

Ted Pattison

**Verteilte Anwendungen mit COM+ und
Microsoft Visual Basic programmieren**

Ted Pattison

Verteilte Anwendungen mit COM+ und Microsoft Visual Basic programmieren

Microsoft Press

Dieses Buch ist die deutsche Übersetzung von: Ted Pattison – Programming Distributed Applications with COM+ and Visual Basic 6.0, Second Edition
Microsoft Press, Redmond, Washington 98052-6399
Copyright 2000 by Microsoft Corporation

Das in diesem Buch enthaltene Programmmaterial ist mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor, Übersetzer und der Verlag übernehmen folglich keine Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieses Programmmaterials oder Teilen davon entsteht.

Das Werk einschließlich aller Teile ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlags unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
01 00

ISBN 3-86063-626-X

© Microsoft Press Deutschland
(ein Unternehmensbereich der Microsoft GmbH)
Konrad-Zuse-Str. 1, 85716 Unterschleißheim
Alle Rechte vorbehalten

Übersetzung: Uwe Thiemann
Korrektorat: anjasiebelkorporat, Siegen
Satz: mediaService, Siegen (www.mediaproject.net)
Umschlaggestaltung: HommerDesign, Haar (www.HommerDesign.com)
Layout und Gesamtherstellung: Kösel, Kempten (www.KoeselBuch.de)

Inhaltsverzeichnis

Danksagung	XIII
Vorwort	XV
Einführung	XVII
Für wen ist dieses Buch?	XVII
Was ist auf der CD enthalten?	XIX
Updates und andere Informationen	XX
1 Ein Überblick über COM+	1
Warum sollten Sie COM+ einsetzen?	2
Von Two-Tier- zu Multi-Tier-Systemen	2
Die Entwicklung der Multi-Tier-Plattform von Microsoft	10
Die Grundlage: COM	11
Von COM zu MTS	13
Von COM und MTS zu COM+	16
COM+-Dienste verstehen	17
Internet Information Services (IIS)	17
Microsoft Message Queuing Services (MSMQ)	18
Queued Components – Warteschlangenkomponenten	19
COM+ Events-Dienst	20
Verteilte Anwendungen mit Visual Basic entwickeln	22
Zusammenfassung	23
2 Schnittstellenbasierte Programmierung verstehen	25
Grundlagen der objektorientierten Programmierung	26
Ändern der öffentlichen Schnittstelle einer Klasse	28
Ein kurzer Leitfaden zur Implementierungsvererbung	29
Trennen der Schnittstelle von der Implementierung	34
Die zwei Gesichter der Vererbung	35
Verwenden von Schnittstellen mit Visual Basic	36
Warum benutzerdefinierte Schnittstellen verwenden?	39
Klassendefinitionen erweitern	42
Verwenden von Schnittstellen in Ihrem Applikations-Design	43
Schnittstellen und COM	43
Zusammenfassung	44

3 COM-Grundlagen	45
Die Entstehung von COM	46
Die Entwicklung binärer Komponenten mit C++	47
Die vTable als speicherinterne Standardrepräsentation	49
Typbibliotheken und Sprachunabhängigkeit	51
Mit IDL arbeiten	52
Die Verbindung zwischen Visual Basic und COM	54
Globale eindeutige Bezeichner (GUIDs)	56
Komponentencode dynamisch laden	58
Objektaktivierung	59
CLSIDs und die Windows-Registrierung	60
Der SCM in Aktion	61
ProgIDs und die CreateObject-Funktion	63
IUnknown und IDispatch verstehen	64
Die IUnknown-Schnittstelle	64
Die QueryInterface-Methode	66
Die IDispatch-Schnittstelle und die Automatisierung	68
Duale Schnittstellen	71
COM außerhalb von Prozessen	73
Die Rolle des Universal-Marshalers	75
Beobachtungen zum prozessexternen COM	77
Zusammenfassung	79
4 Server entwerfen und erstellen	81
Komponenten in Server kompilieren	81
Prozessinterne Server	82
Prozessexterne Server	82
Komponenten für COM+ verpacken	83
Erstellen eines Servers	84
Server-Design	86
Projektübergreifende Typbibliothek-Attribute	87
Die Komponenteneigenschaft <i>Instancing</i>	88
Ändern von Prozedurattributen	91
Friend-Methoden	92
Enumerationen	92
UDTs verwenden	93
Komponenten für Skripting-Clients entwerfen	95
Skripting-Clients und benutzerdefinierte Schnittstellen	95
Beobachtungen zu Skripting-Clients	102
Fehler mit Servern auslösen	102
HRESULT	103
COM-Ausnahmefehler	106
Bedingungen zum Auslösen von Fehlern	109
Fehlerverarbeitung in Skripting-Clients	110
Zusammenfassung	111

5 Komponentenversionsverwaltung	113
Versionsverwaltung unter COM	113
Die Komponentenversionsverwaltung für Scripting-Clients	115
Die Komponentenversionsverwaltung für vTable-gebundene Clients	115
Benutzerdefinierte Schnittstellen erstellen	126
Schnittstellen mit IDL definieren	127
Methodensignaturen mit IDL definieren	131
Enumerationen und UDTs verwenden	133
Kompilieren Ihrer Typbibliothek	135
Typbibliotheken verteilen und konfigurieren	135
Ein weiterer schwieriger Punkt	136
Zusammenfassung	137
6 Konfigurierte Komponenten	139
Konfigurierte Komponenten – Grundlagen	140
COM+-Applikationen	140
Konfigurierte Komponenten erstellen und einsetzen	143
Bibliotheksapplikationen im Vergleich mit Serverapplikationen	146
Prozessexterne Aktivierung	147
Remote-Aktivierung	149
Applikations-Proxies	151
Verteilte Speicherbereinigung	152
Verteilte Applikationen strukturieren	153
Kontexte verstehen	154
Thread-Wechsel-Proxies im Vergleich mit einfachen Proxies	155
Wann werden Kontexte erzeugt?	156
Achten Sie auf den <i>New</i> -Operator	158
Kontexte und Objektreferenzen	158
Das COM+-Programmiermodell	160
Programmieren mit dem Objektkontext	161
Aufrufkontexte verstehen	163
COM+-Komponenten testen und debuggen	164
Konfigurierte Komponenten im Visual Basic-Debugger ausführen	165
Andere nützliche Debug-Techniken	167
Zusammenfassung	169
7 Gemeinsame Ressourcen in COM+-Applikationen nutzen	171
Threads gemeinsam nutzen	172
Eine Win32-Threading-Anleitung	173
Die Threading-Modelle von COM	175
Speicher gemeinsam nutzen	186
Apartments und Standardmodulvariablen	187
Der Shared Property Manager	188
Datenbankverbindungen gemeinsam nutzen	193
Wieso benötigen Sie Datenbankverbindungs-Pooling?	193
Wie funktioniert das Verbindungs-Pooling?	195
Poolfähige Objekte gemeinsam nutzen	199

Wieso ist das Objektpooling wichtig?	200
Objektpooling im Vergleich mit Datenbankverbindungs-Pooling	202
Zusammenfassung	203
8 Transaktionen programmieren	205
Was ist eine Transaktion?	206
Transaktionssysteme	208
Ein Überblick über die »ACID«-Regeln	209
Sperrungen und Feinstrukturierung	211
Schreibsperrungen im Vergleich mit Lesesperrungen	211
Isolationsebenen	212
Lokale Transaktionen programmieren	213
Programmierung verteilter Transaktionen	215
Der Distributed Transaction Coordinator	218
COM+ und deklarative Transaktionen	220
Objekte in einer COM+-Transaktion erstellen	221
Drei wichtige Flags: Happy, Done und Doomed	224
Die Methoden <i>SetComplete</i> und <i>SetAbort</i>	226
Die IContextState-Schnittstelle	227
Das AutoComplete-Attribut	228
Multiobjekt-Transaktionen	230
Datenbankverbindungen und automatische Verknüpfung	234
COM+-Transaktionen und gespeicherte Prozeduren mit Transaktionen	235
Das kurze, aber glückliche Leben eines transaktionalen Objekts	236
Transaktionale Entwürfe	238
COM+-Transaktionen im Vergleich mit lokalen Transaktionen	238
Trennen von Geschäftslogik und Datenzugriffscode	240
Deadlocks	242
Ein letztes Wort zum Tuning	244
Zusammenfassung	245
9 IIS- und ASP-Komponenten erstellen	247
HTTP im Vergleich mit DCOM	248
Die IIS-/ASP-Architektur	250
Das ASP-Gerüst	251
IIS-Applikationen	252
ASP-Anforderungen bearbeiten	253
Visual Basic-Objekte aus ASP-Seiten erstellen	256
Webapplikationen in separate Prozesse unterteilen	259
Isolierte IIS-Applikationen	263
Code zwischen ASP-Seiten und Komponenten aufteilen	265
Die Benutzeroberfläche mit ASP-Seiten erstellen	266
Die Benutzeroberfläche mit kompilierten Komponenten erstellen	267
ASP-Seiten und Komponenten zusammen verwenden	270
Statusverwaltung in Webapplikationen	271
Webapplikationen über den Lastenausgleich skalieren	273
Speichern des applikationsweiten Status im Cache	283
Zusammenfassung	284

10 Messaging und die asynchrone Kommunikation	285
Warum ist das Messaging wichtig?	286
Messaging im Vergleich mit RPC und HTTP	287
MSMQ	290
Die MSMQ-Organisationseinheit	290
MSMQ-Programmierung	292
Das transaktionale Messaging	304
Warteschlangenkomponenten	311
Die Architektur von Warteschlangenkomponenten	313
Warteschlangenkomponenten entwerfen	316
Warteschlangenkomponenten konfigurieren	317
Warteschlangenkomponenten vom Client aus programmieren	319
Warteschlangenkomponenten und die Fehlerbehandlung	322
Der COM+ Events-Dienst	325
Die Architektur von COM+ Events	326
Die Wahl zwischen MSMQ, QC und COM+ -Events	333
Zusammenfassung	334
11 COM+-Sicherheit	335
Sicherheit in Windows 2000	336
Hauptnamen und Autoritäten	337
Lokale Konten im Vergleich mit Domänenkonten	337
Benutzer- und Gruppenkonten	339
Vorkonfigurierte Konten	340
Anmeldungssitzungen und Tokens	342
Ressourcen und DACLS	347
Netzwerk-Authentifizierung und Identitätswechsel	348
Authentifizierungsebenen setzen	351
Sicherheit in COM+	357
Rollen und Autorisierung	357
Deklarative Sicherheit konfigurieren	359
Autorisierung zur Laufzeit durchsetzen	361
Programmgesteuerte Sicherheit	363
Sicherheit in IIS	368
HTTP-Sicherheitsstandards	370
Zertifikate	371
IIS-Authentifizierungsmodi	373
Weitere Quellen	377
Zusammenfassung	377
12 Skalierbare Applikationen entwerfen	379
Skalierbarkeit	379
Die ursprüngliche COM-Vision	380
HTTP in COM+-Applikationen verwenden	382
HTTP von HTML trennen	384
Methodenaufrufe mit HTTP und XML ausführen	386
SOAP und Web Services	388
Applikationsdesign	390

Daten mit COM übergeben	390
Daten mit MSMQ und QC übergeben	393
Daten mit HTTP übergeben	398
Schreibgeschützte Datensatzgruppen im Vergleich mit aktualisierbaren Datensatzgruppen	401
Daten in der mittleren Schicht zwischenspeichern	403
Zusammenfassung	405
Stichwortverzeichnis	407
Der Autor	417

Danksagung

Ich möchte zuerst DevelopMentor danken. Denen, die nicht mit diesem Unternehmen vertraut sind, sei gesagt, dass DevelopMentor Schulungen für Entwickler durchführt, die verteilte Systeme erstellen. DevelopMentor ist eine Gemeinschaft, die aus der lebhaftesten und enthusiastischsten Gruppe von Computerwissenschaftlern besteht, der ich jemals begegnet bin – eine gleichrangige Gruppe der höchsten Ebene technischer Integrität. Ich bin sehr froh, in den letzten sechs Jahren meines Lebens ein Teil dieser Gemeinschaft gewesen zu sein. Viele Ideen, Analogien und Empfehlungen in diesem Buch stammen von einer gemeinsamen Sichtweise und einer Wissensbasis, die innerhalb der DevelopMentor-Gemeinschaft entstanden ist.

Ich möchte Mike Abercombie, Lorrie Trussell und allen anderen Menschen bei DevelopMentor danken. Ich bin dankbar für all eure harte Arbeit, die dieses besondere Umfeld geschaffen hat und mir so viele Möglichkeiten gab. DevelopMentor hat mir erlaubt, Ideen mit einigen Top-Entwicklern der Industrie auszutauschen und meiner Leidenschaft für die Schneidetechnologie nachzugehen.

Mein besonderer Dank gilt Don Box. Ich bin Don sehr verbunden für seine Führung und seinen Rat zu so vielen Aspekten meiner Karriere. Seine Vorstellungen waren für die Strukturierung dieses Buchs sehr wertvoll. Würde er nicht manchmal diese seltsamen Fragen stellen, wenn er ein Notebook kauft, wäre er der intelligenteste Mensch in der Software-Industrie.

Ich danke jenen Themenexperten, die meine Kapitel überprüft haben und wertvolle und konstruktive Kritik lieferten. Sie haben dieses Buch sehr viel besser und detaillierter werden lassen. Ich bekam Unterstützung von den verschiedenen Produktgruppen von Microsoft. Mein Dank gilt Robert Green vom Visual Studio-Team, der die ersten fünf Kapitel dieses Buches überprüft hat. Dank auch George Reilly vom IIS-Team für das Überprüfen des IIS- und ASP-Kapitels. Dick Dievendorff vom MSMQ-Team gebührt ebenfalls Dank, da er das Kapitel lektoriert hat, das Nachrichten thematisiert. Ein spezieller Dank gilt Joe Long vom COM+-Team, der sich die Zeit nahm, um unzählige Fragen zu beantworten, als ich das Kapitel über die internen Details von COM+ schrieb.

Es gab auch viele Leute bei DevelopMentor, die einige Kapitel überprüft haben. Vielen Dank, Bob Beauchemin und Jason Masterman, für Ihre Sachkenntnisse über die Transaktionsverarbeitung und die verteilten COM+-Dienste. Ich danke ebenfalls Tim Ewald für seine ausgezeichneten Empfehlungen bezüglich des architektonischen Entwurfs und für die Erläuterung der unteren Funktionsebenen. Danke, Doug Tenure, für das Feedback zu einigen früheren COM-Kapiteln. Auch Keith Brown, Steve Rodgers und Dan Sinclair gebührt Dank für das gründliche Überprüfen meines Sicherheitskapitels. Dank Aaron Skonnard für die Revision mehrerer Kapitel und die ausgezeichneten Vorschläge zur Erläuterung von Softwareentwicklung, die auf IIS, ASP und XML basiert. Ich danke meinem Kumpel Brian Randell für seine Forschung und seine leidenschaftlichen Ansichten zum Thema der Komponentenversionsverwaltung. Kapitel 5 ist eine adaptierte Version eines Artikels, den ich zusammen mit Brian für das Microsoft System Journal verfasst

habe. Sagen Sie es niemandem (besonders Brian), aber ich glaube, dass einige der Absätze in diesem Kapitel von ihm geschrieben worden sein könnten.

DevelopMentor ist ein wunderbarer Ort, um zu arbeiten, weil das Unternehmen mit legendären Charakteren angefüllt ist. Mike Woodring hat die Island Hopper-Beispiele nur mit Assembler geschrieben. Calvin Caldwell musste einmal zwischen zwei Schritten des ATL COM-Applikationsassistenten zu seiner Ranch fahren, um Geburtshilfe für ein Babykalb zu leisten. Während eines wochenlangen Intensivkurses, führte Dr. Joe Hummel einige eifrige Studenten mit einem Taxi nach Hollywood. Dies geschah in den frühen Morgenstunden, um realistisch zu simulieren, wie Pakete über das Internet gesendet werden. Und dann gibt es diesen Menschen, Fred Wesley, der so schwer und tief denkt, dass keiner von uns wirklich ein Wort von dem versteht, was er sagt. Der Lustige hieran ist, dass er so viel Zeit damit verbringt, über die Theorie des Denkens zu debattieren, dass er niemals zur entsprechenden Praxis gelangt.

Ich möchte auch allen Mitarbeitern bei DevelopMentor danken. Dazu zählen Niels Berglund, Henk de Koning, Jon Flanders, Martin Gudgin, Stuart Halloway, Justin Hoagland, Simon Horrell, Kevin Jones, Paul Kirby, John Lam, Brian Meso, Jose Mojica, Gus Molina, Brad E. Needham, Fritz Onion, Simon Perkins, Brent Rector, Dave Schmitt, George Shepherd, David Smallberg, Dan Weston und Jason Whittington. Sie alle haben auf die eine oder andere Weise zu meinem allgemeinen Verständnis von Informatik und Softwareentwicklung beigetragen.

Schließlich muss ich Chris Sells, dem »bösen Genie« von DevelopMentor Anerkennung zollen. Chris ist mehr als jede andere Person für die Richtung, in die meine Karriere verlief, verantwortlich. Er ist der Mensch, der mich letztlich dazu führte, über Visual Basic und COM zu schreiben. Es waren Chris adäquate Erklärungen zu `const`, der Vorlagenspezialisierung und den `functor`-Objekten, die mich davon überzeugten, dass ich als C++-Programmierer niemals glücklich sein würde.

Ich danke jedem, mit dem ich bei Fawcette Technical Publications gearbeitet habe. Dank allen schwer arbeitenden Menschen, die die VBITS-Tagungen einberiefen: Tena Carter, Janet Nickels, Robert Scoble, Maryam Ghaemmaghami, Jennifer Brucker und Diamond Jim Fawcette.

Dank auch an Joshua Trupin und Joseph Flanigen für die Veröffentlichung meiner Artikel in MSDN Magazine, Microsoft Systems Journal und MIND. Ich danke auch den anderen an diesen Veröffentlichungen beteiligten Menschen, die mir halfen, meine Worte in gedruckte Form zu bringen. Dazu zählen Joanne Steinhart, Terry Dorsey, Etna Novik, Joan Levinson und Michael Longacre.

Ich danke Claudette Moore und Debbie McKenna von der Moore Literary Agency für ihre Unterstützung und Assistenz. Mein besonderer Dank gilt Claire Horne, weil sie mir half, das Konzept zur ersten Auflage dieses Buchs zu erstellen und zu verkaufen.

Ich danke allen Mitarbeitern von Microsoft Press, da sie mir halfen, dieses Buch zu schreiben. Dazu zählen Ben Ryan und ein sehr begabtes Bearbeitungs- und Produktionsteam. Vielen Dank an Kathleen Atkins. Sie vollbringt einen ausgezeichneten Job als leitende Sachbearbeiterin und Projektmanagerin. Dank auch an Sally Stickney für ihre Beiträge zur Bearbeitung des Manuskripts. Ich hatte das Glück, Ina Chang als Manuskriptbearbeiterin und Steve Perry als technischen Bearbeiter zu haben. Ich hoffe, in Zukunft wieder mit all diesen Menschen arbeiten zu dürfen.

Schließlich möchte ich den Anwälten danken, die für Microsoft arbeiten. Zurückblickend bin ich sehr dankbar, dass sie mich davon überzeugt haben, immer »Visual Basic« zu schreiben, anstatt die beiden Buchstaben, die wir alle kennen, als Abkürzung für unser beliebtes Entwicklungswerkzeug zu verwenden. Sie haben diesem Text mindestens zehn Seiten hinzugefügt und somit den Wert meines Buches für den Gelegenheitskäufer, der in den Regalen der örtlichen Buchhandlung stöbert, erhöht.

Einführung

Jeder Entwickler, der Multi-Tier-Applikationen mit der Microsoft Windows-Plattform als Grundlage erstellen möchte, muss viele unterschiedliche Softwarekomponenten verwenden. Einige dieser Komponenten werden von Microsoft angeboten, und Sie oder Ihr Unternehmen werden die anderen Komponenten selbst schreiben. Einige Komponenten können wiederum von Drittanbietern bezogen werden.

Das Komponentenobjektmodell (Component Object Model – COM) ist der Klebstoff, der all diese Komponenten aneinander bindet. Es ermöglicht Programmierern und Unternehmen den effizienten Vertrieb und die Wiederverwendbarkeit ihres Programmcodes. COM+, eine erweiterte Version von COM, stellt eine wertvolle Umgebung für Komponenten zur Verfügung, die in der mittleren Schicht ausgeführt werden. COM+ und einige andere wichtige Windows-Dienste, wie zum Beispiel Microsoft Internet Information Services (IIS), Active Server Pages (ASP) und Microsoft Message Queuing (MSMQ), sind die Elemente, die den Programmierern und Unternehmen gestatten, große Multi-Tier-Applikationen zu entwickeln.

Für wen ist dieses Buch?

Ich glaube, dass detaillierte Kenntnisse über COM und COM+ unbedingt erforderlich sind, um Multi-Tier-Applikationen mit Windows 2000 erstellen zu können. Aus meiner Sicht gibt es keine andere Möglichkeit, skalierbare Applikationen ohne ein solides Verständnis der Infrastruktur, die die Applikationen unterstützt, zu erstellen. Das Ziel dieses Buches besteht deshalb darin, die wichtigen Bereiche von COM+ und Windows 2000 zu erläutern, die Einfluss darauf haben, wie Sie Komponenten für eine verteilte Anwendung entwerfen und schreiben.

Dieses Buch richtet sich an fortgeschrittene Anfänger und erfahrene Visual Basic-Programmierer, die Applikationen für COM+ und Windows 2000 entwickeln möchten. Das Buch konzentriert sich auf die Architektur der Windows-Plattform. Es dient ebenfalls als Ressource für ASP- und C++-Entwickler. Einige Leser haben vielleicht noch nicht mit COM gearbeitet. Andere haben möglicherweise bereits Erfahrungen mit COM und MTS (Microsoft Transaction Server) gesammelt. Mein Ziel in dieser Ausgabe besteht darin, beiden Lesergruppen gerecht zu werden.

Ich habe in den letzten fünf Jahren oft und intensiv darüber nachgedacht, wie eine Beschreibung erstellt werden könnte, die die exakte Menge an Details enthält. Einige technische Details sind sehr wichtig. Andere sind esoterisch und bedeuten lediglich eine Verschwendung der mentalen Kapazitäten.

Viele Programmierer erstellen verteilte Anwendungen mit dem Microsoft Transaction Server (MTS) und Microsoft Message Queue (MSMQ). Eine Vielzahl dieser Programmierer möchte dazu Visual Basic anstelle von C++ verwenden. Diese Programmierer sollten verstehen, auf welche Weise COM alles zusammenfügt. Sie sollten außerdem wissen, wie Komponenten erstellt werden, die sowohl effizient als auch erweiterungsfähig sind. Ich habe versucht, wichtige Themen zu beschreiben und auf irrelevante Inhalte zu verzichten. Ich wollte möglichst viel Territorium abste-

cken, während der Text prägnant bleiben sollte. Während ich unnötige Komplexität vermied, nahm ich das Recht in Anspruch, in die Tiefe zu gehen, wann immer dies angebracht war.

Einige Visual Basic-Programmierer verfügen nicht über die Motivation oder Eignung, um COM und COM+ auf dieser Ebene zu erlernen. Visual Basic kann diesen Programmierern – ohne dass diese Kenntnisse über die zugrunde liegende Technologie besitzen müssen – einen geringen Teil der Plattformfunktionalität zugänglich machen. Diese Programmierer werden jedoch keine umfangreichen Informationssysteme mit COM+, IIS, ASP und MSMQ erstellen. Dieses Buch richtet sich definitiv nicht an sie.

Für die Leser, die bereits mit der ersten Ausgabe vertraut sind

Ich mag es nicht, die zweite Ausgabe eines technischen Buches zu kaufen, nur um festzustellen, dass es einen neuen Umschlag, aber denselben alten Text enthält. Eines der Hauptziele dieser zweiten Ausgabe bestand darin, etwas zu erzeugen, das einen zusätzlichen Wert für jene darstellt, die die erste Ausgabe gelesen haben. Über 75 Prozent des Textes für dieses Buch wurden für die zweite Ausgabe neu geschrieben.

Ich habe die Beschreibung der Grundlagen des klassischen COM umstrukturiert und sie zu Beginn des Buches angeordnet. Kapitel 2, das die schnittstellenbasierte Programmierung beschreibt, unterscheidet sich in beiden Ausgaben kaum. Ich habe die COM-Grundlagen, die in der ersten Ausgabe über mehrere Kapitel verteilt waren, in Kapitel 3 dieses Buches zusammengefasst. Kapitel 4 und 5 beschreiben, wie Visual Basic benutzt wird, um Komponenten zu erstellen und deren Versionen zu verwalten. Das Material in diesen Kapiteln ist, verglichen mit der ersten Ausgabe, erweitert worden, um neue Aspekte der Erstellung von benutzerdefinierten Typbibliotheken mit IDL und des Entwurfs von Komponenten für Scripting-Clients zu beschreiben. Programmierer, die bereits mit COM und MTS vertraut sind, können die Kapitel 2 bis 5 als schnelle Übersicht oder als Referenz verwenden.

Kapitel 6 und 7 erörtern die Architektur der COM+-Laufzeitumgebung. Sie werden lernen, wie konfigurierte Komponenten generiert werden, die die Vorteile der Laufzeitdienste und des von COM+ angebotenen Thread-Poolings nutzen. Kapitel 8, das Transaktionen thematisiert, vergleicht lokale mit verteilten Transaktionen. Kapitel 9 enthält neue Beschreibungen von IIS und ASP. Kapitel 10, das sich mit Nachrichten befasst, erläutert MSMQ. Sie finden hier Analogien zur ersten Ausgabe, aber auch neues Material zu Warteschlangenkomponten und COM+-Ereignissen. Kapitel 11 ist von Grund auf neu geschrieben und widmet sich der Sicherheit. Kapitel 12 behandelt den Applikationsentwurf, der sich auf die Skalierbarkeit und Performance auswirkt.

Welches Hintergrundwissen benötigen Sie?

Ich setze voraus, dass Sie ein wenig Erfahrung mit der objektorientierten Programmierung und der Erstellung von Anwendungen besitzen, die Klassen nutzen. Dabei spielt es keine Rolle, ob Sie Klassen mit Visual Basic, C++ oder Java programmieren. Sie sollten lediglich wissen, wieso Sie eine Klasse unter Verwendung der Kapselung entwerfen. Sie sollten ebenfalls die Beziehung zwischen einer Klasse und einem Objekt verstehen.

Es ist hilfreich, aber nicht unbedingt erforderlich, wenn Sie ein wenig über Computerwissenschaft oder eine Low-Level-Programmiersprache wie zum Beispiel C wissen. Ich kann unmöglich über COM schreiben, ohne Begriffe wie Pointer, Speicheradressen, Stack-Blöcke und Threads zu nennen. Sollten Sie nicht über dieses Wissen verfügen, nehmen Sie sich ein wenig Zeit, um darüber nachzudenken, was auf den unteren Software- und Hardware-Ebenen geschieht. Ihre Schläfen werden möglicherweise gelegentlich ein wenig pochen, doch die von Ihnen investierte Zeit und der Aufwand werden sich lohnen.

Die meisten Leser dieses Buches verfügen über Kenntnisse in der Datenbankprogrammierung. Es ist nur schwer vorstellbar, dass ein fortgeschrittener Anfänger der Visual Basic-Programmierung bestehen kann, ohne in mindestens eine datenbankorientierte Anwendung involviert gewesen zu sein. Wenn ich die Programmierung von Transaktionen für COM+-Objekte beschreibe, gehe ich davon aus, dass Sie ausreichend über die ActiveX Data Object-Programmierung (ADO) sowie das Schreiben von SQL-Anweisungen informiert sind. Wenn Ihnen dieses Wissen fehlt, sollten Sie es sich zunächst aneignen. Der ADO-Code und die in diesem Buch präsentierten SQL-Anweisungen sind nicht sehr kompliziert.

Für das Kapitel über IIS und ASP setze ich voraus, dass Sie die Grundlagen zur Verwendung von HTML und zum Erstellen einfacher ASP-Seiten kennen. Hinsichtlich der Erfahrung der meisten Leser und der Verfügbarkeit von qualitativ hochwertigem Referenzmaterial zu ASP glaube ich, dass diese Details nicht beschrieben werden müssen.

Was ist nicht in diesem Buch enthalten?

Dieses Buch enthält nicht viele Schritt-für-Schritt-Anleitungen. Es wird deshalb nicht jene ansprechen, die wissen möchten, wie eine gewisse Aufgabe gelöst wird, ohne sich für die Gründe für die entsprechende Vorgehensweise zu interessieren. Sie finden überall im Buch Verweise auf das MSDN, um Details zu Praktiken wie zum Beispiel die Verwendung administrativer Werkzeuge für COM+ oder IIS zu erhalten. Ich glaube nicht, dass dies Bereiche sind, in denen die meisten Programmierer Unterstützung benötigen. Mein Ziel ist Ihr Verständnis der Theorie, die sich hinter der Software verbirgt.

Ich werde nicht häufig davon berichten, wie die COM+- oder IIS-Administration mit Hilfe von Skripten oder benutzerdefinierten administrativen Applikationen automatisiert wird. Während sowohl COM+ als auch IIS umfassende Objektmodelle für die Automatisierung administrativer Routinearbeiten zur Verfügung stellen, verbringe ich nicht viel Zeit mit deren Erläuterung. Ich widme mich diesen Themen hier und dort, aber ich erwarte, dass Sie sich die Details mit Hilfe der Dokumentation des MSDN und Plattform SDK aneignen.

Wenn Sie ein Buch mit einer großen Beispieldarstellung suchen, die Sie als Ausgangspunkt verwenden können, ist dies nicht das richtige Buch für Sie. Die meisten meiner Code-Listings sind kurz, zwischen fünf und zehn Zeilen. Wenn ich ein Code-Listing präsentiere, versuche ich immer, Ihre Aufmerksamkeit mit möglichst wenigen Zeilen auf einen besonderen Aspekt zu richten. Ich verzichte auf unwesentliche Elemente, wie zum Beispiel die Fehlerbehandlung. Mein Stil eignet sich somit nicht für jene, die meine Beispiele blind in den eigenen Betriebscode einfügen. Das Ziel meiner Code-Präsentationen ist, Sie das »Fischen« zu lehren, und nicht, Ihnen den Fisch zu geben.

Was ist auf der CD enthalten?

Die CD, die sich in diesem Buch befindet, enthält den Source-Code mehrerer Visual Basic-Anwendungen. Diese Applikationen enthalten die Arbeitsbeispiele des Codes, den ich im Buch verwende. Als Visual Basic-Programmierer habe ich immer wieder feststellen müssen, dass sich mein Verständnis für bestimmte Sachverhalte erweitert, wenn ich die Taste F5 betätige und ein Programm im Einzelschrittmodus ausführen lasse. Ich möchte Ihnen die gleiche Gelegenheit bieten. Die Datei *Samples.htm* informiert Sie über alle Applikationen und deren Funktionsweise.

Das Setup-Verzeichnis auf der CD enthält die Dateien und Anweisungen für die Installation der SQL-Server-Datenbank, die für die Ausführung der Market-Applikation benötigt wird. Die Datei *Setup.htm* beschreibt die erforderlichen Installationschritte. Alle anderen Applikationen mit zu-

sätzlichen Setup-Informationen verfügen ebenfalls über eine *Setup.htm*-Datei in den entsprechenden Verzeichnissen. Ich hoffe, dass Ihnen die Beispielapplikationen eine wertvolle Hilfe sind.

Welche Software benötigen Sie, um die Beispielapplikationen ausführen zu können?

Wenn Sie alle auf der CD-ROM vorhandenen Beispielapplikationen ausführen lassen möchten, müssen Sie Windows 2000 Server oder Windows 2000 Advanced Server installieren. Jede Version enthält COM+ und IIS als Teil der Standardinstallation. Wenn Sie das Nachrichtenbeispiel aus Kapitel 10 ausführen lassen möchten, müssen Sie auch MSMQ installieren, das nicht Bestandteil der Standardinstallation ist.

Sie sollten auch Visual Studio 6 einschließlich Visual Basic 6, Visual C++ und der neuesten Version von MSDN installieren. Wenn Sie Visual Studio installieren, sollten Sie in jedem Fall die Option *Umgebungsvariablen registrieren* selektieren. Die Gründe hierfür werden im fünften Kapitel deutlich. Sie sollten nach der Installation von Visual Studio das Visual Studio Service Pack 3 oder eine spätere Version installieren. Um die Beispielapplikationen der letzten Kapitel ausführen zu können, müssen Sie auch SQL Server 7 installieren. Sie können natürlich auch für diese Software das aktuelle Service Pack einrichten. Schließlich sollten Sie das Microsoft Plattform SDK installieren. Verwenden Sie die Ausgabe vom Januar 2000 oder später. Das Plattform SDK stellt viele wichtige Hilfsmittel und Werkzeuge zur Verfügung, die überall im Buch beschrieben sind.

Updates und andere Informationen

Während meiner Arbeit als Dozent und Autor erstelle und verbessere ich fortwährend Visual Basic-Anwendungen, die sich auf die COM- und COM+-Programmierung beziehen. Es steht Ihnen frei, die aktuelle Beispielkollektion von meiner Website <http://www.SubliminalSystems.com> zu laden. Auf dieser Site finden Sie außerdem weitere Informationen zu diesem Buch, einschließlich eine Liste aller Fehler. Wenn Sie mir Feedback senden möchten, können Sie die E-Mail-Adresse tedp@SubliminalSystems.com verwenden. Ich hoffe, dass Ihnen dieses Buch gefällt.

1 Ein Überblick über COM+

2	Warum sollten Sie COM+ einsetzen?
10	Die Entwicklung der Multi-Tier-Plattform von Microsoft
17	COM+-Dienste verstehen
22	Verteilte Anwendungen mit Visual Basic entwickeln

Gerade ist ein Jahrtausend zu Ende gegangen und ein neues hat begonnen. Wie stark hat dieser Wechsel Ihr Leben als Entwicklungsprofi beeinflusst? Als die Uhren Mitternacht schlugen, gab es vielleicht hier und da einige kleinere Unzulänglichkeiten. Ein paar veraltete Systeme mussten endgültig entsorgt werden. Entgegen aller Vorhersagen in den Medien brach damit jedoch für unsere gesamte Industrie keineswegs der Jüngste Tag an. Das gilt insbesondere für diejenigen von uns, die ihre Anwendungen für Betriebssysteme und mit Entwicklungswerkzeugen der letzten zehn Jahre geschrieben haben. In der Rückschau ist das Ganze fast zum Lachen. Und die Gerüchte über den Ausfall aller Informationssysteme Ihres Unternehmens waren offensichtlich maßlos übertrieben.

Aber weiß die Presse je wirklich, was in unserer Industrie vor sich geht? Sie hat schon die Tatsache versäumt, dass es sich eigentlich um einen Jahrhundertfehler, nicht um einen Jahrtausendfehler handelte. Das Problem wurde durch die Verwendung von zweistelligen Datumswerten anstelle vierstelliger Datumswerte verursacht. Hätten die Menschen unsere heutige Technologie schon vor 100 Jahren besessen, hätten wir am Neujahrsabend 1899 denselben Problemen gegenübergestanden. Aber »Millenniumsfehler« hörte sich natürlich viel sensationeller an und war deshalb verkaufsfördernder für Zeitungen und Zeitschriften.

Erinnern Sie sich, wie die Presse noch vor ein paar Jahren vorhersagte, dass alles in Java neu geschrieben werden müsse, sogar die Verfassung der USA und die Magna Charta? Heute ist die Java-Hysterie abgeklungen und wir sehen Java als das, was es ist – eine junge und vielversprechende Sprache mit Stärken und Schwächen. Sie hat die Industrie nicht im Sturm erobert: eine Sprache, die für einige Aufgabenstellungen ganz besonders, für andere dagegen überhaupt nicht geeignet ist.

Heute spricht man überall von XML als allein glücklich machende Technologie. Ich bin mir sicher, es wird immer mehr Einsatzbereiche für XML geben, sobald es ausgereifter ist. Es wird viele Probleme lösen, die heute sehr komplex und sehr real sind. Aber XML kann niemals Technologien wie COM, Corba und Java ersetzen, wie viele Leute bereits angenommen haben. Wir sollten all diese neuen Technologien im Auge behalten. Es ist jedoch nicht sinnvoll, sich auf eine einzige Technologie zu konzentrieren, die schon in ein paar Jahren all Ihre Probleme lösen kann, wenn Sie Ihr Programm in den nächsten Monaten ausliefern müssen.

Warum sollten Sie COM+ einsetzen?

Ich nehme an, Sie möchten COM+ zur Entwicklung verteilter Anwendungen einsetzen. Viele Unternehmen verwenden COM+ und Microsoft Windows 2000, weil damit eine robuste Entwicklungsplattform bereitgestellt wird. Diese Plattform setzt sich aus mehreren Kerntechnologien zusammen, die die grundlegenden Bausteine zur Konstruktion von Multi-Tier-Geschäftsanwendungen bilden. Und außerdem: je mehr die zugrunde liegende Plattform leistet, desto weniger Code müssen Sie schreiben und debuggen.

Ich brauche Sie vermutlich nicht davon zu überzeugen, dass Multi-Tier-Anwendungen viele Vorteile gegenüber den Two-Tier-Anwendungen bieten, die Ende der 80er und Anfang der 90er Jahre so aktuell waren. Ihr Unternehmen hat möglicherweise bereits entschieden, den Two-Tier-Ansatz zu verwerfen und der Multi-Tier-Strategie den Vorzug zu geben. Dennoch möchte ich zu Beginn dieses Kapitels die wichtigsten Probleme der Two-Tier-Anwendungen ansprechen und erklären, wie man mit Hilfe von Multi-Tier-Anwendungen Lösungen dafür findet. Außerdem werde ich darauf eingehen, warum die Multi-Tier-Entwicklung neue Probleme und zusätzliche Komplexitäten einführt.

Eines der Ziele, die Microsoft bei der Entwicklung von COM+ verfolgte, war, den Unternehmen die Vorteile von Multi-Tier-Anwendungen anzubieten und dabei gleichzeitig die diesem innewohnende Komplexität soweit wie möglich zu verbergen. In den letzten zehn Jahren hat Microsoft viele Fortschritte gemacht, diese Infrastruktur für verteilte Anwendungen zu erstellen.

Die erste Version von COM wurde 1993 verkauft. Seitdem hat sich COM von einer jungen und komplizierten Technologie zur Seele der Multi-Tier-Strategie von Microsoft entwickelt. Dieses Kapitel betrachtet die wichtigsten Meilensteine von COM. Dabei werde ich versuchen, alle Akronyme zu definieren, die die Marketingleute über die Jahre erfunden haben. Sie haben vielleicht schon von OLE, DCOM, ActiveX und MTS gehört. COM+ und DNA sind die zwei neuesten Einträge in der Liste. Aber haben Sie je versucht, jemandem auf einer Party den Unterschied zwischen all diesen Dingen zu erklären? Nicht ganz einfach. Sie bedeuten nämlich für unterschiedliche Leute unterschiedliche Dinge.

Das Kapitel setzt sich mit einer allgemeinen Übersicht über die in die COM+-Plattform integrierten verteilten Dienste fort. Jede nichttriviale Multi-Tier-Anwendung benötigt Dinge wie Transaktionsverarbeitung, integrierte Sicherheit, einen Web-Server, Messaging sowie die Weitergabe von Ereignisbenachrichtigungen. Dieses Kapitel beschreibt, wo diese COM+-Dienste zu finden sind. Bevor wir in den nachfolgenden Kapiteln auf Details eingehen werden, möchte ich Ihnen zeigen, wie sich alles zusammenfügt. Damit verschaffen Sie sich einen Eindruck von COM+ als Ganzem und sehen hoffentlich das Licht am Ende des Tunnels.

Von Two-Tier- zu Multi-Tier-Systemen

Einer der besten Anlässe COM+ zu nutzen besteht darin, die Informationssysteme eines Unternehmens von einer Two-tier-Architektur in eine Multi-Tier-Architektur überzuführen. Dieser Übergang bedingt den Entwurf und die Implementierung einer Middle-Tier-Schicht mit Geschäftsobjekten. Geschäftsobjekte sitzen in der Regel zwischen Client-Anwendungen und Datenbankservern. COM+-Server sind die Plattformen für diese Art von Systemen.

In der Industrie werden Two-Tier-Anwendungen bereits allgemein eingesetzt, deshalb sind die damit einhergehenden Probleme bereits bekannt. Wir wollen die wichtigsten Unzulänglichkeiten einer herkömmlichen Two-Tier-Architektur noch einmal betrachten, wie in Abbildung 1.1 gezeigt.

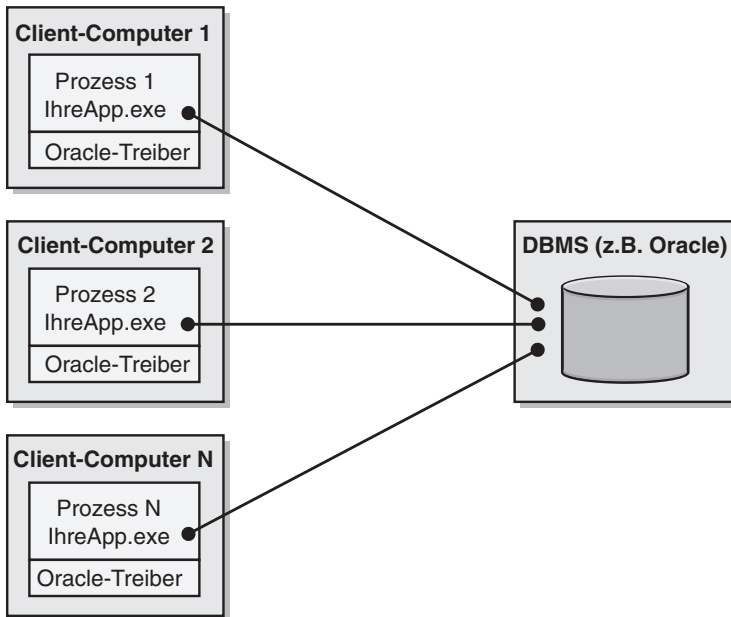


Abbildung 1.1: Typische Two-Tier-Anwendungen bedingen für jeden Benutzer eine separate Verbindung zu einem Datenbankserver. Auf dem Computer des Benutzers muss ein geeigneter Treiber für das jeweilige DBMS (Datenbankmanagementsystem) vorhanden sein.

- Code für die Benutzerschnittstelle ist mit Code für die Geschäftslogik und den Datenzugriff vermischt, so dass es schwierig ist, den Code für die Geschäftslogik und den Datenzugriff in anderen Client-Anwendungen wiederzuverwenden. Darüber hinaus müssen Client-Anwendungen bei Änderungen der Geschäftslogik oder der Datenbank häufig neu erstellt und ausgeliefert werden.
- Jede Client-Anwendung läuft zusammen mit dem Code für die Geschäftslogik und dem Datenzugriff in einem separaten Prozess, so dass Sie die prozessspezifischen Ressourcen wie Threads und Speicher nicht gemeinsam nutzen können. In einer Two-Tier-Anwendung können Sie auch keine Datenbankverbindung gemeinsam nutzen, weil jeder Benutzer eine separate Verbindung zum Datenbankserver benötigt. Dasselbe gilt für Client-Anwendungen, die eine Verbindung zu einem Mainframe-Computer herstellen. Dieser Mangel einer effektiven gemeinsamen Nutzung von Ressourcen durch mehrere Benutzer begrenzt den allgemeinen Durchsatz sowie die Skalierbarkeit eines Systems.
- Jeder Client-Computer braucht einen oder mehrere eigene Treiber, um mit einem Datenbankserver oder einer Mainframe-Anwendung kommunizieren zu können, wodurch die Wartung und die Konfiguration von Desktop-Computern aufwendiger wird. Der Systemadministrator muss auf jedem Computer mehrere Treiber installieren und warten, um beispielsweise Zugriff auf ODBC-Datenbanken (Open Database Connectivity) zu bieten. Wenn Sie Ihre Backend-Datenbank ändern – beispielsweise wenn Sie von Sybase auf Oracle umsteigen –, muss der Administrator diese Änderung auf jedem einzelnen Desktop nachvollziehen. Das kann teuer werden.
- Client-Anwendungen haben es schwer, auf Daten aus mehreren Datenquellen zuzugreifen. In einer Firmenumgebung ist es nicht unüblich, wichtige Geschäftsdaten auf mehrere verschie-

dene Systeme zu verteilen, wie in Abbildung 1.2 gezeigt. Das Ganze wird kompliziert, wenn eine Anwendung Geschäftstransaktionen ausführen will, für die Daten benötigt werden, die auf mehrere Computer verteilt sind. Und noch schwieriger wird es, wenn die Daten in unterschiedlichen Formaten abgelegt sind.

- Client-Anwendungen müssen häufig im selben LAN (Local Area Network) wie der Datenbankserver oder die Mainframe-Anwendung ausgeführt werden, so dass es nicht möglich ist, verteilte Anwendungen zu entwickeln, die geografische Distanzen abdecken. Viele Unternehmen haben Angestellte auf der ganzen Welt, und viele davon müssen Anwendungen für Kunden und Lieferanten schreiben, die nie in eine so kontrollierte Umgebung wie ein Unternehmens-LAN integriert werden könnten.
- Die Benutzer sind auf bestimmte Plattformen festgelegt. Beispielsweise ist es in einem Two-tier-System häufig erforderlich, dass jeder dasselbe Betriebssystem ausführt, beispielsweise eine 32-Bit-Version von Windows. In einer Welt, in der jeder Computer mit jedem anderen Computer verbunden werden kann, wird diese Beschränkung immer mehr zu einem Hindernis.
- Two-tier-Systeme funktionieren nicht mehr, wenn die Computer nicht mehr verbunden sind, beispielsweise wenn der Datenbankserver heruntergefahren wird oder wenn der Laptop aus der Verbindung genommen wird, weil man gerade im Flugzeug oder bei einem Kunden arbeitet. Die Verfügbarkeit eines Systems ist sehr beschränkt, wenn Ausfallzeiten auftreten und verschiedene Computer keine direkte Verbindung mehr zueinander aufnehmen können.

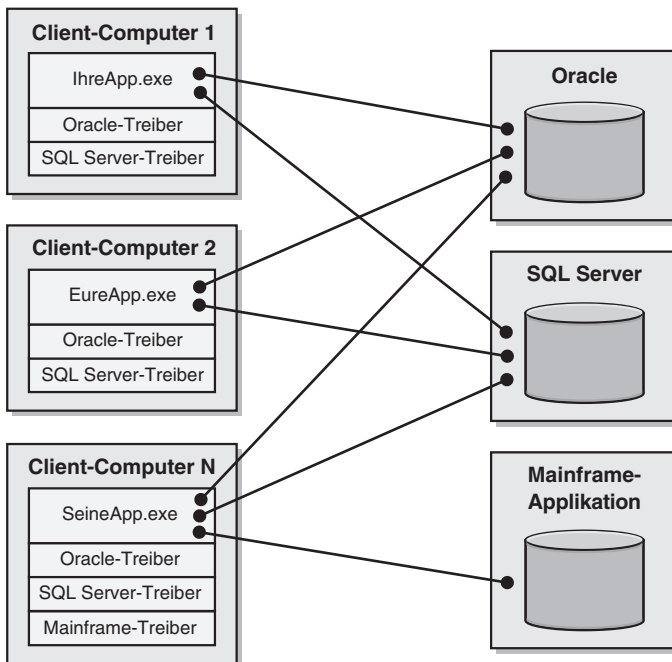


Abbildung 1.2: Eine Two-Tier-Strategie funktioniert nicht zufrieden stellend, wenn ein Unternehmen mehrere Client-Anwendungen oder mehrere Datenbankserver einsetzt.

Aufteilung der Präsentationsebene

Das Problem einer Two-Tier-Architektur kann wie viele andere Probleme in der Informatik gelöst werden, indem man eine weitere Referenzierungsebene einführt. Sie können die Client-Anwendungen von dem Code für die Geschäftslogik und dem Datenzugriff abkoppeln, indem Sie mehrere *Geschäftsobjekte* (*Business Objects*) einführen, wie in Abbildung 1.3 gezeigt. Die Client-Anwendung, die den Code für die Benutzeroberfläche enthält, wird häufig auch als *Präsentationsebene* bezeichnet. Wenn nicht anders angegeben, bezeichnet der Begriff *Client-Anwendung* in diesem Buch eine Anwendung der Präsentationsebene, die Elemente und Code für die Benutzeroberfläche enthält.

Geschäftsobjekte erlauben Ihnen, Ihre Logik zentral abzulegen und sie in unterschiedlichen Client-Anwendungen wiederzuverwenden. In Netzwerken, die auf Windows 2000 und Windows NT basieren, können diese Geschäftsobjekte mit Hilfe von COM eingesetzt werden. Außerdem kann COM die Grundlage für eine Remote-Kommunikation zwischen Client-Anwendungen und Middle-Tier-Objekten bilden.

Eines der wichtigsten Verkaufsargumente für COM ist, dass es den Middle-Tier-Programmierern erlaubt, den Code in ihren Geschäftsobjekten zu aktualisieren, ohne dass die Client-Anwendungen neu kompiliert oder neu bereitgestellt werden müssten. Es ist ganz einfach und völlig schmerzlos, den Code für Ihre Geschäftslogik oder für den Datenzugriff zu ändern, nachdem Ihre Komponenten in die Produktionsumgebung übernommen wurden, da die Client-Anwendungen von den Implementierungsdetails Ihrer Geschäftsobjekte abgeschirmt werden.

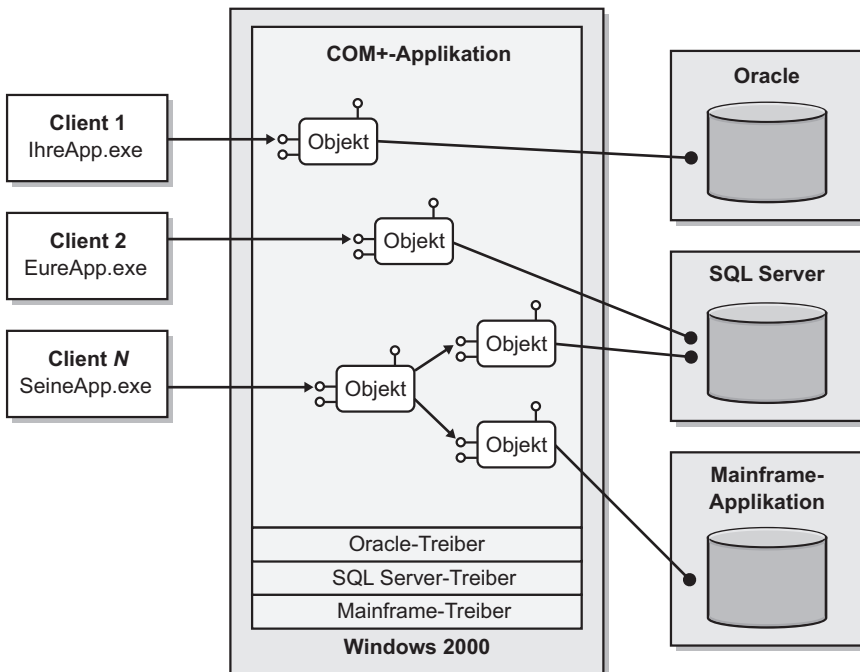


Abbildung 1.3: Durch das Einfügen mehrerer Geschäftsobjekte in eine Zwischenebene werden aufwendige Abhängigkeiten zwischen Client-Anwendungen und Datenbankservern eliminiert.

Die Daten einer Anwendung können Speicherformate ändern, und neue Datenbankserver können eingesetzt werden, ohne dass dadurch Ärger und Verdruss entstünde. Manchmal nimmt man nur an einer einzigen Microsoft Visual Basic DLL (Dynamic Link Library) Änderungen vor. Es ist ganz einfach, eine DLL zu kompilieren und auf einem Produktionsserver in der mittleren Schicht zu ersetzen. Die Client-Anwendungen bleiben in der Produktionsumgebung unverändert.

In einer typischen Multi-Tier-Anwendung, wie in Abbildung 1.3 gezeigt, werden mehrere Middle-Tier-Objekte für viele Benutzer ausgeführt. Alle diese Objekte laufen jedoch häufig innerhalb eines einzigen Prozesses auf einem Windows 2000-Server. Damit wird es möglich, dass mehrere Benutzer prozessabhängige Ressourcen wie beispielsweise Threads, Speicher und Datenbankverbindungen, gemeinsam nutzen. Außerdem brauchen die Computer, auf denen die Client-Anwendungen ausgeführt werden, keine Datenbanktreiber, wie es im Two-Tier-Modellen der Fall ist.

Logische Schichten im Vergleich zum physischen Einsatz

Eine der ersten Fragen, die sich bei der Multi-Tier-Entwicklung stellt, ist, wo jede der Schichten eingesetzt werden sollte. Abbildung 1.4 zeigt zwei mögliche Szenarien. Die Anzahl der Schichten sagt nicht unbedingt etwas darüber aus, wie viele Computer beteiligt sind. In einer kleinen Umgebung könnten der Geschäftscode und der Datenbankserver auf demselben Computer ausgeführt werden. In einem größeren System können die Daten auf einem oder mehreren dafür abgestellten Computern bereitgestellt werden, während die Geschäftsobjekte auf einem separaten Host laufen.

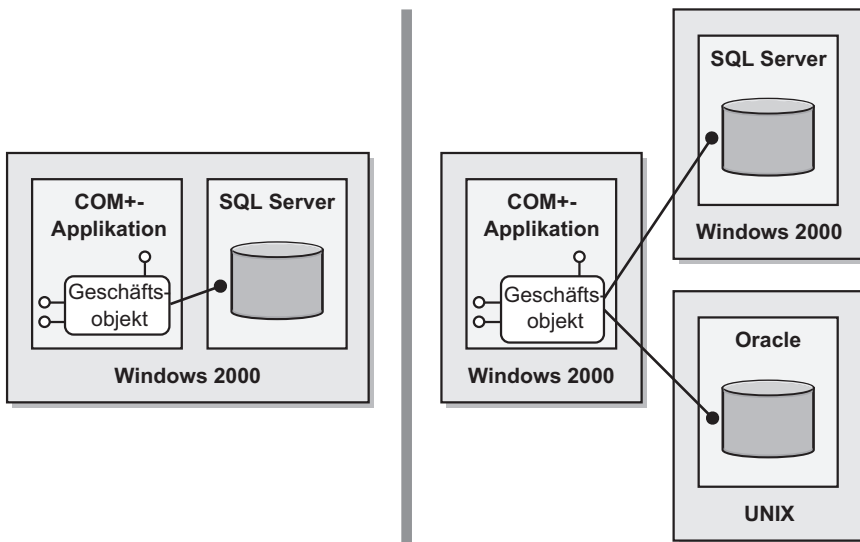


Abbildung 1.4: Die Zuordnung zwischen Geschäftsobjekten und der Schicht für den Datenzugriff kann zunächst ganz einfach sein und dann komplexer werden, ohne dass die Clientanwendungen dadurch beeinflusst werden.

Einige physische Nutzungsschemata sind einfach einzurichten und wenig aufwendig. Andere bieten eine höhere Skalierbarkeit, Zuverlässigkeit und Fehlertoleranz. Wenn man von Multi-Tier-Systemen spricht, meint man damit in der Regel mehrere verschiedene *logische Schichten*. Der physische Einsatz kann variieren und ganz einfach geändert werden, nachdem ein System in die Produktionsumgebung übernommen wurde. Eine der großen Stärken von COM ist, dass es

Ihnen erlaubt, beim physischen Einsatz Änderungen vorzunehmen, ohne dass Anwendungscode geändert oder neu kompiliert werden müsste.

Einige Leute verwenden statt *Multi-Tier* auch die Begriffe *Three-Tier* oder *N-Tier*. Diese Begriffe bedeuten größtenteils dasselbe – dass ein System drei oder mehr Schichten nutzt. Abbildung 1.5 zeigt eine komplexere Menge physischer Schichten. Im Multi-Tier-Modell können die Schichten für Geschäftsdaten und Datenzugriff beliebig komplex sein. Das Beste an diesem Modell ist, dass die Client-Anwendungen lediglich die Schicht mit den Geschäftsobjekten »sehen«. Die gesamte Komplexität hinter den Geschäftsobjekten dringt nicht bis zu ihnen durch. Das primäre Entwurfsziel bei einem Multi-Tier-System ist deshalb, so viel wie möglich von dieser Komplexität vor den Client-Anwendungen, die die Präsentationsschicht bilden, zu verbergen.

Eine weitere leistungsfähige Funktion von COM ist, dass es Client-Anwendungen erlaubt, Objekte über das Netzwerk zu erzeugen und zu nutzen. Hinter den Kulissen verwendet COM das Protokoll RPC (Remote Procedure Call), um Methodenaufrufe über Prozess- und Hostgrenzen hinweg auszuführen. RPC ist das erste von mehreren wichtigen Protokollen, die Sie verstehen sollten, bevor Sie eine verteilte Anwendung mit COM+ entwickeln.

Wenn Sie mehrere Benutzer haben, die COM-Betriebssysteme einsetzen, beispielsweise Windows 2000, Windows NT oder Windows 98, können Sie ähnliche Multi-Tier-Anwendungen wie die in Abbildung 1.5 dargestellte erzeugen. Die Computer, auf denen die Client-Anwendungen laufen, verwenden COM, um über das LAN Verbindungen mit Geschäftsobjekten einzurichten. Nachdem eine Client-Anwendung Geschäftsobjekte über das Netzwerk erzeugt hat, kann sie sie nutzen, um Transaktionen auszuführen und Informationen zu ermitteln.

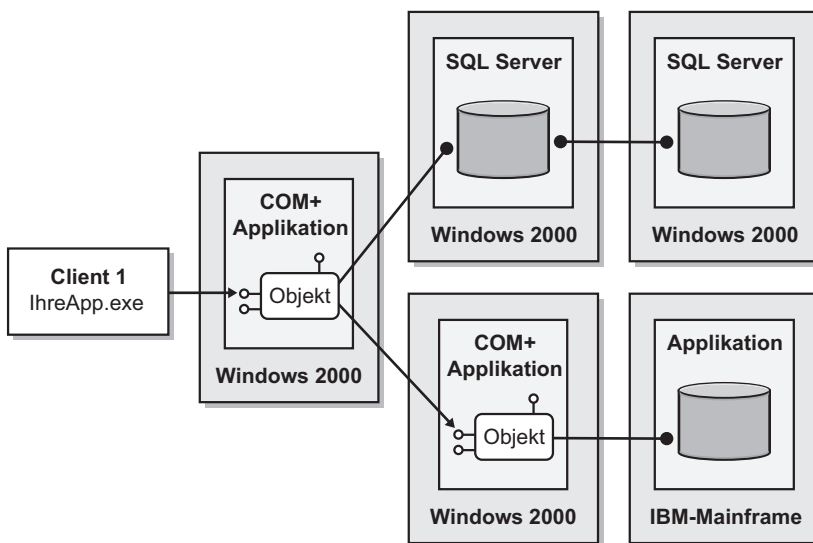


Abbildung 1.5: Eine der Stärken einer Multi-Tier-Architektur ist, dass sie die Komplexität der sich weiterentwickelnden IT-Infrastruktur eines Unternehmens vor den Client-Anwendungen verbirgt.

Dieser Ansatz ist zwar für einige Anwendungen gut geeignet, für andere dagegen unzureichend. Dieser Entwicklungsstil bedingt normalerweise, dass jeder Benutzer eine 32-Bit-Version von Windows einsetzt. Außerdem funktioniert er am zuverlässigsten, wenn all Ihre Client-Anwendungen innerhalb desselben LAN ausgeführt werden. Aber möglicherweise wollen Sie auch Benutzer erreichen, die nicht am lokalen Netzwerk angemeldet sind oder die mit anderen Betriebs-

systemen arbeiten, beispielsweise mit Macintosh, OS/2 oder UNIX. Mit Hilfe einer Web-basierten Entwicklungsstrategie erreichen Sie ein viel größeres Publikum.

Webbasierte Anwendungen

Die Beliebtheit des Internets und der Webbasierten Anwendungen hat die Akzeptanz der Multi-Tier-Architekturen in der Industrie beschleunigt. In einem Webbasierten System werden Client-Anwendungen innerhalb eines Browsers ausgeführt. Browser senden unter Verwendung eines einfachen Protokolls HTTP- (Hypertext Transfer Protocol) Anforderungen an Web-Server. Die Präsentationsschicht in einer Web-basierten Anwendung wird mit Hilfe von HTML (Hypertext Markup Language) realisiert.

Eine typische Web-Anwendung funktioniert eigentlich ganz einfach. Ein Client sendet eine Anforderung an einen Web-Server und der Web-Server reagiert, indem er die Anforderung verarbeitet und eine HTML-basierte Seite an den Benutzer zurückschickt. Das Wunderbare bei der Arbeit mit HTTP und HTML ist, dass sie auf jeder größeren Plattform unterstützt werden. Ihre Anwendungen könnten somit alle Benutzer im Internet erreichen.

Beachten Sie, dass der Web-Server den Eintrittspunkt des Clients in die mittlere Schicht darstellt, wie in Abbildung 1.6 gezeigt. Wenn ein Client eine HTTP-Anforderung einreicht, muss die Middle-Tier-Anwendung Geschäftsobjekte laden und ausführen, um die Vorteile der Multi-Tier-Entwicklung nutzen zu können. Während der Code und die HTML-Seiten, aus denen sich die Präsentationsschicht zusammensetzt, auf dem Server vorhanden sind, können Sie den Code für die Elemente der Benutzeroberfläche, die Geschäftslogik und den Datenzugriff dennoch separat halten. Wie Sie in späteren Kapiteln noch erfahren werden, können Sie eine Gruppe von Geschäftsobjekten erstellen, die von LAN-basierten Clients und Webbasierten Clients gemeinsam genutzt werden können.

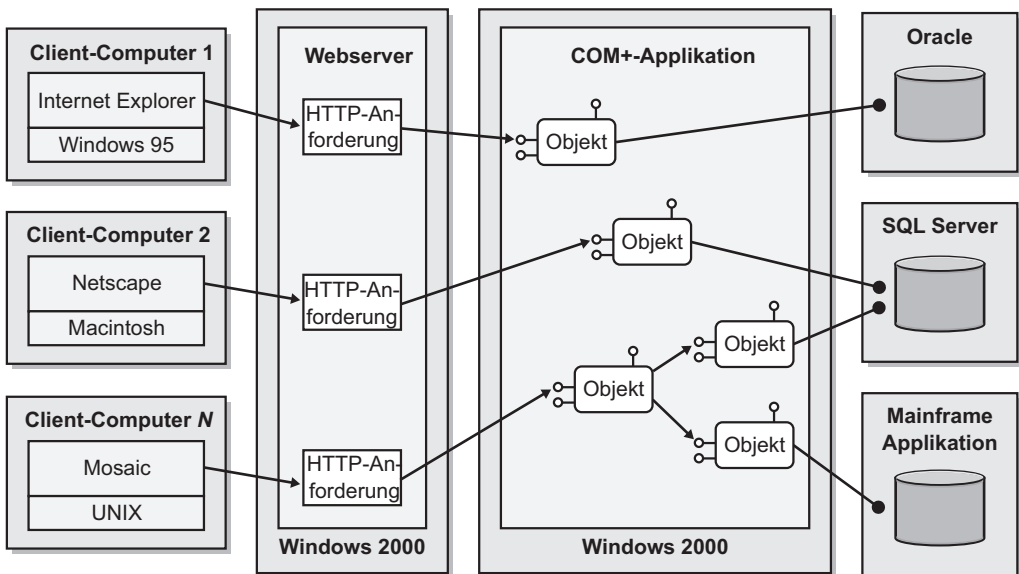


Abbildung 1.6: Die Webbasierte Entwicklung erlaubt es, Clients über das Internet zu erreichen, die die unterschiedlichsten Browser und Betriebssysteme einsetzen.

Wenn Sie eine Webbasierte Anwendung korrekt entwerfen, können Sie alle bisher beschriebenen Vorteile der Multi-Tier-Entwicklung nutzen. Sie können sehr großen Benutzergruppen prozessbezogene Ressourcen zur gemeinsamen Nutzung bereitstellen. Dazu zählen zum Beispiel Threads, Speicher oder Datenbankverbindungen. Sie können auf eine einzelne HTTP-Anforderung reagieren, indem Sie Daten aus mehreren Datenquellen auslesen und Transaktionen für diese Datenquellen ausführen lassen. Darüber hinaus reduziert die Webbasierte Entwicklung die clientseitigen Konfigurationsprobleme ganz wesentlich und eliminiert sie zum Teil.

Wenn Sie eine Webbasierte Anwendung entwickeln, müssen Sie entscheiden, welche Browser und Plattformen Sie unterstützen wollen. Wenn Ihr wichtigstes Ziel darin besteht, ein möglichst großes Publikum anzusprechen, sollten Sie eine reine HTML-Lösung wählen. Wenn Sie eine komplexere Benutzeroberfläche aufbauen wollen, müssen Sie Ihre Benutzer normalerweise auf einen bestimmten Browser oder auf mehrere der modernen Browser beschränken.

Funktionen wie DHTML (Dynamic HTML) und clientseitige JavaScripts werden beispielsweise von neueren Versionen des Microsoft Internet Explorers oder von Netscape Navigator unterstützt. Wenn Sie diese Funktionen nutzen, haben jedoch Benutzer mit älteren Browsern Probleme mit der Anzeige Ihrer Anwendung. In der Regel müssen Sie sich entscheiden, ob es wichtiger ist, ein größeres Publikum anzusprechen, oder ob Sie eine Anwendung mit interessanterer Oberfläche anbieten wollen. Eine wichtige Entscheidung, die Sie möglichst früh in der Entwurfsphase treffen sollten.

Messaging ist notwendig

Es gibt noch einen letzten Aspekt, den Sie in der ersten Entwurfsphase einer Multi-Tier-Anwendung berücksichtigen sollten. Viele der in verteilten Anwendungen genutzten Protokolle, wie beispielsweise RPC oder HTTP, sind synchron. Das bedeutet, die Anwendung, die auf dem Computer des Clients ausgeführt wird, muss nach jeder Anforderung geduldig warten, bis der Server die Anforderung bearbeitet hat. Wenn beim Server sehr viele Anforderungen anstehen, wird der Client dadurch möglicherweise gezwungen, unzumutbar lange zu warten.

Beachten Sie außerdem, dass Anwendungen, die unter Verwendung synchroner Protokolle entwickelt werden, davon abhängig sind, dass der Computer, der eine Anforderung verarbeitet, gleichzeitig mit ihnen online ist. Der Server muss online sein, damit der Client Anforderungen stellen kann, und die Clients müssen online sein und Anforderungen einreichen, um die vom Server angebotenen verfügbaren Verarbeitungszyklen zu nutzen. Geht der Server offline, können die Clients keine Anforderungen mehr einreichen. Wenn die Clients offline sind, ist der Server untätig. Es muss also jeder online sein, wenn das System als Ganzes funktionieren soll.

Häufig müssen mehrere Anwendungen asynchron und ohne eine Verbindung miteinander kommunizieren. Das *Messaging* ist ein Mechanismus, mit dem eine asynchrone Kommunikation zwischen zwei oder mehreren Anwendungen eingerichtet werden kann. Die Client-Anwendungen senden asynchrone Nachrichten über das Netzwerk. Die Nachrichten werden in Warteschlangen abgelegt. Jede dieser Nachrichten stellt normalerweise eine Client-Anforderung oder irgendeine Benachrichtigung dar. Server-Anwendungen überwachen diese Warteschlangen und verarbeiten die Nachrichten beim Eintreffen oder später. Nachrichten und Warteschlangen stellen einfach eine Referenzierungsebene zwischen den Anwendungen zur Verfügung, die die Anforderungen einreichen, und den Anwendungen, die diese Anforderungen verarbeiten.

Eine Anwendung, die auf einem Messaging-Protokoll aufsetzt, kann auch dann ausgeführt werden, wenn mehrere Computer offline sind. Es ist jedoch nicht ganz einfach, eine robuste Messaging-Infrastruktur zu implementieren, in der jede Nachricht garantiert an die Zielwarteschlange geschickt wird. Eine Messaging-Infrastruktur muss zahlreiche Untersysteme bereitstellen, um

Nachrichten transparent zu speichern, wenn Computer offline sind. Sie muss die Nachrichten außerdem später, wenn eine Verbindung eingerichtet werden kann, an deren Ziel weitergeben.

Die meisten Unternehmen wollen weder Zeit noch Geld investieren, um den Code für eine eigene Messaging-Infrastruktur zu entwickeln und zu debuggen. Glücklicherweise gibt es mehrere Messaging-Produkte, die die Vorteile einer asynchronen und verbindungslosen Kommunikation unterstützen, beispielsweise aus der IMB MQSeries-Reihe oder MSMQ (Microsoft Message Queuing Service). Es ist also nicht unbedingt erforderlich, eine eigene Messaging-Infrastruktur zu schaffen.

Die Entwicklung der Multi-Tier-Plattform von Microsoft

Wie ich bereits erwähnt habe, bieten Windows 2000 und COM+ eine Plattform für die Entwicklung von Multi-Tier-Anwendungen. Viele der Kerntechnologien, aus denen sich diese Plattform zusammensetzt, gibt es schon relativ lange. Sie sollten deshalb verstehen, wie sich die verschiedenen Komponenten der Plattform über die Jahre entwickelt haben.

Schon Anfang der 90er Jahre erkannten wichtige Entscheidungsträger bei Microsoft, dass bei der Entwicklung großer Multi-Tier-Systeme überdurchschnittlich viel Geld für die Infrastruktur aufzuwenden ist. Sie stellten fest, dass Unternehmen mehr Zeit aufwenden wollten, um ihre eigene Geschäftslogik zu codieren, anstatt komplexen Code für Aufgaben wie die gemeinsame Nutzung von Ressourcen der mittleren Schicht und die Überwachung verteilter Transaktionen zu schreiben. Die gesamte Multi-Tier-Strategie von Microsoft basiert auf der Annahme, dass die Unternehmen es lieber anderen überlassen, die generischen und doch wichtigen Komponenten eines Gerüsts für verteilte Anwendungen zu erstellen.

Mit der Weiterentwicklung der Multi-Tier-Strategie von Microsoft haben die Versuche, ihr einen Namen und eine konsistente Identität zu verleihen, zu einiger Verwirrung geführt. Die Marketingleute träumen davon, alle ein, zwei Jahre völlig neue Namen und Akronyme für Technologien einzuführen, die es schon längere Zeit gibt. Beispielsweise verwendet Microsoft als neuesten Namen für seine Plattform die Bezeichnung Windows DNA (Windows Distributed interNet Applications Architecture). Aus der Marketingperspektive betrachtet, benötigt Microsoft neue unverbrauchte Namen, um mit ähnlichen Technologien, wie zum Beispiel Corba und Enterprise JavaBeans, konkurrieren zu können. Aus Ihrer Perspektive als Entwickler hat diese Namensänderung keine besondere Bedeutung. Lassen Sie sich dadurch nicht verwirren. Das Rad wird nicht neu erfunden, sondern einfach in »Kreisförmige Fortbewegungseinheit« (KFE) umbenannt.

Jedes in diesem Buch angesprochene Thema fügt sich in den großen Bereich der Windows DNA ein. Ich werde nicht versuchen, alle DNA-bezogenen Technologien abzudecken, wie beispielsweise DHTML oder das clientseitige Skripting. Der Begriff *DNA* vereint in sich alles, was Microsoft je getan hat, um Unternehmen bei der Entwicklung von Multi-Tier-Anwendungen zu unterstützen. Viele Programmierer vermeiden den Begriff völlig, weil sie nicht mit jemandem aus der Marketingabteilung oder einem Manager aus einer der nicht technischen Abteilungen verwechselt werden wollen. Wenn Sie jedoch in Entwicklerkreisen wirklich mitreden möchten, müssen Sie alle wichtigen Technologien verstehen, aus denen sich die Plattform zusammensetzt.

Die Grundlage: COM

Die Multi-Tier-Strategie von Microsoft gründet auf einer Technologie namens COM (Component Object Model). COM bietet zahlreiche Vorteile, aber es handelt sich dabei um eine komplexe Technologie, die verschiedene anspruchsvolle Konzepte und jede Menge Low-Level-Details beinhaltet. Einige dieser Konzepte und viele der Details müssen Sie einfach verstehen, wenn Sie lernen wollen, korrekte Middle-Tier-Komponenten für eine COM+-Anwendung zu entwickeln. Viele andere Details von COM sind dagegen nicht mehr relevant oder nur für Programmierer wichtig, die Anwendungen der Präsentationsschicht entwickeln wollen. Sie brauchen sich nicht mit COM-bezogenen Themen wie OLE (Object Linking and Embedding), ActiveX-Steuerelementen oder Verbindungspunkten (wie beispielsweise Visual Basic-Ereignisse) zu beschäftigen, deshalb gehe ich in diesem Buch auch nicht darauf ein. Das Buch konzentriert sich ausschließlich auf die Details von COM, die für Programmierer wichtig sind, die nicht sichtbare Komponenten der mittleren Schicht entwickeln.

Der Begriff *COM* kann für unterschiedliche Betrachter unterschiedliche Bedeutungen haben. Einerseits handelt es sich um eine Spezifikation für die Entwicklung wiederverwendbarer Software, die in komponentenbasierten Systemen ausgeführt wird. Andererseits ist es eine komplexe Infrastruktur, die es Clients und Objekten erlaubt, über Prozess- und Computergrenzen hinweg miteinander zu kommunizieren. Viele Entwickler, die sich bereits mit COM beschäftigt haben, betrachten es als neuen Programmierstil und als Disziplin, die man beherrschen muss, wenn man in einer auf Microsoft ausgerichteten Umgebung arbeitet.

Das COM-Programmiermodell basiert auf der Verteilung des Klassencodes in binäre Komponenten. Das bedeutet, dass die Komponenten, die konform zu COM sind, ohne jegliche Abhängigkeiten vom Quellcode genutzt werden können. Die Entwickler können ihre Arbeit in Programmdateien weitergeben, ohne ihre eigenen Algorithmen preisgeben zu müssen. Die Wiederverwendung von Code in binärer Form eliminiert außerdem verschiedene Probleme, die beim Kompilieren auftreten, wenn Anwendungen unter Einsatz eines auf der Wiederverwendung von Quellcode basierenden Entwicklungsstils zusammengesetzt werden.

Bevor es die komponentenbasierten Technologien wie COM gab, wurden große Anwendungen entwickelt, indem hunderte und manchmal sogar tausende von Quelldateien in einem einzigen Batch-Job an einen Compiler geschickt wurden, um eine ausführbare Programmdatei daraus zu machen. Dieser Entwicklungsstil und die *monolithischen Anwendungen*, auf die er angewiesen ist, führen zu riesigen Programmdateien und langen Kompilierzeiten. Soll für eine Verbesserung eine einzige Zeile im Quellcode geändert werden, muss die gesamte Anwendung neu kompiliert werden. Damit wird es außerordentlich schwierig, Programmerteams zu koordinieren, die an der Entwicklung einer großen Anwendung zusammen arbeiten. Die Wartung und Erweiterung monolithischer Anwendungen ist bestenfalls schwierig.

Die komponentenbasierte Entwicklung löst viele der mit den monolithischen Anwendungen verbundenen Probleme. Sie erlaubt den Entwicklungsteams, anstelle des Quellcodes Programmdateien weiterzugeben. Binäre Komponenten können unabhängig voneinander aktualisiert und in der Produktionsumgebung ersetzt werden. Hierdurch wird es viel einfacher, eine Anwendung zu warten und zu erweitern, nachdem sie in die Produktion gegangen ist. Die meisten Personen sind sich einig, dass die Nutzung von COM oder einer anderen komponentenbasierten Technologie eine absolute Voraussetzung für die Entwicklung großer Informationssysteme ist.

COM basiert auf der objektorientierten Programmierung (OOP). Dies bedeutet, dass bei COM Clients mit Objekten kommunizieren. COM nutzt das OOP-Paradigma aus, um ein höheres Maß an Wiederverwendbarkeit und Wartungsfähigkeit zu unterstützen, als es mit anderen Modellen der Wiederverwendung möglich ist. COM-Clients und COM-Klassen werden in der Regel in se-

paraten Programmdateien abgelegt. COM definiert eine Infrastruktur, die es Clients erlaubt, Objekte zur Laufzeit anzulegen und zu binden.

Eine Plattform, die auf der objektorientierte Wiederverwendung von Komponenten basiert, muss einen Mechanismus bereitstellen, mit dem dynamisch Klassen geladen werden können. In dieser Hinsicht sind sich Java und COM ähnlich. Ein Client erzeugt Objekte zur Laufzeit, indem er eine bestimmte Klasse angibt, die in eine separate binäre Komponente kompiliert wurde. Ein vom System bereitgestellter Agent ermittelt den Klassencode, lädt ihn und erzeugt dann das Objekt für den Client.

Die binäre Wiederverwendung macht es viel einfacher, kleine Änderungen an Anwendungen vorzunehmen. Beispielsweise kann an einer DLL ein kleinerer Bug-Fix oder eine Leistungsanpassung vorgenommen werden. Die DLL kann dann neu kompiliert und in der Produktionsumgebung ersetzt werden, ohne sich negativ auf die Client-Anwendungen auszuwirken, die diese DLL verwenden. Systeme, die auf der Wiederverwendung von Quellcode basieren, müssen in der Regel jede Codezeile der gesamten Anwendung neu kompilieren, wodurch es mühsam und aufwendig wird, Software zu warten und zu erweitern.

Die Konzepte der binären Wiederverwendung erlauben Ihnen, COM-basierte Anwendungen unter Verwendung sprachunabhängiger Komponenten zu schreiben. Entwickeln mehrere Teams Komponenten für ein einzelnes System, kann jedes Team eine Programmiersprache unabhängig von den anderen Teams auswählen. Heute gibt es zahlreiche COM-fähige Sprachen, unter anderem C++, Visual Basic, Java, Delphi und sogar COBOL. Jedes Team wählt genau die Programmiersprache aus, die seinen Erfahrungen am besten entspricht und die beste Kombination aus Flexibilität, Leistung und Produktivität bietet.

Braucht ein Team beispielsweise Low-Level-Code, verwendet es der Kompatibilität halber C++. Ein anderes Team, das Code für die Geschäftslogik und den Datenzugriff derselben Anwendung entwickelt, könnte beispielsweise Visual Basic einsetzen, um eine möglichst hohe Produktivität zu erzielen. Die Möglichkeit, Sprachen zu kombinieren, macht es den Unternehmen einfach, die vorhandenen Programmier Talente bestmöglich zu nutzen.

Schnittstellenbasierte Programmierung

Die Ingenieure bei Microsoft trafen beim Entwurf von COM viele wichtige Entscheidungen für die Architektur. Eine der weitreichendsten Entscheidungen war jedoch die Forderung, dass Schnittstellen unter COM formal von der Implementierung getrennt werden sollten. Das bedeutet, dass COM auf dem Konzept der *schnittstellenbasierten Programmierung* basiert.

Das Konzept der schnittstellenbasierten Programmierung war keine clevere neue Idee der Ingenieure bei Microsoft. Die Informatiker an den Universitäten hatten diesen Programmierstil, ebenso wie Organisationen, die große, erweiterbare Anwendungen entwickeln mussten, bereits eingesetzt. Die schnittstellenbasierte Programmierung wurde zuerst in Sprachen wie C++ und Smalltalk eingeführt, die keine formale Unterstützung des Konzepts einer separaten, eigenständigen Schnittstelle ausweisen. Die modernen Sprachen und Werkzeuge, wie etwa Java und Visual Basic, besitzen eine integrierte Unterstützung für diesen Programmierstil. Während also Microsoft nicht für die Idee belohnt werden kann, sollte das Unternehmen honoriert werden, weil es die Eleganz der schnittstellenbasierten Programmierung erkannt und als Eckpfeiler für COM genutzt hat.

Eine Schnittstelle ist wie eine Klasse ein bestimmter Datentyp. Sie definiert verschiedene öffentliche Methoden, implementiert diese aber nicht. Eine Schnittstelle definiert in gewisser Hinsicht ein sehr spezifisches Protokoll für die Kommunikation, die zwischen einem Client und einem Objekt stattfindet. Diese Abkopplung der Schnittstelle von den Klassen, die sie implementieren,

gibt den Autoren die Freiheit, vieles zu tun, was andernfalls unmöglich wäre. Ein Entwickler, der eine Client-Anwendung für eine Schnittstellendefinition schreibt, vermeidet Abhängigkeiten von Klassendefinitionen. Kapitel 2 beschreibt einige der wichtigsten Konzepte der schnittstellenbasierten Programmierung. Möglicherweise ist es eines der schwierigsten Kapitel in diesem Buch, aber Sie sollten die dort beschriebenen Konzepte verstehen, um nachvollziehen zu können, wie und warum COM so funktioniert, wie es funktioniert.

DCOM – Distributed COM

COM wurde von Anfang an darauf ausgelegt, die Prozessgrenzen zu überschreiten. Die frühen Versionen von COM erlaubten dies nur, wenn der Client-Prozess und der Server-Prozess auf einem einzigen Computer ausgeführt wurden. Mit der Einführung von Microsoft Windows NT 4.0 wurde COM erweitert, so dass diese Interprozesskommunikation nun über Computergrenzen hinweg möglich war. Microsoft entwickelte ein neues Übertragungsprotokoll für COM, womit es möglich wurde, verteilte Anwendungen in einer LAN-Umgebung einzusetzen. Das war ein wichtiger Meilenstein für die Strategie, die Microsoft in Hinblick auf die Programmierung in großen Unternehmen verfolgt.

Zunächst suchte man bei Microsoft nach einem Marketingbegriff, der deutlich machte, dass COM letztlich genutzt werden konnte, um Objekte über das Netzwerk zu erzeugen. Dem Namen DCOM (Distributed COM, Verteiltes COM) wurde gegenüber Network OLE bei der Veröffentlichung von Windows NT 4 im August 1995 der Vorzug gegeben. Heute verwenden die Entwickler keinen der beiden Begriffe mehr. *DCOM (Distributed COM)* ist momentan der richtige Begriff für das Übertragungsprotokoll von COM, obwohl viele Entwickler glauben, COM selbst sei eine verteilte Technologie und die Angabe von *Distributed* vor *COM* damit redundant.

Wie bereits erwähnt, basiert die Unterstützung, die COM für verteilte Anwendungen bietet, auf einem Interprozessmechanismus namens RPC (Remote Procedure Call). RPC ist ein Industriestandard, der für die unterschiedlichsten Plattformen weiterentwickelt wurde. Microsoft hat RPC mit objektorientierten Erweiterungen ausgestattet, um COM entgegenzukommen. Die resultierende Implementierung auf der Windows-Plattform wird auch als Object RPC (ORPC) bezeichnet.

COM und RPC leben in Symbiose. COM bietet RPC eine objektorientierte Umgebung, und RPC bietet COM die Möglichkeit, Objekte über das Netzwerk zu bedienen. In Kapitel 3 werden wir untersuchen, wie COM RPC nutzt, um Methoden transparent auf Remote-Computern aufzurufen, indem es Anforderungs- und Antwortpakete zwischen den Clients und den Objekten überträgt. Sie erfahren außerdem, wie die Clients ihre Methoden dennoch wie gewohnt aufrufen, als würden die Objekte unmittelbar zur Verfügung stehen. Es ist bemerkenswert, wie es die Architekten von COM geschafft haben, so viele der für RPC erforderlichen Details vor den Programmierern der mittleren Schicht und denen der Client-Seite zu verbergen.

Von COM zu MTS

Die Veröffentlichung von MTS (Microsoft Transaction Server) war ein wichtiger Meilenstein bei der Entwicklung der Plattform. MTS ist eine für Windows NT Server entwickelte Software. MTS erlaubt Geschäftsobjekten, die unter Windows NT Server ausgeführt werden, verteilte Transaktionen von der mittleren Schicht aus auszuführen und zu steuern. MTS ist jedoch sehr viel mehr als ein Transaktionsmonitor; es bietet eine brandneue Laufzeitumgebung für COM-Objekte, die auf der mittleren Schicht ausgeführt werden. MTS hat wichtige Infrastrukturkomponenten eingeführt, die in COM nicht enthalten waren. Insbesondere unterstützt es verteilte Transaktionen, in-

tegrierte Sicherheit sowie ein Thread-Pooling und bietet außerdem eine verbesserte Konfiguration und Administration.

Der Name MTS hat für Verwirrung gesorgt, weil die Software nicht nur für Unternehmen geschaffen wurde, die verteilte Transaktionen ausführen lassen möchten. MTS bot eine Möglichkeit für Microsoft, die nächste Generation seiner Umgebung für verteilte Anwendungen zu verkaufen. Middle-Tier-Objekte, die auf Windows NT Server ausgelegt waren, sollten in der MTS-Umgebung ausgeführt werden. Dies sollte unabhängig davon geschehen, ob diese Objekte Bestandteil von Transaktionen waren. Die Entwickler waren verwirrt, als sie erfuhren, dass sie MTS auch für Objekte einsetzen konnten, die nicht Teil von Transaktionen sind. Um diese Verwirrung unter Windows 2000 aufzuheben, hat Microsoft den Namen der Middle-Tier-Laufzeitumgebung von MTS in COM+ geändert. Die Transaktionsunterstützung, die für MTS entwickelt wurde, ist auch in COM+ enthalten. Wenn Sie bereits wissen, wie man mit MTS Transaktionen programmiert, müssen Sie nicht mehr viel dazulernen, um Transaktionskomponenten für COM+ zu entwickeln. Hier funktioniert alles ähnlich. Die Programmierung von Transaktionen mit COM+ wird in Kapitel 8 beschrieben.

Neben der Unterstützung verteilter Transaktionen erweitert MTS das Sicherheitsmodell von COM. Die MTS-Sicherheit basiert auf dem Konzept von *Rollen*. Eine Rolle ist eine Abstraktion, die ein Sicherheitsprofil für einen oder mehrere Benutzer in einer MTS-Anwendung darstellt. Zur Entwurfszeit kann ein Entwickler Sicherheitsüberprüfungen ausführen, indem er die Rollen in deklarativer oder programmgesteuerter Weise einsetzt. Zur Laufzeit weist ein Administrator Benutzerkonten und Gruppenkonten in einer Windows NT-Domäne Rollen zu. Das rollenbasierte Sicherheitsmodell von MTS bietet mehr Flexibilität und erlaubt eine feinere Strukturierung als das ursprüngliche Sicherheitsmodell von COM.

Eine weitere wichtige Funktion, um die MTS die Plattform ergänzt hat, war ein Schema für die Verwaltung der Parallelität, indem hinter den Kulissen ein Thread-Pooling eingeführt wurde. MTS führte eine Abstraktion ein, eine so genannte *Aktivität*, die einen logischen Ausführungs-Thread in einer MTS-Anwendung darstellt. MTS-Programmierer sollten ihre Anwendungen so schreiben, dass pro Client-Anwendung eine Aktivität bereitgestellt wird. Die MTS-Laufzeitumgebung bindet logische Aktivitäten automatisch an physische Threads. Nachdem die Anzahl der Clients eine gewisse Schwelle erreicht hat, beginnt die MTS-Laufzeitumgebung, einzelne Threads zwischen mehreren Clients zu teilen.

Multi-Tier-Anwendungen, die ein Thread-Pooling-Schema verwenden, sind besser skalierbar als diejenigen, die ein Modell mit einzelnen Threads oder einem Thread pro Client verwenden. Ein Modell mit einem einzigen Thread verhindert jede Möglichkeit der Parallelität, weil es nicht gleichzeitig Methoden für zwei oder mehrere Clients ausführen kann. Ein Modell, das jeweils einen Thread pro Client erzeugt, führt bei größeren Applikationen zu einer nicht akzeptablen Ressourcenverwendung, weil immer wieder physische Threads erzeugt und entsorgt werden müssen. Das Ziel eines Thread-Pooling-Schemas wie dem in MTS implementierten ist es, einen optimalen Ausgleich zwischen höheren Ebenen der Parallelität und einer effizienteren Ressourcennutzung zu schaffen. In dieser Hinsicht leistet MTS ganze Arbeit und bringt alles unter einen Hut.

Eine weitere wichtige Funktion von MTS war eine verbesserte Unterstützung der Computerkonfiguration und Netzwerkverwaltung. Mit COM war es kompliziert und aufwendig, größere COM-basierte Anwendungen in einer Netzwerkumgebung einzusetzen und zu verwalten. Die Administrationswerkzeuge von MTS machten es viel einfacher, die Server-Computer, auf denen die Middle-Tier-Objekte ausgeführt werden, zu konfigurieren und zu verwalten. Anders als COM erlaubt MTS, mehrere Server-Computer von einem einzigen Desktop aus zu verwalten. Darüber

hinaus stellt MTS die Werkzeuge zur Verfügung, mit denen clientseitige und serverseitige Setup-Programme entwickelt werden.

Attributbasierte Programmierung und Abfrage

MTS hat ein wichtiges Konzept in das Programmiermodell der Plattform eingeführt: die attributbasierte Programmierung. Die von der Plattform angebotenen Dienste werden über zu deklarierende Attribute bereitgestellt. Die Attributwerte werden von den Programmierern zur Entwurfszeit festgelegt und können von den Administratoren neu konfiguriert werden, nachdem eine Anwendung in die Produktion übernommen wurde. Die attributbasierte Programmierung unterscheidet sich ganz wesentlich von der Methode, nach der die meisten Betriebssysteme in der Vergangenheit ihre Systemdienste bereitgestellt haben.

Ursprünglich haben die Betriebssysteme ihre Dienste über eine Funktionsmenge bereitgestellt, die aus einer API (Application Programming Interface) aufgerufen werden konnten. In diesem Modell unterstützt eine Anwendung Systemdienste durch explizite Aufrufe der API. Das bedeutet, Sie müssen explizite Aufrufe auf Systemebene in Ihrer Anwendung kompilieren. Wenn Sie die Nutzung eines Systemdienstes ändern wollen, nachdem eine Anwendung bereits in Produktion gegangen ist, müssen Sie Ihren Code abändern und die Anwendung neu kompilieren. Ein Programmiermodell, das auf Attributen basiert, die deklariert werden können, ist sehr viel flexibler.

Betrachten wir ein Beispiel, das verdeutlichen soll, wie deklarative Attribute funktionieren. Eine Komponente in einer MTS-Anwendung hat ein Attribut, das der MTS-Laufzeitumgebung mitteilt, ob sie Transaktionen unterstützt. Wenn ein Programmierer eine Komponente markiert, um damit anzuzeigen, dass sie eine Transaktion bedingt, werden Objekte, die aus der Komponente instanziiert werden, innerhalb des Kontexts einer logischen MTS-Transaktion erzeugt. Wenn ein Client eines dieser Transaktionsobjekte aufruft, führt die MTS-Laufzeitumgebung automatisch einen Aufruf durch, um eine physische Transaktion zu starten. Außerdem initiiert sie den entsprechenden Aufruf, um ein Commit oder ein Rollback für die Transaktion auszuführen. Auf diese Weise ist es für die Programmierer nie erforderlich, explizit Transaktionen zu starten, festzuschreiben oder abzubrechen. Alle erforderlichen Aufrufe werden von der MTS-Laufzeitumgebung hinter den Kulissen vorgenommen.

Warum ist das attributbasierte Programmiermodell besser als das ältere, prozedurale Modell, für das explizite API-Aufrufe erforderlich sind? Erstens ist für das neue Modell weniger Code erforderlich. Sie legen zur Entwurfszeit Ihre Vorgaben fest und die zugrunde liegende Laufzeitumgebung stellt sicher, dass Ihre Anforderungen erfüllt werden. Ein zweiter, weniger offensichtlicher Grund ist, dass die Administratoren auf diese Weise eine Anwendung, die einen Systemdienst nutzt, ganz einfach neu konfigurieren können, auch nachdem diese in die Produktion übernommen wurde. Es ist nicht erforderlich, Code abzuändern oder die Anwendung neu zu kompilieren.

Das attributbasierte Programmiermodell von MTS basiert auf einem Mechanismus, den man als *Interception* oder "Abfangen" bezeichnet. Wenn ein Client ein Objekt aus einer Komponente erzeugt, die in einer MTS-Anwendung konfiguriert wurde, fügt die zugrunde liegende Laufzeitumgebung eine Abfangschicht ein, wie in Abbildung 1.7 gezeigt. Diese Abfangschicht stellt eine Verbindung dar, die es der MTS-Laufzeitumgebung erlaubt, die vom System bereitgestellte Vor- und Nachverarbeitung von Methodenaufrufen auszuführen. Das System übernimmt die Kontrolle unmittelbar bevor und unmittelbar nachdem ein Objekt den Code für eine der entsprechenden Methoden ausgeführt hat.



Abbildung 1.7: Ein attributbasiertes Programmiermodell nutzt den Abfangmechanismus, um die Systemdienste zu unterstützen.

COM oder MTS?

Eines der größten Probleme der Programmierer, die Multi-Tier-Anwendungen für Windows NT Server entwickeln, ist die Entscheidung, ob sie COM oder MTS verwenden sollen. Viele Unternehmen haben sich für MTS entschieden, weil es viele eingebaute Dienste bietet, die es in COM nicht gibt. Andere Unternehmen verwenden COM, weil sie die Verbesserungen der MTS-Laufzeitumgebung nicht verstehen.

COM wird zusammen mit Windows NT Server ausgeliefert, MTS dagegen nicht. Sie müssen also eine zusätzliche Software installieren, das Windows NT Option Pack. Das bedeutet, MTS ist nicht wirklich Teil von COM. COM hat ein eigenes Programmiermodell und eine Laufzeitebene; MTS hat ein anderes Programmiermodell und eine separate Laufzeitebene.

Sie sollten verstehen, dass MTS einfach eine Ebene ist, die auf COM aufgesetzt wurde. COM wurde nie abgeändert, um MTS zu unterstützen. Was bedeutet das also? Die beiden Laufzeitebenen sind nicht so eng miteinander verknüpft, wie sie es sein könnten. Auf diese Weise entsteht eine Plattform, die ein gewisses Maß an Ineffizienz, Mehrdeutigkeit und Verwirrung mit sich bringt. Für die MTS-Programmierer ist alles ein wenig kompliziert, weil sie beide Programmiermodelle kennen müssen. Darüber hinaus funktionieren viele gültige COM-Programmiertechniken nicht korrekt, wenn sie in einer MTS-Anwendung eingesetzt werden.

Von COM und MTS zu COM+

Als die Plattformarchitekten begannen, Änderungen und Verbesserungen für Windows 2000 zu planen, war etwas ganz offensichtlich: COM und MTS mussten in einer einzigen Laufzeitebene und in einem einzigen Programmiermodell zusammengefasst werden. Das war kein triviales Unterfangen, aber es war den Aufwand wert.

Mit der Veröffentlichung von Windows 2000 wurden die besten Konzepte aus COM und MTS in einer neuen Laufzeitumgebung realisiert, COM+. Anders als MTS ist diese neue Laufzeitebene nicht optional. COM+ ist Teil der Standardinstallation von Windows 2000. Die gute Nachricht ist also, dass unter Windows 2000 keine Entscheidung mehr zwischen COM und MTS erforderlich ist. Darüber hinaus ist es viel einfacher, Komponenten für COM+ als für MTS zu entwickeln, weil viele der müßigen Eigenheiten von MTS weggefallen sind.

Wie COM basiert COM+ auf binären Komponenten und der schnittstellenorientierten Programmierung. Methodenaufrufe werden unter Verwendung einer transparenten RPC-Ebene über Prozess- und Computergrenzen hinweg vorgenommen. Und wie COM-Komponenten können COM+-Komponenten aktualisiert und erweitert werden, während sie sich in der Produktionsumgebung befinden, ohne dass die Client-Anwendungen, die sie verwenden, dadurch beeinträchtigt werden.

Wie MTS unterstützt COM+ verteilte Transaktionen und eine rollenbasierte Sicherheit. Es stellt ein integriertes Thread-Pooling-Schema bereit, das so transparent ist wie das von MTS. Das Programmiermodell von COM+ verwendet auch das Konzept der Interception, um den Entwicklern über deklarative Attribute Plattformdienste zur Verfügung zu stellen. COM+ treibt die attributbasierte Programmierung noch einen Schritt weiter als MTS. Neben Transaktionsdiensten und

integrierter Sicherheit unterstützt COM+ Dienste wie das Anlegen benutzerdefinierter Objekte, die Synchronisierung sowie ein Objekt-Pooling. Andere neue COM+-Funktionen, wie etwa Queued Components (Komponenten in Warteschlangen) und COM+-Ereignisse, werden über konfigurierbare Attribute bereitgestellt.

Konfigurierte oder nicht konfigurierte Komponenten

Wenn Sie wollen, dass Ihre Komponenten die COM+-Dienste nutzen, müssen Sie sie in einer COM+-Anwendung installieren. Wird eine Komponente in einer COM+-Anwendung installiert, erhält sie ein Profil in einem Systemkatalog, der COM+-Registrierungsdatenbank (RegDB). Dieser Katalog enthält die Einstellungen für die konfigurierbaren Attribute der Komponenten sowie der COM+-Anwendungen. Eine Komponente, die in einer COM+-Anwendung installiert wurde, wird auch als *konfigurierte Komponente* bezeichnet, da sie COM+-Attributeinstellungen aufweist.

Sie werden bisweilen auch einem älteren Komponententyp begegnen, der keine COM+-Attributeinstellungen besitzt: die nicht konfigurierte Komponente. Nichtkonfigurierte Komponenten sind nicht in COM+-Anwendungen installiert. Stattdessen werden sie so registriert, wie es unter früheren Versionen vom COM üblich war. Nichtkonfigurierte Komponenten können die Vorteile der COM+-Dienste nicht nutzen, aber sie können in anderen Umgebungen als MTS und COM+ ausgeführt werden.

Wenn Sie Komponentencode für eine COM+-Anwendung entwickeln, erzeugen Sie in der Regel konfigurierte Komponenten. Dieser Ansatz erlaubt Ihnen, verschiedene Plattformdienste zu nutzen. Wenn Sie Ihren Code schreiben, werden Sie sehr wahrscheinlich auch nichtkonfigurierten Komponenten begegnen. Beispielsweise setzt sich die ADO-Bibliothek (ActiveX Data Objects) aus nichtkonfigurierten Komponenten zusammen. ADO-Objekte können in einer COM+-Anwendung ausgeführt werden, aber anders als konfigurierte Komponenten können sie auch in Anwendungen eingesetzt werden, die auf früheren Versionen von COM basieren.

COM+-Dienste verstehen

Neben der Laufzeitumgebung beinhalten COM+ und Windows 2000 auch mehrere integrierte Dienste, die für die Programmierer von Multi-Tier-Anwendungen wichtig sind. Einige Multi-Tier-Anwendungen müssen nur einen oder zwei dieser Dienste verwenden. Andere Applikationen benötigen vielleicht alle Dienste. Sie sollten sich ein allgemeines Verständnis verschaffen, wie sich all diese Teile zusammenfügen, so dass Sie in der Entwurfsphase kontrollierte Entscheidungen treffen können. Die folgenden Abschnitte bieten eine kurze Übersicht über die wichtigsten Dienste für verteilte Anwendungen der Plattform.

Internet Information Services (IIS)

IIS (Internet Information Services) ist der Web-Server von Microsoft. Dieses Produkt wurde ursprünglich für Windows NT Server entwickelt. Die neueste Version ist IIS 5, das zusammen mit Windows 2000 ausgeliefert wird. Wie andere Web-Server-Produkte verarbeitet IIS eingehende HTTP-Anforderungen, die von Client-Anwendungen gesendet werden. Frühere Versionen von IIS wurden hauptsächlich zur Unterstützung statischer Web-Seiten verwendet. Heute unterstützen viele webbasierte Anwendungen IIS, um für jede eingehende Anforderung eine genau darauf abgestimmte Verarbeitung vornehmen zu können.

IIS stellt ein eigenes API bereit, das Internet Server API (ISAPI), das von Entwicklern genutzt wird, die Web-Anwendungen mit einer benutzerdefinierten serverseitigen Logik erstellen wollen. Programmierer, die direkt für ISAPI programmieren, erzeugen Software-Module, die auch als *ISAPI-Erweiterungen* oder *ISAPI-Filter* bezeichnet werden. ISAPI-Software bietet die bestmögliche Performance und Flexibilität, weist aber einige wenige signifikante Nachteile auf. Sie muss mit C oder C++ anstelle von Visual Basic entwickelt werden und zwingt die Programmierer in der Regel, sich mit Low-Level-Infrastrukturdetails, wie zum Beispiel das Erstellen eines Thread-Pool-Managers, zu beschäftigen.

Viele Unternehmen möchten nicht direkt auf die ISAPI zugreifen, da die notwendige Erfahrung fehlt oder sie nicht gewillt sind, die hierfür erforderliche Zeit und das nötige Geld zu investieren. Der IIS bietet deshalb eine Alternative zur ISAPI an, die mit Active Server Pages (ASP) bezeichnet wird. Bei der ASP-Umgebung handelt es sich ebenfalls um eine ISAPI-Erweiterung, die es den Programmierern unter anderem erlaubt, mit Hilfe von Skriptingsprachen und Visual Basic eine serverseitige Logik zu entwickeln.

Beachten Sie, dass Web-Anwendungen, die mit Hilfe von ISAPI oder ASP entwickelt wurden, reine HTML-Lösungen bieten können, falls Sie das wollen. Sie treffen die Entscheidung, welche Browser Ihre Anwendung unterstützt. Sie können eine reine HTML-Lösung erstellen, die eine größere Benutzerbasis unterstützt, beispielsweise im Internet. Sie haben jedoch ebenfalls die Möglichkeit, die Stärken eines Browsers auszunutzen, wie zum Beispiel die des Internet Explorers, wenn Sie Ihre Anwendung nur in einem Intranet einsetzen.

Viele Unternehmen haben komplexe Sites aufgebaut und verwenden dafür ausschließlich ASP. Es ist ganz einfach, eine serverseitige Geschäftslogik und den Datenzugriffcode mit Hilfe von Skriptingsprachen wie Microsoft Visual Basic Scripting Edition (VBScript) und JavaScript sowie einem Entwicklungswerkzeug wie Microsoft Visual InterDev zu schreiben. Die Unternehmen haben jedoch auch erkannt, dass es schwierig ist, Logik wiederzuverwenden, zu warten und zu erweitern, die über mehrere Seiten verteilt ist. Das wird offensichtlich, sobald die Site größer wird. Ein besserer Ansatz ist es, die Geschäftslogik und den Datenzugriffcode in kompilierte Komponenten einzubetten.

Die Integration zwischen ASP und COM+ macht es ganz einfach, benutzerdefinierte Geschäftsobjekte auf einer ASP-Seite zu erzeugen und auszuführen. Das bedeutet, Sie können einen Großteil Ihrer Server-Logik mit Hilfe von Komponenten verteilen, statt die in ASP-Seiten eingebetteten Skripts nutzen zu müssen. Durch die Verwendung von Komponenten wird es einfacher, Ihren Code wiederzuverwenden, zu warten und zu erweitern. Solche Komponenten können mit Visual Basic erzeugt werden, das bessere Funktionen für das Testen und das Debugging bereitstellt als jedes ASP-basierte Entwicklungswerkzeug. Viele Unternehmen haben festgestellt, dass die Nutzung von ASP in Kombination mit Visual Basic die beste Abwägung zwischen Produktivität, Wartbarkeit und Leistung bietet.

Microsoft Message Queuing Services (MSMQ)

MSMQ (Microsoft Message Queuing Services) ist ein weiterer wichtiger Bestandteil der Plattform. MSMQ ist ein Middleware-Dienst, der den Nachrichtenaustausch zwischen verschiedenen Prozessen einer Multi-Tier-Anwendung vereinfacht. Wie bereits erwähnt, ist dieser Nachrichtenaustausch wichtig, weil er eine asynchrone und verbindungslose Kommunikation erlaubt, die weder RPC noch HTTP bereitstellen können.

MSMQ ist ein Messaging-Produkt, das auf der asynchronen Weitergabe von Nachrichten in benannte Warteschlangen basiert. Auf oberster Ebene bilden Nachrichten Prozeduraufrufe

zwischen einem Client und einem Server nach, außer dass hier jeder der Beteiligten in Abwesenheit des anderen seine Arbeit dennoch fortsetzen kann. Der größte Unterschied in Hinblick auf das Konzept ist, dass eine Nachricht nur in einer Richtung weitergegeben wird, während ein COM-Methodenaufruf sowohl eine an das Objekt gesendete RPC-Anforderung als auch eine an den Client zurückgeschickte RPC-Antwort beinhaltet.

Mit Hilfe von MSMQ kann eine Client-Anwendung ihre Nachrichten mit den Anforderungen auch dann senden, wenn die Server-Anwendung offline ist. Außerdem bedeutet es, dass der Server auf Anforderungsnachrichten reagieren kann, nachdem alle Client-Anwendungen offline gegangen sind. In Umgebungen, wo Client-Anwendungen und Server aus den unterschiedlichen Gründen die Verbindung zueinander verlieren, erlaubt diese Möglichkeit der verteilten Anwendung, aktiv zu bleiben und ausgeführt zu werden.

Wann sollten Sie MSMQ einsetzen? Wir betrachten hier ein typisches Beispiel. In einer Anwendung für Bestellungen können die Kunden Bestellungen in eine Bestellwarteschlange stellen, auch wenn die Server-Anwendung nicht läuft. Später kann die Server-Anwendung die Warteschlange öffnen und die Aufträge verarbeiten. MSMQ erlaubt es der Server-Anwendung sogar, eine Antwortnachricht an den Sender zu schicken, als handelte es sich dabei um die Antwort auf einen Methodenaufruf. Für das Messaging ist dazu nur ein bisschen mehr Arbeit erforderlich.

MSMQ ist auch praktisch für Unternehmen, in denen Angestellte Laptops verwenden, die ständig an das Netzwerk angeschlossen und davon getrennt werden. Mit Hilfe von MSMQ ist es ganz einfach, Client-Anwendungen zu entwickeln, die Nachrichten in eine Warteschlange im Netzwerk schicken. Geht ein Laptop offline, speichert MSMQ die Nachrichten automatisch in einer temporären lokalen Warteschlange. Wird der Laptop wieder an das Netzwerk angeschlossen, erkennt MSMQ, dass das Netzwerk zur Verfügung steht und gibt die zwischengespeicherten Nachrichten an die entsprechende Warteschlange weiter. Wie Sie sehen, wurden die wichtigsten Aspekte eines Speicher- und Weitergabe-Mechanismus problemlos in die zugrunde liegende Plattform integriert.

Messaging-Produkte wie MSMQ können auch bessere Weitergabegarantien bieten als Produkte, die auf RPC oder HTTP basieren. Eine MSMQ-Nachricht kann innerhalb des Gültigkeitsbereichs einer Transaktion verschickt werden, so dass die Weitergabesemantik *genau einmal* realisiert wird. Das bedeutet, MSMQ ergreift zahlreiche Vorsichtsmaßnahmen, um sicherzustellen, dass Ihre Nachrichten irgendwann ihr Ziel erreichen. Außerdem bieten sie eine zusätzliche Infrastrukturunterstützung, um Fehlerbenachrichtigungen an den Sender zurückzuschicken, wenn für die Nachrichten ein Timeout auftritt oder sie nicht an ihr Ziel weitergeleitet werden können. Sie können sicher sein, dass MSMQ die Nachricht an ihr Ziel weitergibt oder Sie informiert, falls die Nachricht nicht ausgeliefert werden kann. MSMQ eliminiert viele der Probleme, die durch verloren gegangene Nachrichten entstehen, mit denen Anwendungen zu kämpfen haben, die auf RPC und HTTP basieren.

Queued Components – Warteschlangenkompontenten

MSMQ wurde zuerst im November 1997 für Windows NT Server angeboten, nämlich mit Veröffentlichung des Windows NT Option Pack. Seitdem können Visual Basic-Programmierer MSMQ nutzen, um die Vorteile einer asynchronen, verbindungslosen Kommunikation zu genießen. Für die MSMQ-Programmierung ist jedoch zusätzlicher Code erforderlich, um Nachrichten in Client-Anwendungen zu erzeugen, vorzubereiten und zu versenden. Außerdem müssen Sie eine serverseitige Listener-Anwendung schreiben. Verglichen mit der Kommunikation durch COM-Methoden bedingt die Kommunikation mit MSMQ wesentlich mehr Arbeit, um eine Anforderung abzusetzen und eine Antwort zu erhalten.

COM+ stellt einen Dienst namens Queued Components bereit, der es Ihnen ermöglicht, die Vorteile von MSMQ zu nutzen, ohne explizit auf die MSMQ-API zugreifen zu müssen. Der Dienst Queued Components ist einfach eine Produktivitätsschicht, die auf MSMQ aufsetzt. Sie bearbeiten die Warteschlangenkomponenten fast genauso wie die standardmäßigen COM++-Komponenten, mit einer oder zwei Einschränkungen. Beispielsweise können Sie keine Methoden mit Ausgabeparametern oder Rückgabewerten entwickeln.

Nachdem Sie eine Warteschlangenkomponente in einer COM++-Anwendung erzeugt und installiert haben, müssen Sie ein Attribut konfigurieren, das anzeigt, dass die von Ihnen bereitgestellte Schnittstelle für eine Warteschlange vorgesehen ist. In diesem Fall erzeugt der Dienst Queued Components automatisch eine eigene Warteschlange für die Anwendung. Außerdem richtet er einen vom System bereitgestellten Listening-Dienst ein, um eingehende Nachrichten beim Eintreffen zu verarbeiten.

Nachdem die Warteschlangenkomponente auf dem Server korrekt konfiguriert wurde, können Sie eine Client-Anwendung schreiben, die die Komponente nutzt. Eine Client-Anwendung instanziert die Objekte nicht direkt aus einer Warteschlangenkomponente. Stattdessen legt sie ein spezielles clientseitiges Proxy-Objekt an, einen so genannten *Recorder*, der aus der Sicht des Clients wie das eigentliche Objekt erscheint und sich ebenso verhält. Der Client beginnt dann wie gewohnt mit Methodenaufrufen. Der Dienst Queued Components stellt eine Infrastruktur für die Aufzeichnung dieser Methodenaufrufe in einer MSMQ-Nachricht bereit, ebenso wie für die Übertragung dieser Daten über das Netzwerk an den Computer, auf dem die Warteschlangenkomponente installiert ist. Der serverseitige Teil des Diensts Queued Components nimmt die Nachricht entgegen, erzeugt eine Instanz der Warteschlangenkomponente und »spielt« die Methodenaufrufe »ab«.

Wie Sie sehen, ist das wichtigste Entwurfsziel von Queued Components, die Bequemlichkeit von COM-Methodenaufrufen mit den Vorteilen einer asynchronen, verbindungslosen Kommunikation zu kombinieren. Im Wesentlichen verwenden Warteschlangenkomponenten MSMQ als zugrunde liegendes Transportprotokoll anstelle von RPC. Warteschlangenkomponenten können also viele der Beschränkungen eines verbindungsorientierten, synchronen Protokolls kompensieren.

Sie brauchen den Dienst Queued Components nicht, um die Vorteile von MSMQ zu nutzen; Sie können in Ihren Programmen auch direkt auf MSMQ zugreifen. Der Dienst Queued Components ist eine produktivitätsorientierte Umgebung, die viele der müßigen Details der MSMQ-Programmierung verbirgt. Und wie alle anderen Umgebungen bietet er die größere Produktivität auf Kosten der Flexibilität.

Einige Dinge, die mit MSMQ möglich sind, werden von Queued Components nicht unterstützt. Sie sollten die Queued Components nur dann nutzen, wenn alle Funktionen unterstützt werden, die Sie benötigen. Sie ersparen sich dann viele Stunden der MSMQ-Programmierung. Eine der wichtigsten Funktionen ist ein serverseitiger Listener-Dienst. Es ist fast unmöglich, multi-threaded Listener-Dienste nur mit Hilfe von Visual Basic zu entwickeln. Wenn Sie also eine serverseitige Listener-Anwendung benötigen, die mehrere Nachrichten pro Sekunde verarbeitet, und diese nicht mit C++ entwickeln wollen, ist Queued Components genau das, was Sie brauchen.

COM+ Events-Dienst

Für einige Multi-Tier-Anwendungen ist es erforderlich, dass bestimmte Benutzer oder Anwendungen über wichtige Ereignisse benachrichtigt werden – beispielsweise wenn ein neuer Angestellter eingestellt wird, die Aktienpreise sich ändern oder der Lagerbestand unter einen bestimm-

ten Schwellenwert sinkt und nachgekauft werden muss. COM+ Events ist ein Dienst zum Senden und zur Weitergabe von Ereignisbenachrichtigungen zwischen Anwendungen.

Im Modell von COM+ Events werden Anwendungen, die Ereignisbenachrichtigungen versenden, als *Publisher* oder Verleger bezeichnet. Anwendungen, die Ereignisbenachrichtigungen erhalten, werden als *Subscriber* oder Abonnent bezeichnet. COM+-Ereignisse werden häufig auch als *lose verbundene Ereignisse* (LCE – loosely coupled events) bezeichnet, da Verleger und Abonnent keine Kenntnis voneinander haben. Stattdessen wird eine Reihe von Maßnahmen innerhalb des Gültigkeitsbereichs einer Ereignisklasse definiert, und diese Ereignisklasse wird in einer COM+-Anwendung installiert. Publisher und Subscriber kennen die Ereignisklassen, nicht aber einander.

Die Entkopplung von Verlegern und Abonnenten bietet diverse Vorteile. Sie brauchen sich nicht darum zu kümmern, ob jemand zuhört, wenn Sie eine Publisher-Anwendung schreiben, so dass weniger Code geschrieben werden muss. Nachdem die Publisher-Anwendung in Produktion gegangen ist, müssen Sie sie nicht ändern, wenn Sie Subscriber hinzufügen oder entfernen wollen. Darüber hinaus brauchen Sie auch keine Subscriber zu ändern, wenn Sie Publisher hinzufügen oder entfernen möchten, die mit einer Ereignisklasse verknüpft sind.

Hier folgt ein kurzer Überblick darüber, wie dieses Modell funktioniert. Sie schreiben eine Ereignisklasse, die eine oder mehrere Schnittstellen implementiert. Jede Schnittstelle definiert eine Gruppe von Methoden, die eine Ereignismenge darstellen. Sie schreiben Subscriber-Komponenten, um diese Schnittstellen zu implementieren, und auf Ereignisse zu reagieren, wenn sie ausgelöst werden. Schließlich schreiben Sie eine Publisher-Anwendung, um Objekte aus der Ereignisklasse zu erzeugen und verschiedene Methoden aufzurufen, wenn die Anwendung Ereignisse auslösen möchte. Der Ereignisdienst kümmert sich um die Verarbeitung der Methode und die Auslieferung der Ereignisse an jeden Subscriber.

Es gibt zwei Arten von Abonnements: *persistente (beständige) Abonnements* und *transiente (nicht beständige) Abonnements*. Wenn ein Ereignis an ein persistentes Abonnement ausgeliefert wird, erzeugt der COM+-Ereignisdienst ein neues Objekt aus einer konfigurierten Subscriber-Komponente und ruft die dem Ereignis zugeordnete Methode auf. Der Ereignisdienst »weckt« die Subscriber-Anwendung, falls diese noch nicht ausgeführt wird. Beachten Sie, dass bei einem persistenten Abonnement ein Objekt erzeugt und zerstört wird, wenn ein Ereignis ausgelöst wird.

Der zweite Typ ist das transiente Abonnement. Nachdem eine Anwendung ausgeführt wird, kann sie sich bei einer bestimmten Ereignisklasse registrieren, um darauf zu warten, dass bestimmte Ereignisse ausgelöst werden. In diesem Fall wird der Ereignisdienst nicht für jedes Ereignis ein Objekt erzeugen und zerstören. Der wichtigste Unterschied zwischen den beiden Abonnementtypen ist, dass ein transienter Abonnent programmtechnisch ein neues Abonnement erzeugen und einen Callback registrieren muss, wenn er auf Ereignisse warten will.

Die Programmierung eines transienten Abonnenten bedingt mehr Code als die Erstellung eines persistenten Subscribers. Es ist einfacher, einen persistenten Subscriber zu erzeugen, weil Sie die COM+-Administrationswerkzeuge nutzen können. Wenn der Server, auf dem sich die Ereignisklasse befindet, neu gestartet wird, arbeiten persistente Subscriber problemlos weiter. Alle transienten Subscriber dagegen müssen ein neues Abonnement erzeugen und registrieren.

Für eine skalierbare Architektur zur Ereignisbenachrichtigung ist eine asynchrone Kommunikation erforderlich. Wenn der Publisher darauf warten muss, dass die Subscriber ihre Ereignisbenachrichtigung verarbeiten, erweist sich das System möglicherweise als ungeeignet. Darüber hinaus verlangen einige Anwendungen vielleicht, dass die Ereignisbenachrichtigungen in Übereinstimmung mit MSMQ verbindungslos geschickt werden. COM+ Events wurde eng mit Queued Components verknüpft, um die Vorteile der asynchronen, verbindungslosen Kommunikation zu

bieten. Sowohl Publisher als auch Subscriber können so konfiguriert werden, dass sie statt RPC den Queued Components-Dienst verwenden. Für viele Anwendungen sind COM+-Ereignisse nur sinnvoll, wenn sie in Kombination mit diesem Dienst verwendet werden.

Verteilte Anwendungen mit Visual Basic entwickeln

Wenn Sie COM+ und seine Dienste nutzen, um verteilte Anwendungen für Windows 2000 zu entwickeln, befinden Sie sich in einer Umgebung wie in Abbildung 1.8 gezeigt. Wie Sie sehen, gibt es noch viel zu lernen.

Von allen COM-Entwicklungswerkzeugen auf dem Markt bietet Visual Basic den höchsten Produktivitätsgrad. Visual Basic 3 und Visual Basic 4 boten eine durchschnittliche Weiterentwicklung der COM-Features des Produkts, aber erst Visual Basic 5 ließ dieses Entwicklungswerkzeug zu einer wertvollen Option werden, um Komponenten für COM-basierte Systeme zu entwickeln. Visual Basic 6 hat die Unterstützung der COM- und MTS-Entwicklung noch weitergeführt.

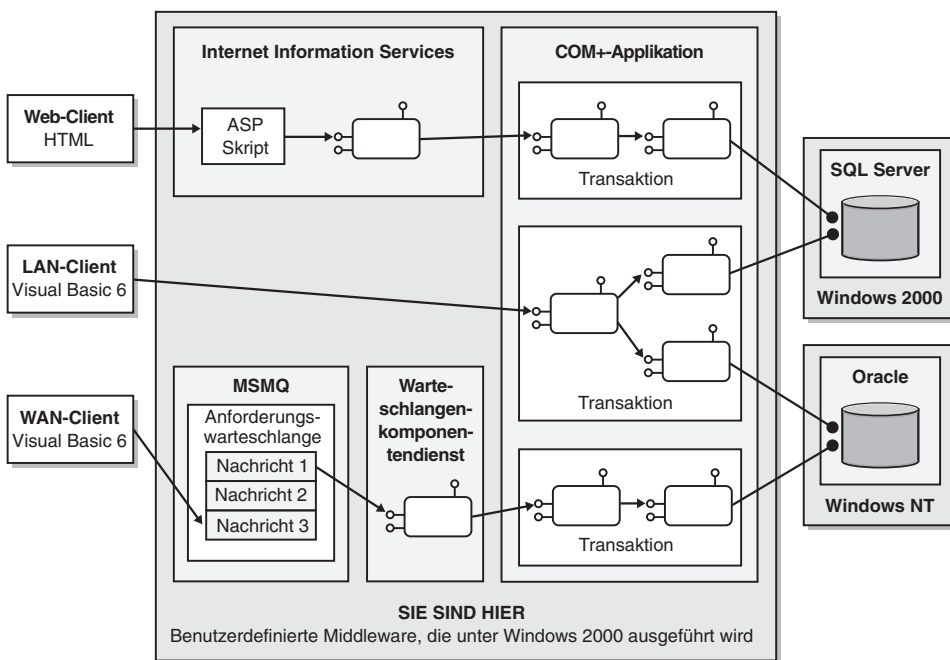


Abbildung 1.8: Mit Hilfe von COM+ und Visual Basic können Sie Middle-Tier-Komponenten entwickeln.

Denken Sie einen Moment lang über die folgenden Fragen nach: Was halten Sie von Visual Basic? Stellt es eine sinnvolle Möglichkeit für die Entwicklung von Code für eine große Anwendung dar? Warum lesen Sie dieses Buch? Beantworten Sie die folgenden Fragen mit Visual Basic, C++ oder SQL (Structured Query Language):

- Mit welcher Sprache ist es am einfachsten, Geschäftslogik zu entwickeln?
- Welche Sprache verursacht die geringsten Kosten für die Code-Wartung?

- Welche Sprache bietet die schnellste Möglichkeit, eine bereits existierende Anwendung zu erweitern?
- Was ist die beste Sprache für Personen, die keine klassische Informatik-Ausbildung besitzen?

Viele Entwickler und Projektleiter antworten auf all diese Fragen mit "Visual Basic". Aber Visual Basic ist nicht immer das Patentrezept. Für bestimmte Komponenten kann beispielsweise C++ oder eine andere Sprache die beste Wahl darstellen. Beispielsweise können viele COM-Schnittstellen, wie etwa die von der OLE DB bereitgestellte, nur über C++ bedient werden, d.h. einige Komponenten müssen in C++ entwickelt werden. Unternehmen haben jedoch viele Möglichkeiten, Visual Basic in einer verteilten Anwendung zu verwenden. Eine der wichtigsten Möglichkeiten ist die Entwicklung von Geschäftslogik und Datenzugriffscode für verteilte Objekte in der mittleren Schicht eines großen Informationssystems.

In der Zeit der Two-tier-Systeme war Visual Basic ein äußerst beliebtes Werkzeug für die Entwicklung der Benutzeroberfläche. Unternehmen verwendeten Visual Basic, um formularbasierte Anwendungen zu entwickeln, die auf den Desktops ausgeführt wurden. Visual Basic-Programmierer verbrachten einen Großteil ihrer Zeit damit, Formulare zu entwerfen und den Code für die Interaktionen mit dem Benutzer zu schreiben. Das hat sich natürlich geändert. Immer mehr Unternehmen verwenden browserbasierte Anwendungen für die Präsentationsschicht und verwenden dazu HTML-basierte Entwicklungswerkzeuge, beispielsweise Visual InterDev. Gewissermaßen haben die Visual Basic-Programmierer nach einem neuen Zuhause gesucht.

Die strahlende neue Zukunft für Visual Basic-Programmierer liegt in der Entwicklung nichtvisueller Komponenten für die mittlere Schicht. Unternehmen, die bisher der Annahme waren, für die COM-Programmierung sei C++ erforderlich, haben ihre Meinung geändert. Immer mehr Entwickler versuchen, so viel Code wie möglich für die mittlere Schicht mit Visual Basic zu schreiben. Darüber hinaus haben die Visual Basic-Programmierer viel Unterstützung von Personen erhalten, die die Plattform entwickeln. Die Architekten und Implementeure, die COM+ auf die Beine gestellt haben, betrachten die Visual Basic-Programmierer als ihre wichtigsten Kunden.

Zusammenfassung

Dieses Kapitel hat einen allgemeinen Überblick geboten und eine Grundlage geschaffen, die Sie benötigen, um große Informationssysteme mit COM+ und Windows 2000 zu entwickeln. Viele Unternehmen brauchen Multi-Tier-Anwendungen, um ihre Angestellten, Händler und Kunden zu erreichen. Aber jeder, der in der Multi-Tier-Entwicklung arbeitet, kann Ihnen sagen, dass es relativ kompliziert ist, eine verteilte Anwendung zu erstellen.

Die Entwickler von COM+ haben ihr Bestes gegeben, um die Komplexität der Entwicklung verteilter Anwendungen zu verbergen. Darüber hinaus ist auch Visual Basic dazu in der Lage, bestimmte Komplexitäten von COM+ und COM zu verbergen. Aber egal wie "verborgen" einige der Eigenheiten der Infrastruktur sind, brauchen Sie dennoch ein solides Wissen über die Kerntechnologien und die Untersysteme der Plattform. Je mehr Sie über die zugrunde liegende Infrastruktur wissen, desto besser. Die Philosophie, die Sie sich aneignen sollten, ist, dass Sie mit COM+ und Windows 2000 zusammenarbeiten werden. COM+ und Windows 2000 werden einen Teil der Arbeit übernehmen, während Sie den anderen Teil übernehmen. Das Geheimnis ist, COM+ und Windows 2000 das zu überlassen, was sie am besten können – die Bereitstellung der Infrastruktur auf unterster Ebene. Mit dieser Infrastruktur können Sie mehr Zeit für das aufwenden, was Sie wirklich tun müssen – die Entwicklung der Geschäftslogik und des Datenzugriffscode, die für Ihre jeweilige Anwendung spezifisch sind.

2 Schnittstellenbasierte Programmierung verstehen

-
- 26 Grundlagen der objektorientierten Programmierung
 - 34 Trennen der Schnittstelle von der Implementierung
-

Es ist nicht ganz einfach, sich einen Überblick über die schnittstellenbasierte Programmierung zu verschaffen. Um diesen Programmierstil zu verstehen, müssen Sie alle alten Gewohnheiten bei der Programmierung vergessen und die Entwicklung der objektorientierten Programmierung (OOP) und Informatik in den letzten zehn Jahren betrachten – eine Darwinsche Geschichte darüber, wie die Schnittstelle das moderne Software-Design überlebensfähiger gemacht hat. Damit Software in dem ständig andere Gesichter zeigenden Dschungel der Betriebsumgebungen überleben kann, braucht sie drei Eigenschaften: *Wiederverwendbarkeit*, *Wartbarkeit* und *Erweiterbarkeit*. Dieses Kapitel bietet einen allgemeinen Überblick über die schnittstellenbasierte Programmierung und betrachtet diese drei Eigenschaften genauer.

COM basiert auf dem Konzept der schnittstellenbasierten Programmierung, einem Programmierstil, der die Schnittstelle von der Implementierung trennt. (COM hätte ohne das Konzept der Schnittstelle keine Bedeutung. Schnittstellen dagegen gibt es auch außerhalb der COM-Welt.) Die schnittstellenbasierte Programmierung wurde in Sprachen wie C++ oder Smalltalk von Software-Ingenieuren vorangetrieben, die erkannten, dass unterschiedliche Schnittstellen dazu führten, dass ihre Programme einfacher zu warten und zu erweitern waren. Dies gilt besonders dann, wenn es sich um sehr große Anwendungen handelt. (Die Entwickler von Java erkannten die Eleganz der schnittstellenbasierten Programmierung und bauten deshalb eine direkte Unterstützung dafür in ihre Sprache ein.)

Schnittstellen lösen viele Probleme der Wiederverwendung von Code in der OOP. Dieses Kapitel spricht einige der Probleme an. Insbesondere, wenn Sie in einem Stil programmieren, der konform zur klassischen OOP ist, kann ein Client unflexible Abhängigkeiten von einer Klassendefinition aufbauen. Diese Abhängigkeiten können bewirken, dass es schwierig wird, die Klasse zu warten oder zu erweitern, ohne den Client unbrauchbar werden zu lassen. Es wird aufwendig oder gar unmöglich, den Code für ein Objekt im Laufe der Zeit zu verbessern. Einige Probleme haben auch mit einer bekannten Eigenschaft der OOP zu tun, der so genannten *Implementierungsvererbung*. Diese leistungsstarke aber häufig fehlerhaft verwendete Funktion ist empfänglich gegenüber ähnlichen Abhängigkeitsproblemen, die die Wartbarkeit und Erweiterbarkeit einer Anwendung negativ beeinflussen. Dieses Kapitel beschreibt die Stärken und Schwächen der Implementierungsvererbung (auch wenn diese Art der Vererbung in Visual Basic nicht unterstützt wird), um einige Probleme zu nennen, für deren Lösung die schnittstellenbasierte Programmierung geschaffen wurde.

Visual Basic 5.0 war die erste Version des Produkts, in dem die Definition und Implementierung benutzerdefinierter Schnittstellen unterstützt wurde. Dieses Kapitel zeigt Ihnen, wie Sie eine Visual Basic-Anwendung unter Berücksichtigung von Schnittstellen programmieren. Nachdem Sie die Grundlagen zur Verwendung von Schnittstellen kennen gelernt haben, zeige ich Ihnen, wie Sie *Polymorphie* realisieren und die *Laufzeittypenüberprüfung* benutzen.

Wie Sie im nächsten Kapitel erfahren werden, müssen COM-Clients über Schnittstellen mit Objekten kommunizieren. Sie sollten jedoch immer beachten, dass Sie in Visual Basic nicht explizit in Hinblick auf benutzerdefinierte Schnittstellen programmieren müssen. Manchmal ist es nicht einmal möglich, Komponenten zu erstellen, die benutzerdefinierte Schnittstellen implementieren. Sie müssen mit Klassen und öffentlichen Methoden arbeiten. Dies ist normalerweise der Fall, wenn Sie Server für Scripting-Clients erstellen.

Die Programmierung mit benutzerdefinierten Schnittstellen bietet Vorteile – jedoch auf Kosten einer zusätzlichen Komplexität. Dieses Kapitel möchte Ihnen beweisen, dass die Vorteile die Kosten überwiegen können.

Grundlagen der objektorientierten Programmierung

Um das Konzept der Schnittstellen kennen zu lernen, sollten Sie als Erstes die Probleme betrachten, für deren Lösung die schnittstellenbasierte Programmierung geschaffen wurde. Viele dieser Probleme haben mit der Beziehung zwischen einer Klasse und den Clients, die die Klasse verwenden, zu tun. Stellen Sie sich die folgenden Fragen: Welche Beziehung besteht zwischen einem Client und einer Klassendefinition? Was muss ein Client über eine Klasse wissen, um sie nutzen zu können? Welche Abhängigkeiten entstehen im Client, wenn ein Programmierer Code unter Verwendung der Methoden und Eigenschaften der Klasse schreibt?

In einem objektorientierten Paradigma ist es normalerweise so, dass ein Client ein Objekt aus einer Klasse instanziert. Der Client erzeugt das Objekt mit dem Operator *New* gefolgt von dem Klassennamen. Nachdem das Objekt angelegt ist, wird es vom Client unter Verwendung der vom Objekt bereitgestellten Eigenschaften und Methoden genutzt. Dies geschieht mit Hilfe einer Variablen, bei der es sich um eine *klassenbasierte Referenz* handelt. Es folgt ein einfaches Beispiel, das eine Variable verwendet, die auf einem Klassentyp basiert, um auf die öffentlichen Elemente eines Objekts zuzugreifen:

```
Dim Dog As CDog
Set Dog = New CDog
' Zugriff auf eine Eigenschaft.
Dog.Name = "Snoopy"
' Eine Methode aufrufen.
Dog.Bark
```

In diesem Beispiel wird das Instanzieren eines *Dog*-Objekts und die Kommunikation mit diesem durch eine klassenbasierte Referenz möglich. Die Kommunikation zwischen dem Client und dem Objekt findet über verschiedene öffentlich verfügbare Eigenschaften und Methoden statt, die auch als die *öffentliche Schnittstelle* des Objekts bezeichnet werden. Der Klassenautor muss die öffentliche Schnittstelle benutzen, um dem Client die Funktionalität des Objekts zur Verfügung zu stellen. Hierdurch wird ein Objekt erst nützlich. Beachten Sie, dass die Methodennamen und Eigenschaftsnamen der öffentlichen Schnittstelle im Programmcode des Clients vermerkt sind.

Zukünftige Versionen der Klasse müssen diese Elemente weiterhin unterstützen, um die im Client entstandenen Abhängigkeiten zu berücksichtigen.

Ein Vorteil bei der Verwendung von Klassen ist, dass sie die Wiederverwendung von Code erlauben. Nachdem eine Klasse geschrieben wurde, können Sie sie an unterschiedlichen Stellen einer Anwendung einsetzen. Klassen reduzieren somit den redundanten Code einer Anwendung. Außerdem vereinfachen sie die Wartung des Codes. Sie können jede nicht öffentliche Eigenschaft oder Methode modifizieren oder entfernen. Und Sie können die öffentlichen Methodenimplementierungen verändern, solange Sie nicht die Aufrufsyntax der Methode verändern. Wird die Implementierung einer Methode in einer Klasse verbessert, zieht jeder Client, der diese Klasse verwendet, sofort Nutzen daraus.

Wenn Sie eine bereits existierende Klassendefinition ändern, sollten Sie niemals die Aufrufsyntax für den Zugriff auf die öffentlichen Methoden oder Eigenschaften verändern. Sie würden in diesem Fall riskieren, dass die Abhängigkeiten des Client-Codes von der ursprünglichen Klassendefinition ungültig werden. Solange Sie die öffentliche Schnittstelle unverändert lassen, können Sie jede Änderung vornehmen, um Ihre Klasse zu verbessern, ohne den Client-Code zu beeinträchtigen.

Fassen wir die Kernaussage des letzten Absatzes noch einmal zusammen: Nachdem Sie die Signatur einer Eigenschaft oder Methode in der öffentlichen Schnittstelle einer Klasse bereitgestellt haben, dürfen Sie diese nicht mehr verändern oder entfernen. Dies bedeutet, dass die öffentliche Schnittstelle zu Beginn eines Projekts korrekt entworfen und während der Weiterentwicklung der Klasse konsequent verwendet werden muss. Auf diese Weise verbessern und erweitern Sie den Objektcode, ohne den Client-Code erneut schreiben zu müssen. Sie warten eine Klasse, indem Sie Fehler entfernen und die Methodenimplementierungen verbessern.

Die Regeln für die Wartung bereits existierender Elemente der öffentlichen Schnittstelle sind genau festgelegt. Doch über welche Flexibilität verfügen Sie, wenn Sie der Klasse neue Funktionen hinzufügen wollen? Was können Sie tun, um ein Objekt in späteren Versionen sicher zu erweitern? Es ist einfach und sicher, einer Klasse neue öffentliche Methoden und Eigenschaften hinzuzufügen, solange eine wichtige Voraussetzung erfüllt ist: Die Klasse und die Clients werden gleichzeitig kompiliert. Alter Client-Code wird wie gewohnt ausgeführt, wenngleich er die neue Funktionalität des Objekts nicht nutzen kann. Neuer Client-Code, der nach der Klassenmodifikation geschrieben wurden, kann jedoch die Elemente nutzen, die der öffentlichen Schnittstelle hinzugefügt wurden. Und das bedeutet, Sie können ein Objekt in der Betriebsumgebung sicher über einen längeren Zeitraum verbessern.

Was passiert, wenn eine Klassendefinition separat von dem Client-Code kompiliert wird, der sie nutzt? Auf diese Weise entsteht ein potentielles Versionsproblem. Betrachten wir ein Beispiel, das dieses Problem verdeutlicht. Angenommen, Sie kompilieren eine Klasse in eine Programmdatei, und den Client-Code, der die Klasse verwendet, in eine andere Programmdatei. Nachdem Sie beide Programmdateien in die Betriebsumgebung übernommen haben, wollen Sie der zweiten Version der Klasse eine neue Methode hinzufügen. Anschließend erzeugen Sie eine zweite Version des Clients, die diese neue Methode nutzt. Was geschieht nun, wenn Sie den neu kompilierten Client vor der neu kompilierten Klasse in Betrieb nehmen? Die neuere Version des Clients würde versuchen, eine Methode auszuführen, die die ältere Version der Klasse nicht unterstützt.

Dieses Beispiel bringt uns zu folgendem Schluss: Wenn Sie Klassen für ein komponentenbasiertes System schreiben, kann das Hinzufügen von Methoden in späteren Versionen einer Klasse ebenso nachteilig sein, als würden Methoden entfernt oder geändert. Die öffentliche Schnittstelle einer Klasse muss konstant bleiben, um in jeder Situation Kompatibilität zu garantieren.

Ändern der öffentlichen Schnittstelle einer Klasse

Probleme im Klassendesign treten auf, wenn Sie die Signatur einer öffentlichen Methode derart verändern, dass ein bereits bestehender Client nicht mehr korrekt ausgeführt wird. Dies geschieht in der Regel, wenn Sie feststellen, dass das ursprüngliche Klassendesign für Ihre Anforderungen nicht mehr geeignet ist. Stellen Sie sich beispielsweise eine Methode vor, die das Verhalten eines sich auf den Rücken rollenden Hundes repräsentiert. Die folgende `RollOver`-Methode wird mit einem 16-Bit-Integer-Parameter definiert, der dem Client erlaubt, bei jedem Aufruf der Methode anzugeben, wie oft sich der Hund drehen soll.

```
' Diese Methode ist in der CDog-Klasse definiert.  
Public Sub RollOver(ByRef Rolls As Integer)  
' Hier folgt der Implementierungscode.  
End Sub
```

```
' Aufrufsyntax des Clients.  
Dim Dog As CDog, Rolls As Integer  
Set Dog = New CDog  
Rolls = 20000  
Dog.RollOver Rolls
```

Was passiert, wenn die Anforderungen an das `Dog`-Objekt im ursprünglichen Entwurf nicht ausreichend berücksichtigt wurden? Was geschieht beispielsweise, wenn die Anzahl der Dreher (`Rolls`) den höchstmöglichen Wert für einen Integer (32767) überschreitet? Vielleicht hat Ihr Konkurrent gerade eine `Dog`-Komponente veröffentlicht, die 100000fache Drehungen des Hundes erlaubt? Sie können jetzt zwar Ihre Implementierung von `RollOver` umschreiben, um es Ihrem Konkurrenten gleichzutun oder ihn gar zu überbieten, aber dann haben Sie ein großes Problem mit Ihrer Methodensignatur. Damit sich Ihr `Dog`-Objekt 100000mal drehen kann, müssen Sie den Parametertyp in `Long` (Long Integer) ändern. Das führt zu einem Design-Problem. Die neuen Clients wollen einen 32-Bit-Long-Wert übergeben, während die älteren, wie der oben gezeigte, eine Abhängigkeit von dem 16-Bit-Integer-Wert aufweisen.

Sie haben nur zwei Möglichkeiten: Sie können die Signatur der Methode ändern und anschließend den gesamten Client-Code umschreiben, der die Methode aufruft. Die zweite Möglichkeit besteht darin, nichts zu verändern und mit den Einschränkungen des Originals zurechtzukommen. Wie Sie sehen, resultiert ein unzureichendes Klassen-Design entweder in nicht mehr verwendbaren Clients oder in nicht erweiterbaren Objekten.

Die intuitive Lösung dieses Problems besteht darin, sicherzustellen, dass das Design der öffentlichen Klassenschnittstelle über alle benötigten Funktionen verfügt und in sich abgeschlossen ist, bevor Sie mit der Programmierung des entsprechenden Client-Codes beginnen. Dies ist jedoch auch für erfahrene Programmierer nicht immer möglich. Wenn eine Klasse eine Entität aus der Realität nachbildet, die sich nicht verändert, kann ein erfahrener Entwickler ein robustes, dauerhaftes Design entwickeln. Häufig kann man jedoch nicht vorhersagen, wie sich externe Veränderungen auf die Anforderungen an eine öffentliche Objektschnittstelle auswirken. Dies gilt besonders für die Entwickler, die Klassen für eine Anwendung erstellen, die über Jahre hinweg in einer sich rapide verändernden Geschäftswelt eingesetzt werden soll. Wenn sich die Geschäftswelt stetig verändert, müssen sich Ihre Klassen ebenfalls verändern. Dieser Umstand erfordert erweiterbare Objekte.

Die Verwendung *klassenbasierter Referenzen* resultiert in einer Schicht von Abhängigkeiten zwischen Clients und Klassen. Diese Abhängigkeiten führen zu einer engen Verknüpfung zwischen verschiedenen Software-Bereichen. Als Autor einer Klasse wollen Sie natürlich die Frei-

heit besitzen, die in späteren Versionen bereitgestellte Funktionalität zu ändern und zu erweitern. Sie bremsen sich jedoch häufig selbst aus, wenn Sie Klassen mit öffentlichen Methoden verwenden. Wenn Sie Monate oder Jahre in einem großen Projekt involviert sind, stehen Sie Entscheidungen gegenüber, die in keinem Fall das gewünschte Ergebnis erzielen. Sollen Sie die von Ihrer Klasse definierte öffentliche Schnittstelle ändern und damit alle Clients zwingen, ihren Code anzupassen und neu zu kompilieren? Oder sollten Sie die Funktionalität Ihrer Objekte bis auf weiteres einfrieren?

Ein kurzer Leitfaden zur Implementierungsvererbung

Viele der in der OOP gebotenen Features sollen den Programmierern eine verstärkte Wiederverwendbarkeit von Code erlauben. Sprachen wie C++, Smalltalk und Java stellen eine verbreitete Funktionalität zur Verfügung, die als Implementierungsvererbung bezeichnet wird und die eine von vielen Möglichkeiten darstellt, die Wiederverwendbarkeit von Code in einem objektorientierten Paradigma zu realisieren. Einige Anwender behaupten, dass eine Sprache die Implementierungsvererbung anbieten muss, um als echte objektorientierte Sprache zu gelten. Diese Aussage führte zu einer hitzigen Debatte sowohl in der Software-Industrie als auch in akademischen Kreisen – an der wir uns hier jedoch nicht beteiligen wollen. Stattdessen konzentrieren wir uns auf die Vorteile und Probleme, die mit dieser leistungsfähigen Funktionalität verknüpft sind.

Bei der Implementierungsvererbung wird eine Klasse definiert, die den Code einer anderen Klasse wiederverwendet. Die Klasse, die wiederverwendet wird, wird als übergeordnete Klasse (Superklasse) bezeichnet. Die Klasse, die von der Wiederverwendbarkeit profitiert, ist die untergeordnete Klasse (Subklasse). Visual Basic unterstützt die Implementierungsvererbung nicht. Ich verwende deshalb ein Java-Beispiel, um zu demonstrieren, wie die Implementierungsvererbung genutzt wird. Betrachten Sie bitte die folgende Java-Klasse CDog.

```
// Übergeordnete Klasse
class CDog
{
    // Eigenschaften des Hundes.
    public String Name;

    // Verhalten des Hundes.
    public void Bark()
    { /* Implementierung der Methode */ }
    public void RollOver(int Rolls)
    { /* Implementierung der Methode */ }
}
```

Die Klasse CDog enthält eine Eigenschaft und zwei Methoden. Angenommen, jede Methode wurde mit einer geeigneten Implementierung definiert. Mit Hilfe der Implementierungsvererbung können Sie den Status und das Verhalten der Klasse wiederverwenden. CDog bildet die übergeordnete Klasse. Eine untergeordnete Klasse, die CDog erweitert, erbt sowohl die Eigenschaften als auch die Methodenimplementierungen der Klasse. Der folgende Java-Code zeigt die für die Implementierungsvererbung erforderliche Syntax.

```
// Untergeordnete Klasse
class CBeagle extends CDog
{
    // Eigenschaften des Beagles.
```

```
// Die Name-Eigenschaft wurde ererbt.
// Eine Farbeigenschaft wird hinzugefügt
    public String Color;

    // Verhalten des Beagles.
    // Die Implementierung der RollOver()-Methode
    // wurde ererbt.
    // Die Implementierung der Bark-Methode
    // wird außer Kraft gesetzt.
    public void Bark()
    {
        /* CBeagle-spezifische Implementierung der Methode */
    }
    // CBeagle erweitert CDog durch Hinzufügen einer neuen Methode
    public void FetchSlippers()
    {
        /* CBeagle-spezifische Implementierung der Methode */
    }
}
```

Wenn CBeagle (die untergeordnete Klasse) CDog (die übergeordnete Klasse) erweitert, erbt sie alle existierenden Implementierungen der Eigenschaften und Methoden. Dies bedeutet, CBeagle kann den Status und das Verhalten, die in CDog definiert sind, wiederverwenden. Sie können CDog anschließend erweitern, indem Sie die in CBeagle bereits existierenden Methoden, wie zum Beispiel Bark, überschreiben und neue Methoden, wie zum Beispiel FetchSlippers, hinzufügen. Sie können natürlich auch der Definition der untergeordneten Klasse neue Eigenschaften hinzufügen.

Sie sollten die Implementierungsvererbung nur dann verwenden, wenn eine logische »ist ein/e«-Beziehung zwischen der untergeordneten und der übergeordneten Klasse besteht. Für unser Beispiel gilt: »Ein Beagle ist ein Hund«. Abbildung 2.1 zeigt eine Beziehung, bei der eine Klasse von einer anderen Klasse abgeleitet ist. Solange die »ist ein/e«-Bedingung erfüllt ist, erweist sich die Implementierungsvererbung als nützlich für die Wiederverwendbarkeit von Code. Die Implementierungsvererbung kann besonders dann praktisch sein, wenn eine Anwendung mehrere Klassen enthält, die ein gemeinsames Verhalten aufweisen müssen. Die Gemeinsamkeiten mehrerer Klassen können in einer einzelnen übergeordneten Klasse gesammelt werden. Nachdem beispielsweise die Klasse CDog geschrieben wurde, kann sie zu CBeagle, CTerrier, CBoxer oder jeder anderen Klasse erweitert werden, die einen Hund repräsentiert (»ist ein« Hund). Der Code, der geschrieben wurde, um den Status und das Verhalten in der Klasse CDog zu definieren, kann in vielen anderen Klassen wiederverwendet werden.

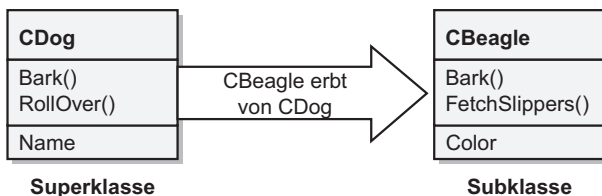


Abbildung 2.1: Die Implementierungsvererbung ermöglicht einer Klasse, den Status und das Verhalten einer anderen Klasse wiederzuverwenden.

Abbildung 2.2 zeigt eine grafische Darstellung der so genannten *Vererbungshierarchie*. Die Hierarchie verdeutlicht die Beziehungen zwischen den unterschiedlichen Klassen einer Applikation. Diese Hierarchie ist recht einfach; Sie können aber auch andere Hierarchien erzeugen, die weit aus komplexer sind. Stellen Sie sich beispielsweise eine Hierarchie vor, in der CWaldi die Klasse CTerrier erweitert, die wiederum CDog erweitert. CDog erweitert die Klasse CMammal, die schließlich CAnimal erweitert. Daran erkennen Sie, dass eine Vererbungshierarchie sehr groß und komplex werden kann. Hierarchien, die aus fünf oder mehr Ebenen bestehen, sind in Produktionscode nicht unüblich.

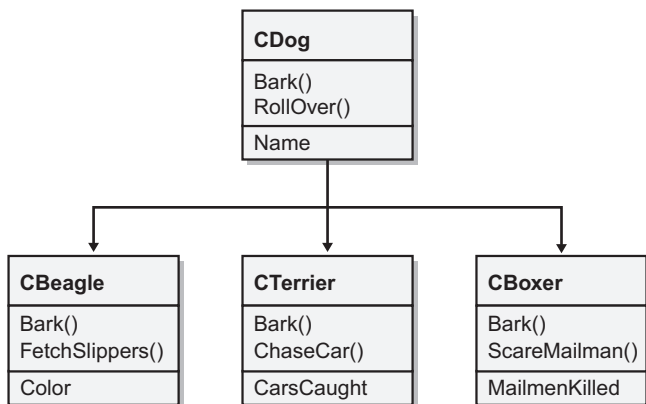


Abbildung 2.2: Eine Vererbungshierarchie zeigt die Beziehungen zwischen den übergeordneten und den untergeordneten Klassen einer Applikation.

Korrekt angewandt, kann die Implementierungsvererbung einen leistungsfähigen Mechanismus für die Wartung von Code darstellen. Wenn Sie die Implementierung einer Methode in einer übergeordneten Klasse verbessern, profitieren alle Klassen, die dieser Klasse in der Vererbungshierarchie folgen, automatisch davon. Eine Fehlerkorrektur in der `CAnimal`-Klasse kann somit eine große Zahl weiterer Klassen verbessern. Wird die Vererbungshierarchie größer und komplexer, können sich Modifizierungen der Klassen auf den obersten Hierarchiestufen signifikant auf viele der unteren Klassen auswirken. Dies kann zur Folge haben, dass eine einzelne Veränderung das Verhalten vieler verschiedener Objekttypen betrifft.

Was ist Polymorphie?

Bisher hat dieses Kapitel beschrieben, wie die Implementierungsvererbung die implizite Wiederverwendbarkeit von Methodenimplementierungen ermöglicht. Daraus resultiert eine größere Wartungsfähigkeit, da redundanter Code eliminiert wird. Ein weiteres leistungsfähiges OOP-Merkmal, das von der Implementierungsvererbung zur Verfügung gestellt wird, trägt die Bezeichnung *Polymorphie*. Die Polymorphie ist das möglicherweise wichtigste Konzept in der objektorientierten Programmierung. Sie erlaubt den Clients, unterschiedliche Objekte einheitlich zu behandeln, auch wenn sie aus unterschiedlichen Klassen erzeugt wurden und verschiedene Verhaltensmuster aufweisen.

Sie können die Implementierungsvererbung verwenden, um die Polymorphie in Sprachen wie C++ und Java zu realisieren. Beispielsweise können Sie eine übergeordnete Klassenreferenz einsetzen, um eine Verbindung zu den Methoden einer untergeordneten Klasseninstanz herzustellen und diese aufzurufen. Abbildung 2.3 zeigt, wie ein Client eine `CDog`-Referenz zur Kommunikation mit drei unterschiedlichen Objekttypen verwendet. Jede untergeordnete Klasse, die von `CDog` abgeleitet ist, ist typkompatibel mit einer `CDog`-Referenz. Ein Client kann daher ebenfalls eine `CDog`-Referenz verwenden, wenn er mit Objekten vom Typ `CBeagle`, `CTerrier` oder `CBoxer` kommuniziert.

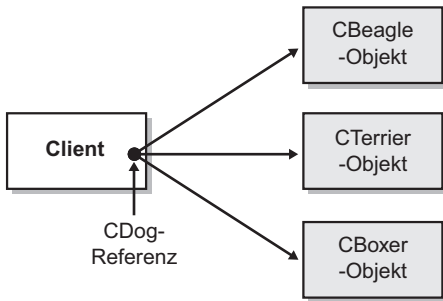


Abbildung 2.3: Sie realisieren Polymorphie, indem Sie eine übergeordnete Klassenreferenz verwenden, um mit den Instanzen einer untergeordneten Klasse zu kommunizieren. Ein Client kann eine CDog-Referenz benutzen, um mit jedem CDog-kompatiblen Objekt zu kommunizieren.

Ein Client kann davon ausgehen, dass jede Klasse, die die CDog-Klasse erweitert, eine Implementierung der Methode Bark zur Verfügung stellt. Dabei ist es dem Client egal, ob die untergeordnete Klasse die von CDog bereitgestellte Bark-Definition nutzt, oder ob sie diese Methode mit einer eigenen Implementierung überschrieben hat. Der Client ruft die Methode einfach über deren Aufrufsyntax auf, die in der CDog-Klasse definiert ist. Und wenn jede untergeordnete Klasse ihre eigene Implementierung von Bark anbietet, kann jeder Objekttyp in seiner eigenen, für ihn spezifischen Weise auf diese Anforderung antworten. Betrachten Sie bitte den folgenden Java-Code.

```
// Methode akzeptiert jedes CDog-kompatible Objekt.
Public void Make Dog Bark(CDog Dog)
{
    // Unterschiedliche Objekte können unterschiedlich antworten.
    Dog.Bark()
}
```

Wenn diese Methode mit einem CBeagle-Objekt aufgerufen wird, kann dies zu ganz anderen Ergebnissen als ein Aufruf mit dem CTerrier-Objekt führen. Der Client-Code weiß, welche Methode aufgerufen werden soll. Er weiß dagegen nicht, wie die Bark-Methode ausgeführt wird. Die Aufrufsyntax ist während des Kompilierens eindeutig definiert. Die Implementierung der Methode wird jedoch erst zur Laufzeit ermittelt. Polymorphie basiert auf dem Konzept des dynamischen Bindens, dem Gegenstück zum statischen Binden. Das dynamische Binden stellt einen Grad kontrollierter Ungewissheit dar, die die Polymorphie sehr leistungsfähig werden lässt.

Viele Entwickler haben erkannt, wie praktisch es ist, Anwendungen zu erstellen, die auf *plug-kompatiblen Objekten* basieren. Selbst wenn unzählige Zeilen Client-Code für die öffentliche CDog-Schnittstelle geschrieben wurden, können Sie einfach ein CBeagle-Objekt durch ein CTerrier-Objekt oder ein CBoxer-Objekt ersetzen. Eine derartige Veränderung wirkt sich nur geringfügig oder gar nicht auf den Client-Code aus, da dieser von der CDog-Klasse und nicht von den Klassen abhängig ist, die die CDog-Klasse erweitern.

Probleme bei der Implementierungsvererbung

Bisher wurden in diesem Kapitel nur die Vorteile der Implementierungsvererbung betrachtet: die implizite Wiederverwendbarkeit der Methodenimplementierungen und die Polymorphie. Es gibt jedoch auch Probleme bei der Implementierungsvererbung. Diese Technologie lässt eine Applikation leider anfälliger für bestimmte Abhängigkeitsprobleme werden, die durch klassenbasierte Referenzen bedingt sind. Der Grund hierfür ist die enge Verknüpfung zwischen einer untergeordneten Klasse und ihrer übergeordneten Klasse.

Durch eine geeignete Anwendung der Kapselung können Sie Implementierungsdetails vor den Clients verstecken. Dies ermöglicht Ihnen, die Implementierungsdetails der Klasse beliebig zu verändern, ohne den Client-Code zu beeinträchtigen. Das Problem der Implementierungsvererbung besteht darin, dass sie die Kapselung von nicht öffentlichen Elementen ungültig werden lässt.

Sprachen, die Implementierungsvererbung anbieten, stellen zusätzlich zum öffentlichen und privaten Niveau ein *geschütztes* Sichtbarkeitsniveau bereit. Eigenschaften und Methoden, die als geschützt deklariert werden, sind für einen Client unsichtbar. Die untergeordnete Klasse kann jedoch darauf zugreifen. Untergeordnete Klassen verfügen somit über eine Zugriffsmöglichkeit auf die Implementierungsdetails, die vor dem Client versteckt werden. Wenn Sie die Namen geschützter Eigenschaften und Methoden einer übergeordneten Klasse in einer untergeordneten Klasse codieren, erzeugen Sie auf diese Weise eine weitere Schicht unflexibler Abhängigkeiten.

Die Implementierungsvererbung ist ein Beispiel für einen Programmierstil, der als *White-Box-Wiederverwendbarkeit* bezeichnet wird. Anwendungen, die auf der White-Box-Wiederverwendbarkeit basieren, weisen häufig eine enge Verknüpfung zwischen den Klassen in der Vererbungshierarchie auf. Sobald eine untergeordnete Klasse eine geschützte Eigenschaft oder Methode verwendet, können Sie die Signatur der übergeordneten Klasse nicht mehr verändern oder entfernen, ohne die Abhängigkeiten in der untergeordneten Klasse zu zerstören. Dies führt zu Unsicherheiten in Applikationen mit großen Vererbungshierarchien. Das Verändern der Klassen auf der obersten Hierarchiestufe erfordert häufig die Anpassung vieler untergeordneter Klassen. In einigen Applikationen kann eine Veränderung von Methodensignaturen oder Eigenschaftentypen, die sich auf der obersten Hierarchiestufe befinden, zu zahlreichen Unterbrechungen der Vererbungskette führen. Diese Unterbrechungen betreffen die Verbindungen zwischen den Klassen, die sich auf den unteren Hierarchiestufen der Vererbungskette befinden. Andererseits führt das Einfrieren von öffentlichen und geschützten Schnittstellen der wichtigsten übergeordneten Klassen zu einem System, das sich nicht weiterentwickeln kann.

Entscheiden Sie sorgfältig, ob eine Eigenschaft oder Methode geschützt sichtbar sein soll. Ein korrektes Design, das die Implementierungsvererbung nutzt, erfordert ausgeprägte Fachkenntnisse und Konsequenz, um das so genannte *unsichere Superklassen-Szenario* zu verhindern. Sie sollten immer wissen, ob eine Klasse von untergeordneten Klassen erweitert wird. Wenn Sie erwarten, dass eine Klasse erweitert wird, ist es wichtig, die Implementierungsdetails mit Hilfe der Kapselung vor den Clients und den untergeordneten Klassen zu verbergen.

Hier sollte keineswegs der Eindruck entstehen, die Implementierungsvererbung sei nicht nützlich! Für manche Entwicklungsszenarien ist sie ein durchaus leistungsfähiges Merkmal. Und am besten ist sie für kleine, kontrollierte Situationen geeignet. Das Erstellen einer großen Vererbungshierarchie, die mit den Anforderungen an eine Applikation wächst, ist nur durch wirklich erfahrene objektorientierte Entwickler realisierbar.

Als C++ und Smalltalk eingeführt wurden, wurde die Implementierungsvererbung von den Fürsprechern der OOP als Patentrezept für die Wiederverwendbarkeit von Code überbewertet. Das führte dazu, dass dieses Feature von vielen Entwicklern falsch eingesetzt wurde, die die Probleme nicht erkannten, die mit der Verknüpfung durch die White-Box-Wiederverwendbarkeit entstanden. In den letzten zehn Jahren lähmte diese unbedachte Verwendung der Implementierungsvererbung die Weiterentwicklung großer Systeme. Erfahrene Entwickler, die erkannt hatten, dass die Implementierungsvererbung lediglich im kleinen Rahmen genutzt werden darf, suchten nach flexibleren Möglichkeiten der Wiederverwendbarkeit in großen Systemen. Die Erweiterungsfähigkeit sollte dadurch nicht beeinträchtigt werden. Diese Suche führte zur Einführung der schnittstellenbasierten Programmierung und eines Entwicklungsstils, der als *Objektkomposition* bezeichnet wird.

Trennen der Schnittstelle von der Implementierung

Die schnittstellenbasierte Programmierung erlaubt eine Wiederverwendung, ohne gleichzeitig zu festen Klassenverknüpfungen zu führen. Die schnittstellenbasierte Programmierung basiert auf der *Black-Box-Wiederverwendbarkeit*, wobei die Kapselung nie verletzt wird. Die Clients kennen nur die Namen und die Aufrufsyntax für eine bestimmte Menge an Anfragen. Die Implementierungsdetails der verwendeten Objekte bleiben ihnen verborgen.

Die Black-Box-Wiederverwendbarkeit wird durch eine formelle Abtrennung der Schnittstelle von der Implementierung realisiert. Die Schnittstelle wird damit zu einem besonders wichtigen Element. Eine Schnittstelle ist ein unabhängiger Datentyp, der selbst definiert wird. Das ist eine Weiterentwicklung der klassischen OOP, wo eine öffentliche Schnittstelle innerhalb des Gültigkeitsbereichs einer Klassendefinition definiert wird.

Sie denken nun möglicherweise, dass all diese Beschreibungen sehr undeutlich sind. Sie fragen sich, was genau eine Schnittstelle ist. Leider existiert keine kurze, präzise Definition, die die Schlüsselkonzepte einer vollständig neuen Vorgehensweise bei der Software-Entwicklung verdeutlicht. Eine Schnittstelle kann unterschiedlich beschrieben werden. Möglicherweise werden Sie sehr schnell verstehen, wie man Schnittstellen definiert, implementiert und verwendet. Die Auswirkung der Schnittstellen auf die Software-Entwicklung sind jedoch für den durchschnittlichen Programmierer sehr viel schwerer zu erkennen. Das Entwickeln mit Schnittstellen zu erlernen, kann bisweilen Monate oder Jahre dauern.

Eine Schnittstelle ist zunächst eine *Sammlung öffentlicher Methodensignaturen*. Sie legt die Aufrufsyntax für verschiedene, logisch zusammenhängende Client-Anfragen fest. Eine Schnittstelle definiert Methodensignaturen, kann jedoch keine Implementierungen oder Dateneigenschaften enthalten. Durch die Bereitstellung einer indirekten Schicht trennt eine Schnittstelle eine Klasse von dem Client ab, der die Klasse verwendet. Dies bedeutet, eine Schnittstelle muss von einer oder mehreren Klassen implementiert werden, um einen Nutzen zu erbringen. Nachdem eine Schnittstelle von einer Klasse implementiert wurde, kann ein Client ein Objekt aus der Klasse erzeugen und über eine Schnittstellenreferenz mit diesem kommunizieren.

Sie können eine Schnittstelle verwenden, um eine Objektreferenz, aber nicht das Objekt selbst zu erstellen. Das ist sinnvoll, weil ein Objekt Dateneigenschaften und Methodenimplementierungen benötigt, die von einer Schnittstelle nicht bereitgestellt werden können. *Eine Schnittstelle ist ein abstrakter Datentyp*, weil sie nicht als Entität erstellt werden kann. Objekte können nur aus erstellbaren Klassen instanziiert werden, die als *konkrete Datentypen* bezeichnet werden.

Aus der Design-Sicht betrachtet, *ist eine Schnittstelle ein Vertrag*. Eine Klasse, die eine Schnittstelle implementiert, garantiert, dass die von ihr bedienten Objekte ein bestimmtes Verhalten unterstützen. Genauer gesagt, muss eine Klasse eine Implementierung für jede von der Schnittstelle definierte Methode zur Verfügung stellen. Bei der Kommunikation mit einem Objekt über eine Schnittstellenreferenz kann ein Client davon ausgehen, dass das Objekt angemessen auf jede in der Schnittstelle definierte Methode reagiert.

Mehrere Klassen können dieselbe Schnittstelle implementieren. Eine Schnittstelle definiert die exakte Aufrufsyntax sowie die offene Semantik jeder Methode. Die offene Semantik ermöglicht den Klassenautoren, ein geeignetes Objektverhalten für jede Methode festzulegen. Wenn die IDog-Schnittstelle (i für das englische Wort *Interface*, das Schnittstelle bedeutet. Anm. d. Ü.) beispielsweise die Methode mit der Bezeichnung `Bark` definiert, können verschiedene Klassenautoren unterschiedlich auf dieselbe Anfrage reagieren. Das ist jedoch nur möglich, solange jeder Autor das Konzept eines bellenden Hundes irgendwie unterstützt. Die Klasse `CBeagle` kann die Methode `Bark`

in einer Weise implementieren, die sich von der für `CTerrier` oder `CBoxer` implementierten unterscheidet. Das bedeutet, dass *Schnittstellen das Konzept der Polymorphie zur Verfügung stellen*. Schnittstellen gleichen der Implementierungsvererbung, weil sie Ihnen ermöglichen, Applikationen zu erstellen, die sich aus plugkompatiblen Objekten zusammensetzen. Schnittstellen bieten jedoch Plug-Kompatibilität ohne das Risiko fester Klassenverknüpfungen, das bei der Implementierungsvererbung und White-Box-Wiederverwendbarkeit auftreten kann.

Die zwei Gesichter der Vererbung

Vererbung ist ein objektorientiertes Konzept, das eine »ist ein/e«-Beziehung zwischen zwei Elementen beschreibt. Dieses Kapitel hat bisher den Ausdruck *Implementierungsvererbung* anstelle des generischeren Begriffs *Vererbung* verwendet. Der Grund dafür ist, dass die Erweiterung einer übergeordneten Klasse durch eine untergeordnete Klasse lediglich eine Möglichkeit darstellt, eine »ist ein/e«-Beziehung zu unterstützen. Wenn eine Klasse eine Schnittstelle implementiert, nutzt sie ebenfalls eine »ist ein/e«-Beziehung. Wenn beispielsweise die `CBeagle`-Klasse die Schnittstelle `IDog` implementiert, ist die Aussage »ein Beagle "ist ein" Hund« korrekt. Sie können ein `CBeagle`-Objekt in allen Situationen verwenden, in denen Sie ein `IDog`-kompatibles Objekt benötigen.

Die schnittstellenbasierte Programmierung basiert auf einer zweiten Form der Vererbung, die als *Schnittstellenvererbung* bezeichnet wird. Im Allgemeinen erfordert die Vererbung keine Wiederverwendbarkeit der Methodenimplementierungen. Vielmehr besteht die einzig wahre Anforderung an die Vererbung darin, dass die Instanz einer untergeordneten Klasse kompatibel mit dem zu erweiternden Basistyp sein muss. Der zu erweiternde Basistyp kann eine Klasse oder eine benutzerdefinierte Schnittstelle sein. Sie können die Basistypenreferenz in beiden Fällen verwenden, um mit unterschiedlichen Objekttypen zu kommunizieren. Auf diese Weise können beide Formen der Vererbung die Polymorphie realisieren.

Sowohl die Implementierungsvererbung als auch die Schnittstellenvererbung unterstützen Polymorphie. Sie unterscheiden sich jedoch in der Verwendung der Kapselung. Die Implementierungsvererbung basiert auf der White-Box-Wiederverwendbarkeit, die einer untergeordneten Klasse alle Details der Klassen offenbart, die von ihr erweitert werden. Die untergeordnete Klasse kann daher die Methodenimplementierungen und Dateneigenschaften einer übergeordneten Klasse implizit wiederverwenden. Die Implementierungsvererbung ist hinsichtlich der Wiederverwendung von Status und Verhalten sehr viel leistungsfähiger als die Schnittstellenvererbung. Diese Wiederverwendbarkeit hat jedoch ihren Preis. Der Verlust der Kapselung in der White-Box-Wiederverwendbarkeit begrenzt ihre Skalierbarkeit in großen Systemen, die auf der Implementierungsvererbung basieren.

Der Begriff *Black-Box-Wiederverwendbarkeit* weist darauf hin, dass die Schnittstellenvererbung das Konzept der Kapselung erzwingt. Die strikte Einhaltung der Kapselung von Implementierungsdetails in Klassen ermöglicht ein skalierbares Applikationsdesign. Die schnittstellenbasierte Programmierung löst viele Probleme, die mit der White-Box-Wiederverwendbarkeit verbunden sind. Um diesen Programmierstil zu verstehen, müssen Sie den Gedanken zulassen, dass die Vorteile den Aufwand überwiegen. Für viele Programmierer ist das zunächst nicht ganz einfach.

Wenn eine Klasse eine Schnittstelle implementiert, verpflichtet sie sich, bestimmte Methoden zur Verfügung zu stellen. Die Autoren untergeordneter Klassen müssen zusätzlichen Code schreiben, wenn sie eine Schnittstelle implementieren. Wenn Sie dies mit der Implementierungsvererbung vergleichen, scheint die eben genannte Vorgehensweise sehr viel mehr Arbeit zu bedeuten. Wenn Sie von einer Klasse erben, ist die meiste Arbeit bereits erledigt. Wenn Sie jedoch von einer

Schnittstelle erben, hat Ihre Arbeit gerade erst begonnen. Auf den ersten Blick erscheint uns die Implementierungsvererbung wie ein Cheeseburger, während die Schnittstellenvererbung wie eine Schüssel gedämpften Brokkolis aussieht. Sie müssen den Wunsch aufgeben, den Cheeseburger zu besitzen, um eine höhere Bewusstseinsebene für Schnittstellen zu erreichen. Der wichtigste Vorteil der Schnittstellenvererbung gegenüber der Implementierungsvererbung besteht darin, dass die Schnittstellenvererbung nicht anfällig für die feste Klassenverknüpfungen ist, die die Erweiterungsfähigkeit einer Applikation beeinträchtigen.

Verwenden von Schnittstellen mit Visual Basic

Visual Basic 5.0 war die erste Version des Produkts, die benutzerdefinierte Schnittstellen unterstützte. Sie nutzen die Vorteile der schnittstellenbasierten Programmierung in einem Visual Basic-Projekt, indem Sie die folgenden drei Schritte beachten:

1. Definieren Sie eine Schnittstelle.
2. Implementieren Sie die Schnittstelle in einer oder mehreren erstellbaren Klassen.
3. Verwenden Sie eine Schnittstellenreferenz in einem Client, um mit Objekten zu kommunizieren.

Wie Sie sehen, sind die grundlegenden Schritte zum Hinzufügen von Schnittstellen zu Ihren Applikationen ganz einfach. Die Verwendung von Schnittstellen ermöglicht Ihnen außerdem, Ihre Applikationen mit Polymorphie auszustatten. Wir werden ein einfaches Beispiel betrachten, das die Visual Basic-Syntax demonstriert, die die oben aufgeführten Schritte realisiert.

Sie erzeugen eine benutzerdefinierte Schnittstelle in Visual Basic, indem Sie gewöhnliche Klassenmodule verwenden. Es wäre sehr praktisch, wenn die Visual Basic-IDE (Integrated Development Environment, Integrierte Entwicklungsumgebung) einen separaten Editor für den Schnittstellenentwurf anböte. Leider gibt es jedoch noch keinen eigenen Editor für die Schnittstellenentwicklung. Sie definieren sowohl Schnittstellen als auch Klassen mit dem Klassenmodul-Editor.

Um eine neue Schnittstelle zu definieren, fügen Sie einem bereits bestehenden Projekt einfach ein neues Klassenmodul hinzu und geben diesem einen geeigneten Namen. Wenn Sie eine Schnittstelle definieren, um das Verhalten eines Hundes auszudrücken, könnte `IDog` oder `itfDog` (`I` oder `itf` für das englische Wort *Interface*. Anm. d. Ü.) eine geeignete Bezeichnung sein. Dies sind die beiden gebräuchlichsten von Visual Basic-Entwicklern verwendeten Namenskonventionen. Wenn Sie an einem Visual Basic-Projekt arbeiten, das entweder eine ActiveX-DLL oder eine ActiveX-EXE ist, sollten Sie auch die *Instancing*-Eigenschaft des Klassenmoduls auf *PublicNotCreatable* setzen. Diese Einstellung ist sinnvoll, weil die Schnittstelle einen abstrakten Datentyp repräsentiert. In einem Standard-EXE-Projekt verfügen Klassenmodule nicht über eine anpassbare *Instancing*-Eigenschaft.

Sie definieren Ihre Schnittstelle, indem Sie die Aufrufsyntax für eine Gruppe öffentlicher Methoden erstellen. Nehmen Sie keine Implementierung für diese Methoden in die Schnittstelle auf. Sie benötigen lediglich die Signaturen. Im Wesentlichen definieren Sie, *wie* der Client die Methoden aufruft, und nicht, *was* nach einem Aufruf geschehen soll. Nachfolgend finden Sie ein Beispiel für die `IDog`-Schnittstelle, die in einem Visual Basic-Klassenmodul definiert ist.

```
' Schnittstelle IDog  
' beschreibt das Verhalten eines Hund-Objekts
```

```
Public Property Get Name() As String
```



```
End Property
```

```
Public Property Let Name() As String  
End Property
```

```
Public Sub Bark()  
End Sub
```

```
Public Sub RollOver(ByRef Rolls As Integer)  
End Sub
```

Eine der ersten Besonderheiten, die Sie während der Deklaration einer Schnittstelle unter Visual Basic bemerken werden, ist das Vorhandensein von `End Sub`, `End Function` oder `End Property` hinter jeder Methodensignatur. Das Schlüsselwort `End` zeigt in der Regel das Ende einer Methodenimplementierung an. Dies ist eine verwirrende Eigenheit der Visual Basic-IDE und ein unglücklicher Nebeneffekt der Verwendung von Visual Basic-Klassenmodulen für die Definition sowohl von Klassen als auch von Schnittstellen. Eine zukünftige Version von Visual Basic wird möglicherweise einen Modultyp zur Verfügung stellen, der die Definition von Schnittstellen erlaubt, die kein `End Sub`, `End Function` oder `End Property` erfordern. Gegenwärtig müssen Sie jedoch eine gute Miene zum bösen Spiel machen.

Eine weitere wichtige Besonderheit besteht darin, dass diese Schnittstelle neben Methoden auch logische Eigenschaften verwenden kann. Dies ist vernünftig, wenn Sie berücksichtigen, dass eine logische Eigenschaft eigentlich eine Methodenmenge und keine Dateneigenschaft ist. Der Client kann die logische `Name`-Eigenschaft, die in der zuvor gezeigten Schnittstelle definiert ist, wie eine gewöhnliche Dateneigenschaft verwenden. Sie muss jedoch als `Property Let`/`Property Get`-Methodenpaar implementiert sein.

Stellen Sie sich jetzt die folgenden Frage: Warum kann eine Schnittstelle keine Datenelemente enthalten? Weil eine Schnittstelle, im Gegensatz zu einer Klasse, niemals zum Erstellen von Objekten verwendet wird. Objekte werden aus einer Klasse erzeugt, die die Implementierungsdetails der Klasse kapselt. Das Datenlayout eines Objekts ist eines der wichtigsten Details, die in einer Klassendefinition eingekapselt werden. Enthielte eine Schnittstelle Datenelemente, würde der Client Abhängigkeiten davon aufweisen. Und Sie wissen bereits, dass Abhängigkeiten nichts Gutes bedeuten.

Obwohl Schnittstellen keine Dateneigenschaften enthalten können, erlaubt Ihnen Visual Basic, eine Eigenschaft in einer Schnittstelle wie folgt zu definieren:

```
Public Name As String
```

Wenn Sie eine Dateneigenschaft in einer Schnittstelle definieren, wandelt Visual Basic diese transparent in eine logische Eigenschaft um. Dies ist eine Annehmlichkeit, die Visual Basic zur Verfügung stellt, wenn Sie benutzerdefinierte Schnittstellen erstellen. Die zuvor definierte `Name`-Eigenschaft erfordert `Property Let` und `Property Get` in jeder Klasse, die die Schnittstelle implementiert. Beachten Sie außerdem, dass die Implementierung einer Schnittstelle keine Auswirkungen auf das Datenlayout einer Klassendefinition hat. Jede Klasse, die diese Schnittstelle implementiert, sollte eine private Dateneigenschaft für das physische Speichern des Hundenamens bereitstellen.

Nachdem Sie die Schnittstellendefinition erstellt haben, besteht der nächste Schritt darin, eine konkrete Klasse zu programmieren, die diese Schnittstelle implementiert. Fügen Sie Ihrem Projekt ein zweites Klassenmodul hinzu und geben Sie ihm einen geeigneten Namen. Sie können

beispielsweise die Klasse `CBeagle` erstellen, die die `IDog`-Schnittstelle implementiert. Sie müssen das Schlüsselwort `Implements` zu Beginn eines Klassenmoduls angeben, wie nachfolgend gezeigt.

```
Implements IDog

Private Sub IDog_Bark()
    ' Hier folgt der Implementierungscode
End Sub
```

Beim Zuordnen von Schnittstellen unter Visual Basic muss jede Methodenimplementierung den Namen der Schnittstelle, gefolgt von einem Unterstrich und dem Methodennamen angeben. Visual Basic verwendet diese ihm eigene Syntax, um einen Einsprungpunkt in ein Objekt zu erzeugen, wenn eine bestimmte Schnittstelle verwendet wird. Der Visual Basic-Compiler erwartet von Ihnen eine ähnliche Implementierung für jede in der Schnittstelle vorhandene Methode und logische Eigenschaft. Diese Vorgehensweise garantiert, dass alle Objekte, die aus der Klasse erzeugt wurden, über Einsprungpunkte für alle Schnittstellenelemente verfügen.

Glücklicherweise vereinfacht die Visual Basic-IDE das Erstellen von Prozedurgerüsten für die Methodenimplementierungen, wenn Sie das Schlüsselwort `Implements` am Anfang des Klassenmoduls angeben. Das Code-Editor-Fenster des Klassenmoduls verfügt über eine Assistentenleiste, auf der Sie zwei Kombinationsfelder finden. Wenn Sie im linken Kombinationsfeld den Namen der Schnittstelle auswählen, können Sie sehr schnell das Gerüst der Methodenimplementierungen generieren lassen, indem Sie im rechten Kombinationsfeld die Methodennamen auswählen. Ein Beispiel hierfür sehen Sie in Abbildung 2.4. Nachfolgend finden Sie eine partielle Implementierung der `CBeagle`-Klasse, die wiederum die `IDog`-Schnittstelle implementiert.

```
Implements IDog
Private Name As String

Private Property Let IDog_Name(ByVal Value As String)
    Name = Value
End Property

Private Property Get IDog_Name() As String
    IDog_Name = Name
End Property

Private Sub IDog_Bark()
    ' Hier folgt der Implementierungscode.
End Sub

Private Sub IDog_RollOver(Rolls As Integer)
    ' Hier folgt der Implementierungscode.
End Sub
```



Abbildung 2.4: Die Assistentenleiste vereinfacht das Erstellen der Prozedurgerüste zur Implementierung einer benutzerdefinierten Schnittstelle.

Die Assistentenleiste generiert Methodensignaturen, die als `Private` deklariert sind. Das bedeutet, diese Methodenimplementierungen stehen nicht den Clients zur Verfügung, die eine `CBeagle`-Referenz verwenden. Sie können lediglich von den Clients genutzt werden, die eine `IDog`-Referenz verwenden. Der oben gezeigte Code demonstriert ebenfalls, wie die `CBeagle`-Klasse eine logische Name-Eigenschaft implementiert, indem eine private Dateneigenschaft definiert und die Methoden `Property Let` und `Property Get` implementiert werden.

Nachdem Sie nun eine Schnittstelle und eine Klasse, die die Schnittstelle implementiert, erstellt haben, können Sie die Schnittstelle nutzen, um mit einem Objekt zu kommunizieren. Beispielsweise kann ein Client mit einem `CBeagle`-Objekt über eine `IDog`-Referenz kommunizieren. Sie können die `IDog`-Referenz nutzen, um eine beliebige Methode aufzurufen, die von der Schnittstelle definiert wird. Hier ein einfaches Beispiel.

```
Dim Dog As IDog ' IDog ist abstrakt.
Set Dog = New CBeagle ' CBeagle ist konkret
' Zugriff auf Objekt über Schnittstellenreferenz.
Dog.Name = "Spot"
Dog.Bark
Dog.RollOver 12
```

Nachdem der Client über die Schnittstellenreferenz mit dem Objekt verbunden wurde, kann er Methoden aufrufen und auf logische Eigenschaften zugreifen. Die Visual Basic-IDE bietet dazu, wie von der Verwendung klassenbasierter Referenzen gewohnt, `IntelliSense`, Typprüfung und `Debugging` an. Beachten Sie bitte, dass Sie hinter dem `New`-Operator keine Schnittstelle verwenden können. Eine Schnittstelle ist kein erstellbarer Typ. Sie müssen eine konkrete Klasse wie `CBeagle` verwenden, um ein Objekt mit dem `New`-Operator zu erstellen.

Warum benutzerdefinierte Schnittstellen verwenden?

Wenn Visual Basic-Programmierer lernen, wie Schnittstellen in einer Applikation verwendet werden, fragen sie sich oft »Warum sollte ich dies jemals tun?« oder »Was geht das mich an?«. Die Programmierung mit klassenbasierten Referenzen scheint, verglichen mit der zusätzlichen Komplexität benutzerdefinierter Schnittstellen, sehr viel natürlicher zu sein. Das vorherige Beispiel wäre einfacher, wäre der Client-Code für die `CBeagle`-Klasse und nicht für die `IDog`-Schnittstelle geschrieben worden. Benutzerdefinierte Schnittstellen bedeuten scheinbar zusätzliche Arbeit, aus der sich keine spürbaren Vorteile ergeben.

Es gibt mehrere wichtige Gründe, weshalb sich ein Visual Basic-Programmierer, der verteilte Anwendungen entwickelt, mit Schnittstellen befassen sollte. Der erste Grund besteht darin, dass Schnittstellen die Grundlage von COM bilden. In COM können Clients keine klassenbasierten Referenzen verwenden. Sie müssen stattdessen über Schnittstellenreferenzen auf COM-Objekte zugreifen. Wie Sie später in diesem Kapitel sehen werden, kann Visual Basic die Komplexität dieser Anforderung sehr gut vor Ihnen verstecken. Wenn Sie eine klassenbasierte Referenz benutzen, generiert Visual Basic im Hintergrund automatisch eine COM-Standardschnittstelle für die Klasse. Dies bedeutet, dass Sie mit Visual Basic arbeiten können, ohne je explizit mit benutzerdefinierten Schnittstellen zu arbeiten. Wenn Sie sich jedoch zur schnittstellenbasierten Programmierung bekennen, werden Sie zu einem sehr viel besseren COM-Programmierer.

Ein weiterer Grund, sich mit Schnittstellen zu beschäftigen, ist, dass diese in der Software-Entwicklung sehr leistungsfähig und flexibel sein können. Sie sollten benutzerdefinierte Schnittstellen unter Visual Basic verwenden, wenn zwischen einer Klasse und einer öffentlichen Schnittstelle keine 1:1-Zuordnung möglich ist. Dies passiert üblicherweise in zwei Szenarios. Im ersten Fall erstellen Sie eine Schnittstelle und implementieren diese in mehrere Klassen. Im zweiten Fall implementieren Sie mehrere Schnittstellen in einer einzelnen Klasse. Beide Techniken bieten Vorteile gegenüber Applikationsentwürfen, in denen die Clients darauf beschränkt sind, Referenzen zu verwenden, die auf konkreten Klassen basieren. Schnittstellenbasierte Entwürfe sind häufig weitaus komplexer und schränken Sie überdies in keiner Weise ein.

Stellen Sie sich eine Situation vor, in der viele Klassen dieselbe Schnittstelle implementieren. Die Klassen CBeagle, CTerrier und CBoxer implementieren alle die Schnittstelle IDog. Eine Applikation kann daraufhin eine Sammlung IDog-kompatibler Objekte verwalten, wie im folgenden Code gezeigt.

```
Dim Dog1 As IDog, Dog2 As IDog, Dog3 As IDog
' Hund-Objekte erstellen und initialisieren
Set Dog1 = New CBeagle
Dog1.Name = "Mo"
Set Dog2 = New CTerrier
Dog2.Name = "Larry"
Set Dog3 = New CBoxer
Dog3.Name = "Curly"
' Hunde zur Auflistung hinzufügen
Dim Dogs As New Collection
Dogs.Add Dog1
Dogs.Add Dog2
Dogs.Add Dog3
```

Die Applikation verhält sich polymorph, weil sie alle IDog-kompatiblen Objekte gleichbehandelt. Der folgende Code durchläuft die Auflistung und ruft für jedes Objekt die Methode Bark auf.

```
Dim Dog As IDog
For Each Dog In Dogs
    Dog.Bark
Next Dog
```

Während der Entwicklung dieser Applikation kann die Auflistung derart modifiziert werden, dass sie verschiedene IDog-kompatible Objekte verwaltet. Dazu zählen Objekte, die aus den Klassen CBeagle, CTerrier, CBoxer und allen zukünftigen Klassen erstellt werden, die die IDog-Schnittstelle implementieren. Die For Each-Schleife des vorherigen Beispiels wurde für die IDog-Schnittstelle ge-

schrieben und weist keine Abhängigkeiten von konkreten Klassen auf. Sie müssen die Schleife nicht verändern, wenn Sie neue Klassentypen in die Applikation einführen.

Eine weitere leistungsfähige Designtechnik besteht darin, mehrere Schnittstellen für eine einzelne Klasse zu implementieren. Auf diese Weise verfügen Sie über Objekte, die mehrere Schnittstellen und somit mehrere Verhalten unterstützen. Diese Vorgehensweise ist zusammen mit der Laufzeittypenüberprüfung ein sehr leistungsfähiges Feature. Stellen Sie sich vor, der Beispielapplikation wird eine weitere Schnittstelle hinzugefügt, die mit `IWonderDog` bezeichnet ist und die folgende Methode definiert:

```
Sub FetchSlippers()
```

Angenommen, Sie haben zwei Klassen, `CBeagle` und `CTerrier`, die beide `IDog` implementieren. Die `CBeagle`-Klasse implementiert `IWonderDog`. Die `CTerrier`-Klasse verzichtet auf eine Implementierung. Ein Client kann ein aus diesen Klassen erzeugtes Objekt zur Laufzeit überprüfen und ermitteln, ob es eine bestimmte Schnittstelle unterstützt. Wenn das Objekt die Schnittstelle unterstützt, kann der Client ihre Funktionalität nutzen. Wenn das Objekt die Schnittstelle nicht unterstützt, ist dies nicht möglich. Der folgende Code demonstriert die Verwendung der `TypeOf`-Syntax von Visual Basic. Diese Syntax prüft, ob die `IWonderDog`-Schnittstelle unterstützt wird.

```
Dim DogA As IDog, DogB As IDog
```

```
Set DogA = New CBeagle
```

```
Set DogB = New CTerrier
```

```
If TypeOf DogA Is IWonderDog Then
```

```
    Dim WonderDogA As IWonderDog
```

```
    Set WonderDogA = DogA
```

```
    WonderDogA.FetchSlippers
```

```
End If
```

```
If TypeOf DogB Is IWonderDog Then
```

```
    Dim WonderDogB As IWonderDog
```

```
    Set WonderDogB = DogB
```

```
    WonderDogB.FetchSlippers
```

```
End If
```

Wenn der Client das `CBeagle`-Objekt abfragt, stellt er fest, dass es zu `IWonderDog` kompatibel ist. Das Objekt unterstützt somit die `IWonderDog`-Schnittstelle. Der Client kann daraufhin eine `IWonderDog`-Referenz erstellen und diese dem `CBeagle`-Objekt zuweisen, indem er die `IDog`-Referenz mit der `Set`-Anweisung bildet. Nachdem der Client über eine `IWonderDog`-Referenz verfügt, kann er die `FetchSlippers`-Methode aufrufen. Beachten Sie, dass zwei Referenzen auf dasselbe Objekt verweisen. Wenn Sie Klassen verwenden, die mehrere Schnittstellen implementieren, wird der Client-Code komplexer, da mehrere Referenzen auf ein einzelnes Objekt erforderlich sind, um die gesamte Funktionalität zu erhalten.

Wenn das `CTerrier`-Objekt auf seine Kompatibilität mit der Schnittstelle `IWonderDog` hin abgefragt wird, stellt der Client fest, dass sie nicht unterstützt wird. Eine entsprechende Referenz wird somit nicht gebildet. Der Client-Code kann dann alle Elemente der Auflistung des `IDog`-kompatiblen Objekts bearbeiten. Im nächsten Beispiel wird die `FetchSlippers`-Methode der Objekte aufgerufen, die die `IWonderDog`-Schnittstelle unterstützen.

```
Dim Dog As IDog, WonderDog As IWonderDog
```

```
For Each Dog In Dogs
```

```

    If TypeOf Dog Is IWonderDog Then
        Set WonderDog = Dog
        WonderDog.FetchSlippers
    End If
Next Dog

```

Wie Sie sich bestimmt vorstellen können, ist diese Möglichkeit zur Ermittlung der Funktionalität eines Objekts zur Laufzeit sehr nützlich, wenn Sie eine Applikation verbessern möchten. Wenn eine spätere Version der CBoxer-Klasse die IWonderDog-Schnittstelle implementiert, kann die zuvor aufgeführte For Each-Schleife die Vorteile dieser Möglichkeit nutzen, ohne umgeschrieben werden zu müssen. Der Client-Code kann die in zukünftigen Versionen des Objekts unterstützte Funktionalität somit bereits jetzt berücksichtigen.

Klassendefinitionen erweitern

Das obige Beispiel zeigt, wie ein Objekt verwendet wird, das mehrere Schnittstelle unterstützt. Sie können auch benutzerdefinierte Schnittstellen einsetzen, um das Verhalten eines Objekts problemlos zu erweitern, wenn ein bereits existierender Satz von Methodensignaturen zu große Einschränkungen bedingt. Die IDog-Schnittstelle kann beispielsweise die RollOver-Methode wie folgt definieren:

```

Public Sub RollOver(ByRef Rolls As Integer)
End Sub

```

Wenn Sie die Funktionalität des Dog-Objekts in der Applikation erweitern müssen, so dass Clients zum Beispiel in die Lage versetzt werden, größere Integer-Werte zu übergeben, könnten Sie eine zweite Schnittstelle mit der Bezeichnung IDog2 erstellen. Stellen Sie sich vor, dass die IDog2-Schnittstelle dieselben Elemente wie IDog, mit Ausnahme der RollOver-Methode, definiert, die wie folgt angegeben wird:

```

Public Sub RollOver(ByRef Rolls As Long)
End Sub

```

Ein neuer Client kann nun überprüfen, ob ein IDog-Objekt das neue Verhalten unterstützt. Wird das neue Verhalten nicht unterstützt, kann der Client zum ursprünglichen Verhalten zurückkehren. Das nachfolgende Beispiel demonstriert diesen Sachverhalt:

```

Sub ExerciseDog(Dog As IDog)
    If TypeOf Dog Is IDog2 Then
        ' Neues Verhalten verwenden, sofern unterstützt
        Dim Dog2 As IDog2, lRolls As Long
        Set Dog2 = Dog
        lRolls = 50000
        Dog2.RollOver lRolls
    Else
        ' Altes Verhalten verwenden, sofern notwendig
        Dim iRolls As Integer
        iRolls = 20000
        Dog.RollOver lRolls
    End If
End Sub

```

Diese Art der Versionsüberprüfung ermöglicht es Ihnen, neue Clients und Objekte in eine Applikation einzuführen, ohne ältere Clients und Objekte zu beeinträchtigen. Eine neue Klasse kann ältere Clients berücksichtigen, indem sie die Schnittstellen früherer Versionen unterstützt. Neue Clients arbeiten mit älteren Klassen, indem sie die älteren Schnittstellen verwenden, sofern dies erforderlich ist. In einer Welt ohne Schnittstellen erfordert die Erweiterung von Objekten häufig die Anpassung aller Clients. Das Anpassen der Clients führt wiederum zum Anpassen aller Objekte. Die Versionsüberprüfung, die durch die schnittstellenbasierte Programmierung möglich wird, erlaubt Ihnen, Änderungen an einer Applikation vorzunehmen, die sich nur geringfügig oder gar nicht auf den Code auswirken.

Verwenden von Schnittstellen in Ihrem Applikations-Design

Dieses Kapitel hat eine einfache Applikation präsentiert, um die Kernkonzepte der schnittstellenbasierten Programmierung zu demonstrieren. Wie aber können Sie diese Konzepte in eine reale Anwendung umsetzen? Wenn Sie eine große Applikation entwerfen, die Kundendaten aus vielen verschiedenen DBM-Systemen verwenden soll, ist es häufig erforderlich, für jedes System separaten Code zu erstellen. Sie können jedoch eine benutzerdefinierte Schnittstelle (`ICustomerDataManager`) erstellen und Client-Code zur Realisierung der Geschäftslogik für diese Schnittstelle schreiben, anstatt eine konkrete `CCustomerDataManager`-Klasse zu verwenden. Auf diese Weise ist es viel einfacher, Ihre Applikation auf unterschiedliche DMS-Systeme zu portieren. Wenn Sie mehrere Klassen erstellen, die die `ICustomerDataManager`-Schnittstelle implementieren, können Sie die Plug & Play-Vorzüge der Polymorphie nutzen. Verschiedene Objekttypen weisen unterschiedliche Verhaltensweisen auf, werden jedoch über dieselbe Schnittstelle kontrolliert.

Vom Standpunkt der Versionsüberprüfung aus betrachtet, ermöglicht Ihnen dieses Design, das Verhalten verschiedener Klassen zu verbessern, indem der Applikation neue Schnittstellen hinzugefügt werden. Schnittstellen wie `ICustomerDataManager2`, `ICustomerDataManager3` und `ICustomerDataManager4` ermöglichen die problemlose Erweiterung des Objektverhaltens. Das Beste daran ist, dass Sie Clients und Objekte unabhängig voneinander überarbeiten können. Ältere Clients und Objekte können frühere Schnittstellen verwenden, während gleichzeitig aktuellere Clients und Objekte über neue Schnittstellen miteinander kommunizieren. All dies wird erst durch die Laufzeittypenüberprüfung der Schnittstellenunterstützung möglich.

Schnittstellen und COM

Die Industrie hat die schnittstellenbasierte Programmierung aufgrund der Einschränkungen anderer Techniken übernommen. Zu diesen Techniken zählen auch die Verwendung klassenbasierter Referenzen und die Implementierungsvererbung. Benutzerdefinierte Schnittstellen führen eine neue Komplexitätsebene in das Applikations-Design und die Programmierung ein. (Und das ist der einzige Aspekt, in dem sich Microsoft und Sun einig sind.) Der Wert dieser Schnittstellen wird jedoch in großen Applikationen offensichtlich. In einem darwinschen Zusammenhang betrachtet, bereitet die schnittstellenbasierte Programmierung die Software auf das Überleben vor. Schnittstellen heben die Abhängigkeiten auf, die verschiedene Software-Teile verknüpfen. Diese Loslösung vereinfacht die Wiederverwendung, Wartung und Erweiterung Ihres Codes.

Das nächste Kapitel präsentiert die Grundlagen von COM. Wie Sie sehen werden, basiert COM auf den nachfolgend aufgeführten Kernkonzepten der schnittstellenbasierten Programmierung.

- **COM erfordert eine formelle Abtrennung der Schnittstelle von der Implementierung.** Clients dürfen ausschließlich über Schnittstellenreferenzen mit Objekten kommunizieren. Damit ist gewährleistet, dass die Clients niemals von den Komponenten (Klassen) abhängig

werden. Auf diese Weise können Programmierer Ihren Objektcode überarbeiten, ohne den Client-Code zu beeinträchtigen.

- **COM-Clients können von den Objekten Informationen über die Laufzeittypen abfragen.** Ein COM-Client kann immer ein Objekt abfragen und feststellen, ob es eine bestimmte Schnittstelle unterstützt. Wenn die angefragte Schnittstelle nicht unterstützt wird, kann der Client entsprechend darauf reagieren. Auf diese Weise können die Programmierer Komponenten und Applikationen unabhängig voneinander überprüfen. Ältere Clients und ältere Objekte können somit harmonisch mit neuen Clients und neuen Objekten zusammenarbeiten. Dies ist der Schlüssel zur Versionsüberprüfung in COM.

Zusammenfassung

Dieses Kapitel hat den Einsatz von Schnittstellen in einer einzelnen Applikation beschrieben. Die gesamte Anwendung wurde in einer einzigen Sprache geschrieben, und der gesamte Quellcode wurde gleichzeitig kompiliert. COM muss jedoch über die Grenzen binärer Komponenten hinweg arbeiten. Darüber hinaus können COM-Clients und COM-Objekte in unterschiedlichen Sprachen erstellt und in unterschiedlichen Prozessen auf unterschiedlichen Computern ausgeführt werden. COM muss viele Probleme auf der physischen Ebene lösen, um die Vorteile der schnittstellenbasierten Programmierung in einer verteilten Umgebung nutzen zu können.

3 COM-Grundlagen

46	Die Entstehung von COM
51	Typbibliotheken und Sprachunabhängigkeit
58	Komponentencode dynamisch laden
64	IUnknown und IDispatch verstehen
73	COM außerhalb von Prozessen

Dieses Kapitel betrachtet die wesentlichen Konzepte und Details von COM (Component Object Model). Hier erfahren Sie, warum Microsoft COM entwickelt hat und welche Design-Ziele damit ursprünglich verfolgt wurden. Wie Sie sehen werden, verbindet COM Clients und Objekte unter Verwendung der Konzepte aus der schnittstellenbasierten Programmierung. Auf der physischen Ebene müssen viele Details berücksichtigt werden, wenn Code für eine Applikation in mehrere Programmdateien verteilt wird und die Interaktion zwischen Client und Objekt über Prozess- und Rechengrenzen hinweg erfolgt.

Dieses Kapitel erklärt, wie die COM-Architekten verschiedene Sprachen in COM integrieren konnten. Zu diesen Sprachen zählt auch Visual Basic. Sie werden erfahren, wie der Visual Basic-Compiler mit einem speziellen Dateityp umgeht, der so genannten Typbibliothek. Typbibliotheken gestatten Sprachen wie Visual Basic, auf COM-Objekte zuzugreifen. Sie werden lernen, wie und warum der Visual Basic-Compiler in jeder ActiveX-DLL und ActiveX-EXE eine Typbibliothek erstellt. Außerdem erklärt das Kapitel, wie Scripting-Clients, zum Beispiel VBScript und JavaScript, auf COM-Objekte zugreifen, indem sie einen bestimmten Mechanismus, die so genannte Automatisierung, nutzen.

In COM wird der Code, der für die Clients geschrieben wird, und der Code, der die Klassen definiert, häufig in separaten Binärdateien aufbewahrt. Die Komponenten zeigen den Clients ihre Implementierungsdetails nicht, Clients müssen aber weiterhin Objekte erzeugen. Doch wie kann eine Client-Applikation ein Objekt erzeugen, wenn dessen Server keine sichtbare, konkrete Klassendefinition bereitstellt? COM verfügt über einen Mechanismus zum Laden von Klassen, den die Clients aufrufen, wenn sie Objekte erzeugen oder aktivieren wollen. Die Clients benötigen dazu keine Informationen über die Implementierungsdetails der Komponenten. Dieses Kapitel erklärt die Dienste und Design-Anforderungen, die die Objektaktivierung ermöglichen.

Die Entstehung von COM

Der Grundstein für COM wurde 1988 gelegt, als mehrere verschiedene Microsoft-Teams mit der Erstellung einer objektorientierten Infrastruktur begannen, die eine auf Komponenten basierende Wiederverwendbarkeit unterstützte. Als eine Gruppe von Ingenieuren aus mehreren Teams zusammengestellt wurde, um beim OLE2-Projekt auszuhelfen, entschlossen diese sich, einige höhere Anforderungen für die zu erstellende Architektur zu erarbeiten.

Diese Ingenieure definierten vier Anforderungen, die von der neuen Komponentenarchitektur erfüllt werden sollten. Erstens sollte die Architektur auf der *binären Wiederverwendung* basieren, weil die Wartung und Erweiterung von Code in großen, monolithischen Applikationen sehr kompliziert sein kann. Systeme, die auf binären Komponenten basieren, können sehr viel einfacher zusammengesetzt, gewartet und erweitert werden.

Zweitens sollte die Architektur auf dem objektorientierten Paradigma aufsetzen. Ein Großteil der binären Wiederverwendbarkeit in Microsoft Windows basierte auf herkömmlichen DLLs, die in der Regel nicht über objektorientierte Erweiterungen verfügten. Diese Art der binären Wiederverwendbarkeit konnte nicht von der Kapselung profitieren, die mit einem klassenbasierten Design erzielt wird.

Drittens sollte die Architektur *sprachunabhängig* sein. Man ging davon aus, dass sie sehr viel leistungsfähiger sei, wenn die Komponentenautoren die gewünschte Programmiersprache voneinander unabhängig auswählen könnten. Jede Programmiersprache zwingt den Programmierer zu Kompromissen hinsichtlich der Produktivität, Flexibilität und Leistung. Die Möglichkeit, für jede Komponente eine eigene Sprache auszuwählen, bietet viele Vorteile, verglichen mit der Verwendung einer einzigen Sprache für die gesamte Applikation.

Und schließlich sollte die Architektur die *Interprozesskommunikation* ermöglichen. Systeme sollten aus Clients und Objekten zusammengesetzt werden können, die auf unterschiedlichen Computern ausgeführt werden, so dass die Architektur eine Grundlage für verteilte Technologien bilden kann. Die Ingenieure wussten, dass ihre Architektur sehr viel produktiver sein würde, wenn sie die Details der Interprozesskommunikation vor den meisten Programmierern verstecken könnten.

Die Bemühungen der Gruppe resultierten in Microsofts zweitem großen Release von OLE (Object Linking and Embedding). Diese Technologie erlaubt den Anwendern, die mit einer bestimmten Applikation erstellten Dokumente mit Dokumenten anderer Applikationen zu verknüpfen oder darin einzubetten. Für OLE ist eine Interprozesskommunikation zwischen den Applikationen erforderlich. Die ursprüngliche Version von OLE basierte auf einem Interprozessmechanismus, der als DDE (Dynamic Data Exchange) bezeichnet wurde. Dieser erste Release wies sehr viele Probleme in Hinblick auf Ressourcen und Leistung auf, so dass er sich in der Industrie nicht wirklich durchsetzen konnte. Microsoft wusste, dass die OLE-Funktionalität grundsätzlich sehr praktisch war, dass aber gleichzeitig die zugrunde liegende Infrastruktur optimiert werden musste, wenn die Technologie für die Betriebsumgebung gerüstet sein sollte. Letztlich brauchte OLE ein völlig neues Konfigurationssystem, wenn es überlebensfähig sein sollte.

1993 wurde OLE2 mit der ersten COM-Generation ausgeliefert. OLE und COM sind unterschiedliche Einheiten. OLE ist eine Technologie für das Verknüpfen und Einbetten von Dokumenten, während COM eine Architektur für die Erstellung komponentenbasierter Systeme ist. Leider wurden die beiden Technologien häufig als eine einzelne Komponente missverstanden. Und wie das Schicksal es wollte, wurde COM irgendwann viel wichtiger als OLE, die Technologie, die es ursprünglich unterstützen sollte.

Die Entwicklung binärer Komponenten mit C++

Die COM-Architekten entwarfen eine Methode, Applikationen basierend auf Komponenten zusammenzusetzen, die in separaten binären Dateien vorhanden sind. Das war jedoch nicht ganz einfach. Man musste während der Entwicklung einige schwierige Entscheidungen treffen. Eine der wichtigsten Entscheidungen war, dass es nicht möglich sein sollte, den Client-Code unter Verwendung klassenbasierter Referenzen zu schreiben. Die dadurch entstehende Verbindung würde sich zu negativ auswirken. Es war offensichtlich, dass dieses neue Objektmodell eine formale Abtrennung der Schnittstelle von der Implementierung realisieren musste.

Schon lange bevor irgendein Visual Basic- oder Java-Programmierer Schlüsselwörter wie *Implements* und *Interface* verwendete, entwickelte die C++-Gemeinde Applikationen nach dem Konzept der schnittstellenbasierten Programmierung. C++ bietet zwar keine direkte Unterstützung dieses Programmierstils, aber mit Hilfe der erweiterten Funktionsmerkmale der Sprache können Sie abstrakte Datentypen (Schnittstellen) erzeugen. Eine C++-Klasse kann Methodensignaturen ohne Implementierung definieren. Man spricht dann auch von einer *abstrakten Basisklasse*. In Kürze werden Sie eine davon kennen lernen, zunächst aber wollen wir betrachten, warum die C++-Entwickler überhaupt abstrakte Basisklassen brauchten.

C++ weist einen schwerwiegenden Nachteil auf, der deutlich wird, wenn komponentenbasierte Applikationen entwickelt werden: Es unterstützt keine Kapselung auf der binären Ebene, weil die Sprache ursprünglich für die Erstellung monolithischer Applikationen entwickelt wurde. Wenn ein C++-Klassenautor ein Datenelement als privat markiert, kann der Client-Code wie erwartet nicht darauf zugreifen. Diese Kapselung wird jedoch nur in der Syntax der Sprache erzwungen. Ein Client, der `New` für eine Klasse aufruft, muss Kenntnisse über das Datenlayout des Objekts besitzen. Der Client ist für das Reservieren des Speichers verantwortlich, in dem das Objekt ausgeführt wird. Das bedeutet, dass der Client von jedem Datenelement einer Klasse wissen muss, ob dies nun öffentlich, geschützt oder privat ist.

Diese Layout-Abhängigkeiten sind in einer monolithischen Applikation kein Problem, da der Client immer die aktuelle Version der Klasse rekompiliert. In der komponentenbasierten Entwicklung dagegen sind sie sehr wohl ein Problem. Das Layout einer Klasse innerhalb einer DLL kann somit nicht verändert werden, ohne die Funktion der Clients einzuschränken, die diese Klasse verwenden. Der folgende C++-Code zeigt einen Client und zwei Versionen einer Klasse, die sich in einer DLL befinden.

```
// In SERVER.DLL - Version 1
// Jedes Objekt belegt 8 Byte Speicher.
class Cdog {
public:
    void Bark(){ /* Implementierung */ }
    void RollOver(int Rolls){ /* Implementierung */ }
private:
    double Weight;
};

// In CLIENT.EXE
// Kompiliert mit DLL-Version 1
// Der Client reserviert 8 Byte für jedes Objekt.
CDog* pDog = new CDog;

// In SERVER.DLL - Version 2
// Jedes Objekt benötigt 16 Byte Speicher.
// Durch das Ersetzen der älteren DLL wird der Client unbrauchbar.
```

```

class Cdog {
public:
    void Bark(){ /* Implementierung */ }
    void RollOver(int Rolls){ /* Implementierung */ }
private:
    double Weight;
    double Age;
};

```

Wenn die erste Version der DLL durch die zweite Version ersetzt wird, tritt ein großes Problem auf. Die Client-Applikation erstellt weiterhin Objekte, die 8 Bytes belegen, doch jedes Objekt glaubt, 16 Bytes groß zu sein. Damit ist die Katastrophe vorprogrammiert. Angenommen, Sie haben die Implementierung von `Bark` in Version 2 umgeschrieben, so dass sie jetzt die `Age`-Eigenschaft abfragt, um die Bellintensität des Hunds zu bestimmen. Die neue Version des Objekts versucht auf einen Speicherbereich zuzugreifen, über den sie nicht verfügt. Dies führt voraussichtlich dazu, dass sich die Applikation seltsam und unvorhersehbar verhält. Die einzige Möglichkeit, dieses Problem zu beheben, besteht darin, alle Client-Applikationen erneut zu erstellen, wenn das Datenlayout des Objekts verändert wird. Dadurch wird einer der wichtigsten Vorzüge der komponentenbasierten Entwicklung zunichte gemacht: Die Möglichkeit, binäre Dateien abzuändern, ohne dass dies Auswirkungen auf andere Dateien hat.

Wie können Sie die DLL ersetzen, ohne die Client-Applikationen, die die DLL verwenden, unbrauchbar werden zu lassen? Dazu muss das Datenlayout des Objekts vor dem Client versteckt werden. Der Client muss außerdem von der Verantwortung befreit werden, den C++-Operator `New` über eine binäre Firewall hinweg aufzurufen. Irgendjemand muss außerhalb der Client-Applikation die Verantwortung übernehmen, das Objekt zu erzeugen.

Letztlich läuft es auf ein Ergebnis hinaus: Die Sprache C++ ist veraltet. Ihre Entwickler haben sie nie auf die Entwicklung komponentenbasierter Applikationen ausgelegt. Glücklicherweise sind die C++-Programmierer die cleversten überhaupt. (Fragen Sie ruhig nach, sie werden es Ihnen bestätigen.) Sie haben entdeckt, dass es durch abstrakte Basisklassen möglich ist, eine Schnittstelle von der Implementierung abzutrennen. Abstrakte Basisklassen erlauben den C++-Programmierern, Client-Applikationen ohne Layout-Abhängigkeiten von den verwendeten konkreten Klassen zu erstellen. Und das wiederum erlaubt es Klassenautoren, ihre Layout-Details in jeder Version nach Belieben neu zu überarbeiten.

Abstrakte Basisklassen als Schnittstellen

In C++ wird eine abstrakte Basisklasse benutzt, um Methodensignaturen zu definieren, die keine Implementierung enthalten. Hier folgt ein Beispiel für eine abstrakte Basisklasse in C++.

```

class IDog{
    virtual void Bark() = 0;
    virtual void RollOver(int Rolls) = 0;
};

```

Eine C++-Elementfunktion (eine Methode), die mit `= 0` deklariert wird, ist eine rein virtuelle Funktion. Dies bedeutet, sie enthält keine Implementierung und ist keine erstellbare Klasse. Die einzige Möglichkeit, eine abstrakte Basisklasse zu verwenden, bietet deshalb die Vererbung, wie nachfolgend gezeigt.

```

class CBeagle: public IDog{
    virtual void Bark()
        { /* Implementierung */ }
    virtual void RollOver(int Rolls);
        { /* Implementierung */ }
};

```

C++-Programmierer, die abstrakte Basisklassen verwenden und zusätzlich eine kleine Regel befolgen, können die Vorzüge der Wiederverwendbarkeit, Wartungsfähigkeit und Erweiterbarkeit nutzen, die bereits im zweiten Kapitel beschrieben wurden. Die zusätzliche Regel besteht darin, die Definition des Datenspeichers und der Methodenimplementierung innerhalb einer abstrakten Basisklasse zu vermeiden. Wenn diese Regel befolgt wird, kann eine abstrakte Basisklasse die Abtrennung der Schnittstelle von der Implementierung vereinfachen. Eine C++-Client-Applikation kann beispielsweise mit einem CBeagle-Objekt über eine IDog-Referenz kommunizieren, so dass die Client-Applikation keine Layout-Abhängigkeiten von der CBeagle-Klasse aufweisen muss.

Wie Sie sehen, kann eine abstrakte C++-Basisklasse als logische Schnittstelle genutzt werden. Obwohl die Sprache die schnittstellenbasierte Programmierung nicht unterstützt, ermöglichen Ihnen fortgeschrittene Techniken, die wichtigsten Vorteile von Schnittstellen zu nutzen. Es waren sogar C++-Programmierer, die zuerst das Konzept der schnittstellenbasierten Programmierung unter Verwendung von abstrakten Basisklassen in großen komponentenbasierten Applikationen erkundeten. Diese Technik wurde seitdem in unzähligen Projekten eingesetzt und ist ausführlich in dem Buch *Large-Scale C++ Software Design* von John S. Lakos beschrieben. Die Grundlagen der in C++ implementierten schnittstellenbasierten Programmierung hatten einen wesentlichen Einfluss auf die Entwicklung von COM.

Die vTable als speicherinterne Standardrepräsentation

Die Entwickler von COM erkannten, dass sie die objektorientierte binäre Wiederverwendbarkeit umsetzen könnten, wenn sie abstrakte Basisklassen verwenden würden. DLLs könnten konkrete Klassendefinitionen enthalten. Client-Applikationen könnten Objekte verwenden, die aus diesen DLLs erstellt würden, vorausgesetzt, sie kommunizierten über abstrakte Basisklassen miteinander. DLLs könnten nach der Inbetriebnahme überarbeitet und ersetzt werden, ohne dass sich das nachteilig auf bereits existierende Client-Applikationen auswirken würde. Diese Schlussfolgerungen waren Meilensteine für die Ingenieure bei Microsoft. Sie hatten eine Möglichkeit geschaffen, die binäre Wiederverwendbarkeit mit C++ in einem objektorientierten Paradigma umzusetzen. Damit waren die beiden ersten Design-Ziele erreicht.

Die schnittstellenbasierte Programmierung setzt auf dem Konzept der Polymorphie auf. Wie Sie aus dem vorhergehenden Kapitel wissen, ist Polymorphie eine kontrollierte Form der Unsicherheit, wobei der Client weiß, welche Methode er aufrufen soll. Er weiß jedoch nicht und kümmert sich auch nicht darum, wie das betreffende Objekt die Methode implementiert. Jede Sprache, die Polymorphie unterstützt, muss eine Möglichkeit bieten, Clients dynamisch an eine von mehreren möglichen Implementierungen zu binden. Sofern der Client für eine bekannte Schnittstelle programmiert wurde, kann eine Applikation jedes typkompatible Objekt zur Laufzeit einsetzen.

Unterschiedliche Sprachen und Compiler realisieren die Anforderungen des *dynamischen Bindens* auf unterschiedliche Weise. Weil C++ auf unterster Ebene aufsetzt, besitzt es eine der schnellsten Bindetechniken, die es gibt. Das dynamische Binden basiert in C++ auf einer höchst effizienten Dispatching-Architektur, die *virtuelle Funktionen* einsetzt.

Der C++-Compiler und der Linker ermöglichen das dynamische Binden, indem sie ein Array aus Funktionszeigern generieren, das als *vTable* (virtuelle Tabelle) bezeichnet wird. Eine *vTable* repräsentiert eine Gruppe von Einsprungpunkten in ein Objekt. Jeder Methode, die in einer abstrakten Basisklasse definiert ist, wird ein Zeiger in der *vTable* zugeordnet. Wenn der Client die Aufrufsyntax einer bestimmten Methode kennt und den entsprechenden Funktionszeiger auf diese Methode verwendet, kann er auf das Objekt zugreifen, ohne die konkrete Klasse zu kennen, aus der das Objekt erzeugt wurde.

An dieser Stelle trafen die COM-Architekten eine weitere wichtige Entscheidung. Sie beschloßen, die *vTables* im Stil von C++ als speicherinterne Standarddarstellung für die gesamte Kommunikation zwischen Client und Objekt zu verwenden. Alle Clients müssen eine Bindung zu den *vTables* vornehmen und die Methoden mit Hilfe der dort bereitgestellten Funktionszeiger aufrufen. Man spricht dabei auch von einer *direkten vTable-Bindung*.

Abbildung 3.1 zeigt den Aufbau einer *vTable*. Grundsätzlich gibt es dabei keinen Unterschied zwischen C++ und COM. Eine *vTable* ist eine Standard-Speicherdarstellung, auf die sich Client und Objekt geeinigt haben. Auf der rechten Seite der Abbildung sehen Sie die logische Darstellung eines COM-Clients, der auf ein COM-Objekt zugreift. Jeder der kleinen Kreise stellt eine von dem COM-Objekt unterstützte Schnittstelle dar. Beachten Sie, dass ein Objekt mehrere Schnittstellen unterstützen kann. Ein Objekt muss für jede von ihm implementierte Schnittstelle eine eigene *vTable* bereitstellen.

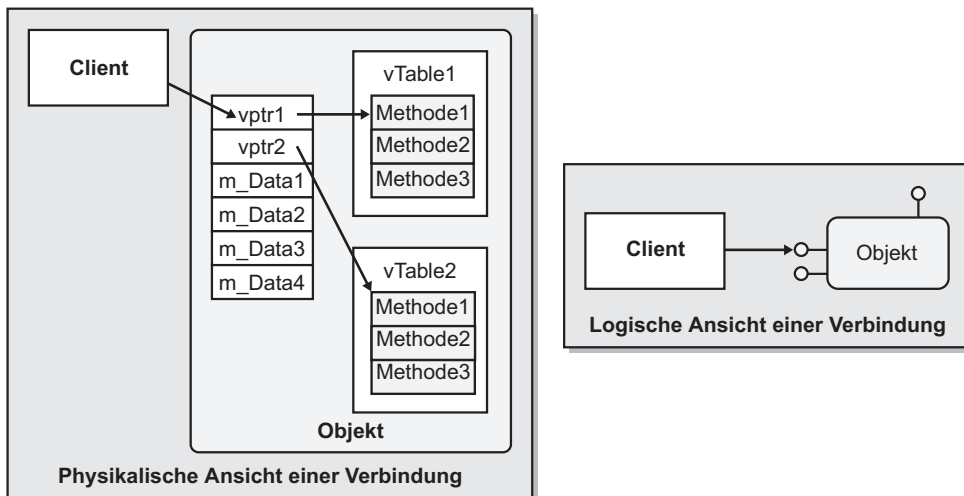


Abbildung 3.1: In C++ und COM werden Clients über *vTables* dynamisch an Objekte gebunden. Jede physische *vTable* repräsentiert eine logische Schnittstelle, die von einem Objekt implementiert wird.

Einige Zeit nach Abschluss des OLE2-Projekts definierte ein Team von Entwicklern, die die COM-Architektur entworfen hatten, eine Spezifikation für ihre Arbeit. Dieses Dokument, auch als *COM-Spezifikation* bezeichnet, definiert die Regeln für die COM-Programmierung. Außerdem definiert es einen Großteil der wichtigen Terminologie.

COM-Objekte werden aus COM-konformen Klassen instanziiert, die man auch als *Co-Klassen* bezeichnet. Eine Co-Klasse ist eine konkrete Implementierung einer oder mehrerer Schnittstellen. Beachten Sie, dass die Begriffe Co-Klasse und Komponente synonym verwendet werden können.

Eine Komponente ist ein erstellbarer Datentyp, aus dem ein Client Objekte erzeugt. In der Praxis legt man für jede Komponente, die man erstellen will, eine *MultiUse*-Klasse an.

Co-Klassen werden zu binären Dateien, so genannten *Servern*, kompiliert und als solche eingesetzt. Der Server enthält den Implementierungscode für eine oder mehrere Co-Klassen. Er muss zusätzlich die COM-Infrastruktur für die Objektaktivierung unterstützen. Wie Sie noch erfahren werden, stellt ein Server zu diesem Zweck bekannte Einsprungpunkte zur Verfügung.

Manche Anwender verwenden den Begriff *Komponente* für eine physische Datei, beispielsweise eine ActiveX-DLL, aber ich definiere eine Komponente als erzeugbare Klasse, die zu einem Server kompiliert wird. Die ActiveX-DLL ist der Server, nicht die Komponente. Um die Verwirrung aufzuklären, stellen Sie sich die folgende Frage: Wenn Sie zwei *MultiUse*-Klassen zu einer ActiveX-DLL kompilieren, haben Sie dann eine Komponente oder zwei? Ich denke, es sind zwei. Andere dagegen sagen, es sei eine. Seien Sie sich aber bewusst, dass hier keine Konsistenz zu erwarten ist. In diesem Buch werden die Begriffe *Komponente*, *Co-Klasse* und *konkrete Klasse* synonym verwendet. Es handelt sich dabei um etwas, das Sie brauchen, um ein Objekt zu erzeugen. Diese Verwendung ist konsistent mit den Programmiermodellen von MTS und COM+.

Typbibliotheken und Sprachunabhängigkeit

In COM kommunizieren Clients und Objekte über die vTable-Bindung miteinander. Das bedeutet, COM-Objekte und COM-Clients müssen die Verwendung von Funktionszeigern beherrschen. Glücklicherweise hilft der Compiler den C++-Programmierern und erledigt die meiste Arbeit im Hintergrund. Der C++-Compiler erzeugt auf der Objektseite automatisch vTables und füllt diese. Außerdem kann der C++-Compiler auch den Code erzeugen, der auf der Clientseite für das Binden benötigt wird und mit dem Methoden über die vTable-Funktionszeiger aufgerufen werden. Die clientseitigen Programmierer müssen ihre Arbeit dann nur noch für die Header-Datei kompilieren, die die Definition der abstrakten Basisklasse enthält.

Doch was ist mit den Programmierern, die mit Visual Basic, Java oder Skriptsprachen arbeiten? Viele Entwicklungswerkzeuge und Sprachen wie diese verfügen nicht über eine integrierte Unterstützung für die Arbeit mit Funktionszeigern. Um COM-Objekte zu erstellen oder zu benutzen, muss eine Programmiersprache die Regeln der COM-Spezifikationen befolgen. Diese Regeln verlangen die vTable-Bindung zur Durchführung der Client-/Objekt-Kommunikation. Viele der höheren Werkzeuge und Sprachen benötigen Hilfe, um an der COM-Programmierung teilnehmen zu können.

Visual Basic stellt diese Hilfe zur Verfügung, indem es eine entsprechende Unterstützung in den Compiler integriert und seiner Laufzeit-Engine eine Visual Basic-zu-COM-Zuordnungsschicht hinzufügt. Schließlich definieren die COM-Spezifikationen eindeutige Regeln. Das Visual Basic-Team wusste genau, was nötig war, um Objekte vTable-konform werden zu lassen. Auf der Client-Seite erstellt der Visual Basic-Compiler automatisch den vTable-Bindungscode, der für den Zugriff auf ein COM-Objekt erforderlich ist. Wenn eine Sprache oder ein Werkzeug eine Zuordnungsschicht verwendet, kann sie/es glücklicherweise eine Vielzahl der zugrunde liegenden Details vor den Programmierern verstecken. Dies ist der Grund dafür, dass die Programmierung von COM mit Visual Basic sehr viel einfacher als mit C++ ist.

Der Visual Basic-Compiler kann nichts mit einer C++-Header-Datei anfangen. Er kann also während des Kompilierens keine abstrakte Basisklasse nutzen, um den Bindungscode für die Clientseite abzuleiten. Stattdessen verwendet Visual Basic Schnittstellendefinitionen, die in einem speziellen Datentyp bereitgestellt werden, den so genannten *Typbibliotheken*. Eine Typbibliothek

teilt dem Visual Basic-Compiler mit, wie eine vTable für eine bestimmte Schnittstelle aufgebaut sein muss.

Eine Typlibibliothek ist ein binärer Behälter, der Typinformationen über Elemente wie Schnittstellen und Co-Klassen aufnimmt. Im Wesentlichen handelt es sich dabei um einen Katalog, der eine Auflistung mit COM-Datentypen definiert. Innerhalb einer Typlibibliothek ist jede Schnittstelle als eine Gruppe mit verschiedenen Methoden definiert. Jede Co-Klasse ist als Gruppe mit einer oder mehreren implementierten Schnittstellen definiert. Wenn Sie eine ActiveX-DLL oder eine ActiveX-EXE kompilieren, erzeugt Visual Basic automatisch eine Typlibibliothek für Sie und fügt sie der ausführbaren Datei des Servers hinzu. Diese Typlibibliothek enthält Definitionen für die vom Server bereitgestellten Co-Klassen sowie eine Definition für jede im Projekt des Servers definierte Schnittstelle.

Beim Kompilieren einer Client-Applikation muss Visual Basic die Typlibibliothek des Servers einsehen können, um den Bindungscode für die Clientseite zu erzeugen. Wenn der Visual Basic-Compiler die Schnittstellendefinitionen aus einer Typlibibliothek lesen kann, kann er auch den für die direkte vTable-Bindung erforderlichen Code erstellen. Sie binden eine Typlibibliothek in ein Visual Basic-Projekt ein, indem Sie im Menü *Projekt* das Dialogfeld *Verweise* öffnen. Dieses Dialogfeld zeigt eine Liste aller auf Ihrer Entwicklungs-Workstation registrierten Typlibibliotheken an.

Wie aber legen Sie eine Typlibibliothek an? Die Programmierer verwenden unterschiedliche Vorgehensweisen. Unter Visual Basic könnte es nicht einfacher sein. Sie kompilieren Ihr Server-Projekt, und Visual Basic erzeugt die Typlibibliothek transparent im Hintergrund. Dazu wird Ihr Quellcode ausgewertet. Viele Programmierer legen die Typlibibliotheken mit Hilfe einer speziellen Sprache an, die als *IDL* (*Interface Definition Language*) bezeichnet wird.

Mit IDL arbeiten

Typlibibliotheken definieren Schnittstellen und Co-Klassen im binären Format. IDL verwendet hierzu ein lesbares Textformat. C++-Entwickler, die Typlibibliotheken anlegen müssen, verwenden dazu IDL. Nachdem ein Entwickler die Schnittstellen- und Co-Klassendefinitionen mit IDL erstellt hat, kann dem MIDL-Compiler (Microsoft IDL) die resultierende IDL-Quelldatei übergeben werden. Der MIDL-Compiler kann unterschiedliche Objekte erzeugen, unter anderem auch eine Typlibibliothek.

Wenn Visual Basic die Typlibibliotheken automatisch erzeugen kann, warum ist dies dann nicht mit dem C++-Compiler möglich? Als das COM-Team begann, die COM-Spezifikation festzulegen, wurde deutlich, dass C++ und C nicht für die Definition von COM-Schnittstellen geeignet war. C und C++ wurden nicht für die Definition von Funktionen ausgelegt, die über Prozessgrenzen hinweg aufgerufen werden können, weil die dort möglichen Parameterdefinitionen extrem ungenau sind. Worauf zeigt zum Beispiel ein Parameter, der von einer C++-Methode als Zeiger definiert wurde? Welche Daten müssen zwischen dem Client-Prozess und dem Objekt-Prozess ausgetauscht werden? Ein reiner Zeiger kann nicht Aufschluss darüber geben, welche Daten zwischen den Prozessen ausgetauscht werden sollen. IDL löst dieses Problem unter Verwendung einer Syntax, die Methodenparameter eindeutig beschreibt.

IDL ist C sehr ähnlich, weist aber einige objektorientierte Erweiterungen auf. Außerdem erlaubt es die Angabe von Attributen für bestimmte Elemente, wie beispielsweise Typlibibliotheken, Co-Klassen, Schnittstellen, Methoden und Parameter. Hier folgt ein einfaches Beispiel dafür, wie IDL aussehen kann:


```

library DogServerLib
{
    interface IDog {
        HRESULT Bark();
        HRESULT RollOver([in] int Rolls);
    };
    interface IWonderDog{
        HRESULT FetchSlippers();
    };
    coclass CBeagle {
        interface IDpg;
        interface IWonderDog;
    };
}

```

Dieses Beispiel zeigt, wie Typbibliotheken, Schnittstellen und Co-Klassen in IDL definiert werden. Beachten Sie bitte, dass jede Methodendefinition einen Rückgabewert vom Typ `HRESULT` definiert. Für COM ist es zwingend erforderlich, dass Methoden `HRESULT` zurückgeben, um eine standardisierte Infrastruktur für Fehlerberichte aufbauen zu können. Insbesondere in verteilten Umgebungen, wo entfernte Objekte aus den unterschiedlichsten Gründen ungültig werden oder nicht mehr erreichbar sein können, ist diese Infrastruktur sehr hilfreich. Kapitel 4 beschreibt die Fehlerweitergabe und die Fehlerbehebung detailliert. Dort werden wir auch noch einmal auf die Arbeitsweise von `HRESULT` zurückkommen.

Wenn C++-Programmierer Komponenten und Schnittstellen erstellen möchten, müssen sie zunächst eine IDL-Quelldatei definieren, die die entsprechenden Definitionen enthält, und diese dem MIDL-Compiler übergeben. Visual Basic-Programmierer müssen dies nicht tun, weil die Visual Basic-IDE die Typbibliotheken direkt aus dem Visual Basic-Code erzeugt. Visual Basic-Programmierer müssen somit niemals mit IDL arbeiten.

Sie können ein produktiver Visual Basic -Programmierer sein, ohne jemals in Berührung mit IDL zu kommen. Wenn Sie jedoch die Grundlagen von IDL kennen lernen, werden Sie zu einem besseren COM-Programmierer. Dies gilt besonders dann, wenn Sie wissen wollen, was hinter den Kulissen passiert, oder wenn Sie mit der Interoperabilität zwischen Komponenten konfrontiert werden, die in anderen Sprachen, wie zum Beispiel C++, geschrieben wurden. Wenn Sie IDL verstehen, erkennen Sie, wie Visual Basic im Hintergrund arbeitet, um COM zu unterstützen.

Ein COM-Dienstprogramm mit der Bezeichnung *Oleview.exe* hilft Ihnen, eine Typbibliothek in einen lesbaren textbasierten IDL-Code umzuwandeln. Abbildung 3.2 zeigt Ihnen, wie Sie das Programm einsetzen, um den IDL-Code von Co-Klassen und Schnittstellen zu untersuchen, die in Ihren ActiveX-DLLs und ActiveX-EXEs erstellt wurden. Verwenden Sie *Oleview.exe* im Expertenmodus (wählen Sie dazu im Menü *View* den Befehl *Expert Mode* aus), wenn Sie versuchen, eine Typbibliothek zu öffnen. Die Typbibliotheken werden andernfalls nicht im Hierarchie-Listenfeld von *Oleview.exe* angezeigt.

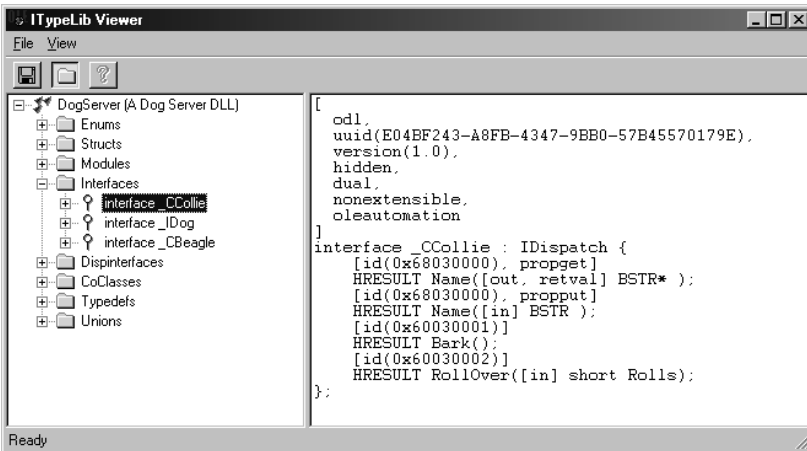


Abbildung 3.2: Oleview.exe erlaubt Ihnen, einige Aspekte Ihrer COM-Server zu untersuchen und zu modifizieren. Dieses Beispiel zeigt, wie Oleview.exe eine Typbibliothek in den ursprünglichen Zustand zurückversetzt.

Die Verbindung zwischen Visual Basic und COM

COM verlangt, dass jedes Objekt mindestens eine Schnittstelle implementiert. Visual Basic unterstützt dies, indem es in jeder erstellbaren Klasse eine Standardschnittstelle generiert. Alle öffentlichen Methoden und Eigenschaften einer Klasse werden in dieser *Standardschnittstelle* definiert. Stellen Sie sich beispielsweise vor, dass Sie eine Klasse `CCollie` (in einer ActiveX-DLL) mit der folgenden öffentlichen Schnittstelle verwenden.

```
Public Name As String

Public Sub Bark()
    ' Implementierung.
End Sub
```

Visual Basic erstellt aus den öffentlichen Methoden und Eigenschaften eine versteckte Schnittstelle mit der Bezeichnung `_CCollie`. Die Tatsache, dass diese Schnittstelle in der Typbibliothek als *hidden* (versteckt) deklariert wird, bedeutet, dass andere Visual Basic-Programmierer die Schnittstelle weder im Objektkatalog noch im IntelliSense-Bereich sehen können. (Visual Basic versteckt übrigens jede Typbezeichnung, die mit einem Unterstrich beginnt.) Visual Basic erstellt anschließend eine Co-Klasse mit der Bezeichnung `CCollie`, die `_CCollie` als *Standardschnittstelle* implementiert. Die grundlegende IDL sieht wie folgt aus:

```
[hidden]
interface _CCollie{
    [propget] HRESULT Name([out, retval] BSTR* Name);
    [propput] HRESULT Name([in] BSTR Name);
    HRESULT Bark();
};
coclass CCollie {
    [default] interface _CCollie;
};
```

Diese transparente 1:1-Zuordnung bewirkt, dass Visual Basic-Klassen ohne das Zutun des Programmierers COM-kompatibel werden. Visual Basic-Programmierer, die sich nicht mit dieser Thematik auseinander setzen, wissen somit nicht, was während der Klassenprogrammierung wirklich geschieht. Jeder Visual Basic-Client kann den folgenden Code enthalten, um auf die Klasse zuzugreifen.

```
Dim Dog As CCollie
Set Dog = New CCollie
Dog.Name = "Lassie"
Dog.Bark
```

In dem zuvor gezeigten Programmcode wird die Variable vom Typ `CCollie` transparent einer `_CCollie`-Referenz zugeordnet. Dies ist sinnvoll, weil der Client über eine Schnittstellenreferenz mit einem Objekt kommunizieren muss. Die COM-Programmierung mit Visual Basic wird dadurch außerdem einfacher. Solange eine 1:1-Zuordnung zwischen Ihren Klassen und den zu exportierenden Schnittstellen besteht, müssen Sie keine benutzerdefinierten Schnittstellen erzeugen. Wenn Sie jedoch keine benutzerdefinierten Schnittstellen verwenden, können Sie die Leistungsfähigkeit des schnittstellenbasierten Designs nicht nutzen.

Kapitel 2 hat Ihnen gezeigt, wie benutzerdefinierte Schnittstellen in eine Visual Basic-Applikation implementiert werden. Wenn Sie eine Schnittstelle in Visual Basic 5 mit einem *PublicNotCreatable*-Klassenmodul definieren und in eine Klasse implementieren, ergibt sich daraus ein IDL-Code, der wie folgt aussehen könnte:

```
interface IDog {
    HRESULT Name([out, retval] BSTR* Name);
    HRESULT Name([in] BSTR Name);
    HRESULT Bark();
};

interface _CBeagle {
};

coclass CBeagle {
    [default] interface _CBeagle;
    interface IDog;
};
```

Beachten Sie, dass `IDog` als implementierte Schnittstelle in der Klasse `CBeagle`, nicht aber als Standardschnittstelle definiert ist. Selbst wenn Ihre Klasse keine öffentlichen Elemente enthält, erstellt Visual Basic automatisch eine Standardschnittstelle mit dem Namen der Klasse, dem ein Unterstrich vorangestellt wird. Sie können dieses Verhalten nicht verändern, um eine andere Schnittstelle als Standard zu definieren. Im nächsten Kapitel erfahren Sie, dass Sie bisweilen eine gewisse Funktionalität in der Standardschnittstelle implementieren müssen, d.h. Sie müssen öffentliche Elemente in Ihre Klasse aufnehmen. Das ist immer dann der Fall, wenn Sie Visual Basic-Objekte für Skripting-Clients erstellen.

Visual Basic 6 arbeitet anders als Visual Basic 5. Auch wenn Sie ein Klassenmodul als *PublicNotCreatable* deklarieren, erzeugt Visual Basic 6 eine Co-Klasse und eine Standardschnittstelle. Wenn Sie beispielsweise die Schnittstelle `IDog` mit einer Eigenschaft und einer Methode in einem Klassenmodul erstellen, das mit *PublicNotCreatable* gekennzeichnet ist, ist der IDL-Code wie folgt aufgebaut:

```
[hidden]
interface _IDog : IDispatch {
    HRESULT Name([out, retval] BSTR* Name);
    HRESULT Name([in] BSTR Name);
    HRESULT Bark();
};

[noncreatable]
coclass IDog {
    [default] interface _IDog;
};
```

Das bedeutet, dass Sie in Visual Basic 6 nicht einfach eine COM-Schnittstelle erstellen können, ohne dass sich in der Typbibliothek eine entsprechende Co-Klasse befindet, wie Sie es von Visual Basic 5 gewöhnt sind. Klassenmodule, deren *Instancing*-Eigenschaft auf *PublicNotCreatable* gesetzt ist, generieren immer eine Schnittstelle und eine nicht erstellbare (*noncreatable*) Co-Klasse. Das Attribut *noncreatable* bedeutet, dass die Klasse nicht von einem COM-Client instanziiert werden kann. Code, der innerhalb desselben Server-Projekts enthalten ist, kann dagegen Objekte aus einer *PublicNotCreatable*-Klasse erzeugen. In Visual Basic 6 sollten Sie sich diese Klasse als eine »Public-Not-Externally-Creatable«-Klasse (öffentliche, nicht extern erstellbare Klasse) vorstellen. Es zeigt sich, dass die Unterschiede zwischen Visual Basic 5 und Visual Basic 6 in der Typbibliothek versteckt sind. Der Code, den Sie für Visual Basic 6 schreiben, unterscheidet sich nicht von dem Code, der für Visual Basic 5 geschrieben wird. In dem zuvor aufgeführten Beispiel ist *IDog* eine Co-Klasse und *_IDog* eine versteckte Schnittstelle. Immer wenn Sie den Typ *IDog* mit dem Schlüsselwort *Implements* verwenden oder wenn Sie *IDog* zum Erstellen von Objektreferenzen benutzen, wandelt Visual Basic *IDog* stillschweigend in *_IDog* um.

Beachten Sie, dass Ihre Klassen auch Schnittstellen implementieren können, die in Typbibliotheken definiert sind, auf die in Ihrem Projekt verwiesen wird. Um sie zu implementieren, müssen Sie die Typbibliothek in das Dialogfeld *Verweise* (im Menü *Projekt*) importieren. Diese Typbibliotheken können mit Visual Basic oder mit IDL und dem MIDL-Compiler erstellt werden. Sobald Ihr Projekt die Schnittstellendefinition erkennt, können Sie das Schlüsselwort *Implements* in einem Klassenmodul verwenden. In großen Projekten, für deren Design benutzerdefinierte Schnittstellen erforderlich sind, kann es sinnvoll sein, die Schnittstellendefinitionen in einer Typbibliothek zu verteilen, die unabhängig von den implementierenden Servern ist. In Kapitel 5 erfahren Sie, wie man Typbibliotheken mit Hilfe von IDL und dem MIDL-Compiler erstellt und verteilt. Insbesondere werden wir dabei darauf eingehen, in welchen Situationen Sie dies tun oder besser unterlassen sollten.

Globale eindeutige Bezeichner (GUIDs)

COM unterstützt ein Schema für die Identifizierung von Datentypen, wie zum Beispiel Schnittstellen und Co-Klassen, das geeigneter ist als die vom Menschen lesbaren Namen, wie z.B. *IDog* oder *CBeagle*. COM verwendet so genannte globale eindeutige Bezeichner (Globally Unique Identifiers – GUIDs). Ein GUID ist ein 128-Bit-Integer. Ein 128-Bit-Integer kann etwa $3,4 \cdot 10^{38}$ Werte darstellen, so dass Sie davon ausgehen können, dass diese Bezeichner für die Verwendung in COM unbegrenzt zur Verfügung stehen. Die meisten Compiler und Datenbanken unterstützen keine 128-Bit-Integer, weshalb GUIDs gewöhnlich in anderen Formaten gespeichert werden.

C++-Programmierer verwenden eine Struktur (gleichbedeutend mit einem benutzerdefinierten Typ in Visual Basic), die einen 128-Bit-Wert durch mehrere kleinere Ganzzahlen repräsentiert.

Ein GUID kann auch durch 32 Zeichen in hexadezimaler Form dargestellt werden, wodurch er leichter lesbar wird. Man spricht hierbei auch vom *Registrierungsformat*, weil es zum Speichern von GUIDs in der Windows-Registrierung verwendet wird, wie im folgenden Beispiel gezeigt.

```
{C46C1BE0-3C52-11D0-9200-848C1D000000}
```

COM stellt eine Funktion namens `CoCreateGUID` bereit, mit der neue GUIDs erzeugt werden. Die Funktion basiert auf einem Algorithmus, der Informationen wie beispielsweise die Netzwerkartenadresse und die Systemzeit verwendet, um eine GUID zu berechnen, die auf der gesamten Welt einmalig ist. Programmierer, die Schnittstellen- oder Co-Klassen-Definitionen in IDL entwickeln, können ein Dienstprogramm namens *GUIDGen.exe* nutzen, um GUIDs zu erzeugen. Diese Vorgehensweise erlaubt ihnen, GUIDs zu kopieren und in die IDL-Quelldateien sowie den C++-Quellcode einzufügen. Visual Basic-Programmierer müssen sich darüber keine Gedanken machen, weil die Visual Basic-IDE GUIDs im Hintergrund erstellt, wann immer dies erforderlich ist.

GUIDs werden in COM in den unterschiedlichsten Situationen benötigt. Sie sollten jedoch zunächst die Verwendung dieser Bezeichner in Schnittstellen und Co-Klassen betrachten. Jede COM-Schnittstelle verfügt über eine entsprechende GUID, die als Schnittstellen-ID (Interface ID – IID) bezeichnet wird. Jede Co-Klasse besitzt eine GUID, die als Klassen-ID (ClassID – CLSID) bezeichnet wird. Wenn Sie IDL untersuchen, werden Sie feststellen, dass jede Schnittstelle und Co-Klasse eine UUID-Eigenschaft (Universally Unique Identifier) besitzt. Lassen Sie sich dadurch nicht verwirren. UUIDs und GUIDs sind dasselbe. Hier folgt ein Beispiel für eine IID und eine CLSID in IDL:

```
[ uuid(3B46B8A8-CA17-11D1-920B-709024000000) ]
interface _Hund {
    // Methoden
};

[ uuid(3B46B8AB-CA17-11D1-920B-709024000000) ]
coclass CBeagle {
    // Schnittstellen
};
```

Wenn Sie eine ActiveX-DLL oder eine ActiveX-EXE erzeugen, werden alle erforderlichen CLSIDs und IIDs transparent erzeugt und in die Typbibliothek des Servers kompiliert. Die GUID stellt den physischen Namen einer Schnittstelle oder einer Co-Klasse dar. Wenn eine Client-Applikation mit der Typbibliothek kompiliert wird, werden die GUIDs ebenfalls in das binäre Abbild des Clients kompiliert. Dies ermöglicht der Client-Applikation, immer dann eine spezifische Co-Klasse und Schnittstelle anzufordern, wenn sie zur Laufzeit ein Objekt erstellen und binden muss.

Visual Basic versteckt die GUIDs gut vor den Programmierern. Wenn Sie in einem Visual Basic-Projekt auf eine Typbibliothek verweisen, benutzen Sie einfach die benutzerfreundlichen Namen der Schnittstellen und Co-Klassen in Ihrem Code. Visual Basic liest die erforderlichen GUIDs während des Kompilierens aus der Typbibliothek aus und fügt diese in die EXE oder DLL ein, wenn Sie den Eintrag *Erstellen* aus dem Menü *Datei* auswählen.

Wenn Sie einen COM-Server erstellen, versteckt Visual Basic die GUIDs ebenfalls vor Ihnen. Das Entwicklungssystem generiert automatisch und dynamisch GUIDs für Ihre Schnittstellen und Co-Klassen, wenn Sie einen Server mit Hilfe des *Erstellen*-Befehls erzeugen. Das ist prak-

tisch, manchmal wäre es jedoch besser, wenn Visual Basic etwas mehr Flexibilität bieten würde. Die Sprache erlaubt Ihnen nicht, eine bestimmte GUID zu verwenden und diese einer Schnittstelle oder einer Co-Klasse zuzuordnen. Beispielsweise könnte es sein, dass Sie eine persönliche Entwickler-GUID wie die folgende verwenden wollen:

```
"{DEADBEEF-BADD-BADD-BADD-2BE2DEF4BEDD}"
```

Leider erlaubt Ihnen die Visual Basic-IDE nicht, diese GUID einer Co-Klasse in Ihrem Visual Basic-Server zuzuweisen. Außerdem verlangt Visual Basic, dass Sie Ihre Server-Projekte mit einer geeigneten Einstellung für den Kompatibilitätsmodus erstellen. Wenn Sie diese Vorgabe nicht befolgen, kann sich Ihre GUID von Version zu Version ändern. Kapitel 5 geht ausführlich auf diese Kompatibilität ein.

Komponentencode dynamisch laden

Was wissen Sie bisher über COM? Sie wissen, dass ein Client mit einem Objekt über eine Schnittstelle kommunizieren muss und dass ein Client zur Laufzeit an ein Objekt gebunden wird und die Methoden über die direkte vTable-Bindung aufruft. Sie wissen auch, dass sich die Visual Basic-Clients Informationen über Komponenten und Schnittstellen verschaffen, indem sie beim Kompilieren die Typbibliotheken auswerten. Diese Regeln bilden die Grundlage von COM.

Stellen Sie sich jedoch die folgenden Fragen: Wie erzeugt ein Client ein Objekt und stellt eine Bindung dazu her, wenn er nur zwei GUIDs, die IID und die CLSID, kennt? Wie erhält ein Client die erste Schnittstellenreferenz auf ein Objekt? COM stellt eine Infrastruktur bereit, die den Client hierbei unterstützt. Der Client muss jedoch wissen, wie man das System auffordert Hilfe zu leisten.

COM stellt eine Bibliothek mit Systemfunktionen bereit, die in ihrer Gesamtheit als die *COM-Bibliothek* bezeichnet werden. Diese Bibliothek wird durch mehrere DLLs realisiert, die auf jedem COM-fähigen Betriebssystem installiert sind. Die COM-Bibliothek ist Teil von Windows 2000, Windows NT 4, Windows 98 und Windows 95. (Die Laufzeitdateien in zukünftigen Versionen von COM+ werden jedoch möglicherweise vom Betriebssystem abgekoppelt.)

Client-Applikationen, die in C++ geschrieben sind, arbeiten normalerweise über Aufrufe der *OLE32.DLL* mit der COM-Bibliothek zusammen. Visual Basic-Programmierer bleiben durch die Laufzeitebene von dieser DLL abgeschirmt. Abbildung 3.3 zeigt die Unterschiede zwischen den Schichten einer in C++ und einer in Visual Basic entwickelten COM-Applikation.

Client-Applikationen müssen die von der COM-Bibliothek bereitgestellten Dienste aufrufen, um Objekte zu erzeugen und eine Verbindung zu diesen herzustellen. Die Vorgehensweise beim Erstellen des Objekts muss sorgfältig geregelt werden, weil ein Client eine Bindung mit einer Schnittstelle und nicht mit einer Klasse eingehen muss. Dies führt zu einem Dilemma: Wie kann ein Client ein Objekt erzeugen, wenn er die Definition der erzeugbaren Klasse nie sieht? Die folgenden Abschnitte beschreiben genau, wie dies mit COM möglich wird.

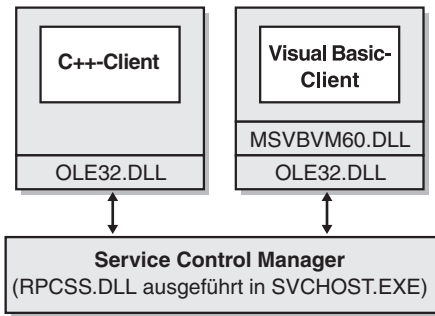


Abbildung 3.3: C++-Programmierer kommunizieren direkt mit der COM-Bibliothek. Visual Basic-Programmierer werden durch die Visual Basic-Laufzeitschicht von dieser Bibliothek abgeschildert.

Objektaktivierung

Eine Client-Applikation kann die CLSID einer Co-Klasse sowie die von der Klasse unterstützten IIDs während des Kompilierens ermitteln, indem Sie eine Typbibliothek überprüft. COM verlangt jedoch, dass keine weiteren Abhängigkeiten zwischen Clients und Co-Klassen bestehen dürfen. Solange ein Client weiß, welche CLSID und IID er benutzen soll, kann er ein Objekt erzeugen und sich daran binden. Nachdem die Bindung hergestellt ist, ruft der Client die Methoden des Objekts auf, indem er auf die der IID zugeordnete vTable zugreift. Das Laden eines Objekts und das Herstellen einer Verbindung zu diesem Objekt wird als *Aktivierung* bezeichnet. Will ein Client ein Objekt aktivieren, muss er die COM-Bibliothek aufrufen und eine CLSID und eine IID übergeben.

Die Unterstützung der Aktivierung muss in der COM-Infrastruktur enthalten sein, weil Clients immer von den Definitionen konkreter Klassen abgeschildert sein müssen. Das ist ein Grundsatz der schnittstellenbasierten Programmierung. Wäre ein Client fähig, die Definition einer Komponente einzusehen, könnte er beim Kompilieren eine Abhängigkeit zum Datenlayout des Objekts aufbauen. Stattdessen gibt COM die Verantwortung für das Reservieren von Speicher und das Erstellen des Objekts an den Server ab. Die Unterstützung der Infrastruktur, die von COM zur Verfügung gestellt wird, spielt die Rolle des Vermittlers. Sie nimmt eine Aktivierungsanforderung vom Client entgegen und leitet diese an den Server weiter. Es ist der Server, der `New` für die Klasse aufruft, wenn das Objekt instanziiert wird.

Der COM-Dienst, der an der Aktivierung beteiligt ist, wird als Service Control Manager (SCM) bezeichnet – von erfahrenen COM-Programmierern als »Schmutzschicht« titulierte. Während der prozessinternen Aktivierung lädt die `OLE32.DLL` den SCM in den Client-Prozess. Um die prozessexterne Aktivierung in Windows 2000 zu unterstützen, wird der SCM von `RPCSS.DLL` in eine Instanz von `SVCHOST.EXE` geladen, wie in Abbildung 3.3 gezeigt. Beachten Sie, dass der SCM unter Windows NT ein Dienst ist, der als `RPCSS.EXE` ausgeführt wird.

Eine Client-Applikation interagiert mit dem SCM immer über die Datei `OLE32.DLL`. C++-Programmierer aktivieren Objekte, indem sie die Funktion `CoCreateInstance` (oder eine verwandte Funktion namens `CoCreateInstanceEx`) aufrufen. Visual Basic-Programmierer aktivieren Objekte mit dem `New`-Operator oder der Funktion `CreateObject`. Wenn eine Client-Applikation `New` oder `CreateObject` für eine Komponente auf dem COM-Server aufruft, besitzt Visual Basic genügend Informationen, um die entsprechende CLSID zu ermitteln und `CoCreateInstance` aufzurufen.

Ruft eine Client-Applikation beispielsweise den Operator `New` für eine Komponente in einer ActiveX-DLL auf, bettet der Visual Basic-Compiler die CLSID der Komponente in das binäre Abbild des Clients ein. Zur Laufzeit wird der Aufruf von `New` von der Visual Basic-Laufzeitumgebung in einen Aufruf von `CoCreateInstance` umgewandelt.

Wenn die Visual Basic-Laufzeitumgebung der Funktion `CoCreateInstance` die CLSID übergibt, verwendet der SCM die Konfigurationsinformationen aus der Windows-Registrierung, um das binäre Abbild des Servers zu finden. Dies ist normalerweise eine DLL oder eine EXE. Der COM-Server benötigt somit entsprechende Registrierungseinträge, zu denen auch der physische Pfad des Servers zählt. Jeder COM-Server ist selbst für das Speichern seiner Konfigurationsinformationen in der Registrierung verantwortlich, wenn er dazu aufgefordert wird.

CLSIDs und die Windows-Registrierung

Der SCM ist von den in der Registrierung bereitgestellten Informationen abhängig. Jede ladbare Komponente, die einem prozessinternen oder prozessexternen Server zugeordnet ist, muss einen Registrierungseintrag besitzen, der Aufschluss über den physischen Pfad zum binären Abbild des Servers gibt.

Der Registrierungsschlüssel `HKEY_CLASSES_ROOT\CLSID` listet die Komponenten auf, die den Client-Applikationen auf der Maschine zur Verfügung stehen. Jeder CLSID-Schlüssel enthält einen oder mehrere untergeordnete Schlüssel, die die Konfigurationsdaten der zugehörigen Komponente enthalten, wie in Abbildung 3.4 gezeigt. Komponenten in einem prozessinternen Server, wie beispielsweise eine ActiveX-DLL, benötigen einen `InprocServer32`-Schlüssel, während Komponenten in einem prozessexternen Server, wie etwa einer ActiveX-EXE, einen `LocalServer32`-Schlüssel erfordern. Diese Schlüssel enthalten den physischen Pfad zu der Server-Datei. SCM verwendet sie, um die Server bei der Objektaktivierung zu finden und zu laden.

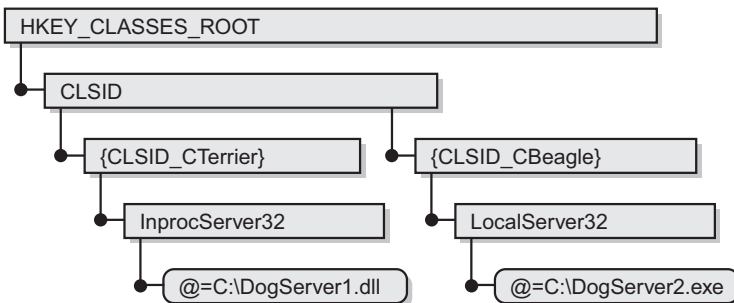


Abbildung 3.4: Jeder erzeugbaren Co-Klasse muss ein Registrierungseintrag zugeordnet sein, der die CLSID mit der Position des zugehörigen Servers verknüpft. SCM verwendet die Information über diese Zuordnung, um während einer Aktivierungsanforderung einen Server zu finden und zu laden.

Es gibt auch noch andere wichtige Registrierungsschlüssel. Der Registrierungsschlüssel `HKEY_CLASSES_ROOT\\TypeLib` verwaltet die physische Position aller registrierten Typbibliotheken auf der lokalen Maschine. Wenn Sie das Dialogfeld *Verweise* öffnen, wertet Visual Basic diesen Schlüssel aus, um die Liste der verfügbaren Referenzen anzuzeigen.

Wie Sie später in diesem Kapitel erfahren werden, spielt jede IID beim Binden eines Objekts an einen Client, der in einem anderen Prozess ausgeführt wird, eine wichtige Rolle. Jeder Schnittstelle, die von einem entfernten Computer aus bedient werden kann, müssen im Registrierungsschlüssel `HKEY_CLASSES_ROOT\\Interface` Konfigurationsdaten zugeordnet werden. Diese Konfigurationsdaten ermöglichen es, Verbindungen einzurichten, die über Prozess- und Maschinengrenzen hinausgehen. Mehr darüber erfahren Sie später in diesem Kapitel.

Alle modernen COM-Server sind selbstregistrierend. Wird ein Server höflich aufgefordert, schreibt er alle von ihm benötigten Informationen in die Registrierung des lokalen Computers. In den ersten Tagen von COM mussten C++-Programmierer den Registrierungscode manuell für die Win32-API schreiben. Glücklicherweise erzeugt Visual Basic automatisch Code für die Selbstregistrierung, wenn Sie einen Server erstellen. Ein Visual Basic-Server ist in der Lage, alle Informationen über seine CLSIDs, IIDs und die Typbibliothek zu registrieren.

Registrieren eines Servers

Prozessinterne Server wie ActiveX-DLLs sind passiv und können sich nicht ohne fremde Hilfe registrieren bzw. die Registrierung rückgängig machen. Zur Registrierung und deren Beendung verwenden Sie das Win32-SDK-Dienstprogramm *REGSVR32.EXE*. Dieses Dienstprogramm wird von der Befehlszeile aus aufgerufen, indem Sie ihm den Pfad zu der DLL mit einem geeigneten Schalter übergeben, etwa wie folgt:

```
REGSVR32.EXE DOGSERVER.DLL  
REGSVR32.EXE /u DOGSERVER.DLL
```

REGSVR32.EXE registriert den Server nicht. Es lädt den Server und ruft eine von der DLL bereitgestellte bekannte Funktion auf. Die DLL reagiert auf diesen Aufruf, indem sie die benötigten Konfigurationsinformationen in die Registrierung schreibt.

Prozesseexterne Server, wie beispielsweise eine ActiveX-EXE, registrieren sich bei jedem Aufruf selbst. Wenn Sie auf das Symbol für eine ActiveX-EXE doppelklicken, wird der Server gestartet und registriert sich selbst. Sobald der Server-Prozess ausgeführt wird, erkennt er, dass er über keinen Client verfügt und entfernt sich daraufhin selbst aus dem Speicher. Die Registrierung eines EXE-Servers kann mit Hilfe eines Befehlszeilenschalters vorgenommen oder beendet werden, wie im Folgenden gezeigt:

```
DOGSERVER.EXE /RegServer  
DOGSERVER.EXE /UnregServer
```

Wenn Sie mit dem Befehl *Erstellen* einen Server generieren, registriert ihn die Visual Basic-IDE unter Verwendung einer der beiden oben beschriebenen Methoden automatisch für die Workstation, an der Sie arbeiten. Wenn Sie eine ActiveX-DLL erzeugen, integriert Visual Basic den Code für die Selbstregistrierung und registriert den Server anschließend mit *REGSVR32.EXE*. Damit wird es einfach, Client-Applikationen zu erstellen und zu testen, nachdem Sie den Server generiert haben. Beachten Sie, dass es etwas anderes ist, einen Server auf einer Entwicklungs-Workstation zu konfigurieren, als ihn auf einer Produktionsmaschine zu registrieren. Beides ist für die korrekte Ausführung Ihres Servers gleich wichtig.

Der SCM in Aktion

Angenommen, Sie besitzen eine ActiveX-DLL mit einer *MultiUse*-Klasse namens *CBeagle*. Die DLL wurde auf der lokalen Maschine installiert und registriert. Eine Client-Applikation ruft *New* für diese Klasse auf, um ein Objekt zu aktivieren. Die Visual Basic-Laufzeitumgebung wandelt den Aufruf von *New* in einen Aufruf von *CoCreateInstance* um und übergibt dem SCM sowohl eine CLSID als auch eine IID.

Der SCM sucht zunächst die CLSID in der Registrierung und dann den Pfad zur zugehörigen DLL-Datei. Er lädt die DLL in den Prozess des Clients. Anschließend ruft der SCM einen be-

kannten Einsprungpunkt in der DLL auf und reicht die Aktivierungsanforderung weiter, indem er die CLSID und IID übergibt.

Der Server reagiert, indem er eine neue Instanz der angeforderten Klasse erstellt. Anschließend gibt er eine Schnittstellenreferenz (für die angeforderte Schnittstelle) an den SCM zurück. Der SCM gibt diese Schnittstellenreferenz wiederum an den Client zurück. Wenn der Client die Schnittstellenreferenz erhält, wird er erfolgreich an das Objekt gebunden. Der Client verwendet somit eine der vTables des Objekts. Er kann jetzt die Methoden des Objekts aufrufen.

Wie Sie sehen, ist der SCM eigentlich nur ein Kuppler. Nachdem er einen Client an ein Objekt gebunden hat, wird er nicht mehr benötigt. Der Client und das Objekt können nun eine lange, fruchtbare Beziehung eingehen. Damit diese Architektur jedoch korrekt arbeitet, muss der SCM mit dem Server kommunizieren können. Jeder COM-Server muss deshalb eine Objektaktivierung unterstützen, indem er einen bekannten Einsprungpunkt anbietet, über den der SCM Aktivierungsanforderungen weiterleiten kann.

Server müssen Class-Factories bereitstellen

Die Regeln für die Unterstützung der Aktivierung in einem COM-Server sind in den COM-Spezifikationen definiert. COM verwendet dazu eine Software-Technik, die als *Factory-Vorlage* bezeichnet wird. Bei dieser Technik ist der Code, der `New` für die Klasse aufruft, in derselben Binärdatei wie die eigentliche Klasse enthalten. Der Client und der SCM müssen somit die Klassendefinition, die sich hinter dem zu erstellenden Objekt verbirgt, nicht kennen. Der wichtigste Vorteil dieser Technik besteht darin, dass sie den Klassenautoren eine Überarbeitung des Codes ermöglicht, ohne die Abhängigkeiten des Clients, wie beispielsweise das Datenlayout eines Objekts, berücksichtigen zu müssen.

Wenn der SCM mit einem Server zusammenarbeitet, um ein Objekt zu aktivieren, muss er einen Verweis auf einen bestimmten Objekttyp anfordern, der als *Class-Factory* bezeichnet wird. Ein Class-Factory-Objekt ist ein Agent, der Instanzen einer Klasse erstellt, die auf Anforderung mit einer spezifischen CLSID verknüpft wird. Ein COM-Server muss für jede unterstützte CLSID ein Class-Factory-Objekt zur Verfügung stellen. Wenn der SCM eine Aktivierungsanforderung erhält, muss er einen Verweis auf das entsprechende Class-Factory-Objekt ermitteln. Dies geschieht auf unterschiedliche Weise, abhängig davon, ob der Server-Code eine prozessinterne DLL oder eine prozessexterne EXE ist. Abbildung 3.5 zeigt, wie ein einzelnes Class-Factory-Objekt eingesetzt werden kann, um mehrere Instanzen einer bestimmten Co-Klasse zu erstellen.

Alle COM-Server, auch die, die mit Visual Basic erstellt werden, müssen Class-Factories für den SCM zur Verfügung stellen. Wenn Sie eine ActiveX-DLL oder eine ActiveX-EXE erzeugen, generiert Visual Basic eine Class-Factory für jede öffentliche, erstellbare Klasse. Visual Basic erzeugt Class-Factories in einer angemessenen Art und Weise. Sie können das Entwicklungswerkzeug hierbei nicht beeinflussen. Sie können die Class-Factories nicht einmal sehen. Verlassen Sie sich einfach darauf, dass sie existieren. Visual Basic erstellt außerdem automatisch die erforderlichen Einsprungpunkte für den SCM, so dass dieser auf Ihre Class-Factories zugreifen kann.

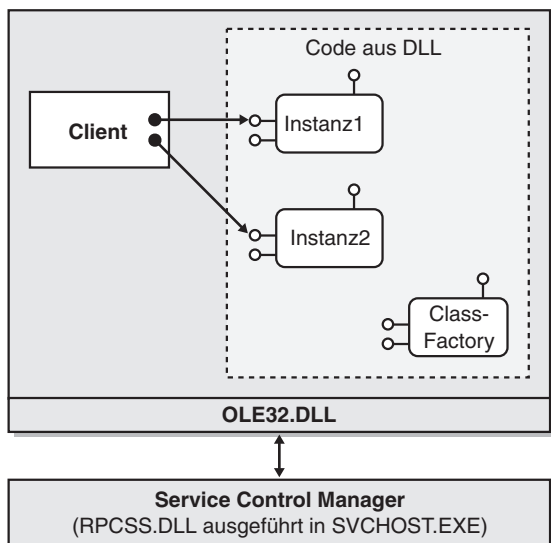


Abbildung 3.5: Der SCM muss mit einem Class-Factory-Objekt zusammenarbeiten, um Instanzen einer bestimmten Co-Klasse erstellen zu können. Dieses Design ermöglicht es dem Code, der für das Erzeugen von Objekten verantwortlich ist, in derselben Binärdatei vorhanden zu sein, aus der auch die Co-Klasse instanziiert wird.

ProgIDs und die CreateObject-Funktion

Eine ProgID ist einfach ein textbasiertes Alias für eine bestimmte CLSID. Eine ProgID setzt sich aus zwei Teilen zusammen. Der erste Teil ist der Name des Server-Projekts, in dem die Komponente enthalten ist. Der zweite Teil ist der vom Benutzer lesbare Klassenname der eigentlichen Komponente. Ein Beispiel für eine ProgID ist `DogServer.CBeagle`. ProgIDs sind insbesondere für Skripting-Clients sehr wichtig. Wenn ein Skripting-Client ein Objekt eines bestimmten Typs aktivieren will, muss er normalerweise die ProgID vorab kennen.

Der Code für die Selbstregistrierung in einem Visual Basic-Server erzeugt Registrierungseinträge, die jeder unterstützten CLSID eine ProgID zuordnen, um die Aktivierung für Applikationen wie zum Beispiel Skripting-Clients zu unterstützen. Beispielsweise gibt es dann den Registrierungsschlüssel `HKEY_CLASSES_ROOT\DogServer.CBeagle`, der einen Wert enthält, welcher einer CLSID zugeordnet wird.

Manchmal begegnen Sie auch ProgIDs, an deren Ende eine Nummer angegeben ist, beispielsweise `DogServer.CBeagle.1` oder `DogServer.CBeagle.2`. Diesen Typ bezeichnet man als versionsabhängige ProgID. ProgIDs ohne angefügte Nummern sind versionsunabhängig. Office-Anwendungen und COM-Server, die in C++-Umgebungen entwickelt werden, wie beispielsweise die ATL (Active Template Library), verwenden häufig versionsabhängige ProgIDs. Visual Basic unterstützt jedoch keine versionsabhängigen ProgIDs. Die von Visual Basic erzeugten ProgIDs erhalten keine Nummer am Ende. Sie sind immer versionsunabhängig.

Wenn Sie in einer Visual Basic-Anwendung die Funktion `CreateObject` aufrufen, übergeben Sie ihr die ProgID für eine bestimmte Komponente. Eine Client-Applikation kann jedoch ein Objekt nicht direkt mit einer ProgID über den SCM aktivieren. Stattdessen muss der Client eine Systemfunktion der COM-Bibliothek aufrufen, um eine ProgID in eine CLSID aufzulösen. Das gilt sowohl für die Visual Basic-Laufzeitumgebung als auch für Skripting-Interpreter.

Nachdem eine Client-Applikation eine ProgID in eine CLSID umgewandelt hat, kann sie `CoCreateInstance` aufrufen. Das alles passiert natürlich im Hintergrund. Beachten Sie, dass der Aufruf von `CreateObject` in der Regel einem Aufruf des Operators `New` entspricht. Für die Implementierung der

Funktion `CreateObject` ist nur ein zusätzlicher Eintrag in die COM-Bibliothek erforderlich, bevor `CoCreateInstance` aufgerufen werden kann.

Bei der Entscheidung, ob Sie Objekte mit dem Operator `New` oder mit der Funktion `CreateObject` erzeugen wollen, gilt es, einen weiteren Unterschied zu berücksichtigen. Wenn Sie `CreateObject` aufrufen, versucht die Visual Basic-Laufzeitumgebung immer, das Objekt über den SCM mit einem Aufruf von `CoCreateInstance` zu aktivieren. Das ist bei der Verwendung des Operators `New` nicht immer der Fall. Wenn eine Klasse mit dem Operator `New` ein Objekt aus einer anderen Klasse im selben Projekt instanziert, erzeugt die Visual Basic-Laufzeitumgebung das Objekt und bindet es, ohne die Hilfe des SCM in Anspruch zu nehmen. Wenn Sie beispielsweise über zwei `MultiUse`-Klassen in einem ActiveX-DLL-Projekt verfügen und eine Klasse mit dem Operator `New` eine Instanz der anderen erzeugt, wird der SCM nicht herangezogen. Das scheint zwar eine sinnvolle Optimierung zu sein, kann aber problematisch für Komponenten sein, die in COM+ oder MTS konfiguriert sind. Kapitel 6 erklärt konfigurierte Komponenten und zeigt, warum es sinnvoll ist, dem SCM alle Aktivierungsanforderungen zu überlassen. Wie Sie sehen werden, gibt es Situationen, in denen Sie den `New`-Operator nicht einsetzen sollten.

IUnknown und IDispatch verstehen

Die beste Eigenschaft eines COM-Objekts besteht wohl darin, dass es berechenbar ist. Nachdem es aktiviert wurde, wartet es geduldig darauf, Ihre Methodenanfragen zu bearbeiten. Das Objekt erkennt, wenn Sie es nicht mehr benötigen, und gibt alle Ressourcen frei, die dann wieder dem System zur Verfügung stehen. Wenn Sie das Objekt fragen, welche Schnittstellen es unterstützt, erhalten Sie eine intelligente Antwort. Das COM-Objekt ermöglicht Ihnen außerdem die Navigation durch all seine Schnittstellen, so dass Sie seine gesamte Funktionalität nutzen können.

Dieser Kapitelabschnitt beschreibt, wie COM-Objekte diese grundlegende Funktionalität mit Hilfe der Schnittstelle `IUnknown` zur Verfügung stellen. Diese Schnittstelle bleibt den Visual Basic-Programmierern zwar völlig verborgen, aber Sie sollten ihre Aufgabe und Funktionalität dennoch verstehen. Außerdem beschäftigen wir uns mit der Standardschnittstelle `IDispatch` und Sie werden erfahren, wie Objekte über einen COM-Mechanismus namens *Automatisierung* den Zugriff auf Skripting-Clients ermöglichen.

Die IUnknown-Schnittstelle

COM verfügt über eine Schnittstelle, von der alle anderen Schnittstellen abgeleitet werden: `IUnknown`. Jede Schnittstelle muss direkt von `IUnknown` oder von einer Schnittstelle abgeleitet werden, in deren Vererbungskette `IUnknown` auf der obersten Ebene angeordnet ist. `IUnknown` ist die einzige Schnittstelle in COM, die sich nicht von einer anderen Schnittstelle ableitet; sie befindet sich an oberster Stelle einer COM-Schnittstellenhierarchie. Das bedeutet, dass sich die drei in `IUnknown` definierten Methoden immer am Anfang einer COM-konformen vTable befinden, wie in Abbildung 3.6 dargestellt. Jede Verbindung zu einem Objekt wird über eine `IUnknown`-kompatible Referenz aufgebaut.

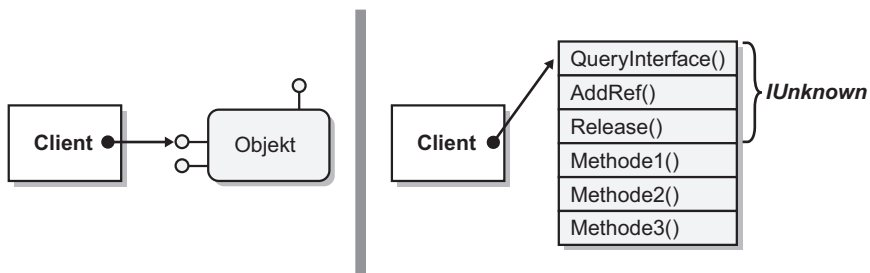


Abbildung 3.6: Die drei Methoden der IUnknown-Schnittstelle befinden sich immer am Anfang einer COM-kompatiblen vTable. Jede Verbindung zu einem Objekt ermöglicht einem Client, diese Methoden aufzurufen.

IUnknown beschreibt das Grundverhalten eines COM-Objekts, im Gegensatz zum domänenspezifischen Verhalten einer benutzerdefinierten Schnittstelle. IUnknown gestattet jedem COM-Objekt, seine eigene Lebensdauer zu verwalten. Ein Client kann außerdem ein Objekt abfragen, um zu ermitteln, ob es eine bestimmte Schnittstelle unterstützt, und um das Objekt dynamisch allen unterstützten Schnittstellen zuzuordnen.

Beachten Sie bitte, dass Visual Basic-Programmierer niemals direkt mit IUnknown in Berührung kommen. Auf der Serverseite verbirgt die Zuordnungsschicht von Visual Basic den Code, der mit IUnknown arbeitet. Sie können Variablen vom Typ IUnknown deklarieren, nicht aber die Methoden dieser Schnittstelle aufrufen, da diese in der Typbibliothek *STDOLE2.TLB* als *restricted* markiert sind.

In COM verwaltet ein Objekt seine eigene Lebensdauer. Das Objekt benötigt jedoch Hilfe, um gezielt entscheiden zu können, ob es sich selbst zerstören oder weiterhin ausgeführt werden soll. Der Client eines Objekts muss *AddRef* aufrufen, wenn ein Schnittstellenverweis dupliziert werden soll. Er muss *Release* aufrufen, wenn eine bestehende Verbindung getrennt werden soll. Wenn alle Clients diese Regeln befolgen, kann ein Objekt eine einfache Implementierung dieser beiden Methoden zur korrekten Verwaltung seiner Lebensdauer bereitstellen. Das Objekt enthält einen Zähler, der Aufschluss über die verbundenen Verweise gibt. Wenn dieser Zähler den Wert 0 erreicht, entfernt sich das Objekt selbst aus dem Speicher, so dass dieser wieder freigegeben wird.

Visual Basic und die Lebensdauerverwaltung

Wenn Sie mit Visual Basic arbeiten, müssen Sie der Lebensdauerverwaltung nicht sehr viel Aufmerksamkeit schenken. Sie sollten lediglich zwei Regeln befolgen: Erhalten Sie den Objektverweis aufrecht, wenn Sie das Objekt weiterhin verwenden möchten, und geben Sie den Verweis frei, wenn Sie das Objekt nicht mehr benötigen. Visual Basic verwaltet alle Aufrufe von IUnknown für Sie. Betrachten Sie bitte das folgende Beispiel.

```
Sub CreateAndUseDog
    Dim Dog As IDog
    Set Dog = New CBeagle ' AddRef wird aufgerufen.
    Dim Dog2 As IDog
    Set Dog2 = Dog        ' AddRef wird aufgerufen.
    Set Dog2 = Nothing     ' Release wird aufgerufen.
    ' Release wird in dem Hund-Objekt aufgerufen,
    ' wenn der entsprechende Verweis den Gültigkeits-
    ' bereich verlässt.
End Sub
```

Dieser Code ruft wiederholt `AddRef` und `Release` auf. Ist ein Objekt aktiviert, wird seine `AddRef`-Methode ausgeführt. Wenn Sie einen zweiten Verweis auf das Objekt erstellen, ruft die Visual Basic-Laufzeitumgebung automatisch `AddRef` auf. Wenn Sie ein Schnittstellenverweis explizit auf `Nothing` setzen, ruft die Visual Basic-Laufzeitumgebung automatisch die `Release`-Methode für Sie auf. Wenn Sie einen Verweis nicht explizit auf `Nothing` setzen, stellt Visual Basic fest, ob ein aktiver Verweis den Gültigkeitsbereich verlässt. In diesem Fall ruft Visual Basic `Release` auf, bevor die Verbindung getrennt wird.

Auf der Objektseite implementiert Visual Basic automatisch `AddRef` und `Release`, um das standardmäßige Herunterzählen der Verweise durchzuführen. Ein Visual Basic-Objekt wird solange ausgeführt, wie aktive Clients mit ihm verbunden sind. Wenn der Client, der die letzte Verbindung hält, `Release` aufruft, zerstört sich das Visual Basic-Objekt selbst. Noch einmal: Sie können in Visual Basic nicht beeinflussen, wie ein Objekt seine Lebensdauer verwaltet. Es ist jedoch wichtig zu verstehen, wie sich Ihre Objekte verhalten.

Die Tatsache, dass Visual Basic `AddRef` und `Release` im Hintergrund aufruft, verleitet Sie vielleicht zu der Annahme, dass Sie die lokalen Objektverweise nicht auf `Nothing` setzen müssen, was aber zahlreiche Probleme nach sich ziehen kann. Die meisten Programmierer hatten schon mit Speicherlecks und anderweitigen Problemen zu kämpfen, wenn sie vergaßen, Objektverweise auf `Nothing` zu setzen, bevor die Variablen, die die Verweise enthalten, den Gültigkeitsbereich verlassen. Unter anderem waren es frühe Versionen von DAO (Data Access Objects) und ADO (ActiveX Data Objects), die als Werkzeuge Probleme bereiteten. Viele Visual Basic-Programmierer haben sich deshalb angewöhnt, die Objektverweise immer auf `Nothing` zu setzen, nachdem sie ein Objekt verwendet haben.

Einer der komplexen Aspekte der Lebensdauerverwaltung betrifft die so genannten *Zirkelverweise*. Ein Zirkelverweis liegt vor, wenn Objekt A beispielsweise einen Verweis auf Objekt B und Objekt B einen Verweis auf Objekt A enthält. Auch nachdem alle beteiligten Clients Ihre Verbindungen zu den Objekten getrennt haben, verbleiben diese aufgrund der ausstehenden gegenseitigen Verweise im Speicher. Wenn Ihr Design Zirkelverweise aufweist, sollten Sie sicherstellen, dass Objekte zerstört werden, wenn Sie diese nicht mehr benötigen. Ein gebräuchliches Verfahren, das verhindert, dass Zirkelverweise zu Objekten mit unbegrenzter Lebensdauer führen, besteht darin, eine explizite Methode in einem der Objekte zu erstellen. Diese Methode unterbricht die Verbindung, so dass alle beteiligten Objekte zerstört werden.

Die QueryInterface-Methode

Die erste und möglicherweise wichtigste Methode von `IUnknown` ist `QueryInterface`, die den Clients die Navigation zwischen den unterschiedlichen Schnittstellen ermöglicht, die von einem Objekt unterstützt werden. Diese dynamische Navigation zwischen den verschiedenen Schnittstellen wird als *Typ-Casting* bezeichnet. Ein Client kann `QueryInterface` auch einfach benutzen, um zu überprüfen, ob ein Objekt eine bestimmte Schnittstelle unterstützt. Die Leistungsmerkmale, die von `QueryInterface` zur Verfügung gestellt werden, sind für COM sehr wichtig. Ohne `QueryInterface` könnte COM keine Polymorphie erzielen und keine Laufzeittypüberprüfung durchführen, was von einem schnittstellenbasierten Programmparadigma verlangt wird.

Ein COM-Objekt muss mindestens eine, kann jedoch beliebig viele Schnittstellen implementieren. Objekte, die mehrere Schnittstellen implementieren, müssen den Clients eine Navigation zwischen diesen Schnittstellen ermöglichen. Dies geschieht über einen Aufruf von `QueryInterface`. Ein Client übergibt dem Objekt die gewünschte IID, wenn er `QueryInterface` aufruft. Das Objekt reagiert darauf, indem es einen Verweis auf die entsprechende Schnittstelle zurückgibt. Wenn der

Client eine Schnittstelle anfordert, die nicht unterstützt wird, führt der Aufruf von `QueryInterface` zu keinem Ergebnis. Der Client bemerkt dies und kann entsprechend darauf reagieren.

Die Visual Basic-Laufzeitumgebung ruft `QueryInterface` auf, wenn Sie ein Objekt einem bestimmten Referenztyp zuweisen. Betrachten Sie bitte das folgende Beispiel:

```
Dim Dog As IDog
Set Dog = New CBeagle
' Um zu einer anderen Schnittstelle zu gelangen:
Dim WonderDog As IWonderDog
Set WonderDog = Dog ' QueryInterface wird aufgerufen.
WonderDog.FetchSlippers
```

Wenn Sie über einen `IDog`-Verweis auf ein Objekt verfügen und einen `IWonderDog`-Verweis abrufen möchten, können Sie einfach die `Set`-Anweisung benutzen, um einen Schnittstellenverweis in einen anderen umzuwandeln. Wenn die Schnittstelle nicht unterstützt wird, führt dies zum auffangbaren Fehler »Datentypen unverträglich«. Nachdem die Umwandlung erfolgreich durchgeführt wurde, können Sie den neuen Schnittstellenverweis verwenden, um die Methoden des Objekts aufzurufen.

Einige Visual Basic-Programmierer bevorzugen die blinde Zuordnung von Schnittstellenverweisen, wie im vorherigen Beispiel gezeigt. Wenn Sie versuchen, eine Schnittstelle zu verwenden, die nicht unterstützt wird, führt die Visual Basic-Laufzeitumgebung einen erfolglosen Aufruf von `QueryInterface` aus, so dass ein auffangbarer Laufzeitfehler ausgelöst wird. Wenn die Möglichkeit besteht, dass die Zuordnung nicht erfolgen kann, sollten Sie darauf vorbereitet sein, den Fehler aufzufangen und darauf zu reagieren. Wenn Sie auf eine Fehlerbehandlung verzichten möchten, können Sie mit der `TypeOf`-Syntax von Visual Basic ermitteln, ob ein Objekt eine bestimmte Schnittstelle unterstützt. Betrachten Sie dazu auch das folgende Beispiel:

```
Dim Dog As IDog
Set Dog = New CBeagle
' Prüfen, ob die Schnittstelle unterstützt wird,
' bevor sie verwendet wird.
If TypeOf Dog Is IWonderDog Then ' Aufruf von QueryInterface.
    Dim WonderDog As IWonderDog
    Set WonderDog = Dog ' Aufruf von QueryInterface.
    WonderDog.FetchSlippers
Else
    ' Hier folgt der Programmcode, der ausgeführt wird,
    ' wenn die Schnittstelle nicht unterstützt wird.
End If
```

Ein Visual Basic-Objekt implementiert `QueryInterface` automatisch. Wenn Sie eine oder mehrere benutzerdefinierte Schnittstellen in einer Klasse verwenden, stellt Visual Basic eine Implementierung der `QueryInterface`-Methode zur Verfügung, die den Clients die Navigation zwischen den Schnittstellenverweisen ermöglicht. Wie im Fall von `AddRef` und `Release` ist die Visual Basic-Implementierung von `QueryInterface` sehr einfach. Sie werden sie jedoch niemals zu Gesicht bekommen. Vertrauen Sie einfach darauf, dass die Methode perfekt arbeitet.

Noch ein Wort zur clientseitigen Aktivierung

Ich habe bereits beschrieben, wie die Objektaktivierung für einen Client, den SCM und einen prozessinternen Server funktioniert. Ich konnte dabei jedoch nicht darauf eingehen, was genau

in der Visual Basic-Laufzeitumgebung passiert, weil ich die Methode `QueryInterface` noch nicht beschrieben hatte. Nachdem Sie wissen, wie und warum ein Client `QueryInterface` aufruft, können Sie vielleicht verstehen, wie Visual Basic während der Objektaktivierung vorgeht. Betrachten Sie bitte den folgenden Code:

```
Dim Spot as CDog
Set Spot = New CDog
```

Wenn die Visual Basic-Laufzeitumgebung `CoCreateInstance` aufruft, übergibt sie die CLSID für die Komponente `CDog` und außerdem die IID für `IUnknown`. Nachdem sie das Objekt mit einem `IUnknown`-Verweis aktiviert hat, ruft sie `QueryInterface` auf, um einen zweiten Verweis zu erhalten, indem sie die IID für die Standardschnittstelle `_CDog` übergibt. Der Client wird an die gewünschte Schnittstelle gebunden, und alles wird wie erwartet fortgesetzt. Was passiert jedoch, wenn Sie eine Komponente mit einer benutzerdefinierter Schnittstelle besitzen und clientseitigen Code schreiben, der wie folgt aufgebaut ist?

```
Dim Fido As IDog
Set Fido = New CBeagle
```

Es stellt sich heraus, dass die Visual Basic-Laufzeitumgebung einige Aufrufe ausführt, die nicht so leicht nachzuvollziehen sind. Zuerst ruft sie `CoCreateInstance` auf, übergibt die CLSID für `CBeagle` und die IID für `IUnknown`, um das Objekt zu aktivieren. Anschließend ruft sie `QueryInterface` auf, um einen zweiten Verweis zu erhalten, indem sie die IID für die Standardschnittstelle `_CBeagle` übergibt. Die Visual Basic-Laufzeitumgebung führt diesen Schritt immer aus, unabhängig davon, ob der Client die Schnittstelle nutzt. Schließlich ruft die Visual Basic-Laufzeitumgebung `QueryInterface` noch einmal auf, um die gewünschte Schnittstelle zu erhalten, indem sie die IID für `IDog` übergibt. Beachten Sie hier, dass die Verwendung des Operators `New` für einen Klassennamen dazu führt, dass die Visual Basic-Laufzeitumgebung `QueryInterface` aufruft, um den Verweis auf die Standardschnittstelle zu erhalten. Dabei ist es egal, ob der Client sie benutzt oder nicht verwendet.

Dieser letzte Punkt erscheint Ihnen möglicherweise etwas seltsam. Sie werden jedoch größere Probleme haben, wenn Sie versuchen, Komponentenversionen zu verwenden, und nicht verstehen, warum das Ganze nicht korrekt läuft. Wir werden in Kapitel 5 noch einmal darauf zu sprechen kommen, nachdem wir uns mit Komponentenversionen auseinander gesetzt haben. Wie Sie sehen, müssen Sie die IID für die Standardschnittstelle einer Komponente konstant halten, wenn Sie spätere Versionen des Servers kompilieren. Das ist auch dann der Fall, wenn die Standardschnittstelle keine Methoden aufweist.

Die IDispatch-Schnittstelle und die Automatisierung

Früher stand COM lediglich für C- und C++-Programmierer zur Verfügung. Das lag daran, dass die vTable-Bindung auf der untersten Ebene durchgeführt wird. IDL und Typbibliotheken ermöglichten Entwicklungswerkzeugen wie Visual Basic 4, die erforderlichen vTable-Bindungen während des Kompilierens zu erstellen. Einige Clients können jedoch immer noch nichts mit Typbibliotheken und den äquivalenten C-/C++-Header-Dateien anfangen. Die gebräuchlichsten Beispiele hierfür sind heute Skripting-Clients, die in VBScript oder JavaScript geschrieben wurden.

Weil ein Skripting-Client nicht fähig ist, Schnittstellendefinitionen aus einer Typbibliothek zu lesen, kann er den auf der Clientseite erforderlichen Bindungs-Code für den Zugriff auf die einem COM-Objekt zugeordneten vTables nicht erzeugen. Diese Clients besitzen dennoch eine Mög-

lichkeit, Methoden in einem speziellen Objekttyp, einem so genannten Automatisierungsobjekt, aufzurufen. Ein Automatisierungsobjekt ist ein Objekt, das eine spezielle Schnittstelle namens `IDispatch` implementiert.

Microsoft war darauf bedacht, die Automatisierung in die Entwicklungsumgebung von COM zu integrieren, da eine derartige Erweiterung zu einer größeren Zahl von COM-Programmierern führen sollte. Die Automatisierung lässt COM sprachunabhängiger werden. Die Originalversion der Automatisierung (die als OLE-Automatisierung bezeichnet wurde) wurde ungefähr zur selben Zeit wie Visual Basic 3 ausgeliefert. Vorhergehende Versionen von Visual Basic enthielten keine COM-Unterstützung. Die Automatisierung war somit die erste Technik, mit deren Hilfe ein Visual Basic-Programmierer clientseitigen Code erzeugen konnte, um auf ein COM-Objekt zuzugreifen.

Das COM-Team und das Visual Basic-Team arbeiteten intensiv zusammen, damit die Automatisierung in Visual Basic 3 eingesetzt werden konnte. Version 4 unterstützte dann Typbibliotheken, so dass Visual Basic nicht länger auf die Automatisierung auf der Clientseite beschränkt war. Viele Sprachen und Werkzeuge nutzen die Automatisierung jedoch immer noch nicht. Dazu zählen beispielsweise die webbasierten Technologien von Microsoft, zu denen Sprachen wie VBScript und JavaScript gehören. Diese Sprachen können keine benutzerdefinierten vTable-Bindungen erzeugen. Programmierer, die diese Sprachen benutzen, sind dann auf die Automatisierung angewiesen, wenn sie auf ein COM-Objekt zugreifen müssen. Wenn Sie Komponenten für Skripting-Clients erzeugen, sollten Sie zuvor verstehen, wie die Automatisierung funktioniert.

Spätes Binden mit `IDispatch`

Die Automatisierung basiert auf einer Schnittstelle, die als `IDispatch` bezeichnet wird. Diese Schnittstelle ermöglicht den Clients zur Laufzeit, in einem Prozess Methodenbindungen zu erzeugen, was auch als späte Bindung bezeichnet wird. Die vTable, die `IDispatch` repräsentiert, ist in Abbildung 3.7 dargestellt. `IDispatch` erweitert `IUnknown` mit vier Methoden. Automatisierungsclients verwenden `GetIDsOfNames` und `Invoke`, um die späte Bindung zu realisieren. Visual Basic-Programmierer arbeiten, wie im Fall von `IUnknown`, nicht direkt mit dieser Schnittstelle. Stattdessen übersetzt die Visual Basic-Zuordnungsschicht Ihren Code und ruft die `IDispatch`-Methode auf. Dasselbe gilt für clientseitigen Code, der in VBScript oder JavaScript erstellt wurde. Der Skripting-Interpreter erledigt alle erforderlichen Aufrufe von `IDispatch` im Hintergrund.

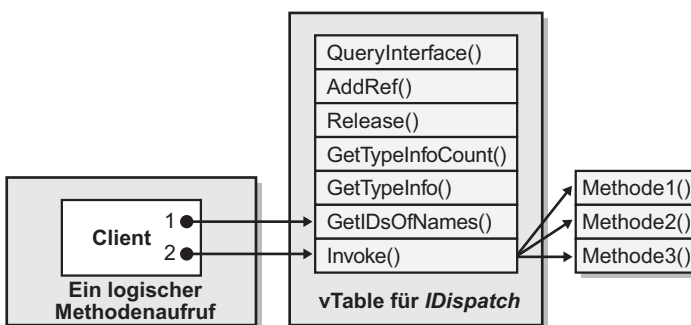


Abbildung 3.7: Automatisierungs-Clients werden an eine vTable gebunden, die auf der `IDispatch`-Schnittstelle basiert. Ein Automatisierungs-Client führt die späte Bindung durch, indem er `GetIDsOfNames` und anschließend `Invoke` aufruft.

Die Automatisierung arbeitet wie folgt: Nachdem ein Client einen `IDispatch`-Verweis erstellt hat, kann er ermitteln, ob ein Objekt eine bestimmte Methode unterstützt. Dazu ruft der Client `GetIDsOfNames` auf. Der Client muss in diesem Aufruf den Namen der Methode als einen Zeichenkettenparameter übergeben. Wenn das Objekt die angeforderte Methode nicht unterstützt, schlägt der Aufruf von `GetIDsOfNames` fehl. Wenn das Objekt die angeforderte Methode unterstützt, gibt

`GetIDsOfNames` einen logischen Bezeichner für die Methode zurück, der als DISPID bezeichnet wird. Eine DISPID ist eine ganze Zahl, die von einem Objekt zur Verfügung gestellt wird, um eine der Methoden des Objekts zu identifizieren. Positive DISPIDs zeigen an, dass eine Methode benutzerdefiniert ist, während negative DISPIDs für spezielle Methoden reserviert sind.

Ein Client, der eine gültige DISPID besitzt, kann über die Automatisierung eine Methode ausführen lassen, indem er `Invoke` aufruft. `Invoke` stellt die einzige Möglichkeit dar, eine benutzerdefinierte Methode über die `IDispatch`-Schnittstelle auszuführen. Wie Sie sehen, ist `IDispatch` eine einzelne physische Schnittstelle, die es den Clients erlaubt, beliebig viele logische Schnittstellen zu ermitteln. `IDispatch` repräsentiert im Wesentlichen eine Standard-`vTable` mit einer höchst flexiblen Aufrufarchitektur. Clients, wie zum Beispiel VBScript, können mit Hilfe dieser Architektur auf viele unterschiedliche Typen von COM-Objekten zugreifen, wenngleich sie nur ein einzelnes `vTable`-Layout kennen. Ein Client muss somit keinen benutzerdefinierten Bindungscode aus den Informationen einer Typbibliothek generieren.

Immer wenn Sie eine *MultiUse*-Klasse in ein ActiveX-DLL- oder ActiveX-EXE-Projekt kompilieren, erzeugt Visual Basic automatisch eine Implementierung von `IDispatch` für die öffentlichen Methoden der Standardschnittstelle. Das bedeutet, alle COM-fähigen Skripting-Clients können Objekte in einem Visual Basic-Server erzeugen und nutzen. Ohne diese Unterstützung wären Sie nicht in der Lage, Visual Basic-Komponenten von Skripting-Clients, wie beispielsweise ASP (Active Server Pages) oder WSH (Windows Scripting Host), zu verwenden.

Während `IDispatch` zwar sehr flexibel ist, ist es verglichen mit der benutzerdefinierten `vTable`-Bindung nicht sehr effizient. Jeder logische Aufruf über `IDispatch` erfordert tatsächlich zwei Aufrufe. Der erste Aufruf von `GetIDsOfNames` erfordert eine Überprüfung einer Zeichenkette im Objekt, damit die korrekte DISPID an den Client zurückgegeben wird. Der Aufruf von `Invoke` bedingt ebenfalls ein wenig Aufwand, der ein Resultat der offenen Ausführungsarchitektur von `IDispatch` ist.

Eine Methode kann über eine beliebige Anzahl von Parametern verfügen. Die Automatisierung berücksichtigt dies, indem sie vom Client verlangt, alle Parameter als ein Array vom Typ *Variant* im Aufruf von `Invoke` zu übergeben. Das Array ermöglicht der Methode, eine beliebige Anzahl von Parametern zu definieren. Der *Variant*-Typ erlaubt, dass die einzelnen Parameter selbstbeschreibend sind. Ein Objekt reagiert auf einen Aufruf von `Invoke`, indem es die DISPID auswertet und die *Variant*-Argumente in die korrekten Typen konvertiert. Nachdem das Objekt den Aufruf von `Invoke` bearbeitet hat, muss es den Rückgabewert vom Typ *Variant* an den Client zurückgeben.

Beachten Sie zwei wichtige Aspekte zur späten Bindung und zur Verwendung von `IDispatch`: Die späte Bindung ist am besten für Clients geeignet, die nicht mit einer direkten `vTable`-Bindung zurechtkommen, führt jedoch, verglichen mit der direkten `vTable`-Bindung, zu relativ langsamen Ausführungszeiten. Wenn Sie mit einem prozessinternen Objekt arbeiten, können Sie davon ausgehen, dass ein Aufruf unter Verwendung der direkten `vTable`-Bindung ein paar hundert Mal schneller ist als ein Aufruf über `IDispatch`.

Ein Visual Basic-Client greift mit dem `Object`-Datentyp über die `IDispatch`-Schnittstelle auf ein Automatisierungsobjekt zu. Beachten Sie, dass Sie den Typ `IDispatch` in Ihrem Visual Basic-Quellcode nicht nutzen können, weil er in der Typbibliothek *STDole2.TLB* als *restricted* gekennzeichnet ist. Sie legen jedoch einen `IDispatch`-Verweis an, wenn Sie mit dem `Object`-Datentyp einen Parameter oder eine Variable definieren. Hier folgt ein Beispiel für die Verwendung des `Object`-Datentyps für den Zugriff auf ein Objekt über `IDispatch`:

```
Dim Dog As Object
Set Dog = CreateObject("DogServer.CBeagle")
Dog.Name = "Rex"
Dog.Bark
```

Wenn das Objekt erstellt ist, wird der `Object`-Variablen (`Dog`) ein `IDispatch`-Verweis zugewiesen. Ein logischer Aufruf einer Eigenschaft oder Methode wird in Aufrufe von `GetIDsOfNames` sowie `Invoke` umgewandelt. Visual Basic könnte die Automatisierung optimieren, indem es die `DISPIDs` der Eigenschaften und Methoden zwischenspeichert, was es jedoch nicht tut. Jeder logische Aufruf führt zu zwei Datenübertragungen zwischen dem Client und dem Objekt. Wenn Sie den zusätzlichen Aufwand für einen Aufruf von `Invoke` hinzufügen, werden Sie einsehen, dass Sie den `Object`-Datentyp nur dann verwenden sollten, wenn dies absolut erforderlich ist.

Wenn Sie den `Object`-Datentyp verwenden, sollten Sie ebenfalls daran denken, dass Sie über keine Typsicherheit verfügen. Die Visual Basic-Umgebung geht während des Kompilierens davon aus, dass jeder Methodenaufruf über `IDispatch` erfolgreich sein wird. Das bedeutet, dass Sie einige der praktischen Funktionen der Visual Basic-IDE verlieren, zu denen die Typüberprüfung während des Kompilierens und IntelliSense zählen – noch ein Grund, den `Object`-Datentyp nicht zu benutzen.

Duale Schnittstellen

Bisher haben Sie zwei Client-Typen kennen gelernt – solche, die über eine direkte `vTable`-Bindung auf benutzerdefinierte Schnittstellen zugreifen, und solche, die über die `IDispatch`-Schnittstelle und eine späte Bindung auf Methoden zugreifen. Für jeden dieser Client-Typen ist ein anderer Schnittstellenstil geeignet. Clients, die eine direkte `vTable`-Bindung vornehmen können, verwenden `IDispatch` nicht, weil es sehr langsam ist. Skripting-Clients können nicht auf benutzerdefinierte Schnittstellen zugreifen, weil sie den dafür erforderlichen clientseitigen Bindungs-Code nicht erzeugen können. Es gibt jedoch noch einen dritten Schnittstellenstil, eine Mischung aus den beiden anderen. Dieser wird auch als *duale Schnittstelle* bezeichnet. Abbildung 3.8 vergleicht die drei Typen.

Eine benutzerdefinierte Schnittstelle wird direkt von `IUnknown` abgeleitet, während eine duale Schnittstelle direkt von `IDispatch` abgeleitet wird. Wichtig ist dabei, dass die `vTable` für eine duale Schnittstelle neben den Methoden von `IDispatch` auch benutzerdefinierte Methoden enthält. Clients können über eine direkte `vTable`-Bindung oder über die späte Bindung auf eine duale Schnittstelle zugreifen. Eine duale Schnittstelle bietet die Geschwindigkeit und Effizienz der direkten `vTable`-Bindung für fortgeschrittene Clients, die mit Werkzeugen wie C++ oder Visual Basic erzeugt wurden. Eine duale Schnittstelle stellt außerdem Einsparungspunkte im Stil von `IDispatch` für Automatisierungs-Clients zur Verfügung, wie beispielsweise VBScript und JavaScript.

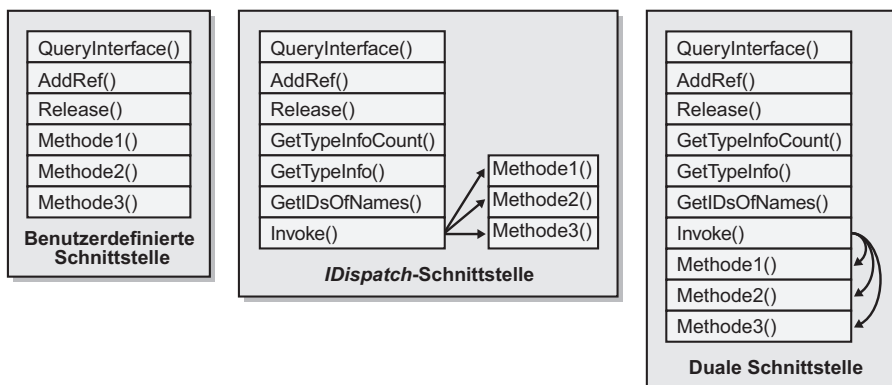


Abbildung 3.8: Eine duale Schnittstelle ist eine Mischung aus einer `IDispatch`-Schnittstelle und einer benutzerdefinierte Schnittstelle.

Visual Basic erstellt seit Version 5 Server unter Verwendung von dualen Schnittstellen. Sie können duale Schnittstellen in der Typbibliothek Ihres Servers nicht anfordern, und Sie können ihre Verwendung auch nicht verhindern. Wenn Sie *Erstellen* aus dem Menü *Datei* auswählen, generiert Visual Basic eine vollständig funktionsfähige Implementierung der dualen Schnittstelle für jede Komponente in Ihrem Server. Damit dies in der richtigen Weise geschieht, muss Visual Basic auch eine Standardimplementierung von `IDispatch` für jede erstellbare Klasse bereitstellen. Das bedeutet, dass Ihre Visual Basic-Objekte sowohl benutzerdefinierten `vTable`-Clients als auch Skripting-Clients gerecht werden.

Ein schneller Überblick über die verfügbaren Bindungstechniken

Clients können in Visual Basic unter Verwendung von drei verschiedenen Bindungstypen erzeugt werden. Das *späte Binden* wird verwendet, wenn Sie den `Object`-Datentyp einsetzen. Dies geschieht unabhängig davon, ob Sie eine Typbibliothek für das Objekt verwenden oder nicht. Das *späte Binden* weist die schlechteste Leistung auf und unterstützt keine Typüberprüfung während des Kompilierens. Sie sollten das *späte Binden* nur dann nutzen, wenn es keine andere Möglichkeit gibt. Unter Visual Basic können Sie in Ihrem clientseitigen Code immer darauf verzichten.

Das *DISPID-Binden* tritt auf, wenn Sie über eine reine `IDispatch`-Komponente verfügen und eine Typbibliothek verwenden können. In Abbildung 3.8 sehen Sie die reine `IDispatch`-Schnittstelle in der Mitte im Vergleich zu der dualen Schnittstelle rechts. Der Client kann die DISPIDs während des Kompilierens aus der Typbibliothek lesen und diese in die ausführbare Client-Datei einbetten. Ein Aufruf von `GetIDsOfNames` ist somit nicht mehr erforderlich. Der Client ruft jedoch weiterhin `Invoke` auf, um Methoden ausführen zu lassen. Die DISPID-Bindung ist schneller als das *späte Binden*, aber immer noch langsamer als die `vTable`-Bindung. Die DISPID-Bindung ermöglicht der Visual Basic-IDE außerdem, während des Kompilierens eine Typüberprüfung durchzuführen und IntelliSense zu nutzen.

Die DISPID-Bindung wird nur dann ausgeführt, wenn Sie über eine Typbibliothek für ein `IDispatch`-Objekt verfügen, das keine duale Schnittstelle verwendet. Es gibt nicht sehr viele Komponenten, die diese Kriterien erfüllen. Das vielleicht häufigste Beispiel sind ältere C++-Komponenten, die unter Verwendung der MFC-Umgebung (Microsoft Foundation Class) erzeugt wurden. Wenn Sie übrigens ein Steuerelement auf einem Formular anordnen, nutzt Visual Basic die DISPID-Bindung anstelle der `vTable`-Bindung. Der Grund hierfür besteht darin, dass Steuerelemente die `IDispatch`-Schnittstelle verwenden müssen, um dynamische Eigenschaften unterstützen zu können.

Die `vTable`-Bindung ist immer die beste Lösung. Sie ist wesentlich schneller als die anderen beiden Bindungstechniken. Visual Basic-Clients nutzen immer die direkte `vTable`-Bindung, wenn die folgenden Bedingungen erfüllt sind.

- Das Client-Projekt enthält einen Verweis auf die Typbibliothek der Komponente.
- Der Verweis bezieht sich auf eine Schnittstelle oder eine erstellbare Klasse (die Standardschnittstelle).
- Das Objekt implementiert `vTables` für eine benutzerdefinierte Schnittstelle oder eine duale Schnittstelle, d.h. das Objekt ist kein reines `IDispatch`-Objekt.

Einige Visual Basic-Programmierer setzen voraus, dass die Funktion `CreateObject` immer in einer späten Bindung resultiert, im Gegensatz zu einer direkten `vTable`-Bindung. Diese Annahme ist falsch. Es ist egal, wie Sie ein Objekt erzeugen oder eine Verbindung dazu herstellen. Wichtig ist der Typ der Variablen oder des Parameters, die Sie verwenden, um eine Verbindung zu einem Objekt herzustellen. Die Regeln bleiben dieselben. Wenn Sie den `Object`-Datentyp benutzen, erfolgt eine *späte Bindung*. Wenn Sie den Namen einer Klasse oder einer benutzerdefinierten Schnittstelle benutzen, erfolgt eine *direkte Bindung*.

nierten Schnittstelle verwenden, erfolgt eine direkte vTable-Bindung, unabhängig davon, ob Sie `CreateObject` oder den `New-Operator` einsetzen.

Hier soll noch darauf hingewiesen werden, dass die Programmierer den Begriff *frühes Binden* für unterschiedliche Sachverhalte verwenden. Einige Entwickler bezeichnen mit dem Begriff die DISPID-Bindung. Andere bezeichnen damit die direkte vTable-Bindung. Beachten Sie, dass die einheitliche Verwendung des Begriffs nicht garantiert ist. Man ist sich lediglich darüber einig, dass sich die frühe Bindung von der späten Bindung unterscheidet.

COM außerhalb von Prozessen

Dieses Buch hat bisher lediglich die Zusammenarbeit zwischen Client und Objekt für einen einzelnen Prozess innerhalb des Gültigkeitsbereichs eines einzelnen Ausführungs-Threads beschrieben. Wenn ein Client an ein prozessinternes Objekt gebunden wird, kann er Methoden direkt über die Verwendung von Funktionszeigern ausführen lassen, die in einer vTable gespeichert sind. Diese Art der Zusammenarbeit ist sehr effizient, da sich der Client-Code und der Objekt-Code einen Thread, einen Aufruf-Stack und dieselben Speicheradressen teilen. Wird das Objekt jedoch in einem anderen Prozess ausgeführt, kann keine dieser Ressourcen gemeinsam genutzt werden.

Die Funktionszeiger, die in den vTables gespeichert sind, haben über die Prozessgrenzen hinaus keine Bedeutung. Ein Client kann keinen Remote-Funktionszeiger verwenden, um auf ein Objekt in einem anderen Prozess zuzugreifen. Doch wie kann COM einen Methodenaufruf vom Client-Prozess zum Objektprozess weiterleiten? COM ermöglicht diese Remote-Kommunikation durch zwei Hilfsobjekte, die als *Proxy* und *Stub* bezeichnet werden.

Abbildung 3.9 zeigt, wie der Proxy und der Stub eingesetzt werden. Der Proxy wird im Client-Prozess ausgeführt, während der Stub im Objektprozess läuft. Der Proxy und der Stub richten einen Kommunikationskanal ein und verwenden *Remote Procedure Calls (RPC)* als Interprozesskommunikationsmechanismus. RPC ist ein verbindungsorientiertes Protokoll, das auf synchronen Anforderungs-/Antwort-Paaren basiert. Der Proxy und der Stub verwenden den RPC-Kanal, um Daten wie beispielsweise die Parameter für eine Methode während der Ausführung dieser Methode zu übertragen. Dazu müssen der Proxy und der Stub die Daten jedoch in einer Form serialisieren, die über die Prozess- und Rechnergrenzen hinweg übertragen werden kann. Diese Serialisierung von Methodenparametern für die Übertragung über die Proxy-/Stub-Schicht wird auch als *Marshaling* bezeichnet.

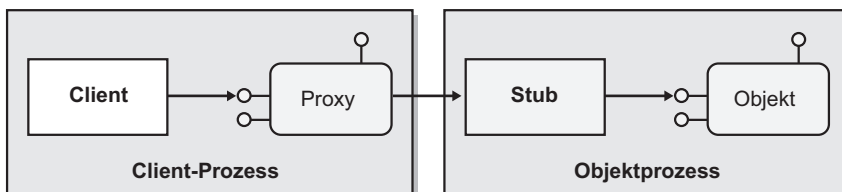


Abbildung 3.9: Die Remoting-Architektur von COM verlangt, dass eine Proxy-/Stub-Schicht zwischen dem Client und dem Objekt verwendet wird. Der Proxy und der Stub richten einen RPC-Kanal ein, der sie miteinander verbindet und Methodendaten zwischen den beiden überträgt.

Wenn der Client eine Methode auf dem Proxy aufruft, leitet der Proxy diese Anforderung an den Stub weiter. Damit diese Anforderung korrekt übertragen wird, muss der Proxy die Eingangsparameter der Methode zunächst unter Verwendung des Marshalings »verpacken« und anschließend zum Stub übermitteln. Wenn der Stub die Anforderung erhält, »entpackt« er die Eingangspara-

meter und führt den lokalen Aufruf auf dem Objekt aus. Nachdem das Objekt die entsprechende Methode ausgeführt und die Kontrolle an den Stub zurückgegeben hat, bereitet der Stub ein Antwortpaket vor, in das unter Verwendung des Marshalings die Ausgangsparameter sowie ein Rückgabewert gepackt werden. Die Daten werden anschließend an den Proxy zurückgegeben. Der Proxy packt die Daten wiederum aus dem Antwortpaket aus und gibt die Programmsteuerung an den Client zurück.

Jetzt sollten Sie sich eine wichtige Frage stellen: Wer erzeugt Proxies und Stubs? Die Antwort ist: sie werden normalerweise von der COM-Laufzeitumgebung erzeugt. Wenn ein Client `CoCreateInstance` aufruft, um ein Objekt in einem anderen Prozess zu erzeugen, erzeugt das System automatisch den Proxy und den Stub. Nachdem der Client das Objekt erfolgreich aktiviert hat, sind auch Proxy und Stub vorhanden. Der Client wird an den Proxy und nicht an das Objekt gebunden. Es ist der Stub, nicht der Client, der an das Objekt gebunden wird.

Größtenteils erkennt die COM-Laufzeitumgebung, wann Proxies und Stubs benötigt werden, und fügt sie bei Bedarf ein. Das Konzept ist einfach. Wird ein Schnittstellenverweis vom Prozess des Objekts exportiert, erzeugt die COM-Laufzeitumgebung einen Stub. Wird ein Schnittstellenverweis in den Prozess des Clients importiert, erzeugt die COM-Laufzeitumgebung einen Proxy. Der Kommunikationskanal, der zwischen Proxy und Stub eingerichtet wird, ermöglicht es, Remote-Methodenaufrufe vom Client zum Objekt und zurück zu übertragen.

Was passiert, wenn in einem COM-Methodenaufruf ein Objektverweis über die Prozessgrenzen hinweg übergeben wird? Auch hier macht die COM-Laufzeitumgebung automatisch das Richtige. Sie weiß, dass ein Proxy-/Stub-Paar erzeugt werden muss, wenn sich das Objekt und der Client, der den Verweis erhält, in unterschiedlichen Prozessen befinden. Sie könnten es nicht leichter haben. Während C++-Programmierer größeren Schwierigkeiten gegenüberstehen, wenn sie explizit Proxies und Stubs erzeugen müssen, indem sie Funktionen der COM-Bibliothek aufrufen, müssen sich Visual Basic-Programmierer darüber keinerlei Gedanken machen. Proxies und Stubs werden immer automatisch im Hintergrund erzeugt.

Der Vorteil dieser Remoting-Architektur besteht darin, dass weder der Client noch das Objekt wissen, dass sie voneinander entfernt sind. Der Client glaubt, der Proxy sei das Objekt. Das Objekt denkt, der Stub sei der Client. Dies ermöglicht COM-Programmierern, Code für Clients und Objekte zu schreiben, unabhängig davon, ob die Objekte von einem prozessinternen oder von einem prozessexternen Server aktiviert werden. Dieses leistungsfähige Feature ist unter dem Namen *örtliche Transparenz* bekannt.

Für jede verbundene Schnittstelle ist ein Proxy-/Stub-Paar vorhanden. Ein Client und ein Objekt können somit durch mehrere Proxy-/Stub-Paare miteinander verbunden sein. Es ist sinnvoll, dass das Proxy-/Stub-Paar mit der Schnittstelle verknüpft ist, weil die Schnittstellendefinition die Methoden beschreibt, auf die ein Remote-Zugriff erfolgen soll.

Der Schlüssel zu dem Code, mit dem Proxies und Stubs erzeugt werden, liegt in den Schnittstellendefinitionen. Wie Sie wissen, können COM-Schnittstellen in IDL-Quelldateien oder Typbibliotheken definiert werden. IDL ist eine Sprache, die ursprünglich dafür vorgesehen war, das Problem des Marshalings von Funktionsparametern für die Übertragung zwischen verschiedenen Maschinen zu lösen. Deshalb bedingt IDL, dass Sie die Parameter mit Attributen wie `[in]`, `[out]` und `[in, out]` spezifizieren. Anhand dieser Parameterattribute erkennen Proxy und Stub, in welche Richtung das Marshaling der Daten für einen Remote-Methodenaufruf erfolgen soll.

Eine Typbibliothek kann wie eine IDL-Quelldatei Schnittstellendefinitionen aufnehmen. Einer der wichtigsten Unterschiede zwischen den beiden ist, dass eine Typbibliothek im binären Format vorliegt. Sie ist zur Laufzeit einfacher zu parsen. COM unterstützt einen Dienst, der Proxies und Stubs nach Bedarf erzeugt, indem er die Schnittstellendefinitionen einer Typbibliothek aus-

wertet. Dieser Dienst wird auch als *Universal-Marshaler* bezeichnet. Es gibt aber auch andere Namen dafür, beispielsweise *Typbibliothek-Marshaler* oder *Automatisierungs-Marshaler*. Der Begriff *Automatisierungs-Marshaler* kann jedoch missverstanden werden, weil der Marshaler nicht verlangt, dass Clients und Objekte über `IDispatch` kommunizieren.

Die Rolle des Universal-Marshalers

Der Proxy und der Stub haben klar definierte Aufgaben. Sie müssen zusammenarbeiten, um dem Client und dem Objekt den Eindruck zu vermitteln, dass sie in einem einzigen Prozess auf einem einzigen Thread ausgeführt werden. Dazu legen Sie im Prozess des Objekts einen Aufruf-Stack an, der identisch mit dem des Client-Prozesses ist. Alle Daten aus dem Aufruf-Stack des Prozesses müssen verpackt und an den Prozess des Objekts geschickt werden. Darüber hinaus erfordern alle Zeiger auf dem Aufruf-Stack des Clients, dass der Proxy die Daten, die von den Zeigern referenziert werden, mit Hilfe des Marshaling verpackt und versendet. Der Stub ist für das Auspacken der Daten und die Einrichtung des Aufruf-Stacks verantwortlich. Dazu müssen möglicherweise auch Datenelemente ausgepackt werden, die nicht stackbasiert sind, und dann stackbasierte Zeiger eingerichtet werden, die auf diese Elemente verweisen.

Wie Sie sich vorstellen können, kann der Code, der das Marshaling für ein Proxy-/Stub-Paar durchführt, relativ kompliziert sein. Glücklicherweise können Sie auf den Universal-Marshaler zurückgreifen, um den erforderlichen Proxy-/Stub-Code zu erzeugen. Der Universal-Marshaler ist Teil der *OLEAUT32.DLL*. Wenn Sie es mit Schnittstellen zu tun haben, die den Universal-Marshaler verwenden, ruft die COM-Laufzeitumgebung den Code in *OLEAUT32.DLL* auf, sobald sie feststellt, dass ein Proxy oder ein Stub benötigt werden.

Beachten Sie, dass C++-Programmierer häufig ihren eigenen, benutzerdefinierten Proxy-/Stub-Code erstellen, statt den Universal-Marshaler zu verwenden. Die C++-Programmierer müssen sehr viel mehr Aspekte berücksichtigen als die Visual Basic-Programmierer. Sie können jedoch davon ausgehen, dass die Schnittstellen, die Sie in Ihren Visual Basic-Komponenten implementieren, auf den Universal-Marshaler zurückgreifen, um ihre Proxies und Stubs zu erzeugen.

Der Universal-Marshaler verwendet Schnittstellendefinitionen in Typbibliotheken, um Proxies und Stubs zu erzeugen. Sie haben in diesem Kapitel bereits erfahren, dass Visual Basic die Schnittstellendefinitionen in Typbibliotheken auch verwendet, um während des Kompilierens Code für die direkte vTable-Bindung zu erzeugen. Jetzt sehen Sie, dass die Typbibliotheken zur Laufzeit noch eine weitere wichtige Rolle spielen. Sie sind der Schlüssel für die Interprozessverbindungen zwischen Clients und Objekten.

Schnittstellen, die den Universal-Marshaler verwenden, sind in Typbibliotheken und in IDL mit dem Attribut `[oleautomation]` definiert. Dadurch unterscheiden sich Schnittstellen, die den Universal-Marshaler verwenden, von Schnittstellen, die einen eigenen, benutzerdefinierten Proxy-/Stub-Code verwenden. Wird eine Typbibliothek registriert, wird ein Schlüssel in der Registrierung angelegt, der dem Universal-Marshaler mitteilt, wo er die Typbibliothek finden kann. Zum Registrierungszeitpunkt werden der Registrierung für jede mit dem Attribut `[oleautomation]` deklarierte IID Konfigurationsinformationen hinzugefügt.

Beachten Sie, dass jedes Proxy-/Stub-Paar auf einer eindeutigen IID basiert. Muss die COM-Laufzeitumgebung einen Proxy oder einen Stub erzeugen, wertet sie den Registrierungsschlüssel *HKEY_CLASSES_ROOTInterface* aus, um Konfigurationsinformationen für die betreffende Schnittstelle zu ermitteln. Abbildung 3.10 zeigt die Registrierungseinträge für eine `oleautomation-IID`. Der Schlüssel *ProxyStubClsid32* teilt der COM-Laufzeitumgebung mit, wo sie den Code für das Erzeugen von Proxies und Stubs findet. Für `oleautomation`-Schnittstellen ist dies die CLSID für den Universal-Marshaler.

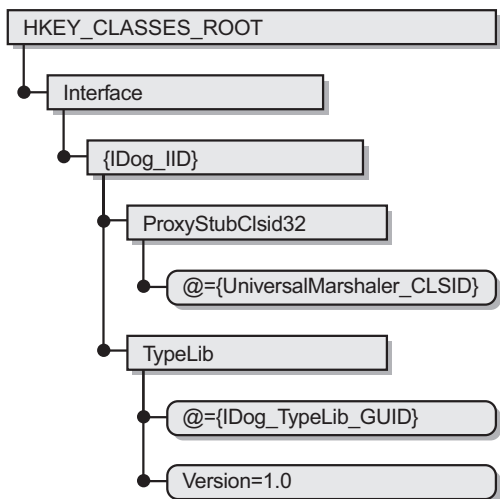


Abbildung 3.10: Die Registrierungseinstellungen für eine mit dem Attribut `oleautomation` gekennzeichnete Schnittstelle weisen die COM-Laufzeitumgebung an, den Universal-Marshaler zu verwenden. Außerdem teilen sie dem Universal-Marshaler mit, welche Typbibliothek die Schnittstellendefinitionen enthält.

Wenn die COM-Laufzeitumgebung feststellt, dass sie einen Proxy oder einen Stub für eine `oleautomation`-Schnittstelle erzeugen muss, gibt sie diese Anforderung an den Universal-Marshaler weiter. Der Universal-Marshaler durchsucht die Registrierung nach der GUID für die der IID zugeordnete Typbibliothek. Der Universal-Marshaler ermittelt die Typbibliothek unter Verwendung der in Abbildung 3.11 gezeigten Informationen. Nachdem der Universal-Marshaler den Pfad ermittelt hat, kann er die Typbibliothek laden, die Schnittstellendefinition lesen und einen Proxy oder einen Stub erzeugen.

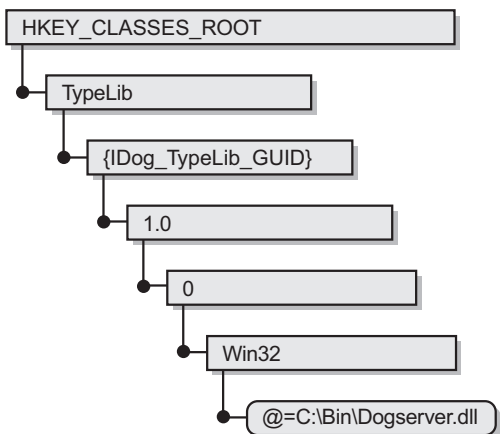


Abbildung 3.11: Der Universal-Marshaler verwendet den Schlüssel `TypeLib`, um den physischen Pfad zu der Typbibliothek zu ermitteln. Dieser Pfad kann auf eine eigenständige `.TLB`-Datei oder auf einen COM-Server mit eingebetteter Typbibliothek verweisen.

In späteren Kapiteln werden Sie erfahren, dass die COM+-Laufzeitumgebung auch in einigen anderen Situationen Proxies und Stubs erzeugt. Ich werde dann auch die Begriffe *Kontexte* und *Apartments* einführen. Ein Kontext ist eine Gruppe von Objekten, die innerhalb eines Apartments ausgeführt werden. Ein Apartment ist eine Gruppe von Kontexten, die innerhalb eines Prozesses ausgeführt werden. Client und Objekt kommunizieren häufig über Proxies und Stubs, auch wenn sie innerhalb desselben Prozesses ausgeführt werden. Ich werde noch erklären, warum das so gehandhabt wird. Hier wollen wir uns auf Proxies und Stubs konzentrieren, die es erlauben, Methodenaufrufe über Prozess- und Rechnergrenzen hinweg auszuführen.

Beachten Sie, dass sowohl für den Client als auch für das Objekt eine Typbibliothek erforderlich ist, um ein Proxy-/Stub-Paar erfolgreich zu erstellen. Werden Client und Objekt auf unterschiedlichen Maschinen ausgeführt, brauchen beide Computer eine lokale Kopie der Typbibliothek. Weil das Objekt immer eine lokale Version des Servers besitzt, kann es auf die Typbibliothek zurückgreifen, die sich innerhalb des binären Abbilds für den Server befindet. Der Client-Computer dagegen braucht den Server nicht. Er benötigt nur die Typbibliothek. Häufig wird für die Installation auf Client-Maschinen eine eigenständige Typbibliothek erzeugt, sofern die Clients Objekte über das Netzwerk aktivieren.

Beobachtungen zum prozessexternen COM

Beachten Sie unbedingt, dass die Low-Level-Details der prozessexternen Kommunikation abstrahiert wurden und im Hintergrund verborgen bleiben. RPC ist wichtig für COM, aber Sie brauchen sich keine Gedanken darüber zu machen, wie COM eine RPC-Verbindung aufbaut und nutzt. Stellen Sie sich nur vor, wie viel komplizierter es wäre, eine verteilte Anwendung zu erstellen, wenn Sie direkt für eine Interprozesskommunikationsebene wie beispielsweise RPC oder Sockets programmieren müssten.

Anschließend sollten Sie drei für die Leistung relevante Aspekte des prozessexternen COM betrachten. Der erste ist, dass prozessexterne Methodenaufrufe sehr viel länger dauern als prozessinterne Aufrufe. Ein prozessexterner Aufruf ist ungefähr eintausend Mal langsamer als ein prozessinterner Aufruf mit direkter vTable-Bindung. Die Proxy-/Stub-Schicht erfordert immer einen Thread-Wechsel und das Marshaling, so dass ein wesentlicher zusätzlicher Aufwand entsteht.

Zweitens gilt, dass Remote-Methodenaufrufe für Visual Basic-Objekte immer synchron erfolgen. Das liegt in der synchronen Natur von RPC begründet. Wenn der Client eine Methode für ein Remote-Objekt aufruft, wird der aufrufende Thread im Prozess des Clients blockiert, bis der Aufruf zurückkehrt. Das bedeutet, der Client muss warten, während die RPC-Anforderung zum Stub übertragen wird, der Stub den Aufruf ausführt und die RPC-Antwortnachricht schließlich zum Proxy zurückkehrt. Unter Windows wird zwar jetzt auch asynchrones RPC unterstützt, aber Sie müssen schon in C++ programmieren, um dies nutzen zu können. Jeder RPC-basierte COM-Methodenaufruf, den Sie aus oder für Visual Basic-Code ausführen, wird synchron ausgeführt.

Beachten Sie als Drittes, dass Objekte, die Sie mit Visual Basic erstellen, ausschließlich als Referenz und nicht als Wert übergeben werden. Denken Sie keinesfalls, dass Sie ein Visual Basic-Objekt einfach von einer Maschine zu einer anderen übertragen können. Ihre Methoden können Parameter definieren, bei denen es sich um Objektverweise, nicht aber um Objekte handelt. Die aktuelle Version von Visual Basic erlaubt Ihnen, das `ByVal`-Schlüsselwort in Parameterdefinitionen vor Objekttypen zu setzen. Diese Parameter werden jedoch weiterhin mit der Semantik der Referenzübergabe interpretiert. Wenn Sie einen Verweis auf ein prozessexternes Objekt benutzen, bedingt der Zugriff auf jede Methode oder Eigenschaft des Objekts einen aufwendigen Datenverkehr zwischen den Komponenten.

Prozessexterne Objekte, die mit Visual Basic erstellt werden, sind immer mit Proxies und Stubs verbunden, die vom Universal-Marshaller erzeugt werden. Diese Technik, die einen Client automatisch an ein prozessexternes Objekt bindet, ist eine Art *Standard-Marshaling*. Viele Programmierer, die andere Sprachen als Visual Basic benutzen, bevorzugen ebenfalls das Standard-Marshaling, weil es einfach einzurichten ist.

C++-Programmierer können auf das Standard-Marshaling verzichten, da ihnen das *benutzerdefinierte Marshaling* zur Verfügung steht. Anwender, die ihren eigenen Marshaling-Code schreiben möchten, indem sie eine spezielle Schnittstelle namens `IMarshal` implementieren, können den Kommunikationskanal in einer Weise optimieren, die das Standard-Marshaling nicht bietet. Ein Programmierer kann beispielsweise eine Wertübergabe-Semantik mit einem benutzerdefinierten Marshaling-Code implementieren. Beachten Sie jedoch, dass das benutzerdefinierte Marshaling auch einige wesentliche Einschränkungen aufweist. Erstens bedingt es die Verwendung von C++ auf der Objektseite. Zweitens wird es für konfigurierte Komponenten weder in COM+ noch in MTS unterstützt. Es kann nur für nicht konfigurierte Komponenten eingesetzt werden.

Sie können als Visual Basic-Programmierer keine Komponenten erstellen, die das benutzerdefinierte Marshaling unterstützen. Sie werden jedoch möglicherweise mit einer häufig verwendeten, nicht konfigurierten Komponente konfrontiert, die das benutzerdefinierte Marshaling unterstützt. Das ADO-Objekt `Recordset` implementiert `IMarshal`. Dies bedeutet, dass Sie die Methodenparameter und Rückgabewerte für die ADO-Komponente `Recordset` definieren können und die gesamten Daten der Datensatzgruppe (nicht nur ein Verweis darauf) über das Netzwerk und die Proxy-/Stub-Schicht übertragen werden. Beachten Sie bitte, dass Sie nicht verbundene, clientseitige Datensatzgruppen benutzen müssen, damit dies möglich ist. Das Marshaling von ADO-Datensatzgruppen ist eine von mehreren Techniken, um Daten in einer effizienten Weise über das Netzwerk zu bewegen. Wir werden dieses Thema ausführlich im elften Kapitel behandeln.

Objektaktivierung und örtliche Transparenz

In der ersten Auflage dieses Buches habe ich erklärt, wie die Objektaktivierung in einem prozessexternen Server wie beispielsweise einer ActiveX-EXE funktioniert. Ich werde diese Details hier nicht nennen, weil sie nicht mehr relevant für Entwickler sind, die mit COM+ verteilte Anwendungen erstellen. Kapitel 6 erklärt, wie die prozessexterne und die Remote-Aktivierung in einer COM+-Anwendung durchgeführt wird. Beachten Sie nur, dass Clients und Objekte nicht erkennen, ob sie im selben Prozess oder überhaupt auf derselben Maschine ausgeführt werden.

Abbildung 3.12 zeigt drei Möglichkeiten, einen Server einzusetzen. Die Komponenten, die Sie in einer ActiveX-DLL erzeugen, müssen daraufhin konfiguriert werden, ob sie prozessintern, lokal oder über das Netzwerk ausgeführt werden. Für Client-Code und Komponentencode ist alles transparent. Sie können sich zwischen den drei Anwendungsszenarien bewegen, ohne Ihre Server oder Client-Applikationen neu erstellen zu müssen. Die Änderung der Position eines Clients oder einer Komponente erfordert nur einen geringen Aufwand, wenn ein einfaches Verwaltungswerkzeug verwendet wird.

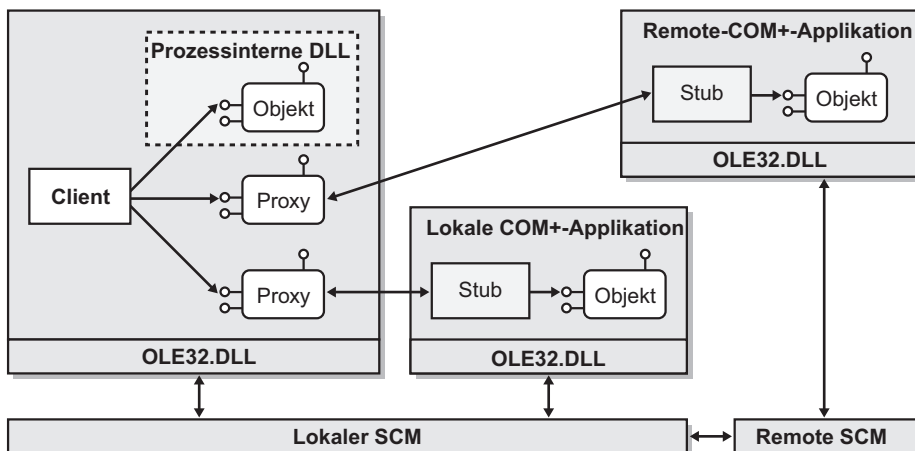


Abbildung 3.12: Örtliche Transparenz bewirkt, dass sich die Programmierer nicht mehr mit den Details der Remote-Aktivierung oder Interprozesskommunikation beschäftigen müssen.

Die Möglichkeit, dass die Programmierer Client-Code und Objekt-Code für eine prozessinterne Beziehung schreiben und dieser Code automatisch über Prozessgrenzen hinweg eingesetzt werden kann, ist eines der leistungsfähigsten Merkmale von COM. Diese örtliche Transparenz bewirkt, dass sich die Programmierer nicht mehr mit den komplexen Details der Interprozesskommunikation beschäftigen müssen. Außerdem bedeutet es, dass die Komponenten im gesamten Netzwerk eingesetzt werden können, was nur geringen oder gar keinen Einfluss auf den Code hat. Sie können einen Client, der für die Aktivierung eines bestimmten Objekts aus einer prozessinternen DLL heraus programmiert wurde, stattdessen ein Remote-Objekt aktivieren lassen, indem Sie einige kleine Änderungen an der Registrierung vornehmen. Sie müssen dazu keine Codezeile neu schreiben.

Die Tatsache, dass Ihr Komponentencode automatisch sowohl in prozessinternen als auch in prozessexternen Situationen funktioniert, bedeutet aber nicht gleichzeitig, dass er effizient ist. Code, der für prozessinterne Objekte erstellt wurde, ist möglicherweise nicht skalierbar, wenn er in einer Applikation eingesetzt wird, die über das Netzwerk ausgeführt wird. Es gibt verschiedene Codierungstechniken, die für prozessinterne Szenarien geeignet sind, aber bei der Verwendung der Proxy-/Stub-Ebene eine nur unzureichende Leistung erzielen. Kapitel 11 erklärt die Bedeutung des Entwurfs von Schnittstellen, die effizient über die Proxy-/Stub-Ebene eingesetzt werden.

Zusammenfassung

Dieses Kapitel hat die wesentlichen Konzepte und Details von COM beschrieben. Sie brauchen ein fundiertes Grundwissen über COM, damit Sie alle neuen Funktionen von COM+ verstehen, die darauf aufsetzen. Im restlichen Buch werde ich immer den Begriff *COM* verwenden, wenn ich von Aspekten der Windows-Plattform spreche, die in diesem Kapitel erklärt wurden. Wenn ich den Begriff *COM+* verwende, spreche ich über eine Laufzeitebene und ein für Windows 2000 spezifisches Programmiermodell. Diese Unterscheidung zwischen COM und COM+ kann verwirrend sein, weil COM+ auf COM aufsetzt. Wenn es um Windows 2000 geht, unterscheiden sich die COM-Laufzeitumgebung und die COM+-Laufzeitumgebung nicht.

Die ursprüngliche COM-Version erfüllt vier wichtige Design-Anforderungen. Sie basiert auf der binären Wiederverwendbarkeit, sie ist objektorientiert, sie ist sprachunabhängig und sie stellt eine Grundlage für die Interprozesskommunikation bereit. Nachdem Sie dieses Kapitel gelesen haben, wissen Sie, dass die Architekten von COM mit unzähligen Details zu tun hatten, um diese Anforderungen zu erfüllen.

COM basiert auf einer formalen Abtrennung der Schnittstelle von der Implementierung. Auf der physischen Ebene dient die *vTable* als Speicherdarstellung einer Schnittstelle. Einige Clients greifen über die direkte *vTable*-Bindung auf Objekte zu, während andere, wie zum Beispiel Skripting-Clients, Automatisierungsobjekte benötigen, die über die *IDispatch*-Schnittstelle die entsprechende Funktionalität bereitstellen.

Visual Basic verbirgt die kompliziertesten Details von COM. Auf der Komponentenseite bietet Visual Basic eine Unterstützung für Programmierer, die sich nicht mit der zusätzlichen Komplexität benutzerdefinierter Schnittstellen belasten wollen. Auf der Client-Seite unterstützt es die direkte *vTable*-Bindung, indem es beim Kompilieren die Schnittstellendefinitionen in den Typbibliotheken auswertet. Sie sollten verstehen, warum Objekte *IUnknown* und *IDispatch* implementieren. Gleichzeitig sollten Sie aber dankbar sein, dass Visual Basic die Details für Sie im Hintergrund ausführt.

Der SCM ist ein dynamischer Klassenlader, den Clients verwenden, um bei Bedarf Objekte zu erzeugen. Die Anforderungen der schnittstellenbasierten Programmierung erlauben es den Servern nicht, konkrete Klassen bereitzustellen. Statt dessen muss jeder Server ein verhältnismäßig kompliziertes Aktivierungsprotokoll unterstützen. Clients geben Aktivierungsanforderungen an den SCM, und der SCM gibt die Anforderung an den Server weiter. Auch hier greift der Visual Basic-Compiler ein und übernimmt die Verarbeitung aller Details im Hintergrund. Sie brauchen nicht sehr viel zu tun, damit die Clients in der Lage sind, Objekte aus Ihren Komponenten zu aktivieren.

Der letzte Teil des Kapitels beschrieb die Grundlagen der Remote-Schicht von COM. Die Proxy-/Stub-Ebene stellt die Architektur für Remote-Methodenaufrufe über Prozess- und Rechnergrenzen hinweg bereit, während Clients und Objekte nicht erkennen, was wirklich geschieht. Diese Details unterliegen der Verantwortlichkeit der COM-Laufzeitumgebung und des Universal-Marshalers. Der Universal-Marshaler erzeugt den Code für Remote-Methodenaufrufe, indem er die Schnittstellendefinitionen aus einer Typbibliothek auswertet. Daraus können Sie schließen, dass die Schnittstellen der Schlüssel zur nahtlosen Verteilung in COM sind.

4 Server entwerfen und erstellen

81	Komponenten in Server kompilieren
86	Server-Design
95	Komponenten für Skripting-Clients entwerfen
102	Fehler mit Servern auslösen

Dieses Kapitel erklärt zunächst, wie Sie Komponenten auf die Veröffentlichung vorbereiten und welche Optionen Sie beim Kompilieren eines Servers haben. Eine der ersten Entscheidungen ist dabei, ob Sie Ihre Komponenten in einer prozessinternen DLL oder einer prozessexternen EXE veröffentlichen sollten. Prozessinterne Objekte sind schneller, während prozessexterne Objekte eine höhere Fehlertoleranz und Sicherheit bieten. Außerdem werden Sie erfahren, warum Sie mit COM+ Ihre Komponenten in ActiveX-DLLs statt in ActiveX-EXEs packen müssen.

Dieses Kapitel erklärt ebenfalls einige wichtige Aspekte des Server-Designs, beispielsweise die Bereitstellung von Klassen und Schnittstellen, das Verwenden von Auflistungen und benutzerdefinierten Typen (UDTs, User Defined Types) sowie das Ändern von Prozedurattributen. Außerdem beschreibt es den Entwurf von Komponenten, die Skripting-Clients unterstützen.

Das Kapitel endet mit einer Erklärung der Fehlerverarbeitung in COM. Sie erfahren, wie Microsoft Visual Basic sein eigenes, internes Modell der Fehlerverarbeitung auf sehr transparente Weise auf die COM-Ausnahmen abbildet. Sie werden jedoch erkennen, dass Fehler von Objekten sprachneutral an Clients weitergegeben werden. Damit können Sie sehr gut abschätzen, welche Arbeit Visual Basic für Sie im Hintergrund erledigt, und wir diskutieren, welche Techniken die Komponenten in Ihren Servern verwenden sollten, um Fehler auszulösen.

Komponenten in Server kompilieren

Ein COM-Server muss eine oder mehrere Komponenten bereitstellen, damit er für eine Client-Applikation Nutzen bringt. Sie können also einen Server einfach als binäre Datei definieren, die mehrere Komponenten enthält. Wie Sie wissen, ist jede Komponente in der Typbibliothek des Servers als Co-Klasse definiert. Neben Co-Klassen kann die Typbibliothek auch andere COM-Datentypen definieren, beispielsweise Schnittstellen, Auflistungen und UDTs.

COM unterstützt zwei Server-Typen. COM-DLLs sind prozessinterne Server und COM-EXEs sind prozessexterne Server. Sie entscheiden, ob Ihre Klassen über eine prozessinterne DLL oder eine prozessexterne EXE bedient werden.

Die Visual Basic-IDE (Integrated Development Environment) bietet drei unterschiedliche Projekttypeneinstellungen für das Erstellen von COM-Servern an. Sie wählen diese Einstellungen aus,

wenn Sie ein neues Projekt öffnen, und ändern sie im Dialogfeld *Projekteigenschaften*, das Sie aus dem Menü *Projekt* aufrufen. Sie verwenden ein ActiveX-DLL-Projekt, um einen prozess-internen Server zu erstellen. Wenn Sie einen spezialisierten Typ eines visuellen prozessinternen Servers erzeugen möchten, öffnen Sie ein ActiveX-Steuerelement-Projekt. (In diesem Buch werden keine ActiveX-Steuerelement-Projekte besprochen.) Öffnen Sie ein ActiveX-EXE-Projekt, wenn Sie einen prozessexternen Server erstellen möchten.

Prozessinterne Server

Ein prozessinterner Server wie beispielsweise eine ActiveX-DLL ist normalerweise die effizienteste Möglichkeit, Komponenten zu verpacken. Ihr Code wird in den Adressraum der Client-Applikation geladen. Ein Objekt, das aus einem prozessinternen Server generiert wird, kann in der Regel direkt, das heißt ohne eine Proxy-/Stub-Schicht an den Client gebunden werden. Beim direkten Binden können sich der Client und das Objekt denselben Thread und dieselben Speicheradressen teilen. Nachdem das Objekt aktiviert wurde, wird die Kommunikation zwischen den beiden so schnell ausgeführt, als wäre die Klasse innerhalb der Client-Applikation definiert. Die bessere Leistung ist häufig einer der größten Vorteile eines prozessinternen Servers.

Ein prozessinterner Server weist jedoch auch einige Einschränkungen auf. Es gibt keine einfache Möglichkeit, unter Verwendung derselben ActiveX-DLL Daten in zwei verschiedenen Client-Applikationen gemeinsam zu nutzen, weil alle Variablen prozessspezifisch sind. Ein prozessinterner Server ist außerdem nicht so stabil wie ein prozessexterner Server, da er fest mit dem Prozess der Client-Applikation verbunden ist, in den er geladen wurde. Ein fehlerhaftes Objekt kann zu einem Absturz der Client-Applikation führen. In diesem Fall stürzt das Objekt ebenfalls ab.

Ein prozessinterner Server ist außerdem etwas unflexibel in Hinblick auf die Sicherheit. Ein Objekt, das aus einem prozessinternen Server erzeugt wurde, muss unter demselben Sicherheitskontext wie die Client-Applikation ausgeführt werden. Wenn beispielsweise zwei Anwender, Bob und Sally, Ihre ActiveX-DLL einsetzen, werden einige Ihrer Objekte unter Bobs Benutzernamen ausgeführt, während andere Objekte unter Sallys Benutzernamen eingesetzt werden. Das ist grundsätzlich kein Problem, aber vielleicht wünschen Sie bisweilen, dass alle Objekte, die von einem bestimmten Server aktiviert wurden, unter einem einzigen Benutzernamen ausgeführt werden. Dies ist aber nicht möglich, wenn Sie Objekte aus einer prozessinternen DLL in Ihre Client-Applikation laden. Dieser Sicherheitsaspekt ist insbesondere dann von Bedeutung, wenn Sie Middle-Tier-Applikationen einsetzen wollen. Wir werden in Kapitel 11 noch einmal auf dieses Thema zurückkommen.

Prozessexterne Server

Ein prozessexterner Server ist in Visual Basic als ActiveX-EXE implementiert. Die ausführbare Datei, die Visual Basic erstellt, wenn Sie eine ActiveX-EXE kompilieren, kann einen eigenen Win32-Prozess starten und steuern. Wenn ein Client ein Objekt aus einem prozessexternen Server aktiviert, sucht oder lädt der Service Control Manager (SCM) den Server-Prozess und initiiert das Erstellen eines neuen Objekts. Nachdem das Objekt und der Client aneinander gebunden wurden, besteht eine Proxy-/Stub-Schicht zwischen ihnen. Wie Sie wissen, führt diese Schicht zu einem wesentlich höheren Rechenaufwand. Aufrufe von Objekten in prozessexternen Servern werden immer langsamer ausgeführt als Aufrufe von Objekten in prozessinternen Servern.

Ein *lokaler Server* wird auf demselben Computer wie die Client-Applikation ausgeführt, während ein *Remote-Server* auf einem anderen Computer läuft. Für COM-Server spielt es jedoch keine Rolle, ob Ihre ActiveX-EXE lokal oder auf einem Computer im Netzwerk ausgeführt wird. Ei-

ne ActiveX-EXE ist immer als lokaler Server konfiguriert. Wenn der lokale Server ausgeführt wird, können Sie einige Konfigurationsänderungen an der Windows-Registrierung vornehmen, so dass auch Remote-Clients den Server nutzen können. Sie müssen keine besonderen Einstellungen an einem ActiveX-EXE-Projekt vornehmen, um zwischen diesen Einsatzoptionen zu unterscheiden.

Prozessexterne Server sind weitaus stabiler als prozessinterne Server. Sie sind mindestens so stabil wie das Betriebssystem, unter dem sie ausgeführt werden. Bei einem prozessinternen Server können sich der Client und das Objekt gegenseitig zum Absturz bringen. Eine prozessexterne Beziehung bietet eine eingebaute Fehlertoleranz. Ein Client kann erkennen, ob ein Objekt abgestürzt ist. Er überprüft dazu den Wert `HRESULT`, der von jeder Methode zurückgegeben wird. Wie dies geschieht, erfahren Sie später in diesem Kapitel. Die Infrastruktur von COM kann außerdem einen abgestürzten Client erkennen und dem entsprechenden Objekt mitteilen, dass die Verbindung zu diesem Client nicht länger gültig ist. Mit einem prozessexternen Server können der Client oder das Objekt somit auch dann ausgeführt werden, wenn die jeweils andere Komponente nicht mehr läuft.

Doch wie entscheiden Sie sich zwischen einer ActiveX-DLL und einer ActiveX-EXE? Denken Sie zunächst über die Leistung nach. Wenn Ihre Objekte ausschließlich von einer einzelnen Client-Applikation verwendet werden, sollten Sie Ihre Co-Klassen in einer ActiveX-DLL aufbewahren. Dies ist die beste Lösung, wenn Ihr Code beispielsweise Zinssätze oder Umsatzsteuern berechnet. Sie können Ihre DLL zusammen mit dem Client auf dem Desktop-Computer des Anwenders installieren. Auf diese Weise erhalten Sie die bestmögliche Leistung.

Aber wie gehen Sie vor, wenn Sie Ihre Objekte über das Netzwerk ausführen möchten? Eine scheinbar offensichtliche Lösung besteht darin, ein ActiveX-EXE-Projekt zu erstellen, da eine ActiveX-EXE Remote-Clients mit Objekten bedienen kann. Sie verfügen jedoch über eine andere Möglichkeit, die weitaus flexibler ist. Der nächste Abschnitt beschreibt, wie Sie mit verteilten Objekten arbeiten, die auf dem Konzept eines von COM+ bereitgestellten Ersatzprozesses basieren.

Komponenten für COM+ verpacken

Die Strategie, die Microsoft für die Entwicklung verteilter Applikationen verfolgt, basiert größtenteils auf dem Konzept der so genannten *Ersatzprozesse*. Ein Ersatzprozess ist eine vom System bereitgestellte Laufzeitumgebung, die Ihren Objekten als Host dient. Beispielsweise stellt COM+ eine Container-Applikation namens *DLLHOST.EXE* bereit. Jede COM+-Server-Applikation wird in einer eigenen separaten Instanz von *DLLHOST.EXE* ausgeführt. Wenn Sie Ihre Objekte in dieser Laufzeitumgebung ausführen wollen, können Sie Ihre Komponenten nicht in einer ActiveX-EXE verpacken. Die einzige Möglichkeit ist, sie in einer ActiveX-DLL zu verpacken.

Nachdem Sie Ihre Komponenten zu einer ActiveX-DLL kompiliert und in einer COM+-Server-Applikation korrekt installiert haben, können Clients Objekte über das Netzwerk aktivieren. Sie müssen Ihre Komponenten zwar in einen prozessinternen Server verpacken, aber COM+ bietet dennoch die Vorteile eines prozessexternen Servers. Sie verfügen über Fehlertoleranz, die Möglichkeit, Speicher gemeinsam zu nutzen oder alle Objekte unter einem bestimmten Benutzerkonto auszuführen.

Es gibt einen wesentlichen Unterschied zwischen einem prozessexternen Server, der von einer oder zwei Client-Applikationen verwendet wird, und einem prozessexternen Server, der Objekte für hunderte von Client-Applikationen ausführt. Ein prozessexterner Server, auf den sehr viele Anwender zugreifen, muss mit sehr viel mehr Problemen im Hinblick auf die Skalierbarkeit zu-recht-kommen. Zu diesen Problemen zählen zum Beispiel das Thread-Pooling, die Sicherheit und

die Verwaltung von Datenbankverbindungen. Eine COM+-Server-Applikation bietet eine skalierbare Architektur, eine ActiveX-EXE dagegen nicht. Aus diesem Grund werde ich mich in diesem Buch hauptsächlich auf ActiveX-DLLs konzentrieren und weniger auf ActiveX-EXEs eingehen. Und auch für Sie wird es besser sein, sich damit zu beschäftigen, wie man Komponenten für COM+-Applikationen erzeugt.

Das Konzept einer generischen Laufzeitumgebung wie der von COM+ bereitgestellten ist aus mehreren Gründen äußerst verlockend. Zunächst nimmt sie den Programmierern ein wenig Arbeit ab, da sie den größten Teil des Infrastruktur-Codes zur Verfügung stellt, der für die Skalierung einer Middle-Tier-Applikation benötigt wird. Zweitens ermöglicht sie Microsoft, die gesamte Infrastruktur in einer einzelnen Codebasis aufzubewahren und auf mehrere unterschiedliche Sprachen aufzuteilen. Schließlich können COM-Objekte in dieser Umgebung auch dann konsistent ausgeführt und verwaltet werden, wenn sie mit verschiedenen Sprachen und Werkzeugen erstellt wurden. Am wichtigsten erscheint mir hier die Tatsache, dass Sie Ihre Komponenten fast immer in ActiveX-DLLs veröffentlichen.

Erstellen eines Servers

Visual Basic macht das Erstellen eines Servers unglaublich einfach. Sogar ein vierjähriges Kind kann eine ActiveX-DLL oder eine ActiveX-EXE in der Visual Basic-IDE mit Hilfe des *Erstellen*-Befehls aus dem Menü *Datei* erzeugen. Nach Auswahl dieses Menüeintrags wird ein Dialogfeld mit unterschiedlichen Optionen und einer OK-Schaltfläche geöffnet. Sie klicken einfach auf OK, um das aktuelle Projekt in einem binären Server erstellen zu lassen.

Bei Ausführung des Befehls *Erstellen* finden einige wichtige Ereignisse im Hintergrund statt. Abbildung 4.1 zeigt, welche Elemente Visual Basic in den Server integriert. Zunächst veröffentlicht Visual Basic automatisch Informationen in einer Typbibliothek und ordnet diese in Ihrer binären Server-Datei an. Die Typbibliothek ist für andere Entwicklungswerkzeuge wichtig, die beim Kompilieren vTable-Bindungen für Ihre Objekte erstellen müssen. Auch der Universal-Marshaler verwendet diese Typbibliothek, um zur Laufzeit Proxies und Stubs anzulegen.

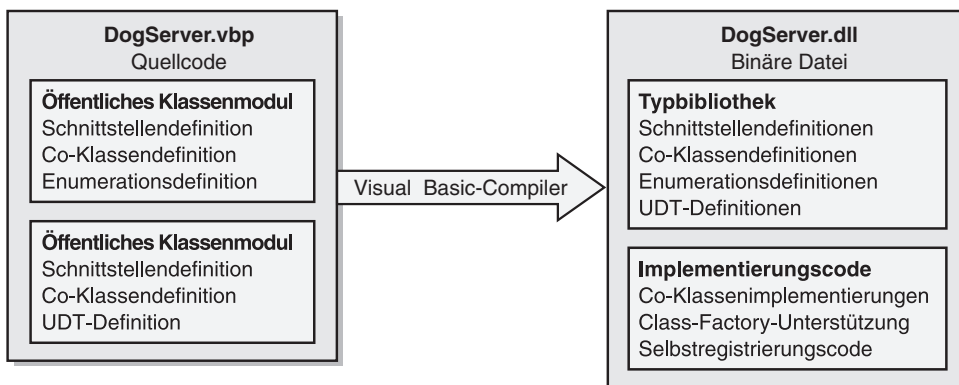


Abbildung 4.1: Wenn Sie mit dem Befehl *Erstellen* im Menü *Datei* einen Server erstellen, erledigt Visual Basic viel Arbeit für Sie.

Zweitens erzeugt Visual Basic für jede Komponente eine Implementierung für alle unterstützten Schnittstellen sowie eine Implementierung von `IUnknown` und `IDispatch`. Drittens integriert Visual Basic eine Unterstützung von Class-Factorys für jede Komponente. Und schließlich fügt Visual

Basic Ihrem Server den Code hinzu, der die Selbstregistrierung vornimmt. Der Code für die Selbstregistrierung fügt die Einträge für jede CLSID und IID in die Registrierung ein. Außerdem nimmt er diverse Einträge für die Typbibliothek vor.

Nachdem Visual Basic den Server erstellt hat, registriert es ihn auf Ihrer Entwicklungs-Workstation. Wenn Sie eine ActiveX-DLL erstellt haben, verwendet Visual Basic das Dienstprogramm *REGSVR32.EXE*, um Ihren Server zu registrieren. Wenn Sie eine ActiveX-EXE erstellen, verwendet das Entwicklungswerkzeug geeignete Befehlszeilenparameter, um die EXE als prozess-externen Server zu registrieren.

Probleme mit der Registrierung

Sicher kann man sagen, dass die Visual Basic-Programmierer unter allen Entwicklern in der Industrie heute am wenigsten den Ruf haben, besonders diszipliniert zu sein. Sie sollten jedoch die Disziplin der korrekten Registrierung und des Entfernens der Registrierung eines Servers auf Ihrer Entwicklungs-Workstation beherrschen.

Wie aber können Sie von sich selbst behaupten, die erforderliche Disziplin aufzubringen? Hier folgt ein praktischer Test. Rufen Sie das Dialogfeld *Verweise* im Menü *Projekt* auf. Wenn Sie zwei oder mehr Typbibliotheken mit genau derselben Beschreibung sehen, wissen Sie, dass Sie mehr darauf achten sollten, was in Ihre Registrierung geschrieben wird. Wenn Sie zehn oder mehr Typbibliotheken mit derselben Beschreibung sehen, sind Sie einer der Programmierer, die der Visual Basic-Gemeinde den Ruf verschafft haben, undiszipliniert zu sein.

Wie Sie wissen, fügt Visual Basic Code für die Selbstregistrierung ein und registriert den Server, wenn Sie eine ActiveX-DLL erstellen. Aber was passiert, wenn Sie eine zweite Version des Servers erstellen und die erste überschreiben? Die Visual Basic-IDE entfernt die Registrierung des ersten Builds, bevor dieser gelöscht wird. Sie brauchen sich kaum darum zu kümmern. Wenn Sie jedoch eine zweite Version der DLL an einer anderen Position erstellen und die erste nicht überschreiben, verbleiben möglicherweise Einträge für die erste DLL in der Registrierung. Und genau das wollen Sie nicht. Und es kommt noch schlimmer: Wenn Sie eine registrierte DLL löschen, ohne deren Registrierung rückgängig zu machen, löschen Sie den Code, der diese DLL-Einträge aus der Registrierung entfernen kann.

Es gibt zwei Situationen, in denen Sie die Registrierung einer DLL immer rückgängig machen sollten: Wenn Sie eine neuere Version an anderer Position erstellt haben, und bevor Sie die DLL löschen. Wenn Sie diese zwei Regeln befolgen, bleibt die Registrierung auf Ihrer Entwicklungs-Workstation jung und geschmeidig. Der Versuch die Registrierung einer DLL entfernen, die nie registriert wurde, hat keine besonderen Konsequenzen. Wenn Sie jedoch vergessen, die Registrierung einer DLL zu entfernen, belastet dies Ihre Windows-Registrierung. Die meisten erfahrenen COM-Entwickler wissen, dass es nur eine wirklich sichere Möglichkeit gibt, die Windows-Registrierung wiederherzustellen, nachdem sie durch undisziplinierte Programmierer unbrauchbar gemacht wurde. Und sicher haben Sie schon erraten, dass diese Möglichkeit mit dem folgenden Befehl beginnt:

FORMAT C:

Eine eigenständige Typbibliothek erstellen

Wie Sie wissen, braucht der Universal-Marshaler die Typbibliothek, um zur Laufzeit Proxies und Stubs zu erzeugen. Aber was passiert, wenn der Server auf einem Computer ausgeführt wird, während alle Client-Applikationen auf anderen Computern laufen? Auf dem Computer mit dem Server kann der Universal-Marshaler einfach die Typbibliothek verwenden, die dem binären Ab-

bild des Servers zugeordnet wurde. Der Client-Computer benötigt jedoch auch eine Kopie der Typbibliothek. Ohne Typbibliothek kann der Universal-Marshaler keinen Proxy erzeugen. Sie könnten die Server-Datei auf jeden Client-Computer kopieren, aber das wäre ineffizient. Die Server-Datei enthält alle Komponentenimplementierungen und die Client-Computer benötigen nur die Schnittstellendefinitionen.

Visual Basic ermöglicht es Ihnen, eine eigenständige Typbibliothek zu erstellen. Dazu markieren Sie das Kontrollkästchen *Remote-Server-Dateien* auf der Registerkarte *Komponente* im Dialogfeld *Projekteigenschaften*. Wenn Sie dieses Kontrollkästchen für ein ActiveX-DLL- oder ActiveX-EXE-Projekt markieren, erzeugt der Befehl *Erstellen* neben der im Server erstellten Typbibliothek eine separate Typbibliothek-Datei mit der Dateinamenerweiterung *.TBL*. Die primäre Motivation für die Auswahl des Kontrollkästchens *Remote-Server-Dateien* besteht darin, eine alleinstehende Typbibliothek für die Distribution an die Client-Computer in einer Netzwerkumgebung zu erstellen. Diese Typbibliothek erlaubt es dem Universal-Marshaler, nach Bedarf Proxies auf einer Client-Maschine anzulegen.

Verschiedene Optionen beim Kompilieren

Bevor wir uns weiter mit dem Entwurf von Servern beschäftigen, will ich kurz auf einige der Optionen eingehen, die Ihnen beim Kompilieren von Servern zur Verfügung stehen. Im Dialogfeld *Projekteigenschaften* gibt es das Kontrollkästchen *Unbeaufsichtigte Ausführung*, das den Visual Basic-Compiler anweist, die Interaktion mit dem Benutzer zu unterdrücken. Wenn Sie dieses Kontrollkästchen markieren, werden beispielsweise alle *MsgBox*-Anweisungen in Ereignisprotokollen für die Applikation umgewandelt.

Wenn Sie von Ihrer Komponente aus eine *MsgBox*-Anweisung ausführen, reagiert der aktuelle Thread nicht mehr, was in einer innerhalb der Betriebsumgebung ausgeführten COM+-Applikation eine Katastrophe sein kann. Sie sollten also vorsichtshalber die Option *Unbeaufsichtigte Ausführung* verwenden. Dies gilt besonders für die *Builds* Ihres Servers, die in Betrieb genommen werden. Falls Sie während des Debuggens Nachrichtendialogfelder anzeigen lassen möchten, sollten Sie die Debug-Versionen ohne diese Option kompilieren.

Wenn Sie das Kontrollkästchen *Unbeaufsichtigte Ausführung* markieren, steht auch das Kontrollkästchen *Im Speicher erhalten* zur Verfügung. Durch Markieren dieses Kontrollkästchens verhindern Sie, dass die Hosting-Applikation den Code Ihrer DLL aus dem Speicher entfernt. Die DLL bleibt während des gesamten Hostings geladen. Diese Option empfiehlt sich für Server, die in Middle-Tier-Umgebungen wie COM+ und Microsoft Internet Information Server (IIS) eingesetzt werden. Schließlich möchten Sie, dass der Code so selten wie möglich in eine Applikation geladen werden muss. Wenn die DLL nicht aus dem Speicher entfernt werden kann, muss sie später nicht erneut geladen werden. Wenn Sie diese Option verwenden, können Sie davon ausgehen, dass die Hosting-Applikation den Code aus Ihrem Server verwendet, solange sie ausgeführt wird. In einer COM+-Anwendung ist das normalerweise eine sichere Annahme.

Server-Design

Im ersten Schritt der Designphase sollten Sie planen, welche Client-Typen Sie unterstützen wollen. Sie sollten die potentiellen Clients Ihrer Komponenten in zwei Kategorien einteilen: Clients, die eine direkte *vTable*-Bindung verwenden, und Skripting-Clients, die die späte Bindung verwenden. Tabelle 4.1 zeigt die Client-Typen, die durch einige der gebräuchlicheren Sprachen erzeugt werden.

Sprachen, die Clients mit direkter vTable-Bindung erzeugen	Sprachen, die spät gebundene Clients erzeugen
Visual Basic 6	VBScript
Visual Basic 5	JavaScript
Visual Basic 4	Visual Basic 3
C++	
Delphi	

Tabelle 4.1: Client-Typen, die durch verschiedene Sprachen erzeugt werden.

Wenn Sie sicher sein können, dass Ihre Komponente nur von Clients einer Kategorie genutzt wird, wird Ihr Leben sehr viel einfacher. Wenn Ihre Komponenten ausschließlich von Clients verwendet werden, die eine direkte vTable-Bindung verwenden, haben sie mehr Möglichkeiten: Sie können Komponenten für benutzerdefinierte Schnittstellen, Auflistungen und UDTs entwickeln. Sie sollten diese Features nicht nutzen, wenn Sie Komponenten für Skripting-Clients erzeugen.

Clients, die eine direkte vTable-Bindung verwenden, beziehen sich auf die Informationen in der Typbibliothek Ihres Servers. Skripting-Clients verwenden keine Typbibliotheken – sie verwenden ProgIDs und die späte Bindung. Wir werden uns zunächst mit den Clients beschäftigen, die Typbibliotheken verwenden, und uns später mit den Design-Aspekten von Skripting Clients beschäftigen.

Beim Entwurf eines Servers sollten Sie zunächst darüber nachdenken, was in der Typbibliothek des Servers veröffentlicht werden soll. Die Information in der Typbibliothek ist das, was die Außenwelt sieht. Gleichzeitig handelt es sich dabei um die grundlegende Methode, wie Ihre Komponenten den Clients Funktionalität zur Verfügung stellen.

Wir beginnen mit der Betrachtung von High-Level-Attributen für eine Typbibliothek. Anschließend beschäftigen wir uns mit Attributen für Komponenten und Methoden. Außerdem erfahren Sie, wie Sie benutzerdefinierte Schnittstellen, Auflistungen und UDTs in Ihren Typbibliotheken veröffentlichen und sie in Methodendefinitionen nutzen.

Projektübergreifende Typbibliothek-Attribute

Wenn Sie ein neues Server-Projekt anlegen, sollten Sie sofort das Dialogfeld *Projekteigenschaften* öffnen und den Projektnamen und die Projektbeschreibung anpassen, wie in Abbildung 4.2 gezeigt. Diese wichtigen Informationen bilden die High-Level-Attribute Ihrer Typbibliothek. Der Projektname legt einen Namensraum für die darin enthaltenen Komponenten an. Client-Applikationen, die auf die Typbibliothek verweisen, können über einen vollständigen Namen wie beispielsweise `DogServer.CBeagle` auf Komponenten in Ihrem Server zugreifen. Beachten Sie, dass der Projektname immer der erste Teil der ProgID für alle Komponenten im Server-Projekt ist.

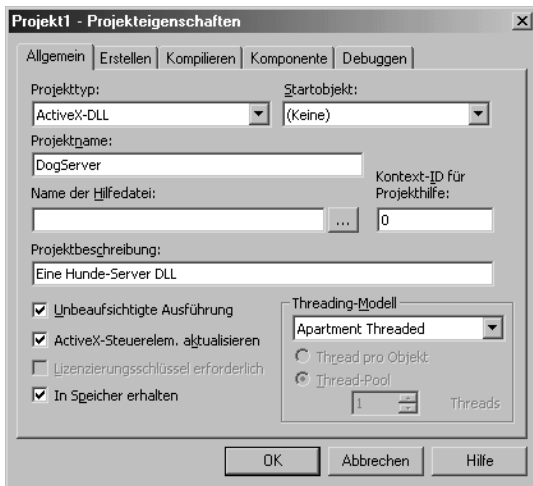


Abbildung 4.2: Im Dialogfeld Projekteigenschaften weisen Sie Ihrem Server einen Projektnamen und eine Projektbeschreibung zu. Diese Einstellungen bilden die Top-Level-Attribute der Typbibliothek Ihres Servers.

Die Projektbeschreibung wird als benutzerfreundliche Beschreibung der eigentlichen Typbibliothek verwendet. Nachdem Ihr Server auf den Workstations anderer Entwickler registriert wurde, sehen diese Entwickler die Beschreibung, wenn sie ihren Projekten mit Hilfe des Dialogfelds *Verweise* Ihren Server hinzufügen. Wenn Sie keine Beschreibung bereitstellen, verwendet Visual Basic den Projektnamen als Beschreibung.

Wenn Sie clientseitigen Code schreiben, sollten Sie statt des Kurznamens `CBeagle` den vollständigen Namen `DogServer.CBeagle` verwenden. Meistens führt die Verwendung des Kurznamens nicht zu Problemen, aber manchmal kann sie Schwierigkeiten verursachen. Beispielsweise führt die Verwendung des Kurznamens zu Mehrdeutigkeiten, wenn das Visual Basic-Projekt Referenzen auf zwei verschiedene Typbibliotheken für eine Client-Applikation verwendet und jede davon die Komponente `CBeagle` enthält. Visual Basic löst diese Mehrdeutigkeit auf, indem es einfach die `CBeagle`-Komponente aus der zuerst angegebenen Typbibliothek verwendet. Wie Sie sich vorstellen können, führt dieser Ansatz zu unerwünschten Ergebnissen. Während Sie zwar im Dialogfeld *Verweise* die Priorität der Typbibliotheken ändern können, auf die verwiesen wird, stellt die Verwendung eines vollständigen Namens eine viel bessere Lösung für das Problem dar.

Die Komponenteneigenschaft *Instancing*

Die *Instancing*-Eigenschaft für Klassenmodule bestimmt, was in der Typbibliothek Ihres Servers implementiert wird. Sie bestimmt, welche Co-Klassen und Schnittstellen für die Außenwelt sichtbar sind. Der Eigenschaft können die folgenden Werte zugewiesen werden, wobei die Reihenfolge angibt, wie wichtig die einzelnen Einstellungen für den COM++-Programmierer sind:

- **MultiUse:** Das Klassenmodul erzeugt eine erstellbare Co-Klasse und eine duale Schnittstelle. Diese Schnittstelle wird aus den öffentlichen Elementen erzeugt und als Standard für die Co-Klasse markiert. Visual Basic erzeugt eine CLSID und eine IID für das Klassenmodul. Die Klasse wird zu einer Komponente, die die Clients nutzen können, um Objekte über den SCM zu instanzieren.
- **PublicNotCreatable:** Das Klassenmodul erzeugt eine nicht erstellbare Co-Klasse und eine duale Schnittstelle. Diese Schnittstelle wird aus den öffentlichen Elementen erzeugt und als Standard für die Co-Klasse markiert. Visual Basic erzeugt eine CLSID und eine IID für das

Klassenmodul. Die Klasse ist keine echte Komponente, weil die Clients sie nicht benutzen können, um Objekte über den SCM zu instanzieren.

- **Private:** Das Klassenmodul hat keinen Einfluss darauf, was in der Typbibliothek veröffentlicht wird. Es hat keine CLSID und keine IID. Die Klasse kann nur von Client-Code im lokalen Projekt verwendet werden. Der SCM wird nicht beteiligt, wenn Objekte für die Klasse instanziiert werden.
- **GlobalMultiUse:** Wie *MultiUse*. Erlaubt den Clients jedoch auch den Zugriff auf die Methoden der Klasse, als handelte es sich dabei um globale Funktionen.
- **SingleUse:** (Nur für prozessexterne Server.) Diese Einstellung ist wie *MultiUse*, aber der Server garantiert dabei, dass jedes Objekt, das aus der Klasse instanziiert wird, in seinem eigenen Prozess bleibt. Das bedeutet, es gibt für jede Instanz dieser Klasse einen separaten ActiveX-EXE-Server-Prozess.
- **GlobalSingleUse:** Wie *SingleUse*. Erlaubt den Clients jedoch auch den Zugriff auf Methoden der Klasse, als handelte es sich dabei um globale Funktionen.

Wenn Sie ActiveX-DLLs für eine COM+-Applikation erzeugen, ist die gebräuchlichste Einstellung für die *Instancing*-Eigenschaft eines Klassenmoduls *MultiUse*. Wahrscheinlich werden Sie nie eine andere Einstellung verwenden. *MultiUse* ist die Standardeinstellung beim Erstellen neuer Klassenmodule. Jede *MultiUse*-Klasse, die Sie in eine ActiveX-DLL kompilieren, kann als COM+-Komponente konfiguriert werden.

Die andere *Instancing*-Einstellung, die Sie möglicherweise verwenden, ist *PublicNotCreatable*. *PublicNotCreatable*-Klassenmodule werden verwendet, wenn eine benutzerdefinierte Schnittstelle erzeugt werden soll. (Kapitel 2 hat erklärt, wie man eine in einem *PublicNotCreatable*-Klassenmodul definierte Schnittstelle erzeugt und implementiert.) Ein *PublicNotCreatable*-Klassenmodul hat eine IID, die die Schnittstelle in der Typbibliothek identifiziert. Anders als bei *MultiUse*-Klassen integriert Visual Basic jedoch keine Unterstützung für die Aktivierung durch den SCM. Client-Applikationen können aus einer *PublicNotCreatable*-Klasse keine Objekte erzeugen.

Wenn Sie mit Visual Basic Server erstellen, die nicht als konfigurierte Komponenten in der COM+-Laufzeitumgebung eingesetzt werden sollen, gibt es noch weitere Gründe, warum Sie ein *PublicNotCreatable*-Klassenmodul verwenden sollten. Beispielsweise können *PublicNotCreatable*-Klassen verwendet werden, um Objekte aus dem Client-Code innerhalb desselben Projekts zu instanzieren. Beachten Sie, dass der Client-Code dazu den Operator *New* verwenden muss. Die Funktion *CreateObject* kann hier nicht eingesetzt werden, weil ein *PublicNotCreatable*-Klassenmodul keine Registrierungseinträge für seine CLSID erhält und auch keine integrierte Unterstützung der Aktivierung besitzt. Wenn Sie den Operator *New* ausführen, instanziiert die Visual Basic-Laufzeitumgebung das Objekt ohne Hilfe des SCM.

Das Anlegen von Objekten aus *PublicNotCreatable*-Klassen kann in COM+ problematisch sein, weil das neue Objekt nicht in einem gültigen Kontext initialisiert wird. Das bedeutet, das Objekt kann keinen der in der COM+-Laufzeitumgebung angebotenen Dienste nutzen. Damit eine Komponente diese COM+-Dienste nutzen kann, müssen ihre Objekte über den SCM aktiviert werden. Mehr darüber erfahren Sie in Kapitel 6. Hier sollten Sie sich nur merken, dass Objekte, die aus *PublicNotCreatable*-Klassen erzeugt wurden, die Hilfestellungen von COM+ nicht nutzen können.

Private Klassen weisen dieselben Nachteile wie *PublicNotCreatable*-Klassen auf, wenn sie in einer COM+-Applikation eingesetzt werden. *Private* Klassen unterstützen keine Aktivierung durch den SCM. Außerdem können sie keine Schnittstelle definieren und sie haben keine IID in der Typbibliothek. Sie sind für alle Clients unsichtbar, die die Typbibliothek einsehen. Der einzige Grund, warum *Private*-Klassen verwendet werden, ist ein Objekt im Hintergrund zu erzeugen, das keinen der von COM+ bereitgestellten Dienste nutzen soll.

SingleUse-Klassen bieten keinerlei Nutzen für COM+-Programmierer, weil sie nur in ActiveX-EXE-Projekten eingesetzt werden können. In COM+ können Sie nur Komponenten verwenden, die in ActiveX-DLLs kompiliert wurden. *SingleUse*-Klassen erfüllen jedoch einen Zweck. Wenn Sie beispielsweise einen einfachen ActiveX-EXE-Server erzeugen, der auf den Desktops der einzelnen Benutzer ausgeführt werden soll, kann eine *SingleUse*-Klasse als interessante Möglichkeit betrachtet werden. Wenn Sie zwei Objekte aus einer *SingleUse*-Klasse erzeugen, wird jedes Objekt in einem separaten Server-Prozess erzeugt. Wenn Sie zwei Instanzen einer Client-Applikation auf dem Desktop eines Benutzers ausführen und diese denselben Objekttyp aktivieren, können Sie eine *SingleUse*-Klasse benutzen, um Ihren Objekten einen zusätzlichen Fehlerschutz zu bieten. *SingleUse*-Klassen unterstützen außerdem eine einfache Methode, das Multi-Threading zu erzielen, weil jeder Prozess einen eigenen Thread erzeugt. Auch diese Technik sollten Sie überlegt und nur für eine sehr begrenzte Anzahl von Clients einsetzen.

MultiUse und *SingleUse* erlauben Ihnen, eine »globale« Co-Klasse zu erstellen. Die Klasse kann auf diese Weise einfacher vom Client verwendet werden. Wenn Sie eine *MultiUse*-Standardklasse besitzen, die beispielsweise einen Steuersatz berechnet, greifen Sie wie folgt auf das Objekt zu:

```
Dim Obj As CTaxCalculator, Tax As Currency
Set Obj = New CTaxCalculator
Steuer = Obj.CalculateTax(28600)
```

Wenn Sie die *Instancing*-Eigenschaft der Klasse auf *GlobalMultiUse* setzen, kann der Client so auf die Methode zugreifen, als handelte es sich dabei um eine globale Funktion, wie im folgenden gezeigt:

```
Dim Tax As Currency
Tax = CalculateTax(28600)
```

Globale Klassen ersparen dem Client das Instanzieren eines Objekts vor dem Aufruf einer Methode. Visual Basic stellt dieses Leistungsmerkmal bereit, indem es die Co-Klasse in der Typbibliothek mit dem Attribut *AppObject* markiert. Auf der Client-Seite erstellt Visual Basic ein unsichtbares Objekt im Hintergrund, sobald dieses benötigt wird, und ordnet alle globalen Aufrufe den Methodenaufrufen für dieses unsichtbare Objekt zu. Visual Basic ist momentan das einzige Entwicklungswerkzeug, das die Bequemlichkeit der transparenten Zuordnung von *AppObject*-Attributen auf der Client-Seite anbietet. Ein Programmierer, der eine Client-Applikation mit C++ oder Java erstellt, kennt keinen Unterschied zwischen einer *GlobalMultiUse*-Klasse und einer *MultiUse*-Klasse. Globale Klassen stellen lediglich einen syntaktischen Anreiz für Visual Basic-Programmierer dar.

Bei Entwürfen mit *GlobalMultiUse*-Klassen sollten Sie einige Aspekte berücksichtigen. Erstens funktionieren sie nur dann korrekt, wenn sich der Client-Code, der sie benutzt, in einem anderen Projekt befindet. Eine Klasse kann die Vorteile der globalen Aspekte einer *GlobalMultiUse*-Klasse nicht nutzen, wenn sich beide Klassen im selben Projekt befinden. Darüber hinaus haben Sie nicht viel Kontrolle darüber, wann das unsichtbare Objekt erzeugt oder zerstört wird. Es wird bei der ersten Benutzung erzeugt und zerstört, nachdem die Client-Applikation beendet wird. Das kann in ineffizientem Code resultieren, wenn ein *GlobalMultiUse*-Objekt wertvolle Ressourcen belegt.

Häufig ist es sehr viel sinnvoller, Objekte aus Standard-*MultiUse*-Klassen zu instanzieren und die Referenzen explizit auf *Nothing* zu setzen, wenn man sie freigeben will.

Ändern von Prozedurattributen

Mit dem Befehl *Prozedurattribute* im Menü *Extras* legen Sie fest, wie Visual Basic Attribute für die Methoden und Eigenschaften Ihrer Klasse setzen soll, während der Code für das zu ändernde Klassenmodul sich im aktiven Fenster befindet. Abbildung 4.3 zeigt das Dialogfeld.

Mit Hilfe der Einstellung *Beschreibung* dokumentieren Sie die Semantik der Methode oder Eigenschaft. Andere Visual Basic-Programmierer sehen diese Beschreibungen, wenn sie die Typbibliothek Ihres Servers im Objektkatalog betrachten. Es ist ganz einfach, Beschreibungen bereitzustellen, und Ihr Server erhält dadurch ein professionelleres Aussehen.

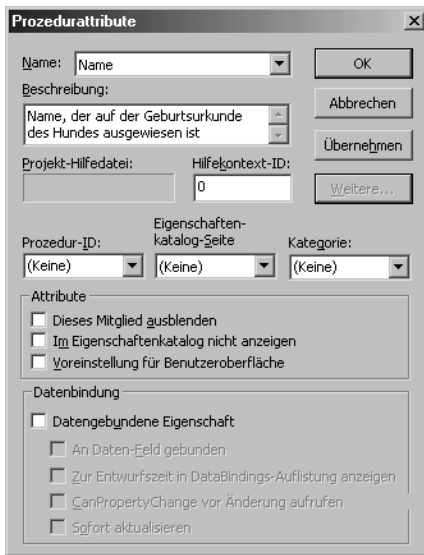


Abbildung 4.3: Im Dialogfeld *Prozedurattribut* ändern Sie die Attribute der Methoden und Eigenschaften eines öffentlichen Klassenmoduls. Alle Einstellungen, die Sie in diesem Dialogfeld vornehmen, werden in der Schnittstellendefinition in der Typbibliothek des Servers festgehalten.

Sie können auch die Prozedur-ID auf einen anderen Wert setzen. Wenn Sie sie beispielsweise eine Eigenschaft auf *Voreinstellung* setzen (das ist der Wert 0), wird die Eigenschaft als Standard-eigenschaft des Objekts erkannt. Und so sieht die resultierende IDL aus:

```
[
    id(00000000),
    propput,
    helpstring("Name, der auf der Gebursturkunde des Hundes ausgewiesen ist")
]
HRESULT Name([in] BSTR Name);
```

Jetzt kann ein Client sowohl explizit als auch implizit auf diese Eigenschaft zugreifen:

```
Dim Dog1 As IDog, Dog2 As IDog
Set Dog1 = New CRetriever
Set Dog2 = New CRetriever
' Der Zugriff auf die Name-Eigenschaft ist explizit möglich
Dog1.Name = "Milli"
' Der Zugriff auf die Name-Eigenschaft ist auch implizit möglich
Dog2 = "Vanilli"
```

Wie Sie sehen, können Sie im Dialogfeld *Prozedurattribute* verschiedene Optionen festlegen. Sie können auch Methoden vor anderen Programmierern verbergen. Kurz gesagt, können Sie Ihren Server verbessern, indem Sie bestimmte Methodenattribute einfügen. Beachten Sie, dass viele der Einstellungen nur für Programmierer relevant sind, die ActiveX-Steuerelemente entwickeln. Viele Optionen bieten keine Vorteile für Programmierer, die nichtvisuelle Komponenten für die mittlere Schicht entwickeln.

Friend-Methoden

Wenn Sie eine Methode in einer Visual Basic-*MultiUse*-Klasse als `friend` markieren, hat dies einen interessanten Effekt. Eine Friend-Methode kann von jedem Code im Server-Projekt aufgerufen werden, steht der Außenwelt jedoch nicht zur Verfügung. Sie ist gewissermaßen innerhalb des Servers öffentlich und außerhalb des Servers privat. In einer öffentlichen Klasse bieten also private Elemente einen modulweiten Zugriff an, während Friend-Elemente einen projektweiten Zugriff zulassen. Öffentliche Elemente stehen jedem Client innerhalb oder außerhalb des Servers zur Verfügung. Wenn Sie eine Methode oder eine Eigenschaft als `friend` deklarieren, wird diese nicht in die Typbibliothek übernommen. Sie hat somit nichts mit COM zu tun.

Verwenden Sie Friend-Elemente jedoch in Visual Basic mit Vorsicht. Viele Programmierer nutzen die Friend-Syntax, ohne zu wissen, dass sie damit das Konzept der Kapselung verletzen. Später, wenn sie eine Friend-Methode löschen oder ändern, müssen sie letztlich den gesamten Client-Code ändern, der Abhängigkeiten von der Methode aufweist.

Der zweite (und wichtigere) Grund, Friend-Methoden zu vermeiden, ist, dass sie in der Typbibliothek nicht veröffentlicht werden. Das ist problematisch, weil die COM-Laufzeitumgebung die Schnittstellendefinitionen der Typbibliothek verwendet, um Proxies und Stubs zu erzeugen. Sie sollten nur dann Friend-Methoden verwenden, wenn Sie absolut sicher sind, dass Sie nie einen Proxy oder einen Stub zwischen dem Client und dem Objekt brauchen. Wie Sie in Kapitel 6 erfahren, werden Objekte, die aus in einer COM+-Applikation konfigurierten Komponenten erstellt werden, über einen einfachen Proxy mit ihrem Client verbunden. Und genau deshalb sollten Sie Friend-Methoden vermeiden, wenn Sie konfigurierte Komponenten erstellen.

Enumerationen

Als COM-Entwickler sollten Sie in Ihren Entwürfen so viele Enumerationen wie möglich verwenden. Enumerationen erlauben Ihnen die Deklaration logischer Konstantengruppen in Ihrer Typbibliothek. Nachdem diese Enumerationen in Ihrer Typbibliothek definiert wurden, können Sie sie für Methodenparameter- und Rückgabewerttypen verwenden. Die Definition einer Enumeration in einem Visual Basic-Klassenmodul könnte wie folgt aussehen:

```
Enum FetchItemEnum
    dsSlippers
    dsNewspaper
    dsBeer
End Enum
```

Jetzt wollen wir diese Auflistung nutzen, indem wir damit einen Methodenparameter definieren. Wenn Sie etwa die folgende Methode

```
Sub Fetch(ByVal Item As FetchItemEnum)
    ' Stellen Sie sich hier Ihre Implementierung vor
End Sub
```


definieren, kann ein Client die Methode aufrufen, indem er einen der Enumerationswerte wie folgt übergibt:

```
Dim Spot As CRetriever
Set Spot = New CRetriever
Spot.Fecht dsBeer
```

Enumerationen machen den Client-Code leichter lesbar. Wenn Sie eine Enumeration für einen Parametertyp verwenden, stellt das in Visual Basic integrierte IntelliSense den clientseitigen Programmieren eine Dropdown-Liste der erhältlichen Auswahlmöglichkeiten zur Verfügung. Damit wird die Programmierung Ihrer Komponenten sehr bequem.

Sie müssen Enumerationen in einem öffentlichen Klassenmodul deklarieren, um sie in der Typbibliothek des Servers zu veröffentlichen. Wenn Sie ein ActiveX DLL-Projekt erzeugen, weisen Sie einem öffentlichen Klassenmodul am besten die Einstellung *MultiUse* oder *PublicNotCreatable* zu. Auflistungen, die in .BAS-Modulen deklariert sind, sind privat für das Projekt und können nicht in öffentlichen Methodendefinitionen verwendet werden.

Die Tatsache, dass die von Ihnen in einem Klassenmodul deklarierten Auflistungen nur für die Typbibliothek und nicht für die Klasse gelten, ist nicht besonders leicht zu verstehen. Die *Instancing*-Eigenschaft der Klasse hat keinen Einfluss auf den Gültigkeitsbereich der Enumeration. Wenn Sie ein separates Klassenmodul erstellen möchten, das lediglich Ihre Enumeration aufnehmen soll, markieren Sie es als *PublicNotCreatable*. Sie können Enumerationen auch einfach in einer *MultiUse*-Klasse deklarieren, die eine Komponente definiert. Beachten Sie jedoch, dass alle Enumerationswerte in Ihrem Projekt Teil eines großen Namensraums sind. Jeder von Ihnen deklarierte Enumerationswert sollte deshalb einen eindeutigen Namen haben.

Hier gilt es einen weiteren Aspekt zu beachten: Wenn eine Client-Applikation mehrere Typbibliotheken verwendet, die jeweils eigene Enumerationen definieren, sind alle kombinierten Enumerationen von allen Servern für die Client-Applikation in einem einzigen, projektübergreifenden Gültigkeitsbereich sichtbar. Die Enumerationen von zwei Servern geraten in Konflikt, wenn sie denselben Namen tragen. Üblicherweise verwendet man serverspezifische Präfixe, um solche Konflikte zu vermeiden. Die Auflistungen im obigen Beispiel wurden mit dem Präfix *ds* angelegt, um dieser Konvention zu genügen. Es besteht jedoch weiterhin ein möglicher Konflikt, den Sie in der Client-Applikation auflösen, indem Sie dem Namen der Auflistung Ihren Projektnamen wie folgt voranstellen:

```
DogServer.dsBeer
```

UDTs verwenden

Vor Visual Basic 6 konnten Sie nur benutzerdefinierte Typen (UDTs, User-Defined Types) definieren, die für ein Projekt privat waren. Seit Visual Basic 6 können Sie COM-Methoden definieren, die einen UDT als Parametertyp oder Rückgabewerttyp verwenden. (Das äquivalente Konstrukt zu einem UDT ist in C und C++ eine Struktur, die mit `structure` oder `struct` angegeben wird.)

Bei der Verwendung von UDTs in COM-Methoden stehen Sie einigen wesentlichen Einschränkungen gegenüber. Erstens müssen Sie alle UDT-Parameter als *ByRef* und nicht als *ByVal* deklarieren. Zweitens müssen Sie eine Version des Universal-Marshalers verwenden, die einen UDT in einen Variant und umgekehrt umwandeln kann. Wenn Sie keine aktuelle Version besitzen, generiert der Universal-Marshaler einen Fehler, wenn er versucht, einen Proxy oder einen Stub für eine IID zu erstellen, die UDTs enthält. Windows 2000- und Windows 98-Computer besitzen die aktuelle Version des Universal-Marshalers. Wenn Sie Windows NT Server einsetzen, müssen Sie

das Service Pack 4 oder eine aktuellere Version installieren. Wenn Sie Windows 95 verwenden, brauchen Sie eine aktuelle Version des DCOM-Upgrades.

Wie Enumerationen müssen auch UDTs in öffentlichen Klassenmodulen deklariert werden, um in der Typbibliothek des Servers veröffentlicht werden zu können. Hier ein einfaches Beispiel für eine UDT-Definition:

```
Type DogInfo
    Name As String
    Rank As String
    SerialNumber As String
End Type
```

Nachdem Sie den UDT definiert haben, können Sie ihn in Methodensignaturen als Parameter oder als Rückgabetyt verwenden, etwa wie folgt:

```
Sub SetDogInfo(ByRef Data As DogInfo)
    'Implementierung
End Sub
Function GetDogInfo() As DogInfo
    'Implementierung
End Function
```

Hier ein Beispiel für einen Client, der die Methode `SetDogInfo` aufruft und einen UDT als Parameter übergibt:

```
Dim Spot As CRetriever
Set Spot = New CRetriever
' UDT-Instanz erzeugen und füllen
Dim SpotInfo As DogInfo
SpotInfo.Name = "Spot"
SpotInfo.Rank = "Sergeant"
SpotInfo.SerialNumber = "K9"
' UDT-Instanz als Parameter übergeben
Spot.SetDogInfo SpotInfo
```

UDTs sind für die unterschiedlichsten Situationen geeignet. In Kapitel 11, das sich genauer mit Marshaling und Schnittstellenentwurf beschäftigt, werden Sie mehrere Techniken kennen lernen, um Daten mit Hilfe von UDTs, Arrays, Property Bags und sogar ADO-Datensatzgruppen effizient über das Netzwerk zu übertragen. Hier sollen Sie nur die grundlegenden COM-Datentypen verstehen, die Sie in einer Typbibliothek definieren können. Bisher haben Sie in diesem Kapitel erfahren, dass Visual Basic Typinformationen für Co-Klassen, Schnittstellen, Enumerationen und UDTs veröffentlicht.

Beachten Sie, dass einige Clients die Typbibliothek Ihres Servers nutzen können, andere dagegen nicht. Wenn Sie davon ausgehen können, dass alle Clients Ihrer Komponente in Visual Basic oder C++ geschrieben sind, können Sie alle Datentypen verwenden, die wir bisher in diesem Kapitel vorgestellt haben, auch Enumerationen und UDTs. Gleichzeitig bedeutet es, dass Sie für Ihre Komponenten und Applikationen abstrakte und konkrete Klassen entwerfen können. Wenn Sie die von benutzerdefinierten Schnittstellen bereitgestellten Entwurfsmerkmale nutzen wollen, ist das in Ordnung. Wenn Sie dagegen die zusätzliche Arbeit und Komplexität vermeiden wollen, die benutzerdefinierte Schnittstellen mit sich bringen, können Sie auch für die (automatisch angelegte) Standardschnittstelle einer Komponente programmieren. Das bleibt ganz Ihnen überlassen.

Komponenten für Skripting-Clients entwerfen

Bisher haben wir uns auf Clients konzentriert, die die aus Typbibliotheken ermittelten Typinformationen nutzen. Diese Clients wissen, wie man auf benutzerdefinierte vTables zugreift und werden zusammen mit CLSIDs und IIDs kompiliert. Skripting-Clients sind ganz anders. Sie wissen nichts von Ihrer Typbibliothek und brauchen diese auch nicht. Sie kennen nur Ihre ProgIDs und die Namen Ihrer Methoden und Eigenschaften.

Der Entwurf von Komponenten für Skripting-Clients bedingt einige Einschränkungen, durch die verhindert wird, dass Sie einige der zuvor in diesem Kapitel beschriebenen Techniken verwenden. Sie sollten keine UDTs einplanen, weil die meisten Skriptingsprachen keine Syntax für den Zugriff auf UDT-Elemente bieten. Sie können Methoden mit Hilfe von Enumerationen definieren, weil der Skripting-Client den Enumerationstyp als Integer betrachtet. Enumerationen sind jedoch dann nicht mehr praktisch, wenn der Client ihre Definitionen nicht aus einer Typbibliothek entnehmen kann.

Wenn Sie eine Methode entwerfen, die ByRef-Parameter verwendet, sollten Sie jeden dieser Parameter als Variant definieren, statt einen präziseren Typ zu verwenden, weil die meisten Skripting-Hosts keine Methoden mit streng typisierten Ausgabeparametern verarbeiten können. Beispielsweise kommt VBScript nur mit ByRef-Parametern vom Typ Variant zurecht. Ein VBScript-Client erzeugt den Fehler »Datentypen unverträglich«, wenn er versucht, eine Methode mit einem ByRef-Parameter aufzurufen, der als Integer, Double oder String deklariert wurde.

Skripting-Clients und benutzerdefinierte Schnittstellen

Viele COM-Programmierer haben angestrengt daran gearbeitet, die Konzepte und Techniken der schnittstellenbasierten Programmierung zu erlernen. Benutzerdefinierte Schnittstellen bedeuten beim Entwurf und bei der Implementierung von Applikationen einen zusätzlichen Aufwand, aber insbesondere für größere Projekte liegen die Vorteile auf der Hand.

Wenn Sie Ihre Schnittstellen separat von Ihren erstellbaren Komponenten definieren, erzielen Sie eine bessere Wiederverwendbarkeit, Wartbarkeit und Erweiterbarkeit. Basierend auf plug-kompatiblen Komponenten können Sie polymorphe Entwürfe realisieren. Ihre Client-Applikationen können beliebige Versionen einer Komponente nutzen, indem zur Laufzeit geprüft wird, ob ein Objekt eine bestimmte Schnittstelle unterstützt. Außerdem sind Sie damit in der Lage, ein robustes Versionsverwaltungsschema zu erzeugen, wodurch es möglich wird, eine Komponente oder eine Client-Applikation zu aktualisieren, ohne dass andere, bereits in Betrieb befindliche Komponenten oder Client-Applikationen dadurch beeinträchtigt werden.

Nachdem Sie die wichtigsten Konzepte der schnittstellenbasierten Programmierung verstanden haben, wird diese erst so richtig interessant. Viele Visual Basic-Programmierer bevorzugen benutzerdefinierte Schnittstellen und setzen sie ein, wo immer dies möglich ist. Es gibt jedoch ein etwas unglückliches Problem damit: Nachdem Sie eine Komponente erstellt haben, die benutzerdefinierte Schnittstellen implementiert, ist es schwierig oder gar unmöglich, sie direkt von einem Skripting-Client aus zu benutzen. Das ist ein Problem, wenn Sie Komponenten erzeugen, die von Skripting-Clients wie beispielsweise den mit ASP (Active Server Pages) erstellten Web-Sites verwendet werden.

Wo ist das Problem?

Der Grund für dieses Problem ist, dass Skripting-Clients in typlosen Sprachen geschrieben werden. Wenn Sie beispielsweise mit VBScript eine ASP-Seite programmieren, sind alle Ihre Variab-

len und Parameter als *Variants* definiert. Wenn Sie eine Verbindung zu einem COM-Objekt einrichten, können Sie nicht angeben, welche Schnittstelle Sie verwenden wollen. Vielmehr ist ein VBScript-Client immer über die Standardschnittstelle mit einem COM-Objekt verbunden. Außerdem muss die Standardschnittstelle eine *IDispatch*-Schnittstelle oder eine duale Schnittstelle sein. *IDispatch* erlaubt einem Skripting-Client, über die späte Bindung auf das Objekt zuzugreifen.

In COM gelangen Clients von einer Schnittstelle zur anderen, indem sie *QueryInterface* aufrufen. C++-Clients können *QueryInterface* direkt aufrufen. Visual Basic-Clients rufen *QueryInterface* indirekt auf, indem sie Objektreferenzen bestimmte Variablen und Parameter zuweisen, die den gewünschten Schnittstellentyp aufweisen. Skripting-Clients dagegen können *QueryInterface* weder direkt noch indirekt aufrufen. Sie können nicht zu einer zweiten Schnittstelle gelangen. Letztlich sind sie auf die Schnittstelle beschränkt, mit der sie ursprünglich verbunden wurden. Wie Sie sehen, bieten Skripting-Clients nicht die erforderliche Unterstützung, um die Vorteile von Komponenten nutzen zu können, die mehrere Schnittstellen implementieren.

Ein weiteres Problem, das ebenfalls mit Visual Basic zu tun hat, macht das Ganze noch schlimmer. Sie können Visual Basic nicht so konfigurieren, dass es eine benutzerdefinierte Schnittstelle als Standardschnittstelle für eine *MultiUse*-Klasse verwendet. Das gilt für benutzerdefinierte Schnittstellen, die in *PublicNotCreatable*-Klassenmodulen definiert sind, sowie für IDL-Schnittstellen.

Die einzige Methode, die Standardschnittstelle für eine *MultiUse*-Klasse zu definieren, ist es, dem eigentlichen Klassenmodul öffentliche Methoden hinzuzufügen. Angenommen, Sie haben eine duale Schnittstelle *IDog* in IDL definiert, die wie folgt aufgebaut ist:

```
Interface IDog : IDispatch {
    HRESULT Bark();
    HRESULT RollOver([in, out] short* Rolls);
};
```

Nachdem Sie Ihren IDL-Code in eine Typbibliothek kompiliert haben, können Sie diese auf Ihrer Entwicklungs-Workstation registrieren. (Mehr dazu im nächsten Kapitel.) Nachdem Sie die Typbibliothek registriert haben, können Sie in einem ActiveX-DLL-Projekt darauf verweisen und Ihre Schnittstelle in eine *MultiUse*-Klasse namens *CDingo* implementieren, etwa wie folgt:

```
Option Explicit
Implements IDog

Private Sub IDog_Bark()
    ' Ihre Implementierung
End Sub

Private Sub IDog_RollOver(ByRef Rolls As Integer)
    ' Ihre Implementierung
End Sub
```

Auch wenn *IDog* die einzige Schnittstelle ist, die Ihre Klasse unterstützt, ist sie nicht als Standardschnittstelle markiert. In Visual Basic haben Sie keine Möglichkeit, *IDog* als Standardschnittstelle zu definieren. Wenn Sie die Typbibliothek betrachten, die Visual Basic in die DLL einbaut, sehen Sie die folgenden Definitionen, die das Problem verdeutlichen:

```
interface _CDingo : IDispatch {
    // Leer, weil es in CDingo keine öffentlichen Methoden gibt
```

```
};

coclass CDingo {
    [default] interface _CDingo;
    interface IDog;
};
```

Visual Basic definiert immer eine duale Schnittstelle aus den öffentlichen Elementen einer *Multi-Use*-Klasse und markiert sie als Standard – auch wenn die Klasse keine öffentlichen Methoden enthält. Benutzerdefinierte Schnittstellen, die Sie mit dem Schlüsselwort `Implements` implementieren, sind immer sekundäre und nicht Standardschnittstellen. Das bedeutet, ein Skripting-Client kann nicht auf die Methoden in Ihrer Schnittstelle zugreifen.

Was bedeutet es, wenn Sie nicht in der Lage sind, zu einer sekundären Schnittstelle zu navigieren? Wenn Sie Komponenten mit Visual Basic entwickeln, können Ihre Skripting-Clients nur die öffentlichen Methoden Ihrer erstellbaren Klassen verwenden. Sie können Ihre Schnittstellen nicht separat von den Klassen definieren, die Ihre Implementierungen enthalten. Außerdem bedeutet es, dass Sie die Konzepte der schnittstellenbasierten Programmierung nicht wirklich nutzen können, wenn Sie mit Skripting-Clients zu tun haben.

Und nach dieser Schwerstarbeit in der Entwurfs- und Codierphase haben Sie noch ein weiteres unangenehmes Problem: Clients, die `QueryInterface` aufrufen können, können auf die benutzerdefinierten Schnittstellen Ihrer Komponenten zugreifen, nicht aber Skripting-Clients. Was tun Sie in dieser Situation? Wir wollen zunächst überlegen, was Sie nicht tun sollten.

Das Problem durch Programmierung beseitigen

Die erste Technik, die wir hier betrachten, ist für Programmierer gedacht, die Herausforderungen lieben. Skripting-Clients können nicht eigenständig zu einer sekundären Schnittstelle gehen. Sie können sie jedoch dabei unterstützen. Betrachten Sie den folgenden Code in einer *MultiUse*-Klasse und achten Sie auf die neue Methode `GetIDog`:

```
Option Explicit
Implements IDog

Private Sub IDog_Bark()
    ' Ihre Implementierung
End Sub

Private Sub IDog_RollOver(ByRef Rolls As Integer)
    ' Ihre Implementierung
End Sub

' Einsprungpunkt für einen Skripting-Client
Public Function GetIDog() As IDog
    Set GetIDog = Me
End Function
```

`GetIDog` ist eine öffentliche Methode in der Standardschnittstelle und steht damit einem Skripting-Client zur Verfügung. Diese Methode hat den Rückgabebetyp `IDog`, woraus ein impliziter Aufruf von `QueryInterface` resultiert. Durch den Aufruf von `GetIDog` gelangt ein Skripting-Client von der Standardschnittstelle zu einer sekundären Schnittstelle. Hier ein Beispiel für VBScript-Code, der eine Verbindung zu `IDog` einrichtet und die `Bark`-Methode aufruft:

```
Dim ProgID, Ref1, Ref2
ProgID = "DogServer.CDingo"
Set Ref1 = Server.CreateObject(ProgID)
Set Ref2 = Ref1.GetIDog()
Ref2.Bark
Ref2.RollOver 3
Set Ref1 = Nothing
Set Ref2 = Nothing
```

Wie Sie sehen, erlaubt diese Technik einem Skripting-Client, auf eine benutzerdefinierte Schnittstelle wie `IDog` zuzugreifen. Beachten Sie, dass Sie `IDog` als duale Schnittstelle definieren sollten, die sich von `IDispatch` ableitet, im Gegensatz zu einer benutzerdefinierten Schnittstelle, die sich von `IUnknown` ableitet. Eine duale Schnittstelle erlaubt dem Skripting-Client, die späte Bindung zu nutzen, wenn er Methoden wie `Bark` oder `RollOver` aufruft.

Oberflächlich betrachtet, scheint dieser Ansatz eine praktische Lösung darzustellen. Er ermöglicht es Ihnen, eine Applikation mit Skripting-Clients zu entwickeln, die die Konzepte der schnittstellenbasierten Programmierung nutzt. Gleichzeitig weist er jedoch ein ernsthaftes Problem auf: Er funktioniert nicht, wenn Client und Objekt in unterschiedlichen Prozessen ausgeführt werden. Er funktioniert nicht einmal, wenn Client und Objekt in unterschiedlichen Threads innerhalb desselben Prozesses ausgeführt werden.

Das Problem bei diesem Ansatz ist, dass die Remote-Schicht von COM verletzt wird. Jede Verbindung zwischen einem Client und einem Objekt basiert auf einer bestimmten IID. Beispielsweise sollte es ein Proxy-/Stub-Paar für die Standardschnittstelle hinter `CDingo` und ein weiteres Proxy-/Stub-Paar für `IDog` geben. In einigen Situationen wird das zweite Proxy-/Stub-Paar für `IDog` niemals erstellt.

Wenn ein Skripting-Client die Methode `GetIDog` für ein Remote-Objekt aufruft, funktioniert das Ganze nicht mehr korrekt, weil die Remote-Schicht von COM versucht, eine Optimierung vorzunehmen. Sie sieht, dass bereits eine `IDispatch`-Verbindung vorhanden ist, und versucht somit nicht, eine zweite aufzubauen. Auch wenn jede duale Schnittstelle eine andere IID hat, fordert der typlose Skripting-Client immer `IDispatch` an. Ein Aufruf von `GetIDog` gibt damit einen redundanten Verweis auf die Standardschnittstelle zurück. Der Skripting-Client navigiert nicht wie gewünscht von einer Schnittstelle zu einer anderen. Wenn der Skripting-Client versucht, `Bark` aufzurufen, schlägt dieser Aufruf fehl, weil die Standardschnittstelle hinter `CDingo` die Methode `Bark` nicht unterstützt.

Möglicherweise ist diese Technik also gar nicht so gut. Sie funktioniert, wenn der Client und das Objekt keine Proxy-/Stub-Schicht zwischen sich benötigen, für alle anderen Fälle jedoch ist sie nicht geeignet. Wann könnte Ihnen dies Probleme bereiten? Angenommen, Sie erstellen eine Komponente und einen ASP-Skripting-Client, der diese Technik nutzt. Sie installieren die Komponente in einer COM+-Bibliotheksapplikation auf derselben Maschine, auf der sich auch der ASP-Code befindet. Das Ganze funktioniert zunächst wunderbar, aber was passiert, wenn der Systemadministrator Ihre Komponente neu konfiguriert, so dass sie aus Gründen der Sicherheit oder Fehlertoleranz in einer COM+-Server-Applikation ausgeführt wird? Ihr Code funktioniert nicht mehr, weil Sie eine Abhängigkeit erzeugt haben, die bei der Neukonfiguration der Komponente zerstört wird. Und genau diese Abhängigkeit wollen sie vermeiden.

Implementieren Sie `IDispatch` nicht mehrfach

Skripting-Clients kennen nur eine Schnittstelle, `IDispatch`. Wenn Sie zwei duale Schnittstellen in einer Komponente implementieren, führt das zu einer Mehrdeutigkeit, weil Sie `IDispatch` mehrfach implementieren. Wie Sie gesehen haben, kann diese Mehrdeutigkeit problematisch sein. Aus

der Design-Perspektive betrachtet, sollte eine Komponente nur eine einzige `IDispatch`-Implementierung aufweisen. Diese Regel wurde von den COM-Entwicklern aufgestellt, als sie ihre Remote-Schicht entwarfen.

Sie wissen, dass Visual Basic-Komponenten die Standardschnittstelle immer als duale Schnittstelle bereitstellen. Dies ist die Schnittstelle, die aus den öffentlichen Methoden der Klasse erstellt wird. Das bedeutet, eine Visual Basic-*MultiUse*-Klasse besitzt von Haus aus eine `IDispatch`-Implementierung, noch bevor Sie irgendwelche sekundären Schnittstellen implementieren. Alle sekundären benutzerdefinierten Schnittstellen, die Sie implementieren, sollten nicht von `IDispatch` sondern von `IUnknown` abgeleitet werden. Die Befolgung dieser Regel stellt sicher, dass Ihre Komponenten nicht mehrfache Implementierungen von `IDispatch` aufweisen. (Mehr über die Definition einer solchen Schnittstelle mit Hilfe von IDL erfahren Sie in Kapitel 5.)

Wir haben aber das Problem noch nicht gelöst. Wenn Sie eine Komponente haben, die benutzerdefinierte Schnittstellen implementiert, wie können Sie dann von einem Skripting-Client aus auf diese Schnittstellen zugreifen? Sie machen aus allen benutzerdefinierten Schnittstellen eine einzige Schnittstelle im Stil von `IDispatch`. Diese Schnittstelle ist eine Obermenge aller Methoden in allen anderen Schnittstellen. Sie implementieren sie unter Verwendung öffentlicher Methoden in einer *MultiUse*-Klasse.

Jetzt wollen wir unser Beispiel erweitern. Angenommen, Sie haben eine *MultiUse*-Klasse, die zwei unterschiedliche benutzerdefinierte Schnittstellen definiert, `IDog` und `IWonderDog`. Weiterhin nehmen wir an, diese beiden Schnittstellen leiten sich statt von `IDispatch` von `IUnknown` ab. Betrachten Sie die Definition der *MultiUse*-Klasse in Listing 4.1: Die öffentlichen Methoden im *MultiUse*-Klassenmodul dienen als Einsprungpunkt für einen Skripting-Client. Für alle Methoden jeder Schnittstelle gibt es eine entsprechende öffentliche Methode, die vom Skripting-Client aufgerufen werden kann. Die öffentliche Methode gibt den Aufruf einfach an die Schnittstellenmethode weiter. Diese Unterstützung von Skripting-Clients kann etwas müßig sein, aber sie löst unser Problem. Ein Skripting-Client kann auf diese Weise alle Methoden aufrufen, die die Komponente implementiert.

```
Implements IDog
Implements IWonderDog

Private Sub IDog_Bark()
    ' Ihre Implementierung
End Sub

Private Sub IDog_RollOver(ByRef Rolls As Integer)
    ' Ihre Implementierung
End Sub

Private Sub IWonderDog_FetchSlippers()
    ' Ihre Implementierung
End Sub

' Skripting-Unterstützung, die der Standardschnittstelle hinzugefügt wurde
Public Sub Bark()
    Call IDog_Bark ' Aufruf weiterleiten.
End Sub

Public Sub RollOver(ByRef Rolls As Variant)
    Dim iRolls As Integer
    iRolls = Rolls
```

```

    Call IDog_RollOver(iRolls) ' Aufruf weiterleiten.
    Rolls = iRolls
End Sub

Public Sub FetchSlippers()
    Call IWonderDog_FetchSlippers ' Aufruf weiterleiten.
End Sub

```

Listing 4.1: Sie können einer MultiUse-Klasse öffentliche Methoden hinzufügen, um die von den Skripting-Clients kommenden Aufrufe an die privaten Methoden benutzerdefinierter Schnittstellen weiterzuleiten.

Hüllkomponenten für Skripting-Clients erstellen

Während der oben beschriebene Ansatz das Problem zwar löst, kann es vorkommen, dass Sie eine andere Technik brauchen, um die Originalkomponente nicht verändern zu müssen. Dies gilt zum Beispiel, wenn Sie es mit einer Komponente zu tun haben, die benutzerdefinierte Schnittstellen implementiert, Sie aber ihren Quellcode nicht ändern können. Statt der Originalkomponente öffentliche Methoden hinzuzufügen, legen Sie eine komplementäre Hüllkomponente an, auf die die Skripting-Clients zugreifen. Diese Hüllkomponente ist ein Adapter, der die Rolle eines »Widerstandsreglers« zwischen einer Skripting-Engine und einer nicht skriptfreundlichen Komponente übernimmt.

Eine Hüllkomponente erzeugt normalerweise ein Objekt aus der Originalkomponente und ordnet ihre öffentlichen Methoden den Methoden aller benutzerdefinierter Schnittstellen zu. Betrachten Sie den Code für die CDingo-Komponente in Listing 4.2. Ein Skripting-Client kann nicht darauf zugreifen, weil die Komponente ihre gesamte Funktionalität über benutzerdefinierte Schnittstellen bereitstellt.

Listing 4.3 zeigt den Code für die Hüllkomponente CDingoWrapper an. Diese Hüllkomponente verwendet den Ereignis-Handler Class_Initialize, um ein CDingo-Objekt zu erzeugen. Während der Initialisierung fordert die Hüllkomponente eine separate Verbindung für jede benutzerdefinierte Schnittstelle an. Die Hüllkomponente verwendet diese Verbindungen, um öffentliche Methodenaufrufe an die Implementierungen der Schnittstellenmethoden weiterzugeben.

Nachdem Sie alle benutzerdefinierten Schnittstellen zu einer großen Standardschnittstelle erweitert haben, kann eine Transaktion auf jede Methode zugreifen. Die Transaktion instanziert einfach ein Objekt unter Verwendung der Hüllkomponente und ruft die Methoden direkt auf. Hier folgt ein Beispiel für VBScript-Code auf einer ASP-Seite:

```

Dim ProgID, Dog
ProgID = "DogServer.CDingoWrapper"
Set Dog = Server.CreateObject(ProgID)
Dog.Bark
Dog.RollOver 3
Dog.FetchSlippers
Set Dog = Nothing

```

Wenn sie die Methoden in der Hüllkomponente definieren und implementieren, müssen Sie die von dem Skripting-Client übergebenen Parameter an die zu verhüllende Komponente weitergeben. Häufig können Sie die Parameter weiterreichen, ohne sie konvertieren zu müssen. In einigen Situationen kann es jedoch erforderlich sein, die Parameter in Typen umzuwandeln, die mit den von Ihnen verwendeten Skripting-Clients kompatibel sind.

Beachten Sie, dass Sie dafür verantwortlich sind, Ihre Hüllkomponente entsprechend anzupassen, wenn jemand die Originalkomponente um eine neue Schnittstelle erweitert. Außerdem sollten Sie erkennen (und dankbar dafür sein), dass Ihre Versionsprobleme nicht so kompliziert sind, wie sie sein könnten, weil die Hüllkomponente nur Skripting-Clients unterstützt. Wir werden im nächsten Kapitel noch genauer auf die Komponentenversionen eingehen.

```
' MultiUse-Klasse CDingo
Implements IDog
Implements IWonderDog

Private Sub IDog_Bark()
    ' Ihre Implementierung
End Sub

Private Sub IDog_RollOver(ByRef Rolls As Integer)
    ' Ihre Implementierung
End Sub

Private Sub IWonderDog_FetchSlippers()
    ' Ihre Implementierung
End Sub
```

Listing 4.2: Diese Klassendefinition erzeugt eine Komponente, auf die Skripting-Clients nicht direkt zugreifen können.

```
' MultiUse-Klasse CDingoWrapper
' CDingo-Hüllkomponente für Skripting-Clients
Private Ref1 As IDog
Private Ref2 As IWonderDog

Private Sub Class_Initialize()
    Set Ref1 = New CDingo    ' Objekt erzeugen und
    Set Ref2 = Ref1         ' Verbindungen einrichten.
End Sub

Public Sub Bark()
    Ref1.Bark ' Aufruf weiterleiten.
End Sub

Public Sub RollOver(ByRef Rolls As Variant)
    Dim iRolls As Integer
    iRolls = Rolls
    Ref1.RollOver(iRolls) ' Aufruf weiterleiten.
    Rolls = iRolls
End Sub

Public Sub FetchSlippers()
    Ref2.FetchSlippers ' Aufruf weiterleiten.
End Sub

Private Sub Class_Terminate()
    Set Ref1 = Nothing
    Set Ref2 = Nothing
End Sub
```

Listing 4.3: Diese Hüllklasse erlaubt es Skripting-Clients, auf die von der CDingo-Klasse implementierten Schnittstellen zuzugreifen.

Beobachtungen zu Skripting-Clients

Skripting-Clients nehmen also etwas so Elegantes wie die schnittstellenbasierte Programmierung und lassen es weniger elegant werden. Wenn sie eine Komponente mit benutzerdefinierten Schnittstellen entwickeln wollen, sollten Sie voraussetzen, dass nur Clients darauf Zugriff haben, die in der Lage sind, `QueryInterface` aufzurufen. Eine solche Komponente bietet keine direkte Unterstützung von Skripting-Clients.

Das Gute daran ist, dass Sie weiterhin abstrakte und konkrete Klassen programmieren können, wenn keine Unterstützung von Skripting-Clients gefordert wird. Beispielsweise könnten Sie mehrere benutzerdefinierte Schnittstellen anlegen, die definieren, wie Ihre Geschäftslogik-Komponenten mit den Komponenten zusammenarbeiten, in denen der Datenzugriffscode enthalten ist. Dieser Ansatz funktioniert ausgezeichnet, solange Sie voraussetzen können, dass alle Ihre Geschäftskomponenten mit Visual Basic oder C++ erstellt werden.

Wenn Sie eine Komponente erzeugen, die Skripting-Clients unterstützt, sollten Sie die Verwendung benutzerdefinierter Schnittstellen besser vermeiden. Legen Sie einfach ihre Komponenten, die öffentliche Methoden verwenden, in *MultiUse*-Klassen an. Skripting-Clients können alle Methoden direkt aufrufen.

In einer wenig perfekten Welt kann man nicht immer alles vorhersehen. Wenn Sie beispielsweise eine Web-Site entwickeln, auf der ein ASP-Client auf eine vorgefertigte Komponente zugreifen muss, die benutzerdefinierte Schnittstellen implementiert, müssen Sie ein bisschen Zusatzarbeit leisten. Einer der besten Ansätze ist es, Hüllkomponenten wie oben bereits gezeigt zu verwenden. Sie erweitern alle benutzerdefinierten Schnittstellen zu einer einzigen Schnittstelle, auf die die Skripting-Clients Zugriff haben. Auf diese Weise können Ihre Skripting-Clients den Weg des geringsten Widerstands gehen.

Fehler mit Servern auslösen

Zweifellos gibt es Situationen, in denen ein Client Ihre Komponente aufruft, um eine bestimmte Aufgabe zu erledigen, Ihr Code aber die Anforderung nicht erfüllen kann. Das könnte beispielsweise passieren, wenn der Datenbankserver offline ist oder wenn der Aufrufer nicht die erforderlichen Berechtigungen besitzt. Vielleicht hat der Client versucht, eine Transaktion auszuführen, die eine Ihrer Geschäftsregeln verletzt. Als Autor von Komponenten müssen Sie diese Probleme vorhersehen. Wenn sie auftreten, müssen Sie Fehler auslösen und diese von Ihrem Code zum Aufrufer zurückgeben lassen.

Wenn Sie bereits Erfahrung mit Visual Basic haben, sind Sie vielleicht schon mit der dort gebotenen Unterstützung für das Auslösen und die Verarbeitung von Fehlern vertraut. Wenn Sie einen Fehler an den Aufrufer zurückgeben möchten, rufen Sie einfach die `Raise`-Methode des integrierten `Err`-Objekts auf. Auf der Clientseite verarbeiten Sie Fehler mit Hilfe von `On Error`-Anweisungen und entsprechenden Fehlerbehandlungsroutinen. Soll ein Fehler über Komponentengrenzen hinweg weitergegeben werden, muss das auf sprachneutrale Weise passieren.

Angenommen, Sie haben mit Visual Basic eine Komponente angelegt, die einen Fehler auslöst. Wenn andere Visual Basic-Programmierer auf Ihre Komponente zugreifen, verarbeiten sie Ihren Fehler genau so wie jeden anderen Laufzeitfehler. Andere clientseitige Programmierer könnten jedoch unter Verwendung anderer Skriptingsprachen auf Ihre Komponente zugreifen, beispielsweise C++, VBScript oder JavaScript. Glücklicherweise stellt COM auch für diese Clients eine Möglichkeit bereit, Ihre Fehler zu verarbeiten.

COM unterstützt zwei sprachunabhängige Mechanismen für die Benachrichtigung über Fehlerursachen. Ein COM-Objekt verwendet sowohl `HRESULTS` als auch COM-Ausnahmen, um den Client darüber zu informieren, ob ein Aufruf nicht erfolgreich ausgeführt werden konnte. `HRESULT` ist der Standardrückgabewert, der dem Client zuverlässig mitteilt, ob ein Aufruf erfolgreich war. COM-Ausnahmen setzen auf `HRESULTS` auf und werden genutzt, um vollständigere Fehlerinformationen sprachneutral von einem Objekt zum Client zurückzugeben.

Bevor das COM-Team ein eigenes Modell zur Ausnahmeverarbeitung schuf, bot Visual Basic eine eigene Unterstützung für das Auslösen und die Verarbeitung von Fehlern. Dabei achtete man streng darauf, dass das Fehlerverarbeitungsmodell von Visual Basic direkt auf die COM-Ausnahmen aufgesetzt werden konnte. Die Visual Basic-Laufzeitumgebung verarbeitet `HRESULTS` und COM-Ausnahmen im Hintergrund. Sie lösen die Fehler einfach in Ihren Komponenten aus und verarbeiten Sie in Ihrem Client-Code. Das sind großartige Neuigkeiten für Visual Basic-Programmierer – Sie müssen Ihren Programmierstil nicht ändern, wenn Sie anfangen, Komponenten zu entwickeln.

HRESULT

`HRESULT` zeigt den Erfolg oder Misserfolg eines Methodenaufrufs in COM an. Es ist besonders in Aufrufen wichtig, die über das Netzwerk ausgeführt werden. COM hat diesen Rückgabewert standardisiert, so dass die Remote-Infrastruktur (die RPC-Schicht) den Client informieren kann, wenn ein Problem während des Versuchs auftritt, ein Objekt zu erreichen. Wenn ein Objekt, das Teil einer aktiven Verbindung ist, ein plötzlicher Tod ereilt, kann der Client dies erkennen und entsprechend darauf reagieren.

In COM müssen alle Remote-Methoden `HRESULT` zurückgeben. Um von einem entfernten Computer aus aufgerufen werden zu können, muss eine Methode über eine Proxy-/Stub-Schicht ausgeführt werden. Wie Sie sehen, gibt jede Methode, die Sie in Visual Basic erzeugen oder aufrufen, `HRESULT` zurück. Das Visual Basic-Team hat sein Bestes getan, alle Spuren der Existenz von `HRESULTS` zu verwischen. Wenn Sie `OleView.exe` aufrufen, um Ihre Server zu überprüfen, werden Sie feststellen, dass jede von Ihnen erstellte Methode mit `HRESULT` als Rückgabewert definiert ist. Dazu zählen auch Visual Basic-Methoden, die als Funktionen definiert sind. Eine Methode könnte beispielsweise wie folgt in Visual Basic definiert sein:

Die Zeile

```
Function GetTaxRate() As Double
```

wird in IDL wie folgt übersetzt:

```
HRESULT GetTaxRate([out, retval] double* );
```

Immer dann, wenn eine Methode einen logischen Rückgabewert verlangt, kommt IDL dieser Anforderung nach, indem es den `[retval]`-Parameter zur Verfügung stellt. Eine COM-Methode kann lediglich einen `[retval]`-Parameter besitzen, der an letzter Position aufgeführt sein muss. Visual Basic und Java sind Beispiele für Sprachen, die diese speziellen Parameter verwenden, so dass diese als Rückgabewerte und nicht als Ausgabeparameter von Programmierern benutzt werden können. Auf diese Weise wird der Aufruf einer Funktion selbstverständlicher und die Programmierer müssen `HRESULT` nicht manuell auswerten. `HRESULT` wird von der Visual Basic-Laufzeitschicht überprüft. Ein Client erhält einen Fehlercode zurück, wenn ein Methodenaufruf misslingt.

Wozu braucht COM einen Standardrückgabewert? Ein Standardrückgabewert erlaubt es der COM-Infrastruktur in bestimmten Situationen, bekannte Fehler an den Client zurückzugeben. Um die Bedeutung von Standardrückgabewerten zu demonstrieren, wollen wir annehmen, dass ein Client mit einem Remote-Objekt verbunden ist, das sich irgendwo im Netzwerk befindet. Es kann passieren, dass das Objekt abstürzt oder nicht mehr erreichbar ist, während der Client weiterhin eine Verbindung dazu aufrechterhält, beispielsweise, wenn der Server abstürzt oder wenn der Server-Computer von einem übereifrigen Administrator wiederholt gebootet wird. In beiden Fällen besitzt die Client-Applikation einen ausstehenden Verweis auf ein Objekt, das nicht mehr existiert. Diese Situation kann auch dann auftreten, wenn Netzwerkprobleme das Übermitteln einer Methodenanforderung zwischen dem Client und dem Objekt unmöglich machen. In jeder dieser Situationen misslingt ein Remote-Methodenaufruf.

Die zugrunde liegende Remote Procedure Call-Schicht (RPC) muss den Client darüber informieren, dass ein Zugriff auf den Server nicht möglich ist. Das ist einer der wichtigsten Gründe, warum das COM-Team die Standardisierung von HRESULT vorgenommen hat. Ein Client kann HRESULT auswerten, um zu überprüfen, ob der Aufruf erfolgreich war. Erkennt der Client, dass der Aufruf nicht erfolgreich war, kann er HRESULT auf weitere Informationen zu dem vorliegenden Problem hin abfragen. HRESULT teilt dem Client mit, ob der Fehler aus dem eigentlichen Objekt oder aus einem anderen Teilsystem stammt, wie beispielsweise der zugrunde liegenden RPC-Schicht.

Wie sich zeigt, können nur C++-Programmierer direkt mit HRESULTs arbeiten. Visual Basic macht alles viel einfacher. Auf der Clientseite wertet die Visual Basic-Laufzeitumgebung HRESULTs für Sie im Hintergrund aus. Wenn der Aufruf ohne Fehler ausgeführt wurde, heißt das, die Visual Basic-Laufzeitumgebung hat HRESULT ausgewertet und festgestellt, dass der Aufruf erfolgreich war. Stellt die Visual Basic-Laufzeitumgebung dagegen fest, dass HRESULT einen Fehler anzeigt, löst sie einen Laufzeitfehler aus.

Hatten Sie als Visual Basic-Programmierer schon einmal mit HRESULTs zu tun? Wohl selten. Die Visual Basic-Laufzeitumgebung erkennt viele bekannte HRESULTs, die von der COM-Bibliothek und anderen COM-bezogenen Teilsystemen zurückgemeldet werden. Nicht immer löst die Visual Basic-Laufzeitumgebung alle Situationen so elegant. Tritt ein nicht bekannter HRESULT auf, löst sie einen Fehler aus und platziert den reinen HRESULT-Wert in Err.Number.

Visual Basic 6 fügt der Laufzeitschicht eine Unterstützung für das Abfangen der meisten dieser HRESULT-Rückgabewerte hinzu, die anzeigen, ob ein prozessexternes Objekt nicht mehr existiert oder nicht erreichbar ist. Wenn die Visual Basic-Laufzeitbibliothek feststellt, dass der HRESULT der RPC-Schicht ein nicht mehr existierendes Objekt meldet, wird der Fehler mit der Nummer 462 zusammen mit der Fehlerbeschreibung »Der Remote-Server-Computer existiert nicht oder ist nicht verfügbar« ausgelöst. Dieser Laufzeitfehler hilft Ihnen in den meisten Situationen, in denen Sie auf ein nicht mehr existierendes Objekt oder auf ein Objekt reagieren müssen, das nicht erreicht werden kann. Visual Basic 5 bietet diese Unterstützung nicht. Wenn Sie Visual Basic 5 verwenden, müssen Sie immer mit HRESULT arbeiten, um ermitteln zu können, ob ein Objekt noch existiert. Sie sollten deshalb immer wissen, was in HRESULT gespeichert wird.

HRESULT ist ein 32-Bit-Wert, der drei unterschiedliche Informationen enthält. Abbildung 4.4 zeigt, wie der physische Speicher von HRESULT unterteilt ist, um die Rangebene, das Umfeld des Fehlers und einen applikationsspezifischen Fehlercode zu speichern. Die Rangebene, die in Bit 31 abgelegt wird, gibt an, ob die Methode erfolgreich aufgerufen wurde. Das ist der Fall, wenn das Bit nicht gesetzt ist. Ist das Bit gesetzt, wurde die Methode nicht ausgeführt.

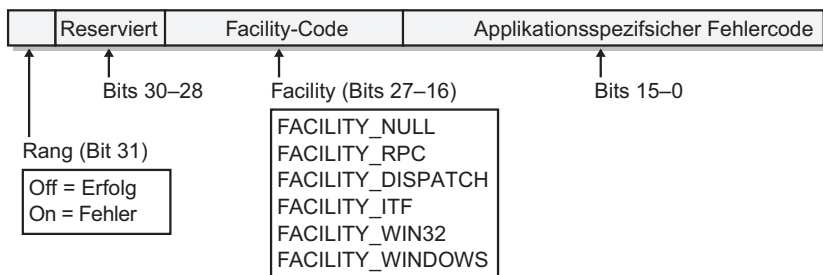


Abbildung 4.4: In COM geben alle Remote-Methoden auf der physischen Ebene HRESULT zurück.

Die Gruppe der Bits im mittleren Abschnitt von HRESULT wird als *Facility-Code* bezeichnet. Sie teilt dem Client mit, wo der Fehler aufgetreten ist. COM-Standardfehler verwenden den Code FACILITY_NULL. Ein Objekt sollte FACILITY_ITF benutzen, wenn es einen schnittstellenspezifischen Fehlercode an den Client übermitteln möchte. Tabelle 4.2 zeigt einige andere mögliche Werte für den Facility-Code. Wie Sie sehen, können unter Windows 2000 Fehler von den unterschiedlichsten Teilsystemen erzeugt werden.

Woher wissen Sie, wann Sie es in Err.Number mit einem HRESULT-Rohwert zu tun haben? Sie erkennen es eigentlich ganz einfach, weil der Wert eine sehr große negative Zahl ist. Wenn Sie einen Wert wie beispielsweise -2147220992 sehen, wissen Sie, dass es sich um einen HRESULT handelt, der auf einen Fehler hinweist. Das oberste Bit eines HRESULT teilt dem Aufrufer mit, ob der Aufruf erfolgreich war. Ist das Bit gesetzt, ist der Aufruf fehlgeschlagen. Visual Basic betrachtet dieses Bit immer als Hinweis darauf, ob die Zahl positiv oder negativ ist, weil es immer Integer mit Vorzeichen verwendet.

Wenn ein Laufzeitfehler auftritt, der Ihnen einen HRESULT-Rohwert übergibt, können Sie versuchen, festzustellen, was passiert ist, indem Sie den Facility-Code und den Fehlercode auswerten. Dazu brauchen Sie eine bitweise Operation mit dem AND-Operator. Betrachten Sie die folgende Funktion:

```
Function GetFacilityCode(ByVal HRESULT As Long) As Integer
    GetFacilityCode = ((HRESULT And &HFFF0000) / 65536)
End Function
```

Diese Funktion extrahiert die Bits von HRESULT, die den Facility-Code bilden. Mit Hilfe einer Bitmaske ignoriert sie alle anderen Bits. Anschließend dividiert sie durch 65536, um den tatsächlichen Facility-Code zu berechnen. Tabelle 4.2 zeigt die in Windows 2000 definierten Werte für den Facility-Code.

Facility-Code	Wert
FACILITY_NULL	0
FACILITY_RPC	1
FACILITY_DISPATCH	2
FACILITY_STORAGE	3
FACILITY_ITF	4
FACILITY_WIN32	7

Facility-Code	Wert
FACILITY_WINDOWS	8
FACILITY_SECURITY	9
FACILITY_CONTROL	10
FACILITY_INTERNET	12
FACILITY_MEDIASERVER	13
FACILITY_MSMQ	14
FACILITY_SETUPAPI	15
FACILITY_SCARD	16

Tabelle 4.2: Facility-Codes in Windows 2000

Durch die Auswertung des Facility-Codes untersuchen Sie die Art des Problems. Facility-Codes, wie zum Beispiel FACILITY_RPC, FACILITY_WIN32 und FACILITY_WINDOWS bedeuten, dass der Fehler von der zugrunde liegenden Infrastruktur gemeldet wurde. Diese Codes könnten auch bedeuten, dass ein Objekt nicht mehr existiert oder nicht erreichbar ist. Leider können Sie nicht sicher sein, dass diese Codes wirklich bedeuten, dass ein Objekt nicht mehr existiert. Sie können diese Facility-Codes auch dann erhalten, wenn die Parameter eines Methodenaufrufs nicht korrekt an das oder von dem Objekt übermittelt werden konnten. In diesem Fall könnte der nächste Aufruf des Objekts durchaus erfolgreich sein.

Wenn Sie feststellen, dass ein HRESULT-Wert einen Facility-Code angibt, sollten Sie in den unteren 16 Bits des Werts nachsehen, um eine genaue Fehlerbeschreibung zu erhalten. Hier folgt eine Funktion, die die unteren 16 Bit von HRESULT betrachtet und alles andere ignoriert:

```
Function GetErrorCode(ByVal HRESULT As Long) As Integer
    GetErrorCode = (HRESULT And 65535)
End Function
```

Nachdem Sie den Facility-Code und den Fehlercode ermittelt haben, können Sie den Fehler in der Headerdatei *WINERROR.H* nachsehen, die Teil des Win32 SDK ist. Diese Datei enthält alle vom System erzeugbaren HRESULTs. Wie Sie sich nun bestimmt vorstellen können, kann die Arbeit auf dieser Ebene für einen durchschnittlichen Visual Basic-Programmierer sehr zeitintensiv sein. Beachten Sie, dass Sie es nicht sehr häufig mit HRESULTs zu tun haben werden – nur wenn Sie einen vom System erzeugten Fehler erhalten, den die Visual Basic-Laufzeitumgebung nicht kennt.

COM-Ausnahmefehler

Ein HRESULT-Wert kann anzeigen, ob eine Methode erfolgreich ausgeführt wurde. Der Wert kann außerdem einen applikationsspezifischen Fehlercode enthalten. HRESULT enthält jedoch keine Textfehlermeldungen oder Informationen darüber, wo der Fehler aufgetreten ist. Sie können *COM-Ausnahmefehler* für die Übermittlung detaillierter Fehlerinformationen zwischen Objekten und Clients benutzen. COM-Ausnahmefehler werden von einigen Funktionen und Schnittstellen der COM-Bibliothek verwaltet.

Bevor wir die internen Details der COM-Ausnahmefehler besprechen, sollten Sie wissen, dass sie vor dem Visual Basic-Programmierer versteckt werden. Das Visual Basic-Team hat das proprietä-

re Fehlerbehandlungsmodell von Visual Basic überarbeitet und für die COM-Ausnahmefehlerbehandlung verwendet. Visual Basic-Programmierer müssen sich also keine Gedanken über COM-Ausnahmefehler machen.

Lassen Sie uns nun erörtern, wie COM-Ausnahmen arbeiten. Jedes COM-Objekt, das Ausnahmefehler auslöst, muss dies bekannt geben, indem es eine Standardschnittstelle mit der Bezeichnung `ISupportErrorInfo` unterstützt. Diese Schnittstelle ermöglicht es den Clients, alle Schnittstellen eines Objekts abzufragen, um zu ermitteln, ob das Objekt Ausnahmefehler für eine bestimmte IID unterstützt. Das Objekt kann dem Client mit Hilfe dieser Schnittstelle mitteilen: »Ja, ich unterstütze COM-Ausnahmefehler für die Schnittstelle, die Du verwendest.«

Wenn ein Objekt einen Ausnahmefehler auslösen möchte, muss es ein Fehlerobjekt erstellen, indem es die Funktion `CreateErrorInfo` aus der COM-Bibliothek aufruft. Das Objekt benutzt diesen Verweis, um das soeben erstellte Fehlerobjekt mit den erforderlichen Informationen, wie zum Beispiel der Fehlerbeschreibung und der Fehlerquelle, zu füllen. Das Objekt ruft anschließend `SetErrorInfo` auf, um das Fehlerobjekt mit dem logischen Thread des Aufrufers zu verknüpfen. (Der Begriff *logischer Thread* bezieht sich auf eine Folge von Methodenaufrufen, die sich über mehrere physische Threads erstrecken können.) Schließlich gibt das Objekt den Wert `HRESULT` zurück, der dem Client mitteilt, dass ein Fehler aufgetreten ist.

Wenn ein Client, der Erfahrung mit Ausnahmefehlern hat, `HRESULT` auswertet und feststellt, dass eine Methode nicht korrekt ausgeführt wurde, fragt er das Objekt ab, um zu ermitteln, ob es die `ISupportErrorInfo`-Schnittstelle unterstützt. In diesem Fall ruft der Client eine Methode über diese Schnittstelle auf, um zu prüfen, ob die aktuell verwendete Schnittstelle unterstützt wird. Wenn das Objekt anzeigt, dass es Ausnahmefehler für diese Schnittstelle unterstützt, kann der Client davon ausgehen, dass ein Fehlerobjekt existiert. Dieses Objekt enthält Informationen über die Art des Problems. Wenn der Client also ein `HRESULT` auswertet und feststellt, dass ein Methodenaufruf nicht erfolgreich war, kann er anschließend `GetErrorInfo` aufrufen, um einen `IErrorInfo`-Schnittstellenverweis auf das Fehlerobjekt entgegenzunehmen. Diese Schnittstelle lässt den Client alle Fehlerinformationen empfangen, die von einem Objekt versendet werden. Wie Sie sehen, erfordern COM-Ausnahmefehler sehr viel Arbeit auf beiden Seiten. Glücklicherweise erledigt Visual Basic diese Arbeit für Sie im Hintergrund, aber Sie können dennoch an diesem sprachneutralen Fehlerverarbeitungsschema teilhaben.

Beachten Sie, dass COM-Ausnahmen höchst effizient sind. Wenn ein Objekt eine COM-Ausnahme auslöst und der Client sich auf einer anderen Maschine befindet, wird die gesamte Fehlerinformation über das Netzwerk geschickt. Man braucht keine zweite Verbindung zwischen dem Client-Computer und dem Server-Computer, um zusätzliche Fehlerinformationen zu ermitteln. Die Fehlerinformation für eine COM-Ausnahme wird immer zusammen mit dem Rückgabewert der Methode über das Netzwerk übertragen. Der Client kann alle diese Fehlerinformationen einem lokalen Cache entnehmen.

Zuordnen von COM-Ausnahmen in Visual Basic

Wenn Sie bereits mit dem Fehlerbehandlungsmodell von Visual Basic gearbeitet haben, wissen Sie, dass es sehr viel einfacher zu handhaben ist als das COM-Ausnahmefehlermodell. Glücklicherweise arbeiten beide Modelle problemlos miteinander. Visual Basic-Programmierer lösen weiterhin Fehler aus und fangen diese ab, wie sie es seit Jahren gewöhnt sind. Die erforderlichen Details werden von der Laufzeitschicht transparent zugeordnet, um dem sprachunabhängigen COM-Modell zu entsprechen.

In Visual Basic verfügt jede Komponente über eine integrierte Unterstützung von `ISupportErrorInfo`. Wenn Sie einen Fehler explizit von einem Visual Basic-Server auslösen lassen, indem Sie `Err.Raise`

benutzen, erstellt die Laufzeitschicht automatisch ein COM-Fehlerobjekt, in dem die erforderlichen Informationen anschließend abgelegt werden. Die Laufzeitschicht führt ebenfalls die notwendigen Aufrufe durch, um eine COM-Ausnahme korrekt auszulösen. Alles was Sie dazu benötigen, ist ein Code wie der des folgenden Beispiels.

```
Dim MyErrorCode As Long, Source As String, Description As String
MyErrorCode = vbObjectError + 512
Source = "DogServer.CBeagle.Bark"
Description = "Die Katze hat die Zunge des Hundes erwischt."
Err.Raise MyErrorCode, Source, Description
```

Wenn Sie `Err.Raise` aufrufen, müssen Sie einen benutzerdefinierten Fehlercode übergeben. Visual Basic stellt die interne Konstante `vbObjectError` als Ausgangspunkt für Ihre benutzerdefinierten Fehlercodes zur Verfügung. Die Konstante `vbObjectError` verweist auf den `HRESULT`-Wert eines Fehlers. Dies geschieht mit dem Facility-Code `FACILITY_ITF`. Das COM-Team empfiehlt jedoch, Ihre eigenen Fehlercodes mit `vbObjectError + 512` zu beginnen, um Konflikte mit den von Microsoft definierten Fehlern zu vermeiden.

Das vorherige Beispiel zeigt auch, wie die Eigenschaften `Source` und `Description` eines Fehlercodes gesetzt werden. Die Quelle (`Source`) sollte der Client-Applikation mitteilen, wo das Problem aufgetreten ist. Das ist besonders während des Debuggens wichtig. Die Beschreibung (`Description`) sollte in einer für den Menschen lesbaren Form angeben, welcher Fehler aufgetreten ist. Häufig zeigt man die Beschreibung in einem Dialogfeld an, so dass der Anwender sofort informiert wird. Wenn es für den Fehler eine Lösung gibt, sollte diese ebenfalls in der Beschreibung vermerkt werden.

Die von Ihren Servern ausgelösten benutzerdefinierten Fehler sollten über entsprechende Fehlercodes verfügen. Die Fehlercodes sollten mit Hilfe einer Enumeration in einer Typbibliothek aufgeführt werden. Sie können Fehlercodes wie folgt definieren:

```
' Dog-Server-Fehlercodes
Enum ErrorCodesEnum
    dsDogUnavailable = vbObjectError + 512
    dsDogUnagreeable
    dsDogNotCapable
    dsDogNotFound
    dsUnexpectedError
End Enum
```

Die clientseitige Zuordnung von COM-Ausnahmen in Visual Basic ist ebenso transparent wie die objektseitige Zuordnung. Wenn eine Methode `HRESULT` zurückgibt, verwendet die Zuordnungsschicht die `ISupportErrorInfo`-Schnittstelle, um zu ermitteln, ob das Objekt COM-Ausnahmen unterstützt. In diesem Fall nimmt die Zuordnungsschicht das COM-Fehlerobjekt entgegen und verwendet es, um ein Visual Basic-Fehlerobjekt mit den erforderlichen Informationen zu füllen. Die Zuordnungsschicht löst anschließend einen Fehler für den Aufrufer aus. Sie können eine `Select Case`-Anweisung in einer Client-Applikation benutzen, um eine Fehlerbehandlung wie die nachfolgend aufgeführte zu erstellen:

```
Sub MakeDogBark(Dog As IDog)
    On Error GoTo MakeDogBark_Err
    Dog.Bark
' Bei Erfolg beenden.
Exit Sub
```



```

' Beim Fehler in den Handler eintreten.
MakeDogBark_Err:
    Select Case Err.Number
        Case dsDogNotCapable:
            ' Erforderliche Verarbeitung im Client vornehmen.
            MsgBox Err.Description, vbCritical, "Fehler: Hund nicht fähig "
        Case Else
            ' Immer einen Handler bereitstellen, der die letzte Möglichkeit verarbeitet.
            MsgBox Err.Description, vbCritical, "Unerwarteter Fehler"
    End Select
End Sub

```

Ihre Komponenten können auch Fehler auslösen, die auf von Microsoft definierten HRESULTs basieren. Wenn Sie beispielsweise einen Laufzeittest in einer Ihrer Methoden vornehmen und feststellen, dass der Aufrufer nicht die erforderliche Berechtigung für die Ausführung besitzt, sollten Sie einen Fehler auslösen. Statt jedoch einen eigenen, benutzerdefinierten Code bereitzustellen, können Sie das Standard-HRESULT verwenden, das anzeigt, dass Ihr Zugriff zurückgewiesen wurde. Hier ein Beispiel, wie Sie HRESULT in Ihrem Code verwenden:

```

Const E_ACCESSDENIED = &H80070005
Err.Raise E_ACCESSDENIED, , "Zugriff wurde verweigert"

```

In der Datei *WINERROR.H* sehen Sie, welche weiteren HRESULTs Microsoft definiert hat. Möglicherweise haben Sie bemerkt, dass das obige Codebeispiel die Konstante HRESULT in einem hexadezimalen Format definiert. Sie können aber auch den äquivalenten Dezimalwert -2147024891 verwenden. Beides funktioniert. *WINERROR.H* verwendet das hexadezimale Format, wodurch es einfacher wird, die Werte zu kopieren und in den Visual Basic-Quellcode einzufügen.

Bedingungen zum Auslösen von Fehlern

Wenn Sie COM-Server veröffentlichen, müssen Sie sowohl die Regeln von COM als auch von Visual Basic befolgen. Dies gilt besonders dann, wenn Sie in einer Umgebung arbeiten, in der einige Komponenten mit anderen Werkzeugen und Sprachen als Visual Basic erstellt werden. Dieser Abschnitt beschreibt einige Regeln, die Sie sich merken sollten.

- Alle COM-Ausnahmen, die Ihren Server verlassen, sollten explizit von Ihrem Code ausgelöst werden. Dies bedeutet, dass nur solche Fehler Ihren Server verlassen sollten, die explizit mit der `Err.Raise`-Syntax erzeugt wurden. Sie müssen verhindern, dass nicht bearbeitete Fehler vom Server ausgehen. Die meisten Visual Basic-Programmierer wissen, dass ein nicht verarbeiteter Fehler die Ausführung einer Client-Applikation abbricht. Ein Visual Basic-Server leitet einen nicht bearbeiteten Fehler einfach an den Client weiter. Wenn beispielsweise eine Methode in Ihrem Server versucht, eine Division durch null durchzuführen, und der entsprechende Fehler nicht bearbeitet wird, leitet die Laufzeitschicht den Fehler direkt an die Client-Applikation weiter.

Der Verzicht auf die Bearbeitung von Ausnahmen, die nicht explizit in einem Server ausgelöst werden, gilt als ein schlechter Programmierstil. Jedes COM-Objekt, das `ISupportInfo` implementiert, kann nur die eigenen Ausnahmen weiterleiten. Sie müssen somit jeden von Visual Basic generierten Fehler in Ihrem Server abfangen. Wenn Ihr Server einen Fehler auslöst, den er nicht bearbeiten kann, sollten Sie einen applikationsspezifischen Fehler (*Unerwarteter Fehler in...*) an den Client weiterleiten.

- Verwenden Sie immer eine erklärende Textbeschreibung, wenn Sie einen Ausnahmefehler auslösen. Versuchen Sie, informative Beschreibungen an dem Ort zu erstellen, an dem der Fehler aufgetreten ist. Die Prozedur, in der der Fehler aufgetreten ist, verfügt gewöhnlich über die besten kontextuellen Informationen. Diese Beschreibung kann von jeder Position in der Aufrufkette an den ursprünglichen Aufrufer zurückgegeben werden. Eine Beschreibung wie zum Beispiel »Bestellungen für DM 2000 werden nicht akzeptiert. Der Kunde verfügt lediglich über ein Kreditlimit von DM 1250.« ist immer besser als »Ein Fehler trat während der Bestellübermittlung auf.«. Die Behandlungsroutine in der Client-Applikation zeigt den Anwendern die Fehlermeldungen an. Denken Sie deshalb daran, dass Fehlercodes für Programmierer und Fehlerbeschreibungen für Anwender gedacht sind.
- Verwenden Sie immer Auflistungen, um Ihre Fehlercodes zu definieren. Dies ermöglicht anderen Programmierern, Ihre Fehlercodes in der Typbibliothek einzusehen. Visual Basic-Programmierer, die Ihre Server benutzen, können Ihre Fehlercodes mit dem Objektkatalog betrachten.
- Definieren Sie Ihre Fehlercodes immer mit `vbObjectError + 512`. Die unteren Fehlercodes sind für COM reserviert und haben spezielle Bedeutungen. Wenn Sie die Konstante `vbObjectError` benutzen und 512 hinzuaddieren, werden Ihre Codes zu keinen Konflikten führen und auch nicht als systemdefinierte Fehlercodes interpretiert.
- Versuchen Sie, eine Dokumentation mit Ihren Servern auszuliefern, die anderen Programmierern die bekannten Fehler beschreibt. Sie können nur wenige Informationen mit einem Auflistungswert wie zum Beispiel `dsDogNotCapable` angeben. Wenn möglich, sollten Sie eine Liste mit Lösungen und Abhilfen für jede Fehlerbedingung erstellen.
- Lassen Sie in der Fehlerbehandlungsroutine Ihres Servers niemals ein Nachrichtendialogfeld mit `MessageBox` anzeigen. Die Entscheidung, mit dem Anwender zu interagieren, sollte immer der Client-Applikation überlassen werden. Eine Fehlerbehandlung, die ein Nachrichtendialogfeld anzeigt, kann zum Absturz eines prozessexternen Servers führen. Wenngleich Sie ein Nachrichtendialogfeld problemlos mit einem prozessinternen Server anzeigen lassen können, sollten Sie dies nicht tun, da eine derartige Vorgehensweise als schlechter Programmierstil verurufen ist. Die Client-Applikation sollte alle Interaktionen mit den Anwendern kontrollieren.

Fehlerverarbeitung in Skripting-Clients

Die Clients Ihrer Komponenten werden möglicherweise als ASP-Skripts oder als Microsoft WSH-Dateien (Windows Script Host) geschrieben, wobei Sprachen wie etwa VBScript oder JavaScript verwendet werden. Wenn Sie Komponenten für solche Skripting-Clients schreiben, können und sollten Sie die Fehler wie oben beschrieben auslösen, wenn eine Methode nicht erfolgreich ausgeführt werden kann. Sie sollten jedoch beachten, dass der Code zur Fehlerverarbeitung in einem Skripting-Client nicht sehr elegant ist. Betrachten wir ein Beispiel für einen Client, der in VBScript geschrieben wurde:

```
' VBScript in einer WSH-Datei
Const dsDogNotCapable = &H80040202
On Error Resume Next
Dim Spot
Set Spot = CreateObject("DogServer.CBeagle")
' Methode aufrufen.
Spot.Bark
' Erfolg oder Misserfolg überprüfen.
Select Case Err.Number
    Case 0
```

```

' Alles ist gut. Der Aufruf hat keinen Fehler produziert.
Case dsDogNotCapable
    MsgBox "Der Hund kann gerade nicht bellen!"
Case Else
    MsgBox "Unerwarteter Fehler #" & Err.Number & ": " & Err.Description
End Select
' Das Fehlerobjekt muss gelöscht werden.
Err.Clear
' Jetzt rufen Sie weitere Methoden auf und überprüfen in ähnlicher Weise, ob Fehler aufgetreten sind.

```

Wenn Ihre Komponente Auflistungen verwendet, um Fehlercodes auszugeben, kann ein Skripting-Client diese Information normalerweise nicht nutzen. Eine einfache Möglichkeit, Abhilfe für das Problem zu schaffen, ist die Verwendung von Konstanten in den Skripting-Clients, um den Fehlercodes lesbaren Namen zuzuordnen. Beachten Sie, dass Sie bei der Fehlerverarbeitung in VBScript auf die Anweisung `On Error Resume Next` beschränkt sind. Das bedeutet, Sie müssen nach jedem Methodenaufruf `Err.Number` auswerten. Ist `Err.Number` ungleich null, ist ein Laufzeitfehler aufgetreten. Beachten Sie, dass Sie das `Err`-Objekt mit der `Clear`-Methode löschen müssen, nachdem ein Laufzeitfehler verarbeitet wurde, bevor Sie einen weiteren Aufruf ausführen.

Zusammenfassung

In diesem Kapitel ging es um die Grundlagen des Entwurfs und des Erstellens von Servern mit Visual Basic. Zunächst müssen Sie entscheiden, ob Sie Ihre Komponenten in ActiveX-DLLs oder ActiveX-EXEs verpacken. Ihre Entscheidung wirkt sich auf die Leistung, Robustheit und Sicherheit Ihres Servers aus. Viele Entwürfe verteilter Anwendungen fordern, dass Ihre Objekte in einem prozessexternen Server ausgeführt werden. Komplexere Laufzeitumgebungen wie COM+ und IIS bieten einen Ersatzprozess an, d.h. Sie können Ihren Code weiterhin als ActiveX-DLL veröffentlichen.

Wie Sie gesehen haben, fügt Visual Basic Ihren Servern während des Kompilierens viel Unterstützung hinzu. Der Visual Basic-Compiler und die Laufzeitschicht geben Ihr Bestes, um die Details zu verbergen, die für COM erforderlich sind. Sie als Autor von Komponenten können sich dann darauf konzentrieren, die gewünschte Funktionalität über die in der Typbibliothek definierten Schnittstellen bereitzustellen und eine Implementierung dieser Schnittstellen in Ihren Komponenten zu realisieren.

Außerdem ging es um komplexe Techniken wie beispielsweise benutzerdefinierte Schnittstellen, Auflistungen und UTDs für den Entwurf Ihrer Komponenten. Sie haben erfahren, dass viele dieser Techniken Probleme bereiten, wenn Sie Skripting-Clients verwenden. Am besten treffen Sie bereits früh in der Entwurfsphase eine Entscheidung, welche Client-Typen Ihre Komponenten nutzen sollen.

Darüber hinaus beschrieb das Kapitel das Auslösen und die Bearbeitung von Fehlern. Visual Basic stellt eine transparente Zuordnung von COM-Ausnahmen zur Verfügung, die das Programmiererleben erleichtert. Visual Basic-Objekte und -Clients können an der Fehlerbehandlung von Komponenten teilnehmen, die in anderen Sprachen geschrieben wurden. Größtenteils verbirgt Visual Basic die müßigen Details von `HResults` und COM-Ausnahmen, so dass Sie Fehler so auslösen und verarbeiten können, wie Sie es als Visual Basic-Programmierer gewohnt sind.

5 Komponentenversionsverwaltung

-
- 113 Versionsverwaltung unter COM
 - 126 Benutzerdefinierte Schnittstellen erstellen
-

Die Komponentenversionsverwaltung ist ein wichtiger Bestandteil der Betriebsumgebung. Um die leistungsfähigsten Features von COM nutzen zu können, müssen Sie in der Lage sein, Ihre Server zu inspizieren und zu ersetzen. Dies darf jedoch keine nachteiligen Auswirkungen auf die Client-Applikationen haben. Das vorliegende Kapitel zeigt Ihnen die von Microsoft Visual Basic zur Verfügung gestellten Optionen, die Ihnen helfen, dieses Ziel zu erreichen. Für einige Leser wird dieses Thema das wichtigste in diesem Buch sein.

Wir werden die wesentlichen Versionsverwaltungsfragen untersuchen, mit denen ein Visual Basic-Programmierer konfrontiert wird, wenn er Komponenten für eine Applikation erstellt und diese pflegt. Sie sollten diese Richtlinien unabhängig davon beachten, ob Ihre Applikation auf COM+, MTS oder früheren Versionen von COM basiert. Sie werden lernen, wann und wie das binäre Kompatibilitätsschema von Visual Basic verwendet wird. Ich werde außerdem einige wichtige Einschränkungen hinsichtlich der binären Kompatibilität nennen und Ihnen eine alternative Technik für die Versionsverwaltung von Visual Basic-Komponenten zeigen. Zu dieser Technik zählt das Definieren von Schnittstellen mit IDL (Interface Definition Language) und das Erstellen von benutzerdefinierten Typbibliotheken.

Mein Ziel besteht darin, Sie in diesem Kapitel mit genügend Wissen auszustatten, damit Sie eines der größten Versprechen von COM realisieren können: die Fähigkeit, Ihren Applikationscode einfach zu warten und zu erweitern, nachdem er in Betrieb genommen wurde.

Versionsverwaltung unter COM

COM wurde auf der Grundlage der Unveränderlichkeit der Schnittstellen entworfen. Sobald eine COM-Schnittstelle (ein benutzerdefiniertes vTable-Layout, das anhand einer IID identifiziert wird) veröffentlicht wird, können ihre Methoden nicht mehr verändert werden. Dies bedeutet, dass Sie die Methodendefinitionen in einer Schnittstelle, die sich bereits in Betrieb befindet, nicht modifizieren, entfernen oder neue hinzufügen können. Wenn Sie diese Regeln verletzen, erklären Sie die Versionsverwaltungsgarantie, die sich aus den COM-Spezifikationen ergibt, für ungültig. Wenn Sie jedoch die Regeln der Unveränderlichkeit von Schnittstellen befolgen, können Sie die Version einer Komponente nach Herzenslust verwalten. Wenn Sie das Verhalten einer Komponente verändern oder erweitern möchten, definieren Sie einfach eine neue Schnittstelle und implementieren diese.

Wenn Ihre Komponente alle zuvor unterstützten Schnittstellen implementiert, wird jede neue Version der Komponente sowohl mit älteren als auch mit neuen Clients problemlos zusammenarbeiten. Ein Client kann zudem ein Objekt zur Laufzeit abfragen, um zu ermitteln, ob eine bestimmte IID unterstützt wird. Ein neuerer Client kann sich deshalb elegant absetzen, wenn er auf eine ältere Version der Komponente trifft. COM gestattet Ihnen somit, eine neue Version eines Clients oder einer Komponente in Betrieb zu nehmen, ohne bereits vorhandenen Code ändern zu müssen.

Visual Basic hat einen großen Teil seines Erfolges dem Umstand zu verdanken, dass es die Komplexität der zugrunde liegenden Plattform verbirgt. In der Absicht, die Programmierung einfacher werden zu lassen, hat das Visual Basic-Team sehr viele komplexe und groteske sowie mit COM verknüpfte Details versteckt. COM fordert zum Beispiel eine formalisierte Trennung der Schnittstelle von der Implementierung. Visual Basic verlangt von einem COM-Programmierer jedoch nicht, mit benutzerdefinierten Schnittstellen zu arbeiten.

Stellen Sie sich vor, Sie erstellen eine ActiveX-DLL mit der Bezeichnung *DOGSERVER.DLL* für eine COM+-Applikation. Wenn Sie eine MultiUse-Klasse namens *CCollie* mit einigen wenigen öffentlichen Methoden erzeugen, generiert Visual Basic eine Co-Klasse und eine Standardschnittstellendefinition für Sie. Der Co-Klasse wird der logische Name *CCollie* zugewiesen und die Schnittstellendefinition erhält die Bezeichnung *_CCollie*. Wenn Sie die DLL erstellen, kompiliert Visual Basic die Co-Klasse und die Schnittstellendefinitionen in eine Typbibliothek. Diese wird in die ausführbare Datei des Servers integriert. Sie, als der Komponentenantor, müssen keine separate Schnittstelle definieren.

Aus der Perspektive der Client-Seite betrachtet, ist die Programmierung unter Visual Basic sehr einfach. Wenn eine Client-Applikation eine Objektreferenz vom Typ *CCollie* erstellt, ordnet Visual Basic diesen Verweis still und heimlich der Schnittstelle *_CCollie* zu. Visual Basic-Clients und -Objekte kommunizieren somit mit Hilfe zulässiger COM-Schnittstellen. Visual Basic-Programmierer müssen jedoch nicht unter dem Aspekt benutzerdefinierter Schnittstellen arbeiten.

Das Visual Basic-Team wusste, dass viele Visual Basic-Programmierer Komponenten erstellen, ohne explizit benutzerdefinierte Schnittstellen zu generieren. Es wollte deshalb ein Versionsverwaltungsschema für Komponenten zur Verfügung stellen, das auf Standardschnittstellen basiert. Ein Ziel dieses Schemas besteht darin, den Komponentenantoren zu gestatten, Methoden zu späteren Versionen einer Komponente hinzuzufügen.

Wie Sie noch sehen werden, wird dieses Versionsverwaltungsschema mit der Projekteinstellung *version compatibility* gesteuert. Sie sollten verstehen, wie Visual Basic diese Einstellung benutzt, um die GUIDs, wie zum Beispiel IIDs und CLSIDs, von Build zu Build zu kontrollieren. Sie sollten außerdem Visual Basics komplizierten Mechanismus kennen lernen, der Ihnen erlaubt, Methoden zu späteren Versionen der Komponente hinzuzufügen. Wir werden diese Themen gleich besprechen.

Lassen Sie uns das Thema Komponentenversionsverwaltung mit einer wichtigen Frage beginnen: Welche Client-Typen werden Ihre Komponente verwenden? Wie ich bereits in Kapitel 4 erwähnt habe, können Sie Client-Code in zwei Kategorien unterteilen: in Scripting-Clients, die mit Hilfe der späten Bindung auf Ihre Objekte zugreifen, und weiterentwickelte Clients, die über die direkte vTable-Bindung auf Ihre Objekte zugreifen.

Wenn Sie davon ausgehen, dass Ihre Komponente lediglich von Clients benutzt wird, die sich einer dieser Kategorien zuordnen lassen, ist die Versionsverwaltung nicht sehr kompliziert. Die Komponentenversionsverwaltung für unmittelbar vTable-gebundene Clients ist schwieriger. Bisweilen werden Sie jedoch beide Client-Typen unterstützen müssen, wenn Sie die Version einer Komponente verwalten. Sie sollten in diesen Situationen verstehen, wie die Versionsverwaltung hinter den Kulissen abläuft.

Die Komponentenversionsverwaltung für Scripting-Clients

Es ist nicht schwierig, die Version einer Komponente für Scripting-Clients zu verwalten, da diese Clients nur eine geringe Abhängigkeit von der Komponente aufweisen. Der Client-Code in einem ASP-Script enthält beispielsweise in der Regel die ProgID der Komponente und die Namen einiger Methoden und Eigenschaften. Scripting-Clients zeigen jedoch keine Abhängigkeiten von IIDs oder benutzerdefinierten vTable-Layouts, da sie die späte Bindung verwenden.

Das Erstellen einer Komponente für einen Scripting-Client ist mit Visual Basic sehr einfach. Sie generieren lediglich eine neue `MultiUse`-Klasse und fügen einige öffentliche Methoden und Eigenschaften hinzu. Visual Basic erstellt immer eine duale Schnittstelle, die öffentliche Methoden und Eigenschaften enthält und als Standardschnittstelle der Komponente dient. Der Zugriff auf diese Schnittstelle ist sowohl mit der späten Bindung als auch mit der direkten vTable-Bindung möglich. Wenn ein Scripting-Client ein neues Objekt von Ihrer Komponente instanziert, ist er immer über die Standardschnittstelle verbunden. Sobald die Verbindung besteht, nutzt der Client die späte Bindung, um auf öffentliche Methoden oder Eigenschaften zuzugreifen.

Ich habe in dem letzten Kapitel die Probleme beschrieben, die auftreten, wenn Sie Scripting-Clients zusammen mit benutzerdefinierten Schnittstellen einsetzen. Wenn Sie Komponenten für Scripting-Clients erstellen möchten, sollten Sie diese nicht unter dem Aspekt benutzerdefinierter Schnittstellen entwickeln. Wenn Sie bereits eine Visual Basic-Komponente besitzen, die benutzerdefinierte Schnittstellen implementiert, ist es sehr schwierig, von einem Scripting-Client aus darauf zuzugreifen. Sie müssen hierzu in der Regel zusätzlichen Code schreiben, der den Scripting-Client mit Methoden verbindet, die nicht Bestandteil der Standardschnittstelle sind.

Wenn Sie einmal akzeptiert haben, dass Scripting-Clients `MultiUse`-Klassen mit öffentlichen Methoden und Eigenschaften benötigen, ist es sehr einfach, Ihre Komponenten zu erstellen und deren Versionen zu verwalten. Sie fügen Ihrer Klasse einige Methoden hinzu und kompilieren und verteilen anschließend Ihre DLL. Wenn Sie eine weitere oder mehrere neue Methoden hinzufügen möchten, benachrichtigen Sie einfach die Klasse und kompilieren und verteilen Ihre DLL erneut. Es spielt sogar keine Rolle, in welchem Versionskompatibilitätsmodus Sie sich befinden, wenn Sie die DLL erneut erstellen. Ältere Scripting-Clients können die ältere Version Ihrer DLL oder die neue Version verwenden. Neuere Clients können die neuere Version und damit auch die neuen Methoden benutzen.

Die einzige Situation, die eine besondere Aufmerksamkeit von Ihnen verlangt, tritt ein, wenn ein neuerer Scripting-Client Kontakt mit einer älteren Version der Komponente aufnimmt. Wenn Sie zum Beispiel eine Methode zu einer älteren Version einer Komponente hinzufügen und diese von einem Scripting-Client aus aufrufen, wird der Client mit Problemen konfrontiert, sobald er versucht, die frühere Version der DLL zu benutzen. Der neue Client würde versuchen, eine Bindung zu einer Methode herzustellen, die nicht existiert. Dies würde zu einem Laufzeitfehler führen. Wenn Sie jedoch auf dieses Szenario vorbereitet sind, werden Sie nicht mit schwerwiegenden Problemen konfrontiert.

Die Komponentenversionsverwaltung für vTable-gebundene Clients

Alles wird etwas komplizierter aber auch interessanter, wenn Sie die Version einer Visual Basic-Komponente verwalten, die die direkte vTable-Bindung benutzt. Sie haben die Auswahl zwischen der binären Kompatibilität oder dem Einsatz von IDL. Die binäre Kompatibilität ist eine gute Wahl, wenn Sie möchten, dass Visual Basic Ihre Schnittstelle im Hintergrund definiert.

Wenn Sie Ihre Schnittstellen getrennt von Ihrer Komponente definieren möchten, sollten Sie IDL benutzen und Typlibibliotheken mit dem MIDL-Compiler generieren.

Wir werden mit der binären Kompatibilität und der in Visual Basic integrierten Unterstützung der Versionsverwaltung beginnen. Sie sollten zunächst verstehen, wie sich die Versionskompatibilitäts-Projekteinstellung von Build zu Build auf Ihre Komponenten auswirkt. Wenn Sie alle Einstellungen korrekt vornehmen, können Sie Ihre Komponenten sicher aktualisieren und erweitern. Wenn Sie Ihr Projekt jedoch mit den falschen Kompatibilitätseinstellungen konfigurieren, kann Ihr Leben geradezu miserabel werden.

Bevor ich die Arbeitsweise dieser Einstellung erläutere, werfen wir noch einmal einen Blick darauf, wie Visual Basic reagiert, wenn Sie eine ActiveX-DLL erstellen. In unserem Beispiel werden wir eine DLL namens *DOGSERVER.DLL* mit einer *MultiUse*-Klasse namens *CCollie* generieren, die einige öffentliche Methoden enthält. Wenn Sie die DLL erzeugen, generiert Visual Basic die Definition für eine Co-Klasse mit der Bezeichnung *CCollie* und eine Standardschnittstelle mit dem Namen *_CCollie*. Diese werden in der Typlibibliothek des Servers veröffentlicht.

Richten Sie Ihre Aufmerksamkeit auf den Inhalt der Typlibibliothek des Servers. Dort ist eine Co-Klasse namens *CCollie* und eine Standardschnittstelle mit der Bezeichnung *_CCollie* vorhanden. Dies sind jedoch nur die logischen Namen für die entsprechenden COM-Typen. COM benötigt GUIDs, um Co-Klassen und Schnittstellen auf der physikalischen Ebene zu identifizieren.

Sie können eine GUID explizit erstellen, indem Sie eine Systemfunktion in der COM-Bibliothek aufrufen, die mit *CoCreateGUID* bezeichnet ist. Alternativ hierzu können Sie ein Hilfsmittel namens *GUIDGEN.EXE* benutzen. COM-Entwickler, die häufig C++ verwenden, generieren GUIDs auf diese Weise. Um die Komponentenerstellung unproblematisch werden zu lassen, entschied das Visual Basic-Team, dass die GUID-Erstellung und -Verwaltung ein transparentes Feature der Entwicklungsumgebung sein sollte.

Während der Kompilierung generiert Visual Basic transparent die erforderlichen GUIDs und weist diese Ihren Co-Klassen und Schnittstellen zu, sofern dies möglich ist. Dieses offenkundige Generieren von GUIDs führt dazu, dass Sie Komponenten schnell und einfach erstellen können. Es kann Visual Basic-Entwickler aber auch in große Schwierigkeiten bringen.

Der logische Name Ihrer Co-Klasse ist *CCollie*. Der physikalische Name ist jedoch eine CLSID, wie zum Beispiel {259D5370-DD2D-11D2-8319-0080C7067BA1}. Visual Basic generiert außerdem eine IID für die Standardschnittstelle *_CCollie*. Darüber hinaus wird diese Standardschnittstelle als [hidden] gekennzeichnet, um sie vor »naiven« Programmierern zu verbergen.

Zusätzlich zu den CLSIDs und IIDs generiert Visual Basic eine GUID um die Typlibibliothek zu identifizieren, die in die DLL integriert wird. Diese GUID ist als LIBID referenziert. Visual Basic kann weitere GUIDs für Enumerationen und benutzerdefinierte Typen (UDTs) erstellen, wenn dies notwendig wird.

Stellen Sie sich vor, Sie fügen diese beiden öffentlichen Methoden zur *CCollie*-Klasse hinzu:

```
Public Sub Bark()  
    ' Implementierung  
End Sub
```

```
Public Sub RollOver(ByRef Rolls As Integer)  
    ' Implementierung  
End Sub
```


Wenn Sie diese Methoden zu `CCollie` hinzufügen, fügen Sie sie gleichzeitig zur Standardschnittstelle `_CCollie` hinzu. Denken Sie daran, dass Client-Applikationen und die COM-Laufzeit immer über die IID und nicht über den logischen Namen `_CCollie` auf die Standardschnittstelle zugreifen.

Wenn Sie die erste Version Ihrer DLL erstellen und anderen Programmierern zur Verfügung stellen, referenzieren diese die integrierte Typlibibliothek. Die Programmierer beginnen dann, mit ihren Programmen auf die `CCollie`-Komponente zuzugreifen. Wenn sie ihren Code kompilieren, weisen ihre Applikationen somit Abhängigkeiten von der CLSID für `CCollie` und von der IID für die Standardschnittstelle auf. Diese Client-Applikationen verwenden die direkte vTable-Bindung, um auf `CCollie`-Objekte zuzugreifen.

Dies alles funktioniert problemlos, bis Sie sich dazu entschließen, die `CCollie`-Komponente zu aktualisieren oder zu erweitern. Bevor Sie eine zweite Version Ihrer DLL ausliefern, müssen Sie Ihr Projekt mit der korrekten Versionskompatibilitätseinstellung konfigurieren. Diese Einstellung kann für jedes Projekt auf der *Komponente*-Registerkarte des *Projekteigenschaften*-Dialogfeldes vorgenommen werden, wie Abbildung 5.1 zeigt. Die drei möglichen Einstellungen sind nachfolgend erläutert:

- **Keine Kompatibilität.** Diese Einstellung veranlasst Visual Basic, alle GUIDs erneut zu generieren. Dazu zählen IIDs, CLSIDs und die LIBID. Die internen Versionsnummern der Typlibibliotheken eines Projekts, das mit dieser Einstellung kompiliert wird, werden auf 1.0 zurückgesetzt.
- **Projekt-Kompatibilität.** Diese Einstellung veranlasst Visual Basic, alle IIDs erneut zu generieren. Die LIBID bleibt jedoch unverändert, so dass andere Programmierer die Typlibibliothek nicht referenzieren müssen, nachdem Sie Ihren Server erneut erstellt haben.
- **Binär-Kompatibilität.** Alle GUIDs einschließlich der IIDs werden für alle Builds beibehalten, um zuvor kompilierte Clients zu unterstützen, die die direkte vTable-Bindung benutzen. Visual Basic führt außerdem Kompatibilitätstests durch, um sicherzustellen, dass sich die Methodensignatur nicht verändert hat.

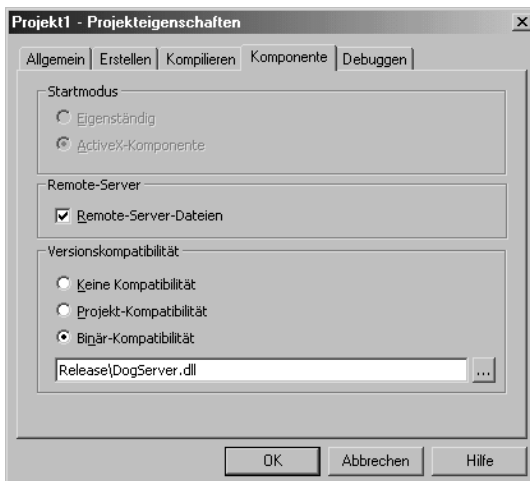


Abbildung 5.1: Sie spezifizieren die Versionskompatibilitätseinstellung für das gesamte Projekt auf der Registerkarte *Komponente* im Dialogfeld *Projekteigenschaften*.

Lassen Sie uns jede dieser Einstellungen erörtern. *Keine Kompatibilität* bedeutet, dass Visual Basic alle GUIDs für jedes Element in der Typlibibliothek erneut generiert und keine Kompatibilitätstests durchführt. Dies geschieht auch, wenn Sie Ihre Komponente zum ersten Mal kompilieren.

Die Einstellungen *Projekt-Kompatibilität* und *Binär-Kompatibilität* führen dazu, dass der Visual Basic-Compiler eine früher kompilierte Version des Servers referenziert. Visual Basic muss die Typbibliothek untersuchen, die sich in einem früheren Build Ihres Servers befindet, um die GUIDs zu vergleichen und Kompatibilitätstests für die letzte Version Ihres Source-Codes durchzuführen. Sie können das Textfeld unter den drei Versionskompatibilitätsoptionsfeldern benutzen, um einen Pfad zu der Serverdatei anzugeben, die als Referenz dienen soll.

Die Projektkompatibilitätseinstellung führt abhängig von der von Ihnen verwendeten Visual Basic-Version zu unterschiedlichen Ergebnissen. Unter Visual Basic 5 behält der Compiler die GUID Ihrer Typbibliothek bei, generiert aber Ihre CLSIDs und IIDs neu. Unter Visual Basic 6 generiert der Compiler lediglich Ihre IIDs neu. Die CLSIDs bleiben für alle Builds unverändert.

Diese Unterschiede zwischen den Versionen waren zunächst beabsichtigt, um die Arbeit mit Scripting-Clients zu vereinfachen, die mit Hilfe des `OBJECT`-Tags (zum Beispiel ActiveX-Steuerelemente) CLSIDs in ihren Source-Code einbetten. Beachten Sie bitte, dass Scripting-Clients, die den `OBJECT`-Tag benutzen, die späte Bindung einsetzen. Dies bedeutet, dass sie Abhängigkeiten von CLSIDs, aber nicht von IIDs aufweisen.

Wenn Sie im *Projekt-Kompatibilität*-Modus kompilieren, führt der Compiler, unabhängig von der Visual Basic-Version, keine Überprüfung durch, um festzustellen, ob sich Ihre Schnittstellen verändert haben. Visual Basic verwirft stattdessen alle IIDs und generiert einen neuen Satz. Sie sollten ebenfalls beachten, dass auch die mit den Enumerationen verknüpften GUIDs und die UDTs neu erstellt werden.

Die Einstellungen *Keine Kompatibilität* und *Projekt-Kompatibilität* führen dazu, dass alle IIDs immer dann geändert werden, wenn Sie Ihre DLL erneut erstellen. Dies ist problematisch, wenn Sie Clients besitzen, die Abhängigkeiten von IIDs aufweisen, die wiederum nicht mehr vorhanden sind. Wenn ein anderer Programmierer eine Client-Applikation für eine frühere Version Ihrer DLL kompiliert, löst die Applikation einen Fehler aus, wenn sie versucht, ein Objekt aus der neuen Version zu aktivieren. Die Benutzer wären sehr verstimmt, da sie mit Fehlermeldungen wie der in Abbildung 5.2 dargestellten konfrontiert würden.

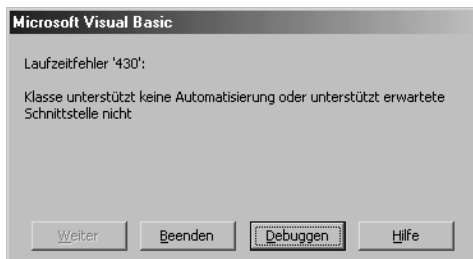


Abbildung 5.2: Wenn ein Client versucht, ein Objekt mit einer IID zu aktivieren, die nicht von der Komponente unterstützt wird, löst Visual Basic den Fehler 430 aus

Was ist die Ursache des Fehlers? Der Client versucht, sich an ein Objekt zu binden. Er benutzt dazu die IID eines vorherigen Builds. Diese IID wird jedoch nicht von späteren Builds der DLL unterstützt. Die offenkundige und einzige Lösung dieser unglücklichen Situation besteht darin, die Versionskompatibilitätseinstellung des Projekts auf *Binär-Kompatibilität* zu setzen.

Binär-Kompatibilität

Binär-Kompatibilität ist die einzige Einstellung, die Ihnen erlaubt, einen Server, der bereits bestehende vTable-gebundene Clients unterstützt, neu zu erstellen. Der Compiler lässt die GUIDs aller Builds unverändert. Hierzu zählen ebenfalls die IIDs. Der Visual Basic-Compiler führt darü-

ber hinaus Kompatibilitätstests durch, um sicherzustellen, dass Sie keine unzulässigen Änderungen an Ihren Schnittstellen, Enumerationen oder UDTs vorgenommen haben.

Die Versionskompatibilitätsregeln, die von Visual Basic im Modus *Binär-Kompatibilität* angewendet werden, lauten vereinfacht zusammengefasst wie folgt: Solange Sie keine bereits vorhandenen Methodensignaturen verändern oder Enumerationswerte respektive UDT-Felder gespeichert haben, wird der Compiler Ihre Änderungen akzeptieren und Ihre Komponente rekompilieren. Wenn sich Visual Basic während der Rekompilierung nicht beschwert, hat die neue Version die Kompatibilitätstests bestanden. Die neue Version der DLL ist dann kompatibel mit früher kompilierten Clients. Wenn Sie jedoch beispielsweise eine Methode entfernen oder den Datentyp eines Parameters ändern, wird der Visual Basic-Compiler dies mit einem Dialogfeld bemängeln, das dem in Abbildung 5.3 dargestellten gleicht.

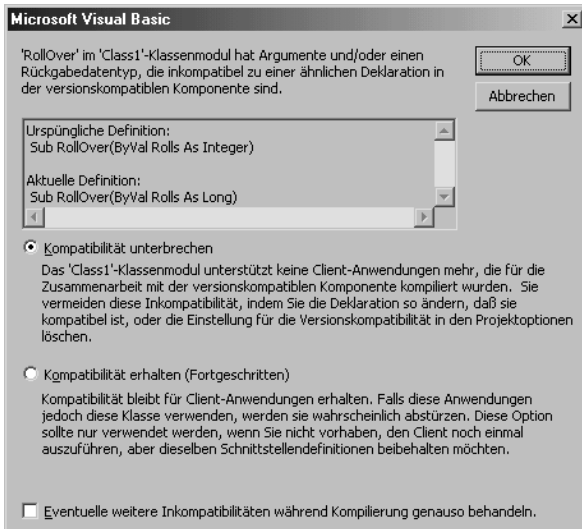


Abbildung 5.3: Visual Basic zeigt dieses Dialogfeld an, wenn Sie sich im *Binär-Kompatibilität*-Modus befinden und versuchen, eine Schnittstelle zu kompilieren, die nicht kompatibel mit der vorherigen Version ist.

Dieses Dialogfeld, das eine Inkompatibilität ermittelt hat, bietet Ihnen zwei Möglichkeiten zur Auswahl an: Die Option *Kompatibilität unterbrechen* hat zwei Funktionen. Sie ändert zunächst alle IIDs und setzt jede Schnittstellenversionsnummer auf 1.0 zurück. Danach wird die Versionsnummer Ihrer Server-Typbibliothek um den Wert 1 erhöht. *Kompatibilität unterbrechen* führt somit zu dem gleichen Ergebnis wie das erneute Erstellen Ihres Projekts mit der Einstellung *Projekt-Kompatibilität*.

Die Option *Kompatibilität erhalten (Fortgeschritten)* ist in den meisten Fällen nicht sehr nützlich. Sie lässt die IID einer Schnittstelle auch dann unverändert, wenn sich die aufrufende Syntax für eine oder mehrere Methoden geändert hat. Sie sollten jedoch an die möglichen Folgen denken, die der Einsatz dieser Option haben kann. Sie kann bei Client-Applikationen, die für frühere Versionen Ihres Servers kompiliert wurden, zu einem katastrophalen Fehler führen.

Der benutzerdefinierte vTable-Bindungscode, den Visual Basic in Client-Applikationen integriert, ist sehr spezifisch. In der Regel geschehen seltsame Dinge, wenn sich ein Client und ein Objekt nicht einig darüber sind, wie ein Parameter während eines Methodenaufrufs übergeben werden soll. Ihr Client kann schnell mit einer unschönen, vom System generierten Fehlermeldung wie der in Abbildung 5.4 dargestellten abstürzen.

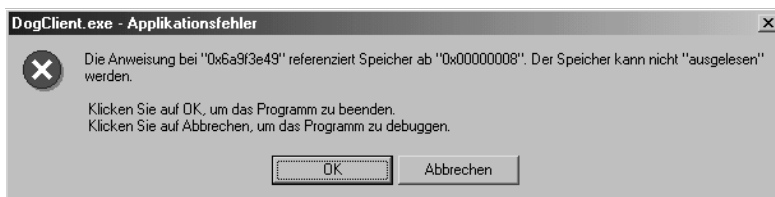


Abbildung 5.4: Dieses Win32-Dialogfeld ist ein Zeichen dafür, dass Ihre Client-Applikation einen grausamen und plötzlichen Tod gestorben ist

Wenn Sie eine der beiden Optionen *Kompatibilität unterbrechen* oder *Kompatibilität erhalten* verwenden, sollten Sie davon ausgehen, dass alle zuvor kompilierten Clients nicht mehr gültig sind. Wenn Sie das in Abbildung 5.3 dargestellte Dialogfeld sehen, sollten Sie Ihren Source-Code modifizieren, so dass dieser kompatibel mit früheren Versionen des Servers wird. Sobald Sie Ihre DLL kompilieren können, ohne dass dieses Dialogfeld erscheint, wissen Sie, dass die DLL mit allen bestehenden Clients kompatibel ist.

Erweitern der Standardschnittstelle

Wenn Sie im *Binär-Kompatibilität*-Modus arbeiten, können Sie bereits vorhandene Methodensignaturen nicht verändern. Dies bedeutet, dass Sie weder den Namen einer Methode, den Typ des Rückgabewerts noch den Typ eines anderen Parameters modifizieren können. Sie können auch keine optionalen Parameter an das Ende der Methodensignatur anfügen. Ihre Intuition sagt Ihnen möglicherweise, dass das Hinzufügen eines optionalen Parameters erlaubt sein müsste, da es keine Änderungen an der Aufrufsyntax des Clients erfordert. Optionale Parameter generieren jedoch einen Variant-Wert, der während der Methodenausführung auf der physikalischen Ebene auf dem Stack abgelegt wird. Aus diesem Grund führt das Hinzufügen eines optionalen Parameters zu einer bestehenden Methode dazu, dass die gesamte Schnittstelle inkompatibel wird.

Solange Sie Ihre Methodensignaturen unverändert lassen, können Sie natürlich jede bereits bestehende Methodenimplementierung problemlos verändern, sofern Sie im *Binär-Kompatibilität*-Modus kompilieren. Wenn Sie darüber nachdenken, ist dies genau das, was COM verkörpert. Solange Sie nicht die physikalische Natur der Schnittstelle ändern, sollten Sie eine Implementierung immer modifizieren können. Visual Basic lässt Sie sogar noch einen Schritt weiter gehen und neue Methoden zu einer *MultiUse*-Klasse hinzufügen.

Wenn Sie neue öffentliche Methoden zu einer zweiten Version einer Komponente hinzufügen, fügen Sie tatsächlich neue Methoden zur Standardschnittstelle hinzu. Visual Basic muss somit eine neue Standardschnittstelle erstellen, die eine Obermenge der ursprünglichen Standardschnittstelle bildet. Die neue Schnittstelle ist dann »versionskompatibel« mit der alten Schnittstelle. Da jede Schnittstelle durch eine eigene vTable repräsentiert wird, muss Visual Basic zwei unterschiedliche IIDs verwenden.

Lassen Sie uns zu unserem Beispiel zurückkehren. Version 1 der *CCollie*-Klasse definiert zwei öffentliche Methoden: *Bark* und *RollOver*. Wenn Sie den Server zum ersten Mal erstellen, generiert Visual Basic automatisch eine Schnittstelle mit der Bezeichnung *_CCollie* (mit einer IID), die diese beiden Methoden definiert. Wenn Sie eine neue Methode namens *FetchSlippers* zu Version 2 von *CCollie* hinzufügen und den Server im *Binär-Kompatibilität*-Modus erneut erstellen, generiert Visual Basic eine neue IID für die neue Standardschnittstelle, die alle drei Methoden enthält. Visual Basic muss jedoch ebenfalls eine Unterstützung in der neuen Version zur Verfügung stel-

len, um Clients zu berücksichtigen, die die alte IID benutzen. Abbildung 5.1 zeigt beide Schnittstellen und die damit verknüpften vTables.

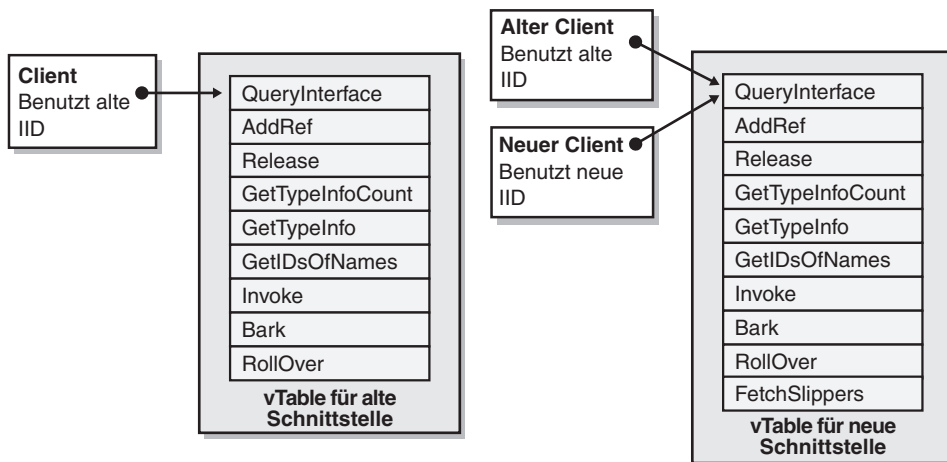


Abbildung 5.5: Das vom Binär-Kompatibilität-Modus angebotene Erweiterbarkeitsschema basiert auf der Benachrichtigung von versionskompatiblen Schnittstellen. Die neue Schnittstelle ist mit der alten versionskompatibel, da ihre vTable eine Obermenge der Methoden bildet, die von der alten vTable repräsentiert werden.

Ab diesem Punkt wird alles ein wenig bizarr. Die vom Compiler für das `CCollie`-Objekt generierte Implementierung von `QueryInterface` gibt einen Verweis auf die neue Schnittstelle aus, wenn ein Client diese oder die alte Schnittstelle verlangt. Clients, die für die ursprüngliche Version der DLL erstellt wurden, werden, basierend auf der neuen IID, an eine vTable gebunden. Sie kennen jedoch nur die ersten beiden der drei Methoden. Da die neue Schnittstelle versionskompatibel mit der alten ist, erfüllt sie die Erwartungen des Clients und arbeitet ohne Probleme.

Beachten Sie bitte, dass die Typbibliothek, die für die versionskompatible DLL erstellt wurde, lediglich die Schnittstellendefinition für die neue IID enthält. Die Schnittstellendefinition für die alte IID ist nicht enthalten. Die COM-Laufzeit muss jedoch möglicherweise Proxy-/Stub-Code basierend auf der alten IID erstellen. Der Proxy-/Stub-Code für Visual Basic-Objekte wird vom Universal-Marshaler zur Laufzeit erstellt. Der Universal-Marshaler generiert Proxies und Stubs, indem er die Schnittstellendefinitionen in den Typbibliotheken untersucht.

Doch wie kann der Universal-Marshaler einen Proxy oder einen Stub für die alte Schnittstelle erstellen, wenn er keinen Zugriff auf die Schnittstellendefinition erhält? Dies geschieht mit Hilfe der Schnittstellenweiterleitung. Der Selbstregistrierungscode in einer versionskompatiblen DLL fügt einen Schlüssel für die alte IID zur Registrierung hinzu, wie in Abbildung 5.6 dargestellt. Dieser Schlüssel enthält Weiterleitungsinformationen, die den Universal-Marshaler auf die neue, versionskompatible IID verweisen. Der Universal-Marshaler kann somit versionskompatible Proxies und Stubs erzeugen, indem er auch dann die mit der neuen IID verknüpfte Schnittstellendefinition verwendet, wenn ein Client eine Verbindung basierend auf der alten IID anfordert.

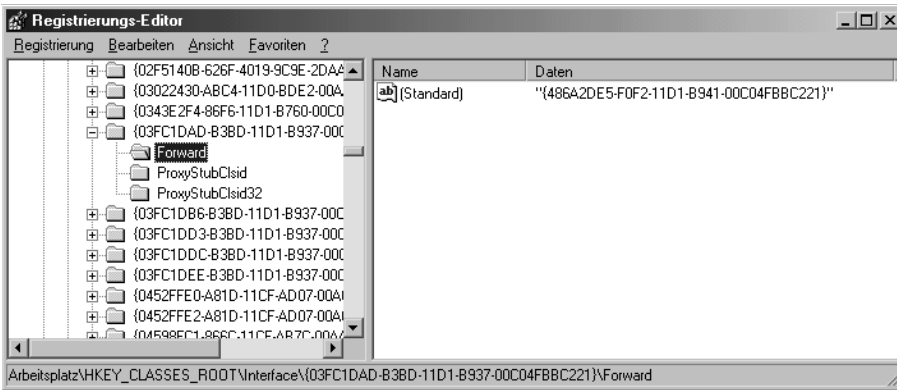


Abbildung 5.6: Visual Basic und der Universal-Marshaler verwenden die Schnittstellenweiterleitung, damit versionskompatible Schnittstellen in der Betriebsumgebung eingesetzt werden können

Visual Basic erlaubt Ihnen, von Version zu Version kompatible Server zu erstellen. Version 4 kann beispielsweise mit Version 3, Version 3 mit Version 2 und Version 2 schließlich mit der Originalversion kompatibel sein. Ein Client, der lediglich die Originalversion der Schnittstelle kennt, kann somit über viele kompatible Schnittstellen hinweg weitergeleitet werden, bevor er auf eine Schnittstelle trifft, die in einer Typlibibliothek auf dem Host-Computer definiert ist.

Doch was geschieht wirklich hinter den Kulissen? Sobald Sie die DLL kompiliert haben, können Sie *OLEVIEW.EXE* benutzen, um die Typlibibliothek zu dekompile und festzustellen, was Visual Basic eigentlich getan hat und wie Visual Basic Ihre GUIDs von Build zu Build verwaltet. Nachfolgend ist eine gekürzte Version aufgeführt, die Sie sehen, wenn Sie Version 1 kompiliert haben:

```
[ // GUID für Standardschnittstelle _ IID1
  uuid(C44FF440-E9A0-11D1-9266-0080C72D2182),
  version(1.0), hidden, dual, oleautomation
]
interface _CCollie : IDispatch {
    HRESULT Bark();
    HRESULT RollOver([in, out] short* Rolls);
};

[ // Diese GUID ist die CLSID.
  uuid(EDE2823B-DE19-11D2-9A2C-0080C7067BA1),
]
coclass CCollie {
    [default] interface _CCollie;
};
```

Wenn andere Programmierer die CCollie-Komponente in einer Client-Applikation benutzen möchten, müssen sie die Typlibibliothek des Servers mit Hilfe des Visual Basic-Dialogfeldes *Verweise* referenzieren. Sobald dies geschehen ist, können sie den folgenden Code schreiben, um eine Instanz der CCollie-Komponente zu aktivieren und darauf zuzugreifen.

```
Dim Dog As CCollie
Set Dog = New CCollie
```

```
Dog.Bark
Dog.RollOver 14
Set Dog = Nothing
```

Jede COM-basierte Verbindung zwischen einem Client und einem Objekt muss auf einer spezifischen IID basieren. Wenn ein Visual Basic-Client `CCollie` referenziert, wird diese Referenz automatisch an die Standardschnittstelle hinter der Komponente weitergeleitet. In unserem Beispiel wird die Verbindung über die IID der Standardschnittstelle für `CCollie` hergestellt. Diese IID wird in jede Client-Applikation kompiliert, die für Version 1 der DLL erstellt wurde.

Nach dem Testen und Debuggen Ihres Codes installieren Sie *DOGSERVER.DLL* in eine COM+-Applikation auf Ihrem Betriebsserver. Nachdem Sie alle Einstellungen korrekt vorgenommen haben, benutzen Ihre Anwender Client-Applikationen, um über das Netzwerk auf die `CCollie`-Komponente zuzugreifen. Version 1.0 der Client-Applikation und der Server arbeiten synchron zusammen. Alles ist gut.

Gehen wir nun davon aus, dass Sie eine zweite Version Ihrer DLL erstellen und eine neue Methode namens `FetchSlippers` zu `CCollie` hinzufügen möchten. Wenn Sie im Modus *Binär-Kompatibilität* versuchen, die DLL erneut zu kompilieren, wird der Visual Basic-Compiler feststellen, dass die Standardschnittstelle kompatibel, aber nicht identisch mit der vorherigen Version ist. Er erstellt somit eine zweite IID für die neue Version der Standardschnittstelle. Nachfolgend ist der IDL-Code des zweiten Builds aufgeführt:

```
[ // Andere GUID für die Standardschnittstelle _ IID2
  uuid(C44FF454-E9A0-11D1-9266-0080C72D2182),
  version(1.1), hidden, dual, oleautomation
]
interface _CCollie : IDispatch {
    HRESULT Bark();
    HRESULT RollOver([in, out] int* Rolls);
    HRESULT FetchSlippers();
};

[ // Diese GUID ist die CLSID.
  uuid(EDE2823B-DE19-11D2-9A2C-0080C7067BA1),
]
coclass CCollie {
    [default] interface _CCollie;
};
```

Die in Fettschrift aufgeführten Elemente werden von Visual Basic vom ersten zum zweiten Build verändert. Visual Basic erstellt eine versionskompatible Komponente. Die Schnittstellendefinition für die erste IID ist nicht in der Typbibliothek enthalten. Wenn Sie jedoch *OLEVIEW* benutzen, können Sie einen weiteren neuen Eintrag in der Typbibliothek des Servers finden, der zurück auf die alte IID verweist.

```
typedef [
    uuid(C44FF440-E9A0-11D1-9266-0080C72D2182),
    version(1.0), public
]
_CCollie CCollie__v0;
```

Dieser Eintrag hilft Visual Basic, eine Unterstützung für die alte IID in späteren Builds zur Verfügung zu stellen. Wie Sie sehen, ordnet Visual Basic die alte IID einfach der neuen IID zu. Alte Clients können sich erfolgreich an neuere Versionen des Objekts binden. Clients, die für die neuere Version der DLL kompiliert werden, arbeiten ebenfalls problemlos, da sie zur Bindung die neue IID verwenden. Versionskompatible Schnittstellen funktionieren nur dann nicht, wenn ein neuer Client versucht, eine Bindung zu einer alten Version des Objekts mit der neuen IID herzustellen. Wir prüfen im nächsten Abschnitt, wie dieses Problem gelöst werden kann. Tabelle 5.1 gibt Aufschluss über die möglichen Kombinationen.

Client-Version	Komponentenversion	Verwendete IID-Version	Kombination möglich?
Alter Client	Alte Komponente	Alte IID	Ja
Alter Client	Neue Komponente	Alte IID	Ja
Neuer Client	Neue Komponente	Neue IID	Ja
Neuer Client	Alte Komponente	keine	Nein

Tabelle 5.1: *Kompatibilität zwischen den verschiedenen Versionen von Client und Komponente*

Es ist offensichtlich, dass die Visual Basic-Unterstützung für COM entworfen wurde, um einfach und unkompliziert zu sein. Diese einfache Verwendung beeinträchtigt jedoch die Flexibilität. Sie können die binäre Kompatibilität benutzen, um alle GUIDs, einschließlich der IIDs, über Builds hinweg beizubehalten und zuvor kompilierte Clients zu unterstützen, die die direkte vTable-Bindung benutzen. Die binäre Kompatibilität ermöglicht ebenfalls das Erstellen von versionskompatiblen Schnittstellen. Wie Sie jedoch wissen, garantiert dies keine vollständige Kompatibilität.

Wenn die binäre Kompatibilität nicht genügt

Wie Sie gesehen haben, kann Visual Basic Ihre Schnittstellen für Sie veröffentlichen und deren Versionen verwalten. Dies geschieht im Hintergrund. Wenn Sie jedoch bereit sind, ein wenig mehr involviert zu werden, steht Ihnen eine andere Option zur Verfügung: Sie können Ihre Schnittstellen mit IDL definieren und in eine benutzerdefinierte Typbibliothek kompilieren. Diese Möglichkeit bietet Ihnen eine weitaus umfassendere Kontrolle und konfrontiert Sie nicht mit einigen der Probleme, die auftreten können, wenn Sie die binäre Kompatibilität verwenden.

Der Einsatz von IDL verlangt von den Komponenteautorenn und den clientseitigen Programmierern, unter dem Aspekt benutzerdefinierter Schnittstellen zu arbeiten. Sie können eine benutzerdefinierte Schnittstelle nicht als Standardschnittstelle hinter einer MultiUse-Klasse konfigurieren. Dies bedeutet, dass Sie die Prinzipien der schnittstellenbasierten Programmierung verstehen und zwischen konkreten und abstrakten Typen unterscheiden müssen. Es ist bei diesem Programmierstil sehr aufwendig, immer auf dem Laufenden zu bleiben. Die Vorteile des Definierens von Schnittstellen mit IDL gleichen diesen Aufwand jedoch in vielen Fällen wieder aus. Dies gilt besonders für große Projekte, in denen die Unzulänglichkeiten der binären Kompatibilität mehr als offenkundig sind.

Was sind die signifikantesten Probleme, die mit dem exklusiven Programmieren der Standardschnittstelle Ihrer Komponente und dem Verwenden des binären Kompatibilitätsschemas von Visual Basic verknüpft sind? Zunächst verlieren Sie einen Teil der Kontrolle über die Verwaltung Ihrer Schnittstellendefinitionen, Enumerationen und UDTs. Zweitens verlieren Sie die Möglichkeit, die Polymorphie zu nutzen und Plug-kompatible Komponenten zu erstellen. Schließlich können Ihre Clients keine älteren Versionen Ihres Servers verwenden. Lassen Sie uns diese Probleme ein wenig detaillierter betrachten:

Sie verlieren die Kontrolle über die Definition und Verwaltung der Elemente, die in Ihre Typbibliothek aufgenommen werden, da Visual Basic alle Arbeiten für Sie im Hintergrund ausführt. Sie können lediglich die Versionskompatibilitätseinstellung für das gesamte Projekt verändern. Wenn Sie jedoch direkt mit IDL arbeiten, um Ihre Schnittstellen getrennt von Ihren Servern und Komponenten zu definieren, können Sie entscheiden, wann Ihre IIDs geändert, beibehalten und veröffentlicht werden sollen. Dies ist dann für jede Schnittstelle möglich.

Wenn Sie auf die Standardschnittstelle hinter jeder Ihrer `MultiUse`-Klassen angewiesen sind, können Sie keine Plug-kompatiblen Komponenten erstellen, da eine beschränkende Eins-zu-eins-Beziehung zwischen Ihren Komponenten und Schnittstellen besteht. Wenn Sie eine Applikation basierend auf einem polymorphen Design mit Plug-kompatiblen Komponenten entwerfen möchten, müssen Sie mit benutzerdefinierten Schnittstellen arbeiten. (Sie erzielen die Polymorphie ebenfalls mit Hilfe der späten Bindung. Lassen Sie uns jedoch davon ausgehen, dass Sie mit Clients arbeiten möchten, die die direkte `vTable`-Bindung benutzen.) Sobald Sie eine benutzerdefinierte Schnittstelle erstellt haben, können Sie diese in verschiedene Komponenten implementieren.

Wenn Sie die binäre Kompatibilität nutzen, verzichten Sie außerdem auf einen der wesentlichen Vorzüge der schnittstellenbasierten Programmierung. Ein neuer Client kann nicht so variiert werden, dass er eine ältere Version der Komponente verwendet. Dies bedeutet, dass Sie jeden Server in der Betriebsumgebung ersetzen müssen, wenn Sie eine neue Version einer Client-Applikation einsetzen möchten.

Lassen Sie mich Ihnen ein wenig mehr Hintergrundwissen vermitteln, so dass Sie diesen letzten Punkt auch wirklich verstehen. Wie Sie wissen, erstellt Visual Basic (im binären Kompatibilitätsmodus) immer dann eine neue IID, wenn Sie eine oder mehrere Methoden zur neuen Version einer Komponente hinzufügen. Die Komponente unterstützt sowohl die alte IID als auch die neue. In dem neuen Build der Typbibliothek wird jedoch nur die Schnittstellendefinition für die neue IID veröffentlicht. Wenn ein alter Client eine Verbindung basierend auf der alten IID anfordert, wird das `vTable`-Layout anhand der neuen IID generiert. Dies ist möglich, weil die `vTable` hinter der neuen IID eine Obermenge der `vTable` der alten IID ist. Die neuere `vTable` ist somit versionskompatibel mit der älteren.

Sie wissen bereits, wie Visual Basic auf der physikalischen Ebene mit versionskompatiblen Schnittstellen arbeitet. Lassen Sie uns deshalb nun eine höhere Ebene betrachten, um einen Eindruck davon zu erhalten, was alles missglücken kann. Als Sie die erste Version von `CCollie` mit zwei öffentlichen Methoden kompiliert haben, hat Visual Basic eine Standardschnittstelle namens `_CCollie` veröffentlicht. Diese Standardschnittstelle hat eine spezifische IID erhalten, die wir für unser Beispiel IID1 nennen. Als Sie eine weitere Methode mit dem Namen `FetchSlippers` hinzugefügt und den Server im binären Kompatibilitätsmodus erneut erstellt haben, hat Visual Basic eine zweite IID für die neue Schnittstellendefinition mit allen drei Methoden generiert. Lassen Sie uns diese IID2 nennen. Wenn eine Visual Basic-Client-Applikation die DLL referenziert, die die neueste Version der Komponente enthält, weiß sie lediglich von IID2. Die Applikation kann nicht eine Überprüfung für IID2 durchführen und dann einfach IID1 verwenden, wenn sie auf eine DLL trifft, die die ursprüngliche Version der Komponente enthält. Dies kann zu ernsthaften Problemen führen. Der Client kennt lediglich die IID der Standardschnittstelle, diese kann sich jedoch mit jedem neuen Build der DLL ändern.

Ist dies wirklich ein Problem? Was geschieht, wenn Sie Version 1 der DLL zusammen mit einer Client-Applikation veröffentlichen? Stellen Sie sich vor, dass die DLL in eine COM+-Applikation installiert wurde. Stellen Sie sich weiter vor, dass Sie ein Upgrade herausbringen, das sowohl eine neue Version der DLL als auch eine neue Version des Clients enthält. Während des Upgra-

de-Vorgangs werden die Clients aktualisiert. Aufgrund einer unerwarteten Verzögerung wird die DLL auf dem Server jedoch nicht aktualisiert. Wenn die Benutzer nun versuchen, ein Objekt zu aktivieren, erhalten Sie die Fehlermeldung 430 »Klasse unterstützt keine Automatisierung oder unterstützt erwartete Schnittstelle nicht.« Etwas sollte jetzt deutlich geworden sein: Sie sollten versionskompatible Schnittstellen nur dann verwenden, wenn Sie garantieren können, dass neue Clients nur zusammen mit der letzten Version der DLL veröffentlicht werden.

Wenn Sie Ihre neuen Clients auf ältere Versionen Ihrer Komponenten umstellen können möchten, müssen Sie mit benutzerdefinierten Schnittstellen arbeiten. Stellen Sie sich zum Beispiel vor, Sie definieren eine Schnittstelle mit der Bezeichnung `IDog` und implementieren diese in die Originalversion der `CCollie`-Komponente. In der zweiten Version möchten Sie eine weitere Methode hinzufügen. Sie definieren deshalb eine zweite Schnittstelle namens `IDog2`, die alle Methoden von `IDog` und die neue Methode enthält. Wenn Sie `IDog2` zusätzlich zu `IDog` in die zweite Version Ihrer Komponente implementieren, können Sie den Client-Code wie nachfolgend aufgeführt schreiben:

```
Dim Ref1 As IDog
Set Ref1 = New CCollie
' Prüfen, ob Komponente IDog2 unterstützt.
If TypeOf Ref1 Is IDog2 Then
    ' Zu neuer IID umleiten.
    Dim Ref2 As IDog2
    Set Ref2 = Ref1
    ' Über Ref2 (neue IID) auf Objekt zugreifen.
Else
    ' Wechsel zur nächstniedrigeren Ebene.
    ' Über Ref1 (alte IID) auf Objekt zugreifen.
End If
```

Dies ist ein Beispiel für einen Client-Code, der besser auf das Schema abgestimmt werden kann, das von der binären Kompatibilität verwendet wird. Die Arbeit mit benutzerdefinierten Schnittstellen bietet für große Geschäftssysteme, die ständig verändert werden, sehr viele Vorteile (wie im zweiten Kapitel beschrieben).

Benutzerdefinierte Schnittstellen erstellen

Bevor wir uns dem nächsten Thema zuwenden, möchte ich noch einmal einen Schlüsselpunkt erwähnen: Wenn Sie benutzerdefinierte Schnittstellen verwenden, müssen Sie voraussetzen, dass alle Clients die direkte `vTable`-Bindung benutzen, um auf Objekte zuzugreifen, die aus Ihrer Komponente erzeugt werden.

Im vorherigen Kapitel wurde bereits beschrieben, dass benutzerdefinierte Schnittstellen nicht gut mit Scripting-Clients zusammenarbeiten. Sie sollten über öffentliche Methoden in der `MultiUse`-Klasse auf die Scripting-Clients zugreifen. Wir gehen für den Rest dieses Kapitels davon aus, dass unser gesamter Client-Code in Visual Basic oder mit einem anderen Werkzeug geschrieben wird, das die direkte `vTable`-Bindung und das Navigieren zwischen den verschiedenen von einem Objekt angebotenen Schnittstellen unterstützt.

Sie haben zwei Auswahlmöglichkeiten für die Erstellung von benutzerdefinierten Schnittstellen. Sie können Ihre Schnittstellen mit Visual Basic und `PublicNotCreatable`-Klassenmodulen oder mit IDL definieren. Lassen Sie uns mit der einfacheren Möglichkeit – mit Visual Basic – beginnen.

Visual Basic lässt das Erstellen von benutzerdefinierten Schnittstellen sehr einfach werden. Sie haben die hierfür notwendigen Grundlagen im zweiten Kapitel kennen gelernt. Sie müssen jedoch weiterhin entscheiden, ob Ihre Schnittstellen in demselben Serverprojekt wie Ihre Komponenten definiert werden sollen. Eine andere Option besteht darin, eine alleinstehende Typbibliothek mit Schnittstellendefinitionen zu generieren, die von Ihren Servern getrennt wird.

In einem kleineren Projekt ist es nicht immer sinnvoll, Schnittstellendefinitionen in einer separaten Typbibliothek zu veröffentlichen. Dies gilt besonders dann, wenn nur ein kleines Team oder ein einzelner Entwickler an dem Projekt arbeitet. Es ist einfacher, die Schnittstellen in dem Serverprojekt zu definieren, in dem sich auch Ihre Komponenten befinden. Wenn sich für Ihr Projekt keine Vorteile aus der getrennten Wartung von Schnittstellen und Komponenten ergeben, steht der administrative Aufwand des Erstellens von separaten Dateien möglicherweise in keinem Verhältnis zu den Schwierigkeiten, die sich daraus ergeben können.

Für größere Projekte ist es hingegen sinnvoll, benutzerdefinierte Schnittstellen in einer gesonderten Typbibliothek zu veröffentlichen. Diese Vorgehensweise vereinfacht den Einsatz derselben Schnittstellendefinition für mehrere Server. Was geschieht zum Beispiel, wenn Sie zwei ActiveX-DLLs erstellen möchten, die je eine Komponente enthalten, und beide Komponenten dieselbe Schnittstelle implementieren? Wenn ein Entwickler eine Schnittstelle in einer gesonderten Typbibliothek veröffentlicht, können zwei voneinander unabhängige DLL-Autoren diese Bibliothek einfach referenzieren und die Schnittstelle implementieren. Mehrere COM-Server können somit Objekte bedienen, die typkompatibel sind. Dies ist besonders dann sehr nützlich, wenn ein System auf Plug-kompatiblen Komponenten basiert.

Das Erstellen einer alleinstehenden Typbibliothek, die Schnittstellendefinitionen enthält, ist unter Visual Basic ziemlich einfach, aber auch ein wenig umständlich. Sie erstellen zunächst ein ActiveX-DLL-Projekt und definieren Ihre Schnittstellen in Klassenmodulen, die als `PublicNotCreatable` gekennzeichnet sind. Sie selektieren anschließend im Dialogfeld *Projekteigenschaften* auf der Registerkarte *Komponente* das Kontrollkästchen *Remote-Server-Dateien*, um sicherzustellen, dass eine alleinstehende Typbibliothek (mit einer *.TLB*-Erweiterung) generiert wird, sobald Sie die DLL erstellen. Sie können die DLL dann löschen und die *.TLB*-Datei an andere Programmierer weitergeben.

Das Umständliche bei der Verwendung der Visual Basic-IDE ist, dass Sie mindestens eine erstellbare Klasse in jedes ActiveX-DLL-Projekt einbinden müssen. Die Visual Basic-IDE geht davon aus, dass Sie lediglich Server erstellen möchten. Sie müssen deshalb einen Trick anwenden, um eine Typbibliothek zu erstellen, die nur Schnittstellen enthält. Sie erzeugen dazu ein Klassenmodul ohne Inhalt, dessen *Instancing*-Eigenschaft auf *MultiUse* gesetzt wird. Dies ist etwas verwirrend, da jeder Benutzer, der Ihre Typbibliothek verwenden wird, diese Klasse ignorieren muss. Diese Vorgehensweise bietet jedoch die einzige Möglichkeit, eine *.TLB*-Datei mit der Visual Basic-IDE zu generieren.

Schnittstellen mit IDL definieren

Wenngleich Sie benutzerdefinierte Schnittstellen in einem Visual Basic-Projekt mit Hilfe von `PublicNotCreatable`-Klassenmodulen erstellen können, verfügen Sie über eine weitaus größere Kontrolle, wenn Sie direkt mit IDL arbeiten. Sobald Sie Ihre Schnittstellendefinitionen in einer IDL-Sourcedatei erstellt haben, können Sie eine benutzerdefinierte Typbibliothek mit dem MIDL-Compiler generieren.

IDL hat eine etwas verwirrend anmutende Geschichte. Die Sprache wurde ursprünglich entwickelt, um RPC-Schnittstellen zu definieren. Die in IDL vorgenommenen RPC-Schnittstellendefi-

nitionen können an einen Compiler übergeben werden, um Code zu generieren, der Funktionsaufrufe über das Netzwerk zu einem entfernten Ziel weiterleitet. Microsoft hat IDL für COM-Schnittstellen erweitert, indem Attribute und einige objektorientierte Features hinzugefügt wurden. Der MIDL-Compiler kann darüber hinaus Fernverarbeitungscode für COM-Schnittstellen erzeugen.

Einige Jahre zuvor verwendeten COM-Programmierer zwei unterschiedliche Sprachen, um Komponenten und Schnittstellen zu beschreiben: IDL und ODL (Object Definition Language). Sie benutzten IDL, um den Fernverarbeitungscode zu erstellen, und ODL, um Typbibliotheken zu generieren. Die Programmierer verwendeten außerdem ein älteres Hilfsmittel namens *MKTYPLIB.EXE*, um eine ODL-Datei in eine Typbibliothek zu kompilieren.

Mit der Veröffentlichung von Windows NT 4 fasste Microsoft die ursprünglichen Versionen von IDL und ODL in der damals aktuellen Version von IDL zusammen. Sie verwenden den MIDL-Compiler, um sowohl die Typbibliotheken als auch den Fernverarbeitungscode zu erstellen. IDL und die neueste Version von MIDL sollten anstelle von ODL und *MKTYPLIB* verwendet werden, wenn Sie Typbibliotheken erzeugen möchten. Beachten Sie außerdem, dass Sie die immer die letzte Version von MIDL, die von dem Platform-SDK zur Verfügung gestellt wird, benutzen sollten. Frühere Versionen von MIDL unterstützen nicht jedes der neuesten Features von IDL. (Die Beispiele dieses Kapitels basieren auf MIDL 5.03.0280).

Sie sollten nun erfahren, wie eine Visual Basic-kompatible Typbibliothek mit IDL erstellt wird. Sie entscheiden zunächst, welche Definitionstypen Sie in Ihrer Typbibliothek veröffentlichen möchten. Sie können Schnittstellen, Enumerationen und UDTs zu Ihrer IDL-Sourcdatei hinzufügen. Co-Klassen, die in IDL definiert werden, führen jedoch zu einem Problem. Obwohl Sie eine Co-Klasse in IDL definieren und in eine benutzerdefinierte Typbibliothek kompilieren können, können Sie die Co-Klassendefinition nicht aus einem Visual Basic-Projekt heraus verwenden. Die einzigen von Visual Basic verwendeten Co-Klassendefinitionen sind die, die die Programmiersprache automatisch in den mit den Servern verknüpften Typbibliotheken erstellt. Dies bedeutet, dass Sie keinen IDL-Code schreiben können, um Einfluss darauf zu nehmen, wie Visual Basic eine Co-Klasse definiert.

Sie sollten eine IDL-Sourcdatei für jede Typbibliothek benutzen, die Sie erstellen möchten. Sie können diese IDL-Sourcdatei mit einem einfachen Texteditor, wie zum Beispiel Notepad, erstellen und modifizieren. Wenn Sie Ihre IDL-Dateien in der Visual C++-Entwicklungsumgebung bearbeiten, wird die Syntax des Sourcecodes farbig hervorgehoben. Dies kann sehr angenehm sein, wenn Sie eine neue Sprache lernen.

Wenn Sie eine Typbibliothek mit der Bezeichnung *DOGLIBRARY.TLB* generieren möchten, sollten Sie damit beginnen, eine IDL-Sourcdatei namens *DOGLIBRARY.IDL* zu erstellen. Nachfolgend ist das Anfangsgerüst für diese IDL-Sourcdatei aufgeführt:

```
// DOGLIBRARY.IDL
[
    uuid(46373B81-4106-11d3-AB39-2406D0000000),
    helpstring("The Dog App Type Lib"),
    version(1.0)
]
library DogLibrary {
    importlib("STDOLE2.TLB");
    // Definitionen für Enumerationen und UDTs werden hier angeordnet.
    // Schnittstellendefinitionen werden hier angeordnet.
};
```

Diese Vorlage enthält den Grundcode für die Definition der Attribute einer Typbibliothek. Sie erweitern diese Vorlage, indem Sie Definitionen zum Bibliotheksabschnitt hinzufügen. Sie sollten beachten, dass der Bibliotheksabschnitt ein `uuid`-Attribut (eine GUID) erfordert. Mit Visual Studio wird ein Hilfsmittel mit der Bezeichnung *GUIDGEN.EXE* ausgeliefert, das das Generieren von GUIDs vereinfacht. Sie können die generierten GUIDs außerdem in die Zwischenablage kopieren und anschließend in Ihre IDL-Sourcdatei einfügen. Benutzen Sie jedoch das Format der Registrierung, wenn Sie Ihre GUIDs kopieren. Entfernen Sie dazu einfach die Zeichen { und }, die von GUIDGEN an der GUID angeordnet werden.

Der Bibliotheksabschnitt enthält zwei optionale Attribute. Das `[helpstring]`-Attribut dient als Beschreibung der Typbibliothek. Visual Basic-Programmierer sehen diese Beschreibung, wenn sie sich die Typbibliothek im *Verweise*-Dialogfeld anzeigen lassen. Sie können außerdem ein `[version]`-Attribut verwenden.

Der Vorlagencode der Bibliothek importiert außerdem eine Typbibliothek namens *STDOLE2.TLB*. Sie müssen diese Bibliothek importieren, da sie COM-Standardschnittstellen, wie zum Beispiel `IUnknown` und `IDispatch`, definiert. Sie markiert `IUnknown` darüber hinaus als `[hidden]` und `IDispatch` als `[restricted]`. Die Methoden beider Schnittstellen werden mit `[restricted]` gekennzeichnet. Der Visual Basic-Compiler und die -Laufzeit sind darauf angewiesen, dass diese Methoden mit `[restricted]` markiert sind. Es ist deshalb wichtig, dass Sie *STDOLE2.TLB* in jede Visual Basic-kompatible Bibliothek importieren, die Sie erstellen. Sie müssen möglicherweise auch andere Typbibliotheken importieren, wenn die Definitionen in Ihrer IDL-Sourcdatei externe Typen, wie zum Beispiel ADO-Datensatzgruppen, verwenden.

Lassen Sie uns nun eine Schnittstelle mit IDL definieren. Nachfolgend ist ein geeigneter Ausgangspunkt für eine Visual Basic-kompatible Schnittstellendefinition aufgeführt:

```
[
    uuid(A0E89184-40BE-11d3-AB39-2406D0000000),
    oleautomation,
    object
]
interface IDog : IUnknown {
    // Methodensignaturen
};
```

Die Schnittstelle enthält drei wichtige Attribute. Das `[uuid]`-Attribut muss angegeben werden. Es bildet die IID der Schnittstelle. Sie können die IID, wie jede andere GUID auch, mit dem *GUIDGEN*-Programm generieren. Achten Sie außerdem darauf, dass die Schnittstelle mit dem `[oleautomation]`-Attribut definiert wird. Dieses Attribut lässt lediglich Datentypen in der Schnittstelle zu, die mit Visual Basic kompatibel sind. Es weist die COM-Laufzeit außerdem an, den Universal-Marshaler zum Erstellen von Proxy-/Stub-Code zu verwenden. Das `[object]`-Attribut informiert den MIDL-Compiler schließlich darüber, dass die Schnittstelle eine COM-Schnittstelle und keine RPC-Schnittstelle ist. Ältere Versionen des MIDL-Compilers verlangen nicht die Angabe des `[object]`-Attributs, wenn eine Schnittstelle in einem Typbibliotheksgerüst definiert wird.

Beachten Sie, dass die Schnittstelle von `IUnknown` abgeleitet wird. Visual Basic-kompatible Schnittstellen müssen direkt von `IUnknown` oder `IDispatch` abgeleitet werden. Eine Schnittstelle, die von einer anderen benutzerdefinierten Schnittstelle abgeleitet wird, ist nicht kompatibel mit Visual Basic. Was geschieht beispielsweise, wenn Sie eine benutzerdefinierte Schnittstelle mit der Bezeichnung `IDog2` von einer anderen benutzerdefinierten Schnittstelle namens `IDog` ableiten? Betrachten Sie zur Beantwortung dieser Frage die folgenden beiden Schnittstellendefinitionen:

```
// Kann in eine VB-Klasse implementiert werden
interface IDog : IUnknown {
    HRESULT Bark();
    HRESULT RollOver([in, out] long* Rolls);
};

// Kann nicht in eine VB-Klasse implementiert werden
interface IDog2 : IDog {
    HRESULT FetchSlippers();
};
```

Sie können diese Version von IDog2 nicht in eine Visual Basic-Klasse implementieren, da sie von IDog abgeleitet ist. Visual Basic unterstützt nur eine Ebene der Schnittstellenvererbung. Wenn Sie IDog2 in eine Visual Basic-Klasse implementieren möchten, muss sie von IUnknown oder IDispatch abgeleitet sein.

Ungeachtet der fehlenden Visual Basic-Unterstützung für mehrere Ebenen der Schnittstellenvererbung, werden Sie oft eine Schnittstelle generieren wollen, die die Obermenge einer anderen Schnittstelle bildet. Wie gehen Sie beispielsweise vor, wenn Sie die Funktionalität einer Schnittstelle erweitern möchten? Nachfolgend finden Sie den IDL-Code, der eine Lösung für dieses Problem präsentiert:

```
interface IDog2 : IUnknown {
    HRESULT Bark();
    HRESULT RollOver([in, out] long* Rolls);
    HRESULT FetchSlippers();
};
```

Sie müssen die Methodendefinitionen in beiden Versionen der Schnittstelle duplizieren. Aus der Perspektive der objektorientierten Programmierung betrachtet, ist dies ist keine elegante Lösung, führt aber zu dem gewünschten Ergebnis. Sobald Sie IDog und IDog2 auf diese Weise definiert haben, können Sie beide in Version 2 der CCollie-Komponente implementieren, wie nachfolgend gezeigt:

```
Implements IDog
Implements IDog2
```

Jede Schnittstelle erfordert unglücklicherweise eine separate Gruppe mit Einstiegspunkten in das Klassenmodul für jede duplizierte Methode. Beide Schnittstellen definieren zum Beispiel eine Methode namens Bark. Sie werden jedoch in den meisten Fällen beide Einstiegspunkte für Bark an eine einzelne Methodenimplementierung weiterleiten wollen. Dies kann zu einer lästigen und langwierigen Routinearbeit werden. Sie müssen Ihre Klasse modifizieren, so dass sie der nachfolgend aufgeführten gleicht:

```
Public Sub IDog_Bark()
    ' Implementation code
End Sub

Public Sub IDog2_Bark()
    ' Forward call
    Call IDog_Bark
End Sub
```

Sie sollten ebenfalls beachten, dass IDog und IDog2 von IUnknown anstelle von IDispatch abgeleitet sind. Der Grund hierfür ist einfach. Sie müssen nur dann von IDispatch ableiten, wenn Sie eine duale Schnittstelle zur Unterstützung von Scripting-Clients erstellen möchten. Scripting-Clients sind jedoch immer über die Standardschnittstelle mit Objekten verbunden. Scripting-Clients können außerdem nicht von einer Schnittstelle zu einer anderen navigieren. Ein Scripting-Client kann nicht auf eine Methode in einer benutzerdefinierten Schnittstelle zugreifen. Dies bedeutet, dass die in IDL definierten Schnittstellen lediglich von Clients verwendet werden können, die die direkte vTable-Bindung benutzen. Das Ableiten von IUnknown ist alles, was Sie tun müssen, und es erfordert weniger Aufwand als das Ableiten von IDispatch.

Methodensignaturen mit IDL definieren

Lassen Sie uns nun betrachten, wie die eigentlichen Methoden in einer Schnittstelle definiert werden. Sie sollten lernen, wie der Typ und die Richtung jedes Parameters definiert wird. Stellen Sie sich vor, Sie benötigen eine Schnittstelle mit einigen Methoden, die der nachfolgend aufgeführten Schnittstelle gleicht:

```
Sub Test1(ByVal i As Long)
Sub Test2(ByRef i As Long)
Function Test3() As Long
```

Der erforderliche IDL-Code ist dann wie folgt aufgebaut:

```
HRESULT Test1([in] long i);
HRESULT Test2([in, out] long* i);
HRESULT Test3([out, retval] long*);
```

Die Leser unter Ihnen, die nicht mit C vertraut sind, sollten wissen, dass das Zeichen * einen Parameter definiert, der mit einem Zeiger übergeben wird. Sie müssen diese Zeigersyntax immer dann verwenden, wenn Sie Ausgabeparameter vom Objekt zum Client übergeben möchten. Sie sollten ByRef-Parameter mit Hilfe der Zeigersyntax und dem [in, out]-Attribut definieren. Ein Funktionsrückgabewert sollte mit der Zeigersyntax sowie dem [out, retval]-Attribut und als äußerster rechter Parameter definiert werden.

IDL stellt Äquivalente für alle gebräuchlichen VBA-Typen einschließlich String, Date und Variant, zur Verfügung. Tabelle 5.2 zeigt die Zuordnung von Visual Basic- zu IDL-Datentypen.

Visual Basic-Datentyp	IDL-Datentyp
Integer	short
Long	long
Byte	unsigned char
Single	float
Double	double
Boolean	VARIANT_BOOL
String	BSTR
Variant	VARIANT
Currency	CURRENCY

Visual Basic-Datentyp	IDL-Datentyp
Date	Date
Array	SAFEARRAY
Object	*IDispatch

Tabelle 5.2: Zuordnung zwischen Visual Basic- und IDL-Datentypen

Visual Basic-Arrays sind so genannte `SAFEARRAYs` in IDL. Ein `SAFEARRAY` ist eine Datenstruktur mit vielen verknüpften Metadaten. Ein Array kann als `ByRef`-Parameter oder als Funktionsrückgabewert übergeben werden. Beachten Sie, dass der Universal-Marshaler die Inhalte eines `SAFEARRAYs` mit einer einzigen Übertragung über das Netzwerk bewegen kann. Nachfolgend sind zwei Visual Basic-Methoden aufgeführt. Die erste Methode empfängt ein Array und die zweite Methode gibt ein Array zurück:

```
Sub Test4(ByRef x() As Long)
Function Test5() As Long()
```

Die entsprechenden IDL-Methoden lauten wie folgt:

```
HRESULT Test4([in, out] SAFEARRAY(long)* x);
HRESULT Test5([out, retval] SAFEARRAY(long)* );
```

Doch wie gehen Sie vor, wenn Sie ein Objekt übergeben möchten? Nun, Sie sollten zunächst wissen, dass Sie eine Objektreferenz anstelle des eigentlichen Objekts übergeben. Auf der physikalischen Ebene übergeben Sie einen Zeiger auf die `vTable`, die mit einer Schnittstelle, wie zum Beispiel `IDog`, verknüpft ist. Dies bedeutet, dass Objektreferenzen immer als Zeiger übergeben werden. Ein `ByVal`-Parameter ermöglicht, dass der Zeiger vom Client zum Objekt übergeben werden kann, wie im folgenden IDL-Code dargestellt:

```
HRESULT Test6([in] IDog* Dog);
```

Ein `ByRef`-Parameter oder ein Funktionsrückgabewert, der auf einer Objektreferenz basiert, muss als Zeiger übergeben werden, der auf einen anderen Zeiger verweist. Dies bedeutet, dass Sie zwei Multiplikationszeichen (*) benötigen. Betrachten Sie bitte die folgenden Methodendefinitionen:

```
HRESULT Test7([in, out] IDog** Dog);
HRESULT Test8([out, retval] IDog**);
```

Der Code, den Sie unter Visual Basic schreiben, um diese Methoden zu implementieren, müsste wie folgt aufgebaut sein:

```
Sub Test6(ByVal Dog As IDog)
Sub Test7(ByRef Dog As IDog)
Function Test8() As IDog
```

Sie erfahren mehr über das Schreiben von Methodendefinitionen in IDL, indem Sie die Online-MIDL-Dokumentation lesen, die mit MSDN (Microsoft Developer Network) ausgeliefert wird. Es gibt jedoch eine nützliche Abkürzung, die Sie sehr viel schneller durch die IDL-Lernkurve führt. Sie beginnen mit der Definition einiger Methodensignaturen in einem Visual Basic-Klas-

senmodul. Kompilieren Sie Ihren Code dann in eine ActiveX-DLL und verwenden Sie *OLEVIEW*, um die in der DLL generierte Typlibibliothek zu dekompile. Sie können anschließend die Methodensignaturen aus dem *Type Library Viewer* von *OLEVIEW* kopieren und direkt in eine IDL-Schnittstellendefinition einfügen.

Sobald Sie die Methodendefinitionen in Ihre IDL-Sourcdatei eingefügt haben, sollten Sie die `[id]`-Attribute entfernen, da diese nur dann benötigt werden, wenn Ihre Clients über `IDispatch` verbunden werden. Sie können außerdem ein `[helpstring]`-Attribut zu jeder Methodendefinition hinzufügen, um deren Semantik für andere Programmierer zu dokumentieren.

Achten Sie darauf, dass ein gewisser Teil der Syntax, die sich aus der Dekompilierung einer Typlibibliothek aus einem Visual Basic-Server ergibt, nicht immer funktioniert. Der Grund hierfür besteht darin, dass Visual Basic Typlibibliotheken unter Verwendung älterer ODL-Datentypen erstellt. Auch *OLEVIEW* generiert gelegentlich ODL- anstelle von IDL-Syntax. Wenn Sie eine Typlibibliothek dekompile, wird nicht die gesamte ODL-Syntax korrekt vom MIDL-Compiler kompiliert. Dies geschieht nicht oft, ist aber, wenn es passiert, sehr frustrierend. Die beiden häufigsten Probleme treten auf, wenn Sie Single-Datentypen und UDT-Definitionen verwenden.

Der Single-Datentyp wird von ODL, aber nicht von IDL definiert. Wenn Sie Code kopieren und einfügen, der von *OLEVIEW* produziert wurde, sollten Sie den Single-Datentyp durch dessen IDL-Äquivalent *Float* ersetzen. Die dazu notwendige Vorgehensweise wird beschrieben, wenn erklärt wird, wie UDT-Definitionen erstellt werden. Merken Sie sich bitte zunächst, dass Sie nicht einfach UDT-Definitionen aus einem dekompileten Code kopieren und in Ihre eigenen IDL-Sourcdateien einfügen können.

Sobald Sie IDL beherrschen, können Sie die Signaturen manuell erstellen oder modifizieren. Achten Sie jedoch darauf, die nachstehend aufgeführten Regeln zu befolgen:

- Alle Methoden müssen `HRESULTS` zurückgeben.
- Markieren Sie `ByVal`-Parameter mit `[in]` und `ByRef`-Parameter mit `[in, out]`.
- Sie können die mit `[in]` gekennzeichneten Parameter nicht als Zeiger definieren.
- Ein Parameter, der mit `[out]` gekennzeichnet ist, muss ebenfalls mit `[retval]` markiert werden und der äußerst rechte Parameter sein.
- Methodennamen können nicht mit einem Unterstrich (`_`) beginnen.
- Verwenden Sie keine Integer-Werte ohne Vorzeichen als Parameter.

Enumerationen und UDTs verwenden

Da Sie nun wissen, wie eine Schnittstelle in einer Typlibibliothek definiert wird, sollten Sie ebenfalls erfahren, wie Enumerationen und UDT-Definitionen hinzugefügt werden. Achten Sie darauf, dass Sie einen Typ definieren müssen, bevor Sie diesen in einer IDL-Datei benutzen. Stellen Sie sich zum Beispiel vor, Sie möchten einen UDT und eine Methode in einer Schnittstelle definieren, die den UDT als Parameter verwendet. Wenn Sie die Schnittstelle vor dem UDT definieren, wird die IDL-Datei nicht kompiliert. Wenn Sie all Ihre Enumerationen und UDTs vor der Schnittstelle definieren, können Sie dieses Problem vermeiden. Alternativ hierzu können Sie die Weiterleitungsdeklarationssyntax benutzen, die Ihnen in IDL zur Verfügung steht.

Fügen Sie nun Ihrer Typlibibliothek eine Enumeration hinzu. Eine Enumeration definiert eine Gruppe mit Integer-Konstanten. Wie Sie wissen, eignen sich Enumerationen für die Definition von Parametern und Rückgabewerttypen. Viele Programmierer benutzen Enumerationen ebenfalls, um Fehlercodegruppen zu definieren. Nachfolgend ist ein Beispiel für eine Enumerationsdefinition in IDL aufgeführt:

```
typedef
[uuid(CC316146-9B37-4EF6-9E6D-2A68ACDCA908)]
enum {
    // 0x80040200 = vbObjectError + 512
    dsDogUnavailable = 0x80040200,
    dsDogUnagreeable = 0x80040201,
    dsDogNotCapable = 0x80040202,
    dsDogNotFound = 0x80040203,
    dsUnexpectedError = 0x80040204,
} DogErrorCodes;
```

Beachten Sie bitte, dass die Enumeration mit dem Schlüsselwort `typedef` definiert wird. Dieses Beispiel zeigt, wie eine Gruppe mit Fehlercodes, die von der `CCollie`-Komponente ausgelöst werden können, definiert wird. Der Anfangswert dieser Enumeration ist `vbObjectError + 512`. Dies ist der übliche Ausgangspunkt für Ihre benutzerdefinierten Fehlercodes (wie in Kapitel 4 beschrieben).

Lassen Sie uns jetzt das Definieren von UDTs erörtern. Sie sollten UDTs mit dem Schlüsselwort `struct` in IDL definieren. Bevor Sie jedoch UDTs in Ihren Methodensignaturen benutzen, sollten Sie zwei signifikante Punkte berücksichtigen. Erstens müssen alle Benutzer Visual Basic 6 oder eine spätere Version benutzen. Alle vorherigen Versionen von Visual Basic bieten keine Unterstützung für UDTs in COM-Methodenaufrufen. Zweitens muss Ihr Code auf Computern ausgeführt werden, die eine aktuelle Version des Universal-Marshalers ausführen, wie im vierten Kapitel beschrieben.

Die Arbeit mit UDTs in IDL kann zunächst ein wenig verwirrend sein. Sie sollten damit beginnen, sich die neueste Version des MIDL-Compilers zu beschaffen, um Probleme mit einer inkompatiblen Syntax zu vermeiden. Die Technik, die ich Ihnen zeigen werde, ist für viele frühere Versionen von MIDL nicht zulässig.

Sie müssen das `struct`-Schlüsselwort benutzen, sollten aber das `typedef`-Schlüsselwort vermeiden. Während Sie `typedef` verwenden, um ein UDT in IDL zu definieren, kann das Schlüsselwort zu einigen Problemen für den MIDL-Compiler führen. Sie definieren ein UDT wie folgt in IDL:

```
// UDT, der innerhalb der Typbibliothek DogLibrary definiert ist
[uuid(173CF18E-99DA-11D2-AB73-E8BE3D000000)]
struct DogData {
    BSTR Name;
    BSTR Rank;
    BSTR SerialNumber;
};
```

Jede UDT-Definition benötigt eine eigene identifizierende GUID. Lassen Sie uns nun den UDT in einer Methodensignatur verwenden. Hierfür ist ein wenig zusätzliche Aufmerksamkeit erforderlich. Betrachten Sie die folgenden Methodendefinitionen:

```
// Methodensignaturen, die den UDT nutzen
HRESULT Test9([in, out] struct DogData* Data);
HRESULT Test10([out, retval] struct DogData*);
```

UDT-Instanzen müssen als Referenz, und nicht als Wert übergeben werden. Dies bedeutet, dass Sie UDT-Parameter mit `[in, out]` oder `[out, retval]` definieren müssen. UDT-Parameter müssen außerdem als Zeiger, also mit dem Multiplikationszeichen (*) definiert werden. Beachten Sie eben-

falls, dass das `struct`-Schlüsselwort innerhalb der Methodendefinition verwendet wird. Der Parametertyp in diesen Methoden ist somit `struct DogData*` anstatt `DogData*`.

Sobald Sie den UDT in der Typbibliothek definiert haben, kann er in der implementierenden Klasse verwendet werden und erscheint dort ähnlich, wie nachfolgend dargestellt:

```
Private Sub IDog3_Test9(Data As DogData)
Private Function IDog3_Test10() As DogData
```

Kompilieren Ihrer Typbibliothek

Wenn Sie Ihren IDL-Code fertig gestellt haben, sollten Sie eine Typbibliothek generieren, indem Sie die IDL-Sourcdatei an den MIDL-Compiler übergeben. Wenn der MIDL-Compiler nicht auf Ihrer Entwicklungsmaschine vorhanden ist, müssen Sie ihn installieren. Sie können dies tun, indem Sie die letzte Version des Plattform SDKs installieren. Beachten Sie bitte, dass es sehr wichtig ist, die Option *Umgebungsvariablen registrieren* während der SDK-Installation zu selektieren, so dass der MIDL-Compiler im Systempfad abgelegt wird.

Sie können eine Typbibliothek ebenfalls von der Befehlszeile aus erstellen. Dies ist sehr einfach. Nachfolgend ist ein Beispiel für eine mögliche Eingabe aufgeführt:

```
MIDL /win32 DogLibrary.idl
```

Ein weiterer gebräuchlicher Trick besteht darin, eine Batch-Datei zu erstellen und ausführen zu lassen, die wie folgt aufgebaut ist:

```
MIDL /win32 DogLibrary.idl
Pause
```

Wenn ein Problem während des Kompilierens der IDL-Sourcdatei auftritt, meldet der MIDL-Compiler den Fehler im Konsolenfenster. Wenn der MIDL-Compiler ohne Fehler ausgeführt wird, generiert er eine Typbibliothek namens *DogLibrary.tlb*.

Der MIDL-Compiler besitzt wenige Befehlszeilenparameter. Sie benötigen die meisten dieser Parameter jedoch nicht. Diese sind für Entwickler gemacht, die andere Dateien als Typbibliotheken generieren möchten. Wenn Sie die gesamte Liste der zur Verfügung stehenden MIDL-Befehlszeilenparameter einsehen möchten, lassen Sie einfach das folgende Kommando von der Befehlszeile aus ausführen:

```
MIDL /?
```

Typbibliotheken verteilen und konfigurieren

Sobald Sie Ihre Typbibliothek erstellt haben, müssen Sie diese auf jedem System registrieren, das die Bibliothek verwendet. Dazu zählen sowohl Rechner der Betriebsumgebung als auch Entwicklungsmaschinen.

Die Typbibliothek muss auf den Computern der Betriebsumgebung vorhanden sein, da sie vom Universal-Marshaler benutzt wird, um Proxies und Stubs zu generieren. Nachdem eine Typbibliothek registriert wurde, ordnen bestimmte Registrierungseinträge jede `[oleautomation]`-IDD einer LIBID und die LIBIDs dem Pfad und Dateinamen der Typbibliothek zu. Der Universal-Marshaler verwendet diese Informationen, um die Schnittstellendefinition zur Laufzeit zu lokalisieren.

Die Schnittstellendefinition wird von dem Universal-Marshaler benutzt, um Proxy-/Stub-Code zu erstellen.

Sie müssen die Typlibibliothek auf Entwicklungsrechnern registrieren, damit Hilfsmittel, wie zum Beispiel Visual Basic, eine Assistenten-Unterstützung und Microsoft IntelliSense zur Verfügung stellen können. Wenn eine Typlibibliothek registriert ist, finden Sie ihre Beschreibung in dem Dialogfeld *Verweise*. Beachten Sie, dass der Visual Basic-Compiler die Typlibibliothek benötigt, um vTable-Bindungscode in Client-Applikationen zu erstellen.

Das Registrieren einer Typlibibliothek kann schwieriger als das Registrieren einer ActiveX-DLL sein. Sie können eine Typlibibliothek nicht mit *REGSVR32.EXE* registrieren, da sie keinen Selbstregistrierungscode enthält. Sie rufen stattdessen eine Funktion in der COM-Bibliothek auf, die mit *RegisterTypeLib* bezeichnet ist. (Sie müssen eigentlich zuerst eine Funktion namens *LoadTypeLib* aufrufen.) Wenn Sie *RegisterTypeLib* aufrufen, fügt die Funktion einen Eintrag für die Typlibibliothek zur Registrierung hinzu. Sie fügt außerdem Registrierungseinträge für jede Schnittstelle hinzu, die in der Typlibibliothek mit dem `[oleautomation]`-Attribut definiert ist.

Es ist sehr viel einfacher, *RegisterTypeLib* in C++ als in Visual Basic aufzurufen, da diese Funktion über Parameter verfügt, die auf Zeigerdatentypen basieren. Glücklicherweise gibt es mehrere Hilfsmittel, die Sie benutzen können, um *RegisterTypeLib* aufzurufen.

Das Programm *REGTLIB.EXE* wird mit Visual Studio ausgeliefert. Sie können dieses Hilfsmittel von der Befehlszeile aus starten, indem Sie den Pfad und Namen Ihrer Typlibibliothek übergeben. Sie können ebenfalls die Registrierung einer Typlibibliothek mit dem Programm wieder rückgängig machen.

Eine zweite Möglichkeit, eine Typlibibliothek in der Visual Basic-Entwicklungsumgebung zu registrieren, besteht darin, die Option *Durchsuchen* im Dialogfeld *Verweise* zu benutzen. Klicken Sie einfach auf *Durchsuchen* und selektieren Sie die gewünschte Typlibibliothek in dem anschließend angezeigten Dialogfeld. Wenn Sie die Typlibibliothek zu Ihrem Projekt hinzufügen, ruft Visual Basic die Funktion *RegisterTypeLib* auf. Beachten Sie, dass die Visual Basic-IDE nicht dazu verwendet werden kann, die Registrierung einer Typlibibliothek wieder rückgängig zu machen.

Die letzte Möglichkeit, eine Typlibibliothek zu registrieren, besteht darin, sie zu einer COM++-Applikation oder einem MTS-Paket hinzuzufügen. Dies ist etwas kompliziert, da Sie gleichzeitig einen DLL-Server und eine Typlibibliothek hinzufügen müssen. Wenn Sie die Typlibibliothek hinzufügen, wird diese automatisch auf dem lokalen Computer registriert. Die von COM+ und MTS generierten clientseitigen Setup-Programme installieren und registrieren die Typlibibliothek außerdem automatisch auf den Client-Rechnern. (Die Installation von Komponenten und Typlibibliotheken in COM++-Applikationen ist ausführlich im nächsten Kapitel beschrieben.)

Ein weiterer schwieriger Punkt

Stellen Sie sich vor, Sie haben Ihre Schnittstellen in IDL definiert und in eine Typlibibliothek kompiliert. Sie erstellen nun einen Server mit einer Komponente, die wie folgt aufgebaut ist:

```
Option Explicit
Implements IDog

Private Sub IDog_Bark()
    ' Ihre Implementierung
End Sub

Private Sub IDog_RollOver()
```

Sobald Sie all Ihre Schnittstellen in einer benutzerdefinierten Typlibibliothek definiert haben, definiert die Typlibibliothek Ihres Servers lediglich Co-Klassen. Sie schließen wahrscheinlich hieraus, dass die Typlibibliothek des Servers nicht zum Definieren von Schnittstellen verwendet wird. Es scheint, als wären alle wichtigen IIDs, für die Ihre Clients programmiert wurden, in der benutzerdefinierten Typlibibliothek und nicht in der Typlibibliothek Ihres Servers vorhanden.

Deshalb nun eine wichtige Frage: Sollten Sie den Versionskompatibilitätsmodus berücksichtigen, wenn Sie die folgenden Versionen Ihres Servers kompilieren? Es ist offensichtlich, dass Sie die Rekompilierung nicht mit der Einstellung *Keine Kompatibilität* durchführen sollten, da sich die CLSID ändert. Doch spielt es eine Rolle, ob Sie die Einstellung *Binär-Kompatibilität* oder die Einstellung *Projekt-Kompatibilität* verwenden?

Die intuitive Antwort auf diese Frage lautet »Nein«. Es sollte theoretisch keine Rolle spielen, da beide Kompatibilitätsmodi die CLSID über die verschiedenen Builds hinweg beibehalten (solange Sie Visual Basic 6 benutzen). Wenn Sie im Modus *Projekt-Kompatibilität* kompilieren, ändern sich die IIDs in der Typlibibliothek des Servers. Sie gehen jedoch davon aus, dass dies nicht wichtig ist, da Sie diese IIDs nicht berücksichtigen.

Diese Annahme wird Sie in einige Schwierigkeiten bringen. Wie Sie wissen, ist es weitaus wichtiger, wie etwas in der Praxis und nicht in der Theorie funktioniert. In der Praxis werden Sie häufig im Modus *Binär-Kompatibilität* rekompilieren müssen. Dies gilt auch dann, wenn Ihre Komponente keine öffentlichen Methoden enthält. Wenn Sie die *Projekt-Kompatibilität* benutzen, können Fehler auftreten.

Doch wieso ist dies so? Im dritten Kapitel erwähnte ich einen esoterischen Punkt, der die Vorgehensweise der Visual Basic-Laufzeit auf der Client-Seite während der Objektaktivierung und nach einem Client-Aufruf von `New` betraf. Die Visual Basic-Laufzeit ruft `CoCreateInstance` unter Verwendung von `IUnknown` auf. Dann erfolgt ein Aufruf von `QueryInterface`, um einen Verweis auf die Standard-IID hinter der Komponente abzurufen. Dies gilt unabhängig davon, ob der Client die Standardschnittstelle oder eine benutzerdefinierte Schnittstelle verwendet. Ein Visual Basic-Client weist somit auch dann Abhängigkeiten von der IID der Standardschnittstelle Ihrer Komponente auf, wenn er nicht explizit für die Standardschnittstelle programmiert wurde. Wenn sich die Standard-IID über verschiedene Builds hinweg ändert, werden ältere Clients während des Versuchs, ein Objekt aus der neuen Version des Servers zu aktivieren, einen Fehler zurückgeben.

Verbringen Sie nicht zu viel Zeit damit, sich zu fragen, wieso Visual Basic auf diese Weise arbeitet. Es ist nicht wichtig, ob dies einen Sinn für Sie ergibt. Es ist eben so. Was Sie hiervon mitnehmen sollten, ist, dass die binäre Kompatibilität immer dann erforderlich ist, wenn Sie Visual Basic-Clients besitzen, die die direkte `vTable`-Bindung und den `New`-Operator verwenden. (Das Problem ergibt sich nicht, wenn Clients die Objekte mit der `CreateObject`-Funktion erstellen.) Sie müssen Ihre Server auch dann im Modus *Binär-Kompatibilität* erneut erstellen, wenn Ihre Komponenten die Funktionalität exklusiv über Schnittstellen zur Verfügung stellen, die in einer benutzerdefinierten Typlibibliothek definiert sind. Doch auch dann gilt: Sie sind ein COM+-Programmierer. Es ist nicht Ihre Aufgabe, nach dem Warum zu fragen.

Zusammenfassung

Die Komponentenversionsverwaltung ist ein wichtiger Aspekt für die Entwicklung und Pflege von dynamischen Systemen. COM wurde von Grund auf entworfen, um die Verwaltung der Komponentenversionen zu ermöglichen. Visual Basic fügt gewiss eine eigene spezielle Note zur

COM-Versionsverwaltung hinzu. Sie müssen ständig über viele Details informiert sein und haben ebenso viele Verantwortlichkeiten. Das wichtigste ist jedoch, die richtige Frage zu Beginn eines jeden Projekts zu stellen.

Möchten Sie Scripting-Clients unterstützen? Möchten Sie Clients unterstützen, die die direkte vTable-Bindung benutzen? Sollten Sie Visual Basic benutzen, um Ihre Schnittstellen von diesem Hilfsmittel definieren und deren Versionen im Hintergrund verwalten zu lassen? Ergeben sich Vorteile, wenn Sie Ihre Schnittstellen in IDL definieren?

Jeder Entwickler wird diese Fragen anders beantworten. Solange Sie jedoch das Wissen besitzen, um diese Fragen korrekt beantworten zu können, können Sie ein Versionsverwaltungsschema aufstellen, das sehr effektiv für das Projekt ist. Besonders wichtig ist auch, dass der Visual Basic-Fehler 430 eine entfernte Erinnerung statt die tägliche Ursache für Schmerz und Frustration sein wird.

6 Konfigurierte Komponenten

140	Konfigurierte Komponenten – Grundlagen
154	Kontexte verstehen
160	Das COM+-Programmiermodell
164	COM+-Komponenten testen und debuggen

Das erste Kapitel hat eine Übersicht über die grundlegenden Prinzipien des COM+-Programmiermodells präsentiert. Im dritten Kapitel haben wir etwas genauer untersucht, wie das Programmiermodell von COM-Schnittstellen das dynamische Laden von Klassen und das transparente Erstellen von RPC-basierten Verbindungen zwischen Clients und Objekten einsetzt. Dieses Kapitel konzentriert sich auf einen wichtigen Aspekt eines Modells, das früher nicht Bestandteil von COM war. Gemeint sind Plattformdienste, auf die Sie über deklarative Attribute zugreifen. Das Kapitel möchte Ihnen zeigen, wie sich dieses Prinzip darauf auswirkt, wie Sie Ihren Code schreiben und Ihre Komponenten einsetzen.

Deklarative Attribute wurden zuerst mit dem MTS eingeführt. Sie boten dort eine Unterstützung für verteilte Transaktionen und die integrierte Sicherheit. COM+ baut auf dieser Idee auf, indem es zusätzliche Dienste über deklarative Attribute zur Verfügung stellt. Die Leser, die von MTS zu COM+ wechseln, werden einige Hinweise finden, die die wichtigsten Unterschiede zwischen den beiden Plattformen illustrieren. Wenn Sie direkt mit Microsoft Windows 2000 beginnen, müssen Sie die Details der Unterschiede zwischen MTS und COM+ nicht beachten. COM+ wurde neu entwickelt, um einfacher und weniger verwirrend zu sein.

Dieses Kapitel erklärt, wie COM+ Attributeinstellungen für Komponenten und Applikationen überwacht und diese nutzt. Sie erfahren außerdem, wie Ihnen COM+ erlaubt, Objekte aus konfigurierten Komponenten in prozessexternen und Remote-Szenarien zu aktivieren. Die Grundlagen des Erstellens von COM+-Komponenten mit Visual Basic werden ebenso beschrieben wie die Konfiguration dieser Komponenten für die Ausführung in einer COM+-Applikation.

Es ist wichtig, dass Sie verstehen, wie die COM+-Laufzeit Kontexte und das Abfangschema nutzt. Das Abfangschema gestattet COM+, die Dienste zur Verfügung zu stellen, die über die deklarativen Attribute angefordert werden. Zusätzlich zum Setzen von Attributen, müssen Sie bisweilen Code für die Typbibliothek *COM+ Services Type Library* schreiben. Sie werden eine gewisse Zeit damit verbringen, die gebräuchlichen Schnittstellen und die vom System zur Verfügung gestellten Objekte kennen zu lernen, die Sie benutzen, um mit der COM+-Laufzeit zu interagieren.

Am Ende des Kapitels werden wir einen kurzen Blick darauf werfen, wie Sie die mit Visual Basic erstellten konfigurierten Komponenten testen und debuggen. Sie werden feststellen, dass das Debuggen von konfigurierten Komponenten mit dem Visual Basic-Debugger recht einfach, aber mit einigen Einschränkungen verbunden ist. Sie werden außerdem einige andere Debug-Techniken

kennen lernen, die langwierig und kompliziert sind, aber eine bessere Simulation der Vorgänge bieten, mit denen Sie konfrontiert werden, wenn Sie Ihren Code in der Betriebsumgebung ausführen lassen.

Konfigurierte Komponenten – Grundlagen

Das Offenlegen von Systemdiensten über konfigurierte Attribute unterscheidet sich von dem Bereitstellen solcher Dienste über die Funktionen einer traditionellen API. In dem älteren API-Modell müssen Sie Code schreiben, um dem System mitzuteilen, welche Dienste Sie benutzen möchten und wie diese Dienste eingesetzt werden sollen. Wenn Sie die Art und Weise ändern möchten, in der eine Applikation einen Systemdienst nutzt, müssen Sie Code modifizieren und erneut kompilieren. Die Applikation muss daraufhin ebenfalls erneut distribuiert werden. Visual Basic-Programmierer sind darüber hinaus häufig benachteiligt, da ihnen die Programmiersprache nicht erlaubt, verschiedene Systemdienste über die C-basierte Win32-API aufzurufen. Die neue Möglichkeit, bei der COM+ und die deklarativen Attribute Systemdienste zur Verfügung stellen, ist weitaus flexibler und lässt Ihre Arbeit einfacher werden.

Die Arbeit mit deklarativen Attributen bietet zwei wesentliche Vorteile. Zunächst müssen Sie nicht sehr viel Code schreiben. Sie zeigen Ihre Wünsche, indem Sie Attribute zur Entwurfszeit konfigurieren und COM+ die eigentliche Arbeit zur Laufzeit ausführen lassen. Der zweite Vorteil besteht darin, dass Sie Attribute konfigurieren können, nachdem eine Applikation in die Betriebsumgebung übernommen wurde. Sie müssen nicht Code modifizieren oder Ihre Applikation erneut kompilieren, wenn Sie die Art und Weise ändern möchten, in der Sie einen Systemdienst nutzen. Dies ist besonders dann sehr vorteilhaft, wenn Sie Hardwareänderungen an der physikalischen Anwendung Ihrer Applikation vornehmen und eine Datenbank-Verbindungszeichenfolge modifizieren oder eine Middle-Tier-Applikation optimieren müssen, um die Vorteile eines leistungsfähigeren Computers nutzen zu können.

COM+-Applikationen

In COM+ benötigt jede konfigurierte Komponente ein Heim. Das Programmiermodell definiert ein Abstrakt, das als COM+-Applikation bezeichnet wird. Jede COM+-Applikation definiert bestimmte konfigurierte Attribute, die dem System beispielsweise mitteilen, wo Objekte aktiviert werden müssen und wie die Sicherheit einzurichten ist. Jede Komponente, die über deklarative Attribute verfügt, ist immer exakt einer COM+-Applikation zugeordnet. Sie können sich eine COM+-Applikation als eine Gruppe mit einer oder mehreren konfigurierten Komponenten vorstellen, die eingerichtet wurden, um auf einem bestimmten Computer ausgeführt zu werden. Zusätzlich zu den eigenen Attributen, erbt eine konfigurierte Komponente ebenfalls die Attribute von ihrer Host-Applikation. Wie Sie später noch sehen werden, überwacht COM+ auch die konfigurierten Attribute von Schnittstellen und Methoden.

HINWEIS: VON MTS ZU COM+

MTS verwendet den Begriff »Paket« in derselben Weise, wie COM+ den Terminus »Applikation« benutzt.

Im dritten Kapitel wurde beschrieben, welche Verwirrung mit dem Begriff »Komponente« einhergeht. Einige Personen benutzen diesen Ausdruck für eine konkrete Klasse, während andere Benutzer eine physikalische Serverdatei, wie zum Beispiel eine ActiveX-DLL meinen. Wieder andere gehen soweit, den Begriff »Komponente« mit dem Terminus »Objekt« zu vertauschen. Da der Begriff »Komponente« verschiedene Bedeutungen hat, müssen Sie häufig zwischen den Zeilen lesen, um

herauszufinden, wie er benutzt wird. In diesem Buch wird der Ausdruck für eine konkrete Klasse verwendet, die durch eine spezifische CLSID identifiziert wird. Im Bereich der COM+-Entwicklung mit Visual Basic ist eine Komponente eine *MultiUse*-Klasse, die in eine ActiveX-DLL kompiliert wird. Dies gilt besonders für COM+, da jede COM+-Applikation eine *Components*-Auflistung enthält, die über eine Gruppe mit konkreten Klassen verfügt. Sie sollten jedoch wachsam sein. Die COM+-Dokumentation im MSDN verwendet häufig den Begriff *Komponente*, um eine physikalische Serverdatei statt einer konkreten Klasse zu beschreiben. So viel zur Konsistenz.

Die COM+-Registrierungsdatenbank (RegDB)

Wenn Sie eine neue COM+-Applikation erstellen, generiert das System ein neues Profil für diese Applikation. Dies geschieht in einem Systemkatalog, der COM+-Registrierungsdatenbank (RegDB) genannt wird. Dieser Katalog enthält Attributseinstellungen für Applikationen, Komponenten, Schnittstellen und Methoden. Die Details zum Speichern der Daten in der RegDB sowie der Zugriff darauf werden in einer vom System zur Verfügung gestellten Komponente mit der Bezeichnung *Katalog-Manager* zusammengefasst, wie in Abbildung 6.1 dargestellt. Der Katalog-Manager erzeugt eine neue GUID (die Applikations-ID oder AppID), um jede Applikation zu identifizieren.

Es gibt zwei gebräuchliche Möglichkeiten, um eine COM+-Applikation zu erstellen und zu verwalten. Sie können dies manuell tun, indem Sie administrative Hilfsmittel mit der Bezeichnung *Komponentendienste* verwenden. Die zweite Möglichkeit ist programmgesteuert. Sie können administrative Skripts oder umfassende Visual Basic-Applikationen für eine Gruppe mit Komponenten schreiben, die vom System bereitgestellt werden. Diese Komponenten befinden sich in der COMAdmin-Bibliothek (Component Services Administration). Beide administrativen Techniken nutzen den Katalog-Manager.

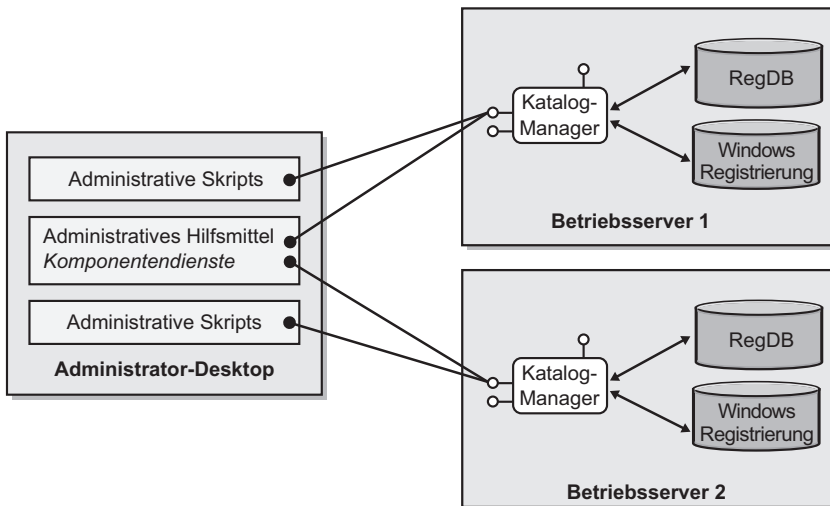


Abbildung 6.1: Der COM+-Katalog-Manager liest Konfigurationsinformationen aus der RegDB und der Windows-Registrierung und schreibt Daten hinein.

Eine Komponente ist eine konfigurierte Komponente, sobald sie einer COM+-Applikation hinzugefügt wurde. Wie Sie im dritten Kapitel gesehen haben, basieren frühere Versionen von COM auf Registrierungsschlüsseln, um Konfigurationsinformationen über Typbibliotheken, CLSIDs,

ProgIDs und IIDs zu speichern. COM+ benutzt dieselben Registrierungsschlüssel zusätzlich zu anderen Attributseinstellungen, die in der RegDB gespeichert sind. Das COM+-Team entschied aus verschiedenen Gründen, die herkömmliche Methode zum Speichern von COM-Konfigurationsinformationen beizubehalten. Das Team kam außerdem überein, dass doppelt vorhandene Einstellungen in der RegDB eine schlechte Idee seien. Dies bedeutet, dass Konfigurationsinformationen über konfigurierte Komponenten sowohl innerhalb als auch außerhalb der RegDB abgelegt sind.

HINWEIS: VON MTS ZU COM+

Konfigurierte Komponenten und MTS

Der Ausdruck Komponente wurde erst mit Erscheinen von COM+ präzisiert. Sie sollten sich eine Komponente, die in einem MTS-Paket installiert ist, jedoch als konfigurierte Komponente vorstellen. Beide unterscheiden sich nicht. MTS-Komponenten sind wie COM+-Komponenten, da sie über deklarative Attribute verfügen.

Das Konzept eines Standard-Katalog-Managers ist sehr nützlich. Die Komponentenregistrierung wird hierdurch einfacher und sicherer, da Administratoren und Entwickler nicht dafür verantwortlich sind, Daten in die Windows-Registrierung respektive in die RegDB zu schreiben oder daraus zu lesen. Sie müssen sich keine Gedanken darüber machen, wie die Registrierungsdaten physikalisch in der RegDB gespeichert sind. Der Katalog-Manager ist außerdem eine COM+-Komponente, die die Remote-Aktivierung unterstützt. Ein Administrator kann einfach eine COM+-Applikation erstellen und die Komponenten auf entfernten Maschinen mit Skripts oder dem administrativen Hilfsmittel *Komponentendienste* konfigurieren. Dies bedeutet, dass COM+ ein Standardschema für die Netzwerkverwaltung zur Verfügung stellt.

Dies ist sehr wichtig, weil ein Administrator sehr viele Betriebsserver von einem einzelnen Desktop-Computer aus konfigurieren kann. Es ist sehr einfach, eine Komponente von einem Computer zu entfernen und auf einem anderen Computer einzusetzen, wenn eine Applikation Änderungen an der physikalischen Installation notwendig werden lässt. Dies ist häufig der Fall, wenn ein Unternehmen weitere Computer zu einer Applikation hinzufügt, um einen höheren Durchsatz und eine bessere Fehlertoleranz und Sicherheit zu erzielen. In den frühen Tagen von COM war die Konfiguration von Servern sehr viel schwieriger, da diese Arbeit gewöhnlich die Überprüfung aller beteiligten Computer erforderte.

Konfigurierte Komponenten im Vergleich zu nicht konfigurierten Komponenten

Zusätzlich zu konfigurierten Komponenten werden Sie mit einem älteren Komponententyp konfrontiert, der keine COM+-Attribute besitzt – die nicht konfigurierten Komponenten. Diese Komponenten sind nicht in COM+-Applikationen installiert. Sie werden stattdessen wie in früheren Versionen von COM registriert. Sie sollten zum Beispiel eine prozessinterne DLL, die nicht konfigurierte Komponenten enthält, mit *REGSVR32.EXE* registrieren. Beachten Sie, dass nicht konfigurierte Komponenten die Vorteile der COM+-Dienste nicht nutzen können. Sie können jedoch in anderen Umgebungen als MTS und COM+ ausgeführt werden.

Wenn Sie Komponentencode für eine auf COM+ basierende Applikation schreiben, produzieren Sie in der Regel konfigurierte Komponenten. Dies erlaubt Ihnen, die Vorteile verschiedener Plattformdienste zu beanspruchen. Wenn Sie Ihren Code erstellen, werden Sie voraussichtlich ebenfalls mit nicht konfigurierten Komponenten konfrontiert. ADO ist beispielsweise eine Bibliothek, die aus nicht konfigurierten Komponenten besteht. ADO-Objekte können in einer auf COM+ basierenden Applikation ausgeführt werden. Dies gilt jedoch ebenfalls für Applikationen, die auf früheren Versionen von COM basieren.

Sie sollten außerdem beachten, dass Visual Basic fast nichts über die COM+-Administration weiß. Wenn Sie eine ActiveX-DLL erstellen, fügt die Visual Basic-IDE der DLL einen herkömmlichen Selbstregistrierungscode hinzu und ruft *REGSVR32.EXE* auf. Ihre *MultiUse*-Klassen werden somit als nicht konfigurierte Komponenten eingerichtet, wenn Sie eine ActiveX-DLL auf einem Windows 2000-Computer erstellt haben. Sie müssen Ihre Klassen explizit zu einer COM+-Applikation hinzufügen, um sie in konfigurierte Komponenten zu wandeln.

Konfigurierte Komponenten erstellen und einsetzen

Einer der wichtigsten Gründe für viele Unternehmen, Visual Basic zum Erstellen von Middleware zu verwenden, besteht darin, dass es sehr einfach ist, Komponenten für eine COM+-Applikation zu erzeugen. Sie müssen lediglich eine einfache ActiveX-DLL mit einer oder mehreren *MultiUse*-Klassen generieren. Sie fügen dann einige öffentliche Methoden hinzu, implementieren diese und erstellen die DLL. Es ist so einfach.

Bevor wir damit beginnen, folgen zunächst einige Empfehlungen, die Sie während der Erstellung von konfigurierten Komponenten berücksichtigen sollten:

- Seien Sie vorsichtig, wenn Sie den New-Operator benutzen.
- Setzen Sie das Threading-Modell der DLL nicht auf Single-Threaded.
- Benutzen Sie keine öffentlichen Variablen in .BAS-Modulen.
- Halten Sie Datenbankverbindungen nicht über mehrere Methodenaufrufe hinweg geöffnet.

Sie wissen nun, was Sie nicht tun sollten. Die Gründe hierfür sind jedoch schwierig zu erklären. Ich werde versuchen, diese Empfehlungen im Laufe der nächsten Kapitel zu erläutern.

Die einfachste Möglichkeit, eine neue COM+-Applikation zu erstellen und ihr die Komponenten aus einer DLL hinzuzufügen, besteht darin, das administrative Hilfsmittel *Komponentendienste* zu verwenden, das in Abbildung 6.2 dargestellt ist. Wenn Sie mit diesem Programm nicht vertraut sind, sollten Sie einen Blick auf die Plattform-SDK-Dokumentation im MSDN werfen und nach »*Creating and Configuring COM+ Applications*« suchen. Dieser Abschnitt führt Sie durch die COM+-Administration. Er kann Ihnen außerdem eine Einführung zum Schreiben von administrativen Skripts und Applikationen mit der COMAdmin-Bibliothek bieten.

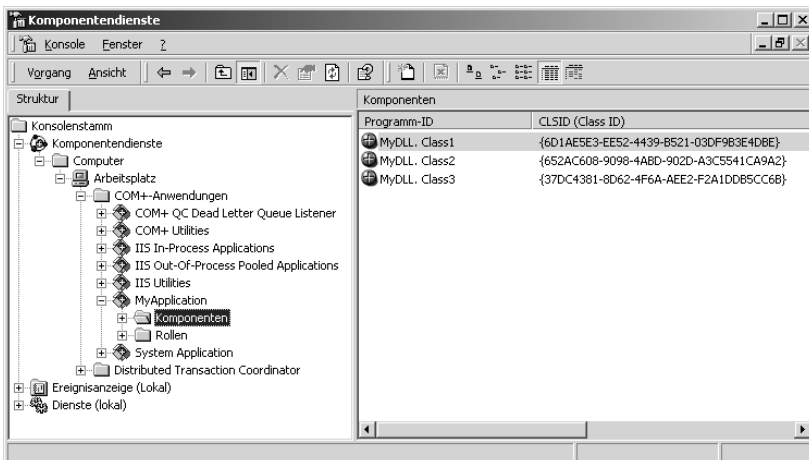


Abbildung 6.2: Sie sollten lernen, wie COM+-Applikationen und konfigurierte Komponenten mit dem administrativen Hilfsmittel Komponentendienste erstellt werden.

Beachten Sie auch, dass der Abschnitt »COM+ *Administration Reference*« in der Plattform-SDK-Dokumentation kurz jedes konfigurierbare Attribut beschreibt. Das Thema mit dem Titel »*Applications Collection*« führt jedes Applikationsattribut auf. Das Thema »*Components Collection*« listet alle Komponentenattribute auf. Sie sollten sich ein wenig Zeit nehmen, um mit diesen Ressourcen vertraut zu werden.

Wenn Sie eine neue COM+-Applikation mit dem administrativen Hilfsmittel *Komponentendienste* erstellen, möchte ein Assistent von Ihnen wissen, ob die Identitätseigenschaft der Applikation gesetzt werden soll. In einer Betriebsumgebung sollte dieser Eigenschaft ein spezielles Benutzerkonto zugewiesen werden, dass dafür vorgesehen ist, Objekte der mittleren Schicht auszuführen. Während Sie jedoch Ihre Komponenten schreiben und testen, ist die Standardeinstellung *Interaktiver Benutzer* die einfachste und übliche Wahl.

Wenn Sie eine Komponente zu einer COM+-Applikation hinzufügen, erhält die Komponente neue Attribute. Jedem Attribut wird ein Standardwert zur Entwurfszeit zugewiesen. Sie können die Liste der Komponentenattribute im MSDN einsehen, um eine Übersicht über alle Standardwerte zu erhalten. In einigen Fällen weisen die Attribute Werte auf, die Sie nicht erwarten. Sie müssen dann ein wenig zusätzliche Arbeit leisten, um die korrekte Konfiguration vorzunehmen.

Sie können diese konfigurierbaren Attribute (mit Ausnahme der Transaktionsunterstützung) nicht vorab aus der Visual Basic-IDE heraus definieren. Sie müssen Ihre Komponenten zu einer COM+-Applikation hinzufügen und die Attribute nach der Inbetriebnahme konfigurieren. Die Einstellungen können programmgesteuert oder manuell mit dem administrativen Hilfsmittel *Komponentendienste* vorgenommen werden. Zukünftige Versionen der Microsoft-Entwicklungswerkzeuge und von COM+ werden Ihnen möglicherweise erlauben, Komponentenattribute zur Entwurfszeit zu konfigurieren. Gegenwärtig gibt es jedoch noch keine elegante Lösung.

Zusätzlich zum korrekten Einrichten Ihrer Komponenten und Applikationsattribute, müssen Sie ebenfalls Code schreiben, der mit der COM+-Laufzeit interagiert. Sie müssen dazu die Bibliothek *COM+ Services Type Library* in Ihrem ActiveX-DLL-Projekt referenzieren, wie in Abbildung 6.3 dargestellt. Diese Typbibliothek enthält Definitionen für die Komponenten, Schnittstellen, UDTs und Enumerationen, die von COM+ bestimmt werden.

In den meisten Fällen werden die DLLs, die Sie für die mittlere Schicht erstellen, konfigurierte Komponenten enthalten, weil diese die Vorteile der COM+-Dienste nutzen können. Konfigurierte Komponenten weisen jedoch eine Abhängigkeit von der COM+-Laufzeit auf. Wenn eine Komponente auf die Typbibliothek *COM+ Services Type Library* zugreift und konfigurierte Attribute benutzt, kann sie nicht in anderen Umgebungen ausgeführt werden.

Sie werden in einigen seltenen Szenarien möglicherweise Komponenten erstellen wollen, die keine Abhängigkeiten von der COM+-Laufzeit aufweisen. Dies ist zum Beispiel der Fall, wenn Sie eine Komponente besitzen, die die Umsatzsteuer berechnet oder Benutzereingaben prüft, und wenn diese Komponente sowohl auf einem Desktop-Computer mit Windows 95 als auch in einer COM+-Applikation verwendet werden soll. Wenn Sie Abhängigkeiten von COM+-Attributen und der Typbibliothek *COM+ Services Type Library* vermeiden, können Sie Ihre Klassen entweder als konfigurierte Komponenten oder als nicht konfigurierte Komponenten ausführen lassen. Denken Sie nur daran, dass nicht konfigurierte Komponenten keine COM+-Dienste nutzen können.

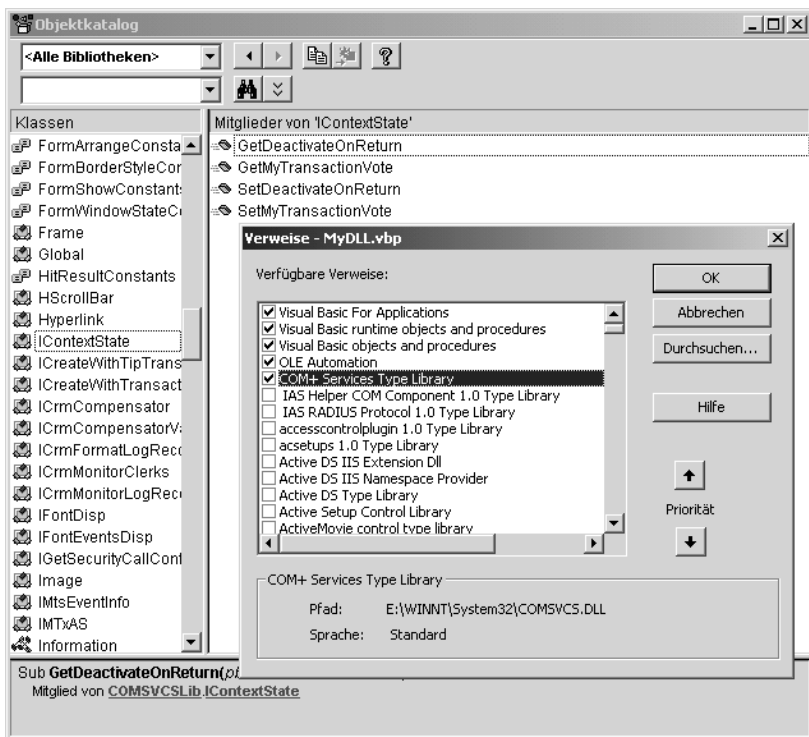


Abbildung 6.3: Die Typbibliothek COM+ Services Type Library enthält mehrere Schnittstellen- und Enumerationsdefinitionen, die Sie benutzen, um mit der COM+-Laufzeit zu interagieren.

Sie fragen sich vielleicht: »Liegt das an mir, oder ist all diese Computertheorie ein wenig verworren?« Lassen Sie uns einen Blick darauf werfen, wie eine Komponente erstellt wird, die einige konfigurierbare Attributeinstellungen nutzt. Dieses Beispiel zeigt Ihnen, wie die Theorie in die Praxis umgesetzt wird.

Objektkonstruktor-Zeichenfolgen verwenden

Jede konfigurierte Komponente verfügt über eine Objektkonstruktor-Zeichenfolge. Sie können diese für verschiedene Aufgaben benutzen. In unserem Beispiel wird die Zeichenfolge jedoch verwendet, um eine OLE-DB-Datenbankverbindungszeichenfolge aufzunehmen. Die beiden wichtigen konfigurierbaren Attribute des Beispiels sind *ConstructionEnabled* und *ConstructorString*. Beachten Sie, dass Sie beide Attribute mit dem administrativen Hilfsmittel *Komponentendienste* auf der Registerkarte *Aktivierung* des Eigenschaftendialogfeldes der Komponente konfigurieren können.

Wenn der SCM ein Objekt aus einer konfigurierten Komponente erstellt, der eine deklarative Konstruktorzeichenfolge zugewiesen wurde, führt die COM+-Laufzeit einen Aufruf an das Objekt durch und stellt eine Möglichkeit zur Verfügung, den Zeichenfolgenwert aus RegDB zu laden.

Lassen Sie uns nun detailliert untersuchen, wie die COM+-Laufzeit mit dem neuen Objekt interagiert. Der SCM prüft während der Objekterstellung, ob das *ConstructionEnabled*-Attribut der Komponente aktiviert ist. Er führt in diesem Fall *QueryInterface* auf dem neuen Objekt aus, um

einen Verweis auf eine Schnittstelle namens `IObjectConstruct` abzurufen. Wenn Ihre Komponente diese Schnittstelle implementiert, ruft die COM+-Laufzeit die `Construct`-Methode auf und übergibt dieser einen Verweis auf ein Konstruktorobjekt. Betrachten Sie den folgenden Code:

```
Implements COMSVCSLib.IObjectConstruct
Private MyConnectionString As String

Private Sub IObjectConstruct_Construct(ByVal pCtorObj As Object)
    Dim cs As IObjectConstructString
    Set cs = pCtorObj
    MyConnectionString = cs.ConstructString
End Sub
```

Wie Sie sehen, lädt diese benutzerdefinierte Implementierung der `Construct`-Methode die Objekt-konstruktor-Zeichenfolge in eine Modulvariable mit der Bezeichnung `MyConnectionString`. Beachten Sie, dass die vom `pCtorObj`-Parameter übergebene Referenz an die `IObjectConstructString`-Schnittstelle weitergeleitet wird. Diese Schnittstelle stellt nur eine Eigenschaft zur Verfügung, die mit `ConstructString` bezeichnet ist. Sie dient dazu, den konfigurierten Wert von RegDB abzurufen und in einer im gesamten Modul gültigen Variablen zu speichern. Nachdem das Objekt erstellt wurde, kann die Konstruktorzeichenfolge von jeder anderen Methodenimplementierung in der Klasse benutzt werden.

Der wichtigste Aspekt der Verwendung einer Konstruktorzeichenfolge besteht darin, dass die eigentliche Zeichenfolge nicht in die DLL kompiliert wird. Der Administrator kann die Verbindungszeichenfolge einfach mit dem administrativen Hilfsmittel *Komponentendienste* neu konfigurieren. Wenn sich die Verbindungsinformationen der Datenbank ändern, müssen Sie somit keinen Code modifizieren. Dieses Beispiel demonstriert, wie COM+ dafür sorgt, dass Sie mehr deklarative Arbeit verrichten, wodurch der mit der Programmierung und erneuten Kompilierung verbundene Aufwand verringert wird. Das Beispiel zeigt Ihnen ebenfalls, wie eine in der Typbibliothek *COM+ Services Type Library* definierte Schnittstelle implementiert wird.

Bibliotheksapplikationen im Vergleich mit Serverapplikationen

Im dritten Kapitel wurde beschrieben, wie der SCM Objekte aus einem prozessinternen Server, wie zum Beispiel einer ActiveX-DLL, aktiviert. Frühere Versionen des SCM kennen die erweiterten Attribute jedoch nicht, die von COM+ zur Verfügung gestellt werden. Das COM+-Team schrieb den SCM für Windows 2000 deshalb um, so dass die Aktivierungseinstellung einer konfigurierten Komponentenapplikation überprüft werden konnte. Diese Einstellung verrät dem SCM, wo und wie Objekte aktiviert werden.

Jede COM+-Applikation muss entweder als Bibliotheksapplikation oder als Serverapplikation konfiguriert werden. Der Unterschied zwischen diesen beiden Aktivierungseinstellungen ist recht einfach. Werfen Sie dazu einen Blick auf Abbildung 6.4. Die aus Bibliotheksapplikationskomponenten erstellten Objekte werden aktiviert und in dem Prozess ihres Erzeugers ausgeführt. Die aus Serverapplikationskomponenten erstellten Objekte werden in ihrem eigenen Ersatzprozess aktiviert. Clients können Objekte außerdem aus lokalen Serverapplikationen sowie aus Remote-Serverapplikationen, die über das Netzwerk ausgeführt werden, aktivieren. Die Möglichkeit, eine Komponente in einem dieser Szenarien einzusetzen, ohne Modifikationen vornehmen oder erneut kompilieren zu müssen, ist unter dem Begriff *Ortsunabhängigkeit* oder örtliche *Transparenz* bekannt.

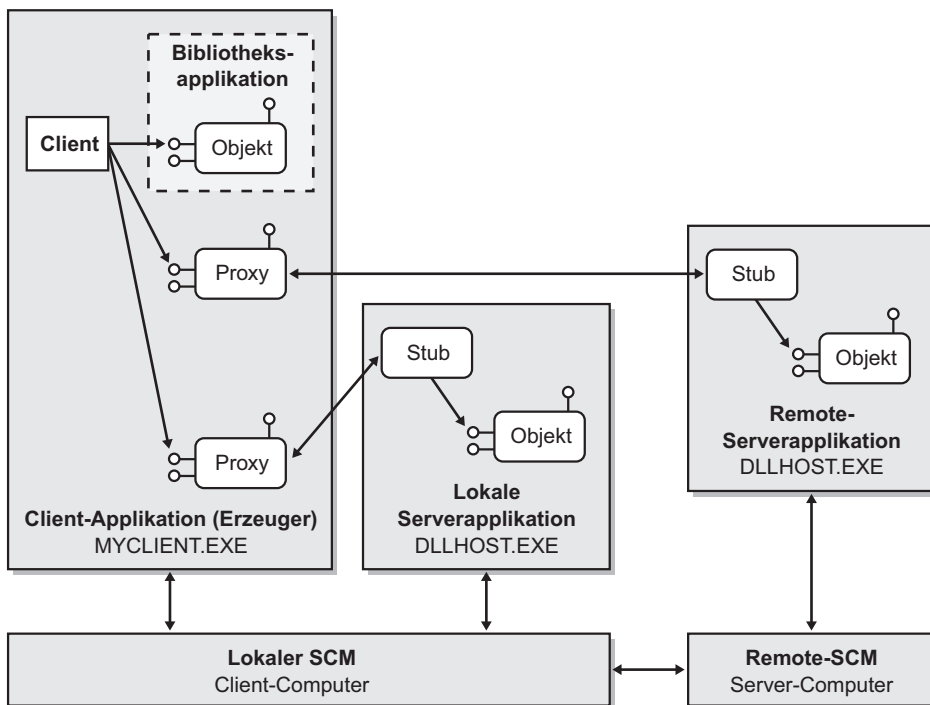


Abbildung 6.4: COM+ stellt eine so genannte Ortsunabhängigkeit zur Verfügung. Komponenten können konfiguriert werden, um prozessintern, lokal oder über ein Netzwerk ausgeführt zu werden. Eine Änderung am Code oder ein erneutes Kompilieren ist hierfür nicht notwendig.

Prozessexterne Aktivierung

Jede Serverapplikation wird in einer separaten Instanz der Ersatzcontainerapplikation *DLLHOST.EXE* ausgeführt. Wenn ein Client ein Objekt aus einer Komponente erstellt, die sich in einer Serverapplikation befindet, lokalisiert der SCM die entsprechende Instanz von *DLLHOST.EXE*. Wenn die Serverapplikation noch nicht ausgeführt wird, startet sie der SCM, bevor er das Objekt erstellt. Dieses Schema erlaubt COM+, Applikationen auf Anforderung zu starten.

HINWEIS: VON MTS ZU COM+

Der Name der Containerapplikation hat sich verändert.

In MTS lautet der Name der Ersatzcontainerapplikation *MTX.EXE* und nicht *DLLHOST.EXE*.

HINWEIS: VON MTS ZU COM+

Die Refresh Components-Anweisung ist nicht mehr erforderlich.

Sowohl in COM als auch in COM+ muss der *CLSID*-Schlüssel für eine Komponente in einer ActiveX-DLL einen *InProcServer32*-Unterschlüssel besitzen, der auf den Pfad des Servers verweist. Wenn Sie eine Visual Basic-ActiveX-DLL erstellen, fügt der Compiler der DLL automatisch Selbstregistrierungscode hinzu. Sobald die DLL erstellt wurde, wird sie mit dem Programm *REGSVR32.EXE* registriert.

Der SCM benutzt die Informationen im *CLSID*-Schlüssel, wenn er ein Objekt generieren muss. Der SCM in Windows NT 4 weiß jedoch nicht, wie ein MTS-Objekt mit Hilfe konfigurierter Attribute korrekt erzeugt wird. Der MTS verlangt, dass *CLSIDs* anders konfiguriert werden, als dies unter COM und COM+ üblich ist. Das MTS-Team musste somit eine Möglichkeit schaffen, dem SCM die Aktivierungsanforderungen zu nehmen.

Wenn Sie einem MTS-Paket eine Komponente hinzufügen, ändert der Katalog-Manager die Einträge unter dem *CLSID*-Schlüssel, um die Aktivierungsanforderungen zur MTS-Laufzeit umzuleiten. Eine Komponente in einem MTS-Bibliothekspaket verlangt einen *InProcServer32*-Schlüssel, der auf *MTXEX.DLL* verweist. Eine Komponente in einem MTS-Serverpaket benötigt einen *LocalServer32*-Schlüssel, der auf die Containerapplikation *MTX.EXE* verweist.

Dies ist recht verwirrend. Ihnen stehen unter Windows NT 4 drei Möglichkeiten zur Verfügung, um eine *CLSID* zu registrieren: die ursprüngliche COM-Methode und zwei MTS-Vorgehensweisen. Um alles noch ein wenig schlimmer zu machen, kennt der in eine Visual Basic-DLL integrierte Selbstregistrierungscode lediglich die COM-Vorgehensweise zum Registrieren einer *CLSID*. Wenn Sie *REGSVR32.EXE* für eine Visual Basic-DLL aufrufen, nachdem diese in einem MTS-Paket installiert wurde, geraten die *CLSID*-Einträge durcheinander.

Viele Programmierer geraten in Schwierigkeiten, wenn sie DLLs auf einer MTS-Maschine erstellen und prüfen. Der Grund hierfür besteht darin, dass Visual Basic immer *REGSVR32.EXE* aufruft, wenn ein Build erfolgreich erstellt worden ist. Sie können dieses Verhalten in der Visual Basic-IDE nicht unterbinden. Wenn Sie versuchen, eine MTS-Komponente zu testen, die zu einer COM-Standardkomponente rekonfiguriert wurde, werden Sie ein Fehlverhalten feststellen.

Um dieses Problem zu lösen, bietet der MTS eine *Refresh Components*-Anweisung an. Wenn Sie diese Anweisung für ein MTS-Paket ausführen lassen, durchsucht der MTS-Katalog-Manager einfach die Windows-Registrierung und stellt sicher, dass jede *CLSID* gemäß der MTS-Vorgehensweise und nicht mit der COM-Methode konfiguriert wurde. Es gibt darüber hinaus einen Visual Basic-Zusatz, der den MTS-Katalog-Manager automatisch aufruft, um MTS-Komponenten immer dann zu aktualisieren, wenn der verknüpfte ActiveX-DLL-Server erneut erstellt wird.

In COM+ werden die *CLSIDs* aus einer ActiveX-DLL immer gemäß der ursprünglichen Version von COM registriert. Es gibt immer einen *InProcServer32*-Schlüssel, der den Pfad zu Ihrer DLL aufnimmt. Dies gilt sowohl für konfigurierte als auch für nicht konfigurierte Komponenten. Konfigurierte Komponenten unterscheiden sich von nicht konfigurierten Komponenten nur darin, dass sie zusätzliche Informationen über ihre Attribute und Aktivierungsanforderungen in RegDB besitzen. Die guten COM+-Neuigkeiten lauten jedoch, dass Sie sich keine Gedanken mehr um das *Refresh Components*-Kommando machen müssen, wie dies in MTS erforderlich war.

Sobald der SCM den Prozess der Ziel-Serverapplikation lokalisiert hat, leitet dieser die Anforderung für das neue Objekt weiter. Im dritten Kapitel wurde jedoch erwähnt, dass der Code zum Erstellen von Objekten in den Class-Factories Ihrer DLLs aufbewahrt wird. Wenn eine Serverapplikation gestartet wird, lädt die COM+-Laufzeit alle verknüpften DLLs, um jede Class-Factory zu registrieren. Wenn die Class-Factory für die angeforderte *CLSID* geladen wurde, kann die Serverapplikation das Objekt erzeugen.

Die Serverapplikation verwendet eine Class-Factory, um ein Objekt zu erstellen, und gibt eine Objektreferenz an den SCM zurück. Die Referenz wird dann an den Client weitergeleitet. Nach einer erfolgreichen prozessexternen Aktivierung ist der Client aber tatsächlich mit einem Proxy

verbunden, der – für den Client nicht sichtbar – eine RPC-basierte Verbindung zu dem Objekt aufrechterhält.

Der Prozess für eine Serverapplikation bleibt standardmäßig drei Minuten im Speicher vorhanden, nachdem das letzte Objekt freigegeben wurde. Sie können dieses Zeitintervall jedoch im administrativen Hilfsmittel *Komponentendienste* auf der Registerkarte *Erweitert* im Eigenschaftendialogfeld einer Serverapplikation konfigurieren. (Sie können eine Serverapplikation auch ohne Zeitbeschränkung ausführen oder sofort herunterfahren lassen, nachdem das letzte Objekt freigegeben wurde.)

COM+ lässt Sie nicht nur den Zeitpunkt bestimmen, zu dem ein Prozess heruntergefahren werden soll. Es unterstützt ebenfalls das frühe Starten einer Serverapplikation. Der frühe Start einer Applikation ist eine Optimierung, da der erste Client nicht darauf warten muss, dass der Serverprozess gestartet wird. Sie können eine Serverapplikation wie auch die meisten anderen administrativen Routineaufgaben manuell oder programmgesteuert vorab starten.

Remote-Aktivierung

Die Aktivierung eines Objekts in einer Remote-Serverapplikation ist weitaus komplexer als eine prozessinterne oder lokale Aktivierung, da hierfür die Beteiligung und Koordinierung des SCM auf zwei verschiedenen Computern erforderlich ist. Wenn der clientseitige SCM feststellt, dass ein Client ein Objekt aus einer Komponente aktivieren möchte, die unter einer anderen Netzwerkadresse registriert ist, leitet er die Aktivierungsanforderung an den SCM des Servercomputers weiter. Der serverseitige SCM ist für das Erstellen des angeforderten Objekts und die Rückgabe einer Referenz verantwortlich.

Wenn der serverseitige SCM eine Aktivierungsanforderung von einem Client über das Netzwerk erhält, initiiert er eine prozessexterne Standardaktivierung von der Serverapplikation aus, die über die angeforderte CLSID verfügt. Aus der Sicht des Servercomputers wird die Aktivierung lokal durchgeführt. Nachdem das Objekt erstellt wurde, wird eine Referenz von der Serverapplikation an den serverseitigen SCM, dann an den clientseitigen SCM und schließlich zurück an den Client übergeben. Der Client wird, wie bei der lokalen prozessexternen Aktivierung, über ein Proxy-/Stub-Paar mit dem Objekt verbunden.

Beachten Sie, dass Client-Computer nicht unter Windows 2000 ausgeführt werden müssen. Diese Rechner benötigen lediglich eine Version von Windows, die mit DCOM (Distributed COM) arbeitet. Windows NT 4 und Windows 98 werden bereits mit DCOM eingerichtet. Wenn Sie Client-Applikationen auf Computern mit Windows 95 ausführen lassen möchten, müssen Sie die entsprechende DCOM-Software zunächst installieren, indem Sie die entsprechenden Dateien von der Microsoft Website aus dem Internet laden (www.microsoft.com/com/resources/downloads.asp).

Wie Sie im dritten Kapitel gelernt haben, ist die mit Visual Basic erstellte Software auf den Universal-Marshaler angewiesen, um Proxies und Stubs erzeugen zu können. Der Universal-Marshaler erstellt einen Proxy auf dem Client-Computer, indem er die entsprechende Schnittstellendefinition in einer Typbibliothek untersucht. Dies bedeutet, dass eine Kopie der Typbibliothek auf jeden Client-Computer kopiert und registriert werden muss. Ein Client-Computer muss außerdem Informationen über IIDs, CLSIDs und ProgIDs besitzen.

Sie können jeden clientseitigen Computer, der eine DCOM-Version unter Windows ausführt, konfigurieren, um eine Aktivierung anhand der CLSID an einen Remote-Computer weiterzuleiten. Der SCM leitet eine Aktivierungsanforderung weiter, wenn er feststellt, dass eine CLSID über eine verknüpfte AppID mit einer zulässigen Netzwerkadresse verfügt.

Sie müssen diese clientseitigen Registrierungseinträge wahrscheinlich niemals manuell hinzufügen. Sie sollten dennoch wissen, wie Sie vorgehen müssten, um dies zu tun. Fügen Sie zuerst einen *AppID*-Schlüssel mit dem benannten Wert *RemoteServerName* hinzu, wie in Abbildung 6.5 dargestellt. Setzen Sie den *RemoteServerName*-Wert anschließend auf eine IP-Adresse, eine DNS-Adresse (Domain Name System) oder einen NETBIOS-Namen. Fügen Sie schließlich für jede CLSID den benannten Wert *AppID* zum Registrierungsschlüssel hinzu, um diese mit dem Remote-Servernamen zu verknüpfen. Wenn alles richtig konfiguriert ist, leitet der clientseitige SCM jede Aktivierungsanforderung für alle CLSIDs an den Remote-Servercomputer weiter. Auf diese Weise wurde die Weiterleitung mit DCOM durchgeführt, bevor Windows 2000 erhältlich war. Windows 2000 speichert die AppID ebenso wie DCOM in der Windows-Registrierung. Die CLSID- und ProgID-Informationen werden jedoch in der RegDB abgelegt.

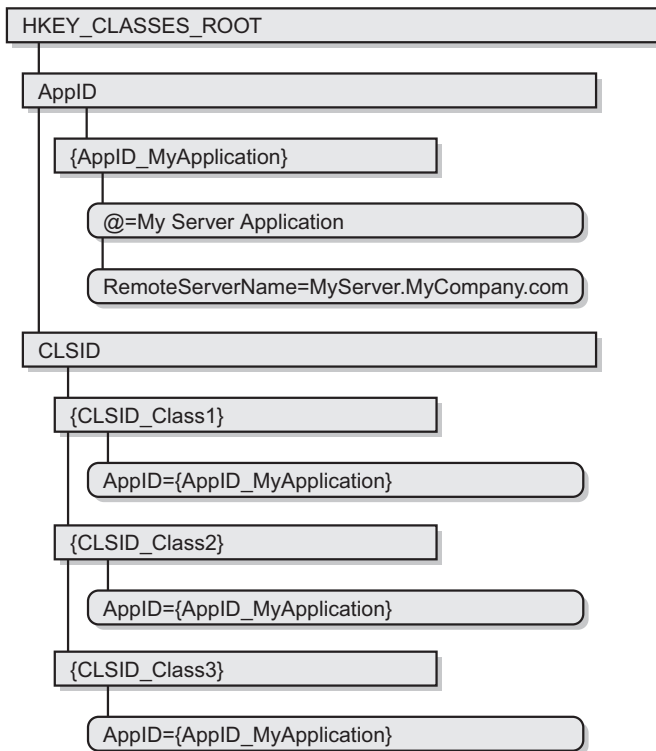


Abbildung 6.5: Eine CLSID, die für die Remote-Aktivierung konfiguriert wurde, muss eine AppID mit einem zulässigen RemoteServerName-Wert besitzen. Die CLSID sollte nicht über einen InProcServer-Unterschlüssel oder einen LocalServer32-Unterschlüssel verfügen.

Wenn Sie eine CLSID mit einer AppID für die Remote-Aktivierung konfigurieren, darf der CLSID-Schlüssel keinen *InProcServer32*- oder *LocalServer32*-Unterschlüssel aufweisen. Der clientseitige SCM leitet die Aktivierung nur dann an einen Remote-Computer weiter, wenn diese beiden Werte fehlen. Programmierer werden immer wieder mit Problemen konfrontiert, wenn sie eine DLL auf der Client-Maschine registrieren und unwissentlich den *InProcServer32*-Unterschlüssel hinzufügen. Der SCM führt in diesem Fall eine lokale prozessinterne Aktivierung anstelle eines Aufrufs über das Netzwerk durch, um das Objekt zu erstellen.

Sie sollten einen weiteren Hinweis zur Remote-Aktivierung berücksichtigen. Sie müssen nicht den *RemoteServerName*-Wert für die Auswahl eines Zielservers verwenden. Sie können stattdessen die Adresse eines Remote-Computers programmgesteuert mit der *CreateObject*-Funktion von Visual Basic angeben. Visual Basic 6 hat dieser Funktion einen zweiten optionalen Parame-

ter hinzugefügt, der den Namen eines Remote-Servers akzeptiert. Dieser Parameter überschreibt immer den *RemoteServerName*-Wert in der Windows-Registrierung. Dem Parameter kann wie dem *RemoteServerName*-Wert eine IP-Adresse, eine DNS-Adresse oder ein NETBIOS-Name übergeben werden.

Applikations-Proxies

Einige Abschnitte zuvor wurde erwähnt, dass Sie wahrscheinlich niemals clientseitige Konfigurationsinformationen manuell zur Windows-Registrierung hinzufügen müssen. Der Grund hierfür besteht darin, dass Ihnen COM+ erlaubt, ein clientseitiges Setup-Programm zu erstellen, das automatisch Typbibliotheken kopiert und alle erforderlichen Konfigurationsinformationen in die Registrierung schreibt. Dieses clientseitige Setup-Programm ist unter dem Begriff Anwendungsproxy bekannt.

Sie generieren einen Anwendungsproxy, indem Sie das *Exportieren*-Kommando des administrativen Hilfsmittels *Komponentendienste* ausführen lassen. Wenn Sie die *Exportieren*-Anweisung auswählen, um einen Anwendungsproxy zu erstellen, generiert COM+ eine .MSI-Datei (eine Windows Installer-Datei). Beachten Sie, dass das Setup-Programm unter Windows 2000 anders ausgeführt wird als auf Computern, die frühere DCOM-Versionen benutzen. Unter Windows 2000 schreibt das Programm Einträge für die CLSID und ProgID in die RegDB und Einträge für die Typbibliothek und IID in die Windows-Registrierung. Da frühere Versionen von Windows nicht über die RegDB verfügen, fügt das Setup-Programm hier alle Einträge zur Windows-Registrierung hinzu.

Sie werden eine neue Applikation im Ordner *COM+-Anwendungen* des administrativen Hilfsmittels *Komponentendienste* bemerken, nachdem ein Anwendungsproxy auf einem Windows 2000-Computer installiert wurde. Jeder Windows 2000-Computer besitzt eine *Applications*-Auflistung, die zusätzlich zu Serverapplikationen und Bibliotheksapplikationen Applikations-Proxies enthält. Der Unterschied besteht darin, dass ein Anwendungsproxy die CLSIDs für eine Gruppe mit Komponenten aufnimmt, die auf anderen Maschinen vorhanden sind.

Sie sollten einige weitere wichtige Aspekte beachten, die Applikations-Proxies betreffen. Sie können zum Beispiel explizit den Remote-Servernamen setzen, der in den Anwendungsproxy integriert wird. Wenn Sie diesen Wert nicht explizit setzen, verwendet COM+ einfach den Namen des Computers, auf dem der Anwendungsproxy erstellt wird. Wenn Sie wünschen, dass der Anwendungsproxy auf einen anderen Computer verweist, sollten Sie den Remote-Servernamen mit dem administrativen Hilfsmittel *Komponentendienste* ändern, bevor der Anwendungsproxy generiert wird. Sie können den Remote-Servernamen auf der Registerkarte *Optionen* des Dialogfeldes *Eigenschaften von Arbeitsplatz* angeben.

Ein zweiter Aspekt ist, dass Ihr Client-Computer möglicherweise eine aktualisierte Version des Windows Installers benötigt, um einen Anwendungsproxy installieren zu können. Windows NT, Windows 98 und Windows 95 benötigen eine aktualisierte Version. Wenn Sie einen Fehler während der Installation eines Applikations-Proxies erhalten, finden Sie das erforderliche Update im Plattform SDK für Windows 2000. Sobald Sie die aktuelle Version des Windows Installers eingerichtet haben, sollte der Anwendungsproxy problemlos installiert werden können.

Stellen Sie drittens sicher, dass alle relevanten Typbibliotheken zur Serverapplikation hinzugefügt worden sind, bevor der Anwendungsproxy erstellt wird. Achten Sie darauf, Komponenten zu einer Serverapplikation hinzuzufügen, indem Sie sie installieren und nicht importieren. Die Installation bettet die Typbibliotheken im Gegensatz zum Importieren in Ihre ActiveX-DLL ein. Wenn Sie benutzerdefinierte Typbibliotheken mit IDL und dem MIDL-Compiler erstellen, sollten Sie

diese ebenfalls zur Serverapplikation hinzufügen, bevor der Anwendungsproxy generiert wird. Dies kann zunächst ein wenig kompliziert sein, da Ihnen das administrative Hilfsmittel *Komponentendienste* lediglich erlaubt, eine *.TLB*-Datei nur dann in eine Serverapplikation zu installieren, wenn Sie Komponenten aus einer DLL installieren.

Der vierte und letzte Aspekt betrifft einen Fehler, der Sie daran hindert, Applikations-Proxies in bestimmten Situationen zu erstellen. Die Client-Computer benötigen Typlibibliotheken, aber nicht die DLLs, die Ihre konfigurierten Komponenten enthalten. Wenn Sie die Option *Remote-Server-Dateien* in einem ActiveX-DLL-Projekt selektieren, generiert die Visual Basic-IDE eine allein stehende *.TLB*-Datei und die Typlibibliothek, die in die DLL integriert wird. Sie können eine allein stehende *.TLB*-Datei nur dann zu einer Serverapplikation hinzufügen, wenn Sie auch die Komponenten aus einer DLL hinzufügen. Der Grund hierfür besteht darin, dass der Anwendungsproxy daran gehindert werden soll, die gesamte DLL auf jeden Client-Computer zu kopieren.

Es gibt jedoch ein Problem: Wenn Sie die *.TLB*-Datei zusätzlich zur DLL zu einer Serverapplikation hinzufügen, kann COM+ den Anwendungsproxy nicht erstellen, da eine Inkompatibilität zwischen den von COM+ und den von Visual Basic generierten Typlibibliotheken besteht. Dieser Fehler kann sehr frustrierend sein, da unter dem MTS alles funktioniert, unter COM+ jedoch nicht. Ich habe leider keine Informationen darüber, wann dieser Fehler korrigiert wird.

Was sind die Folgen dieses Fehlers? Wenn Sie Applikations-Proxies benutzen, um clientseitige Computer einzurichten, müssen Sie all Ihre DLLs auf den Client-Computer kopieren. Dies kann zu einem Sicherheitsproblem führen, wenn Ihre DLLs sensitive Kennwörter enthalten. Ein Benutzer könnte Ihr Sicherheitsschema übergehen, indem er die DLL in einer lokalen Serverapplikation installiert und eine bestimmte Methode ausführt.

Eine Lösung dieses Problems besteht darin, all Ihre Kennwörter aus Ihren kompilierten DLLs zu entfernen und stattdessen in deklarativen Konstruktorzeichenfolgen zu benutzen. Eine zweite Lösung ist, COM+-Applikations-Proxies nicht zu verwenden, wenn Sie Client-Computer konfigurieren. Sie können ersatzweise Typlibibliotheken kopieren und die erforderlichen Konfigurationsinformationen mit einer anderen Technik auf den Client-Computern vermerken. Visual Basic bietet eine alternative Technik zum Einrichten von Client-Computern an, die *.VBR*-Dateien und das Programm *CLIREG32.EXE* nutzt.

Verteilte Speicherbereinigung

Was geschieht, wenn eine Client-Applikation abstürzt, die eine ausstehende Referenz auf ein Objekt in einer Remote-Serverapplikation besitzt? Das Objekt muss feststellen können, ob eine Client-Applikation noch gültig ist. Im vierten Kapitel wurde erwähnt, dass ein Client ermitteln kann, ob ein Objekt noch aktiv ist, indem er die entsprechenden *HRESULT*-Werte untersucht. Ein Objekt benötigt jedoch ein wenig mehr Hilfe, um festzustellen, ob ein Client abgestürzt ist.

COM stellt eine Infrastruktur für die verteilte Speicherbereinigung zur Verfügung, die den serverseitigen Computer ermitteln lässt, ob ein Client mit einer ausstehenden Objektreferenz abgestürzt ist. Wenn der serverseitige Computer dies feststellt, informiert er das Objekt mit einem Aufruf von *Release*. Dies bedeutet, dass ein Remote-Objekt automatisch aus dem Speicher entfernt wird, wenn sein Client abstürzt. Die Speicherbereinigung ist in verteilten Applikationen, die über sehr lange Zeiträume hinweg ausgeführt werden sollen, extrem wichtig.

Der COM-Mechanismus für die verteilte Speicherbereinigung basiert darauf, dass die Client-Maschine eine Nachricht an die Server-Maschine sendet, deren Wortlaut mit »Ich lebe noch.« übersetzt werden könnte. Diese Nachricht wird in Abständen von zwei Minuten wiederholt vom Cli-

ent zum Server geschickt. Wenn der Server sechs Minuten (drei nicht erhaltene Nachrichten) nichts vom Client hört, informiert er das Objekt, dass der Client abgestürzt ist.

Dieser so genannte Ping-Algorithmus wurde optimiert, um einen übermäßigen Netzwerkverkehr zu vermeiden. Ping-Nachrichten werden nicht für individuelle Schnittstellenreferenzen oder individuelle Objekte gesendet. Der Client-Computer überträgt stattdessen einen einzelnen maschinenweiten Ping an den Server. Dieser Ping enthält Informationen über jede Verbindung zwischen den beiden Maschinen. Beachten Sie bitte, dass ein Ping nicht alle Informationen über jede ausstehende Schnittstellenreferenz übermittelt. Stattdessen werden Informationen gesendet, die Aufschluss darüber geben, was sich seit dem letzten Ping verändert hat. Dieser »Delta«-Algorithmus reduziert die über das Netzwerk zu übertragenden Daten auf das Wesentliche.

Verteilte Applikationen strukturieren

Beachten Sie, dass jede Serverapplikation mit genau einem Prozess pro Computer verknüpft ist. Die aus den Komponenten derselben Serverapplikation erstellten Objekte werden im selben Prozess ausgeführt und können deshalb prozessspezifische Ressourcen gemeinsam nutzen. Die aus unterschiedlichen Serverapplikationen generierten Objekte werden in separaten Prozessen ausgeführt und können somit nicht prozessspezifische Ressourcen gemeinsam nutzen.

Wie also entscheiden Sie sich zwischen Serverapplikationen und Bibliotheksapplikationen? Und wie viele Serverapplikationen sollten Sie erstellen, wenn Sie sich für diese entscheiden? Sollten Sie fünf Komponenten zur selben oder jede Komponente zu einer eigenen Serverapplikation hinzufügen? Sie können zwischen verschiedenen Möglichkeiten wählen, von denen jede Vor- und Nachteile hinsichtlich der Leistung, Fehlertoleranz und Sicherheit bietet.

Der wichtigste Vorteil einer Bibliotheksapplikation besteht darin, dass sie eine bessere Leistung bietet, da sie die Notwendigkeit zu prozessübergreifenden Aufrufen beseitigt oder vermindert. Bibliotheksapplikationen unterstützen jedoch nicht die Remote-Aktivierung. Die Client-Applikation muss auf dem lokalen Computer ausgeführt werden, um ein Objekt aus einer Bibliotheksapplikation aktivieren zu können.

Jede Serverapplikation benötigt einen eigenen Prozess und somit eine eigene Ebene der Fehlertoleranz. Sie können sich eine Serverapplikation als eine eigene Fehlerisolutions-Domäne vorstellen. Ein Objekt, das abstürzt, kann alle Objekte in derselben Serverapplikation »mit sich reißen«. Ein Absturz in einem Prozess wirkt sich aber nicht unbedingt auf die Objekte aus, die in anderen Prozessen ausgeführt werden.

Um eine zusätzliche Ebene der Fehlertoleranz zur Verfügung zu stellen, verfügt jeder COM+-Prozess über eine so genannte Failfast-Richtlinie. Wenn die COM+-Laufzeit feststellt, dass während der Ausführung einer Methodenimplementierung ein unerwarteter Fehler auftritt, geht sie davon aus, dass der gesamte Prozess fehlerhaft ist. COM+ beendet den Prozess daraufhin. Die Failfast-Richtlinie wird auch dann angewendet, wenn eine Komponente einen Win32-Fehler oder eine C++-Ausnahme auslöst. Sie können die Failfast-Richtlinie jedoch nicht selbst initiieren, indem Sie einen Visual Basic-Fehler mit `Err.Raise` auslösen. Visual Basic-Fehler (die, wie Sie wissen, in Wirklichkeit COM-Ausnahmen sind) werden ohne Probleme an den Aufrufer weitergeleitet. Beachten Sie deshalb, dass COM+ eine Fehlermeldung in das Windows-Ereignisprotokoll schreibt, wenn es einen Prozess gemäß der Failfast-Richtlinie terminiert.

Sie können all Ihre Komponenten einer einzigen Serverapplikation hinzufügen oder über mehrere Serverapplikationen verteilen. Der Nachteil besteht darin, dass Aufrufe über Prozessgrenzen hinweg sehr viel mehr Zeit als prozessinterne Aufrufe benötigen. Wenn Sie sich dazu entschließen, mehrere Serverapplikationen auf verschiedenen Computern zu benutzen, können Sie den

Bearbeitungsaufwand zwischen mehreren Prozessoren aufteilen. Der offenkundige Nachteil dieser Lösung ist, dass Aufrufe über Computergrenzen hinweg noch aufwendiger als lokale prozessübergreifende Aufrufe sind.

Wann also würden Sie eine Bibliotheksapplikation einsetzen? Zwei Szenarien sind üblich. Im ersten Szenario besitzen Sie eine Hilfskomponente, die von zwei oder mehreren Serverapplikationen gemeinsam genutzt werden soll. Wenn Sie die Komponente zu einer Bibliotheksapplikation hinzufügen, kann ein Objekt, das in jeder beliebigen Serverapplikation ausgeführt wird, ein prozessinternes Objekt aus der Hilfskomponente erstellen. Sie können eine Komponente somit mehreren Serverapplikationen für die gemeinsame Nutzung zur Verfügung stellen, während gleichzeitig aufwendige prozessübergreifende Aufrufe vermieden werden.

Im zweiten Szenario benutzen Sie eine Bibliotheksapplikation, um Objekte in einer anderen Host-Applikation als *DLLHOST.EXE* zu erstellen. Stellen Sie sich zum Beispiel vor, Sie benutzen eine Komponente, die in einer Web-Applikation eingesetzt werden soll. Der Client ist eine ASP-Seite, die im IIS-Webserverprozess *INETINFO.EXE* ausgeführt wird. Wenn Sie die Komponente in einer Bibliotheksapplikation konfigurieren, werden Ihre Objekte im Webserverprozess, und nicht in einer Instanz von *DLLHOST.EXE* erstellt, wie in Abbildung 6.6 dargestellt. Sie erzielen dann eine bessere Leistung, da Sie die Notwendigkeit prozessübergreifender Aufrufe beseitigt haben. Die Fehlertoleranz ist in diesem Fall jedoch nicht sehr gut. Ihre Komponente könnte den gesamten Webserverprozess abstürzen lassen.

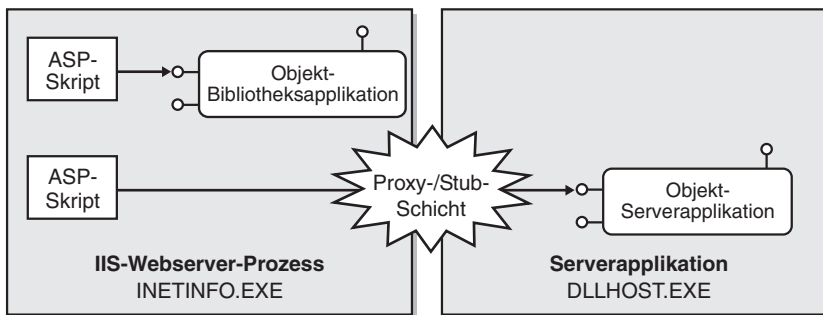


Abbildung 6.6: In einer auf einer ASP-Seite basierenden Webapplikation bieten Bibliotheksapplikationen in der Regel eine bessere Leistung, da sie die Notwendigkeit prozessübergreifender Aufrufe beseitigen. Serverapplikationen bieten eine bessere Fehlertoleranz und die Möglichkeit, Komponenten auf einem separaten Computer ausführen zu lassen.

Kontexte verstehen

Dieser Abschnitt ist für Ihr Verständnis der Arbeitsweise von konfigurierten Komponenten wichtig. Die attributbasierte Programmierung ist auf den so genannten Abfangmechanismus angewiesen. Wenn ein Client ein Objekt aus einer Komponente erstellt, die in einer COM+-Applikation konfiguriert ist, fügt die zugrunde liegende Laufzeit eine Abfangschicht zwischen den beiden ein. Diese Schicht repräsentiert eine Verbindung, die der COM+-Laufzeit gestattet, eine vom System zur Verfügung gestellte Vor- und Nachbearbeitung für einzelne Methodenaufrufe durchzuführen. Das System übernimmt die Programmsteuerung direkt bevor und nachdem ein Objekt die eigene Methodenimplementierung ausführt.

Die COM+-Abfangschicht basiert auf so genannten Kontexten. Ein Kontext ist, einfach beschrieben, eine Gruppe mit Objekten, die in einem Prozess ausgeführt werden. Prozesse sind eigentlich

in Apartments und Apartments in Kontexte unterteilt. (Apartments werden im nächsten Kapitel besprochen.) Die COM+-Laufzeit initiiert den Abfangvorgang, wenn ein Aufruf die Kontextgrenzen überschreitet.

Jedes in einer COM+-Applikation erstellte Objekt wird innerhalb des Gültigkeitsbereichs eines spezifischen Kontextes generiert, wie in Abbildung 6.7 dargestellt. COM+ fängt Aufrufe jedoch nicht ab, wenn sich ein Aufrufer und ein Objekt im selben Kontext befinden. COM+ fängt nur solche Aufrufe ab, die die Kontextgrenzen verlassen. Der Begriff Kontextwechsel bezieht sich auf einen vom System bereitgestellten Code, der automatisch ausgeführt wird, wenn ein Aufruf die Kontextgrenze überschreitet.

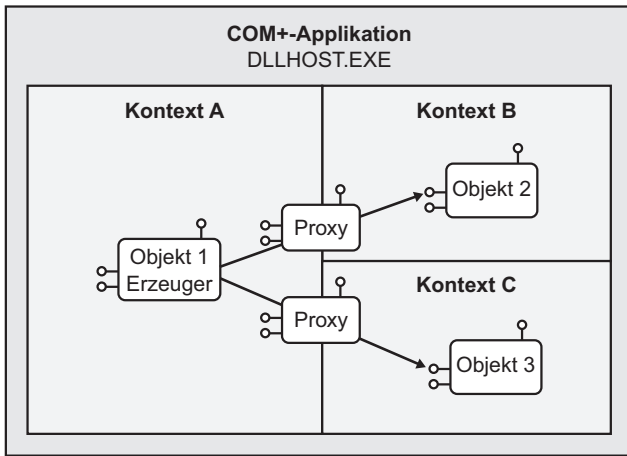


Abbildung 6.7: Die COM+-Laufzeit richtet ein Abfangschema ein, das auf Kontexten basiert. Ein vom System zur Verfügung gestellter Proxy führt immer dann einen Kontextwechsel durch, wenn ein Aufruf über die Kontextgrenze hinweg ausgeführt wird.

Thread-Wechsel-Proxies im Vergleich mit einfachen Proxies

Wenn ein Objekt ein anderes Objekt in einem separaten Prozess erstellt, befinden sich beide Objekte per definitionem in unterschiedlichen Kontexten. Die COM+-Laufzeit erstellt deshalb den entsprechenden Abfangcode in der Proxy-/Stub-Schicht. Beachten Sie, dass diese Windows 2000-Version einer Proxy-/Stub-Schicht weiter entwickelt als die am Ende des dritten Kapitels beschriebene Schicht ist. Das Proxy-/Stub-Paar hat nun zwei Verantwortlichkeiten: Es muss zuerst einen Methodenaufruf über Prozess- und Host-Grenzen hinweg verarbeiten, indem es einen Thread-Wechsel durchführt und Parameter verpackt und versendet. Zweitens muss es den erforderlichen Kontextwechsel durchführen. Dies ist, verglichen mit den früheren Windows-Versionen, in denen die Proxy-/Stub-Schicht nichts von Kontexten oder Abfangvorgängen weiß, eine große Veränderung.

Windows 2000 führt einen neuen Proxytyp ein, um der Situation gerecht zu werden, in der der Aufrufer und das Objekt in unterschiedlichen Kontexten, aber im selben Prozess auf demselben Thread ausgeführt werden. Wie Sie in Abbildung 6.6 sehen, ist hierfür kein Proxy-/Stub-Paar, sondern lediglich ein einfacher Proxy erforderlich. Dieser Proxy führt einen Kontextwechsel, aber keinen Threadwechsel durch. Die COM+-Laufzeit erstellt, wie für den Thread-Wechsel-Proxy,

einen einfachen Proxy, indem sie zur Laufzeit eine Schnittstellendefinition in einer Typbibliothek überprüft.

HINWEIS: VON MTS ZU COM+

Kontext-Wrapper gibt es nicht mehr.

Die Abfangschicht wird in MTS mit Hilfe eines nicht sichtbaren, vom System zur Verfügung gestellten Objekts mit der Bezeichnung Kontext-Wrapper eingerichtet. Der MTS-Kontext-wrapper führt, ebenso wie die Kontextproxies in COM+, die Vor- und Nachbearbeitung für jeden Methodenaufruf durch. In MTS wird jedoch nur ein Kontext-Wrapper erzeugt, wenn ein Client ein neues Objekt aktiviert. Dies bedeutet, dass nur ein Kontext-Wrapper für jedes MTS-Objekt vorhanden ist. In COM+ kann ein Objekt über mehrere verknüpfte Kontextproxies verfügen. Das neue Abfangschema in Windows 2000 ist weitaus flexibler, da die COM+-Laufzeit Kontextproxies für ein Objekt generiert, wenn diese benötigt werden.

Wann werden Kontexte erzeugt?

Kontexte werden gewöhnlich von dem SCM generiert, wenn ein Client ein neues Objekt mit einem Aufruf der COM-Bibliotheksfunktion `CoCreateInstance` aktiviert. Wie Sie aus den vorherigen Kapiteln wissen, ruft Visual Basic `CoCreateInstance` auf, wenn eine Client-Applikation ein Objekt mit der `CreateObject`-Funktion oder dem `New`-Operator erstellt. (Es gibt eine Besonderheit des `New`-Operators, die ich gleich erläutern werden.) Sie sollten jedoch beachten, dass der SCM nicht immer einen neuen Kontext erstellt, wenn Sie ein Objekt erzeugen.

Lassen Sie uns die grundlegenden Regeln besprechen. Jedes in einer COM+-Applikation aktivierte Objekt wird innerhalb eines zulässigen Kontextes erstellt. Der SCM verfügt über drei verschiedene Möglichkeiten, um auf eine Aktivierungsanforderung zu reagieren:

- Das neue Objekt wird in dem Kontext des Erstellers erzeugt.
- Das Objekt wird in einem neu erstellten Kontext erzeugt.
- Das neue Objekt wird in dem Standardkontext eines anderen Apartments erstellt.

Die beiden ersten Möglichkeiten sind weitaus gebräuchlicher als die dritte. Werfen wir also zuerst einen Blick hierauf. Der SCM muss während der Aktivierung entscheiden, ob ein neues Objekt eigene kontextuelle Informationen benötigt, um seine COM+-Attribute zu beschreiben. Wenn COM+ einen Kontext erstellt, erzeugt es ein privates Objekt, das als Objektkontext bezeichnet wird. COM+ benutzt dieses Objekt, um kontextspezifische Informationen für alle Objekte zu speichern, die sich auf den entsprechenden Kontext beziehen. Wenn ein neues Objekt eigene kontextuelle Informationen benötigt, erstellt COM+ einen neuen Kontext für das Objekt. Wenn ein neues Objekt diese eigenen kontextuellen Informationen nicht benötigt, fügt COM+ das Objekt zum Kontext des Erstellers hinzu.

In den meisten Fällen wird ein Objekt, das aus einer nicht konfigurierten Komponente erzeugt wird, im Kontext des Erstellers angelegt, während ein Objekt, das aus einer konfigurierten Komponente erstellt wird, einen eigenen neuen Kontext erhält. Abbildung 6.8 zeigt was geschieht, wenn ein COM+-Objekt ein ADO-Objekt wie zum Beispiel eine Verbindung oder eine Datensatzgruppe erstellt. Wie jedes andere Objekt auch, das aus einer nicht konfigurierten Komponente erzeugt wird, benötigt ein ADO-Objekt keinen eigenen Kontext, da es nicht über deklarative Attribute verfügt.

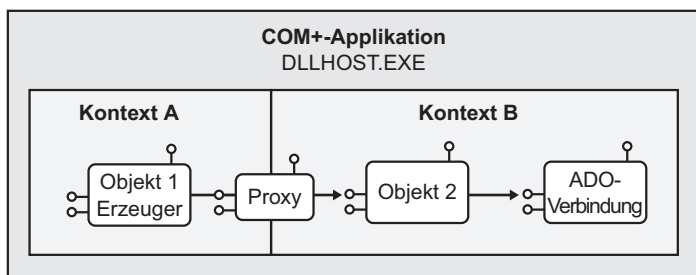


Abbildung 6.8: Ein Objekt, das aus einer konfigurierten Komponente erstellt wird, erhält in den meisten Fällen einen eigenen neuen Kontext, während ein Objekt, das aus einer nicht konfigurierten Komponente generiert wird, im Kontext des Erzeugers erstellt wird.

Ein Objekt, das aus einer konfigurierten Komponente erstellt wird, benötigt in beinahe jeder Situation einen eigenen Kontext, da die COM+-Laufzeit auf einige Informationen angewiesen ist, um den Abfangcode korrekt ausführen und zuverlässige Systemdienste anbieten zu können. Der Objektkontext enthält wertvolle Informationen, zum Beispiel über die Verknüpfung eines Objekts mit einer spezifischen Aktivität und einer bestimmten Transaktion.

HINWEIS: VON MTS ZU COM+

Aufrufe von `CreateInstance` sind nicht mehr erforderlich.

Eine der signifikantesten Veränderungen, die die Programmierung betrifft und mit der Sie konfrontiert werden, wenn Sie von MTS zu COM+ wechseln, besteht darin, dass Sie die `CreateInstance`-Methode der `ObjectContext`-Schnittstelle nicht mehr aufrufen müssen. In MTS mussten Sie `CreateInstance` aufrufen, um eine Objektkontextanforderung an die MTS-Laufzeit anstatt an den SCM zu schicken.

In COM+ ist der SCM kontextfähig. Dies bedeutet, dass, im Gegensatz zum SCM unter Windows NT 4, der in Windows 2000 integrierte SCM weiß, wie ein Objekt in einem entsprechenden Kontext aktiviert werden muss. Aufrufe von `CreateObject` werden korrekt durchgeführt. Aufrufe von `CreateInstance` werden weiterhin unterstützt, jedoch nur aus Gründen der Abwärtskompatibilität. Ein Aufruf von `CreateInstance` führt zu denselben Ergebnissen wie ein Aufruf von `CreateObject`.

Von den drei Möglichkeiten, die dem SCM zur Verfügung stehen, um eine Aktivierungsanforderung zu bearbeiten, wird die letzte nur selten genutzt. Der SCM erstellt nur dann ein neues Objekt im Standardkontext eines anderen Apartments, wenn der Erzeuger und das zu erstellende Objekt inkompatible Threading-Modelle besitzen und deshalb nicht auf dem selben Thread ausgeführt werden können. Ein Beispiel hierfür ist ein Apartment-Threaded-Objekt, das ein Objekt aus einer nicht konfigurierten Komponente erstellt, die Free-Threaded ist. Das neue Objekt kann nicht im Kontext des Erzeugers ausgeführt werden, da der falsche Apartmenttyp vorliegt. (Ein Kontext muss sich in einem spezifischen Apartment befinden.) COM+ muss jedoch keinen neuen Kontext für das Objekt erstellen, da es keine COM+-Attribute aufweist. Die COM+-Laufzeit erstellt in solchen Situationen ein neues Objekt im Standardkontext eines anderen Apartments. Wenn Sie nur Apartment-Threaded-Komponenten verwenden wie die, die mit Visual Basic generiert werden, müssen Sie sich um das dritte Szenario keine Gedanken machen.

Achten Sie auf den New-Operator

Sie sollten vorsichtig sein, wenn Sie Code in eine konfigurierte Komponente schreiben, der `New` auf einer anderen konfigurierten Komponente aufruft. Wenn Sie den `New`-Operator verwenden, müssen Sie zwei Szenarien berücksichtigen. Im ersten Szenario befinden sich die Erzeugerkomponente und die Komponente, die zum Erstellen des neuen Objekts benutzt wird, in separaten DLLs. In diesem Fall wird ein Aufruf von `New`, genau wie ein Aufruf von `CreateObject`, an den SCM weitergeleitet. Dies erlaubt dem SCM, das neue Objekt in der korrekten Weise zu erstellen. Im zweiten Szenario werden die Erzeugerkomponente und die Komponente, aus der das neue Objekt generiert wird, in dieselbe DLL kompiliert. Das hierbei auftretende Problem ist sehr subtil, kann aber zu großer Frustration führen, wenn Sie zu ermitteln versuchen, weshalb Ihr Code nicht korrekt ausgeführt wird.

Wenn ein Visual Basic-Objekt `New` für einen Klassennamen aufruft, der in dieselbe DLL kompiliert wurde, erstellt die Visual Basic-Laufzeit das Objekt und bindet es an sich selbst, ohne den SCM zu involvieren. Das neue Objekt ist somit nicht korrekt vom SCM erzeugt worden und die COM+-Laufzeit weiß überhaupt nicht, dass es existiert. COM+ besitzt nicht die Möglichkeit, das Abfangschema in der angemessenen Weise einzurichten. Ihr Code kann sich somit seltsam verhalten, wenn Sie `New` zur falschen Zeit aufrufen.

Wie Sie sehen, können Sie den `New`-Operator bisweilen mit den Methoden einer COM+-Komponente verwenden. Sie können `New` immer benutzen, um Objekte aus Komponenten zu erstellen, die sich außerhalb der DLL des Erzeugers befinden. Sie können `New` zum Beispiel immer aufrufen, um ein neues ADO-Objekt zu generieren.

Kontexte und Objektreferenzen

Die COM+-Regeln, die Objektreferenzen betreffen, sind sehr einfach. Ein Client wird direkt an das Objekt gebunden, wenn beide im selben Kontext vorhanden sind. Ein Client muss über einen vom System zur Verfügung gestellten Proxy mit einem Objekt kommunizieren, wenn sich beide in unterschiedlichen Kontexten befinden. Doch wann und wie werden diese Proxies erstellt? Die Antwort lautet: Die COM+-Laufzeit erstellt Proxies transparent, wenn diese benötigt werden.

Als COM+-Programmierer stehen Ihnen in der Regel zwei Möglichkeiten zur Verfügung, die zu einer transparenten Erstellung eines kontextfähigen Proxies führen: Sie aktivieren neue Objekte aus einer konfigurierten Komponente, indem Sie die `CreateObject`-Funktion verwenden, und Sie übergeben Objektreferenzen mit Hilfe von Parametern und Rückgabewerten in Standardmetho-
denaufrufen.

Wenn Sie ein Objekt aus einer konfigurierten Komponente erstellen, wird die COM+-Laufzeit sehr wahrscheinlich einen neuen Kontext erzeugen. Die COM+-Laufzeit gibt jedoch keine direkte Referenz an den Erzeuger zurück. Wenn COM+ feststellt, dass das Objekt und der Erzeuger in unterschiedlichen Kontexten vorhanden sind, generiert es einen kontextfähigen Proxy und gibt diesen an den Erzeuger zurück. Dies bedeutet, dass alle Aufrufe vom Erzeuger an das Objekt über den Proxy geleitet werden. Der Proxy führt den erforderlichen Kontextwechsel immer dann zur Laufzeit aus, wenn die Steuerung zwischen diesen beiden Kontexten übergeben und zurückgegeben wird.

Wie Sie gesehen haben, erstellt COM+ automatisch einen Proxy, wenn ein Objekt in einem anderen Kontext als dem des Erzeugers generiert wird. Die zweite übliche Situation, in der COM+ einen Proxy erstellt, tritt ein, wenn eine Objektreferenz in einem Standardmethodenaufruf über Kontextgrenzen hinweg übergeben wird. Da die COM+-Laufzeit alle Aufrufe, die die Kontextgrenzen verlassen, abfängt, kann sie ermitteln, ob der Empfänger einer Objektreferenz einen neuen Proxy benötigt.

Lassen Sie mich Ihnen ein Beispiel zeigen, um das soeben Beschriebene zu demonstrieren. Werfen Sie einen Blick auf Abbildung 6.9. Stellen Sie sich vor, Objekt 1 erstellt Objekt 2 mit einem Aufruf der Funktion `CreateObject`. Objekt 1 befindet sich in Kontext A, während Objekt 2 in Kontext B vorhanden ist. Stellen Sie sich nun vor, dass Objekt 1 die folgende Methode von Objekt 2 aufruft:

```
Function CreateObject3() As Class3
    Dim obj As Class3
    Set obj = CreateObject("MyDll.Class3")
    Set CreateObject3 = obj
End Function
```

Wenn Objekt 2 das dritte Objekt erstellt, wird eine Referenz an den Proxy übergeben, der weiß, wie ein Kontextwechsel zwischen Kontext B und Kontext C durchgeführt werden muss. Sie sollten beachten, dass Objekt 1 diesen Proxy nicht nutzen kann, da es sich in Kontext A befindet. Objekt 1 benötigt einen anderen Proxy, der einen Kontextwechsel zwischen Kontext A und Kontext C durchführen kann. Alles funktioniert jedoch problemlos. Wenn der Aufruf von `CreateObject3` die Steuerung von Kontext B an Kontext A zurückgibt, erzeugt die COM+-Laufzeit automatisch einen neuen Proxy für das dritte Objekt, das anschließend in Kontext A verwendet werden kann.

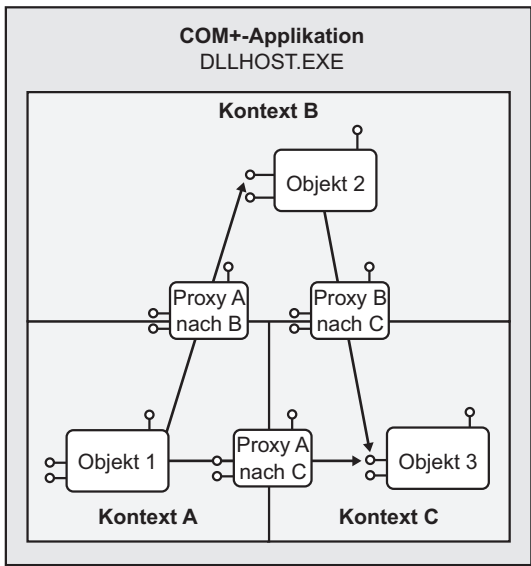


Abbildung 6.9: Objektreferenzen sind immer kontextrelativ. Sie können sich einen kontextfähigen Proxy als einen Adapter zwischen zwei spezifischen Kontexten vorstellen.

Dieses Beispiel zeigt einen wichtigen Aspekt des COM+-Programmiermodells: Alle Objektreferenzen sind kontextrelativ. Wenn Sie die Regeln befolgen, ist alles ganz einfach. Wenn eine Objektreferenz von zwei oder mehreren Objekten gemeinsam genutzt werden soll, können Sie Methoden entwickeln, um die Referenz zwischen den Objekten zu übergeben. Sie sollten dennoch aufmerksam sein. Sie können in Schwierigkeiten geraten, wenn Sie versuchen, Objektreferenzen, die andere Techniken verwenden, für die gemeinsame Nutzung freizugeben. Es ist zum Beispiel eine schlechte Idee, eine Objektreferenz in einer öffentlichen Variablen eines `.BAS`-Moduls anzulegen.

Lassen Sie uns zu unserem Beispiel zurückkehren: Was geschieht, wenn Objekt 2 das dritte Objekt erstellt und die zurückgegebene Referenz in einer *.BAS*-Modulvariablen speichert? Sie würden in diesem Fall feststellen, dass die Referenz nur von dem Code verwendet werden kann, der in Kontext B ausgeführt wird.

Wenn Objekt 1 auf die *.BAS*-Modulvariable zugreift und die Referenz benutzt, führt dies zu einem Fehlverhalten. Objekt 1 benötigt einen Proxy, der den Kontextwechsel zwischen Kontext A und Kontext C durchführt. Die in der *.BAS*-Modulvariablen gespeicherte Referenz verweist jedoch auf einen Proxy, der für den Kontextwechsel zwischen Kontext B und Kontext C verantwortlich ist. Wie Sie sehen, lässt dieser Programmierstil das von COM+ eingerichtete Abfangschema unbrauchbar werden. Es ist ebenso keine gute Idee, eine Objektreferenz mit anderen gemeinsam genutzten Speichermechanismen und unter Verwendung des SPM (Shared Property Manager) zu speichern. (Sie erfahren im nächsten Kapitel, wie der SPM eingesetzt wird. Sie benutzen ihn, um reine Daten, und nicht um Objektreferenzen zu speichern.)

HINWEIS: VON MTS ZU COM+

Aufrufe an SafeRef sind nicht länger erforderlich.

In MTS muss jedes Objekt mit einem verknüpften Kontext-Wrapper *SafeRef* aufrufen, wenn einer anderen Entität eine Referenz auf sich selbst übergeben wird. Der Aufruf von *SafeRef* übergibt einen Verweis auf den Kontext-Wrapper anstelle des reinen Verweises auf das Objekt selbst. Wenn *SafeRef* nicht zum richtigen Zeitpunkt aufgerufen wird, kann dies das MTS-Abfangschema verletzen.

Die Art und Weise, in der COM+ das Abfangschema einrichtet, ist weitaus transparenter und flexibler. *SafeRef* muss nicht mehr aufgerufen werden. Solange Sie die in diesem Kapitel genannten Regeln befolgen, erstellt die COM+-Laufzeit immer dann kontextfähige Proxies, wenn diese benötigt werden.

Das COM+-Programmiermodell

Wenn Sie konfigurierte Komponenten erstellen, sollten Sie die Typbibliothek *COM+ Services Type Library* in Ihrem ActiveX-DLL-Projekt referenzieren. Diese Typbibliothek definiert viele der Schnittstellen, die Sie benutzen, wenn Sie mit der COM+-Laufzeit interagieren. Tabelle 6.1 führt die kontextbezogenen Schnittstellen auf, die vorwiegend von Visual Basic-Programmierern eingesetzt werden.

Das COM+-Programmiermodell unterscheidet zwei Kontexttypen: Objektkontexte und Aufrufkontexte. Ein Objektkontext wird erzeugt, wenn der SCM einen neuen Kontext erstellt, der in der Regel über die verschiedenen Methodenaufrufe hinweg unverändert bleibt. Ein Aufrufkontext enthält aufrufspezifische Informationen und bewegt sich von Kontext zu Kontext, wenn die Steuerung von einer Methode an die nächste übergeben wird.

Schnittstelle	Typ	Verwendet	Steht in MTS zur Verfügung?
ObjectContext	Objektkontext und Aufrufkontext	Transaktionssteuerung. Programmgesteuerte Sicherheit. Abruf von Verweisen auf ASP-Objekte, wie zum Beispiel Request, Response und Application.	Ja
ContextInfo	Objektkontext	Abruf von kontextuellen Informationen über den aktuellen Kontext, die Aktivität und die Transaktion.	Nein

Schnittstelle	Typ	Verwendet	Steht in MTS zur Verfügung?
IContextState	Objektkontext	Transaktionssteuerung.	Nein
SecurityCallContext	Aufrufkontext	Programmgesteuerte Sicherheit.	Nein

Tabelle 6.1: COM+-Kernschnittstellen, die von Visual Basic-Programmierern verwendet werden.

Programmieren mit dem Objektkontext

Jeder Kontext erhält einen Objektkontext. Dieses vom System zur Verfügung gestellte COM-Objekt nimmt kontextbezogene Informationen auf (siehe Abbildung 6.10). Der Objektkontext bietet außerdem Methoden an, die Ihnen die Interaktion mit der COM+-Laufzeit gestatten. Einige dieser Methoden erlauben Ihren Objekten, die COM+-Laufzeit nach Informationen abzufragen, während andere Methoden die COM+-Laufzeit anweisen, bestimmte Aufgaben für Sie durchzuführen.

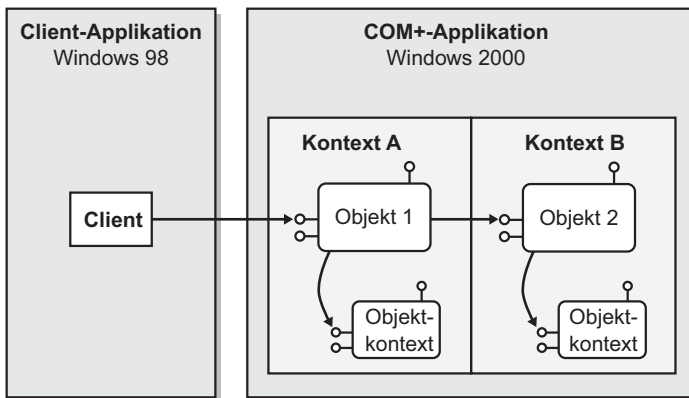


Abbildung 6.10: Ein Objekt kann einen Verweis auf seinen Objektkontext abrufen, indem es *GetObjectContext* aufruft.

Lassen Sie uns einen Blick auf ein kurzes Beispiel werfen. Das Schreiben von COM+-Transaktionen wird in einem späteren Kapitel beschrieben. Ich zeige Ihnen an dieser Stelle lediglich ein einfaches Beispiel für ein Objekt, das seine Arbeit beendet hat und die aktuelle Transaktion abschließen möchte. Betrachten Sie den folgenden Code:

```
Dim oc As COMSVCSLib.ObjectContext
Set oc = GetObjectContext
oc.SetComplete
```

Der Aufruf von *GetObjectContext* erlaubt einem Objekt, einen Verweis auf den eigenen Objektkontext abzurufen. Das Objekt in diesem Beispiel verwendet seinen Objektkontext, um der COM+-Laufzeit mitzuteilen, dass seine Arbeit getan ist und die Transaktion abgeschlossen werden sollte. Beachten Sie, dass Sie keine explizite *ObjectContext*-Referenz erstellen müssen. Nachfolgend ist eine gekürzte Version des obigen Codes aufgeführt, die die gleiche Funktion erfüllt:

```
GetObjectContext.SetComplete
```

Ein Aufruf von `GetObjectContext` wird sehr schnell durchgeführt, weil die COM+-Laufzeit einen Verweis auf den Objektkontext an einem Ort speichert, auf den sehr schnell zugegriffen werden kann. Dieser Ort wird als lokaler Thread-Speicher bezeichnet. Sie sollten nicht versuchen, Ihren Code zu optimieren, indem Sie die Anzahl der Aufrufe an `GetObjectContext` verringern. Die Geschwindigkeit wird nicht erhöht, wenn Sie zum Beispiel `GetObjectContext` aufrufen und die zurückgegebene Referenz in einer modulweit gültigen Variablen speichern. Diese Lösung ließe Ihren Code komplexer werden, ohne eine spürbar verbesserte Leistung zu bieten.

Beachten Sie, dass ein Objekt niemals aus einem fremden Kontext heraus auf einen Objektkontext zugreifen sollte. In Abbildung 6.10 sollte das zweite Objekt zum Beispiel keinen Verweis auf den Objektkontext für Kontext A abrufen. Dies bedeutet, dass Sie eine Referenz niemals mit einer Methode an einen Objektkontext übergeben sollten. Der einzige Kontext, den ein Objekt verwenden sollte, ist der, den es mit einem Aufruf von `GetObjectContext` abrufen.

Auch das MTS-Programmiermodell enthält die `ObjectContext`-Schnittstelle. Diese Schnittstelle arbeitet sowohl in MTS als auch in COM+ mit Objektkontexten und Aufrufkontexten. Als die Software-Architekten COM+ neu entwarfen, entschlossen sie sich dazu, einige neue Schnittstellen einzuführen, die entweder nur für Objektkontexte oder ausschließlich für Aufrufkontexte verwendet werden sollten.

Die `ContextInfo`-Schnittstelle ist beispielsweise eine spezifische Objektkontextschnittstelle. Sie erlaubt Ihnen, kontextuelle Informationen (primär die identifizierenden GUIDs) für den aktuellen Kontext sowie für die aktuelle Aktivität und Transaktion zu ermitteln. Sie können über die `ObjectContext`-Schnittstelle einen Verweis auf ein vom System zur Verfügung gestelltes Objekt abrufen, das die `ContextInfo`-Schnittstelle implementiert. Nachfolgend ist ein Beispiel für den Einsatz der `ContextInfo`-Schnittstelle aufgeführt, das die GUID ermittelt, die den aktuellen Kontext identifiziert:

```
Dim ci As COMSVCSLib.ContextInfo
Dim MyContextID As String
Set ci = GetObjectContext.ContextInfo
MyContextID = ci.GetContextId
```

Wie Sie sehen, ist das Objekt, das `ContextInfo` implementiert, im Objektkontext enthalten. Dies bedeutet, dass zwei separate Objekte vorhanden sind. In der COM-Sprache ausgedrückt: es existieren zwei separate Identitäten.

Der COM+-Objektkontext stellt nicht nur ein Unterobjekt zur Verfügung, das `ContextInfo` implementiert. Er implementiert auch andere Schnittstellen als `ObjectContext`. Auf einige dieser Schnittstellen kann nur über C++ zugegriffen werden. `IContextState` ist jedoch eine Schnittstelle, auf die Sie mit Visual Basic zugreifen können.

Die `IContextState`-Schnittstelle ist, wie `ContextInfo`, eine spezifische Objektkontextschnittstelle. Sie benutzen diese Schnittstelle, um Kontrolle über Ihre Transaktionen zu erhalten. Sie werden möglicherweise feststellen, dass diese Schnittstelle eine gewisse Funktionalität anbietet, die ebenfalls in der `ObjectContext`-Schnittstelle enthalten ist. Die Details hierzu werden im achten Kapitel beschrieben. Ich werde Ihnen jedoch bereits an dieser Stelle ein kurzes Beispiel für den Abruf und den Einsatz einer `IContextState`-Referenz zeigen:

```
Dim cs As COMSVCSLib.IContextState
Set cs = GetObjectContext
cs.SetMyTransactionVote TxCommit
```

Sie rufen die Referenz ab, indem Sie `GetObjectContext` aufrufen und den Rückgabewert dem `IContextState`-Typ zuordnen. Wenn Sie `GetObjectContext` aufgerufen haben, können Sie außerdem beliebig oft zwischen der `ObjectContext`- und der `IContextState`-Schnittstelle wechseln. Dies ist möglich, da beide zwei Schnittstellen repräsentieren, die von einer einzigen COM-Identität implementiert werden.

```
Dim oc As ObjectContext, cs As IContextState
Set oc = GetObjectContext
Set cs = oc
```

In diesem Beispiel richtet die zweite Zeile eine `ObjectContext`-Verbindung zu dem Objektkontext ein. Die nächste Zeile führt zu einem Aufruf von `QueryInterface`, der einen zweiten Verweis auf dasselbe Objekt ermittelt. Wenn Sie verstanden haben, wieso diese Art der Zuordnung funktioniert, werden Sie glücklich sein, so viel Zeit und Energie investiert zu haben, um mehr über die schnittstellenbasierte Programmierung zu lernen.

Aufrufkontexte verstehen

Während der Objektkontext an einem einzigen Ort verbleibt, bewegt sich der Aufrufkontext immer mit der Kette von Methodenaufrufen weiter. Lassen Sie uns hierzu ein Beispiel betrachten, um ein besseres Verständnis der wichtigen Konzepte zu erhalten. Stellen Sie sich vor, Sie haben eine formbasierte Client-Applikation erstellt, die auf einem Windows 98-Computer ausgeführt wird. Diese Client-Applikation generiert ein Objekt in einer COM+-Serverapplikation auf einem Windows 2000-Server. Dieses Objekt erstellt weitere Objekte (von denen einige auf einem anderen Windows 2000-Server vorhanden sind), wie in Abbildung 6.11 dargestellt. Wenn die Client-Applikation einen Methodenaufruf durchführt, der diese Objekte passiert, entsteht eine logische Aufrufkette über drei verschiedene Computer hinweg. Wenn die Steuerung über Kontext-, Prozess- und Computergrenzen hinweg übergeben wird, geschieht dies zusammen mit vielen kontextuellen Informationen.

Diese Übergabe von kontextuellen Informationen zusammen mit Methodenaufrufen wird durch die Kausalität möglich, die bereits Bestandteil von COM war, bevor es MTS und COM+ gab. Eine Kausalität ist als eine Kette von Objekt-RPC-Methodenaufrufen (ORPC) definiert. Wenn der Client auf dem Windows 98-Computer eine Methode aufruft, erstellt die COM-Laufzeit eine neue Kausalität und generiert eine GUID (die Kausalitäts-ID), um die Methode zu identifizieren, bevor der erste Ausgangsaufruf durchgeführt wird. Wenn der Aufruf über das Netzwerk übertragen wird, werden auch die Parameterwerte der Methode übermittelt. Die Kausalitäts-ID und andere sicherheitsbezogene Informationen werden ebenfalls automatisch weitergeleitet. Die Kausalität bleibt bestehen, wenn sich der Steuerungsfluss von Objekt 1 zu Objekt 2 weiter zu Objekt 3 und schließlich zurück zum Client bewegt. Es scheint, als würde ein großer Aufrufstack über das Netzwerk hinweg entstehen.

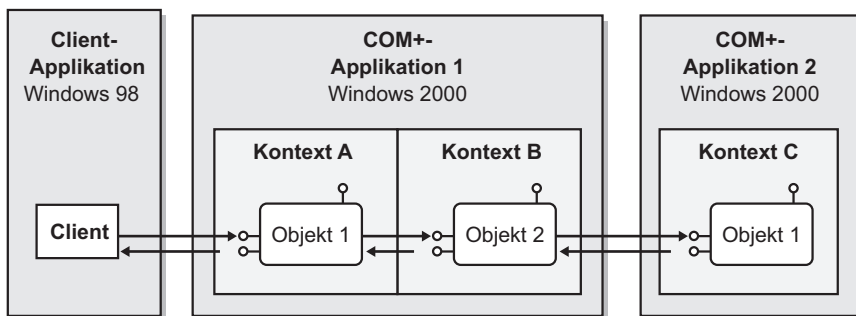


Abbildung 6.11: Wenn eine Kette von Methodenaufrufen über mehrere Objekte hinweg durchgeführt wird, repräsentiert dies eine Kausalität, die verwandte Informationen mit sich trägt. Diese Informationen werden als Aufrufkontext bezeichnet. In diesem Szenario wird immer dann eine neue Kausalität erzeugt, wenn der Client eine Methode aufruft.

Der primäre Grund für die Berücksichtigung des Aufrufkontextes bezieht sich auf die Sicherheit. Eine Schnittstelle nutzt den Aufrufkontext besonders häufig: `SecurityCallContext`. Diese Schnittstelle lässt ein Objekt ermitteln, ob die COM+-Sicherheit aktiviert ist und wo sich der Aufrufer befindet. Nachfolgend ist ein Beispiel für den Einsatz der Schnittstelle aufgeführt:

```

Dim scc As COMSVCSLib.SecurityCallContext
Dim IsMyAppSecure As Boolean
Set scc = GetSecurityCallContext
IsMyAppSecure = scc.IsSecurityEnabled()

```

Sie werden möglicherweise feststellen, dass einige Methoden sowohl in der `SecurityCallContext`- als auch in der `ObjectContext`-Schnittstelle vorhanden sind. Der Grund hierfür besteht darin, dass MTS entwickelt wurde, um einen umfassenden Zugang über die `ObjectContext`-Schnittstelle zu ermöglichen. Sie können natürlich jede dieser beiden Schnittstellen benutzen. Die `SecurityCallContext`-Schnittstelle enthält jedoch eine Funktionalität, die in der `ObjectContext`-Schnittstelle nicht zur Verfügung steht. Sie sollten deshalb konsequent die `SecurityCallContext`-Schnittstelle für Ihre programmgesteuerte Sicherheit benutzen.

COM+-Komponenten testen und debuggen

Sobald Sie eine Komponente in eine ActiveX-DLL kompiliert haben, können Sie diese in eine COM+-Serverapplikation installieren. Dies geschieht mit dem administrativen Hilfsmittel *Komponentendienste*. Eine einfache Möglichkeit, Ihren Komponentencode zu überprüfen, besteht darin, eine formbasierte Applikation mit Visual Basic zu erstellen, die als Testumgebung für Ihre COM+-Objekte dient. Sie können diese Testumgebung erstellen, indem Sie ein neues Standard-EXE-Projekt generieren und die Referenz auf Ihre ActiveX-DLL darin einbinden. Wenn Sie diesen Test-Client starten, werden alle aus einer konfigurierten Komponente erstellten Objekte von der COM+-Laufzeit in eine private Instanz von `DLLHOST.EXE` geladen.

Sie können den Test-Client mit der entsprechenden kompilierten EXE oder in der Visual Basic-IDE starten. Sie sollten in beiden Fällen das administrative Hilfsmittel *Komponentendienste* benutzen können, um zu überprüfen, ob Ihre Objekte innerhalb Ihrer Serverapplikation erzeugt worden sind.

Sie beenden Ihre Arbeit in der Regel damit, Ihre DLLs erneut generieren zu lassen, wenn Sie Ihren Code in der COM+-Laufzeitumgebung modifizieren und testen. Eine Serverapplikation ist jedoch so konfiguriert, dass sie drei Minuten im Speicher verbleibt, nachdem all ihre Objekte freigegeben wurden. Wenn Sie versuchen, die DLL erneut erstellen zu lassen, während die Serverapplikation noch geladen ist, erhalten Sie die Fehlermeldung »Zugriff verweigert«. Sie können dieses Problem umgehen, indem Sie einen Rechtsklick auf die Serverapplikation im administrativen Hilfsmittel *Komponentendienste* durchführen und die Anweisung *Herunterfahren* auswählen. Sie können die Serverapplikation während der Entwicklung auch so konfigurieren, dass diese sofort heruntergefahren wird, nachdem alle Objekte freigegeben wurden.

Sie müssen auch das Test-Client-Projekt schließen, bevor Sie die DLL erneut erstellen lassen können, da das Projekt einen Verweis auf die Typbibliothek in der DLL verwendet. Sobald Sie das Client-Projekt, das die DLL referenziert, geschlossen und darüber hinaus sichergestellt haben, dass die Serverapplikation nicht ausgeführt wird, sollten Sie die DLL problemlos neu erstellen können.

Einige Programmierer haben allerdings festgestellt, dass sie das »Zugriff verweigert«-Problem noch ein wenig weiterführen können. Wenn Sie dafür gesorgt haben, dass die DLL in den Webserverprozess *INETINFO.EXE* geladen wird, müssen Sie *INETINFO.EXE* zunächst herunterfahren, bevor Sie die DLL erneut erstellen lassen. Wenn Sie Komponenten für eine Webapplikation erzeugen, sollten Sie diese nicht testen, während sie als nicht konfigurierte Komponenten registriert oder für die Ausführung in einer Bibliotheksapplikation konfiguriert sind. Wenn Sie Ihre Komponente in einer Serverapplikation konfiguriert haben, wird Ihre DLL nicht in *INETINFO.EXE* geladen, so dass ein erneutes Erstellen kein Problem sein sollte.

Konfigurierte Komponenten im Visual Basic-Debugger ausführen

Der Debugger in Visual Basic 6 überprüft Komponenten in einer simulierten Version der COM+-Laufzeitumgebung. Diese Unterstützung wurde ursprünglich für die MTS-Entwicklung hinzugefügt, arbeitet aber auf beiden Plattformen. Wie Sie noch sehen werden, ist diese Debugging-Strategie die definitiv schnellste und sehr einfach zu benutzen. Diese Technik weist jedoch im Bereich der Simulation der COM+-Laufzeitumgebung einige Nachteile auf, so dass ich Ihnen auch andere Techniken vorstellen werde.

Der Visual Basic-Debugger simuliert die COM+-Laufzeitumgebung, indem er Objekte aus den Komponenten in bestimmten Bibliotheksapplikationen erstellt. Dies erlaubt Visual Basic, Objekte in den Debugger (*VB6.EXE*) zu laden und auszuführen. Wenn Ihre Komponenten bereits zu einer Bibliotheksapplikation hinzugefügt wurden, werden keine Änderungen vorgenommen. Wenn sich Ihre Komponenten in einer COM+-Serverapplikation befinden, wird diese vom Visual Basic-Debugger in eine Bibliotheksapplikation konvertiert. Wurden Ihre Komponenten zu einer COM+-Applikation hinzugefügt, generiert die Visual Basic-IDE eine versteckte Bibliotheksapplikation für die Komponenten.

Alle drei Methoden arbeiten ohne Unterschied. Wenn Sie ein ActiveX-DLL-Projekt starten, das korrekt konfiguriert wurde, wird die COM+-Laufzeit zusammen mit Ihrem Komponentencode in den Prozess des Visual Basic-Debuggers geladen. Wichtig ist, dass Sie eine Applikation und all Ihre Komponenten mit dem administrativen Hilfsmittel *Komponentendienste* konfigurieren können, bevor Sie den Debug-Vorgang initiieren. Schließlich möchten Sie, dass all Ihre Attribute korrekt eingerichtet sind, bevor Sie Ihren Code testen und debuggen. Dies bedeutet, dass Sie im-

mer zuerst Ihre Komponenten zu einer COM+-Applikation hinzufügen und dort konfigurieren sollten, bevor Sie mit dem Debuggen beginnen.

In Visual Basic 6 verfügt jedes *MultiUse*-Klassenmodul in einem ActiveX-DLL-Projekt über eine *MTSTransactionMode*-Eigenschaft. Wenn Sie diese Eigenschaft auf einen anderen Wert als *NotAnMTSObject* setzen, können Sie Ihre COM+-Komponenten in der Visual Basic-IDE debuggen. Sie haben zum Beispiel die Möglichkeit, die *MTSTransactionMode*-Eigenschaft für Ihr Klassenmodul auf *NoTransactions* setzen. Visual Basic führt Ihre Objekte dann unter der Kontrolle der COM+-Laufzeit aus.

Hätte das Visual Basic-Team zwei Klasseneigenschaften statt nur einer Eigenschaft zur Verfügung gestellt, wäre alles etwas weniger verwirrend. Wenn eine *MultiUse*-Klasse beispielsweise zusätzlich zur Transaktionsmodus-Einstellung eine *COM+Debugging*-Eigenschaft besäße, die auf *True* oder *False* gesetzt werden könnte, wäre alles ein wenig einfacher. Sie sollten jedoch beachten, dass die *MTSTransactionMode*-Eigenschaft sowohl das Debuggen als auch die Transaktionsunterstützung kontrolliert. Dies ist sehr verwirrend, da beide Vorgänge nicht wirklich etwas miteinander zu tun haben.

In diesem Kapitel wurde bereits erwähnt, dass *New* nicht für eine Klasse aufgerufen werden sollte, die in dieselbe DLL kompiliert wird. Der *New*-Operator führt jedoch zu weitaus größeren Schwierigkeiten, wenn Sie konfigurierte Komponenten innerhalb der Visual Basic-IDE debuggen. Im Visual Basic-Debugger sind die Regeln, die den Einsatz des *New*-Operators verbieten, sehr viel konsequenter. Sie müssen auf den Operator verzichten, wenn der Client und die Klasse, die zum Erstellen des neuen Objekts verwendet werden, in derselben Projektgruppe vorhanden sind. Anders ausgedrückt, können ein Aufruf von *New* und die Klasse, für die Sie *New* aufrufen, nicht innerhalb derselben Sitzung des Visual Basic-Debuggers ausgeführt werden. Lassen Sie mich ein wenig Hintergrundwissen erklären, um dieses Problem zu erläutern.

Stellen Sie sich vor, Sie bewahren ein Client-Applikationsprojekt und ein ActiveX-DLL-Projekt in derselben Projektgruppe auf. Der Grund hierfür ist, dass sie beide Projekte in einer einzelnen Sitzung der Visual Basic-IDE debuggen möchten. Wenn die Client-Applikation in dem DLL-Projekt den *New*-Operator für die Klasse aufruft, die Sie als konfigurierte Komponente debuggen möchten, erstellt der Visual Basic-Debugger das neue Objekt ohne die Hilfe des SCM. Das neue Objekt wird somit nicht von der COM+-Laufzeit in einem gültigen Kontext erstellt.

Diese Situation führt zu unerwünschten Nebeneffekten, die leicht zu erkennen sind. All Ihre Aufrufe von *GetObjectContext* und *GetSecurityCallContext* geben eine Null-Referenz zurück, die gewöhnlich in mehreren Laufzeitfehlern resultiert. Wenn Sie wiederholt mit Fehler 91 (*»Objektvariable oder With-Blockvariable nicht festgelegt«*) konfrontiert werden, versucht Ihr Code wahrscheinlich, Methoden mit Null-Referenzen aufzurufen.

Sie können Ihr Test-Client-Projekt und eine ActiveX-DLL, die sich in derselben Projektgruppe befinden, somit nicht ausführen, wenn die Client-Applikation den *New*-Operator verwendet. Ebenso ist es nicht möglich, zwei ActiveX-DLL-Projekte in derselben Projektgruppe zu debuggen, wenn der Code in einer der beiden DLLs den *New*-Operator benutzt, um Objekte aus einer Klasse der anderen DLL zu erstellen. Wenn Sie sicher sein möchten, dass der SCM Ihre Objekte in einem zulässigen Kontext erstellt, müssen Sie die *CreateObject*-Funktion anstelle des *New*-Operators verwenden.

Sie haben in diesem Kapitel viele Gründe kennen gelernt, um die *CreateObject*-Funktion dem *New*-Operator in der COM+-Entwicklung vorzuziehen. Viele COM+-Programmierer benutzen ausschließlich *CreateObject*, so dass sie sich keine Gedanken um Probleme machen müssen, die durch den *New*-Operator entstehen.

Nachdem ich nun darüber gesprochen habe, was sie vermeiden sollten, finden Sie nachfolgend eine Liste der Schritte, die Sie befolgen sollten, um konfigurierte Komponenten problemlos debuggen zu können:

1. Starten Sie eine Sitzung der Visual Basic-IDE und öffnen Sie ein ActiveX-DLL-Projekt oder eine Projektgruppe mit ActiveX-DLL-Projekten, die Ihre konfigurierten Komponenten enthalten. Sorgen Sie dafür, dass `CreateObject` benutzt wird, wenn Sie die Objekte der DLL-Projekte aktivieren.
2. Stellen Sie sicher, dass jedes Projekt mit der Einstellung *Binär-Kompatibilität* eingerichtet wird. Dies bedeutet, dass Sie jede DLL zunächst erstellen müssen, bevor Sie sie debuggen.
3. Setzen Sie die *MTSTransactionMode*-Eigenschaft jeder *MultiUse*-Klasse auf einen anderen Wert als *NotAnMTSObject*. Wenn Sie keine transaktionalen Komponenten erstellen, sollten Sie die Einstellung *NoTransactions* verwenden.
4. Achten Sie darauf, dass Sie Ihre Komponenten in eine COM+-Applikation installieren und all Ihre Attribute angemessen konfigurieren. Wenn Ihre Komponenten nicht in einer COM+-Applikation konfiguriert sind, werden den Attributen Standardeinstellungen zugewiesen.
5. Haben Sie Ihre Komponenten zu einer Serverapplikation hinzugefügt, sollten Sie sicherstellen, dass der Serverapplikationsprozess nicht ausgeführt wird, wenn Sie den Visual Basic-Debugger starten. Wenn die Serverapplikation ausgeführt wird, können Sie die *Herunterfahren*-Anweisung im administrativen Hilfsmittel *Komponentendienste* benutzen.
6. Setzen Sie die gewünschten Haltepunkte und starten Sie den Visual Basic-Debugger, indem Sie aus dem Menü *Ausführen* den Eintrag *Starten, vollständige Kompilierung* auswählen.
7. Starten Sie die Client-Applikation. Sie können den Client mit der kompilierten ausführbaren Datei oder aus einer anderen Sitzung der Visual Basic-IDE heraus starten. Sie können ihn darüber hinaus in derselben Visual Basic-IDE-Sitzung, die auch das DLL-Projekt ausführt, starten. Wenn Sie sich für letztere Möglichkeit entscheiden, sollte Ihr Client `CreateObject` anstelle des `New`-Operators verwenden.

Es ist üblich, wiederholt zwischen der Ausführung Ihres Komponentencodes in *DLLHOST.EXE* und dem Visual Basic-Debugger umzuschalten. Sorgen Sie lediglich dafür, dass die Client-Applikation alle Objekte freigibt und die Serverapplikation oder der Visual Basic-Debugger heruntergefahren wird, bevor Sie einen Wechsel durchführen. Dies gewährleistet, dass die Aktivierungseinstellung Ihrer Anwendung von einer Serverapplikation auf eine Bibliotheksapplikation und umgekehrt umgestellt wird.

Es gibt einige wichtige Einschränkungen, die den Visual Basic-Debugger betreffen. Zunächst kann Ihr Code nicht in einer Instanz von *DLLHOST.EXE* ausgeführt werden. Ihr Code läuft in *VB6.EXE*. Zweitens ist der Visual Basic-Debugger auf einen einzelnen Thread beschränkt. Es gibt keine Möglichkeit, Parallelität zu simulieren. Wenn Sie drittens Ihre Komponenten in einer COM+-Applikation konfiguriert haben, werden Ihrer Applikation und Ihren Komponentenattributen Standardwerte während des Debug-Vorgangs zugewiesen.

Andere nützliche Debug-Techniken

Bisweilen werden Sie Ihren Code in *DLLHOST.EXE*, und nicht mit dem Visual Basic-Debugger überprüfen wollen. Die einzige Möglichkeit, Ihren Code in einer Instanz von *DLLHOST.EXE* zu debuggen, besteht darin, Ihre Visual Basic-Komponenten im Visual C++-Debugger ausführen zu lassen. Die primäre Motivation hierfür besteht darin, dass Sie feststellen können, wie sich Ihr Code in Situationen verhält, in denen Objekte, die auf separaten Threads ausgeführt werden,

gleichzeitig versuchen, auf dieselbe Ressource zuzugreifen. In MTS müssen Sie den C++-Debugger benutzen, um sicherheitsrelevanten Code zu testen. Dies ist unter COM+ nicht notwendig.

Um kompilierte Visual Basic-Komponenten im Visual C++-Debugger ausführen zu können, müssen Sie Ihre ActiveX-DLL mit symbolischen Debug-Informationen erstellen. Sie setzen diese Option auf der Registerkarte *Kompilieren* im *Projekteigenschaften*-Dialogfeld. Im Anschluss daran werden zusätzliche Informationen zu Ihrer kompilierten DLL hinzugefügt, die bestimmte Instruktionen mit den Zeilen aus Ihrem Visual Basic-Quellcode verknüpfen. Laden Sie Ihre DLL und Ihr Visual Basic-Klassenmodul anschließend in die Visual C++-IDE. (Details hierzu finden Sie im MSDN unter dem Titel »Testen von kompilierten Komponenten«.)

Sie sollten noch zwei weitere Debug-Techniken kennen. Sie können zum Beispiel einfache Debug-Nachrichten mit der `MsgBox`-Anweisung generieren oder in das Windows-Ereignisprotokoll schreiben. Lassen Sie uns mit der `MsgBox`-Anweisung beginnen.

Sie halten den Einsatz von `MsgBox` möglicherweise für einen Trick, auf den unerfahrene Programmierer zurückgreifen. Es ist jedoch eine Möglichkeit, die Werte Ihrer Variablen einzusehen und festzustellen, wie Ihr Code zur Laufzeit verzweigt. Wenn Sie eine konfigurierte Komponente, die in `DLLHOST.EXE` ausgeführt wird, schnell und einfach debuggen müssen, kann `MsgBox` die beste Lösung sein.

Um die `MsgBox`-Anweisung verwenden zu können, müssen Sie zwei Aspekte berücksichtigen: Sie müssen zunächst gewährleisten, dass Ihre Serverapplikation unter der Identität des interaktiven Benutzers ausgeführt wird, so dass Ihre Objekte auf die lokale Anzeigekonsolle Ihres Computers zugreifen können. Sie müssen zweitens sicherstellen, dass die Option *Unbeaufsichtigte Ausführung* nicht selektiert ist, wenn Sie Ihren Server erstellen. Wenn Sie diese beiden Regeln befolgen, können Sie ein Nachrichtendialogfeld benutzen, um eine Debug-Nachricht wie nachfolgend gezeigt zu senden:

```
Dim Msg As String
Msg = "Fügen Sie hier Ihre Debug-Nachricht ein"
MsgBox Msg, vbMsgBoxSetForeground, "Debugger für Arme"
```

Der Nachteil dieser Art des Debuggens besteht darin, dass Sie die `MsgBox`-Anweisungen wieder entfernen müssen, bevor Sie Ihren Code distribuieren. Wenn Sie dies vergessen, kann einer dieser Aufrufe dazu führen, dass ein Thread in Ihrer Applikation nicht mehr reagiert. Sie können natürlich die bedingte Kompilierung einsetzen, um zwischen den Builds, die Debug-Nachrichten generieren, und den Builds, die auf diese Nachrichten verzichten, umzuschalten.

Eine weitere nützliche Technik besteht darin, Debug-Nachrichten an das Windows-Ereignisprotokoll zu senden. Visual Basic bietet ein integriertes `App`-Objekt an, dass die `LogEvent`-Methode zur Verfügung stellt. Sie können eine Nachricht einfach an das Windows-Ereignisprotokoll anhängen, indem Sie den folgenden Code benutzen:

```
Dim Msg As String
Msg = "Fügen Sie hier Ihre Debug-Nachricht ein"
App.LogEvent Msg, vbLogEventTypeInformation
```

Wenn Sie Ihre Komponenten testen, können Sie die Windows-Ereignisanzeige benutzen, um Ihre Debug-Nachrichten einzusehen. Sie werden, nebenbei bemerkt, möglicherweise feststellen, dass das Protokollieren von Windows-Ereignissen nicht nur für den Debug-Vorgang nützlich ist. Sie können Ereignisse auch einsetzen, um wiederholt auftretende Fehler in der Betriebsumgebung zu überwachen. Sie könnten ebenfalls alle unerwarteten Fehler, die von Ihren Komponenten ausge-

löst werden, protokollieren. Beachten Sie, dass dem zweite Parameter der `LogEvent`-Methode einer der Werte `vbLogEventTypeInformation`, `vbLogEventTypeWarning` oder `vbLogEventTypeError` zugewiesen werden kann.

Zusammenfassung

Was also hat Sie dieses Kapitel gelehrt? Sie sollten nun verstehen, dass das COM+-Programmiermodell von deklarativen Attributen und Kontexten handelt. Sie sollten nach Möglichkeiten suchen, mehr deklarative als programmgesteuerte Arbeit zu verrichten. Das Geheimnis besteht darin, die COM+-Laufzeit so viel wie möglich arbeiten zu lassen. Sie sollten außerdem versuchen, keine Änderungen an dem zu kompilierenden Code vorzunehmen, so dass Sie Ihre DLLs nicht häufig erneut kompilieren und distribuieren müssen.

Das Wissen um die Funktionsweise der COM+-Administration ist sehr wichtig. Auch nachdem Sie Ihre COM+-Applikation erstellt und Ihre Komponenten hinzugefügt haben, müssen Sie bisweilen bestimmte Attribute explizit konfigurieren. Einige von Ihnen werden administrative Skripts schreiben, während andere das administrative Hilfsmittel *Komponentendienste* benutzen werden. Beide Lösungen können zu dem gewünschten Ergebnis führen. In den nächsten Kapiteln werden Sie weitere Applikations- und Komponentenattribute kennen lernen. Gegenwärtig ist es jedoch wichtig, dass Sie lediglich verstehen, wann und wie diese Attribute konfiguriert werden müssen.

Sie haben in diesem Kapitel erfahren, wie COM+ Attributwerte und kontextuelle Informationen in einer ausgeführten Applikation überwacht. Ein Verständnis dieser Architektur ist wichtig, damit Sie wissen, wie COM+ intern arbeitet. Jeder Kontext besitzt einen eigenen Objektkontext, der kontextuelle Informationen speichert. Sie haben außerdem gesehen, wie eine programmgesteuerte Interaktion mit dem Objektkontext erfolgt und wie sich der Aufrufkontext bewegt, wenn die Programmsteuerung von Methode zu Methode übergeben wird.

Das Kapitel endet mit der Beschreibung verschiedener Ressourcen zum Testen und Debuggen von konfigurierten Komponenten. Diese Informationen und andere Ideen, die in diesem Kapitel präsentiert wurden, sollten den Weg für die folgenden Kapitel ebnen. Ich glaube, Sie werden mit mir darin einig sein, dass die attributbasierte Programmierung die Welt, in der Sie leben, ändern wird. Ihre Komponenten sind definitiv nicht mehr an einem weit entfernten Ort.

7 Gemeinsame Ressourcen in COM+-Applikationen nutzen

-
- 172 Threads gemeinsam nutzen
 - 186 Speicher gemeinsam nutzen
 - 193 Datenbankverbindungen gemeinsam nutzen
 - 199 Poolfähige Objekte gemeinsam nutzen
-

Eine typische COM+-Applikation kann auf vielen unterschiedlichen Hardware-Konfigurationen ausgeführt werden. Dies ist der Vorteil von COM und der Ortsunabhängigkeit. Während Ihnen jedoch viele Optionen zur Verfügung stehen, können Sie nur eine physikalische Konfiguration gleichzeitig benutzen. Vielleicht führen Sie Ihre Applikation auf einem Servercomputer mit einem 166 MHz-Prozessor aus. Möglicherweise wird die Applikation auf einem Rechner mit vier 400 MHz-Prozessoren eingesetzt. Vielleicht sind Sie in der glücklichen Lage, eine Servergruppe mit vielen Prozessoren, die über viele Computer verteilt sind, verwenden zu können. Doch welche Konfiguration Sie auch immer besitzen, Sie sollten Ihre Applikationen immer so schreiben, dass diese den größtmöglichen Nutzen aus der gegenwärtig zur Verfügung stehenden Hardware ziehen. Dies ist das Geheimnis einer Applikation, die skalierbar ist.

Was ist Skalierbarkeit? Abgesehen davon, dass Skalierbarkeit ein Begriff ist, der häufig vom Marketing missbraucht wird, ist mit diesem Ausdruck die Fähigkeit einer Applikation gemeint, eine wachsende Anzahl von Benutzern zu bedienen, ohne die Antwortzeiten zu verlängern und den gesamten Systemdurchsatz zu verringern. Weist Ihre Applikation in einem System mit zwanzig Benutzern eine Antwortzeit von einer Sekunde auf, ist sie skalierbar, wenn sie dieselbe Antwortzeit für 10000 Benutzer benötigt. Wenn Ihre Applikation all diese Benutzer nicht bedienen kann, weil sie auf einem Server mit einem 166 MHz-Prozessor ausgeführt wird, können Sie nicht feststellen, ob sie skalierbar ist oder nicht. Wenn Ihre Applikation jedoch unabhängig von der eingesetzten Hardware keine angemessene Antwortzeit bietet, ist sie definitiv nicht skalierbar.

Es gibt immer zwei Möglichkeiten, eine Applikation zu skalieren: das Einrichten einer leistungsfähigeren Hardware und das Schreiben von Code, der die Vorteile der gegebenen Hardware nutzt. Um skalierbare Komponenten und Applikationen schreiben zu können, müssen Sie wissen, wie die COM+-Laufzeit arbeitet. Dieses Kapitel erklärt, wie die COM+-Laufzeit eingesetzt wird, um möglichst viele Benutzer gleichzeitig zu bedienen.

Lassen Sie uns die einzelnen Bestandteile einer verteilten Applikation betrachten. Eine verteilte Applikation besteht aus einem oder mehreren Prozessen, die auf einem oder mehreren Computern ausgeführt werden. Wenn Objekte die Atome Ihrer Applikation sind, bilden Prozesse die Moleküle. Sie unterteilen Ihre Applikation in Prozesse, indem Sie Ihre Komponenten in verschiedenen Serverapplikationen und Bibliotheksapplikationen konfigurieren. In einigen Szenari-

en benötigen Sie möglicherweise mehr Prozesse, in anderen weniger. In diesem Kapitel konzentrieren wir uns auf die Faktoren, die jedem individuellen Prozess erlauben, die größtmögliche Anzahl von Clients zu bedienen.

Eine mehrschichtige Applikation kann eine größere Skalierbarkeit als eine Two-Tier-Applikation bieten, da mehrere Clients die Möglichkeit besitzen, prozessspezifische Ressourcen gemeinsam zu verwenden. Die COM+-Laufzeit kann Threads, Speicher, Datenbankverbindungen und poolfähige Objekte für die gemeinsame Nutzung freigeben. Der Haupttenor dieses Kapitels ist, dass Code, der auf Weisung eines Clients ausgeführt wird, niemals eigennützig sein sollte. Er sollte so spät wie möglich auf seine Ressourcen zugreifen und diese schnellstmöglich wieder freigeben. Applikationen, die Ressourcen effizient gemeinsam nutzen, sind besser skalierbar als Applikationen, die hierauf verzichten.

Das Kapitel konzentriert sich auf prozessspezifische Skalierbarkeitsaspekte. Wir werden in den folgenden Kapiteln Designs untersuchen, die einer verteilten Applikation erlauben, über mehrere Computer verteilt zu werden. Wie Sie sehen werden, sollten Sie bestimmte Programmiertechniken vermeiden, wenn Sie Ihre Applikation in einer Servergruppe ausführen lassen möchten. Wir werden unsere Diskussion jedoch zunächst auf einen einzelnen Prozess beschränken, der auf einem einzelnen Computer ausgeführt wird.

Threads gemeinsam nutzen

Das Threading ist ein wichtiger Aspekt für jede Middle-Tier-Applikation, die mit mäßigem bis massivem Datenverkehr konfrontiert wird. Ein hohes Verkehrsaufkommen tritt in der Regel dann auf, wenn Online-Benutzer viele Client-Anfragen pro Sekunde auslösen. Ein skalierbarer Prozess muss dazu in der Lage sein, jede Anfrage mit einem akzeptablen Reaktionsvermögen zu bearbeiten. Ein Prozess ist nicht skalierbar, wenn sein Reaktionsvermögen und sein Gesamtdurchsatz mit einer zunehmenden Anzahl von Anfragen pro Sekunde linear abnimmt.

Ein Prozess, der auf einem einzelnen Thread basiert, kann nicht skalierbar sein, da er nicht mehr als eine Methode gleichzeitig ausführen kann. Anders ausgedrückt lässt dieser Prozess keine Parallelität zu. Während der Perioden, in denen ein massives Verkehrsaufkommen herrscht, werden die Anfragen einfach in einer Warteschlange abgelegt und seriell bearbeitet. Ein Single-Threaded-Modell ist außerdem extrem restriktiv, wenn ein Prozess auf einem Computer mit mehreren Prozessoren ausgeführt wird. Der Prozess kann einfach nicht die Vorteile nutzen, die von der Hardware angeboten werden.

Ein Prozess, der einen neuen Thread für jeden Client erstellt, weist darüber hinaus Probleme hinsichtlich der Skalierbarkeit auf. Physikalische Threads müssen erstellt und zerstört werden, wenn Clients kommen und gehen. Das Betriebssystem muss außerdem mehr Zeit damit verbringen, Threads in die Prozessoren zu schalten und wieder daraus zu entfernen, wenn die Anzahl der Clients (und Threads) zunimmt. Wenn dieser administrative Aufwand größer wird, verringert sich der Gesamtdurchsatz des Prozesses.

Jahre der Erfahrung haben die Industrie zu einer wichtigen Erkenntnis geführt: Um die höchsten Ebenen der Skalierbarkeit zu erzielen, muss ein Middle-Tier-Prozess ein gewisses Thread-Pooling-Schema zur Verfügung stellen. Das Ziel eines solchen Schemas besteht darin, einen optimierten Ausgleich zwischen den höheren Ebenen der Parallelität und einer effizienten Ressourcenverwendung zu erreichen. Das Schreiben des Codes zum Verwalten eines Thread-Pools und das effiziente Verteilen dieser Threads auf mehrere Clients ist jedoch kein triviales Unterfangen.

COM+ stellt jedoch glücklicherweise ein integriertes Schema für das Thread-Pooling zur Verfügung. Dies geschieht in einer für den Programmierer transparenten Weise. Sie müssen sich somit

nicht darum kümmern, wie der COM+-Thread-Pool-Manager arbeitet, wenn Sie Visual Basic-Komponenten schreiben. Sie erstellen Ihre Komponenten vorwiegend aus einer Single-Threaded-Perspektive und lassen die COM+-Laufzeit Ihre Objekte auf mehrere Threads verteilen.

Das COM+-Threading-Schema ist vorwiegend transparent. Sie müssen dennoch einige wichtige Threading-Konzepte beachten, wenn Sie Visual Basic-Komponenten entwickeln. Ich werde aus diesem Grund die grundlegenden Threading-Konzepte von COM und des Microsoft-Windows-Betriebssystems erläutern. Dies führt wiederum zu einer Diskussion über das COM+-Threading-Pool-Schema, das auf einer Abstraktion basiert, die als Aktivität bezeichnet wird. Mein Ziel ist, Ihnen die Informationen zukommen zu lassen, die Sie benötigen, um die Vorteile des aktivitätsbasierten Parallelitätsmodells nutzen zu können und um Programmiertechniken zu vermeiden, die die Skalierbarkeit Ihrer Applikationen und Ihren Code beeinträchtigen.

Eine Win32-Threading-Anleitung

Das Win32-Programmiermodell basiert auf zwei höheren Abstraktionen, die unter den Begriffen Prozesse und Threads bekannt sind. Ein Prozess ist eine ausgeführte Instanz einer Applikation, die einen Adressraum im virtuellen Speicher und verschiedene andere Ressourcen besitzt. Ein Thread ist eine zeitlich einteilbare Entität, die das Eigentum von nur einem Prozess ist. Das Betriebssystem kennt jeden Thread in allen Prozessen und ist verantwortlich für die Einteilung der Zeit eines Threads im Prozessor des Systems, wie in Abbildung 7.1 dargestellt.

Jeder Thread erhält seinen eigenen Aufrufstack. Der Scheduler reserviert Bearbeitungszyklen, indem er jedem Thread einen Zeitabschnitt zuweist. Wenn der Zeitabschnitt beendet ist, wird der gegenwärtig ausgeführte Thread unterbrochen, und ein anderer Thread ist an der Reihe. Ein unterbrochener Thread kann genug Informationen auf seinem Aufrufstack speichern, um sich daran zu erinnern, was er getan hat und wie weit er kam, bevor er aus dem Prozessor entfernt wurde. Wenn der Thread den nächsten Zeitabschnitt erhält, kann er dort beginnen, wo er unterbrochen wurde. Die Kombination von Threads und dem Scheduler ist sehr leistungsfähig, da sie einem Computer mit nur einem Prozessor erlaubt, scheinbar mehrere Aufgaben gleichzeitig durchzuführen.

Jeder Prozess beginnt mit einem primären Thread. Dieser vom System erstellte Thread dient als Einsprungspunkt in die Applikation. In einer typischen, auf Windows basierenden Applikation mit einer Benutzeroberfläche wird der primäre Thread benutzt, um das Hauptfenster der Applikation zu erstellen und eine so genannte Nachrichtenpumpe einzurichten, die zum Überwachen der vom Betriebssystem versendeten, eingehenden Nachrichten verwendet wird. Wenn ein Benutzer beispielsweise auf ein Hauptfenster klickt, sendet das System eine WM_MOUSEBUTTONDOWN-Nachricht an eine Warteschlange, die mit dem Fenster verknüpft ist. Der primäre Thread liest diese Nachricht aus der Warteschlange und reagiert darauf. Ein Thread, der eine Pumpe einrichtet, um die eingehenden Nachrichten zu überwachen, wird als Benutzeroberflächen-Thread bezeichnet.

Wenn eine Benutzeroberfläche mit C oder C++ geschrieben wird, muss die Nachrichtenpumpe häufig manuell eingerichtet werden, indem eine Schleife geschrieben wird, die die mit dem Fenster verknüpfte Nachrichtenwarteschlange überwacht. Visual Basic-Programmierer müssen glücklicherweise nicht direkt mit einer Nachrichtenpumpe arbeiten. Die Visual Basic-Laufzeit richtet diese im Hintergrund ein. Das Erstellen einer Visual Basic-Applikation mit einer Benutzeroberfläche ist sehr einfach, da die Visual Basic-Laufzeit die Windows-Nachrichten in Visual Basic-Ereignisse übersetzt.

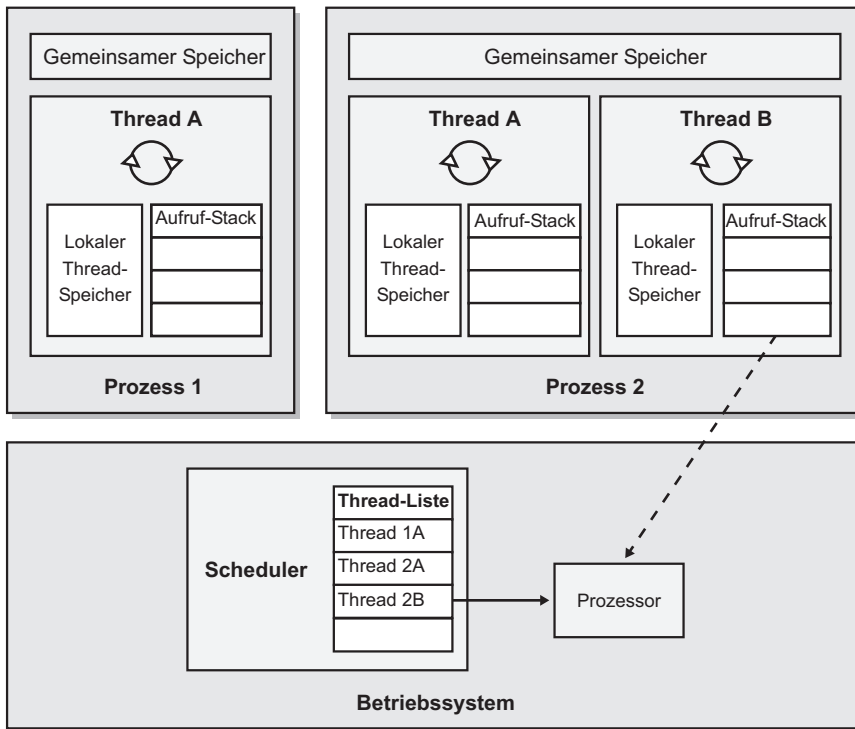


Abbildung 7.1: Jeder Win32-Prozess besitzt einen oder mehrere Threads. Jeder Thread verfügt über einen verknüpften Aufrufstack und kann Daten in einem privaten Bereich speichern, der als lokaler Thread-Speicher (TLS – Thread-Local Storage) bezeichnet wird.

Sobald eine Applikation erstellt ist und ausgeführt wird, erlaubt Ihnen die Win32-API zusätzliche Threads zu produzieren. Dies geschieht mit einer Win32-API-Funktion namens `CreateThread`. C- und C++-Programmierer erzeugen sekundäre Threads in der Regel, um Hintergrundaufgaben mit einer niedrigen Priorität durchzuführen. Ein sekundärer Thread überwacht in den meisten Fällen keine Nachrichtenwarteschlange und erfordert deshalb nicht sehr viel Aufwand. Dieser Thread-Typ wird oft als Arbeits-Thread bezeichnet. Der Vorteil eines zweiten Threads besteht darin, dass eine Hintergrundaufgabe durchgeführt werden kann, ohne das Reaktionsvermögen der Benutzeroberfläche zu blockieren.

Parallelität und Synchronisierung

Wenn Sie Multi-Threaded-Applikationen schreiben, indem Sie `CreateThread` aufrufen, müssen Sie sehr vorsichtig sein. Wenn zwei Threads gleichzeitig in einem einzelnen Prozess ausgeführt werden, können Sie mit Problemen konfrontiert werden, die in einem Single-Threaded-Prozess nicht auftreten. Besonders gemeinsam genutzter Speicher, auf den mehrere Threads zugreifen, ist anfällig für Inkonsistenz und Fehler. Der Grund hierfür ist ein Phänomen, das als Parallelität bezeichnet wird. In einer präemptiven Multithreading-Umgebung, wie zum Beispiel Windows 2000, werden Threads von dem Scheduler willkürlich aus dem Prozessor entfernt. Es gibt keine Garantie dafür, dass der Thread seine Arbeit beendet hat. Wenn ein anderer Thread in den Prozessor geschaltet wird, kann dieser mit gemeinsam genutztem Speicher und einem unzulässigen Status

konfrontiert werden, der von dem vorherigen, in seiner Arbeit unterbrochenen Thread zurückgelassen wurde.

Stellen Sie sich zum Beispiel vor, dass die Variablen x und y die Position eines Punktes repräsentieren. Die erste Position des Punktes ist (10,10). Thread A beginnt nun, die Position des Punktes in (20,20) zu ändern. Wenn Thread A unterbrochen wird, nachdem er die x -Position, aber bevor er die y -Position geändert hat, wird der logische Punkt in einem unzulässigen Zustand – der Position (20,10) – zurückgelassen. Die einzigen zulässigen Positionen des Punktes sind (10,10) und (20,20). Thread B sieht den Punkt jedoch an Position (20,10). Wie Sie sehen, lässt die Parallelität eine Applikation für Dateninkonsistenz anfällig werden.

Das Multithreading lässt eine Applikation auch für Datenbeschädigungen anfällig werden. Werfen Sie einen Blick auf ein anderes Beispiel mit einem besonders unglücklichen Ende. Wenn Thread A einen neuen Eintrag in eine verkettete Liste einfügt, muss er einige Zeiger modifizieren, um die Aufgabe durchführen zu können. Wenn Thread A inmitten der Operation unterbrochen wird, kann Thread B während des Durchsuchens der Liste mit unzulässigen Zeigern konfrontiert werden. Wenn Thread B versucht, die unzulässigen Zeiger zu benutzen, kann der gesamte Prozess abstürzen.

Das Multithreading macht es schwierig, Code zu schreiben, der korrekt und robust ist. Ein Win32-Programmierer, der eine Multi-Threaded-Applikation erstellen möchte, muss jeden gemeinsam genutzten Speicher sperren und synchronisieren, der zu Inkonsistenz oder Datenbeschädigungen führen kann. Die Win32-API bietet aus diesem Grund einige Synchronisierungselemente an. Kritische Abschnitte, Mutexe, Semaphore und Ereignisse sind alles Beispiele für Win32-Synchronisierungsobjekte. Es ist nicht immer einfach, mit diesen Objekten zu arbeiten. Sie lassen erfahrene Programmierer jedoch sicheren und robusten Code schreiben, der die Vorzüge des Multithreadings nutzen kann.

Die mit der Parallelität verbundenen Probleme treten nur dann auf, wenn zwei oder mehrere Threads auf dieselben Datenelemente im gemeinsam genutzten Speicher zugreifen. Sie können diese Probleme vermeiden, indem Sie lokale Variablen auf dem Aufrufstack anstatt des gemeinsam genutzten Speichers verwenden. Diese lokalen Variablen sind für einen bestimmten Thread privat. Die Win32-API erlaubt einem Thread ebenfalls, persistente Daten in einem privaten Speicher abzulegen, der als lokaler Thread-Speicher (TLS – Thread-Local Storage) bezeichnet wird. Der Einsatz von TLS löst viele Parallelitätsprobleme, da Sie sich nicht um die Synchronisierung kümmern müssen.

Wenngleich der TLS die Parallelitätsprobleme löst, verursacht er selbst einige Probleme. Auf die Daten, die im TLS gespeichert sind, kann nur der Thread zugreifen, der diese Daten besitzt. Objekte, die Daten im TLS ablegen, erzeugen somit eine Abhängigkeit von dem Thread, der sie erstellt hat. Visual Basic-Objekte nutzen den TLS massiv, so dass jedes Visual Basic-Objekt von dem Thread abhängig ist, der es erstellt hat. Ein Thread kann niemals auf ein Visual Basic-Objekt zugreifen, das von einem anderen Thread erzeugt wurde. Dieser Umstand wird auch als Thread-Affinität bezeichnet. Wie Sie später noch sehen werden, kann die Thread-Affinität in Umgebungen, die das Thread-Pooling nutzen, zu einem einschränkenden Faktor werden.

Die Threading-Modelle von COM

Viele Programmierer – besonders die, die Geschäftslogik in Sprachen wie zum Beispiel Visual Basic schreiben – denken nicht über Parallelität und Synchronisierung nach. Sie gehen entweder davon aus, dass ihr Code unter der Kontrolle eines einzigen Threads ausgeführt wird, oder sie beschäftigen sich erst gar nicht mit dem Threading. Sie kümmern sich nicht um die Synchronisie-

rung ihrer Daten während der Methodenaufrufe und arbeiten deshalb sehr viel produktiver als die Programmierer, die Ihre Zeit mit den Problemen der Parallelität verbringen.

Sie können Code auf zwei unterschiedliche Arten schreiben. Sie können eine Komponente mit Synchronisierungscode erstellen, die threadfähig ist, oder Sie schreiben eine Komponente aus der Single-Threaded-Perspektive. Die Entwickler von COM suchten nach einer Möglichkeit, beide Komponententypen in eine einzige Applikation zu integrieren. Sie wollten Komponenten integrieren, die keinen eigenen Synchronisierungscode in einen Multi-Threaded-Prozess einbringen. Ihre Lösung bestand darin, eine vom System zur Verfügung gestellte Synchronisierung für jedes Objekt anzubieten, das diese benötigt.

COM bietet die automatische Synchronisierung mit Hilfe einer Abstraktion an, die als Apartment bekannt ist. Apartments erlauben Komponenten ohne benutzerdefinierten Synchronisierungscode, sicher in Multi-Threaded-Prozessen ausgeführt zu werden. Auch Objekte, die eine Thread-Affinität aufweisen, wie zum Beispiel die mit Visual Basic erstellten Objekte, können sicher in Multi-Threaded-Prozessen ausgeführt werden.

In COM muss jedes Objekt in einem spezifischen Apartment ausgeführt werden. Jedes Apartment muss in einem bestimmten Prozess vorhanden sein. Sie können sich ein Apartment als eine Gruppe mit Objekten vorstellen, die kompatible Synchronisierungsanforderungen aufweisen. Das ist jedoch nicht alles. Jeder COM-fähige Thread ist mit genau einem Apartment verknüpft, so dass Sie ein Apartment als eine Gruppe mit Objekten und Threads in einem Prozess definieren können.

Sie sollten beachten, dass es Apartments bereits länger als MTS oder COM+ gibt. Dies bedeutet, dass Apartments Teil des Programmiermodells waren, bevor Kontexte eingeführt wurden. Mit dem Erscheinen von Windows 2000 können Sie ein Apartment als Gruppe mit Kontexten definieren, die in einem Prozess ausgeführt werden. Prozesse können somit in Apartments und Apartments können in Kontexte unterteilt werden.

Vor Windows 2000 gab es nur zwei Apartmenttypen: Das Multi-Threaded-Apartment (MTA) und das Single-Threaded-Apartment (STA). Das MTA stellt eine Umgebung für Komponenten mit benutzerdefiniertem Synchronisierungscode zur Verfügung. Ein STA ist eine sichere Umgebung für die Ausführung von Objekten, die nicht unter Berücksichtigung der Sperrung und Synchronisierung geschrieben wurden. STAs wurden entwickelt, um auf die vom Programmierer unterstützte Synchronisierung verzichten zu können.

Im Verlauf der letzten Jahre wurden zwei Begriffe benutzt, um beide Threading-Modelle zu beschreiben. Die Win32-SDK-Dokumentation verwendet die Begriffe Multi-Threaded-Apartment und Single-Threaded-Apartment, während andere Autoren und Entwickler die Begriffe Free-Threaded und Apartment-Threaded benutzen. Eine Free-Threaded-Komponente wird im MTA und eine Apartment-Threaded-Komponente im STA ausgeführt. Wenn Sie nur den Ausdruck Apartment hören, können Sie in der Regel davon ausgehen, dass ein STA gemeint ist.

Doch wie verhindert ein STA die Parallelität? Wie der Name verrät, basiert ein STA auf einem einzigen Ausführungs-Thread. Nur dieser Thread kann mit den Objekten im STA in Berührung kommen. Mehr als diese Einschränkung ist nicht notwendig, um die Parallelität zu unterbinden und eine sichere Umgebung für Objekte zu schaffen, die eine Thread-Affinität aufweisen. Ein STA muss jedoch Methodenaufrufe bearbeiten können, die von anderen Apartments stammen. Um dieser Anforderung gerecht zu werden, erhält das STA ein wenig Hilfe von der zugrunde liegenden RPC-Schicht und einer Windows-Nachrichtenwarteschlange.

Wenn die COM-Laufzeit ein STA erstellt, generiert sie ebenfalls ein unsichtbares Fenster. Beachten Sie, dass das STA dieses Fenster nicht wirklich benötigt. Es benötigt lediglich die mit dem Fenster verknüpfte Nachrichtenwarteschlange. Diese Warteschlange erlaubt dem STA, eine Stan-

Standard-Nachrichtenpumpe einzurichten, wie in Abbildung 7.2 dargestellt. Die RPC-Schicht reagiert auf Methodenaufrufe, deren Ziel das STA ist, indem sie eine Standard-Windows-Nachricht in dieser Warteschlange hinterlegt. Dies geschieht mit der Win32-Funktion `PostMessage`. Der Thread im STA verwendet eine Nachrichtenpumpe in derselben Weise wie der primäre Thread einer Applikation, die eine Benutzeroberfläche besitzt. Dies ermöglicht dem STA, auf Methodenanforderungen zu reagieren, indem die Nachrichten auf einer FIFO-Basis (First In, First Out) bearbeitet werden.

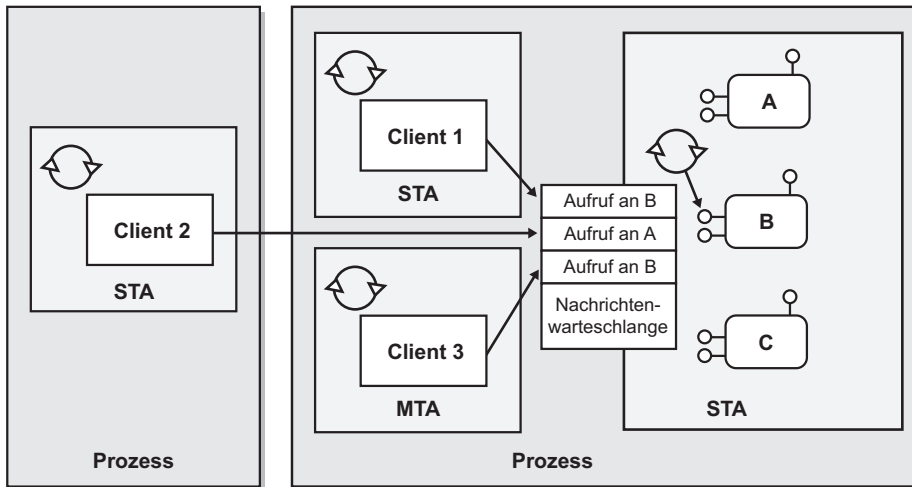


Abbildung 7.2: Jedes STA enthält einen Benutzeroberflächen-Thread, eine Windows-Nachrichtenwarteschlange und eine Nachrichtenpumpe. Diese Architektur gestattet den STAs, eingehende Aufrufe nacheinander zu bearbeiten und Objekte auszuführen, die eine Thread-Affinität aufweisen.

Wenn das STA damit beschäftigt ist, einen Aufruf zu bearbeiten, werden alle anderen eingehenden Anforderungen in die Warteschlange gestellt, bis sie an der Reihe sind. Der Thread des STAs arbeitet sich vielleicht bis zum Ende der Warteschlange durch und wechselt dann, wenn die Warteschlange leer ist, in den Leerlaufmodus. Diese Aufrufarchitektur erfordert sehr viel Aufwand. Der Vorteil des STAs besteht jedoch darin, dass sich Programmierer nicht mit der Parallelität und Synchronisierung beschäftigen müssen.

Als das Visual Basic-Team eine Multi-Threaded-Unterstützung für COM zu seinem Produkt hinzufügte, musste es eine schwierige Entscheidung treffen. Es musste bestimmen, ob die Programmierer mit der Synchronisierung konfrontiert werden sollten. Das Team entschied, dass eine vom Programmierer unterstützte Synchronisierung in einem Werkzeug, das die größtmögliche Produktivität bietet, nicht akzeptabel sei. Es entschloss sich dazu, dass alle Visual Basic-Objekte gemäß des STA-Modells ausgeführt werden sollten. Das Team ging außerdem davon aus, dass es akzeptabel sei, den TLS in dem Code zu verwenden, der automatisch in jeder Visual Basic-Komponente erstellt wird. Dies ist der Grund für die Thread-Affinität aller Visual Basic-Objekte.

Sie müssen eine andere Sprache, wie zum Beispiel C++, benutzen, um Free-Threaded-Komponenten zu erstellen. Ein C++-Programmierer, der Free-Threaded-Komponenten schreibt, muss in der Regel auf die Parallelität reagieren, indem er benutzerdefinierten Synchronisierungscode mit Hilfe von Win32-Sperrelementen generiert. Dieser Programmierstil kann zunehmend komplexer werden, führt aber zu Komponenten, die potentiell schneller sind und besser reagieren.

Windows 2000 führt ein drittes Threading-Modell ein, das als threadneutrales Apartment (TNA) bezeichnet wird. Visual Basic-Programmierer können dieses neue Modell aus Gründen der Thread-Affinität leider nicht nutzen. C++-Programmierer, besonders jene, die zuvor Komponenten für das MTA erstellt haben, werden dieses neue Modell jedoch mit Interesse aufnehmen.

Vor Windows 2000 verlangten die COM-Regeln einen Thread-Wechsel für Aufrufe, die eine Apartmentgrenze verließen. Dies galt in beinahe allen Situationen. Während diese Regeln und deren Befolgung das in das STA-Modell integrierte Synchronisierungsschema gewährleisten, sind sie für Objekte, die im MTA ausgeführt werden, unnötig kompliziert. Ein Aufruf von einem Client in einem STA an ein Objekt, das sich in einem MTA befindet, kann beispielsweise zu einem aufwendigen und nicht erforderlichen Thread-Wechsel führen. Ein Objekt, das unter dem TNA-Modell ausgeführt wird, kann jedoch direkt von jedem Thread in einem Prozess benutzt werden. Das TNA ist deshalb das bevorzugte Threading-Modell für Windows 2000. Sie können dieses Modell natürlich nur dann verwenden, wenn Ihre Werkzeuge und Ihre Sprache Objekte erstellen können, die keine Thread-Affinität aufweisen.

Visual Basic-Komponenten und das ThreadingModel-Attribut

Wie Sie wissen, besitzt jede Komponente einen verknüpften *CLSID*-Schlüssel in der Windows-Registrierung. Der SCM benutzt die Informationen in diesem Schlüssel während der Objektaktivierung. Der Registrierungsschlüssel *CLSID\InprocServer32* kann mit einem *ThreadingModel*-Attribut gesetzt werden, um die Apartmenttypen anzuzeigen, die kompatibel mit der Komponente sind. Eine Visual Basic-Komponente weist entweder das *ThreadingModel*-Attribut Apartment oder kein Threading-Modell auf, wie in Abbildung 7.3 dargestellt. Darüber hinaus sind einige weitere Einstellungen für das *ThreadingModel*-Attribut möglich, die jedoch für die Entwicklung mit Visual Basic nicht relevant sind.

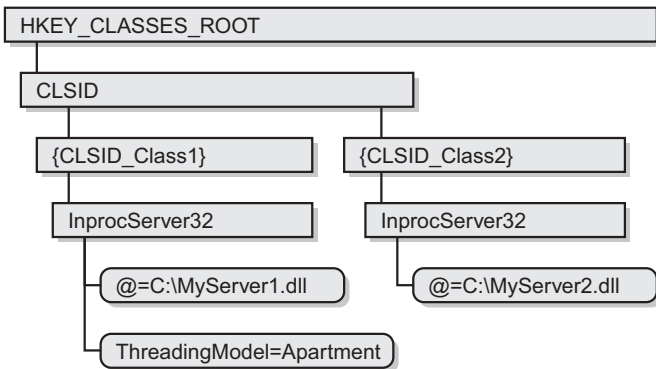


Abbildung 7.3: Sie können den Apartmenttyp einer Komponente spezifizieren, indem Sie ein *ThreadingModel*-Attribut im *CLSID\\InprocServer32*-Schlüssel angeben. Komponenten ohne dieses Attribut werden als *Single-Threaded*-Komponenten erkannt.

Sie können einer Komponente in einem ActiveX-DLL-Projekt explizit das *ThreadingModel*-Attribut zuweisen, indem Sie die *Threading-Modell*-Einstellung auf der Registerkarte *Allgemein* im Dialogfeld *Projekteigenschaften* setzen. Zwei Einstellungen sind hier möglich: Die Standardeinstellung *Apartment Threaded* gewährleistet, dass jede Komponente mit dem *ThreadingModel*-Attribut *Apartment* gekennzeichnet wird. Diese Einstellung zeigt an, dass ein neues Objekt in ein STA geladen werden muss. Das neue Objekt kann jedoch in jedes beliebige STA geladen werden. Sofern der Erzeuger des Objekts in einem STA ausgeführt wird, wird das Objekt in demselben Apartment erstellt. Dies bedeutet, dass beide ohne einen Thread-Wechsel miteinander kommunizieren können.

Wenn Sie sich für die andere *Threading-Modell*-Einstellung *Single Threaded* entscheiden, wird jede Komponente ohne ein *ThreadingModel*-Attribut konfiguriert. Wenn der SCM ein Objekt aus einer Komponente erstellt, die kein *ThreadingModel*-Attribut aufweist, geschieht dies in einer möglichst sicheren Weise. Da der SCM keine Voraussage bezüglich der Komponente treffen kann, wird jedes Objekt auf demselben Thread erstellt. Genauer gesagt wird jedes Objekt in dem ersten STA erstellt, das im Host-Prozess generiert wurde. Das erste STA wird auch als Haupt-STA bezeichnet. Eine Komponente ohne *ThreadingModel*-Attribut wird häufig Single-Threaded- oder Main-Threaded-Komponente genannt.

Single-Threaded-DLLs sind problematisch, da sie zu unnötigen Thread-Wechseln führen. Wenn ein Erzeuger in einem anderen STA als dem Haupt-STA ausgeführt wird, muss eine Proxy-/Stub-Schicht, die zwischen den Threads umschaltet, zwischen dem Erzeuger und dem Objekt eingefügt werden. Single-Threaded-DLLs sind besonders in Umgebungen wie zum Beispiel COM+ und IIS nicht zu empfehlen. Apartment-Threaded-DLLs sind immer zu bevorzugen, da Sie einem Client in einem STA gestatten, sich an ein Objekt im selben STA zu binden. Auf diese Weise wird der Aufwand einer Proxy-/Stub-Schicht vermieden, die zwischen den Threads umschalten muss.

HINWEIS: Rufen Sie CreateThread niemals aus Visual Basic heraus auf.

Ein Win32-Thread kann nicht in die COM-Programmierung einbezogen werden, ohne einen expliziten COM-Bibliotheksaufwurf von `CoInitialize` oder `CoInitializeEx` durchzuführen. Diese Funktionen verknüpfen einen Thread mit einem Apartment. Es ist nicht gestattet und in der Regel fatal, einen Thread in eine COM-Aktivität einzubeziehen, bevor eine dieser Funktionen aufgerufen wurde. Die COM+-Laufzeit führt diesen Aufruf glücklicherweise automatisch durch, wenn sie einen Thread erstellt. Wenn Sie eine Standard-EXE oder eine ActiveX-EXE erzeugen, nimmt die Visual Basic-Laufzeit alle erforderlichen Aufrufe von `CoInitialize` im Hintergrund vor.

Die direkte Arbeit mit Win32-Threads und COM-Apartments ist schwierig und aus praktischen Gründen nicht mit der Visual Basic-Programmiersprache möglich. Als jedoch Visual Basic 5 ausgeliefert wurde, versuchten viele Programmierer, die `CreateThread`-Funktion in der Win32-API mit Hilfe des neuen Visual Basic-Operators `AddressOf` aufzurufen. Diese Programmierer konnten `CreateThread` erfolgreich aufrufen, ihre Applikationen starben jedoch oft einen mysteriösen Tod.

Das Problem des Aufrufs von `CreateThread` besteht darin, dass er einen neuen Win32-Thread erstellt, der nichts von COM oder Apartments weiß. Es ist deshalb nicht gestattet, diesen Thread zu benutzen, um eine Methode auf einem COM-Objekt aufzurufen. Dies würde zu einem Problem führen, da alle Visual Basic-Objekte, einschließlich der Formulare und Steuerelemente, ebenfalls COM-Objekte sind. Lediglich der Thread, der in dem Apartment des Objekts ausgeführt wird, kann direkt auf das Objekt zugreifen. Wenn der neu erstellte Thread zum Beispiel versucht, einen Wert in einem Textfeld zu verändern, stürzt die Applikation ab. Dies ist ein guter Grund dafür, `CreateThread` niemals in Visual Basic-Code zu verwenden.

Ein entschlossener Visual Basic-Programmierer, der die COM-Threading-Modelle kennt, könnte versuchen, einen Schritt weiter zu gehen. Sie können einen Thread mit einem STA verknüpfen, indem Sie eine Funktion, wie zum Beispiel `CoInitialize`, direkt aufrufen. Sobald Sie den Thread jedoch mit einem Apartment verknüpft haben, nimmt die Komplexität zu. Sie sind dann gezwungen, zum Beispiel Code zu schreiben, um Schnittstellenreferenzen (die auf dieser Ebene tatsächlich Schnittstellenzeiger sind) mit Hilfe des Marshalings von einem Apartment zu einem anderen zu übermitteln. Wenn Sie die Schnittstellenreferenzen in einem COM-Methodenaufwurf übergeben, werden der Proxy und der Stub automatisch generiert. Wenn Sie die Referenzen jedoch mit einer anderen Technik übergeben, vergrößern sich die Anforderungen

beträchtlich. Sie müssten zwei komplizierte Aufrufe an die COM-Bibliothek vornehmen – einen Aufruf von `CoMarshalInterface`, um den Stub im Apartment des Objekts zu erstellen, und einen Aufruf von `CoUnmarshalInterface`, um einen Proxy im Apartment des Aufrufers zu erzeugen. Sie müssten darüber hinaus sehr viel über COM wissen, um entscheiden zu können, wann diese Aufrufe notwendig sind. Addieren Sie hierzu die Komplexität der COM+-Regeln und Kontexte, und Sie werden zu einer wichtigen Folgerung gelangen: Rufen Sie niemals `CreateThread` aus Visual Basic heraus auf. Wenn Sie wirklich Ihre eigenen Threads generieren müssen, sollten Sie C++ benutzen und sehr viel über die zugrunde liegende COM-Konfiguration wissen.

Ich muss gestehen, dass dieser gesamte Hinweis lediglich eine Meinung wiedergibt. Einige Ikonen der Visual Basic-Welt, wie zum Beispiel Matt Curland, befürworten das Erstellen von Threads mit Visual Basic-Code. Matt beschreibt Ihnen die Details der hierzu erforderlichen Vorgehensweise in seinem Buch »Advanced Visual Basic 6: Power Techniques for Everyday Programs« (veröffentlicht von Addison Wesley).

Wenn Sie mit allen im dritten Kapitel behandelten COM-Details vertraut sind und eine umfassendere Erläuterung suchen, kann Ihnen Matts Buch helfen. Matt hat einige Jahre im Visual Basic-Team gearbeitet und besitzt ein umfangreiches Wissen darüber, wie die Visual Basic-Laufzeit mit der COM-Bibliothek interagiert. Er beschreibt viele Techniken, die direkte Aufrufe in die COM-Bibliothek aus Visual Basic-Code heraus durchführen.

Das aktivitätsbasierte Parallelitätsmodell

Apartments waren in früheren Versionen von COM der primäre Mechanismus für die vom System zur Verfügung gestellte Synchronisierung. Im COM+-Programmiermodell ist eine neue Abstraktion mit der Bezeichnung Aktivität für die vom System angebotene Synchronisierung verantwortlich. Eine *Aktivität* besteht aus mehreren Kontexten, deren Objekte vor dem gleichzeitigen Zugriff geschützt werden. Das Erstellen eines oder mehrerer Objekte in einer Aktivität befreit Sie davon, benutzerdefinierten Synchronisierungscode zu schreiben. Sie können sich eine Aktivität deshalb als eine Synchronisierungsdomäne vorstellen. Eine Aktivität repräsentiert gewissermaßen einen logischen Thread.

Eine Aktivität verrichtet, hinsichtlich der seriellen Bearbeitung von Methodenaufrufen denselben Job wie ein STA. Die Aktivität erfüllt ihre Mission jedoch sehr viel effizienter. Sie muss nicht jedes Objekt einem spezifischen Thread zuweisen. Die COM+-Laufzeit unterbindet die Parallelität in einer Aktivität stattdessen mit einem internen Sperrschema.

Die COM+-Laufzeit überwacht die Aktivitäten in den einzelnen Prozessen mit Hilfe einer internen Datenstruktur. Jede Aktivität besitzt eine identifizierende GUID und eine exklusive Sperrung. Jeder Thread muss diese Sperrung anfordern, bevor der Zugriff auf eine Aktivität und deren Objekte erfolgen kann. Sobald der Thread die Sperrung vorgenommen hat, werden alle anderen Threads blockiert, sofern sie versuchen, auf die Aktivität zuzugreifen. Wenn ein Thread Zugang zu einer Aktivität erhalten hat, muss er die Sperrung wieder freigeben, sobald er die Aktivität verlässt. Dies gewährt dem nächsten Thread den Zugang zur Aktivität. Wie Sie sehen, verhindert dieses Sperrschema mehr als ein STA die Parallelität. Dies geschieht jedoch, ohne den Zugriff auf einen einzelnen Thread zu beschränken oder ein unsichtbares Fenster mit einer verknüpften Nachrichtenwarteschlange erforderlich werden zu lassen. Das Schema bietet somit eine automatische Synchronisierung ohne die zusätzlich Last, die mit einem STA verbunden ist.

Apartments sind noch immer ein Bestandteil des COM+-Programmiermodells. Sie spielen jedoch eine weniger bedeutende Rolle als früher. Sie werden inzwischen primär dazu verwendet, Objekte zu verwalten, die eine Thread-Affinität aufweisen. Eine Aktivität beschränkt aber nicht den Zugriff auf einen einzelnen Thread. Nur ein STA kann garantieren, dass ein Objekt mit einer Affi-

nität zu einem spezifischen Thread niemals von einem anderen Thread berührt wird. Jedes Visual Basic-Objekt, das in einer COM+-Applikation ausgeführt wird, muss deshalb in einem STA vorhanden sein. Ein Visual Basic-Objekt sollte sich ebenfalls in einer Aktivität befinden (obwohl dies nicht notwendig ist).

Das Synchronization-Attribut

Die Verknüpfung zwischen einem Kontext und einer Aktivität wird während der Erstellung des Kontextes hergestellt. Wie Sie wissen, generiert die COM+-Laufzeit immer dann einen neuen Kontext, wenn sie ein Objekt erzeugt, das eigene kontextuelle Informationen benötigt. Jede konfigurierte Komponente besitzt ein Synchronization-Attribut, das der COM+-Laufzeit mitteilt, ob ein neuer Kontext mit einer Aktivität verknüpft werden soll. Wenn die COM+-Laufzeit einen neuen Kontext erstellt, der eine Aktivität benötigt, zeigt das Synchronization-Attribut außerdem an, ob der Kontext die Aktivität des Erzeugers erben oder eine neue eigene Aktivität erhalten soll. Tabelle 7.1 beschreibt, wie COM+ neue Objekte und Kontexte mit Aktivitäten verknüpft.

In der Praxis müssen Sie sich nicht umfassend mit den Unterschieden zwischen den Synchronisierungseinstellungen befassen, wenn Sie Komponenten mit Visual Basic erstellen. Sie sollten die Voreinstellung Required verwenden. Der Grund hierfür wird gleich erläutert. C++-Programmierer können jedoch Komponenten ohne Thread-Affinität erstellen, die nicht in einem STA ausgeführt werden müssen. Dies offeriert sehr viel mehr Optionen für das Synchronization-Attribut. C++-Programmierer wählen deshalb in der Regel eine Synchronisierungseinstellung aus, die davon abhängig ist, ob die Programmierer eine automatische Synchronisierung wünschen. (Das Entwickeln von COM+-Komponenten mit C++ ist ein Thema für ein anderes Buch.)

Synchronization-Attribut	Verknüpfungsmethode
Not Supported	Das Objekt wird in einem Kontext erstellt, der keine Aktivität besitzt.
Supported	Wenn der Erzeuger in einer Aktivität ausgeführt wird, wird das Objekt in einem Kontext erstellt, der die Aktivität ebenfalls nutzt. Wenn der Erzeuger nicht in einer Aktivität ausgeführt wird, wird das Objekt in einem Kontext erstellt, der keine Aktivität besitzt.
Required	Wenn der Erzeuger in einer Aktivität ausgeführt wird, wird das Objekt in einem Kontext erstellt, der die Aktivität ebenfalls nutzt. Wenn der Erzeuger nicht in einer Aktivität ausgeführt wird, wird das Objekt in einem Kontext mit einer neuen Aktivität erstellt.
Requires New	Das Objekt wird in einem Kontext mit einer neuen Aktivität erstellt.
Disabled	Der SCM ignoriert die Synchronization-Eigenschaft bei der Entscheidung, ob das Objekt einen eigenen Kontext benötigt. Das Objekt befindet sich dann nur in einer Aktivität, wenn es im Kontext des Erzeugers angeordnet wird und dieser Kontext eine Aktivität besitzt.

Tabelle 7.1: Wie COM+ neue Objekte und Kontexte mit Aktivitäten verknüpft.

HINWEIS: VON MTS ZU COM+

Alle MTS-Objekte werden in einer Aktivität ausgeführt.

Aktivitäten wurden erstmals mit dem MTS eingeführt. Im MTS muss jedes Objekt in einer Aktivität ausgeführt werden. Eine MTS-Komponente weist somit gewissermaßen immer die Synchronization-Einstellung Required auf. Der MTS verlangt außerdem von jedem Objekt, dass es in einem STA ausgeführt wird. Dies gleicht den Umständen, die für Visual Basic-Objekte in COM+ gelten. Im Gegensatz zu COM+ kann der MTS jedoch nicht die Vorteile von C++-Komponenten mit weiterentwickelten Threading-Fähigkeiten nutzen.

Apartment-Threaded-Objekte und Aktivitäten

Abbildung 7.4 zeigt das Layout von Visual Basic-Objekten in einem typischen Serverapplikationsprozess mit mehreren Clients. Die grundlegende Prämisse hinter dem COM+-Parallelitätsmodell besteht darin, dass jeder Client seine eigene Aktivität erhalten sollte. Sie sollten Designs vermeiden, in denen ein einzelner Client mehr als eine Aktivität anfordert. Sie sollten ebenfalls Designs vermeiden, in denen zwei Clients derselben Aktivität zugeordnet sind.

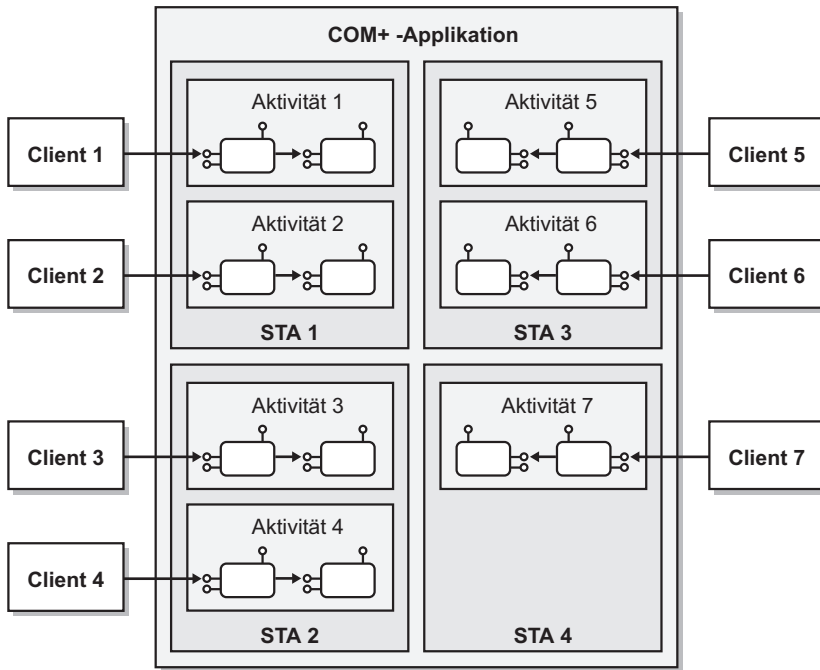


Abbildung 7.4: Jeder Client sollte mit genau einer Aktivität verknüpft werden. Die COM+-Laufzeit bindet logische Aktivitäten an physikalische STA-Threads, wenn Apartment-Threaded-Objekte involviert sind.

In welchem Fall würde ein Client mit mehreren Aktivitäten verknüpft? Dies kann geschehen, wenn ein Client, der außerhalb der COM+-Laufzeit ausgeführt wird, zwei separate Objekte erstellt. Beachten Sie, dass der Client nicht in einer Aktivität ausgeführt wird. Wenn der Client zwei Objekte aus Komponenten erstellt, die der Synchronisierung bedürfen, generiert die COM+-Laufzeit eine separate Aktivität für jedes Objekt. Eine bessere Lösung bestünde darin, den Client das erste Objekt und dieses Objekt das zweite Objekt erstellen zu lassen. Das zweite Objekt würde dann in derselben Aktivität wie das erste Objekt erzeugt, sofern das Synchronization-Attribut seiner Komponente auf Required gesetzt wäre.

Wieso ist es für beide Objekte wichtig, sich dieselbe Aktivität zu teilen? Dies hat mit dem Verbrauch von weniger Ressourcen und dem Vermeiden von unnötigen Thread-Wechseln zu tun. Hinsichtlich der Ressourcennutzung sollte Ihnen klar sein, weshalb Sie jeden Client auf einen einzelnen Thread begrenzen sollten. Ein Client benötigt nicht mehr als einen Thread. Ein Objekt kann außerdem ein anderes Objekt viel schneller aufrufen, wenn kein Thread-Wechsel erforderlich ist.

Wenn Objekt A Objekt B aufrufen wird, sollten Sie sicherstellen, dass beide im selben STA vorhanden sind. Die COM+-Laufzeit garantiert, dass zwei Apartment-Threaded-Objekte in demselben STA ausgeführt werden, wenn Sie sich in derselben Aktivität und in demselben Prozess befinden. Aus diesem Grund sollten Sie immer die Synchronization-Einstellung *Required* verwenden.

Sie wissen nun, dass es nicht gut für einen Client ist, mehr als eine Aktivität zu besitzen. Sie sollten außerdem darauf verzichten, zwei oder mehrere Clients einer Aktivität zuzuordnen. Die COM+-Architekten setzen voraus, dass jede Aktivität mit genau einem Client verknüpft ist. Wenn ein Client einen Aufruf in eine Aktivität durchführt, sollte er auf die Rückkehr des Aufrufs warten, bevor ein weiterer Aufruf erfolgt. Dieses Modell wurde nicht entwickelt, um Situationen gerecht zu werden, in denen zwei oder mehrere Clients unter Verwendung von zwei separaten Threads Aufrufe in dieselbe Aktivität durchführen. Die wichtigste Regel, die Sie in Erinnerung behalten sollten, ist, dass Sie niemals Aufrufe in eine Aktivität durchführen sollten, bevor der vorherige Aufruf die Programmsteuerung zurückgibt.

Ein Objekt, das von mehreren Clients gemeinsam genutzt wird, wird häufig als Singleton bezeichnet. Entwickler arbeiten in der Regel mit Singletons, damit Daten von mehreren Clients gemeinsam genutzt werden können. Während der Einsatz von Singleton-Objekten in einigen Einzelbenutzer-Szenarien akzeptabel ist, sollten Sie in COM+-Applikationen darauf verzichten. Jeder Client sollte seine eigenen privaten Objekte erhalten. Wenn mehrere Clients auf dieselben Daten zugreifen müssen, sollten Sie diese Daten im gemeinsamen Speicher oder in einer Datenbank ablegen. Sie sollten dann ein Design entwerfen, das jeden Client ein privates Objekt erstellen lässt, das wiederum auf diese gemeinsam genutzten Daten zugreift.

Sie sollten beachten, dass COM+ eine Aktivität nicht auf einen einzelnen Prozess beschränkt. Abbildung 7.5 zeigt ein Beispiel für eine Aktivität, die Prozesse überbrückt. Wenn Objekt A in einer Aktivität der Serverapplikation 1 ausgeführt wird und Objekt B in der Serverapplikation 2 erstellt, erben das neue Objekt und dessen Kontext die Aktivität des Erzeugers. Dieses Modell setzt jedoch voraus, dass die Komponente, mit der Objekt B erzeugt wird, die Synchronisierung verlangt.

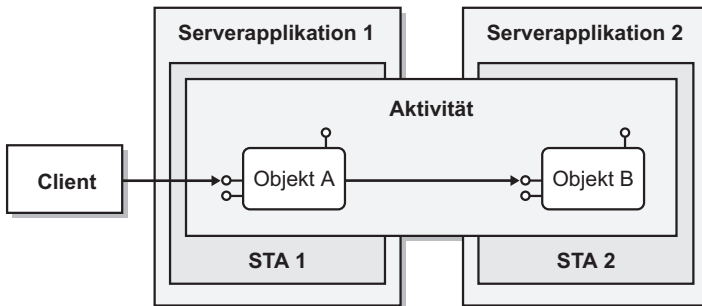


Abbildung 7.5: Eine Aktivität kann Prozess- und Computergrenzen überbrücken.

Wie Sie an der Abbildung erkennen können, überträgt COM+ kontextuelle Informationen über die Aktivität von Objekt A zu Objekt B. Dies geschieht während der Aktivierung. Im achten Kapitel werden Sie feststellen, dass sich Objekte in derselben Aktivität befinden müssen, um in derselben COM+-Transaktion vorhanden sein zu können. Zwei Objekte können jedoch auch dann an derselben Transaktion teilnehmen, wenn Sie auf verschiedenen Computern ausgeführt werden. Sie müssen sich nur in derselben Aktivität befinden.

Sie sollten sich darüber hinaus der Beschränkungen von Aktivitäten bewusst sein. Die COM+-Laufzeit garantiert keine aktivitätsweite Synchronisierung über Prozesse hinweg. Das von COM+ verwendete Sperrschema arbeitet lediglich innerhalb des Gültigkeitsbereichs eines einzelnen Prozesses. Es wäre für COM+ zu aufwendig, Sperrungen in anderen Prozessen zu überprüfen, bevor ein Thread auf eine Aktivität zuzugreifen darf. Was aber würde zum Beispiel in Abbildung 7.5 geschehen, wenn ein Client die Absicht hätte, eine Methode von Objekt A aufzurufen, während ein anderer Client einen Aufruf in Objekt B durchführen würde? COM+ kann nicht verhindern, dass in einer Aktivität zwei Aufrufe gleichzeitig ausgeführt werden. Diese Einschränkung sollte jedoch niemals zu einem Problem werden, da Ihre Designs so konzipiert sein sollten, dass diese Situation nicht eintritt. So lange Sie zwei separaten Clients nicht gestatten, mit derselben Aktivität verknüpft zu sein, ist das Schema zum Sperren einzelner Prozesse kein Thema.

Der STA-Thread-Pool

Jede COM+-Serverapplikation pflegt zwei separate Thread-Pools. Ein Thread-Pool bedient STA-Objekte, während der zweite Thread-Pool alle anderen Objekte bedient. Als Visual Basic-Programmierer sollten Sie sich ein wenig umfassender mit dem STA-Thread-Pool beschäftigen. Die COM+-Laufzeit richtet den STA-Thread-Pool gemäß der folgenden Regeln ein:

- Die anfängliche Poolgröße ist 7 addiert mit der Anzahl der Prozessoren.
- Die maximale Poolgröße ist 10 multipliziert mit der Anzahl der Prozessoren.
- Die Thread-Poolgröße erhöht sich mit der Anzahl der in die Warteschlange gestellten Anforderungen.
- In Zeiten mit einem geringen Verkehrsaufkommen wird die Poolgröße, basierend auf einem vom System zur Verfügung gestellten Code, der von Leerlauf-Timern ausgelöst wird, verringert.

HINWEIS: VON MTS ZU COM+

Die Thread-Poolgröße in MTS

Unter MTS 2 pflegt jeder Serverpaketprozess einen einzelnen Pool mit STA-Threads. Dieser Pool hat eine anfängliche Größe von 1 und eine maximale Größe von 100. Die Poolgröße wird mit der Anzahl der Aktivitäten erhöht. Im Gegensatz zu COM+ wird die Größe des MTS-Thread-Pools niemals verringert.

Nachdem sie diese Regeln kennen gelernt haben, stellen viele Programmierer die folgende Frage: Wie kann meine Applikation skalierbar sein, wenn sie auf 10 Threads begrenzt ist? Ihre Intuition sagt Ihnen vielleicht, dass das Erhöhen der Thread-Zahl das Blatt zum Besseren wenden wird. Dies ist jedoch nicht der Fall. Sie können nur dann in diesem Maße skalieren, wenn Ihre Applikation auf einem Computer mit nur einem Prozessor ausgeführt wird. Das Erhöhen der Thread-Zahl von 50 auf 100 oder 1000 würde lediglich den Gesamtdurchsatz Ihrer Applikation einschränken.

Das in COM+ integrierte Thread-Pool-Schema optimiert sich selbst hinsichtlich der Anzahl der zur Verfügung stehenden Prozessoren auf dem Host Computer. Die beste Möglichkeit, einer zunehmenden Anzahl von Benutzern gerecht zu werden, besteht darin, Ihre Hardware zu aktualisieren. Der Erwerb eines neuen Server-Computers, der mehr oder schnellere Prozessoren als das alte Gerät besitzt, ist ein geeigneter Ausgangspunkt. Sie sollten aber auch sicherstellen, dass der Computer über genug physikalischen Speicher verfügt, um die erwartete Benutzerzahl bedienen zu können.

Hier eine weitere Frage, die häufig gestellt wird: Wie viele Benutzer können auf eine COM+-Applikation zugreifen? Die Antwort ist von einigen Faktoren abhängig. Mit der richtigen Hardware

kann eine Applikation mehrere hundert Benutzer bedienen. Später in diesem Buch wird der Einsatz von HTTP im Vergleich zu COM und RPC für die Client-/Server-Kommunikation beschrieben. Wie Sie sehen werden, weist HTTP viele Skalierungsvorteile im Gegensatz zu RPC auf. Dies ist sehr wichtig, wenn eine Applikation über Tausende von Benutzern verfügt. Viele verteilte Applikationen benutzen deshalb COM und RPC für die Server-/Server-Kommunikation und HTTP für die Client-/Server-Kommunikation.

Aktivitäten und STA-Threads

Sie sollten darauf achten, dass Ihre Applikation nur eine begrenzte Anzahl von STA-Threads besitzt. COM+ weist zunächst jede Aktivität einem eigenen Thread zu. Wenn die Anzahl der Aktivitäten die STA-Zahl jedoch übersteigt, muss COM+ mehrere Aktivitäten demselben STA zuordnen. Merken Sie sich bitte, dass ein Client mit einem STA-Thread verknüpft wird, sobald eine Aktivität erstellt wird. In einer Applikation mit vielen Benutzern besitzt jeder STA eine eigene Gruppe mit Clients.

Denken Sie daran, dass STAs physikalische Threads und Aktivitäten logische Threads sind. Wenn sich zwei Aktivitäten dasselbe selbe STA teilen, verwaltet die COM+-Laufzeit diese Mehrfachbelegung innerhalb und außerhalb von Aktivitäten. Die von COM+ zur Verfügung gestellte Mehrfachbelegung ist sehr kompliziert.

Lassen Sie uns hierzu ein Beispiel betrachten. Werfen Sie einen Blick auf Abbildung 7.6. Stellen Sie sich vor, dass Client 1 eine Methode von Objekt A aufruft. Ein wenig später versucht Client 2, eine Methode von Objekt B aufzurufen. Objekt B befindet sich in einer anderen Aktivität als Objekt A. Beide Aktivitäten teilen sich jedoch dasselbe STA.

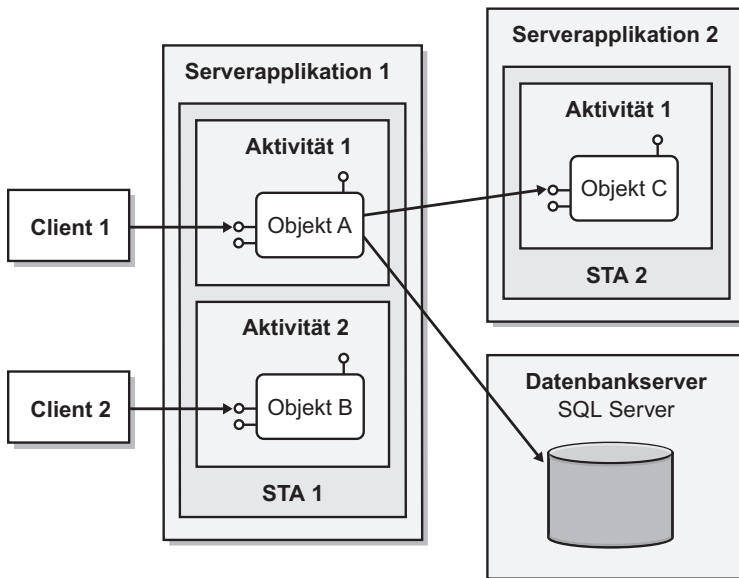


Abbildung 7.6: Eine Aktivität gibt die Kontrolle über ihren STA-Thread ab, wenn sie einen ausgehenden Aufruf durchführt oder von diesem zurückkehrt

Der Aufruf von Client 2 wird blockiert, während der STA-Thread damit beschäftigt ist, den Methodenaufruf für Client 1 auszuführen. Sie bemerken sicherlich, dass diese Blockierung nicht von

aktivitätsbasierten Sperrungen erzwungen wird. Die Blockierung erfolgt aufgrund der Single-Threaded-Natur des STAs. Doch was geschieht, wenn Objekt 1 einen ausgehenden Aufruf an einen anderen Prozess durchführt, während die Methode des Objekts ausgeführt wird? Vielleicht führt Objekt 1 einen COM-Aufruf an ein anderes Objekt in einer anderen Serverapplikation oder einen OLE-DB-Aufruf an einen Datenbank-Server durch. Ist der Aufruf von Client 2 in diesem Fall blockiert, während Objekt 1 auf die Rückkehr des ausgehenden Aufrufs wartet? Die Antwort lautet glücklicherweise »Nein«. Das Mehrfachbelegungsschema ist intelligent genug, um den STA-Thread freizugeben. Sobald der Aufruf von Objekt 1 den STA verlässt, kann eine andere Aktivität die Kontrolle über den Thread übernehmen.

Diese Sperrsemantik ist sehr komplex. Die Ausführung des Aufrufs an Objekt B kann nach dem Aufruf an Objekt A beginnen, aber vor diesem beendet sein. Dieses Verhalten hilft, die Parallelität zu verbessern. Es lockert jedoch die Regeln der Synchronisierung. Wenn Objekte in einem STA ausgehende Aufrufe durchführen, kann der STA nicht garantieren, dass die Aufrufe seriell bearbeitet werden. Aufrufe werden jedoch aus der Perspektive jeder Aktivität serialisiert.

Ich möchte Ihnen ein anderes Beispiel dafür geben, wie die losen Synchronisierungsregeln einen Programmierer überraschen können. Stellen Sie sich vor, ein Programmierer erstellt ein Objekt aus einer Visual Basic-Komponente, die keine Synchronisierung unterstützt. Das Objekt wird in einem STA erstellt, besitzt aber keine verknüpfte Aktivität. Stellen Sie sich weiter vor, dass zwei verschiedene Clients dieses Objekt referenzieren. (Ich weiß, dass Sie zwei Clients niemals gemeinsam ein Objekt benutzen lassen würden. Stellen Sie sich einfach vor, einer Ihrer Bekannten würde dies tun.)

Client 1 führt einen Methodenaufruf auf dem Objekt durch. Wenn die Ausführung dieser Methode beginnt, ändert sie eine modulweit gültige Variable auf den Wert 5 und initiiert einen ausgehenden Aufruf. Client 2 ruft nun eine Methode auf, die ausgeführt wird, während das Objekt auf die Rückkehr des ausgehenden Aufrufs wartet. Wenn die von Client 2 aufgerufene Methode die modulweit gültige Variable auf den Wert 6 ändert, wird Client 1 mit inkonsistenten Daten konfrontiert.

Wenn der ausgehende Aufruf initiiert wird, ist der Wert der Variablen 5. Wenn der Aufruf zurückkehrt, lautet der Wert 6. Der Aufruf von Client 1 wird somit nicht seriell durchgeführt. Würde das Objekt in einer Aktivität erstellt, gäbe es keine Probleme. Der Aufruf von Client 2 wäre in diesem Fall blockiert, bis der Aufruf von Client 1 zurückkehrt. Dieses Beispiel bietet einen weiteren guten Grund, die Synchronization-Voreinstellung *Required* für alle Visual Basic-Komponenten beizubehalten. Es zeigt außerdem, wie Clients, die Objekte gemeinsam verwenden, Sie in Schwierigkeiten bringen können.

Speicher gemeinsam nutzen

In der ersten Design-Phase einer verteilten Applikation müssen Sie entscheiden, wie Daten auf mehrere Clients verteilt werden sollen. Der Einsatz von gemeinsam genutztem Speicher in einer COM+-Applikation führt in der Regel am schnellsten zu diesem Ziel. Gemeinsam genutzter Speicher ist schneller als der Zugriff auf eine Datenbank, da die Daten sehr nahe am Ort des Geschehens sind. Sie müssen darüber hinaus nicht durch eine Datenbank-Engine navigieren, um Datenelemente zu lesen oder zu schreiben. Der Einsatz von gemeinsam genutztem Speicher in einer COM+-Applikation verlangt jedoch Ihre Aufmerksamkeit hinsichtlich der Synchronisierung. Verschiedene Objekte, die auf unterschiedlichen Threads ausgeführt werden, greifen schließlich auf dieselben Informationen zu.

Apartments und Standardmodulvariablen

Viele erfahrene Visual Basic-Programmierer sind überrascht, wenn sie ihren Code zum ersten Mal in einer Multi-Threaded-Umgebung wie zum Beispiel einer COM+-Applikation ausführen, da die öffentlichen Variablen, die in einem Standardmodul (.BAS) definiert sind, nicht wirklich globale Variablen sind. Diese öffentlichen Variablen sind eher auf der STA-Ebene als auf der Prozessebene gültig. Dies ändert einiges für die Programmierer, die daran gewöhnt waren, globale, applikationsweite Variablen in früheren Versionen von Visual Basic zu definieren. Die Regeln ändern sich jedoch, wenn Sie beginnen, für die mittlere Schicht zu programmieren.

Wenn Ihre Komponente in einem Multi-Threaded-Prozess ausgeführt wird, greifen zwei Objekte nur dann auf dieselben öffentlichen Standardmoduldaten zu, wenn sie sich in demselben STA befinden. Objekte in separaten STAs arbeiten mit unterschiedlichen Instanzen dieser öffentlichen Variablen, wie in Abbildung 7.7 dargestellt. Dies führt aufgrund der willkürlichen Weise, in der Objekte den STA-Threads zugeordnet werden, zu einem unerwarteten Verhalten in einer COM+-Applikation.

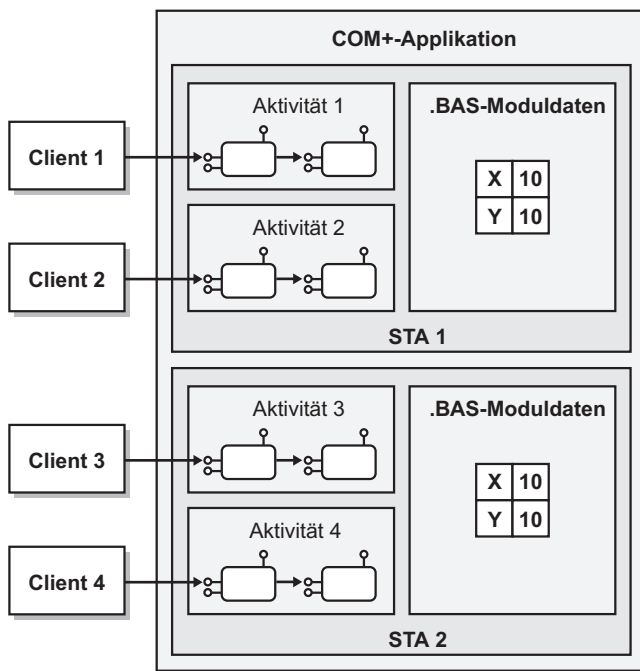


Abbildung 7.7: Jede Variable, die Sie in einem Standardmodul eines ActiveX-DLL-Projekts definieren, wird im TLS aufbewahrt. Dies bedeutet, dass Objekte, die in unterschiedlichen STAs ausgeführt werden, Daten nicht gemeinsam nutzen können.

Als die Visual Basic-Entwickler eine Unterstützung für das STA-Threading-Modell zu ihrer Programmiersprache hinzufügten, mussten sie entscheiden, wie die in Standardmodulen definierten, öffentlichen Variablen behandelt werden sollten. Wenn sie sich dazu entschlossen hätten, diese Daten im gemeinsam genutzten Speicher abzulegen, hätte jemand für die Synchronisierung des Zugriffs auf die Daten verantwortlich sein müssen. Das Team entschied wie im Fall der in die Visual Basic-Komponenten integrierten COM-Unterstützung, dass es nicht akzeptabel sei, eine vom Programmierer unterstützte Synchronisierung zu verlangen. Die in einem Standardmodul definierten Daten sollten deshalb im TLS gespeichert werden.

Wenn die Visual Basic-Laufzeit mit einem neuen STA konfrontiert wird, erstellt sie eine neue Kopie aller in einem Standardmodul definierten Variablen im TLS. Die Visual Basic-Laufzeit ruft außerdem einmal `Sub Main` in jedem STA auf. Sie können die Daten somit für jedes STA initialisieren. Beachten Sie, dass es mehrere Kopien von jedem Datenelement gibt. Während es möglich ist, Konstanten oder Variablen zu benutzen, die nach der Initialisierung nur gelesen werden können, verfügen Sie nicht über die Möglichkeit, ein Standardmodul zu verwenden, um gemeinsam genutzte Daten zu definieren, die fortwährend von den Clients aktualisiert werden. Sie müssen den Lese-/Schreibspeicher in einer COM+-Applikation mit Hilfe einer anderen Technik für die gemeinsame Nutzung freigeben.

Der Shared Property Manager

Die Typbibliothek *COM+ Services Type Library* enthält eine Komponente, die als Shared Property Manager (SPM) bezeichnet wird. Der SPM erlaubt Ihnen, gemeinsam genutzte Datenelemente in einer COM+-Applikation zu erstellen und darauf zuzugreifen. Alle Objekte, die in demselben Prozess ausgeführt werden, können auf jedes Datenelement zugreifen. Der SPM implementiert darüber hinaus ein Sperrschema, um den Datenzugriff zu synchronisieren. Dieses Sperrschema kann hinsichtlich der Parallelität sehr wichtig für die Aufrechterhaltung der Datenkonsistenz sein.

Der SPM stellt eine einfache aber effektive Form der Synchronisierung zur Verfügung. Anstatt eigenständige Eigenschaften zu benutzen, verlangt der SPM von Ihnen, jede Eigenschaft innerhalb des Gültigkeitsbereichs einer Eigenschaftengruppe zu definieren. Eine Eigenschaftengruppe ist ein Namensraum für mehrere Eigenschaften. Eine Eigenschaftengruppe ist ebenfalls ein Element der Synchronisierung. Was geschieht beispielsweise, um zu unserem früheren Beispiel zurückzukehren, wenn Ihre Daten auf einen Punkt verweisen und Sie ein Schreibobjekt benutzen möchten, um sowohl die x- als auch die y-Koordinate zu verändern, bevor ein Leseobjekt beide Änderungen sehen kann? Eine Eigenschaftengruppe bietet ein Sperrschema an, das dem Schreibobjekt gestattet, die Arbeit zu beenden, bevor ein Leseobjekt die Änderungen einsehen kann.

Abbildung 7.8 zeigt die Arbeitsweise des SPM. Unterschiedliche Objekte werden in verschiedenen Aktivitäten ausgeführt. Jedes Objekt erstellt seine eigene Instanz einer spezifischen Eigenschaftengruppe. Die Daten für die Gruppe werden jedoch nur einmal im Speicher für den gesamten Prozess abgelegt. Zusätzlich zu den Eigenschaftswerten besitzt jede Eigenschaftengruppe eine eigene exklusive Sperrung. Jede gemeinsam genutzte Eigenschaftengruppeninstanz implementiert einen internen Sperrmanager, der diese exklusive Sperrung durchführt, um den Zugriff auf die Eigenschaftsdaten innerhalb der Gruppe zu synchronisieren.

Der SPM ist eine Name-/Wert-Liste, die Sie Eigenschaften und Eigenschaftengruppen anhand eines Namens erstellen lässt. Lassen Sie uns hierzu ein Beispiel betrachten. Bevor Sie Eigenschaften und Eigenschaftengruppen erstellen und abrufen können, müssen Sie eine Instanz des Managers für gemeinsam genutzte Eigenschaftengruppen generieren. Werfen Sie einen Blick auf den folgenden Code:

```
Dim spam As SharedPropertyGroupManager
Set spam = New SharedPropertyGroupManager
Dim MyPoint As SharedPropertyGroup
Dim AlreadyExists As Boolean
' Erstellen und/oder binden an die Gruppe MyPointData
Set MyPoint = spam.CreatePropertyGroup("MyPointData", _
    LockMethod, _
    Process, _
    AlreadyExists)
```

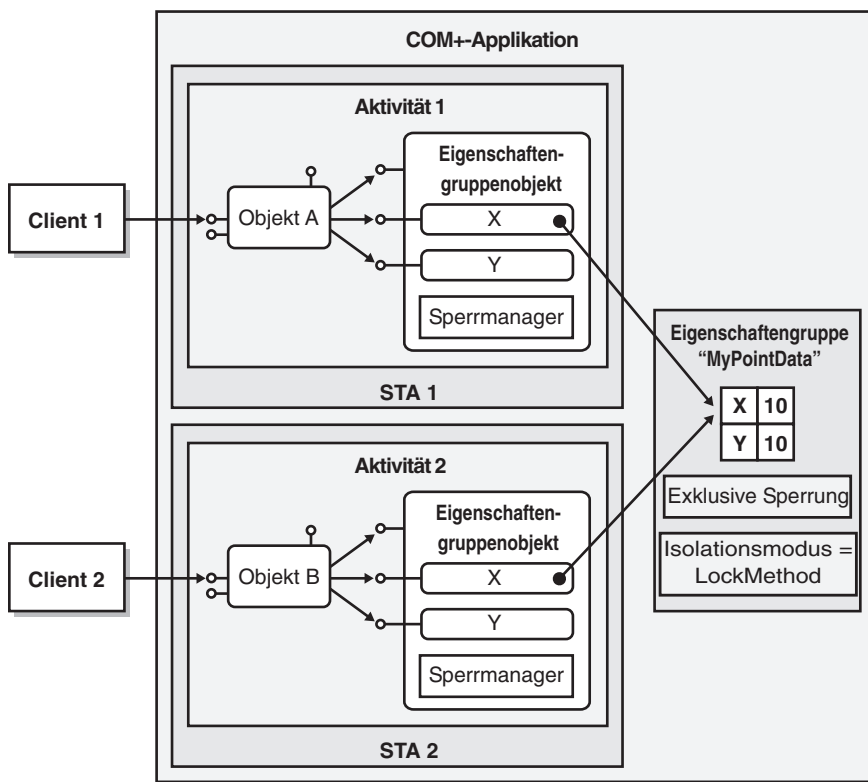



Abbildung 7.8: Der SPM bietet einen synchronisierten Zugriff auf gemeinsam genutzte Daten über die Eigenschaften in einer Eigenschaftengruppe an. Wenn Sie eine Eigenschaftengruppe erstellen, definieren Sie deren Sperrverhalten, indem Sie den Isolationsmodus setzen.

Die `CreatePropertyGroup`-Methode erwartet vier Parameter. Der erste Parameter lässt Sie den Namen der Eigenschaftengruppe übergeben. Wenn bereits eine Eigenschaftengruppe mit demselben Namen besteht, wird Ihr Code daran gebunden. Wenn die Eigenschaftengruppe nicht existiert, erstellt sie der SPM und bindet Ihren Code daran. Der letzte Parameter von `CreatePropertyGroup` ist ein boolescher Ausgabeparameter, der dem Aufrufer mitteilt, ob die Eigenschaftengruppe vor dem Aufruf bereits vorhanden war.

Der zweite und dritte Parameter spezifizieren einen Isolationsmodus und einen Freigabemodus für Ihre Eigenschaftengruppe. Der Isolationsmodus erlaubt Ihnen, das Sperrverhalten für die Gruppe einzurichten, während der Freigabemodus dem SPM mitteilt, wie lange die Daten der Gruppe im Speicher verwahrt werden sollen. Beachten Sie, dass beide Parameter als `ByRef`-Werte übergeben werden und dass der SPM ihre Eingabewerte ignoriert, wenn die Eigenschaftengruppe bereits existiert. Nur der Client, der die Gruppe erstellt, kann diese Werte einsehen. Wenn die Eigenschaftengruppe bereits vorhanden ist, gibt der SPM den aktuellen Isolationsmodus und Freigabemodus in diesen beiden Parametern als Ausgabewerte an den Aufrufer zurück.

Wenn Ihr Objekt eine Eigenschaftengruppeninstanz generiert und versucht, auf einige Eigenschaften dieser Gruppe zuzugreifen, muss es die Gruppe zunächst exklusiv sperren. Die Sperrung wird intern mit der Aktivität verknüpft, in der Ihre Objekte ausgeführt werden. Sobald die Aktivität Ihres Objekts die Sperrung durchgeführt hat, werden die Objekte in den anderen Aktivitäten

blockiert, wenn sie versuchen, auf die Eigenschaften in der Gruppe zuzugreifen. Die nächste Frage lautet: Wie lange hält die Aktivität die Sperrung aufrecht? Die Antwort ist davon abhängig, wie Sie das Sperrverhalten der Gruppe konfiguriert haben.

Wenn Sie eine Gruppe erstellen, richten Sie das Sperrverhalten mit dem Isolationsmodus `LockMethod` oder `LockSetGet` ein. Benutzen Sie `LockMethod`, wenn Sie die Konsistenz einiger Eigenschaften in der Gruppe für eine gewisse Zeit wahren möchten. Wenn der Isolationsmodus einer Gruppe auf `LockMethod` gesetzt wird, erhält jedes Objekt die exklusive Sperrung für die Dauer des aktuellen Methodenaufrufs. Die COM+-Laufzeit stellt Code in der Abfangschicht bereit, der die Sperrung wieder freigibt, sobald die Methode die Programmsteuerung zurückgibt. Dieses Schema erlaubt einem Objekt, mehrere Eigenschaften in einer Gruppe auszulesen oder zu setzen, ohne die Lese- und Schreibvorgänge der anderen Objekte berücksichtigen zu müssen.

Es gibt einige gute Gründe für das Erstellen einer Eigenschaftengruppe auf der Isolationsebene `LockMethod`. Vielleicht möchten Sie eine Eigenschaft auslesen und anschließend deren Wert ohne das Risiko aktualisieren, dass ein anderer Client den Eigenschaftswert zwischen Ihren beiden Operationen ausliest oder aktualisiert. Vielleicht weisen die Eigenschaften in Ihrer Gruppe auch gegenseitige Abhängigkeiten voneinander auf, wie im Fall der mit einem Punkt verknüpften Daten. Sie könnten zum Beispiel die x- und y-Position eines Punktes in einer Weise ändern wollen, die verhindert, dass ein anderer Client einen nicht gültigen Wert liest. In solchen Fällen benötigt jedes Objekt eine höhere Isolationsebene, wenn mehrere Aktualisierungen durchgeführt werden.

Wenn Sie `LockSetGet` übergeben, fordert der SPM die exklusive Sperrung der Gruppe immer dann an, wenn Sie auf eine individuelle Eigenschaft zugreifen. Im Gegensatz zu `LockMethod` wird eine Sperrung jedoch nicht für die Dauer einer Methodenausführung aufrecht erhalten. Beachten Sie, dass dies in einigen Designs dazu führen kann, dass Ihre Daten für die Inkonsistenz anfällig werden. Wenn Sie die höheren Ebenen der Isolation nicht benötigen, sollten Sie wissen, dass das kurze Aufrechterhalten von Sperrungen in höheren Ebenen der Parallelität resultiert.

Sie sollten die bestmögliche Balance zwischen Konsistenz und Parallelität finden. Es ist oft wünschenswert, verschiedene Eigenschaften in verschiedenen Gruppen anzuordnen. Einige Ihrer Gruppen können `LockMethod` benutzen, während andere `LockSetGet` verwenden. Sie erzielen auf diese Weise eine bessere Aufgliederung Ihres Sperrschemas, um die Konsistenz zu erhalten, die Sie benötigen, damit Benutzer nicht unnötig blockiert werden.

Wenn Sie den Freigabemodus einer Gruppe setzen, können Sie entweder den Wert `Process` oder `Standard` übergeben. Der Wert `Process` zeigt an, dass die Eigenschaftengruppe für die Dauer des Prozesses im Speicher aufbewahrt werden soll. Der Wert `Standard` gibt zu verstehen, dass die Eigenschaftengruppe selbst ihre Zeitdauer mit einem Standardreferenzzähler verwaltet. Wenn Sie `Standard` übergeben, werden die Daten der Eigenschaftengruppe aus dem Speicher entfernt, sobald der letzte Client sein Eigenschaftengruppenobjekt freigibt. Sie werden Eigenschaftengruppenobjekte oft erstellen und freigeben, aber die mit der Gruppe verknüpften Daten im Speicher behalten wollen. Dies bedeutet, dass Sie `Process` häufiger als `Standard` verwenden werden.

Der SPM ist natürlich erst dann wirklich nützlich, wenn Sie gemeinsam genutzte Eigenschaften erstellen und verwenden. Sie können eine Eigenschaft mit der `CreateProperty`- oder `CreatePropertyByPosition`-Methode erstellen. Nachfolgend finden Sie ein Beispiel, das zwei Eigenschaften erstellt und deren Standardwerte setzt:

```
Dim spam As SharedPropertyGroupManager
Set spam = New SharedPropertyGroupManager
Dim MyPoint As SharedPropertyGroup
Dim AlreadyExists As Boolean
Set MyPoint = spam.CreatePropertyGroup("MyPointData", _
```

```

LockMethod, _
Process, _
AlreadyExists)
' Erstellen und/oder Binden an Eigenschaft X
Dim X_Position As SharedProperty
Set X_Position = MyPoint.CreateProperty("X", AlreadyExists)
If AlreadyExists = False Then X_Position.Value = 10
' Erstellen und/oder Binden an Eigenschaft Y
Dim Y_Position As SharedProperty
Set Y_Position = MyPoint.CreateProperty("Y", AlreadyExists)
If AlreadyExists = False Then Y_Position.Value = 10

```

CreateProperty bindet Ihren Code an eine Eigenschaft, sofern diese bereits vorhanden ist. Sie können eine existierende Eigenschaft anhand ihres Namens oder ihrer Position abrufen. Wenn die Eigenschaft nicht vorhanden ist, erstellt sie der SPM und weist ihrer Value-Eigenschaft den Standardwert 0 (als langen Integer) zu. Wenn Sie einen anderen Vorgabewert wünschen, können Sie diesen explizit setzen, sobald die Eigenschaft erstellt wird.

Die Value-Eigenschaft eines gemeinsam genutzten Eigenschaftensobjekts wird als Variant gespeichert. Dies gewährt Ihnen ein wenig mehr Flexibilität. Sie können jeden Variant-fähigen Datentyp wie zum Beispiel einen Integer, Long, Double oder String, in einer Eigenschaft speichern. Sie können außerdem ein Variant-fähiges Array speichern. Das folgende Beispiel speichert ein Zeichenfolgen-Array in einer gemeinsam genutzten Eigenschaft.

```

Dim data() As String
ReDim data(3)
data(0) = "Bob"
data(1) = "Carol"
data(2) = "Ted"
data(3) = "Alice"
MyProperty.Value = data

```

Sobald Sie mit dem SPM vertraut sind, werden Sie feststellen, dass es sehr einfach ist, mit Objekten, die in unterschiedlichen Aktivitäten ausgeführt werden, auf gemeinsam genutzte Daten zuzugreifen. In einem Moment der Schwäche könnten Sie sich sogar dazu verleiten lassen, einer gemeinsam genutzten Eigenschaft eine Visual Basic-Objektreferenz zuzuweisen. Sie müssen diesem Impuls jedoch widerstehen. Wie Sie wissen, fügen sich gemeinsam genutzte Objekte nicht sehr gut in das aktivitätsbasierte Programmiermodell ein. Der Grund hierfür sind die Probleme mit der Thread-Affinität. Es gibt außerdem Probleme mit der Kontextrelativität. Wenn Sie ein Objekt in einer Eigenschaftengruppe speichern, können Sie die entsprechende Referenz nur aus demselben Kontext heraus legal benutzen. Der SPM sollte deshalb nicht für das Speichern von Objekten verwendet werden, die mit Visual Basic erstellt wurden.

Während Sie auf das Speichern von Objektreferenzen in eine Eigenschaftengruppe verzichten sollten, ist es üblich, eine Komponente für die Verwaltung der Daten einer spezifischen Gruppe zu erstellen. Es ist nur so, dass Ihre Applikation Objekte aus dieser Komponente erstellt und diese auch dann freigibt, wenn die Daten der Eigenschaftengruppe im Speicher vorhanden bleiben. Die Daten der Eigenschaftengruppe definieren ein logisches Objekt, das aus einfachen Datentypen und Arrays besteht. Sie können Objekte einfach erstellen, wenn diese benötigt werden, und mit den Daten in der Eigenschaftengruppe füllen. Solange Objekte innerhalb des Gültigkeitsbereichs einer einzelnen Aktivität existieren, können Sie die Probleme mit der Parallelität vermeiden.

Die Grenzen des SPM

Der SPM stellt Daten zur Verfügung, auf die Sie schnell zugreifen können. Sie sollten jedoch beachten, dass die Daten im Falle eines Systemfehlers oder des Absturzes Ihrer Applikation verloren gehen. Der Grund für die hohe Geschwindigkeit des SPM besteht darin, dass keine Daten auf einem Datenträger gespeichert werden. Wenn die Daten, die von mehreren Clients gemeinsam genutzt werden, wiederherstellbar sein sollen, müssen Sie eine persistente Speicherform, wie zum Beispiel eine Datenbank verwenden.

Daten, die vom SPM verwaltet werden, stehen nur den Objekten zur Verfügung, die in einem einzelnen Prozess ausgeführt werden. Wenn Daten über Prozesse oder Computer hinweg gemeinsam genutzt werden sollen, müssen Sie eine andere Technik benutzen. Auch hier bietet sich eine Datenbank an.

Merken Sie sich bitte, dass Lese- und Schreibvorgänge im SPM niemals innerhalb einer Transaktion ausgeführt werden. Der SPM bietet eine gewisse Isolation, kann aber einmal initiierte Vorgänge nicht rückgängig machen. Wenn Sie eine gemeinsam genutzte Eigenschaft aus einer COM+-Transaktion heraus modifizieren, bleiben Ihre Änderungen auch dann bestehen, wenn Sie die Transaktion zurücksetzen. Glauben Sie auch nicht, dass die Sperrungen Ihrer Eigenschaftengruppe bis zum Ende der aktuellen COM+-Transaktion aufrechterhalten werden. Die Sperrungen im SPM haben nichts mit COM+-Transaktionen zu tun. Der SPM gibt seine Sperrungen frei, sobald die Ausführung der aktuellen Methode beendet ist. Dies geschieht in den meisten Fällen, bevor die aktuelle Transaktion beendet ist.

Beachten Sie auch, dass der SPM im Gegensatz zu den meisten Datenbankprodukten niemals Lesesperrungen verwendet. Er setzt ausschließlich die exklusive Sperrung ein. Sobald ein Objekt die Sperrung einer Gruppe anfordert, ist jedes andere Objekt blockiert. Eine Datenbank kann eine optimierte Form der Parallelität zur Verfügung stellen, indem nur die Benutzer blockiert werden, die Schreiboperationen ausführen möchten. Diese zusätzliche Ebene der Sperrung bietet eine bessere Parallelität, da viele Leser gleichzeitig auf dieselben Daten zugreifen können.

Während des Betatests von Windows 2000 verwendete COM+ einen Dienst, der mit In-Memory Database (IMDB) bezeichnet war. Dieser Dienst war eine verbesserte Version des SPM. Mit seiner Hilfe konnten Operationen für den gemeinsam genutzten Speicher in Transaktionen zusammengefasst werden. Er bot außerdem eine detailliertere Sperrung. Der IMDB-Dienst war jedoch mit verschiedenen Problemen behaftet. Das Feedback der Kunden führte schließlich zu dem Entschluss des COM+-Teams, IMDB aus der finalen Version von Windows 2000 zu entfernen.

Der Zugriff auf IMDB geschah über einen OLE-DB-Provider und ADO. Der Dienst stellte jedoch keine Datenbank-Engine für die Ausführung von Abfragen zur Verfügung. Er war deshalb aus der Sicht erfahrener Datenbankentwickler nicht sehr funktionell. Der eigentliche Untergang von IMDB war jedoch darin begründet, dass diese Datenbank keine lokale Microsoft SQL Server-Datenbank überbieten konnte. Sie war schwieriger zu bedienen und weniger funktionell. Sie bot auch keine größere Geschwindigkeit als die Pflege von gemeinsam genutzten Daten in einer SQL Server-Tabelle.

Der IMDB-Dienst wurde nicht erwähnt, damit Sie wissen, was Sie nicht benutzen können, sondern um die Vorteile einer lokalen Datenbank zu demonstrieren, mit deren Hilfe Sie die Einschränkungen des SPM umgehen können. Der SPM wird immer den schnellsten Zugriff auf gemeinsam genutzte Daten bieten. Wenn Sie jedoch Beständigkeit, Daten, die über Prozesse hinweg gemeinsam genutzt werden, Lesesperrungen oder transaktionale Operationen wünschen, ist eine lokale Datenbank eine gute Wahl. Als COM+-Entwickler sollten Sie eine Datenbank niemals unterbewerten. Benutzen Sie sie, wenn dies erforderlich ist.

Datenbankverbindungen gemeinsam nutzen

Wenn Sie damit beginnen, Datenzugriffscode für die mittlere Schicht zu schreiben, werden Sie einige Besonderheiten bemerken, die sich von dem Datenzugriffscode einer Two-Tier-Desktop-Applikation unterscheiden. Sie sollten wissen, was Datenbankverbindungs-Pooling ist, wie diese Technik arbeitet und wie sie die Art und Weise ändert, in der Sie Datenzugriffscode schreiben. Dieser Abschnitt beschreibt den Entwurf und die Erstellung Ihres ADO-Codes für den Einsatz von Objekten der mittleren Schicht unter Verwendung des OLE-DB-Verbindungs-Pooling.

Wieso benötigen Sie Datenbankverbindungs-Pooling?

Das Einrichten einer Datenbankverbindung erfordert in der Regel die Bemühungen von zwei Computern. Der Client muss die Berechtigung des Benutzers an das DBMS (Datenbank-Managementsystem) übergeben, und das DBMS muss die Identität des Benutzers überprüfen und eine Zugriffsnummer an den Client zurückgeben. Dies erfordert Datenverkehr über das Netzwerk und Bearbeitungszyklen auf beiden Computern. Die Idee des Datenbankverbindungs-Poolings ist sehr einfach: Da eine Verbindung relativ aufwendig zu öffnen ist, sollte eine Applikation skalierbar sein, um einer zunehmenden Zahl von Benutzern gerecht zu werden. Dies ist jedoch nur dann möglich, wenn die Verbindungen in einem Cache abgelegt und von mehreren Clients gemeinsam genutzt werden können.

In einer zweischichtigen Applikation muss jeder Computer eine eigene Verbindung zum DBMS einrichten, wie in Abbildung 7.9 dargestellt. Die Tatsache, dass jede Client-Applikation in einem separaten Prozess auf einem anderen Computer ausgeführt wird, lässt die gemeinsame Nutzung von Verbindungen unmöglich werden. In diesem Applikationstyp wird eine Datenbankverbindung häufig eingerichtet, wenn der Benutzer die Client-Applikation startet. Die Verbindung bleibt geöffnet, bis die Applikation beendet ist.

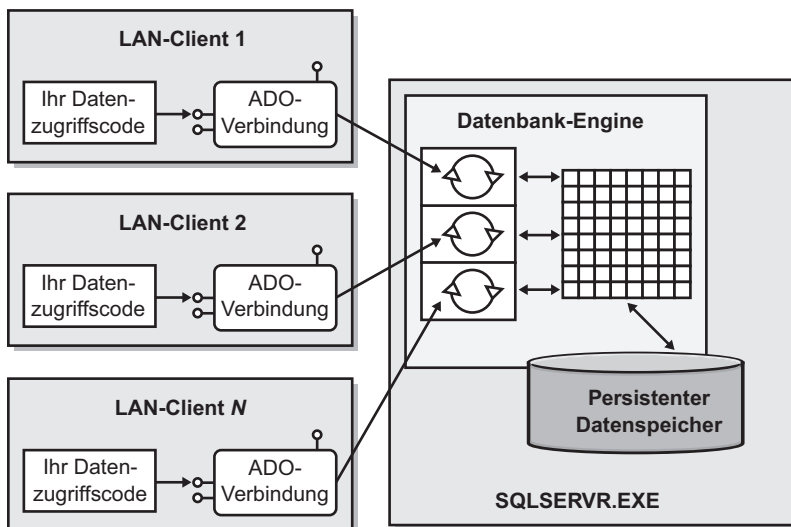


Abbildung 7.9: In einer zweischichtigen Applikation muss jeder Client eine eigene Verbindung zum DBMS herstellen. Da jede Client-Applikation auf einem separaten Computer ausgeführt wird, können Verbindungen nicht gemeinsam von den Benutzern verwendet werden.

Um die serverseitigen Ressourcen zu schonen, kann der Entwickler einer zweischichtigen Applikation entscheiden, eine Client-Verbindung zu schließen, wenn der Benutzer zehn Minuten verstreichen lässt, ohne auf die Datenbank zuzugreifen. Nachdem die Verbindung geschlossen ist, könnte der Benutzer jedoch eine Anweisung ausführen, um auf die Datenbank zuzugreifen. In diesem Fall kann die Client-Applikation transparent eine andere Verbindung herstellen. Diese Lösung erlaubt dem System, einer zunehmenden Zahl von Benutzern gerecht zu werden. Jeder Benutzer wird jedoch mit einer Zeitverzögerung konfrontiert, wenn eine neue Verbindung mit dem DBMS eingerichtet werden muss.

Die Architektur einer dreischichtigen Applikation bietet zahlreiche Vorteile gegenüber einer zweischichtigen Architektur. Ein wesentlicher Vorteil ist die Fähigkeit, Datenbankverbindungen gemeinsam zu nutzen. Wenn Clients Anforderungen von der Präsentationsschicht aus versenden, benutzen Objekte, die auf Weisung von vielen verschiedenen Benutzern ausgeführt werden, eine allgemeine Gruppe mit bereits geöffneten Verbindungen. Diese gemeinsame Nutzung erlaubt eine Skalierung, die in einer zweischichtigen Applikation nicht möglich ist.

Abbildung 7.10 zeigt die komplexe Architektur einer dreischichtigen Applikation, die das Datenbankverbindungs-Pooling benutzt. Wenn Sie diese Art von Applikation generieren, führt der Einsatz des Verbindungs-Pooling dazu, dass Ihr Code schneller ausgeführt wird und eine zunehmende Anzahl von Benutzern trotz weniger Datenbankverbindungen bedient werden kann. Die beste Nachricht ist jedoch, dass Sie keinen Code schreiben müssen, um Datenbankverbindungen auf mehrere Benutzer zu verteilen und zu verwalten. Diese Funktionalität ist bereits in OLE-DB und ODBC integriert. Wenn Ihre OLE-DB-Provider und ODBC-Treiber dieses Feature unterstützen, kann Ihr ADO-Code Vorteile daraus ziehen, sofern er in einer Middle-Tier-Umgebung wie zum Beispiel COM+ oder IIS ausgeführt wird.

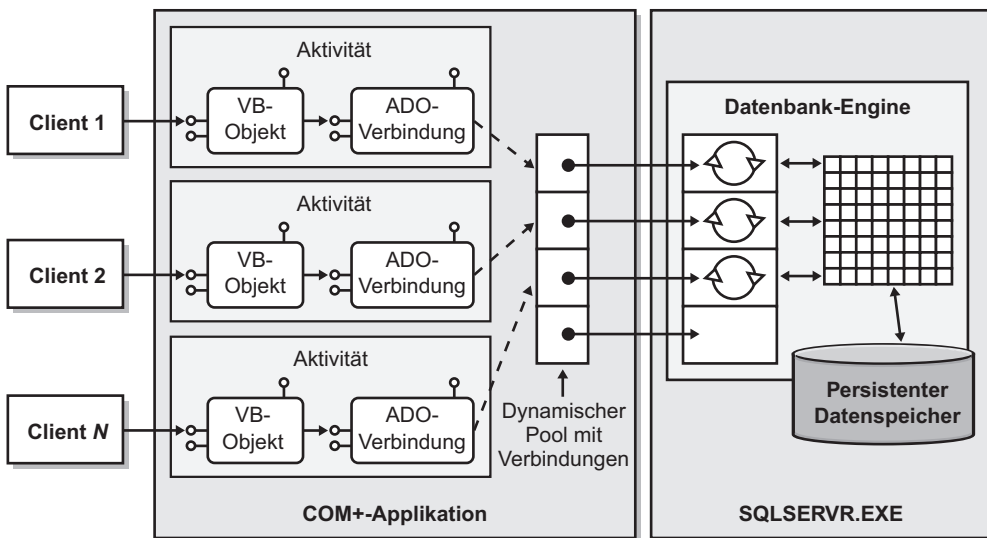


Abbildung 7.10: In einer dreischichtigen Applikation können Datenbankverbindungen gemeinsam genutzt werden. Sowohl ODBC als auch OLE-DB stellen die Basis für die Verbindungsverwaltung und das automatische Verbindungs-Pooling zur Verfügung.

Wie funktioniert das Verbindungs-Pooling?

Das automatische Verbindungs-Pooling war einige Zeit lediglich Bestandteil von ODBC. Mit Veröffentlichung von MDAC 2 (Microsoft Data Access Components) wurde das Verbindungs-Pooling auch zu OLE-DB hinzugefügt. Das OLE-DB-Verbindungs-Pooling wird mit Hilfe einer OLE-DB-Dienstkomponente ermöglicht. Diese Komponente arbeitet mit dem OLE-DB-Provider zusammen, den Sie verwenden, um mit dem DBMS zu kommunizieren. Sie öffnen und schließen ADO-Verbindungen einfach, und das Verbindungs-Pooling wird für Sie im Hintergrund durchgeführt.

Lassen Sie uns hierzu ein Beispiel betrachten. Stellen Sie sich vor, Sie arbeiten mit dem nativen OLE-DB-Provider für SQL Server (SQLOLEDB). Wenn Ihr Datenzugriffscode auf eine einzelne Datenbank eines Computers zugreift, der SQL Server ausführt, kann OLE-DB Ihre Verbindungen in einem einzigen Pool anordnen und verwalten. Wenn ein Objekt in einer COM+-Serverapplikation zum ersten Mal ein neues `Connection-ADO`-Objekt erstellt und die `Open`-Methode aufruft, muss OLE-DB eine Verbindung zum DBMS herstellen. Wenn Sie jedoch die `Close`-Methode aufrufen, entfernt OLE-DB die Verbindung zum DBMS nicht. Sie wird stattdessen in einem Pool angeordnet, so dass sie von anderen Objekten verwendet werden kann, die eine Verbindung benötigen.

Wenn der Code, der auf Weisung eines anderen Benutzers ausgeführt wird, die `Open`-Methode eines zweiten `Connection`-Objekts aufruft, sieht OLE-DB in dem Pool nach, um festzustellen, ob eine bereits vorhandene Verbindung zur Verfügung steht. Wenn es eine bereits hergestellte Verbindung findet, muss Ihre Applikation keinen Kontakt zum DBMS aufnehmen, um eine neue Verbindung einzurichten. Sie kann ihre Arbeit stattdessen unter Verwendung der ersten Verbindung durchführen. Auf diese Weise wird die Ausführungsgeschwindigkeit erhöht.

Lassen Sie uns die Beschränkungen des von OLE-DB verwendeten Verbindungs-Pooling-Schemas betrachten. Ein Verbindungs-Pool kann nur von den Objekten im selben Prozess benutzt werden. Wenn all Ihre Objekte in einer einzigen Instanz von `DLLHOST.EXE` oder `INETINFO.EXE` ausgeführt werden, können sie gemeinsam denselben Pool verwenden.³

Sie sollten ebenfalls beachten, dass jede Verbindung in einem Pool mit demselben OLE-DB-Provider, Server-Computer, derselben Datenbank, Benutzer-ID und demselben Kennwort verknüpft sein muss. Wenn Ihre Applikation eine Verbindung zu SQL Server und ORACLE herstellt, verwahrt OLE-DB einen separaten Pool für jedes DBMS. Sie müssen natürlich auch in diesem Fall dieselbe Benutzer-ID und dasselbe Kennwort verwenden, um all Ihre Verbindungen in demselben Pool anzuordnen. Wenn Sie wünschen, dass jedes Objekt in einer Serverapplikation den gleichen Verbindungs-Pool benutzt, muss jede Komponente dieselbe Verbindungszeichenfolge verwenden.

Wenn all Ihre Verbindungen aus dem Pool benutzt werden und ein Client eine Methode aufruft, um eine weitere Verbindung zu öffnen, richtet OLE-DB eine neue Verbindung mit dem DBMS ein. Der Pool kann somit dynamisch vergrößert werden. OLE-DB weist darüber hinaus jeder Verbindung im Pool ein Timeout-Intervall zu. Eine Verbindung, die mehr als 60 Sekunden im Leerlauf verbringt, wird von OLE-DB automatisch geschlossen. Der Pool kann somit auf eine effizientere Größe verkleinert werden, wenn einem hohen Verkehrsaufkommen eine Periode mit wenig Aktivität folgt.

Sie sollten auch wissen, dass abfangbare ADO-Fehler auftreten können, wenn das DBMS die maximale Anzahl von Verbindungen erreicht. Sie sind dann auf sich allein gestellt, da Ihnen weder OLE-DB noch ADO hilft, die aufgrund dieses Problems ausgelösten Fehler zu bearbeiten. Wenn SQL Server so konfiguriert ist, dass nur 50 gleichzeitige Verbindungen zulässig sind, und das Ver-

bindungs-Pooling-Schema beispielsweise versucht, die einundfünfzigste Verbindung zu öffnen, wird Ihr ADO-Code während des Aufrufs von `Open` mit einem Fehler konfrontiert. Sie müssen Code für diesen Eventualfall schreiben. Eine Möglichkeit der Reaktion auf diesen Fehler besteht darin, etwas später erneut zu versuchen, die Verbindung zu öffnen. Die ultimative Lösung besteht aber in der Regel darin, den Datenbankadministrator zu bitten, das DBMS für eine größere Anzahl von Verbindungen zu rekonfigurieren.

Es gibt eine weitere wichtige Einschränkung, die sich auf das Datenbankverbindungs-Pooling-Schema bezieht. Was geschieht zum Beispiel, wenn Ihre COM+-Applikation einen Pool mit zehn Verbindungen besitzt und der Computer, auf dem die SQL Server-Datenbank ausgeführt wird, neu gebootet wird? Jede Verbindung in dem Pool ist dann unbrauchbar. Sollten Sie eine COM+-Serverapplikation herunterfahren, wenn die Datenbank neu gestartet wird? Dies ist keine elegante Lösung, viele Unternehmen reagieren jedoch auf diese Weise auf das Problem.

Die Vorteile des Datenbankverbindungs-Pooling nutzen

Das Schreiben von Datenzugriffscode ist also nicht so einfach, wie Sie sich vielleicht vorgestellt haben. Ein Aufruf der `Close`-Methode bedeutet nicht immer, dass eine Verbindung geschlossen wird. Die Verbindung wird stattdessen wieder freigegeben und im Pool angeordnet. Ein Aufruf von `Open` bedeutet auch nicht immer, dass eine Verbindung geöffnet wird. Die Verbindung wird aus dem Pool entnommen. Wie Sie gesehen haben, kann ein Aufruf von `Open` zu einer zunehmenden Poolgröße führen, sofern alle vorhandenen Verbindungen bereits im Einsatz sind.

Ihre Objekte sollten eine Datenbankverbindung nicht nur für sich selbst verwenden. Die wichtigste Gewohnheit, die Sie sich aneignen sollten, ist, Verbindungen spät anzufordern und früh freizugeben. Je früher Ihre Objekte die Verbindungen wieder freigeben, desto größer ist die Chance, dass jedes Objekt eine vorhandene Verbindung aus dem Pool erhält. Wenn Ihre Objekte die Verbindungen jedoch länger als notwendig aufrechterhalten, muss Ihre Applikation möglicherweise zusätzliche Verbindungen einrichten, die nicht notwendig wären. Dies kann zu einer geringeren Performance und zu einer eingeschränkten Ressourcenverwendung führen.

Sie sollten Ihre Komponenten so entwerfen, dass sie Verbindungen innerhalb einer Methode anfordern und wieder freigeben. Hier ein guter Ausgangspunkt für eine Methode, die Datenzugriffscode enthält:

```
Sub MyMethod()  
    Dim conn As ADODB.Connection  
    Set conn = New ADODB.Connection  
    conn.Open sConnect ' (1) Verbindung einrichten  
    ' Fügen Sie Ihren Code hier ein. (2) Lese- und Schreiboperationen durchführen.  
    conn.Close ' (3) Verbindung schließen.  
    Set conn = Nothing  
End Sub
```

Das Deklarieren Ihrer ADO-Verbindungsobjekte im Deklarationsabschnitt Ihres Klassenmoduls hilft Ihrem Design in der Regel nicht und kann Sie in Schwierigkeiten bringen. In einer COM+-Applikation sollten Ihre `Connection`-ADO-Objekte immer am Ende jeder Anforderung geschlossen werden. Wenn Sie `Class_Initialize` und `Class_Terminate` benutzen, um ein `Connection`-Objekt auf der Klassenebene zu öffnen und zu schließen, halten Sie die Verbindungen wahrscheinlich länger als erforderlich. Betrachten Sie hierzu die folgende Klassendefinition der Komponente `CCustomerManager`:


```

' Klasse CCustomerManager
Private conn As ADODB.Connection

Sub Class_Initialize()
    Set conn = New ADODB.Connection
    conn.Open sConnect
End Sub

Sub AddCustomer(ByVal Customer As String)
    ' Fügen Sie hier den Code zum Hinzufügen eines neuen
    ' Kundendatensatzes mit conn ein.
End Sub

Sub Class_Terminate()
    conn.Close
    Set conn = Nothing
End Sub

```

Die CCustomerManager-Komponente erhält eine Verbindung, wenn sie erstellt wird, und gibt die Verbindung wieder frei, wenn sie terminiert wird. Der Client-Code, der diese Komponente benutzt, könnte wie folgt aufgebaut sein:

```

Dim CustMgr As CCustomerManager
Set CustMgr = New CCustomerManager
CustMgr.AddCustomer "Bob"
Set CustMgr = Nothing

```

Das Problem dieser Lösung besteht darin, dass Verbindungen geöffnet bleiben, während Sie darauf warten, dass die Visual Basic-Laufzeit Objekte erstellt und zerstört. Die Verbindung wird länger als erforderlich aufrechterhalten. Wenn der Client darüber hinaus vergisst, das CCustomerManager-Objekt auf Nothing zu setzen, benötigt die Visual Basic-Speicherbereinigung mehr Zeit, um die Verbindung zu schließen. Wenn Sie vergessen, Ihre Objekte auf Nothing zu setzen, werden Sie außerdem mit anderen Problemen konfrontiert, die dazu führen, dass die Objekte nicht korrekt terminiert werden. Sie sollten Ihre Verbindungen deshalb immer innerhalb von Methoden, die explizit vom Client aufgerufen werden, öffnen und schließen.

Stellen Sie sich nun vor, was geschieht, wenn Sie eine zweite Komponente besitzen, die mit CProductManager bezeichnet ist und der CCustomerManager-Komponente gleicht. Der Client wäre wie folgt aufgebaut:

```

Dim CustMgr As CCustomerManager
Set CustMgr = New CCustomerManager
CustMgr.AddCustomer "Bob"
Dim ProdMgr As CProductManager
Set ProdMgr = New CProductManager
ProdMgr.AddProduct "Dog"

```

Dieser Code führt zu Problemen, da er versucht, eine zweite Verbindung zu erhalten, bevor die erste Verbindung innerhalb des Gültigkeitsbereichs einer einzelnen Client-Anforderung freigegeben wird. Dieser Code verlangt zwei Verbindungen, obwohl eine genügen würde.

Sie können während der Design-Phase in eine Situation geraten, in der Sie ein Connection-ADO-Objekt im Deklarationsabschnitt eines Klassenmoduls deklarieren sollten. Dies ist möglich, wenn Sie mehrere Datenzugriffsmethoden eines Objekts in einer einzigen Anforderung aufrufen

müssen. Der Client könnte zum Beispiel dazu gezwungen sein, die `AddCustomer`-Methode wiederholt aufzurufen. In diesem Fall könnten Sie darauf verzichten, eine Verbindung aus dem Pool in einer einzigen Anforderung freizugeben und erneut abzurufen. Betrachten Sie die beiden neuen Methoden der folgenden Klassendefinition von `CCustomerManager`:

```
' Klasse CCustomerManager
Private conn As ADODB.Connection

Sub AcquireConnection()
    Set conn = New ADODB.Connection
    conn.Open sConnect
End Sub

Sub AddCustomer(ByVal Customer As String)
    ' AcquireConnection muss zuvor aufgerufen worden sein.
    ' Fügen Sie hier den Code zum Hinzufügen eines neuen
    ' Kundendatensatzes mit conn ein.
End Sub

Sub ReleaseConnection()
    conn.Close
    Set conn = Nothing
End Sub
```

Um dieses neue Design zu benutzen, muss der Client explizit Methoden aufrufen, um die Verbindung anzufordern und freizugeben. Der Client-Code wird somit wie folgt umgeschrieben:

```
Dim CustMgr As CCustomerManager
Set CustMgr = New CCustomerManager
CustMgr.AcquireConnection
CustMgr.AddCustomer "John"
CustMgr.AddCustomer "Paul"
CustMgr.AddCustomer "George"
CustMgr.AddCustomer "Ringo"
CustMgr.ReleaseConnection
Set CustMgr = Nothing
```

Wie Sie sehen, überträgt dieses Design die Verantwortlichkeit für die korrekte Vorgehensweise auf den Client. Viele Programmierer träumen von einer Welt, in der sich Objekte um sich selbst kümmern. Sie glauben, dass dies ein Versprechen der objektorientierten Programmiersprachen ist. Sie begründen Ihre Meinung damit, dass ein Objekt dazu fähig sein sollte, die eigenen Ressourcen transparent abzurufen und wieder freizugeben. Diese Praxis resultiert jedoch nicht in effizientem Code. Die soeben gezeigten Beispiele demonstrieren das Gegenteil. Sie sollten Verbindungen innerhalb von Methoden öffnen und schließen. Ihr Design kann andernfalls sehr komplex werden.

Wenn Ihre Designs komplexer werden, sind Sie dafür verantwortlich, Datenbankverbindungen schnellstmöglich freizugeben, nachdem Sie diese aus dem Pool erhalten haben. Dies erfordert in der Regel eine besondere Aufmerksamkeit für Details und eine durchdachte Zusammenarbeit der Objekte in Ihrer Applikation.

Poolfähige Objekte gemeinsam nutzen

Dieses Kapitel hat bisher beschrieben, wie mehrere Clients die Threads, den Speicher und Datenbankverbindung gemeinsam nutzen. Zusätzlich zu diesen Ressourcen bietet COM+ einen Recycling-Mechanismus für Clients an, um Objekte gemeinsam zu nutzen. Dieses Schema wird als Objektpooling bezeichnet. Das Objektpooling kann die Skalierbarkeit einer Applikation verbessern, da es während der Dauer eines Prozesses zu weniger Objekterstellungen und -zerstörungen führt. Visual Basic kann jedoch keine Komponenten generieren, die poolfähige Objekte erstellen. Diese Einschränkung ist mit den Problemen der Thread-Affinität verknüpft. Sie sollten dennoch verstehen, wie das Objektpooling arbeitet und weshalb es so wichtig ist.

Jeder Pool repräsentiert eine Gruppe mit Objekten, die aus derselben CLSID erzeugt wurden. Sobald Client A ein Objekt nicht mehr benötigt, deaktiviert die COM+-Laufzeit das Objekt und ordnet es wieder im Pool an. Wenn Client B ein Objekt aus derselben CLSID benötigt, gibt es keinen Grund dafür, ein neues Objekt zu erstellen. Die COM+-Laufzeit kann einfach ein Objekt aus dem Pool nehmen und es innerhalb der Aktivität von Client B aktivieren. Das Objektpooling gleicht dem Datenbankverbindungs-Pooling in viererlei Hinsicht. Viele verschiedene Clients erhalten eine spezifische Ressource und geben diese wieder frei. Ein vom System zur Verfügung gestellter Code ordnet wenige Ressourcen einer großen Zahl von Clients zu.

Das Objektpooling erfordert den Einsatz der Just-In-Time-Aktivierung. Wenn ein Client ein Objekt nicht mehr benötigt, muss die COM+-Laufzeit die Verknüpfung zwischen dem Client und dem Objekt lösen. Dieser Vorgang wird auch als Deaktivierung bezeichnet. Wenn der Client ein anderes Objekt benötigt, muss die COM+-Laufzeit dieses Objekt suchen und mit dem Client verknüpfen. Dieser Vorgang wird Aktivierung genannt. Das besondere an diesem Schema ist, dass der Client nicht den Eindruck hat, dass Objekte deaktiviert und reaktiviert werden. Er fordert lediglich eine Verbindung zu einem logischen Objekt an und führt einige Methodenaufrufe durch. Der Client weiß somit nicht, dass sich das physikalische Objekt, das den ersten Methodenaufruf ausführt, von dem Objekt unterscheiden kann, das den zweiten Methodenaufruf ausführt.

Poolfähige Objekte werden aus konfigurierten Komponenten mit dem `JustInTimeActivation`-Attribut `True` erstellt. Wenn COM+ einen neuen Kontext für ein Objekt erstellt, das für die Just-In-Time-Aktivierung konfiguriert wurde, fügt es ein spezielles Boolesches Flag hinzu, dass als Done-Bit bezeichnet wird, wie in Abbildung 7.11 dargestellt. Immer dann, wenn ein Aufruf von dem Objekt zurückkehrt, untersucht die COM+-Abfangschicht das Done-Bit. Wenn das Done-Bit auf `True` gesetzt ist, deaktiviert die COM+-Laufzeit das Objekt. Nun wird alles ein wenig kompliziert, da der Client, der das Objekt erstellt hat, mit einem Proxy verbunden ist, der über kein Objekt auf der anderen Seite verfügt. Während des nächsten Methodenaufrufs führt die COM+-Laufzeit jedoch eine Just-In-Time-Aktivierung durch, um die Anforderung des Clients zu bedienen.

Wie Sie sehen, ist das Done-Bit ein sehr wichtiger Bestandteil dieses Schemas. Das Done-Bit sollte in den meisten Fällen auf `True` gesetzt werden, um ein poolfähiges Objekt freizugeben, so dass es schnellstmöglich wieder im Pool angeordnet werden kann. Ihnen stehen zwei Möglichkeiten zur Verfügung, um das Done-Bit auf `True` zu setzen. Die erste Möglichkeit besteht darin, ein nur in der Methode gültiges Attribut namens `AutoComplete` zu konfigurieren. Wenn dieses Attribut aktiviert wird, setzt der Abfangcode das Done-Bit auf `True`, bevor die Methode ausgeführt wird. Sie können das Done-Bit auch programmgesteuert manipulieren. Die `IContextState`-Schnittstelle besitzt eine Methode namens `SetDeactivateOnReturn`. Diese Methode setzt das Done-Bit entweder auf `True` oder `False`. Beachten Sie, dass die `ObjectContext`-Schnittstelle über eine Methode mit der Bezeichnung `SetComplete` verfügt, die das Done-Bit auf `True` setzt und vorschlägt, die aktuelle Transaktion abzuschließen.

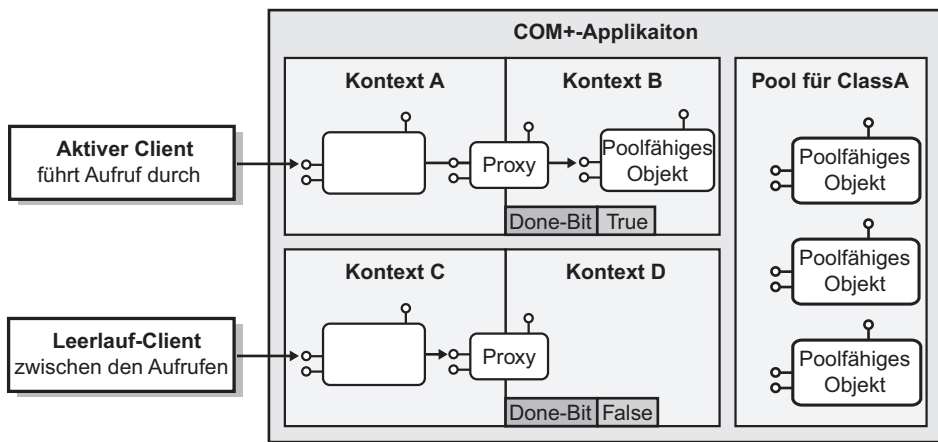


Abbildung 7.11: Das Objektpooling wird mit Hilfe der Just-In-Time-Aktivierung gesteuert. Ein Objekt in einem Pool wird in der Regel mit jedem Methodenaufruf aktiviert und deaktiviert.

Wenngleich Sie poolfähige Objekte nicht mit Visual Basic erstellen können, sollten Sie dennoch verstehen, wie das Done-Bit arbeitet, sofern Sie transaktionale Komponenten erstellen möchten. Transaktionale Komponenten gleichen poolfähigen Komponenten darin, dass sie ebenfalls die Just-In-Time-Aktivierung verlangen. Im achten Kapitel wird dieses Thema detailliert erläutert und verraten, wieso Visual Basic-Objekte, die in einer Transaktion ausgeführt werden, deaktiviert werden müssen.

Wieso ist das Objektpooling wichtig?

Das primäre Motiv für das Objektpooling besteht darin, dass einige Objekte der mittleren Schicht nur aufwendig erstellt und initialisiert werden können. Wenn einige dieser »aufwendigen« Objekte nur einmal erstellt und von mehreren Clients wiederverwendet werden können, kann der Server-Computer die Bearbeitungszyklen schonen, die mit dieser fortwährenden Objekterstellung und Initialisierung verbunden sind. Stellen Sie sich ein poolfähiges Objekt wie jede andere gemeinsam genutzte Ressource der mittleren Schicht vor.

Erwarten Sie nicht, dass Ihre Komponenten eine signifikant verbesserte Performance aufweisen, wenn Sie das Objektpooling einsetzen. Das Objektpooling spart Bearbeitungszyklen, die mit der kontinuierlichen Erstellung und Zerstörung von Objekten verknüpft sind. Es erfordert jedoch zusätzliche Bearbeitungszyklen für die Verwaltung des Pools. Das System muss ein Objekt nach dessen Deaktivierung im Pool anordnen und ein anderes Objekt für jede Just-In-Time-Aktivierung aus dem Pool entnehmen. Für die meisten einfachen Komponenten ist der Unterschied zwischen dem Einsatz des Poolings und dem Verzicht darauf nur gering. Für bestimmte Komponententypen ergeben sich jedoch enorme Vorteile aus dem Objektpooling.

Während der Entwurfsphase sollten Sie sich selbst fragen, ob eine Komponente ein geeigneter Kandidat für das Objektpooling ist. Sie sollten besonders darüber nachdenken, ob das Erstellen und Initialisieren dieses Objekttyps sowohl aufwendig als auch generisch ist. Wenn die Antwort auf diese Fragen ja lautet, kann das Objektpooling sehr vorteilhaft sein. Lautet die Antwort nein, ist das Objektpooling wahrscheinlich nicht sehr nützlich.

Lassen Sie uns ein Beispiel für eine Komponente betrachten, für die das Objektpooling Vorteile bereithält. Stellen Sie sich vor, Sie entwickeln eine Komponente, die eine Verbindung zu einer Mainframe-Applikation herstellt. Jedes Objekt benötigt fünf Sekunden, um die Verbindung einzurichten. Dies ist der aufwendige Teil. Nachdem das Objekt die Verbindung zur Mainframe-Applikation eingerichtet hat, kann das Objekt und seine Verbindung von mehreren verschiedenen Clients verwendet werden. Das ist der generische Teil. Das Objekt besitzt keinen client-spezifischen Status. Es ist in seinem initialisierten Status für jeden Client nützlich.

Abbildung 7.12 zeigt die Registerkarte *Aktivierung* des Eigenschaftendialogfeldes einer Komponente (das über das administrative Hilfsmittel *Komponentendienste* aufgerufen wird). Sie können hier die Poolingeinstellungen für eine Komponente in einer COM+-Applikation konfigurieren. (Diese Einstellungen sind für Visual Basic-Komponenten immer gesperrt.) Beachten Sie, dass die minimale Poolgröße auf 10 gesetzt ist. Wenn Sie Ihre Komponente auf diese Weise konfigurieren, erstellt und initialisiert die COM+-Laufzeit automatisch zehn Objekte, sobald die Applikation gestartet wird. Die Initialisierungszeit dieser Objekte wird dann während der Ausführung der Applikation amortisiert.

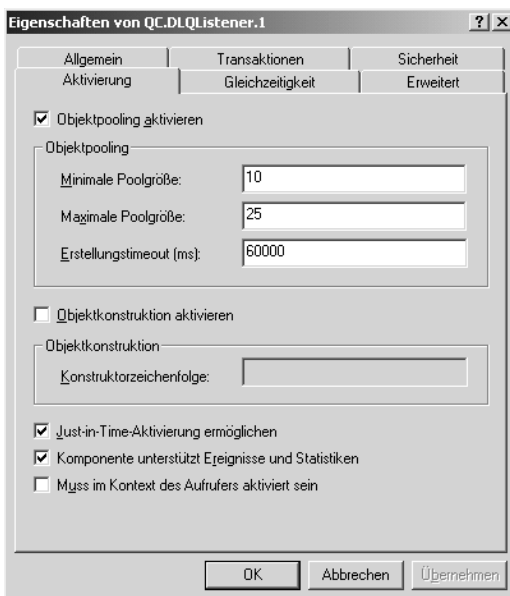


Abbildung 7.12: Sie können eine C++-Komponente für das Objektpooling konfigurieren, indem Sie das Eigenschaftendialogfeld der Komponente benutzen.

Der primäre Vorteil des Objektpoolings besteht darin, dass kein Client fünf Sekunden warten muss, bis ein Objekt in der mittleren Schicht eine Verbindung eingerichtet hat. Der Client erhält einfach ein Objekt mit einer bereits bestehenden Verbindung aus dem Pool. In einem Szenario wie diesem kann das Objektpooling die Reaktionszeit des Clients sowie den Gesamtdurchsatz der Applikation signifikant verbessern.

Zusätzlich zur Einsparung von Bearbeitungszyklen für das Erstellen und die Implementierung von Objekten, kann das Objektpooling eine Ressourcenbegrenzung zur Verfügung stellen. Diese begrenzt die Anzahl der Clients, die mehrere Objekte und deren Ressourcen verwenden. Sie können zum Beispiel die maximale Anzahl der ausstehenden gleichzeitigen Verbindungen zur Mainframe-Applikation beschränken, indem Sie eine Ressourcenbegrenzung für die Komponente konfigurieren.

Die Einstellung *Maximale Poolgröße* begrenzt die Anzahl der Benutzer, die die Komponente gleichzeitig verwenden können. Wenn Sie den Wert 25 angeben, können nur 25 Clients gleichzeitig auf die Komponente zugreifen. Sobald 25 Clients Objekte aktiviert haben, wird jeder weitere Client blockiert, bis einer der anderen Clients ein Objekt für den Pool freigibt. Ein blockierter Client wartet einfach auf das nächste verfügbare Objekt.

Wenn das Done-Bit gesetzt ist, wird ein Objekt immer dann an den Pool zurückgegeben, wenn eine Methode zum Client zurückkehrt. Das Setzen des Done-Bits ist nicht erforderlich, aber in den meisten Situationen empfehlenswert. Wenn das Done-Bit gesetzt ist, erhält und benutzt jeder Client ein Objekt für jeden einzelnen Methodenaufruf. Das Objekt wird dann nach dem Aufruf wieder freigegeben. In unserem Beispiel bedeutet dies, dass nur 25 Clients gleichzeitig einen Methodenaufruf durchführen können. Wenn diese Objekte das Done-Bit nicht sehen, gibt die maximale Größe die Anzahl der Clients an, die eine Verbindung zu einem Objekt herstellen können. Wie Sie sehen, ermöglicht ein gesetztes Done-Bit die Ressourcenbegrenzung und eine komplexere Parallelität.

Zusätzlich zur maximalen Poolgröße besitzt eine Komponente eine konfigurierbare Einstellung mit der Bezeichnung *Erstellungstimeout (ms)*. Ein Client wird nur für die Dauer des Erstellungstimeouts blockiert. Wenn ein Client diese Zeit während des Versuchs, ein Objekt aus dem Pool zu aktivieren, überschreitet, wird er mit einem Laufzeitfehler konfrontiert. Sie sollten hierauf reagieren, indem Sie eine Fehlerbehandlungsroutine für den Code schreiben, der das Objekt aktivieren möchte.

Lassen Sie uns das Objektpooling zusammenfassen. Es ist eine Technik für die Wiederverwendung von Objekten, deren Initialisierung sowohl aufwendig als auch generisch ist. Ich habe Ihnen hierzu ein Beispiel für eine Komponente demonstriert, die eine Verbindung zu einer Mainframe-Applikation herstellen muss. Es gibt viele andere Situationen, in denen Sie mit Komponenten konfrontiert werden, die diese Kriterien ebenfalls erfüllen. Es ist jedoch wichtig, dass Sie daran denken, dass eine Komponente die Vorteile des Objektpoolings nicht nutzen kann, wenn eines dieser Kriterien nicht erfüllt ist.

Objektpooling im Vergleich mit Datenbankverbindungs-Pooling

Das Objektpooling löst einige Probleme, die mit dem in OLE-DB integrierten Datenbankverbindungs-Pooling-Schema verbunden sind. Sie können die Anzahl der verfügbaren Verbindungen nicht mit dem Datenbankverbindungs-Pooling begrenzen. Das Objektpooling ist besser, da Sie jeden Prozess derart einrichten können, dass er die optimale Anzahl von Verbindungen nutzt. Die COM+-Laufzeit blockiert automatisch jeden Client, bis ein anderer Client eines der im Pool befindlichen Objekte freigibt.

Beachten Sie ebenfalls, dass poolfähige Objekte feststellen können, ob ihre Verbindung noch besteht. Dies bedeutet, dass ein Pool-Objekt eine neue Verbindung herstellen kann, wenn die bisherige Verbindung nicht mehr existiert. Sie können somit poolfähige Objekte entwickeln, die »sich selbst helfen«. Sie können außerdem bestimmte Probleme vermeiden, die mit poolfähigen Datenbankverbindungen auftreten. Wenn ein SQL Server-Computer neu gebootet wird und ein Pool mit Datenbankverbindungen nicht mehr vorhanden ist, benötigt der Code, der diese Verbindungen verwendet, eine Fehlerbehandlungsroutine. Code, der Objekte aus einem Pool benutzt, kann von solchen Problemen abgesichert werden.

Wenn Ihnen diese Fakten wichtig sind, werden Sie sich vielleicht dazu entschließen, poolfähige Objekte zu Ihrer Applikation hinzuzufügen. Diese Komponenten können den Code kapseln, der

mehrere Verbindungen zu einem Datenbank-Server oder einer Mainframe-Applikation herstellt und verwaltet. Sie können die Komponenten ebenfalls verwenden, um Verbindungen, die auf Sockets basieren, zu anderen Applikationen herzustellen. Eine poolfähige Komponente muss jedoch verschiedene Anforderungen erfüllen. Sie muss mit C++ erstellt werden. Ihr `ThreadingModel`-Attribut muss auf `Neutral` oder `Both` gesetzt werden, und sie kann keine Thread-Affinität aufweisen. Es gibt jedoch noch weitere Anforderungen.

Der Einsatz von C++ mit der ATL-Bibliothek (Active Template Library) bietet die einfachste Möglichkeit, COM+-Komponenten zu erstellen, die das Pooling unterstützen. Der ATL-Assistent führt die meisten Konfigurationsarbeiten selbstständig durch. Das Hinzufügen der Unterstützung für eine poolfähige Komponente, die an einer Transaktion beteiligt ist, erhöht den Programmieraufwand jedoch enorm. Dies ist ein Bereich der COM+-Entwicklung, in dem Sie Visual Basic leider nicht benutzen können.

Zusammenfassung

Was haben wir also gelernt? Die Skalierbarkeit einer Applikation beginnt auf der Prozessebene. Innerhalb eines Prozesses muss Code vorhanden sein, der für die effiziente gemeinsame Nutzung von Ressourcen verantwortlich ist. Ein Teil dieses Codes wird von Microsoft zur Verfügung gestellt, den anderen Teil müssen Sie selbst schreiben.

Das Verteilen von gemeinsam genutzten Daten auf mehrere Benutzer ist schwierig. Die Parallelität und Datenkonsistenz sind von Natur aus gegensätzlich. Wenn Sie Code schreiben, um eine dieser Techniken zu fördern, vernachlässigen Sie die andere. Der Schlüssel zum Erstellen der schnellstmöglichen Applikation ist das Verständnis dafür, wann die Sperrung eingesetzt und wann sie vermieden werden muss. Ich werde dieses Thema im nächsten Kapitel, das Transaktionen behandelt, fortführen.

Nebenbei bemerkt, verfügen Sie nun über die besten Voraussetzungen, um mehr über die Arbeitsweise der Transaktionen zu erfahren. Das Wissen über Aktivitäten, die Just-In-Time-Aktivierung und das Done-Bit ist wichtig, um das transaktionale Programmiermodell verstehen zu können. Diese Konzepte sind sehr wichtig, wenn Sie beabsichtigen, transaktionale Komponenten zu entwerfen und zu erstellen.

Im Laufe der letzten Jahre wurde die Visual Basic-Fähigkeit, Komponenten der mittleren Schicht zu produzieren, massiv weiterentwickelt. Einige Unternehmen verzichten mittlerweile sogar vollständig darauf, ihre Geschäftslogik und ihren Datenbankzugriffscode in anderen Sprachen zu schreiben. Einige Bereiche der COM+-Entwicklung erfordern jedoch noch immer C++. Wenn Sie eine poolfähige Komponente oder Code benötigen, der direkt unter Verwendung von OLE-DB geschrieben werden soll, müssen Sie auf die Entwickler zurückgreifen, die keine Angst vor Zeigern und Semikola haben. Wenn sie Ihnen das geben können, was Sie benötigen, sollten Sie ihnen jeden Betrag zahlen und definitiv nicht fragen, was sie für Sie getan haben.

8 Transaktionen programmieren

206	Was ist eine Transaktion?
213	Lokale Transaktionen programmieren
215	Programmierung verteilter Transaktionen
220	COM+ und deklarative Transaktionen
238	Transaktionale Entwürfe

Am Ende des letzten Kapitels habe ich die folgende Aussage getroffen: Wenn Sie schnelle Applikationen entwickeln wollen, müssen Sie wissen, wann Sperrungen gesetzt werden müssen und wann Sie das Sperren unterlassen sollten. Bestimmte Sperrungen sind unvermeidlich, wenn es um Applikationen mit vielen Clients und begrenzten Ressourcen geht. Ein Client muss darauf warten, dass ein anderer Client einen Thread freigibt, und ein Thread muss darauf warten, dass ein anderer Thread den Prozessor freigibt. Jeder Client braucht einen exklusiven Zugriff auf verschiedene Ressourcen, was der Parallelität und dem Gesamtdurchsatz einer Applikation nicht zuträglich ist.

Was Ressourcen wie Prozessoren, Threads und Datenbankverbindungen betrifft, können Sie den Anforderungen einer größeren Benutzerzahl gerecht werden, indem Sie mehr Hardware installieren. Wichtig ist, das Verhältnis zwischen Clients und Ressourcen sinnvoll zu dimensionieren, um Engpässe zu vermeiden und akzeptable Antwortzeiten zu garantieren. Entstehen die Engpässe jedoch für die Daten, auf denen Ihr Unternehmen basiert, gibt es für dieses Problem keine so einfache Lösung.

Angenommen, Sie entwickeln eine Applikation für ein Online-Handelsgeschäft, das Tiere verkauft, beispielsweise Hunde, Katzen und Vögel. Sie haben die Tabelle *Products*, die für jede Tierart eine Zeile enthält, und jede Zeile verwaltet den Lagerbestand im Feld *Quantity*. Es gibt jedoch ein Problem: Es gibt nur eine Stelle in der gesamten Applikation, wo der Lagerbestand der Hunde geführt wird. Der Datensatz, der die Menge der Hunde verwaltet, stellt einen Engpass für die Clients dar. Wenn mehrere Clients gleichzeitig versuchen, diesen Wert zu lesen oder zu schreiben, entstehen Probleme in Hinblick auf Konsistenz und Parallelität. Wenn immer mehr Clients auf den Lagerbestand der Hunde zugreifen wollen, wird es immer schwieriger zu verhindern, dass der Engpass die Leistung Ihrer Applikation beeinträchtigt.

Beinahe jede Geschäftsapplikation benutzt Daten in der einen oder anderen Form. Einige Applikationen fragen Daten lediglich ab und analysieren diese, während andere Anwendungen bestimmte Daten modifizieren. Applikationen, die Daten modifizieren und eine große Anzahl konkurrierender Benutzer bedienen, werden auch als Online-Transaktionsbearbeitungssysteme

(*Online Transaction Processing-Systeme* – OLTP) bezeichnet. Dieses Kapitel beschreibt solche Applikationen.

Sie sollten sich zunächst mit den Grundlagen der OLTP-Systeme beschäftigen, bevor Sie beginnen, Transaktionen für eine COM+-Applikation zu programmieren. Dieses Kapitel vermittelt Ihnen die Grundlagen der Transaktionsverarbeitung. Es erklärt, warum eine Transaktion die »ACID-Regeln« befolgen muss und was eine Transaktionsisolationsebene ist. Außerdem lernen Sie zwei Ansätze zur Programmierung von Transaktionen kennen. Beim ersten schreiben Sie lokale Transaktionen für ein einzelnes Datenbank-Management-System (DBMS) wie beispielsweise Microsoft SQL Server. Beim zweiten entwickeln Sie verteilte Transaktionen unter Verwendung einer komplexeren Infrastruktur, die COM+ und DTC (Distributed Transaction Coordinator) bereitstellen. Jede dieser Techniken ist für ganz bestimmte Situationen geeignet, Sie müssen also die richtigen Fragen stellen, um sich für einen geeigneten Ansatz zu entscheiden.

Es ist schwierig, unter Verwendung von SQL-Anweisungen wirklich optimierte OLTP-Applikationen zu entwickeln, die auf die DBM-Systeme unterschiedlicher Hersteller portierbar sind. Wenn es Ihnen hauptsächlich um die Leistung und die Skalierbarkeit Ihrer Applikation geht, müssen Sie häufig auf die Portierbarkeit verzichten. So schrecklich es sich anhört, es ist häufig einfach erforderlich, SQL-Anweisungen und gespeicherte Prozeduren für jedes einzelne DBMS zu schreiben. Aus diesem Grund verwende ich für die Beispiele in diesem Kapitel SQL Server und dessen eigene Sprache, Transact-SQL.

Ich habe SQL Server aus zwei Gründen ausgesucht. Erstens ist es in meinem MSDN-Abonnement enthalten, und zweitens habe ich mit diesem DBMS die meiste Erfahrung. Ich weiß, dass viele von Ihnen andere DBMS-Produkte verwenden, beispielsweise Oracle oder DB2. Grundsätzlich spielt es bei den Beispielen in diesem Kapitel keine wichtige Rolle, welches DBMS Sie verwenden. Wenn ich auf Besonderheiten zu Transact-SQL oder zum Sperrungsverhalten und zur Sperrungsstruktur hinweise, sollten Sie überprüfen, wie diese Funktionsmerkmale in Ihrem DBMS unterstützt werden.

Was ist eine Transaktion?

Eine typische Anweisung in einer OLTP-Applikation nimmt mehrere Lese- und Schreiboperationen vor, um eine einzelne Arbeitseinheit zu realisieren. Im Wesentlichen führt die Anweisung eine Transaktion aus. Eine Transaktion kann als *Folge von Anweisungen interpretiert werden, die Daten für einen einzigen Client lesen und ändern*. Wie Sie sehen, sind die Anforderungen einer OLTP-Applikation, die mehrere Transaktionen pro Sekunde ausführt, relativ hoch.

In diesem Kapitel verwende ich eine Beispielapplikation, die es den Clients ermöglicht, Verkaufsaufträge abzusetzen, um Produkte aus der *Animal Market*-Datenbank zu bestellen. Abbildung 8.1 zeigt das Datenbankschema für diese Applikation. Um alles möglichst einfach zu gestalten, wird in jedem Auftrag eines Kunden eine bestimmte Menge von nur einem einzigen Produkttyp bestellt. Ein Beispiel dafür wäre ein Auftrag von Bob, der fünf Hunde für 100 \$ (20 \$ pro Hund) bestellt. Die Applikation nimmt bestimmte Überprüfungen vor und führt die folgenden Aktualisierungen aus, um die Bestellung korrekt aufzuzeichnen:

- Die bestellte Menge wird vom Lagerbestand der Tabelle *Products* subtrahiert.
- Der Kaufpreis wird dem Kundenkonto in der Tabelle *Customers* belastet.
- Der Tabelle *Orders* wird ein neuer Datensatz hinzugefügt.

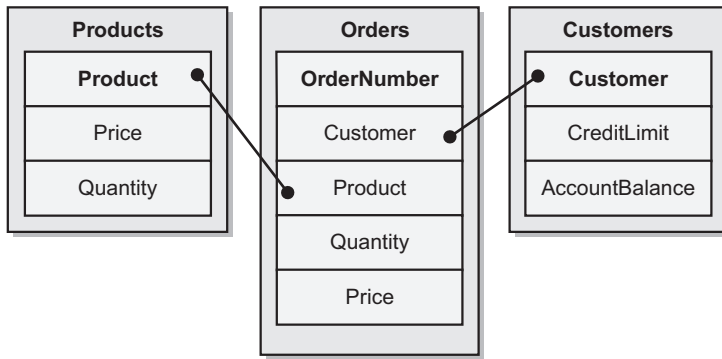


Abbildung 8.1: Die Applikation Animal Market basiert auf diesem Datenbankschema.

In einer realen Welt wird ein Kunde in der Regel mehrere unterschiedliche Produkte in einem Auftrag bestellen. Ich habe diese Möglichkeit bewusst nicht berücksichtigt, weil dafür eine vierte Tabelle erforderlich wäre, die die Auftragsdetails verwaltet. Mein einfacherer Datenbankentwurf erlaubt Ihnen, sich mehr auf wichtige OLTP-Konzepte zu konzentrieren und weniger auf die Theorie relationaler Datenbanken achten zu müssen.

Hier die SQL-Anweisungen, die Sie brauchen, um die Datenbank anzuweisen, die Bestellung aufzuzeichnen:

```
-- Lagerbestand mindern.
UPDATE Products
SET Quantity = Quantity - 5
WHERE Product = 'Dog'
-- Kundenkonto belasten.
UPDATE Customers
SET AccountBalance = AccountBalance + 100
WHERE Customer = 'Bob'
-- Neuen Auftragsdatensatz hinzufügen.
INSERT Orders(Customer, Product, Quantity, Price)
VALUES('Bob', 'Dog', 5, 100)
```

Möglicherweise wollen Sie auch einige Überprüfungen vornehmen, bevor Sie einen Auftrag festschreiben. Hier folgt ein Beispiel, wie man einen Batch mit Transact-SQL-Anweisungen schreibt, der vor einer Aktualisierung eine Überprüfung vornimmt. Der Batch löst einen Fehler aus, wenn die bestellte Menge nicht auf Lager ist.

```
DECLARE @Quantity int

SELECT @Quantity = Quantity
FROM Products
WHERE Product = 'Dog'

-- Benutzerdefinierten Fehler 50001 auslösen,
-- wenn weniger als fünf Hunde auf Lager sind.
If (@Quantity<5)
BEGIN
    RAISERROR(50001,16,1)
    RETURN
```

END

```
UPDATE Products  
SET Quantity = @Quantity - 5  
WHERE Product = 'Dog'
```

Dieses Beispiel unterstützt die in SQL Server gebotene Möglichkeit, benutzerdefinierte Fehlermeldungen an den Client zurückzugeben. Beim Aufruf von `RAISERROR` können benutzerdefinierte Fehlermeldungen verwendet werden, die einer Applikation mit der vom System bereitgestellten gespeicherten Prozedur `sp_addmessage` hinzugefügt wurden. Sie sollten wissen, wie man aus SQL-Anweisungen heraus korrekt Fehler auslöst und wie man sie in einer Microsoft Visual Basic-Komponente auffängt.

Ein realer Kaufauftrag umfasst vermutlich andere Überprüfungen als die oben gezeigten. Den Rest überlasse ich einfach Ihrer Vorstellungskraft. Sie haben genug SQL gesehen, um der Diskussion der Transaktionsentwicklung folgen zu können.

Transaktionssysteme

Die Entwicklung einer OLTP-Applikation ohne Transaktionsunterstützung von Seiten Ihres DBMS oder der zugrunde liegenden Plattform wäre sehr schwierig, weil während einer Transaktion sehr viel falsch laufen kann. Die Benutzer setzen häufig Anfragen ab, die die Geschäftsregeln verletzen. Was passiert beispielsweise, wenn in einem Auftrag ein Produkt bestellt wird, das nicht auf Lager ist? Was passiert, wenn ein Kunde nicht genügend Kredit besitzt, um die Bestellung aufgeben zu können? Und es können auch Systemfehler auftreten. Was passiert, wenn das Transaktionsprotokoll voll ist und die Datenbank nur noch gelesen werden kann? Was passiert, wenn der DBMS-Computer nicht erreichbar ist, weil das System oder das Netzwerk ausgefallen ist? Alle diese Szenarien verhindern, dass der Benutzer eine Transaktion korrekt und vollständig ausführt.

Wenn Sie keine Unterstützung von Ihrem DBMS erfahren, müssen Sie sehr viel Code schreiben, der jedes mögliche Fehlerszenario abdeckt. Wenn eine Transaktion die erste Aktualisierung erfolgreich vornehmen kann, die zweite dagegen nicht, muss die erste Operation rückgängig gemacht werden. Angenommen, Sie nehmen fünf Hunde aus dem Lagerbestand, stellen aber dann fest, dass der Kunde nicht über genügend Kredit verfügt, um diese Hunde zu kaufen. Sie brauchen eine ausgleichende `UPDATE`-Anweisung für die Tabelle `Products`, um den Lagerbestand wieder zu korrigieren. Oder stellen Sie sich nur eine komplexere Transaktion vor, wo der Fehler erst in der zehnten Operation auftritt. Sie müssen neun Aktualisierungen rückgängig machen, um die Datenbank wieder in ihren ursprünglichen Zustand zurückzusetzen.

Ein weiteres Problem beim Rückgängigmachen von Schreiboperationen gemäß der oben beschriebenen Vorgehensweise ist, dass dabei die Parallelität nicht berücksichtigt wird. Wenn sehr viele Clients Ihre Applikation mit vielen Anfragen pro Sekunde bestürmen, müssen Sie alle möglichen Sperrungslogiken schreiben, die Konsistenz und Synchronisierung gewährleisten. Betrachten Sie beispielsweise die folgende SQL-Anweisung:

```
UPDATE Table1  
SET Field1 = Field1 * 2  
WHERE ID = 'ID28'
```

Diese Anweisung scheint ganz einfach zu sein. Sie verdoppelt den Wert von `Field1`. Aber wie machen Sie diese Aktualisierung rückgängig? Wenn Sie garantieren können, dass keine andere Transaktion den Wert von `Field1` geändert hat, verwenden Sie dazu die folgende Anweisung:

```
UPDATE Table1  
SET Field1 = Field1 * 0.5  
WHERE ID = 'ID28'
```

Ändert jedoch eine andere Transaktion zwischen der ersten SQL-Anweisung und der ausgleichenden Zurücksetzung den Wert, geraten die Daten in einen inkonsistenten Zustand. Um sich dagegen zu schützen, müssen Sie einen Sperrmechanismus implementieren, der verhindert, dass andere Transaktionen den Feldwert zwischen der Ausführung der beiden Anweisungen ändern.

Glücklicherweise brauchen Sie in Ihrer Applikation nicht dafür zu sorgen. OLTP-Anwendungsprogrammierer können die in der zugrunde liegenden Infrastruktur bereitgestellten Transaktionsdienste nutzen. DBM-Systeme wie SQL Server, Oracle und DB2 bieten umfangreiche Unterstützung für die Transaktionsverarbeitung. Andere Datenquellen, wie beispielsweise Mainframe-Applikationen und Nachrichtenwarteschlangen, bieten häufig eine eigene Form der Transaktionsunterstützung.

Diese eingebaute Unterstützung macht es viel einfacher, eine Transaktion zu schreiben. Sie starten eine Transaktion, nehmen verschiedene Lese- und Schreiboperationen vor und schreiben die Transaktion dann fest oder machen sie rückgängig. Das zugrunde liegende System übernimmt einen Großteil der schwierigen Arbeit beim so genannten Rollback, also dem Rückgängigmachen der Transaktion, und bei den Sperrungen.

Ein Überblick über die »ACID«-Regeln

Wenn Sie sich bereits mit der Theorie der Transaktionsverarbeitung beschäftigt haben, dann haben Sie zweifellos schon von den ACID-Regeln gehört. Jede Transaktion muss vier Bedingungen erfüllen. Sie muss *atomar* (atomic), *konsistent* (consistent), *isoliert* (isolated) und *beständig* (durable) sein. Diese Bedingungen werden als ACID-Regeln bezeichnet. Wenn Sie eine Transaktion für ein OLTP-System schreiben, müssen alle daran beteiligten Komponenten diesen Regeln genügen. Möglicherweise haben Sie sie schon unzählige Male gesehen, aber wir wollen sie hier noch einmal wiederholen.

- **Atomar (atomic)** Eine Transaktion muss »Alles« oder »Nichts« sein. Entweder sollten alle oder keine Daten aktualisiert werden.
- **Konsistent (consistent)** Einzelne Operationen innerhalb einer Transaktion können Daten in einem Zustand hinterlassen, der die Integritätsvorgaben des Systems verletzt. Nachdem ein OLTP-System eine Transaktion beendet hat, müssen sich die Daten jedoch in einem zulässigen Zustand befinden.
- **Isoliert (isolated)** Eine Transaktion kann die noch nicht vollzogenen Änderungen einer anderen Transaktion nicht einsehen. Das bedeutet, dass die nicht festgeschriebenen Änderungen einer Transaktion von allen anderen Transaktionen isoliert sein müssen.
- **Beständig (durable)** Wenn eine Transaktion festgeschrieben wird, müssen alle daran beteiligten Datenquellen die vorgenommenen Änderungen dauerhaft speichern. Die Änderungen müssen im Falle eines Systemfehlers wiederhergestellt werden können.

Betrachten wir einen Moment die Bedeutung der dritten ACID-Regel. Das System muss die nicht festgeschriebenen Änderungen einer Transaktion vor anderen Transaktionen verbergen, weil sie möglicherweise irgendwann rückgängig gemacht werden. Betrachten wir ein Beispiel für ein Sys-

tem, das keine Isolation realisiert. Wenn Transaktion A eine Änderung vornimmt, vergeht eine bestimmte Zeitspanne, bis die Änderung festgeschrieben ist. Wenn Transaktion B die nicht festgeschriebenen Änderungen von Transaktion A sieht und nutzt, können jedoch Probleme auftreten. Was passiert, wenn Transaktion A rückgängig gemacht wird? Transaktion B erhält Daten, die es eigentlich nicht geben darf. Das Lesen nicht festgeschriebener Daten wird auch als *Dirty Read* bezeichnet. Wie Sie sehen, muss ein System die nicht festgeschriebenen Daten isolieren, um die Datenkonsistenz zu gewährleisten, wenn Transaktionen gleichzeitig ausgeführt werden.

Als Programmierer verlassen Sie sich darauf, dass Ihnen Transaktionssysteme die erforderlichen Isolationsebenen mit Hilfe von Sperrungen bereitstellen. Ein DBMS wie SQL Server beinhaltet einen Transaktionsmanager und einen Sperrmanager, wie in Abbildung 8.2 gezeigt. Der Transaktionsmanager erzwingt die ACID-Regeln. Zusammen mit dem Sperrmanager sorgt er dafür, dass jede Transaktion die erforderliche Isolation erhält.

Wenn Transaktion A einen Datensatz ändert, werden alle anderen Transaktionen, die diesen Datensatz lesen wollen, blockiert, bis Transaktion A abgeschlossen ist. Nachdem Transaktion A festgeschrieben wurde, erlaubt der Sperrmanager Transaktion B, den Datensatz mit seinem neuen Status zu lesen. Wurde Transaktion A rückgängig gemacht, liest Transaktion B den Datensatz in seinem ursprünglichen Status. In beiden Fällen verhindert der Sperrmanager, dass Transaktion B die nicht festgeschriebenen Änderungen sieht. Transaktion B muss warten, bis Transaktion A freigegeben wird.

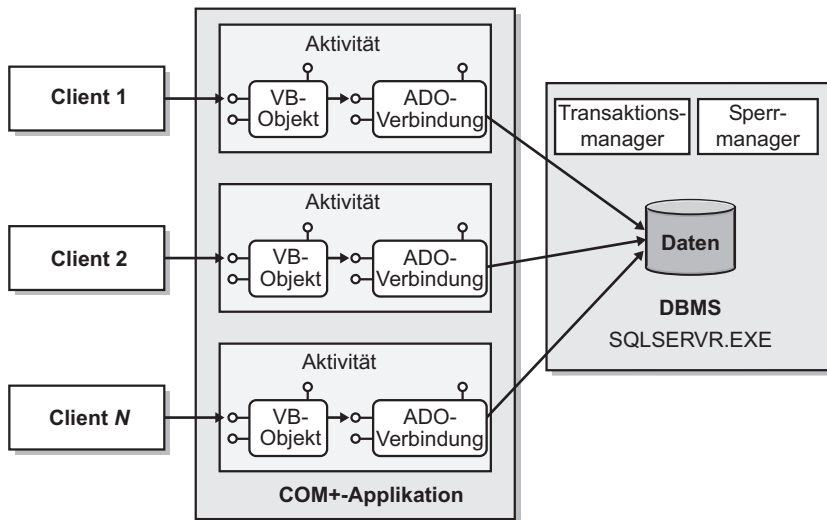


Abbildung 8.2: Der Sperrmanager eines DBMS erzwingt die Synchronisierung durch eine Sperrrichtlinie.

An dieser Stelle will ich Sie auf einige Beobachtungen hinweisen. Für die Datenkonsistenz braucht man Isolation und für die Isolation braucht man Sperrungen. Sperrungen behindern die Parallelität, weil sie zu Blockierungen führen. Blockierungen verringern die Antwortzeiten und den Durchsatz einer Applikation. Aus diesen Tatsachen können wir zwei wichtige Regeln für die Optimierung einer OLTP-Applikation ableiten:

- Fordern Sie keine unnötigen Sperrungen für Datenelemente an.
- Achten Sie darauf, dass die Sperrungen nur so lange wie nötig aufrechterhalten werden.

Sperrungen und Feinstrukturierung

Die meisten Sperrmanager verwenden mehrere Stufen für Datensperrungen. Grob strukturierte Sperrungen (wie beispielsweise für eine ganze Tabelle) isolieren größere Datenmengen. Fein strukturierte Sperrungen (wie beispielsweise für einzelne Datensätze) isolieren kleinere Datenmengen. Versucht eine Transaktion, jeden Datensatz in einer Tabelle zu ändern, sperrt der Sperrmanager am besten die ganze Tabelle, anstatt jeden einzelnen Datensatz zu sperren und wieder freizugeben. Das gilt besonders für Tabellen mit mehr als 100000 Zeilen.

Eine Tabellensperrung hat jedoch auch Nachteile – sie beeinträchtigen die Parallelität ganz wesentlich. Nachdem eine Transaktion eine exklusive Sperrung für eine Tabelle angefordert hat, muss Transaktion B warten, wenn sie auf einen Datensatz der Tabelle zugreifen will. Das ist insbesondere dann ungeschickt, wenn zwei Transaktionen nicht an denselben Datensätzen interessiert sind. Sperrungen auf der Datensatzebene sperren nur die Daten, die wirklich von der Transaktion genutzt werden. Sperrungen auf der Datensatzebene sind vorzuziehen, wenn eine Transaktion einen einzigen Datensatz in einer großen Tabelle ändern will. Feiner strukturierte Sperrungen haben weniger Einfluss auf die Parallelität.

SQL Server fügt eine zusätzliche Sperrebene ein, auch als *Seitensperrung* bezeichnet wird. Eine Seitensperrung ist irgendwo zwischen Datensatzsperrungen und Tabellensperrungen anzuordnen. SQL Server speichert die Datensätze in Seiten. Eine Standardseitengröße hilft SQL Server, die Festplattenein- und -ausgabe zu optimieren, wenn Daten zwischen Arbeitsspeicher und Festplattenspeicher verschoben werden. Außerdem wird damit eine Sperrstruktur realisiert, die eine schnellere Leistung erlaubt als die Datensatzsperrung, aber weniger Einschränkungen als die Tabellensperrung nach sich zieht.

SQL Server 6 und frühere Versionen unterstützen nur Sperrungen auf der Seiten- und Tabellenebene. SQL Server 6.5 führte ein Schema ein, in dem Sie Datensatzsperrungen auf der letzten Seite einer Tabelle benutzen konnten, wenn Sie Zeilen eingefügt hatten. Dieses Funktionsmerkmal ist jedoch standardmäßig nicht aktiviert. Sie müssen es für jede Tabelle separat aktivieren. SQL Server 7 ist die erste Version, die echte Datensatzsperrungen unterstützt.

Größtenteils kann man sich darauf verlassen, dass der Optimierer von SQL Server die richtige Sperrebene für Sie auswählt. In einigen Situationen wollen Sie SQL Server jedoch explizit mitteilen, dass eine Zeilensperrung oder eine Tabellensperrung gesetzt werden soll. SQL Server beinhaltet verschiedene Sperrparameter, die die Strukturierung der Sperre beeinflussen. Sie platzieren einen solchen Parameter wie folgt in Ihrem SQL-Code:

```
UPDATE Products WITH (ROWLOCK)
SET Quantity = Quantity - 1
WHERE Name = 'Dog'
```

Schreibsperrungen im Vergleich mit Lesesperrungen

Sperrmanager benutzen zwei Sperrtypen: Schreibsperrungen und Lesesperrungen. Ein Sperrmanager setzt Schreibsperrungen (auch als *exklusive Sperrungen* bezeichnet), um nicht festgeschriebene Änderungen einer Transaktion zu isolieren. Der Sperrmanager setzt Lesesperrungen (auch als *gemeinsam genutzte Sperrungen* bezeichnet) für Datenelemente, wenn sie gelesen werden.

Eine Schreibsperrung steht im Konflikt mit anderen Schreibsperrungen und mit Lesesperrungen. Eine Transaktion, die eine Schreibsperrung hält, blockiert alle anderen Transaktionen, so dass diese keine Daten in das gesperrte Datenelement schreiben oder daraus lesen können. Das Datenelement bleibt gesperrt, bis die Transaktion festgeschrieben oder rückgängig gemacht wird. Das ist sinn-

voll, weil das System nicht gespeicherte Änderungen aus Gründen der Datenkonsistenz isolieren muss. Diese Isolation hat jedoch einen Preis. Das Blockieren verringert die Parallelität und den Durchsatz des Systems.

Lesesperren geraten nicht in Konflikt mit Schreibsperrern. Viele Transaktionen können dasselbe Datenelement gleichzeitig mit Lesesperren belegen. Eine Transaktion kann jedoch ein Datenelement, das bereits mit Lesesperren belegt ist, nicht mit einer zusätzlichen Schreibsperre belegen. Auf diese Weise wird gewährleistet, dass eine Transaktion ein Datenelement nicht überschreiben kann, während eine andere Transaktion das Element ausliest. Tabelle 8.1 fasst zusammen, wie der Sperrmanager Sperranforderungen bearbeitet, wenn Datenelemente bereits mit Sperrungen belegt sind.

	Keine Sperre	Lesesperre	Schreibsperre
Lesesperranfrage	Sperre wird zugelassen	Sperre wird zugelassen	Anfrage wird blockiert
Schreibsperranfrage	Sperre wird zugelassen	Anfrage wird blockiert	Anfrage wird blockiert

Tabelle 8.1: Wie der Sperrmanager Sperranforderungen verwaltet.

Isolationsebenen

Es ist nicht schwierig zu erkennen, wie ein Sperrmanager Schreibsperrern verwendet. Wenn eine Transaktion eine Änderung vornimmt, wird das betreffende Datenelement exklusiv für alle anderen Transaktionen gesperrt. Das System weiß, dass diese exklusive Sperre bis zum Abschluss der Transaktion aufrechterhalten werden muss. Das Timing des Sperrmanagers in Hinblick auf die Anforderung und Freigabe von Lesesperren ist jedoch nicht so offensichtlich. Wie lange soll eine Lesesperre vom System aufrechterhalten werden, wenn eine Transaktion ein Datenelement liest? Soll die Lesesperre für die gesamte Dauer der Transaktion aufrechterhalten werden, oder nur für die Dauer der Leseoperation?

Sie können festlegen, wie lange eine Lesesperre aufrechterhalten wird, indem Sie die *Isolationsebene* der Transaktion konfigurieren. Während längere Lesesperren die Datenkonsistenz und Transaktionsisolation zuverlässig gewährleisten, wirken sie sich gleichzeitig negativ auf die Parallelität und den Durchsatz aus. In manchen Situationen kann die Freigabe von Lesesperren vor dem Abschluss einer Transaktion eine wichtige Optimierung darstellen. Hier die vier von SQL Server unterstützten Isolationsebenen:

- **Read Uncommitted** Die Transaktion kann alle Datenelemente auslesen. Dies geschieht unabhängig davon, ob für das Datenelement bereits Schreibsperrern gesetzt sind. Das Lesen von Daten mit gesetzter Schreibsperre wird auch als *Dirty Read* bezeichnet. Die Transaktion setzt außerdem keine Lesesperren (und gibt keine frei). Diese Isolationsebene bietet den schnellsten Zugriff, ist jedoch äußerst anfällig für inkonsistente Lesevorgänge.
- **Read Committed** Es können ausschließlich Datenelemente ausgelesen werden, die festgeschrieben wurden. Eine auf dieser Isolationsebene ausgeführte Transaktion muss warten, bis eine von einer anderen Transaktion verwendete Schreibsperre freigegeben wird, bevor sie auf das Datenelement zugreifen kann. Lesesperren werden zugelassen, während die Transaktion das Datenelement ausliest, aber vor Abschluss der Transaktion freigegeben. Dies ist die Standardisolationsebene von SQL Server und ADO (ActiveX Data Objects), wenn Sie eine Transaktion ohne COM+ ausführen.
- **Repeatable Read** Diese Ebene entspricht mit einer Ausnahme der *Read Committed*-Ebene. Die Ausnahme besteht darin, dass alle Lesesperren bis zum Abschluss der Transaktion auf-

rechterhalten werden. Alle Daten, die von der Transaktion gelesen werden, können später mit demselben Ergebnis noch einmal gelesen werden; das bedeutet, dass alle Daten, die von der Transaktion gelesen werden, im selben Zustand verbleiben, bis die Transaktion festgeschrieben oder rückgängig gemacht wird. Diese Ebene wird nicht vom SQL Server 6.5 unterstützt, der stattdessen die Ebene *Serializable* verwendet.

- **Serializable** Diese Ebene entspricht der *Repeatable Read*-Ebene, bietet jedoch eine zusätzliche Qualifikation: Eine Abfrage, die mehrfach in einer serialisierbaren Transaktion ausgeführt wird, sollte immer dasselbe Resultat erzielen. Phantomdatenelemente können nicht von anderen Transaktionen eingefügt werden, bis die Transaktion abgeschlossen ist. Wenn eine Transaktion, die auf dieser Ebene ausgeführt wird, beispielsweise eine Abfrage initiiert, um die Anzahl der Datensätze in einer Tabelle zu ermitteln, kann keine andere Transaktion einen Datensatz in die Tabelle einfügen, bis die erste Transaktion beendet ist. Dies ist die einzige Standardisolationsebene, die von SQL Server verwendet wird, wenn dieser von einem Objekt in einer COM+-Transaktion aufgerufen wird.

Jede Ebene, mit Ausnahme von *Serializable*, wirkt sich nachteilig auf die Isolation aus, um Parallelität und Durchsatz zu verbessern. Wenn Sie Transaktionen auf der *Serializable*-Isolationsebene ausführen, bevorzugt der Transaktionsmanager Datenkonsistenz und Isolation auf Kosten von Parallelität und Durchsatz. Eine Transaktion ist serialisierbar, wenn sie nicht von anderen Transaktionen beeinflusst wird. Wenn jede Transaktion serialisierbar ist, hat dies denselben Effekt auf eine Applikation, als würde lediglich eine Transaktion ausgeführt. Die Ausführung all Ihrer Operationen auf der Ebene *Serializable* ist für die Datenkonsistenz am besten – aber für die Parallelität am schlechtesten. Später in diesem Kapitel werden Sie Methoden kennen lernen, verschiedene Operationen innerhalb derselben Transaktion mit unterschiedlichen Isolationsebenen auszuführen.

Lokale Transaktionen programmieren

Es ist nicht sehr kompliziert, eine Transaktion für ein einzelnes DBMS zu schreiben. Stellen Sie sich vor, Sie entwickeln eine Applikation für eine SQL Server-Datenbank. Eine Möglichkeit, eine lokale SQL Server-Transaktion zu starten und zu kontrollieren, ist die Verwendung von Methoden, die das ADO-Objekt *Connection* bereitstellt. Betrachten Sie den folgenden Code:

```
Dim conn As Connection
Set conn = New Connection
conn.Open MyConnectionString
conn.IsolationLevel = adXactSerializable
conn.BeginTrans
    ' Die SQL-Anweisungen wurden der Einfachheit halber weggelassen.
    conn.Execute sqlDecrementInventory
    conn.Execute sqlChargeCustomer
    conn.Execute sqlAddOrder
conn.CommitTrans
```

Das ist zwar vielleicht der einfachste Ansatz, aber die Steuerung einer Transaktion über ein ADO-Connection-Objekt führt zu ineffizienten Sperrungen, weil sie mindestens drei Verbindungen zum DBMS brauchen, um eine Transaktion abzuschließen. Sie erhalten die Sperrungen aufrecht, während die Aufrufe zwischen Ihrem ADO-Code in der Zwischenschicht und dem DBMS übertragen werden.

Sie sollten wann immer möglich jede Transaktion innerhalb einer einzigen Verbindung zum DBMS ausführen, insbesondere wenn sich Ihr ADO-Code und das DBMS auf unterschiedlichen Computern befinden.

Es gibt zwei allgemeine Ansätze, um eine lokale Transaktion innerhalb einer einzigen Verbindung zu dem DBMS auszuführen. Sie können einen SQL-Batch absetzen, der mehrere Anweisungen beinhaltet, oder Sie rufen eine gespeicherte Prozedur auf, die eine geeignete SQL-Logik enthält. Bei beiden Ansätzen können Sie die Transaktion mit Transact-SQL programmtechnisch steuern. Sie können beispielsweise den folgenden Code verwenden, um eine Transaktion auszuführen:

```
-- Isolationsebene Serializable verwenden
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
-- Transaktion zum Kauf von fünf Hunden für 100 $ ausführen.
BEGIN TRANSACTION
    -- Produktmenge im Lagerbestand verringern.
    UPDATE Products SET Quantity = Quantity - 5
    WHERE Product = 'Dog'
    -- Benutzerkonto mit dem Preis belasten.
    UPDATE Customers SET AccountBalance = AccountBalance + 100
    WHERE Customer = 'Bob'
    -- Auftragsdatensatz hinzufügen.
    INSERT Orders(Customer, Product, Quantity, Price)
    VALUES('Bob', 'Dog', 5, 100)
COMMIT TRANSACTION
```

Dieses einfache Beispiel enthält keinen Programmcode, der auf Probleme reagiert und einen Rollback ausführt. Es zeigt jedoch, wie mehrere SQL-Anweisungen als eine einzelne Transaktion ausgeführt werden. Wenn Sie diesen Code unter Verwendung eines ADO-Connection- oder Command-Objekts ausführen lassen, verwendet der SQL Server seinen eigenen integrierten Transaktionsmanager, um die ACID-Regeln zu erzwingen. Entweder werden alle oder keine Änderungen gespeichert. Beachten Sie, dass dieser Batch Ihnen auch ermöglicht, eine Transaktion innerhalb einer einzigen Datenübertragung auszuführen und freizugeben.

SQL Server-Transaktionen verwenden die Standardisolationsebene Read Committed. Der Batch aus dem vorherigen Beispiel setzt die Standardisolationsebene auf Serializable. Alle Anweisungen in dieser Transaktion werden auf dieser Ebene ausgeführt. Auch SQL Server 7 erlaubt Ihnen, Parameter zu setzen, die die Isolationsebene für jede einzelne Anweisung festlegen.

Eine Alternative zur Bereitstellung von SQL-Batches in Ihrem Datenzugriffscode sind gespeicherte Prozeduren. Gespeicherte Prozeduren können dieselbe Transact-SQL-Logik enthalten, und sie bringen normalerweise aufgrund des Cachings im Speicher des DBMS die beste Leistung. Wenn Sie eine gespeicherte Prozedur definieren, können Sie Parameter für die Eingabe, die Ein- und Ausgabe, die Ausgabe sowie einen Rückgabewert vorgeben. Wenn Sie eine gespeicherte Prozedur aufrufen, die Ausgabeparameter oder einen Rückgabewert besitzt, sollten Sie sie mit einem ADO-Command-Objekt aufrufen.

Vielfach wurde geklagt, gespeicherte Prozeduren passten nicht in das Design von mehrschichtigen Applikationen. Die Kritiker sagen, die Logik für Auswertungen und Geschäftsregeln sollte statt in Transact-SQL in einer Sprache wie beispielsweise Visual Basic verwaltet werden. Wenn Sie dem zustimmen, geht es Ihnen mehr um die Wartbarkeit des Codes als um die Leistung der Applikation. Gespeicherte Prozeduren sind immer schneller. Wenn Sie Ihre Logik hauptsächlich in Visual Basic schreiben wollen, sollten Sie wirklich darauf achten, wie viele Verbindungen Sie zu dem DBMS herstellen.

Wenn Sie beispielsweise versuchen, zu viel Logik in Visual Basic zu verwalten, führt das zu einem Entwurf, bei dem Sie eine Datenübertragung über das Netzwerk durchführen, um eine Transaktion zu starten, eine weitere Übertragung initiieren, um Daten zur Auswertung zu erhalten, eine dritte, um eine UPDATE-Anweisung auszuführen, und eine vierte, um die Transaktion festzuschreiben. In einer größeren, komplexeren Transaktion kann dieser Ansatz schnell zu 20 bis 30 Übertragungen pro Transaktion führen. Er ist einfach nicht skalierbar. Während die Aufrufe zwischen Ihrem Datenzugriffscode und dem DBMS übertragen werden, werden Sperrungen aufrechterhalten, die den Durchsatz Ihrer Applikation reduzieren und die Skalierbarkeit beeinträchtigen. Die Reduzierung der Verbindungen ist ein wesentlicher Aspekt der Skalierbarkeit.

HINWEIS: Abfangen der Fehler des nativen SQL Server-OLE-DB-Providers.

Datenbankprogrammierer haben häufig ein Problem, wenn sie zum ersten Mal Batches für SQL Server absetzen oder gespeicherte Prozeduren ausführen. Wenn Sie den eigenen OLE-DB-Provider verwenden, übersehen Sie sehr leicht Fehler, die von einem Batch oder einer gespeicherten Prozedur ausgelöst werden. Wenn Sie beispielsweise einen Batch mit mehreren SELECT-, INSERT-, UPDATE- und DELETE-Anweisungen schreiben, gibt die zugrunde liegende Datenzugriffsschicht für jede Anweisung ein separates Ergebnis zurück. Das passiert auch bei allen SQL-Anweisungen, wie beispielsweise Aktionsabfragen, die keine Zeilen zurückgeben. Jedes Ergebnis beinhaltet einen HRESULT, den Wert RowsAffected und möglicherweise eine Datensatzgruppe und eine Fehlermeldung, die der Ausführung der Anweisung zuzuordnen ist. Die Tatsache, dass der eigene OLE-DB-Provider mehrere Ergebnisse zurückgibt, bedeutet, dass Sie jedes Ergebnis verarbeiten müssen, um sicherzustellen, dass Sie die von den später im Batch definierten Anweisungen ausgelösten Fehler verarbeiten. Hier ein einfaches Beispiel dafür, wie Ihr Code aussehen sollte:

```
Dim rs As Recordset
Set rs = conn.Execute sqlMyBigBatch
Do Until rs Is Nothing
    Set rs = rs.NextRecordset ' Alle Ergebnisse verarbeiten.
Loop
```

Programmierung verteilter Transaktionen

Wie Sie gesehen haben, kann ein einzelnes DBMS-System die ACID-Regeln erzwingen, wenn es alle Daten verwahrt, die von einer Transaktion bearbeitet werden. Der SQL Server stellt einen internen Transaktionsmanager bereit, der das Festschreiben und Rückgängigmachen unterstützt. Außerdem garantiert er, dass die festgeschriebenen Daten im Falle eines Systemfehlers wiederhergestellt werden können.

Was passiert jedoch, wenn sich nicht alle an einer Transaktion beteiligten Daten auf demselben Computer oder im selben Datenbankformat befinden? OLTP-Applikationen, die solche verteilten Transaktionen ausführen, sind sehr viel komplizierter zu implementieren. Angenommen, Sie haben die Products-Tabelle in einer SQL Server-Datenbank, die Customers-Tabelle in einer Oracle-Datenbank und die Orders-Tabelle in DB2, wie in Abbildung 8.3 gezeigt. Um eine verteilte Transaktion auszuführen, brauchen Sie Software, die das Festschreiben/Rückgängigmachen sowie die Wiederherstellung über mehrere Datenquellen hinweg koordiniert.

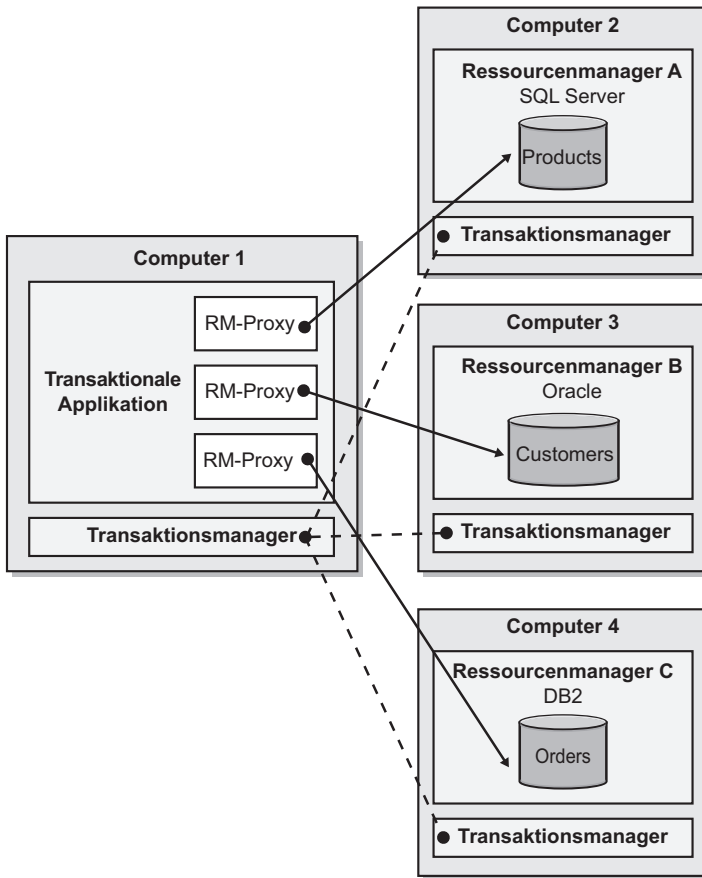


Abbildung 8.3: Eine transaktionale Applikation sowie Transaktionsmanager arbeiten zusammen, um eine verteilte Transaktion zu überwachen und das Zwei-Phasen-Commit-Protokoll über mehrere Ressourcenmanager hinweg auszuführen.

Nur wenige Software-Produkte nutzen eine ähnlich komplexe Architektur, um das Problem der Koordinierung verteilter Transaktionen zu lösen. Zu diesen Produkten zählen CICS von IBM, TUXEDO von BEA Systems und COM+ von Microsoft. Sie alle verwenden eine komplexe Abstraktion, die als *Transaktions-Manager* (TM) bezeichnet wird, sowie ein spezielles Protokoll, das *Two-Phase-Commit* genannt wird.

Ein Großteil der Terminologie verteilter Transaktionen wurde in den frühen Tagen von CICS geprägt. Die Applikation, die die Lese-/Schreiboperationen ausführt und letztlich die Transaktion steuert, wird als *transaktionale Applikation* bezeichnet. Datenquellen werden als *Ressourcen-Manager* (RM) bezeichnet. Jeder *Knoten* (Computer) im System führt entweder eine lokale Sitzung des TM aus oder kommuniziert mit einer Remote-Sitzung eines TM. Eine Applikation, die auf dieser Infrastruktur aufsetzt, wird häufig auch als *Transaktionsmonitor* bezeichnet.

Damit Sie das Ganze besser verstehen, wollen wir betrachten, wie diese Applikationen und Dienste zusammenarbeiten. Eine transaktionale Applikation ruft ihren lokalen TM auf, um mit der Transaktion zu beginnen. Der TM erzeugt eine Transaktion, indem er eine eindeutige ID erzeugt und verschiedene Datenstrukturen initialisiert, um die Lebensdauer der Transaktion zu ver-

walten. Dieser TM führt das Zwei-Phasen-Commit-Protokoll aus und wird auch als koordinierender Transaktionsmanager (Coordinating Transaction Manager – CTM) bezeichnet.

Neben dem Aufruf des TM zum Starten einer verteilten Transaktion richtet eine transaktionale Applikation auch die Verbindungen zu einem oder mehreren RMs ein. Dabei wird einem RM die ID der verteilten Transaktion übergeben und er wird angewiesen, dem CTM die Kontrolle über die Transaktion zu übergeben.

In einigen Situationen kommuniziert ein verbundener RM direkt mit dem CTM. In anderen Situationen wird ein RM mit einem lokalen TM verbunden, einem so genannten *Participating Transaction Manager* (PTM). Während der Zuordnung eines Remote-RM muss der CTM ebenfalls eine Verbindung zum PTM herstellen. Nachdem alle Verbindungen zugeordnet wurden, wird die Kommunikation eingerichtet, wie in Abbildung 8.3 bereits gezeigt wurde.

Die transaktionale Applikation und der CTM arbeiten zusammen, um die ACID-Regeln für die gesamte Transaktion zu erzwingen. Die transaktionale Applikation gibt Befehle an den CTM weiter, und der CTM koordiniert das Festschreiben respektive das Rückgängigmachen für alle Teilnehmer.

Der CTM erzwingt die Einhaltung der ACID-Regeln in einer verteilten Transaktion, indem er das Zwei-Phasen-Commit-Protokoll ausführt. Wenn die transaktionale Applikation anzeigt, dass sie die Transaktion festschreiben möchte, führt der CTM die erste Phase aus, in der die Transaktion vorbereitet wird. In dieser Phase sendet der CTM eine »Bist du vorbereitet?«-Nachricht an jeden Teilnehmer. Wenn ein RM anzeigt, dass seine Änderungen dauerhaft sind und diese auf Anforderung festgeschrieben oder rückgängig gemacht werden können, sendet der RM eine »Ich bin vorbereitet!«-Nachricht. In diesem Fall weiß der CTM, dass die Änderungen festgeschrieben werden können.

Ein RM zeigt mit der »Ich bin vorbereitet!«-Nachricht an, dass sich seine Änderungen in einem dauerhaften Zustand befinden und dass er diese Änderungen auf Weisung entweder festschreiben oder rückgängig machen kann. Sobald jeder Teilnehmer mit einer »Ich bin vorbereitet!«-Nachricht geantwortet hat, weiß der CTM, dass die Änderungen übernommen werden können. Der CTM kann nun Phase 1 abschließen.

Und jetzt folgt, was in einer erfolgreichen Transaktion am Ende von Phase 1 passiert. Nachdem von jedem Teilnehmer eine »Ich bin vorbereitet!«-Nachricht empfangen wurde, schreibt der CTM in sein Protokoll, dass die Transaktion festgeschrieben wurde, und gibt eine Erfolgsmeldung an die transaktionale Applikation zurück. Jetzt kann die transaktionale Applikation andere Arbeiten erledigen und die von der Transaktion gesetzten Sperrungen werden aufrechterhalten.

In Phase 2 sendet der CTM eine »Zeit zum Festschreiben!«-Nachricht an jeden Teilnehmer. Wenn ein RM diese Nachricht erhält, kann er die Sperrungen freigeben und alle Informationen verwerfen, die zum Rückgängigmachen der Transaktion benötigt werden. Nachdem die Änderungen festgeschrieben wurden, sendet jeder RM eine »Ich bin fertig!«-Nachricht an den CTM. Nachdem der CTM diese Nachricht von jedem Teilnehmer empfangen hat, geht er davon aus, dass die Arbeit erfolgreich erledigt wurde und dass er ebenfalls die Informationen verwerfen kann, die zum Rückgängigmachen der Transaktion benötigt werden.

Als Programmierer von COM+-Applikationen müssen Sie sich jedoch nicht mit den internen Details des Zwei-Phasen-Protokolls beschäftigen, da es lediglich ein Implementierungsdetail darstellt. Wenn Sie eine verteilte Transaktion in COM+ festschreiben, können Sie sicher sein, dass das System die Einhaltung der ACID-Regeln erzwingt. Es gibt nichts im COM+-Programmiermodell, das Sie veranlassen könnte, über das Zwei-Phasen-Commit-Protokoll nachzudenken.

Wenn Sie dennoch mehr über dieses Thema erfahren möchten, sollten Sie das Buch *Principles of Transaction Processing* von Bernstein und Newcomer lesen. Dieses Buch behandelt viele Aspek-

te des Zwei-Phasen-Commit-Protokolls (und einige andere interessante Themen), die besonders für Systementwickler und Administratoren wichtig sind. Wenn Sie vorhaben, immer wieder OLTP-Applikationen zu entwickeln, sollten Sie dieses Buch unbedingt lesen.

Der Distributed Transaction Coordinator

Der Transaction Manager von Microsoft trägt die Bezeichnung Distributed Transaction Coordinator (DTC). Dieses Produkt wurde ursprünglich mit SQL Server 6.5 ausgeliefert, wird jedoch mittlerweile auch von anderen Produkten, wie zum Beispiel COM+ und Microsoft Message Queue (MSMQ), verwendet. Der DTC wird unter Microsoft Windows 2000 und Microsoft Windows NT als Systemdienst ausgeführt. Abbildung 8.4 zeigt, dass der DTC als koordinierender TM und außerdem als teilnehmender TM agieren kann.

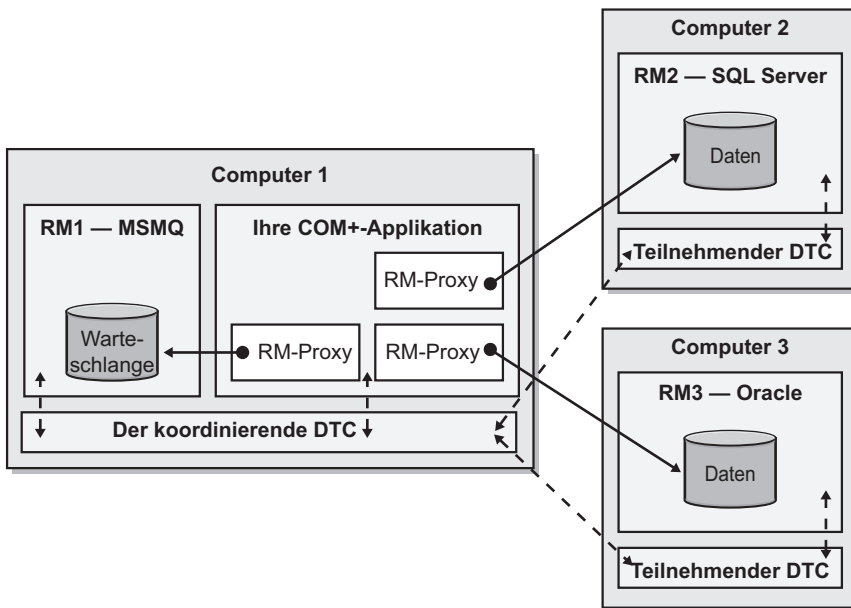


Abbildung 8.4: Der DTC – Microsofts Transaction Manager – wird von Produkten wie COM+, MSMQ und SQL Server verwendet.

Der DTC benutzt ein Protokoll, das *OLE Transactions* genannt wird. Dieses Protokoll definiert mehrere COM-Schnittstellen, über die Applikationen und RMs mit dem DTC kommunizieren können. X/Open ist ebenfalls ein gebräuchlicher Standard, der von Produkten wie Tuxedo, Encina und TOPEND verwendet wird. X/Open standardisiert, wie OLE Transactions die Kommunikation von Applikationen und RMs mit dem Transaction Manager realisiert.

In den frühen Tagen von MTS waren SQL Server und Oracle die beiden einzigen RMs, die eine Kommunikation mit dem DTC erlaubten. Heute gibt es zahlreiche RMs, die mit dem DTC kommunizieren können. Einige RMs sind auf OLE Transactions ausgelegt. Andere, X/Open-basierte RMs benötigen Bridging-Software, um korrekt zu arbeiten. Details über die Interoperabilität zwischen diesen beiden Standards entnehmen Sie bitte dem MTS SDK oder erkundigen Sie sich beim Hersteller Ihres Ressourcenmanagers. Von jetzt an setze ich voraus, dass Sie einen oder mehrere RMs besitzen, die mit COM+ und dem DTC zusammenarbeiten.

Direkte Zusammenarbeit mit dem DTC

Wenn Sie verteilte Transaktionen mit dem DTC ausführen möchten, stehen Ihnen zwei Auswahlmöglichkeiten zur Verfügung. Die einfache Möglichkeit besteht darin, deklarative Transaktionen zu verwenden und die COM+-Laufzeitumgebung alle komplizierten Aufrufe an den DTC ausführen zu lassen. Die zweite (und tapferere) Möglichkeit ist es, eine Applikation zu erstellen, die direkt mit dem DTC zusammenarbeitet. Wenn Sie sich für die zweite Möglichkeit entscheiden, sollten Sie Ihren Code in C oder C++ schreiben.

Jetzt möchte ich kurz abschweifen und beschreiben, was erforderlich ist, wenn Sie direkt Code für den DTC schreiben wollen. Als Visual Basic-Programmierer werden Sie niemals auf dieser Ebene programmieren müssen. Wenn Sie jedoch einen kurzen Blick hinter die Kulissen werfen, erhalten Sie eine Vorstellung davon, welche Aufgaben COM+ in Ihrem Auftrag ausführt, wenn Sie eine deklarative Transaktion ausführen.

Die transaktionale Applikation muss zunächst eine Verbindung zum DTC herstellen, indem sie `DtcGetTransactionManager` aufruft, um einen `ITransactionDispenser`-Verweis auf das DTC-Proxy-Objekt zu erhalten. Die `ITransactionDispenser`-Schnittstelle enthält eine Methode namens `BeginTransaction`. Wenn Sie diese Methode aufrufen, erstellt der DTC ein neues Transaktionsobjekt und gibt einen Verweis darauf an Ihre Applikation zurück. Dieses Transaktionsobjekt verfügt über eine `Commit`-Methode und eine `Abort`-Methode, auf die Sie über die `ITransaction`-Schnittstelle zugreifen. Wenn Sie eine Transaktion erstellen, können Sie die Isolationsebene und ein Zeitüberschreitungsintervall angeben.

Sie stellen anschließend eine Verbindung zu einem oder mehreren Ressourcen-Managern her. Beispielsweise können Sie mit einem OLE DB-Provider eine Verbindung zu einem DBMS, wie SQL Server oder Oracle, einrichten. Beachten Sie dabei, dass der ODBC-Treiber als RM-Proxy agiert. Sobald Sie eine Verbindung zu mehreren RMs hergestellt haben, müssen Sie diese Verbindungen explizit zuordnen, indem Sie OLE DB aufrufen. Nachdem Ihre Verbindungen zugeordnet sind, können Sie SQL-Operationen ausführen, wie zum Beispiel `INSERT`, `UPDATE` und `DELETE`. Alle Les- und Schreibaktivitäten werden für Ihre verteilte Transaktion ausgeführt.

Wenn die transaktionale Applikation `Abort` aufruft, teilt der DTC allen Teilnehmern mit, dass ein Rollback ausgeführt werden muss und alle Sperrungen freigegeben werden müssen. Wenn die transaktionale Applikation ihre Arbeit auf allen RMs vollständig beendet hat, ruft sie `Commit` auf, um den lokalen DTC anzuweisen, das Zwei-Phasen-Commit-Protokoll auszuführen. Der koordinierende DTC ruft parallel dazu alle Teilnehmer auf und fragt diese, ob sie darauf vorbereitet sind, ihre Arbeit festzuschreiben. Die Tatsache, dass diese Aufrufe parallel erfolgen, bedeutet, dass der lokale DTC nicht darauf warten muss, dass der erste Teilnehmer reagiert, bevor er die Anforderung an den nächsten Teilnehmer senden kann. Dieser Ansatz verbessert die Leistung ganz wesentlich.

Nachdem der koordinierende DTC von jedem mit ihm verknüpften Teilnehmer die Nachricht »Ich bin vorbereitet!« erhalten hat, beginnt die zweite Phase. Der lokale DTC vermerkt zuerst in seinem Protokoll, dass die Transaktion festgeschrieben wurde, und gibt eine Erfolgsmeldung an die transaktionale Applikation zurück. Hier endet Phase 1 und Phase 2 beginnt.

In Phase 2 sendet der koordinierende DTC parallel an alle Teilnehmer eine »Zeit zum Festschreiben«-Nachricht. Die Teilnehmer schreiben ihre Änderungen fest, geben ihre Sperrungen frei und senden die Nachricht »Ich bin fertig.« an den koordinierenden DTC. Nachdem dieser alle Nachrichten empfangen hat, kann er die Transaktion vergessen, weil ein Wiederherstellen des ursprünglichen Zustands aufgrund eines Systemfehlers jetzt nicht mehr erforderlich ist.

Wie Sie sehen, ist die Entwicklung einer verteilten Applikation äußerst komplex, und Sie müssen auf sehr viele Details achten. Außerdem ist dieser Ansatz der Entwicklung lokaler Transaktionen

sehr ähnlich, weil dabei ein prozedurales Paradigma angewendet wird. Sie nehmen explizite Aufrufe vor, um die Transaktion zu starten, festzuschreiben und rückgängig zu machen. Glücklicherweise beinhaltet COM+ ein deklaratives Programmiermodell, das alles viel einfacher macht. Beachten Sie jedoch, dass COM+-Transaktionen im Hintergrund genau so ausgeführt werden, wie hier beschrieben. COM+ ist so praktisch, weil es die müßigen Details der Kommunikation mit dem DTC und die Zuordnung von RMs verbirgt.

COM+ und deklarative Transaktionen

COM+ macht es viel einfacher, verteilte Transaktion mit dem DTC auszuführen, indem ein Programmiermodell bereitgestellt wird, das auf deklarativen Transaktionen basiert. Sie sind für das Erstellen von Objekten innerhalb des Gültigkeitsbereichs einer COM+-Transaktion verantwortlich. Wenn Sie solche Objekte erstellen, generiert die COM+-Laufzeitumgebung in Zusammenarbeit mit dem DTC eine verteilte Transaktion. Außerdem ordnet COM+ Ihre RM-Verbindungen im Hintergrund zu.

Nachdem Sie Ihre Objekte in einer Transaktion erstellt haben, verwenden Sie einige Methoden, die von der `ObjectContext`-Schnittstelle oder von der `IContextState`-Schnittstelle angeboten werden. Mit diesen Methoden kontrollieren Sie den Fluss der Transaktion. Wie Sie noch sehen werden, sind Sie dafür verantwortlich, die Transaktionen so schnell wie möglich wieder freizugeben.

Zunächst wollen wir betrachten, wie man ein oder mehrere Objekte in einer COM+-Transaktion erzeugt. Jede konfigurierte Komponente verfügt über ein *Transaction*-Attribut. Abbildung 8.5 zeigt die Registerkarte *Transaktionen* des Eigenschaftendialogfeldes einer Komponente. Sie öffnen das Dialogfeld mit dem administrativen Hilfsmittel *Komponentendienste*. Auf dieser Registerkarte modifizieren Sie das *Transaction*-Attribut für jede Komponente in einer COM+-Applikation. Wenn Sie Visual Basic 5 benutzen, müssen Sie zunächst Ihre Komponenten installieren und das *Transaction*-Attribut anschließend manuell setzen.

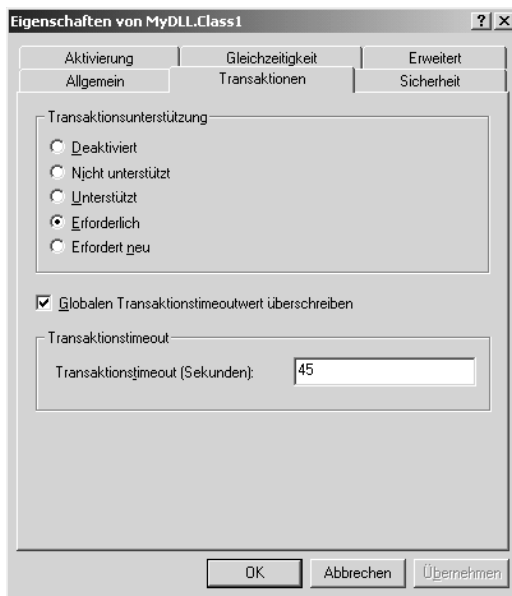


Abbildung 8.5: Wenn die COM+-Laufzeitumgebung ein Objekt aus einer Komponente erstellt, liest sie deren *Transaction*-Attribut aus, um zu ermitteln, ob das Objekt in einer neuen oder in einer bereits bestehenden Transaktion erstellt werden soll.

Visual Basic 6 fügt dem Eigenschaftsblatt aller öffentlichen Klassenmodule innerhalb eines ActiveX-DLL-Projekts eine neue Eigenschaft mit der Bezeichnung *MTSTransactionMode* hinzu. Wenn Sie diese Eigenschaft wie in Abbildung 8.6 dargestellt setzen, übernimmt Visual Basic das entsprechende COM+-*Transaction*-Attribut in die Typbibliothek Ihres Servers. Wenn Sie Ihre DLL in einer COM+-Applikation installieren, werden Ihre Komponenten automatisch mit dem entsprechenden *Transaction*-Attribut konfiguriert.

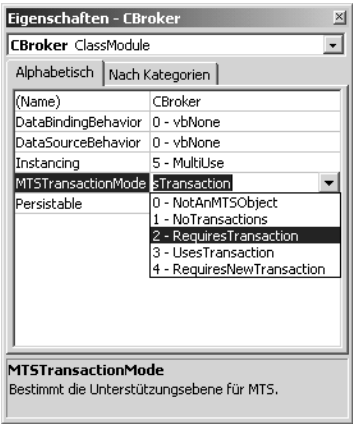


Abbildung 8.6: Wenn Sie Ihre DLL in einer COM+-Applikation installieren, wird jede Komponente mit dem Transaktionsattribut konfiguriert, das dem MTSTransactionMode-Eigenschaftswert entspricht.

Beachten Sie bitte den Unterschied zwischen den MTSTransactionMode-Eigenschaftswerten für ein Visual Basic-Klassenmodul und den *Transaction*-Attributswerten in COM+. Der Unterschied besteht lediglich in den Bezeichnungen, die von COM+ und Visual Basic verwendet werden. Jede der vier MTSTransactionMode-Eigenschaftseinstellungen hat ein entsprechendes *Transaction*-Pendand, wie in Tabelle 8.2 gezeigt.

MTSTransactionMode-Eigenschaftswert	COM+- <i>Transaction</i> -Attribut
RequiresTransaction	Erforderlich
RequiresNewTransaction	Erfordert neu
UsesTransaction	Unterstützt
NoTransactions	Nicht Unterstützt
NotAnMTSObject	Kein entsprechendes <i>Transaction</i> -Attribut
N/A	Deaktiviert

Tabelle 8.2: MTSTransactionMode-Eigenschaftswerte und äquivalente COM+-Einstellungen für das *Transaction*-Attribut.

Objekte in einer COM+-Transaktion erstellen

Wenn die COM+-Laufzeitumgebung ein neues Objekt erzeugt, wertet sie das *Transaction*-Attribut der Komponente aus, um zu überprüfen, ob das neue Objekt in einer Transaktion erzeugt werden soll. Wenn die COM+-Laufzeitumgebung ein Objekt erzeugt, das in einer Transaktion ausgeführt werden muss, ermittelt sie gleichzeitig, ob das Objekt in einer neuen Transaktion oder in der Transaktion des Erzeugers aktiviert werden soll. Im siebten Kapitel wurde beschrieben, wie COM+ unter Verwendung des *Synchronization*-Attributs Objekte mit Aktivitäten verknüpft.

COM+ benutzt das *Transaction*-Attribut auf dieselbe Weise, um Objekte mit Transaktionen zu verknüpfen.

Nachdem ein Objekt in einer Transaktion erzeugt wurde, verbleibt es dort bis zu seiner Zerstörung. Wenn die Transaktion festgeschrieben oder abgebrochen wird, zerstört die COM+-Laufzeitumgebung alle darin enthaltenen Objekte, wozu die Just-in-time-Aktivierung verwendet wird. Das bedeutet, dass ein Visual Basic-Objekt in einer Transaktion zerstört werden muss, nachdem die Transaktion abgeschlossen ist. Wie Sie sehen werden, ist diese Vorgehensweise für die korrekte Semantik einer Transaktion notwendig. Die Objekte und der von ihnen angeforderte Status müssen zerstört werden, damit die Isolationsanforderungen der ACID-Regeln erfüllt sind.

Ein transaktionales Objekt kann von einem anderen Objekt in derselben COM+-Applikation oder von einem externen Client erzeugt werden. Wenn die COM+-Laufzeitumgebung eine Aktivierungsanforderung erhält, um ein neues Objekt zu erzeugen, stellt sie fest, ob der Erzeuger des Objekts in einer bereits existierenden COM+-Transaktion ausgeführt wird. Außerdem wertet sie das *Transaction*-Attribut der Komponente aus. Aus diesen beiden Informationen kann sie ableiten, wie weiter zu verfahren ist.

Einer Komponente kann eine der folgenden Einstellungen für das *Transaction*-Attribut zugewiesen werden:

- **Erforderlich** Das Objekt wird immer in einer Transaktion erstellt. Es wird in der Transaktion seines Erzeugers erstellt, sofern diese vorhanden ist. Wenn der Erzeuger in keiner Transaktion ausgeführt wird, erstellt die COM+-Laufzeitumgebung eine neue Transaktion für das Objekt.
- **Erfordert neu** Die COM+-Laufzeitumgebung erstellt eine neue Transaktion für das Objekt.
- **Unterstützt** Das Objekt wird in der Transaktion seines Erzeugers erstellt, sofern diese vorhanden ist. Wenn der Erzeuger in keiner Transaktion ausgeführt wird, wird das Objekt ohne eine Transaktion generiert.
- **Nicht unterstützt** Das Objekt wird nicht in einer Transaktion erstellt.

Ein externer Client kann eine Transaktion initiieren, indem er ein Objekt aus einer konfigurierten Komponente aktiviert, die mit der Option *Erforderlich* oder mit *Erfordert neu* ausgezeichnet ist. Die COM+-Laufzeitumgebung stellt fest, dass das neue Objekt in einer Transaktion ausgeführt werden muss und dass der Erzeuger nicht in einer Transaktion ausgeführt wird. Die COM+-Laufzeitumgebung generiert deshalb eine neue COM+-Transaktion und anschließend ein neues Objekt innerhalb dieser Transaktion.

Das erste Objekt in einer Transaktion wird als Root-Objekt der Transaktion bezeichnet und spielt eine wichtige Rolle. Das Root-Objekt kann weitere *sekundäre Objekte* in einer Transaktion erzeugen. Wenn Sie eine Methodenimplementierung für ein COM+-Objekt schreiben, um ein weiteres Objekt in der Transaktion erstellen zu lassen, müssen Sie das Objekt zunächst aus Komponenten erstellen, die mit *Erforderlich* oder *Unterstützt* markiert sind.

Beachten Sie einen kleinen Unterschied zwischen *Erforderlich* und *Erfordert neu*. Eine Komponente, die mit *Erforderlich* markiert ist, kann genutzt werden, um sowohl Root-Objekte als auch sekundäre Objekte zu erzeugen. Eine Komponente, die mit *Erfordert neu* markiert ist, kann nur Root-Objekte erzeugen. Wenn Sie mehr über die Rolle des Root-Objekts und der sekundären Objekte gelernt haben, werden Sie den Unterschied verstehen.

Häufig werden COM+-Transaktionen erzeugt, die mehrere Objekte enthalten. Zunächst wollen wir jedoch eine Transaktion betrachten, die nur ein Objekt enthält – das Root-Objekt. Nachdem Sie genau wissen, wie das Root-Objekt funktioniert, werden Sie besser verstehen, was passiert, wenn mehrere Objekte gleichzeitig in einer Transaktion ausgeführt werden.

Eine Geschichte von zwei Transaktionen

Wenn ein Client ein Objekt aus einer Komponente erstellt, die mit *Erforderlich* markiert ist, erstellt die COM+-Laufzeitumgebung eine neue COM+-Transaktion, in der das neue Objekt platziert wird. Gleichzeitig führt die COM+-Laufzeitumgebung einen Aufruf des DTC aus, um eine neue verteilte Transaktion zu erstellen. Sie sollten sich die COM+-Transaktion als *logische Transaktion* und die DTC-Transaktion als *physische Transaktion* vorstellen. Beachten Sie bitte, dass die COM+-Laufzeitumgebung die physische Transaktion nicht erzeugt, wenn sie die logische Transaktion erzeugt.

Die COM+-Laufzeitumgebung erstellt die logische Transaktion, wenn sie das Root-Objekt erzeugt. Sie unterbindet das Generieren der physischen Transaktion, bis das Root-Objekt aktiviert ist. Es entsteht also eine Verzögerung zwischen der Erstellung beider Transaktionen. Die COM+-Laufzeitumgebung verzögert das Erstellen der physischen Transaktion so lange wie möglich. Dies ist eine Optimierung, um die Transaktionszeiten zu verkürzen.

In Kapitel 7 habe ich das Konzept der Just-in-time-Aktivierung vorgestellt, das wichtig ist, um deklarative Transaktionen zu verstehen. Hier noch einmal eine Zusammenfassung: Der Client verwendet nie direkt ein Objekt, das die Just-in-time-Aktivierung unterstützt. Stattdessen greift der Client auf einen vom System bereitgestellten Proxy zu, der es der COM+-Laufzeitumgebung erlaubt, einen speziellen Trick auf der Abfangebene auszuführen. Die COM+-Laufzeitumgebung aktiviert vor jedem Aufruf ein Objekt und deaktiviert es, bevor es die Kontrolle an den Client zurückgibt. Der Client erkennt nicht, was passiert, aber jeder Methodenaufruf wird von einem neuen und anderen Objekt bedient (vorausgesetzt, Ihre transaktionalen Objekte werden nicht in einem Pool bereitgestellt).

Zwischen dem Zeitpunkt, zu dem der Client ein Objekt erzeugt, und dem Zeitpunkt, zu dem der Client den ersten Methodenaufruf absetzt, entsteht eine Verzögerung. In MTS startet die Laufzeitumgebung die DTC-basierte Transaktion erst dann, wenn das erste Root-Objekt aktiviert wurde und der erste Methodenaufruf erfolgt. COM+ hat eine neue Optimierung eingeführt: Die COM+-Laufzeitumgebung startet die DTC-basierte Transaktion, nachdem das Root-Objekt den ersten »interessanten« Aufruf abgesetzt hat. Ein interessanter Aufruf ist einfach beschrieben ein Aufruf, für den die COM+-Laufzeitumgebung die DTC-basierte Transaktion starten muss. Ein Beispiel für einen interessanten Aufruf ist die Einrichtung einer Datenbankverbindung, die automatisch zugeordnet werden muss, oder die die *TransactionID* abfragt. In beiden Fällen muss die COM+-Laufzeitumgebung den DTC aufrufen und eine verteilte Transaktion starten, um ihre Arbeit erledigen zu können.

Wenn die COM+-Laufzeitumgebung erkennt, dass es an der Zeit ist, die verteilte Transaktion zu starten, setzt sie einen *BeginTransaction*-Aufruf an den DTC ab. Die COM+-Laufzeitumgebung erzeugt außerdem die physische Transaktion mit der Isolationsebene *Serializable* und einem Standard-Timeout-Intervall von 60 Sekunden. Das bedeutet, die Transaktion arbeitet auf der höchsten Ebene der Isolation und Sie haben eine Minute Zeit, um Ihre Arbeiten abzuschließen.

Dieses Timeout-Intervall kann sowohl für die ganze Maschine als auch für einzelne Komponenten gesetzt werden. Timeout-Intervalle sollten jedoch kurz gewählt werden, weil sie beim Erkennen und Auflösen von so genannten Deadlocks (gegenseitigen Sperrungen) helfen. (Wir werden später in diesem Kapitel noch über Deadlocks sprechen.)

Wenn die COM+-Laufzeitumgebung den DTC aufruft, um eine physische Transaktion zu erstellen, erzeugt dieser ein Transaktionsobjekt und gibt einen *ITransaction*-Verweis an die COM+-Laufzeitumgebung zurück. Die COM+-Laufzeitumgebung verwahrt diesen Verweis für die gesamte Dauer der logischen MTS-Transaktion. Die *ITransaction*-Schnittstelle erlaubt der COM+-Laufzeitumgebung, eine der Methoden *Commit* oder *Abort* aufzurufen. Wie Sie sehen, bietet die COM+-Laufzeitumgebung die Möglichkeit, das Ergebnis der Transaktion zu kontrollieren.

Als Visual Basic-Programmierer können Sie keinen `ITransaction`-Verweis auf das Transaktionsobjekt zurückgeben lassen und auch nicht `Commit` oder `Abort` aufrufen. Sie fragen sich deshalb bestimmt, wie Sie die Transaktion kontrollieren können. Sie müssen wissen, wann die COM+-Laufzeitumgebung `Commit` und `Abort` aufruft. Wenn dies geschieht, können Sie die Entscheidung der MTS-Laufzeitumgebung beeinflussen.

Alles, was deklarative Transaktionen betrifft, basiert auf einer großen Voraussetzung: Die COM+-Laufzeitumgebung entscheidet über das Festschreiben oder den Abbruch der Transaktion, wenn das Root-Objekt deaktiviert wird. Sie können vor dieser Entscheidung jedoch den Status der Transaktion ändern. Dazu rufen Sie die vom System bereitgestellten Methoden auf, mit deren Hilfe ein Objekt festlegen kann, ob die Transaktion abgeschlossen werden soll oder nicht.

Drei wichtige Flags: Happy, Done und Doomed

Abbildung 8.7 zeigt ein Diagramm mit einem Client, einer Transaktion, einem Root-Objekt und dem Kontext des Root-Objekts. Außerdem zeigt sie einige wichtige interne Flags, die von Skripting-Clients verwaltet werden. Diese Flags sind einfach vom System definierte Variablen, die den Wert `True` oder `False` haben. Aus diesem Grund werden sie auch als *Bits* bezeichnet. Sie sollten verstehen, wie die Flags die Entscheidung der COM+-Laufzeitumgebung über das Festschreiben oder den Abbruch einer Transaktion beeinflussen.

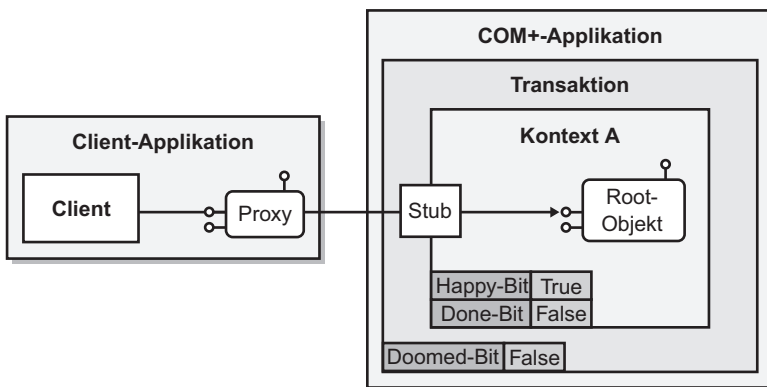


Abbildung 8.7: Das *Happy-Bit* nimmt die Entscheidung des Objekts auf, ob die Transaktion festgeschrieben werden soll. Die COM+-Laufzeitumgebung erkennt, wann sie das Objekt deaktivieren soll, indem sie das *Done-Bit* auswertet.

Die COM+-Laufzeitumgebung erlaubt jedem transaktionalen Objekt, zu wählen, ob die Transaktion festgeschrieben oder rückgängig gemacht werden soll. Diese Entscheidung wird in einem kontextabhängigen Bereich gespeichert, der auch als *Happy-Bit* bezeichnet wird. (Weniger humorvolle Programmierer können es auch als *Konsistenz-Bit* bezeichnen.) Dieses Boolesche Flag teilt der COM+-Laufzeitumgebung mit, ob das Objekt die erledigte Arbeit festschreiben will. Der Standardwert für das *Happy-Bit* ist `True`.

Das andere wichtige kontextabhängige Flag eines Objekts in einer Transaktion ist das *Done-Bit*, mit dessen Hilfe ein Objekt der COM+-Laufzeitumgebung mitteilen kann, dass es seine Arbeit abgeschlossen hat. Wenn das Root-Objekt sein *Done-Bit* auf `True` setzt, löst es seine eigene Deaktivierung aus. Grund dafür ist, die Transaktion so schnell wie möglich freizugeben. Der Standardwert des *Done-Bits* ist `False`.

Es gibt noch ein wichtiges Flag, das *Doomed-Bit*. Dabei handelt es sich um ein einzelnes Flag, das von jedem Objekt und Kontext in der Transaktion genutzt wird. Dieses Flag wird zunächst auf `False` gesetzt, wenn eine COM+-Transaktion erzeugt wird. Es ist so wichtig, weil die COM+-Laufzeitumgebung es nutzt, um zu entscheiden, ob die Transaktion festgeschrieben oder abgebrochen werden soll.

Wann wertet die COM+-Laufzeitumgebung das *Doomed-Bit* aus? Und wie können Sie den Wert des *Doomed-Bits* ändern? Um diese Fragen zu beantworten, müssen Sie wissen, dass das Root-Objekt in jeder COM+-Transaktion eine wichtige Rolle spielt. Die COM+-Laufzeitumgebung wertet das *Doomed-Bit* aus, wenn das Root-Objekt deaktiviert wird. Wird das Root-Objekt während einer aktiven Transaktion deaktiviert, wertet die COM+-Laufzeitumgebung das Bit aus und gibt die Transaktion durch einen Aufruf von `Commit` oder `Abort` frei. Ist das *Doomed-Bit* auf `False` gesetzt, ruft die COM+-Laufzeitumgebung `Commit` auf. Ist das *Doomed-Bit* auf `True` gesetzt, ruft die COM+-Laufzeitumgebung `Abort` auf.

Die Deaktivierung des Root-Objekts sollte die Transaktion immer beenden. Wird Root deaktiviert, wird die Transaktion freigegeben. Solange das Root-Objekt aktiviert bleibt, kann die Transaktion auch weiterhin existieren und all ihre Sperrungen aufrechterhalten.

Jetzt betrachten wir die beiden Flags, die im Kontext jedes transaktionalen Objekts verwaltet werden. Das erste ist das *Happy-Bit* und hat zunächst den Wert `True`. Wird ein in einer COM+-Transaktion ausgeführtes Objekt deaktiviert, wertet die COM+-Laufzeitumgebung ihr *Happy-Bit* aus. Ist es auf `False` gesetzt, setzt die COM+-Laufzeitumgebung das *Doomed-Bit* der Transaktion auf `True`. Ist das *Doomed-Bit* der Transaktion auf `True` gesetzt, kann es nicht mehr geändert werden. Das hat weitreichende Auswirkungen. Wenn das Root-Objekt oder jedes sekundäre Objekt in einer Transaktion in einem *Unhappy-Status* deaktiviert wird, ist die Transaktion zum Scheitern verurteilt.

Wir werden nun einige Szenarien erörtern, die das oben Beschriebene verdeutlichen. Angenommen, ein Client erzeugt ein Objekt aus einer Komponente, deren Transaktionsunterstützungsattribut auf *Erforderlich* gesetzt ist. Der Client führt einen Methodenaufruf durch, um eine Verbindung zu dem DBMS herzustellen. Er fügt anschließend einen Datensatz ein. Daraufhin wird eine COM+-Transaktion erzeugt und die COM+-Laufzeitumgebung wird veranlasst, den DTC aufzurufen, um auch die physische Transaktion zu erzeugen. Was passiert, wenn der Client jetzt den Verweis auf das Root-Objekt einfach freigibt? Wird das Root-Objekt deaktiviert, wertet die COM+-Laufzeitumgebung das *Happy-Bit* aus. Das *Happy-Bit* hat immer noch den Ausgangswert `True`. Die COM+-Laufzeitumgebung ändert deshalb das *Doomed-Bit* nicht. Das *Doomed-Bit* bleibt `False` und die COM+-Laufzeitumgebung ruft `Commit` für die Transaktion auf. Sie können ein einfaches Beispiel ausführen und diese Ergebnisse bestätigen, indem Sie die Transaktionsstatistik im Verwaltungswerkzeug *Komponentendienste* ausführen.

Es ist also ganz einfach: Sie erzeugen ein Objekt in einer Transaktion, rufen eine Methode auf, die irgendeine Arbeit ausführt, und geben das Objekt dann frei. Es sind nur drei Schritte erforderlich, um mit dem DTC eine Transaktion erfolgreich zu starten und zu beenden. Dieses Beispiel zeigt, wie und wann die COM+-Laufzeitumgebung mit dem DTC kommuniziert. Wir mussten keinen Code schreiben, um die Transaktion zu beginnen oder zu beenden, weil die logische COM+-Transaktion alles für Sie erledigt.

Jetzt schreiben wir Code, um die COM+-Laufzeitumgebung zu veranlassen, eine Transaktion rückgängig zu machen. Dazu setzen Sie einfach das *Happy-Bit* des Objekts auf `False`. Eine Möglichkeit dafür ist, `DisableCommit` in einer Methode aufzurufen:

```
GetObjectContext.DisableCommit
```

Wenn der Client mit diesem Code eine Methode für das Root-Objekt aufruft, setzt die COM+-Laufzeitumgebung den Wert des Happy-Bits auf `False`. Wenn der Client die Verbindung freigibt, wird das Root-Objekt deaktiviert. Während der Deaktivierung sieht die COM+-Laufzeitumgebung, dass das Happy-Bit `False` ist und setzt den Wert des Doomed-Bits für die Transaktion auf `True`. Wird das Root-Objekt deaktiviert, ruft die COM+-Laufzeitumgebung `Abort` für die Transaktion auf.

`DisableCommit` wird von einer anderen Methode mit der Bezeichnung `EnableCommit` ergänzt. Diese Methode setzt das Happy-Bit einfach wieder auf `True`. Sie können diese Methoden wiederholt aufrufen. Das Happy-Bit ist einfach nur ein Boolescher Wert. Die vor der Deaktivierung des Objekts zuletzt aufgerufene Methode bestimmt also, wie die COM+-Laufzeitumgebung mit der Transaktion verfährt. Durch den Aufruf einer dieser Methoden legen Sie einfach fest, ob die Transaktion erfolgreich ausgeführt werden soll. Sie können `EnableCommit` und `DisableCommit` beliebig oft aufrufen. Wichtig ist nur der letzte Aufruf vor der Deaktivierung des Objekts.

Die Methoden **SetComplete** und **SetAbort**

Zusätzlich zu `DisableCommit` und `EnableCommit` beinhaltet die `ObjectContext`-Schnittstelle zwei weitere wichtige Methoden, mit denen Sie Ihre Transaktionen kontrollieren: `SetComplete` und `SetAbort`. Wie die beiden anderen Methoden, ändern auch diese Methoden das Happy-Bit, um eine Entscheidung festzulegen. `SetComplete` setzt das Happy-Bit auf `True`, `SetAbort` setzt es auf `False`. Diese Methoden unterscheiden sich jedoch von `DisableCommit` und `EnableCommit`, weil sie das Done-Bit auf `True` setzen. Wie Sie aus Kapitel 7 wissen, hat das Done-Bit einen ganz wesentlichen Einfluss auf den Lebenszyklus eines Objekts. `DisableCommit` und `EnableCommit` ändern das Done-Bit ebenfalls, setzen es aber auf `False`, was in der Regel keine Auswirkung hat, weil das der Standardwert ist.

Nachdem das Root-Objekt die Ausführung eines Methodenaufrufs beendet hat, gibt es die Kontrolle zurück an die Abfangschicht. Damit hat die COM+-Laufzeitumgebung die Gelegenheit, das Done-Bit auszuwerten, bevor sie die Kontrolle an den Client zurückgibt. Ist das Done-Bit auf `True` gesetzt, deaktiviert die COM+-Laufzeitumgebung das Objekt. Das ist äußerst wichtig, weil damit die Freigabe der Transaktion beschleunigt wird. Das Root-Objekt kann also die Freigabe einer Transaktion aktiv unterstützen, indem es `SetComplete` oder `SetAbort` aufruft und dann die Kontrolle an den Client zurückgibt.

Ruft das Root-Objekt einer Transaktion `SetAbort` auf, wird die Transaktion abgebrochen. Die COM+-Laufzeitumgebung deaktiviert das Objekt in einem »Unhappy«-Status, setzt das Doomed-Bit auf `True` und bricht die Transaktion ab. Ruft stattdessen das Root-Objekt einer Transaktion die Methode `SetComplete` auf, deaktiviert die COM+-Laufzeitumgebung das Objekt und versucht, die Transaktion sofort festzuschreiben. Die COM+-Laufzeitumgebung wertet dennoch alle Happy-Bits aus, um zu erkennen, ob sie die Transaktion ebenso wie beim Aufruf von `EnableCommit` und `DisableCommit` festschreiben oder abbrechen soll. `SetComplete` und `SetAbort` zwingen die COM+-Laufzeitumgebung, die Transaktion sehr viel schneller zu beenden, so dass die aufwendigen Sperrungen, die Ihre Transaktion aufrechterhält, früher freigegeben werden. Sie brauchen nicht darauf zu warten, dass der Client das Root-Objekt freigibt.

Es ist wichtig, `SetComplete` und `SetAbort` zum richtigen Zeitpunkt auszuführen. Wenn Sie Ihre Transaktion beeinflussen, indem Sie `DisableCommit` oder `EnableCommit` aufrufen, werden die Transaktion und alle damit verknüpften Sperrungen beibehalten, bis der Client das Root-Objekt freigibt. Ein Aufruf von `SetComplete` oder `SetAbort` ist besser, da das Root-Objekt die COM+-Laufzeitumgebung veranlasst, die Transaktion freizugeben. Wenn Sie in einer OLTP-Umgebung arbeiten, besteht Ihre wichtigste Aufgabe darin, die Parallelität und den Durchsatz zu verbessern, indem Sie die Zeit verkürzen, in der eine Transaktion Datenelemente sperrt.

Sie haben gesehen, dass die COM+-Laufzeitumgebung das Root-Objekt deaktiviert und die Transaktion freigibt, wenn Sie `SetComplete` oder `SetAbort` aufrufen. Das führt zu einer wichtigen Frage: Wie wirkt sich die Deaktivierung des Root-Objekts auf den Client aus? Wenn ein Client eine Methode ausführt, die einen Aufruf von `SetComplete` oder `SetAbort` enthält, wird das Objekt deaktiviert und zerstört. Würde der Client erfahren, dass das Objekt nicht mehr vorhanden ist, hätten Sie ein Problem. Glücklicherweise versteckt das Just-in-time-Aktivierungsschema die Tatsache, dass das Objekt nicht mehr existiert, vor dem Client.

Wenn der Client weiterhin Methoden aufruft, die `SetComplete` oder `SetAbort` verwenden, erzeugt die COM+-Laufzeitumgebung jedes Mal ein neues Root-Objekt und eine neue Transaktion und gibt diese frei. COM+ stellt damit eine Just-in-time-Aktivierung gefolgt von einer schnellstmöglichen Deaktivierung bereit. Dies alles geschieht in dem Zeitfenster zwischen der Vorverarbeitung und Nachbearbeitung des Methodenaufrufs. Eine COM+-Transaktion und die darin enthaltenen Objekte sollten also nur kurz im Universum aufblitzen. In der Welt von OLTP führen kürzere Transaktionen zu einer verbesserten Parallelität und einem besseren Durchsatz.

Die IContextState-Schnittstelle

Bisher habe ich vier Methoden der `ObjectContext`-Schnittstelle beschrieben, die es Ihnen ermöglichen, eine Transaktion zu steuern. COM+ führt eine neue Schnittstelle ein, `IContextState`, die eine ähnliche Funktionalität aufweist. Nachfolgend sind die vier Methoden der Schnittstelle aufgeführt:

- **SetMyTransactionVote** Setzt das Happy-Bit auf `True` oder `False`.
- **GetMyTransactionVote** Liest den Status des Happy-Bits.
- **SetDeactivateOnReturn** Setzt das Done-Bit auf `True` oder `False`.
- **GetDeactivateOnReturn** Liest den Status des Done-Bits.

Beachten Sie, dass diese Methoden es Ihnen ermöglichen, die Happy- und Done-Bits sowie die Methoden der `ObjectContext`-Schnittstelle zu ändern. Beispielsweise können Sie einen Aufruf von `SetComplete` durch einen Aufruf von `SetMyTransactionVote` und `SetDeactivateOnReturn` ersetzen. Solange Sie wissen, wie man das Happy-Bit und das Done-Bit auf `True` setzt, ist es egal, wie Sie es bewerkstelligen.

Was ist aber der Unterschied zwischen der Verwendung der `ObjectContext`-Schnittstelle und der `IContextState`-Schnittstelle bei der Steuerung einer Transaktion? Erstens ermöglicht Ihnen `IContextState`, das Happy- und das Done-Bit unabhängig voneinander zu setzen. Zweitens erlaubt Ihnen `IContextState`, den aktuellen Status dieser Bits zu ermitteln, was mit der `ObjectContext`-Schnittstelle nicht möglich ist. Der dritte Unterschied ist, dass die COM+-Laufzeitumgebung Fehler auslöst, wenn Sie eine `IContextState`-Methode für ein nicht korrekt konfiguriertes Objekt aufrufen. Aufrufe von `GetMyTransactionVote` und `SetMyTransactionVote` schlagen fehl, wenn das Objekt nicht innerhalb einer Transaktion ausgeführt wird. Aufrufe von `SetDeactivateOnReturn` und `GetDeactivateOnReturn` schlagen fehl, wenn das Objekt die Just-in-time-Aktivierung nicht unterstützt.

Wie verwenden Sie also `ObjectContext`, wann `IContextState`, wenn Sie eine transaktionale Komponente entwickeln? Eigentlich ist das egal. Programmierer, die bisher mit MTS gearbeitet haben, bevorzugen möglicherweise `SetComplete` und `SetAbort`, weil ihnen dieser Ansatz am vertrautesten ist. Andere Programmierer wiederum ziehen `IContextState` vor, weil sie den Status des Happy-Bits und des Done-Bits abfragen wollen. C++-Programmierer, die poolfähige Objekte erstellen, die nicht an Transaktionen beteiligt werden, sollten `IContextState` verwenden, wenn sie das Done-Bit manipulieren wollen, ohne gleichzeitig das Happy-Bit zu verändern.

Solange Sie wissen, wie sich die COM++-Laufzeitumgebung verhält und wie Sie das Happy-Bit und das Done-Bit setzen, können Sie jede der beiden Schnittstellen verwenden. Im restlichen Kapitel verwende ich die Methoden der ObjectContext-Schnittstelle. Merken Sie sich einfach, dass Sie genauso gut die Methoden von IContextState verwenden könnten.

Das AutoComplete-Attribut

Achten Sie unbedingt darauf, Transaktionen so schnell wie möglich wieder freizugeben, wenn Sie die Komponente für ein Root-Objekt erzeugen. Die gebräuchlichste Methode ist, das Done-Bit dazu durch einen Aufruf von SetComplete oder SetAbort auf True zu setzen. Diese Aufrufe sind erforderlich, weil das Done-Bit standardmäßig auf False gesetzt wird.

Jede Methode einer konfigurierten Komponente hat ein AutoComplete-Attribut, mit dem Sie den Standardwert des Done-Bits auf True setzen. Sie konfigurieren dieses Attribut mit dem Eigenschaftendialogfeld der Methode im administrativen Hilfsmittel *Komponentendienste*. Selektieren Sie dazu das Kontrollkästchen *Automatisch dieses Objekt deaktivieren, wenn diese Methode zurückkehrt*. Dieses Attribut setzt das Standardverhalten eines Root-Objekts, so dass es automatisch auch dann deaktiviert wird, wenn Sie keinen Aufruf von SetComplete oder SetAbort vornehmen. Dieses Attribut setzt außerdem das Happy-Bit auf False, wenn die Methode einen Fehler an den Aufrufer zurückgibt.

Dieses Funktionsmerkmal ist insbesondere für vorentwickelte Komponenten wichtig, die keine Aufrufe von SetComplete oder SetAbort beinhalten. Vielleicht wollen Sie eine solche Komponente als Root-Objekt einer Transaktion verwenden, ohne dazu Code abändern zu müssen. Sie können alle Methoden für AutoComplete konfigurieren, um sicherzustellen, dass jede Methode eine Transaktion startet und freigibt.

Wenn Sie eine transaktionale Komponente schreiben, steht Ihnen die Verwendung des AutoComplete-Attributs frei. Es bietet eine gewisse Bequemlichkeit, aber Sie haben mehr Kontrolle, wenn Sie das Happy-Bit und das Done-Bit selbst setzen. Darüber hinaus sollten Sie beachten, dass das AutoComplete-Attribut zwar den Wert des Done-Bits auf True setzt, damit aber nicht garantiert ist, dass das Bit True bleibt. Sie müssen dies überwachen, weil Aufrufe von EnableCommit, DisableCommit und SetDeactivateOnReturn das Done-Bit möglicherweise wieder auf False setzen und eine Transaktion damit länger existiert, als Sie das vorgesehen hatten.

Die ObjectControl-Schnittstelle

Die COM++-Laufzeitumgebung verwaltet den Lebenszyklus jedes Objekts, das aus einer konfigurierten Komponente erzeugt wird. Der Lebenszyklus eines Objekts, das die Just-in-time-Aktivierung unterstützt, erfolgt in vier Phasen: Erzeugung, Aktivierung, Deaktivierung und Entsorgung. Wenn Sie COM++-Transaktionen entwickeln, sollten Sie wissen, wann die Übergänge zwischen den einzelnen Phasen erfolgen, um die entsprechenden Maßnahmen zu ergreifen.

Ein Visual Basic-Klassenmodul stellt die Prozedur Initialize bereit. Der Code, den Sie in Initialize schreiben, wird ausgeführt, wenn das Visual Basic-Objekt erzeugt wird. Initialize wird jedoch ebenfalls ausgeführt, bevor das Objekt aktiviert wird. Wenn Sie warten, bis das Objekt aktiviert wurde, ist die von der COM++-Laufzeitumgebung bereitgestellte Kontextinformation umfangreicher und zuverlässiger.

Die Prozedur Terminate in einer Visual Basic-Klasse wird vor dem Zerstören eines Objekts ausgeführt, aber nachdem das Objekt deaktiviert wurde. Terminate ist Initialize dahingehend ähnlich, dass es ausgelöst wird, wenn sich das Objekt nicht mehr in einem aktiven Status befindet. Weder Initialize noch Terminate geben Ihnen die Kontrolle, die Sie brauchen, um den Lebenszyklus in-

nerhalb einer COM+-Transaktion zu verwalten. Glücklicherweise kann Ihnen die COM+-Laufzeitumgebung dabei helfen, indem sie Ihr Objekt unmittelbar nach der Aktivierung benachrichtigt, und dann noch einmal unmittelbar vor der Deaktivierung.

Und so funktioniert das Ganze: Die COM+-Typbibliothek *Services* beinhaltet eine Definition für eine Schnittstelle namens *ObjectControl*. Wenn die COM+-Laufzeitumgebung ein Objekt mit Unterstützung der Just-in-time-Aktivierung erzeugt, ruft sie *QueryInterface* auf, um festzustellen, ob das Objekt diese Schnittstelle unterstützt. Wenn das Objekt *ObjectControl* implementiert, ruft die COM+-Laufzeitumgebung Methoden dieser Schnittstelle auf, um das Objekt in wichtigen Übergangsphasen seines Lebenszyklus zu benachrichtigen. Das bedeutet, Sie sollten *ObjectControl* in jedem Objekt implementieren, von dem Sie glauben, dass es diese Benachrichtigungen erhalten sollte. Die *ObjectControl*-Schnittstelle enthält die drei folgenden Methoden:

- **Activate** Diese Methode wird von der COM+-Laufzeitumgebung aufgerufen, nachdem die Aktivierung erfolgt ist und unmittelbar bevor der erste Methodenaufruf erfolgt.
- **Deactivate** Diese Methode wird von der COM+-Laufzeitumgebung unmittelbar vor der Deaktivierung des Objekts aufgerufen.
- **CanBePooled** Für Komponenten, die das Objekt-Pooling unterstützen, erlaubt diese Methode einem Objekt, der COM+-Laufzeitumgebung mitzuteilen, ob es zerstört oder im Pool bereitgestellt werden soll. Sie wird von der COM+-Laufzeitumgebung nach der Deaktivierung aufgerufen. Wenn Sie die *ObjectControl*-Schnittstelle in eine Visual Basic-Komponente implementieren, ist die Implementierung dieser Methode erforderlich, aber bedeutungslos.

Damit Ihr Objekt diese Benachrichtigungen erhält, sollten Sie die *ObjectControl*-Schnittstelle in die *MultiUse*-Klassenmodulen Ihrer ActiveX-DLLs implementieren. Hier folgt ein Beispiel für ein Visual Basic-Klassenmodul, das diese Schnittstelle implementiert:

```
Implements ObjectControl
```

```
Private Sub ObjectControl_Activate()  
    ' Ihr Code für die Initialisierung nach der Aktivierung  
End Sub
```

```
Private Sub ObjectControl_Deactivate()  
    ' Ihr Code für die Aufräumarbeiten vor der Deaktivierung  
End Sub
```

```
Private Function ObjectControl_CanBePooled() As Boolean  
    ' Objekt-Pooling wird für Visual Basic-Komponenten nicht unterstützt  
    ObjectControl_CanBePooled = False  
End Function
```

Die COM+-Laufzeitumgebung ruft *Activate* unmittelbar vor dem Aufruf der ersten Methode auf. Die *Activate*-Methode wird ausgeführt, während das Objekt vollständig aktiv ist. Wenn Ihre Komponente für die Just-in-time-Aktivierung konfiguriert werden soll, ist es immer am besten, den Code für die Objektinitialisierung in *Activate* statt in *Initialize* unterzubringen. Analog hierzu sollten Sie den Code für Ihre Aufräumarbeiten in *Deactivate* statt in *Terminate* anordnen.

Beachten Sie, dass es nur zwei Komponententypen gibt, die eine Just-in-time-Aktivierung benötigen. Poolfähige Komponenten brauchen die Just-in-time-Aktivierung, um die Objekte effizienter in den Pool zurückzugeben. Komponenten, die in COM+-Transaktionen ausgeführt werden, brauchen die Just-in-Time-Aktivierung, um die Transaktionszeiten kurz zu halten.

Wenn Sie eine Komponente erzeugen, die nicht poolfähig ist und die keine Transaktionen unterstützt, brauchen Sie die Just-in-time-Aktivierung nicht. Es ist eine kleine Optimierung, wenn die Just-in-time-Aktivierung für Komponenten deaktiviert wird, die diese nicht brauchen. Beachten Sie, dass die `ObjectControl`-Schnittstelle für Objekte vorgesehen ist, die die Just-in-time-Aktivierung unterstützen. Wenn Sie die Just-in-time-Aktivierung deaktiviert haben, sollten Sie die Visual Basic-Prozeduren `Initialize` und `Terminate` für die Initialisierung sowie für Aufräumarbeiten verwenden.

Multiobjekt-Transaktionen

Wie Sie wissen, kann ein Root-Objekt sekundäre Objekte innerhalb derselben Transaktion erstellen, indem es `CreateInstance` auf Komponenten ausführt, deren Transaktionsunterstützungsattribut auf *Erforderlich* oder auf *Unterstützt* gesetzt ist. (Ich bin mir sicher, dass Sie es bereits wissen, aber ich möchte hier noch einmal darauf hinweisen, dass der Aufruf von `New` für eine Klasse, die innerhalb derselben DLL kompiliert wird, ein ernsthaftes Problem in COM+ verursacht.) Abbildung 8.8 zeigt ein Root-Objekt und zwei sekundäre Objekte in einer MTS-Transaktion. Jedes Objekt verfügt über einen eigenen Wrapper und damit über ein eigenes Happy- und ein Done-Bit.

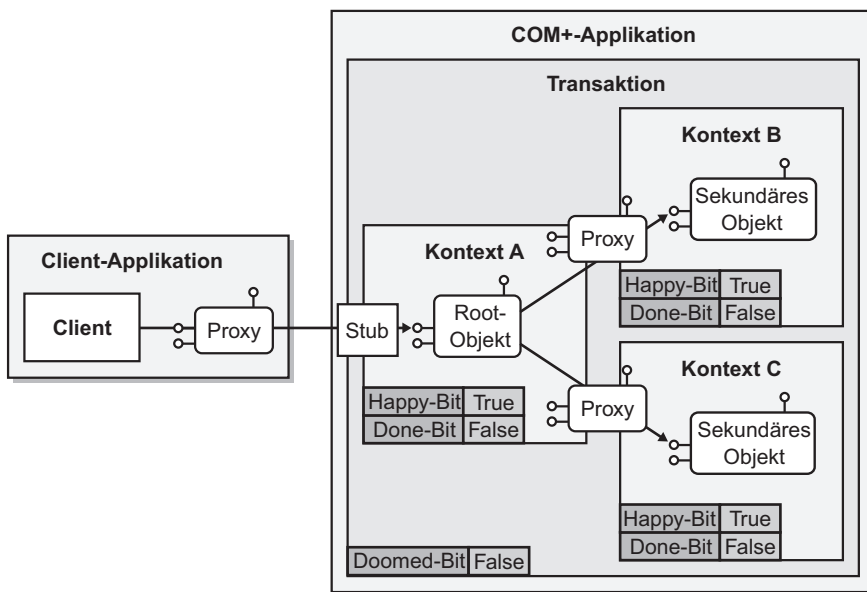


Abbildung 8.8: Multiobjekt-Transaktionen haben immer ein Root-Objekt. Sekundäre Objekte spielen eine andere Rolle als das Root-Objekt.

Eine COM+-Transaktion ist eine demokratische Gemeinschaft, in der jedes Objekt mit einer Stimme darüber entscheidet, ob die Transaktion erfolgreich ausgeführt werden soll. Ein sekundäres Objekt folgt denselben Regeln wie ein Root-Objekt. Wenn ein sekundäres Objekt deaktiviert wird, wertet die COM+-Laufzeitumgebung sein Happy-Bit aus. Wenn das Bit auf `False` gesetzt ist, setzt die COM+-Laufzeitumgebung das `Doomed-Bit` der Transaktion auf `True`. In diesem Fall wird die Transaktion abgebrochen. Alle anderen Objekte können das `Doomed-Flag` nicht zurück auf `True` setzen. Jedes Objekt in einer Transaktion kann somit verhindern, dass die Transaktion festgeschrieben wird.

Sie werden häufig Projekte entwickeln, in denen das Root-Objekt sekundäre Objekte erstellt. Nachdem ein sekundäres Objekt erstellt wurde, kann das Root-Objekt dessen Methoden aufrufen, um die Transaktion zu beenden. Wenn Sie vorhaben, Multiobjekt-Transaktionen zu entwickeln, sollten Sie verstehen, welche Vorgänge innerhalb einer Transaktion ausgeführt werden, so dass Sie die Kommunikation zwischen den darin enthaltenen Objekten koordinieren können. Insbesondere sollten Sie die Verantwortlichkeiten sekundärer Objekte sowie des Root-Objekts kennen. Sekundäre Objekte spielen eine völlig andere Rolle als das Root-Objekt.

Lassen Sie uns einen Blick auf die Situation werfen, in der ein Root-Objekt mehrere Methoden eines sekundären Objekts erfolgreich aufruft. Solange das sekundäre Objekt sein Done-Bit nicht auf `True` setzt, bleibt es aktiviert, bis es vom Root-Objekt freigegeben wird. Das Root-Objekt kann mehrere erfolgreiche Aufrufe an das sekundäre Objekt ausführen und anschließend `SetComplete` aufrufen, um die Transaktion festzuschreiben. Wenn das Root-Objekt `SetComplete` aufruft, werden beide Objekte deaktiviert. Das sekundäre Objekt wird jedoch zuerst deaktiviert, gefolgt vom Root-Objekt. Solange kein Objekt sein Happy-Flag auf `False` setzt, indem es `SetAbort` oder `DisableCommit` aufruft, wird die Transaktion festgeschrieben.

Wenn ein sekundäres Objekt nicht explizit angibt, welches Ergebnis es von der Transaktion erwartet, erteilt es der COM+-Laufzeitumgebung passiv die Zustimmung, die Transaktion festzuschreiben, weil sein Happy-Bit standardmäßig auf `True` gesetzt ist. Das bedeutet, dass Ihnen der Aufruf von `SetComplete` oder `SetAbort` in einem sekundären Objekt freisteht. Diese Vorgehensweise unterscheidet sich wesentlich von der für ein Root-Objekt, wo ein Aufruf von `SetComplete` oder `SetAbort` unverzichtbar ist, um die Transaktion freizugeben.

Was geschieht jedoch, wenn das sekundäre Objekt die Transaktion abbrechen möchte? Wenn das sekundäre Objekt `DisableCommit` aufruft, setzt es sein Happy-Bit auf `False`. Jetzt liegt dem Root-Objekt eine komplizierte Situation vor. Wenn das sekundäre Objekt in einem »Unhappy«-Zustand deaktiviert wird, bringt es die Transaktion zum Scheitern. Jetzt hat das Root-Objekt zwei Möglichkeiten: Erstens kann es versuchen, eine andere Methode für das sekundäre Objekt aufzurufen und es davon zu überzeugen, dass es sein Happy-Bit auf `True` setzen sollte. Falls das nicht möglich ist, kann sich das Root-Objekt geschlagen geben. In diesem Fall sollte es `SetAbort` aufrufen und dem Client einen Fehler zurückgeben.

Wirklich wichtig ist, dass das Root-Objekt immer wissen sollte, wann ein sekundäres Objekt »unhappy« ist. Wenn das Root-Objekt `SetComplete` in einer Transaktion aufruft, in der ein oder mehrere sekundäre Objekte »unhappy« sind, erkennt die COM+-Laufzeitumgebung den Konflikt. Das Root-Objekt versucht dann, eine Transaktion festzuschreiben, die abgebrochen werden muss. Die COM+-Laufzeitumgebung bewältigt diese Situation, indem sie alle Objekte deaktiviert und dem Client einen `mtsErrCtxAborted`-Fehler zurückmeldet.

Statt `DisableCommit` aufzurufen, kann ein sekundäres Objekt den Abbruch erzwingen, indem es `SetAbort` aufruft. Dies führt zu einem interessanten Effekt, weil das sekundäre Objekt deaktiviert wird und die Transaktion scheitern lässt, bevor die Kontrolle an das Root-Objekt zurückgeht. Jetzt wird das Root-Objekt in einer gescheiterten Transaktion ausgeführt. Versucht das Root-Objekt, in einer gescheiterten Transaktion ein weiteres Objekt zu aktivieren, erhält es einen `mtsErrCtxAborting`-Fehler. Wenn das Root-Objekt einfach die Kontrolle an seinen Aufrufer zurückgibt, ohne `SetAbort` aufzurufen, verbleibt es in einem unvollständigen Zustand. Weitere Methodenaufrufe für das Root-Objekt resultieren dann wahrscheinlich in `mtsErrCtxAborted`- und `mtsErrCtxAborting`-Fehlern, die von der COM+-Laufzeit ausgelöst werden.

Diese Probleme lassen lediglich einen Schluss zu: Wenn ein sekundäres Objekt eine Transaktion abbrechen will, indem es `DisableCommit` oder `SetAbort` aufruft, sollte das Root-Objekt ebenfalls `SetAbort` aufrufen und alle Versuche, die Transaktion erfolgreich zu beenden, abbrechen. Diese Lö-

sung hat jedoch ebenfalls einen Haken, weil das Root-Objekt das Doomed-Flag nicht auswerten kann. Es kann die COM+-Laufzeitumgebung auch nicht fragen, ob ein sekundäres Objekt »unhappy« ist. Ihre sekundären Objekte sollten deshalb einem Root-Objekt aktiv mitteilen, wann sie `DisableCommit` oder `SetAbort` aufrufen. Sie können den Rückgabewert oder die Ausgabeparameter in den Methoden Ihrer sekundären Objekte verwenden, um anzuzeigen, ob sie »unhappy« sind, oder Sie können Fehler vom sekundären Objekt auslösen lassen, die an das Root-Objekt zurückgeleitet werden.

Wenn ein sekundäres Objekt beispielsweise entscheidet, die gesamte Transaktion in `Method1` rückgängig zu machen, könnten Sie die folgende Aufruffolge verwenden:

```
GetObjectContext.DisableCommit
Dim ErrorCode As Long, Description As String
ErrorCode = myErrorEnum1 ' Im Format (vbObjectError + 512)
Description = "Die angeforderte Menge steht nicht zur Verfügung."
Err.Raise ErrorCode, , Description
```

Wenn Sie diese Regel befolgen, kann eine Methodenimplementierung im Root-Objekt auf die Fehler reagieren, die von jedem sekundären Objekt ausgelöst werden, nachdem dessen Happy-Bit auf `False` gesetzt wurde. Das bedeutet, dass eine Fehlerbehandlungsroutine im Root-Objekt `SetAbort` aufrufen und einen eigenen Fehler auslösen sollte, um die Fehlermeldung des sekundären Objekts an den Client zurückzuleiten. Wenn das Root-Objekt die Methoden des sekundären Objekts aufruft, ohne eine Fehlermeldung zu erhalten, kann es davon ausgehen, dass alles in Ordnung ist und `SetComplete` aufgerufen werden kann.

Das folgende Beispiel zeigt eine typische Methode im Root-Objekt. Beachten Sie, dass alle Ausföhrungspfade einen Aufruf von `SetComplete` oder `SetAbort` nach sich ziehen.

```
Sub RootMethod1()
    On Error GoTo MyHandler
    Dim Secondary1 As CSecondary
    Set Secondary1 = CreateObject("MyDll.CSecondary")
    Secondary1.Method1
    Secondary1.Method2
    ' Transaktion festschreiben, wenn alle Aufrufe erfolgreich sind.
    GetObjectContext.SetComplete
Exit Sub
MyHandler:
    ' Transaktion rückgängig machen und Fehler zurückmelden.
    GetObjectContext.SetAbort
    ' Fehlerinformationen an den Basis-Client zurückleiten.
    Err.Raise Err.Number, , Err.Description
End Sub
```

Dieser Code zeigt natürlich nur eine von vielen möglichen Lösungen. Wenn Sie sich für eine andere Lösung entscheiden, sollten Sie darauf achten, die Kommunikation zwischen den sekundären Objekten und dem Root-Objekt korrekt zu koordinieren. Rufen Sie im Root-Objekt immer `SetComplete` oder `SetAbort` auf, aber nie `SetComplete`, wenn ein sekundäres Objekt »unhappy« ist.

Beachten Sie außerdem einige weitere wichtige Aspekte: Erstens sollten Sie wissen, dass es einen Fehler in der ersten Version von COM+ in Windows 2000 gibt, der verhindert, dass sekundäre Objekte Fehlerbeschreibungen an das Root-Objekt zurückgeben, nachdem `SetAbort` aufgerufen wurde. (Weitere Informationen finden Sie im Hinweis »Der `SetAbort`-Fehler«.) Das bedeutet, Sie sollten `DisableCommit` möglicherweise `SetAbort` vorziehen. Beachten Sie außerdem, dass ein Aufruf von `DisableCommit` in einem sekundären Objekt nicht unbedingt zum Scheitern der Transaktion föhrt.

HINWEIS: Der SetAbort-Fehler

Leider gibt es in der ersten Version von COM+ einen Fehler, der auftreten kann, wenn ein sekundäres Objekt `SetAbort` aufruft und einen Fehler an das Root-Objekt zurückmeldet. Dieser Fehler befindet sich in dem einfachen Proxy, den COM+ erstellt, um das Root-Objekt mit einem sekundären Objekt im selben STA zu verbinden. Der Fehler ist spezifisch für Visual Basic-Komponenten.

Der Fehler überschreibt alle benutzerdefinierten Fehlerbeschreibungen, die Sie von dem sekundären Objekt zur Root zurückmelden, und ersetzt sie durch die wenig aussagekräftige Meldung *Methode '~' von Objekt '~' ist fehlgeschlagen*. Eine Lösung dafür ist, den Aufruf von `SetAbort` oder `SetDeactivateOnReturn(True)` in sekundären Objekten zu vermeiden, die Fehler auslösen. Um eine Transaktion rückgängig zu machen, sollte ein sekundäres Objekt stattdessen `DisableCommit` oder `SetMyTransactionVote(txAbort)` aufrufen. Solange das sekundäre Objekt das Done-Bit nicht setzt, kann es einen Fehler mit zuverlässiger Beschreibung an das Root-Objekt zurückmelden und damit immer noch Einfluss darauf nehmen, ob die Transaktion rückgängig gemacht werden soll.

Dieser Fehler hat zweierlei Konsequenzen: Erstens gibt es in einem großen Teil des für MTS geschriebenen Codes sekundäre Objekte, die `SetAbort` aufrufen. Dieser Fehler existiert in MTS nicht, deshalb haben sich viele Entwickler angewöhnt, `SetAbort` aufzurufen und Fehler auszulösen, sobald ein Problem auftritt. Möglicherweise müssen Sie Ihren MTS-Code also abändern, wenn Sie ihn nach COM+ portieren wollen.

Das zweite Problem ist, dass eine Komponente entweder als Root-Objekt oder als sekundäres Objekt geschrieben werden muss. Das Root-Objekt sollte immer `SetComplete` oder `SetAbort` aufrufen. Ein sekundäres Objekt dagegen sollte niemals `SetAbort` aufrufen. Der Fehler verhindert, dass Sie eine Komponente erstellen, die sowohl als Root-Objekt als auch als sekundäres Objekt verwendet werden kann.

Dieser Fehler wird im ersten Service-Pack für COM+ korrigiert. Nachdem er aus COM+ entfernt wurde, können Sie Ihren MTS-Code einfacher portieren und eine Komponente schreiben, die sowohl als Root-Objekt als auch als sekundäres Objekt verwendet werden kann.

Wenn ein sekundäres Objekt `DisableCommit` aufruft und die Kontrolle an das Root-Objekt zurückgibt, zeigt es damit an, dass es seine Arbeit im gegenwärtigen Zustand nicht festschreiben kann. `DisableCommit` deaktiviert jedoch das sekundäre Objekt nicht, nachdem es die Kontrolle an das Root-Objekt zurückgegeben hat. Diese Vorgehensweise unterscheidet sich von einem Aufruf von `SetAbort`, bei dem die COM+-Laufzeitumgebung das sekundäre Objekt deaktiviert, bevor das Root-Objekt die Kontrolle zurückerhält. Ein Aufruf von `SetAbort` verurteilt die Transaktion zum Scheitern. Wenn ein sekundäres Objekt `DisableCommit` aufruft, sagt es auf diese Weise: »Ich bin im Moment nicht zufrieden mit meiner Arbeit. Vielleicht ruft das Root-Objekt aber eine andere Methode auf, die meine Meinung ändert.«

Wie Sie sich leicht vorstellen können, erfordert der Einsatz von `DisableCommit` ein weitaus komplizierteres Kommunikationsprotokoll zwischen den Objekten innerhalb einer Transaktion. Wenn ein sekundäres Objekt `DisableCommit` aufruft, kann das Root-Objekt versuchen, das sekundäre Objekt umzustimmen, indem es zusätzliche Methoden ausführt. Das Root-Objekt muss jedoch letztendlich `SetComplete` oder `SetAbort` aufrufen. Es muss deshalb eine Möglichkeit finden, das sekundäre Objekt zufrieden zu stimmen, oder den Abbruch der Transaktion einleiten.

Sie sollten einen Aufruf von `DisableCommit` im Root-Objekt vermeiden, da Sie die Kontrolle sicherlich nicht an den Client zurückgeben möchten, solange noch eine Transaktion aussteht. Lassen Sie den Client niemals bestimmen, wann die Sperrungen freigegeben werden. Dies kann zu unnötig langen Blockierungen von Datenelementen führen.

Datenbankverbindungen und automatische Verknüpfung

Nachdem Sie nun die Grundlagen des Festschreibens von COM+-Transaktionen und den Roll-back-Mechanismus kennen gelernt haben, sollten Sie erfahren, wie Ihre Datenbankverbindungen verknüpft werden. Wenn Sie eine Verbindung mit einem RM einrichten, muss die Verbindung mit Hilfe des DTC zugeordnet werden, wie in Abbildung 8.9 gezeigt. Das bedingt die Einrichtung einer Kommunikationsleitung zwischen dem RM und einer oder mehreren Sitzungen des DTC. Wie Sie in diesem Kapitel bereits erfahren haben, werden diese Kommunikationsleitungen genutzt, um das Zwei-Phasen-Commit-Protokoll auszuführen.

Das Verknüpfen einer Verbindung in einer COM+-Applikation ist sehr einfach. Sie müssen lediglich zwei Regeln befolgen: Sie müssen die Verbindung zunächst ausgehend von einem Objekt, das in einer Transaktion ausgeführt wird, einrichten. Sie müssen zweitens sicherstellen, dass Sie einen RM verwenden, der mit dem DTC zusammenarbeitet, und dass Sie einen RM-Proxy benutzen, der das automatische Verknüpfen unterstützt. SQL Server ist zum Beispiel ein RM, der mit dem DTC zusammenarbeitet. Der native OLE-DB-Provider für SQL Server ist ein RM-Proxy, der das automatische Verknüpfen unterstützt.

Wenn Sie eine typische Verbindung ausgehend von einem Objekt in einer Transaktion mit ADO einrichten, bemerkt der RM-Proxy, dass das automatische Verknüpfen benötigt wird. Er führt dann alle erforderlichen Aufrufe für die Verknüpfung im Hintergrund durch. Sie öffnen eine Datenbankverbindung in der gewohnten Weise, und Ihre Schreib- und Leseoperationen werden automatisch zu einem Bestandteil einer verteilten Transaktion. Es könnte nicht einfacher sein.

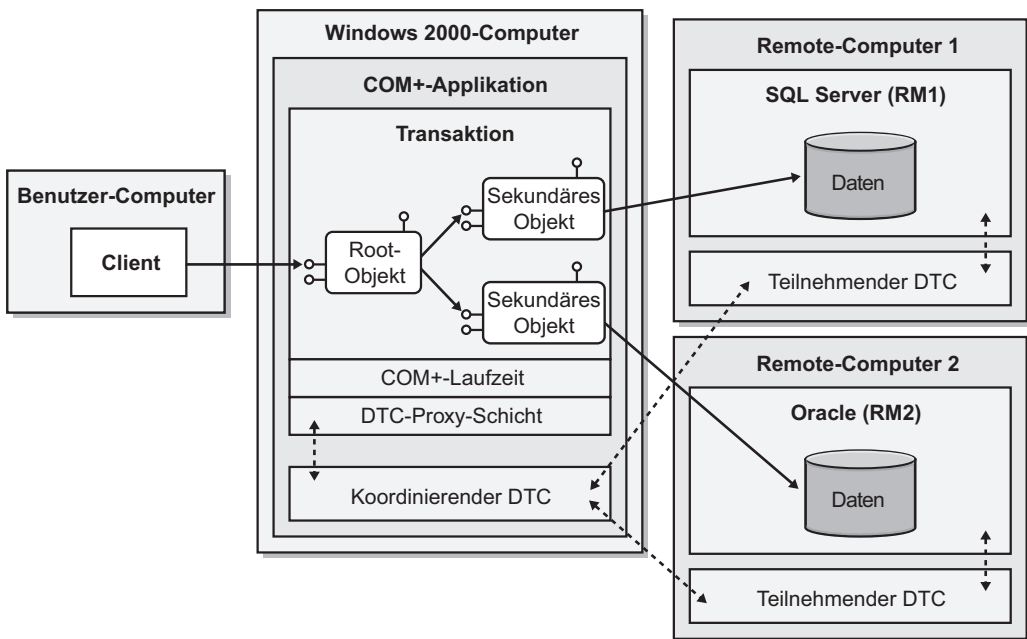


Abbildung 8.9: Wenn Sie eine Verbindung zu einem RM, wie zum Beispiel SQL Server oder Oracle, von einem Objekt innerhalb einer COM+-Transaktion herstellen lassen, verknüpfen die COM+-Laufzeit und der RM-Proxy die Verbindung automatisch mit dem DTC.

Betrachten wir ein typisches Beispiel. Wenn Sie mit SQL Server 7 arbeiten und über ADO eine Verbindung zu dem nativen OLE-DB-Provider herstellen, können Sie den folgenden Code verwenden:

```
Dim conn As ADODB.Connection, sConnectionString As String
Set conn = New ADODB.Connection
conn.Open sConnectionString
```

Dies ist derselbe Code, den Sie auch schreiben würden, um eine ADO-Verbindung einzurichten. Die COM+-Laufzeitumgebung arbeitet mit dem nativen OLE-DB-Provider zusammen, um alle ADO-Verbindungen, die aus einer COM+-Transaktion heraus eingerichtet wurden, automatisch zuzuordnen. Der RM-Proxy kommuniziert mit dem DBMS, um einen Kommunikationskanal mit dem koordinierenden DTC einzurichten, wie in Abbildung 8.9 gezeigt. Sie richten einfach Ihre Verbindungen ein und greifen auf die Daten zu. Alle Ihre Änderungen werden für eine einzelne verteilte Transaktion ausgeführt, die von der COM+-Laufzeit gesteuert wird.

In Kapitel 7 haben Sie die Grundlagen des Datenbankverbindungs-Poolings kennen gelernt. Neben dem Standard-Pooling für Datenbankverbindungen erlaubt COM+ auch das Pooling verknüpfter Verbindungen. Wie Sie gesehen haben, ist eine Datenübertragung zu dem DBMS erforderlich, wenn eine Verbindung eingerichtet werden muss. Werden analog Verbindungen in einer verteilten Transaktion verknüpft, sind noch weitere Übertragungen zu dem DBMS erforderlich. Das Pooling zugeordneter Verbindungen ist eine weitere Optimierung, um die Anzahl der Übertragungen über das Netzwerk zu reduzieren.

Wenn Sie, ausgehend von einem Objekt, das in einer Transaktion ausgeführt wird, eine verknüpfte Verbindung schließen, gibt COM+ die Verbindung an einen speziellen Pool zurück. Größtenteils enthält dieser Pool nur eine einzige physische Verbindung. Fordert ein anderes Objekt in derselben Transaktion eine Verbindung unter Verwendung derselben Verbindungszeichenkette an, verwendet COM+ einfach diese verknüpfte Verbindung erneut. Dies bedeutet, dass mehrere Objekte in einer Transaktion dieselbe verknüpfte Verbindung nutzen können. Dieses Schema verhindert redundante Datenübertragungen zum selben DBMS, die andernfalls für die Verknüpfung mit derselben Transaktion erforderlich wären. Die COM+-Laufzeit kümmert sich um die Aufräumarbeiten für die verknüpfte Verbindung, wenn die Transaktion freigegeben wird.

COM+-Transaktionen und gespeicherte Prozeduren mit Transaktionen

Wenn Programmierer gespeicherte Prozeduren von einem Objekt in einer COM+-Transaktion aufrufen wollen, fragen sie sich häufig, was passiert, wenn die gespeicherte Prozedur Aufrufe enthält, die eine lokale Transaktion steuern, beispielsweise `BEGIN TRAN`, `COMMIT TRAN` oder `ROLLBACK TRAN`. Wie beeinflussen diese Aufrufe das Verhalten des Codes, wenn sie bereits in einer verteilten Transaktion ausgeführt werden? Die Antwort ist einfacher, als Sie vielleicht denken. Aufrufe von `BEGIN TRAN` und `COMMIT TRAN` werden ignoriert, während Aufrufe von `ROLLBACK TRAN` berücksichtigt werden.

Ein Aufruf von `BEGIN TRAN` wird ignoriert, weil die Transaktion bereits gestartet wurde. Darüber hinaus unterstützen weder DTC noch SQL Server das Konzept verschachtelter Transaktionen. Ein Aufruf von `BEGIN TRAN` muss ignoriert werden, weil nicht gestartet werden kann, was bereits gestartet wurde.

Außerdem sollten Sie beachten, dass die gespeicherte Prozedur eine verteilte Transaktion nicht festschreiben kann, wenn sie diese nicht gestartet hat. Die gespeicherte Prozedur ist mit einem sekundären Objekt vergleichbar. Sie kann verlangen, dass die Transaktion abgebrochen wird, aber

sie kann nicht endgültig entscheiden, dass die Transaktion festgeschrieben werden soll. Das bedeutet, dass auch Aufrufe von `COMMIT TRAN` ignoriert werden.

Wenn eine gespeicherte Prozedur `ROLLBACK TRAN` aufruft, wird die Transaktion verworfen. In dieser Hinsicht verhält sich eine gespeicherte Prozedur, die `ROLLBACK TRAN` aufruft, genau wie ein sekundäres Objekt, das `SetAbort` oder `DisableCommit` aufruft. Ein Root-Objekt bekommt Probleme, wenn es `SetComplete` aufruft, nachdem eine gespeicherte Prozedur die Transaktion rückgängig gemacht hat. Es ist also wichtig, dass eine gespeicherte Prozedur einen Fehler auslöst, der darauf hinweist, dass sie die Transaktion rückgängig machen will. Wenn Ihre gespeicherte Prozedur solche Fehler auslöst, können Sie die Transaktion innerhalb des Root-Objekts korrekt rückgängig machen.

Das kurze, aber glückliche Leben eines transaktionalen Objekts

Dieses Kapitel dreht sich immer wieder um ein Thema: Wenn man eine bessere Parallelität und einen höheren Durchsatz erzielen will, muss man Sperrungen so schnell wie möglich freigeben. Wenn Sie transaktionale Objekte verwenden, werden die von Ihnen angeforderten Sperrungen aufrechterhalten, bis Sie das Root-Objekt deaktivieren. Sie sollten das Root-Objekt so schnell wie möglich deaktivieren, indem Sie eine Methode wie `SetComplete` oder `SetAbort` aufrufen. Dieser Aspekt des Programmiermodells kann jedoch auf den ersten Blick seltsam erscheinen. COM+ muss während der Aufräumarbeiten alle der Transaktion zugeordneten Objekte zerstören. Das Root-Objekt und alle sekundären Objekte existieren nur während der Dauer eines einzigen Methodenaufrufs.

Dieses Programmiermodell unterscheidet sich wesentlich von dem der klassischen objektorientierten Programmierung (OOP) und COM, und man muss dafür ein wenig umdenken. Die OOP- und COM-Paradigmen sind nicht für Situationen geeignet, in denen der Status nach jedem Abschluss eines Methodenaufrufs verworfen wird. Objektorientierte Clients setzen voraus, dass sie einen Verweis auf ein langlebiges Objekt erhalten können. Wenn ein Client einen Status innerhalb eines Objekts ändert, nehmen die objektorientierten Programmierer an, dass das Objekt diese Änderungen über mehrere Methodenaufrufe hinweg behält. Bei transaktionalen Objekten in COM+ ist das jedoch nicht der Fall. Die Objekte müssen während der Aufräumarbeiten zerstört werden, und mit ihnen ihr Status.

Diese transparente Zerstörung transaktionaler Objekte wird häufig auch als *statuslose Programmierung* bezeichnet. In der kurzen Geschichte der MTS- und COM+-Programmierung konnte nie eindeutig definiert werden, warum die statusarme Programmierung ein wesentlicher Aspekt des Programmiermodells ist. Einige Bücher und Artikel behaupteten, dass die statusarme Programmierung den Speicherbedarf auf dem Computer verringere, auf dem die COM+-Applikation ausgeführt wird. Es wurde behauptet, das Zerstören von Objekten und der geringe Speicherbedarf führe zu einer besseren Skalierbarkeit, da die Ressourcen effizienter genutzt werden könnten. Dieses Argument ist aber sowohl verwirrend als auch falsch.

Als Visual Basic-Programmierer setzen Sie das Done-Bit nur auf `True`, um die Semantik einer Transaktion sicherzustellen. Der Gedanke dabei ist, dass ein transaktionales Objekt ein Datenelement nur während der RM eine Sperre hält, in einem konsistenten Status sieht. Verwahrt ein COM+-Objekt eine Kopie dieses Datenelements, nachdem die Sperre freigegeben wurde, könnte eine andere Transaktion die ursprünglichen Daten im RM modifizieren. Die ursprünglichen Daten und die Kopie könnten sich somit voneinander unterscheiden und die ACID-Regeln der

Transaktion verletzen. Die Zerstörung von Objekten an den Transaktionsgrenzen bedeutet also, dass alle Kopien eines Datenelements sofort verworfen werden müssen, wenn der RM seine Sperrungen freigibt. Deshalb fordert COM+, dass alle transaktionalen Objekte zerstört werden müssen, wenn eine Transaktion freigegeben wird.

Wenn Sie ein Objekt besitzen, das nicht an einer Transaktion beteiligt ist und keine Sperrungen verlangt, kann der Objektstatus über Methodenaufrufe hinweg beibehalten werden. Wenn Sie das Transaktionsunterstützungsattribut einer Komponente auf *Nicht unterstützt* setzen und weder `SetComplete` noch `SetAbort` aufrufen, können Sie »statusbehaftete« Objekte erstellen.

Ein eher eigenartiger Aspekt bei der Programmierung deklarativer Transaktionen ist, dass ein Objekt zerstört wird, bevor seine Transaktion freigegeben wird. Das bedeutet, ein Objekt kennt nie wirklich das Ergebnis der Transaktion, in der es ausgeführt wurde. Selbst das Root-Objekt wird zerstört, bevor die COM+-Laufzeitumgebung das Zwei-Phasen-Commit-Protokoll ausführt. Das kann beim Entwurf transaktionaler Applikationen zu interessanten Konstellationen führen.

Angenommen, Sie haben ein Root-Objekt und ein sekundäres Objekt. Wird das Root-Objekt aktiviert, erzeugt es das sekundäre Objekt und ruft eine Methode auf, die bestimmte Arbeiten verrichtet. Angenommen, beide Objekte sind »happy«. Wenn das Root-Objekt `SetComplete` aufruft und zurückkehrt, deaktiviert die COM+-Laufzeitumgebung beide Objekte und führt das Zwei-Phasen-Commit-Protokoll aus.

Was passiert, wenn die COM+-Laufzeitumgebung in Phase 1 einen Fehler erkennt? Sie kann keinen Fehler zum Root-Objekt zurückmelden, weil das Root-Objekt bereits zerstört wurde. Sie kann höchstens einen `mtsErrCtxAborted`-Fehler an den Client zurückgeben. Wenn es sich bei Ihrem Client jedoch um eine formularbasierte Visual Basic-Applikation oder eine ASP-Seite handelt, ist es nicht besonders elegant, ihr einen Fehlercode zu senden, der in der COM+-Typbibliothek *Services* definiert ist. Häufig ist es ganz praktisch, Ihrem Entwurf eine zusätzliche, nicht transaktionale Komponente hinzuzufügen, um diesen Fehler beheben zu können. Diese neue Komponente spielt die Rolle eines *Transaktionsbeobachters*.

Abbildung 8.10 zeigt zwei Clients, die Transaktionen ausführen. Client 1 ist direkt mit dem Root-Objekt verbunden, während Client 2 über ein nichttransaktionales Objekt indirekt mit dem Root-Objekt verbunden ist. Einer der größten Vorteile dieses letzteren Entwurfs ist, dass der Client auf diese Weise vor `mtsErrCtxAborted`-Fehlern abgeschirmt wird, die in Phase 1 des Zwei-Phasen-Commit-Protokolls auftreten können. Der Client ruft eine Methode für das nicht transaktionale Objekt auf, das nicht transaktionale Objekt ruft eine Methode für das Root-Objekt auf, um eine Transaktion auszuführen. Das nicht transaktionale Objekt beobachtet das Ergebnis der Transaktion und kann feststellen, ob diese festgeschrieben oder verworfen wurde.

Ein nicht transaktionales Objekt kann auch den clientspezifischen Status über mehrere Methodenaufrufe hinweg beibehalten. Das kann in bestimmten Entwürfen von Vorteil sein, weil bei einem statusbehafteten Objekt die Notwendigkeit entfällt, Parameter zu übergeben und Objekte neu zu initialisieren.

Sie sollten natürlich nicht immer statusbehaftete Objekte verwenden. In vielen Situationen (insbesondere bei webbasierten Applikationen) leben Ihre Objekte für die Dauer einer einzigen Client-Anforderung, egal ob sie transaktional sind oder nicht. Dennoch stellen statusbehaftete Objekte für bestimmte Entwürfe die beste Lösung dar. Wenn Ihnen eine Situation vorliegt, wo es sinnvoll scheint, ein statusbehaftetes Objekt über mehrere Aufrufe hinweg beizubehalten, sollten Sie darauf achten, dass das Objekt nicht transaktional sein darf, und dass Sie das Done-Bit nicht auf `True` setzen dürfen – und bedenken Sie, dass es keinen Grund gibt, `SetComplete` für ein Visual Basic-Objekt aufzurufen, das nicht innerhalb einer Transaktion ausgeführt wird.

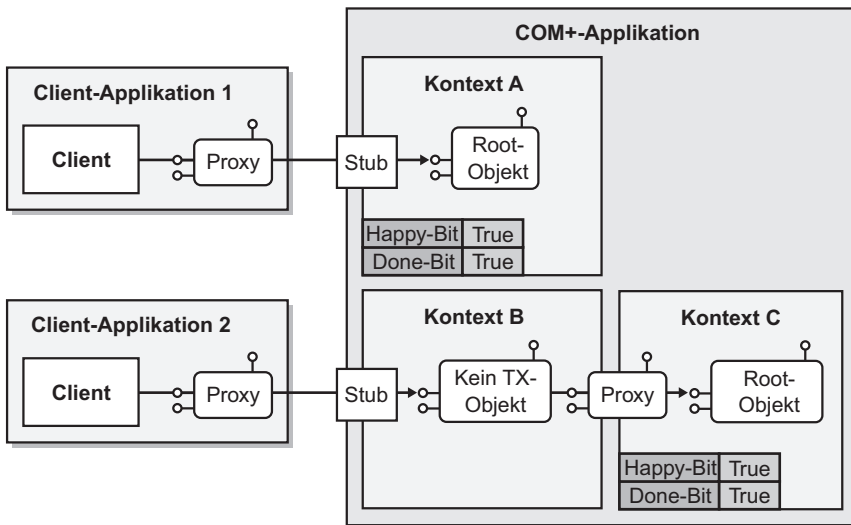


Abbildung 8.10: Es kann sehr vorteilhaft sein, ein nicht transaktionales Objekt zwischen Root-Objekt und Client einzuführen.

Transaktionale Entwürfe

Ich möchte in diesem Kapitel noch auf einige weitere Aspekte hinweisen. Ich habe bereits erklärt, wie man lokale Transaktionen sowie verteilte Transaktionen mit COM+ und DTC programmiert. Jetzt wollen wir diese beiden Ansätze vergleichen, so dass Sie sich gezielt zwischen ihnen entscheiden können. Außerdem möchte ich Ihnen einige weitere Möglichkeiten vorstellen, Ihre Geschäftslogik vom Datenzugriffscode zu trennen. Wie Sie sehen werden, bietet jeder Ansatz seine Vorteile in Hinblick auf die Leistung und die Wartbarkeit einer Applikation. Ich schließe mit einer Beschreibung, wie Sie Deadlocks vermeiden und eine Transaktion optimieren.

COM+-Transaktionen im Vergleich mit lokalen Transaktionen

Die Entwickler von MTS und COM+ haben viel Arbeit in die Infrastruktur zur Ausführung verteilter Transaktionen gesteckt. Die Dokumentation von Microsoft sagt deshalb auch viel mehr zu COM+-Transaktionen als zu lokalen Transaktionen. Beide Ansätze sind jedoch sinnvoll. Viele DBM-Systeme und Mainframe-Applikationen verfügen über eigene TMs, die die ACID-Regeln selbst erzwingen, ohne den DTC dazu zu brauchen. Sie sollten alle diese wichtigen Aspekte verstehen, so dass Sie für Ihre Projekte die richtige Wahl treffen können.

Die Ausführung einer logischen Transaktion innerhalb einer einzigen Verbindung führt häufig zu einer besseren Leistung als eine COM+-Transaktion. Der Grund hierfür besteht darin, dass für eine COM+-Transaktion zusätzliche Aufrufe zwischen der COM+-Applikation, dem TCT und dem RM erforderlich sind, um die Verbindungen zuzuordnen und das Zwei-Phasen-Commit-Protokoll auszuführen. Man kann jedoch schlecht sagen, um wie viel sich die Leistung steigert, weil das von Ihrer Hardware und Ihrer Netzwerkkonfiguration abhängig ist. Möglicherweise tritt eine ganz wesentliche Leistungssteigerung auf, möglicherweise ist sie aber auch kaum wahrnehmbar. Die einzige Methode, das wirklich festzustellen, ist die Verwendung von Benchmark-Tests.

Wenn Ihre Daten über mehrere Computer verteilt oder in unterschiedlichen Datenbankformaten abgelegt sind, haben Sie keine andere Wahl, als COM+-Transaktionen und den DTC zu verwenden. Lokale Transaktionen sind dafür nicht geeignet. Aber auch wenn Sie momentan noch nicht mit mehreren DBM-Systemen zu tun haben, könnten Sie COM+-Transaktionen einsetzen, um den geplanten Einsatz anderer DBM-Systeme in Ihrer Applikation nicht zu beeinträchtigen.

Ein weiterer Grund, sich für COM+-Transaktionen anstelle von lokalen Transaktionen zu entscheiden, ist, dass COM+-Transaktionen den Entwurf Ihrer Applikation einfacher machen und den Codieraufwand reduzieren. Bei einer Komponente, die als sekundäres Objekt verwendet wird, müssen Sie überhaupt keinen Code schreiben. Sie konfigurieren die Komponente einfach so, dass sie Transaktionen unterstützt, und die gesamte Lese/Schreib-Aktivität wird für die vom Root-Objekt gestartete verteilte Transaktion ausgeführt. Wenn Sie eine Komponente anlegen, die die Rolle des Root-Objekts übernimmt, müssen Sie normalerweise Code einfügen – vielleicht Aufrufe von `SetComplete` oder `SetAbort` –, aber das ist einfacher als in einem prozeduralen Paradigma, wo der Programmierer dafür verantwortlich ist, Transaktionen explizit zu starten, festzuschreiben oder rückgängig zu machen.

Sie müssen also zwischen der Leistung und Skalierbarkeit einerseits und der Flexibilität und einfachen Nutzung andererseits entscheiden. Brauchen Sie maximale Geschwindigkeit? Dann sollten Sie sicherstellen, dass Ihre gesamten Daten in einem DBMS abgelegt werden, und Sie sollten lokale Transaktionen verwenden. Wollen Sie verteilte Transaktionen ausführen? Dann verwenden Sie COM+-Transaktionen. Schätzen Sie es, wie deklarative Transaktionen Ihnen das Leben leichter machen? Dann verwenden Sie COM+-Transaktionen.

Wenn Sie sich für COM+-Transaktionen entschieden haben, können Sie die zusätzlich erforderlichen Verbindungen immer durch schnellere Hardware kompensieren. Und es gibt viele Möglichkeiten, die Ausführung verteilter Transaktionen mit COM+ und DTC zu optimieren. Das Windows DNA Performance Kit, das Teil des Microsoft Platform SDK ist, ist ein ausgezeichnetes Werkzeug für das Tuning von COM+-Transaktionen. (Es ist nicht in der Standardinstallation enthalten, Sie müssen es also innerhalb einer benutzerdefinierten Installation einrichten.) Das Windows DNA Performance Kit bietet Anleitungen zum Applikationsentwurf und zum Tuning Ihrer Hardware, die die Leistung ganz wesentlich verbessern. Außerdem werden diese Empfehlungen mit zahlreichen Benchmark-Tests belegt.

Hier nun einige wenige Anleitungen, die Sie im Windows DNA Performance Kit finden:

- Legen Sie das DTC-Transaktionsprotokoll an einer Speicherposition ab, die schnellstmöglichen Zugriff bietet. Die Standardposition des Protokolls ist nicht so schnell, wie sie sein könnte.
- Legen Sie keine Transaktionen über Prozessgrenzen hinweg an.
- Wenn sich alle Ihre Daten in einem einzigen DBMS befinden, beispielsweise in SQL Server, können Sie die Leistung verbessern, indem Sie den Windows 2000-Server neu konfigurieren, auf dem Sie Ihre COM+-Applikation ausführen, damit der DTC auf dem Computer eingesetzt wird, auf dem auch SQL Server läuft.

Kombinieren Sie COM+-Transaktionen nicht mit lokalen Transaktionen

Sie sollten vermeiden, innerhalb einer einzigen Abfrage die Ausführung einer COM+-Transaktion mit einer lokalen Transaktion zu kombinieren. Was passiert beispielsweise, wenn ein Root-Objekt ein nicht transaktionales Objekt erzeugt und dann beide Objekte eine Verbindung zum selben DBMS einrichten? Wenn das nicht transaktionale Objekt eine lokale Transaktion startet, kann das sehr leicht zu Schwierigkeiten führen. Das Problem dabei ist, dass das DBMS zwei un-

terschiedliche Transaktionen sieht und annimmt, die verteilte Transaktion gehöre zu einem Client und die lokale Transaktion zu einem anderen.

Warum ist es ein Problem, wenn eine einzelne Abfrage eines Clients in zwei unterschiedlichen Transaktionen resultiert? Das größte Problem ist, dass sich die Sperrungen der beiden Transaktionen gegenseitig blockieren. Wenn das Root-Objekt eine `SELECT`-Anweisung ausführt, wodurch eine Tabellensperre erzeugt wird, könnte die lokale Transaktion einen Versuch blockieren, auf die gesperrte Tabelle zuzugreifen. Leider wird die Tabellensperre aufrechterhalten, bis alle Objekte deaktiviert wurden. Wie Sie sehen, führt das zum Deadlock (zur gegenseitigen Sperrung). Sie sollten nie auf eine Client-Anforderung antworten, indem Sie gleichzeitig eine verteilte Transaktion und eine lokale Transaktion für dieselbe Datenbank ausführen.

Trennen von Geschäftslogik und Datenzugriffscode

Sie müssen bereits früh in der Entwurfsphase erkennen, wie wichtig es ist, die Geschäftslogik von dem Datenzugriffscode zu trennen. In großen Projekten sollten Sie die gesamte Geschäftslogik in einer Klassengruppe und Ihren gesamten Datenzugriffscode in einer anderen Klassengruppe ablegen. In kleineren Applikationen könnte eine formale Abtrennung einen unverhältnismäßigen Aufwand darstellen.

Das transaktionale Programmiermodell von COM+ zwingt Sie nicht, den einen oder den anderen Ansatz zu verfolgen. Beispielsweise könnten Sie eine Transaktion entwerfen, wobei die Geschäftslogik in einer Komponente und der Datenzugriffscode in einer anderen Komponente abgelegt wird. Eine Komponente mit der Geschäftslogik kann als Root-Objekt verwendet werden, indem ein komplexer Arbeitsflusserzeuger generiert wird. Diese Geschäftskomponente kann COM+-Transaktionen ausführen, indem sie speziell darauf abgestimmte Datenzugriffskomponenten verwendet, die so konfiguriert werden, dass sie in der Transaktion ihres Erstellers ausgeführt werden. Diese formale Abtrennung von Geschäftslogik und Datenzugriffscode führt zu einer besseren Wartbarkeit, weil Ihre Geschäftslogik nicht mit ADO-Aufrufen durchsetzt ist. Außerdem ist es einfacher, Ihre Applikationen über unterschiedliche DBM-Systeme zu portieren.

Sie könnten für eine kleinere Applikation die Entscheidung treffen, dass eine formelle Abtrennung der Geschäftslogik und des Datenzugriffscode nicht wichtig ist. Da Sie bereits den Code der Präsentationsschicht von der nicht sichtbaren Logik getrennt haben, glauben Sie nicht, dass eine weitere Unterteilung Ihrer nicht sichtbaren Logik sinnvoll ist. Sie können Ihr einfaches Design deshalb aufrechterhalten und Ihren Code schneller in Betrieb nehmen. Wenn Sie davon ausgehen können, dass Sie für das Projekt nur ein spezifisches DBMS benötigen, müssen Sie sich nicht mit der Portierung auf verschiedene DBM-Systeme beschäftigen.

Egal, was Sie für die Abtrennung der Geschäftslogik von dem Datenzugriffscode verwenden, Sie sollten dabei nie die Leistung sowie die Notwendigkeit, möglichst wenige Verbindungen zur Datenbank aufzubauen, vergessen. Angenommen, Sie schreiben eine Methode, um eine bestellte Menge eines bestimmten Produkts aus dem Lagerbestand abzugeben. Intuitiv würden Sie den Code so schreiben, dass diese Aufgabe in zwei einzelnen Datenbankoperationen ausgeführt wird. Zuerst führen Sie eine `SELECT`-Anweisung aus, um eine ADO-Datensatzgruppe für die `Products`-Tabelle zu öffnen. Damit ist es einer Visual Basic-Bibliothek in der mittleren Schicht möglich, festzustellen, ob der aktuelle Lagerbestand ausreicht, um eine Bestellung zu erfüllen. Ist die Bestellmenge größer als der Lagerbestand, soll das Objekt die Transaktion rückgängig machen und einen Fehler an den Aufrufer zurückmelden. Ist die Bestellmenge kleiner oder gleich dem Lagerbestand, kann die Komponente ihre Aufgabe erfüllen, indem sie eine `UPDATE`-Anweisung für die `Products`-Tabelle ausführt.

Der Vorteil bei diesem Ansatz ist, dass Sie Ihre Auswertungslogik innerhalb einer Visual Basic-Komponente unterbringen können. Der Nachteil ist, dass Sie zwei Datenübertragungen zum DBMS brauchen, um eine Operation auszuführen, die auch innerhalb einer einzigen Übertragung möglich wäre. Sie müssen also zwischen Leistung und Wartbarkeit abwägen. Wenn Sie die beste Leistung brauchen, sollten Sie einen SQL-Batch schreiben oder eine gespeicherte Prozedur ausführen.

Wenn Sie Logik in SQL schreiben wollen, müssen Sie sich zwischen SQL-Batches und gespeicherten Prozeduren entscheiden. Viele Visual Basic-Programmierer bevorzugen gespeicherte Prozeduren, weil sie etwas schneller sind und die Visual Basic-IDE es Ihnen relativ schwer macht, SQL-Batches in Visual Basic-Quellcode zu verwalten und zu bearbeiten. Wenn Sie gespeicherte Prozeduren verwenden, müssen Sie nur den ADO-Code schreiben, um diese aufzurufen.

Wenn Sie Auswertungslogik in Visual Basic schreiben, können Sie darüber hinaus innerhalb einer einzigen Übertragung zum DBMS alle Daten abrufen, die Sie für Ihre Auswertung und die Berechtigungsüberprüfung mit Visual Basic in der mittleren Schicht benötigen. Wenn Sie in der Auswertung Fehler erkennen, können Sie die Transaktion rückgängig machen und einen Fehler an den Aufrufer zurückmelden. Tritt bei der Auswertung kein Fehler auf, richten Sie eine zweite Verbindung zu dem DBMS ein, um die Transaktion festzuschreiben. Die Abwägung bei diesem Ansatz ist, dass Sie möglicherweise zwischen den beiden Verbindungen Sperrungen aufrechterhalten müssen.

Visual Basic-Klassen sollten nicht Tabellen zugeordnet werden

Programmierer, die nach der reinen Objektorientierung suchen, vergessen häufig die Anforderungen eines OLTP-Systems. Das wichtigste bei einem skalierbaren Ansatz ist, sich eine »Ein- und Ausgabe«-Denkweise anzugewöhnen. Der »Für-jede-Klasse-eine-Tabelle«-Ansatz ist in der Regel nicht ausreichend, weil er Leistung und kurze Sperrungen für eine bessere Wartbarkeit opfert.

Wenn Sie beispielsweise einzelne Klassen für die *Products*-Tabelle, die *Customers*-Tabelle und die *Orders*-Tabelle einführen, richtet jede Klasse möglicherweise unabhängig von den anderen eine eigene Verbindung ein und setzt SQL-Anweisungen ab. Das bedeutet mindestens drei Verbindungen zu der Datenbank. Wenn jede Klasse mehrere eigene Verbindungen einrichtet, wird dieser Ansatz sehr schnell sehr aufwendig.

Sie können die Anzahl der Übertragungen über das Netzwerk reduzieren, indem Sie ein Modell entwickeln, das eine einzige Visual Basic-Klasse für jede Datenbankverbindung verwendet. Eine solche Klasse wird häufig auch als *Sitzungskomponente* bezeichnet. Eine Sitzungskomponente hat die Aufgabe, alle Ihre Operationen innerhalb möglichst weniger Verbindungen auszuführen. Im Allgemeinen stellt die Sitzungskomponente für jede Transaktion oder jeden Befehl, die bzw. der auf dem DBMS ausgeführt werden soll, eine Methode bereit.

Der Vorteil beim Entwurf für einzelne Sitzungen ist, dass damit die beste Leistung und Skalierbarkeit erzielt wird. Leider gibt es auch einige Nachteile. Sie müssen mehr Logik und Auswertungen in SQL schreiben, und weniger in Visual Basic. Damit wird es schwieriger, Ihre Applikationen von einem DBMS auf einen anderen zu portieren. Außerdem müssen Sie Fehler im SQL-Code auslösen und in Visual Basic-Code auffangen. Vielleicht sind Sie auch dazu gezwungen, so lange auf Ihren UML-Code-Editor zu verzichten, bis dieser ein Kontrollkästchen zum Aktivieren der Option »Hören Sie auf, wie ein objektorientierter Mensch aus den 80er Jahren zu denken. Denken Sie stattdessen wie ein effizientes OLTP-Design-Werkzeug« bereithält.

Deadlocks

Die Entwicklung einer OLTP-Applikation erfordert in der Regel den Ausgleich zwischen zwei gegensätzlichen Kräften. Sie wünschen sich einerseits die bestmögliche Parallelität. Andererseits brauchen Sie Isolation, um die Konsistenz Ihrer Daten gewährleisten zu können. Das bedeutet, dass das System Datenelemente sperren muss, um bestimmte Transaktionen zu blockieren, während andere Transaktionen ihre Arbeit verrichten.

Sperrungen sind wichtig. Ohne Sperrungen könnten Sie keine Datenkonsistenz garantieren. Wenn eine Transaktion blockiert ist, wartet sie einfach darauf, bis sie wieder an der Reihe ist und ihre Arbeit erledigen kann. Manchmal haben die von einer Transaktion angeforderten Sperrungen jedoch nicht den gewünschten Effekt. In einigen Situationen geraten die Sperrungen, die von zwei oder mehr Transaktionen aufrechterhalten werden, in Konflikt, weil sie durch Warten nicht mehr aufgelöst werden können. Man spricht dann auch von einem *Deadlock*.

Stellen Sie sich beispielsweise eine Transaktion A vor, die eine Sperrung des Datenelements X anfordert. Sie versucht ebenfalls, Datenelement Y mit einer Schreibsperrung zu belegen, um ihre Arbeit zu beenden. Wenn Transaktion B das Datenelement Y mit einer Schreibsperrung anfordert und darauf wartet, dass die Schreibsperrung von Datenelement X freigegeben wird, ist eine Patt-Situation erreicht. Ohne das Eingreifen von Außen warten beide Transaktionen ewig darauf, dass die jeweils andere Transaktion ihre Sperrungen aufhebt.

Ein einzelner DBMS, wie zum Beispiel der SQL Server, kann Deadlock-Situationen schnell entdecken. Er löst die Situation, indem er eine der beiden Transaktionen beendet und eine Fehlermeldung an den Erzeuger der Transaktion sendet. Wenn die Sperrungen der beendeten Transaktion freigegeben sind, kann die andere Transaktion ihre Arbeit abschließen.

In einer verteilten Transaktion können einige Deadlock-Situationen nicht von einem einzelnen Ressourcen-Manager entdeckt werden. Deshalb verfügen COM+-Transaktionen über ein Timeout-Intervall von sechzig Sekunden. Kann eine Transaktion nicht innerhalb einer Minute ausgeführt werden, geht der DTC davon aus, dass ein Deadlock aufgetreten ist. Ihr Code sollte in jedem Fall Deadlock-Fehler bearbeiten können.

Sie sollten sich wirklich gut überlegen, ob Sie das Timeout-Intervall für die Transaktion einer Komponente auf einen höheren als den Standardwert von 60 Sekunden setzen sollten. In den meisten Systemen sind Transaktionen kurzlebig – in der Regel kürzer als eine Sekunde. Wenn Sie eine Komponente mit einem Timeout-Intervall von 20 Minuten konfigurieren und ein Deadlock entsteht, setzen Sie möglicherweise unbeabsichtigt mehrere Sperrungen, die nichts anderes tun, als die anderen Transaktionen 19 Minuten und 59 Sekunden zu blockieren. Achten Sie also darauf, dass Sie das Timeout-Intervall für eine transaktionale Komponente nie auf einen sehr hohen Wert setzen.

Deadlocks vermeiden

Sie können einige Standardtechniken einsetzen, um Deadlocks zu verhindern. Ein *zyklischer Deadlock* beispielsweise kann auftreten, wenn zwei Programmierer separate Transaktionen für dieselben Daten geschrieben haben. Ein Programmierer schreibt beispielsweise eine Transaktion, die die *Products*-Tabelle und anschließend die *Customers*-Tabelle modifiziert. Wenn ein anderer Programmierer eine zweite Transaktion erstellt, die in umgekehrter Reihenfolge auf dieselben Tabellen zugreift, ist die Wahrscheinlichkeit eines Deadlocks sehr groß. Sie können die Anzahl solcher zyklischen Deadlocks verringern, indem Sie einen konsistenten Datenzugriffsfluss zwischen den Transaktionen eines einzelnen Systems aufrechterhalten, die dieselben Daten benutzen.

Deadlocks, die aus Sperrkonvertierungen resultieren, kommen ebenfalls häufig vor und erfordern Ihre Aufmerksamkeit. Stellen Sie sich vor, Sie schreiben eine Transaktion, die eine bestimmte Menge eines Produkts aus dem Lagerbestand entfernt. Ihre Transaktion führt zunächst eine `SELECT`-Anweisung für die `Products`-Tabelle aus, um festzustellen, ob die angefragte Menge im Lager vorhanden ist. Wenn diese Menge gleich oder größer als die angefragte Menge ist, führen Sie die `UPDATE`-Anweisung aus, um die Stückzahl aus dem Lagerbestand zu entfernen. Wir gehen davon aus, dass die `SELECT`-Anweisung und die `UPDATE`-Anweisung für dieselben Daten in der `Products`-Tabelle aufgerufen werden.

In einer umfassenden OLTP-Applikation besteht die hohe Wahrscheinlichkeit, dass zwei separate Transaktionen die `SELECT`-Anweisung ausführen und Lesesperren vergeben, bevor eine der beiden Transaktionen die Schreibsperre zum Ausführen der `UPDATE`-Anweisung erteilen kann. Dies führt zu einem Deadlock. Jede Transaktion wartet darauf, dass die andere Transaktion die Lesesperre entfernt, damit eine Schreibsperre eingerichtet werden kann. Das Problem besteht darin, dass die Transaktion ein Datenelement zunächst mit einer Lesesperre belegt und anschließend versucht, die Sperrung in eine Schreibsperre umzuwandeln. Wenn zwei Transaktionen gleichzeitig eine Lesesperre vergeben, kann keine der beiden die Sperre in eine Schreibsperre konvertieren.

Viele Programmierer nehmen an, dass sie vor Deadlocks sicher sind, wenn sie eine Transaktion mit der Isolationsebene `Serializable` ausführen, aber auch das stellt keinen Schutz gegen die Probleme der Sperrkonvertierung dar. SQL Server unterstützt einen weiteren Sperrtyp, der als Aktualisierungssperre bezeichnet wird und das eben genannte Problem löst. Sie sollten eine Aktualisierungssperre verwenden, wenn Sie eine Lesesperre während einer Transaktion in eine Schreibsperre umwandeln müssen.

Eine Aktualisierungssperre führt zu Konflikten mit Schreibsperren und anderen Aktualisierungssperren. Eine Transaktion ist während des Versuchs, eine Aktualisierungssperre zu vergeben, blockiert, wenn eine andere Transaktion bereits eine Schreibsperre oder Aktualisierungssperre erteilt hat. Im Gegensatz zu einer Schreibsperre führt eine Aktualisierungssperre nicht zu Konflikten mit Lesesperren. Wenn eine Transaktion ein Datenelement mit einer Lesesperre belegt hat, kann eine andere Transaktion eine Aktualisierungssperre für dieses Datenelement anfordern. Wenn eine Transaktion eine Aktualisierungssperre aufrechterhält, können andere Transaktionen Lesesperren anfordern. Eine Aktualisierungssperre kann im Gegensatz zu einer Schreibsperre das Problem der Sperrungskonvertierung lösen, ohne die Parallelität negativ zu beeinflussen.

Unter Transact-SQL können Sie eine Aktualisierungssperre mit dem Schlüsselwort (`UPDLOCK`) in einer `SELECT`-Anweisung explizit anfordern. Nachfolgend finden Sie ein Beispiel, das die Aktualisierungssperre mit ADO in einer `SELECT`-Anweisung verwendet:

```
Dim sSQL As String
sSQL = "SELECT Quantity" & _
      " FROM Products (UPDLOCK)" & _
      " WHERE Product = 'Dog'"
Dim rs As ADODB.Recordset
Set rs = New ADODB.Recordset
rs.CursorLocation = adUseClient
rs.CursorType = adOpenForwardOnly
rs.LockType = adLockReadOnly
' Davon ausgehen, dass die Verbindung geöffnet ist.
rs.Open sSQL, conn
Dim Quantity As Long
Quantity = rs.Fields("Quantity")
' Die Aktualisierungssperre ist eingerichtet.
' Das UPDATE-Statement kann nun ausgeführt werden.
```

SQL Server verwendet auch Aktualisierungssperren, wenn Sie pessimistische Sperrungen mit serverseitigen Cursors benutzen. Der folgende Code setzt (UPDLOCK) nicht ein, führt jedoch zum selben Ergebnis:

```
Dim sSQL As String
sSQL = "SELECT Quantity" & _
      " FROM Products " & _
      " WHERE Product = 'Dog'"
Dim rs As ADODB.Recordset
Set rs = New ADODB.Recordset
rs.CursorLocation = adUseServer
rs.CursorType = adOpenDynamic
rs.LockType = adLockPessimistic
' Davon ausgehen, dass die Verbindung geöffnet ist.
rs.Open sSQL, conn
Dim Quantity As Long
Quantity = rs.Fields("Quantity")
' Die Aktualisierungssperre ist eingerichtet.
' Das UPDATE-Statement kann nun ausgeführt werden.
```

Beide Beispiele zeigen, wie Deadlocks ausgeschlossen werden, die durch eine Sperrkonvertierung entstehen. Wenn Ihr DBMS keine Aktualisierungssperren unterstützt, unterstützt er einen anderen Sperrtyp, um Deadlocks zu verhindern, die aus der Sperrkonvertierung resultieren. Sie können mit einer UPDATE-Anweisung, die einem Feld den eigenen Wert zuweist, immer eine Schreibsperre setzen. Damit wird der Sperrmanager gezwungen, eine Schreibsperre für den betreffenden Datensatz einzurichten. Das ist die rohe Gewalt, aber durchaus üblich.

Während Sie Ihre Applikationen so schreiben können, dass sie bestimmte Deadlocks vermeidet, sind andere Deadlocks in großen OLTP-Systemen unvermeidlich. Einige Deadlocks werden vom DBMS erkannt, andere werden aufgefangen, wenn eine COM+-Transaktion ihren Timeout-Wert erreicht hat. Sie müssen planen, Deadlock-Fehler zu verarbeiten, sobald sie auftreten, und sie so gut wie möglich auflösen. Manchmal kann man einen Deadlock auflösen, indem man die Transaktion von einer Fehlerbehandlungsroutine aus noch einmal absetzt. Das verbirgt den Deadlock vor den Benutzern einer Applikation. In anderen Situationen könnten Sie gezwungen werden, einen Fehler an den Aufrufer zurückzugeben, mit dem Hinweis, es später noch einmal zu versuchen.

Ein letztes Wort zum Tuning

In diesem Kapitel haben wir einen sehr großen Bereich abgedeckt. Als Letztes möchte ich die gesamte OLTP-Theorie zusammenfassen, indem ich den Ausführungsfluss einer typischen Transaktion aufzeichne. Ich werde hier das Beispiel eines Kaufauftrags der *Animal Market* Applikation verwenden. Wenn Sie den Ausführungsfluss einer Transaktion entwerfen, sollten Sie überlegen, welche Operationen die meisten Engpässe verursachen könnten. Diese Operationen sollten zuletzt ausgeführt werden.

Beispielsweise sollten Sie Operationen auf niedrigeren Isolationsebenen immer zuerst ausführen. Wenn Sie Zeilen oder Tabellen sperren müssen, die andere Benutzer blockieren, sollten Sie dies am Ende der Transaktion tun. Das ist wichtig, um die Sperrzeiten möglichst kurz zu halten, und wie Sie wissen, verringern kürzere Sperrzeiten den Durchsatz und halten die Antwortzeiten kurz. Hier ein möglicher Ausführungsfluss für unser Beispiel mit dem Kaufauftrag:

- Wir führen eine SELECT-Anweisung für die Customers-Tabelle mit Read Uncommitted aus, um sicherzustellen, dass der Kunde existiert. Falls sich der Kundename nicht in der Datenbank befindet, machen wir die Transaktion rückgängig und lösen einen Fehler aus.

- Wir führen eine `SELECT`-Anweisung für die `Products`-Tabelle mit `Read Uncommitted` aus, um sicherzustellen, dass das Produkt existiert, und um seinen Preis zu ermitteln. Dabei gehen wir davon aus, dass sich die Produktpreise während der normalen Geschäftsstunden nicht ändern. Falls sich das Produkt nicht in der Datenbank befindet, machen Sie die Transaktion rückgängig und lösen einen Fehler aus.
- Wir fügen einen neuen Datensatz in die `Orders`-Tabelle ein. Von den drei zu aktualisierenden Tabellen weist diese die geringsten Engpässe auf.
- Wir aktualisieren die `Customers`-Tabelle. Diese Tabelle weist den nächstgeringeren Engpass auf. Wir führen eine `SELECT`-Anfrage mit einer Aktualisierungssperre aus, um die Kreditgrenze des Kunden und den Kontostand zu lesen. Wir machen die Transaktion rückgängig und lösen einen Fehler aus, falls der Kunde nicht genügend Kredit für die Bestellung besitzt. Verfügt der Kunde über genügend Kredit, aktualisieren wir den Kundendatensatz und belasten das Konto mit dem Kaufpreis.
- Wir aktualisieren den Lagerbestand in der `Products`-Tabelle. Diese Tabelle weist den höchsten Engpass auf. Wir führen eine `SELECT`-Abfrage mit Aktualisierungssperre aus, um den Lagerbestand für das Produkt zu lesen. Wir machen die Transaktion rückgängig und lösen einen Fehler aus, wenn der Lagerbestand des Produkts nicht groß genug ist. Ist der Lagerbestand ausreichend, aktualisieren wir den Produktdatensatz, um den Lagerbestand zu reduzieren.
- Sind keine Fehler aufgetreten, schreiben wir die Transaktion fest.

Zusammenfassung

Dieses Kapitel hat Techniken für die Programmierung lokaler Transaktionen sowie von COM+-Transaktionen vorgestellt. Sie sollten immer den Ansatz wählen, der für Ihr jeweiliges Projekt am besten geeignet ist. Die transaktionale Programmierung versucht, den Durchsatz hoch und die Antwortzeiten kurz zu halten, während die Anzahl der Benutzer steigt. Um das zu gewährleisten, vermeiden Sie unnötige Sperrungen und versuchen, möglichst geringe Sperrzeiten zu realisieren.

Deklarative Transaktionen sind das Herz des COM+-Programmiermodells. Sie sind relativ einfach zu handhaben und sie lösen das unglaublich schwierige Problem, verteilte Transaktionen für unterschiedliche Ressourcenmanager auszuführen. Die COM+-Laufzeitumgebung übernimmt die Kommunikation mit dem DTC und stellt sicher, dass die Verbindungen korrekt zugeordnet sind. Sie müssen nur noch lernen, wie man das Ergebnis einer Transaktion steuert, indem man das Happy-Bit und das Done-Bit setzt.

Sie wollen jetzt vermutlich nichts mehr über die Reduzierung der Verbindungen hören, aber Sie sollten wirklich versuchen, so wenige Verbindungen wie möglich zwischen einer COM+-Applikation und Ihren RM zu verwenden, weil das der Schlüssel zu einer optimalen Leistung ist. Sie müssen sich die OLTP-Denkweise aneignen, um skalierbare Applikationen zu entwickeln. Beachten Sie, dass die Verwendung von Sitzungskomponenten in Kombination mit gespeicherten Prozeduren normalerweise die beste Leistung bringt.

Klar sollte auch sein, dass für die Optimierung einer OLTP-Applikation ein solides Grundwissen hinsichtlich einer Datenbank-API (wie zum Beispiel ADO), sowie genaue Kenntnisse über die von Ihnen verwendeten RMs bedingt. Nehmen Sie sich die Zeit zu lernen, wie die einzelnen RMs mit Transaktionen, Sperrungen und Parallelität umgehen, so dass Sie Ihre Visual Basic-Komponenten und den SQL-Code so schreiben können, dass Hardware und Software am besten genutzt werden.

9 IIS- und ASP-Komponenten erstellen

248	HTTP im Vergleich mit DCOM
250	Die IIS-/ASP-Architektur
265	Code zwischen ASP-Seiten und Komponenten aufteilen
271	Statusverwaltung in Webapplikationen
284	Zusammenfassung

Als Entwickler können Sie nicht das Phänomen »Web« ignorieren. Immer mehr Unternehmen verzichten auf die zweischichtige Entwicklung und wenden sich Browser-Applikationen für das eigene Intranet zu. Sehr viele Unternehmen erstellen Websites für den Außendienst, die Lieferanten, die Kunden oder die Aktionäre. Online-Verkäufe durch Unternehmen wie zum Beispiel Amazon.com haben enorme Dimensionen angenommen. Es scheint, als ob jeden Tag ein neuer 22-Jähriger zu einem Multimillionär wird, weil sein "Dot-Com"-Betrieb die öffentliche Erstemission erreicht hat.

Der Erfolg des Webs basiert auf offenen und allgegenwärtigen Standards wie zum Beispiel HTTP und HTML, die einer Applikation erlauben, das größtmögliche Publikum zu erreichen. Wenn Sie eine Web-Applikation erstellen, ist jede Person mit einem Browser, einer IP-Adresse und einer Internetverbindung ein potentieller Benutzer. Einige Sites nutzen erweiterte Features, wie zum Beispiel DHTML (Dynamic HTML) und clientseitige Skripts. Andere Sites verwenden generische Verfahren, um einfache Browser zu unterstützen und auf diese Weise ein größeres Publikum zu erreichen.

Im Laufe der nächsten Jahre wird XML (Extensible Markup Language) die Applikationen, die auf HTTP basieren, auf die nächste Ebene führen. HTTP ist ein exzellentes Protokoll für die Übertragung von formatierten Inhalten und wird nun zusehends zu einem Medium für die Übertragung von XML-Daten von Unternehmen zu Unternehmen.

Dieses Kapitel beschreibt, wie Web-Applikationen für Windows 2000 Server erstellt werden. Wir werden die Architektur von IIS (Microsoft Internet Information Services) und ASP (Active Server Pages) untersuchen, da beide die Bausteine für die Kommunikation mit HTTP auf der Windows-Plattform zur Verfügung stellen.

Da ich mich auf die erweiterten Aspekte der Entwicklung von Web-Applikationen konzentrieren möchte, werde ich voraussetzen, dass Sie ein wenig HTML-Erfahrung besitzen und serverseitige Skripts in ASP-Seiten erstellen können. Ich setze außerdem voraus, dass Sie Grundkenntnisse im Bereich integrierter ASP-Objekte, wie zum Beispiel Request, Response, Session, Application und Server besitzen. Sie sollten darüber hinaus wissen, wie HTML-Formulare verwendet und wie die mit einer ASP-Anforderung verknüpften Eingabedaten erfasst werden. Wenn Sie nicht mit den ASP-

Grundlagen vertraut sind, sollten Sie sich zunächst mit einigen angemessenen Informationsquellen beschäftigen, die dieses Buch ergänzen.

HTTP im Vergleich mit DCOM

Viele Unternehmen, die mit verteilte Applikationen arbeiten, benutzen HTTP als Protokoll für die Client-/Server-Kommunikation. Wir werden zu Beginn dieses Kapitels die wichtigsten Aspekte untersuchen, die in bestimmten Situationen dazu führen, dass HTTP und nicht DCOM bevorzugt wird. Sie sollten dann verstehen, wann eine verteilte Applikation einen Webserver, wie zum Beispiel IIS, benutzen muss.

Der erste Grund für den Einsatz von HTTP ist offensichtlich: Es erleichtert die plattformübergreifende Entwicklung. HTTP ist allgegenwärtig – Implementierungen stehen auf allen Plattformen zur Verfügung. HTTP-Verarbeitungshilfen sind außerdem weit verbreitet. Sie können deshalb eine große Zahl potentieller Benutzer erreichen. Applikationen, die jedoch DCOM verwenden, können lediglich Benutzer mit einer 32-Bit-Version von Windows erreichen. Wenn Sie zusätzlich die Tatsache berücksichtigen, dass die mit HTML erstellten Benutzeroberflächen mit vielen verschiedenen Browsern und Betriebssystemen ausgeführt werden können, bietet HTTP eine sehr attraktive Möglichkeit, um plattformübergreifende Applikationen zu erzeugen.

HTTP bedient nicht nur HTML-Benutzeroberflächen. Viele Applikationen verwenden HTTP als RPC-ähnliches Protokoll, um mit POST-Anweisungen beliebige Informationen an den Webserver zu senden und die Antwortnachrichten abzufangen. Das gebräuchlichste Format zum Versenden dieser Informationen ist XML. XML ist ein textbasiertes Datenformat, das Applikationen, die von verschiedenen Organisationen entwickelt wurden, erlaubt, Informationen auf eine plattformunabhängige Weise auszutauschen. Visual Basic-Programmierer werden froh darüber sein, dass der Microsoft-XML-Parser in Windows 2000 integriert ist und das Lesen, Schreiben, Versenden und Empfangen von XML unterstützt.

Ein weiterer Nachteil, der mit dem Einsatz von DCOM in einer Applikation verbunden ist, besteht darin, dass DCOM sehr viele clientseitige Konfigurationsarbeiten verlangt. Client-Computer müssen mit Informationen über AppIDs, CLSIDs, ProgIDs und IIDs konfiguriert werden. Sie verlangen außerdem die Registrierung von Typbibliotheken. Das Einrichten und die Pflege dieser clientseitigen Konfiguration können sehr aufwendig sein.

Hier ein Beispiel dafür, wie problematisch clientseitige Konfigurationsanforderungen sein können. Stellen Sie sich vor, Sie haben eine Visual Basic-Komponente erstellt und in eine COM+-Serverapplikation installiert. Sie haben die Komponente mit einer *MultiUse*-Klasse und öffentlichen Methoden erzeugt. Wie Sie wissen, können Sie einen Applikations-Proxy generieren, um die Typbibliothek auf der Client-Maschine zu installieren. Was aber geschieht, wenn Sie einige neue Methoden zur zweiten Version der Klasse und damit zu einer aktualisierten Version der DLL hinzufügen möchten? Wenn Sie die Methoden einfügen und die Komponente im Modus *Binär-Kompatibilität* erneut kompilieren, benötigen alle Client-Computer die aktualisierte Version der Typbibliothek. Sie müssten somit einen neuen Applikations-Proxy generieren und erneut installieren. Diese clientseitigen Konfigurationsanforderungen überraschen viele Entwickler und Administratoren.

Die mit DCOM und der clientseitigen Konfiguration verknüpften Probleme führen somit zu einem spürbaren administrativen Aufwand. Sie können dies verhindern, indem Sie HTTP und einen Standard-Webbrowser benutzen, da diese Elemente die clientseitige Konfiguration vermindern oder vollständig ausschließen. Wie bereits erwähnt wurde, ist HTTP ohne einen Webbrowser oder HTML ebenfalls eine realisierbare Option.

Ein weiteres Problem besteht darin, dass DCOM nicht für die Verbindung zwischen Computern verwendet werden kann, die in einem privaten Windows 2000- oder Windows NT-Netzwerk ausgeführt werden. Es ist schwierig und bisweilen unmöglich, Benutzer aus dem Internet zu authentifizieren und zu autorisieren. DCOM kann darüber hinaus nicht einfach über eine Firewall hinweg konfiguriert werden. Die übliche Firewall-Software erlaubt HTTP-Anforderungen, die Ports 80 und 443 zu passieren. Das Konfigurieren der von RPC und DCOM geforderten Ports ist mit dieser Software jedoch sehr kompliziert.

DCOM leidet außerdem darunter, ein sehr verbindungsorientiertes Protokoll zu sein. Client-Computer, die DCOM benutzen, verlangen in der Regel mindestens sechs Datenübertragungen zwischen Client und Server, um eine Anweisung ausführen zu können. Die ersten drei oder vier Übertragungen erstellen das Objekt. Weitere Übertragungen sind notwendig, um eine sichere Verbindung zum Serverprozess herzustellen. Die letzte Übertragung ruft die gewünschte Methode auf. Auch das Beenden der Verbindung erfordert mindestens zwei Übertragungen. Die erste Übertragung führt `Release` (von `IUnknown`) aus, während die zweite Übertragung die TCP-Verbindung beendet. HTTP ist sehr viel effizienter. Ein Client, der HTTP benutzt, kann seine erste Methode mit nur zwei Übertragungen ausführen. Die erste Übertragung richtet die anfängliche TCP-Verbindung ein und der zweite Datenaustausch sendet die eigentliche Anforderung und gibt die Antwort zurück.

Im sechsten Kapitel wurde beschrieben, wie DCOM einen Ping-Mechanismus verwendet, um eine verteilte Speicherbereinigung zu implementieren. Das »Pingen« zwischen Client und Server ist notwendig um Objekte freizugeben, wenn Client-Applikationen abstürzen. Dieser Ping-Mechanismus ist wichtig, aber auch aufwendig. Er begrenzt die Anzahl der DCOM-Clients, die gleichzeitig eine Verbindung zu einem einzelnen Servercomputer herstellen können.

DCOM führt darüber hinaus zu Problemen, wenn mehrere Clients gemeinsam verschiedene Servercomputer benutzen sollen. Die verbindungsorientierte Natur von DCOM verknüpft einen Client mit einem spezifischen Server, so dass Sie keinen Lastenausgleich für jede Anforderung vornehmen können. Die Apartment-Threaded-Natur einer Visual Basic-Komponente verbindet einen Client außerdem mit einem spezifischen Thread auf einem bestimmten Server. Dies bedeutet, dass eine Applikation, die auf Visual Basic und DCOM basiert, das Thread-Pool-Schema, das einer COM+-Serverapplikation zur Verfügung steht, nicht vollständig nutzen kann.

Der Lastenausgleich für eine auf HTTP basierende Applikation ist sehr viel einfacher zu realisieren. Eine solche Applikation ist auch besser skalierbar als eine DCOM-Applikation. Später in diesem Kapitel werden mehrere HTTP-Techniken für den Lastenausgleich vorgestellt, die eine gebräuchliche und effektive Applikationsskalierung bieten. Wie Sie sehen werden, sollten Visual Basic-Objekte immer im Gültigkeitsbereich einer einzelnen HTTP-Anforderung erstellt und freigegeben werden. Der HTTP-Lastenausgleich skaliert eine Applikation, damit diese tausende von Benutzern bedienen kann.

Die für HTTP zur Verfügung stehenden Lastenausgleichsschemata erhöhen nicht nur die Anzahl der potentiellen Benutzer. Sie können außerdem die Verfügbarkeit von Applikationen verbessern, da jede Anforderung eines bestimmten Benutzers an einen anderen Server weitergeleitet werden kann. Wenn ein Webserver-Computer abstürzt oder gegenwärtig gewartet wird und somit nicht zur Verfügung steht, können die Clients auch weiterhin die Applikationen ohne Unterbrechung benutzen. Dies ist nicht möglich, wenn ein Client über DCOM mit einem bestimmten Objekt verbunden ist, das auf einem spezifischen Servercomputer ausgeführt wird.

Trotz dieser Argumente, die dafür sprechen, dass Sie HTTP anstelle von DCOM verwenden sollten, weist auch die Webentwicklung besondere Herausforderungen auf. Ständig neu veröffentlichte Standards erschweren das Erstellen und Warten von Webapplikationen. Es ist auch nicht

einfach, Sitzungen mit einem clientspezifischen Status zu verwalten, wenn Sie ein verbindungsloses Protokoll wie zum Beispiel HTTP verwenden. Auch das Erstellen eines effektiven Rückrufbenachrichtigungs-Systems für Client-Applikationen, die in einem Browser am anderen Ende des Internets ausgeführt werden, ist etwas kompliziert. Ich bin mir jedoch sicher, dass Sie Herausforderungen lieben.

Die IIS-/ASP-Architektur

IIS (Internet Information Services) ist mehr als ein einfacher Webserver. Es ist ein hochentwickelter Server, der auf vielen bekannten Protokollen und Internetstandards basiert. Zusätzlich zu HTTP kennt IIS weitere Protokolle, wie zum Beispiel FTP (File Transfer Protocol), NNTP (Network News Transfer Protocol) und SMTP (Simple Mail Transfer Protocol). Dieses Kapitel konzentriert sich darauf, wie IIS den HTTP-Verkehr bewältigt. Wir werden insbesondere untersuchen, wie sich IIS in die Rolle des Webserver für die Windows-Plattform fügt. Die Informationen in diesem Kapitel basieren auf IIS 5, das mit Windows 2000 Server und Windows 2000 Professional ausgeliefert wird.

Die erste Version von IIS war lediglich ein Dienst, der statische HTML-Seiten bediente. Neuere Versionen von IIS sind weiter entwickelt. Sie bedienen nicht nur statische Webseiten, sondern führen eine aufwendige serverseitige Verarbeitung als Reaktion auf HTTP-Anforderungen durch.

Sie können zwischen einigen unterschiedlichen Vorgehensweisen auswählen, um die serverseitige Verarbeitung mit IIS durchzuführen. IIS kann, wie auch viele UNIX-Webserver, CGI-Applikationen (Common Gateway Interface) ausführen. Diese CGI-Unterstützung vereinfacht das Portieren bereits bestehender Webapplikationen auf die Windows-Plattform. CGI ist jedoch nicht die beste Lösung zum Erstellen von Applikationen für IIS. Die auf der nativen IIS-API basierende Webentwicklung ist flexibler und bietet eine bessere Performance. Die native IIS-API ist auch als ISAPI (Internet Server API) bekannt.

Software, die auf die ISAPI zugreift, ist in zwei Ausführungen vorhanden: ISAPI-Erweiterungen und ISAPI-Filter. ISAPI-Erweiterungen werden als Reaktion auf HTTP-Anforderungen für eine bestimmte Dateinamenserweiterung ausgeführt. ISAPI-Filter werden als Reaktion auf den gesamten HTTP-Verkehr für einen spezifischen IIS-Computer, eine bestimmte Site oder eine Applikation ausgeführt.

Software, die direkt auf die ISAPI zugreift, bietet die beste Performance und größte Flexibilität. Sie weist jedoch einige entmutigende Anforderungen auf. ISAPI-Software muss in DLLs kompiliert werden, die Aufruffunktionen als bekannte Einsprungspunkte implementieren. Dies wird nicht von Visual Basic unterstützt, so dass die ISAPI-Entwicklung in der Regel mit C oder C++ durchgeführt wird. Die ISAPI-Programmierung zwingt Sie außerdem, Low-Level-Arbeiten wie zum Beispiel das Implementieren eines Thread-Pool-Managers und das Schreiben eines benutzerdefinierten Synchronisierungscode zu verrichten. Viele Unternehmen verzichten auf die Programmierung von ISAPI-Software, da ihnen das notwendige Wissen fehlt und sie nicht bereit sind, in Zeit und Geld zu investieren, um dieses Wissen zu erhalten.

ASP (Active Server Pages) ist eine beliebte Alternative zur ISAPI-Entwicklung. ASP kann jedoch nicht vollständig auf die ISAPI verzichten. ASP ist eine Laufzeitschicht, die auf ISAPI aufsetzt und die Low-Level-Details der direkten ISAPI-Programmierung verbirgt. ASP selbst ist eine ISAPI-Erweiterung, die Ihnen erlaubt, serverseitige Logik mit einer einfachen Skriptsprache zu schreiben. ASP bietet die Annehmlichkeit eines integrierten Thread-Pool-Schemas und verlangt nicht von Ihnen, einen benutzerdefinierten Synchronisierungscode zu schreiben.

Da sich dieses Buch auf die Software-Entwicklung mit Visual Basic konzentriert, wird das Erstellen von ISAPI-Filtern und ISAPI-Erweiterungen hier nicht beschrieben. Sie erfahren stattdessen, was mit ASP-Seiten und Visual Basic-Komponenten möglich ist.

Das ASP-Gerüst

Lassen Sie uns die grundlegenden ASP-Konzepte betrachten. Das ASP-Gerüst ist um drei wichtige Voraussetzungen herum errichtet worden. Da eine Webapplikation erstens dazu tendiert, Seiten sowohl aus statischen als auch aus dynamischen Daten zu generieren, ist eine ASP-Seite in der Regel eine Kombination aus HTML-Text (statische Daten) und einem serverseitigen Skriptcode (Code zum Erzeugen dynamischer Daten). Da Entwickler zweitens Sites mit einer komponentenbasierten Software programmieren möchten, muss eine ASP-Seite Objekte aus COM-Standardkomponenten laden und ausführen können. Da Webentwickler drittens aufgrund der Tatsache, dass HTTP ein verbindungsloses Protokoll ist, Hilfe im Bereich der Statusverwaltung benötigen, muss das ASP-Gerüst ein Schema für die Verwaltung des clientspezifischen Status und des applikationsweiten Status über HTTP-Anforderungen hinweg anbieten.

Wie Sie wissen, ist eine ASP-Seite eine Textdatei, die HTML und serverseitige Skripts enthält. Serverseitige Skripts können mit unterschiedlichen Skriptsprachen einschließlich VBScript und JScript (Microsofts JavaScript-Implementierung) geschrieben werden. Die ASP-Laufzeit reagiert auf eine .ASP-Dateianforderung, indem sie einen Filter ausführt, der spezielle Tags sucht, in denen Anweisungen zur Code-Analyse und Ausführung auf dem Server enthalten sind. Die ASP-Laufzeit führt diese serverseitigen Skripts aus und gibt einen HTML-Datenstrom an den Client zurück. Dieser Datenstrom kann statischen HTML-Code aus der ASP-Seite und HTML-Code der als Ergebnis einer serverseitigen Verarbeitung dynamisch generiert wurde, kombinieren. Ein ASP-Entwickler kann serverseitige Skripts schreiben, um Geschäftslogik auszuführen, auf Datenbanken zuzugreifen und mit COM-Objekten zu interagieren.

Die ASP-Laufzeit bietet, zusätzlich zum einfachen Ausführen von Skripts und Visual Basic-Objekten auf dem Webserver, eine Statusverwaltung an. Die ASP-Entwickler erkannten die schwierige Verwaltung von applikationsweiten Variablen und dem clientspezifischen Status während der Entwicklung von Applikationen, die auf einem verbindungslosen Protokoll wie zum Beispiel HTTP basieren. Das ASP-Gerüst verwendet eine Name-/Wert-Liste, die als *Application*-Objekt bezeichnet wird, mit deren Hilfe alle Clients einer spezifischen Applikation die im Speicher befindlichen Daten gemeinsam nutzen können. Das Problem der Verwaltung des clientspezifischen Status wird mit einer weiteren Name-/Wert-Liste gelöst, die als *Session*-Objekt bekannt ist.

Die Sitzungsverwaltung von ASP wird durch die Übergabe von Cookies zwischen dem IIS-Webserverprozess und dem Client-Browser möglich. Das Generieren und die Verwaltung dieser Cookies sowie das Ablegen des clientspezifischen Status in einem Cache ist für jeden ASP-Entwickler nachvollziehbar. Jeder Benutzer, der bereits mit dem ASP-Objekt *Session* gearbeitet hat, kann Ihnen sagen, wie einfach dies ist. Die ASP-Laufzeit verrichtet beinahe die gesamte Arbeit im Hintergrund.

ASP-Applikationsvariablen und ASP-Sitzungsvariablen sind für jeden ASP-Entwickler und für alle Visual Basic-Entwickler zugänglich, die Komponenten für IIS erstellen. Dieses ASP-Statusverwaltungs-Feature vereinfacht das Erstellen von sitzungsbasierten Applikationen, in denen Clients einen serverseitigen Status über eine Serie von ASP-Anforderungen hinweg etablieren. Sie sollten jedoch verstehen, wie der Einsatz der ASP-Sitzungsverwaltung die Skalierbarkeit einer Applikation begrenzen kann. Sie werden im Verlauf dieses Kapitels erfahren, wieso ASP-Sitzungsvariablen nicht für groß angelegte Applikationen geeignet sind.

IIS-Applikationen

Der Haupt-Webserverprozess für IIS wird als Windows-Dienst ausgeführt und mit einer ausführbaren Datei namens *INETINFO.EXE* gestartet. Alle eingehenden HTTP-Anforderungen werden zunächst an diesen Webserverprozess gesendet. Wenn IIS erkennt, dass sich eine Anforderung an eine statische Webseite (wie zum Beispiel eine Datei mit einer *.HTM*-Endung) richtet, gibt es diese Seite an den Client zurück. Wenn IIS jedoch feststellt, dass die Anforderung an eine IIS-Erweiterung (Wie zum Beispiel die ASP-Laufzeit) weitergeleitet werden muss, führt es einen Aufruf an ein vom System zur Verfügung gestelltes Hilfsobjekt durch, das als WAM (Web Application Manager) bezeichnet wird.

Wenn Sie auf einem Windows 2000 Server (mit installiertem IIS) einen Blick in den Ordner *COM+-Anwendungen* werfen, sehen Sie eine Bibliotheksapplikation namens *IIS In-Process Applications*. Diese Applikation enthält eine Free-Threaded-Komponente mit der Bezeichnung *IISWAM.W3SVC*, die zusammen mit IIS installiert wird. Diese Komponente ist ein WAM, der die eingehenden Anforderungen für ISAPI-Erweiterungen an die entsprechende DLL weiterleitet. Wenn ein WAM beispielsweise eine Anforderung mit einer *.ASP*-Erweiterung erhält, wird diese Anforderung an *ASP.DLL* umgeleitet.

IIS verwendet eine Abstraktion, die IIS-Applikation genannt wird und den physikalischen Verzeichnissen bestimmte URLs zuweist. Eine IIS-Applikation wird deshalb auch als virtuelles Verzeichnis bezeichnet. Die Installation von IIS erstellt automatisch eine Standardwebsite, die als Root-Applikation dient. Der Pfad zum obersten Verzeichnis der Site lautet in der Regel *C:\Inet-Pub\wwwroot*. Sie können zusätzliche IIS-Applikationen konfigurieren, indem Sie virtuelle Verzeichnisse mit dem administrativen Hilfsmittel *Internetdienste-Manager* erzeugen. Sie können zum Beispiel einem physikalischen Verzeichnis Ihrer Wahl eine URL wie *http://LocalHost/My-Application* zuordnen. Sobald Sie ein virtuelles Verzeichnis erstellt haben, können Sie Ihre HTML- und ASP-Seiten diesem Verzeichnis hinzufügen. Jedes Element, das sich dann von diesem Punkt aus abwärts in der Hierarchie des Verzeichnisses befindet, ist Bestandteil der IIS-Applikation (bis Sie eine andere IIS-Applikation erreichen).

Wie Sie sehen, ist es sehr einfach eine IIS-Applikation einzurichten und auszuführen. Wenn Sie nicht mit der Konfiguration und Verwaltung von IIS-Applikationen vertraut sind, sollten Sie die Dokumentation lesen, die mit IIS 5 ausgeliefert wird. Die Dokumentation selbst ist ein virtuelles Verzeichnis, auf das Sie mit dem URL *http://LocalHost/iishelp* zugreifen. Es bietet eine Schritt-für-Schritt-Anleitung zum Erstellen und Verwalten von IIS-Applikationen an.

IIS überwacht die Konfigurationsinformationen für IIS-Entitäten wie zum Beispiel Computer, Sites und Applikationen in einer speziellen Datenbank, die IIS-Metadatenbank genannt wird. Das Konzept der IIS-Metadatenbank gleicht dem der COM+-Registrierungsdatenbank (RegDB) und der Windows-Registrierung. Sie ist jedoch ein separater Datenspeicher, der nur von IIS verwendet wird.

Die IIS-Metadatenbank ist eine hierarchische Datenbankdatei, die Schlüssel-/Wertpaare enthält. Einige Metadatenbankschlüssel werden mit dem Internetdienste-Manager konfiguriert. Auf andere Metadatenbankschlüssel können Sie nur programmgesteuert über IIS-Admin-Objekte zugreifen, die von den ADSI-Schnittstellen (Active Directory Services Interfaces) zur Verfügung gestellt werden.

Sie werden später in diesem Kapitel ein kurzes Beispiel kennen lernen, das die IIS-Admin-Objekte mit ADSI verwendet. Sie finden eine vollständige Übersicht über das Schreiben von administrativen Skripts und Programmen in der IIS-Dokumentation oder dem Plattform-SDK, wenn Sie dort nach »*Administering IIS Programmatically*« suchen. Diese Ressourcen bieten zusätzliche Informationen über den Zugriff auf Metadatenbankschlüssel mit WSH-Dateien (Windows Script Host) oder mit Visual Basic.

ASP-Anforderungen bearbeiten

Abbildung 9.1 zeigt die Bearbeitungsreihenfolge einer typischen ASP-Anforderung. Die Anforderung wird zunächst von einem WAM-Objekt bearbeitet und dann zur ASP-Laufzeit weitergeleitet. Die ASP-Laufzeit reagiert, indem sie ein internes Seitenobjekt erstellt. Das Seitenobjekt führt einen Filter auf der angeforderten ASP-Seite aus, um jede darin enthaltene serverseitige Verarbeitungsanweisung zu extrahieren und zu verarbeiten. Beachten Sie, dass die Schnittstellen der WAM-Objekte und internen Seitenobjekte nicht dokumentiert sind. Sie müssen niemals direkt mit diesen Schnittstellen interagieren. Wenn Sie jedoch wissen, wie diese Systemobjekte miteinander interagieren, verstehen Sie besser, wie eine ASP-Anforderung bearbeitet wird.

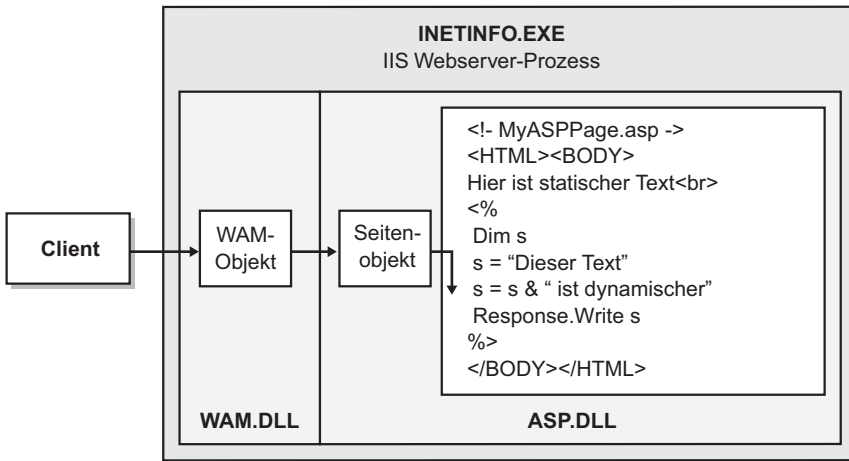


Abbildung 9.1: WAM-Objekte leiten .ASP-Dateianforderungen an die ASP-Laufzeit weiter, die daraufhin ein internes Seitenobjekt erstellt, um die serverseitigen Skripts aus der ASP-Seite zu filtern und ausführen zu lassen.

Das WAM-Objekt ist ein Free-Threaded-Objekt. Wenn es einen Aufruf an eine ISAPI-Erweiterung wie zum Beispiel *ASP.DLL* vornimmt, verwendet es einen MTA-Thread (MTA=Multi-Threaded-Apartment), der von einem Thread-Pool zur Verfügung gestellt wird. Dieser Pool wird von der IIS-Laufzeit verwaltet. Einer der komplizierten Aspekte des Erstellens von ISAPI-Erweiterungen ist die von diesem MTA-Thread-Pool verursachte Parallelität und Synchronisierung. Die ASP-Laufzeit beseitigt einen Teil dieser Komplexität, indem sie jede ASP-Anforderung in einen STA-Thread (Single-Threaded-Apartment) schaltet, bevor der Code einer ASP-Seite ausgeführt wird. Die ASP-Laufzeit verwaltet einen separaten Pool mit STA-Arbeits-Threads. (ASP verwendet eigentlich das COM+-Thread-Pooling, um dies zu tun.) Die ASP-Designer entwickelten dieses Thread-Pool-Schema, um einen optimierten Ausgleich zwischen der Parallelität und der Ressourcenverwendung zu schaffen und um eine vom Programmierer unterstützte Synchronisierung zu vermeiden.

Abbildung 9.2 zeigt, wie die ASP-Laufzeit jede Anforderung aus einem MTA-Thread in einen STA-Thread schaltet. Ein Weiterleitungsmechanismus in der ASP-Laufzeit ordnet jede Anforderung in einer zentralen Anforderungswarteschlange an. Die STA-Threads in einem von ASP verwalteten Pool überwachen diese Warteschlange und bedienen die Anforderung auf einer FIFO-Basis (First-In, First-Out). Beachten Sie, dass die Größe dieses Thread-Pools dynamisch ist. Die

ASP-Laufzeit erstellt zusätzliche Threads, wenn ein hohes Verkehrsaufkommen herrscht, und sie gibt Threads frei, wenn sich alles wieder beruhigt hat.

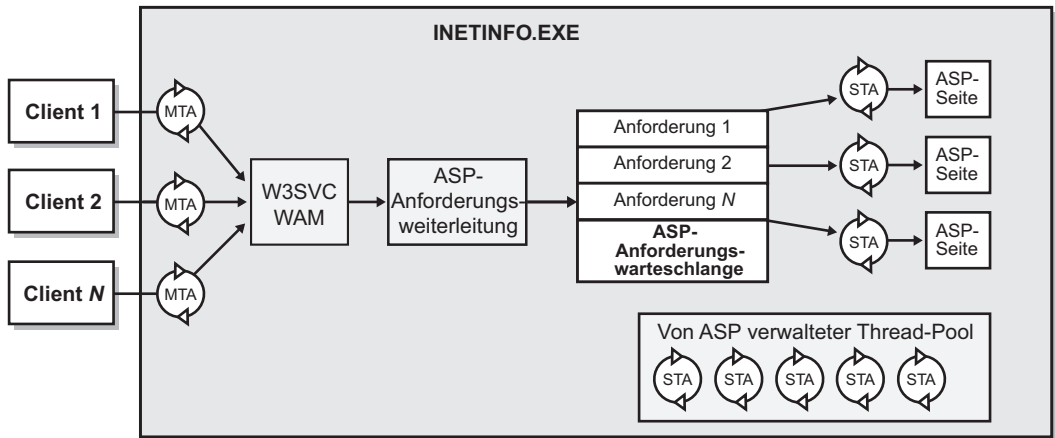


Abbildung 9.2: ASP-Anforderungen werden in einer zentralen Anforderungswarteschlange abgelegt, wo sie von STA-Arbeits-Threads auf einer FIFO-Basis (First-In, First-Out) bedient werden.

Der Wechsel zu einem STA-Thread verhindert die Parallelität, wirkt sich aber ebenfalls auf die Performance aus. Eine ISAPI-Erweiterungs-DLL, die jede Anforderung auf einem einzelnen MTA-Thread bearbeitet, bietet schnellere Reaktionszeiten. Diese Leistungsunterschiede werden offensichtlich, wenn Benchmarks für eine ISAPI-Erweiterungs-DLL mit ASP-Anforderungen durchgeführt werden. Die meisten Unternehmen sind jedoch bereit, für eine Umgebung, die das Schreiben von Software vereinfacht, auf ein wenig Performance zu verzichten. Sie gehen davon aus, dass zusätzliche Hardware den Rechenaufwand des ASP-Threading-Schemas kompensieren kann.

Beachten Sie, dass IIS 5 eine Optimierung bietet, die in früheren Versionen von IIS nicht zur Verfügung stand. Wenn die ASP-Anforderungsweiterleitung mit einer ASP-Seite konfrontiert wird, die keine serverseitigen Verarbeitungs-Tags enthält, bearbeitet sie die Seite auf dem MTA-Thread und sendet sie nicht zur Anforderungswarteschlange. Dies verbessert die Performance. In früheren IIS-Versionen mussten Sie Seiten mit einer *.HTM*- oder *.HTML*-Endung benutzen, um den Rechenaufwand eines Thread-Wechsels zu vermeiden. Inzwischen können alle Seiten, unabhängig davon, ob sie serverseitige Verarbeitungs-Tags enthalten, eine *.ASP*-Endung aufweisen. Der Thread-Wechsel wird nur dann ausgeführt, wenn er erforderlich ist.

IIS bietet eine Möglichkeit an, die maximale Größe des Thread-Pools und die maximale Größe der Anforderungswarteschlange zu konfigurieren. Die Thread-Pool-Größe wird mit dem *AspProcessorThreadMax*-Schlüssel in der IIS-Metadatenbank kontrolliert. Die Standardeinstellung für diesen Schlüssel ist 25 pro Prozessor und pro Prozess. Auf einer Maschine mit vier Prozessoren bedeutet dies, dass jeder Prozess, der ASP enthält, bis zu 100 Arbeits-Threads aufweisen kann. Für die meisten ASP-Applikationen, die sich in Betrieb befinden, gilt jedoch, dass diese Voreinstellung nicht verändert werden sollte. Sie sollten sie erst dann ändern, wenn Sie über alle Aspekte der Feineinstellung eines Thread-Pools nachgedacht haben. Das Reaktionsvermögen und der Durchsatz Ihrer Site kann versehentlich beeinträchtigt werden, wenn Sie diese Einstellung auf einen zu hohen Wert setzen.

Der primäre Grund, den Wert von *AspProcessorThreadMax* zu erhöhen, besteht darin, eine Applikation zu optimieren, in der ASP-Seiten Aufrufe an COM-Objekte initiieren, die zu einer längeren Ausführung in anderen Prozessen oder auf anderen Maschinen führen. In den meisten anderen Szenarien führt das Erhöhen dieses Wertes zu einer geringeren Performance. Der Grund hierfür ist der zusätzliche Rechenaufwand, der für das Einfügen von Threads in einen Webserver-Prozessor und das Entfernen dieser Threads aus dem Prozessor benötigt wird. Beachten Sie, dass Sie diese Einstellung nicht mit dem Internetdienste-Manager modifizieren können. Sie müssen sie programmgesteuert mit einem administrativen Skript oder einer Visual Basic-Applikation ändern.

In einigen Situationen kann auch die Größe der ASP-Anforderungswarteschlange eine Feineinstellung erfordern. Wenn der Webserver mehr Anforderungen pro Sekunde erhält, als er bearbeiten kann, wird die Anforderungswarteschlange immer größer. Wenn sie zu groß wird, nehmen die Reaktionszeiten für die Bedienung eingehender Anforderung dramatisch zu. IIS setzt eine Maximalkapazität für die Warteschlange, so dass der Anforderungsrückstand nicht zu groß wird.

Die ASP-Laufzeit erlaubt der Warteschlange standardmäßig 3000 Anforderungen aufzunehmen. Sobald die Warteschlangenkapazität erreicht wird, werden weitere eingehende ASP-Anforderungen mit dem Fehler »*Server Too Busy*« zurückgewiesen. Dies ist vernünftig, da IIS jede Anforderung zurückweisen kann, die es nicht in einer angemessenen Zeit bearbeiten kann.

Das Einstellen der Warteschlangengröße ändert den Zeitpunkt, ab dem der Webserver mit der Zurückweisung eingehender Anforderung beginnt. Sie müssen deshalb einen Ausgleich zwischen einer umfassenden Verfügbarkeit und kurzen Reaktionszeiten finden. Das Ziel besteht also darin, die Warteschlangengröße zu konfigurieren, um eine kurzzeitige Zunahme der Anforderungen zuzulassen, aber den Warteschlangenkückstand während einer extremen Arbeitsauslastung zu begrenzen. Wenn Ihr Webserver überlastet wird, ist es besser, Fehlermeldungen an die Clients zurückzusenden, als diese zu lange warten zu lassen.

Sie können den *AspRequestQueueMax*-Schlüssel und den *AspProcessorThreadMax*-Schlüssel programmgesteuert mit Hilfe der IIS-Admin-Objekte und ADSI konfigurieren. Sie haben zum Beispiel die Möglichkeit, die Typbibliothek *Active DS Type Library* in einem Standard-EXE-Projekt zu referenzieren und den folgenden Code zu schreiben:

```
Dim MyWebServer As ActiveDs.IADs
Set MyWebServer = GetObject("IIS://LocalHost/W3SVC")
MyWebServer.Put "AspProcessorThreadMax", 30
MyWebServer.Put "AspRequestQueueMax", 1500
MyWebServer.SetInfo
```

Lassen Sie uns zusammenfassen, wie der ASP-Thread-Pool für eine IIS-Standardinstallation auf einer Maschine mit nur einem Prozessor arbeitet. Der Pool kann 25 STA-Arbeits-Threads pro Prozess aufnehmen. (Wenn Sie weitere Threads wünschen, sollten Sie einen Rechner mit zwei oder vier Prozessoren als Webserver verwenden.) Wenn eine Anforderung eintrifft, wird sie in der Anforderungswarteschlange angeordnet. Die ASP-Laufzeit entnimmt nun einen Leerlauf-STA-Thread aus dem Pool, sofern dieser verfügbar ist. Wenn alle STA-Arbeits-Threads mit der Verarbeitung anderer Anforderungen beschäftigt sind, werden eingehende Anforderungen in die Warteschlange gestellt und auf einer FIFO-Basis (First-In, First-Out) bearbeitet. Solange die Warteschlange nicht die Standardkapazität von 3000 Anforderungen erreicht, werden alle Anforderung bearbeitet.

Da Sie nun das in die ASP-Laufzeit integrierte Threading-Schema verstehen, sollten wir darüber sprechen, wie Sie als Visual Basic-Programmierer den größtmöglichen Nutzen daraus ziehen.

Visual Basic-Objekte aus ASP-Seiten erstellen

Lassen Sie uns diesen Abschnitt mit einer Untersuchung der Beziehung zwischen einem Visual Basic-Objekt und der ASP-Laufzeitumgebung beginnen. Da Sie Visual Basic-DLLs für den Einsatz auf einem Webserver erstellen und konfigurieren werden, sollten Sie verstehen, wie die Objektaktivierung und die Lebenszyklus-Verwaltung arbeiten, wenn Sie COM-Objekte aus einer ASP-Seite heraus generieren. ASP erleichtert glücklicherweise das Erzeugen von Webapplikationen, die die auf dem Webserver ausgeführten Visual Basic-Komponenten benutzen.

Die folgende ASP-Seite benutzt ein in VBScript geschriebenes serverseitiges Skript, um ein Visual Basic-Objekt zu erstellen und mit diesem zu interagieren.

```
<%@ LANGUAGE="VBSCRIPT"%>
<HTML><BODY>
<H2>Customer List</H2>
<%
    Dim sProgID, CustomerManager
    sProgID = "MyWebDll.CCCustomerManager"
    Set CustomerManager = Server.CreateObject(sProgID)
    Response.Write CustomerManager.GetCustomerListHTML()
%>
</BODY></HTML>
```

Wie Sie sehen, ist es sehr einfach, ein Visual Basic-Objekt aus einer ASP-Seite heraus zu erstellen. Beachten Sie, dass der ASP-Seitenfilter die eingebetteten Tags `<%` und `%>` zu Beginn und am Ende eines serverseitigen Skripts erkennt. Sie können serverseitige Skripts ebenfalls mit dem `SCRIPT`-Tag und dem `RunAt=Server`-Attribut in eine ASP-Seite einfügen.

Ich muss betonen, dass Sie das Ausführen von Visual Basic-Komponenten aus einem serverseitigen Skript heraus nicht davor bewahrt, plattformübergreifende Lösungen entwickeln zu müssen. Sie können so viel COM-Code auf dem Webserver ausführen lassen, wie Sie möchten, ohne clientseitige Abhängigkeiten vom Microsoft Internet Explorer oder der Windows-Plattform zu erzeugen. Solange Sie plattformunabhängigen HTML-Code schreiben, kann der Client-Computer verschiedene Browser und Betriebssysteme benutzen. Sie entscheiden, welche HTML-Tags und serverseitigen Skripttechniken zu browserspezifischen Abhängigkeiten führen.

ASP-Skripte durch Visual Basic-Komponenten ersetzen

Sie können einige Vorteile nutzen, wenn Sie die Geschäftslogik und den Datenzugriffscode aus einer ASP-Seite entfernen und in einer Visual Basic-DLL anordnen. ASP-Seiten sind besonders für die Programmierung im Kleinen geeignet. Das Verwalten von Applikationen, die tausende Zeilen von Code enthalten, kann jedoch sehr schwierig werden. Nachfolgend sind einige Hilfen aufgeführt, an die Sie denken sollten, wenn Sie entscheiden, ob Ihre Logik in einer ASP-Seite oder in einer Visual Basic-Komponente angeordnet werden soll:

- Sie erreichen höhere Ebenen der Kapselung und der Wiederverwendung von Code, indem Sie eine Programmiersprache wie zum Beispiel Visual Basic einsetzen, die klassenbasiert und objektorientiert ist.
- Die Visual Basic-IDE bietet eine Typüberprüfung während der Kompilierung und den effektiven Einsatz von IntelliSense. Dies führt zu einer größeren Produktivität und zu weniger Stunden, die mit dem Debuggen verbracht werden (besonders wenn Sie auf eine COM-Bibliothek wie zum Beispiel ADO zugreifen).

- Visual Basic-Code wird im Gegensatz zu einem ASP-Skript kompiliert. Dies bedeutet, dass die Logik in einer Visual Basic-DLL schneller als ein serverseitiges ASP-Skript ausgeführt werden kann. Sie können jedoch nicht immer davon ausgehen, dass der Einsatz Ihrer Logik in einer Visual Basic-Komponente zu einer höheren Verarbeitungsgeschwindigkeit als die einer ASP-Seite führt. Das Erstellen und Zerstören von Visual Basic-Objekten erfordert eigene Bearbeitungszyklen. Das Verschieben von Code aus einer ASP-Seite in eine Visual Basic-Komponente kann die Performance für eine Anforderung mit minimaler Logik sogar beeinträchtigen. Wenn der Code für eine Anforderung jedoch zunimmt, wird das kompilierte Format des Visual Basic-Codes die äquivalente Logik in einer ASP-Seite immer überbieten.
- Visual Basic-Code verwendet die direkte vTable-Bindung, wenn Sie in Ihren Programmen auf COM-Bibliotheken wie zum Beispiel ADO zugreifen. ADO-Skripte benutzen `IDispatch` und die späte Bindung für jeden Methodenaufruf, wenn sie mit einem COM-Objekt interagieren. Wenn für eine Anforderung zum Beispiel zwanzig oder mehr Aufrufe an ADO-Objekte erforderlich sind, kann die durch die späte Bindung entstehende Leistungseinbuße signifikant sein.
- Die mit Visual Basic erstellten COM-DLLs können alle Vorteile der Integration zwischen IIS und der COM+-Laufzeitumgebung nutzen.
- Sie können von einer ActiveX-DLL aus direkt auf die Win32-API zugreifen. Mit einem Skript in einer ASP-Seite ist dies nicht möglich. (Natürlich kann Ihr Visual Basic-Code den Webserverprozess zum Absturz bringen, wenn Sie eine Win32-API-Funktion inkorrekt aufrufen.)

Sie sollten ebenfalls an die Besonderheiten der Scripting-Clients denken, die wir im vierten Kapitel besprochen haben. Sie müssen auf einige Eigentümlichkeiten gefasst sein, wenn Sie ASP-Code mit einer Skriptsprache, wie zum Beispiel VBScript schreiben. Variablen in einem serverseitigen Skript können nicht explizit typisiert werden. Sie werden implizit als Variants zugeordnet. Daran müssen Sie sich erst gewöhnen, wenn Sie ein Visual Basic-Programmierer sind.

VBScript kann die Ausgabeparameter von Methoden außerdem nur dann abrufen, wenn sie als Variants deklariert sind. Wenn Sie versuchen, eine Methode auf einem Visual Basic-Objekt aufzurufen, dessen Ausgabeparameter als String oder Double deklariert ist, schlägt der VBScript-Code fehl und generiert den Fehler »Datentypen unverträglich«. Sie sollten Ihre Visual Basic-Methoden deshalb so entwerfen, dass alle Ausgabeparameter (ByRef-) Variants sind, sofern Sie diese von einem ASP-Skript aus aufrufen möchten.

Sie können Fehler von einem Visual Basic-Objekt an eine ASP-Seite zurückleiten. Die Fehlerbehandlung in VBScript ist jedoch, verglichen mit der von Visual Basic, nicht sehr progressiv. Die Fehlerbehandlung wird in der Regel mit einer `On Error Resume Next`-Anweisung und der Überprüfung der Fehlernummer nach jedem Aufruf durchgeführt. Dies ist sehr hinderlich, funktioniert aber. Hier ein Beispiel für die Bearbeitung eines Fehlers mit VBScript in einer ASP-Seite:

```
<%
    On Error Resume Next
    Dim sProgID, CustomerManager
    sProgID = "MyWebDll.CCustomerManager"
    Set CustomerManager = Server.CreateObject(sProgID)
    Response.Write CustomerManager.GetCustomerListHTML()
    ' Feststellen, ob ein Aufruf auf einen Laufzeitfehler gestoßen ist.
    If Err.Number <> 0 Then
        Response.Write "<b>Es gab einen Systemfehler</b><br>"
        Response.Write Err.Description
    End If
%>
```

Lebenszyklusanforderungen für Apartment-Threaded-Objekte

Wie Sie wissen, können Sie Visual Basic verwenden, um Apartment-Threaded-Komponenten zu erstellen, die im STA-Modell ausgeführt werden. Wenn Sie eine Komponente generieren möchten, die ein weiterentwickeltes Threading-Modell unterstützt, müssen Sie andere Sprachen, wie zum Beispiel C++ benutzen. Die guten Nachrichten für Visual Basic-Programmierer lauten, dass dies zu keiner besonderen Einschränkung in einer IIS-Applikation führt, da der ASP-Thread-Pool auf STA-Threads basiert. Eine ASP-Seite und das von ihr erzeugte Visual Basic-Objekt werden häufig im selben Thread ausgeführt. Sie werden deshalb nicht mit einer aufwendigen COM-Proxy-/Stub-Schicht konfrontiert.

Wenn Sie eine ActiveX-DLL für eine IIS-Applikation erstellen, sollten Sie darauf achten, dass die *Threading-Modell*-Einstellung Ihres Projekts den Vorgabewert *Apartment-Threaded* verwendet. Ändern Sie den Wert nicht auf *Single-Threaded*. Ihre Objekte müssen andernfalls im Haupt-STA des IIS-Webserverprozesses ausgeführt werden. Die Objekte werden in diesem Fall außerdem auf einen anderen Thread als die ASP-Seite, die sie erstellt hat, geladen. Dies führt zu einer Leistungsminderung, da eine unnötige Proxy-/Stub-Schicht erstellt wird. Single-Threaded-DLLs sind außerdem nicht empfehlenswert, da sie zu Blockierungen führen.

Denken Sie auch daran, dass Visual Basic-Objekte eine Thread-Affinität aufweisen. Dies bedeutet, dass ein Visual Basic-Objekt nur von dem Thread verwendet werden kann, der das Objekt erstellt hat. Dies ist kein Problem, wenn Sie ein Objekt innerhalb des Gültigkeitsbereichs einer HTTP-Anforderung erstellen oder freigeben. Wenn Sie jedoch versuchen, ein Visual Basic-Objekt über mehrere Anforderungen hinweg aufrechtzuerhalten, indem sie es einer ASP-Session-Variablen zuweisen, führt dies dazu, dass der Client fest mit einem spezifischen Arbeits-Thread im ASP-Thread-Pool verbunden wird.

Um zu demonstrieren, wieso diese Situation nicht empfehlenswert ist, werden wir ein Beispiel betrachten, das die Aspekte nennt, an die die ASP-Designer denken mussten, als sie ihr Gerüst erstellten. Stellen Sie sich eine Anforderung von einem Client vor, die ein Objekt erstellt und einer ASP-Session-Variablen zuweist. Das Objekt wird auf einem der STA-Threads aus dem ASP-Thread-Pool generiert, zum Beispiel auf Thread 17. Was geschieht nun, wenn die nächste Anforderung von demselben Client auf Thread 8 bearbeitet wird? Ein Problem tritt auf, da Thread 8 nicht auf ein Apartment-Threaded-Objekt zugreifen kann, das auf Thread 17 erstellt wurde. Die Anforderung müsste von Thread 8 zu Thread 17 wechseln, um auf das Objekt zugreifen zu können.

Die ASP-Designer mussten zwischen zwei Übeln wählen. Sie entschieden, dass die feste Verbindung von Clients mit Threads der Situation vorzuziehen sei, in der eine einzelne Anforderung einen Zugriff auf mehrere Threads aus dem von ASP verwalteten Pool verlangt. Sie fügten Code zur ASP-Laufzeit hinzu, um zu ermitteln, wann ein Apartment-Threaded-Objekt einer ASP-Session-Variablen zugewiesen wird. Sobald einer Client-Sitzung ein Apartment-Threaded-Objekt zugewiesen wird, leitet die ASP-Laufzeit während der Sitzung jede Anforderung über denselben STA-Arbeits-Thread weiter. Dies ist keine glückliche Lösung, da das ASP-Thread-Pool-Schema nur dann optimal arbeitet, wenn es den ersten verfügbaren Thread im Pool benutzen kann.

Das ASP-Thread-Pool-Modell leitet jeden Thread in den Pool, um eine eingehende ASP-Anforderung bedienen zu können. Wenn Sie jedoch einer ASP-Session-Variablen ein Visual Basic-Objekt zuweisen, muss ASP den Thread lokalisieren (und möglicherweise blockieren), der das Objekt erstellt hat. Die ASP-Laufzeit kann nun alle zukünftigen Anforderungen über denselben Thread serialisieren. Diese Situation gestattet einer Applikation jedoch nicht, ihr Potential hinsichtlich der Performance und Parallelität auszuschöpfen. Ein Client kann auch dann die Anfor-

derung eines anderen Clients blockieren, wenn sich mehrere STA-Arbeits-Threads des Pools im Leerlauf befinden.

Die wichtige Schlussfolgerung, die sich hieraus ergibt, lautet, dass Visual Basic-Objekte lediglich im Gültigkeitsbereich einer Seite verwendet werden sollten. Versuchen Sie niemals, ein Visual Basic-Objekt (oder ein anderes Apartment-Threaded-Objekt) mit einer ASP-Session-Variablen aufrechtzuerhalten, da Ihre Applikation in diesem Fall nicht korrekt skaliert werden könnte. Dies ist vielleicht möglich, wenn nur wenige Benutzer auf Ihre Site zugreifen. Sie sollten jedoch immer an die Folgen Ihrer Aktionen denken.

Sie kennen nun die mit Apartment-Threaded-Objekten und ASP-Session-Variablen verknüpften Probleme. Das Zuweisen eines Visual Basic-Objekts zu einer ASP-Application-Variable führt zu einem weitaus schwierigeren Problem. Wenn Sie einer ASP-Application-Variable ein Apartment-Threaded-Objekt zuweisen, müssen die Anforderungen vieler verschiedener Clients über einen einzelnen Thread weitergeleitet werden. (Es gibt zu diesem Zweck einen speziellen STA-Thread, so dass immer ein Thread-Wechsel durchgeführt wird und ein Konflikt auftritt.) Dies führt zu einem kritischen und nicht akzeptablen Blockierungsverhalten. Wenn Sie Ihre Visual Basic-Objekte aber am Ende jeder Anforderung freigeben, können Sie dieses unerwünschte Resultat vermeiden.

Webapplikationen in separate Prozesse unterteilen

IIS erlaubt Ihnen, Ihre ASP-Seiten im Webserverprozess (*INETINFO.EXE*) oder in einem isolierten Prozess ausführen zu lassen. Die Standard-Website und andere IIS-Applikationen besitzen eine konfigurierbare *Anwendungsschutz*-Einstellung, die bestimmt, ob sie in *INETINFO.EXE* oder in einem separaten, isolierten Prozess ausgeführt werden. Abbildung 9.3 zeigt, wie die *Anwendungsschutz*-Einstellung für eine IIS-Applikation mit dem Internetdienste-Manager gesetzt wird.

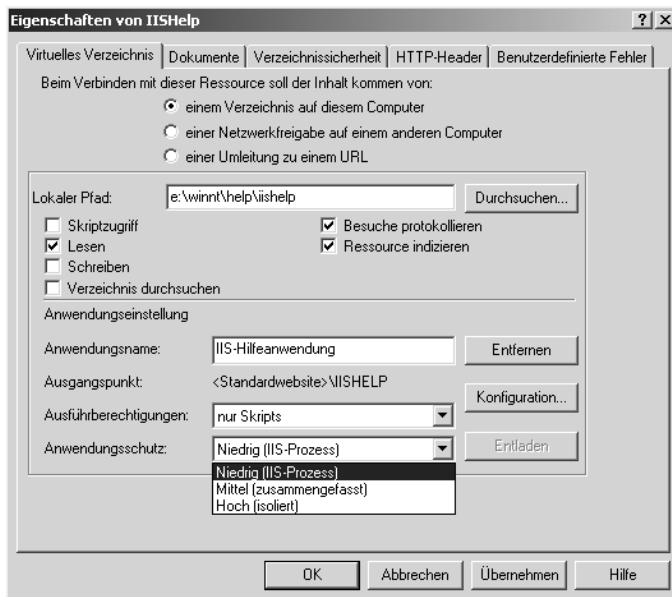


Abbildung 9.3: Sie können eine IIS-Applikation in *INETINFO.EXE* oder in einem isolierten Prozess ausführen lassen.

Um die bestmögliche Performance zu erzielen, sollten Sie Ihre ASP-Seiten und Visual Basic-Objekte in *INETINFO.EXE* ausführen lassen, wie in Abbildung 9.4 dargestellt. Sie lassen ASP-Seiten in *INETINFO.EXE* ausführen, indem Sie die *Anwendungsschutz*-Einstellung des Hauptverzeichnisses oder einer IIS-Applikation auf den Wert *Niedrig* (*IIS-Prozess*) setzen. Beachten Sie, dass dies nicht die Standardeinstellung für das Hauptverzeichnis oder eine IIS-Applikation in IIS 5 ist.

Wenn Sie Visual Basic-Objekte zusammen mit Ihren ASP-Seiten in *INETINFO.EXE* ausführen lassen möchten, können Sie Ihre Komponenten zu einer COM+-Bibliotheksapplikation hinzufügen. Sie erzielen denselben Effekt, wenn Sie Ihre DLLs mit *REGSVR32.EXE* registrieren und als nicht konfigurierte Komponenten ausführen lassen. Objekte, die aus nicht konfigurierten Komponenten erzeugt werden, sind etwas schneller als Objekte aus konfigurierten Komponenten. Der Grund hierfür besteht darin, dass sie keinen vom System zur Verfügung gestellten Proxy mit einem Abfangcode benötigen. Sie sollten jedoch daran denken, dass nicht konfigurierte Komponenten keine deklarativen Attribute besitzen und auch nicht mit der COM+-Laufzeit interagieren können.

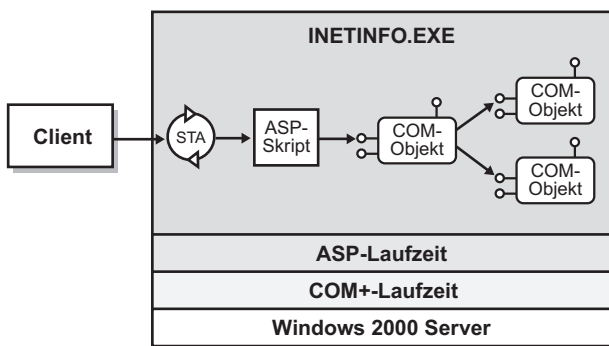


Abbildung 9.4: Das Ausführen Ihrer ASP-Seiten und Visual Basic-Objekte in *INETINFO.EXE* bietet die beste Performance.

Wenngleich das Ausführen Ihres Codes in *INETINFO.EXE* die beste Leistung bietet, könnten Sie andere Faktoren dazu verleiten, Ihre Visual Basic-Objekte und ASP-Seiten in separaten Prozessen ausführen zu lassen. Vielleicht sind Sie an einer höheren Fehlertoleranz interessiert. Möglicherweise möchten Sie Ihre Visual Basic-Komponenten auch auf einem separaten Computer ausführen lassen, um eine sichere Umgebung zu schaffen oder damit verschiedene Webserver mehrere Geschäftsobjekte gemeinsam nutzen können. Wenn Sie eine Applikation in mehrere Prozesse aufteilen, führt dies in der Regel zu einer geringeren Performance.

Die beiden primären Gründe für Unternehmen, Visual Basic-Code nicht in *INETINFO.EXE* zu verwenden, betreffen die Fehlertoleranz und die Aktualisierung des Applikations-Codes. Die Fehlertoleranz ist ein Grund, da der Code in einer Visual Basic-DLL *INETINFO.EXE* abstürzen lassen kann. Wenn Sie beispielsweise eine Win32-API-Funktion aufrufen, die Zeiger nicht korrekt bearbeitet, können Sie eine vollständige Website abstürzen lassen. Jede Visual Basic-DLL, die *Declare*-Anweisungen und API-Aufrufe enthält, muss umfassend getestet werden, bevor sie in Betrieb genommen wird.

Sie sollten beachten, dass viele Entwickler und Administratoren die Möglichkeit, dass Visual Basic *INETINFO.EXE* zum Absturz bringen kann, überbewerten. Wenn eine ActiveX-DLL einen Standard-Laufzeitfehler nicht abfangen kann, wird der Fehler einfach an die ASP-Seite zurückgegeben, die das Objekt erstellt hat. Wenn die ASP-Seite den Fehler nicht abfängt, meldet die ASP-Laufzeit den Fehler an den Browser. Wenngleich nicht bearbeitete Fehler kein Aushänge-

schild für eine Applikation sind, kann Visual Basic-Code in der Regel sicher und ohne das Risiko eines Absturzes in *INETINFO.EXE* ausgeführt werden.

Der zweite Grund für den Verzicht auf Visual Basic-Code in *INETINFO.EXE* betrifft die Aktualisierung von Applikationscode. Wenn eine Visual Basic-DLL in *INETINFO.EXE* geladen wird, muss der Webserverprozess heruntergefahren werden, bevor die DLL erneut erstellt und ersetzt wird. Einige Unternehmen glauben, dass das Herunterfahren von *INETINFO.EXE* nicht akzeptabel für einen Betriebsserver ist. Wir werden später in diesem Kapitel einige Optionen erörtern, die Ihnen gestatten, eine Visual Basic-DLL im laufenden Betrieb zu ersetzen, ohne den Webserverprozess herunterzufahren. Während der Entwicklung werden Sie ebenfalls darauf verzichten wollen, DLLs in *INETINFO.EXE* zu laden, da dies das erneute Kompilieren und das Testen Ihres Codes erschwert.

Ihnen stehen verschiedene Möglichkeiten zur Verfügung, um Ihren Visual Basic-Code in einem separaten Prozess ausführen zu lassen. Die erste Lösung besteht darin, Ihre Visual Basic-Komponenten in einer COM+-Serverapplikation zu konfigurieren, wie in Abbildung 9.5 dargestellt. Ihre Visual Basic-Objekte werden dann in einer Instanz der COM+-Containerapplikation *DLLHOST.EXE* erstellt. Jeder Aufruf, der aus einer ASP-Seite heraus an ein Objekt erfolgt, wird mit Hilfe eines Proxy-/Stub-Paares über die Prozessgrenzen hinweg ausgeführt.

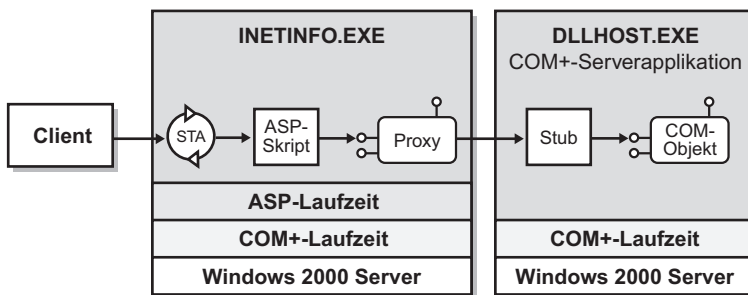


Abbildung 9.5: Wenn eine ASP-Seite ein Objekt aus einer Komponente erstellt, die in einer COM+-Serverapplikation konfiguriert ist, werden die ASP-Seite und das Objekt über ein Proxy-/Stub-Paar miteinander verbunden.

Diese Lösung vereinfacht das erneute Erstellen einer DLL während der Entwicklung. Nachdem Sie Ihre DLL getestet haben, können Sie die COM+-Serverapplikation herunterfahren und die DLL neu erstellen. Sie können sogar zwischen der Ausführung Ihrer Objekte in *DLLHOST.EXE* und im Visual Basic-Debugger wechseln. Wenn Sie Ihre Komponenten für das MTS-/COM+-Debugging eingerichtet haben (wie im sechsten Kapitel beschrieben), können Sie den Code eines Visual Basic-Objekts, das aus einer ASP-Seite in *INETINFO.EXE* instanziiert wurde, im Einzelschrittmodus ausführen lassen. Wenn Sie Visual Basic-Objekte debuggen, die aus ASP-Seiten erzeugt wurden, sollten Sie den folgenden Hinweis lesen.

HINWEIS: Die aus ASP-Seiten erzeugten Visual Basic-Komponenten debuggen

Visual Basic-Entwickler können mit einem Problem konfrontiert werden, wenn sie zum ersten Mal versuchen, Visual Basic-Objekte zu debuggen, die aus ASP-Seiten unter Windows 2000 erstellt wurden. Wenn eine ASP-Seite versucht, ein Objekt im Visual Basic-Debugger zu erzeugen, meldet die ASP-Laufzeit den Fehler »*The call to Server.CreateObject failed while checking permissions. Access is denied to this object.*«. Die Sicherheit wird im elften Kapitel eingehend beschrieben. Ich möchte dieses Problem jedoch an dieser Stelle erläutern, damit Sie

Ihre Arbeit nicht unterbrechen müssen, wenn Sie Visual Basic-Komponenten für eine IIS-Applikation debuggen.

Eine IIS-Applikation verfügt über eine Standardsicherheitseinstellung, die den anonymen Zugriff erlaubt. Wenn ein nicht authentifizierter Client eine ASP-Anforderung versendet, wird die ASP-Seite als anonymes Benutzerkonto unter dem *IUSR_MachineName*-Konto ausgeführt. Der mit dem Visual Basic-Debugger (*VB6.EXE*) verknüpfte Prozess benutzt Standardzugriffsberechtigungen, die für alle Rechner konfiguriert werden. Das *IUSR_MachineName*-Konto kann so lange keinen Aufruf an *VB6.EXE* durchführen, bis es eine maschinenweit gültige Standardzugriffsberechtigung erhält. Sie modifizieren die Sicherheitsberechtigungen mit einem Hilfsmittel namens *DCOMCNFG.EXE*. Wenn Sie dieses Programm von der Befehlszeile aus starten, können Sie die maschinenweit gültigen Standardzugriffsberechtigungen auf der Registerkarte *Standardsicherheit* ändern. *DCOMCNFG.EXE* stellt Ihnen einen Windows-Standard-Editor zur Bearbeitung von Zugriffssteuerungslisten (ACL – Access Control List) zur Verfügung, mit dem Sie Zugriffsberechtigungen an *IUSR_MachineName* oder an alle anderen Benutzerkonten vergeben. Denken Sie daran, dass Sie auf diese Weise ein potentiell Sicherheitsloch öffnen, das für eine Entwicklungs-Workstation möglicherweise akzeptabler als für einen Betriebsserver ist.

Eine weitere Möglichkeit, dieses Sicherheitsproblem zu lösen, besteht darin, den anonymen Zugriff auf die Standard-Website mit dem Internetdienste-Manager zu sperren und während des Debuggens die integrierte Windows-Authentifizierung zu verwenden. In diesem Fall werden die ASP-Seiten nicht als *IUSR_MachineName*, sondern unter dem Konto des interaktiven Benutzers ausgeführt. Der Browser wird ebenfalls unter diesem Konto ausgeführt. Wenn Sie das gesamte Debuggen auf einer einzelnen Maschine durchführen, wird auch der Visual Basic-Debugger unter dem Konto des interaktiven Benutzers ausgeführt. Die COM-Sicherheitsüberprüfungen gestatten den ASP-Seiten somit, Objekte in *VB6.EXE* zu erstellen und mit diesen zu interagieren.

Beachten Sie, dass das in diesem Hinweis beschriebene Problem nur dann relevant ist, wenn Sie eine ActiveX-DLL in der Visual Basic-IDE debuggen. Visual Basic-Objekte, die in einer COM+-Applikation ausgeführt werden, sind von diesem Problem nicht betroffen.

Ein weiterer guter Grund, Ihre Komponenten zu einer COM+-Serverapplikation hinzuzufügen, besteht darin, Objekte auf einem separaten Computer ausführen zu lassen, der als Applikationsserver vorgesehen ist. Sie könnten dies beispielsweise tun, wenn Sie einen einzelnen Applikationsserver benutzen möchten, der von vielen verschiedenen Webservern verwendet wird. Vielleicht möchten Sie auch die Arbeitslast der Applikation auf mehrere Computer verteilen oder Ihre Geschäftsobjekte auf einem Computer ausführen lassen, der sicherer als ein Webserver ist. Sobald Sie Ihre Visual Basic-Komponenten in eine COM+-Serverapplikation auf dem Applikationsserver-Computer installiert haben, ist es sehr einfach, einen Applikationsproxy zu erstellen und diesen auf jedem Webserver zu installieren, der auf den Proxy zugreifen muss. ASP-Seiten können anschließend Objekte über das Netzwerk erzeugen.

Wie Sie wissen, sind prozesseexterne Aufrufe sehr aufwendig. Aufrufe, die die Computergrenzen verlassen, sind noch aufwendiger. Wenn Ihre ASP-Seiten Objekte aufrufen sollen, die in anderen Prozessen vorhanden sind, müssen Sie Ihre Komponenten so entwerfen, dass die Anzahl der Übertragungen über das Netzwerk minimiert wird. Dies ist ein wichtiger Aspekt der Entwicklung von verteilten Applikationen.

Isolierte IIS-Applikationen

Sie haben eine Möglichkeit kennen gelernt, Ihre Visual Basic-Objekte in einem separaten Prozess außerhalb von *INETINFO.EXE* ausführen zu lassen. IIS gestattet Ihnen ebenfalls, ASP-Seiten außerhalb von *INETINFO.EXE* ausführen zu lassen. Wenn Sie die *Anwendungsschutz*-Einstellung einer IIS-Applikation auf *Mittel (zusammengefasst)* oder *Hoch (isoliert)* setzen, werden Ihre ASP-Seiten in einer Instanz der COM+-Containerapplikation *DLLHOST.EXE* ausgeführt.

Abbildung 9.6 zeigt was geschieht, wenn Sie die *Anwendungsschutz*-Einstellung einer IIS-Applikation auf *Hoch (isoliert)* setzen. IIS muss einen kleinen Trick anwenden, um ASP-Seiten in einem isolierten Prozess ausführen zu können. Wenn Sie die Einstellung auf *Hoch (isoliert)* setzen, erstellt IIS eine neue COM+-Serverapplikation mit einer speziellen WAM-Komponente. Sie können dies, nachdem Sie die Einstellung geändert haben, überprüfen, indem Sie den Ordner *COM+-Anwendungen* im administrativen Hilfsmittel *Komponentendienste* aktualisieren und dessen Inhalt betrachten.

Wenn IIS die erste Anforderung für die isolierte IIS-Applikation erhält, erstellt das *W3SVC* WAM-Objekt in *INETINFO.EXE* eine Instanz dieser speziellen WAM-Komponente. Dies zwingt die COM+-Laufzeit, eine neue Instanz von *DLLHOST.EXE* zu starten. Danach werden alle Anforderungen für die isolierte IIS-Applikation über diese beiden WAM-Objekte weitergeleitet. Dies führt dazu, dass alle mit der isolierten IIS-Applikation verknüpften ASP-Seiten in einem eigenen privaten Prozess ausgeführt werden.

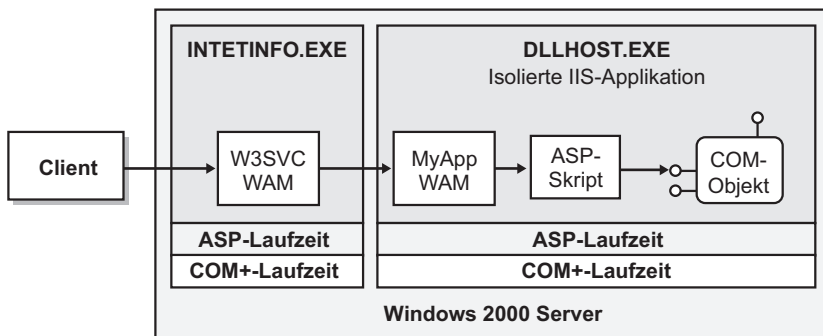


Abbildung 9.6: Wenn eine IIS-Applikation die *Anwendungsschutz*-Einstellung *Hoch (isoliert)* benutzt, wird sie in einer privaten Instanz von *DLLHOST.EXE* ausgeführt.

Denken Sie daran, dass Leistungseinbußen für jede Anforderung bestehen, da ein prozessexterner COM-Aufruf von einem WAM-Objekt an ein anderes WAM-Objekt notwendig ist. Im Gegensatz zur in Abbildung 9.5 dargestellten Konfiguration, müssen Aufrufe von ASP-Seiten an Visual Basic-Objekte jedoch nicht die Prozessgrenzen verlassen. Solange Ihre Komponenten in Bibliotheksapplikation angeordnet oder als nicht konfigurierte Komponenten registriert sind, werden Ihre Objekte in demselben Prozess und auf demselben Thread wie die ASP-Seite erstellt, die die Objekte erzeugt. Solch ein Modell bietet auch dann eine akzeptable Leistung, wenn eine ASP-Seite mehrere Aufrufe an ein Objekt vornimmt.

Das letzte Konfigurationsschema ist einfach eine geringfügige Variation des bereits vorgestellten Schemas. Wenn eine IIS-Applikation mit der *Anwendungsschutz*-Einstellung *Mittel (zusammengefasst)* ausgeführt wird, befindet sie sich zusammen mit jeder anderen IIS-Applikation, die dieselbe Einstellung aufweist, in einer gemeinsam genutzten Instanz von *DLLHOST.EXE*. Beachten Sie, dass *Mittel (zusammengefasst)* die Standardeinstellung für das Hauptverzeichnis

und jede neue IIS-Applikation ist. Abbildung 9.7 zeigt, wie Anforderungen an die entsprechenden Ziele weitergeleitet werden.

Eine Motivation für die isolierte Ausführung von IIS-Applikationen betrifft die Sicherheit. *INETINFO.EXE* wird als Windows-Dienst unter der Identität des Systems ausgeführt. Isolierte IIS-Applikationen werden jedoch unter der Identität eines Benutzers mit eingeschränkten Rechten ausgeführt. Dies bedeutet, dass die Isolation einer IIS-Applikation das Risiko von Angriffen auf Ihre Website verringern kann. Dieses Thema wird im elften Kapitel ausführlich besprochen, wenn von der IIS-Sicherheit die Rede ist.

Sie haben nicht wenige Konfigurationsszenarien kennen gelernt. Sie können all Ihre ASP-Seiten und Visual Basic-Objekte in *INETINFO.EXE* ausführen. Sie können Ihre ASP-Seiten in *INETINFO.EXE* und Ihre Visual Basic-Objekte in einer COM+-Serverapplikation ausführen. Und Sie können Ihre ASP-Seiten und Objekte außerhalb von *INETINFO.EXE* ausführen lassen. Sie verfügen somit über die Möglichkeit, Ihre Webapplikationen in jeder sinnvollen Weise zu unterteilen.

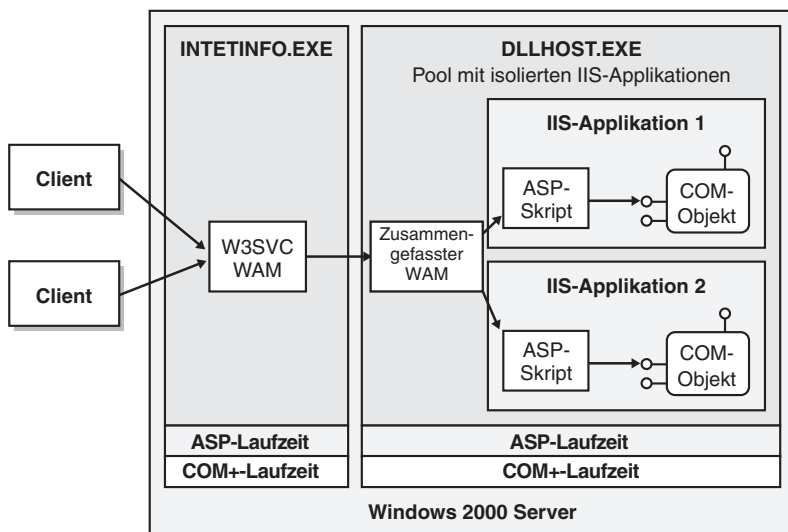


Abbildung 9.7: Jede IIS-Applikation, deren Anwendungsschutz-Einstellung auf Mittel (zusammengefasst) gesetzt ist, wird in einer gemeinsam genutzten Instanz von *DLLHOST.EXE* ausgeführt.

Das Besondere dieser Konfigurationsoptionen ist, dass sie sich nicht unbedingt darauf auswirken, wie Sie Ihre ASP-Seiten oder Ihre Visual Basic-Komponenten schreiben. (Die einzige Ausnahme ist das Erstellen von Visual Basic-Komponenten, die aus Gründen der Performance eine prozessinterne Abhängigkeit aufweisen.) Wenn Sie Ihre Applikation fertig gestellt haben, können Sie von einem Konfigurationsschema zu einem anderen wechseln, ohne Code umschreiben zu müssen. Sie können einfach die *Anwendungsschutz*-Einstellung einer IIS-Applikation oder die Aktivierungseinstellung einer COM+-Applikation ändern.

Sie haben zum Beispiel die Möglichkeit, Ihren Code in der Entwicklungsumgebung zu debuggen, indem Sie Ihre Visual Basic-Komponenten in einer COM+-Serverapplikation ausführen lassen. Dies vereinfacht das Entfernen Ihrer DLLs aus dem Speicher und deren erneute Kompilierung. Wenn die Zeit gekommen ist, Ihren Code in die Betriebsumgebung zu übernehmen, können Sie Ihre Site so konfigurieren, dass Ihre ASP-Seiten und Visual Basic-Objekte in *INETINFO.EXE* ausgeführt werden, um die bestmögliche Performance zu erzielen.

Code zwischen ASP-Seiten und Komponenten aufteilen

Nachdem wir nun die Grundlagen der Entwicklung und Konfiguration von Visual Basic-Komponenten für eine IIS-Applikation besprochen haben, werden wir uns nun einigen wichtigen Faktoren zuwenden, die sich darauf auswirken, wie Sie eine Applikation für IIS entwickeln. Es ist sehr wichtig, dass Sie entscheiden, wie Sie Ihren Applikationscode in ASP-Seiten und Visual Basic-Komponenten unterteilen.

Wenn Sie die komplexe Architektur einer neuen Webapplikation entwerfen, können Sie Ihre Applikation gewöhnlich in drei logische Schichten unterteilen. Sie besitzen Datenzugriffscode, Geschäftslogik und Benutzeroberflächencode. In der ersten Design-Phase sollten Sie entscheiden, wie und wo Ihr Code für diese Schichten aufbereitet werden soll.

Für die meisten von uns sollte die Entscheidung über den Aufbewahrungsort des Datenzugriffs-codes und der Geschäftslogik sehr einfach sein. Beide sollten in Komponenten erstellt und in COM-DLLs verteilt werden. Die Komponenten sollten anschließend in COM+-Applikationen oder als nicht konfigurierte Komponenten eingerichtet werden. Sobald sich die Komponenten, die Ihre Geschäftslogik und Ihren Datenzugriffscode enthalten, am vorgesehenen Ort befinden, können die für die Benutzeroberfläche verantwortlichen Entwickler Desktop- und Webapplikationen erstellen. Die Client-Applikationen, die die Benutzeroberfläche definieren, verwenden Ihre Geschäftsobjekte als Einstiegspunkt in die IT-Infrastruktur Ihres Unternehmens.

Das Erstellen einer großen Webapplikation erfordert Fachkenntnisse über viele verschiedene Technologien. Eine große Zahl dieser Technolgoien bezieht sich auf die Präsentationsschicht. In diesem Buch wird nicht beschrieben, wie eine hochentwickelte und interaktive Benutzeroberfläche generiert wird (ein Vorhaben, dass ein umfassendes Wissen über DHTML und JavaScript erfordert). Wir konzentrieren uns stattdesse auf den Code, der in IIS ausgeführt wird und der den Benutzeroberflächen-Anteil zurück zum Browser schickt.

Wie Sie wissen, sind ASP-Seiten sehr wichtig, da sie Ihnen gestatten, serverseitige Skripts auszuführen, die Ihnen wiederum erlauben, COM-Objekte als Reaktion auf Client-Anforderungen zu aktivieren. ASP-Seiten werden ebenfalls häufig dazu verwendet, den Inhalt, die Layoutdetails und den Benutzeroberflächencode von Webseiten aufzunehmen. Doch wie viele Elemente sollten Sie in ASP-Seiten anordnen? Die Antwort ist wie so oft von verschiedenen Faktoren abhängig.

Einige Webentwickler definieren gerne jeden Aspekt der Benutzeroberfläche innerhalb von ASP-Seiten. Andere unternehmen jede Anstrengung, um den Inhalt ihrer ASP-Seiten zu minimieren. Viele Entwickler bevorzugen den gemischten Einsatz von ASP-Seiten und Komponenten, um HTML dynamisch zu generieren. Für welche Lösung Sie sich auch entscheiden, ein Teil Ihrer Applikation muss für die Definition der Benutzeroberfläche verantwortlich sein.

Jede nicht triviale Webapplikation muss die folgenden Aspekte berücksichtigen: Sie muss Eingaben akzeptieren, die der Benutzer in ein HTML-Formular eingibt. Sie muss Benutzeranforderungen interpretieren und die entsprechenden Anweisungen ausführen. Sie muss außerdem HTML-Code generieren, der Seiteninhalte und Layoutdetails enthält. Dieser HTML-Code muss zurück zum Browser gesendet werden.

Lassen Sie uns einige Möglichkeiten zum Erstellen des Codes für die Präsentationsschicht untersuchen. Jede Lösung bietet Vor- und Nachteile hinsichtlich der Produktivität, Wiederverwendbarkeit, Wartbarkeit, Erweiterbarkeit und Performance. Unabhängig davon, für welche Möglichkeit Sie sich entscheiden, müssen Sie immer Code für die integrierten ASP-Objekte (wie zum Beispiel

Request und Response) schreiben und Geschäftsobjekte erstellen sowie ausführen. Sie müssen außerdem HTML-Code für den Seiteninhalt und das Layout generieren und diesen an den Browser zurückschicken.

Doch wie entscheiden Sie, ob der Code der Präsentationsschicht in einer ASP-Seite, in kompiliertem Code oder sowohl in einer ASP-Seite als auch in kompiliertem Code angeordnet werden soll? Stellen Sie sich dazu die folgenden Fragen: Wie schnell möchten Sie Ihre Site aufbauen und ausführen lassen? Was ist Ihnen wichtiger: die Entwicklungsproduktivität oder die Applikationsperformance? Wie wichtig ist die Wiederverwendbarkeit von Code? Sind Sie bereit, mehr Zeit zu Beginn des Projekts zu investieren, um die Wartbarkeit und Erweiterbarkeit Ihrer Site zu verbessern? Möchten Sie in der Lage sein, verschiedene Aspekte Ihrer Site zu ändern, ohne Code erneut kompilieren zu müssen? Ihre Antworten auf diese Fragen werden Sie zu einer Entscheidung führen.

Die Benutzeroberfläche mit ASP-Seiten erstellen

Das Einbetten des Codes Ihrer Präsentationsschicht in eine ASP-Seite wird mit einem Hilfsmittel wie zum Beispiel Microsoft Visual InterDev durchgeführt und bietet gewisse Vor- und Nachteile. Positiv ist, dass Sie das Layout der Webseiten schnell erstellen und modifizieren können. Sie können serverseitige Skripts schreiben, die ASP-Objekte benutzen, wie zum Beispiel Request, Response, Session und Application. Sie können außerdem benutzerdefinierte Geschäftsobjekte erstellen und ausführen lassen. Und schließlich können Sie Code und statischen HTML-Code hinzufügen, um Seiten zu definieren, die an den Browser zurückgeschickt werden.

Andererseits kann die Visual InterDev-Lösung schwierig zu verwalten sein, wenn Sie große Websites entwickeln. Dies gilt besonders, wenn die Site häufig verändert wird. Da Ihr gesamter Code für die Präsentationsschicht in Textdateien definiert und eingerichtet ist, kann die Suche nach effektiven Wiederverwendungstechniken sehr kompliziert werden. Sie verfügen außerdem über keinen objektorientierten Mechanismus, mit dem Sie verschiedene Bestandteile Ihrer Logik von anderen Abschnitten getrennt kapseln können. Jede Seite kann stattdessen zu einer Mixtur aus Seiteninhalten, Layoutdetails, Code für die Bearbeitung von Benutzereingaben und Aufrufen an Geschäftsobjekte werden.

Die übliche Technik des Visual InterDev-Entwicklers besteht darin, *Save As* für eine .ASP-Datei aufzurufen und die Kopie anschließend anzupassen. Diese Lösung vereinfacht das Verändern einer Seite der Site. Größere Änderungen, die die gesamte Site betreffen, können jedoch schwieriger sein. Stellen Sie sich zum Beispiel vor, Sie müssten das Layout einer allgemein genutzten Tabellenauswertung leicht verändern. Wenn Sie viele .ASP-Dateien mit redundanten Attributseinstellungen für diese Auswertung besitzen, müssen Sie jede Seite modifizieren, um die Änderung für die gesamte Site durchführen zu können. Wie Sie sehen, schadet dieses Problem der Wartbarkeit Ihrer Site.

Kleine Websites werden möglicherweise nicht mit diesem Problem konfrontiert. Für große Sites ist es jedoch häufig erforderlich, viele .ASP-Dateien zu ändern, um ein neues Feature hinzuzufügen oder das Verhalten eines clientseitigen Skripts zu modifizieren. Diese Lösung kann somit die Erweiterbarkeit und die Wartbarkeit beeinträchtigen.

Ein weiterer Nachteil großer ASP-Seiten besteht darin, dass sie langsam sein können. Diese Seiten können aufgrund der interpretativen Natur eines Skripts nicht schneller als kompilierter Code ausgeführt werden. Jeder Aufruf, den ein ASP-Skript an ein COM-Objekt vornimmt, wird außerdem über IDispatch geleitet und verwendet die späte Bindung. Dies bedeutet, dass Aufrufe an Geschäftsobjekte und an die integrierten ASP-Objekte langsamer als die gleichen Aufrufe sind,

die aus einer kompilierten Visual Basic-DLL heraus initiiert werden. Die mit der späten Bindung verknüpfte Leistungseinbuße kann signifikant sein, wenn eine einzelne ASP-Seite hunderte oder tausende von COM-Aufrufen durchführt.

Die Benutzeroberfläche mit kompilierten Komponenten erstellen

Wenn Sie einen Teil oder die gesamte Benutzeroberfläche einer Applikation mit Visual Basic-Komponenten generieren möchten, sollten Sie in Ihrem Programm die ASP-Objekte verwenden. Beachten Sie bitte, dass Ihnen in einer ActiveX-DLL beinahe die gleichen Möglichkeiten wie in einer ASP-Seite zur Verfügung stehen. Sie können deshalb Komponenten schreiben, die Werte aus Abfragezeichenfolgen und HTML-Formulareingabefeldern abrufen. Dies geschieht mit dem Request-Objekt. Sie benutzen das Response-Objekt, um HTML-Code und clientseitige Skripte zu erstellen, die an den Browser zurückgeschickt werden. Wenn Sie mit dem ASP-Objektmodell vertraut sind, wissen Sie, dass Ihnen sehr viele Möglichkeiten zur Verfügung stehen.

Lassen Sie uns betrachten, wie Sie mit Ihren Visual Basic-Komponenten direkt auf die integrierten ASP-Objekte zugreifen können. Sie müssen zunächst sowohl die Typbibliothek *COM+ Services Type Library* als auch die Typbibliothek *Active Server Pages Object Library* in Ihrem ActiveX-DLL-Projekt referenzieren. Sie rufen dann die ASP-Objektreferenzen über den Objektnamen aus der Item-Auflistung derObjectContext-Schnittstelle ab. Zu den ASP-Objekten, die in der Item-Auflistung verfügbar sind, zählen Request, Response, Session, Application und Server.

Werfen Sie einen Blick auf die folgende Methodenimplementierung:

```
Sub MyMethodCalledFromASP()  
    Dim rsp As Response  
    Set rsp = GetObjectContext.Item("Response")  
    rsp.Write "Hello World!"  
End Sub
```

Dieser Code wird aufgrund der Integration zwischen der ASP-Laufzeit und der COM+-Laufzeit korrekt ausgeführt. Wenn ein serverseitiges Skript in einer ASP-Seite ein Objekt aus einer konfigurierten Komponente erzeugt, übergibt die ASP-Laufzeit die Verweise auf die fünf integrierten ASP-Objekte an die COM+-Laufzeit. Die COM+-Laufzeit verwendet diese Referenzen, um die Item-Auflistung zu füllen. Dies geschieht, wenn sie den Kontext des neuen Objekts initialisierten. Das ist ein weiteres Beispiel dafür, wie die COM+-Laufzeitumgebung kontextuelle Informationen vom Erzeuger zu einem neu erstellten Objekt leitet. Sobald das Objekt initialisiert ist, stehen die ASP-Objekte für die Dauer der ASP-Anforderung zur Verfügung.

Jede konfigurierte Komponente besitzt ein *IISIntrinsics*-Attribut, das auf *True* gesetzt werden muss, damit ihre Objekte auf die integrierten ASP-Objekte zugreifen können. Der Standardwert dieses Attributs ist glücklicherweise *True*. (Die COM+-Dokumentation führt als Standardwert fälschlicherweise *False* auf.) Sie sollten auch beachten, dass Sie das *IISIntrinsics*-Attribut nicht mit dem administrativen Hilfsmittel *Komponentendienste* konfigurieren können. Sie haben lediglich die Möglichkeit, es programmgesteuert einzurichten. Sie müssen das Attribut natürlich nicht ändern, um den soeben gezeigten Code zu schreiben. Sie könnten jedoch eine Optimierung durchführen, indem Sie den Wert für konfigurierte Komponenten, die nicht auf ASP-Objekte zugreifen, auf *False* setzen.

Ich möchte Sie an dieser Stelle auf zwei wichtige Beobachtungen hinweisen, die Visual Basic-Komponenten betreffen, die wiederum integrierte ASP-Objekte benutzen: Erstens müssen Acti-

veX-DLLs, die die `ObjectContext`-Schnittstelle verwenden, um ASP-Referenzen abzurufen, auf die Typbibliothek *COM+ Services Type Library* zugreifen und sollten deshalb als konfigurierte Komponenten eingesetzt werden. Ein Visual Basic-Objekt muss zweitens immer dann einen Aufruf an die ASP-Laufzeit durchführen, wenn es auf ein ASP-Objekt zugreifen möchte. Wenn das Visual Basic-Objekt und die ASP-Seite, die das Objekt erstellt hat, in unterschiedlichen Prozessen existieren, ist dieser prozessexterne Aufruf sehr aufwendig.

Wichtig ist, dass eine Visual Basic-Komponente, die ASP-Objekte benutzt, in eine COM+-Bibliotheksapplikation und nicht in eine COM+-Serverapplikation installiert werden muss. Die Performance Ihres Codes ist andernfalls nicht akzeptabel. Sie sollten eine Komponente, die ASP-Objekte benutzt, während der Entwicklung, wenn die Performance nicht wichtig ist, in einer COM+-Serverapplikation oder dem Visual Basic-Debugger testen. Die Komponente sollte jedoch einer COM+-Bibliotheksapplikation hinzugefügt werden, sobald sich Ihr Code in der Betriebsumgebung befindet. Der Grund hierfür besteht darin, dass die Objekte der Komponente in diesem Fall im selben Prozess wie die ASP-Seite, die die Objekte erstellt hat, ausgeführt werden. Wie bereits erwähnt wurde, können Sie ASP-Seiten zusammen mit Visual Basic-Objekten in drei verschiedenen Umgebungen ausführen lassen: in *INETINFO.EXE*, in einer isolierten IIS-Applikation oder in der zusammengefassten IIS-Applikation.

Erstellen eines benutzerdefinierten Gerüsts für das Erzeugen einer Benutzeroberfläche

Wenn Ihr Ziel darin besteht, möglichst viel Logik innerhalb kompilierter Komponenten aufzubewahren, können Sie eine favorisierte Strategie von Entwicklern übernehmen, die Ihr »COM ist Liebe«-T-Shirt jeden Tag tragen. Diese Programmierer glauben, dass die gesamte Applikationslogik in COM-DLLs zusammengefasst werden sollte. Lassen Sie mich solche Entwickler zum Zwecke dieser Diskussion als COM-Fanatiker bezeichnen. Diese COM-Fanatiker glauben, dass ASP-Seiten böse sind, da sie nicht in einer kompilierten, binären Form vorliegen. COM-Fanatiker unternehmen enorme Anstrengungen, um die Inhalte Ihrer ASP-Seiten zu minimieren. Sie haben eine Möglichkeit entdeckt, eine vollständige Website auf eine einzelne ASP-Seite zu beschränken:

```
<%  
    Dim ProgID, req  
    ProgID = "WebFramework.ClientRequest"  
    Set req = Server.CreateObject(ProgID)  
    req.ProcessRequest  
%>
```

Stellen Sie sich vor, dass dieses ASP-Skript eine benutzerdefinierte Komponente namens `ClientRequest` verwendet, die sich in einem ActiveX-DLL-Projekt mit der Bezeichnung `WebFramework` befindet. Die Komponente bietet nur eine öffentliche Methode namens `ProcessRequest` an. Diese Methode ist der einzige Einstiegspunkt in eine Webapplikation, die auf dem benutzerdefinierten Gerüst unserer COM-Fanatiker beruht. Die `ProcessRequest`-Methode bedient in diesem Gerüst jede Client-Anforderung. Werfen Sie einen Blick auf den folgenden Pseudo-Code:

```
Sub ProcessRequest()  
    ' Ermitteln, welche Anweisung der Benutzer wünscht.  
    ' Geschäftsobjekte instanzieren und aufrufen.  
    ' HTML mit Inhalt und Layout erstellen.  
    ' HTML zurück zum Browser schicken.  
End Sub
```


Der COM-Fanatiker muss ein Applikationsdesign kreieren, dass dem Client erlaubt, jede Anweisung auszuführen, die die Applikation anbietet. Alle Anforderungen treffen jedoch über nur einen Einstiegsunkt ein. Wie stellt die `ProcessRequest`-Methode also fest, welche Anweisung ausgeführt werden soll?

Eine beliebte Technik besteht darin, die Anweisungspräferenz des Clients in einer Abfragezeichenfolge zu übergeben. Dies erfordert jedoch zusätzliche Arbeit. Sie müssen die Abfragezeichenfolge an Ihre URLs anhängen, wenn Sie sie dynamisch in Ihre Seiten schreiben. Sie können Ihre URLs zum Beispiel wie folgt angeben:

```
MyPage.asp?Command=GetCustomerList
```

Listing 9.1 ist eine einfache Implementierung von `ProcessRequest`. Dieser Code interpretiert die Client-Anforderung, indem er den Anweisungswert aus der Abfragezeichenfolge extrahiert. Das Beispiel zeigt außerdem eine einfache Code-Strukturierung, die HTML generiert und als Antwort zurück zum Browser überträgt. Dies ist ein möglicher Ausgangspunkt für einen COM-Fanatiker, der ein Gerüst erstellen möchte. Unternehmen versuchen in der Regel eine Lösung wie diese zu realisieren, wenn sie ein benutzerdefiniertes Web-Gerüst zusammenstellen. Sie möchten auf diese Weise eine bessere Wiederverwendbarkeit, Wartbarkeit und Erweiterbarkeit erzielen. Das Erstellen eines Gerüsts, das diesen Anforderungen gerecht wird, bedingt jedoch eine signifikante Investition zu Beginn eines Projekts. Sie werden außerdem mit vielen anderen Design-Problemen konfrontiert, wenn Sie sich für diese Lösung entscheiden.

```
Sub ProcessRequest()  
    Dim req As Request  
    Set req = GetObjectContext("Request")  
    Select Case req("Command")  
        Case "", "GetStartPage"  
            GetStartPage  
        Case "GetCustomerList"  
            GetCustomersList  
        Case Else  
            ' Hier auf unbekannte Anweisung reagieren.  
    End Select  
    Set req = Nothing  
End Sub  
  
Private Sub GetStartPage()  
    ' HTML-Code für Startseite generieren.  
End Sub  
  
Private Sub GetCustomerList()  
    ' Geschäftsobjekt erstellen.  
    ' Abfrage ausführen, um Kundendaten abzurufen.  
    ' Formatierten HTML-Code für Kundendaten generieren.  
End Sub
```

Listing 9.1: Eine einfache Implementierung von `ProcessRequest`

Einige Unternehmen zahlen viel Geld für den Versuch, ein benutzerdefiniertes Gerüst zu entwerfen und zu implementieren. Dieser Versuch wird schließlich aufgegeben, und die Unternehmen entscheiden, dass es kostengünstiger ist, zu den ASP-Seiten zurückzukehren, um die Benutzer-

oberfläche zu definieren. Wenn Sie solch ein Gerüst erstellen möchten, sollten Sie die Vorabkosten und die Bedeutung eines gut durchdachten Designs nicht unterbewerten.

Es gilt jedoch, noch weitere Aspekte zu berücksichtigen. Wenn Sie sich für solch ein komponentenbasiertes Gerüst entscheiden, ist es schwierig, produktivitätsorientierte Seitendesigner, wie zum Beispiel Microsoft FrontPage und Visual InterDev zu verwenden. Wenn Sie einen Seitendesigner benutzen, können Sie in der Regel keine *.ASP*- oder *.HTM*-Dateien verwenden. Sie müssen in diesem Fall HTML-Code-Abschnitte kopieren und in eine Datenbank einfügen. Sie benutzen diese Code-Abschnitte anschließend, um Webseiten dynamisch zusammenzusetzen.

Sie sollten Ihre Inhalte oder HTML-Layoutdetails nicht in Ihre DLLs kompilieren. Wenn Sie zum Beispiel Ihre HTML-Layout-Tags in Ihre DLLs kompilieren, müssen Sie Ihren Code immer dann erneut kompilieren und verteilen, wenn Sie eine geringfügige kosmetische Änderung an Ihrer Site vornehmen möchten. Da Sie mit dem Gerüst versuchen, die Wartbarkeit Ihrer Site zu verbessern, erreichen Sie Ihr Ziel nicht, wenn Ihr Gerüst zu neuen Wartungsproblemen führt. Sie werden nicht mit diesen Problemen konfrontiert, wenn die Benutzeroberfläche in ASP-Seiten definiert ist.

Um die bestmögliche Wartbarkeit und Erweiterbarkeit zu erzielen, können Sie ein Gerüst erstellen, das eine datengesteuerte Lösung verwendet. Sie sollten Geschäftsdaten (wie zum Beispiel eine Kundenliste) datengesteuert in Ihrer Applikation anzeigen lassen. Ihnen stehen jedoch noch weitere Möglichkeiten zur Verfügung. Sie sollten die datengesteuerte Lösung auch dann verwenden, wenn Sie Elemente wie zum Beispiel Seitentitel, die Seitenformatierung und die Navigation definieren. Diese Daten sind natürlich keine Geschäftsdaten im eigentlichen Sinne. Sie bilden Metadaten, die von der Benutzeroberfläche Ihrer Applikation benutzt werden.

Wenn Sie eine datengesteuerte Lösung verwenden, können Sie ein Gerüst erstellen, das dem Webmaster der Site erlaubt, eine neue Seite hinzuzufügen. Er fügt dazu einen neuen Datensatz in eine Datenbank ein, die Ihre Metadaten enthält. Sie können zum Beispiel eine Datenbanktabelle namens *Pages* generieren, die Felder wie zum Beispiel *Title*, *Header* und *Body* definiert. Der Code in Ihrem Gerüst kann diese Elemente dann zur Laufzeit dynamisch in eine Webseite einarbeiten.

Sie können außerdem einfach eine siteweite Änderung an Elementen wie zum Beispiel der Seitenformatierung und der Navigationsleiste vornehmen, wenn die Metadaten, die diese Elemente definieren, ebenfalls in der Datenbank vorhanden sind. Sobald Sie siteweite Formatierungsänderungen vornehmen und neue Seiten hinzufügen können, ohne Ihren Code erneut kompilieren zu müssen, haben Sie sehr gute Arbeit geleistet.

Wie bereits erwähnt wurde, sollten Sie HTML-Metadaten und clientseitige Skripte in einem nicht kompilierten Medium speichern, wenn Sie eine datengesteuerte Lösung verwenden. Eine Datenbank ist in der Regel die beste Wahl. Das Gerüst kann die Metadaten in der Datenbank während des Applikationsstarts oder bei Bedarf auslesen.

Stellen Sie sich vor, Sie haben sich dazu entschlossen, alle HTML-Metadaten in einem DBMS (wie zum Beispiel Microsoft SQL Server oder Oracle) zu speichern. Wenn Ihre ASP-Applikation gestartet wird, können Sie alle Metadaten aus der Datenbank abrufen und in ASP-Application-Variablen laden. Diese Vorgehensweise bietet einen schnellen und einfachen Zugriff auf die Metadaten und lässt Sie in kurzer Zeit benutzerdefinierte Seiten erstellen.

ASP-Seiten und Komponenten zusammen verwenden

Die Wahl zwischen ASP-Seiten und Visual Basic-Komponenten ist keine Alles-oder-nichts-Entscheidung. Viele Unternehmen glauben, dass der beste Ausgleich darin besteht, ASP-Seiten zu benutzen, um das Grundgerüst ihrer Seiten zu definieren, und Komponenten zu verwenden, um die mehr komplexen oder prozessorintensiven Aspekte der Benutzeroberfläche zu generieren.

Hier einige Beispiele für Möglichkeiten, die Ihnen zur Verfügung stehen. Sie können eine generische Komponente erstellen, die ein HTML-Formular mit einigen Eingabefeldern generiert, indem das Schema einer Datenbanktabelle zur Laufzeit untersucht wird. Sie können eine benutzerdefinierte Visual Basic-Komponente erstellen, die formatierte HTML-Tabellen generiert, indem sie sich durch eine ADO-Datensatzgruppe bewegt und diese analysiert. Wenn Sie wirklich progressiv sind, können Sie eine Komponente schreiben, die einen XML-Parser einsetzt, um ADO-Datensatzgruppen in präsentierbare HTML-Inhalte zu konvertieren.

Wie Sie sehen, müssen Sie einige wenige Aspekte berücksichtigen, wenn Sie entscheiden, wie viel Benutzeroberflächenlogik in ASP-Seiten und Visual Basic-Komponenten angeordnet werden soll. Sie müssen darüber hinaus die Anforderungen des Projekts analysieren und die Fähigkeiten Ihrer Team-Mitglieder beurteilen.

Statusverwaltung in Webapplikationen

Wieso führt die Statusverwaltung zu derart vielen Problemen in einer Webapplikation? Der Grund hierfür besteht in der Regel darin, dass HTTP im Wesentlichen ein verbindungsloses und statusarmes Protokoll ist. Die einfache Natur des HTTP-Protokolls setzt voraus, dass der Webserver alle Informationen über den Client in dem Augenblick vergisst, indem er die Bearbeitung einer Anforderung abgeschlossen hat. DCOM und RPC bieten im Gegensatz hierzu ein sehr verbindungsorientiertes Protokoll an, das die Pflege des clientspezifischen Status auf dem Server-Computer sehr einfach werden lässt.

Viele Webentwickler speichern HTTP-Cookies auf dem Client-Computer, um den clientspezifischen Status über mehrere Anforderungen hinweg zu verwalten. Ihre serverseitige Software könnte beispielsweise einen Cookie generieren, um jede Client-Sitzung zu überwachen. Sie müssten dann Applikationslogik schreiben, um Cookies mit Anforderungen und Antworten zu versenden. Die Entwickler müssten in einigen Applikationen einen Cookie zu Beginn jeder Sitzung generieren und übermitteln und in jeder eingehenden Anforderung nach Cookies suchen, damit ein bestimmter Client identifiziert werden kann.

Wie Sie wissen, stellt das ASP-Gerüst bestimmte Features zur Statusverwaltungs über zwei integrierte ASP-Objekte zur Verfügung. Das `Session`-Objekt ist eine Name-/Wert-Liste und ermöglicht das einfache Überwachen des clientspezifischen Status. Das ASP-Gerüst hält eine Annehmlichkeit für Sie bereit, wenn Sie dieses Objekt benutzen, da ein Cookie für jede Client-Sitzung im Hintergrund verwaltet wird. Das `Application`-Objekt ist eine Name-/Wert-Liste, die Ihnen gestattet, einen Status zu speichern, der dann jeder Applikation zur Verfügung steht, die in derselben IIS-Applikation ausgeführt wird.

Ich setze voraus, dass Sie wissen, wie die ASP-Variablen `Session` und `Application` in einer ASP-Seite gelesen und geschrieben werden. Sie sollten ebenfalls wissen, dass es möglich ist, aus einer Visual Basic-Komponente heraus auf diese beiden Variablen zuzugreifen. Lassen Sie mich eine wichtige Regel wiederholen, die bereits in diesem Kapitel genannt wurde: Speichern Sie niemals ein Visual Basic-Objekt in einer ASP-Session- oder ASP-Application-Variablen.

Sie müssen mit diesem Wissen oft entscheiden, wie der clientspezifische Status über mehrere Anforderungen hinweg aufrechterhalten werden soll. Wenn Sie zum Beispiel eine Warenkorb-Webapplikation entwickeln, müssen Sie eine sitzungsbasierte Statusvorrichtung erzeugen, die es einem Client über mehrere Anforderungen hinweg ermöglicht, Produkte zu selektieren und Zahlungsinformationen anzugeben.

Die erste Frage, die Sie in der Design-Phase beantworten sollten, lautet, ob es akzeptabel ist, den clientspezifischen Status auf dem Webserver-Computer zu speichern. In einigen Szenarien ist

dies nicht möglich, zum Beispiel wenn Sie ASP-Seiten und Visual Basic-Komponenten für eine Webgruppen-Umgebung schreiben. Das Speichern eines clientspezifischen Status auf einem Webserver führt nur dann zum Erfolg, wenn alle Anforderungen eines einzelnen Clients über denselben IIS-Computer geleitet werden. Viele Lastenausgleichsschemata, die von Webgruppen genutzt werden, weisen eine solche Einschränkung nicht auf. Komplexe Umgebungen, wie zum Beispiel Webgruppen, benötigen gewöhnlich statusarme Techniken, die den clientspezifischen Status in einer Backend-Datenbank speichern oder mit Hilfe von Cookies respektive HTML-Steurelementen zurücksenden und auf dem Client speichern.

Wir werden die Diskussion über die mit Webgruppen verbundenen Probleme später in diesem Kapitel fortsetzen und uns nun einem einfacheren Szenario zuwenden. Wenn Sie davon ausgehen können, dass Sie ASP-Seiten und Visual Basic-Komponenten für eine Website auf nur einem IIS-Computer schreiben, können Sie die ASP-Session-Variablen benutzen, um den clientspezifischen Status zu verwalten.

Wenn Sie ASP-Session-Variablen einsetzen, um Ihren Status über Anforderungen hinweg aufrechtzuerhalten, sollten Sie einfache Werte (einfache Datentypen) anstelle von Visual Basic-Objekten speichern. Wenn Sie Ihren Status mit Datentypen wie zum Beispiel Integer, Double und String in ASP-Session-Variablen speichern, kann IIS die zukünftigen Anforderungen mit jedem verfügbaren STA-Arbeits-Thread aus dem Pool bearbeiten. Wenn Sie sich für diese Lösung entscheiden, können Sie die ASP-Session-Variablen sogar benutzen, um komplexe Datenstrukturen wie zum Beispiel Arrays und Variants zu speichern.

Betrachten Sie bitte das folgende Beispiel, das ein Visual Basic-Objekt benutzt, um den Status aus einer ASP-Session-Variablen zu lesen und darin zu speichern. Sie können den Code verwenden, um einen einfachen Zeichenfolgenwert in eine ASP-Session-Variable namens `MyState` zu schreiben:

```
Dim sess As Session
Set sess = GetObjectContext("Session")
sess("MyState") = "Einige auf diesen Client bezogene Daten"
```

Sie können in einer zukünftigen Anforderung für denselben Client den Status erneut aus dieser ASP-Session-Variablen auslesen, wie das folgende Beispiel zeigt:

```
Dim sess As Session, s As String
Set sess = GetObjectContext("Session")
s = sess("MyState")
```

In einem etwas komplizierteren Szenario könnten Sie dazu gezwungen sein, einige Zeilenelemente für den Warenkorb eines Clients zu speichern. Dies ist möglich, indem Sie ein Zeichenfolgenarray in einer ASP-Session-Variablen speichern, wie nachfolgend gezeigt:

```
Dim sess As Session, LineItems() As String
Set sess = GetObjectContext("Session")
Redim LineItems(1 To 3)
LineItems(1) = "Product=Dog;Quantity=5"
LineItems(2) = "Product=Cat;Quantity=10"
LineItems(3) = "Product=Bird;Quantity=25"
sess("ShoppingCart") = LineItems
```

Eine zukünftige Anforderung könnte dem Array zusätzliche Zeilenelemente hinzufügen oder eine Transaktion ausführen, um die Elemente für den Kauf zu bearbeiten. Diese Art der Verwen-

derung von ASP-Session-Variablen erlaubt Ihnen, statusbehaftete Applikationen und Sites zu erstellen, die symbolische Warenkörbe verwenden (wie zum Beispiel *Amazon.com*).

Der Schlüsselaspekt des Entwurfs von Applikationen, die ASP-Session-Variablen benutzen, ist, dass Sie vorhersagen müssen, wie Ihre Applikation eingesetzt wird. Jeder Client muss sicher sein können, dass seine Anforderungen von demselben IIS-Computer bedient werden. Im nächsten Abschnitt werden wir einen Blick auf einige Faktoren werfen, die diese Vorhersage nicht realisierbar erscheinen lassen.

Webapplikationen über den Lastenausgleich skalieren

Eine Website ist wie ein aufstrebender Hollywood-Schauspieler. Zu Beginn ihrer Karriere besteht ihr größtes Problem in der Paranoia, für den Rest des Lebens unbeachtet zu sein. Wenn sie jedoch berühmt wird, offenbaren sich neue Herausforderungen. Die eingehenden Anforderungen nehmen dramatisch zu und ihre Fans haben hochgesteckte Erwartungen bezüglich der Performance der Site. Die Fans sind außerdem sehr kritisch, wenn diese Erwartungen nicht erfüllt werden. Eine Website kann diese Herausforderungen leider nicht immer verkraften. Einige Websites reagieren auf Ihren Ruhm, indem Sie sich in Rauch auflösen und sich nie mehr erholen. Die Website, die diesem Druck standhalten kann, wird zu einem Ort, den tausende von Benutzern immer wieder besuchen.

Andere Websites (häufig die Kinder berühmter Websites) sind bereits ab dem Tag berühmt, an dem sie gestartet wurden. Wieder andere erleben eine Veränderung ihrer Benutzerbasis von hundert auf tausende und von tausenden wieder zu hundert von Benutzern. Um die Erwartungen ihrer Fans zu erfüllen, müssen diese Sites entsprechend skalierbar sein.

Das Konzept der Skalierbarkeit wurde bereits in den vorherigen Kapiteln behandelt. Es ist jedoch wichtig, noch einmal einen Blick darauf zu werfen. Eine einfache Definition von Skalierbarkeit lautet: die Fähigkeit des Systems, einer zunehmenden Zahl von Clients gerecht zu werden und jedem Client eine zufriedenstellende Reaktionszeit zukommen zu lassen. Um dieses Ziel zu erreichen, muss eine Website für eine adäquate Anzahl von Verarbeitungszyklen pro Sekunde sorgen. Dies ist notwendig, damit die Menge der eingehenden Anforderungen bewältigt werden kann. Wie Sie wahrscheinlich bereits erraten haben, erfordern mehr Clients auch eine zunehmende Zahl von Verarbeitungszyklen. Daraus ergibt sich die folgende Frage: »Woher kommen all diese Verarbeitungszyklen?«.

Sie können die Skalierbarkeit Ihrer Site zunächst mit einer Ein-Server-Lösung vergrößern. Sie können Ihren Webserver auf einen Computer mit einem schnelleren Prozessor oder mehreren Prozessoren aufrüsten. Diese Vorgehensweise wird als Aufwärtsskalierung bezeichnet. Sie haben zum Beispiel die Möglichkeit, einen neuen Webserver-Computer mit zwei oder vier schnellen Prozessoren zu kaufen. Der neue Computer bearbeitet dann eine größere Clientbasis als das vorherige Gerät. Bisweilen führt eine Ein-Server-Lösung jedoch nicht zu dem gewünschten Ziel. Sie können lediglich eine Aufwärtsskalierung durchführen. Die High-End-Computer in diesem Bereich sind außerdem extrem teuer. Sobald Sie einen bestimmten Grenzwert erreichen, verlangt die kosteneffektive Skalierbarkeit den Einsatz von mehreren Prozessoren, die auf mehrere Computer verteilt sind. In diesem Fall wird häufig davon gesprochen, dass Sie eine Auswärts- und nicht eine Aufwärtsskalierung benötigen.

Hier kommt der Lastenausgleich ins Spiel. Sie müssen die Arbeitslast der eingehenden HTTP-Anforderungen auf mehrere, weniger kostspielige Computer verteilen. Wir werden hierzu einige Lösungen betrachten. Für eine Webapplikation ist es besser, sich dort mit dem Lastenausgleich zu beschäftigen, wo die HTTP-Anforderung die Site erreicht. Dies bedeutet, dass Sie eine Technik

benötigen, um eingehende HTTP-Anforderung auf mehrere IIS-Computer zu verteilen. Eine Site, die auf diese Weise vorgeht, wird häufig als Webgruppe, Web-Farm oder als Webserver-Array bezeichnet.

Web-Farmen erstellen

Webentwickler verfügen über einige Techniken für die Verteilung von HTTP-Anforderungen auf mehrere Server. Eine einfach Möglichkeit besteht darin, eine Website mit einem der Site zugewiesenen Routing-Server zu erstellen. Der Routing-Server besitzt benutzerdefinierten Code, um Clients umzuleiten wie in Abbildung 9.8 dargestellt.

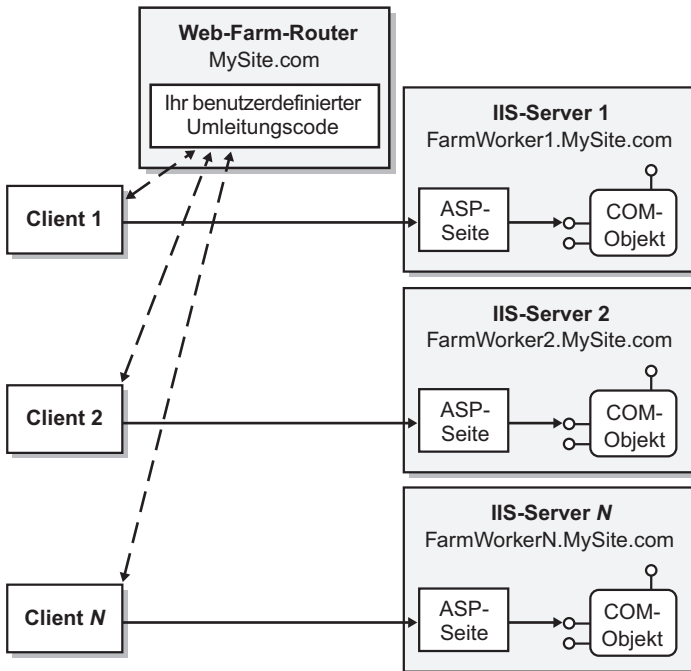


Abbildung 9.8: Sie können eine sitzungsbasierte Web-Farm erstellen, indem Sie einen Routing-Server benutzen, um mehrere Clients auf mehrere IIS-Servercomputer zu verteilen.

Dem Routing-Server wird ein bekannter DNS-Name (wie zum Beispiel *MySite.com*) und eine ihm zugeordnete IP-Adresse zugewiesen. Die anderen Server in der Farm besitzen ihre eigenen IP-Adressen und optionale DNS-Namen. Wenn die erste Anforderung eines Clients den Routing-Server erreicht, wird sie an einen der anderen Server in der Farm umgeleitet. Sie können Clients vom Routing-Server umleiten lassen, indem Sie das *Session_OnStart*-Ereignis in der Datei *global.asa* benutzen. Listing 9.2 zeigt eine mögliche Implementierung.

```
Sub Session_OnStart
    Const SERVER_COUNT = 3
    Dim sServer, sURL
    Randomize
    Select Case (Fix(Rnd * SERVER_COUNT) + 1)
        Case 1
            sServer = "FarmWorker1"
        Case 2
            sServer = "FarmWorker2"
        Case 3
```

```

        sServer = "FarmWorker3"
    End Select
    sURL = "http://" & sServer & ".MySite.com/MyApp/default.asp"
    Response.Redirect sURL
End Sub

```

Listing 9.2: Umleiten von Clients mit Hilfe des Session_OnStart-Ereignisses

Diese Technik verlangt, dass die erste Anforderung des Clients an den Routing-Server für eine Seite mit einer *.ASP*-Erweiterung durchgeführt wird. Das Session_OnStart-Ereignis wird nicht für eine Seite mit einer *.HTM* oder *.HTML*-Endung ausgelöst. Der zuvor aufgeführte Algorithmus leitet jeden neuen Client zu einem von drei Servern in der Farm um. Wenn ein Client zu einem IIS-Computer in der Farm umgeleitet wird, wird eine Sitzung erzeugt, und der Client sendet alle zukünftigen Anforderungen an dieselbe Adresse. Sie können sich diese Vorgehensweise deshalb als eine sitzungsbasierte Lastenausgleichstechnik vorstellen.

Der Algorithmus verwendet eine Zufallszahl für die Umleitung. Sie können jedoch einen etwas umfassenderen Lastenausgleichsmechanismus entwerfen. Jeder Server in der Farm könnte zum Beispiel Performance-Daten an den Routing-Server zurücksenden. Wenn jeder Server in regelmäßigen Abständen die Zahl der aktiven Sitzungen an den Routing-Server zurückschickt, kann der Lastenausgleichscode jeden neuen Client an den Server umleiten, der die am wenigsten aktiven Clients bearbeitet.

Das so genannte Round-Robin-DNS ist eine weitere sitzungsbasierte Lastenausgleichstechnik. Dabei wird jeder logische DNS-Name (wie zum Beispiel *MySite.com*) mehreren IP-Adressen zugeordnet. Wenn ein Browser versucht, den DNS-Namen auszuwerten, sendet der DNS-Server eine der Adressen aus der Liste zurück. Der DNS-Server entnimmt die Adressen im Rotationsverfahren, um mehrere Clients auf mehrere Server zu verteilen. Sobald ein Browser einen DNS-Namen in eine IP-Adresse aufgelöst hat, legt er die IP-Adresse für die Dauer der Client-Sitzung in einem Cache ab. Das Round-Robin-DNS ist etwas schneller als die zuvor vorgestellte Umleitungstechnik, führt aber zu denselben Ergebnissen. Verschiedenen Clients werden unterschiedliche IP-Adressen zugewiesen, um die Arbeitslast auf mehrere Server zu verteilen.

Um eine Web-Farm mit einer dieser sitzungsbasierten Lastenausgleichstechniken einzurichten, sollten Sie relative URLs in Ihren ASP-Seiten und in Ihrem Visual Basic-Code verwenden. Sie sollten zum Beispiel eine URL wie */MyOtherPage.asp* anstelle einer absoluten URL benutzen, die den DNS-Namen oder die IP-Adresse eines Servers enthält. Dies gewährleistet, dass alle Anforderungen eines Clients an denselben Server geschickt werden, sobald eine Sitzung gestartet wurde.

Während diese beiden sitzungsbasierten Lastenausgleichsformen sehr einfach einzurichten sind, weisen sie einige Einschränkungen auf. Der Lastenausgleich wird erstens nur für jeden Client einmal zu Beginn der Sitzung durchgeführt. Das Lastenausgleichsschema kann dadurch ein wenig verzerrt werden. Alle Benutzer, die zum Beispiel zu *FarmWorker1* umgeleitet wurden, könnten in die Mittagspause gehen, während die Benutzer, die an *FarmWorker2* weitergeleitet wurden, weiterhin Anforderungen senden. In diesem Fall könnte ein Server übermäßig belastet werden, während sich der andere Server im Leerlauf befindet.

Ein weiteres Problem betrifft die Verfügbarkeit einer Webapplikation. Der sitzungsbasierte Lastenausgleich lässt die IP-Adressen der Server in der Farm für die Clients zugänglich werden. Dies ist eine Einschränkung, da jeder IIS-Computer auf diese Weise zu einer Fehlerquelle wird.

Was geschieht, wenn ein Webserver in der Farm abstürzt oder aus Wartungsgründen vom Netzwerk getrennt wird? Der Lastenausgleichsalgorithmus muss dies schnellstmöglich erkennen und

darauf reagieren. Dies kann jedoch problematisch sein. Wenn der Routing-Server oder der DNS-Server eine IP-Adresse für einen Server ausgibt, der nicht mehr online ist, erhalten die Clients eine »Server nicht verfügbar«-Fehlermeldung. Wenn Sie ein Round-Robin-DNS-System verwenden, kann die Problemlösung sehr viel Zeit erfordern, sobald Sie entdecken, dass ein Server abgestürzt ist. Alles wird jedoch noch schlimmer, wenn die IP-Adresszuordnungen an andere DNS-Server über das Internet weitergeleitet werden müssen.

Wenn ein Webserver nicht mehr online ist, betrifft dies jeden Client, für den der Server einen sitzungsspezifischen Status speichert. Wenn der Webserver also abstürzt, ist es möglich, dass die Client-Sitzungen mit ihm abstürzen. Wie Sie sehen, können die bisher vorgestellten Lastenausgleichstechniken die Verfügbarkeit und Fehlertoleranz einer Webapplikation einschränken. Sie besitzen glücklicherweise die Möglichkeit, weiterentwickelte Verfahren für den Lastenausgleich zu verwenden, die diese Probleme umgehen. Die Lösung besteht darin, jedem Client eine einzelne IP-Adresse zur Verfügung zu stellen.

Eine bessere Web-Farm entwickeln

Wie Sie gesehen haben, kann das Offenlegen mehrerer IP-Adressen für eine einzelne Website der Verfügbarkeit und dem Lastenausgleich schaden. Es ist besser, eine einzelne IP-Adresse zur Verfügung zu stellen, die mehreren physikalischen Servern zugeordnet wird. Diese Lösung erfordert jedoch Netzwerkcode auf der Systemebene, um eingehende IP-Pakete zu mehreren Servern umzuleiten. Die meisten Unternehmen entscheiden sich für eine vorgefertigte und nicht für eine eigene Lösung. Wir werden einige gebräuchliche Optionen in diesem Abschnitt untersuchen.

Wir betrachten zunächst, wie Sie den Lastenausgleich mit einem Hardware-Produkt wie zum Beispiel dem LocalDirector-Switch von Cisco oder dem BigIP-Router von F5 realisieren. Danach wenden wir uns NLB (Network Load Balancing) zu. Dies ist eine in Windows 2000 Advanced Server enthaltene Software-Lösung. Beachten Sie bitte, dass viele andere Unternehmen ähnliche Produkte anbieten. Auf diesem Markt herrscht eine große Konkurrenz, so dass Sie selbst recherchieren sollten, um den Anbieter zu ermitteln, der Ihnen das beste Preis-/Leistungsverhältnis bieten kann.

Abbildung 9.9 zeigt die Basiskonfiguration einer Site mit einer Hardware-Lösung. Ein Router oder Switch wartet auf eingehende Anforderungen an eine einzelne virtuelle IP-Adresse und kann diese an mehrere IIS-Server-Computer weiterleiten. Jeder Computer in der Farm besitzt seine eigene eindeutige IP-Adresse. Im Gegensatz zu den Lastenausgleichstechniken, die wir zuvor besprochen haben, können die Clients die IP-Adressen der physikalischen Server jedoch nicht einsehen. Der Lastenausgleich wird immer dann durchgeführt, wenn eine Anforderung über die »virtuelle« IP-Adresse eingeht. Diese Vorgehensweise wird als anforderungsbasierter Lastenausgleich bezeichnet.

NLB ist ein Windows-Dienst, der eine Software-Lösung für den anforderungsbasierten Lastenausgleich repräsentiert. NLB kann von vielen Applikationstypen eingesetzt werden, die den IP-Verkehr nutzen. Die verbleibenden Abschnitte dieses Kapitels beschäftigen sich mit dem Erstellen einer Web-Farm unter Verwendung von NLB.

Der NLB-Dienst basiert auf der Software *Convoy Cluster*, die Microsoft von Valence Research erworben hat. Im Gegensatz zu LocalDirector oder BigIP benötigt NLB keine proprietäre Hardware. Der Dienst wird als Windows-Gerätetreiber auf jeden Computer der Farm installiert, wie in Abbildung 9.10 dargestellt. NLB kann eine Web-Farm auf bis zu 32 Server erweitern.

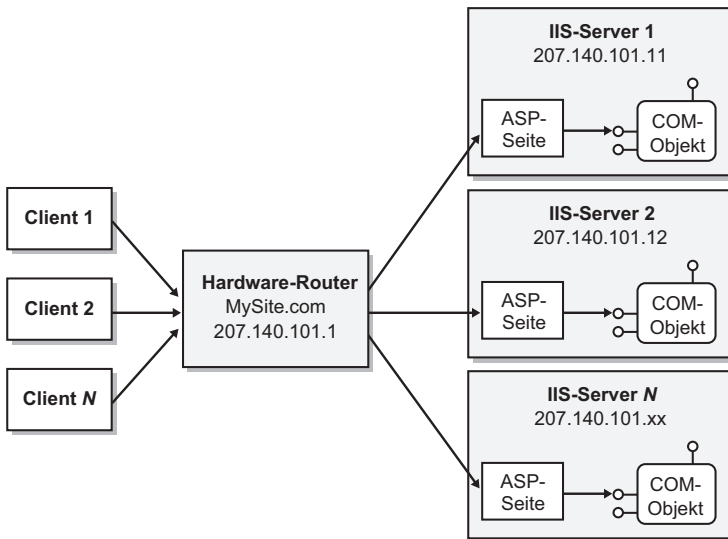


Abbildung 9.9: Der anforderungsbasierte Lastenausgleich kann eine Hardware-Lösung anbieten. Client-Anforderungen werden an eine einzelne virtuelle IP-Adresse gerichtet und an mehrere physikalische IP-Adressen weitergeleitet.

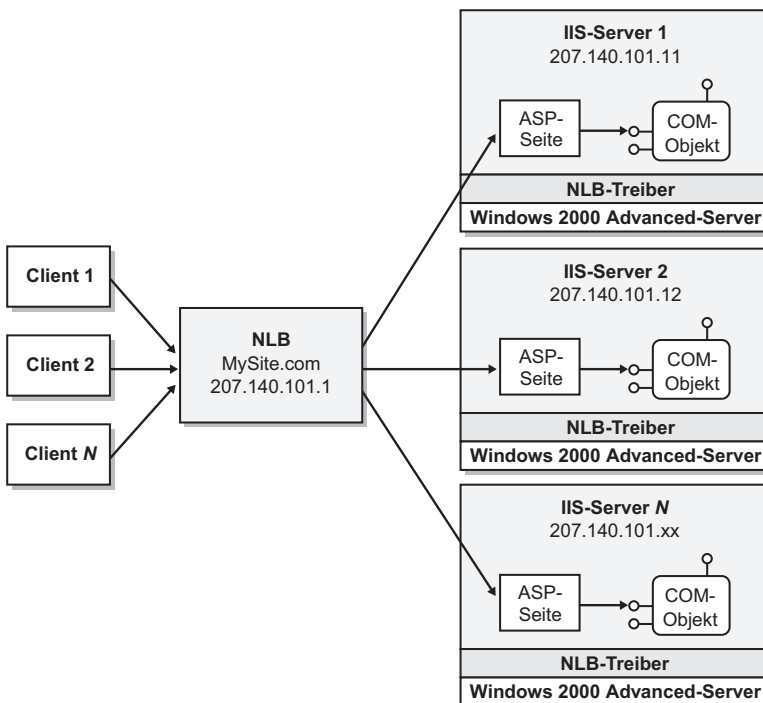


Abbildung 9.10: NLB (Network Load Balancing) erlaubt mehreren Computern, auf denen Windows 2000 Advanced Server ausgeführt wird, eine einzelne IP-Adresse zu überwachen.

Wenn Sie eine Web-Farm mit NLB einrichten, kommunizieren alle Server ständig miteinander. Sie tauschen Performance-Statistiken aus und teilen die Verantwortlichkeit für die eingehenden Anforderungen untereinander auf. Jede eingehende Anforderung wird von jedem Server in der Farm erkannt. NLB verfügt über einen proprietären Algorithmus, um zu bestimmen, welcher Server eine Anforderung bearbeitet.

NLB weist einige Vorteile gegenüber einer Hardware-Lösung auf. Es benötigt keine proprietäre Hardware und es bietet keine Angriffsfläche für Fehler. Wenn Sie eine Hardware-Lösung benutzen möchten, müssen Sie in der Regel ein zweites ausfallübergreifendes Gerät erwerben, um die Verfügbarkeit und Fehlertoleranz zu verbessern. Einige Unternehmen glauben, dass eine ausfallübergreifende Hardware-Lösung zu teuer für ihre Anforderungen sei. Das Argument für eine Hardware-Lösung lautet jedoch, dass diese nur einmalige Kosten verursacht. Bei einer Software-Lösung müssen Sie in der Regel immer dann zusätzliche Lizenzgebühren bezahlen, wenn Sie einen weiteren Server zu Ihrer Web-Farm hinzufügen. Lassen Sie mich noch einmal erwähnen, dass Sie in jedem Fall recherchieren sollten, welche Lösung für Ihre Applikation geeignet ist.

LocalDirector, BigIP und NLB unterscheiden sich signifikant darin voneinander, wie sie auf der Systemebene konfiguriert sind. Aus einer höheren Perspektive betrachtet, produzieren sie jedoch dieselben Ergebnisse. Jeder Client sendet eine Anforderung mit einer virtuellen IP-Adresse. Die Anforderung wird an einen von mehreren IIS-Server-Computern in der Web-Farm weitergeleitet. Der anforderungsbasierte Lastenausgleich weist viele Vorteile gegenüber dem sitzungsbasierten Lastenausgleich auf. Der anforderungsbasierte Lastenausgleich ist feiner strukturiert, da sein Algorithmus häufiger ausgeführt wird. Dies führt zu einer gleichmäßigeren Verteilung der Anforderungen auf mehrere Server.

Die primäre Motivation für den Einsatz eines anforderungsbasierten Lastenausgleichs besteht darin, eine bessere Verfügbarkeit und Fehlertoleranz zu erzielen. Sowohl NLB als auch Hardware-Lösungen können feststellen, wann ein Server nicht mehr online ist. Beide Lösungen können schnell darauf reagieren, um zu verhindern, dass zukünftige Anforderungen an eine nicht verfügbare IP-Adresse geleitet werden. Ein Administrator kann darüber hinaus einen Server während der Betriebsstunden sicher vom Netzwerk trennen. Er könnte zum Beispiel ein sukzessives Upgrade durchführen, für das alle Web-Farm-Computer nacheinander vom Netzwerk getrennt werden müssen. Das sukzessive Upgrade erlaubt Aktionen wie zum Beispiel das Herunterfahren von *INETINFO.EXE*, um einige Visual Basic-DLLs zu überarbeiten. Die Idee hinter dem sukzessiven Upgrade ist, dass die Ausführung der Website als Ganzes niemals unterbrochen wird.

Der anforderungsbasierte Lastenausgleich weist jedoch auch einige Nachteile auf. Ein wesentlicher Nachteil ist, dass die Statusverwaltung in einer Web-Farm komplizierter wird, wenn Sie nicht voraussetzen können, dass die Client-Anforderungen immer von demselben IIS-Computer bedient werden. Wenn dies nicht möglich ist, sollten Sie nicht versuchen, den Status mit einer ASP-Session-Variablen aufrechtzuerhalten. Sie müssen dann eine weiterentwickelte Lösung für die Sitzungsverwaltung entwerfen. Es gibt außerdem einige wichtige Aspekte, die den Einsatz von ASP-Application-Variablen in einer Umgebung betreffen, die den anforderungsbasierten Lastenausgleich verwendet.

Den Status in einer Web-Farm aufrechterhalten

Wenn die Benutzer Ihrer Webapplikation lediglich Informationen einsehen möchten, müssen Sie sich nicht viele Gedanken um die Erhaltung des Status machen. Wenn Ihre Webapplikation jedoch zum Beispiel einen Warenkorb anbietet, müssen Sie sorgfältig darüber nachdenken, wie der Status eines Client aufgebaut und über mehrere Anforderungen hinweg gespeichert wird.

Sie sollten die Design-Phase beginnen, indem Sie sich eine einfache Frage stellen: Wo können Sie den clientspezifischen Status in Ihrer Applikation aufbewahren? Sie haben drei Möglichkeiten: Sie können den Status erstens auf dem Client-Computer mit Hilfe von Cookies oder in dem HTML-Code speichern, den Sie an den Browser senden. Sie können den Status zweitens auf der mittleren Schicht in IIS speichern. Sie verfügen drittens über die Möglichkeit, den Status in einem Back-End-DBMS wie zum Beispiel SQL Server abzulegen.

Wenn Sie, basierend auf nur einem IIS-Computer, eine Webapplikation für eine Site entwerfen, ist das Speichern des Status unter IIS mit ASP-Session-Variablen in der Regel die einfachste Lösung. Wenn Sie eine Web-Farm besitzen, die den sitzungsbasierten Lastenausgleich verwendet, können Sie auch den clientspezifischen Status unter IIS speichern. Wenn Sie jedoch über eine Web-Farm verfügen, die den anforderungsbasierten Lastenausgleich benutzt, arbeiten die meisten Techniken zum Speichern der Daten mit IIS nicht korrekt.

Wenn Sie den Status unter IIS nicht speichern können, müssen Sie auf eine der beiden anderen Optionen ausweichen. Der Hauptvorteil, der mit dem Speichern des Status auf dem Client verbunden ist, besteht darin, dass dies sehr schnell geschieht. Sie müssen keinen zeitaufwendigen Trip zur Backend-Datenbank unternehmen. Einige Vorteile der Statusspeicherung in einer Backend-Datenbank sind, dass die Daten beständig sind und Sie so viel speichern können, wie Sie möchten. Das Speichern des Status auf dem Client ist mit einer begrenzten Kapazität verknüpft und führt zu Sicherheitsproblemen. Viele Unternehmen möchten keine sensiblen Informationen auf dem Client-Computer speichern. Einige clientseitige Sicherheitseinstellungen könnten auch Sie davon abhalten, auf die Festplatte des Clients zu schreiben.

Sie sollten beachten, dass Sie möglicherweise nur einen geringen Teil des Status auf dem Client verwahren müssen, wenn Sie den Status in einer Backend-Datenbank speichern. Es ist zum Beispiel üblich, mit jeder Anforderung explizit eine Kunden-ID oder einen anderen Primärschlüssel zwischen dem Client und IIS zu übergeben. Sie können dann einem oder mehreren Datensätzen in Ihrer Datenbank einen Benutzer zuordnen. Bisweilen genügt auch eine HTTP-Übertragung der Client-IP-Adresse oder eines Benutzernamens. Wie Sie sehen, ist es möglich, einen »kleinen« clientseitigen Status aufrechtzuerhalten und den überwiegenden Teil Ihrer Daten sicher in einer Datenbank wie zum Beispiel SQL Server zu speichern.

Während der Design-Phase sollten Sie sich die folgenden Fragen stellen, wenn Sie entscheiden, ob Ihr clientspezifischer Status in eine Backend-Datenbank geschrieben werden soll. Wie wirkt sich der Zugriff auf die Backend-Datenbank auf die Performance aus? Bildet die Backend-Datenbank einen signifikanten Performance-Flaschenhals? In diesem Fall werden Sie möglicherweise darüber nachdenken, die Anzahl der Übertragungen zur Datenbank zu reduzieren. Sie sollten einige Benchmarks durchführen und die Reaktionszeiten der Anforderungen, die zu einem Datenbankzugriff führen, mit den Zeiten vergleichen, die für alle anderen Anforderungen benötigt werden.

Sie können die Anzahl der Datenübertragungen zur Datenbank verringern, aber nicht vollständig vermeiden. Hier noch einige Fragen, die Sie während der Design-Phase berücksichtigen sollten. Welche Folgen hat es für Sie, wenn der clientspezifische Status verloren geht oder beschädigt wird? Möchten Sie, dass derselbe clientspezifische Status verfügbar ist, wenn ein Benutzer zu einem anderen Computer wechselt? Ist Ihr clientspezifischer Status so sensitiv, dass Sie ihn vor den Benutzern verbergen müssen? Müssen Sie so viel clientspezifischen Status speichern, dass dies auf dem Client nicht möglich ist?

Sie können in der Regel immer einen angemessenen Kompromiss hinsichtlich der Menge finden, die in Ihrer Datenbank gespeichert werden soll. Sie könnten auf die erste Anforderung des Clients reagieren, indem Sie auf die Datenbank zugreifen, um den Benutzernamen und das Kenn-

wort zu überprüfen und einige vorkonfigurierte Benutzerpräferenzen abzurufen. Wenn der Benutzer dann beginnt, Artikel zum Warenkorb hinzuzufügen, können Sie diese Aktion überwachen, indem Sie den Status auf dem Client-Computer speichern. Wenn der Benutzer die Artikel schließlich erwirbt, kann Ihre Applikation zum zweiten Mal auf die Datenbank zugreifen, um die entsprechende Transaktion aufzuzeichnen. In diesem Szenario führt die Client-Sitzung vielleicht zu 20 bis 30 Anforderungen, doch nur 2 erfordern einen Zugriff auf die Backend-Datenbank.

Übergeben des Status zwischen IIS und dem Browser

Unabhängig davon, ob Sie einen großen Teil des clientspezifischen Status oder nur einen Primärschlüssel auf dem Client speichern, müssen Sie wissen, wie Daten zwischen dem Browser und IIS übergeben werden. In diesem Abschnitt werden wir drei hierzu erforderliche, gebräuchliche Techniken untersuchen.

Die einfachste Möglichkeit, den Status auf dem Client zu speichern, besteht darin, HTTP-Cookies zu verwenden. Die ASP-Objekte `Response` und `Request` vereinfachen das Schreiben und Auslesen eines Cookie-Wertes. Sie können einen Cookie beispielsweise wie folgt in eine HTTP-Antwort schreiben:

```
Dim rsp As Response
Set rsp = GetObjectContext.Item("Response")
rsp.Cookies("UserID") = "BrianR"
rsp.Cookies("UserName") = "Brian Randell"
rsp.Cookies("FavoriteFood") = "CheeseBurger"
rsp.Cookies("FavoriteQuantity") = "3"
```

Wenn Sie Cookie-Werte in eine HTTP-Antwort schreiben, werden diese mit jedem Anforderungs-/Antwortpaar vor- und zurückbewegt. Cookies können sehr einfach mit dem ASP-Objekt `Request` gelesen werden. Die Cookies in dem zuvor aufgeführten Beispiel bleiben für die Dauer der Browser-Sitzung bestehen.

Sie können persistente Cookie-Werte ebenfalls auf der Festplatte eines Benutzers speichern. In diesem Fall bleiben die Cookie-Werte über mehrere Browser-Sitzungen hinweg vorhanden. Dies bedeutet, dass sich Ihre Site alle Arten von Informationen merken und benutzerspezifische Präferenzen für eine längere Zeit speichern kann. Benutzer scheinen Sites, die dies tun, zu schätzen. Um einen persistenten Cookie-Wert auf der Festplatte des Benutzers zu speichern, fügen Sie einfach ein Ablaufdatum wie folgt hinzu:

```
rsp.Cookies("UserID") = "BrianR"
rsp.Cookies("UserID").Expires = "July 1, 2001"
```

Sie können in einer Warenkorb-Applikation Cookies benutzen, um sukzessive eine komplexe Datenstruktur für jeden bestellten Artikel aufzubauen. Hier ein Beispiel, das mehrere Werte in einen Cookie schreibt, um die Zeilenelemente eines Warenkorbs zu speichern:

```
rsp.Cookies("LineItemCount")= "3"
rsp.Cookies("LineItems")("Item1") = "Product=Dog;Quantity=5"
rsp.Cookies("LineItems")("Item2") = "Product=Cat;Quantity=10"
rsp.Cookies("LineItems")("Item3") = "Product=Bird;Quantity=25"
```

Sie sollten einige wichtige Einschränkungen bezüglich der Verwendung von HTTP-Cookies beachten. Cookies können nicht über Domänen hinweg benutzt werden. Ihre Speicherkapazität ist außerdem begrenzt. Die meisten Browser unterstützen Cookies bis zu einer Kapazität von 4096

Bytes. Während Sie mehrere Cookies auf einem Client speichern können, sind größere Datenmengen möglicherweise zu komplex, um damit arbeiten zu können. Sie können außerdem nicht voraussetzen, dass alle Clients Cookies unterstützen. Einige Browser sind einfach zu alt für solch eine Unterstützung. Einige Benutzer haben Cookies darüber hinaus in ihren Browsern deaktiviert, nachdem sie Warnungen von Verschwörungstheoretikern gelesen haben.

Wenn Sie Clients berücksichtigen müssen, die Cookies nicht unterstützen, können Sie Daten nicht über mehrere Sitzungen des Browsers hinweg auf dem Client speichern. Sie müssen außerdem eine andere Technik benutzen, um den clientspezifischen Status zu übergeben. Eine beliebte Technik verwendet Abfragezeichenfolgen, um benannte Werte an Ihre URLs anzuhängen. Anstatt Ihre HTML-Formularaktion beispielsweise auf `\MyPage.asp` zu setzen, können Sie eine Abfragezeichenfolge wie folgt an den URL anfügen:

```
\MyPage.asp?UserID=BrianR.
```

Das Anfügen von benannten Werten an URLs erfordert einen zusätzlichen Aufwand. Sie müssen diese Name-/Wert-Paare dynamisch am Ende jedes URLs einbetten, wenn Sie den HTML-Code für Ihre Seiten generieren. Sie sind außerdem gewöhnlich auf 2 Kilobyte beschränkt, wenn Sie den Status mit Abfragezeichenfolgen verwalten.

Ihnen steht jedoch noch eine weitere Technik für die clientseitige Statusverwaltung zur Verfügung. Diese Technik verwendet ein verstecktes Feld in einem HTML-Formular. Wenn ein Benutzer keine Cookies akzeptiert und Sie mehr Status als die Menge, die mit einer Abfragezeichenfolge an einen URL angehängt werden kann, speichern möchten, könnte dies die Lösung sein, nach der Sie gesucht haben. Beachten Sie, dass diese Technik den Einsatz eines HTML-Formulars und einer *Abschicken*-Schaltfläche erfordert. Hier ein Beispiel dafür, wie Ihr Formular aufgebaut sein könnte:

```
<FORM ACTION="MyPage.asp" METHOD="Post">
  <INPUT TYPE="Hidden" NAME="UserID" VALUE="BrianR">
  <INPUT TYPE="Submit" VALUE="Get Past Purchases">
</FORM>
```

Einige weitere Worte zur Optimierung

Produkte wie zum Beispiel NLB, LocalDirector und BigIP erlauben Ihnen, auf den anforderungsbasierten Lastenausgleich zu verzichten und auf den sitzungsbasierten Lastenausgleich auszuweichen. Wenn Sie zum Beispiel die *Affinity*-Einstellung in NLB setzen, können Sie den Dienst anweisen, jeden Client an denselben physikalischen Server weiterzuleiten, nachdem die erste Anforderung gesendet wurde. Dies scheinen gute Neuigkeiten zu sein, sofern Sie eine große, bereits bestehende Webapplikation besitzen, die Abhängigkeiten von ASP-Session-Variablen aufweist.

Die schlechte Nachricht lautet, dass die Alternative, die zurück zum sitzungsbasierten Lastenausgleich führt, die Performance und Verfügbarkeit einer Applikation beeinträchtigt. Sie verlieren sehr viele Vorteile des anforderungsbasierten Lastenausgleichs. Alles wird langsamer, da der Routing-Mechanismus die Client-/Server-IP-Adresszuordnungen verwahren und jede Anforderung überprüfen muss. Ihre Site wird außerdem für Probleme mit der Fehlertoleranz und für eine verzerrte Verteilung anfällig, die bereits besprochen wurde. Weitere Probleme treten auf, wenn Client-Anforderungen über einen Proxy-Server geleitet werden.

Es ist zum Beispiel für hunderte oder tausende von Clients möglich, mit nur einem Proxy-Server verknüpft zu werden. Dies bedeutet, dass Ihr Lastenausgleichsmechanismus mit einer IP-Adresse konfrontiert werden kann, der sehr viele Clients zugeordnet sind. Vielleicht geraten Sie in eine

Situation, in der 3000 Benutzer, die sich denselben Proxy-Server teilen, einen Ihrer Webserver wie ein Kartenhaus zusammenfallen lassen, während neun andere Server in der Farm nichts zu tun haben.

Um alles noch ein wenig schlimmer zu machen, besitzt AOL (America Online) eine Proxy-Farm, die so groß ist, dass sie sich über mehrere Klasse-C-Adressen erstreckt. Eine Klasse-C-Adresse bildet jedoch die Affinitätsebene in NLB. Dies bedeutet, dass die Anforderungen von AOL-Benutzern häufig auch dann über mehrere NLB-Computer geleitet werden, wenn die Affinität aktiviert ist. In diesem Fall verursachen die `Session`-Variablen sogar Probleme, wenn Sie den sitzungsbasierten Lastenausgleich verwenden.

Meine Meinung ist, dass Sie nicht zum sitzungsbasierten Lastenausgleich zurückkehren sollten, wenn Sie den anforderungsbasierten Lastenausgleich verwenden können. Dies wäre so, als würden Sie einen Porsche mit 60 km/h über die Autobahn fahren. Sie sollten realisieren, dass all die »sitzungsbewussten« Features primär solche Applikationen unterstützen, die Abhängigkeiten von dem Status der mittleren Schicht aufweisen. Ihre Webapplikation lässt sich viel besser skalieren, wenn Sie auf diese Abhängigkeiten in Ihrem Design verzichten.

Stellen Sie sich vor, dass Sie während der Design-Phase sehr diszipliniert waren und eine Webapplikation ohne Abhängigkeiten von der ASP-Sitzungsverwaltung erstellt haben. Sie sollten an diesem Punkt eine weitere Optimierung vornehmen: Sie sollten die Sitzungsverwaltung deaktivieren. IIS führt viele Sitzungsverwaltungstätigkeiten durch, wenn Sie nicht das Sitzungsverwaltungs-Feature deaktivieren. Wenn Sie dieses Feature aktiviert lassen, kann dies zu einer Verschwendung wertvoller Ressourcen wie zum Beispiel Verarbeitungszyklen, Speicher und Netzwerkbandbreite führen.

Doch was geschieht, wenn Sie die ASP-Sitzungsverwaltung nicht deaktivieren? Wenn ein Client eine ASP-Seite anfordert, prüft das ASP-Gerüst, ob der Client mit einer aktiven Sitzung verknüpft ist. Wenn der Client zum ersten Mal eine Anforderung sendet, initiiert das ASP-Gerüst mehrere Aufgaben, um die Client-Sitzung zu überwachen. ASP generiert eine eindeutige `SessionID` und übergibt diese wiederholt in HTTP-Cookies. ASP sucht außerdem nach dem `Session_OnStart`-Ereignis und führt es aus, wenn es vorhanden ist. Wenn das ASP-Gerüst eine zulässige `SessionID` in einer eingehenden Anforderung entdeckt und feststellt, dass der Benutzer an einer aktiven Sitzung teilnimmt, muss es die aktuelle Anforderung mit jedem clientspezifischen Status verknüpfen, der möglicherweise aufgrund der früheren Anforderungen in ASP-`Session`-Variablen gespeichert wurde.

Das ASP-Gerüst verrichtet außerdem zusätzliche Arbeit, um alle Anforderungen für eine Sitzung zu serialisieren. Diese Unterstützung verhindert Parallelitätsprobleme, die sich auf den Sitzungsstatus beziehen. Diese Serialisierung von Client-Anforderungen ist jedoch, sofern sie nicht erforderlich ist, aufwendig und einschränkend. Stellen Sie sich zum Beispiel vor, dass Ihre Applikation zwei Frames im Browser definiert. Jeder Frame basiert auf einer ASP-Seite. Eine ASP-Seite muss vollständig verarbeitet sein, bevor die ASP-Laufzeit mit der Bearbeitung der zweiten Seite beginnt. Wenn die Sitzungsverwaltung deaktiviert ist, kann IIS beide ASP-Seiten gleichzeitig auf zwei unterschiedlichen Threads bearbeiten. Dies ist besonders dann wichtig, wenn der Webserver-Computer mehrere Prozessoren besitzt.

Wie Sie sehen, sind die ASP-Sitzungsverwaltungs-Features sehr aufwendig. Wenn Sie sie nicht benötigen, sollten Sie sie ausschalten, indem Sie das Kontrollkästchen *Sitzungszustand aktivieren* im Internetdienste-Manager deaktivieren. Sie können die Deaktivierung für eine IIS-Applikation oder eine vollständige Site durchführen. Sie können die Sitzungsverwaltung außerdem deaktivieren, indem Sie die folgende Deklaration zu Beginn jeder ASP-Seite einfügen:

```
<%@ EnableSessionState=False %>
```

Speichern des applikationsweiten Status im Cache

Das letzte Thema, das ich in diesem Kapitel behandeln möchte, ist die Verwaltung des applikationsweiten Status in einer Website. Das ASP-Gerüst bietet das `Application`-Objekt an, damit mehrere Clients gemeinsam mehrere Variablen benutzen können. Jede Anforderung kann auf alle ASP-Applikationsvariablen in einer IIS-Applikation zugreifen.

Die Verwendung von ASP-Application-Variablen gleicht dem Einsatz des SPM (Shared Property Manager). Sie bietet die Möglichkeit, Daten im Cache-Speicher innerhalb eines Prozesses abzufragen und diese Daten mehreren Clients zur gemeinsamen Verfügung bereitzustellen. Sie können ASP-Application-Variablen und den SPM tatsächlich in vielen Situationen vertauschen, um zum selben Ziel zu gelangen. Lassen Sie mich die Vorteile jeder Lösung aufzählen.

Wenn Sie aktualisierbare Daten im Cache ablegen, kann der SPM eine besser strukturierte Sperrung anbieten, sofern Sie Ihre im Cache befindlichen Daten in mehrere Eigenschaftsgruppen unterteilt haben. Das vom ASP-Objekt `Application` verwendete Sperrschema erlaubt nur eine grob strukturierte Sperrung. Ein weiterer potentieller Vorteil des SPM besteht darin, dass er keine Abhängigkeiten von der ASP-Laufzeit aufweist. Sie können den SPM benutzen, um eine konfigurierte Komponente zu erstellen, die ein Cache-Schema zur Verfügung stellt und anschließend die Komponente in einer COM+-Applikation zusammen mit Web- und DCOM-Clients benutzt.

Viele ASP-Entwickler sind der Meinung, dass ASP-Application-Variablen einfach zu benutzen sind. Sie können auf diese Variablen außerdem direkt von einer ASP-Seite aus zugreifen. Ein Zugriff auf den SPM ist von dieser Position aus jedoch nicht möglich. Das ASP-Application-Objekt vereinfacht außerdem die Freigabe des Speichers, der nicht mehr benötigt wird. Sowohl das ASP-Application-Objekt als auch das ASP-Session-Objekt bieten eine `Contents`-Eigenschaft an, die über eine `Remove`- und eine `RemoveAll`-Methode verfügt. Der SPM weist keine ähnliche Funktionalität auf.

Unabhängig davon, ob Sie das ASP-Application-Objekt oder den SPM benutzen, um applikationsweite Daten zu überwachen, müssen Sie während der anfänglichen Design-Phase entscheiden, ob Ihr Code in einer Web-Farm-Umgebung ausgeführt werden soll. Wenn Clients über mehrere IIS-Servercomputer weitergeleitet werden, ist es sehr schwierig, aktualisierbare Daten im Cache abzulegen. Sie müssen häufig davon ausgehen, dass die Daten im Cache nur gelesen werden können, nachdem Sie aus einer Datenbank geladen wurden.

Das Ablegen von schreibgeschützten Daten im Cache des Webserver ist eine der besten Möglichkeiten, um die Reaktionszeiten zu verbessern und den Gesamtdurchsatz zu erhöhen. Cache-Techniken sind sehr nützlich, da sie die erforderliche Anzahl von Datenübertragungen zur und von der Datenbank verringern. Sie können darüber hinaus serverseitige Bearbeitungszyklen schonen. Nachdem beispielsweise die *Products*-Tabelle in eine ADO-Datensatzgruppe ausgelesen wurde, können Sie die Daten in eine HTML-Tabelle oder in das XML-Format umwandeln und als Zeichenfolgenwert im Cache einer ASP-Application-Variablen ablegen. Diese Technik verringert die Anzahl der notwendigen Datenkonvertierungen für jede Anforderung.

Wenn Sie ein Cache-Schema für die mittlere Schicht und schreibgeschützte Daten entwerfen möchten, sollten Sie darüber nachdenken, wie lange Ihre Daten aktuell bleiben. In einigen Fällen können Sie sich dazu entschließen, die Daten entweder nur einmal während des Starts der Anwendung oder während des ersten Zugriffs in ASP-Application-Variablen laden. In anderen Applikationen lassen Sie die Daten vielleicht in regelmäßigen Abständen aktualisieren.

Dazu ein Beispiel, das Ihnen zeigt, wie Sie ein zusammenhängendes Cache-Schema entwerfen. Stellen Sie sich vor, Sie möchten eine HTML-Tabelle für eine Artikelliste in einer `Application`-Variablen zwischenspeichern. Sie gehen davon aus, dass die sich die Artikeldaten derart oft verändern, dass die HTML-Tabelle in Abständen von fünf Minuten aktualisiert werden soll.

In Ihrem Design erstellen Sie eine ASP-Application-Variable, um die eigentliche Artikelliste aufzunehmen, und eine weitere Application-Variable, um zu überwachen, wann die Liste zuletzt von der Datenbank aktualisiert wurde. Jede ASP-Seite oder Visual Basic-Komponente kann somit den Zeitpunkt der letzten Aktualisierung ermitteln, wenn sie die Liste abrufen. Wenn der Code, der in einer Anforderung ausgeführt wird, feststellt, dass das Intervall zwischen der aktuellen Zeit und dem Zeitpunkt der letzten Aktualisierung größer als das Aktualisierungsintervall ist, kann er die Datenbank erneut abfragen und die Liste auf diese Weise für sich selbst und alle anderen Anforderung der nächsten fünf Minuten aktualisieren. Wenn Sie ein progressiver COM+-Designer sind, können Sie das Aktualisierungsintervall auch in der Konstruktorzeichenfolge einer konfigurierten Komponente aufbewahren. Sie können das Intervall dann einstellen, ohne den Code erneut kompilieren zu müssen.

Zusammenfassung

Dieses Kapitel hat einige wichtige Themen zum Erstellen umfassend skalierbarer Applikationen mit Visual Basic behandelt. Applikationen, die HTTP für die Client-/Server-Kommunikation benutzen, sind skalierbarer als Applikationen, die DCOM und RPC verwenden. Das Web und HTTP eröffnen Ihnen außerdem eine Welt der plattformübergreifenden Entwicklung.

Wenn Sie Websites mit Windows 2000 Server entwerfen, sollten Sie die Architektur von IIS und ASP verstehen. Diese Kerntechnologien gestatten Ihnen, den Clients ASP-Seiten und Visual Basic-Komponenten über das Internet und das unternehmenseigene Intranet zur Verfügung zu stellen. Wie Sie gesehen haben, betreffen viele Technologieaspekte das Erstellen von ASP-Seiten und Visual Basic-Komponenten. Viele Programmiertechniken und Konfigurationsschemata hindern eine Applikation leider daran, den Anforderungen an die Reaktionszeiten und den Gesamtdurchsatz mit Hilfe einer angemessenen Skalierung gerecht zu werden.

Ich hoffe, Sie erinnern sich daran, dass Sie in der frühen Design-Phase lange und intensiv über die Skalierbarkeit nachdenken sollten. Viele Designer, die entwickeln, ohne an die Skalierbarkeit zu denken, bereuen nun die Abhängigkeiten, die ihre Applikationen von ASP-Session-Variablen aufweisen. Ihre Applikationen werden möglicherweise niemals in der Oberliga mitspielen.

Es ist häufig nicht möglich vor auszusehen, wie beschäftigt und erfolgreich Ihre Site sein wird. Wenn Sie wissen, dass Ihre Site niemals mehr als nur wenige Anforderungen pro Minute erhalten wird, müssen Sie sich nicht um viele Details kümmern, die in diesem Kapitel genannt wurden. Wenn Sie jedoch nicht sicher sind, sollten Sie jede Chance wahrnehmen. Beginnen Sie den Entwurf einer Webapplikation mit der Annahme, dass Ihre Site eines Tages die beliebteste Site des Internets sein wird.

10 Messaging und die asynchrone Kommunikation

286	Warum ist das Messaging wichtig?
290	MSMQ
311	Warteschlangenkomponenten
325	Der COM+ Events-Dienst
333	Die Wahl zwischen MSMQ, QC und COM+ -Events

Messaging ist traditionell ein wesentlicher Bestandteil der verteilten Applikationsprogrammierung. Dies gilt besonders für große Systeme. Das Messaging bietet eine zusätzliche Dimension, die Sie mit synchronen Protokollen, wie zum Beispiel RPC und HTTP, nicht erreichen können. Dieses Kapitel beschreibt, wie Messaging-Produkte, wie zum Beispiel IBM MQSeries und Microsoft Message Queuing (MSMQ), den Unternehmen helfen, die Vorteile des Messaging zu nutzen, ohne selbst eine komplexe Kommunikationsinfrastruktur aufbauen zu müssen.

Dieses Kapitel erklärt, wieso und wann Sie das Messaging in einer verteilten Applikation verwenden sollten. Es beschreibt außerdem, wie Nachrichtenwarteschlangen einer OLTP-Applikation (Online Transaction Processing) Reaktionsvermögen und Zuverlässigkeit hinzufügen können. Dies ist mit anderen Mitteln nicht möglich. Ich werde Ihnen darüber hinaus zeigen, wie Applikationen geschrieben werden, die MSMQ-Nachrichten mit Visual Basic versenden und empfangen. Ich kann Ihnen natürlich nicht jeden Aspekt der MSMQ-Programmierung in nur einem Kapitel demonstrieren. Mein Ziel besteht jedoch darin, Ihnen eine Einführung zu bieten, indem ich Ihnen die gebräuchlichsten Techniken zum Erstellen von Applikationen zeige, die MSMQ-Nachrichten senden und empfangen.

Das Kapitel beschreibt ebenfalls die transaktionale Programmierung mit MSMQ. Sie müssen transaktionale Nachrichten an transaktionale Warteschlangen übermitteln, um eine einmalige Sendung zu erhalten. Sie lernen zwei einfache Techniken zum Versenden und Empfangen transaktionaler Nachrichten mit MSMQ und Visual Basic kennen. Sie können entweder den internen Transaktionsmechanismus von MSMQ oder eine COM+-Transaktion verwenden, die den DTC (Distributed Transaction Coordinator) nutzt.

Nachdem MSMQ in diesem Kapitel beschrieben wurde, werden Sie den Warteschlangenkomponenten-Dienst kennen lernen, der Bestandteil von COM+ und Windows 2000 ist. Dieser Dienst ist ein benutzerfreundliches Gerüst, das auf MSMQ aufsetzt. Warteschlangenkomponenten bieten die Vorteile von MSMQ, ohne Sie mit der Komplexität der MSMQ-Programmierung zu konfrontieren. Sie werden noch sehen, dass ein Verständnis von MSMQ sehr hilfreich sein kann, wenn Sie erfahren möchten, wie Warteschlangenkomponenten arbeiten.

Dieses Kapitel beschreibt ebenfalls den COM+ -Events-Dienst, mit dessen Hilfe Sie Applikationen erstellen, die Ereignisbenachrichtigungen für andere Applikationen zur Verfügung stellen. So, wie Sie die Grundlagen von MSMQ verstehen sollten, bevor Sie mehr über Warteschlangenkomponenten erfahren, sollten Sie wissen, wie Warteschlangenkomponenten entworfen und konfiguriert werden, wenn Sie die COM+-Ereignisse bestmöglich nutzen möchten.

Das Kapitel bietet keine vollständige Referenz zu all diesen Themen. Es erklärt jedoch die entsprechenden Strukturen und deren Zusammenwirken und hilft Ihnen, sich während der Design-Phase zwischen MSMQ, Warteschlangenkomponenten und COM-Standardmethodenaufrufen zu entscheiden. Sie lernen die Stärken und Einschränkungen von COM+-Ereignissen kennen und erfahren, ob diese für Ihre Applikation geeignet sind. Wenn Sie Applikationen schreiben, die MSMQ oder Warteschlangenkomponenten verwenden, sollten Sie die Plattform-SDK-Dokumentation und vielleicht auch andere Quellen lesen, die das Material in diesem Kapitel ergänzen.

Warum ist das Messaging wichtig?

Die grundlegende Idee, die sich hinter dem Messaging verbirgt, ist recht einfach. Eine Nachricht ist eine Anforderung oder ein anderer Benachrichtigungstyp, der von einer Applikation zu einer anderen gesendet wird. Eine Warteschlange ist ein benannter, geordneter Nachrichtenspeicher, der auf einem bestimmten Computer vorhanden ist. Einige Applikationen führen Anforderungen durch, indem sie Nachrichten an die Warteschlange senden, während andere Applikationen die Nachrichten aus der Warteschlange entfernen und die Verarbeitung durchführen, die von dem Versender angefordert wurde. Abbildung 10.1 zeigt, wie eine Nachrichtenwarteschlange gewöhnlich in einer verteilten Applikation verwendet wird.

Eine Versender-Applikation erstellt und bereitet eine Nachricht vor, indem Sie unterschiedliche Eigenschaften in dem Header der Nachricht setzt und parametrisierte Informationen in ein Gerüst einfügt, das als Nachrichtenkörper bezeichnet wird. Die Versender-Applikation schreibt die Nachricht anschließend in die Warteschlange. Eine Empfänger-Applikation entfernt die Nachricht aus der Warteschlange und liest sie, so dass die Nachricht nur einmal bearbeitet wird. Die Empfänger-Applikation interpretiert dann die Anforderung, entnimmt die parametrisierten Informationen aus dem Nachrichtenkörper und führt jede Anweisung aus, die von dem Nachrichtensender angefordert wurde.

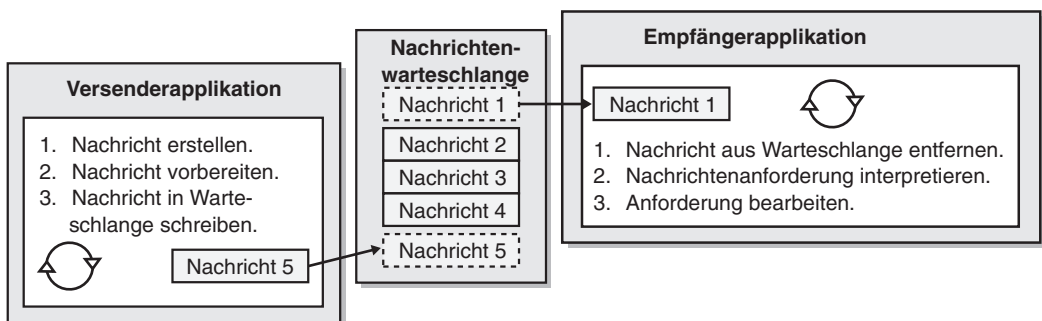


Abbildung 10.1: Eine Versender-Applikation erstellt und bereitet eine Nachricht vor, die in die Warteschlange geschrieben wird. Eine Empfänger-Applikation entnimmt die Nachricht aus der Warteschlange, interpretiert die Anforderung der Nachricht und führt die Anweisungen aus, die zur Bearbeitung der Anforderung erforderlich sind.

Nachrichtenwarteschlangen unterstützen auch das Konzept einer Leser-Applikation. Dieser Applikationstyp kann eine Nachricht in der Warteschlange überprüfen, ohne diese zu entfernen. Dieser Vorgang wird als *Peeking* bezeichnet. Das *Peeking* gestattet einer Applikation, die Nachrichten auszuwählen, die aus der Warteschlange entfernt werden sollen.

Eine Applikation kann mehrere dieser Rollen annehmen. Es ist üblich, dass eine Applikation sowohl als Leser als auch als Empfänger für eine einzelne Warteschlange eingesetzt wird. Die Applikation kann mit Hilfe des *Peekings* nachsehen, welche Nachrichten in der Warteschlange enthalten sind, und lediglich die Nachrichten entnehmen, die bestimmte Kriterien erfüllen. Eine Applikation kann auch Nachrichten aus einer Warteschlange empfangen, während sie andere Nachrichten an eine zweite Warteschlange sendet. Es ist zum Beispiel üblich, eine Arbeitsablaufsapplikation zu erstellen, in der verschiedene Programme sowohl über Nachrichteneingangswarteschlangen als auch über Nachrichtenausgangswarteschlangen verfügen. Die Ausgangswarteschlange eines Programms ist die Eingangswarteschlange eines anderen Programms. Nachdem ein Programm eine Anforderung bearbeitet hat, leitet es die Anforderung einfach an das nächste Programm im Arbeitsablauf weiter.

Ein Messaging-Subsystem ist ein nützliches Element in einem verteilten System, da es Applikationen gestattet, Anforderungen auszuführen und Informationen zu anderen Applikationen zu senden. Dies geschieht in einer verbindungslosen, asynchronen Weise.

Messaging im Vergleich mit RPC und HTTP

Das Übergeben von Nachrichten gleicht in gewisser Hinsicht einem synchronen Protokoll, wie zum Beispiel RPC oder HTTP. Andererseits bestehen gravierende Unterschiede zwischen diesen Verfahren. Wie Sie in diesem Buch gesehen haben, verwendet COM RPC, um Interprozess-Anforderungen zwischen Clients und Objekten zu übermitteln. Im neunten Kapitel wurde beschrieben, wie die Computer in einer Webapplikation unter Verwendung von HTTP miteinander kommunizieren. Es wichtig, dass Sie verstehen, dass RPC und HTTP synchrone Protokolle sind. Diese synchrone Natur führt zu einigen erwähnenswerten Einschränkungen, deren Erläuterung ein wenig Zeit erfordert.

Lassen Sie uns einige bekannte Probleme der synchronen Kommunikation betrachten, die nicht auftreten, wenn Sie das Messaging verwenden. Das erste Problem eines synchronen Protokolls, wie zum Beispiel RPC oder HTTP, besteht darin, dass der Thread einer Client-Applikation während einer Anforderung blockiert ist. Die Client-Applikation muss warten, bis der Servercomputer die angeforderte Aufgabe ausgeführt hat und eine Antwort zurückgibt. Wenn der Servercomputer außerdem mit vielen Anforderungen von anderen Clients im Rückstand ist, kann eine Client-Applikation für eine längere Zeit blockiert sein, wenn sie auf eine Antwort wartet.

Ein Messaging-Protokoll zwingt eine Client-Applikation hingegen nicht, zu warten. Eine Client-Applikation kann ihre Arbeit sofort fortsetzen, nachdem sie eine asynchrone Anforderung abgeschickt hat. Sie können die Steuerung auf diese Weise schneller an den Client zurückgeben, um die Reaktionszeit zu verbessern. Der Nachteil besteht darin, dass Sie den Client nicht umgehend über die Geschehnisse informieren können, da die Nachricht nicht bearbeitet wird, bis der Client die Steuerung zurückerhält. Die Hauptvoraussetzung des Messaging ist somit, dass etwas getan werden muss, aber nicht sofort. Für den Server ist es akzeptabel, die Anforderung in wenigen Sekunden, einer Stunde oder nach einigen Tagen zu bearbeiten.

Das zweite Problem eines synchronen Protokolls besteht darin, dass es das Einrichten einer Verbindung zwischen zwei bestimmten Computern erfordert. Im Fall einer einfachen Anforderung, die von einem Client-Computer zu einem Servercomputer gesendet wird, müssen die Client-

Applikation und die Server-Applikation gleichzeitig online sein. Wenn zum Beispiel keine Verbindung zum Server hergestellt werden kann, kann der Client keine Anforderungen versenden. Wenn der Server keine Verbindung zum Client herstellen kann, kann er keine Anforderungen bearbeiten. Keine der beiden Seiten kann somit die notwendigen Operationen ausführen, wenn die jeweils andere Seite nicht gestartet wurde und ausgeführt wird.

Die Anforderungen an die verbindungsorientierte Kommunikation sind in Systemen, in denen Anforderungen über mehrere Server geleitet werden, sogar noch höher. Abbildung 10.2 zeigt eine physikalische Installation, in der eine Client-Anforderung zwischen drei Servercomputern bewegt wird.

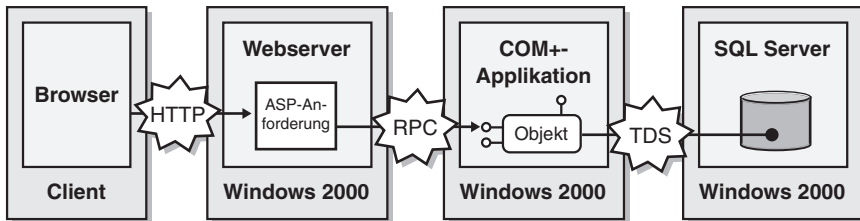


Abbildung 10.2: Applikationen, die ausschließlich synchrone verbindungsorientierte Protokolle verwenden, weisen keine ausgeprägte Fehlertoleranz auf. Jeder Servercomputer muss online sein, damit das gesamte System betriebsfähig ist.

Der Client sendet mit Hilfe des HTTP-Protokolls eine Anforderung an den Webserver. Diese Anforderung startet wiederum eine ASP-Seite auf dem Webserver. Die ASP-Seite verwendet DCOM und RPC, um eine Methode eines Geschäftsobjekts aufzurufen, das auf einem Applikationsserver ausgeführt wird. Der Applikationsserver stellt mit dem nativen OLE-DB-Provider eine ADO-Verbindung zu SQL Server her. Der Provider benutzt dazu ein proprietäres synchrones Protokoll mit der Bezeichnung Tabular Data Streams (TDS). Beachten Sie, dass alle drei Servercomputer online und einsatzbereit sein müssen, damit der Client eine Anforderung abschicken und eine Erfolgsmeldung empfangen kann. Wenn einer der Servercomputer vom Netzwerk getrennt wird, reagiert auch die Applikation nicht mehr.

Wie Sie sehen, können verteilte Applikationen, die ausschließlich synchrone Protokolle verwenden, keine ausgeprägte Fehlertoleranz und Verfügbarkeit bieten. Ein Applikationsserver oder ein DBMS (Database Management System) könnte aufgrund eines Systemabsturzes oder bedingt durch Wartungsarbeiten vom Netzwerk getrennt werden.

Ein Messaging-Subsystem bietet eine bessere Fehlertoleranz und Verfügbarkeit, da es als Puffer zwischen der Applikation, die eine Anforderung versendet, und einer zweiten Applikation dient, die die Anforderung bearbeitet. Client-Applikationen können unabhängig davon, ob die Empfängerapplikation online ist, weiterhin Anforderungen an die Warteschlange senden. Wenn die Server-Applikation wieder online ist, kann sie die Bearbeitung der akkumulierten Nachrichten fortsetzen. Ein Server kann seine Arbeit außerdem auch dann weiterführen, wenn alle Client-Applikationen offline sind. Wie Sie sehen, agiert die Warteschlange als Puffermechanismus, der beiden Seiten erlaubt, die eigene Arbeit in Abwesenheit der jeweils anderen Seite zu verrichten.

Das Beispiel in Abbildung 10.3 gleicht dem in Abbildung 10.2, mit einer Ausnahme. Die Kommunikation zwischen dem Webserver und dem Applikationsserver wird mit MSMQ-Nachrichten und nicht mit DCOM oder RPC durchgeführt. Der Einsatz eines asynchronen Protokolls bedeutet, dass Clients auch dann Anforderungen abschicken können, wenn der Applikationsserver oder der Datenbankserver offline ist. Die Infrastruktur von MSMQ speichert Nachrichten ein-

fach auf dem Webserver, bis diese an den Applikationsserver weitergeleitet werden können. Der Applikationsserver oder der Datenbankserver können somit vom Netzwerk getrennt werden, ohne Wartezeiten für die gesamte Applikation zu verursachen.

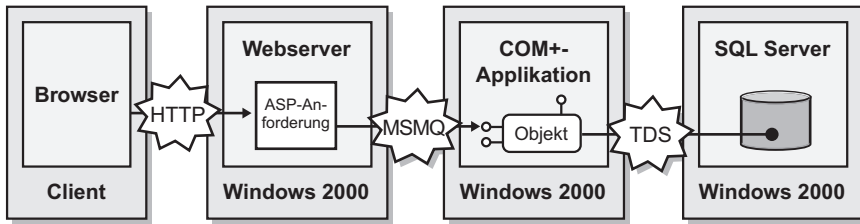


Abbildung 10.3: Das Messaging bietet eine bessere Fehlertoleranz, da Clients auch dann weiterhin Anforderungen abschicken können, wenn ein oder mehrere Servercomputer offline sind.

Das dritte Problem synchroner Protokolle besteht darin, dass Anforderungen, die zuerst einge-
hen, in der Regel auch zuerst bedient werden. Es gibt keine Möglichkeit, den Aufrufen mit RPC
oder HTTP eine Priorität zuzuweisen. Eine Anforderung mit einer höheren Priorität muss auf die
Bearbeitung warten, wenn zuvor Anforderungen mit einer geringeren Priorität eingegangen sind.
Ein Messaging-Subsystem, wie zum Beispiel MSMQ, löst dieses Problem, indem den Nachrich-
ten Prioritätsebenen zugewiesen werden. Nachrichten mit einer höheren Priorität werden an der
obersten Position der Warteschlange angeordnet, während Nachrichten mit einer niedrigen Prio-
rität am Ende platziert werden. Der Server kann somit zuerst auf die wichtigen Nachrichten rea-
gieren.

Das vierte und möglicherweise signifikanteste Problem besteht darin, dass RPC und HTTP für
Fehler anfällig sind, die zu inkonsistenten Ergebnissen in OLTP-Applikationen führen. Lassen
Sie uns überprüfen, was beispielsweise geschehen kann, wenn eine Client-Applikation eine RPC-
basierte Methode aufruft, um eine COM+-Transaktion auszuführen. Dies kann zu drei Fehlern
führen: Zunächst kann die RPC-Anforderungsnachricht des Methodenaufrufs zwischen dem
Client und dem Server verloren gehen. Zweitens kann die COM+-Applikation abstürzen, wäh-
rend der Methodenaufruf die Transaktion ausführt. In beiden Szenarien werden die von der
Transaktion beabsichtigten Änderungen nicht gespeichert. Der dritte Fehler tritt auf, wenn die
Antwortnachricht des Methodenaufrufs auf dem Weg zum Client fehlschlägt oder verloren geht,
nachdem die Methode ausgeführt und die Transaktion gespeichert wurde.

Was geschieht aber, wenn ein Client mit DCOM und RPC eine Methode auf einem transaktiona-
len Objekt über das Netzwerk ausführt und keine Antwort erhält? Es gibt keine Möglichkeit, um
festzustellen, was geschehen ist. Wenn der Client dieselbe Anforderung noch einmal einreicht,
wird die Transaktion möglicherweise ein zweites Mal gespeichert. Sie erkennen an diesem Bei-
spiel, dass DCOM und RPC keine Garantien für eine einmalige Versendung bieten können.
HTTP ist für die gleichen Probleme anfällig.

Das transaktionale Messaging ermöglicht das Abschicken von Transaktionsanforderungen mit ei-
ner so genannten Einmal-Übermittlungssemantik. Sie können eine Transaktionsanforderung mit
einer Einmal-Semantik ausführen lassen, indem sie die Anforderung in drei kleinere Transaktio-
nen unterteilen. In der ersten Transaktion sendet die Client-Applikation eine Anforderungsnach-
richt an eine Anforderungswarteschlange auf dem Server. In der zweiten Transaktion empfängt
eine so genannte Listener-Applikation die Anforderungsnachricht, bearbeitet diese und sendet
eine Antwortnachricht an die Anforderungswarteschlange des Clients. In der letzten Transaktion
empfangt die Client-Applikation die Antwortnachricht, um festzustellen, was geschehen ist.

Wichtig ist, dass RPC nicht besser als die Nachrichtenübergabe ist, da es dem Client nicht erlaubt, den exakten Status einer Anforderung zu ermitteln. Später in diesem Kapitel, wenn die Grundlagen der MSMQ-Programmierung behandelt wurden, werde ich zu diesem Thema zurückkehren und beschreiben, wie eine Applikation eingerichtet wird, die Ihnen das Ausführen von Transaktionen mit einer Einmal-Semantik gestattet.

Nachdem Sie nun erfahren haben, wie Warteschlangen Ihre verteilten Applikationen verbessern, verwenden wir dieses Wissen, um mit MSMQ zu arbeiten. Wenn Sie mit MSMQ programmieren, werden Sie feststellen, dass dies einen größeren Aufwand als die Programmierung von COM-basierten Methodenaufrufen erfordert. Wenn Sie DCOM und RPC benutzen, erzielen Sie die Interprozess-Kommunikation möglicherweise ohne diesen Aufwand. Die Einschränkungen von RPC rechtfertigen jedoch die Energie, die Sie in MSMQ investieren werden.

MSMQ

MSMQ 2 ist in Windows 2000 integriert. Es ist nicht mit früheren Versionen von Windows kompatibel. MSMQ 1 kann unter Windows NT, Windows 95 und Windows 98 ausgeführt werden. Es ist nicht schwierig, eine Zusammenarbeit zwischen diesen beiden Versionen von MSMQ zu erreichen, wenn sich bereits Windows NT- und Windows 98-Computer in einer Windows 2000-Domäne befinden.

MSMQ 2 benutzt Windows 2000 Active Directory, um einen globalen Katalog mit Konfigurationsdaten zu pflegen. Der MSMQ-Katalog gilt für viele Windows 2000-Domänen und überwacht die Profile von MSMQ-Entitäten, wie zum Beispiel Computern, öffentlichen Warteschlangen und Sicherheitszertifikaten. Jeder MSMQ-Katalog definiert gewissermaßen eine MSMQ-Organisationseinheit.

Die MSMQ-Organisationseinheit

Um eine Organisationseinheit mit MSMQ-Computern zu erstellen, müssen Sie MSMQ zunächst auf einem Windows 2000 Computer installieren, der als Domänencontroller konfiguriert ist, auf dem Active Directory installiert ist. Ein Windows 2000-Domänencontroller wird häufig auch als Active Directory-Server bezeichnet. Diese erste Installation von MSMQ auf einem Active Directory-Server führt dazu, dass der Katalog der MSMQ-Organisationseinheit in Active Directory generiert wird. Eine Windows 2000-Gruppe kann über mehrere Active Directory-Server verfügen, auf denen MSMQ installiert ist. Jeder Active Directory-Server besitzt eine Kopie des MSMQ-Katalogs.

Nachdem Sie MSMQ auf einem Active Directory-Server installiert haben, ist es sehr einfach, MSMQ auf anderen Computern zu installieren, damit diese der MSMQ-Organisationseinheit hinzugefügt werden. Bevor Sie MSMQ installieren, müssen Sie lediglich sicherstellen, dass sich die anderen Computer in einer Domäne der Gruppe befinden, der auch der erste Active Directory-Server zugehört.

Die hier beschriebene Konfiguration wird auch MSMQ 2-Enterprise-Modus genannt. Sie können eine Gruppe mit Computern auch im MSMQ 2-Arbeitsgruppenmodus einrichten. Dieser Modus erlaubt Ihnen, MSMQ ohne Windows 2000-Domänencontroller oder Active Directory zu benutzen. Der MSMQ 2-Enterprise-Modus lässt das MSMQ-Netzwerk jedoch sehr viel leistungsfähiger werden. Das Kapitel beschreibt deshalb nur den Einsatz von MSMQ für Computer, die Active Directory in einer Windows 2000-Domäne benutzen.

Ihnen stehen verschiedene Möglichkeiten zur Verfügung, um MSMQ auf einem Computer zu konfigurieren, der kein Active Directory-Server ist. Sie können einen MSMQ-Computer als Routing-Server, als unabhängigen Client oder als abhängigen Client einrichten. Ein Routing-Server kann seine eigenen Warteschlangen besitzen. Wie der Name impliziert, kann er auch als Speicher- und Weiterleitungsagent für solche Nachrichten agieren, deren Ziel die Warteschlange eines anderen MSMQ-Computers ist. Dies ist hilfreich, wenn Client-Applikationen ihre Nachrichten nicht direkt zum Zielsystem senden können. Der Routing-Server ist eine Zwischenstation zwischen dem Computer, der eine Nachricht sendet, und dem Zielsystem, der die Zielwarteschlange verwahrt. Beachten Sie, dass ein Active Directory-Server, auf dem MSMQ installiert ist, dieselben Aufgaben wie ein Routing-Server durchführt. Er speichert jedoch zusätzlich zu diesen Verantwortlichkeiten eine Kopie des MSMQ-Katalogs.

Sie können einen MSMQ-Computer auch als unabhängigen Client konfigurieren, der keine Speicher- und Weiterleitungsfunktion erfüllt, aber eigene lokale Warteschlangen erstellen und verwalten kann. Diese Installation ist eine gute Wahl für einen Remote-Computer, wie zum Beispiel ein Notebook, da eine Client-Applikation auch dann Nachrichten an eine Netzwerkwarteschlange senden kann, wenn das Notebook nicht daran angeschlossen ist. In einer solchen Situation erstellt MSMQ eine temporäre lokale Warteschlange im Hintergrund und speichert versendete Nachrichten darin. Später, wenn der unabhängige Client-Computer wieder eine Verbindung zum Netzwerk herstellt, leitet die MSMQ-Infrastruktur die Nachrichten automatisch an die Zielwarteschlange weiter.

Ein abhängiger Client kann keine lokalen Warteschlangen pflegen. Er ist nicht betriebsfähig, wenn er vom Netzwerk getrennt wird. Er muss eine Verbindung zu einem Active Directory-Server oder einem Routing-Server herstellen, um Nachrichten empfangen und versenden zu können. Ein Desktop-Computer, der immer mit dem LAN verbunden ist, ist in der Regel die einzige Maschine, die Sie als abhängigen Client verwenden können.

MSMQ verwendet das Konzept einer Windows 2000-Site. Dies ist eine Gruppe mit Computern in einem oder mehreren Subnets, die mit einer sehr hohen Geschwindigkeit aufeinander zugreifen können. Eine Site ist in der Regel eine Gruppe mit Computern, die im selben LAN ausgeführt werden. Jede Windows 2000-Site, die MSMQ-Computer enthält, sollte einen oder mehrere Active Directory-Server besitzen, so dass Sie den MSMQ-Katalog abfragen können, ohne Aufrufe über mehrere Sites hinweg durchführen zu müssen. Active Directory speichert automatisch eine Kopie des MSMQ-Katalogs auf jedem Domänencontroller und aktualisiert diese Daten mit Hilfe der Replikation.

Sie sollten verstehen, dass das Versenden von Nachrichten über mehrere Sites hinweg langsamer und weniger zuverlässig als der Nachrichtenversand zwischen zwei Computern innerhalb derselben Site ist. MSMQ erlaubt den Administratoren, jeder Verbindung zu einer anderen Site eine gewisse Belastung zuzuweisen. MSMQ kann dann einen Algorithmus ausführen, um den optimalen Weg für die Versendung von einer Site zu einer anderen zu ermitteln. Dieses administrative Detail ist ein wichtiger Aspekt der Optimierung von MSMQ-Organisationseinheiten, sofern die Computer ihre Nachrichten über mehrere Sites hinweg versenden, die sich wiederum über große geographische Distanzen erstrecken.

Der Warteschlangen-Manager

Die Kommunikationsinfrastruktur von MSMQ verwendet eine Komponente, die als Warteschlangen-Manager bezeichnet wird. Der Warteschlangen-Manager wird als Windows-Dienst ausgeführt und über MQSVC-EXE gestartet. MSMQ installiert einen Warteschlangen-Manager auf jedem Computer, der eine Speicher- und Weiterleitungsverantwortlichkeit oder eigene lokale

Warteschlangen besitzt. Der einzige MSMQ-Typ, der keinen eigenen Warteschlangen-Manager erhält, ist der abhängige Client.

Jedem Warteschlangen-Manager wird während der Installation eine eindeutige GUID zugewiesen. MSMQ verwendet diese GUID, um Anforderungen zu überwachen, die von einem spezifischen Warteschlangen-Manager erstellt wurden und sich in dessen Besitz befinden. Es verwendet diese GUID ebenfalls, um Nachrichten zu überwachen, die von dem Warteschlangen-Manager stammen. Beachten Sie bitte, dass ein Computer niemals mehr als einen Warteschlangen-Manager besitzt. Dies bedeutet, dass die GUID eines Warteschlangen-Managers auch den Computer identifiziert, auf dem der Manager installiert ist. MSMQ überwacht diese GUID zusammen mit den Informationen, die für die Auflösung der Computeradresse benötigt werden, im Active Directory.

Der Warteschlangen-Manager ermöglicht somit die Kommunikation in MSMQ. Eine Versender-Applikation muss eine Nachricht an den Warteschlangen-Manager übertragen. Wenn der Warteschlangen-Manager der Besitzer einer Zielwarteschlange ist, speichert er die Nachricht in dieser Warteschlange. Wenn der die Zielwarteschlange nicht besitzt, leitet er die Nachricht an den nächsten Warteschlangen-Manager weiter. Wenn der Computer mit der Zielwarteschlange nicht erreicht werden kann, speichert der Warteschlangen-Manager die Nachricht temporär und leitet sie später weiter, wenn eine Kommunikation eingerichtet werden kann. Die Empfänger- und die Leser-Applikationen kommunizieren mit dem Warteschlangen-Manager auf dem Computer, der die Warteschlange verwahrt, um Nachrichten zu empfangen oder zu lesen.

Ich habe bereits erwähnt, dass MSMQ einen globalen Katalog in Active Directory pflegt. Jeder Warteschlangen-Manager verwahrt ebenfalls zusätzliche Daten für lokale Warteschlangen. Dies geschieht an einem Ort, der als LQS (Lokaler Warteschlangenspeicher – Local Queue Store) bezeichnet wird. Der LQS besteht aus einer Datei für jede lokale Warteschlange sowie aus konfigurierten Daten. Der Warteschlangen-Manager benutzt diese Daten, um seine lokalen Warteschlangen zu verwalten. Beachten Sie, dass der LQS häufig Informationen dupliziert, die sich im Active Directory befinden. Dies geschieht, da der Warteschlangen-Manager den LQS verwenden kann, um die Anzahl der Datenübertragungen über das Netzwerk zu verringern und gelegentlich Nachrichten zu bearbeiten, wenn keine Verbindung zu einem Active Directory-Server hergestellt werden kann.

MSMQ-Programmierung

Ihnen stehen zwei Möglichkeiten zur Auswahl, um mit MSMQ zu programmieren. MSMQ bietet sowohl eine C-API als auch eine COM-Komponentenbibliothek an. Als Visual Basic-Programmierer verwenden Sie natürlich die COM-Komponentenbibliothek, wenn Sie mit MSMQ programmieren. Beachten Sie aber, dass einige MSMQ-Funktionen lediglich über die C-API ausgeführt werden können. Sie können die meisten Anforderungen für die Nachrichtenübergabe in einer verteilten Applikation jedoch mit der Komponentenbibliothek erfüllen. In den wenigen Situationen, in denen Sie die C-API benötigen, können Sie die `Declare`-Anweisung von Visual Basic verwenden, um einen direkten Aufruf einer API-Funktion vorzunehmen. Dieses Kapitel beschreibt jedoch ausschließlich den Einsatz der MSMQ-Komponentenbibliothek. Sobald Sie einen Verweis auf die MSMQ-Typbibliothek (die *Microsoft Message Queue 2.0 Object Library*) in Ihr Visual Basic-Projekt übernommen haben, können Sie diese MSMQ-Komponenten in Ihrem Code verwenden.

MSMQ stellt mehrere Komponenten zur Verfügung, die einige Eigenschaften und Methoden besitzen. Ich werde nicht jede Möglichkeit der MSMQ-Programmierung mit MSMQ-Komponenten

beschreiben. Stattdessen werde ich Ihnen MSMQ-Programmierbeispiele vorstellen, die den gebräuchlichen Einsatz dieser Technik in einer verteilten Applikation demonstrieren.

Ich setze voraus, dass Sie MSMQ auf einem Windows 2000-Computer installiert haben, der als Active Directory-Server, Routing-Server oder unabhängiger Client konfiguriert ist. Diese MSMQ-Installationstypen erlauben Ihnen nicht nur, MSMQ-Programme zu schreiben und zu testen. Sie gestatten Ihnen ebenfalls, das administrative Hilfsmittel *Computerverwaltung* zu benutzen, um bestimmte administrative MSMQ-Aufgaben zu verrichten. Dieses Hilfsmittel (das in Abbildung 10.4 dargestellt ist) lässt Sie Warteschlangen und Nachrichten verwalten. Sie starten es, indem Sie das Startmenü öffnen und dort *Programme* und anschließend *Verwaltung* auswählen. Klicken Sie nun auf *Computerverwaltung*.

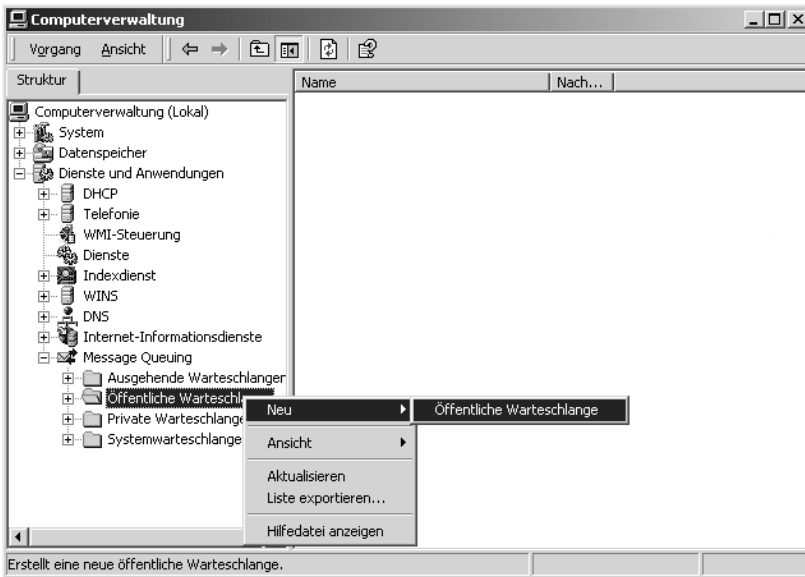


Abbildung 10.4: Das administrative Hilfsmittel *Computerverwaltung* lässt Sie lokale Warteschlangen erstellen und verwalten.

Erstellen einer öffentlichen Warteschlange

Wir beginnen unsere Tour durch die MSMQ-Programmierung mit der Erstellung einer Warteschlange. MSMQ lässt Sie sowohl öffentliche als auch private Warteschlangen erstellen. Ihnen stehen zwei Möglichkeiten zur Verfügung, um eine öffentliche Warteschlange zu erstellen: manuell, mit dem administrativen Hilfsmittel *Computerverwaltung* oder programmgesteuert.

Wir besprechen zunächst die manuelle Erstellung. Starten Sie das administrative Hilfsmittel *Computerverwaltung* und führen Sie einen Rechtsklick auf den Ordner *Öffentliche Warteschlange* aus. Wählen Sie den Eintrag *Öffentliche Warteschlange* aus dem *Neu*-Menü aus. Sie geben der Warteschlange einen Namen und bestimmen, ob die Warteschlange transaktional sein soll. Transaktionale Warteschlangen werden später in diesem Kapitel erörtert. Erstellen Sie zunächst eine Warteschlange, die nicht transaktional ist. Deaktivieren Sie dazu das Kontrollkästchen *Auf Transaktionen ausgelegt*. Betätigen Sie die Schaltfläche *OK*, um die Warteschlange erstellen zu lassen.

Nachdem Sie die Warteschlange erzeugt haben, können Sie deren Attribute überprüfen, indem Sie einen Rechtsklick auf die Warteschlange ausführen und anschließend *Eigenschaften* auswählen. Im Anschluss daran wird ein Registerdialogfeld angezeigt, in dem Sie die Eigenschaften der Warteschlange modifizieren können. Sie werden dabei feststellen, dass MSMQ automatisch eine GUID generiert hat, die die Warteschlange identifiziert. Die mit einer öffentlichen Warteschlange verknüpfte GUID wird auch als Instanz-ID bezeichnet.

Sie können eine Warteschlange auch programmgesteuert erstellen, indem Sie ein `MSMQQueueInfo`-Objekt benutzen. Sie erstellen das Objekt zunächst und weisen ihm anschließend einen zulässigen `PathName` zu. Der `PathName` einer Warteschlange sollte aus dem Namen des Computers und dem Namen der Warteschlange bestehen. Der folgende Code verwendet ein `MSMQQueueInfo`-Objekt, um eine neue Warteschlange zu erstellen:

```
Dim qi As MSMQQueueInfo
Set qi = New MSMQQueueInfo
qi.PathName = "MyComputer\MyQueue"
qi.Label = "My Queue"
qi.Create
```

Nachdem Sie die `PathName`-Eigenschaft gesetzt haben, können Sie die Warteschlange mit einem Aufruf der `Create`-Methode erstellen lassen. Dieses Beispiel setzt auch die `Label`-Eigenschaft der neuen Warteschlange. Die `Label`-Eigenschaft ist optional, hilft Ihnen jedoch später, wenn Sie versuchen, die Warteschlange zu lokalisieren.

Der `Create`-Methode können zwei optionale Parameter übergeben werden. Der erste Parameter zeigt an, ob die neue Warteschlange transaktional ist. Der zweite Parameter, `IsWorldReadable`, lässt Sie bestimmen, ob die Warteschlange von anderen Benutzern als dem Besitzer der Warteschlange gelesen werden kann. Der Vorgabewert für diesen Parameter ist `False`, so dass nur der Besitzer Nachrichten von der Warteschlange empfangen kann. Wenn Sie dem Parameter den Wert `True` übergeben, kann die Warteschlange von allen Benutzern ausgelesen werden. Unabhängig von dem Wert, den Sie `IsWorldReadable` zuweisen, können alle Benutzer Nachrichten an diese Warteschlange senden. Sie können darüber hinaus Sicherheitsberechtigungen für die Warteschlange setzen, indem Sie die wahlweise Zugriffssteuerungsliste (DACL – Discretionary Access Control List) der Warteschlange modifizieren. Dies geschieht mit Hilfe des administrativen Hilfsmittels *Computerverwaltung* und dort im Eigenschaftendialogfeld der Warteschlange auf der Registerkarte *Sicherheit*.

Beachten Sie, dass die den `PathName` einer lokalen Warteschlange ohne den Computernamen angeben können. Sie sollten dies tun, wenn Sie generischen Code schreiben möchten, der auf unterschiedlichen Computern ausgeführt werden soll. Ein einzelner Punkt (wie in `.\MyQueue`) signalisiert, dass der Warteschlangenpfad auf dem lokalen Computer definiert ist. Sie können diese abgekürzte Form verwenden, wenn Sie eine lokale Warteschlange erstellen und öffnen. Dazu ein Beispiel:

```
Dim qi As MSMQQueueInfo
Set qi = New MSMQQueueInfo
qi.PathName = ".\MyQueue"
qi.Label = "Market Order Submission Queue"
qi.Create
```

Zusätzlich zum Erstellen von Warteschlangen können Sie mit dem `MSMQQueueInfo`-Objekt nach bereits bestehenden Warteschlangen suchen oder diese öffnen. Sie führen eine Abfrage von Active Directory mit einem `MSMQQuery`-Objekt durch, um zu ermitteln, ob eine Warteschlange mit einer bestimmten Bezeichnung vorhanden ist. Sie rufen dazu die `LookupQueue`-Methode auf, die ein `MSMQQueueInfos`-Objekt zurückgibt. Dieses Objekt ist eine Auflistung von `MSMQQueueInfo`-Objekten, die den von Ihnen angegebenen Kriterien entsprechen. Das folgende Beispiel führt die Suche nach der Bezeichnung einer Warteschlange durch:

```
Dim qry As MSMQQuery
Set qry = New MSMQQuery
Dim qis As MSMQQueueInfos
Set qis = qry.LookupQueue(Label:="Market Order Submission Queue")
Dim qi As MSMQQueueInfo
Set qi = qis.Next
If Not (qi Is Nothing) Then
    ' Warteschlange öffnen und Bestellnachricht senden.
Else
    ' Warteschlange kann nicht lokalisiert werden.
End If
```

In diesem Beispiel führt ein Client eine Abfrage für einen MSMQ-Katalog in Active Directory aus, um eine Warteschlange mit einer bekannten Bezeichnung zu lokalisieren. Diese dynamische Suchlösung bietet eine gewisse Flexibilität, da die Client-Applikation nicht den Namen des Computers kennen muss, auf dem die Warteschlange vorhanden ist. Sie können somit Warteschlangen über das Netzwerk bewegen, ohne Applikationen umschreiben zu müssen.

Lassen Sie uns nun eine Warteschlange öffnen und eine Nachricht senden. Das nächste Objekt, das Sie kennen lernen werden, ist ein `MSMQQueue`-Objekt. Die Beziehung zwischen `MSMQQueueInfo`-Objekten und `MSMQQueue`-Objekten kann ein wenig verwirrend sein. Sie denken wahrscheinlich, dass ein `MSMQQueue`-Objekt eine physikalische Warteschlange repräsentiert. Stattdessen repräsentiert es jedoch ein Warteschlangen-Handle. Sie können beispielsweise drei unterschiedliche `MSMQQueue`-Objekte für dieselbe physikalische Warteschlange öffnen:

```
Dim qi As MSMQQueueInfo
Set qi = new MSMQQueueInfo
qi.PathName = ".\MyQueue"
Dim qSend As MSMQQueue
Set qSend = qi.Open(MQ_SEND_ACCESS, MQ_DENY_NONE)
Dim qPeek As MSMQQueue
Set qPeek = qi.Open(MQ_PEEK_ACCESS, MQ_DENY_NONE)
Dim qReceive As MSMQQueue
Set qReceive = qi.Open(MQ_RECEIVE_ACCESS, MQ_DENY_NONE)
```

Ein `MSMQQueueInfo`-Objekt stellt somit eine physikalische Warteschlange dar, während ein `MSMQQueue`-Objekt ein geöffnetes Handle auf die Warteschlange repräsentiert. Wenn Sie `Open` aufrufen, müssen Sie den Zugriffstyp spezifizieren, der dem ersten Parameter übergeben wird. Sie können ein Peeking ausführen und Nachrichten aus einer Warteschlange entnehmen, wenn Sie diese mit `MQ_RECEIVE_ACCESS` öffnen. Wenn Sie jedoch Nachrichten senden und aus derselben Warteschlange Nachrichten empfangen oder ein Peeking für diese ausführen möchten, müssen Sie zwei `MSMQQueue`-Objekte öffnen. Denken Sie daran, die `Close`-Methode des `MSMQQueue`-Objekts aufzurufen, wenn Sie es nicht mehr benötigen.

Sie können den zweiten Parameter von `Open` verwenden, um den Freigabemodus der Warteschlange zu bestimmen. Der Vorgabewert für diesen Parameter ist `MQ_DENY_NONE`, der angibt, dass mehrere Applikationen gleichzeitig die Warteschlange öffnen können, um Nachrichten zu empfangen. Sie sollten diese Einstellung verwenden, wenn Sie eine Warteschlange mit `MQ_PEEK_ACCESS` oder `MQ_SEND_ACCESS` öffnen. Wenn Sie die Warteschlange jedoch mit `MQ_RECEIVE_ACCESS` öffnen, können Sie den Freigabemodus auf `MQ_DENY_RECEIVE_SHARE` setzen, um zu verhindern, dass alle Applikationen gleichzeitig Nachrichten empfangen. Wenn eine Applikation eine Warteschlange mit `MQ_RECEIVE_ACCESS` und `MQ_DENY_RECEIVE_SHARE` öffnet, kann keine andere Applikation die Warteschlange im Empfangsmodus öffnen. Eine Applikation, die diesen Modus benutzt, kann als einzige Anwendung Nachrichten von der Warteschlange empfangen.

Private Warteschlangen erstellen

Wenn Sie eine öffentliche Warteschlange erstellen, weist MSMQ dieser eine Instanz-ID (eine GUID) zu und veröffentlicht die Warteschlange in Active Directory. Andere Applikationen können die Warteschlange daraufhin öffnen, indem Sie den Computernamen und den Warteschlangennamen der `PathName`-Eigenschaft zuweisen. Dies erlaubt einer Applikation, die Warteschlange über das Netzwerk hinweg zu lokalisieren und eine Abfrage mit dem `MSMQQuery`-Objekt durchzuführen. Das Veröffentlichen einer öffentlichen Warteschlange kann jedoch ein wenig Zeit und Festplattenkapazität erfordern und ist nicht immer notwendig.

Stellen Sie sich eine Applikation vor, die aus unzähligen unabhängigen Clients besteht, die alle eine lokale Antwortwarteschlange benötigen. In dieser Situation sollten private Warteschlangen verwendet werden. Private Warteschlangen müssen lokal erstellt werden. Sie werden nicht im Active Directory, sondern ausschließlich im LQS des Computers veröffentlicht, auf dem sie sich befinden.

Wie Sie später in diesem Kapitel sehen werden, können Sie die Rückgabeadresse für eine private Antwortwarteschlange im Header der Anforderungsnachricht versenden, so dass die Empfängerapplikation eine Antwortnachricht zurückgeben kann. Auf diese Weise erzeugen Sie eine bidirektionale Kommunikation zwischen einer Client-Applikation und dem Server. Wenn Sie private Warteschlangen verwenden, müssen Sie außerdem nicht alle Antwortwarteschlangen im Active Directory veröffentlichen, so dass Sie Festplattenkapazität und Zeit sparen.

Sie erstellen eine private Warteschlange, indem Sie die Zeichenfolge `Private$` zur `PathName`-Eigenschaft der Warteschlange hinzufügen:

```
Dim qResponseInfo As MSMQQueueInfo
Set qResponseInfo = New MSMQQueueInfo
qResponseInfo.PathName = ".\Private$\MyResponseQueue"
qResponseInfo.Create
```

MSMQ-Applikationen können Nachrichten an private Warteschlangen senden, die auf anderen Computern vorhanden sind, sofern sie diese Warteschlangen finden. Dies ist nicht so einfach wie das Lokalisieren öffentlicher Warteschlangen, da Sie eine private Warteschlange nicht über die `PathName`-Eigenschaft öffnen können. Der Grund hierfür besteht darin, dass private Warteschlangen nicht im Active Directory veröffentlicht werden. Sie werden später in diesem Kapitel eine Technik kennen lernen, um die Informationen über eine Antwortwarteschlange mit einer Anforderungsnachricht an andere Applikationen zu übermitteln.

Warteschlangen lokalisieren

MSMQ weist sowohl öffentlichen als auch privaten Warteschlangen Instanz-IDs zur Entwurfszeit zu. Jede Warteschlange behält ihre Instanz-ID bis zum Ende. Wenn Sie eine Warteschlange mit `PathName` oder einer anderen Technik öffnen, muss ein Warteschlangen-Manager die Instanz-ID zunächst zuordnen, bevor Sie auf die Warteschlange zugreifen können.

MSMQ generiert eine Instanz-ID für eine private Warteschlange in einem anderen Format als die Instanz-ID einer öffentlichen Warteschlange. Wie Sie wissen, generiert MSMQ eine eindeutige GUID, die als Instanz-ID einer öffentlichen Warteschlange dient. Eine private Instanz-ID wird jedoch aus der eindeutigen GUID des lokalen Warteschlangen-Managers und einer computerspezifischen Warteschlangen-ID erzeugt. Hier ein Beispiel für eine Instanz-ID einer privaten Warteschlange:

```
f38f2a17-218e-11d2-b555-c48e04000000\00000022
```

Wenn Sie die Instanz-ID einer Warteschlange kennen, können Sie die Warteschlange mit der `FormatName`-Eigenschaft anstelle der `PathName`-Eigenschaft öffnen. Nachfolgend finden Sie ein Beispiel für zwei unterschiedliche `FormatName`-Eigenschaften für eine öffentliche und eine private Warteschlange.

```
PUBLIC=067ce2cb-26fc-11d2-b56b-f4552d000000  
PRIVATE=f38f2a17-218e-11d2-b555-c48e04000000\00000022
```

Sobald Sie wissen, wie die `FrontName`-Eigenschaft generiert wird, können Sie mit dem folgenden Code eine private Warteschlange auf einem anderen Computer öffnen:

```
Dim MyFormatName As String  
MyFormatName = "PRIVATE=f38f2a17-218e-11d2-b555-c48e04000000\00000022"  
Dim qi As MSMQQueueInfo  
Set qi = new MSMQQueueInfo  
qi.FormatName = MyFormatName  
Dim qSend As MSMQQueue  
Set qSend = qi.Open(MQ_SEND_ACCESS, MQ_DENY_NONE)
```

In einigen Fällen können Sie eine Warteschlange entweder mit `FormatName` oder mit `PathName` öffnen. Wenn Sie jedoch eine private Warteschlange öffnen, können Sie `PathName` nicht verwenden. Eine private Warteschlange besitzt keine Instanz-ID, die in Active Directory veröffentlicht wird. Wie Sie jedoch an dem vorherigen Beispiel gesehen haben, können Sie eine Nachricht an eine private Warteschlange über das Netzwerk senden, indem Sie die `FormatName`-Eigenschaft vor einem Aufruf der `Open`-Methode setzen. Sie sollten `FormatName` anstelle von `PathName` benutzen, wenn die Informationen der Zielwarteschlange nicht in Active Directory veröffentlicht wurden oder wenn ein Active Directory-Server nicht zur Verfügung steht.

Ihre letzte Option für den Zugriff auf eine Warteschlange besteht darin, einen direkten Formatnamen zu benutzen, der ein Protokoll, eine Computeradresse und einen Zielwarteschlangennamen spezifiziert. Wenn Sie diese Technik verwenden, sind Sie auf den LQS des Zielcomputers angewiesen, der die Namensauflösung vornimmt. Hier einige Beispiele für einen direkten Formatnamen:

```
DIRECT=TCP:209.27.34.23\MyQueue  
DIRECT=TCP:209.27.34.23\private$\MyResponseQueue  
DIRECT=OS:MyComputer\MyQueue
```

Sie benutzen direkte Formatnamen in der Regel nur als letzte Möglichkeit, wenn Sie andere Techniken nicht einsetzen können. Sie müssen direkte Formatnamen zum Beispiel verwenden, um die Kommunikation zwischen zwei MSMQ-Computern, die im Arbeitsgruppenmodus eingerichtet wurden, zu ermöglichen. Sie müssen direkte Formatnamen außerdem für eine Kommunikation über MSMQ-Organisationseinheiten hinweg einsetzen. Direkte Formatnamen sind somit notwendig, um Nachrichten zwischen zwei Computern zu versenden, die nicht denselben Active Directory-Katalog gemeinsam nutzen.

Der Einsatz von direkten Formatnamen führt zu einer wesentlichen Einschränkung: MSMQ kann keine Nachricht temporär auf dem Routing-Server speichern, wenn diese von dem Versender an den Empfänger übermittelt wird. Die Nachricht muss also direkt zwischen dem Versender-Computer und dem Computer mit der Zielwarteschlange übermittelt werden. Dies bedeutet, dass zwei Maschinen eine direkte Verbindung einrichten müssen. Beide Rechner können somit nicht die Vorzüge des verteilten Speicher- und Weiterleitungsmechanismus nutzen, der in die Infrastruktur von MSMQ integriert ist.

Nachrichten senden

Lassen Sie uns unsere erste Nachricht senden. MSMQ lässt diese Aufgabe sehr einfach werden. Sie können ein neues `MSMQMessage`-Objekt erstellen und einige Eigenschaften setzen, um die Nachricht vorzubereiten. Sie rufen anschließend die `Send`-Methode dieses Objekts auf, und MSMQ leitet Ihre Nachricht an die Zielwarteschlange weiter. Dazu ein Beispiel, das eine Nachricht über das Netzwerk an eine Warteschlange sendet:

```
Dim qi As MSMQQueueInfo
Set qi = New MSMQQueueInfo
qi.PathName = "MyServerComputer\MyQueue"
Dim q As MSMQQueue
Set q = qi.Open(MQ_SEND_ACCESS, MQ_DENY_NONE)
' Neue Nachricht erstellen.
Dim msg As MSMQMessage
Set msg = New MSMQMessage
' Nachricht vorbereiten.
msg.Label = "Meine Bezeichnung"
msg.Body = "Meine parametrisierten Anforderungsinformationen"
msg.Priority = MQ_MAX_PRIORITY
' Nachricht senden, um Warteschlange zu öffnen.
msg.Send q
q.Close
```

Wie Sie sehen, erleichtert die MSMQ-Komponentenbibliothek das Öffnen einer Warteschlange und das Versenden einer Nachricht. Die Nachricht des vorherigen Beispiels wurde auf das Versenden vorbereitet, indem drei Eigenschaften gesetzt wurden. Die `Label`-Eigenschaft ist eine Zeichenfolge des Nachrichten-Headers, der eine Nachricht identifiziert oder bezeichnet. Die beiden anderen Eigenschaften bilden den Nachrichtenkörper und die Nachrichtenpriorität.

In MSMQ wird ein Nachrichtenkörper als ein Array aus Byte-Werten gespeichert. Der Nachrichtenkörper wird gewöhnlich dazu verwendet, parametrisierte Daten zwischen dem Versender und dem Empfänger zu übermitteln. Das Beispiel zeigt, dass Sie dem Nachrichtenkörper einfach eine Visual Basic-Zeichenfolge zuweisen können. Der Empfänger kann diese Zeichenfolge ebenso einfach auslesen. In vielen Fällen verwenden Sie jedoch einen Nachrichtenkörper, der weitaus komplexer ist. Sie könnten zum Beispiel mehrere Parameter vom Versender zum Empfänger übermitteln. Dieses Thema wird noch einmal im zwölften Kapitel in dem Abschnitt behandelt,

der beschreibt, wie Daten übergeben werden und welche Optionen Ihnen zur Verfügung stehen, um parametrisierte Informationen im MSMQ-Nachrichtenkörper abzugeben.

Die letzte Eigenschaft, die in unserem Beispiel verwendet wird, setzt die Nachrichtenpriorität. Einer Nachricht kann ein Prioritätswert zwischen 0 und 7 zugewiesen werden. Je höher der Wert, desto höher die Priorität. MSMQ speichert Nachrichten mit einer höheren Priorität an der obersten Position der Warteschlange. Eine Nachricht mit der Priorität 6 wird beispielsweise unter allen Nachrichten der Priorität 7 angeordnet. Sie wird auch unter allen Nachrichten der Priorität 6 angeordnet, die bereits in die Warteschlange geschrieben wurden. Die neue Nachricht wird über den Nachrichten mit der Priorität 5 platziert. Die MSMQ-Typbibliothek enthält die Konstanten `MQ_MAX_PRIORITY` (7) und `MQ_MIN_PRIORITY` (0). Der Vorgabewert für die Priorität einer neuen Nachricht ist `MQ_DEFAULT_PRIORITY` (3).

Bevor wir uns dem nächsten Abschnitt zuwenden, möchte ich Ihnen einige andere wichtige Nachrichteneigenschaften vorstellen. Wenn eine Nachricht an eine Warteschlange gesendet wird, weist ihr MSMQ eine `ID`-Eigenschaft zu. Die `ID`-Eigenschaft ist ein 20-Byte-Array, das die Nachricht eindeutig identifiziert. MSMQ generiert die `ID` aus zwei verschiedenen Werten. Die ersten 16 Bytes der `ID` sind die `GUID` des lokalen Warteschlangen-Managers. Dies bedeutet, dass der erste Teil der Nachrichten-`ID` für alle Nachrichten, die vom selben Computer aus versendet werden, gleich ist. Die letzten vier Bytes der `ID` bilden einen eindeutigen Integer, der von dem Warteschlangen-Manager erzeugt wird. In den meisten Fällen müssen Sie sich nicht um den Inhalt dieses Byte-Arrays kümmern. Sie sollten immer davon ausgehen, dass jede `ID` eindeutig ist. Wenn Sie jedoch zwei `IDs` miteinander vergleichen möchten, um zu ermitteln, ob diese dieselbe Nachricht repräsentieren, können Sie die VBA-Funktion `StrComp` mit dem `vbBinaryCompare`-Flag verwenden.

Jede Nachricht hat außerdem eine `CorrelationID`-Eigenschaft, die ebenfalls ein 20-Byte-Array verwendet. Lassen Sie uns ein Beispiel untersuchen, das zeigt, dass diese Eigenschaft sehr nützlich ist. Stellen Sie sich vor, dass eine Client-Applikation Anforderungsnachrichten an einen Server sendet. Der Server bearbeitet die Anforderungen und sendet eine Antwortnachricht für jede Anforderung zurück. Doch woher weiß die Client-Applikation, welche Anforderungsnachrichten miteinander verknüpft sind? Die `CorrelationID`-Eigenschaft löst dieses Problem.

Wenn der Server eine Anforderung bearbeitet, kann er der `ID` der eingehenden Anforderungsnachricht die `CorrelationID` der ausgehenden Antwortnachricht zuweisen. Wenn die Client-Applikation eine Antwortnachricht empfängt, kann sie die `CorrelationID` dieser Nachricht mit der `ID` jeder Anforderungsnachricht vergleichen. Der Versender kann Nachrichten auf diese Weise in eine Beziehung zueinander setzen. Die `CorrelationID` ist somit nützlich, wenn Sie Ihre eigenen Antwortnachrichten erstellen. MSMQ weist die korrekte `CorrelationID` außerdem automatisch zu, wenn es bestimmte vom System generierte Nachrichten, wie zum Beispiel Bestätigungsnachrichten, auf das Versenden vorbereitet.

Nachrichten empfangen und lesen

Um eine Nachricht zu empfangen, öffnen Sie zunächst ein `MSMQQueue`-Objekt für den Empfangszugriff und rufen anschließend die `Receive`-Methode auf, um die erste Nachricht aus einer Warteschlange auszulesen und anschließend zu entfernen:

```
Dim qi As MSMQQueueInfo
Set qi = New MSMQQueueInfo
qi.PathName = ".\MyQueue"
Dim q As MSMQQueue
Set q = qi.Open(MQ_RECEIVE_ACCESS, MQ_DENY_NONE)
```

```

Dim msg As MSMQMessage
' Versuchen, die erste Nachricht in der Warteschlange zu empfangen.
Set msg = q.Receive(ReceiveTimeout:=1000)
If Not (msg Is Nothing) Then
    ' Sie haben die erste Nachricht aus der Warteschlange entfernt.
    MsgBox msg.Body, vbInformation, msg.Label
Else
    ' Ein Timeout wurde erreicht,
    ' da keine Nachrichten in der Warteschlange
    ' enthalten waren.
End If
q.Close

```

Es gibt einen interessanten Unterschied zwischen dem Versenden und dem Empfangen einer Nachricht mit MSMQ. Sie rufen die `Send`-Methode des `MSMQMessage`-Objekts, aber die `Receive`-Methode des `MSMQQueue`-Objekts auf. (Dies führt zu keinen Problemen. Es ist lediglich eine kleine Inkonssequenz im MSMQ-Programmiermodell.) Wenn eine Nachricht in der Warteschlange vorhanden ist, nimmt die `Receive`-Methode die Nachricht aus der Warteschlange heraus und gibt ein neu erstelltes `MSMQMessage`-Objekt zurück. Wenn keine Nachricht in der Warteschlange vorhanden ist, ist das Verhalten von `Receive` von dem gesetzten Zeitüberschreitungswert abhängig.

Ein Aufruf von `Receive` wird standardmäßig ohne Zeitüberschreitungswert ausgeführt und blockiert die weitere Programmausführung, bis eine Nachricht in die Warteschlange gestellt wird. Wenn Sie nicht möchten, dass der Thread, der `Receive` aufruft, lange Zeit blockiert ist, können Sie einen Zeitüberschreitungswert spezifizieren. Sie verwenden dazu den `ReceiveTimeout`-Parameter, dem Sie die Anzahl der Millisekunden übergeben, die vergehen sollen, bis die `Receive`-Methode aufgrund einer leeren Warteschlange abgebrochen wird.

Wenn `Receive` für eine leere Warteschlange aufgerufen wird und das Timeout-Intervall abgelaufen ist, bevor eine Nachricht in die Warteschlange gelangt, gibt die Methode einen Null-Verweis und kein `MSMQMessage`-Objekt zurück. Der Code im letzten Beispiel setzt den Timeout auf eintausend Millisekunden. Er zeigt außerdem, wie Sie ermitteln, ob eine Nachricht die Warteschlange vor Ablauf der Zeit erreicht hat. Wenn Sie keine Wartezeit wünschen, können Sie `ReceiveTimeout` auf 0 setzen. Ein `ReceiveTimeout`-Wert von -1 gibt zu verstehen, dass Sie unendlich lange warten möchten. (Dies ist der Vorgabewert, der verwendet wird, wenn Sie keinen Wert angeben.)

Sie können `Receive` wiederholt in einer `Do Loop`-Schleife aufrufen, um jede Nachricht synchron aus der Warteschlange auszulesen. Das folgende Beispiel zeigt die dazu notwendige Vorgehensweise. Es empfängt alle Nachrichten von einer Warteschlange und füllt ein Listenfeld mit den Nachrichtenbezeichnungen.

```

Dim qi As MSMQQueueInfo
Set qi = New MSMQQueueInfo
qi.PathName = ".\MyQueue"
Dim q As MSMQQueue
Set q = qi.Open(MQ_RECEIVE_ACCESS, MQ_DENY_RECEIVE_SHARE)
Dim msg As MSMQMessage
Set msg = q.Receive(ReceiveTimeout:=0)
Do Until msg Is Nothing
    lstReceive.AddItem msg.Label
    Set msg = q.Receive(ReceiveTimeout:=0)
Loop
q.Close

```


Sie können den Freigabemodus auf `MQ_DENY_RECEIVE_SHARE` setzen, so dass Ihre Applikation nicht mit anderen Applikationen in Konflikt gerät, während sie Nachrichten aus der Warteschlange entnimmt. Verwenden Sie den Zeitüberschreitungswert 0, wenn Sie das Ende der Warteschlange erreichen und direkt im Anschluss daran andere Aufgaben durchführen möchten.

Bisweilen werden Sie die Nachrichten überprüfen wollen, bevor Sie diese aus der Warteschlange entnehmen. Dazu verwenden Sie die Peeking-Methoden des `MSMQQueue`-Objekts in Verbindung mit einem impliziten Cursor, mit dessen Hilfe Sie durch die Nachrichten einer Warteschlange navigieren. Nach dem Öffnen einer Warteschlange für den Empfangszugriff oder den Peeking-Zugriff können Sie `Peek`, `PeekCurrent` oder `PeekNext` aufrufen.

`Peek` ähnelt `Receive`, da es die erste Nachricht in der Warteschlange liest. `Peek` entfernt die Nachricht jedoch nicht. Wenn Sie `Peek` wiederholt aufrufen, erhalten Sie somit immer die gleiche Nachricht. `Peek` hat außerdem keine Auswirkungen auf den impliziten Cursor, der von dem `MSMQQueue`-Objekt verwendet wird. Es ist deshalb üblich, häufiger mit `PeekCurrent` und `PeekNext` als mit `Peek` zu arbeiten.

Sie können den impliziten Cursor mit einem Aufruf von `PeekCurrent` auf die erste Nachricht einer Warteschlange setzen. Sie sollten, wie für `Receive`, einen Zeitüberschreitungswert verwenden, wenn Sie den Code aufgrund einer leeren Warteschlange nicht blockieren möchten. Nach dem Aufruf von `PeekCurrent` können Sie mit `PeekNext` die jeweils nächste Nachricht selektieren:

```
Dim qi As MSMQQueueInfo
Set qi = New MSMQQueueInfo
qi.PathName = ".\MyQueue"
Dim q As MSMQQueue
Set q = qi.Open(MQ_PEEK_ACCESS, MQ_DENY_NONE)
Dim msg As MSMQMessage
Set msg = q.PeekCurrent(ReceiveTimeout:=0)
Do Until msg Is Nothing
    ' Nachrichtenbezeichnung zu einem Listenfeld hinzufügen.
    lstPeek.AddItem msg.Label
    Set msg = q.PeekNext(ReceiveTimeout:=0)
Loop
q.Close
```

Die `ReceiveCurrent`-Methode wird häufig zusammen mit `PeekCurrent` und `PeekNext` verwendet. Sie können beispielsweise alle Nachrichten einer Warteschlange auswerten, indem Sie jede einzelne Nachricht mit einer Peeking-Methode lesen. Auf diese Weise können Sie die Eigenschaften der aktuellen Nachricht mit den Kriterien der Nachrichten vergleichen, die Sie empfangen und bearbeiten möchten. Nach einem Aufruf von `PeekCurrent` oder `PeekNext` könnten Sie die `Label`-Eigenschaft der aktuellen Nachricht mit einer spezifischen Bezeichnung vergleichen, nach der Sie suchen. Wenn eine Nachricht über diese Beschriftung verfügt, können Sie sie mit `ReceiveCurrent` aus der Warteschlange entnehmen und anschließend bearbeiten.

Nachrichten mit MSMQ-Ereignissen empfangen

Die letzten Beispiele haben synchrone Techniken zum Überprüfen und Entfernen von Nachrichten einer Warteschlange verwendet. Diese Techniken vereinfachen das Auslesen und Entfernen von Nachrichten, die sich zu einem bestimmten Zeitpunkt in der Warteschlange befinden. Mit den Techniken können Sie auch Nachrichten bearbeiten, die zu einem späteren Zeitpunkt zur Zielwarteschlange gesendet werden. Diese Lösung beinhaltet jedoch ebenfalls den aufrufenden Thread. Wenn Sie eine Single-Threaded-Applikation besitzen, kann die Applikation in der Zwischenzeit nichts anderes tun.

Sie können MSMQ-Ereignisse als Alternative zu dieser synchronen Verarbeitung von Nachrichten verwenden. MSMQ-Ereignisse lassen Ihre Applikation auf asynchrone Benachrichtigungen reagieren, die von MSMQ ausgelöst werden, wenn Nachrichten in der Warteschlange eintreffen. Sie können deshalb auf eine neue Nachricht reagieren, ohne sich mit einem Thread beschäftigen zu müssen, der den Aufruf von `Receive` blockiert.

Lassen Sie uns die MSMQ-Ereignisse untersuchen. Der MSMQ-Ereignismechanismus basiert auf der `MSMQEvent`-Komponente. Um Ereignisse benutzen zu können, müssen Sie zunächst ein `MSMQEvent`-Objekt erstellen und eine Ereignissenke einrichten. Sie verknüpfen das `MSMQEvent`-Objekt anschließend mit einem `MSMQQueue`-Objekt, das für den Peeking- oder Empfangszugriff geöffnet wurde. Sie erstellen diese Verknüpfung, indem Sie die `EnableNotification`-Methode des `MSMQQueue`-Objekts aufrufen und dieser einen Verweis auf das `MSMQEvent`-Objekt übergeben. Nach dem Aufruf von `EnableNotification` benachrichtigt MSMQ Ihre Applikation, wenn eine Nachricht in der Warteschlange eingetroffen ist, indem es das `Arrived`-Ereignis auslöst.

Sie erstellen eine Ereignissenke mit dem Visual Basic-Schlüsselwort `WithEvents` und deklarieren die Referenzvariable des Quellobjekts im allgemeinen Abschnitt eines Formularmoduls oder eines Klassenmoduls. Der folgende Code zeigt, wie eine Ereignissenke für ein neues `MSMQEvent`-Objekt in einem Formularmodul eines Standard-EXE-Projekts eingerichtet wird:

```
Private RequestQueue As MSMQQueue
Private WithEvents RequestEvent As MSMQEvent

Private Sub Form_Load()
    Dim qi As MSMQQueueInfo
    Set qi = New MSMQQueueInfo
    qi.PathName = ".\MyQueue"
    Set RequestQueue = qi.Open(MQ_RECEIVE_ACCESS, MQ_DENY_NONE)
    Set RequestEvent = New MSMQEvent
    RequestQueue.EnableNotification RequestEvent
End Sub
```

Nachdem Sie die Ereignissenke des `MSMQEvent`-Objekts eingerichtet und `EnableNotification` aufgerufen haben, werden Sie mit einem `Arrived`-Ereignis benachrichtigt, sobald MSMQ eine Nachricht in der Warteschlange findet. Nachfolgend wird eine Implementierung der `Arrived`-Ereignisbehandlungsroutine aufgeführt, die die Nachrichten bearbeitet, sobald diese in der Warteschlange eintreffen:

```
Sub RequestEvent_Arrived(ByVal Queue As Object, ByVal Cursor As Long)
    Dim RequestQueue As MSMQQueue
    Set RequestQueue = Queue ' Wechseln zu MSMQQueue, um die späte Bindung zu vermeiden.
    Dim msg As MSMQMessage
    Set msg = RequestQueue.Receive(ReceiveTimeout:=0)
    If Not (msg Is Nothing) Then
        ' Nachricht bearbeiten, wenn sie eintrifft.
    End If
    RequestQueue.EnableNotification RequestEvent
End Sub
```

Beachten Sie bitte, dass dieses Beispiel immer dann `EnableNotification` aufruft, wenn ein `Arrived`-Ereignis ausgelöst wird. Dies ist notwendig, da ein Aufruf von `EnableNotification` lediglich eine Benachrichtigung für die nächste Nachricht einrichtet. Wenn Sie ständig Benachrichtigungen erhalten möchten, müssen Sie `EnableNotification` wiederholt in der `Arrived`-Ereignisprozedur aufrufen.

Beachten Sie auch, dass der Code in dem Beispiel jede Nachricht empfängt, die in der Warteschlange gespeichert wurde, als das `MSMQEvent`-Objekt eingerichtet wurde. MSMQ löst somit sowohl Ereignisse für bereits bestehende als auch für zukünftige Nachrichten aus.

Listener-Applikationen mit Visual Basic erstellen

Ich arbeite gerne mit Visual Basic, muss aber zwei Einschränkungen dieses Entwicklungswerkzeuges nennen, die sich während der Erstellung von serverseitigen Listener-Applikationen zeigen. In einem kritischen System sollte die Listener-Applikation als Windows-Dienst eingesetzt werden. Dies führt zu einer besseren Verfügbarkeit. Wenn Ihre Listener-Applikation außerdem sehr viele eingehende Nachrichten bearbeiten muss, sollten Sie den Systemdurchsatz erhöhen, indem Sie eine größere Anzahl von Arbeits-Threads verwenden. Diese Arbeits-Threads empfangen die Nachrichten aus der Warteschlange und bearbeiten Anforderungen. Das Überwachen einer Warteschlange mit einem Thread-Pool, der fünf bis zwanzig Threads enthält, führt in der Regel zum bestmöglichen Durchsatz.

Einige Hardcore-Programmierer haben Möglichkeiten gefunden, eine Visual Basic-Applikation in einen Windows-Dienst umzuwandeln und zusätzliche COM-Threads zu erstellen. Dies sind jedoch keine gebräuchlichen Techniken. Sie werden nicht von Visual Basic unterstützt und resultieren deshalb in Code, der sehr schwierig zu pflegen ist.

Viele Unternehmen verwenden alternative Lösungen wie zum Beispiel C++, um eine benutzerdefinierte Listener-Applikation mit einer maximalen Verfügbarkeit und dem bestmöglichen Durchsatz zu erstellen. Eine andere Möglichkeit besteht darin, den Multi-Threaded-Listener-Dienst zu benutzen, der in die Architektur der Warteschlangenkomponenten integriert ist. Wenn Sie den Warteschlangenkomponenten-Dienst einsetzen, können Sie ein Multi-Threaded-Verhalten erzielen, während Ihr gesamter Applikationscode in Visual Basic vorliegt. Die dazu notwendige Vorgehensweise wird später in diesem Kapitel besprochen.

Sie sollten ebenfalls daran denken, dass es bisweilen möglich ist, eine serverseitige Listener-Applikation zu verwenden, die aus einem Standard-EXE-Projekt generiert wurde und auf nur einem Thread ausgeführt wird. Sie müssen nicht auf die C++-Entwicklung oder Warteschlangenkomponenten ausweichen. Dies gilt besonders dann, wenn Sie eine Listener-Applikation, die nur wenige Nachrichten pro Minute empfängt, oder eine Stapelverarbeitungsapplikation erstellen, die nur einmal täglich (zum Beispiel um Mitternacht) ausgeführt wird, um die Nachrichten zu bearbeiten, die sich während des Tages in der Anforderungswarteschlange gesammelt haben. In diesen Fällen kann eine einfache Visual Basic-Applikation eine weitaus kostengünstigere Lösung als eine Listener-Applikation für die mittlere Schicht bieten. Wie Sie später sehen werden, kann die reine MSMQ-Programmierung ebenfalls mehr Optionen und eine größere Flexibilität als Warteschlangenkomponenten zur Verfügung stellen.

Arbeiten mit der Antwortwarteschlange

Wenn eine Client-Applikation eine Nachricht an eine Anforderungswarteschlange sendet, muss häufig eine Antwortnachricht von der serverseitigen Listener-Applikation an den Versender zurückgeschickt werden. Eine Antwortnachricht kann zum Beispiel eine Benachrichtigung enthalten, die den Versender informiert, dass eine Transaktion erfolgreich ausgeführt wurde oder fehlgeschlug. Eine Antwortnachricht kann auch angeforderte Daten zurück zum Versender schicken.

Wenn Sie eine Nachricht in der Versender-Applikation vorbereiten, können Sie Informationen zum Header der Nachricht hinzufügen, die der Listener-Applikation mitteilen, wie eine Antwortwarteschlange auf dem Versender-Computer lokalisiert wird. Jedes `MSMQMessage`-Objekt verfügt über eine `ResponseQueueInfo`-Eigenschaft. Sie richten diese Eigenschaft mit Hilfe eines `MSMQQueueInfo`-Ob-

jekts ein. Die Empfängerapplikation benutzt `MSMQQueueInfo`, um die Warteschlange des Versenders zu öffnen und eine Bestätigungsnachricht zu versenden. Der Code in Listing 10.1 zeigt, wie die Listener-Applikation eine Anforderungsnachricht empfängt, die Antwortwarteschlange des Versenders öffnet und eine Antwortnachricht versendet.

```
Dim q1 As MSMQQueueInfo, RequestQueue As MSMQQueue
Set q1 = New MSMQQueueInfo
q1.PathName = "MyComputer\MyQueue"
Set RequestQueue = q1.Open(MQ_RECEIVE_ACCESS, MQ_DENY_RECEIVE_SHARE)
Dim msgRequest As MSMQMessage
Set msgRequest = RequestQueue.Receive(ReceiveTimeout:=0)
If Not (msgRequest Is Nothing) Then
    ' (1) Nachricht bearbeiten (auf eine Implementierung wurde aus Gründen der Übersichtlichkeit verzichtet).
    ' (2) Benachrichtigung zurück an den Client senden (Implementierung folgt).
    Dim q2 As MSMQQueueInfo, ResponseQueue As MSMQQueue
    Set q2 = msg.ResponseQueueInfo
    Set ResponseQueue = q2.Open(MQ_SEND_ACCESS, MQ_DENY_NONE)
    Dim msgResponse As MSMQMessage
    Set msgResponse = New MSMQMessage
    msgResponse.Label = "Receipt for: " & msgRequest.Label
    msgResponse.Body = "Your sales transaction has been completed"
    msgResponse.CorrelationId = msgRequest.Id
    msgResponse.Send ResponseQueue
    ResponseQueue.Close
End If
RequestQueue.Close
```

Listing 10.1: *Empfangen einer Nachricht mit einer Listener-Applikation*

Wenn Sie eine Listener-Applikation erstellen, die eine Antwortnachricht an die Client-Applikation zurückgibt, sollten Sie die ID der eingehenden Anforderungsnachricht der `CorrelationId` der ausgehenden Antwortnachricht zuordnen. Diese Technik wird in Listing 10.1 demonstriert. Wenn die Client-Applikation eine Antwortnachricht empfängt, kann sie deren `CorrelationId` mit der ID jeder Anforderungsnachricht vergleichen. Dies erlaubt dem Versender, Nachrichten in Beziehung zueinander zu setzen.

Sie sollten ebenfalls darüber nachdenken, private Warteschlangen anstelle von öffentlichen Warteschlangen zu verwenden, wenn Sie Ihre Antwortwarteschlangen einrichten. Dies gilt besonders für verteilte Applikationen mit hunderten oder tausenden von Clients. Im Gegensatz zu öffentlichen Warteschlangen werden private Warteschlangen nicht im Active Directory veröffentlicht. Der Einsatz von öffentlichen Warteschlangen verschwendet Verarbeitungszyklen, die Übertragungskapazität im Netzwerk und Festplattenspeicher und führt zu einem administrativen Mehraufwand.

Das transaktionale Messaging

Während der Design-Phase sollten Sie entscheiden, wie der MSMQ-Warteschlangen-Manager Ihre Nachrichten über das Netzwerk versenden soll. Sie können sich für eine Technik, die schnell und wenig zuverlässig ist, oder für eine Methode entscheiden, die langsamer ist, aber mehr Garantien bietet. Performance und Zuverlässigkeit vertragen sich nie.

Jede MSMQ-Nachricht besitzt eine `Delivery`-Eigenschaft mit zwei möglichen Einstellungen. Die Standardeinstellung ist `MQMSG_DELIVERY_EXPRESS`, die die Nachricht schnell, aber unzuverlässig versendet. Expressnachrichten werden lediglich im Speicher verwahrt, während sie über verschiedene Routing-Server hinweg zu ihrer Zielwarteschlange bewegt werden. Wenn ein Computer abstürzt, während er eine Expressnachricht speichert, kann diese Nachricht verloren gehen.

Um zu gewährleisten, dass eine Nachricht nicht verloren geht, während sie zur Zielwarteschlange geleitet wird, können Sie die `Delivery`-Eigenschaft auf `MQMSG_DELIVERY_RECOVERABLE` setzen. Die Nachricht wird dann auf der Festplatte zwischengespeichert, während sie von einem Computer zu einem anderen übertragen wird. Die Ein- und Ausgabeoperationen der Festplatte, die für wiederherstellbare Nachrichten erforderlich sind, führen zu einer signifikanten Leistungseinbuße. Die Nachrichten gehen jedoch im Falle eines Systemfehlers nicht verloren. Wenn Sie nichttransaktionale Nachrichten versenden, müssen Sie die `Delivery`-Eigenschaft explizit setzen, um eine wiederherstellbare Versendung zu erzielen. Wenn Sie transaktionale Nachrichten übermitteln, wird die `Delivery`-Eigenschaft automatisch auf `MQMSG_DELIVERY_RECOVERABLE` gesetzt.

Doch auch wenn Sie die `Delivery`-Eigenschaft für die Wiederherstellung setzen, können Sie mit Problemen konfrontiert werden. Eine einzelne Nachricht könnte mehr als einmal zur Zielwarteschlange gesendet werden. In diesem Fall würde die Listener-Applikation dieselbe Anforderung wiederholt bearbeiten. Dies kann zu einem unerwünschten Verhalten führen. Sie können Nachrichten zuverlässiger übermitteln, indem Sie transaktionale Warteschlangen benutzen.

Wenn Sie eine Nachricht in einer Transaktion versenden, erhalten Sie einige Garantien, die nichttransaktionale Nachrichten nicht gewähren können. Der Warteschlangen-Manager ergreift zum Beispiel besondere Vorsichtsmaßnahmen, um zu gewährleisten, dass Nachrichten nicht verloren gehen oder wiederholt bearbeitet werden. Transaktionale Nachrichten treffen darüber hinaus in der Reihenfolge ein, in der sie versendet wurden. Der Versender kann außerdem immer feststellen, ob eine Nachricht auf dem Weg zur Zielwarteschlange fehlgeschlagen ist.

Lassen Sie uns ein Beispiel für eine Übermittlung in der korrekten Reihenfolge betrachten. Stellen Sie sich vor, Sie senden eine Nachricht A und anschließend eine Nachricht B zur selben Warteschlange. Nachricht A erreicht die Warteschlange vor Nachricht B, sofern beide Nachrichten Bestandteil derselben Transaktion sind. Dies bedeutet, dass Nachricht A näher am oberen Bereich der Warteschlange angeordnet wird. Beachten Sie jedoch, dass andere Nachrichten von anderen Transaktionen die Position mit Ihrer Nachricht tauschen können. MSMQ gewährleistet die Anordnung von Nachrichten nur für eine einzelne Transaktion. Beachten Sie auch, dass transaktionalen Nachrichten aufgrund dieser garantierten Reihenfolge keine Prioritäten zugewiesen werden können. In MSMQ weist jede transaktionale Nachricht den `Priority`-Wert 0 auf.

In MSMQ müssen transaktionale Nachrichten an transaktionale Warteschlangen gesendet werden. Sie können das `Transaktionsattribut` nicht mehr verändern, nachdem eine Warteschlange erstellt wurde. Es ist deshalb wichtig, dass Sie bereits während der Erstellung angeben, ob eine Warteschlange transaktional sein soll. Sie können den ersten Parameter der `Create`-Methode einer `MSMQQueueInfo`-Komponente benutzen, um diese Angabe vorzunehmen.

Ihnen stehen zwei Möglichkeiten zur Auswahl, um mit MSMQ transaktionale Nachrichten zu versenden und zu empfangen. Sie können den internen MSMQ-Transaktionsmechanismus oder externe Transaktionen benutzen, die vom Distributed Transaction Coordinator (DTC) koordiniert werden. Der DTC verwendet dazu das Zwei-Phasen-Commit-Protokoll. Jede Technik bietet bestimmte Vorteile.

MSMQ stellt einen eigenen internen Transaktionsmechanismus zur Verfügung, der die beste Performance für das transaktionale Messaging bietet. Wenn Sie interne Transaktionen benutzen, kann MSMQ diese nicht mit anderen Ressourcen-Managern, wie zum Beispiel einer SQL Server-

Datenbank koordinieren. Interne Transaktionen nutzen nicht den DTC. Stattdessen verwendet MSMQ ein effizienteres Protokoll, das speziell für transaktionale Nachrichten optimiert wurde. Interne Transaktionen sind deshalb sehr viel schneller als extern koordinierte Transaktionen.

Da MSMQ das OLE- Transaktions-Protokoll unterstützt, ist es ein zulässiger Ressourcen-Manager, der an DTC-Transaktionen teilnehmen kann. Sie können es mit anderen Ressourcen-Managern verwenden, wenn Sie deklarative COM+-Transaktionen ausführen lassen. Ihre Verbindung zu einer transaktionalen Warteschlange kann wie eine Verbindung zu einer SQL Server-Datenbank mit dem DTC verknüpft werden. Externe Transaktionen werden auch als koordinierte Transaktionen bezeichnet, da sie vom DTC verwaltet werden.

Koordinierte Transaktionen sind langsamer als interne Transaktionen. Sie können die Nachrichtenübermittlung und Operationen anderer Ressourcen-Manager jedoch vereinen. Sie können zum Beispiel eine Transaktion schreiben, die eine Anforderungsnachricht empfängt, eine SQL Server-Datenbank modifiziert und eine Antwortnachricht versendet. Da diese drei Operationen Bestandteil einer einzigen Transaktion sind, erzwingt der DTC die Einhaltung der ACID-Regeln für alle Schreibaktivitäten. (Im achten Kapitel finden Sie eine detaillierte Beschreibung der ACID-Regeln.)

Interne MSMQ-Transaktionen

MSMQ bietet Ihnen eine schnelle Möglichkeit an, um eine einzelne transaktionale Nachricht zu versenden. Übergeben Sie in einem Aufruf von `Send` einfach die Konstante `MQ_SINGLE_MESSAGE`. Dazu ein Beispiel:

```
msg.Send q, MQ_SINGLE_MESSAGE
```

Sie werden häufig einzelne Nachrichten innerhalb einer Transaktion versenden wollen, wenn Sie die Einmal-Übermittlung wünschen. Wenn Sie eine Nachricht nicht innerhalb einer Transaktion versenden, kann diese verloren gehen oder dupliziert werden.

Um mehrere Nachrichten innerhalb einer internen Transaktion zu versenden, erstellen Sie ein neues `MSMQTransactionDispenser`-Objekt. Sie können anschließend dessen `BeginTransaction`-Methode aufrufen, um eine neue Transaktion zu starten. Dieser Aufruf gibt ein `MSMQTransaction`-Objekt zurück. Sie können einen Verweis auf dieses Objekt übergeben, wenn Sie eine Schreiboperation für eine Nachricht, wie zum Beispiel `Send`, `Receive` oder `ReceiveCurrent` ausführen. Der nachfolgende Programmcode sendet zwei Nachrichten zu einer transaktionalen Warteschlange, während eine `Receive`-Operation für eine andere transaktionale Warteschlange ausgeführt wird.

```
Dim td As MSMQTransactionDispenser
Set td = New MSMQTransactionDispenser
Dim tx As MSMQTransaction
Set tx = td.BeginTransaction()
Set msg1 = RequestQueue.Receive(Transaction:=tx)
msg2.Send ResponseQueue, tx
msg3.Send ResponseQueue, tx
' Transaktion festschreiben.
tx.Commit
```

Denken Sie daran, dass Nachrichten innerhalb derselben Transaktion, die an dieselbe Warteschlange gesendet werden, eine Übermittlung in der korrekten Reihenfolge gewährleisten. In diesem Beispiel können Sie davon ausgehen, dass `msg2` über `msg3` in der Warteschlange angeordnet

wird. Es gibt jedoch keine Garantie dafür, dass die Nachrichten anderer Transaktionen nicht zwischen den beiden Nachrichten angeordnet werden.

Die zeitliche Abstimmung für das Hinzufügen von transaktionalen Nachrichten zu einer transaktionalen Warteschlange und für das Entfernen dieser Nachrichten aus der Warteschlange ist kompliziert und erfordert sehr viel Aufmerksamkeit von Ihnen. Nachrichten werden erst dann zur Antwortwarteschlange gesendet, wenn die Transaktion festgeschrieben wurde. Es ist wichtig zu erkennen, dass eine transaktionale Nachricht während des Festschreibens einer Transaktion nicht an die Zielwarteschlange übermittelt wird. Die Reise der Nachricht, deren Ausgangspunkt der Versender-Computer ist, beginnt zu diesem Zeitpunkt erst.

Wenn Sie eine Nachricht in einer Transaktion empfangen, wird diese sofort aus der Warteschlange entfernt. Eine andere Applikation, die die Warteschlange untersucht, sieht die Nachricht auch dann nicht, wenn die Transaktion festgeschrieben wurde. Wenn die Transaktion jedoch abgebrochen wird, gelangt die Nachricht zurück zur Warteschlange. Dieses Verhalten tritt auch bei transaktionalen Warteschlangen auf. Dies geschieht unabhängig davon, ob Sie interne oder externe Transaktionen verwenden.

Sie müssen bestimmte Regeln befolgen, wenn Sie transaktionale Warteschlangen programmieren. Wenn Sie eine dieser Regeln verletzen, erhalten Sie einen Laufzeitfehler.

- Sie können eine Nachricht aus einer Transaktion nicht aus einer Transaktion heraus an eine nicht transaktionale Warteschlange senden.
- Sie können eine Nachricht nicht an eine transaktionale Warteschlange senden, wenn die Nachricht kein Bestandteil einer Transaktion ist.
- Sie können innerhalb derselben Transaktion keine Nachrichten zur selben Warteschlange senden oder von dort empfangen.
- Sie können transaktionale Nachrichten nur dann empfangen, wenn die Empfängerapplikation auf demselben Computer wie die transaktionale Warteschlange ausgeführt wird.

MSMQ-Transaktionen werden außerdem auf einer niedrigeren Isolationsebene als die Transaktionen in einem DBMS, wie zum Beispiel SQL Server, ausgeführt. Im achten Kapitel wurde bereits erwähnt, dass SQL Server die Operationen innerhalb einer COM+-Transaktion immer auf der Isolationsebene *Serializable* ausführt. MSMQ unterstützt diese Ebene nicht. Der Grund hierfür ist die Sperrungsverwaltung während einer Transaktion.

MSMQ verwendet die Isolationsebene *Repeatable Read* für eine Empfängerapplikation, wenn andere Applikationen Nachrichten an dieselbe transaktionale Warteschlange senden. Der Empfänger könnte zum Beispiel eine Transaktion starten und feststellen, dass die Warteschlange leer ist. Im Laufe dieser Transaktion sieht die Empfängerapplikation Nachrichten, die von anderen Transaktionen eingefügt wurden. Würde MSMQ die Transaktionen auf der Isolationsebene *Serializable* ausführen, wäre die Warteschlange gesperrt und jede Übermittlungsoperation blockiert, bis die empfangende Transaktion beendet ist. Ein Sperrschema wie dieses würde jedoch zu einer nicht akzeptablen Parallelitätseinschränkung in einem Warteschlangensystem führen.

Wenn jedoch zwei Transaktionen von zwei verschiedenen Empfängerapplikationen ausgeführt werden, kann dies auf der Isolationsebene *Read Uncommitted* geschehen. Lassen Sie uns dazu ein Beispiel betrachten. Stellen Sie sich eine einzelne Nachricht in einer transaktionalen Warteschlange vor. Wenn Transaktion A diese Nachricht aus der Warteschlange empfängt, findet Transaktion B eine leere Warteschlange vor. Wenn Transaktion A später abgebrochen wird, wird die Nachricht zurück in die Warteschlange geschrieben. Die Warteschlange war somit niemals leer. Transaktion B hatte jedoch diesen Eindruck, da sie die Warteschlange in einem nicht festgeschriebenen Zustand (*uncommitted*) überprüft hat.

Sie sollten die fehlende Möglichkeit, serialisierbare Transaktionen in MSMQ auszuführen, nicht als einen Mangel betrachten. Andere Messaging-Produkte, wie zum Beispiel MQSeries, weisen dasselbe Verhalten auf. Dies ist lediglich ein natürliches Problem transaktionaler Warteschlangen. Die Isolation wird geopfert, um eine bessere Parallelität zu erzielen. Sie können deshalb nicht so viele Vorhersagen treffen wie mit einem DBMS, dessen Operationen auf der Isolations-ebene *Serializable* ausgeführt werden.

Externe MSMQ-Transaktionen

Die MSMQ-Typbibliothek enthält eine Komponente zum Erstellen DTC-basierter Transaktionen, die als *MSMQCoordinatedTransactionDispenser* bezeichnet wird. Die Schnittstelle dieses so genannten Dispensers ist mit der *MSMQTransactionDispenser*-Komponente identisch, die bereits in diesem Kapitel vorgestellt wurde. Beide Komponenten verfügen über eine einzelne Methode, die *BeginTransaction* heißt und ein *MSMQTransaction*-Objekt zurückgibt. In den meisten Fällen ist es egal, welche dieser beiden Komponenten Sie verwenden.

Der Unterschied zwischen diesen beiden Komponenten besteht darin, dass der koordinierte Dispenser mit dem DTC zusammenarbeitet, während der interne Dispenser nicht mit dem DTC zusammenarbeitet. Wenn Sie eine externe Transaktion speichern, führt der DTC das Zwei-Phasen-Protokoll für MSMQ aus. MSMQ agiert hierbei als Ressourcen-Manager. Der koordinierte Dispenser lässt Sie außerdem einen anderen Ressourcen-Manager als MSMQ in einer Transaktion verknüpfen. Derzeit wird leider keine einfache Möglichkeit angeboten, um einen anderen Ressourcen-Manager als den SQL Server unter Verwendung des *MSMQTransaction*-Objekts zu verknüpfen. Sie müssen eine Datenbankverbindung explizit mit einem Aufruf von ODBC oder OLE-DB auf der Systemebene verknüpfen. Dies sollte jedoch in Betriebscode, der mit Visual Basic erstellt wurde, vermieden werden. Dies bedeutet, dass die meisten Visual Basic-Programmierer keine Vorteile aus der direkten Verwendung des koordinierten Dispensers ziehen.

Externe Transaktionen können diese Vorteile jedoch nutzen, indem Sie Nachrichten aus einer COM+-Transaktion heraus versenden und empfangen. Die COM+-Laufzeitumgebung kann in einer deklarativen Transaktion eine Verknüpfung zu MSMQ und zu anderen Ressourcen-Managern, wie zum Beispiel zu SQL-Server und Oracle herstellen. Nachfolgend finden Sie ein Beispiel zum Versenden einer Nachricht aus einer konfigurierten Komponente heraus, die eine Transaktion erfordert.

```
' Wir setzen voraus, dass diese Komponente eine Transaktion erfordert.
Dim oc As ObjectContext
Set oc = GetObjectContext()
' (1) In eine SQL Server-Datenbank schreiben.
' (2) Nachricht senden.
Dim qi As MSMQQueueInfo
Set qi = New MSMQQueueInfo
qi.PathName = "MyComputer\MyQueue"
Dim q As MSMQQueue
Set q = qi.Open(MQ_SEND_ACCESS, MQ_DENY_NONE)
Dim msg As MSMQMessage
Set msg = New MSMQMessage
msg.Body = "The database has been updated"
msg.Send q, MQ_MTS_TRANSACTION
q.Close
' (3) Transaktion festschreiben.
oc.SetComplete ' oder SetAbort aufrufen.
```


Sie können in einem Aufruf von `Send`, `Receive` oder `ReceiveCurrent` die Konstante `MQ_MTS_TRANSACTION` übergeben, um anzuzeigen, dass Sie Ihre Schreiboperation als Teil der aktuellen COM+-Transaktion ausführen lassen möchten. Wenn Sie keinen Transaktionsparameter übergeben, wird dieser Wert vorgegeben. Beachten Sie, dass eine Übergabe von `MQ_MTS_TRANSACTION` zu einer transaktionalen Operation führt, wenn Sie den Aufruf innerhalb einer COM+-Transaktion vornehmen.

Nachrichten innerhalb einer COM+-Transaktion werden, wie auch interne Transaktionen, nicht versendet, bis die Transaktion festgeschrieben wurde. Wenn Sie Nachrichten innerhalb einer Transaktion empfangen, werden diese aus der Warteschlange entfernt. Wenn Sie `SetAbort` aufrufen, um eine Transaktion rückgängig zu machen, werden Ihre Nachrichten zurück in die Warteschlange geschrieben. Denken Sie daran, dass die Nachrichten nicht unbedingt wieder an derselben Position angeordnet werden. Die vorherige Beschreibung der MSMQ-Isolationsebenen gilt ebenso für externe Transaktionen wie für interne Transaktionen.

Transaktionen mit einer Einmal-Semantik ausführen

Nachdem Sie nun gesehen haben, wie transaktionale Nachrichten übergeben werden, werden wir die Probleme untersuchen, die mit der Ausführung von COM+-Transaktionen ohne MSMQ verbunden sind. COM und RPC sind für Kommunikationsfehler anfällig, die zu widersprüchlichen Ergebnissen führen können. In OLTP-Applikationen ist dies häufig nicht akzeptabel. MSMQ bietet jedoch eine Lösung dieses Problems an.

Lassen Sie uns betrachten, was zu Fehlern führen kann, wenn ein Client einen COM-Methodenaufruf durchführt, um eine COM+-Transaktion auszuführen. Es gibt drei mögliche Fehlerquellen: Die Anforderungsnachricht des Methodenaufrufs kann erstens zwischen dem Client und dem Server fehlschlagen. Die COM+-Applikation kann zweitens abstürzen, während der Methodenaufruf in einer Transaktion ausgeführt wird. In beiden Szenarien werden die beabsichtigten Änderungen der Transaktion nicht festgeschrieben. Der dritte Fehler tritt auf, wenn die Antwortnachricht des Methodenaufrufs an den Client fehlschlägt oder verloren geht, nachdem die Transaktion erfolgreich durchgeführt und festgeschrieben wurde.

Hier das Problem: Was geschieht, wenn ein Client eine Transaktion abschickt, indem er einen COM-Methodenaufruf auf einem Remote-COM+-Objekt ausführt, aber keine Antwort erhält? Die Client-Applikation weiß in diesem Fall nicht, ob die Transaktion festgeschrieben wurde. Wenn der Client dieselbe Anforderung erneut abschickt, wird die Transaktion vielleicht ein zweites Mal festgeschrieben. Dies führt zu einem Problem, das nicht von COM und RPC gelöst werden kann.

Transaktionale Nachrichtenwarteschlangen bieten eine Möglichkeit, eine Transaktionsanforderung mit der Einmal-Übermittlungssemantik zu versenden. Sie können eine Transaktion mit dieser Semantik ausführen lassen, indem Sie sie in drei unterschiedliche Phasen unterteilen. Die Nachrichten werden in diesen Phasen von transaktionalen Warteschlangen versendet und empfangen. Abbildung 10.5 zeigt, wie diese Warteschlangen eingerichtet werden.

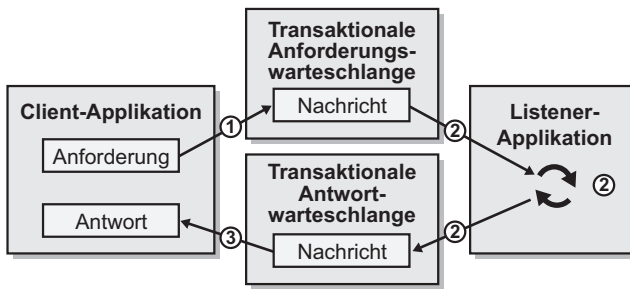


Abbildung 10.5: Das Ausführen einer Transaktionsanforderung mit der Einmal-Semantik erfordert zwei transaktionale Warteschlangen und drei verschiedene Transaktionen.

Der Client schickt eine Nachricht zuerst an eine Anforderungswarteschlange. Die Listener-Applikation führt daraufhin drei Schritte in nur einer komplexen Transaktion aus: Sie empfängt die Client-Nachricht aus der Anforderungswarteschlange, führt die vom Client angeforderte Transaktion aus und schickt eine Bestätigungsnachricht an die Antwortwarteschlange des Versenders zurück. Der Server muss diese drei Schritte erfolgreich ausführen. Die komplexe Transaktion wird andernfalls rückgängig gemacht. In diesem Fall wird die Client-Nachricht an die Anforderungswarteschlange zurückgegeben. Dies bedeutet, dass die Listener-Applikation die Nachricht erneut aus der Anforderungswarteschlange empfangen kann.

Sobald die angeforderte Transaktion erfolgreich abgeschlossen wurde, wird eine entsprechende Bestätigungsnachricht in der Antwortwarteschlange angeordnet. Die Client-Applikation empfängt eine transaktionale Nachricht aus der Antwortwarteschlange, die Aufschluss darüber gibt, ob die Transaktion erfolgreich abgeschlossen wurde.

Transaktionale Warteschlangen erlauben dem Versender, den Status einer Anforderung zu ermitteln: Eine Anforderung kann darauf warten, bearbeitet zu werden. Sie kann gegenwärtig bearbeitet werden oder sie wurde bereits bearbeitet. Wenn der Versender entdeckt, dass die ursprüngliche Nachricht weiterhin in der Anforderungswarteschlange enthalten ist, wurde die Anforderung noch nicht bearbeitet. Wenn der Versender eine entsprechende Nachricht in der Antwortwarteschlange findet, weiß er, dass die Anforderung bearbeitet wurde. Wenn keine Nachricht in beiden Warteschlangen vorhanden ist, wird die Anforderung gegenwärtig bearbeitet. RPC weist Einbußen gegenüber dem transaktionale Messaging auf, da es dem Client nicht gestattet, den exakten Status einer Anforderung zu ermitteln.

Inkorrekte Nachrichten und Fehlerbehandlung

Wenn Sie Applikationen unter Berücksichtigung des transaktionalen Messagings entwerfen, sollten Sie auf ein häufig auftretendes Design-Problem achten. Dieses Problem tritt auf, wenn eine Listener-Applikation die Transaktion abbricht, in der sie die Anforderungsnachricht des Clients empfängt. Eine Listener-Applikation könnte zum Beispiel dazu gezwungen sein, eine Transaktion aufgrund einer Geschäftsregelverletzung oder eines Datenbank-Servers abubrechen, der nicht mehr online ist. Wenn die Anforderungsnachricht jedoch einfach in die Anforderungswarteschlange zurückgeschrieben wird, führt dies zu einer rekursiven Schleife. Die Listener-Applikation wiederholt dann beliebig oft den Empfang der Nachricht, den Abbruch der Transaktion und das Zurückschreiben der Nachricht in dieselbe Anforderungswarteschlange. Dieses Szenario ist als *inkorrekte Nachrichtenproblem* bekannt.

Ein übliches Verfahren, auf inkorrekte Nachrichten zu reagieren, besteht darin, zwei separate Transaktionen auszuführen, während die Transaktionsanforderung des Clients bearbeitet wird. Die Listener-Applikation erstellt eine äußere Transaktion, um die Anforderungsnachricht zu empfangen. Sie führt dann eine COM+-Transaktion aus, um die Datenbank zu aktualisieren, und

sendet eine Antwortnachricht an den Client zurück. Die COM+-Transaktion wird als separate interne Transaktion ausgeführt. Dazu muss eine Listener-Applikation geschrieben werden, die Objekte aus Komponenten erstellt, deren Transaktionsunterstützungs-Eigenschaft auf *Erfordert neu gesetzt* wird.

Das Ausführen von zwei separaten Transaktionen kann das inkorrekte Nachrichtenproblem lösen. Wenn eine Geschäftsregelverletzung zu einem Zurücksetzen der inneren Transaktion führt, kann die Listener-Applikation feststellen, dass die Client-Transaktion fehlgeschlagen ist. Die äußere Transaktion kann jedoch noch festgeschrieben werden. Dies gibt der Listener-Applikation die Möglichkeit, die ursprüngliche Anforderungsnachricht zu löschen und eine Antwortnachricht mit einer Fehlerbenachrichtigung an den Client zurückzusenden.

Es ist nicht sehr schwierig, eine Listener-Applikation zu entwerfen, die inkorrekte Nachrichten verhindert, die sich wiederum aus Geschäftsregelverletzungen ergeben. Sie müssen jedoch ebenfalls auf Anforderungsnachrichten reagieren, die aufgrund anderer Probleme, wie zum Beispiel Systemfehler, nicht bearbeitet werden können. Wenn Sie eine Applikation entwerfen, die das transaktionale Messaging verwendet, sollten Sie den mit den inkorrekten Nachrichten und der Fehlerbehandlung verbundenen Aufwand nicht unterschätzen.

Was geschieht zum Beispiel, wenn ein Client eine zulässige Transaktionsanforderung abschickt, die die Listener-Applikation diese jedoch nicht bearbeiten kann? Wie schreiben Sie eine Listener-Applikation, die eine transaktionale Anforderungsnachricht bearbeiten kann, während der Datenbankserver offline ist? Sie können die Transaktion nicht einfach abbrechen und die Anforderungsnachricht in die Anforderungswarteschlange zurückschreiben. Wie Sie wissen, führt dies zu inkorrekten Nachrichten, so dass die Listener-Applikation eine unendliche Schleife durchläuft. Sie können die Transaktion auch nicht einfach festschreiben und die Anforderungsnachricht löschen, wenn Sie keinen Plan für den Eventualfall besitzen, der verhindert, dass sich die Client-Anforderung in Luft auflöst. Schließlich verschickt der Client eine zulässige Transaktionsanforderung, für deren korrekte Bearbeitung Sie sorgen müssen.

Eine Möglichkeit, auf dieses Problem zu reagieren, besteht darin, eine Fehlernachricht in eine spezielle Fehlerbehandlungswarteschlange zu schreiben. Wenn Sie zum Beispiel eine transaktionale Anforderungsnachricht empfangen und feststellen, dass ein Systemfehler aufgetreten ist, können Sie eine Kopie der Nachricht an die Fehlerbehandlungswarteschlange senden, die in der Regel von einem Administrator überwacht wird. Dies erlaubt Ihnen, die ursprüngliche Nachricht aus der Anforderungswarteschlange zu entfernen und die Client-Anforderungen weiterhin aufzuzeichnen. Sobald der Datenbankserver wieder online ist, kann der Administrator ein Hilfsprogramm ausführen, das Sie geschrieben haben, um die Anforderungsnachrichten zu bearbeiten, die sich in der Fehlerbehandlungswarteschlange angesammelt haben.

Dies ist nur eine einfache Lösung des Fehlerbehandlungsproblems, das mit dem transaktionalen Messaging verbunden ist. Viele Messaging-Applikationen weisen ein sehr viel weiter entwickeltes Design auf. Dieses Beispiel zeigt Ihnen jedoch die wesentlichen Aspekte und verdeutlicht die Zeit sowie den Aufwand, der für das Erstellen einer transaktionalen Listener-Applikation notwendig ist.

Warteschlangenkomponenten

Der Warteschlangenkomponenten-Dienst (QC – Queued Components), der ein Bestandteil von COM+ und Windows 2000 ist, bietet viele Vorteile der MSMQ-Programmierung. Dies geschieht jedoch in einer transparenten Weise. QC integriert COM und MSMQ, um die Einfachheit von COM-Methodenaufrufen zusammen mit den Vorteilen von Messaging und MSMQ zur Verfügung

zu stellen. Wie Sie in diesem Kapitel gesehen haben, erfordert die MSMQ-Programmierung mehr Aufmerksamkeit als COM, wenn Client-Anforderungen verschickt und beantwortet werden müssen.

Wenn Sie MSMQ benutzen, müssen Sie mehr über die Mechanik der Kommunikation nachdenken. Sie müssen Nachrichten vorbereiten und explizit über das Netzwerk an eine spezifische Warteschlange senden. Auf der Empfängerseite müssen Sie eine Listener-Applikation schreiben, um diese MSMQ-Nachrichten zu sammeln und die mit den Nachrichten verknüpften Anforderungen zu bearbeiten. Wenn Sie sich dann noch für die zusätzliche Komplexität entscheiden, die mit dem Einrichten einer bidirektionalen Kommunikation (einschließlich zum Beispiel der Antwortwarteschlangen und `CorrelationIDs`) verbunden ist, wird die MSMQ-Programmierung sehr viel langwieriger als das Abschicken von Anforderungen mit COM-Methodenaufrufen.

Eine der größten COM-Stärken ist der Verzicht auf die Details der Remote-Kommunikation und von RPC. Wie Sie im dritten Kapitel gesehen haben, arbeiten die COM-Laufzeit und der Universal-Marshaler zusammen, um eine Remote-Verbindung herzustellen. Dazu wird eine Proxy-/Stub-Schicht transparent zwischen einem Client und einem Objekt eingerichtet. Diese Proxy-/Stub-Schicht gewährleistet die Ortsunabhängigkeit, da sie Methodenaufrufe zwischen dem Client und dem Objekt übermitteln kann.

Die Remote-Kommunikation mit MSMQ gleicht in einer Hinsicht der Remote-Kommunikation mit COM. Jemand sendet eine Nachricht von einem Computer, und eine andere Person »lauscht«¹ dieser Nachricht auf einem anderen Computer. COM ist jedoch sehr viel einfacher zu bedienen als MSMQ. Die Standard-Proxy-/Stub-Schicht von COM bereitet RPC-Nachrichten im Hintergrund vor und übermittelt und empfängt diese. Mit MSMQ müssen Sie diese Arbeiten überwiegend selbst verrichten.

Eine Warteschlangenkomponente ist eine COM-Komponente. Die Methoden einer Warteschlangenkomponente können jedoch, mit MSMQ als zugrunde liegender Transportschicht, von einer Maschine zu einer anderen übertragen werden. Dies bedeutet, dass Warteschlangenkomponenten nicht von denselben RPC-Beschränkungen wie COM-Komponenten betroffen sind. Wie Sie sich bestimmt erinnern, weist RPC einige Einschränkungen auf, da es von verbindungsorientierter, synchroner Natur ist.

Der Einsatz einer Warteschlangenkomponente ist relativ einfach. Eine Applikation auf dem Client-Computer erstellt ein spezielles QC-Proxyobjekt und ruft eine oder mehrere Methoden auf. Die COM+-Laufzeit leitet die Methodenaufrufe weiter, indem sie diese in einer MSMQ-Nachricht aufzeichnet und an eine vom System zur Verfügung gestellte Warteschlange auf dem Servercomputer sendet. QC bietet die Konfiguration, die für einen integrierten Listener-Dienst auf dem Servercomputer erforderlich ist. Dieser Listener-Dienst gestattet der COM+-Laufzeit, die Warteschlange zu überwachen und Nachrichten zu sammeln, wenn diese eintreffen. Nachdem eine Nachricht empfangen wurde, erstellt COM+ eine Instanz der Warteschlangenkomponente und spielt die auf dem Client-Computer aufgezeichnete Nachricht ab.

Wie Sie sehen, bietet QC die Vorteile des Messagings, während auf die Details der MSMQ-Programmierung verzichtet wird. Im nächsten Abschnitt beschäftigen wir uns mit den Details der QC-Verwendung. Beachten Sie bitte, dass QC weniger Flexibilität als die direkte Programmierung von MSMQ zur Verfügung stellt. Wenn Sie die Vor- und Nachteile der Verwendung von QC und MSMQ verstehen, können Sie die beste Lösung für Ihr Projekt selektieren.

Die Architektur von Warteschlangenkomponenten

Lassen Sie uns mit einer erneuten Betrachtung der Architektur der COM-Proxy-/Stub-Standard-schicht beginnen, wie in Abbildung 10.6 dargestellt. Diese Architektur erlaubt Ihnen, eine RPC-Verbindung zwischen einem Client und einem Objekt einzurichten. Wenn ein Client eine Methode aufruft, bereitet der Proxy eine RPC-Anforderungsnachricht vor und sendet diese an den Stub. Nachdem die Methode des Objekts ausgeführt wurde, bereitet der Stub eine RPC-Antwortnachricht vor und schickt diese zum Proxy zurück. Die Details der Vorbereitung, Übermittlung und des Empfangs von RPC-Nachrichten werden von dem Proxy und dem Stub versteckt.

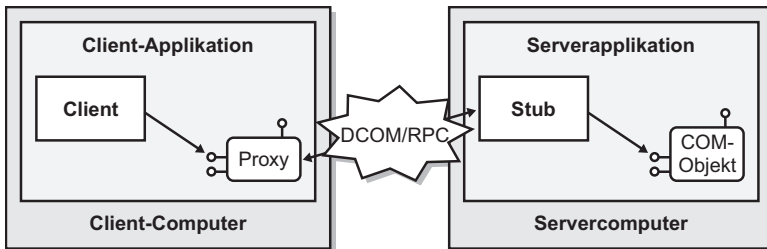


Abbildung 10.6: Die COM-Weiterleitungsarchitektur verlangt, dass eine Proxy-/Stub-Schicht zwischen dem Client und dem Objekt eingefügt wird. Der Proxy und der Stub sind auf RPC angewiesen, um eine synchrone, verbindungsorientierte Kommunikation durchführen zu können.

Abbildung 10.7 zeigt die Architektur von QC. Wenn die Client-Applikation ein Objekt aus einer Warteschlangenkomponente instanziiert, wird sie nicht an dieses Objekt gebunden, das die Methode implementiert. Sie wird stattdessen an ein spezielles Proxy-Objekt gebunden, das als »Recorder« bezeichnet wird. Warteschlangenkomponenten weisen somit eine Architektur auf, die der COM-Proxy-/Stub-Standardschicht gleicht. Ein Client kennt den Unterschied zwischen einem COM-Objekt, einem COM-Standardproxy und einem QC-Recorder nicht. Die Implementierungsdetails eines Recorders unterscheiden sich jedoch von sehr von denen eines COM-Standardproxies.

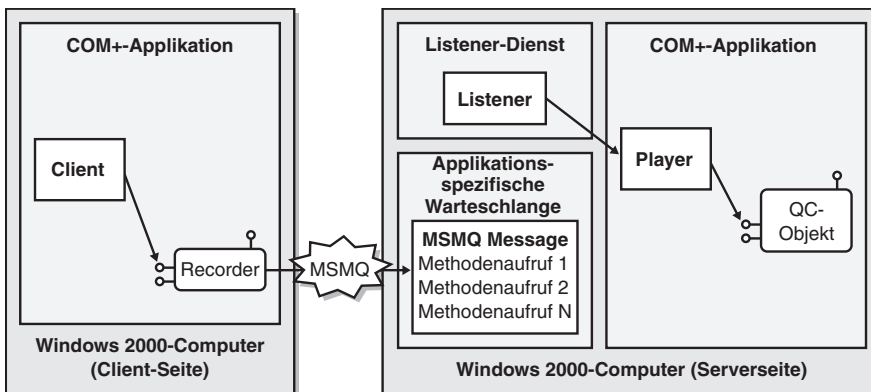


Abbildung 10.7: Eine Warteschlangenkomponente kann MSMQ als zugrunde liegendes Transportmittel benutzen. Auf der Client-Seite werden Methodenaufrufe automatisch aufgezeichnet und von der QC-Recorder-Komponente versendet. Auf der Serverseite ruft der QC-Listener-Dienst die QC-Player-Komponente auf, um alle Aufrufe abzuspielen, die in den MSMQ-Nachrichten aufgezeichnet sind.

Während ein COM-Standardproxy weiß, wie über RPC mit dem Stub kommuniziert wird, leitet das QC-Recorder-Objekt Methodenaufrufe über MSMQ weiter. Auf dem Client-Computer erstellt die COM+-Laufzeit Recorderobjekte aus einer konfigurierten Komponente namens *QC.Recorder*. Die Standardinstallation von COM+ fügt diese vom System zur Verfügung gestellte Komponente zu einer COM+-Bibliotheksapplikation mit der Bezeichnung *COM+ Utilities* hinzu.

Lassen Sie mich einen wichtigen Unterschied zwischen einem COM-Standardproxy und dem QC-Recorder erwähnen. Die COM-Standard-Proxy-/Stub-Schicht benötigt eine eingerichtete RPC-Verbindung. Jeder vom Client initiierte COM-Methodenaufwurf wird in Echtzeit an das Objekt weitergeleitet. Wenn ein Client zum Beispiel zwei Methoden ausführt, muss der erste Aufruf zum Client zurückgekehrt sein, bevor der zweite Aufruf ausgeführt werden kann. Warteschlangenkompontenten wurden hingegen so entworfen, dass die erforderlichen Aufgaben ausgeführt werden, sobald eine Verbindung zum Servercomputer hergestellt werden kann.

Wenn eine Client-Applikation eine Instanz des QC-Recorders erstellt und mit der Ausführung von Methoden beginnt, werden die Aufrufe nicht in Echtzeit weitergeleitet. Ein Recorderobjekt puffert stattdessen alle Aufrufe auf dem Client-Computer, indem diese in nur einer MSMQ-Nachricht aufgezeichnet werden. Die QC-Laufzeit übermittelt die Nachricht erst dann an den Servercomputer, wenn das Recorderobjekt freigegeben wurde. Beachten Sie bitte, dass QC die transaktionalen Messaging-Fähigkeiten von MSMQ nutzt. Jede Nachricht wird innerhalb des Gültigkeitsbereichs einer MSMQ-Transaktion versendet. Dies bedeutet, dass die Kommunikation mit QC die Einmal-Übermittlungsssemantik erfüllt.

Ich werde die Details der Konfiguration des Servercomputers gleich erläutern. Lassen Sie uns jedoch zunächst die größeren Zusammenhänge betrachten. Die Warteschlangenkompontente muss auf dem Servercomputer in eine COM+-Serverapplikation installiert werden, die für Warteschlangen konfiguriert ist. Wenn Sie eine COM+-Serverapplikation für Warteschlangen konfigurieren, erstellt COM+ automatisch eine applikationsspezifische öffentliche Eingabewarteschlange für diese Applikation. (COM+ generiert außerdem einige zusätzliche Fehlerbehandlungswarteschlangen.) Sobald eine Serverapplikation für Warteschlangen konfiguriert wurde, kann sie auch als Listener eingerichtet werden. COM+ richtet in diesem Fall einen Dienst ein, der die Warteschlange überwacht und auf eingehende Nachrichten reagiert.

Stellen Sie sich vor, dass eine Warteschlangenkompontente und ihr COM+-Serverapplikations-Host richtig konfiguriert wurden und die Applikation ausgeführt wird. Eine Client-Applikation erstellt nun ein Recorderobjekt und ruft drei Methoden auf. Das Recorderobjekt reagiert, indem es die zum Abspielen dieser drei Methoden erforderlichen Daten in den Körper einer MSMQ-Nachricht serialisiert. Wenn die Client-Applikation das Recorderobjekt freigibt, versucht die QC-Laufzeit, die Nachricht an die Eingabewarteschlange der Applikation, die sich auf dem Servercomputer befindet, zu senden. Wenn der Client-Computer vom Netzwerk getrennt ist, speichert der lokale Warteschlangen-Manager die Nachricht und leitet sie später weiter, sobald wieder eine Verbindung zum Netzwerk besteht. Wenn die Nachricht schließlich bei der Applikations-Eingabewarteschlange des Servercomputers eintrifft, erkennt dies der QC-Listener-Dienst, so dass er mit der Bearbeitung der Nachricht beginnen kann.

Der QC-Listener-Mechanismus verwendet eine nicht konfigurierte Komponente namens *QC.Listener*. Während der Listener die Warteschlange überwacht, delegiert er die Verantwortlichkeit für die Bearbeitung jeder MSMQ-Nachricht an ein Player-Objekt. Der Listener erstellt ein neues Player-Objekt für jede eingehende MSMQ-Nachricht. Beachten Sie, dass der Listener die Player-Objekte aus einer vom System zur Verfügung gestellten, konfigurierten Komponente namens *QC.ListenerHelper* generiert. Die Standardinstallation von COM+ fügt die *QC.ListenerHelper*-Komponente zu der bereits erwähnten COM+-Bibliotheksapplikation mit der Bezeichnung *COM+ Utilities* hinzu.

Ein Player-Objekt erzeugt eine Instanz der Warteschlangenkomponente und führt jede auf dem Client-Computer aufgezeichnete Methode aus. Die MSMQ-Nachricht trägt alle Informationen mit sich, die der Player benötigt, um seine Arbeit zu verrichten. Nachdem das Player-Objekt alle Methoden abgespielt und die Instanz der Warteschlangenkomponente freigegeben hat, wird er von der COM+-Laufzeit freigegeben.

Lassen Sie uns noch einmal zusammenfassen, wie mehrere Methodenaufrufe mit einer Warteschlangenkomponente weitergeleitet werden. Jede Methodenaufrufserie erfordert bestimmte Voraussetzungen. Ein Recorderobjekt muss auf dem Client-Computer erstellt und zerstört werden. Die COM+-Laufzeit auf dem Client-Computer muss eine MSMQ-Nachricht an den Servercomputer versenden. Der Listener-Dienst auf dem Servercomputer muss ein Player-Objekt erstellen und freigeben. Während seiner Lebensdauer muss das Player-Objekt Informationen aus der MSMQ-Nachricht lesen, eine Instanz der angeforderten Warteschlangenkomponente erstellen und jede Methode abspielen, bevor das Objekt freigegeben wird.

Lassen Sie mich einige Schlüsselaspekte dieser Architektur erwähnen. Playerobjekte sind thread-neutral. Sie werden im MTA-Thread-Pool einer COM+-Serverapplikation ausgeführt. Wenn Sie Ihren Code mit Visual Basic schreiben, werden die aus Ihren Warteschlangenkomponenten erzeugten Objekte in die Threads des STA-Thread-Pools geladen. Die schlechte Nachricht lautet, dass es einen Thread-Wechsel zwischen einem Player-Objekt und einem Visual Basic-Objekt gibt. Die gute Nachricht lautet, dass diese Architektur all Ihre Objekte über alle Threads im STA-Thread-Pool verteilt. Dies bedeutet, dass QC ein Multi-Threaded-Verhalten aufweist. Ihre Applikation kann mehrere MSMQ-Nachrichten gleichzeitig bearbeiten, was, verglichen mit einer Single-Threaded-Listener-Applikation, zu einem höheren Durchsatz führt.

Vergessen Sie auch nicht, dass sowohl die Recorder-Komponente als auch die Player-Komponente so konfiguriert sein müssen, dass eine Transaktion erforderlich ist. Dies führt zu wichtigen Einschränkungen für den Servercomputer. Der Listener startet eine neue COM+-Transaktion, wenn er ein neues Player-Objekt erstellt. Wenn Sie das Transaktionsunterstützungs-Attribut einer Warteschlangenkomponente auf *Unterstützt* oder *Erforderlich* setzen, wird jede Warteschlangenkomponenteninstanz in der Transaktion des Players erzeugt. Dies ist wichtig, da der Abbruch der Player-Transaktion den QC-Dienst davon abhält, eine aus der Applikations-Eingabewarteschlange empfangene Nachricht erfolgreich zu bearbeiten. Dieses Thema wird später ausführlich behandelt, wenn wir uns der Fehlerbehandlung zuwenden.

Einschränkungen der Warteschlangenkomponenten

Lassen Sie uns nun einige erwähnenswerte Einschränkungen des QC-Dienstes untersuchen. Die erste Einschränkung besteht darin, dass QC auf die MSMQ-Laufzeit angewiesen ist. MSMQ muss auf jedem Client-Computer und Servercomputer installiert sein, der QC benutzt.

Die zweite wichtige Einschränkung besteht darin, dass die QC-Architektur konfigurierte Komponenten auf dem Client- und Servercomputer verwendet. Die Listener-Konfiguration auf dem Server erfordert außerdem die COM+-Containerapplikation *DLLHOST.EXE*. Dies bedeutet, dass QC nur von solchen Applikationen benutzt werden kann, die auf einem Windows 2000-Computer ausgeführt werden. Diese unglückliche Einschränkung hält viele Unternehmen davon ab, QC für die Client-/Server-Kommunikation zu verwenden. Wenn Ihre Benutzer frühere Versionen von Windows ausführen, kann QC nur dem Ziel dienen, eine asynchrone, getrennte Kommunikation von einem Servercomputer zu einem anderen zu erreichen. Sie können QC dann zum Beispiel lediglich benutzen, um Anforderungen von einem Webserver an einen Applikationsserver zu senden.

Eine weitere Einschränkung betrifft mehrere verschiedene Vorgänge, die mit MSMQ, aber nicht mit QC möglich sind. Im Gegensatz zu MSMQ erlaubt Ihnen QC nicht, Active Directory nach einer Warteschlange abzufragen und dabei Kriterien wie zum Beispiel eine Warteschlangenbezeichnung zu verwenden. QC gestattet auch kein Peeking für Nachrichten. Sie dürfen auch keine CorrelationIDs benutzen, um Antwortnachrichten in Beziehung zueinander zu setzen. QC bietet für eine Applikation möglicherweise all die Funktionalität, die Sie benötigen. Für eine andere Applikation entscheiden Sie vielleicht, MSMQ und dessen Features zu nutzen, die nicht von QC unterstützt werden.

Warteschlangenkomponenten entwerfen

Sie erstellen eine Warteschlangenkomponente ebenso wie jede andere konfigurierte Komponente, indem Sie eine *MultiUse*-Klasse zu einem ActiveX-DLL-Projekt hinzufügen. Eine Warteschlangenkomponente muss mindestens eine Schnittstelle mit Visual Basic oder IDL definieren, die Warteschlangen unterstützt. Sobald Sie eine Warteschlangenschnittstelle definiert haben, können Sie diese in eine *MultiUse*-Klasse implementieren. Ich werde in diesem Kapitel jedoch nicht auf komplizierte Details eingehen. Ich werde Ihnen zeigen, wie eine Warteschlangenkomponente unter Verwendung einer *MultiUse*-Klasse mit öffentlichen Methoden generiert wird. Wir werden nicht mit einer separaten benutzerdefinierten Schnittstelle arbeiten, da ich Ihnen die einfachste Möglichkeit zum Erstellen einer Warteschlangenkomponente mit Visual Basic zeigen möchte.

Eine Warteschlangenschnittstelle weist eine wichtige Einschränkung auf: Sie kann lediglich benutzt werden, um in nur eine Richtung zu kommunizieren. Die von einer Warteschlangenkomponente implementierte Schnittstelle kann ausschließlich Methoden mit Eingabeparametern enthalten. Die Parameter können somit lediglich vom Client-Computer zum Servercomputer übertragen werden. Im Gegensatz zu COM-Methodenaufrufen, die RPC-basierte Anforderungs-/Antwortpaare benutzen, kann eine Warteschlangenkomponente keine Parameter von einem Objekt an den Client zurückgeben. Hier einige Beispiele für Methoden, die in einer Warteschlangenschnittstelle definiert werden können:

```
Public Sub Method1()  
    ' Implementierung  
End Sub
```

```
Public Sub Method2(ByVal i as Integer)  
    ' Implementierung  
End Sub
```

```
Property Let Property1(ByVal s As String)  
    ' Implementierung  
End Property
```

Wie Sie sehen, muss jeder Parameter mit dem *ByVal*-Attribut gekennzeichnet werden. Eine Schnittstelle ist keine Warteschlangenschnittstelle, wenn sie Methoden enthält, die einen Ausgabeparameter oder einen Rückgabewert benutzen. Stellen Sie sich zum Beispiel vor, Sie möchten eine Warteschlangenkomponente unter Verwendung einer *MultiUse*-Klasse mit öffentlichen Methoden erstellen. Die öffentlichen Methoden, die in der Standardschnittstelle definiert sind, müssen Warteschlangen unterstützen. Die folgenden Methodensignaturen würden dazu führen, dass die Klasse nicht als Warteschlangenkomponente verwendet werden kann:


```

Sub Method3(ByRef i As Integer)
    ' Implementierung
End Sub

' ByRef ist die Aufruf-Standardkonvention.
Sub Method4(i As Integer)
    ' Implementierung
End Sub

Function Method5() As Integer
    ' Implementierung
End Function

Property Get Property1() As Integer
    ' Implementierung
End Property

```

Der Schlüssel zum Entwerfen von Warteschlangenkomponenten ist Methodensignaturen zu vermeiden, die das Zurücksenden von Daten vom Objekt zum Client erfordern. Wenn Sie die Implementierung für eine *MultiUse*-Klasse schreiben, die diese Regel befolgt, und die Klasse anschließend in eine ActiveX-DLL kompilieren, können Sie die DLL in eine COM+-Serverapplikation installieren und als Warteschlangenkomponente konfigurieren.

Warteschlangenkomponenten konfigurieren

Wenn Sie den Servercomputer einrichten, besteht Ihre erste Aufgabe darin, eine COM+-Serverapplikation zu erstellen und richtig zu konfigurieren. Stellen Sie sich vor, Sie erstellen eine COM+-Serverapplikation namens *MyApplication*. Sie können diese Applikation manuell für Warteschlangen und als Listener konfigurieren, indem Sie das administrative Hilfsmittel *Komponentendienste* benutzen, wie in Abbildung 10.8 dargestellt. Sie können das Erstellen und Konfigurieren einer COM+-Serverapplikation außerdem automatisieren, indem Sie Skripte oder die COM+-Admin-Objekte verwenden.

Wenn Sie eine COM+-Serverapplikation für Warteschlangen konfigurieren, generiert COM+ automatisch eine öffentliche transaktionale Warteschlange mit demselben Namen. COM+ erstellt außerdem mehrere private Warteschlangen, um den QC hinsichtlich der Fehlerbehandlung zu unterstützen.

Zwei weitere Aktionen sind notwendig, um eine Warteschlangenkomponente einzurichten. Sie müssen erstens die Serverapplikation als Listener konfigurieren, so dass COM+ die Applikationswarteschlange für eingehende Nachrichten überwacht. Zweitens müssen Sie die Applikation starten. Der Listener-Dienst arbeitet nur dann, wenn die Applikation ausgeführt wird. Sie können die Serverapplikation manuell starten, indem Sie das administrative Hilfsmittel *Komponentendienste* benutzen. Um die Applikation programmgesteuert zu starten, verwenden Sie die COM+-Admin-Objekte. Hier ein Beispiel für Visual Basic-Code, der eine COM+-Serverapplikation startet:

```

Dim cat As COMAdminCatalog
Set cat = New COMAdmin.COMAdminCatalog
cat.StartApplication "MyApplication"

```



Abbildung 10.8: Sie können eine COM++-Serverapplikation mit Hilfe des administrativen Hilfsmittels Komponentendienste auf der Registerkarte Warteschlange im Eigenschaftendialogfeld der Applikation für Warteschlangen und als Listener konfigurieren.

Sie können eine COM++-Serverapplikation ebenfalls mit einer WSH-Batchdatei (WSH= Windows Script Host) starten, die in VBScript geschrieben ist. Sie konfigurieren den Servercomputer in diesem Fall mit dem Taskplannerdienst, so dass die Batchdatei nach dem Systemstart ausgeführt wird. Sie können eine Warteschlangenapplikation auch in der Nacht ausführen lassen, wenn die Client-Anforderungen im Batchmodus bearbeitet werden sollen.

Zusätzlich zur Applikation müssen Sie jede Warteschlangenkomponente konfigurieren. Sie installieren die Komponente dazu zunächst in der Warteschlangen-Serverapplikation. Dann richten Sie die Schnittstelle für Warteschlangen ein. Stellen Sie sich vor, Sie möchten die Standardschnittstelle einer Komponente mit der Bezeichnung *MyQueuedComponent* verwenden. Nachdem die Komponente installiert wurde, finden Sie die Standardschnittstelle (die den Namen *_MyQueuedComponent* trägt) im Schnittstellen-Ordner des administrativen Hilfsmittels *Komponentendienste*, wie in Abbildung 10.9 dargestellt.

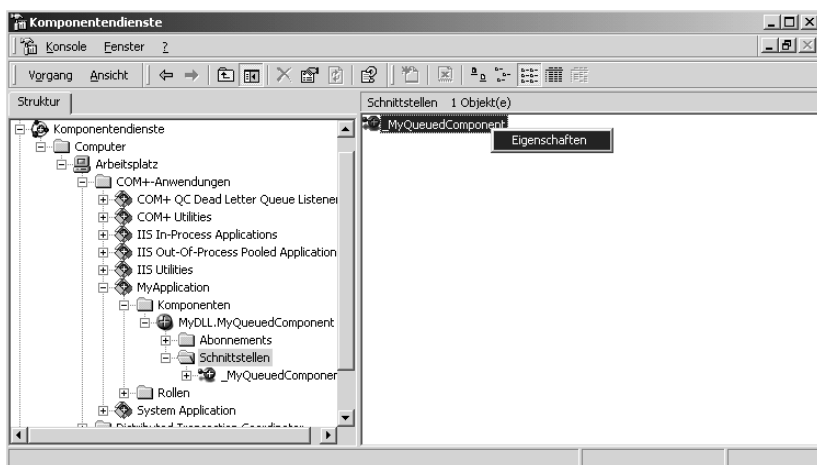


Abbildung 10.9: Jede konfigurierte Komponente besitzt einen Schnittstellen-Ordner.

Nachdem Sie die Schnittstelle lokalisiert haben, können Sie deren Eigenschaftendialogfeld aufrufen. Auf der Registerkarte *Warteschlange* konfigurieren Sie die Schnittstelle für Warteschlangen, wie in Abbildung 10.10 dargestellt. Wenn das Kontrollkästchen *Eingereiht* gesperrt ist, hat COM+ festgestellt, dass die Schnittstelle nicht für Warteschlangen konfiguriert werden kann. Dies geschieht, wenn eine Methode der Schnittstelle einen ByRef-Parameter oder einen Rückgabewert besitzt. Beachten Sie, dass das Einrichten einer Schnittstelle für Warteschlangen, wie auch die meisten anderen Aspekte der COM+-Administration, programmgesteuert durchgeführt werden kann.

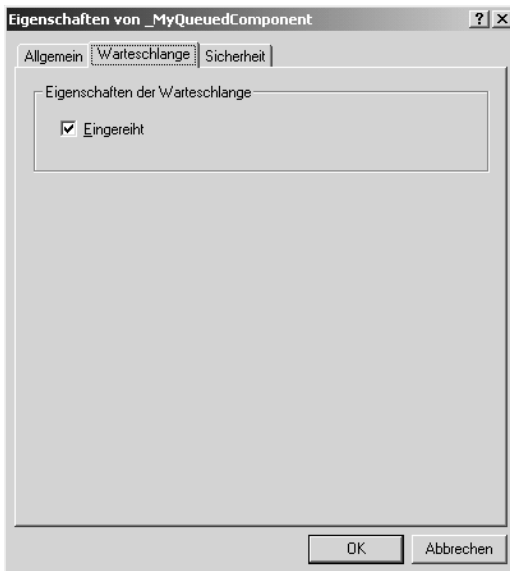


Abbildung 10.10: Die Schnittstelle einer Komponente muss für Warteschlangen konfiguriert werden, damit eine asynchrone Kommunikation erfolgen kann.

Lassen Sie mich schnell die Konfigurationsanforderungen für den Servercomputer zusammenfassen: Sie benötigen erstens eine COM+-Serverapplikation, die für Warteschlangen und als Listener (Option *Abfragen*) konfiguriert ist. Sie müssen zweitens die Warteschlangenkomponekte in der Applikation installieren und sicherstellen, dass eine oder mehrere Schnittstellen für Warteschlangen konfiguriert sind. Die Reihenfolge dieser beiden Schritte ist nicht wichtig. Schließlich starten Sie die Serverapplikation. Wenn Sie eine Serverapplikation als Listener konfigurieren, während die Applikation ausgeführt wird, müssen Sie sie herunterfahren und neu starten, damit die Änderung berücksichtigt wird.

Warteschlangenkomponekten vom Client aus programmieren

Der wohl wichtigste Aspekt der clientseitigen Programmierung ist das korrekte Erstellen des Recorderobjekts. Wenn Sie möchten, dass Ihre Warteschlangenkomponekten MSMQ als zugrunde liegendes Transportmittel verwenden, muss eine Client-Applikation einen QC-Recorder anstelle eines COM-Standardproxies erstellen.

Stellen Sie sich vor, Sie haben die Warteschlangenkomponekte und deren COM+-Host-Serverapplikation richtig konfiguriert. Sie haben die COM+-Export-Anweisung benutzt, um einen Applikationsproxy für diese Applikation zu erstellen, und Sie haben den Applikationsproxy auf dem

Client-Computer installiert. Sie können nun Ihre Warteschlangenkomponenten vom Client-Computer aus benutzen.

Beachten Sie, dass die für Warteschlangenkomponenten erforderliche clientseitige Konfiguration der Konfiguration anderer COM-Komponenten gleicht. Der Client-Computer muss eine registrierte Kopie der Typbibliothek und Konfigurationsinformationen über ProgIDs, CLSIDs und IIDs besitzen. Es ist ebenfalls hilfreich, über eine clientseitige AppID mit einem zulässigen Remote-ServerName zu verfügen. All diese Details werden Ihnen abgenommen, wenn Sie einen Applikationsproxy auf dem Client-Computer installieren.

Wenn Sie ein QC-Recorderobjekt als Proxy für eine Warteschlangenkomponente erstellen, können Sie das Objekt nicht mit einer Standardtechnik wie zum Beispiel dem Aufruf von `New` oder `CreateObject` instanzieren. Betrachten Sie hierzu den folgenden Code:

```
Dim obj1 As MyDll.MyQueuedComponent
Set obj1 = New MyDll.MyQueuedComponent
Dim obj2 As MyDll.MyQueuedComponent
Set obj2 = CreateObject("MyDll.MyQueuedComponent")
```

Beide Aktivierungstechniken führen zur COM-Standardaktivierung. Eine Instanz der Warteschlangenkomponente wird auf dem Servercomputer erstellt und mit dem Client verbunden. Dies geschieht über die COM-Standardproxy-/Stub-Schicht. Jeder Methodenaufruf an das Objekt wird nun mit RPC in Echtzeit ausgeführt. In diesem Szenario besitzen Sie eine Warteschlangenkomponente, die Sie jedoch wie eine COM-Standardkomponente verwenden.

Um das Recorderobjekt korrekt auf dem Client-Computer zu erzeugen, müssen Sie eine spezielle Aktivierungstechnik benutzen. Betrachten Sie dazu den folgenden Code:

```
Dim rec As MyDll.MyQueuedComponent
Set rec = GetObject("Queue:/New:MyDll.MyQueuedComponent")
```

Wie Sie sehen, können Sie einen Recorder mit einem Aufruf der Visual Basic-Funktion `GetObject` generieren. Damit Sie jedoch verstehen, wieso dies möglich ist, muss ich das Zeichenfolgenargument erklären, das an `GetObject` übergeben wird. Ich muss außerdem das Konzept eines Monikers erläutern.

Moniker sind bereits seit einiger Zeit ein Bestandteil von COM. Ein Moniker ist ein Objekt, das ein anderes Objekt sucht oder erstellt und an einen Client bindet. Ein Moniker kann mit einer Zeichenfolge wie der, die wir an die `GetObject`-Funktion übergeben haben, identifiziert werden. Ich werde nicht viele Worte über Moniker in diesem Buch verlieren, da Sie diese nicht mit Visual Basic implementieren können. Sie können jedoch bereits vorhandene Moniker von einer Client-Applikation aus, die in Visual Basic geschrieben wurde, verwenden, indem Sie `GetObject` aufrufen. Das COM+-Team hat Windows 2000 zwei neue Moniker hinzugefügt, um den Bedürfnissen von QC gerecht zu werden: den Queue-Moniker und den New-Moniker. Wenn Sie `GetObject` aufrufen und eine Zeichenfolge übergeben, die den Queue-Moniker identifiziert, wird ein spezielles Moniker-Objekt im Hintergrund erstellt. Dieses Objekt generiert ein QC-Recorderobjekt und integriert dieses in den Client. Der Queue-Moniker ist auf den New-Moniker angewiesen, um seine Arbeit verrichten zu können. Im vorherigen Beispiel wurde eine Queue-Moniker-Zeichenfolge in diesem Format erstellt:

```
Queue:/New:MyDll.MyQueuedComponent
```

Ich habe hier die ProgID der Warteschlangenkompone nte an den New-Moniker übergeben. Sie können auch eine CLSID mit oder ohne geschweifte Klammern übergeben. Dies bedeutet, dass es drei verschiedene Formate gibt, die auf dieselbe Weise arbeiten.

Beachten Sie, dass der Queue-Moniker einige Parameter akzeptiert, die die Warteschlange öffnen und benutzerdefinierte Nachrichten versenden. Sie finden eine Liste aller verfügbaren Optionen in der Warteschlangenkompone nten-Dokumentation des Plattform SDKs. Hier ein Beispiel für einen Parameter, der den Servernamen spezifiziert und die deklarative Remote-Servernamenseinstellung überschreibt:

```
Queue:ComputerName=MyServer/New:MyDll.MyQueuedComponent
```

Lassen Sie uns nun den Lebenszyklus eines Recorderobjekts besprechen. Wenn das Recorderobjekt aktiv ist, werden die Methodenaufrufe einfach aufgezeichnet und auf dem Client-Computer gepuffert. Die QC-Laufzeit versucht nicht, MSMQ-Nachrichten über das Netzwerk zu versenden, bis Sie das Recorderobjekt freigeben. Betrachten Sie hierzu den folgenden Code:

```
Dim rec As MyDll.MyQueuedComponent
Set rec = GetObject("Queue:/New:MyDll.MyQueuedComponent")
rec.Method1
rec.Method2 33
rec.Prop1 = "Some Value"
Set rec = Nothing
```

Die QC-Laufzeit versucht nicht erst dann, wenn die letzte Zeile erreicht ist und der Client den Verweis auf das Recorderobjekt freigibt, die Nachricht zu versenden. Wenn der Client-Computer keine Verbindung zum Servercomputer oder einem anderen Routing-Server in der Organisationseinheit herstellen kann, speichert MSMQ die Nachricht natürlich lokal im Cache, bis diese an das richtige Ziel weitergeleitet werden kann.

Die QC-Dokumentation hebt hervor, dass Sie Objekte aus Warteschlangenkompone nten mit dem New-Moniker und ohne den Queue-Moniker erstellen können. Dazu ein Beispiel:

```
Dim obj As MyDll.MyQueuedComponent
Set obj = GetObject("New:MyDll.MyQueuedComponent")
```

Das hier demonstrierte Instanzieren eines Objekts aus einer Warteschlangenkompone nte führt zu demselben Ergebnis wie bereits vorgestellte Techniken, die den New-Operator oder die CreateObject-Funktion verwenden. Der New-Moniker ohne den Queue-Moniker initiiert einen Aufruf von CoCreateInstance. Dies bedeutet, dass der Client kein Recorderobjekt erzeugt. Er generiert stattdessen direkt eine Remote-Instanz einer Warteschlangenkompone nte und wird über ein Standardproxy-/Stub-Paar mit der Kompone nte verbunden. Diese Technik verwendet eine synchrone RPC-Kommunikation. Sie erfordert außerdem eine Verbindung zwischen dem Client-Computer und dem Servercomputer. Wenn zwischen diesen beiden Computern keine direkte Verbindung hergestellt werden kann, schlägt die Aktivierungsanforderung fehl.

Es war ein Design-Ziel des COM+-Teams, eine Infrastruktur zu erzeugen, die einen einfachen Wechsel zwischen synchronen und asynchronen Aufrufen ermöglichen sollte. Wie Sie sehen, können Sie über eine verbindungsorientierte Kommunikation (COM) oder eine nicht verbundene, asynchrone Kommunikation (QC) auf eine Warteschlangenkompone nte zugreifen, sobald Sie die Kompone nte korrekt eingerichtet haben. Sie können zwischen diesen beiden unterschied-

lichen Verfahren wechseln, indem Sie einfach die Zeichenfolge ändern, die die Client-Applikation an `GetObject` übergibt.

Bidirektionale Kommunikation

Das Einrichten einer bidirektionalen Kommunikation ist mit QC möglich, kann aber sehr kompliziert sein. Die Unterstützung, die QC für die Rückgabe von Antwortnachrichten an einen Client zur Verfügung stellt, ist nicht sehr intuitiv. Dies gilt besonders, wenn Sie daran denken, wie einfach es ist, QC für das Versenden von Nachrichten in eine Richtung zu benutzen. Lassen Sie mich ein Beispiel nennen, um die Anforderungen zu demonstrieren. Die Client-Applikation muss nicht nur ein Recorderobjekt für die Warteschlangenkomponente auf dem Server, sondern ebenfalls ein zweites Recorderobjekt erstellen, das eine lokale Warteschlangenkomponente verwendet. Der Client muss dieses Recorderobjekt als einen Parameter in einer ausgehenden Methode übermitteln. Die Warteschlangenkomponente auf dem Server kann anschließend das Recorderobjekt benutzen, um mehrere Methodenaufrufe an den Client-Computer zurückzugeben.

Eine Applikation wird sehr komplex, wenn Sie QC benutzen möchten, um Antwortnachrichten an den Client zurückzusenden. Sie müssen eine Warteschlangenkomponente und eine für Warteschlangen konfigurierte COM+-Serverapplikation auf dem Client-Computer und auf dem Servercomputer erstellen und einrichten. Diese Lösung ist besonders für Client-Applikationen geeignet, die COM+-Serverapplikationen sind. Sie ist für Client-Applikationen mit einer Benutzeroberfläche jedoch nicht sehr elegant.

Was ist so schlimm daran, einen Rückruf von einer Client-Applikation mit einer Benutzeroberfläche einzurichten? Das Problem ist, dass die Client-Applikation in einem Prozess ausgeführt wird, während die Rückrufobjekte in einer separaten Instanz von *DLLHOSTEXE* generiert und ausgeführt werden. Dies ist nicht einfach, da Sie die Kommunikation zwischen den beiden Prozessen koordinieren müssen.

Warteschlangenkomponenten und die Fehlerbehandlung

Lassen Sie uns betrachten, wie QC arbeitet, wenn alles planmäßig verläuft. Das Recorderobjekt auf dem Client-Computer sendet eine MSMQ-Nachricht an den Servercomputer. Denken Sie daran, dass diese Nachricht an eine transaktionale Eingabewarteschlange gesendet wird. Der Listener-Dienst erstellt anschließend ein Player-Objekt, um die Nachricht zu empfangen. Die QC-Player-Komponente wird so konfiguriert, dass Sie eine Transaktion erfordert. Ein Player-Objekt wird deshalb in einer COM+-Transaktion ausgeführt und kann einen transaktionalen Empfang durchführen. Wenn das Player-Objekt eine Instanz der Zielwarteschlangenkomponente generieren und alle Methoden abspielen kann, gibt es seine Einwilligung zum Festschreiben der Transaktion. Wenn der Player die Transaktion festschreibt, geht QC davon aus, dass die Client-Anforderung bearbeitet wurde, so dass die MSMQ-Nachricht gelöscht wird.

Was aber geschieht, wenn Fehler auftreten? Die QC-Infrastruktur stellt eine Fehlerbehandlung zur Verfügung und verhindert inkorrekte Nachrichten. Wie bereits erwähnt wurde, erstellt COM+ zusätzlich zur primären Applikations-Eingabewarteschlange sechs weitere private Warteschlangen für die Fehlerbehandlung, wenn Sie eine COM+Applikation für Warteschlangen konfigurieren. COM+ generiert diese zusätzlichen Warteschlangen, um das Problem der inkorrekten Nachrichten auszuschließen.

Denken Sie darüber nach, was geschehen würde, wenn das Player-Objekt einfach die Transaktion abbrechen würde, sobald ein Fehler während des Abspielens einer Methode aufträte. Die MSMQ-Nachricht würde an die primäre Eingabewarteschlange zurückgegeben und die QC-In-

frakstruktur würde versuchen, dieselbe Nachricht immer wieder in einer endlosen Schleife zu bearbeiten. QC stellt die zusätzlichen Warteschlangen zur Verfügung, um dieses Problem zu lösen.

Einige Ereignisse können verhindern, dass ein Player-Objekt eine Client-Anforderung erfolgreich bearbeitet und die entsprechende MSMQ-Nachricht löscht. Ein Player-Objekt bricht die Transaktion erstens ab, wenn es mit einem Laufzeitfehler konfrontiert wird, während die Methoden aus der MSMQ-Nachricht abgespielt werden. Zweitens kann auch jedes Objekt, das in derselben Transaktion wie das Player-Objekt erstellt wurde, die Transaktion abbrechen. Wenn Ihre Warteschlangenkomponekte beispielsweise so konfiguriert ist, dass Transaktionen erforderlich sind oder unterstützt werden, werden die Objekte der Komponente in derselben Transaktion wie das Player-Objekt erzeugt. Wenn Ihr Objekt `SetAbort` aufruft, wird die Transaktion abgebrochen und die MSMQ-Nachricht nicht gelöscht. Schließlich wird die Transaktion abgebrochen, wenn das Player-Objekt seine Arbeit nicht innerhalb des computerweit gültigen Timeout-Intervalls für Transaktionen verrichten kann. Dieser Intervall dauert standardmäßig 60 Sekunden. Ein Player-Objekt muss jede Methode abspielen, Ihr Objekt freigeben und die Transaktion festschreiben, bevor der Timeout-Wert erreicht ist. Wenn dies nicht möglich ist, bricht die COM+-Laufzeit die Transaktion automatisch ab.

Wenn die Transaktion des Player-Objekts abgebrochen wurde, gibt QC die Nachricht nicht einfach an dieselbe Warteschlange zurück. In diesem Fall träte das Problem der inkorrekten Nachrichten auf. QC bewegt die Nachricht stattdessen zur nächsten Warteschlange in einer Kette mit Wiederholungswarteschlangen. COM+ erstellt zusätzlich zur primären Eingabewarteschlange der Applikation fünf Wiederholungswarteschlangen und eine finale Abschlusswarteschlange.

Wenn Sie eine Warteschlangenapplikation namens `MyApplication` erstellen, generiert COM+ eine öffentliche Eingabewarteschlange mit der Bezeichnung `myapplication` und fünf Wiederholungswarteschlangen mit den Namen `myapplication_0`, `myapplication_1`, `myapplication_2`, `myapplication_3` und `myapplication_4`. Außerdem wird eine Abschlusswarteschlange namens `myapplication_deadqueue` erzeugt, die Nachrichten speichert, die nicht bearbeitet werden können.

Wenn ein Player-Objekt eine Nachricht aus der primären Eingabewarteschlange nicht bearbeiten kann, ordnet QC die Nachricht in `myapplication_0` an. QC versucht dann nach einer Minute, die Nachricht erneut zu bearbeiten. Dieser Versuch wird dreimal wiederholt. Wenn diese Versuche fehlschlagen, platziert QC die Nachricht in `myapplication_1`, wartet zwei Minuten und führt anschließend drei Versuche aus, die Nachricht zu bearbeiten. Wenn die Nachricht erneut Probleme verursacht, ordnet sie QC nacheinander in allen fünf Wiederholungswarteschlangen und dann in der finalen Abschlusswarteschlange an.

Jede Ausnahmefehlerwarteschlange weist einen längeren Wartezyklus als die vorherige Warteschlange auf. Die Wartezeit erhöht sich von einer Minute auf zwei Minuten, von zwei Minuten auf vier Minuten, von vier Minuten auf acht Minuten und schließlich auf 16 Minuten. Nachdem sich die Nachricht 16 Minuten in `myapplication_4` befunden hat, unternimmt QC drei letzte Versuche und verschiebt die Nachricht dann in die finale Abschlusswarteschlange. Dort unterliegt es Ihrer Verantwortung, die Nachricht zu bearbeiten.

COM+ stellt außerdem ein Hilfsmittel namens `MessageMover` zur Verfügung, dass das Verschieben der Nachrichten zwischen den Warteschlangen automatisiert. `MessageMover` ist eine vom System zur Verfügung gestellte Komponente, die Bestandteil der COM+-Laufzeit (`COMSVCS.DLL`) ist. Sie greifen in Ihrer Visual Basic-Applikation auf den `MessageMover` zu, indem Sie einen Verweis auf die Typbibliothek *COM+ Services Type Library* einbinden.

Lassen Sie uns ein Beispiel betrachten, das zeigt, wie nützlich diese Hilfsmittel-Komponente sein kann. Stellen Sie sich vor, Sie kommen montags in Ihr Büro, um festzustellen, dass Ihr Applikations-Datenbankserver seit 48 Stunden offline ist. Wenn Clients am Wochenende Warteschlan-

genkomponenten verwendet haben, um Anforderungen abzuschicken, könnte die finale Abschlusswarteschlange voller MSMQ-Nachrichten sein, die nicht bearbeitet wurden. Mit MessageMover können Sie einfach eine administrative Applikation erstellen, um all diese Nachrichten von der finalen Abschlusswarteschlange zurück in die primäre Eingabewarteschlange zu verschieben. Sobald der Datenbankserver wieder online ist, können Sie diese Verschiebung vornehmen, so dass QC die Nachrichten bearbeiten kann.

Das Programmieren mit dem MessageMover ist einfach. Der folgende Code kann einer Visual Basic-Applikation hinzugefügt werden, um alle Nachrichten aus der finalen Abschlusswarteschlange zurück in die primäre Eingabewarteschlange zu verschieben.

```
Dim MessageMover As MessageMover
Dim MessageCount As Long
Set MessageMover = New COMSVCSLib.MessageMover
MessageMover.SourcePath = ".\private$\myapplication_deadqueue"
MessageMover.DestPath = ".\myapplication"
MessageCount = MessageMover.MoveMessages
MsgBox MessageCount & " messages have been moved"
```

Sie können auch eine Ausnahmeklasse erstellen, deren Methoden aufgerufen werden, um die Bearbeitung der Nachrichten zu automatisieren, die in der finalen Abschlusswarteschlange abgelegt werden. Sie erstellen eine Ausnahmeklasse, indem Sie eine Komponente generieren, die eine Schnittstelle mit der Bezeichnung `IPlaybackControl` aus der Typbibliothek *COM+ Services Type Library* implementiert. Diese Schnittstelle weist zwei Methoden auf: `FinalServerRetry` und `FinalClientRetry`. Sie können mit einer Ausnahmeklasse auf fehlgeschlagene Nachrichten auf der Client-Seite sowie auf der Serverseite reagieren. Eine Ausnahmeklasse muss ebenfalls die Warteschlangenschnittstelle implementieren, die der Client benutzt, um Methoden in der MSMQ-Nachricht aufzuzeichnen.

Sie können eine Ausnahmeklasse mit einer Warteschlangenkomponekte verknüpfen, indem Sie die Registerkarte *Erweitert* des Komponenten-Eigenschaftendialogfelds im administrativen Hilfsmittel *Komponentendienste* verwenden. Geben Sie in das Textfeld unter *Warteschlangen-Ausnahmeklasse* die ProgID oder CLSID der Ausnahmeklasse ein.

Wenn QC eine Nachricht in die finale Abschlusswarteschlange verschiebt, prüft der Dienst, ob die Warteschlangenkomponekte eine Ausnahmeklasse besitzt. QC erstellt in diesem Fall ein Objekt und ruft `QueryInterface` auf, um eine `IPlaybackControl`-Verbindung herzustellen. Im Anschluss daran ruft QC `FinalServerRetry` auf. Wie Sie dies implementieren, ist Ihre Entscheidung. Sie können zum Beispiel den `MessageMover` benutzen, um die Nachricht in eine andere Warteschlange zu verschieben. Sie können aber auch eine E-Mail an einen Administrator versenden oder ein Windows-Ereignis protokollieren.

Nachdem `FinalServerRetry` auf dem Ausnahmeklassenobjekt aufgerufen wurde, ruft QC ein weiteres Mal `QueryInterface` auf, um noch eine Verbindung zu der Schnittstelle herzustellen, die in der Nachricht aufgezeichneten Methoden definiert. QC kann anschließend die Methoden abspielen, die ursprünglich vom Client in MSMQ aufgezeichnet wurden. Dieses Schema gibt dem Autor einer Ausnahmeklasse die Möglichkeit, Code für den Eventualfall zu schreiben, in dem die Client-Methoden nicht erfolgreich abgespielt werden können. Ihnen stehen verschiedene Möglichkeiten zur Verfügung, um diese Methoden zu implementieren. Beachten Sie, dass QC davon ausgeht, dass Sie sich um alles kümmern, so dass die MSMQ-Nachricht gelöscht wird, wenn der Aufruf von `FinalServerRetry` und die Aufrufe der individuellen Methoden durchgeführt wurden. Wenn Sie einen Fehler in der Ausnahmeklasse auslösen, wird die Nachricht in die finale Abschlusswarteschlange verschoben.

Wie Sie sehen, stellt QC ein äußerst kompliziertes Gerüst zur Verfügung, um auf inkorrekte Nachrichten und Ausnahmefehler zu reagieren. Dies ist einer Hinsicht hilfreich, da das manuelle Erstellen eines solchen Gerüsts kein einfaches Unterfangen ist. Andererseits bietet das Gerüst keine ausgeprägte Flexibilität, die Sie in einigen Situationen jedoch benötigen. Wenn Ihre Applikation inkorrekte Nachrichten in einer anderen Weise behandeln muss, werden Sie wahrscheinlich auf QC verzichten wollen und stattdessen auf die reine MSMQ-Programmierung ausweichen. Der Nachteil dieser Lösung besteht jedoch darin, dass Sie Ihre eigene Listener-Applikation schreiben und Ihr eigenes Fehlerbehandlungsgerüst erstellen müssen.

Der COM+ Events-Dienst

COM+ Events gleicht QC darin, dass dieser Dienst ein Gerüst ist, der zu COM+ und Windows 2000 hinzugefügt wurde. Der Dienst hilft Ihnen, Benachrichtigungen für bestimmte Benutzer und/oder Applikationen zur Verfügung zu stellen, sobald interessante Ereignisse auftreten. Er ist eine neue Lösung für ein sehr altes Problem.

Viele Applikationen müssen andere Applikationen benachrichtigen, wenn interessante Dinge geschehen. Eine Applikation muss vielleicht eine andere darüber informieren, dass sich der Artikelpreis geändert hat oder dass der Artikelbestand einen gewissen Punkt erreicht hat. COM+ Events hilft Applikationen, Ereignisse auszulösen und darauf zu reagieren.

Im COM+ Events-Modell werden Applikationen, die Ereignisse auslösen, Verleger genannt. Komponenten und Applikationen, die Ereignisse empfangen und darauf reagieren, sind Abonnenten. Die Architektur des COM+ Events-Dienstes ist locker miteinander verbunden, da Verleger und Abonnent nichts voneinander wissen müssen. Das COM+ Events-System wird aus diesem Grund auch LCE (Loosely Coupled Events – Locker verbundene Ereignisse) genannt. Diese Trennung wird mit einer Referenzierungsschicht zwischen den Verlegern und Abonnenten erzielt, die auch als Ereignisklasse bezeichnet wird.

Eine Ereignisklasseninstanz ist ein Objekt, dessen Implementierung vom System zur Verfügung gestellt wird. Eine Ereignisklasse implementiert jedoch eine oder mehrere Schnittstellen, die von Ihnen definiert werden. Jede Schnittstelle definiert eine Gruppe mit Methoden, die mehrere Ereignisse repräsentieren. Wenn eine Verleger-Applikation ein Ereignisklassenobjekt instanziert, kann sie Ereignisse auslösen. Ein Verleger weiß jedoch nicht, welche Abonnenten mit einer Ereignisklasse verknüpft sind. Ein Abonnent wird über Konfigurationsdaten mit einer Ereignisklasse verknüpft. Diese Daten werden als Abonnement bezeichnet. Ein Ereignisklassenobjekt verwendet die im COM+-Katalog verwahrten Abonnementdaten, um die Ereignisse von registrierten Abonnenten zu lokalisieren und auszulösen. Abonnenten und Abonnements wissen nicht, welche Verleger Ereignisse auf der Ereignisklasse auslösen. Die Verleger und Abonnenten weisen somit keine Abhängigkeiten voneinander auf.

Diese locker verbundene Architektur bietet mehrere Vorteile. Sie müssen sich erstens keine Gedanken um Abonnenten machen, wenn Sie eine Verleger-Applikation erstellen. Dies bedeutet, dass es nur wenig Code zu schreiben und zu pflegen gibt. Sobald außerdem eine Verleger-Applikation in den Betrieb übernommen wurde, müssen Sie sie nicht mehr modifizieren, wenn Sie Abonnenten hinzufügen oder entfernen möchten. Sie müssen außerdem keine Abonnenten oder Abonnements modifizieren, wenn Sie die Verleger hinzufügen oder entfernen möchten, die Ereignisse auf der Ereignisklasse auslösen.

Die Architektur von COM+ Events

Die Architektur des COM+ Events-Modells ist in Abbildung 10.11 dargestellt. Das COM+ Events-System arbeitet mit vier Hauptentitäten: Verleger, Ereignis, Klassen, Abonnenten und Abonnements.

Ein Verleger ist eine Applikation, die Ereignisse auslöst. Ein Ereignisklassenobjekt agiert als Verteiler für die Versendung von Ereignisbenachrichtigungen. Sie definieren die Schnittstellen einer Ereignisklasse, aber COM+ implementiert die Ereignisklasse. Das Ereignisklassenobjekt lokalisiert jeden registrierten Abonnent und leitet Ereignisse weiter, wenn diese von einem Verleger ausgelöst werden. Ein Abonnent ist eine Klasse oder eine Applikation, die Ereignisse empfängt und darauf reagiert. Ein Abonnement besteht aus Konfigurationsdaten, die im COM+-Katalog aufbewahrt werden und einen Abonnenten in einer spezifischen Ereignisklasse an eine Schnittstelle binden. Ein Abonnement ist somit eine Anforderung, bestimmte Ereignisse zu empfangen.

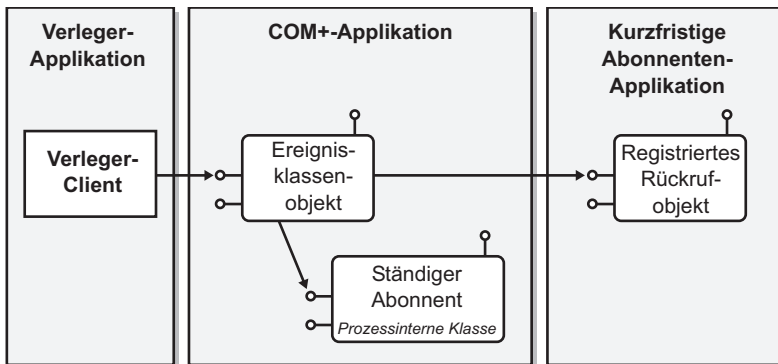


Abbildung 10.11: Die Architektur von COM+ Events basiert auf einem Verleger, einer Ereignisklasse, Abonnenten und Abonnements.

Es gibt zwei Abonnementtypen: ständige und kurzfristige Abonnements. Sie können ein ständiges Abonnement benutzen, um eine Komponente als Abonnent zu konfigurieren. Wenn ein Ereignis an ein ständiges Abonnement gesendet wird, erstellt der COM+ Events-Dienst ein neues Objekt aus einer konfigurierten Abonnentenkomponente und ruft die mit dem Ereignis verknüpfte Methode auf. Der Ereignisdienst »weckt« den Abonnent der Komponentenapplikation, sofern dieser noch nicht ausgeführt wird. Beachten Sie, dass immer dann ein Objekt für ein ständiges Abonnement erstellt und zerstört wird, wenn ein Ereignis ausgelöst wird.

Wenn Sie ein Rückrufobjekt mit einem kurzfristigen Abonnement registrieren, können Sie eine ausgeführte Applikation zu einem Abonnenten machen. Sobald eine Applikation ausgeführt wird, kann sie ein Rückrufobjekt mit einer spezifischen Ereignisklasse registrieren. Dies erlaubt einem kurzfristigen Abonnenten, eine Ereignissenke einzurichten, an die die Ereignisklasse Benachrichtigungen senden kann. Im Falle eines kurzfristigen Abonnements muss der Ereignisdienst kein Objekt erstellen und zerstören, wenn ein Ereignis ausgelöst wird. Ein großer Unterschied zwischen diesen beiden Abonnementtypen besteht darin, dass Sie ein kurzfristiges Abonnement dazu zwingt, ein neues Abonnement zu erstellen und ein aktives Rückrufobjekt programmgesteuert zu registrieren. Im Gegensatz zu einem kurzfristigen Abonnent können Sie ein Abonnement für einen ständigen Abonnent manuell konfigurieren. Dies geschieht mit dem administrativen Hilfsmittels *Komponentendienst*.

Beachten Sie, dass ein Ereignissystem auf einem spezifischen Computer vorhanden sein muss. Es gibt keine Integration mit Active Directory, um ein Ereignissystem als eine netzwerkweit gültige Ressource zu veröffentlichen. Dies bedeutet, dass das Auslösen und Verteilen von Ereignissen immer auf einem bestimmten Computer ausgeführt wird. Dies erhöht die Fehlertoleranz, da dieser Computer nur eine Angriffsfläche für Fehler im gesamten Ereignissystem präsentiert.

Ereignisklassen erstellen

Eine Ereignisklasse wird als eine spezielle konfigurierte Komponente registriert. Eine Ereignisklasse besteht aus einer oder mehreren Ereignisschnittstellen, die einer vom System zur Verfügung gestellten Implementierung einer Klasse zugeordnet sind. Sie definieren eine Ereignisklassenschnittstelle wie jede andere COM-Schnittstelle. Sie können dazu Visual Basic oder IDL verwenden. Der Einsatz von Visual Basic ist jedoch einfacher.

Um eine Ereignisklasse zu erstellen, müssen Sie eine Co-Klasse definieren, die die Schnittstelle oder Schnittstellen mit Ihren Ereignissen implementiert. Dies ist ein wenig seltsam, da Sie keine Implementierung für die Co-Klasse zur Verfügung stellen. COM+ übernimmt dies für Sie. Nachfolgend finden Sie eine oberflächliche Version des IDL-Sourcecodes zum Definieren einer Schnittstelle und Co-Klasse für eine Ereignisklasse:

```
library MyTypeLibrary {
    [ uuid(IID_MyEventInterface) ]
    interface MyEventInterface : IUnknown {
        HRESULT OnThisHappened( BSTR Param1 );
        HRESULT OnThatHappened( BSTR Param1, double Param2 );
    }
    [ uuid(CLSID_MyEventClass) ]
    coclass MyEventClass {
        [default] interface _MyEventInterface;
    }
}
```

Die mit der IDL-Definition einer Ereignisklasse verbundene Schwierigkeit besteht darin, dass Sie nicht einfach eine Typlibibliothek mit *MIDL.EXE* generieren und in einer COM+-Applikation installieren können. Die Typlibibliothek muss in eine selbstregistrierende DLL eingebettet werden. Dies ist mit C++-Code und der Visual C++-Entwicklungsumgebung, aber nicht mit der Visual Basic-IDE möglich.

Die einfache Möglichkeit, eine Ereignisklasse mit Visual Basic zu definieren, besteht darin, eine *MultiUse*-Klasse mit öffentlichen Methoden in einem ActiveX-DLL-Projekt zu erstellen. Wie Sie wissen, erzeugt Visual Basic eine Definition für die Standardschnittstelle und die Co-Klasse in der Typlibibliothek der DLL. Die sich daraus ergebende DLL ist selbstregistrierend und erfüllt alle Anforderungen für die Installation der Ereignisklasse in einer COM+-Applikation.

Wenn Sie eine Ereignisklasse zum ersten Mal mit Visual Basic erstellen, wird Ihnen alles ein wenig seltsam erscheinen, da Sie Methodensignaturen ohne Implementierungen definieren. Sie möchten vielleicht die *Instancing*-Eigenschaft der Klasse auf *PublicNotCreatable* setzen, da Sie ein Modul mit Methodensignaturen, aber ohne Implementierung definieren. Sie sollten dieser Versuchung jedoch widerstehen und die *Instancing*-Eigenschaft der Klasse, die auf *MultiUse* gesetzt ist, beibehalten. Nachfolgend finden Sie eine *MultiUse*-Klassendefinition, die zum Definieren einer Ereignisklasse verwendet werden kann:

```

' MultiUse-Klasse MyEventClass
Sub OnThisHappened(ByVal Param1 As String)
    ' Keine Implementierung
End Sub

Sub OnThatHappened(ByVal Param1 As String, ByVal Param2 As Double)
    ' Keine Implementierung
End Sub

```

Wenn Sie die Typbibliothek der DLL mit OLEVIEW.EXE dekompileieren, sehen Sie Zeilen, die den folgenden gleichen:

```

[ uuid(IID_Default_MyEventClass) ]
interface _MyEventClass : IDispatch {
    HRESULT OnThisHappened( BSTR Param1 );
    HRESULT OnThatHappened( BSTR Param1, double Param2 );
}
[ uuid(CLSID_MyEventClass) ]
coclass MyEventClass {
    [default] interface _MyEventClass;
}

```

Dieses Beispiel ist sehr einfach, da hier eine Ereignisklasse erstellt wird, die nur eine Schnittstelle implementiert. Dies ist die einfachste Lösung. Denken Sie daran, dass Sie Ereignisklassen definieren können, die mehrere Schnittstellen implementieren. Sie können außerdem die Polymorphie nutzen, indem Sie ein Design erzeugen, in dem mehrere Ereignisklassen dieselbe Schnittstelle implementieren.

Ereignisklassen installieren und konfigurieren

Sobald Sie die *MultiUse*-Klasse, die Ihre Ereignisklasse definiert, in eine ActiveX-DLL kompiliert haben, können Sie die Klasse in einer COM+-Applikation installieren. Wenn Sie eine Komponente mit dem administrativen Hilfsmittel *Komponentendienste* in einer COM+-Applikation installieren, können Sie zwischen der Installation einer neuen Komponente und der Installation einer neuen Ereignisklasse wählen. Sie sollten die Option zum Installieren einer neuen Ereignisklasse selektieren. Beachten Sie, dass die COM+-Admin-Objekte ebenfalls eine spezielle Methode für die Installation einer Ereignisklasse anbieten.

Wenn Sie eine neue Ereignisklasse installieren, verwendet COM+ nicht die von Ihrer DLL zur Verfügung gestellte Implementierung. Aus diesem Grund müssen Sie keine Implementierung für die Methoden der Komponente einbinden. COM+ erstellt stattdessen eine synthetische Klassenimplementierung aus einer vom System bereitgestellten Komponente in *ES.DLL*. COM+ verwendet die Schnittstellendefinition und die CLSID für die Ereignisklasse aus der DLL Ihrer Typbibliothek.

Eine Ereignisklasse ist eine konfigurierte Komponente, die dieselben Attribute wie andere konfigurierte Standardkomponenten aufweist. Sie besitzt jedoch einige zusätzliche Attribute, die sich ausschließlich auf eine Ereignisklasse beziehen. Abbildung 10.12 zeigt die Registerkarte *Erweitert* des Eigenschaftendialogfeldes einer Ereignisklasse. Sie können hier die LCE-Attribute von solchen konfigurierten Komponenten konfigurieren, die keine Ereignisklassen sind.

Sie können in das *Verleger-ID*-Feld einen benutzerfreundlichen Namen für die Ereignisklasse eingeben. Beachten Sie, dass die Verleger-ID mit der Ereignisklasse und nicht mit einer Verleger-Applikation verknüpft wird. Wenn Sie ein Abonnement für eine Ereignisklasse erstellen, können Sie die Ereignisklasse entweder anhand ihrer CLSID oder ihrer Verleger-ID identifizieren.

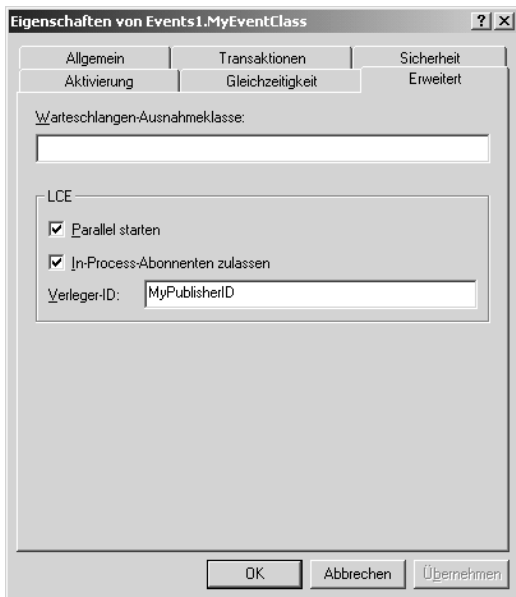


Abbildung 10.12: Die Registerkarte *Erweitert* des Eigenschaftendialogfeldes einer Ereignisklasse weist einige zusätzliche Einstellungen für die Konfiguration locker verbundener Ereignisse auf.

Wenn Sie das Kontrollkästchen *In-Process-Abonnenten zulassen* deaktivieren, verhindern Sie, dass Abonnenten in demselben Prozess wie die Ereignisklasse ausgeführt werden. Diese Option ist standardmäßig selektiert. Sie können Sie in einigen Situationen deaktivieren, um eine bessere Sicherheit und Fehlertoleranz zu erzielen.

Die letzte Option auf der Registerkarte *Erweitert* ist *Parallel starten*. Sie informiert das COM+-Ereignissystem, ob verschiedene Abonnenten auf verschiedenen Threads benachrichtigt werden sollen. Diese Option ist standardmäßig nicht selektiert. Dies bedeutet, dass das Ereignissystem nur einen Thread verwendet, um die Abonnenten sukzessive zu benachrichtigen. Wenn Sie diese Option selektieren, benutzt das COM+-Ereignissystem mehrere Threads, um die Abonnenten gleichzeitig zu benachrichtigen.

Ständige Abonnements konfigurieren

Sie können eine Komponente als Abonnent konfigurieren, indem Sie ein ständiges Abonnement erstellen. Abonnements werden wie Ereignisklassen mit dem Ereignissystem registriert, und ihre Konfigurationsdaten werden im COM+-Katalog gespeichert. Ein ständiges Abonnement muss entweder mit der CLSID oder der Verleger-ID einer Ereignisklasse verknüpft werden. Das Abonnement muss darüber hinaus mit der IID konfiguriert werden, die die Ereignisse definiert.

Stellen Sie sich vor, Sie möchten ein ständiges Abonnement für die Ereignisklasse erstellen, die bereits in diesem Kapitel beschrieben wurde. Sie können eine Abonnentenkomponente in einem anderen ActiveX-DLL-Projekt erstellen, indem Sie eine *MultiUse*-Klasse verwenden. Vergessen Sie aber nicht, dass diese Abonnentenkomponente dieselbe Schnittstelle wie die Ereignisklasse implementieren muss.

Wenn Sie das ActiveX-DLL-Projekt für die Abonnentenkomponente erstellen, sollten Sie die Typbibliothek in der DLL referenzieren, die die Ereignisklasse enthält. Dies erlaubt Ihnen, die Standardschnittstelle zu implementieren, die von Ihrer Ereignisklasse verwendet wird. Hier ein Beispiel für eine Abonnentenkomponente, die dieselbe Schnittstelle wie die Ereignisklasse implementiert:

Implements MyEventClass

```
Private Sub MyEventClass_OnThisHappened(ByVal Param1 As String)
    ' Implementierung für Ereignisantwort hier einfügen.
End Sub

Sub MyEventClass_OnThatHappened(ByVal Param1 As String, Param2 As Double)
    ' Implementierung für Ereignisantwort hier einfügen.
End Sub
```

Sobald Sie diese Komponente in eine ActiveX-DLL kompiliert haben, können Sie sie in eine COM+-Applikation installieren und als ständigen Abonnent konfigurieren. Sie installieren eine Abonnentenkomponente ebenso wie eine Standardkomponente in einer COM+-Applikation. Installieren Sie die Komponente nicht als Ereignisklasse.

Eine Option besteht darin, die Abonnentenkomponente in dieselbe COM+-Applikation zu installieren, in der sich auch die Ereignisklasse befindet. Wenn Sie den Abonnent jedoch in einem separaten Prozess ausführen lassen möchten, sollten Sie die Abonnentenkomponente in eine andere COM+-Serverapplikation auf demselben Computer installieren.

Sobald Sie die Abonnentenkomponente installiert haben, können Sie für diese ein Abonnement erstellen, indem Sie einen Rechtsklick auf dem *Abonnements*-Ordner der Komponente im administrativen Hilfsmittel *Komponentendienste* ausführen. Wählen Sie anschließend den Eintrag *Abonnement* aus dem Menü *Neu* aus. Im Anschluss daran wird der Assistent zum Erstellen von neuen COM-Abonnements aufgerufen. Der Assistent präsentiert Ihnen eine Liste der Schnittstellen (IIDs), die von der Abonnentenkomponente implementiert wurden, und bittet Sie, eine Schnittstelle für das Abonnement zu selektieren. Der Assistent zeigt anschließend eine Liste mit allen Ereignisklassen auf dem lokalen Computer an, die diese Schnittstelle ebenfalls implementieren. Auch Ihre Ereignisklasse sollte in dieser Liste aufgeführt sein, sofern Sie die Klasse korrekt konfiguriert haben. Die letzte Seite des Assistenten lässt Sie dem Abonnement einen benutzerfreundlichen Namen zuweisen und erlaubt Ihnen, das Abonnement zu aktivieren.

Sobald Sie eine Komponente als ständigen Abonnent konfiguriert haben, erstellt das COM+-Ereignissystem immer dann eine Instanz der Komponente, wenn ein Verleger ein Ereignis auf der verknüpften Ereignisklasse auslöst. Denken Sie daran, dass das Ereignissystem die Abonnentenkomponente benutzt, um immer dann ein Objekt zu erstellen und zu zerstören, wenn ein Ereignis ausgelöst wird. Vergessen Sie auch nicht, dass ständige Abonnements im Gegensatz zu kurzfristigen Abonnements das Herunterfahren des Systems überstehen.

Verleger erstellen

Wenn Sie eine Ereignisklasse und einen oder mehrere ständige Abonnenten generiert haben, ist das Erstellen einer Verleger-Applikation sehr einfach. Jede Applikation, die ein Ereignisklassenobjekt instanziiert, kann Ereignisse auslösen. Wenn Sie eine Verleger-Applikation erzeugen, generieren Sie einfach eine Instanz der Ereignisklasse und rufen eine Methode auf, um Ereignisse auszulösen. Wenn Sie planen, die Verleger-Applikation auf einem anderen Computer als die Ereignisklasse ausführen zu lassen, sollten Sie das COM+-Export-Feature verwenden, um einen Applikationsproxy zu erstellen. Dies ist erforderlich, um Konfigurationsinformationen zur Ereignisklasse auf dem Computer der Verleger-Applikation zu installieren. Hier ein einfaches Beispiel für den Code, der in einer Verleger-Applikation ein Ereignis auslöst. Wie Sie sehen, ist der Code nicht besonders kompliziert:

```

Sub FireEventFromPublisher()
    Dim MyEventClassObject As CMyEventClass
    Set MyEventClassObject = New CMyEventClass
    MyEventClassObject.OnThisHappened "this thing just happened"
    Set MyEventClassObject = Nothing
End Sub

```

Die Verleger-Applikation löst ein Ereignis aus, indem sie eine Methode des Ereignisklassenobjekts aufruft. Das Ereignis selbst wird jedoch von dem Ereignissystem an den Abonnent weitergeleitet. Das Ereignissystem ist dafür verantwortlich, im COM+-Katalog nach Abonnementinformationen zu suchen und das Ereignis an jeden Abonnent weiterzuleiten. Wie Sie sich bestimmt erinnern, können Sie eine Ereignisklasse so konfigurieren, dass sie Ereignisse für Abonnenten nacheinander oder gleichzeitig auslöst. In beiden Fällen ist die Verleger-Applikation blockiert, bis alle Abonnenten ihre Ereignisbenachrichtigungen bearbeitet haben. Das Ereignissystem stellt somit nicht selbst ein asynchrones Verhalten zur Verfügung. Ich werde im nächsten Abschnitt beschreiben, wie COM+-Ereignisse mit QC kombiniert werden, um Ereignisse asynchron von einer Verleger-Applikation auslösen zu lassen.

Ereignisse asynchron auslösen

COM+-Ereignisse werden standardmäßig synchron, also nacheinander ausgelöst. Eine Verleger-Applikation muss warten, bis der Abonnent das Ereignis bearbeitet und die Steuerung an das Ereignissystem zurückgegeben hat. Auch wenn Sie das Attribut zum parallelen Auslösen von Ereignissen für eine Ereignisklasse aktiviert haben, ist der Verleger blockiert, bis jeder Abonnent sein Ereignis bearbeitet hat.

Eine skalierbare Ereignisbenachrichtigungsarchitektur erfordert die asynchrone Kommunikation. Eine verteilte Applikation ist häufig nicht geeignet, wenn die Verleger-Applikation warten muss, während jeder Abonnent seine Ereignisbenachrichtigungen bearbeitet. Einige Systeme können außerdem erfordern, dass die Ereignisbenachrichtigungen asynchron und verbindungslos versendet werden.

Alles bisher über das COM+ Events-System Gesagte basiert auf COM und RPC, und nicht auf MSMQ. COM+ Events wurde glücklicherweise in QC integriert, um die Ereignisbenachrichtigung mit den Vorteilen der asynchronen, verbindungslosen Kommunikation zu verbinden. Ereignisklassen und ständige Abonnenten können so konfiguriert werden, dass sie QC anstelle der synchronen COM-Aufrufe verwenden. COM+ Events führt für Unternehmen mit Servercomputern, die häufig offline sind oder mit einem hohen Verkehrsaufkommen konfrontiert werden, zu vielen Einschränkungen, sofern die Ereignisse nicht mit QC ausgelöst und empfangen werden.

Wenn Sie das COM+ Events-System mit QC benutzen möchten, werden alle Einschränkungen der Warteschlangenkomponenten zu Einschränkungen des COM+ Events-Modells. Ihre Ereignisschnittstellen müssen dieselben Restriktionen wie jede andere Warteschlangenschnittstelle auch ertragen. Dies bedeutet, dass alle Ereignisse mit ByVal-Parametern und ohne Rückgabewerte definiert werden müssen. Verleger-Applikationen und Abonnenten sind außerdem auf Windows 2000-Computer beschränkt.

Kurzfristige Abonnenten

In diesem Kapitel wurden bereits die grundlegenden Schritte zum Erstellen eines kurzfristigen Abonnenten beschrieben. Da ständige Abonnenten einfacher als kurzfristige Abonnenten verwendet werden können, möchte ich nur oberflächlich erläutern, wie kurzfristige Abonnenten arbeiten. Wie Sie wissen, können Sie einen ständigen Abonnenten vollständig manuell konfigurieren.

Es ist schwieriger, eine kurzfristige Abonnentenapplikation zu erzeugen, da Sie Code schreiben müssen, um das kurzfristige Abonnement zur Laufzeit zu erstellen. Dies erfordert die Manipulation des COM+-Katalogs, indem Code geschrieben wird, der auf die COM+-Admin-Objekte zugreift.

Ein kurzfristiges Abonnement gleicht einem ständigen Abonnement in der Hinsicht, dass es mit einer spezifischen Ereignisklasse und einer bestimmten IID konfiguriert werden muss. Der kurzfristige Abonnent ist wie ein ständiger Abonnent, da er eine Komponente zur Verfügung stellen muss, die dieselbe Schnittstelle wie die Ereignisklasse implementiert. Die beiden Abonententypen sind jedoch hinsichtlich des Objektlebenszyklus und der Erstellungsweise sehr unterschiedlich.

Sobald die ausgeführte Abonnentenapplikation unter Verwendung einer spezifischen Ereignisklasse und einer bestimmten Schnittstelle ein kurzfristiges Abonnement erstellt, muss sie ein Rückrufobjekt registrieren, das diese Schnittstelle ebenfalls implementiert. Dieses Rückrufobjekt stellt eine Senke zur Verfügung, die vom Ereignissystem benutzt wird, um Ereignisbenachrichtigungen zu versenden. Im Gegensatz zur Vorgehensweise für ein ständiges Abonnement, muss das Ereignissystem nicht immer dann ein Objekt erstellen und zerstören, wenn ein Ereignis ausgelöst wird. Das Ereignissystem kann somit viele Ereignisse für ein langlebiges Rückrufobjekt auslösen.

Kurzfristige Abonnements weisen einige Nachteile auf. Das COM+ Events-System erlaubt Ihnen leider nicht, einen kurzfristigen Abonnent asynchron zu benachrichtigen. Ereignisse können nur mit COM und nicht mit MSMQ weitergeleitet werden. Dies bedeutet, dass kurzfristige Abonnenten darauf beschränkt sind, innerhalb desselben LANs wie der Servercomputer mit der Ereignisklasse ausgeführt zu werden. Synchrone Rückrufe begrenzen außerdem die Skalierbarkeit.

Auch die Konfiguration der Sicherheit kann schwierig sein, wenn Sie kurzfristige Abonnenten einsetzen. Der Grund hierfür besteht darin, dass die Client-Applikation oder ein benutzerdefiniertes, serverseitiges Registrierungsobjekt unter der Identität eines Kontos ausgeführt werden muss, das administrative Zugriffsberechtigungen besitzt. Nur dann kann in den COM+-Katalog, der sich auf dem Computer mit der Ereignisklasse befindet, geschrieben werden.

Der letzte Nachteil eines kurzfristigen Abonnements besteht darin, dass es nicht mehr nützlich ist, wenn die Abonnentenapplikation oder der Computer mit der Ereignisklasse heruntergefahren wird. Wenn Sie ein kurzfristiges Abonnement nicht explizit aus dem COM+-Katalog entfernen, wird es automatisch gelöscht, sobald das System neu gestartet wird.

Zusätzliche COM+ Events-Features

Lassen Sie mich kurz einige zusätzliche Features des COM+ Events-Modells nennen. Das Modell unterstützt das Filtern von Ereignissen, so dass Sie selektieren können, welche Ereignisse an die Abonnenten geschickt werden sollen. Ein Abonnement besitzt ein FilterCriteria-Attribut, das Sie für jeden Parameter setzen können, um Ereignisse anhand bestimmter Kriterien zu filtern. Sie erhalten eine weitaus größere Kontrolle, wenn Sie ein benutzerdefiniertes Filterobjekt erstellen und zwischen einer Verleger-Applikation und einem Ereignisobjekt platzieren. Dies gibt Ihnen die Möglichkeit, Ereignisse für einzelne Abonnenten zu filtern. Sie können außerdem ermitteln, welche Abonnenten zuerst Ereignisbenachrichtigungen erhalten.

Weitere Informationen über COM+ Events finden Sie in dem Bericht »*The COM+ Event Service Eases the Pain of Publishing and Subscribing to Data*« von David Platt, der 1999 in der September-Ausgabe des MSJ (Microsoft Systems Journal) erschien. Sie finden diesen Artikel ebenfalls in der MSDN-Bibliothek (suchen Sie nach »Platt«). David stellt auch Details über QC zur Verfügung. Dies geschieht in dem Artikel »*Building a Store-and-Forward Mechanism with Windows 2000 Queued Components*«, der 1999 in der Juni-Ausgabe von MSJ erschien. Auch

diesen Artikel finden Sie im MSDN. David beschreibt in diesen Artikeln einige Details, die nicht in diesem Kapitel behandelt wurden. Er bietet außerdem einige Schritt-für-Schritt-Anleitungen an, die Ihnen helfen können, wenn Sie lernen möchten, wie Visual Basic-Komponenten konfiguriert werden, die diese beiden COM+-Dienste nutzen.

Die Wahl zwischen MSMQ, QC und COM+ -Events

Nachdem die Grundlagen von MSMQ, QC und COM+ Events beschrieben wurden, werden wir nun einen Schritt zurückgehen und einige wichtige Design-Aspekte erläutern. Stellen Sie sich vor, Sie müssen eine asynchrone, verbindungslose Kommunikation zwischen zwei Applikationen einrichten. Wie Sie gesehen haben, müssen Sie nicht direkt mit MSMQ programmieren, wenn Sie QC verwenden. Sie verzichten dann jedoch auf ein wenig Flexibilität. Wenn QC die Funktionalität bietet, die Sie benötigen, können Sie sehr viel Zeit und Aufwand sparen. Sie sollten jedoch daran denken, dass die MSMQ-Programmierung viele Möglichkeiten bietet, die von QC nicht unterstützt werden.

Die Kommunikation mit Computern, die Windows NT, Windows 95 und Windows 98 ausführen, ist häufig ein wichtiger Aspekt. Wenn dies notwendig ist, müssen Sie direkt mit MSMQ arbeiten. Viele Unternehmen haben festgestellt, dass QC die beste Wahl für eine Server-/Server-Kommunikation ist. QC vereinfacht zum Beispiel das Versenden von asynchronen Anforderungen, die nur in eine Richtung von einem Webserver zu einem Applikationsserver übermittelt werden. Sie sollten für eine Applikation, die eine komplexe asynchrone Kommunikation benötigt, viel Zeit und Geld investieren, um Ihren Code mit MSMQ zu schreiben.

Sie haben gesehen, dass der COM+ Events-Dienst ein wertvolles Gerüst für die Weiterleitung von Ereignisbenachrichtigungen anbietet. Es ist jedoch wichtig, dass Sie dessen Stärken und Einschränkungen verstehen. Viele Entwickler überschätzen die Fähigkeiten dieses Dienstes und entscheiden dann, nachdem Sie viel Zeit für die Erarbeitung von Prototypen und die Durchführung stressiger Tests geopfert haben, dass die direkte MSMQ-Programmierung zu besser adaptierbaren und leistungsorientierten Lösungen führt.

Microsoft hat sehr viel Werbung für QC und COM+ Events betrieben und beide Dienste als neue wertvolle Features bezeichnet. Über MSMQ wurden jedoch nur wenige Worte verloren. MSMQ war schließlich bereits Bestandteil der Plattform und somit kein heißes, neues Verkaufsargument.

Da ich kein Marketing-Experte bin, versuche ich, Sie davon zu überzeugen, dass MSMQ eines der größten Verkaufsargumente der Windows 2000-Plattform ist. Ich möchte nicht suggerieren, dass QC und COM+ Events nicht nützlich sind. Ich mag QC, da es Visual Basic-Entwicklern einen Multi-Threaded-Listener-Dienst und eine nützliche Fehlerbehandlung zur Verfügung stellt. Ich empfehle Ihnen lediglich, MSMQ zu benutzen, wenn dies Ihr Design erfordert.

Lassen Sie uns dieses Kapitel mit einem Beispiel beenden, das MSMQ und QC gemeinsam verwendet. Stellen Sie sich vor, Sie müssten hunderten von Client-Applikationen Ereignisbenachrichtigungen zukommen lassen, wenn ein Warenpreis unter einen bestimmten Grenzwert fällt. Sie beginnen damit, eine Warteschlangenkomponeute zu entwerfen, die auf einem Windows 2000-Servercomputer ausgeführt werden soll. Eine Applikation, die auf einem anderen Windows 2000-Computer ausgeführt wird, könnte diese Warteschlangenkomponeute benutzen, um asynchron eine Ereignisbenachrichtigung auszulösen.

Wenn der QC-Listener-Dienst die sich hieraus ergebende Nachricht bearbeitet, erstellt er eine Instanz Ihrer Warteschlangenkomponeute und benachrichtigt Sie über registrierte Clients. Wenn Ihre Warteschlangenkomponeute für MSMQ programmiert wurde, können Sie Benachrichtigungen an Client-Applikationen senden, die unter Windows NT, Windows 95 oder Windows 98 aus-

geführt werden. Diese Lösung fördert die Skalierbarkeit, da die Ereignisse asynchron ausgelöst und verteilt werden.

Bei einer Lösung wie dieser müssen Sie eine Möglichkeit finden, um Clients zu registrieren, die Benachrichtigungen empfangen möchten. Eine Möglichkeit besteht darin, eine private Warteschlange auf jedem Desktop zu erstellen. Der Formatname jeder Warteschlange wird dann in einer Datenbank oder einer XML-Datei überwacht, die sich auf dem Computer mit den Warteschlangenkompontenten befindet, die wiederum Ereignisbenachrichtigungen weiterleiten. Wenn eine Warteschlangenkompontente eine Ereignisbenachrichtigung empfängt, kann sie alle Formatnamen durchlaufen und eine Nachricht für jeden Client vorbereiten und versenden. MSMQ unterstützt das Verteilen von Nachrichten leider nicht, so dass Sie eine separate Nachricht für jede individuelle clientseitige Warteschlange vorbereiten und versenden müssen.

Zusammenfassung

Sie haben in diesem Kapitel einige Techniken kennen gelernt, um Visual Basic für die Integration des Messagings und der asynchronen Kommunikation in eine verteilte Applikation zu benutzen. Ich hoffe, dass Sie in Erinnerung behalten haben, warum und wie Daten mit MSMQ-Nachrichten über das Netzwerk bewegt werden.

Das Messaging ist für Szenarien geeignet, in denen RPC und HTTP fehlschlagen. Die asynchrone Kommunikation kann die Antwortzeiten und den Systemdurchsatz signifikant erhöhen. Sie verbessert außerdem die Verfügbarkeit, da sie eine zuverlässige Kommunikation zwischen Computern zur Verfügung stellt, deren Verbindung aus verschiedenen Gründen getrennt werden kann.

Die transaktionale Programmierung führt das Messaging auf eine andere Ebene. Wie Sie gesehen haben, kann MSMQ eine Zuverlässigkeit bieten, die mit RPC oder HTTP nicht möglich ist. Anforderungsnachrichten können mit der Einmal-Übermittlungssemantik übertragen werden. Sie haben aber auch gesehen, dass transaktionale Nachrichten eine gewisse Komplexität bedingen. Jemand muss sich um inkorrekte Nachrichten und die Fehlerbehandlung kümmern.

Sie sollten außerdem mit den Stärken und Schwächen jedes COM+-Dienstes vertraut sein, so dass Sie eine kluge Entscheidung darüber treffen können, ob MSMQ oder QC verwendet werden soll, wenn Sie eine asynchrone oder eine nicht verbundene Kommunikation benötigen. In einigen Applikationen kann QC Ihren Anforderungen gerecht werden und Ihnen helfen, Zeit und Geld zu sparen. Andere Applikationen können die MSMQ-Programmierung erfordern, um eine spezifische Funktionalität zu implementieren oder Clients zu erreichen, die unter einer älteren Version von Windows ausgeführt werden. Sie haben gesehen, dass Sie bisweilen kreativ sein und beide Dienste in nur eine Lösung integrieren können.

11 COM+-Sicherheit

336	Sicherheit in Windows 2000
357	Sicherheit in COM+
368	Sicherheit in IIS
377	Weitere Quellen
377	Zusammenfassung

Die Bedeutung der Sicherheit in einer verteilten Applikation kann nicht oft genug betont werden. Die Daten, die für den reibungslosen Arbeitsablauf verantwortlich sind, bilden den wertvollsten Besitz Ihres Unternehmens. Wenn Sie den Zugriff auf diese Daten über eine COM+-Applikation ermöglichen, sollten Sie unbedingt wissen, wie Sie Sicherheitsrichtlinien entwerfen und deren Einhaltung erzwingen. Eine effektive Sicherheitsrichtlinie muss die Bösen von den Aktionen abhalten, die Sie nicht zulassen möchten. Die gleichen Richtlinien müssen vertrauenswürdigen Benutzern jedoch erlauben, die Applikation in der beabsichtigten Weise zu verwenden, ohne frustrierende »Zugriff verweigert«-Meldungen zu erhalten.

Innerhalb eines privaten Netzwerks, wie zum Beispiel in einem unternehmenseigenen LAN, können Sie eine Sicherheitsrichtlinie entwerfen, um gewöhnlichen Benutzern eine niedrigere Zugriffsebene zu gewähren, während den Managern eine höhere Zugriffsebene zugewiesen wird. Sie können den Managern zum Beispiel Lese- und Schreib-Zugriffsberechtigungen für Daten gewähren, auf die andere Benutzer nicht zugreifen dürfen. Wenn Sie eine verteilte Applikation in einem öffentlichen Netzwerk wie zum Beispiel dem Internet publizieren, wird die Sicherheit noch wichtiger. Wenn Sie die Applikation nicht richtig absichern, kann ein Konkurrent oder eine andere Ihnen nicht geneigte Partei die Daten entwenden oder zerstören, die den Fortbestand Ihres Unternehmens sichern.

Das Realisieren der Sicherheit in einer dreischichtigen Applikation unterscheidet sich von der entsprechenden Vorgehensweise für eine klassische zweischichtige Applikation. Es erfordert ein Umdenken. Sie müssen viele Kernkonzepte des Windows-Sicherheitsmodells verstehen, bevor Sie die zusätzlichen sicherheitsbezogenen Features nutzen können, die COM+ anbietet.

COM+ übernimmt einige wichtige Sicherheitsaspekte von dem zugrunde liegenden Betriebssystem. Dieses Kapitel beginnt deshalb mit einer kurzen Einführung zur Windows-Sicherheit, um Ihnen einen Eindruck von der Authentifizierung und Autorisierung zu vermitteln. Sie erfahren außerdem, wie Anmeldungssitzungen mit Tokens verknüpft sind und wie ein Prozess eine Identität etabliert. Sobald Sie diese Kernkonzepte verstanden haben, wird es einfacher für Sie sein, die Sicherheits-Features von COM+ zu benutzen.

Die COM+-Sicherheit basiert auf Rollen. Wie Sie sehen werden, bietet das Rollenmodell sowohl eine deklarative als auch eine programmgesteuerte Sicherheit an. Sie werden in einigen Applikationen jede sicherheitsbezogene Programmierung vermeiden wollen und stattdessen eine administrative Sicherheit mit dem administrativen Hilfsmittel *Komponentendienste* realisieren. In anderen Applikationen, die bedingte Autorisierungsüberprüfungen oder benutzerdefinierte Überwachungsrichtlinien erfordern, können Sie einen sicherheitsbezogenen Code schreiben, um Ihre deklarativen Richtlinien zu vervollständigen.

Die Sicherheit ist besonders in der heutigen Welt mit ihren miteinander verbundenen Computern wichtig. Nachdem erörtert wurde, wie die COM+-Sicherheit in einem privaten Netzwerk arbeitet, erfahren Sie, wie COM+ mit der IIS-Sicherheit interagiert. Viele von Ihnen werden schließlich Webapplikationen erstellen, um Benutzer über das Internet zu erreichen. Wenn Sie eine Client-Applikation über ASP-Seiten zugänglich machen möchten, sollten Sie die IIS-Authentifizierung verstehen. Sie sollten außerdem wissen, wie Zugriffsberechtigungen für Visual Basic-Komponenten konfiguriert werden. Lassen Sie mich noch einmal erwähnen, dass es wichtig ist, vertrauenswürdigen Benutzern die benötigten Zugriffsberechtigungen zukommen zu lassen, während die Bösen davon abgehalten werden müssen, Chaos anzurichten.

Das umfangreiche Thema Windows-Sicherheit kann nicht in nur einem Kapitel vollständig beschrieben werden. Ich werde jedoch das Hintergrundwissen vermitteln, das notwendig ist, um verteilte Applikationen zu entwickeln und einzusetzen, die die integrierten Sicherheits-Features von Windows 2000 und COM+ nutzen. Wenn Sie alles über die Windows-Sicherheit erfahren möchten, sollten Sie weitere Bücher lesen. Eine exzellente Quelle, die die internen Details des Windows-Sicherheitsmodells beschreibt, ist das Buch »*Programming Windows Security*« von Keith Brown (Addison Wesley, 2000). Keith ist einer der führenden Experten in seinem Bereich. Sein Buch vervollständigt dieses Kapitel.

Sicherheit in Windows 2000

Um die Möglichkeiten der COM+-Sicherheit erfassen zu können, müssen Sie einige Kernkonzepte des zugrunde liegenden Windows-Sicherheitsmodells verstehen. Wenn Sie bereits vertraut mit diesem Thema sind, können Sie diesen Abschnitt überspringen. Nehmen Sie sich andernfalls ein wenig Zeit, um diesen Abschnitt sorgfältig zu lesen, bevor Sie sich mit der COM+-Sicherheit beschäftigen.

Der erste Schritt, der für ein Verständnis der Sicherheit erforderlich ist, besteht darin, die Arbeitsweise der Authentifizierung kennen zu lernen. Sie sollten wissen, was ein Sicherheitshauptname (security principal) ist und wie sich dieser von einem Konto unterscheidet. Sie sollten auch den Zweck einer Autorität kennen und verstehen, wie diese eine Kontendatenbank verwaltet. Wie Sie noch sehen werden, sind Domänencontroller spezielle Computer, die eine zentrale Rolle spielen, wenn einem Hauptnamen erlaubt wird, seine Identität nachzuweisen.

Ich werde ebenfalls beschreiben, wie Windows die Sicherheit zur Laufzeit erzwingt. Dies geschieht mit Autorisierungsüberprüfungen. Windows benutzt Anmeldungssitzungen und Tokens, um Autorisierungsinformationen über Benutzer in einem Cache zu speichern. Windows benutzt außerdem Kontrolllisten (DACL – Discretionary Access Control List) für den wahlweisen Zugriff, um Benutzern und Gruppen bestimmte Berechtigungen für spezifische Ressourcen zuzuweisen. Diese grundlegenden Bausteine erlauben Ihnen, eine effektive Sicherheitsrichtlinie zu konfigurieren.

Dieser Abschnitt konzentriert sich ebenfalls auf die zusätzliche Komplexität, mit der Sie konfrontiert werden, wenn Sie die Authentifizierung über das Netzwerk durchführen. Windows bietet

glücklicherweise transparente Authentifizierungsdienste an. Eine Client-Applikation, die auf einem Desktop-Computer ausgeführt wird, kann sicher mit einer COM+-Applikation auf einem Servercomputer kommunizieren, wenn ein so genannter Vertrauenspfad zwischen beiden existiert. Sie werden sehen, dass das allumfassende Konzept des Vertrauens jeden Aspekt der Sicherheit durchdringt.

Hauptnamen und Autoritäten

Die Windows-Sicherheit basiert auf einer zentralen Entität, die als Hauptname bekannt ist. In Windows 2000 sind Benutzer und Computer so genannte Sicherheitshauptnamen. (Unter Windows NT sind Computer nicht wirklich Hauptnamen.) In diesem Sicherheitsmodell identifiziert sich ein Hauptname gegenüber einem anderen Hauptnamen mit Hilfe eines Prozesses, der Authentifizierung genannt wird. Ohne die Authentifizierung wäre es nicht möglich, die Einhaltung einer Sicherheitsrichtlinie durchzusetzen. Sie wüssten nicht, wem Sie Zugriff gewähren oder verweigern würden.

Die Authentifizierung wird von einer Autorität durchgeführt. Eine Autorität ist ein Windows-Dienst, der die Identität eines Hauptnamens überprüft. Wenn Bob (Hauptname 1) zum Beispiel sein Kennwort in ein Anmeldedialogfeld eingibt und versucht, sich an einem lokalen Computer (Hauptname 2) anzumelden, wird eine Autorität aufgerufen, um zu ermitteln, ob sich wirklich Bob anmeldet. Die Autorität verwahrt eine verschlüsselte Kopie von Bobs Kennwort und vergleicht diese mit der Eingabe in das Anmeldedialogfeld. Stimmen beide überein, hat Bob seine Identität bewiesen und die Autorität informiert den lokalen Computer, dass sich wirklich Bob angemeldet hat.

Jede Autorität besitzt eine Kontendatenbank mit sicherheitsbezogenen Konfigurationsdaten. Die Autorität verwahrt ein separates Konto für jeden ihr bekannten Hauptnamen. Wenn ein Konto erstellt wird, generiert die Autorität einen eindeutigen Bezeichner, der als Sicherheits-ID (SID) bezeichnet wird. Eine SID gleicht einer GUID in der Hinsicht, dass sie ein vom System generierter, eindeutiger Wert ist. Die Struktur einer SID unterscheidet sich jedoch von der einer GUID. Eine SID enthält immer die ID der Autorität, die sie erstellt hat.

Ein Hauptnamenkonto speichert sicherheitsbezogene Daten für einen spezifischen Benutzer oder Computer. Ein Benutzerkonto kann zum Beispiel Gruppenzuweisungen und ein Kennwort überwachen. Der Umstand, dass die Autorität Benutzerkennworte überwacht, erlaubt den Benutzern, die eigene Identität zur Laufzeit nachzuweisen. Ein Kennwort ist nur ein Beispiel für Hauptnamensbestätigungen zum Nachweis der eigenen Identität. In einigen Szenarien verwenden Hauptnamen exotischere Bestätigungsformen, um sich zu identifizieren. Dazu zählen digitale Zertifikate und so genannte Smartcards.

Lokale Konten im Vergleich mit Domänenkonten

Jeder Windows 2000- und Windows NT-Computer führt eine Autorität aus, die über eine Kontendatenbank verfügt. Einige Computer führen eine lokale Sicherheitsautorität (LSA – Local Security Authority) aus, die lokale Konten pflegt. Andere Computer führen eine Domänenautorität aus, die Domänenkonten verwaltet. Der Autoritätsdienst wird in beiden Fällen von *LSASS.EXE* nach dem Computerstart gestartet.

Unter Windows 2000 können Hauptnamen mit verschiedenen Namensformaten identifiziert werden. Windows 2000 und Active Directory verwenden das Benutzer-Hauptnamensformat (UPN – User Principal Name) *hauptname@autorität*. Ein Domänenkonto liegt beispielsweise im

Format *Bob@MeineDomäne* vor. In den meisten Fällen können Sie einen Hauptnamen auch mit dem älteren Windows NT-Format *Autorität\Hauptname* identifizieren, wie zum Beispiel in *MeinComputer\Bob* oder *MeineDomäne\Bob*. Bisweilen müssen Sie sogar das alte Format verwenden. Dies ist zum Beispiel der Fall, wenn Sie einen Kontennamen zuweisen, der als Identität einer COM+-Applikation dienen soll.

Sie können die Sicherheit auch innerhalb einer relativ kleinen Computergruppe ohne Domänenautorität konfigurieren. Ein Benutzer kann zum Beispiel das lokale *Gast*-Konto freigeben, damit andere Benutzer ohne Identitätsbestätigung auf lokale Ressourcen wie zum Beispiel ein gemeinsam genutztes Verzeichnis zugreifen können. Diese Lösung bietet jedoch keine feine Strukturierung (granularity). Wenn Sie Zugriff auf eine Ressource gewähren, muss dies für jeden Anwender im Netzwerk geschehen. Wenn Sie nur bestimmten Benutzern oder einer Gruppe Zugriff gewähren möchten, muss jeder Benutzer von einer Autorität bestätigt werden.

Sie können die Identität der Benutzer in einer Umgebung ohne Domänenautorität bestätigen, indem Sie lokale Kontenkopien verwenden. Jeder Benutzer muss ein Konto auf jedem Computer im Netzwerk besitzen. Jedes dieser Konten, die mit einem Benutzer verknüpft sind, muss über denselben Benutzernamen und dasselbe Kennwort verfügen. Wenn Bob zum Beispiel an Computer A angemeldet ist und versucht, auf eine Ressource von Computer B zuzugreifen, arbeiten die Sicherheitsautoritätsdienste der beiden Computer zusammen, um Bobs Identität zu bestätigen. Wenn die Autorität auf Computer B feststellt, dass es ein lokales Konto mit demselben Benutzernamen und Kennwort gibt, mit denen sich Bob an Computer A angemeldet hat, ist die Identität von Bob bestätigt.

Das Problem der Kontenkopien besteht darin, dass sie schwer zu administrieren sind. Ein Netzwerk mit zehn Benutzern und zehn Computern erfordert zum Beispiel einhundert Konten. Stellen Sie sich vor, wie schwer es für Bob wäre, sein Kennwort zu ändern. Er müsste sich an jedem Computer des Netzwerks anmelden, um dies zu tun. Wie Sie sehen, ist die Verwaltung der Sicherheit mit Kontenkopien nur in kleinen Netzwerken durchführbar. Diese Lösung ist nicht skalierbar, da die administrative Last mit jedem neuen Benutzer und Computer zunimmt.

Um eine Sicherheitsrichtlinie zu erstellen, die einer mittleren oder großen Benutzerpopulation gerecht wird, sollten Sie Ihr Netzwerk mit einer oder mehreren Domänenautoritäten einrichten. Eine Domänenautorität verwaltet eine netzwerkweit gültige Kontendatenbank und führt Authentifizierungsdienste für die Hauptnamen in der Domäne durch. Da eine Domäne eine Gruppe mit Benutzern und Computern repräsentiert, wird sie bisweilen auch als *Sicherheitsgrenze* bezeichnet.

Ein Computer, der eine Domänenautorität ausführt, ist ein Domänencontroller. In Windows 2000 verwahren und pflegen die Domänencontroller ihre Kontendatenbanken mit Hilfe von Active Directory. Beachten Sie, dass eine Domäne mit mehreren Domänencontrollern konfiguriert werden kann, um die Performance, Fehlertoleranz und Verfügbarkeit zu verbessern. Wenn eine Domäne über mehr als einen Domänencontroller verfügt, verwendet Active Directory Replikationsmechanismen, um eine Kopie der Kontendatenbank auf jedem Controller zu speichern.

Das Windows-Sicherheitsmodell basiert auf dem Konzept des Vertrauens. Ohne Vertrauen wäre die Authentifizierung bedeutungslos. Benutzer müssen ihrer Domänenautorität und den Administratoren der Domänenautorität vertrauen. Bob vertraut der Domänenautorität zum Beispiel sein Kennwort an und erwartet, dass die Domänenautorität keine anderen Benutzer bestätigt, die seine Identität annehmen möchten.

Auch Computer müssen der Domänenautorität vertrauen. Wenn Sie einer Domäne einen Computer hinzufügen, vertraut die lokale Sicherheitsautorität des Computers der Domänenautorität. Wenn Bob zum Beispiel versucht, auf eine Ressource eines anderen Computers in der Domäne

zuzugreifen, muss die lokale Autorität dieses Computers darauf vertrauen, dass die Domänenautorität Bobs Identität korrekt überprüft. Diese Wechselbeziehung wird häufig als *Vertrauenspfad* bezeichnet.

Domänenautoritäten können auch einander vertrauen. Dies ermöglicht die Authentifizierung von Benutzern einer Domäne, wenn diese versuchen, auf die Ressourcen eines Computers in einer anderen Domäne zuzugreifen. Verglichen mit Windows NT, ist es unter Windows 2000 sehr viel einfacher, einen Vertrauenspfad zwischen verschiedenen Domänen einzurichten. Alle Windows 2000-Domänen derselben Gruppe weisen ein implizit bidirektionales Vertrauen zueinander auf. Unter Windows NT müssen Vertrauensbeziehungen über Domänen hinweg in beide Richtungen manuell konfiguriert werden. Unter Windows 2000 müssen Sie Vertrauensbeziehungen nur dann manuell konfigurieren, wenn Sie einen Vertrauenspfad zwischen Domänen erstellen möchten, die nicht derselben Gruppe angehören.

Benutzer- und Gruppenkonten

Wenn eine Ressource, wie zum Beispiel eine COM+-Applikation, abgesichert werden soll, ist es üblich, Zugriffsberechtigungen für Gruppenkonten und weniger für einzelne Benutzerkonten zu konfigurieren. Wenn Sie für einzelne Benutzerkonten die Ressourcen-Zugriffsberechtigungen konfigurieren, müssen Sie diese fortwährend ändern, wenn neue Benutzer dem Unternehmen beitreten, daraus ausscheiden oder die Rollen innerhalb der Organisation ändern.

Wenn Sie mit Gruppenkonten arbeiten, verringert dies die administrative Last. Eine Gruppe repräsentiert in der Regel mehrere Benutzer, die eine bestimmte Rolle spielen oder eine Organisationseinheit bilden. Wenn Sie Zugriffsberechtigungen für Gruppen konfigurieren, müssen Sie diese nicht oft ändern. Es ist außerdem einfacher, Benutzer aus einer Gruppe zu entfernen oder zu dieser hinzuzufügen.

Sie können Gruppenkonten innerhalb des Gültigkeitsbereichs einer lokalen Sicherheitsautorität oder innerhalb des Gültigkeitsbereichs einer großen Domäne oder Domänengruppe erstellen. Eine lokale Gruppe (die auch als Alias bezeichnet wird) besitzt ein verknüpfte Konto in der lokalen Kontendatenbank und wird nur auf dem lokalen Computer definiert. Eine Domänengruppe ist, verglichen mit einer lokalen Gruppe, eine Gruppe mit einem umfassenderen Gültigkeitsbereich.

Windows 2000 kennt drei Domänengruppentypen. Die Typen unterscheiden sich abhängig davon, ob eine Gruppe über viele Domänen hinweg in einer Domänengruppe oder lediglich innerhalb einer einzelnen Domäne bekannt ist. Ich werde in diesem Kapitel nicht zwischen den verschiedenen Domänengruppentypen unterscheiden.

Es ist wichtig, die Funktion einer lokalen Gruppe zu verstehen. Eine lokale Gruppe existiert nur auf einem einzelnen Computer. Sie benutzen sie, um Berechtigungen für den Zugriff auf Ressourcen des lokalen Computers zu konfigurieren. Sie können lokale Benutzer, Domänenbenutzer und Domänengruppen zur lokalen Gruppe hinzufügen. Sie können lokale Gruppen jedoch nicht ineinander verschachteln.

In Windows 2000 können Sie Domänengruppen ineinander verschachteln. (Beachten Sie, dass hierzu die Domänenautorität im nativen Modus und nicht im gemischten Modus ausgeführt werden muss.) Das Verschachteln von Domänengruppen erlaubt einem Unternehmen, organisatorischen Gruppen bestimmte Rollen zuzuweisen. Stellen Sie sich zum Beispiel vor, dass Ihr Unternehmen eine Gruppe für Verkäufer und eine weitere Gruppe für das Marketing-Personal verwaltet. Sie müssen den Benutzern beider Gruppen äußerst komplexe Zugriffsberechtigungen für dieselben gemeinsam genutzten Verzeichnisse und Dateien gewähren. Sie könnten natürlich

beiden Gruppen die gleichen Zugriffsberechtigungen zuweisen. Es ist jedoch einfacher, eine einzelne rollenspezifische Gruppe zu generieren und dieser die gewünschten Zugriffsberechtigungen zuzuweisen. Dann fügen Sie dieser Gruppe die anderen Gruppen mit den Benutzern hinzu, die die konfigurierten Zugriffsberechtigungen erhalten sollen.

In einer Windows NT-Domäne können Sie Gruppenkonten nicht verschachteln. Sie können jedoch Domänengruppen in lokale Gruppen einbetten. Das von Windows NT verwendete Schema bietet nicht soviel Flexibilität wie das Windows 2000-Schema. Sie können aber ähnliche Ergebnisse in beiden Umgebungen produzieren. Die fehlende Möglichkeit, Domänengruppen unter Windows NT zu verschachteln, bedeutet, dass lokale Gruppen hier nützlicher als unter Windows 2000 sind. Lokale Gruppen arbeiten jedoch sowohl unter Windows 2000 als auch unter Windows NT in derselben Weise.

Vorkonfigurierte Konten

Windows 2000 verfügt über einige vorkonfigurierte Benutzer- und Gruppenkonten, denen automatisch bestimmte gebräuchliche Privilegien und Zugriffsberechtigungen zugewiesen werden. Ich werde kurz die Konten beschreiben, die für die Administration und Sicherheit einer COM+-Applikation besonders wichtig sind.

Vorkonfigurierte Konten werden erstellt, wenn die Kontendatenbank generiert wird. Dies geschieht für eine lokale Sicherheitsautorität, wenn das Betriebssystem installiert wird. Vorkonfigurierte Konten werden für eine Domänenautorität erzeugt, wenn ein Server zu einem Domänencontroller erweitert wird, wodurch eine neue Domäne entsteht.

Jede lokale Sicherheitsautorität besitzt eine lokale *Administratoren*-Gruppe, die standardmäßig über umfassende Privilegien und Zugriffsberechtigungen verfügt. Viele dieser Privilegien und Zugriffsberechtigungen können nicht rekonfiguriert werden. Mitglieder dieser Gruppe können standardmäßig COM+-Applikationen erstellen und konfigurieren. Sie können ebenfalls Applikationen und Dienste auf dem lokalen Computer installieren und Applikationen und Dienste über das Netzwerk installieren und starten.

Jede lokale Sicherheitsautorität verfügt über ein *Administrator*-Konto, das automatisch zur *Administratoren*-Gruppe hinzugefügt wird. Beachten Sie, dass das *Administrator*-Konto nichts weiter als ein Mitglied der *Administratoren*-Gruppe ist. Es besitzt keine explizit zugewiesenen Privilegien oder Berechtigungen. Dieses Konto ist dennoch wichtig, da es Ihnen erlaubt, sich mit administrativen Privilegien und Zugriffsberechtigungen anzumelden, nachdem das Betriebssystem installiert wurde. Nach der anfänglichen Konfiguration des Computers löschen viele Administratoren dieses Konto oder benennen es um, da es häufig für Einbruchsversuche verwendet wird.

Jede Domänenautorität besitzt eine *Domänen-Administratoren*-Gruppe. Das *Administrator*-Konto der Domänenautorität ist automatisch ein Bestandteil dieser Gruppe. Jeder Benutzer in dieser Gruppe kann außerdem andere Benutzer- und Gruppenkonten hinzufügen. Der Zweck dieses Kontos besteht darin, die administrativen Privilegien und Zugriffsberechtigungen auf Administratoren auszuweiten, die dann verantwortlich für die Verwaltung der Domäne sind.

Wenn Sie einen Computer zu einer Domäne hinzufügen, wird die *Domänen-Administratoren*-Gruppe zur lokalen *Administratoren*-Gruppe hinzugefügt. Dies bedeutet, dass ein Mitglied der *Domänen-Administratoren*-Gruppe implizit Administratorprivilegien und -zugriffsberechtigungen für jeden Computer in der Domäne erhält.

Jede Domänenautorität weist eine *Domänen-Benutzer*-Gruppe auf, die standardmäßig alle Domänenbenutzerkonten enthält. Demgemäß verfügt jede lokale Autorität über eine lokale *Benut-*

zer-Gruppe, die standardmäßig alle lokalen Benutzerkonten enthält. Wenn ein Computer zur Domäne hinzugefügt wird, wird auch die *Domänen-Benutzer*-Gruppe zur lokalen *Benutzer*-Gruppe hinzugefügt. Beachten Sie, dass die lokale *Benutzer*-Gruppe für die lokale Anmeldung privilegiert ist. Dies bedeutet, dass sich jeder Benutzer mit einem zulässigen Domänenbenutzerkonto an jedem Computer in der Domäne (mit Ausnahme des Domänencontrollers) anmelden kann.

Jede lokale Sicherheitsautorität und Domänenautorität besitzt darüber hinaus ein *Gast*-Konto. Dieses Konto ist standardmäßig gesperrt. Es gewährt Zugriff für die Hauptnamen, die über kein Konto verfügen, dessen Identität bestätigt werden kann. Stellen Sie sich zum Beispiel vor, Sie möchten ein Verzeichnis gemeinsam mit Benutzern verwenden, die sich in einer Domäne befinden, die keine Vertrauensbeziehung zu Ihrer Domäne aufweist. Oder Sie möchten ein Verzeichnis gemeinsam mit Benutzern verwenden, die sich in keiner Domäne befinden. Da kein Vertrauenspfad zwischen Ihrem Computer und der Autorität dieser Benutzer besteht, benötigen Sie das *Gast*-Konto.

Wenn Benutzer auf eine Ressource Ihres Computers zuzugreifen möchten, versucht Ihre lokale Sicherheitsautorität, die Identität dieser Benutzer zu identifizieren. Wenn die lokale Sicherheitsautorität keine übereinstimmenden Benutzernamen in der Kontendatenbank finden konnte, prüft sie, ob das *Gast*-Konto freigegeben ist. Wenn das Konto gesperrt ist, wird den nicht identifizierten Benutzern der Zugriff verweigert. Wenn das Konto freigegeben ist, weist die lokale Sicherheitsautorität die nicht identifizierten Benutzer dem *Gast*-Konto zu. Indem Sie das *Gast*-Konto freigeben und die entsprechenden Zugriffsberechtigungen konfigurieren, ermöglichen Sie den anonymen Zugriff.

Ein Problem kann jedoch auftreten, wenn Sie den anonymen Zugriff mit Hilfe des *Gast*-Kontos gewähren. Wenn ein Benutzerkonto einer nicht vertrauenswürdigen Autorität denselben Benutzernamen wie ein Konto in Ihrer lokalen Sicherheitsautorität oder Domänenautorität besitzt, prüft das Windows-Sicherheitsmodell, ob beide Konten dasselbe Kennwort aufweisen. Wenn die Kennwörter identisch sind, wird der Benutzer unter der Identität des vertrauenswürdigen Kontos authentifiziert. Wenn die Kennwörter nicht übereinstimmen, wird der Benutzer dem *Gast*-Konto nicht zugewiesen. Ihre lokale Sicherheitsautorität verweigert dem Benutzer dann einfach den Zugriff. Wie Sie sehen, sind übereinstimmende Kontennamen der Grund dafür, dass das *Gast*-Konto keine geeignete Lösung für den anonymen Zugriff ist. Viele Unternehmen und Organisationen besitzen nicht ohne Grund ein Benutzerkonto namens Bob.

Jede Autorität kennt zwei universelle SIDs namens *Jeder* und *Authentifizierte Benutzer*. Diese SIDs gleichen Gruppenkonten in der Hinsicht, dass Sie sie benutzen können, um Zugriffsberechtigungen zu konfigurieren. Sie sind jedoch nicht wirklich Gruppen, da sie keine Konten in einer Kontendatenbank besitzen. Die lokale Sicherheitsautorität verknüpft diese SIDs immer dann mit der Identität eines Hauptnamens, wenn dies erforderlich ist.

Die SID *Jeder* repräsentiert alle Benutzer. Dazu zählen ebenfalls Benutzer, die nicht authentifiziert und somit dem *Gast*-Konto zugewiesen wurden. Die SID *Authentifizierte Benutzer* enthält alle Hauptnamen, die von einer vertrauenswürdigen Autorität identifiziert wurden. Viele Administratoren konfigurieren Sicherheitsberechtigungen auch dann mit der SID *Jeder*, wenn die SID *Authentifizierte Benutzer* eine bessere Wahl wäre. Es ist wichtig, dass Sie die Unterschiede zwischen den beiden SIDs verstehen.

Anmeldungssitzungen und Tokens

Nachdem nun die Grundlagen der Hauptnamen und der Authentifizierung behandelt wurden, werden wir uns darauf konzentrieren, wie die Einhaltung einer Sicherheitsrichtlinie zur Laufzeit durchgesetzt wird. Wir beginnen mit einem Blick auf Anmeldungssitzungen und Tokens.

Lassen Sie mich mit einem einfachen Beispiel beginnen: Stellen Sie sich vor, Bob meldet sich unter einem lokalen Konto an seinem Computer an. Wenn er sein Kennwort eingibt, versucht die lokale Sicherheitsautorität, seine Identität zu authentifizieren. Wenn die Authentifizierung erfolgreich ist, erstellt die Autorität eine interaktive Anmeldungssitzung und ein Token für Bob, wie in Abbildung 11.1 dargestellt.

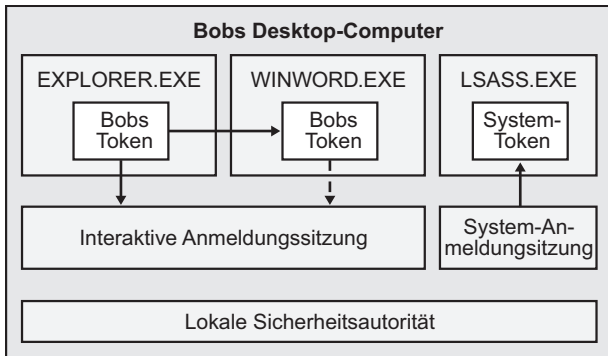


Abbildung 11.1: Jeder Prozess wird mit einem Token erstellt, das den Prozess mit einer eingerichteten Anmeldungssitzung verknüpft. Der Hauptname dieser Anmeldungssitzung dient als Prozessidentität.

Das Windows-Sicherheitsmodell unterscheidet zwischen Anmeldungssitzungen und Tokens. Sie müssen diese Differenzierung jedoch erst dann vornehmen, wenn Sie Systemcode in C++ schreiben. Als Visual Basic-Programmierer sollten Sie sich mehr mit den Tokens beschäftigen, da diese von Windows benutzt werden, um Sicherheitsüberprüfungen zur Laufzeit durchzuführen.

Während Bob viele verschiedene Tokens auf seinem lokalen Computer besitzen kann, verfügt er über nur eine Anmeldungssitzung. Eine Anmeldungssitzung ist gewissermaßen eine Auflistung mit Tokens, die mit einer spezifischen Identität verknüpft sind. Die lokale Autorität speichert die SID für Bobs Benutzerkonto sowie die SIDs für alle lokalen Gruppen und Domänengruppen, in denen Bob mit seinen Tokens Mitglied ist. Die lokale Sicherheitsautorität speichert auch die Privilegien, die Bob gewährt wurden. Wie Sie sehen, agiert ein Token als Symbol für Bob, da es Informationen enthält, mit deren Hilfe das System Autorisierungsüberprüfungen vornimmt, wenn Bob auf eine Ressource zugreifen möchte.

Einer der wichtigsten Vorteile der Verwendung von Tokens in Verbindung mit Anmeldungssitzungen besteht darin, dass ein Prozess die mit einem Hauptnamen verknüpften Authentifizierungsinformationen dynamisch modifizieren kann, wobei dies keine Auswirkungen auf andere Prozesse hat. Ein Prozess kann zum Beispiel zur Laufzeit ein Privileg zu einem Token hinzufügen und aktivieren, ohne Einfluss darauf zu nehmen, wie sich die entsprechende Anmeldungssitzung anderen Prozessen präsentiert. Wenn Prozesse die Anmeldungssitzungen jedoch direkt und nicht über Tokens benutzen, kann ein Prozess die Modifikationen eines anderen sehen.

Wichtig ist auch, dass jeder Prozess mit einem speziellen Token erzeugt wird, das die Identität des Prozesses definiert. Dieses Token wird als Prozess-Token bezeichnet. Ein Prozess wird unter derselben Identität wie der Hauptname ausgeführt, der mit der Anmeldungssitzung des Prozess-Tokens verknüpft ist.

Lassen Sie uns zu dem Beispiel in Abbildung 11.1 zurückkehren. Wenn sich Bob erfolgreich an seinem Computer anmeldet, startet das System eine Instanz der Windows-Shell-Applikation (*EXPLORER.EXE*) und generiert ein Prozess-Token aus Bobs interaktiver Anmeldungssitzung. Wenn Bob eine andere Applikation, wie zum Beispiel Microsoft Word (*WINWORD.EXE*) von der Shell aus startet, kopiert das System das Prozess-Token von einem Prozess in einen anderen. Beide Tokens verweisen jedoch zurück auf Bobs interaktive Anmeldungssitzung. Beide Prozesse werden somit unter Bobs Identität ausgeführt.

Der mit einer interaktiven Anmeldungssitzung verknüpfte Benutzer nimmt die Identität des interaktiven Benutzers an. (Diese Identität ist für die später folgende Diskussion über die Konfiguration der Identität einer COM+-Applikation entscheidend.) Wenn kein Benutzer an einen Computer angemeldet ist, existiert keine interaktive Anmeldungssitzung. Der interaktive Benutzer ist in diesem Fall nicht auf dem lokalen Computer vorhanden.

Was geschieht, wenn sich Bob an seinem Computer abmeldet? Seine interaktive Anmeldungssitzung wird von Windows heruntergefahren. In diesem Fall beendet Windows alle Prozesse, die unter der Identität eines Tokens von Bob ausgeführt werden. Dieses Beispiel sollte ein wenig Licht auf die Beziehung zwischen Anmeldungssitzungen und Tokens geworfen haben. Stellen Sie sich eine Anmeldungssitzung als eine Gruppe mit Tokens vor. Eine Anmeldungssitzung erlaubt dem Betriebssystem, mehrere Prozesse zu verwalten, die unter der Identität desselben Hauptnamens ausgeführt werden.

Es gibt zusätzlich zur interaktiven Anmeldungssitzung mehrere andere Anmeldungssitzungstypen. Dazu zählen System-Anmeldungssitzungen, Netzwerk-Anmeldungssitzungen, Dienst-Anmeldungssitzungen und Batch-Anmeldungssitzungen. Sie sollten wissen, wann diese Typen erstellt und wie sie eingesetzt werden.

Wenn ein Computer hochgefahren wird, erstellt der Startprozess eine System-Anmeldungssitzung. Dies ist eine umfassend privilegierte Anmeldungssitzung, die keine Einschränkungen beachten muss. Viele wichtige Windows-Dienste werden unter der Identität des Systems ausgeführt. Ein Beispiel für solch einen Dienst ist die lokale Sicherheitsautorität (*LSASS.EXE*), wie in Abbildung 11.1 dargestellt. Lassen Sie mich noch einmal erwähnen, dass Sie das Konzept des Vertrauens nicht vergessen dürfen. Jeder Code, der unter der Identität des Systems ausgeführt wird, muss vertrauenswürdig sein. Beachten Sie, dass die System-Anmeldungssitzung vom Computerstart bis zum Herunterfahren des Geräts ausgeführt wird.

Sie gehen vielleicht davon aus, dass die System-Anmeldungssitzung unter Verwendung eines lokalen Kontos in der Kontendatenbank erstellt wird. Dies ist jedoch nicht der Fall. Windows 2000 startet und beendet die System-Anmeldungssitzung und benötigt dazu kein Konto oder Kennwort. Benötigte Windows ein Systemkonto in der Kontendatenbank, würde hierdurch ein weiteres Sicherheitsloch für die »Bösen« geöffnet, durch das die Systemidentität ermittelt und die Sicherheitsrichtlinie umgangen werden könnte.

Die System-Anmeldungssitzung besitzt jedoch kein Konto, so dass sie nicht von anderen Hauptnamen im Netzwerk erkannt werden kann. Beachten Sie, dass die unter der Identität des Systems ausgeführten privilegierten Prozesse die Identität des lokalen Computerkontos benutzen, wenn sie Aufrufe über das Netzwerk durchführen. Dies erlaubt anderen Computern, die Anforderungen von privilegierten Diensten zu authentifizieren.

Lassen Sie uns ein Beispiel betrachten, um diesen letzten Punkt zu klären. Wenn sich Bob mit seinem Domänenkonto an seinem lokalen Computer anmeldet, kann ihn seine lokale Sicherheitsautorität nicht ohne die Hilfe der Domänenautorität authentifizieren. Die lokale Sicherheitsautorität auf Bobs Computer wird mit einem von der System-Anmeldungssitzung erstellten Token ausgeführt (wie in Abbildung 11.2 dargestellt). Wenn die lokale Sicherheitsautorität die

Domänenautorität aufruft, verwendet sie jedoch die Identität des lokalen Computerkontos. Wenn die Domänenautorität somit die Anforderung zur Authentifizierung von Bob empfängt, kann sie feststellen, ob die Anforderung von einem Computer stammt, der ein Mitglied der Domäne ist.

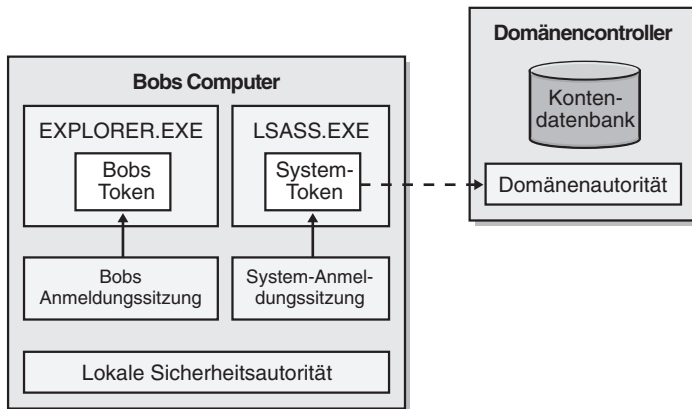


Abbildung 11.2: Wenn ein Prozess, der unter der Identität des Systems ausgeführt wird, einen anderen Computer aufruft, wird dieser Aufruf unter der Identität des lokalen Computerkontos durchgeführt.

Lassen Sie mich zwei Besonderheiten einer Anmeldungssitzung erwähnen. Eine Anmeldungssitzung ist immer mit einem spezifischen Hauptnamen verknüpft und auf einen Computer begrenzt. Was geschieht also, wenn Bob auf eine Netzwerkressource zugreifen möchte? Der Remote-Computer kann kein Token erstellen, um eine Autorisierungsüberprüfung mit Hilfe einer Anmeldungssitzung eines anderen Computers durchzuführen. Die lokale Sicherheitsautorität des Remote-Computers muss eine Netzwerk-Anmeldungssitzung und anschließend ein mit dieser Sitzung verknüpftes Token erzeugen, um zu bestimmen, ob Bob Zugriff auf die angeforderte Ressource erhalten soll.

Abbildung 11.3 zeigt, dass Bob eine interaktive Anmeldungssitzung auf seinem lokalen Computer und eine Netzwerk-Anmeldungssitzung auf einem anderen Computer unterhält. Beachten Sie, dass sich die Tokens, die aus Bobs interaktiver Anmeldungssitzung generiert wurden, in der Regel von den aus Bobs Netzwerk-Anmeldungssitzung erstellten Tokens unterscheiden.

Die Tokens, die aus Bobs Netzwerk-Anmeldungssitzung erstellt werden, gleichen den Tokens, die aus der lokalen Anmeldungssitzung generiert werden, in der Hinsicht, dass sie die SID von Bobs Benutzerkonto und die SID jeder Domänengruppe enthalten, deren Mitglied Bob ist. Die Privilegien und die SIDs lokaler Gruppen sind jedoch immer unterschiedlich. Bob kann zum Beispiel ein Mitglied der lokalen *Administratoren*-Gruppe seines Computers sein. Dies bedeutet jedoch nicht, dass er ein Mitglied der lokalen *Administratoren*-Gruppe eines entfernten Computers ist. Bob ist wahrscheinlich eher ein Mitglied der lokalen *Benutzer*-Gruppe als der lokalen *Administratoren*-Gruppe des entfernten Computers.

Die letzten beiden Anmeldungssitzungstypen, die hier beschrieben werden sollen, sind Dienst-Anmeldungssitzungen und Batch-Anmeldungssitzungen. Beide Typen sind sich sehr ähnlich. Sie werden vorwiegend dazu verwendet, Prozess-Tokens für lokale Prozesse zu erstellen. Sie erlauben Ihnen außerdem, einen Prozess unter einer anderen Identität als der des Systems oder des interaktiven Benutzers auszuführen.

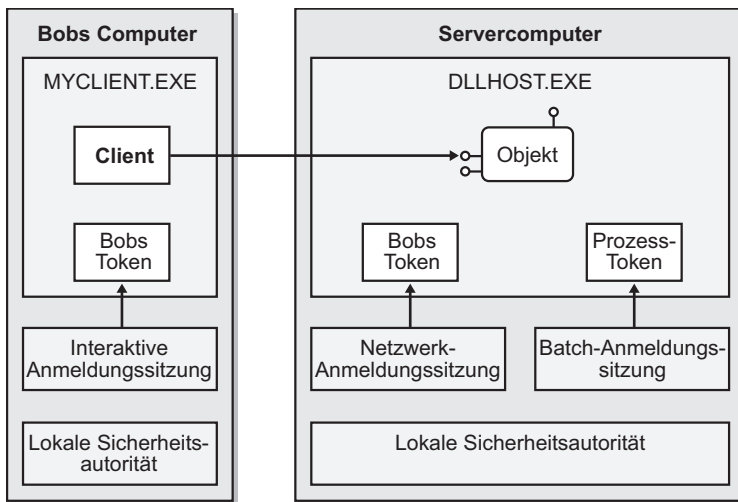


Abbildung 11.3: Ein Benutzer kann über Netzwerk-Anmeldungssitzungen und lokale interaktive Anmeldungssitzungen verfügen.

Wenn Sie einen Dienst auf einem Windows 2000-Computer konfigurieren, können Sie den Prozess des Dienstes unter der Identität des Systems oder eines Benutzerkontos ausführen lassen. Wenn Sie den Dienst unter der Systemidentität ausführen, erhält dieser ein Prozess-Token, das mit der System-Anmeldungssitzung verknüpft ist. Wenn Sie den Dienst mit einem Benutzerkonto ausführen, erstellt das Betriebssystem eine Dienst-Anmeldungssitzung und verwendet diese, um das Prozess-Token für den Prozess des Dienstes zu generieren.

Die meisten Prozesse benötigen nicht alle Berechtigungen und Privilegien der System-Anmeldungssitzung. Windows gestattet Ihnen außerdem nicht, eine Applikation unter der Identität des Systems ausführen zu lassen. Diese Konfigurationsoption steht nur Windows-Diensten zur Verfügung. Es ist jedoch üblich, auch Windows-Dienste mit einem weniger privilegierten Benutzerkonto auszuführen, sofern dies möglich ist. Wenn ein Dienst unter der Identität des Systems ausgeführt wird, obwohl dies nicht notwendig ist, erhalten die Bösen einmal mehr die Möglichkeit, Chaos auf Ihrem Computer anzurichten.

Im Gegensatz zu einem Prozess, der mit einem Windows-Dienst verknüpft ist, kann der Prozess für eine COM+-Applikation nicht unter der Identität der System-Anmeldungssitzung ausgeführt werden. Er muss die Identität eines spezifischen Benutzers annehmen. Sie können die Identität einer COM+-Applikation mit Hilfe der Registerkarte *Identität* im Eigenschaftendialogfeld der Applikation konfigurieren, wie in Abbildung 11.4 dargestellt. Sie können eine Serverapplikation so konfigurieren, dass sie unter der Identität des interaktiven Benutzers oder eines spezifischen Benutzerkontos ausgeführt wird.

Das Ausführen einer COM+-Serverapplikation als interaktiver Benutzer ist während der Entwicklung üblich. Es ist jedoch keine gute Wahl für eine COM+-Applikation, die auf einem Server in der Betriebsumgebung ausgeführt wird. Lassen Sie uns einige Probleme untersuchen, die mit der Ausführung einer Applikation unter der Identität des interaktiven Benutzers verbunden sind.

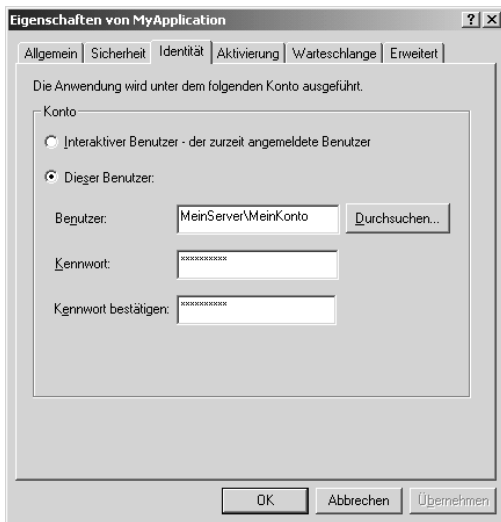


Abbildung 11.4: Sie können auf der Identitäts-Registerkarte des Eigenschaftendialogfeldes einer Serverapplikation bestimmen, welcher Hauptname als Identität der Applikation dienen soll.

Sie möchten den Prozess für eine COM+-Serverapplikation in der Regel unabhängig von dem Benutzer ausführen lassen, der sich an dem Servercomputer angemeldet hat. Der Prozess für eine COM+-Serverapplikation, die mit der Identität des interaktiven Benutzers konfiguriert wurde, kann nur während einer interaktiven Anmeldungssitzung ausgeführt werden. Wenn niemand an dem Servercomputer angemeldet ist, können die Client-Applikationen keine Objekte über das Netzwerk aktivieren. Sie sollten ebenfalls beachten, dass der Prozess für die COM+-Serverapplikation automatisch heruntergefahren wird, wenn sich der aktuelle Benutzer abmeldet. Die Identität der interaktiven Anmeldungssitzung besitzt außerdem abhängig davon, welcher Benutzer angemeldet ist, unterschiedliche Privilegien und Zugriffsberechtigungen.

Eine COM+-Serverapplikation kann somit nicht unter der Identität der System-Anmeldungssitzung oder einer Dienst-Anmeldungssitzung ausgeführt werden. Sie sollten sie auch nicht unter der Identität des interaktiven Benutzers ausführen.

Dies führt uns zur letzten Anmeldungssitzung, der Batch-Anmeldungssitzung. Batch-Anmeldungssitzungen gleichen den Dienst-Anmeldungssitzungen mit der Ausnahme, dass sie für Applikationen und nicht für Dienste verwendet werden. Wenn Sie eine COM+-Serverapplikation für die Ausführung unter der Identität eines Benutzerkontos wie zum Beispiel *MeinServer\MeinBenutzerkonto* konfigurieren, wird die mit der Serverapplikation verknüpfte Instanz von *DLL-HOST.EXE* immer mit einem Prozess-Token gestartet, dass aus einer Batch-Anmeldungssitzung generiert wird. Beachten Sie, dass Sie das Windows NT-Format für den Kontonamen verwenden müssen, wenn Sie die Identität einer COM+-Serverapplikation konfigurieren. COM+ akzeptiert nicht das neuere UPN-Format (*MeinBenutzerkonto@MeinServer*).

Im sechsten Kapitel wurde bereits erwähnt, dass der SCM (Service Control Manager) den Prozess einer COM+-Serverapplikation startet, wenn dies notwendig ist. Der SCM muss außerdem die Batch-Anmeldungssitzung generieren, um das Prozess-Token erstellen zu können. Um jedoch die Batch-Anmeldungssitzung erzeugen zu können, benötigt der SCM das Kennwort des Benutzerkontos.

Wenn Sie die Identität einer COM+-Serverapplikation auf ein spezifisches Benutzerkonto setzen, müssen Sie das Benutzerkennwort zur Verfügung stellen. Der COM+-Katalogmanager schreibt das verschlüsselte Kennwort an eine verborgene Position in den COM+-Katalog. Der SCM kann

das Kennwort von dort abrufen, wenn er es benötigt. Ein weiterer wichtiger Aspekt ist, dass das Benutzerkonto das Privileg benötigt, in einer Stapelverarbeitung angemeldet zu werden, um als Identität einer Batch-Anmeldungssitzung dienen zu können. Wenn Sie die Identität einer COM+-Serverapplikation auf ein spezifisches Benutzerkonto setzen, weist COM+ dem Konto dieses Privileg automatisch zu.

Die letzten Aspekte der Identitätseinstellung einer COM+-Applikation betreffen das Testen und Debuggen von Komponenten. In der Entwicklungsumgebung ist es üblich, eine COM+-Serverapplikation unter der Identität des interaktiven Benutzers ausführen zu lassen. Dies ist einfacher, als ein zugeordnetes Benutzerkonto zu erstellen und zu konfigurieren, das als Identität der Applikation dienen soll. Wenn Sie Ihre Visual Basic-Objekte in *DLLHOST.EXE* ausführen lassen, geschieht dies mit Ihren Privilegien und Zugriffsberechtigungen.

Ein weiterer Aspekt, der für das Ausführen von Code unter der Identität des interaktiven Benutzers während der Entwicklung spricht, besteht darin, dass Ihre Objekte den Benutzern des lokalen Computers Nachrichtendialogfelder anzeigen können. Wenn ein Nachrichtendialogfeld von einem unter der Identität einer Batch-Anmeldungssitzung ausgeführten Objekt aufgerufen wird, wird die Nachricht auf einem virtuellen Bildschirm angezeigt, den kein Benutzer jemals sieht.

Während es während der Entwicklung hilfreich sein kann, ein Nachrichtendialogfeld von einem Objekt in einer COM+-Applikation aufrufen zu lassen, kann dies in einer DLL, die in die Betriebsumgebung übernommen wird, unerwünschte Folgen haben. Denken Sie daran, dass auch die Kompilierung des DLL-Builds für die Betriebsumgebung eine Vorsichtsmaßnahme ist, wenn Sie die Option *Unbeaufsichtigte Ausführung* (die im vierten Kapitel beschrieben wurde) aktiviert haben.

Ressourcen und DACLS

Nachdem nun die Grundlagen der Anmeldungssitzungen und Tokens behandelt wurden, erfahren Sie, wie Windows Autorisierungsüberprüfungen zur Laufzeit durchführt. Viele Ressourcentypen, wie zum Beispiel Verzeichnisse und Dateien auf einer NTFS-Partition, können konfiguriert werden, um bestimmten Benutzern und Gruppen Zugriff zu gewähren oder zu verweigern. Eine Ressource, die abgesichert werden kann, besitzt sicherheitsbezogene Konfigurationsdaten, die als Sicherheitsdeskriptoren bezeichnet werden. Der Sicherheitsdeskriptor verwahrt die SID des Objektbesitzers. Der Besitzer wiederum bestimmt, wer Zugriff auf das Objekt erhält. Ein Sicherheitsdeskriptor kann außerdem eine Sicherheitszugriffskontrollliste (SACL – Security Access Control List) und eine Liste für die wahlweise Zugriffskontrolle (DACL – Discretionary Access Control List) verwahren.

Eine SACL ist eine Liste mit Konfigurationsdaten, die für Überwachungszwecke benötigt werden. Sie können die Überwachung zum Beispiel so einrichten, dass Sie immer wissen, wann ein Benutzer eine bestimmte Datei modifiziert oder löscht. Ich werde nicht weiter auf SACLs eingehen, da COM+ diese Listen nicht verwendet. Sie erfahren später in diesem Kapitel, wie COM+ eine programmgesteuerte Unterstützung für die Überwachung zur Verfügung stellt.

Eine DACL ist eine Liste mit Zugriffsberechtigungen, die dem System mitteilt, welche Tokens in welcher Weise auf eine Ressource zugreifen dürfen. Denken Sie daran, dass eine DACL Bestandteil eines Sicherheitsdeskriptors ist, so dass sie sich auf eine spezifische Ressource, wie zum Beispiel eine Datei oder ein gemeinsam genutztes Verzeichnis, bezieht. Jeder Eintrag in der DACL weist einer spezifischen Berechtigung die SID eines Benutzers oder einer Gruppe zu. Ein Eintrag in einer DACL, die für ein gemeinsam genutztes Verzeichnis verwendet wird, kann zum Beispiel anzeigen, dass Bob eine Lese- und Schreibberechtigung besitzt. Ein anderer Eintrag könnte Aufschluss darüber geben, dass die *Marketing*-Gruppe Leseberechtigungen besitzt.

Das Windows-Subsystem, das DACLs benutzt, um Autorisierungsüberprüfungen durchzuführen, wird als SRM (Security Reference Monitor – Sicherheitsreferenzüberwachung) bezeichnet. Wenn Windows einen Aufruf an den SRM initiiert, um eine Autorisierungsüberprüfung zur Laufzeit durchzuführen, vergleicht der SRM die SIDs eines Tokens mit den SIDs einer DACL, um festzustellen, ob ein Token (und der damit verknüpfte Benutzer) berechtigt ist, auf die Ressource zuzugreifen. Der SRM ermittelt somit, ob der Hauptname die Zugriffsberechtigungen für die beabsichtigten Aktionen besitzt.

Erlauben Sie mir zwei abschließende Bemerkungen über das Autorisieren mit Hilfe von DACLs. Wenn der Sicherheitsdeskriptor für eine Ressource eine DACL besitzt, müssen die Benutzer über explizite Zugriffsberechtigungen verfügen, um auf die Ressource zugreifen zu können. Wenn ein Sicherheitsdeskriptor für eine Ressource darüber hinaus keine eigene DACL besitzt, erhält er eine leere DACL, so dass allen Benutzern implizit alle Zugriffsberechtigungen für die Ressource gewährt werden.

Netzwerk-Authentifizierung und Identitätswechsel

Die lokale Authentifizierung ist sehr einfach zu verstehen. Die Authentifizierung über ein Netzwerk ist jedoch weitaus komplizierter. Der Authentifizierungsmechanismus muss sensitive Informationen über das Netzwerk versenden. Dies muss in einer Weise geschehen, die Benutzer mit nicht legalen Absichten davon abhält, Nachrichten, die von einem Computer zu einem anderen gesendet werden, zu lesen oder unerlaubt zu ändern.

Die Netzwerk-Authentifizierungsdienste wurden entwickelt, um sowohl aktive als auch passive Attacken abzuwehren. Ein Beispiel für eine aktive Attacke ist beispielsweise ein Benutzer, der versucht, Bobs Netzwerk-Anmeldungssitzung zu erhalten, indem er sich als Bob ausgibt und Methodenaufrufe durchführt. Der Authentifizierungsdienst sucht nach Anzeichen für solche unerlaubten Eingriffe und weist Methodenaufrufe ab, wenn diese Eingriffe erfolgen. Netzwerk-Authentifizierungsdienste verhindern ebenfalls passive Attacken, die zum Beispiel von einem Benutzer initiiert werden, der Nachrichten »belauscht«, um sensitive Informationen zu erhalten. Sie können sich gegen Lauschangriffe schützen, indem Sie den Authentifizierungsdienst anweisen, Nachrichten zu verschlüsseln, bevor diese über das Netzwerk versendet werden.

Um die Netzwerk-Authentifizierung nutzen zu können, müssen sich zwei separate Parteien, die auf unterschiedlichen Computern ausgeführt werden, über einen verschlüsselten Sitzungsschlüssel einigen. Dieser Schlüssel wird verwendet, um Nachrichten auf der Versenderseite zu verschlüsseln und auf der Empfängerseite zu entschlüsseln. Wenn zwei Parteien einen Sitzungsschlüssel benutzen, um Nachrichten zu übergeben, kommunizieren sie über einen so genannten sicheren Kanal.

In einem Windows 2000-Netzwerk wird die Authentifizierung und das Generieren von sicheren Kanälen mit Hilfe eines ladbaren Sicherheitsmoduls durchgeführt, das die Bezeichnung SSP (Security Support Provider – Sicherheitsunterstützungs-Provider) trägt. Damit die SSPs kompatibel miteinander sind, verlangt Windows von ihnen, sich nach der Standard-API SSPI (Security Support Provider Interface) zu richten.

Abbildung 11.5 zeigt die Schichten, die zwischen einer COM+-Applikation und dem SSP bestehen, der für die Ausführung der Authentifizierungsdienste verwendet wird. Diese Schichtarchitektur ist nützlich, da Ihre Applikationen immer von dem Code abgesichert sind, der die Authentifizierungsdienste ausführt. Sie müssen außerdem keinen eigenen Code schreiben oder einen SSP erwerben. Sie können einfach einen der SSPs benutzen, die mit dem Betriebssystem ausgeliefert werden.

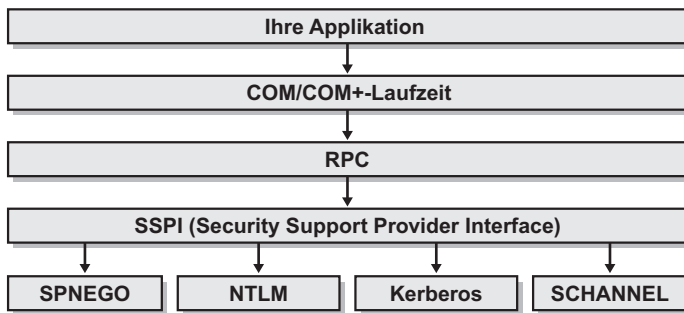


Abbildung 11.5: SSPs sind Plug-In-Module, die Authentifizierungsdienste ausführen.

In den meisten Fällen teilen Sie dem SSP die gewünschte Authentifizierungsebene mit, indem Sie ein deklaratives Applikationsattribut setzen, das Authentifizierungsebene genannt wird. Der Nachteil unterschiedlicher Ebenen besteht darin, dass einige Authentifizierungsdienste sehr viel Verarbeitungszeit benötigen. Das Setzen der Authentifizierungsebene wirkt sich somit auf die Performance aus.

NTLM und Kerberos sind häufig verwendete SSPs, die Authentifizierungsdienste für COM++-Applikationen in einer privaten Netzwerkumgebung zur Verfügung stellen. NTLM ist der Standard-SSP für Windows NT, und Kerberos ist der bevorzugte SSP für Windows 2000. Windows 2000 unterstützt jedoch auch weiterhin NTLM aus Gründen der Abwärtskompatibilität mit Windows NT. SCHANNEL (Secure Channel) ist ein SSP, der häufig mit HTTP verwendet (und später in diesem Kapitel beschrieben) wird.

Der Standard-SSP für Windows 2000 wird SPNEGO (Simple Protected Negotiator) genannt. SPNEGO ist nicht wirklich ein Authentifizierungsdienst-Provider. Es ist ein Dienst, der zwei Computern hilft, den besten SSP auszuwählen, wenn eine authentifizierte Kommunikation erforderlich ist.

Beachten Sie, dass der SSP von beiden Computern unterstützt werden muss, damit ein sicherer Kanal eingerichtet werden kann. Wenn zwei Windows 2000-Computer einen sicheren Kanal einrichten, wird in der Regel Kerberos verwendet. NTLM wird benutzt, wenn ein Windows NT-Computer eine sichere Verbindung zu einem Windows 2000- oder einem anderen Windows NT-Computer herstellt. Die beiden involvierten Computer wählen den SSP vollständig im Hintergrund aus.

NTLM und Kerberos weisen einige Ähnlichkeiten auf. Die Authentifizierungsprotokolle beider SSPs können zum Beispiel einen vertrauenswürdigen Domänencontroller benutzen, um einen sicheren Kanal zwischen zwei Computern einzurichten. Abbildung 11.6 zeigt die Parteien, die daran beteiligt sind, einen sicheren Kanal zwischen einer Client-Applikation und einer COM++-Serverapplikation einzurichten. Nachdem sich beide Computer über den zu benutzenden SSP geeinigt haben, führt der SSP ein Handshake-Protokoll aus, um einen verborgenen Sitzungsschlüssel zu generieren, der von beiden Maschinen verwendet wird. Die ermöglicht das Einrichten eines sicheren Kanals.

Die Systemebendetails der Arbeitsweise von NTLM und Kerberos sind äußerst komplex. NTLM verwendet ein Anforderungs-/Antwortprotokoll, während Kerberos ein Authentifizierungsschema einsetzt, das die Dienste, die auf einem Windows 2000-Domänencontroller ausgeführt werden, dazu anregt, so genannte Tickets zu verteilen. Die Details hierzu werden nicht weiter erläutert. (Keith Browns Buch ist eine exzellente Quelle für die Leser unter Ihnen, die mehr erfahren möchten.) Für Visual Basic-Programmierer sind die Details, die das Einrichten eines sicheren Kanals mit NTLM und Kerberos betreffen, lediglich Implementierungsdetails. Diese Details werden vor Ihnen versteckt.

Während das Wissen um die Implementierungsdetails eines SSPs nicht sehr wichtig ist, sollten Sie beachten, dass NTLM gegenüber Kerberos einige Einschränkungen aufweist. NTLM unterstützt erstens keine gegenseitige Authentifizierung. NTLM kann einem Servercomputer gewiss bestätigen, dass dieser Methodenaufrufe von einem authentifizierten Client erhält. Es kann jedoch nicht garantieren, dass die Client-Methodenaufrufe nur auf dem vorgesehenen Servercomputer ausgeführt werden. Kerberos unterstützt hingegen die gegenseitige Authentifizierung, so dass ein Benutzer mit nicht legalen Absichten Bobs Methodenaufrufe nicht mit Hilfe einiger Routing-Tricks an einen nicht vertrauenswürdigen Server umleiten kann.

Das Ticketübergabeschema von Kerberos ist weitaus skalierbarer als das Anforderungs-/Antwortprotokoll von NTLM. Kerberos weist ein intelligenteres Cache-Schema auf, das zu einer geringeren Zahl von Datenübermittlungen zwischen Domänencontrollern und anderen Computern führt. Kerberos weist somit eine bessere Leistung in einer größeren Netzwerkumgebung auf. Kerberos ist darüber hinaus sehr viel schneller als NTLM, wenn ein sicherer Kanal zwischen zwei Computern eingerichtet wird, die sich in verschiedenen vertrauenswürdigen Domänen befinden.

Die letzte Einschränkung von NTLM besteht darin, dass es nicht delegieren kann. Dies bedeutet, dass NTLM den Identitätswechsel einer verteilten Applikation begrenzt. Sie werden jedoch später in diesem Kapitel von mir erfahren, dass Sie in einer COM+-Applikation auf das Delegieren verzichten sollten. Wenn Sie dann mit mir einer Meinung sind, ist diese Einschränkung von NTLM nicht besonders relevant.

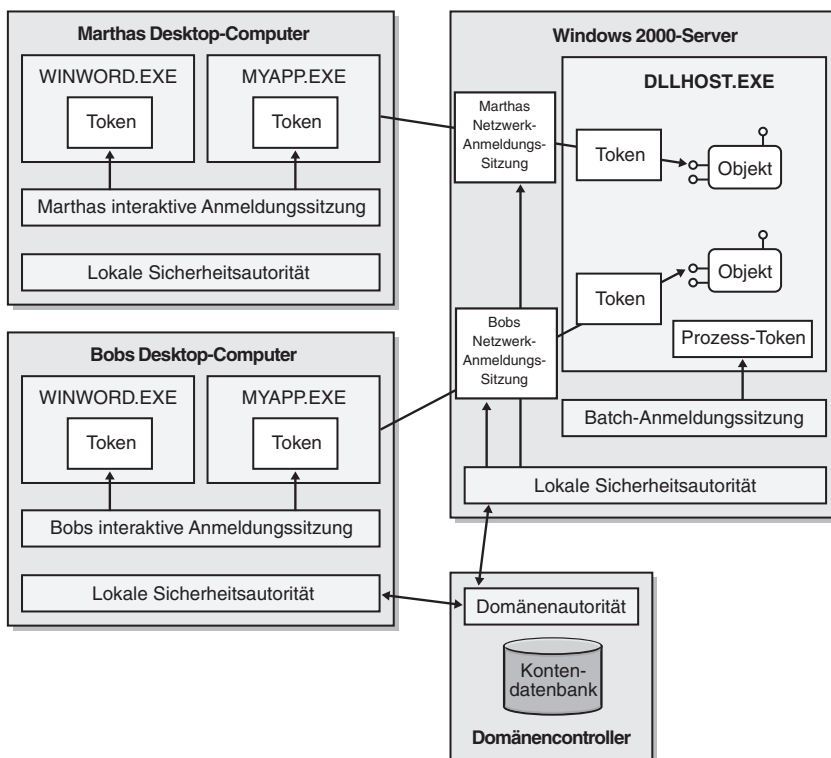


Abbildung 11.6: Die Netzwerk-Authentifizierung mit NTLM oder Kerberos erfordert die Teilnahme eines vertrauenswürdigen Domänencontrollers, damit ein sicherer Kanal zwischen zwei Computern eingerichtet werden kann.

Authentifizierungsebenen setzen

Jede COM+-Serverapplikation besitzt ein `AuthenticationLevel`-Attribut, das Sie setzen können, um dem SSP mitzuteilen, in welchem Maße die Authentifizierungsdienste ausgeführt werden sollen. Jeder Aufruf, der der angeforderten Authentifizierungsebene nicht gerecht wird, schlägt dann fehl. Höhere Ebenen der Authentifizierung bieten einen besseren Schutz gegen unerlaubte Änderungen und Lesevorgänge, wirken sich aber auch auf die Performance aus. Tabelle 11.1 führt die Verantwortlichkeiten des SSPs für jede Authentifizierungsebene auf.

Authentifizierungsebene	Konfigurationswert	Beschreibung	Schutz gegen
<i>(keine)</i>	1	Methodenaufrufe werden nicht authentifiziert. Diese Einstellung deaktiviert die Sicherheit.	
<i>Verbinden</i>	2	Die Authentifizierung wird nur dann durchgeführt, wenn eine Verbindung besteht.	Nicht gestattete Änderungen
<i>Aufruf</i>	3	Die Authentifizierung wird nur für das erste RPC-Paket jedes Methodenaufrufs garantiert. (Diese Ebene wird nicht unterstützt. Stattdessen wird immer die Paketebene verwendet.)	Nicht gestattete Änderungen
<i>Paket</i>	4	Die Authentifizierung wird für jeden Paket-Header in jedem Methodenauf-ruf garantiert.	Nicht gestattete Änderungen
<i>Paketintegrität</i>	5	Wie Authentifizierungsebene <i>Paket</i> . Zusätzlich wird eine verschlüsselte Paketdatenprüfsumme benutzt, mit deren Hilfe unerlaubte Änderungen ermittelt werden.	Nicht gestattete Änderungen
<i>Paketdatensicherheit</i>	6	Wie Authentifizierungsebene <i>Paketintegrität</i> . Zusätzlich wird eine Prüfsumme für die Datensicherheit verwendet.	Nicht gestattete Änderungen und Leseversuche

Tabelle 11.1: SSP-Authentifizierungsebenen

Sie können die Authentifizierungsebene einer COM+-Serverapplikation mit dem administrativen Hilfsmittel *Komponentendienste* auf der Registerkarte *Sicherheit* des Applikations-Eigenschaftendialogfeldes einstellen, wie in Abbildung 11.7 dargestellt. Die Authentifizierungsebene wird für die gesamte COM+-Serverapplikation berücksichtigt, wenn diese gestartet wird. Die COM+-Laufzeit setzt die Authentifizierungsebene während der Applikationsinitialisierung, indem sie den konfigurierten Wert aus dem COM+-Katalog liest. Dies geschieht im Hintergrund.

Das Konfigurieren der Authentifizierungsebene einer COM+-Serverapplikation genügt jedoch nicht. COM+ (oder COM auf einem Computer mit einer Vorgängerversion von Windows 2000) untersucht auch die Authentifizierungsebene der Client-Applikation. Sie sollten wissen, dass die prozessspezifische Authentifizierungsebene auf beiden Seiten den Mindestwert repräsentiert. Wenn der Client-Prozess und der Serverprozess unterschiedliche Werte aufweisen, verwendet COM die höhere Marke.

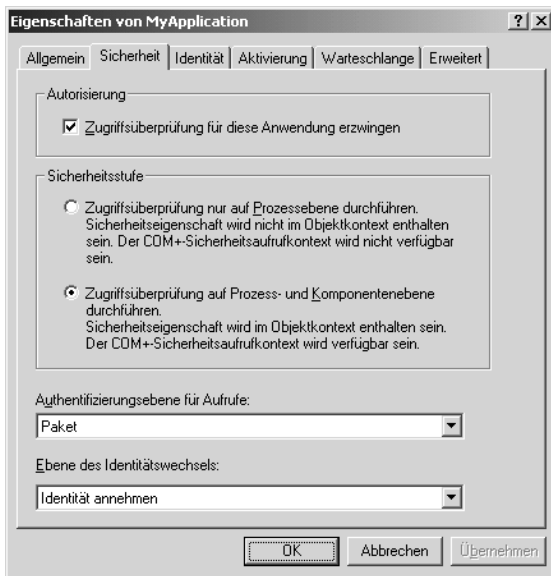


Abbildung 11.7: Sie können auf der Registerkarte Sicherheit die rollenbasierte Authentifizierungsüberprüfung aktivieren und die Applikationssicherheit, Authentifizierung sowie die Identitätswechselebenen konfigurieren.

Wenn Sie die Authentifizierungsebene der Serverapplikation konfigurieren, können Sie davon ausgehen, dass die Authentifizierung auf dieser oder auf einer höheren Ebene ausgeführt wird. Wenn Sie die Authentifizierung deaktivieren oder auf einer niedrigeren Ebene ausführen lassen möchten, müssen Sie sicherstellen, dass die Client-Applikation keine höhere Authentifizierungsebene anfordert.

Wenn Sie eine Visual Basic-Client-Applikation installieren, die auf einem Standard-EXE-Projekt basiert, stehen Ihnen zwei Möglichkeiten zur Verfügung, um die Authentifizierungsebene der Applikation zu konfigurieren: eine einfache, aber nicht korrekte Vorgehensweise oder eine schwierige, aber bessere Methode. Die einfache und nicht korrekte Vorgehensweise besteht darin, die maschinenweit gültige Standard-Authentifizierungsebene zu setzen. Dies ist mit einem administrativen Hilfsmittel namens *DCOMCNFG.EXE* möglich. Sie öffnen auf diese Weise jedoch ein Sicherheitsloch auf dem Client-Computer. Dies gilt besonders dann, wenn Sie die maschinenweit gültige Authentifizierungsebene auf *(keine)* setzen.

Die schwierige, aber bessere Methode, die Authentifizierungsebene einer Client-Applikation zu konfigurieren, besteht darin, benutzerdefinierte *AppID*-Schlüssel zur Windows-Registrierung hinzuzufügen. Diese Technik kann für Client-Applikationen verwendet werden, die unter Windows 2000 oder Windows NT ausgeführt werden. Abbildung 11.8 zeigt die Schlüssel und die benannten Werte, die Sie zur Registrierung hinzufügen müssen. Beachten Sie, dass Sie den Namen der ausführbaren Client-Applikationsdatei einer benutzerdefinierten *AppID* zuordnen und den *AppID*-Schlüssel mit einem benannten Wert konfigurieren müssen, der die konfigurierte Authentifizierungsebene aufnimmt.

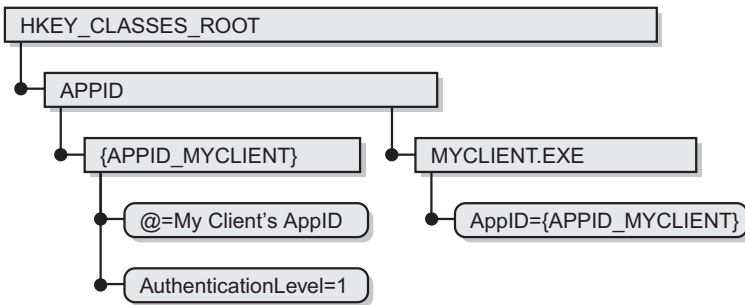


Abbildung 11.8: Diese AppID-Einstellungen sind erforderlich, wenn Sie die *AuthenticationLevel*-Einstellung für eine clientseitige Applikation konfigurieren, die auf einem Standard-EXE-Projekt basiert. Wenn keine AppID vorhanden ist, benutzt die Applikation die maschinenweit gültige *AuthenticationLevel*-Einstellung.

Sie sollten einige Aspekte berücksichtigen, die das Hinzufügen dieser Einträge zur Windows-Registrierung auf dem Client-Computer betreffen. Der Schlüssel, der die GUID für die AppID verwahrt, muss dem *AuthenticationLevel*-Attribut einen Wert vom Typ *DWORD* (nicht *String*) zuweisen. Der Wert muss außerdem mit einem der in Tabelle 11.1 aufgeführten Konfigurationswerte übereinstimmen. Nachdem Sie den *AppID*-Schlüssel zur Registrierung hinzugefügt haben, können Sie dessen Authentifizierungsebene mit *DCOMCNFG.EXE* einstellen.

Ich habe bisher nur beschrieben, wie die Authentifizierungsebene deklarativ eingerichtet wird. In einigen Fällen kann es jedoch vorteilhaft oder notwendig sein, die Authentifizierungsebene programmgesteuert zu erhöhen oder zu verringern. Sie müssen in solchen Situationen Code in einer Systemsprache wie zum Beispiel C++ schreiben, der direkt mit der COM-Sicherheits-API kommuniziert. Die COM-Bibliothek bietet Funktionen wie zum Beispiel *CoInitializeSecurity* und Schnittstellen an, die Ihnen gestatten, die Authentifizierungsebene und andere Sicherheitseinstellungen für einzelne Prozesse zu konfigurieren. Diese Funktionen erlauben Ihnen außerdem, feiner strukturierte Änderungen vorzunehmen. Sie können zum Beispiel die sicherheitsbezogenen Einstellungen für einen Kanal ändern, der mit einem Proxy-/Stub-Paar verbunden ist. Sie verfügen somit über die Möglichkeit, die Authentifizierungsebene mit Hilfe einzelner Aufrufe zu modifizieren.

Einige Unternehmen haben eine benutzerdefinierte DLL mit C++ erstellt, um Visual Basic-Komponenten zu ergänzen und eine zusätzliche programmgesteuerte Kontrolle zu erhalten. Wenn Sie Ihren gesamten Code jedoch mit Visual Basic generieren möchten, sollten Sie voraussetzen, dass die Authentifizierungsebene ein statischer, prozessweit gültiger Wert ist.

Lassen Sie mich diesen Abschnitt mit einer letzten Bemerkung zu den Authentifizierungsebenen beenden. Denken Sie immer daran, dass die mit COM durchgeführte Remote-Kommunikation immer auf ein privates Netzwerk begrenzt sein sollte. Die mit COM-Methodenaufrufen verknüpften Nachrichten in diesem Netzwerk sind in der Regel weniger schutzlos als in einem öffentlichen Netzwerk, wie zum Beispiel dem Internet. In einer kontrollierten Umgebung ist es nicht immer notwendig, die Authentifizierungsebene auf eine der höheren Werte (wie zum Beispiel *Paketdatensicherheit*) zu setzen. Jedes Unternehmen muss jedoch eine eigene Entscheidung treffen. Wenn Sie die Authentifizierungsdienste auf einer unteren Ebene ausführen lassen, erhalten Sie eine bessere Performance.

Identitätswechselebene einstellen

Eine Remote-Verbindung zwischen einem Client und einem Objekt weist nicht nur eine Authentifizierungsebene, sondern ebenfalls eine Identitätswechselebene auf. Die Authentifizierungsebene dient der Datensicherheit im Netzwerk und erlaubt dem Servercomputer, die Identität des Aufrufers zu ermitteln. Die Identitätswechselebene schützt den Client. Ein Client kann mit Hilfe der Authentifizierung seine Sicherheitsinformationen über das Netzwerk übermitteln. Die Identitätswechselebene ermittelt, in welcher Weise der Serverprozess das Client-Token benutzen kann. Höhere Identitätswechselebenen bieten dem Server eine effektivere Nutzung der Client-Sicherheitsinformationen. Tabelle 11.2 führt die Optionen zum Konfigurieren der Identitätswechselebene auf.

Identitätswechselebene	Beschreibung
<i>Anonym</i>	Der Server kann die Client-Informationen nicht einsehen. Diese Ebene wird im Netzwerk nicht unterstützt. Unter Windows 2000 und Windows NT wird stattdessen die Identitätswechselebene <i>Identifizieren</i> verwendet. Die Ebene wird lokal unterstützt, führt aber dazu, dass alle Methodenaufrufe fehlschlagen. Sie sollten diese Einstellung nicht verwenden.
<i>Identifizieren</i>	Der Server kann die Sicherheitsinformationen einsehen, aber lediglich für die Identifizierung und Autorisierung benutzen.
<i>Identität annehmen</i>	Der Server kann die Sicherheitsinformationen des Clients benutzen, um auf lokale Ressourcen zuzugreifen.
<i>Delegieren</i>	Der Server kann die Sicherheitsinformationen des Clients benutzen, um auf lokale und entfernte Ressourcen zuzugreifen. (Diese Option ist über NTLM nicht verfügbar.)

Tabelle 11.2: Identitätswechselebene

Beachten Sie, dass die Identitätswechselebene im Gegensatz zur Authentifizierungsebene vollständig vom Client bestimmt wird. Wenn Sie die Identitätswechselebene in einer COM+-Serverapplikation konfigurieren, führt die Einstellung nur dann zu einem Ergebnis, wenn die Objekte der Applikation als Clients agieren, die die Methoden anderer Objekte in anderen Prozessen aufrufen.

Visual Basic-Programmierer können die Identitätswechselebene nicht einfach zur Laufzeit ändern. Wenn Sie eine Visual Basic-Client-Applikation ausführen, gilt für die Identitätswechselebene der maschinenweit gültige Standardwert. Sie können diesen Wert mit *DCOMCNFG.EXE* modifizieren.

Lassen Sie uns einen Schritt zurückgehen und die Philosophie des Identitätswechsels untersuchen. Der Identitätswechsel ist ein Mechanismus, mit dessen Hilfe eine Client-Applikation die eigenen Sicherheitsinformationen an andere Prozesse delegieren kann, so dass diese Prozesse eine Aufgabe unter der Identität des Clients ausführen können. Auf der physikalischen Ebene muss der Identitätswechsel einige wichtige Anforderungen erfüllen. Die Serverapplikation muss die Ausführung einer Client-Anforderung unter der Identität des Prozess-Tokens abrechnen und die Identität des Client-Tokens annehmen. Der Thread, der die Anforderung bedient, kann unter der Identität des Clients lediglich solche Aufgaben durchführen, für die der Client autorisiert ist. Wenn außerdem eine der angeforderten Operationen vom System überwacht wird, erscheint der Benutzername des Clients im Windows-Ereignisprotokoll.

In einigen Situationen wird der Identitätswechsel problemlos durchgeführt. Dies gilt zum Beispiel, wenn Sie Zugriff auf einen Dateiserver benötigen. Wenn Bob über das Netzwerk auf ein gemeinsam genutztes Verzeichnis zugreift und versucht, eine Datei zu löschen, wird er dabei von ei-

nem Weiterleitungsdienst unterstützt, der unter der Identität des Systems ausgeführt wird. Bobs Anforderung wird jedoch nicht auf einem Thread unter der Identität des Systems bearbeitet. Der Weiterleitungsdienst verwendet stattdessen den Identitätswechsel, bevor er versucht, die Datei zu löschen, um den Thread auf ein Token zu schalten, das von Bobs Netzwerk-Anmeldungssitzung generiert wurde. Dieses Schema funktioniert, da Bob Zugriffsrechte besitzen muss, um die Datei zu löschen. Dies bedeutet aber auch, dass Bob während dieser Zeit überwacht werden kann.

Ein Webserver ist ein weiterer Dienst, der den Identitätswechsel benutzt. IIS wird zum Beispiel als Dienst unter der Identität des Systems ausgeführt. Es wäre jedoch nicht sehr sicher, wenn IIS jede Client-Anforderung unter dieser Identität bearbeiten würde. Der Dienst verwendet deshalb den Identitätswechsel, um Arbeits-Threads eine weniger privilegierte Identität zuzuweisen. (Dieses Thema wird später in diesem Kapitel ausführlich behandelt.)

Wenn mehrschichtige Applikationen mit COM+ erstellt werden, funktioniert der Identitätswechsel in der Theorie besser als in der Praxis. Der erste Grund, auf den Identitätswechsel zu verzichten, besteht in diesem Fall darin, dass er schwierig einzurichten und zu verwenden ist. Dies gilt besonders dann, wenn ein Objekt, das in einer COM+-Serverapplikation ausgeführt wird, die Identität des Clients annehmen und ein Aufruf durchgeführt werden soll, der den Host verlässt.

Stellen Sie sich zum Beispiel vor, dass Bob eine Client-Applikation ausführt und ein Objekt aus einer COM+-Serverapplikation, die über das Netzwerk ausgeführt wird, aktivieren möchte. Bobs interaktive Anmeldungssitzung speichert eine verschlüsselte Kopie seines Kennworts auf seinem Computer. Wenn Bob ein Objekt aktiviert, authentifiziert ihn der Servercomputer, um eine Netzwerk-Anmeldungssitzung zu erzeugen. Der lokale Computer muss eine zwischengespeicherte Kopie von Bobs Kennwort während der Authentifizierung zur Verfügung stellen. Das Kennwort in Bobs interaktiver Anmeldungssitzung präsentiert somit gewissermaßen dessen Netzwerkinformationen.

Damit der Servercomputer einen dritten Computer aufrufen und Bobs Identität annehmen kann, muss auch er eine verschlüsselte Kopie von Bobs Kennwort speichern. Dieser Identitätswechseltyp, der sich über mehrere Netzwerkstationen erstreckt, wird *Delegation* genannt. Sie können nur mit Kerberos delegieren. NTLM unterstützt dies nicht. Die Identitätswechselebene der Client-Applikation muss dazu auf *Delegieren* gesetzt sein. Das Benutzerkonto des Clients und das Benutzerkonto, das als Identität der Serverapplikation dient, müssen darüber hinaus mit speziellen Eigenschaftseinstellungen in Active Directory konfiguriert werden. Die *Konto kann nicht delegiert werden*-Eigenschaft des Client-Benutzerkontos muss deaktiviert sein. Die *Konto wird für Delegierungszwecke vertraut*-Eigenschaft des Server-Applikationskontos muss aktiviert sein.

Wenn Sie die administrativen Aufgaben gelöst haben, müssen Sie den Identitätswechsel programmgesteuert für jede einzelne Anforderung durchführen. Dies geschieht in der Methodenimplementierung Ihrer Komponenten. Es gibt verschiedene programmgesteuerte Techniken, um einen Thread vom Prozess-Token auf das Client-Token zu schalten. Diese Techniken können jedoch einfacher mit C++ als mit Visual Basic verwendet werden.

Sie haben nun einige technische Gründe für einen Verzicht auf den Identitätswechsel und das Delegieren kennen gelernt. Es gibt jedoch auch einige Design-Gründe. Wenn Sie den Identitätswechsel in einer COM+-Applikation benutzen, delegieren Sie die Verantwortung lediglich an den Administrator, der die Datenschicht verwaltet. Anstatt dafür zu sorgen, dass die Sicherheitsrichtlinie eingehalten wird, überlassen Sie alles der Person, die die Datenschichtressourcen, wie zum Beispiel Datenbanken und gemeinsam genutzte Verzeichnisse, verwaltet. Diese Sicherheitsphilosophie ist mit einer verteilten Applikation einfach nicht vereinbar.

Der Identitätswechsel zwingt den Datenschichtadministrator, den Benutzerkonten und/oder Gruppenkonten einer Organisation explizit Zugriffsberechtigungen zu gewähren oder zu verwei-

gern. Wenn der Administrator eine neue Gruppe erstellt, muss er möglicherweise die Zugriffsberechtigungen in mehreren Datenbankservern und Dateiservern rekonfigurieren. Das Verwalten einer Sicherheitsrichtlinie wird mit einer zunehmenden Zahl von Datenschichtservern schwieriger. Sie können außerdem nicht immer die Absichten eines Benutzers kennen, wenn Sie die Zugriffsberechtigungen für eine Ressource konfigurieren.

Wenn Sie einer Gruppe zum Beispiel Lese- und Schreibberechtigungen für eine Datei gewähren, haben Sie keinen Einfluss darauf, welche Bereiche der Datei modifiziert werden können. Sie können nicht zwischen Benutzern unterscheiden, die geringfügige Änderungen vornehmen oder Benutzern, die den gesamten Dateiinhalt löschen möchten.

Wenn Sie eine verteilte Applikation entwickeln, können Sie die administrative Last verringern, indem Sie alle Autorisierungsüberprüfungen dann durchführen lassen, wenn die Client-Anforderungen die mittlere Schicht erreichen. Die meisten Unternehmen, die verteilte Applikationen erfolgreich für eine große Benutzerzahl einsetzen, wissen, dass dies die einzig vernünftige Lösung ist.

Es kann außerdem sehr viel effektiver sein, Zugriffsberechtigungen für Komponenten und Methoden anstatt für Ressourcen wie zum Beispiel Datei- und Datenbanktabellen zu konfigurieren. Sowohl Komponenten als auch Methoden weisen eine verwandte Semantik auf, die das Erkennen der Benutzerabsichten vereinfacht. Eine Methode könnte zum Beispiel eine geringfügige Änderung an einer Datei vornehmen, während eine andere Methode den gesamten Dateiinhalt löscht. Sie entscheiden, welche Methoden von den einzelnen Benutzern ausgeführt werden dürfen. Sie erhalten auf diese Weise eine äußerst effektive Sicherheitsrichtlinie.

Aus einer Performance-Perspektive betrachtet, ist es besser, einen Methodenaufruf in einer COM+-Applikation sofort abzuweisen, als eine Anforderung weiterzuleiten und von einem Datei- oder Datenbankserver abweisen zu lassen. Ein Aufruf sollte frühstmöglich scheitern. In größeren Applikationen sind die Verarbeitungszyklen des Datenbankservers darüber hinaus häufig auch die wertvollsten Zyklen. Ist dies der Fall, sollten Sie Verarbeitungszyklen nicht verschwenden, indem Sie Autorisierungsüberprüfungen auf dem Datenbankserver ausführen lassen, obwohl diese in die mittlere Schicht verlagert werden könnten.

Ein weiterer Aspekt, der die Performance betrifft, besteht darin, dass der Identitätswechsel und das Delegieren die Möglichkeit einschränken, das Datenbankverbindungs-Pooling durchzuführen, da jede Verbindung in einem Pool denselben Benutzer und dasselbe Kennwort besitzen muss. Aus der Perspektive des DBMS betrachtet, besitzt jede Verbindung im Pool dieselbe Benutzeridentität.

Lassen Sie uns zusammenfassen: Sie sollten Autorisierungsüberprüfungen auf der mittleren Schicht möglichst nahe am Client durchführen. Dies verlangt besonders dann ein Umdenken von vielen Unternehmen, wenn diese intensiv mit zweischichtigen Applikationen gearbeitet haben. Ich empfehle Ihnen, eine Sicherheitsrichtlinie basierend auf der Idee zu entwerfen, dass die Datenschicht den Benutzerkonten vertraut, die als Identitäten für die Applikationen der mittleren Schicht dienen. Diese Middle-Tier-Benutzerkonten sind dafür vorgesehen, Objekte in COM+-Applikationen ausführen zu lassen. Sie unterscheiden sich darin von anderen Benutzerkonten, dass sie keiner Person zugewiesen sind. Datenbankadministratoren wissen somit nicht, wer auf ihre Datenbank zugreift.

Denken Sie daran, dass der Datenbankadministrator diesen Middle-Tier-Benutzerkonten nur typische Benutzerberechtigungen gewähren muss. Ein Benutzerkonto der mittleren Schicht benötigt in der Regel Lese- und Schreibberechtigungen für Tabellen und Ausführungsberechtigungen für gespeicherte Prozeduren. Es ist aber nicht erforderlich, den Middle-Tier-Konten administrative Berechtigungen für Aktivitäten wie zum Beispiel das Erstellen und Löschen von Tabellen zu gewähren.

Datenbankadministratoren verlieren auf diese Weise einen Teil der Kontrolle, die sie früher besaßen. Sie sind außerdem nicht mehr dafür verantwortlich, Zugriffsberechtigungen für die Benutzer- und Gruppenkonten des Unternehmens zu konfigurieren. Sie können einzelne Benutzer nicht mehr mit Hilfe der verschiedenen Kontrollmechanismen überwachen, die von den meisten kommerziellen DBM-Systemen zur Verfügung gestellt werden. Während einige Personen (besonders die Datenbankadministratoren) diesen Kontrollverlust nicht begrüßen, ist dieser notwendig, wenn Sie eine umfassende dreischichtige Applikation installieren.

Sicherheit in COM+

Nachdem nun die Besonderheiten der Windows-Sicherheit erörtert wurden, können wir uns den spezifischen COM+-Sicherheits-Features zuwenden. Die COM+-Sicherheit kann deklarativ und programmgesteuert sein. Wie Sie sehen werden, basieren beide Formen auf einer wichtigen Abstraktion, die *Rolle* genannt wird.

Die deklarative Sicherheit überträgt die Verantwortung für die Einhaltung der Sicherheitsrichtlinien auf den Systemadministrator. Wenn eine Applikation installiert wird, entscheidet ein Administrator, wer auf die Applikation zugreifen darf. Wenn sich die Sicherheitsanforderungen für die Applikation im Laufe der Zeit ändern, kann der Administrator darauf reagieren, ohne sicherheitsbezogenen Applikationscode um- oder neu schreiben zu müssen. Dies führt ebenfalls dazu, dass die entsprechenden DLLs nicht so oft erneut kompiliert und verteilt werden müssen. Sie können die COM+-Sicherheit, wie auch die meisten anderen Aspekte der COM+-Administration, mit Hilfe des administrativen Hilfsmittels *Komponentendienste* oder über das COM+-Admin-Objektmodell konfigurieren.

In COM+, können Sie die deklarativen Autorisierungsüberprüfungen mit der programmgesteuerten Sicherheit ergänzen. In bestimmten Situationen können programmgesteuerte Autorisierungsüberprüfungen eine bessere Kontrolle bieten, da Sie über die Möglichkeit verfügen, eine bedingte Logik zu verwenden. Sie sollten ebenfalls beachten, dass die Windows-Sicherheit viele deklarative Überwachungs-Features zur Verfügung stellt, die jedoch nicht von COM+ unterstützt werden. Wenn Sie möchten, dass Ihre Applikation die Aktionen der Benutzer überwacht, müssen Sie auf die programmgesteuerte Sicherheit ausweichen. Die gute Neuigkeit lautet jedoch, dass das Schreiben von Code für die benutzerdefinierte Überwachung sehr einfach ist.

Rollen und Autorisierung

Eine zentrale Figur im COM+-Sicherheitsmodell ist die Rolle. Lassen Sie uns untersuchen, warum das COM+-Team entschied, die gesamte Autorisierung mit Hilfe von Rollen durchzuführen. In diesem Kapitel wurde bereits beschrieben, wie Gruppen in andere Gruppen eingebettet werden. Die übliche Motivation hierfür besteht darin, einem einzelnen Sicherheitsprofil mehrere Organisationseinheiten zuzuordnen. Sie könnten zum Beispiel entscheiden, die Verkäufergruppe und die Marketing-Gruppe in der *DatabaseReader*-Gruppe anzuordnen. Wenn Sie anschließend einige Zugriffsberechtigungen für die *DatabaseReader*-Gruppe konfigurieren, werden diese Berechtigungen implizit für die Verkäufer- und Marketing-Gruppe übernommen.

Mehrere Ebenen der Gruppierung können die Sicherheitsadministration in großen Organisationen vereinfachen, da sie die Anzahl der Konten verringern, für die Sie Zugriffsberechtigungen konfigurieren müssen. Zugriffsberechtigungen müssen außerdem nicht erneut konfiguriert werden, wenn Benutzer und Gruppen hinzugefügt oder entfernt werden. Das rollenbasierte Modell

der COM+-Sicherheit basiert auf diesen Prinzipien. Anstatt jedoch Gruppen innerhalb anderer Gruppen anzuordnen, betten Sie Gruppen in Rollen ein.

COM+-Rollen unterscheiden sich vollständig von Gruppen. Sie werden innerhalb des Gültigkeitsbereichs einer COM+-Applikation definiert. Gruppen hingegen werden in einer Kontendatenbank definiert und sind daher mit einer spezifischen Autorität verknüpft. Rollen werden mit Zeichenfolgen identifiziert, Gruppen mit SIDs. Das COM+-Sicherheitsmodell trennt Rollen von SIDs, um eine größere Flexibilität zu bieten.

Sie definieren Rollen zur Entwurfszeit. Eine Rolle ist eine Gruppe mit Benutzern in einer COM+-Applikation, die bestimmte Zugriffsberechtigungen gemeinsam nutzen. Jede Rolle besitzt einen beschreibenden Namen und definiert ein Sicherheitsprofil für einen bestimmten Benutzertyp. Sie könnten zum Beispiel eine Applikation entwickeln, die drei Rollen enthält: Reader, Writer und Manager. Die von Ihnen (deklarativ oder programmgesteuert) eingerichteten Autorisierungsüberprüfungen werden immer für Rollen durchgeführt. Rollen sind eine einfach zu benutzende Abstraktion für Designer und Programmierer. Rollen verhindern, dass Sie sich mit Konten (und deren SIDs) während der Entwicklung beschäftigen müssen.

Ein Administrator weist den Rollen in Ihrer Applikation bestimmte Benutzer und Gruppen zur Entwurfszeit zu, wie in Abbildung 11.9 dargestellt. Beachten Sie, dass ein Benutzer oder eine Gruppe zu mehreren Rollen hinzugefügt werden kann. Jede Rolle kann somit viele verschiedene Benutzer als Mitglieder aufweisen. Diese Viele-zu-Viele-Beziehung zwischen Benutzern und Rollen führt zu einer außerordentlichen Flexibilität.

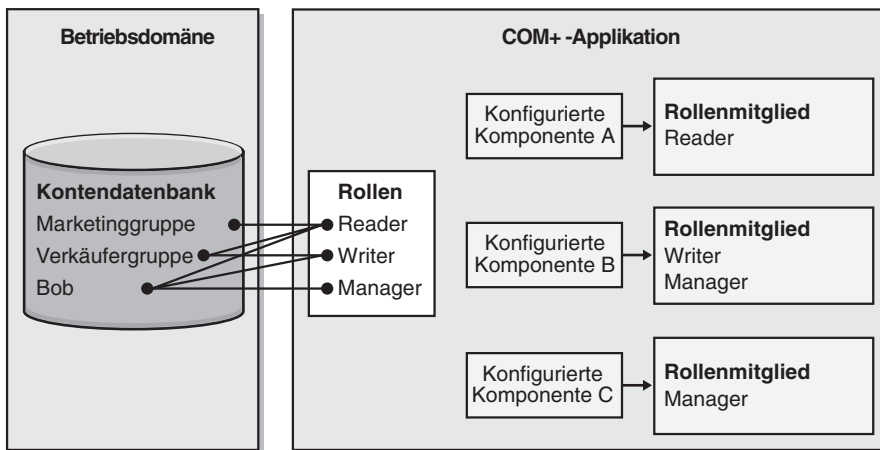


Abbildung 11.9: Sie verwenden Rollen während der Entwicklung, um Sicherheitsprofile zu definieren und Autorisierungsüberprüfungen zu konfigurieren. Nach der Installation ordnet ein Administrator den in einer COM+-Applikation definierten Rollen Benutzer und Gruppen zu.

Ein großer Vorteil des rollenbasierten Modells besteht darin, dass Sie Zugriffsberechtigungen für die gesamte Applikation in der Entwicklungsumgebung vorkonfigurieren können. Zugriffsberechtigungen können ausschließlich Rollen zugewiesen werden, und Rollen sind innerhalb des Gültigkeitsbereichs einer COM+-Applikation definiert. Rollenbasierte Zugriffsberechtigungen weisen somit keine Abhängigkeiten von SIDs auf. Dies ist sehr wichtig. Die Rollen im COM+-Sicherheitsmodell vereinfachen den Einsatz einer einzelnen COM+-Applikation in mehreren Betriebsumgebungen.

Wenn Sie eine COM+-Applikation mit einer vorkonfigurierten Sicherheitsrichtlinie distribuieren, sollten Sie eine Dokumentation hinzufügen, die die Semantik jeder Rolle beschreibt. Administratoren benötigen diese Informationen, um den von Ihnen definierten Rollen Benutzer und Gruppen hinzuzufügen.

Wenn Sie eine gesicherte Applikation entwickeln, die in vielen verschiedenen Betriebsumgebungen eingesetzt werden soll, sollten Sie darüber nachdenken, die Anzahl der Rollen zu begrenzen und die Komplexität der Rollensemantik zu minimieren. Stellen Sie sich eine Applikation vor, die 27 Rollen definiert und vier oder fünf Seiten Dokumentation enthält, um jede Rolle zu beschreiben. Wie viele Administratoren werden 100 Seiten Dokumentation lesen wollen, um die Applikation in die Betriebsumgebung übernehmen zu können? Denken Sie daran, dass Ihre Sicherheitsrichtlinien nicht in der von Ihnen beabsichtigten Weise eingehalten werden können, wenn ein Administrator nicht weiß, wie Ihren Rollen bestimmte Benutzer und Gruppen korrekt zugewiesen werden.

Deklarative Sicherheit konfigurieren

Lassen Sie uns kurz betrachten, wie deklarative Sicherheitsüberprüfungen für eine COM+-Serverapplikation eingerichtet werden. Um rollenbasierte Autorisierungsüberprüfungen zu konfigurieren und durchzuführen, müssen Sie diese Schritte befolgen:

1. Aktivieren Sie die Autorisierungsüberprüfung, indem Sie die Option *Zugriffsüberprüfung für diese Anwendung erzwingen* auf der Registerkarte *Sicherheit* im Eigenschaftendialogfeld der Applikation selektieren.
2. Aktivieren Sie die Sicherheit auf derselben Registerkarte auf der Komponentenebene (Standard) oder auf der Prozessebene.
3. Setzen Sie die Authentifizierungsebene und die Identitätswechselebene.
4. Erstellen Sie mehrere Rollen in der Serverapplikation.
5. Fügen Sie jeder Komponente, Schnittstelle und Methode, für die Sie Zugriff gewähren möchten, eine Rolle hinzu.
6. Weisen Sie den von Ihnen erstellten Rollen Benutzer- und Gruppenkonten hinzu. (Ein Administrator tut dies in der Regel zur Entwurfszeit.)

Wenn Sie eine neue COM+-Serverapplikation erstellen, sind deklarative Autorisierungsüberprüfungen standardmäßig deaktiviert. Sie aktivieren sie, indem Sie die Option *Zugriffsüberprüfung für diese Anwendung erzwingen* auf der Registerkarte *Sicherheit* im Eigenschaftendialogfeld des Serverpakets selektieren. Dies ist die applikationsweit gültige Einstellung zum Einschalten der Sicherheit. Wenn die Sicherheit deaktiviert ist, kann jeder, der auf Ihren Computer zugreifen darf, Objekte erstellen und Methoden ausführen.

COM+-Applikationen besitzen außerdem ein Sicherheitsstufenattribut, das standardmäßig für Überprüfungen auf der Komponentenebene und Prozessebene konfiguriert ist. Sie können dieses Attribut so setzen, dass lediglich Überprüfungen auf der Prozessebene durchgeführt werden. In den meisten Fällen genügt die Standardeinstellung jedoch. Ich werde gleich beschreiben, was geschieht, wenn Sie eine Applikation so konfigurieren, dass ausschließlich Überprüfungen auf der Prozessebene ausgeführt werden. Lassen Sie uns jedoch zunächst davon ausgehen, dass Sicherheitsüberprüfungen auf der Komponentenebene ausreichend sind.

Das Entwerfen von Rollen ist bisweilen recht schwierig. Sobald Sie jedoch entschieden haben, welche Rollen verwendet werden sollen, können diese einfach zu einer COM+-Applikation hinzugefügt werden. Sie führen im administrativen Hilfsmittel *Komponentendienste* lediglich einen

Rechtsklick auf dem *Rollen*-Ordner aus und selektieren in dem folgenden Menü *Neu* und anschließend *Rolle*. Beachten Sie, dass Sie mit Hilfe von COM+-Admin-Objekten ebenfalls Rollen hinzufügen und rollenbasierte Autorisierungsüberprüfungen konfigurieren können.

Sobald Sie die entsprechenden Rollen zu einer COM+-Applikation hinzugefügt haben, können Sie deklarative Autorisierungsüberprüfungen konfigurieren. Sie bestimmen auf diese Weise, welche Benutzer auf verschiedene Methoden zugreifen dürfen. Sie können rollenbasierte Autorisierungsüberprüfungen auf der Methodenebene, auf der Schnittstellenebene und/oder auf der Komponentenebene hinzufügen.

Jede Komponente besitzt ein `ComponentAccessChecksEnabled`-Attribut, das standardmäßig aktiviert ist. Dies bedeutet bei aktivierter Sicherheit, dass den aufrufenden Benutzern kein Zugriff gewährt wird, wenn sie sich nicht in einer autorisierten Rolle befinden, die mit der Methode verknüpft ist, auf die der Zugriff erfolgen soll. Wenn sich ein Benutzer nicht in einer Rolle befindet, die für den Aufruf der Methode autorisiert ist, gibt COM+ den Standardfehler »Zugriff verweigert« zurück.

Abbildung 11.10 zeigt, wie die für eine Methode verfügbaren Rollen in ihren Eigenschaftendialogfeldern erscheinen. Sie entscheiden, ob die Zugriffsberechtigungen auf der Komponentenebene, der Schnittstellenebene oder der Methodenebene konfiguriert werden. Jede Lösung bietet Vorteile für bestimmte Situationen und kann zusammen mit den anderen Lösungen verwendet werden. Wenn Sie eine Rolle auf der Schnittstellenebene zuweisen, wird die Rolle implizit mit Ihren Schnittstellen und Methoden verknüpft. Das Design Ihrer Komponenten wird dann Einfluss darauf nehmen, ob Sie umfassende Überprüfungen auf der Komponentenebene oder feiner strukturierte Überprüfungen auf der Methodenebene verwenden. Beachten Sie, dass Überprüfungen auf der Methodenebene ein willkommener Zusatz in COM+ sind. MTS unterstützt keine deklarativen Überprüfungen auf der Methodenebene, so dass die Entwickler auf programmgesteuerte Überprüfungen ausweichen müssen.

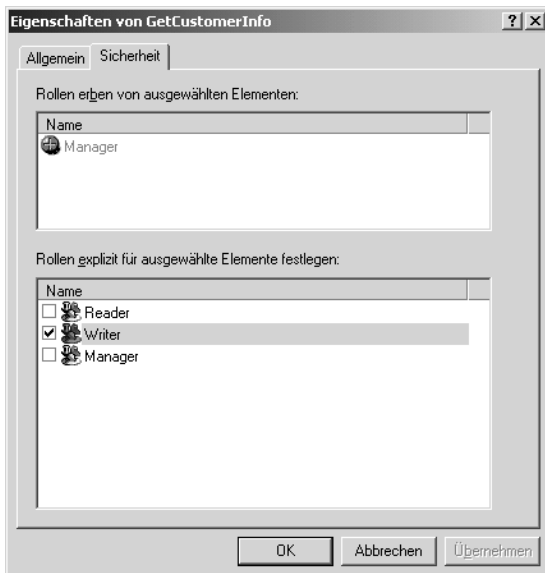


Abbildung 11.10: Das administrative Hilfsmittel Komponentendienste vereinfacht das Hinzufügen von Rollen und die Konfiguration von rollenbasierten Autorisierungsüberprüfungen

Autorisierung zur Laufzeit durchsetzen

Wenn eine COM+-Serverapplikation mit aktivierter Sicherheit nicht ausgeführt wird, wird ein Client mit zwei anfänglichen Barrieren konfrontiert, die das Aktivieren eines Objekts und den Aufruf einer Methode betreffen. Der Client muss über die erforderlichen Startberechtigungen und die prozessweit gültigen Zugriffsberechtigungen verfügen.

COM+ konfiguriert diese beiden Zugriffsberechtigungen automatisch im Hintergrund. Es erzeugt eine DACL, indem es die SIDs aus allen in der Applikation definierten Rollen entnimmt. Beachten Sie, dass auch die SID der Systemidentität zu dieser DACL hinzugefügt wird. Das System vergleicht die SIDs im Client-Token mit den SIDs in der DACL. Dies geschieht zur Laufzeit. Wenn das Benutzerkonto des Clients explizit oder implizit mit mindestens einer Rolle verknüpft ist, besitzt es sowohl Startberechtigungen als auch Zugriffsberechtigungen. Andernfalls kann es die Applikation nicht starten und auch keinen Methodenaufruf durchführen.

Wenn die COM+-Laufzeit feststellt, dass der Client eine Aktion initiiert, für die keine Autorisierung besteht, weist sie den Aufruf ab und gibt den HRESULT-Wert `E_ACCESSDENIED` zurück. Wenn der Client eine Visual Basic-Applikation ist, fängt die Visual Basic-Laufzeit diesen Fehler ab und löst den Fehler mit der Fehlernummer 70 und dem Fehlertext »Zugriff verweigert« aus.

Wenn Sie wissen, wie die DACL Autorisierungsüberprüfungen zur Laufzeit beeinflusst, können Sie verstehen, was in einer COM+-Applikation mit deaktivierter Sicherheit geschieht. COM+ bereitet eine leere DACL für die Startberechtigungen und prozessweit gültigen Zugriffsberechtigungen der Applikation vor. Die leere DACL zeigt an, dass alle SIDs implizite Zugriffsberechtigungen besitzen. Jeder Benutzer kann somit die Applikation starten und die Methoden der Applikation aufrufen.

Beachten Sie, dass Benutzer, die von Ihrem Server nicht authentifiziert werden können, möglicherweise mit Problemen konfrontiert werden, wenn sie Methodenaufrufe ausführen lassen möchten. Dies gilt auch dann, wenn Sie die applikationsweit gültige Sicherheitseinstellung deaktiviert haben. Wenn Sie den nicht authentifizierten Benutzern gestatten möchten, Methoden auszuführen, müssen Sie die Authentifizierungsebene der Applikation auf (*keine*) setzen. Denken Sie daran, dass diese Authentifizierungsebene in der Regel das Einrichten einer clientseitigen AppID erfordert, so dass Sie auch die Authentifizierungsebene der Client-Applikation auf (*keine*) setzen müssen.

Wenn eine Applikation die Sicherheit aktiviert hat und Sie das Sicherheitsstufenattribut gesetzt haben, um Überprüfungen nur auf der Prozessebene durchführen zu lassen, initiiert COM+ eine einzelne applikationsweite Autorisierungsüberprüfung. Dies geschieht, wenn ein Aufruf die Applikationsgrenze erreicht. COM+ führt diese Überprüfung mit Hilfe der bereits beschriebenen DACL durch. Beachten Sie, dass Sie keine detaillierte Kontrollmöglichkeit besitzen, wenn eine Applikation auf diese Weise konfiguriert wird. Wenn das Client-Token mindestens eine SID enthält, die auf eine mit einer Applikationsrolle verknüpften SID verweist, erhält der Client einen vollständigen Zugriff auf die gesamte Applikation.

Sie sollten die Standardeinstellung des Sicherheitsstufenattributs in den meisten Fällen unverändert lassen, so dass COM+ die Autorisierungsüberprüfungen durchführt, die Sie auf der Methoden-, Schnittstellen- und Komponentenebene konfiguriert haben. Die COM+-Laufzeit verwendet einen einfachen Algorithmus, mit dessen Hilfe bestimmt wird, ob ein Aufruf erfolgreich ausgeführt werden soll. Dieser Algorithmus überprüft, ob sich der Aufrufer in mindestens einer Rolle befindet, die mit der gewünschten Methode (oder der entsprechenden Schnittstelle bzw. Komponente) verknüpft ist. In diesem Fall wird der Aufruf erfolgreich durchgeführt. Befindet sich der

Aufrufer in keiner autorisierten Rolle, weist die COM+-Laufzeit den Aufruf ab, indem sie einen »Zugriff verweigert«-Fehler an den Client zurückgibt.

Lassen Sie uns nun ein etwas komplizierteres Szenario betrachten. Stellen Sie sich vor, Komponente A und Komponente B sind in einer Serverapplikation konfiguriert, deren Autorisierungsüberprüfung aktiviert ist. Eine Client-Applikation erstellt ein Objekt aus Komponente A, das wiederum ein Objekt aus Komponente B erzeugt, wie in Abbildung 11.11 dargestellt. Die Client-Applikation wird unter der Identität eines Benutzerkontos ausgeführt, das sich in einer Rolle befindet, die den Zugriff auf Komponente A autorisiert. Das Benutzerkonto befindet sich hingegen nicht in einer Rolle, die den Zugriff auf Komponente B autorisiert. Was geschieht nun, wenn der Client eine Methode des aus Komponente A erstellten Objekts ausführen lassen möchte und dieses Objekt versucht, eine Methode des aus Komponente B erzeugten Objekts aufzurufen? Der Aufruf wird erfolgreich durchgeführt, da keine internen Applikationssicherheitsüberprüfungen implementiert sind. Sie sollten die Folgen dieses Beispiels verstehen, wenn Sie Ihre Sicherheitsrichtlinien entwerfen.

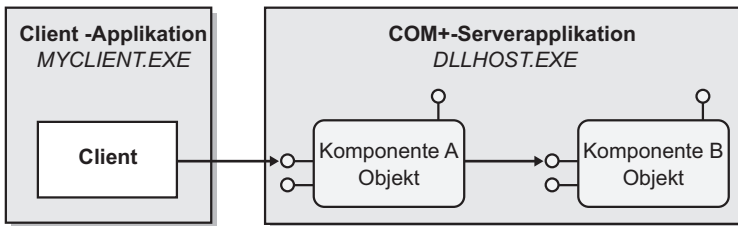


Abbildung 11.11: COM+ führt Autorisierungsüberprüfungen nur dann durch, wenn ein Aufruf die Applikationsgrenze überschreitet.

Lassen Sie uns eine geringfügige Variation des Beispiels aus Abbildung 11.11 betrachten. Wenn Sie Komponente A in eine Serverapplikation und Komponente B in eine Bibliotheksapplikation installieren, wird automatisch eine Sicherheitsüberprüfung zwischen den beiden Komponenten ausgeführt. Dies geschieht natürlich nur dann, wenn beide Applikationen Sicherheitsüberprüfungen aktiviert haben. Sie sollten jedoch vorsichtig sein, wenn Sie die Sicherheit für Komponenten konfigurieren, die nicht direkt von Clients verwendet werden, um Objekte zu erstellen. Solche Komponenten sollten jeden Zugriff verweigern. Sie sollten jedoch anderen Komponenten in Ihrer Applikation trauen, wenn diese Komponenten Methoden auf Weisung autorisierter Clients aufrufen.

Aufrufe zwischen Serverprozessen können ebenso kompliziert sein. Was geschieht zum Beispiel, wenn eine Client-Applikation, die unter der Identität von Bob ausgeführt wird, einen Aufruf an Serverapplikation A durchführt, die wiederum Serverapplikation B aufruft (wie in Abbildung 11.12 dargestellt)? Sie erwarten vielleicht, dass Serverapplikation B die Sicherheitsüberprüfungen mit Bobs Sicherheitsinformationen durchführt. Dies ist jedoch nicht der Fall. COM+ führt Sicherheitsüberprüfungen für jeden Aufruf durch. Sie sollten den Unterschied zwischen dem direkten Aufrufer, der sich an der nächsten Position in der Aufrufkette befindet, und dem ursprünglichen Aufrufer verstehen, der sich am Anfang der Aufrufkette befindet. COM+ führt die Sicherheitsüberprüfungen immer mit den Informationen des direkten Aufrufers durch. Beachten Sie, dass COM+ die Identität von Bob an die Serverapplikation B weitergibt. Diese kontextuellen Informationen sollten jedoch ausschließlich für die Überwachung verwendet werden.

Bob ist in diesem Beispiel der ursprüngliche Aufrufer. Das Hauptnamenskonto, das als Identität für Serverapplikation A dient, ist der direkte Aufrufer. Serverapplikation B führt die eigenen

Sicherheitsüberprüfungen mit Hilfe des direkten Aufrufers durch. Dieses Sicherheitsschema erfordert, dass Serverapplikation B darauf vertraut, dass Serverapplikation A überprüft hat, ob Bobs Aktionen autorisiert sind.

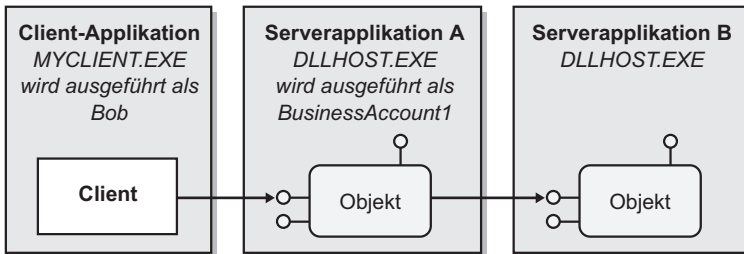


Abbildung 11.12: COM+ führt rollenbasierte Autorisierungsüberprüfungen durch, wenn ein Aufruf die Grenzen der Applikation passiert, deren Sicherheit aktiviert ist. Sicherheitsüberprüfungen werden jedoch immer mit dem Token des direkten Aufrufers und nicht mit dem Token des ursprünglichen Aufrufers durchgeführt.

Wie Sie sehen, müssen Sie vorsichtig sein. Wenn Sie einer Rolle Zugriffsberechtigungen für eine Komponente zuweisen, autorisieren Sie implizit die Mitglieder der Rolle für den Zugriff auf jede von der Komponente verwendete Ressource. Diese Ressourcen enthalten häufig andere Komponenten, Datenbanken, Webserver und Dateiserver. Dieses Modell sollte Sie ermutigen, Autorisierungsüberprüfungen frühestmöglich durchzuführen und auf solche Autorisierungsüberprüfungen zu verzichten, die von der ursprünglichen Authentifizierungsposition aus abwärts durchgeführt werden.

Programmgesteuerte Sicherheit

COM+ bietet Ihnen die programmgesteuerte Sicherheit an, um die deklarative Sicherheit zu ergänzen und zu erweitern. Sie können Ihre eigenen Sicherheitsüberprüfungen mit Hilfe der Object-Context- oder SecurityCallContext-Schnittstelle schreiben. Die beiden Schnittstellen weisen eine sich überlappende Funktionalität auf, so dass in den meisten Fällen eine Schnittstelle genügt.

Betrachten Sie ein kurzes Beispiel, das eine SecurityCallContext-Referenz abruft und in einer Methodenimplementierung verwendet. Dieses Beispiel prüft, ob die Methode in einer gesicherten Applikation ausgeführt wird. Der Aufruf schlägt andernfalls fehl:

```
' Definiert einen Standard-HRESULT für den »Zugriff verweigert«-Fehler.
Const E_ACCESSDENIED = &H80070005
```

```
Dim scc As SecurityCallContext
Set scc = GetSecurityCallContext
If scc.IsSecurityEnabled() Then
    ' Gesicherte Operation ausführen.
Else
    ' »Zugriff verweigert«-Fehler für Aufrufer auslösen.
    Err.Raise E_ACCESSDENIED, _
        "MyDll.MyClass.MyMethod", _
        "Zugriff verweigert"
End If
```

IsSecurityEnabled gibt Aufschluss über zwei Informationen. Es teilt Ihnen erstens mit, ob der aktuelle Aufruf in einer COM+-Applikation mit aktivierter Sicherheit ausgeführt wird. Sie erfahren zweitens, ob die Komponente konfiguriert wurde, um Autorisierungsüberprüfungen auf der Komponenten-, Schnittstellen- oder Methodenebene durchzuführen. Möchten Sie Ihren Code ausschließlich in einer gesicherten COM+-Applikation ausführen lassen, können Sie den Code des vorherigen Beispiels benutzen, um einen »Zugriff verweigert«-Fehler an den Aufrufer zurückzugeben, wenn die Applikation nicht gesichert ist. Beachten Sie, dass das Beispiel ein Standard-HRESULT benutzt, um einen Fehler für den Client auszulösen. Dies ist derselbe HRESULT-Wert, den COM+ verwendet, wenn es eine deklarative Autorisierungsüberprüfung abweist.

Sie können auch Code schreiben, um zu prüfen, ob der Aufrufer einer bestimmten Rolle zugeordnet ist. Die IsCallerInRole-Methode akzeptiert einen Zeichenfolgenparameter für die zu testende Rolle. Die Methode gibt True zurück, wenn dieser Rolle das Benutzerkonto des Aufrufers oder dessen Benutzergruppenkonto zugeordnet ist. Sie sollten jedoch aufmerksam sein, da IsCallerInRole immer True zurückgibt, wenn die Methode in einer Applikation mit deaktivierter Sicherheit aufgerufen wird. Einige Programmierer rufen IsCallerInRole zusammen mit IsSecurityEnabled auf, wie der folgende Code zeigt:

```
Dim scc As SecurityCallContext
Set scc = GetSecurityCallContext
If scc.IsSecurityEnabled() And _
    scc.IsCallerInRole("Manager") Then
    ' Eine gesicherte Manager-Operation ausführen.
Else
    Err.Raise E_ACCESSDENIED, _
        "MyDll.MyClass.MyMethod", _
        "Zugriff verweigert"
End If
```

Sie entscheiden, ob ein gemeinsamer Aufruf von IsSecurityEnabled mit IsCallerInRole erforderlich ist. Der Vorteil, der sich hieraus ergibt, besteht darin, dass Sie einen Benutzer davon abhalten können, eine sensitive Operation auszuführen, wenn eine COM+-Applikation nicht korrekt konfiguriert wurde. Der Vorteil des Aufrufs von IsCallerInRole ohne IsSecurityEnabled besteht darin, dass ein Administrator Ihre programmgesteuerten Autorisierungsüberprüfungen einfach deaktivieren kann, indem er die deklarativen Autorisierungsüberprüfungen deaktiviert. Dies kann wichtig sein, wenn der Administrator die Diagnose einer Applikation durchführen muss.

Wie Sie sehen, lässt Sie IsCallerInRole benutzerdefinierte Autorisierungsüberprüfungen für einzelne Methoden durchführen. Sie erhalten auf diese Weise eine umfassendere Kontrolle als Ihnen die deklarative Sicherheit bieten kann, da Ihre Autorisierungsüberprüfungen bedingte Logik enthalten können. Sie können zum Beispiel eine SubmitOrder-Methode schreiben, so dass nur die Benutzer in der Manager-Rolle eine Bestellung abschicken können, wenn die Anforderung DM 5000 übersteigt. Der folgende Code demonstriert dies:

```
Const E_ACCESSDENIED = &H80070005
Dim scc As SecurityCallContext
Set scc = GetSecurityCallContext
Dim Amount As Currency
' Bestellsumme ermitteln.
Amount = GetAmount()
' Manager-Rolle für jeden Betrag über DM 5.000 prüfen.
If Amount > 5000 Then
    If Not scc.IsSecurityEnabled() Or _
```



```

Not scc.IsCallerInRole("Manager") Then
    ' (caller <> manager) AND (amount > $5,000)
    ObjCtx.SetAbort ' Transaktion abbrechen, wenn bereits eine ausgeführt wird.
    Dim ErrDesc As String
    ErrDesc = "Manager required when amount exceeds $5,000"
    Err.Raise E_ACCESSDENIED, , ErrDesc
End If
End If
' Gesicherte Operation jetzt durchführen.

```

Beachten Sie, dass COM+ einen `mtsErrCtxRoleNotFound`-Fehler generiert, wenn Sie `IsCallerInRole` in einer COM+-Applikation mit aktivierter Sicherheit aufrufen und einen Rollennamen übergeben, der nicht existiert.

Sowohl COM+ als auch MTS unterstützen die von der `ObjectContext`-Schnittstelle zur Verfügung gestellte sicherheitsbezogene Funktionalität. Die `SecurityCallContext`-Schnittstelle ist jedoch neu in COM+. `IsSecurityEnabled` und `IsCallerInRole` werden von beiden Schnittstellen unterstützt, und beide Versionen arbeiten in der gleichen Weise. Die `SecurityCallContext`-Schnittstelle stellt aber zusätzlich die `IsUserInRole`-Methode und mehrere Auflistungen mit Informationen über vorherige Aufrufer zur Verfügung. Diese Methode und die Auflistungen werden von der `ObjectContext`-Schnittstelle nicht angeboten.

Sie können die `IsUserInRole`-Methode benutzen, um eine benutzerdefinierte Autorisierungsüberprüfung für einen vorherigen Aufrufer durchzuführen. Der folgende Code prüft zum Beispiel, ob der ursprüngliche Aufrufer der `Manager`-Rolle zugeordnet ist.

```

Dim oc As ObjectContext
Set oc = GetObjectContext()
Dim scc As SecurityCallContext
Set scc = GetSecurityCallContext()

Dim OriginalCaller As String
OriginalCaller = oc.Security.GetOriginalCallerName()
If scc.IsSecurityEnabled() And _
    scc.IsUserInRole(OriginalCaller, "Manager") Then
    ' Manager-Operation ausführen.
Else
    Err.Raise E_ACCESSDENIED, _
        "MyDll.MyClass.MyMethod", _
        "Zugriff verweigert"
End If

```

Obwohl die `IsUserInRole`-Methode sehr verlockend ist, sollten Sie sie sparsam verwenden. Im Gegensatz zur `IsCallerInRole`-Methode erfordert die `IsUserInRole`-Methode eine Datenübermittlung zum Domänencontroller und zurück. Aus der Design-Perspektive betrachtet, steht die Verwendung von `IsUserInRole` im Widerspruch zu Autorisierungsüberprüfungen, die möglichst nahe am Benutzer durchgeführt werden. Wenn Sie diese Methode zu häufig benutzen, sollten Sie darüber nachdenken, ein neues Design für Ihre Applikation zu entwerfen, um Autorisierungsüberprüfungen für Positionen durchzuführen, die in der Aufrufkette weiter zurückliegen.

Benutzerdefinierten Überwachungscode hinzufügen

Die programmgesteuerte COM+-Sicherheit erlaubt Ihnen nicht nur, bedingte Autorisierungsüberprüfungen durchzuführen. Sie stellt außerdem kontextuelle Informationen über die Identität der

Clients zur Verfügung, die in der Aufrufkette weiter zurückliegen. Sie können Identitätsinformationen über den Objektersteller und den Objektaufrufer abrufen. (Eine COM+-Applikation muss für Zugriffsüberprüfungen auf der Komponentenebene konfiguriert sein, damit die folgenden Beispiele nachvollzogen werden können.)

Obwohl die *Security*-Eigenschaft der *ObjectContext*-Schnittstelle Informationen über den Ersteller und den Aufrufer enthält, sind beide in der Regel kongruent. Die mit dem Ersteller verknüpften kontextuellen Informationen wurden ursprünglich für MTS erzeugt und werden nun in COM+ missbilligt. Sie sollten lediglich den Namen des Aufrufers berücksichtigen. Der folgende Code ruft den Benutzerkontonamen des Aufrufers ab:

```
Dim oc As ObjectContext
Set oc = GetObjectContext()
Dim Caller As String
Caller = oc.Security.GetOriginalCallerName()
```

COM+ stellt die Konto- und Autoritätsnamen der in der Aufrufkette zurückliegenden Aufrufer zur Verfügung, um kontextuelle Informationen für die benutzerdefinierte Überwachung anzubieten. Sie können eine Überwachung innerhalb einer Methodenimplementierung durchführen, indem Sie den Namen des ursprünglichen Aufrufers in das Windows-Ereignisprotokoll oder eine Datenbank schreiben.

In einigen Situationen möchten Sie vielleicht überwachen, ob Benutzer eine Transaktion erfolgreich abgeschlossen haben. In anderen Situationen werden Sie protokollieren wollen, wann authentifizierte Benutzer Operationen durchführen möchten, für die keine Autorisierung besteht. Wenn Sie zum Beispiel mit einer programmgesteuerten Autorisierungsüberprüfung feststellen, dass ein Benutzer eine nicht erlaubte Operation durchführen möchte, können Sie den folgenden Code schreiben, um diesen Versuch in das Windows-Ereignisprotokoll zu schreiben:

```
Dim Caller As String, LogMsg As String
Dim oc As ObjectContext
Set oc = GetObjectContext
Caller = oc.Security.GetOriginalCallerName()
LogMsg = Caller & " has attempted to look at everybody's salary."
App.LogEvent LogMsg, vbLogEventTypeError
```

Dieses Beispiel benutzt die *Security*-Eigenschaft der *ObjectContext*-Schnittstelle. Die *GetOriginalCallerName*-Methode der Eigenschaft gibt den Kontonamen des Aufrufers im Format *Autorität/Hauptname* zurück, wie zum Beispiel in *MeineDomäne\KeithB*. Das *Security*-Objekt bietet darüber hinaus die Methoden *GetDirectCallerName*, *GetDirectCreatorName* und *GetOriginalCreatorName* an. *GetDirectCreatorName* und *GetOriginalCreatorName* sind unter COM+ nicht mehr zulässig und sollten deshalb nicht verwendet werden. In den meisten Fällen werden Sie den Namen des ursprünglichen Aufrufers überwachen wollen.

Der Einsatz der *ObjectContext*-Schnittstelle ist die einfachste Möglichkeit, um Informationen über Aufrufer, die in der Aufrufkette zurückliegen, zu erhalten. Die *SecurityCallContext*-Schnittstelle ist ein wenig komplizierter, kann aber detailliertere Informationen zur Verfügung stellen. Dazu zählt auch der Name jedes vorherigen Aufrufers.

Ein *SecurityCallContext*-Objekt bietet eine *Item*-Auflistung mit mehreren nützlichen Elementen an, die in Tabelle 11.3 aufgeführt sind. Sie können diese Auflistung benutzen, um die Anzahl der Aufrufer zu ermitteln, die in der Aufrufkette zurückliegen. Sie können außerdem die kleinste Authentifizierungsebene der Applikation und Informationen über den ursprünglichen Aufrufer und

den direkten Aufrufer abrufen. Wenn Sie Informationen über einen bestimmten Aufrufer erhalten möchten, müssen Sie die `SecurityIdentity`-Schnittstelle benutzen. Diese Schnittstelle ist in der Typbibliothek *COM+ Services Type Library* definiert.

Element	Typ
NumCallers	Integer
MinAuthenticationLevel	Integer
Callers	SecurityCallers-Referenz
DirectCaller	SecurityIdentity-Referenz
OriginalCaller	SecurityIdentity-Referenz

Tabelle 11.3: Elemente der Item-Auflistung des `SecurityCallContext`-Objekts

Ein `SecurityIdentity`-Objekt verfügt über eine Item-Auflistung mit den in Tabelle 11.4 aufgeführten Elementen. Sie können dieses Objekt benutzen, um die für den Aufruf notwendigen Informationen über den Kontonamen des Aufrufers, die SID, den SSP, die Authentifizierungsebene und die Identitätswechselebene zu ermitteln. Hier ein einfaches Beispiel, das den Kontonamen und die Authentifizierungsebene des ursprünglichen Aufrufers ermittelt:

```
Dim scc As SecurityCallContext
Set scc = GetSecurityCallContext()
Dim Caller As SecurityIdentity
Set Caller = scc.Item("OriginalCaller")
Dim UserName As String, AuthLevel As Integer
UserName = Caller.Item("AccountName")
AuthLevel = Caller.Item("AuthenticationLevel")
```

Element	Typ
SID	Binäres Array
AccountName	String
AuthenticationService	Integer
AuthenticationLevel	Integer
ImpersonationLevel	Integer

Tabelle 11.4: Elemente der Item-Auflistung des `SecurityIdentity`-Objekts

COM+ erlaubt Ihnen, jeden Aufrufer in der logischen Aufrufkette zu untersuchen. Dieser Vorgang wird als Kausalität bezeichnet (siehe sechstes Kapitel). Was geschieht zum Beispiel, wenn eine Client-Applikation einen Aufruf an eine Serverapplikation vornimmt, wobei diese Applikation eine zweite und die zweite eine dritte Applikation aufruft? COM+ reicht in diesem Fall kontextuelle Informationen über jeden Aufrufer die Aufrufkette hinab weiter. Die Serverapplikation, die sich am Ende der Aufrufkette befindet, kann somit kontextuelle Informationen über die Identität aller Aufrufer ermitteln.

Um diese Informationen zu erhalten, müssen Sie das `Callers`-Element aus der `SecurityCallContext`-Schnittstelle abrufen. Dieses Element ist eine Objektreferenz vom Typ `SecurityCallers`. Ein `Security-`

Callers-Objekts bietet eine Auflistung mit einer *SecurityIdentity*-Referenz für jeden Aufrufer an. Hier ein Beispiel, das jeden Aufrufer in der Kausalität überprüft:

```
Dim scc As SecurityCallContext
Set scc = GetSecurityCallContext()
Dim Callers As SecurityCallers
Set Callers = scc.Item("Callers")
Dim Caller As SecurityIdentity
Dim Item As Integer
For Item = 0 To (Callers.Count - 1)
    Set Caller = Callers.Item(Item)
    ' Auf Identität des Aufrufers zugreifen.
Next Item
```

Lassen Sie mich noch einmal erwähnen, dass COM+ all diese kontextuellen Informationen über die in der Aufrufkette zurückliegenden Aufrufer aus Überwachungsgründen zur Verfügung stellt. Wenn Sie *IsUserInRole* aufrufen müssen, um eine Autorisierungsüberprüfung für einen Aufrufer inmitten der Kausalität durchzuführen, sollten Sie möglicherweise Ihr Sicherheitsschema überdenken. Sie sollten Ihre Autorisierungsüberprüfungen in diesem Fall an einer anderen Position durchführen lassen, wenn der Benutzer authentifiziert wird.

Sicherheit in IIS

Dieses Kapitel schließt mit einer Diskussion über die IIS-Sicherheit und deren Bezug zu den anderen sicherheitsbezogenen Themen, die bisher besprochen wurden. Sie können Webapplikationen erstellen, die ASP-Seiten als Zugang zu einer COM+-Applikation verwenden. Sie verfügen außerdem über die Möglichkeit, Webapplikationen mit einer HTML-Benutzeroberfläche zu erzeugen. Andere Webapplikationen benutzen ausschließlich HTTP und nicht HTML. Doch unabhängig davon, welche Webapplikation Sie generieren, erreichen Ihre Clients Ihre Visual Basic-Komponenten nur über IIS. Dies sollte Motivation genug für Sie sein, um zu lernen, wie sich die IIS-Sicherheit in die COM+-Sicherheit integriert.

Abbildung 11.13 zeigt die Vorgänge innerhalb einer typischen Webapplikation, die mit ASP-Seiten und Visual Basic-Komponenten erzeugt wurde. Beachten Sie, dass *INETINFO.EXE* der Einstiegspunkt des Clients in die mittlere Schicht ist. *INETINFO.EXE* ist somit der Prozess, der den Client authentifizieren muss. Dies bedeutet, dass Sie einen der Authentifizierungsmodi verwenden müssen, die von IIS unterstützt werden.

IIS muss außerdem jede HTTP-Anforderung unter der Identität eines spezifischen Sicherheitshauptnamens ausführen. *INETINFO.EXE* wird unter der Identität des Systems ausgeführt. Würde IIS die Client-Anforderungen unter Verwendung des eigenen Prozess-Tokens bearbeiten, würde jede Anforderung unter der Systemidentität ausgeführt und wiese praktisch keine Einschränkungen auf. Dies ist offensichtlich keine gute Lösung. IIS verwendet stattdessen den Identitätswechsel, so dass die Client-Anforderungen unter der Identität eingeschränkter Benutzerkonten ausgeführt werden.

Die Threading-Architektur von IIS und ASP wurde bereits im neunten Kapitel beschrieben. Sie erinnern sich bestimmt daran, dass IIS eine Anforderung auf einem MTA-Thread bearbeitet, wenn ein Client eine einfache *.HTM*-Seite verlangt. Wenn ein Client jedoch eine *.ASP*-Seite anfordert, übergibt IIS die Anforderung an *ASP.DLL*. *ASP.DLL* schaltet die Anforderung auf einen STA-Thread, um die Seite ausführen zu können. Wenngleich die Threading-Architektur kompliziert ist, wird der Identitätswechsel auch dann korrekt in IIS durchgeführt, wenn die Anforde-

rung von einem STA-Thread aus dem ASP-Thread-Pool bearbeitet wird. Dies bedeutet, dass Client-Anforderungen immer unter der Identität eines Benutzerkontos und niemals unter der Identität des Systems ausgeführt werden.

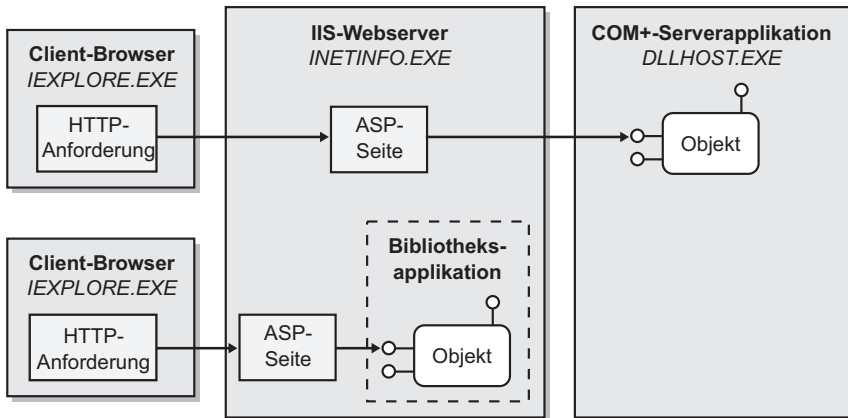


Abbildung 11.13: Clients, die HTTP-Anforderungen an IIS senden, werden von INETINFO.EXE authentifiziert. IIS verwendet den Identitätswechsel, um jede Client-Anforderung unter der Identität eines spezifischen Benutzerkontos auszuführen.

IIS verrichtet einen zuverlässigen, aber keinen absolut sicheren Job, wenn Client-Anforderungen einem benutzerspezifischen Token zugeordnet werden. Es besteht ein geringes Restrisiko, dass ein Benutzer mit nicht legalen Absichten den Thread, der die Benutzeranforderung bearbeitet, auf das System-Token zurücksetzt. Wie Sie sich bestimmt vorstellen können, möchten Sie nicht, dass dieser Benutzer seinen Code unter der Identität des Systems ausführen lässt. Er wäre dann mit keiner Einschränkung konfrontiert.

Im neunten Kapitel wurde ebenfalls beschrieben, wie IIS WAM-Komponenten benutzt, um isolierte IIS-Applikationen auszuführen. Eine Motivation für das isolierte Ausführen einer IIS-Applikation besteht darin, die Fehlertoleranz zu verbessern. Darüber hinaus wird das soeben erwähnte Risiko einer Attacke durch einen Benutzer beseitigt.

WAM-Komponenten für isolierte IIS-Applikationen werden für die Ausführung in einer speziellen COM+-Serverapplikation konfiguriert. IIS erstellt automatisch eine dieser COM+-Serverapplikationen im Hintergrund, wenn dies erforderlich ist. Die Applikationsidentität wird daraufhin mit einem bestimmten Benutzerkonto eingerichtet, das den Namen *IWAM_Machine* trägt, wobei *Machine* für den NETBIOS-Namen des lokalen Computers steht. Dieses Konto ist ein lokales Benutzerkonto, das mit umfassenderen Einschränkungen als das Systemkonto versehen ist. Selbst wenn der »böse« Benutzer nun seine Anforderung auf das Prozess-Token schalten würde, könnte er nicht unter der Identität des Systems arbeiten. Wie Sie sehen, kann das Ausführen einer isolierten IIS-Applikation eine zuverlässige Vorsichtsmaßnahme sein. Viele Unternehmen führen Ihre IIS-Applikationen deshalb außerhalb von *INETINFO.EXE* aus, indem sie die *Anwendungsschutz*-Einstellung auf *Mittel* (zusammengefasst) oder *Hoch* (isoliert) setzen.

Sie wissen nun, dass IIS Client-Anforderungen ausführt, indem die Identität verschiedener Benutzerkonten verwendet wird. Doch welches Benutzerkonto benutzt IIS für eine Anforderung? Dies ist davon abhängig, welcher Authentifizierungsmodus von IIS verwendet wird. Bevor Sie erfahren, welche Modi IIS benutzt, um Clients zu authentifizieren, werden wir einen Schritt zurück gehen und über einige wichtige Aspekte der HTTP-Authentifizierung sprechen.

HTTP-Sicherheitsstandards

HTTP ist ein weitverbreitetes Protokoll, das es Ihnen erlaubt, Clients zu erreichen, die unter verschiedenen Betriebssystemen und Browsern ausgeführt werden. Es gibt Ihrer Applikation die Möglichkeit, jeden Benutzer des Internets zu erreichen. Diese Vorteile sind jedoch mit einigen Unsicherheiten verbunden.

Wenn Sie HTTP benutzen, können Sie nicht davon ausgehen, dass die Identität des Clients mit einem Benutzerkonto einer Windows 2000-Domäne authentifiziert werden kann. Sie können auch nicht voraussetzen, dass der Client unter Windows ausgeführt wird. Sie müssen damit rechnen, dass der Client-Computer und der Webserver über eine Firewall miteinander kommunizieren. Ein Authentifizierungsmechanismus wie zum Beispiel Kerberos oder NTLM genügt somit den Anforderungen des Internets nicht.

Lassen Sie mich kurz zusammenfassen, wie die Windows-Authentifizierung in einem privaten Netzwerk zwischen einem Benutzer und einem Remote-Servercomputer durchgeführt wird. Wie Sie wissen, muss ein Vertrauenspfad zwischen der Benutzerautorität und dem Servercomputer vorhanden sein. Wenn der Benutzer und der Servercomputer derselben Domäne oder verschiedenen Domänen mit einer Vertrauensbeziehung zugeordnet sind, kann der Servercomputer den Benutzeraufruf mit Kerberos oder NTLM authentifizieren. Kerberos bietet darüber hinaus die Möglichkeit, dem Benutzer die Identität des Servers zu bestätigen.

Die Netzwerk-Authentifizierung mit Kerberos oder NTLM benötigt einen Domänencontroller. Der Domänencontroller führt ein Protokoll aus, das dem Client und dem Server hilft, einen allgemeinen Sitzungsschlüssel zu ermitteln. Dieses »gemeinsame Geheimnis« ermöglicht das Einrichten eines sicheren Kanals. Nachrichten werden mit Hilfe des Sitzungsschlüssels auf der Versenderseite verschlüsselt und auf der Empfängerseite entschlüsselt. Dieser Vorgang wird häufig als symmetrische Verschlüsselung bezeichnet.

Das Übermitteln eines vertraulichen Schlüssels zwischen zwei Internet-Computern ist nicht durchführbar. Jede Autorität müsste dazu eine bidirektionale Vertrauensbeziehung zu jeder anderen Autorität aufweisen.

HTTP wird glücklicherweise ständig weiterentwickelt, so dass in den Bereichen Sicherheit und Authentifizierung ein akzeptabler Standard erreicht wurde. SSL (Secure Sockets Layer) wurde als Authentifizierungsprotokoll, das vorwiegend für HTTP verwendet wird, anerkannt. SSL führt die Authentifizierung plattformübergreifend durch und kann dazu verwendet werden, Benutzer von unterschiedlichen Organisationen auch dann zu authentifizieren, wenn sie keiner allgemeinen Autorität zugeordnet sind. Die SSL-Authentifizierung ist bekannter als Kerberos oder NTLM. Doch auch sie basiert auf dem Konzept des Vertrauens.

Im SSL-Authentifizierungsschema muss keine Partei das Geheimnis verraten. Die Verschlüsselung basiert hier auf zwei Schlüsseln. Ein Schlüssel verschlüsselt die Nachrichten, während der zweite Schlüssel zum Entschlüsseln der Nachrichten verwendet wird. Eine Entität kann somit einen öffentlichen Schlüssel weitergeben und den privaten Schlüssel verborgen aufbewahren. Dieses Schema wird auch als asymmetrische Verschlüsselung bezeichnet.

Lassen Sie uns die Funktionsweise der asymmetrischen Verschlüsselung anhand eines Beispiels erörtern. Stellen Sie sich vor, ich gebe Ihnen meinen öffentlichen Schlüssel. Ich kann eine Nachricht mit meinem privaten Schlüssel verschlüsseln und an Sie senden. Sie können meine Nachricht anschließend mit meinem öffentlichen Schlüssel entschlüsseln. Wenn Sie dem Umstand vertrauen, dass ich der einzige mit einem privaten Schlüssel bin, der die Nachricht verschlüsseln konnte, können Sie sicher sein, dass die Nachricht von mir stammt. Sie können nun meinen öffentlichen Schlüssel benutzen, um Ihre Nachrichten zu verschlüsseln, bevor Sie diese an mich

verschicken. Wenn Sie darauf vertrauen, dass ich der einzige bin, der den privaten Schlüssel zum Entschlüsseln der Nachricht besitzt, können Sie davon ausgehen, dass die Nachricht von niemand anderem gelesen wird.

Es gibt einige Aspekte der asymmetrischen Verschlüsselung, die Sie beachten sollten. Das soeben beschriebene Beispiel führt keine gegenseitige Authentifizierung durch. Sie wissen von mir, doch ich weiß nichts über Sie. Müsste ich Sie authentifizieren, könnte ich mir hinsichtlich Ihrer Identität sicher sein, wenn Sie mir ebenfalls einen öffentlichen Schlüssel zukommen lassen. Das Beispiel sollte jedoch verdeutlicht haben, dass eine gegenseitige Authentifizierung nicht notwendig ist, um einen sicheren Kanal einzurichten. Solange zwei Parteien ihre öffentlichen Schlüssel austauschen, können beide sicher miteinander kommunizieren.

Beachten Sie, dass die asymmetrische Verschlüsselung verglichen mit der symmetrischen Verschlüsselung nur äußerst aufwendig zu berechnen ist (ca. der tausendfache Aufwand). Dies bedeutet, dass der Einsatz von öffentlichen und privaten Schlüsseln zu signifikanten Laufzeiteinbußen führt. SSL ist andererseits nicht auf die asymmetrische Verschlüsselung begrenzt. SSL kann ebenfalls einen vertraulichen Sitzungsschlüssel mit Hilfe der asymmetrischen Verschlüsselung übergeben, um einen schnelleren Kanal einzurichten, der auf der symmetrischen Verschlüsselung basiert. Wie Sie sehen, versucht SSL eine bessere Performance zu erzielen, wenn dies möglich ist.

Zertifikate

Wie Sie gesehen haben, gestatten Ihnen öffentliche und private Schlüssel das Einrichten eines sicheren Kanals. Dies ist ohne symmetrische Verschlüsselung und Domänencontroller möglich. Die Weitergabe von öffentlichen Schlüsseln führt jedoch zu einem anderen Sicherheitsproblem. Was geschieht, wenn Ihnen ein Benutzer einen öffentlichen Schlüssel schickt und Sie glauben lässt, dass dies mein Schlüssel sei? Wenn Sie sich täuschen lassen und Nachrichten mit diesem öffentlichen Schlüssel verschlüsseln und versenden, kann der Benutzer die Nachrichten lesen. Um dieses Sicherheitsloch zu schließen, erfordert die Weitergabe von öffentlichen Schlüsseln so genannte Zertifikate und eine zentrale Autorität, die von allen öffentlichen Schlüsselempfängern als vertrauenswürdig erachtet werden muss.

Ein Zertifikat (das auch als digitales Zertifikat bezeichnet wird) ist ein Dokument, das die Verknüpfung zwischen einem öffentlichen Schlüssel und einer Entität, wie zum Beispiel einer Person oder einem Unternehmen, beglaubigt. Ein Zertifikat bestätigt, dass der von dem Zertifikat verwahrte öffentliche Schlüssel der in dem Zertifikat genannten Entität zugeordnet ist. Der gebräuchliche Internet-Standard für Zertifikate ist in der ITU-Empfehlung (International Telecommunications Union) X.509 definiert. Ein X.509-Zertifikat besteht aus den folgenden Feldern:

- Version
- Seriennummer
- Signaturalgorithmus-ID
- Name des Ausstellers
- Gültigkeitsperiode
- Subjektnamen (Benutzer)
- Öffentliche Schlüsselinformationen des Subjekts
- Eindeutiger Bezeichner des Ausstellers
- Eindeutiger Bezeichner des Subjekts
- Ausstellerunterschrift

Ein Zertifikat muss mit dem privaten Schlüssel des Ausstellers unterzeichnet sein. Ohne diesen Schlüssel besitzt das Zertifikat keinen Wert. Die Unterschrift des Ausstellers kann mit dessen öffentlichem Schlüssel verifiziert werden. Wenn eine Entität dem Aussteller vertraut, kann sie sicher sein, dass der im Zertifikat enthaltene öffentliche Schlüssel dem im Zertifikat genannten Subjekt zugeordnet ist.

Zertifikate werden von einer Zertifikatautorität (CA – Certificate Authority) ausgegeben. Eine CA ist ein vertrauenswürdiger Dienst oder eine vertrauenswürdige Entität, die sich für die Identitäten derer verbürgt, denen sie Zertifikate ausstellt. Eine CA verbürgt sich genauer gesagt für die Echtheit des öffentlichen Schlüssels, der im Zertifikat aufgeführt ist. Damit dies möglich ist, muss ein Vertrauenspfad vorhanden sein. Sie müssen einer CA vertrauen, um dem von der CA ausgestellten digitalen Zertifikat vertrauen zu können.

Viele Drittanbieter, wie zum Beispiel VeriSign, agieren als CAs. Wenn Sie ein Zertifikat von einem CA-Drittanbieter erhalten möchten, müssen Sie Ihre Identität beweisen. Wenn die CA für Sie bürgen soll, muss sie sicher sein, dass Sie die Person sind, für die Sie sich ausgeben. Wenn Sie eine Zertifikatanforderung an eine CA senden, müssen Sie ebenfalls einen öffentlichen Schlüssel einschicken, für den Sie einen entsprechenden privaten Schlüssel besitzen. Die CA erstellt dann das Zertifikat mit Ihrem Namen, Ihrem öffentlichen Schlüssel und der digitalen Unterschrift und lässt Ihnen das Dokument zukommen.

Beachten Sie, dass ein Zertifikat erst dann nützlich ist, wenn der Zertifikatempfänger den öffentlichen Schlüssel des Ausstellers besitzt. Doch wie rufen Sie den öffentlichen Schlüssel der CA sicher ab? Die öffentlichen Schlüssel von CAs, wie zum Beispiel VeriSign, sind bereits in Browsern wie zum Beispiel Netscape Communicator und Microsoft Internet Explorer installiert. Wenn beide Parteien einen öffentlichen Schlüssel der CA besitzen, besteht ein Vertrauenspfad zwischen ihnen.

CAs können untergeordnete CAs zertifizieren, die daraufhin ihre eigenen Zertifikate ausstellen können. Es ist somit möglich, eine Hierarchie mit CAs zu erstellen. Die CA auf der obersten Ebene muss ohne Zertifikat von allen anderen CAs als vertrauenswürdige erachtet werden. Ihr öffentlicher Schlüssel muss bekannt sein. Die CA-Hierarchie erlaubt das Erstellen von Vertrauenspfaden zwischen nicht angegliederten Entitäten. Microsoft besitzt ein Produkt mit der Bezeichnung MCS (Microsoft Certificate Server), das den Unternehmen gestattet, eigene Zertifikate auszustellen.

SSL und HTTPS

Microsoft unterstützt vier verwandte Protokolle für die zertifikatbasierte Sicherheit: SSL 2 und 3 (Secure Sockets Layer), TLS (Transport Layer Security) und PCT 1 (Private Communication Technology).

Microsoft hat PCT erstellt, um mit SSL 2 zu konkurrieren. SSL 3 ist im Laufe der letzten Jahre jedoch zu dem Authentifizierungs-Standardprotokoll geworden, das mit HTTP verwendet wird. PCT ist nicht mehr gebräuchlich und sollte deshalb vermieden werden. TLS 1 ist eine neuere Version von SSL 3, so dass ich den Ausdruck SSL sowohl für SSL als auch für TLS verwenden werde.

SCHANNEL (Secure Channel) ist Microsofts Implementierung dieser vier Protokolle. SCHANNEL ist ein SSP, der sich wie NTLM und Kerberos in die Windows-Sicherheitsarchitektur fügt. SCHANNEL gestattet Applikationen wie zum Beispiel IIS und Internet Explorer, SSL zu verwenden, wenn die Authentifizierung mit Hilfe der asymmetrischen Verschlüsselung durchgeführt wird.

Für den Einsatz von SSL stehen zwei Möglichkeiten zur Verfügung. Die übliche Methode besteht darin, SSL mit einem Serverzertifikat zu benutzen. Der Client kann dann den Webserver authentifizieren. Der Server weiß nicht, wer der Client ist. Das Serverzertifikat genügt jedoch, um einen sicheren Kanal einzurichten. Der Webserver kann Nachrichten mit einem authentifizierten Client austauschen, ohne das Risiko fürchten zu müssen, dass ein nicht autorisierter Benutzer die Nachricht entziffert.

Die zweite Möglichkeit besteht darin, Client-Zertifikate zusammen mit einem Serverzertifikat zu verwenden. Der sich hieraus ergebende Vorteil ist, dass der Webserver die Clients authentifizieren kann. Jeder Client kann einem Benutzerkonto zugewiesen werden, so dass Standard-Authentifizierungstechniken, wie zum Beispiel Zugriffsbeschränkungen, mit Hilfe der rollenbasierten Sicherheit genutzt werden können. Dieses Schema ist jedoch nur schwierig einzurichten, da jeder Client über ein eigenes Zertifikat verfügen muss. Der MCS kann Ihnen jedoch helfen, wenn Sie Client-Zertifikate weitergeben möchten.

Client-Zertifikate können den Benutzerkonten einer Windows 2000-Domäne zugeordnet werden. Ihnen stehen dazu zwei Optionen zur Verfügung. Sie können jedem Zertifikat ein eigenes Benutzerkonto zuordnen, oder Sie weisen mehrere Zertifikate einem allgemeinen Benutzerkonto zu. Sie verfügen zum Beispiel über die Möglichkeit, jedes Zertifikat eines Benutzers in der Verkaufsabteilung einem einzelnen Benutzerkonto zuzuordnen. Die Dokumentation, die mit IIS und MCS ausgeliefert wird, enthält viele Informationen zum Erstellen und Weitergeben von Zertifikaten an die Clients.

Sobald auf dem Webserver (und optional auch auf dem Client) ein Zertifikat vorhanden ist, können Sie HTTPS (Secure HTTP) verwenden. HTTPS ist einfach HTTP, das über SSL ausgeführt wird. HTTPS unterscheidet sich darin von HTTP, dass es immer mit Hilfe eines vertraulichen Sitzungsschlüssels verschlüsselt kommuniziert. HTTPS verwendet standardmäßig Port 443. HTTP nutzt Port 80. Wenn der Client eine Verbindung zu IIS mit HTTPS einrichtet, werden die öffentlichen Schlüssel in den Zertifikaten mit einem vereinbarten Protokoll ausgetauscht, das als SSL-Authentifizierung bezeichnet wird.

Wenn Sie ein Serverzertifikat besitzen, können die Clients entweder über HTTP oder HTTPS miteinander kommunizieren. Sie können jedoch eine vollständige IIS-Website oder ein virtuelles Verzeichnis für die Verwendung von HTTPS konfigurieren. Wenn Sie über besondere Sicherheitsanforderungen verfügen, können Sie HTTPS für jede Anforderung benutzen. Denken Sie jedoch daran, dass HTTPS langsamer als HTTP ist, da es die asymmetrische Verschlüsselung einsetzt, um einen vertraulichen Sitzungsschlüssel einzurichten. Wenn einige Bereiche Ihrer Site weniger sensitiv als andere sind, sollten Sie ein Konglomerat aus HTTP und HTTPS verwenden.

IIS-Authentifizierungsmodi

Nachdem nun die Grundlagen der Zertifikate und von SSL beschrieben wurden, wenden wir uns den fünf IIS-Authentifizierungsmodi zu. Diese sind in Tabelle 11.5 in der Reihenfolge aufgeführt, in der sie von IIS zur Laufzeit verwendet werden.

Der erste Modus, der von IIS gewählt wird, ist der schnellste. Der anonyme Zugriffsmodus verwendet keine Authentifizierung. Der Webserver kennt die Identität des Clients somit nicht. IIS muss jedoch den Identitätswechsel einsetzen, damit die Anforderung nicht unter der Identität des Systemkontos ausgeführt wird. IIS benutzt standardmäßig ein Konto namens `IUSR_Machine`, das zusammen mit IIS installiert wird. `IUSR_Machine` ist lediglich das anonyme Standardbenutzerkonto. Sie können dem anonymen Benutzerkonto ein anderes Benutzerkonto mit dem Internetdienst-Manager zuweisen.

Da IIS immer versucht, eine Client-Anforderung ohne die Authentifizierung des Clients zu bearbeiten, müssen Sie den anonymen Zugriff verhindern, wenn Sie möchten, dass IIS die Client-Anforderung für die Ausführung unter der Identität des Benutzerkontos authentifiziert.

Authentifizierungsmodus	Beschreibung	Identitätswechsel	Standardmäßig aktiviert?
Anonymer Zugriff	Client wird nicht authentifiziert.	IUSR_MACHINE	Ja
Zertifikat-Authentifizierung	Authentifiziert mit SSL und Client-Zertifikaten	Benutzerkonten	Ja
Integrierte Windows-Authentifizierung	Authentifiziert mit Kerberos oder NTLM.	Benutzerkonten	Ja
Digest-Authentifizierung	Authentifiziert mit einem Kennwort, das in verschlüsselter Form übergeben wird, und einem Windows 2000-Domänencontroller.	Benutzerkonten	Nein
Grundauthentifizierung	Authentifiziert mit einem Kennwort, das als gewöhnlicher Text übergeben wird.	Benutzerkonten	Nein

Tabelle 11.5: IIS-Authentifizierungsmodi

Es gibt zwei Möglichkeiten, um zu verhindern, dass IIS eine Anforderung im anonymen Zugriffsmodus ausführt. Sie können den anonymen Zugriff für eine vollständige Site, ein virtuelles Verzeichnis oder individuelle Dateien mit dem Internetdienste-Manager sperren. Die zweite Möglichkeit, IIS zur Authentifizierung von Clients zu bewegen, besteht darin, die Berechtigungen für .HTM- und .ASP-Dateien zu ändern, so dass das *IUSR_Machine*-Konto nicht über die entsprechenden Zugriffsberechtigungen verfügt, um solche Seiten zu lesen. Sie können Zugriffsberechtigungen für individuelle .ASP-Dateien oder für virtuelle Verzeichnisse konfigurieren, sofern sich diese Elemente auf einer NTFS-Partition befinden. Wenn IIS mit dem *IUSR_Machine*-Konto auf eine Datei zugreifen möchte und dies misslingt, versucht es, den Client in einem der anderen Modi zu authentifizieren.

Der zweite Modus, der von IIS gewählt wird, ist die Zertifikat-Authentifizierung. Das Einrichten dieses Modus erfordert den größten administrativen Aufwand. Die Zertifikat-Authentifizierung ist jedoch die bevorzugte Technik zum Authentifizieren von Clients und zum Ausführen der Client-Anforderungen unter der Identität bestehender Benutzerkonten. Beachten Sie, dass die Informationen über das Client-Zertifikat programmgesteuert abgerufen werden können. Diese Informationen sind sowohl über das ASP-Objektmodell als auch über die Schnittstellen in der Typbibliothek *COM+ Services Type Library* erhältlich.

Der integrierte Windows-Authentifizierungsmodus verwendet Kerberos oder NTLM. Der Client und der Webserver vereinbaren das zu benutzende Protokoll, und im Anschluss daran wird ein sicherer Kanal eingerichtet. Die Einschränkung, die mit diesem Modus verbunden ist, besteht darin, dass Windows und der Internet Explorer benötigt werden. Es verwendet außerdem Ports, die von den meisten Firewall-Programmen nicht zugelassen werden. Während diese Lösung nicht für das Internet geeignet ist, bietet sie sich für Client-Computer und Webserver an, die in einem privaten Windows-Netzwerk ausgeführt werden und einer allgemeinen Autorität vertrauen.

Die Digest-Authentifizierung ist neu in IIS 5. Sie ist standardmäßig gesperrt, so dass Sie sie zunächst aktivieren müssen, bevor Sie sie verwenden. Wenn ein Benutzer zum ersten Mal eine Verbindung zu einer Site herstellt, wird er dazu aufgefordert, einen Benutzernamen und ein Kennwort einzugeben. Die Digest-Authentifizierung gleicht NTLM darin, dass sie ein Anforderungs-/

Antwortprotokoll verwendet und das Kennwort nicht im gewöhnlichen Textformat über das Internet überträgt. Sie weist jedoch einige signifikante Probleme auf, so dass diese Authentifizierung für die meisten Situationen ungeeignet ist. Das größte Problem ist, dass der Webserver auf einem Domänencontroller ausgeführt werden muss. Dies begünstigt illegale Attacken, die dem Domänencontroller und jedem Element innerhalb der Domäne schaden können. Ein weiteres Problem der Digest-Authentifizierung besteht darin, dass die Benutzerkonten in der Kontendatenbank rekonfiguriert werden müssen, damit die Kennwörter in verschlüsselter Form gespeichert werden können.

Auch der Grundauthentifizierungsmodus ist standardmäßig gesperrt. Der Benutzer wird hier wie bei der Digest-Authentifizierung aufgefordert einen Benutzernamen und ein Kennwort einzugeben. Der Browser überträgt den Benutzernamen und das Kennwort im HTTP-Authentifizierungs-Header. Dazu wird ein Base64/UUEncoded-Codierungsschema verwendet. Dieses Codierungsschema wurde nicht entwickelt, um Nachrichten zu schützen, so dass Sie davon ausgehen müssen, dass das Kennwort im gewöhnlichen Textformat über das Internet versendet wird. Sie sollten den Grundauthentifizierungsmodus deshalb nur mit HTTPS benutzen. Wenn Sie HTTPS nicht verwenden, ist Ihr Kennwort jedem Benutzer zugänglich.

Die Stärken der Grundauthentifizierung bestehen darin, dass sie ein etablierter HTTP-Standard ist, der von den meisten Browsern unterstützt wird. Die Grundauthentifizierung kann durch Firewalls und über Proxyserver hinweg ausgeführt werden und erfordert keinen Konfigurationsaufwand. (Dies gilt besonders, wenn sie mit der Zertifikat-Authentifizierung verglichen wird.)

Sie sollten eine weitere interessante Besonderheit beachten, die den Einsatz der Grundauthentifizierung mit IIS und ASP betrifft. Sie können über ASP-Skripte und Visual Basic-Komponenten auf das Kennwort zugreifen, indem Sie die `AUTH_PASSWORD`-Variable in der `ServerVariables`-Auflistung benutzen. Sie können diese Option nicht deaktivieren, so dass Sie den Personen, die Skripte und Komponenten schreiben, sehr vertrauen müssen. Ein Entwickler könnte zum Beispiel Benutzernamen- und Kennwortpaare in eine Protokolldatei schreiben und diese Informationen verwenden, um eine Attacke zu starten.

Welchen der fünf Authentifizierungsmodi sollten Sie also für Ihre Applikation auswählen? Die Antwort auf diese Frage ist von einigen Faktoren abhängig. Jeder Modus weist gewisse Stärken und Schwächen auf. Die Zertifikat-Authentifizierung und die Grundauthentifizierung werden vorwiegend verwendet, um Clients über das Internet zu authentifizieren.

Sie haben vielleicht den Eindruck gewonnen, dass der anonyme Zugriff ausschließlich für Situationen geeignet ist, in denen die Identität des Clients nicht wichtig ist. Dies ist jedoch nicht immer so. Sie werden Ihre Clients bisweilen auch dann authentifizieren wollen, wenn es nicht sehr effizient ist, ein Windows-Benutzerkonto für jeden Client zu pflegen. Dies ist häufig der Fall, wenn eine Applikation einer sehr großen Anzahl von Benutzern gerecht werden muss. Wenn Sie Ihre Applikation für tausende oder zehntausende von Benutzern skalieren möchten, können Sie ein Microsoft-Produkt namens Site Server benutzen, das die Überwachung von Benutzerprofilen in einer benutzerdefinierten Datenbank ermöglicht. Eine andere Option besteht darin, ein benutzerdefiniertes Sicherheitsmodell manuell zu erstellen.

Sie können zum Beispiel einen sicheren Kanal erstellen, wenn Sie den anonymen Zugriff mit HTTPS verwenden. Ein Client kann dann einen Benutzernamen und ein Kennwort sicher über das Internet versenden. Ihre Applikation könnte eine Anmeldungsroutine zur Verfügung stellen, um die benutzerdefinierte Authentifizierung durchzuführen. Dazu würde eine applikationsspezifische Datenbank durchsucht, um den Benutzernamen und das Kennwort zu überprüfen. Sie könnten nun einen benutzerdefinierten Sitzungsschlüssel generieren, den Sie mit Hilfe von HTTP-Cookies übermitteln. Ein solches benutzerdefiniertes Sicherheitsgerüst erfordert sehr viel Code, bietet aber eine außerordentliche Flexibilität und Skalierbarkeit.

IIS-Authentifizierung mit rollenbasierter Sicherheit verwenden

Das Authentifizieren des Clients genügt jedoch nicht. Um Ihre Applikation zu sichern, müssen Sie Ihren .ASP-Dateien und konfigurierten Komponenten die richtigen Zugriffsberechtigungen zuweisen. Wenn Ihre .ASP-Dateien (und andere darauf bezogene Dateien) auf einer NTFS-Partition gespeichert sind, können Sie sie konfigurieren, um Benutzern, Gruppen und Alias-Konten Lese-Berechtigungen oder Lese-/Ausführungs-Berechtigungen zu gewähren respektive zu verweigern. Es ist relativ einfach, Zugriffsberechtigungen für einzelne Dateien und Verzeichnisse mit dem Windows Explorer zu konfigurieren.

Wenn Sie jedoch Zugriffsberechtigungen auf der Datei- und Verzeichnisebene konfigurieren, können Sie nicht die Vorteile der rollenbasierten Sicherheit nutzen. Sie erinnern sich bestimmt, dass die rollenbasierte Sicherheit sehr nützlich ist, da sie Ihnen erlaubt, die Autorisierung vorab in einer COM+-Applikation zu konfigurieren. Die COM+-Applikation kann anschließend in viele Betriebsumgebungen übernommen werden. Vergessen Sie auch nicht, dass die programmgesteuerte Seite der rollenbasierten Sicherheit die Vorteile bedingter Autorisierungsüberprüfungen und der benutzerdefinierten Überwachung bietet.

Es gibt drei Möglichkeiten zum Konfigurieren einer COM+-Applikation, die von ASP-Seiten-Clients benutzt wird: Sie können sie als Bibliotheksapplikation, als lokale Serverapplikation oder als Remote-Serverapplikation konfigurieren. Lassen Sie uns die Vor- und Nachteile dieser Möglichkeiten betrachten.

Wenn Sie eine Bibliotheksapplikation benutzen, werden Ihre Objekte auf denselben STA-Thread wie die ASP-Seite geladen, die die Objekte erstellt hat. COM+ fügt jedoch einen einfachen Proxy zwischen der ASP-Seite und dem Objekt ein. Dies erlaubt COM+, die deklarativen Autorisierungsüberprüfungen immer dann auszuführen, wenn eine ASP-Seite einen Aufruf an ein Objekt vornimmt. Sie sollten beachten, dass diese Autorisierungsüberprüfungen mit einer Anmeldungssitzung und einem Token ausgeführt werden, die gemäß eines IIS-Authentifizierungsmodus erstellt wurden. Der Vorteil einer Bibliotheksapplikation besteht darin, dass sie die beste Performance bietet.

Wenn Sie eine lokale Serverapplikation benutzen, werden Ihre Objekte in eine private und lokale Instanz von *DLLHOST.EXE* geladen. COM+ fügt auch hier eine Abfangschicht zwischen der ASP-Seite und dem Objekt ein, um rollenbasierte Autorisierungsüberprüfungen durchführen zu können. Das Ausführen von Autorisierungsüberprüfungen in *DLLHOST.EXE* erfordert das Erstellen eines weiteren Tokens. Beide Prozesse befinden sich jedoch auf demselben Computer, so dass eine zweite Anmeldungssitzung nicht notwendig ist.

Betrachten Sie noch einmal Abbildung 11.13. Die oberste Anforderung wird von *INETINFO.EXE* an *DLLHOST.EXE* übergeben. Beachten Sie, dass der Identitätswechsel der Benutzeridentität erlaubt, sich von einem Serverprozess auf einen anderen auszuweiten. Wenn IIS den Client beispielsweise als Bob authentifiziert, wird die ASP-Seite in *INETINFO.EXE* unter der Identität von Bob ausgeführt. Wenn der Aufruf an *DLLHOST.EXE* vorgenommen wird, können die Autorisierungsüberprüfungen ebenfalls mit Bobs Token durchgeführt werden. Dies ist aufgrund eines neuen Features in Windows 2000 möglich, das als dynamisches Cloaking bezeichnet wird. (Unter Windows NT und IIS4 steht Ihnen dieses Feature nicht zur Verfügung.)

Sie haben bisher Optionen kennen gelernt, die Bibliotheksapplikationen und lokale Serverapplikationen verwenden. Ich werde versuchen, Ihnen die nachfolgend beschriebene dritte Option auszureden. Wenn Sie eine Remote-Serverapplikation benutzen möchten, werden Sie wahrscheinlich sehr bald äußerst frustriert sein. Das Durchführen von Autorisierungsüberprüfungen auf einem Computer, der kein Webserver ist, erfordert das Verteilen von Anmeldungssitzungen über das Netzwerk hinweg. Dies kann eine Herausforderung oder sogar nicht durchführbar sein.

Das Problem besteht darin, dass die meisten von IIS verwendeten Authentifizierungsmodi eine Anmeldungssitzung ohne Netzwerkinformationen erstellen. Die Anmeldungssitzung enthält somit kein Kennwort und auch keine anderen Identitätsbestätigungen. Ohne diese Netzwerkinformationen kann der Webserver eine Anmeldungssitzung nicht an ein anderes Gerät weitergeben. Ein Delegieren ist somit nicht möglich.

Es gibt viele Möglichkeiten, den von IIS verwendeten Authentifizierungsmodus zu konfigurieren. Die meisten dieser Möglichkeiten führen jedoch zu keiner Anmeldungssitzung mit Netzwerkinformationen. Wenn Sie sich jedoch bemühen, können Sie mit Hilfe der Grundauthentifizierung, der Zertifikat-Authentifizierung oder mit Hilfe von Kerberos Anmeldungssitzungen delegieren.

Sie sollten jedoch darüber nachdenken, auf das Delegieren zu verzichten, da diese Methode der Praxis widerspricht, möglichst nahe am Client zu authentifizieren und zu autorisieren. Das Delegieren ist sehr aufwendig, da die Client-Identität vom Webserver abwärts an andere Server weitergegeben werden muss. Dies führt zu einer Leistungseinbuße. Versuchen Sie, die Authentifizierung und Autorisierung immer auf dem Webserver selbst durchzuführen, sofern dies möglich ist.

Weitere Quellen

Die Sicherheit ist ein bisweilen entmutigendes Thema. Ich muss leider gestehen, dass dieses Kapitel lediglich die Spitze des Eisberges ist. Es gibt zu den Bereichen Domänenadministration, Zertifikatvergabe und Verschlüsselung sehr viele weitere Informationen. Sie sollten außerdem wissen, welche IIS-Attacken üblich sind, so dass Sie den »bösen Jungs« immer ein wenig voraus sind. Ich muss mich deshalb entschuldigen, weil ich Sie mit sehr vielen Hausaufgaben zurücklasse.

Wenn Sie die administrative Seite von IIS und MCS meistern möchten, sollten Sie die Dokumentation lesen, die mit diesen Produkten ausgeliefert wird. Das Windows 2000-Ressource-Kit ist ebenfalls eine äußerst nützliche Ressource. Es beschreibt detailliert Active Directory und behandelt ebenfalls IIS.

Wenn Sie wirklich die Interna des Windows-Sicherheitsmodells verstehen möchten und sicherheitsbezogenen Code in C++ schreiben, sollten Sie »Programming Windows Security« von Keith Brown lesen. Dieses Buch ist auch dann sehr nützlich, wenn Sie nicht C++-Code-Beispiele lesen möchten, da es die Kernkonzepte der Windows-Sicherheit beschreibt.

Zusammenfassung

Das Kapitel begann mit einer Diskussion über die Kernkonzepte der Windows-Sicherheit. Alles basiert auf Hauptnamen. Autoritäten sind Dienste, die einem Hauptnamen gestatten, einen anderen Hauptnamen zu authentifizieren. Anmeldungssitzungen und Tokens sind Datenstrukturen, die zur Laufzeit erstellt werden, um Authentifizierungsinformationen über Hauptnamen zwischenspeichern. Jeder Prozess erhält ein Prozess-Token, das den Prozess mit einem Hauptnamen verknüpft und die Prozessidentität einrichtet. Eine DACL ist eine konfigurierbare Liste mit Zugriffsberechtigungen, die das Ausführen von Autorisierungsüberprüfungen für ein Hauptnamens-Token zur Laufzeit ermöglicht.

Die Netzwerk-Authentifizierung wird von Plug-In-Modulen durchgeführt, die auch unter der Bezeichnung SSP bekannt sind. Kerberos und NTLM sind SSPs. Diese Provider stellen die Authentifizierungsdienste für solche Computer zur Verfügung, die innerhalb eines privaten Netzwerks

miteinander kommunizieren. Die Authentifizierungsebene bestimmt die Intensität der Aktivität eines Authentifizierungsdienstes. Sie sollten immer möglichst nahe am Client authentifizieren und autorisieren. Dies bedeutet, dass Sie COM+-Applikationsdesigns vermeiden sollten, die den Identitätswechsel benutzen oder Anmeldungssitzungen delegieren.

Das COM+-Sicherheitsmodell basiert auf Rollen. Rollen trennen Zugriffsberechtigungen von SIDs. Dies vereinfacht den Einsatz einer einzelnen Applikation in mehreren Betriebsumgebungen. Sie können deklarative rollenbasierte Autorisierungsüberprüfungen vorab konfigurieren und Code schreiben, um bedingte Autorisierungsüberprüfungen durchzuführen und die Benutzeraktivität zu überwachen.

Wenn Sie eine Webapplikation erstellen, sollten Sie lernen, wie sich IIS in die COM-Sicherheit integriert. IIS verwendet den Identitätswechsel, um zu vermeiden, dass Client-Anforderungen unter der Identität des Systems ausgeführt werden. Das Benutzerkonto, dessen Identität von IIS angenommen wird, wird mit Hilfe des IIS-Authentifizierungsmodus bestimmt. Die Zertifikat-Authentifizierung und die Grundauthentifizierung sind zwei gebräuchliche Verfahren, um den Windows-Benutzerkonten bestimmte Internet-Benutzer zuzuweisen. Es ist jedoch nicht üblich, Anforderungen unter der Identität des anonymen Internet-Benutzerkontos (das standardmäßig *IUSR_Machine* ist) auszuführen.

Sobald ein Client authentifiziert ist, bestimmen die von Ihnen konfigurierten Zugriffsberechtigungen, welche Rechte der Client besitzt. Wenn Sie die rollenbasierte Sicherheit wünschen, sollten Sie konfigurierte Komponenten benutzen, die in einer Bibliotheksapplikation oder einer Serverapplikation installiert sind. Sie sollten jedoch keine Remote-Serverapplikation einsetzen. Das Ausführen von Autorisierungsüberprüfungen auf einem Applikationsserver, der kein Webserver ist, kann sehr aufwendig sein, da es den Identitätswechsel und möglicherweise das Delegieren von Anmeldungssitzungen erfordert. Sie sollten Autorisierungsüberprüfungen immer auf dem Webserver selbst durchführen.

12 Skalierbare Applikationen entwerfen

379	Skalierbarkeit
382	HTTP in COM+-Applikationen verwenden
390	Applikationsdesign
405	Zusammenfassung

Sie haben in diesem Buch viele COM- und COM+-Details kennen gelernt. Da dies das letzte Kapitel des Buches ist, sollten wir versuchen, all diese Details und deren Beziehung zueinander in einem Zusammenhang zu betrachten. Es ist wichtig, die Details zu kennen. Es ist jedoch ebenso wichtig zu wissen, wie all diese kleinen Puzzle-Teile ineinander greifen. Wie nutzen Sie die Informationen, die Sie in diesem Buch erhalten haben, um eine Applikation, die Ihren Anforderungen gerecht wird, in die Betriebsumgebung zu übernehmen? Ich werde Ihnen in diesem Kapitel einige Einsatzmöglichkeiten präsentieren und deren Auswirkungen auf die Skalierbarkeit erläutern.

Ein weiteres Thema dieses Kapitels beschreibt die Auswahl eines geeigneten Protokolls. Das Arsenal eines COM+-Entwicklers bietet DCOM, HTTP und MSMQ an. Jedes Protokoll weist besondere Stärken und Schwächen auf. Sie sollten die Funktionalität, die von diesen Protokollen angeboten wird, verstehen, um problemlos damit arbeiten zu können.

Ich werde ebenfalls die Techniken für die Übergabe von Daten über das Netzwerk beschreiben. Das Übergeben von Daten an den Server und zurück an den Client ist ein wesentlicher Aspekt jeder verteilten Applikation. Das Kapitel schließt mit einigen Worten zu verschiedenen Themen, wie zum Beispiel das Übermitteln von aktualisierbaren Datensatzgruppen und das Zwischenspeichern von Daten in der mittleren Schicht.

Skalierbarkeit

Der Begriff Skalierbarkeit ist mit mehreren Bedeutungen verknüpft. Skalierbarkeit ist einerseits eine Größe, die Aufschluss über die Anzahl der Benutzer gibt, die gleichzeitig von Ihrer Applikation bedient werden können. Andererseits kann Skalierbarkeit ein Indikator dafür sein, ob Ihre Applikation die Verfügbarkeitsanforderungen einer kritischen Geschäftsapplikation erfüllt. Sie können Skalierbarkeit außerdem als die Fähigkeit Ihrer Applikation definieren, Benutzer zu bedienen, die sich außerhalb des LANs Ihres Unternehmens befinden und/oder kein entsprechendes Benutzerkonto in der Windows 2000-Domäne Ihres Unternehmens besitzen. Ein Webentwickler könnte Skalierbarkeit als die Fähigkeit einer Applikation definieren, jeden Benutzer des Internets zu erreichen. Sie entscheiden, welche Aspekte der Skalierbarkeit nützlich sind und von Ihrer Applikation erfüllt werden sollten.

Wie vielen Benutzern müssen Sie heute gerecht werden? Wie viele Benutzer wird Ihre Applikation in Zukunft bedienen müssen? Wie viele Anforderungen verschickt jeder Benutzer durchschnittlich an den Server? Indem Sie diese Fragen beantworten, sollten Sie annähernd bestimmen können, mit welchem Datenverkehr Sie rechnen müssen. Wenn Sie hunderte oder tausende von Anforderungen pro Sekunde bearbeiten müssen, sollten Sie eine Applikation entwerfen, die solche Mengen bewältigen kann.

Sie sollten ebenfalls darüber nachdenken, wo sich Ihre Benutzer befinden. Kann Ihre Applikation den Benutzern gerecht werden, die sich außerhalb eines administrierten LANs befinden? Vielleicht wird Ihre Applikation von Angestellten verwendet, die über ISDN, DSL oder Kabelmodems kommunizieren. Möglicherweise muss Ihre Applikation Internetbenutzer berücksichtigen, die über MSN eine Verbindung zu Benutzern auf der anderen Seite des Planeten herstellen möchten.

Erreicht Ihre Applikation Benutzer, die verschiedene Betriebssysteme und Browser ausführen? Ist Ihre Applikation ausreichend flexibel und zuverlässig, um ein Szenario zu unterstützen, in dem der Systemadministrator Ihres Unternehmens die Computer Ihrer Benutzer möglicherweise nicht konfigurieren kann? Wenn Sie jeden Internetbenutzer erreichen möchten, werden Sie mit Vor- und Nachteilen konfrontiert, die die Technologien der clientseitigen Computer betreffen.

Die »ultimative« skalierbare Applikation kann für Millionen von Internetbenutzern gleichzeitig Transaktionen ausführen und Daten abrufen. Jeder Benutzer kann ein beliebiges Betriebssystem ausführen und erfordert keine benutzerdefinierte Konfiguration. Dies ist ein sehr hoher Anspruch, den die meisten Applikationen in der Regel nicht erfüllen müssen. Sie sollten die Kosten kennen, die mit den verschiedenen Aspekten der Skalierbarkeit verknüpft sind. Sie müssen dann nicht für etwas bezahlen, das Sie nicht benötigen. Es ist deshalb wichtig, dass Sie Ihre Anforderungen an die Skalierbarkeit frühestmöglich während der Entwurfsphase definieren.

Bevor Sie eine verteilte Applikation entwickeln, sollten Sie entscheiden, wie Clients mit Ihren Servern kommunizieren sollen. Welches zugrunde liegende Protokoll wird den Clients erlauben, Anforderungen an den Server zu schicken – DCOM, HTTP oder MSMQ? Ihre Wahl wird sich entscheidend auf die Skalierbarkeit Ihrer Applikation auswirken.

Wir haben in diesem Buch die interne Arbeitsweise von DCOM, HTTP und MSMQ untersucht. Wie Sie wissen, sind diese Protokolle die Bausteine einer verteilten Applikation. Jedes Protokoll weist Vor- und Nachteile auf. Lassen Sie uns betrachten, wie sich jedes dieser Protokolle in das Design großer Applikationen fügt.

Die ursprüngliche COM-Vision

In den neunziger Jahren wurde COM zu einer sehr wichtigen Technologie für das Erstellen komponentenbasierter Applikationen. Es wurde von vielen Microsoft-Produktgruppen übernommen, und viele Windows-Entwickler folgten diesem Beispiel. Beinahe jeder Windows-Entwickler verwendet COM deshalb in der einen oder anderen Form.

Viele Entwickler glaubten, dass DCOM zu einem der führenden Industrie-Applikationsprotokolle für die Client-/Serverkommunikation werden würde. Diese Entwickler teilten damals die Vision, dass große Applikationen mit Hilfe eines physikalischen Modells wie dem in Abbildung 12.1 dargestellten erstellt werden könnten.

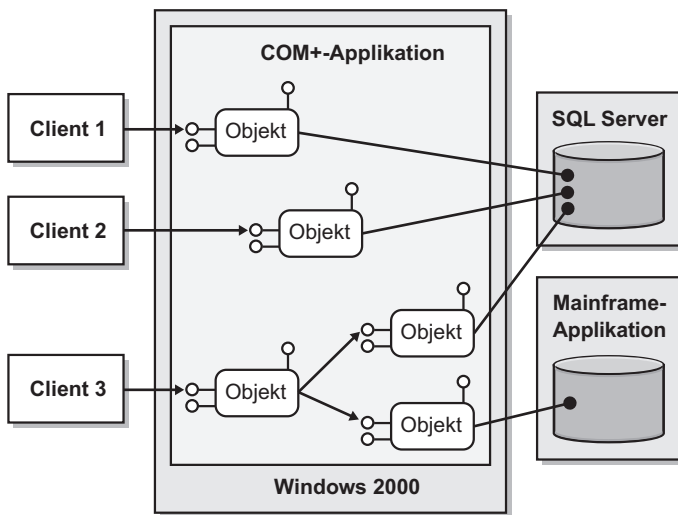


Abbildung 12.1: Eine Applikation, die DCOM für die Client-/Serverkommunikation verwendet, kann nicht einfach über ein LAN hinaus skaliert werden. DCOM erschwert außerdem die Skalierung einer Applikation für tausende von Benutzern.

COM ist ortsunabhängig, da es ein transparentes Protokoll (DCOM) anbietet. Dieses Protokoll vereinfacht das Einrichten einer Remote-Verbindung zwischen einem Client-Computer und einem Servercomputer. COM ist somit gewissermaßen ein Einstiegspunkt in die Welt der verteilten Programmierung. Dieser Umstand verleitet viele Personen zu der Annahme, dass DCOM offensichtlich dem Zweck diene, mehrschichtige Applikationen zu erstellen.

Als sich die neunziger Jahre dem Ende zuneigten, wurde deutlich, dass DCOM hinsichtlich der Skalierung großer Applikationen mit zu vielen Einschränkungen versehen ist. DCOM kann beispielsweise die Vorteile von Lastenausgleichsprodukten, wie sie zum Beispiel NLB von Microsoft oder LocalDirector von Cisco bieten, nicht in demselben Maße wie HTTP nutzen. DCOMs verbindungsorientierte Natur verhindert darüber hinaus den anforderungsbasierten Lastenausgleich, wodurch die Anzahl der gleichzeitigen Benutzer begrenzt und eine Applikation davon abgehalten wird, die bestmögliche Fehlertoleranz und Verfügbarkeit zu erzielen.

Damit ist die Liste der DCOM-Nachteile jedoch noch nicht vollständig. DCOM verträgt sich nicht sehr gut mit Firewalls, und sein Sicherheitsmodell ist nicht für öffentliche Netzwerke wie zum Beispiel das Internet geeignet. DCOM erfordert außerdem eine Windows-Version und eine umfassende benutzerdefinierte Konfiguration und Administration auf jedem Client-Computer.

All diese Probleme mit DCOM führen zu einer Folgerung: Während DCOM für eine kleine bis mittlere Benutzergruppe in einem LAN geeignet ist, wird es zunehmend problematischer und unzuverlässiger, je mehr Clients zu dem Netzwerk hinzugefügt werden oder wenn die Clients weit entfernt vom Server angeordnet werden.

Ich hoffe, dass ich Sie nicht enttäuscht habe. Schließlich habe ich COM als wichtige Technologie bezeichnet und Sie dazu veranlasst, hunderte von Seiten mit COM-Details zu studieren. Ich werde Ihnen nicht plötzlich empfehlen, COM nicht zu verwenden. Wir sollten jedoch einen Schritt zurückgehen und die Geschehnisse von einer anderen Perspektive aus betrachten.

COM und COM+ weisen vier Besonderheiten auf:

- Sie ermöglichen Ihnen, Applikationen aus solchen Komponenten zusammenzustellen, die in einer Sprache Ihrer Wahl geschrieben wurden.
- COM und COM+ bieten darüber hinaus eine Laufzeitumgebung an, die Systemdienste über Kontexte und die attributbasierte Programmierung zur Verfügung stellt.

- COM und COM+ ermöglichen außerdem Aufrufe zwischen den Prozessen, die auf demselben Computer ausgeführt werden. Ein Beispiel hierfür ist eine ASP-Seite, die eine Methode eines Objekts aufruft, das wiederum in einer COM+-Serverapplikation ausgeführt wird.
- Schließlich wird ein Netzwerkprotokoll (DCOM) für die Kommunikation über Computergrenzen hinweg zur Verfügung gestellt.

Vielleicht entscheiden Sie, dass DCOM nicht notwendig für Ihre Applikation ist. Die ersten drei Aspekte von COM und COM+ sind jedoch nützlich für Sie.

HTTP in COM+-Applikationen verwenden

HTTP ist zu dem führenden Industrieprotokoll für die Client-/Serverkommunikation geworden. Ein Schlüssel zum Erfolg dieses Protokolls ist, dass es einfach zu implementieren ist und jede wichtige Plattform unterstützt. HTTP ist allgegenwärtig und somit sehr nützlich.

Das Phänomen des weltweiten Netzwerks (World Wide Web) wirkt sich besonders darauf aus, wie wir verteilte Applikationen entwickeln. Die gesamte Industrie richtet ihre Aufmerksamkeit gegenwärtig auf Technologien für Webserver und Browser. Unternehmen investieren Millionen in die Forschung und Entwicklung, um die Fähigkeiten von HTTP in vielen wichtigen Bereichen, wie zum Beispiel denen des Lastenausgleichs und der Sicherheit zu erweitern.

Microsoft hat, wie viele andere Anbieter auch, die strategische Bedeutung von HTTP erkannt. Die Architekten von COM+ haben ihre Plattform auf der folgenden Voraussetzung errichtet: Eine Applikation muss HTTP für die Client-/Server-Kommunikation benutzen, um die beste Skalierbarkeit zu erzielen.

Abbildung 12.2 zeigt eine COM+-Applikation mit einer skalierbaren Konfigurationsarchitektur. Beachten Sie, dass diese Applikation nicht DCOM benutzt. Die Clients schicken stattdessen Anforderungen mit HTTP an IIS. Die Web-Farm spielt eine zentrale Rolle in dieser Architektur, da sie eine COM+-Applikation hunderte von Anforderungen pro Sekunde bearbeiten lässt und eine bessere Fehlertoleranz und Verfügbarkeit bietet.

Diese Architektur stellt die Mittel für eine Auswärtsskalierung (scale out) zur Verfügung. Sie können einer zunehmenden Zahl von Clients gerecht werden, indem Sie zusätzliche Computer zur Web-Farm hinzufügen. Wie bereits im neunten Kapitel erwähnt wurde, bietet eine Web-Farm, die den anforderungsbasierten Lastenausgleich verwendet, eine bessere Fehlertoleranz und Verfügbarkeit. Das neunte Kapitel hat außerdem einige wichtige Aspekte genannt, die Sie berücksichtigen sollten, wenn Sie ASP-Seiten und Komponenten für eine Web-Farm-Umgebung mit einem anforderungsbasierten Lastenausgleich entwickeln. Viele Techniken, die ASP-Session-Objekte und andere Cache-Techniken für die mittlere Schicht benutzen, führen jedoch nicht zu dem erwarteten Ergebnis. Dies ist der Preis, den Sie zahlen müssen, wenn Sie großangelegte Applikationen erstellen möchten.

Abbildung 12.2 zeigt, dass sich der Datenbankserver nicht in derselben Weise wie die Server, die IIS- und COM+-Applikationen ausführen, für die Auswärtsskalierung eignet. Der Datenbankserver-Computer verursacht in den meisten Fällen Konflikte. Dies bedeutet, dass Sie eine Aufwärtsskalierung (scale up) und nicht eine Auswärtsskalierung für den Datenbankserver durchführen sollten. Um einer zunehmenden Zahl von Benutzern gerecht zu werden, sollten Sie einen Multiprozessor-Computer erwerben, der eine angemessene Zahl von Verarbeitungszyklen und ausreichend Speicher und Festplattenkapazität anbietet.

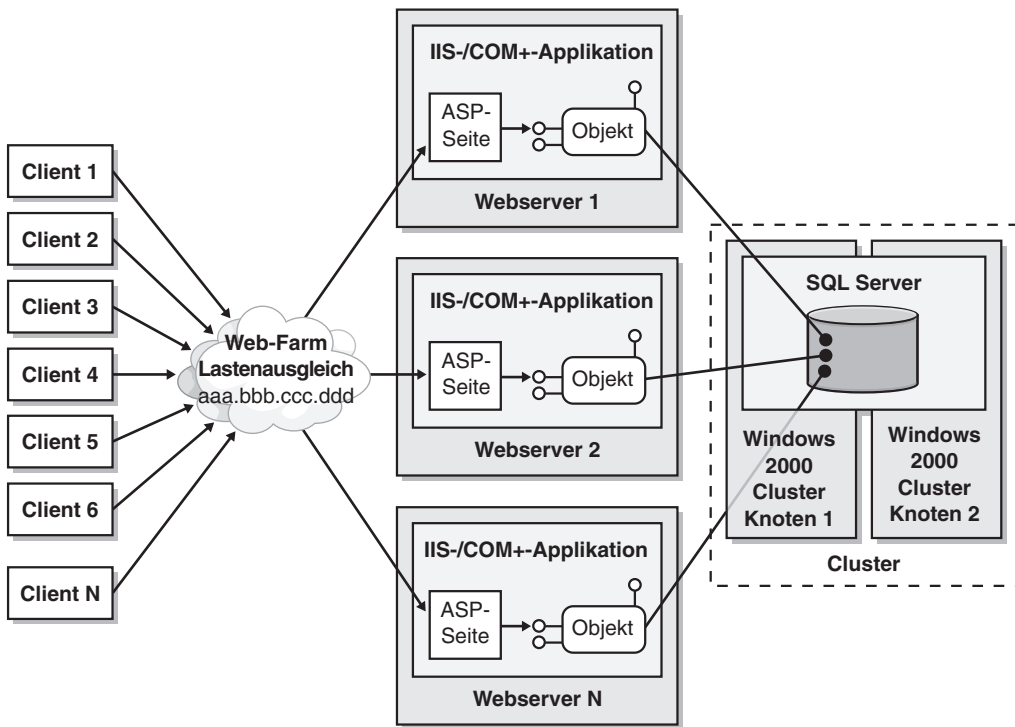


Abbildung 12.2: Eine umfassend skalierbare Architektur für eine COM+-Applikation verwendet eine Web-Farm-Infrastruktur und den anforderungsbasierten Lastenausgleich. Diese Architektur zwingt Sie, HTTP für die Client-/Server-Kommunikation zu verwenden.

Sie erkennen an diesem Beispiel, dass eine Auswärtsskalierung für Webserver, aber nicht für Datenbankserver möglich ist. Sie sollten daraus schließen, dass ein großer Teil der Verarbeitung auf den Webserver verlagert werden sollte, sofern dies möglich ist. Suchen Sie nach Möglichkeiten, bestimmte Aufgaben, wie zum Beispiel sicherheitsbezogene Autorisierungsüberprüfungen, vom Datenbankserver auf die Webserver der Farm zu verlagern.

Der Datenbankserver ist ebenfalls für Fehler anfällig. Wenn der Datenbankserver abstürzt, ist Ihre Applikation blockiert. Sie können darauf reagieren, indem Sie Cluster-Technologien verwenden. Cluster Service ist ein Windows 2000-Dienst, dessen primäre Aufgabe darin besteht, eine ausfallübergreifende Unterstützung für Applikationen wie zum Beispiel Datenbank- und Nachrichtensysteme zur Verfügung zu stellen. Cluster Service gestattet mehreren physikalischen Computern, die Identität eines einzelnen Computers anzunehmen. Jeder Computer ist in diesem Fall ein Knotenpunkt, der die Steuerung übernimmt, sobald ein anderer Knoten ausfällt. Cluster Service unterstützt ausfallübergreifende Cluster für zwei Knoten in Windows 2000 Advanced Server. Unter Windows 2000 Datacenter Server werden vier Knoten unterstützt.

Im zehnten Kapitel wurde beschrieben, dass Sie auf Probleme mit der Fehlertoleranz und Verfügbarkeit reagieren können, indem Sie das Messaging verwenden. Sobald eine HTTP-Anforderung den Webserver erreicht, können Sie entweder MSMQ oder QC benutzen, um die Client-Anforderung abzufangen. Eine Applikation bleibt auf diese Weise auch dann betriebsbereit, wenn der Datenbankserver offline ist. Das Messaging kann darüber hinaus die Reaktionszeiten verkürzen, wenn der Datenbankserver sehr viele Anforderungen bearbeiten muss und in Verzug gerät. Sie

sollten auch verstehen, dass MSMQ nicht darauf beschränkt ist, Nachrichten zwischen verschiedenen Computern zu versenden. Sie können MSMQ-Nachrichten ebenfalls zwischen zwei Applikationen verschicken, die auf demselben Webserver-Computer ausgeführt werden. Die dazu notwendige Vorgehensweise ist recht einfach und bietet die Vorteile der asynchronen Ausführung sowie der Einmal-Übermittlung.

Microsoft wird voraussichtlich ein Produkt namens Application Center Ende des Jahres 2000 veröffentlichen. AppCenter Server ist eine Produktsammlung für Unternehmen, die Applikationen in eine Server-Farm-Umgebung installieren und dort administrieren. AppCenter Server wird eine neue Unterstützung für die Administration und Überwachung von NLB und Cluster Service sowie eine dritte Form des Lastenausgleichs enthalten. Dieser Lastenausgleich wird mit dem CLB-Dienst (Component Load Balancing) durchgeführt werden.

CLB verteilt mehrere COM-Clients über einen Cluster mit Servercomputern. Jeder Servercomputer in dem Cluster wird gewöhnlich mit denselben COM+-Applikationen konfiguriert. CLB basiert auf einem Lastenausgleichsalgorithmus, der ausgeführt wird, wenn ein Client ein COM-Objekt aktiviert. CLB ist somit eine sitzungsbasierte und nicht eine anforderungsbasierte Lastenausgleichstechnik.

In einer Applikation, die CLB benutzt, senden alle Clients Aktivierungsanforderungen an einen zentralen Routing-Server. Der Routing-Server agiert als ein Objektvermittler, da er jede Aktivierungsanforderung an einen der COM+-Servercomputer im Cluster weiterleitet. Das administrative Hilfsmittel *Komponentendienste* enthält bereits verschiedene Konfigurationsoptionen für CLB, da das COM+-Team ursprünglich davon ausging, dass CLB mit der ersten Version von Windows 2000 veröffentlicht werde.

Beachten Sie, dass CLB auf DCOM basiert und somit dieselben Einschränkungen aufweist. CLB ist deshalb vorwiegend für LAN-Umgebungen geeignet. Der von Produkten wie zum Beispiel NLB und Ciscos LocalDirector angebotene anforderungsbasierte Lastenausgleich eignet sich deshalb besser für eine verteilte Applikation.

HTTP von HTML trennen

Lassen Sie uns noch einmal die Anforderungen für die Skalierung einer Applikation betrachten. Nachfolgend sind einige übliche Anforderungen an großangelegte Applikationen aufgeführt:

- Die Applikation muss tausende oder zehntausende von Benutzern gleichzeitig bedienen können.
- Die Applikation muss eine ausgeprägte Fehlertoleranz und Verfügbarkeit bieten.
- Ein Client muss durch eine Firewall auf die Applikation zugreifen können.
- Internetbenutzer müssen ebenfalls auf die Applikation zugreifen können.
- Benutzer, die nicht an Ihr Unternehmen gebunden sind, müssen sicher auf die Applikation zugreifen können.
- Benutzer müssen auf die Applikation zugreifen können, ohne einen Systemadministrator zu benötigen, der ihre Computer konfiguriert.
- Die Applikation muss auch solchen Benutzern zur Verfügung stehen, die andere Betriebssysteme als Windows, wie zum Beispiel Linux oder Macintosh verwenden.

Wie Sie gesehen haben, erfüllt HTTP diese Anforderungen. Der Einsatz von HTML mit HTTP ist jedoch nicht erforderlich. Nur die letzte und, in geringerem Maße, die vorletzte Anforderung zwingen Sie, HTML zu benutzen.

Fragen Sie sich, ob Ihre Applikation wirklich HTML benötigt. Wenn Sie eine E-Commerce-Site erstellen und jeden Benutzer im Internet erreichen möchten, ist HTML erforderlich. Wenn Sie eine Geschäftsapplikation für Ihre Angestellten, Lieferanten oder Kunden generieren, ist HTML in der Regel optional.

Nichts im skalierbaren und in Abbildung 12.2 dargestellten Konfigurationsmodell zwingt Sie, HTML zu benutzen. Wenn Sie voraussetzen können, dass Ihre Benutzer eine 32-Bit-Version von Windows ausführen lassen, besitzen Sie einige Alternativen. Die Präsentationsschicht Ihrer Applikation könnte zum Beispiel eine traditionelle Desktop-Applikation sein, die mit Visual Basic erstellt wird. Sie könnten somit den Visual Basic-Formulardesigner benutzen. Der einzige Trick, den Sie erlernen müssten, ist das Schreiben von Code in Visual Basic, um eine HTTP-Anforderung an IIS zu schicken. Wie Sie bald sehen werden, ist dies nicht besonders schwierig.

Sie sollten sich ebenfalls fragen, ob Sie mit HTML oder mit Visual Basic eine bessere Benutzeroberfläche erzeugen können. Meine Erfahrung ist, dass HTML-Applikationen (auch solche, die DHTML und clientseitige Java-Skripte verwenden) nicht die Eleganz und Bedienerfreundlichkeit der meisten Visual Basic-Applikationen bieten. Werden Sie wirklich die Entwicklung von HTML-Oberflächen für einen Browser vermissen, wenn Sie Ihre Client-Applikation mit Visual Basic erstellen? Ich kenne sehr viele Entwickler, die diese Frage verneinen würden.

Visual Basic vereinfacht das Erstellen von Formularen, um Benutzereingaben entgegenzunehmen und zu prüfen. Viele Entwickler haben sich daran gewöhnt, Grid- und Strukturansicht-Steuer-elemente zu verwenden und den Benutzern tabulatorgesteuerte Dialogfelder zu präsentieren. HTML und Browser bieten diese Features in der Regel nicht an. Wenn Sie die Einschränkungen von HTML umgehen möchten, indem Sie ActiveX-Steuer-elemente für den Microsoft Internet Explorer zur Verfügung stellen, müssen Ihre Benutzer eine 32-Bit-Version von Windows verwenden. Sie könnten jedoch ebenfalls den nächsten logischen Schritt gehen und Visual Basic benutzen, um die gesamte Client-Applikation zu generieren.

Denken Sie an die zusätzlichen Komplikationen, die sich ergeben, wenn Sie mit Browsern arbeiten. Wenn der clientseitige Status verwaltet werden soll, müssen Sie zwischen Cookies, Abfragezeichenfolgen und versteckten Formularen wählen. Soll eine Visual Basic-Applikation auf dem Desktop des Benutzers ausgeführt werden, ist die Statusverwaltung sehr viel einfacher. Darüber hinaus ist es einfacher, eine ADO-Datensatzgruppe oder ein XML-Dokument auf der Festplatte des Benutzers zu speichern. Sie müssten sich auch nicht mit der geringen Sicherheit eines Browsers beschäftigen.

Ich spreche nicht davon, dass es HTML und die Browser-Entwicklung bald nicht mehr gibt. HTML ist noch immer eine der besten Möglichkeiten, die Probleme der plattformübergreifenden Entwicklung zu lösen und die clientseitige Konfiguration durchzuführen. Sie entscheiden jedoch, wann HTML erforderlich ist und wann Sie darauf verzichten können.

Lassen Sie mich noch einen Aspekt an dieser Stelle erwähnen, der mehr politisch als technisch ist. Stellen Sie sich vor, Sie befolgen meinen Rat und erstellen eine Webapplikation aus einer Desktop-Applikation mit Visual Basic. Jemand aus dem Management oder der Marketing-Abteilung bezichtigt Sie nun, nicht mehr zeitgemäß zu arbeiten. In einem Design-Meeting könnte zum Beispiel die folgende Frage gestellt werden: »Wieso ist die Applikation nicht webbasiert?« Sie müssten die anderen davon überzeugen, dass die Applikation webbasiert ist. Sie verwenden lediglich etwas mehr HTTP und etwas weniger HTML. Wenn Sie einige Schlagworte anbringen möchten, um Ihre Aussage zu bekräftigen, sollten Sie antworten: »HTML war letztes Jahr. Diese Applikation ist von Grund auf mit XML erstellt worden.« Auf diese Weise können Sie die Skeptiker für sich gewinnen.

Methodenaufrufe mit HTTP und XML ausführen

Es gibt einige Möglichkeiten, um eine HTTP-Anforderung von einer Client-Applikation zu verschicken, die mit Visual Basic erstellt wurde. Ich werde Ihnen ein Beispiel präsentieren, dass den Microsoft XML-Parser verwendet. Dies ist die einfachste und flexibelste Lösung. Version 2 von Microsofts XML-Parser wird mit Windows 2000 ausgeliefert. Microsoft veröffentlicht jedoch fortwährend neue Versionen in regelmäßigen Abständen. Sie sollten die aktuelle Version benutzen, die Sie aus dem Internet herunterladen können (msdn.microsoft.com/xml/default.asp).

Ich kann in diesem Buch leider nicht mehr beschreiben, wie XML verwendet wird. Ich werde jedoch ein einfaches Beispiel präsentieren, um Ihnen zu zeigen, wie eine XML-basierte HTTP-Anforderung von einer Visual Basic-Applikation an IIS gesendet wird. Dazu verwenden Sie ein einfaches XML-Dokument, in dem die Eingabeparameter übergeben werden. Eine ASP-Seite liest diese Eingabeparameter aus, führt die angeforderte Methode aus und gibt Ausgabeparameter in einem anderen XML-Dokument an den Client zurück.

Das Beispiel basiert auf einer konfigurierten Komponente namens `CCustomerManager`. Stellen Sie sich vor, dass diese konfigurierte Komponente in einer COM+-Serverapplikation auf einem Webserver-Computer installiert ist. Wir führen eine Methode der Komponente aus, die mit `GetCustomer` bezeichnet ist und einen Eingabeparameter sowie zwei Ausgabeparameter besitzt. Hier die Methodeneimplementierung:

```
' Parameter sind als Variants definiert, damit sie von der ASP-Seite ausgelesen werden können.
Sub GetCustomer(ByVal Name, ByRef CreditLimit, ByRef AccountBalance)
    Dim conn As Connection, rs As Recordset
    Set conn = New Connection
    conn.Open sConnect
    Set rs = New Recordset
    rs.Open "SELECT CreditLimit, AccountBalance" & _
        " FROM Customers" & _
        " WHERE Customer='" & Name & "'", conn
    CreditLimit = rs("CreditLimit")
    AccountBalance = rs("AccountBalance")
    rs.Close
    Set rs = Nothing
    conn.Close
    Set conn = Nothing
End Sub
```

Da die `CCustomerManager`-Komponente in eine COM+-Serverapplikation installiert wurde, können COM-Clients über das Netzwerk darauf zugreifen. Wir möchten diese Methode jedoch auch von einer Client-Applikation ausführen lassen, die HTTP verwendet. Wir beginnen deshalb damit, eine ASP-Seite namens *GetCustomer.asp* zum Webserver hinzuzufügen. Dies geschieht mit dem folgenden Code:

```
<%@ Language=VBScript %>
<%
    Response.ContentType = "text/xml"
    set dom = Server.CreateObject("MSXML.DOMDocument")
    set obj = Server.CreateObject("WebMarket.CCustomerManager")
    dom.load Request
    dim Name, CreditLimit, AccountBalance
    Name = dom.selectSingleNode("//Name").text
```

```

obj.GetCustomer Name, CreditLimit, AccountBalance
%>

<Response>
  <CreditLimit><%=CreditLimit%></CreditLimit>
  <AccountBalance><%=AccountBalance%></AccountBalance>
</Response>

```

Diese ASP-Seite enthält einige interessante Elemente. Sie muss eine Instanz der `CCustomerManager`-Komponente erstellen, um die `GetCustomer`-Methode ausführen zu können. Zuvor muss die Seite jedoch den Eingabeparameter auslesen. Das Beispiel setzt voraus, dass der Client den Eingabeparameter in einem XML-Dokument versendet hat, das in den Körper der HTTP-Anforderung eingebettet ist.

Die ASP-Seite erstellt ein `DOMDocument`-Objekt, um das eingehende XML-Dokument zu bearbeiten. Das Objekt wird durch einen Aufruf der `load`-Methode und die Übergabe des ASP-Request-Objekts gefüllt. Dies ist nur dann möglich, wenn der Client ein entsprechend formatiertes XML-Dokument im Körper der Anforderung verschickt hat. (Diese Technik kann mit IIS 5 und Windows 2000, aber nicht mit IIS 4 und Windows NT genutzt werden.) Sobald das XML-Dokument geladen wurde, kann die ASP-Seite den Eingabeparameter extrahieren und die Methode ausführen. Nachdem `GetCustomer` ausgeführt wurde, bereitet die ASP-Seite ein XML-Antwortdokument vor, das an den Client zurückgegeben wird. Beachten Sie, dass die ASP-Seite explizit den Inhaltstyp der Antwort auf "text/xml" und nicht auf "text/html" setzt.

Der letzte Schritt besteht darin, eine HTTP-Anforderung für die Ausführung von `GetCustomer` von einer Desktop-Applikation aus zu verschicken, die mit Visual Basic geschrieben wurde:

```

Const sURL = http://localhost/MyApplication/
' XMLHttpRequest-Objekt erstellen, um die Anforderung zu verschicken.
Dim httpRequest As MSXML.XMLHttpRequest
Set httpRequest = New MSXML.XMLHttpRequest
' XML-Dokument vorbereiten, um die Eingabeparameter zu übergeben.
Dim Name As String, xmlRequest As String
Name = txtName.Text
xmlRequest = "<Request>" & _
    "<Name>" & Name & "</Name>" & _
    "</Request>"
' Verbindung zu einem Webserver öffnen und Anforderung versenden.
httpRequest.open "POST", sURL & "GetCustomer.asp", False
httpRequest.send xmlRequest
' Sind Fehler aufgetreten?
If httpRequest.status <> 200 Then
    ' Fehler auslösen, wenn der zurückgegebene Status nicht 200 ist.
    Err.Raise MyErrorCode, "HTTP transport error: " & httpRequest.status & _
        " - Description: " & httpRequest.statusText
End If
' DOMDocument-Objekt benutzen, um die Ausgabeparameter des Objekts zu lesen.
Dim ResponseDoc As DOMDocument
Set ResponseDoc = httpRequest.responseXML
Dim CreditLimit As Currency, AccountBalance As Currency
CreditLimit = ResponseDoc.selectSingleNode("//CreditLimit").Text
AccountBalance = ResponseDoc.selectSingleNode("//AccountBalance").Text

```

Microsofts XML-Parser stellt eine `XMLHttpRequest`-Komponente zur Verfügung, mit deren Hilfe ein Client eine HTTP-Anforderung an den Webserver senden kann. Wie Sie sehen, konstruiert die Client-Applikation ein einfaches XML-Dokument, das die Eingabeparameter der Methode enthält. Der Client ruft anschließend die `open`-Methode gefolgt von der `send`-Methode des `XMLHttpRequest`-Objekts auf, um den Code in der ASP-Seite auf dem Webserver ausführen zu lassen.

Beachten Sie, dass der dritte Parameter der `open`-Methode Aufschluss darüber gibt, ob die HTTP-Anforderung asynchron versendet werden soll. Unser Beispiel übergibt der Methode den Wert `False`, so dass der Aufruf synchron durchgeführt wird. Der Aufruf von `send` blockiert so lange, bis der Webserver eine Antwort zurückgibt. Nachdem der Aufruf zurückgekehrt ist, erstellt die Client-Applikation ein `DOMDocument`-Objekt, um die Ausgabeparameter aus dem XML-Antwortdokument zu extrahieren.

Ich habe dieses Beispiel präsentiert, um etwas zu verdeutlichen: Eine Client-Applikation kann HTTP ebenso wie DCOM benutzen, um mit einem Servercomputer zu kommunizieren. Das besondere an diesem Beispiel ist, dass Sie eine Client-Applikation generieren können, die zwischen DCOM und HTTP wechseln kann. In einer LAN-Umgebung kann DCOM geeigneter sein, da es schnellere Reaktionszeiten bietet. HTTP steht Ihnen jedoch ebenfalls zur Verfügung, wenn Sie es benötigen.

Beachten Sie bitte, dass die Übermittlung von HTTP-Anforderungen auch solchen Clients zur Verfügung steht, die Windows nicht benutzen. Der Client muss lediglich wissen, wie HTTP und XML eingesetzt werden. Dies eröffnet eine Welt, in die DCOM nicht eintreten kann.

Beachten Sie ebenfalls, dass die programmgesteuerte Übermittlung von XML-basierten HTTP-Anforderungen nicht auf Applikationen mit einer Benutzeroberfläche beschränkt ist. Stellen Sie sich zum Beispiel ein Szenario vor, in dem eine COM+-Applikation, die auf einem öffentlichen Webserver ausgeführt wird, ein Geschäftsobjekt aufrufen muss, das sich auf einem anderen Applikationsserver befindet. Wenn Ihr Unternehmen aus Gründen der Sicherheit eine Firewall zwischen dem öffentlichen Webserver und dem Applikationsserver eingerichtet hat, sollten Sie HTTP verwenden. Würden Sie stattdessen DCOM benutzen, würde die Firewall den Aufruf erschweren. Wenn der Applikationsserver den Clients die Geschäftsobjekte jedoch über HTTP anbietet, kann der öffentliche Webserver problemlos die Methoden dieser Objekte ausführen lassen.

SOAP und Web Services

Das vorherige Beispiel zeigt drei wichtige Aspekte der Methodenausführung mit HTTP.

- Sie können COM+-Applikationen über eine umfassend skalierbare Web-Farm-Architektur zur Verfügung stellen.
- Sie können Desktop-Applikationen integrieren, die mit Visual Basic erstellt wurden.
- Diese Lösung ist sehr umständlich. Sie müssen eine ASP-Seite für jede Methode erstellen und zwei XML-Dokumente vorbereiten und auswerten, um jeden Methodenaufruf durchführen zu können. Der Code ist einfach, aber das Erstellen all dieser ASP-Seiten und des erforderlichen clientseitigen Codes kann sehr viel Zeit beanspruchen.

Verglichen mit DCOM ist diese Art der Methodenausführung mit HTTP nicht besonders geordnet. DCOM versteckt die Details, die das Verpacken der Parameter in RPC-Nachrichten und das Entpacken dieser Nachrichten am Ziel betreffen. Ein Client, der DCOM verwendet, ruft einfach eine Methode auf, und alles Weitere geschieht im Hintergrund. Sie werden nun vielleicht die folgende Frage stellen: »Wenn dies alles so einfach mit DCOM und RPC ist, wieso kann HTTP dann nicht vereinfacht werden?«

Dies ist bereits geschehen. Viele Unternehmen haben einen Standard für Methodenaufrufe mit HTTP entwickelt. Vielleicht haben Sie schon von SOAP (Simple Object Access Protocol) gehört. SOAP ist eine neue Spezifikation, deren Ziel ein Standard für solche Clients ist, die Methoden über HTTP ausführen. Dazu werden mehrere HTTP-Standard-Header und eine XML-Nutzlast für die Übermittlung von Methodenparametern und Rückgabewerten definiert.

Einer der vielversprechendsten Aspekte von SOAP ist, dass es heterogene Systeme überbrücken kann. Denken Sie daran, dass SOAP vollständig auf HTTP und XML basiert. Sie sind somit nicht auf die Windows-Plattform angewiesen, weder auf der Client-Seite noch auf der Serverseite.

Eine Visual Basic-Applikation kann zum Beispiel eine SOAP-Anforderung an einen Servercomputer senden, der ein UNIX-Betriebssystem und einen Apache-Webserver ausführt. Die Methodenimplementierung kann in Perl geschrieben sein. Eine Client-Applikation, die nicht unter Windows ausgeführt wird, kann umgekehrt eine SOAP-Anforderung an einen Servercomputer schicken, der IIS und Windows 2000 Server benutzt. SOAP verlangt lediglich, dass der Client und der Server HTTP und XML verstehen müssen.

Eine der treibenden Philosophien von SOAP ist, keine neuen Technologien zu entwickeln. Die meisten Plattformen verfügen bereits über funktionelle Webserver und kommunizieren über HTTP. Darüber hinaus unterstützen immer mehr Plattformen XML. SOAP versucht, die allgegenwärtige Natur von HTTP und XML zu nutzen. Es berücksichtigt nicht den Lastenausgleich und die Sicherheit, da die zugrunde liegende Transportschicht hierfür verantwortlich ist. SOAP erweitert HTTP einfach, indem es definiert, wie Clients Anforderungen vorbereiten und wie der Server darauf reagieren muss. Auf der untersten grundlegenden Ebene ist SOAP wie eine COM-Schnittstelle. Es lässt die Client-Applikation und den Webserver vereinbaren, wie die Kommunikation durchgeführt werden soll.

SOAP befindet sich noch im Anfangsstadium. Die SOAP-Spezifikationen weisen keinen Nutzen auf, solange keine clientseitigen und serverseitigen Implementierungen vorhanden sind. Sie können die SOAP-Spezifikation abrufen und diese Implementierungen selbst schreiben. (Als dieses Buch geschrieben wurde, war die letzte Spezifikation unter <http://msdn.microsoft.com/xml/general/soapspec-V1.asp> erhältlich.) Wenn Sie selbst eine SOAP-Applikation schreiben, wird Ihr Code dem zuvor präsentierten Beispiel gleichen. Sie werden jedoch sehr viel Code hinzufügen müssen, um zum Beispiel applikationsspezifische Fehler weiterzuleiten und komplexe Datenstrukturen zu verpacken.

Das wahre Potential von SOAP ist jedoch, dass es Software-Unternehmen wie zum Beispiel Microsoft erlaubt, eine entsprechende Unterstützung zu deren Entwicklungssystemen und zugrunde liegenden Plattformen hinzuzufügen. Für die Entwickler ist dies vorteilhaft, da Microsoft und andere Anbieter im Laufe der Zeit dazu beitragen werden, dass ein Methodenaufruf über HTTP ebenso einfach wie über DCOM durchgeführt werden kann.

Microsoft verwendet gerne den Begriff Web Service, der einfach eine Applikation beschreibt, die ihre Funktionalität über SOAP-basierte HTTP-Anforderungen zur Verfügung stellt. Das Erstellen solcher Web Services, die clientseitige und serverseitige Implementierungen von SOAP verwenden, ist kein einfaches Unterfangen. Microsoft muss, wie viele andere Unternehmen auch, komplexe Aspekte berücksichtigen, wie zum Beispiel das Parameter-Marshaling und das Weiterleiten von Fehlern, damit SOAP später genauso souverän arbeiten wird wie DCOM heute.

Microsoft arbeitet gegenwärtig an einer Zusammenstellung von Entwicklungswerkzeugen und Hilfsmitteln für Entwickler, die SOAP in ihre eigenen Applikationen integrieren möchten. Diese Werkzeuge und Hilfsmittel werden in das Web Services SDK übernommen. (Der Name dieses SDKs könnte bereits geändert worden sein, wenn Sie dies lesen.) Ich konnte leider keine Beta-Version dieses Pakets erhalten, so dass ich Sie nicht darüber informieren kann, wie dieses SDK

SOAP unterstützen wird. Ich kann Ihnen auch nicht mitteilen, wann das Web Services SDK erhältlich sein wird. Vielleicht können Sie es bereits unter <http://msdn.microsoft.com/downloads/default.asp> herunterladen.

Ich kann jedoch mit Zuversicht behaupten, dass SOAP ein wesentlicher Bestandteil von Microsofts zukünftiger Entwicklungsstrategie sein wird. Das Web Services SDK wird sehr wahrscheinlich mehrmals überarbeitet, so dass im Laufe der Zeit die Features der jeweiligen Vorgängerversionen erweitert werden. Sie können außerdem davon ausgehen, dass zukünftige Versionen von COM+ und Visual Basic SOAP unterstützen werden. Vielleicht wird es eines Tages möglich sein, eine deklarative Attributseinstellung für eine Komponente oder einen Eigenschaftswert zur Laufzeit zu ändern, um wiederholt zwischen DCOM und HTTP umzuschalten.

Applikationsdesign

Wir werden in diesem Abschnitt einige wichtige Design-Aspekte behandeln, die die Entwicklung mehrschichtiger Applikationen betreffen. Sie lernen zunächst Techniken für die Übergabe von Daten mit DCOM, MSMQ und HTTP kennen. Im Anschluss daran werde ich Techniken für die Übergabe nicht verbundener Datensatzgruppen und zum Zwischenspeichern von Daten in der mittleren Schicht beschreiben.

Wenn Daten über das Netzwerk übergeben werden sollen, ist der Datenverkehr zwischen den Komponenten Ihr Gegner. Unabhängig von dem von Ihnen verwendeten Protokoll sollten die mit einem großen Aufwand verbundenen Aufrufe über das Netzwerk nur dann durchgeführt werden, wenn dies erforderlich ist. Es ist natürlich besser, viele Daten mit nur einer Übertragung zu übermitteln, als das Netzwerk mehrmals zu belasten.

Es ist ebenfalls wichtig, dass Sie die unterschiedlichen Datenübermittlungsmechanismen von DCOM, MSMQ und HTTP kennen. Die Parameter der COM-Methodenaufrufe sind häufig streng typisiert. Wie Sie im dritten Kapitel gesehen haben, ermöglichen streng typisierte Parameter dem Universal-Marshaler, die Datenübermittlung über das Netzwerk zu optimieren. Wenn Sie Daten mit MSMQ oder HTTP übergeben, sind diese weniger typisiert. Die zugrunde liegende MSMQ-Transportschicht erachtet den Körper einer MSMQ-Nachricht als einzelnes, typloses Gerüst. Auch die zugrunde liegende HTTP-Transportschicht sieht eine HTTP-Anforderung als ein einzelnes, typloses Gerüst an. Die Übergabe von Daten mit MSMQ und HTTP unterscheidet sich deshalb wesentlich von der Datenübergabe mit DCOM.

Wenn Sie an COM und streng typisierte Schnittstellen gewöhnt sind, erfordert die Kommunikation, die mit MSMQ und HTTP über Applikationsgrenzen hinweg durchgeführt wird, eine neue Perspektive. Sie können die Bedeutung Ihrer Daten nicht mit Methodensignaturen übermitteln, sondern müssen andere Techniken benutzen. Dokumentationen und selbstbeschreibende Datenstrukturen werden somit noch wichtiger. Wenn Sie Daten mit Hilfe einer weniger typisierten Transportschicht übertragen, können ADO-Datensatzgruppen und XML-Schemata nützlich sein. Diese verfügen über Metadaten, die die Nutzlast beschreiben, die zwischen den Applikationen bewegt wird. Denken Sie daran, dass eine Transportschicht, die mit weniger typisierten Daten arbeitet, applikationsspezifische Typbibliotheken benötigt, die von Ihnen installiert und konfiguriert werden müssen.

Daten mit COM übergeben

Eine der besten Eigenschaften von COM besteht darin, dass die Details der Interprozesskommunikation verborgen werden. Der Code, den Sie für ein prozessinternes Objekt schreiben, arbeitet

auch mit einem prozessexternen Objekt. Sie müssen keine Änderungen in Ihrem Visual Basic-Klassenmodul vornehmen. Für den clientseitigen Code, den Sie schreiben, ist es nicht wichtig, ob das Objekt in der Nähe oder weit entfernt ist.

Wie Sie wissen, ist COM einfacher zu verwenden als HTTP oder MSMQ, wenn Daten über die Prozess- und Computergrenzen hinweg übergeben werden müssen. Wie Sie im zehnten Kapitel gesehen haben, versucht QC, MSMQ ebenso einfach wie COM werden zu lassen. MSMQ weist jedoch auch weiterhin einige Einschränkungen und eine nur geringe Flexibilität auf. COM-Methodenaufrufe sind vielseitiger.

In diesem Buch wurde bisweilen erwähnt, dass DCOM für viele Situationen nicht geeignet ist. Bisweilen können und sollten Sie jedoch die Proxy-/Stub-Architektur von COM nutzen. Dies gilt zum Beispiel, wenn Sie einen synchronen Aufruf zwischen zwei Prozessen auf demselben Computer durchführen möchten. COM ist aber nicht einfacher zu bedienen als MSMQ und HTTP. Es weist außerdem nur selten eine bessere Performance auf.

Sie werden vielleicht in Situationen geraten, in denen Servercomputer innerhalb einer LAN-Hochgeschwindigkeitsumgebung über DCOM miteinander kommunizieren sollen. Sie können zum Beispiel mit einer Applikation, die den anforderungsbasierten Lastenausgleich verwendet, auf eine COM+-Serverapplikation zugreifen, die auf einem Servercomputer ausgeführt wird, damit die Webserver-Computer in einer Farm gemeinsam einen Sitzungsstatus benutzen können.

Wie Sie sehen, ist es bisweilen vorteilhaft, Daten mit COM zu übergeben. Wenn Sie eine Komponente entwickeln, auf die ein anderer Prozess zugreift, sollte diese über Methoden verfügen, die Daten möglichst effizient über das Netzwerk übermitteln. Lassen Sie uns einige praktische Aspekte für den Entwurf von Methoden betrachten, die über eine Proxy-/Stub-Schicht ausgeführt werden.

COM vereinfacht das Übermitteln von Daten, die auf primitiven VBA-Typen, wie zum Beispiel Integer, Double und String basieren. Sie generieren einfach Methodensignaturen mit diesen Typen, um Parameter und Rückgabewerte zu definieren. Es ist auch relativ einfach, komplexere Datentypen, wie zum Beispiel Arrays, UDTs und Variants zu übergeben. Sie können sogar UDT-Arrays oder ineinander verschachtelte Arrays übergeben. Der Universal-Marshaler kümmert sich um die Details, die das Verpacken der Daten und deren Übermittlung zum Empfänger betreffen.

Wenn Sie die Schnittstellen für Ihre Komponente entwerfen, sollten Sie vorsichtig sein, wenn Sie Objektreferenzen übergeben möchten. Weder Visual Basic noch der Universal-Marshaler unterstützen die Wertübergabe von Objekten über die Proxy-/Stub-Architektur von COM. Dies bedeutet, dass Sie ein Visual Basic-Objekt nur als Referenz übergeben können.

Wenn Sie ein Objektparameter mit `ByVal` deklarieren, wird die Referenz vom Client-Prozess zum Prozess der aufgerufenen Methode übermittelt. Wenn Sie ein Objektparameter mit `ByRef` deklarieren, wird die Referenz verpackt und in beide Richtungen übermittelt. Das COM-Objekt (und alle darin enthaltenen Daten) verlässt den Prozess jedoch nicht, in dem es erstellt wurde.

Wenn Sie eine Visual Basic-Objektreferenz über Prozess- oder Computergrenzen hinweg übergeben, wird das Objekt selbst nicht bewegt. Lediglich ein COM-fähiger Zeiger wird verschickt. Dieser Zeiger lässt den Empfänger eine Verbindung einrichten, die zurück zum referenzierten Objekt reicht. Wenn der Empfänger über die Objektreferenz auf eine Eigenschaft oder Methode zugreifen möchte, muss er über ein anderes Proxy-/Stub-Paar zu dem Objekt zurückkehren. Diese Form des Marshaling wird Standard-Marshaling genannt. Beachten Sie, dass die Übergabe von Objektreferenzen nicht mit dem Standard-Marshaling durchgeführt werden sollte, wenn dies zu unverhältnismäßig vielen Datenübermittlungen über das Netzwerk führt.

COM stellt eine Wertübergabe für eine spezielle Objektkategorie zur Verfügung. Um ein Objekt als Wert übergeben zu können, muss es eine COM-Standardschnittstelle mit der Bezeichnung `IMarshal` implementieren. Das Objekt verwendet dann das benutzerdefinierte Marshaling. Der Vorteil, der sich für Sie ergibt, wenn Sie ein solches Objekt benutzen, besteht darin, dass Sie dessen Daten mit einem einfachen COM-Methodenaufruf über Prozess- und Computergrenzen hinweg übergeben können.

Objekte, die das benutzerdefinierte Marshaling anbieten, weisen außerordentliche Anforderungen auf. Sie müssen in C++ und nicht in Visual Basic geschrieben sein. Sie müssen außerdem als nicht konfigurierte Komponenten installiert sein und ausgeführt werden.

ADO-Datensatzgruppen mit COM übergeben

Für die meisten Visual Basic-Programmierer ist das `ADO-Recordset`-Objekt ein übliches Beispiel für eine Komponente, die das benutzerdefinierte Marshaling implementiert. Sie können ADO-Datensatzgruppen mit Hilfe von COM-Methodenaufrufen über Prozess- und Computergrenzen hinweg übergeben. Dies ist sowohl mit Parametern als auch mit Rückgabewerten möglich. Wenn Sie eine ADO-Datensatzgruppe als Wert übergeben, müssen Sie einen clientseitigen statischen Cursor verwenden. Sie müssen die Verbindung zur Datensatzgruppe außerdem wieder trennen, indem Sie deren `ActiveConnection`-Eigenschaft auf `Nothing` setzen.

Lassen Sie uns ein Beispiel hierzu betrachten. Die `GetCustomerTable`-Methode der `CCustomerManager`-Komponente verwendet ihren Rückgabewert, um eine nicht verbundene Datensatzgruppe zurück an den Aufrufer zu übergeben.

```
Function GetCustomerTable() As Recordset
    Dim conn As Connection, rs As Recordset
    Set conn = New Connection
    conn.Open sConnect
    Set rs = New Recordset
    rs.CursorLocation = adUseClient
    rs.CursorType = adOpenStatic
    rs.LockType = adLockReadOnly
    Set rs.ActiveConnection = conn
    rs.Open "SELECT Customer, CreditLimit, AccountBalance" & _
        " FROM Customers"
    Set rs.ActiveConnection = Nothing
    conn.Close
    Set conn = Nothing
    Set GetCustomerTable = rs
End Function
```

Nachdem Sie nun diese Methode in der `CCustomerManager`-Komponente kennen gelernt haben, sollte Ihnen das Schreiben des clientseitigen Codes, der die Datensatzgruppe mit Hilfe des Marshaling von einem Prozess zu einem anderen übermittelt, kein Problem mehr bereiten:

```
Dim CustomerManager As CCustomerManager
Set CustomerManager = New CCustomerManager
Dim rs As Recordset
Set rs = CustomerManager.GetCustomerTable()
' Recordset-Objekt ist nun im Client-Prozess verfügbar.
```

Diese die Prozess- und Computergrenzen passierende Übermittlung einer ADO-Datensatzgruppe eignet sich besonders für kleine, mittlere und große Datengruppen. Sobald eine Datensatzgruppe

mit dem Marshaling in den Empfängerprozess übermittelt wurde, befinden sich die Daten in der Nähe und können somit schnell benutzt werden. Eine ADO-Datensatzgruppe trägt darüber hinaus zusätzliche Metadaten mit sich, die den Inhalt der Datensatzgruppe beschreiben (wie zum Beispiel den Namen und die Datentypen der Spalten). Ich werde dieses Thema später in diesem Kapitel noch einmal behandeln und einige komplexe Design-Aspekte beschreiben, die das Übermitteln von ADO-Datensatzgruppen über das Netzwerk betreffen.

Daten mit MSMQ und QC übergeben

Wie Sie im zehnten Kapitel gesehen haben, nutzen MSMQ und QC die Vorteile der asynchronen und nicht verbundenen Kommunikation. Dies bietet Ihnen die Möglichkeit, Anforderungen und Benachrichtigungen zwischen Applikationen zu versenden. Beachten Sie, dass MSMQ, ebenso wie DCOM, nicht einfach eine Firewall passieren und nur aufwendig für Internet-Clients konfiguriert werden kann. Sie können MSMQ jedoch in einigen Situationen für die Client-/Server-Kommunikation verwenden. MSMQ und QC sind darüber hinaus für die Server-/Server-Kommunikation geeignet.

Die Applikation, die die Nachricht sendet, bereitet den Nachrichtenkörper vor. Häufig müssen anforderungsspezifische Parameter übermittelt werden, so dass Sie während der Design-Phase darüber nachdenken sollten, wie die Daten in dem Körper einer MSMQ-Nachricht verpackt werden sollen. Sie müssen diese Parameter außerdem auf der Empfängerseite wieder entpacken, bevor Sie die Anforderung bearbeiten können. Im zehnten Kapitel wurde das Versenden und Empfangen von Nachrichten beschrieben. Die dort aufgeführten Beispiele übergeben jedoch lediglich VBA-Zeichenfolgen im Nachrichtenkörper. Lassen Sie uns deshalb einige Techniken untersuchen, die Nachrichten mit komplexeren Datenstrukturen übergeben.

Der Körper einer MSMQ-Nachricht ist ein Variant-Wert, der als Byte-Array gespeichert und übermittelt wird. Sie können die gewöhnlichen VBA-Datentypen (zum Beispiel Boolean, Byte, Integer, Long, Single, Double, Currency, Date und String) lesen und schreiben, indem Sie den Nachrichtenkörper verwenden. MSMQ überwacht die Typen, die Sie im Nachrichten-Header benutzen. Dies vereinfacht das Speichern eines einzelnen Wertes im Nachrichtenkörper. Es löst jedoch nicht das Problem, das auftritt, wenn mehrere verschiedene Daten gleichzeitig verpackt werden sollen. Um mehrere verschiedene Daten in einer Nachricht anordnen zu können, sollten Sie verstehen, wie das Byte-Array hinter dem Nachrichtenkörper verwendet wird.

Das Verwenden des Arrays, das sich hinter dem Nachrichtenkörper verbirgt, kann kompliziert sein, da es sich dabei lediglich um ein Byte-Array handelt. Wenn Sie dem Nachrichtenkörper einen anderen Array-Typ zuweisen, konvertiert MSMQ dieses Array in ein Byte-Array. Wenn Ihre Daten einmal in ein Byte-Array konvertiert wurden, können Sie diese auf der Empfängerseite leider nicht wieder in das ursprüngliche Array umwandeln. Dies bedeutet, dass eine einfache Technik wie zum Beispiel das Versenden Ihrer Parameter in einem Zeichenfolgen-Array nicht benutzt werden kann.

QC weist darüber hinaus eine Einschränkung auf, die das Übermitteln von Arrays betrifft. Visual Basic erlaubt Ihnen nicht, Array-Parameter mit `ByVal` zu definieren. Sie müssen stattdessen `ByRef` verwenden. `ByRef` in einer Methodensignatur verhindert aber, dass eine Schnittstelle für die Arbeit mit Warteschlangen verwendet werden kann. Dies bedeutet, dass die Methoden einer Warteschlangenkomponeute keine Parameter oder Rückgabewerte benutzen können, um Arrays zu übermitteln.

Trotz dieser Einschränkung und verglichen mit der direkten MSMQ-Programmierung vereinfacht eine Warteschlangenkomponeute die direkte Datenübergabe. Sie definieren `ByVal`-Parameter

einfach in Ihren Methodensignaturen, und die QC-Laufzeit verpackt alle Daten in einer MSMQ-Nachricht für Sie. Beachten Sie, dass eine Client-Applikation dieselbe Methode eines Recorderobjekts wiederholt aufrufen kann. Dies kann zu demselben Ergebnis wie die Übergabe eines Arrays über das Netzwerk führen.

Das Anordnen von Name-/Wertpaaren in Zeichenfolgen ist bei MSMQ-Programmierern sehr beliebt. Diese Lösung erlaubt Ihnen, parameterisierte Informationen mit einer einzelnen Zeichenfolge im Nachrichtenkörper zu verpacken. Die versendende Partei ist dann jedoch dafür verantwortlich, dass die Zeichenfolge konstruiert wird. Der Empfänger muss die Name-/Wertpaare extrahieren, indem die Zeichenfolge ausgewertet wird.

Eine Zeichenfolge, die Name-/Wertpaare enthält ist ein Beispiel für locker typisierte Daten. Der Versender und der Empfänger müssen sich über das Zeichenfolgenformat einigen. Wenn Sie sich für diese Lösung entscheiden, können XML-Schemata und der XML-Parser auf jeder Seite dazu beitragen, dass diese locker typisierte Natur des MSMQ-Körpers einfacher zu verwalten ist.

Nachrichtenkörper mit dem PropertyBag-Objekt verpacken

PropertyBag-Objekte sind nützlich, da sie das Anordnen der parameterisierten Informationen in einem Nachrichtenkörper und das darauf folgende Entpacken automatisieren. Ein PropertyBag-Objekt erlaubt Ihnen, mit Name-/Wertpaaren zu arbeiten. Sie lesen benannte Eigenschaften aus dem PropertyBag-Objekt aus und weisen dem Objekt benannte Eigenschaften zu. Das Objekt serialisiert Ihre Daten in ein internes Byte-Array.

Jedes PropertyBag-Objekt besitzt eine Contents-Eigenschaft, die ein internes Byte-Array repräsentiert. Sie können mit der WriteProperty-Methode benannte Werte in dieses Byte-Array schreiben. Nachdem Sie alle Parameter im PropertyBag-Objekt gespeichert haben, verwenden Sie die Contents-Eigenschaft, um das Byte-Array in den Nachrichtenkörper zu serialisieren, wie nachfolgend gezeigt:

```
Dim msg As MSMQMessage
Set msg = New MSMQMessage
Dim PropBag As PropertyBag
Set PropBag = New PropertyBag
PropBag.WriteProperty "Customer", "Bob"
PropBag.WriteProperty "Product", "MohairSuit"
PropBag.WriteProperty "Quantity", 4
msg.Body = PropBag.Contents
msg.Send q
```

Das PropertyBag-Objekt schreibt Ihre benannten Werte mit einem eigenen proprietären Algorithmus in eine Folge aus Byte-Werten. Sobald Sie ein Byte-Array mit der Versenderapplikation verpackt haben, benötigen Sie in zweites PropertyBag-Objekt auf der Empfängerseite, um das Array wieder zu entpacken. Sie entpacken die Nachricht, indem Sie das Byte-Array in ein neues PropertyBag-Objekt laden und die ReadProperty-Methode aufrufen:

```
Set msg = q.Receive()
Dim PropBag As PropertyBag
Set PropBag = New PropertyBag
PropBag.Contents = msg.Body
Dim Customer As String, Product As String, Quantity As Long
Customer = PropBag.ReadProperty("Customer")
Product = PropBag.ReadProperty("Product")
Quantity = PropBag.ReadProperty("Quantity")
```

Das `PropertyBag`-Objekt erleichtert das Leben des Programmierers, da es die Parameter selbstständig verpackt und entpackt. Sie können diese Technik natürlich auch für andere Aufgaben verwenden. Sie können zum Beispiel ein `Byte-Array`-Parameter für eine COM-Standardmethode definieren und Name-/Wertpaare zwischen einem Client und einem COM-Objekt übergeben. Sie können `PropertyBag`-Objekte ebenfalls benutzen, um persistente Name-/Wertpaare in ASP-Application-Variablen und ASP-Session-Variablen zu speichern.

Persistente Objekte verwenden

Eine weitere Technik zum Übermitteln von parameterisierten Informationen mit MSMQ oder QC benutzt persistente Objekte. Dies sind Objekte, deren Daten im Körper einer MSMQ-Nachricht gespeichert und übermittelt werden können. MSMQ kann Objekte aber nur dann im Nachrichtenkörper speichern, wenn diese eine von zwei COM-Standardschnittstellen implementieren: `IPersistStream` oder `IPersistStorage`.

Die Schnittstellendefinitionen für `IPersistStream` und `IPersistStorage` enthalten Parameter, die nicht mit Visual Basic kompatibel sind. Sie können diese Schnittstellen nicht einfach mit dem `Implements`-Schlüsselwort implementieren. Visual Basic 6 verfügt glücklicherweise über persistente Klassen. Wenn Sie eine persistente Klasse erstellen, implementiert Visual Basic automatisch `IPersistStream`. Persistente Klassen ermöglichen das direkte Auslesen und Schreiben von Objekten aus und in einen Nachrichtenkörper. Sie können darüber hinaus Warteschlangenkomponenten erstellen, die persistente Objekte als Parameter übergeben. Wenn Sie mit MSMQ oder QC arbeiten, werden die mit einem persistenten Objekt verknüpften Daten in den Körper einer MSMQ-Nachricht geschrieben. Im Anschluss daran wird ein Klonobjekt auf der Versenderseite erstellt, sobald die Empfängerapplikation die Nachricht erhält.

Jede öffentliche Klasse in einem ActiveX-DLL- und ActiveX-EXE-Projekt besitzt eine *Persistable*-Eigenschaft. Sie setzen diese Eigenschaft zur Entwurfszeit auf *Persistable*, um eine Klasse als persistent zu kennzeichnen. Wenn Sie eine persistente Klasse generieren, können Sie dem Klassenmodul eine `ReadProperties`- und eine `WriteProperties`-Methode hinzufügen. Sie können das Gerüst dieser Methoden mit Hilfe der Listenfelder *Objekt* und *Prozedur* in das Code-Editor-Fenster einfügen. Sie können ebenfalls die `InitProperties`-Methode benutzen, obwohl diese für den Einsatz unter MSMQ oder QC nicht benötigt wird.

Mit den Methoden `ReadProperties` und `WriteProperties` lesen Sie Ihre Eigenschaften in ein internes `PropertyBag`-Objekt ein oder weisen dem Objekt Ihre Eigenschaften zu. Visual Basic erstellt dieses Objekt automatisch im Hintergrund und verwendet es, um `IPersistStream` zu implementieren. Denken Sie daran, dass Ihr Objekt ebenfalls `IPersistStream` implementieren muss, damit MSMQ die Objektdaten in einen Nachrichtenkörper schreiben kann. Wenn MSMQ die Methoden der `IPersistStream`-Schnittstelle aufruft, leitet Visual Basic diese Aufrufe an Ihre Implementierung von `ReadProperties` und `WriteProperties` weiter.

Das Verwenden persistenter Klassen mit MSMQ ist einfacher als es scheint. Sie können zum Beispiel eine persistente Klasse erstellen und dieser die Eigenschaften hinzufügen, die in den Nachrichtenkörper übernommen werden sollen. Sie stellen im Anschluss daran eine Implementierung von `ReadProperties` und `WriteProperties` zur Verfügung. Nachfolgend ist der Programmcode eines persistenten Visual Basic-Klassenmoduls aufgeführt, der eine Bestellanforderung erzeugt:

```
' COrderRequest: eine persistente Klasse
Public Customer As String
Public Product As String
Public Quantity As Long
```

```

Private Sub Class_ReadProperties(PropBag As PropertyBag)
    Customer = PropBag.ReadProperty("Customer", "")
    Product = PropBag.ReadProperty("Product", "")
    Quantity = PropBag.ReadProperty("Quantity", "")
End Sub

Private Sub Class_WriteProperties(PropBag As PropertyBag)
    PropBag.WriteProperty "Customer", Customer
    PropBag.WriteProperty "Product", Product
    PropBag.WriteProperty "Quantity", Quantity
End Sub

```

Wie Sie sehen, ist es nicht schwierig, eine persistente Klasse zu erstellen. Nachdem Sie eine permanente Klasse wie die zuvor aufgeführte Klasse erstellt haben, können Sie diese benutzen, um einen Nachrichtenkörper zu verpacken:

```

Dim msg As MSMQMessage
Set msg = New MSMQMessage
' Bestellanforderungsobjekt erstellen und vorbereiten.
Dim Order As COrderRequest
Set Order = New COrderRequest
Order.Customer = txtCustomer.Text
Order.Product = txtProduct.Text
Order.Quantity = txtQuantity.Text
' Objekt dem Nachrichtenkörper zuweisen.
msg.Body = Order ' WriteProperties wird aufgerufen.
msg.Send q

```

Wenn Sie dem Nachrichtenkörper ein Objekt zuweisen, führt MSMQ dessen `QueryInterface`-Methode aus, um zu ermitteln, ob das Objekt die `IPersistStream`-Schnittstelle oder die `IPersistStorage`-Schnittstelle unterstützt. Wenn Ihr Objekt die `IPersistStream`-Schnittstelle unterstützt, weiß MSMQ, dass es eine Methode über diese Schnittstelle aufrufen kann, die mit `Save` bezeichnet ist. Visual Basic leitet diesen Aufruf an Ihre Implementierung von `WriteProperties` weiter. In dieser Implementierung schreiben Sie die benannten Eigenschaftswerte in das `PropertyBag`-Objekt. Die Parameter werden automatisch als ein Byte-Array in den Nachrichtenkörper kopiert.

In der Empfängerapplikation können Sie ein permanentes Objekt aus dem Nachrichtenkörper extrahieren, indem Sie einen neuen Verweis generieren und den Nachrichtenkörper diesem Verweis zuordnen:

```

Set msg = q.Receive(ReceiveTimeout:=0)
Dim Order As COrderRequest
Set Order = msg.Body
Dim Customer As String, Product As String, Quantity As Long
Customer = Order.Customer
Product = Order.Product
Quantity = Order.Quantity

```

Wenn Sie mit dem `Set`-Schlüsselwort einen Nachrichtenkörper einem Verweis zuordnen, erstellt MSMQ eine neue Instanz des Objekts und ruft dessen `Load`-Methode über die `IPersistStream`-Schnittstelle auf. Visual Basic leitet diesen Aufruf an Ihre Implementierung von `ReadProperties` weiter. Sie verwenden erneut das `PropertyBag`-Objekt, das als Parameter übergeben wurde, um Ihre Daten zu extrahieren.

QC lässt das Übermitteln von persistenten Objekten sogar noch einfacher werden, da Sie Nachrichten nicht explizit empfangen müssen. Wenn Sie ein persistentes Objekt an die Methode einer Warteschlangenkomponente übergeben, erstellt die QC-Laufzeit automatisch das geklonte Objekt, bevor Ihre Methode ausgeführt wird.

Wenn Sie diese Technik einsetzen, sollten Sie daran denken, dass Sie die DLL, die die persistente Klasse enthält, sowohl auf dem Versendercomputer als auch auf dem Computer des Empfängers installieren und konfigurieren müssen. Dies ist notwendig, da die MSMQ-Laufzeit eine lokale Kopie des Objekts auf beiden Computern generieren muss. Beachten Sie, dass diese Konfigurationsanforderung dazu führen kann, dass die anderen in diesem Kapitel vorgestellten Techniken dem Einsatz von persistenten Klassen vorgezogen werden.

ADO-Datensatzgruppen mit MSMQ übergeben

Sie können ebenfalls ADO-Datensatzgruppen im Körper einer MSMQ-Nachricht übergeben. Eine ADO-Datensatzgruppenkomponente, wie zum Beispiel eine persistente Klasse, implementiert `IPersistStream`, so dass Sie dem Nachrichtenkörper einfach eine ADO-Datensatzgruppe zuweisen können. MSMQ und ADO arbeiten dann zusammen, um die mit der Datensatzgruppe verknüpften Daten im Nachrichtenkörper zu verpacken. Sie können außerdem eine Methode mit Hilfe eines Datensatzgruppenparameters in einer Warteschlangenkomponente definieren. Dies ist jedoch nur dann möglich, wenn Sie den Parameter mit dem `ByVal`-Schlüsselwort kennzeichnen.

Diese Technik ist, wie die Übermittlung von Datensatzgruppe in einem COM-Methodenaufruf, nur dann geeignet, wenn Sie eine ADO-Datensatzgruppe mit einem clientseitigen statischen Cursor verwenden. Sie ist leistungsfähig und einfach zu verwenden. Betrachten Sie dazu den folgenden Code:

```
Dim conn As Connection
Set conn = New Connection
conn.Open sConnect
Dim rs As Recordset
Set rs = New Recordset
rs.CursorLocation = adUseClient
rs.CursorType = adOpenStatic
rs.LockType = adLockReadOnly
rs.Open "SELECT Customer, CreditLimit, AccountBalance" & _
    " FROM Customers"
' Datensatzgruppe in einer neuen Nachricht verpacken.
Dim msg As MSMQMessage
Set msg = New MSMQMessage
msg.Body = rs
' Nachricht an Warteschlange senden.
msg.Send ResponseQueue
ResponseQueue.Close
rs.Close
Set rs = Nothing
conn.Close
Set conn = Nothing
```

Auf der Empfängerseite können Sie die Datensatzgruppe ebenso einfach aus dem Nachrichtenkörper extrahieren und benutzen. Hier ein Beispiel für eine Client-Applikation, die eine Datensatzgruppe extrahiert und an ein Grid-Steuerelement bindet:

```

Set q = qi.Open(MQ_RECEIVE_ACCESS, MQ_DENY_NONE)
Set msg = q.Receive()
Dim rs As Recordset
Set rs = msg.Body
' Datensatzgruppe an Grid binden.
Set grdDisplay.DataSource = rs

```

Denken Sie an die Design-Möglichkeiten, die sich Ihnen mit dieser Lösung eröffnen. Stellen Sie sich zum Beispiel vor, Sie entwerfen eine Applikation für eine entfernte Verkaufsabteilung. Wenn sich Benutzer am Netzwerk anmelden, können Sie die aktuelle Kundenliste in die lokalen Warteschlangen der Benutzer-Notebooks laden. Trennen die Benutzer die Verbindung später, können sie ihre Arbeit fortsetzen und neue Bestellungen hinzufügen und Kundeninformationen aktualisieren. Sie können all diese Vorgänge mit Hilfe lokaler ADO-Datensatzgruppen im Hintergrund überwachen. Wenn sich ein Benutzer wieder am Netzwerk anmeldet, können die Datensatzgruppen transparent an eine Lisetner-Applikation weitergeleitet werden, die die neuen Bestellungen und die Aktualisierung der Kundendaten in die Datenbank übernimmt.

Daten mit HTTP übergeben

Sie können Code mit Visual Basic und Microsoft XML schreiben, um ein XML-Dokument in einer HTTP-Anforderung zu übermitteln. Sie haben hierzu bereits einige Beispiele gesehen. Ein XML-Dokument ist einfach eine Zeichenfolge. Die Herausforderung besteht darin, zu entscheiden, welche Informationen in dem Dokument angeordnet werden sollen. Das Entwerfen des XML-Schemas, auf dem das XML-Dokument basiert, ist sehr wichtig.

Im Laufe der nächsten Jahre wird XML immer häufiger für die Übergabe von Daten zwischen Applikationen, Organisationen und Anbietern eingesetzt werden. Es wird mehr Möglichkeiten für Unternehmen bieten, Informationen auf elektronischem Wege untereinander auszutauschen. Die elektronische Bestellung ist ein einfaches Beispiel. XML bietet jedoch eine weitaus umfassendere Funktionalität an. XML erlaubt Ihnen, COM+-Applikationen mit heterogenen Applikationen zu verwenden, die mit Corba und Enterprise JavaBeans erstellt wurden.

XML ist zu wichtig, als dass Sie es ignorieren könnten. Wenn Sie sich noch nicht mit dieser Technologie beschäftigt haben, sollten Sie damit beginnen. Der Microsoft XML-Parser hilft Ihnen dabei. Je mehr Sie über XML wissen, desto besser vorbereitet sind Sie später. Denken Sie ebenfalls daran, dass XML-Dokumente gegenwärtig vorwiegend mit HTTP übermittelt werden. Sie können Sie jedoch ebenfalls mit COM und MSMQ übergeben.

ADO-/XML-Datensatzgruppen mit HTTP übergeben

Sie sollten nach Möglichkeiten suchen, die Vorteile der leistungsfähigen Integration von Microsoft XML und ADO 2.5 zu nutzen. Diese Integration gestattet Ihnen, ADO-Datensatzgruppen in HTTP-Anforderungen vom Webserver zurück an die Client-Applikation zu übermitteln. Wie Sie noch sehen werden, können Sie auch eine aktualisierte Datensatzgruppe mit Hilfe einer HTTP-Anforderung von einer Client-Applikation zurück an den Webserver senden.

Lassen Sie uns ein Beispiel betrachten. Hier eine Implementierung der `GetCustomerTable`-Methode der `CCustomerManager`-Komponente:

```

Function GetCustomerTable() As Recordset
    Dim conn As Connection, rs As Recordset
    Set conn = New Connection
    conn.Open sConnect

```

```

Set rs = New Recordset
rs.CursorLocation = adUseClient
rs.CursorType = adOpenStatic
rs.LockType = adLockBatchOptimistic
Set rs.ActiveConnection = conn
rs.Open "SELECT Customer, CreditLimit, AccountBalance" & _
      " FROM Customers"
Set rs.ActiveConnection = Nothing
conn.Close
Set conn = Nothing
Set GetCustomerTable = rs
End Function

```

Diese Implementierung der `GetCustomerTable`-Methode gleicht der bereits gezeigten Implementierung. Der einzige Unterschied besteht darin, dass hier der `LockType`-Eigenschaft der Datensatzgruppe der Wert `adLockBatchOptimistic` zugewiesen wird. Dies erlaubt Ihnen, die nicht verbundene Datensatzgruppe in der Client-Applikation zu aktualisieren.

Lassen Sie uns nun eine ASP-Seite betrachten, die einer Visual Basic-Applikation erlaubt, `GetCustomerTable` aufzurufen und die Datensatzgruppe mit Hilfe von HTTP abzurufen.

```

<%@ Language=VBScript %>
<%
    Response.ContentType = "text/xml"
    set obj = Server.CreateObject("WEBMARKET.CCustomerManager")
    set rs = obj.GetCustomerTable()
    rs.Save Response, 1
%>

```

Die ASP-Seite benutzt das `CCustomerManager`-Objekt auf, um das Datensatzgruppenobjekt abzurufen. Sie serialisiert die Datensatzgruppe anschließend in den Körper der HTTP-Antwort, indem sie die `Save`-Methode aufruft und das ASP-Response-Objekt als ersten Parameter übergibt.

Der erste Parameter der `Save`-Methode einer ADO-Datensatzgruppe akzeptiert ein Objekt, das die `IStream`-Schnittstelle implementiert. In IIS implementiert das ASP-Response-Objekt `IStream`. Das ADO-Datensatzgruppenobjekt kann das ASP-Response-Objekt somit benutzen, um einen serialisierten Datenstrom darin abzulegen. Wenn die ASP-Seite `Save` aufruft, serialisiert die ADO-Laufzeit die Daten der Datensatzgruppe und schreibt sie in die HTTP-Antwort.

Beachten Sie, dass die ASP-Seite den Wert 1 als zweiten Parameter an die `Save`-Methode übergibt. Dieser Wert weist die ADO-Laufzeit an, die Datensatzgruppe im ADO-XML-Format zu serialisieren. In einer Visual Basic-Applikation, die die ADO-Typbibliothek referenziert, sollte jedoch die Konstante `adPersistXML` anstelle des Wertes 1 übergeben werden.

Betrachten Sie nun den Code in der Client-Applikation, der die Datensatzgruppe vom Webserver abrufen.

```

Const sURL = http://localhost/MyApplication/
Dim rs As Recordset
Set rs = New Recordset
rs.Open sURL & "GetCustomerTable.asp"
' Programmgesteuerter Zugriff auf die Datensatzgruppe erfolgt jetzt,
' oder die Datensatzgruppe wird an ein Grid-Steuerelement gebunden.

```

Beachten Sie, dass nur wenig Code erforderlich ist, wenn Sie ADO 2.5 benutzen. Die `Open`-Methode eines `ADO-Recordset`-Objekts akzeptiert den URL einer ASP-Seite, die eine Datensatzgruppe in serialisierter Form als XML-Dokument zurücksenden kann. Sie müssen keinen Code schreiben, der den XML-Parser verwendet.

Nachdem der Client die Datensatzgruppe erhalten hat, können Sie die Datensatzgruppe zum Beispiel an ein Grid-Steuerelement binden und Sortier-, Filter- und Suchoperationen durchführen. Wenn die Datensatzgruppe mit einer `LockType`-Einstellung von `adLockBatchOptimistic` erstellt wurde, können Sie sie in der Client-Applikation aktualisieren und zum Webserver zurücksenden, um die Benutzeränderungen in die Datenbank zu übernehmen.

Lassen Sie uns ein Beispiel betrachten, das eine aktualisierte Datensatzgruppe zum Webserver zurückschickt. Der Code ist ein wenig komplizierter als das letzte Beispiel, da Sie ein `DOMDocument`-Objekt benutzen müssen.

```
Const sURL = http://localhost/MyApplication/  
Dim rs As Recordset  
' Datensatzgruppe aus einem Grid-Steuerelement verwenden.  
Set rs = grdCustomers.DataSource  
Dim RequestDoc As MSXML.DOMDocument  
Set RequestDoc = New MSXML.DOMDocument  
rs.Save RequestDoc, adPersistXML  
Dim httpRequest As MSXML.XMLHttpRequest  
Set httpRequest = New MSXML.XMLHttpRequest  
httpRequest.Open "POST", sURL & "PostCustomerTable.asp", False  
httpRequest.send RequestDoc  
' Auf Fehler überprüfen.  
If httpRequest.Status <> 200 Then  
    ' Fehler auslösen, wenn zurückgegebener Status nicht 200 ist.  
    Err.Raise MyErrorCode, "HTTP transport error: " & httpRequest.Status & _  
        " - Description: " & httpRequest.statusText  
End If
```

Das Zurücksenden der Datensatzgruppe an den Webserver ist ein wenig kompliziert, da die Daten in ein XML-Dokument serialisiert werden müssen. Die in diesem Beispiel verwendete Technik ruft die `Save`-Methode des `Recordset`-Objekts auf. Die Client-Applikation übergibt ein `DOMDocument`-Objekt als ersten Parameter. Das `DOMDocument`-Objekt implementiert, ebenso wie das `ASP-Response`-Objekt, die `IStream`-Schnittstelle. Dies bedeutet, dass die ADO-Laufzeit serialisierte XML-Daten in das `DOMDocument`-Objekt schreiben kann. Das `DOMDocument`-Objekt kann anschließend als Parameter an die `Send`-Methode des `XMLHttpRequest`-Objekts übergeben werden. Das XML-Dokument, das die Daten des aktualisierten `Recordset`-Objekts enthält, wird nun zum Webserver übermittelt. Hier der Code in der ASP-Seite, die die eingehende Anforderung auf dem Webserver bearbeitet:

```
<%@ Language=VBScript %>  
<%  
    dim rs  
    set rs = Server.CreateObject("ADODB.Recordset")  
    rs.Open Request  
    set obj = Server.CreateObject("WEBMARKET.CCustomerManager")  
    obj.PostCustomerTable rs  
%>
```

Die ASP-Seite erstellt ein neues `Recordset`-Objekt und lädt dies mit den XML-Daten aus dem `ASP-Request`-Objekt. Dies ist möglich, da auch das `ASP-Request`-Objekt in IIS 5 die `IStream`-Schnittstelle implementiert. Die ASP-Seite erstellt nun eine Instanz von `CCustomerManager` und ruft die `PostCustomerTable`-Methode auf, der die Datensatzgruppe übergeben wird. Das letzte Glied in der Kette ist somit die Implementierung der `PostCustomerTable`-Methode der `CCustomerManager`-Komponente.

```
Sub PostCustomerTable(ByVal rs As Recordset)
    Dim conn As Connection
    Set conn = New Connection
    conn.Open sConnect
    Set rs.ActiveConnection = conn
    rs.UpdateBatch
    Set rs.ActiveConnection = Nothing
    conn.Close
    Set conn = Nothing
End Sub
```

Die Implementierung von `PostCustomerTable` richtet eine Verbindung zur Datenbank ein und verknüpft diese mit der eingehenden Datensatzgruppe. Nachdem die Datensatzgruppe mit einer aktiven Verbindung verknüpft wurde, führt die Methodenimplementierung die `UpdateBatch`-Methode aus. Die ADO-Laufzeit arbeitet im Anschluss daran mit Ihrem OLE-DB-Provider zusammen, um die erforderlichen `INSERT`-, `UPDATE`- und `DELETE`-Anweisungen zu erstellen und auszuführen. Wie Sie sehen, helfen Ihnen aktualisierbare Datensatzgruppen dabei, weniger zu programmieren, da alle notwendigen SQL-Anweisungen im Hintergrund generiert und ausgeführt werden.

Schreibgeschützte Datensatzgruppen im Vergleich mit aktualisierbaren Datensatzgruppen

Das Übermitteln von ADO-Datensatzgruppen über das Netzwerk war schon immer ein kontroverses Thema. Einige Puristen glauben, dass die Übergabe von nicht verbundenen Datensatzgruppen an eine Client-Applikation das dreischichtige Modell verletzt. Diese Personen sind der Meinung, dass Datenzugriffscode und Datenbankschemainformationen in der mittleren Schicht verwahrt werden sollten. Andere hingegen argumentieren, dass die praktischen Vorteile die theoretischen Nachteile ausgleichen.

Das Übermitteln einer ADO-Datensatzgruppe an eine Client-Applikation weist mehrere Vorteile auf. Sobald Sie eine Datensatzgruppe auf den Desktop-Computer des Benutzers geladen haben, können Sie die clientseitige ADO-Cursor-Engine benutzen, um Filter-, Sortier- und Suchoperationen durchzuführen. Sie können Datensatzgruppen an datengebundene Steuerelemente binden. Ich weiß, dass viele Visual Basic-Programmierer niemals gestehen würden, gebundene Steuerelemente zu verwenden. Diese können jedoch bisweilen sehr nützlich sein.

Das Übermitteln von Datensatzgruppen an den Client hält auch einige Vorteile hinsichtlich der Skalierbarkeit bereit. Der Client muss beispielsweise keine Anforderung an den Webserver verschicken (der wiederum den Datenbankserver aufrufen müsste), um Sortier-, Filter und Suchoperationen durchzuführen. Die clientseitige ADO-Cursor-Engine führt diese Operationen lokal aus, wodurch die Client-Applikation ein besseres Reaktionsvermögen aufweist. Die Client-Applikation sendet darüber hinaus nur wenige Anforderungen an Servercomputer, so dass der Netzwerkverkehr reduziert wird. Sie können außerdem bestimmte Aufgaben auf die Desktop-Computer auslagern, wodurch wertvolle Verarbeitungszyklen auf dem Datenbankservercomputer gewonnen werden.

Sie sollten den Unterschied zwischen schreibgeschützten und aktualisierbaren Datensatzgruppen verstehen. Der Einsatz von schreibgeschützten Datensatzgruppen ist in der Regel weniger kompliziert. Aktualisierbare Datensatzgruppen verlangen hingegen die Berücksichtigung der Parallelität und der vollständigen Sperrung. Sie erfordern außerdem Code, der auf Laufzeitfehler reagiert, die aus Aktualisierungskonflikten entstehen.

Wenn Sie `UpdateBatch` aufrufen, um eine aktualisierte Datensatzgruppe an die Datenbank zurückzusenden, sollten Sie auf Fehler vorbereitet sein. Viele Programmierer vermeiden aktualisierbare Datensatzgruppen, da sie glauben, dass diese mehr Probleme verursachen als zu lösen. Andere Programmierer sind der Meinung, dass aktualisierbare Datensatzgruppen in gewissen Situationen nützlich sind. Sie entscheiden selbst über den Nutzen von Datensatzgruppen.

Wenn Sie aktualisierbare Datensatzgruppe verwenden, sollten Sie wissen, wie ADO die vollständige Sperrung einsetzt, um auf Aktualisierungskonflikte zu reagieren. Wenn eine Client-Applikation eine Zeile in einer nicht verbundenen Datensatzgruppe modifiziert, werden diese Änderungen zwischengespeichert. ADO verwahrt jedoch eine Kopie der Originalzeilen und der neuen Zeilenwerte. Beide Werte werden zusammen mit der Datensatzgruppe an den Webserver zurückgesendet. Wenn die Middle-Tier-Komponente schließlich `UpdateBatch` aufruft, um die Änderungen in die Datenbank zu schreiben, ermittelt ADO, ob andere Benutzer ebenfalls Änderungen vorgenommen haben, nachdem die Datensatzgruppe aus der Datenbank kopiert wurde.

ADO entdeckt Aktualisierungskonflikte, indem es `WHERE`-Klauseln zu den `INSERT`-, `UPDATE`- und `DELETE`-Anweisungen hinzufügt. ADO kann auf diese Weise feststellen, ob jede Zeile in der ursprünglichen Form aktualisiert wird. ADO überprüft den Wert, der Aufschluss über die von einer Änderung betroffenen Zeilen gibt, für jede einzelne Anweisung, um zu gewährleisten, dass jede Aktualisierung erfolgreich durchgeführt wird. Wenn ADO feststellt, dass eine `UPDATE`-Anweisung keine Zeile betrifft, muss ein anderer Benutzer den Datensatz modifiziert haben. Dieser Aktualisierungskonflikt führt dazu, dass ADO einen Laufzeitfehler in dem Aufruf von `UpdateBatch` auslöst.

Der Einsatz von aktualisierbaren Datensatzgruppen wird kompliziert, wenn partielle Fehler auftreten. Was geschieht zum Beispiel, wenn ein Client eine aktualisierte Datensatzgruppe mit zwanzig Änderungen zurückschickt, von denen drei Änderungen aufgrund von Konflikten nicht in die Datenbank übernommen werden können? Sollten die anderen siebzehn Aktualisierungen in diesem Fall zurückgesetzt werden? Diese Frage ist nicht einfach zu beantworten, da die Antwort zu einem großen Teil von der Applikation abhängig ist. Der ADO-Mechanismus für die vollständige Sperrung ist generisch. Ihr Code für den Eventualfall muss jedoch Ihren Anforderungen entsprechend angepasst sein. Sie werden bisweilen ein äußerst komplexes Konfliktlösungsschema implementieren müssen. Dieses Schema könnte auch das Zurücksenden einer Gruppe mit Datensätzen beinhalten, die aufgrund von Konflikten nicht an den Client zurückgeschrieben werden können.

Wenn Sie darüber nachdenken, ob Sie aktualisierbare Datensatzgruppen verwenden möchten, sollten Sie die betroffenen Daten untersuchen und mögliche Konflikte ermitteln. Bisweilen führen Datensatzgruppen zu keinen oder zu nur wenigen Konflikten. Dies ist zum Beispiel der Fall, wenn Sie eine Kundendatensatzgruppe übergeben, die alle Konten eines spezifischen Außendienstmitarbeiterbereichs repräsentiert. Wenn kein anderer Außendienstmitarbeiter dieselben Datensätze aktualisieren muss, können Sie davon ausgehen, dass keine Konflikte auftreten werden. Wenn Sie dieselben Kundendatensätze jedoch an viele verschiedene Benutzer übergeben müssen, werden voraussichtlich Konflikte auftreten. Der Einsatz von aktualisierbaren Datensatzgruppen ist dann sehr viel komplizierter.

Stellen Sie sich vor, Ihre Applikation verlangt ein weiterentwickeltes Sperrungs- und Parallelitätsschema. Vielleicht wünschen Sie ein Applikationsdesign, in dem ein Benutzer einen Kunden-

datensatz zur Bearbeitung abrufen und für andere Benutzer eine gewisse Zeit sperren kann. Wenn Sie davon ausgehen, dass der durchschnittliche Benutzer fünf Minuten für die Bearbeitung des abgerufenen Datensatzes benötigt, müssen Sie den Datensatz irgendwie sperren, damit ihn andere Benutzer nicht aktualisieren können.

Beachten Sie, dass die von ADO für clientseitige Cursors angebotene vollständige Sperrung (optimistic locking) nicht die Funktionalität bietet, die Sie benötigen. Nicht verbundene Datensatzgruppen sperren die Datenbank nicht. Ihr einziger Beitrag zur Konfliktvermeidung besteht darin, dass sie überprüfen, ob Aktualisierungskonflikte aufgetreten sind. Wir benötigen jedoch für unser Beispiel eine Datenbanksperrung, um andere Benutzer davon abzuhalten, den Datensatz zu aktualisieren.

Das Verwenden eines serverseitigen Cursors und einer eingeschränkten Sperrung (pessimistic locking) ist keine gute Lösung. Eingeschränkte Sperrungen sollten nur für kurze Perioden – in der Regel für die Dauer einer Transaktion – aufrechterhalten werden. Ein Problem der serverseitigen eingeschränkten Sperrung besteht darin, dass sie ein Recordset-Objekt zusammen mit einer hergestellten Verbindung erfordert. Dies ist in der mittleren Schicht nicht möglich. Das zweite Problem betrifft die Fehlertoleranz. Wenn die Client-Applikation abstürzt oder der Benutzer nicht anwesend ist, nachdem eine eingeschränkte Sperrung eingerichtet wurde, kann diese Sperrung für eine lange Zeit bestehen bleiben.

Die Lösung dieses Problems besteht darin, ein benutzerdefiniertes Sperrschema zu schreiben. Wenn die Benutzer einen Datensatz logisch sperren können sollen, können Sie ihnen ein Timeout-Intervall zuweisen. Die Benutzer müssen ihre Änderungen dann innerhalb dieser Zeit vornehmen.

Ein benutzerdefiniertes Sperrschema verlangt von Ihnen, zusätzliche Felder zu Ihren Datenbanktabellen hinzuzufügen. Jede Tabelle, die das Schema unterstützen soll, benötigt zusätzliche Spalten für Informationen über den Besitzer und den Zeitpunkt der Sperrung. Sie schreiben anschließend Code, um einer Sperrung einen Besitzer zuzuweisen und die Zeit der Sperrung zu aktualisieren.

Wenn ein Benutzer versucht, einen Datensatz abzurufen und zu bearbeiten, muss Ihr Code ermitteln, ob der Datensatz bereits von einem anderen Benutzer gesperrt wurde. Sie müssen den Zeitpunkt der Sperrung feststellen und ermitteln, ob eine bereits bestehende Sperrung das Timeout-Intervall überschritten hat. Wenn ein anderer Benutzer einen Datensatz sperrt, können Sie einen Fehler zurückgeben. Wenn jedoch keine Sperrung besteht, können Sie den Besitzer und den Zeitpunkt der Sperrung im Datensatz aktualisieren, bevor Sie den Datensatz für die Dauer des von Ihnen konfigurierten Intervalls an einen Benutzer zurückgeben.

Wie Sie sehen, wird die von ADO und OLE-DB angebotene automatische Sperrung den Anforderungen der komplexen Parallelitätsrichtlinien nicht gerecht. Sie benötigen häufig ein benutzerdefiniertes Schema wie das soeben vorgestellte, das wiederum zusätzliche Arbeiten während der Design- und Implementierungsphase erfordert.

Daten in der mittleren Schicht zwischenspeichern

Es gibt zwei Gründe, die dafür sprechen, Daten auf dem Webserver zwischenspeichern: Die Reaktionszeiten werden verkürzt (da weniger Datenübermittlungen vom und zum Datenbankserver notwendig sind), und auf dem Datenbankserver werden Verarbeitungszyklen gespart.

Lassen Sie uns direkt beginnen und beobachten, wie eine ADO-Datensatzgruppe in eine Zeichenfolge der mittleren Schicht konvertiert wird. Diese Technik erlaubt Ihnen, eine ADO-/XML-Datensatzgruppe in einer ASP-Application-Variablen oder mit dem SPM in einer gemeinsam

genutzten Eigenschaft zwischenspeichern. ADO 2.5 hat das Stream-Objekt eingeführt, das Sie unterstützt. Betrachten Sie den folgenden Code:

```
Dim conn As Connection
Set conn = New Connection
conn.Open sConnect
Dim rs As Recordset
Set rs = New Recordset
rs.CursorLocation = adUseClient
rs.CursorType = adOpenStatic
rs.LockType = adLockReadOnly
Set rs.ActiveConnection = conn
rs.Open "SELECT Customer, CreditLimit, AccountBalance" & _
    " FROM Customers"
Set rs.ActiveConnection = Nothing
conn.Close
Set conn = Nothing
' Datensatzgruppe im XML-Format in einer Zeichenfolgenvariablen speichern.
Dim strm As Stream, MyXML As String
Set strm = New Stream
rs.Save strm, adPersistXML
MyXML = strm.ReadText
rs.Close
Set rs = Nothing
' MyXML jetzt in einer ASP-Application-Variablen speichern.
```

Wie bereits erwähnt wurde, ist das Stream-Objekt neu in ADO 2.5. In früheren Versionen von ADO konnten Sie persistente Datensatzgruppen in Dateien, aber nicht im Speicher ablegen. Der Grund hierfür bestand darin, dass die mit der Datensatzgruppe verknüpften Daten nicht in eine Variable geschrieben werden konnten. Das Stream-Objekt in ADO 2.5 lässt diese Aufgabe jedoch relativ einfach werden.

Sobald Sie die XML-Daten in eine ASP-Application-Variable oder eine gemeinsam genutzte Variable des SPM geschrieben haben, können Sie sie in den folgenden Anforderungen benutzen. Sie können mit diesen Daten ein ADO-Recordset-Objekt oder ein DOMDocument-Objekt laden. Sie können darüber hinaus die XML-Daten auf dem Webserver manipulieren und in einer HTTP-Anforderung an den Client zurückgeben.

Beachten Sie, dass Sie ADO-Recordset-Objekte und ADO-Stream-Objekte nicht in ASP-Application-Variablen oder gemeinsam genutzten Eigenschaften des SPM ablegen sollten. Diese Elemente weisen Threading-Eigenschaften auf, die zu Konflikten führen können. XML-Text kann jedoch ohne Bedenken in diesen Variablen der mittleren Schicht gespeichert werden, da es sich dabei um eine Zeichenfolge handelt. Sie können außerdem schreibgeschützte XML-Daten in ein DOMFreeThreadedDocument-Objekt laden. Dieses Objekt verfügt über weiter entwickelte Threading-Features, die das Speichern des Objekts in einer ASP-Application-Variablen oder einer gemeinsam genutzten Eigenschaft des SPMs ermöglichen. Diese Technik kann Verarbeitungszyklen schonen, da Sie die XML-Daten in einem DOMDocument-Objekt nicht für jede einzelne Anforderung erneut laden müssen.

Sie sollten für einige Applikationen XSL-Konvertierungen (Extensible Style Language) auf dem Webserver durchführen. Dies gilt besonders dann, wenn Sie Browser unterstützen, die XML nicht nutzen können. Diese Clients benötigen formatierte HTML-Tabellen. Sie können die XML-Daten, die die Eingabe für die XSL-Konvertierung repräsentieren, zwischenspeichern. Sie können außerdem die resultierende HTML-Tabelle zwischenspeichern, die von der XSL-Konvertie-

nung generiert wird. Wenn viele Clients dieselbe Tabelle benötigen, müssen Sie diese nicht noch einmal erstellen. Sie können sie für viele Anforderungen benutzen. Der Nachteil ist, dass Sie viel Speicher auf dem Webserver benötigen. Speicher kann jedoch häufig einfacher und günstiger als Verarbeitungszyklen erworben werden.

Wenn Sie Daten in der mittleren Schicht zwischenspeichern, sollten Sie wissen, wie oft die Daten aktualisiert werden müssen. Einige Daten ändern sich so oft, dass ein Zwischenspeichern nicht zweckmäßig ist. Andere Daten bleiben für Tage oder Monate unverändert. Wieder andere Daten, wie zum Beispiel die Daten eines Lagers, ändern sich niemals und können deshalb als schreibgeschützt erachtet werden. Wenn Sie sich mit der Natur Ihrer Daten beschäftigt haben, entscheiden Sie, wie oft diese aktualisiert werden sollen. Statische Daten eignen sich besonders für Cache-Techniken.

Im neunten Kapitel wurden bestimmte Aspekte beschrieben, die das regelmäßige Aktualisieren zwischengespeicherter Daten betreffen. Einige dieser Aspekte sind nicht besonders kompliziert. Das Entwerfen eines Cache-Schemas für eine Web-Farm-Umgebung kann jedoch sehr kompliziert sein. Dies gilt besonders für Applikationen, die den Sitzungsstatus für solche Benutzer speichern müssen, die fortwährend über verschiedene Server in der Web-Farm hinweg weitergeleitet werden. In den meisten Fällen besteht die einfachste Lösung darin, den Sitzungsstatus in eine Datenbank zu schreiben.

Zusammenfassung

Sie sollten einige wichtige Ausführungen dieses Kapitels in Erinnerung behalten. Der Begriff Skalierbarkeit hat viele Bedeutungen. Während die meisten Applikationen verschiedene Formen der Skalierbarkeit benötigen, sind einige wenige Applikationen genügsamer. Sie sollten deshalb frühstmöglich definieren, welche Anforderungen Ihre Applikation erfüllen soll.

In diesem Kapitel wurde ebenfalls beschrieben, welche wichtigen Rollen HTTP und die Web-Farm-Architektur für die Skalierung einer COM+-Applikation spielen. Dies bedeutet, dass die meisten Applikationen HTTP und nicht DCOM für die Client-/Serverkommunikation verwenden sollten. Vergessen Sie auch nicht, dass HTTP ohne HTML benutzt werden kann. HTML ist nur dann wichtig, wenn Sie plattformübergreifend arbeiten und einfache Browser unterstützen möchten. Visual Basic-Desktop-Applikationen, die HTTP benutzen, werden immer beliebter. Sie können in eine skalierbare Architektur integriert werden und eine Benutzeroberfläche zur Verfügung stellen, die weiter entwickelt als die auf Webbrowsern basierenden Oberflächen sind.

Sie haben in diesem Buch viel über XML erfahren. Ich möchte Sie dazu ermutigen, weitere Erfahrungen in diesem Bereich zu sammeln. Sie sollten Microsofts XML-Parser aus dem Internet laden und mit dessen Dokumentation vertraut werden. Das Buch »XML in Action« von William J. Pardi (Microsoft Press, 1999) erklärt XML von Grund auf und ist somit für den Einstieg in diese Sprache geeignet.

Wenn Sie sich intensiv mit XML beschäftigen möchten, sollten Sie »Essential XML« von Don Box, Aaron Skonnard und John Lam (Addison Wesley, 2000) lesen. Ich bin natürlich ein wenig voreingenommen, da einer der Autoren das Vorwort zu diesem Buch geschrieben hat. »Essential XML« ist jedoch ein Buch, das alle XML-Details beschreibt. Es erläutert nicht nur die Syntax des XML-Parsers, sondern geht noch weiter und beschreibt, wieso XML gegenwärtig so wichtig für die Industrie ist. Ein weiteres wichtiges Buch über XML, das auch Details zu SOAP und BizTalk enthält, ist »XML-Lösungen programmieren« von Jake Sturm (Microsoft Press, 2000).

Die zweite Ausgabe dieses Buches nähert sich dem Ende. Dies bedeutet, dass ich noch einmal das Erlernte zusammenfassen sollte. COM und COM+ sind Technologien, die Ihnen helfen, ver-

teilte Applikationen zu erstellen. COM unterstützt Sie dabei, Applikationen aus Komponenten zusammenzustellen, die in verschiedenen Sprachen geschrieben sein können. COM+ ist eine Laufzeitumgebung, die wertvolle Dienste, wie zum Beispiel das Thread-Pooling, deklarative Transaktionen und rollenbasierte Autorisierungsüberprüfungen anbietet.

COM+ und Windows 2000 bilden außerdem eine Plattform, die viele nützliche Dienste zur Verfügung stellt. IIS und ASP sind wichtige Dienste, da sie Ihnen ermöglichen, Funktionalität über HTTP bereitzustellen und eine Applikation über eine Web-Farm-Architektur zu skalieren. MSMQ und QC sind ebenfalls wichtig, da mit ihrer Hilfe die asynchrone und nicht verbundene Kommunikation durchgeführt wird. Wenn Sie all diese Bereiche gemeinsam nutzen, erhalten Sie eine Infrastruktur, um groß angelegte Applikationen zu erstellen, die den Anforderungen der heutigen Geschäftswelt gerecht werden. Es liegt nun an Ihnen, das Erlernte anzuwenden.

Stichwortverzeichnis

A

- Abfangen 15
- Abfangmechanismus 154
- Abonnements 325
 - konfigurieren 329
- Abonnenten 325
 - kurzfristige 331
- Abort 219
- Abort-Methode 219
- Abschlusswarteschlangen 323
- abstrakte Basisklassen 48
- abstrakter Datentyp 34
- Access Control List 262
- ACID-Regeln 209
- ACL 262
- Activate 229
- Active Directory Services Interfaces 252
- Active Directory-Server 290
- Active Server Pages 70, 250
- ActiveX
 - Data Objects 17
 - Steuerelemente 11
- AddRef-Methode 65
- AddressOf 179
- ADO-Bibliothek 17
- ADO-Datensatzgruppen
 - übergeben 392, 397
- ADO-Objekte 17
- ADSI-Schnittstellen 252
- Aktivierung 59
 - prozessexterne 147
- Aktivität 14, 180
- Aktualisierungssperren 243
- Alias 339
- Anmeldungssitzungen 342
- Anweisungen
 - SetErrorInfo 107
- Anwendungen
 - monolithische 11
 - verteilte 22
- Apartments 77
 - threadneutral 178
- API 15
- Application Programming Interface 15
- Application-Objekt 251
- Applikationen
 - transaktionale 216
 - verteilte strukturieren 153

- Applikations-Proxies 151
- Arbeits-Threads 174
- Arrived-Ereignis 302
- ASP 18, 70
 - Anforderungen bearbeiten 253
 - Architektur 250
 - Gerüst 251
 - Seiten 251
 - Skripte ersetzen 256
- AspProcessorThreadMax-Schlüssel 254
- AspRequestQueueMax-Schlüssel 255
- Atomar 209
- Auflistungen 92
- Aufrufkontext 160, 163
- Aufwärtsskalierung 273
- Ausnahmefehler 106
 - Bedingungen 109
- AUTH_PASSWORD-Variable 375
- Authentifizierung 337
- Authentifizierungsebenen 349
 - setzen 351
- AutoComplete-Attribut 228
- Automations-Marshaler 75
- Automatisierung 64
- Autorisierung
 - Laufzeit 361
- Autoritäten 337

B

- Base64/UUEncoded-Codierungsschema 375
- Basisklasse
 - abstrakte 47, 48
- Batch-Anmeldungssitzungen 346
- BeginTransaction 219
 - Methode 306
- benutzerdefiniertes Marshaling 78
- Benutzergruppe 339
- Benutzer-Hauptnamensformate 337
- Benutzeroberfläche
 - erstellen 266
 - Threads 173
- Beziehungen
 - ist ein/e 30
- Bibliotheksapplikationen 146
- Binär-Kompatibilität 117
- Binden
 - dynamisches 32, 49

- Bindung
 - DISPID 72
 - frühe 73
 - späte 69
- Black-Box-Wiederverwendbarkeit 34
- Business Objects 5

C

- C++
 - binäre Komponenten entwickeln 47
- CanBePooled 229
- Casting 66
- Class_Initialize 100
- Class-Factories 62
- ClassID 57
- CLB-Dienst 384
- Client-Anwendung 5
- Cloaking
 - dynamisch 376
- Close 295
- CLSID 57
 - Windows-Registrierung 60
- Cluster Service 383
- CoCreateGUID 116
- CoCreateInstance 59
- CoCreateInstanceEx 59
- Coklasse 50
- COM 11, 46
 - Anforderungen 46
 - Ausnahmefehler 106
 - Bibliothek 58
 - Clients 11
 - Distributed 13
 - Grundlagen 45
 - Klassen 11
 - Methodenaufrufe 391
 - MTS 13
 - Sicherheit 14
 - Threading-Modelle 175
- COM+ 16
 - Applikationen 140
 - Dienste 17
 - Einsatz 2
 - Events 20
 - Architektur 326
 - Komponenten
 - debuggen 164
 - testen 164
 - Programmiermodell 160
 - Registrierungsdatenbank 17
 - Server 2
 - Services Type Library 144
 - Transaktionen 235
 - vs. lokale Transaktionen 238
 - Überblick 2
- COMAdmin 141
- CoMarshalInterface 180
- Commit 219
 - Methode 219

- Component Load Balancing 384
- Component Object Model 11, 45, 46
- Component Services Administration 141
- Computerkonfiguration 14
- COM-Spezifikation 50
- Construct-Methode 146
- Contents-Eigenschaft 394
- Coordinating Transaction Manager 217
- CorrelationID-Eigenschaft 299
- CoUnmarshalInterface 180
- CreateErrorInfo-Funktion 107
- Create-Methode 294
- CreateProperty 190
- CreatePropertyByPosition 190
- CreatePropertyGroup 189
- CreateThread 174
- CTM 217

D

- DACL 294
- Datenbank-Managementsystem 193
- Datenbankverbindungen 234
 - gemeinsame 193
- Datenbankverbindungs-Pooling 193
 - Vorteile 196
- Datentypen
 - abstrakte 34
 - konkrete 34
- DBMS 193, 206
- DCOM 13
- DDE 46
- Deactivate 229
- Deadlocks 242
- Debug-Techniken 167
- Deklarative Sicherheit
 - konfigurieren 359
- Delegation 355
- DHTML 9
- Dienst-Anmeldungssitzungen 345
- Dirty Read 210, 212
- DisableCommit 225
- Discretionary Access Control List 294, 336, 347
- Dispenser 308
- DISPID 70
 - Bindung 72
- Distributed COM 13
- Distributed Transaction Coordinator 206, 218
- DLLHOST.EXE 147
- DNA 10
- Domänenautoritäten 337, 338
- Domänencontroller 338
- Done-Bit 199, 224
 - lesen 227
 - setzen 227
- Doomed-Bit 225
- DTC 206
- DtcGetTransactionManager 219
- DtcGetTransactionManager-Funktion 219
- Duale Schnittstelle 71

Dynamic Data Exchange 46
Dynamic HTML 9
dynamisches Binden 32, 49

E

Eigenschaften
 Contents 394
 CorrelationID 299
 Instancing 88
 MTSTransactionMode 221
 ResponseQueueInfo 303
Empfänger-Applikation 286
EnableNotification-Methode 302
Enumerationen 92
 verwenden 133
Ereignisklassen 325
 erstellen 327
 installieren 328
 konfigurieren 328
Ereignisse
 Arrived 302
 asynchrone 331
Err-Objekt 102
Ersatzprozesse 83
Erweiterbarkeit 25
Events-Dienst 325
externe Transaktionen 305

F

FACILITY_NULL 105
Facility-Code 105
Factory
 Vorlage 62
Failfast-Richtlinie 153
Fehler
 aufwerfen 102
Fehler-Handler 102
Formatnamen 297
Friend-Methoden 92
Funktionen
 CreateErrorInfo 107
 DtcGetTransactionManager 219
 GetErrorInfo 107

G

Geschäftsobjekte 5
geschütztes Sichtbarkeitsniveau 33
Gespeicherte Prozeduren
 Transaktionen 235
GetDeactivateOnReturn 227
GetErrorInfo-Funktion 107
GetIDsOfNames-Methode 69
GetMyTransactionVote 227
GetObjectContext 161

globale eindeutige Bezeichner 56
Globally Unique Identifiers 56
GlobalMultiUse 89
GlobalSingleUse 89
Grid-Steuerelement 397
Grundauthentifizierung 375
Gruppe 339
GUID 56
GUIDGEN.EXE 116

H

Happy-Bit 224, 225
 lesen 227
 setzen 227
Hauptnamen 337
HKEY_CLASSES_ROOT\CLSID 60
HKEY_CLASSES_ROOT\TypeLib 60
HRESULT 103
HTML 8
HTTP 8
HTTP-Anforderungen 17
HTTPS 373
HTTP-Sicherheitsstandards 370
Hypertext Markup Language 8
Hypertext Transfer Protocol 8

I

IContextState 162
 Schnittstelle 227
ID-Eigenschaft 299
Identitätswechsel 354
 einstellen 354
IDispatch 64
 Schnittstelle 69
IDL 52
 Datentypen 131
 Quelldatei 52
IID 57
IIS 17
 Admin-Objekte 252
 API 18
 Applikation 252
 isolierte 263
 Architektur 250
 Dokumentation 252
 Metadatenbank 252
IIS In-Process Applications 252
IISIntrinsics-Attribut 267
IISWAM.W3SVC 252
IMarshal 78
IMB MQSeries 10
IMDB 192
Implementierungsvererbung 25, 29
 Probleme 32
INETINFO.EXE 252
Initialize 228
In-Memory Database 192

- InprocServer32-Schlüssel 60
- Instancing-Eigenschaft 88
- Instanz-ID 294
- Interception 15
- Interface Definition Language 52
- Interface ID 57
- interne Transaktionen 305
- Internet Information Services 17, 250
- Internet Server API 18, 250
- Interprozess-Kommunikation 46
- Invoke-Methode 70
- IOBJECTCONSTRUCT 146
- IOBJECTCONSTRUCTSTRING 146
- IPersistStorage 395
- IPersistStream 395
- ISAPI 18, 250
- ISAPI-Erweiterungen 18, 250
- ISAPI-Filter 18, 250
- Isolationsebenen 212
 - Read Committed 212
 - Read Uncommitted 212
 - Repeatable Read 212
 - Serializable 213
- ISupportErrorInfo-Schnittstelle 107
- IsWorldReadable 294
- ITransactionDispenser-Schnittstelle 219
- ITransaction-Schnittstelle 219, 223
- IUnknown 64
- IUnknown-Schnittstelle 64
- IUSR_MachineName 262
- IWAM_Machine 369

J

- JavaScript 9
- JustInTimeActivation-Attribut 199
- Just-In-Time-Aktivierung 199

K

- Kanäle
 - sichere 348
- Katalog-Manager 141
- Kausalität 367
 - ID 163
- keine Kompatibilität 117
- Klasse
 - übergeordnete 29
 - untergeordnete 29
- Klassendefinition
 - erweitern 42
- Klassen-ID 57
- Kommunikation
 - bidirektionale 322
- Komponenten 51
 - Architektur 46
 - binäre, mit C++ entwickeln 47
 - Dienste 141, 145
 - Kompilieren 81

- konfigurierte 17, 140
 - konfigurierte debuggen 165
 - nicht-konfigurierte 17
- Komponentenversionsverwaltung 113
 - Scripting-Clients 115
 - vTable-Clients 115
- Konfigurierte Komponenten
 - debuggen 165
 - einsetzen 143
 - erstellen 143
- konkreter Datentyp 34
- Konten
 - vorkonfigurierte 340
- Kontexte 77, 154
- Kontextwechsel 155

L

- Laufzeitfehler 104
- Laufzeittypenüberprüfung 26
- LCE 325
- Lesesperre 211
- Listener-Anwendung 19
- Listener-Applikationen 289
 - erstellen 303
- Local Queue Store 292
- Local Security Authority 337
- LocalServer32-Schlüssel 60
- LockMethod 190
- LockSetGet 190
- Lokaler Warteschlangenspeicher 292
- LookupQueue-Methode 295
- Loosely Coupled Events 325
- LQS 292
- LSA 337

M

- Main-Threaded-Komponente 179
- Marshaling 73
 - benutzerdefiniert 78, 392
 - Standard 78
- MCS 372
- Messaging 9
 - Infrastruktur 10
 - transaktionales 304
- Methoden
 - Abort 219
 - AddRef 65
 - BeginTransaction 306
 - Close-Methode 295
 - Commit 219
 - Create 294
 - EnableNotification 302
 - GetIDsOfNames 69
 - Invoke 70
 - LookupQueue 295
 - Open 295
 - Peek 301

- Methoden (*Fortsetzung*)
 - PeekCurrent 301
 - PeekNext 301
 - QueryInterface 66
 - ReceiveCurrent 301
 - Release 65
 - Save 396
 - Send 298
 - SetAbort 226
 - SetComplete 226
 - WriteProperty 394
- Methodensignaturen
 - definieren mit IDL 131
- Microsoft Certificate Server 372
- Microsoft IDL 52
- Microsoft Message Queue 2.0 Object Library 292
- Microsoft Message Queuing Services 10, 18
- Microsoft Transaction Server 13
- Microsoft Visual InterDev 18
- MIDL-Compiler 52
- Moniker 320
- MQ_DENY_NONE 296
- MQ_DENY_RECEIVE_SHARE 296
- MQ_MTS_TRANSACTION 309
- MQ_PEEK_ACCESS 296
- MQ_RECEIVE_ACCESS 295
- MQ_SEND_ACCESS 296
- MSMQ 10, 18, 290, 393
 - Katalog 290
 - Nachrichtenkörper 298
 - Nachrichtenpriorität 299
 - Organisationseinheiten 290
 - Transaktionen
 - externe 308
 - interne 306
- MSMQ 2-Arbeitsgruppenmodus 290
- MSMQ 2-Enterprise-Modus 290
- MSMQMessage-Objekt 298
- MSMQQueueInfo-Objekt 294
- MSMQTransactionDispenser-Objekt 306
- MSMQTransaction-Objekt 306
- MTS 13
 - Sicherheit 14
- MTSTransactionMode 221
- MTSTransactionMode-Eigenschaft 221
- Multiobjekt-Transaktionen 230
- Multi-Tier
 - Architektur 7
 - Plattform 10
 - Systeme 2
- MultiUse 88
- MultiUse-Klasse 51

N

- Nachrichten
 - empfangen 299, 301
 - inkorrekte 310
 - Körper 286, 298
 - lesen 299

- Priorität 299
- Pumpen 173
- senden 298
- Netzwerk
 - Anmeldungssitzungen 344
 - Authentifizierungsdienste 348
 - Verwaltung 14
- New 26
 - Moniker 320
 - Operator 158
- NotAnMTSObject 221
- NoTransactions 221

O

- Object
 - Datentyp 70
- Object Definition Language 128
- Object Linking and Embedding 11, 46
- Object RPC 13
- ObjectControl-Schnittstelle 228
- Objekt
 - Aktivierung 59
 - erweiterbare 28
 - instantiieren 26
 - Komposition 33
 - öffentliche Schnittstelle 26
 - plug-kompatible 32
 - poolfähig 199
 - sekundär 222
- Objektkonstruktor-Zeichenfolgen 145
- Objektkontext 160
 - programmieren 161
- objektorientiert 11
- Objekt-RPC-Methodenaufrufe 163
- ODL 128
- OLE 11, 46
 - Transactions-Protokoll 218
- OLE32.DLL 59
- OLEAUT32.DLL 75
- Oleview.exe 53
- OLTP 206
- On Error 102
- Online Transaction Processing-Systeme 206
- OOP 11, 25, 26
- Open-Methode 295
- ORPC 13, 163
- örtliche Transparenz 74, 146
- Ortsunabhängigkeit 146

P

- Parallelität 14, 174
- Parallelitätsmodell
 - aktivitätsbasiert 180
- Participating Transaction Manager 217
- PeekCurrent-Methode 301
- Peeking 287
- Peek-Methode 301

- PeekNext-Methode 301
- Ping-Algorithmus 153
- Polymorphismus 26, 31
- PostMessage 177
- Präsentationsebene 5
- Präsentationsschicht 8
- Private 89
- Private\$ 296
- ProgID 63
 - versionabhängige 63
- Programmierung
 - attributbasierte 15
 - objektorientierte 26
 - schnittstellen-basierte 12
 - statuslose 236
- Projekt-Kompatibilität 117
- Property Get 37
- Property Let 37
- PropertyBag-Objekt 394
- Protokoll
 - HTTP 8
 - RPC 7
 - synchron 9
- Proxy 73
- Prozeduattribute
 - ändern 91
- Prozesse 173
- Prozessexterne Server 82
- Prozessinterne Server 82
- PublicNotCreatable 88
- Publisher 21

Q

- QC 393
- QC.Listener 314
- QC.ListenerHelper 314
- QC.Recorder 314
- QueryInterface 96
- QueryInterface-Methode 66
- Queued Components 20, 311
- Queue-Moniker 320

R

- Raise 102
- Rangebene 104
- ReadProperties 395
- ReceiveCurrent-Methode 301
- ReceiveTimeout 300
- Recorder 20, 313
- Referenzen
 - klassenbasierte 26, 28
- Refresh Components 147
- RegDB 17, 141
- Registrierung
 - Probleme 85
- Registrierungsdatenbank 141
- Registrierungsformat 57

- REGSVR32.EXE 61
- Release-Methode 65
- Remote
 - Aktivierung 149
 - Kommunikation 73
 - Server 82
 - Server-Dateien 127
- Remote Procedure Call 7, 13
- Remote Procedure Calls 73
- Repeatable Read 212
- RequiresNewTransaction 221
- RequiresTransaction 221
- ResponseQueueInfo-Eigenschaft 303
- Ressourcen-Manager 216
- retval 103
- RM 216
- Rollback 209
- Rollen 14, 357
- Root
 - Applikation 252
 - Objekt 222
- Round-Robin-DNS 275
- RPC 7, 13, 73
- RunAt=Server-Attribut 256

S

- SACL 347
- SAFEARRAY 132
- SafeRef 160
- Save-Methode 396
- SCHANNEL 372
- Schnittstellen 25, 34
 - benutzerdefinierte erstellen 126
 - duale 71
 - ID 57
 - IDispatch 69
 - IDL-Definition 127
 - ISupportErrorInfo 107
 - ITransaction 219, 223
 - ITransactionDispenser 219
 - IUnknown 64
 - öffentliche 26
 - abändern 28
 - Vererbung 35
 - verwenden mit Visual Basic 36
 - Weiterleitung 121
- Schreibsperre 211
- SCM 59
- Secure Channel 372
- Secure HTTP 373
- Security Access Control List 347
- Security Reference Monitor 348
- Security Support Provider 348
- Security Support Provider Interface 348
- Seitensperre 211
- Send-Methode 298
- Server 51
 - Applikationen 146
 - Design 86

- Server (*Fortsetzung*)
 - erstellen 84
 - lokaler 82
 - prozesseextern 82
 - prozessintern 82
 - registrieren 61
- ServerVariables-Auflistung 375
- Service Control Manager 59
- Session_OnStart-Ereignis 274
- Session-Objekt 251
- SetAbort-Methode 226
- SetComplete-Methode 226
- SetDeactivateOnReturn 227
- SetErrorInfo-Anweisung 107
- SetMyTransactionVote 227
- Shared Property Manager 160, 188
- Sicherheit
 - COM+ 357
 - IIS 368
 - programmgesteuerte 363
 - Windows 2000 336
- Sicherheitsautoritäten
 - lokale 337
- Sicherheitsdeskriptoren 347
- Sicherheitsgrenze 338
- Sicherheitshauptnamen 337
- Sicherheits-ID 337
- Sicherheitsreferenzüberwachung 348
- Sicherheitsunterstützungs-Provider 348
- Sicherheitszugriffskontrolllisten 347
- SID 337
- Simple Object Access Protocol 389
- Singleton 183
- SingleUse 89
- Site Server 375
- Sitzungskomponente 241
- Sitzungsschlüssel 348
- Skripting-Clients 102
 - Komponenten 95
- SOAP 389
- sp_addmessage 208
- späte Bindung 69
- Speicher
 - gemeinsamer 186
- Speicherbereinigung
 - verteilte 152
- Sperren
 - Aktualisierungs- 243
 - exklusive 211
 - gemeinsame 211
 - Konvertierungen 243
- Sperrkonvertierungen 243
- Sperrungen
 - Schreibsperre 211
- SPM 160, 188
 - Grenzen 192
- Sprachunabhängigkeit 46, 51
- SQL Server 206
- SRM 348
- SSL-Authentifizierung 373
- SSP 348
- SSPI 348

- Standard-Marshaling 78, 391
- Standardmodulvariablen 187
- Standardrückgabewert 104
- Standardschnittstelle 96
- Standardschnittstellen 54
 - erweitern 120
- STA-Thread-Pool 184
- Status
 - applikationsweiten cachen 283
 - übergeben 280
- statuslose Programmierung 236
- Stream-Objekt 404
- Stub 73
- Subklasse 29
- Subscriber 21
- Subscription
 - persistent 21
 - transient 21
- Superklasse 29
- Synchronization-Attribut 181
- System-Anmeldungssitzungen 343
- Systemdienste 15

T

- Tabellen
 - virtuelle 50
- Tabular Data Streams 288
- TDS 288
- ThreadingModel-Attribut 178
- Threading-Modelle 175
- Threads 173
 - Affinität 175
 - Sharing 172
- Thread-Speicher
 - lokaler 162
- Thread-Wechsel-Proxies 155
- TNA 178
- Tokens 342
- Transaction
 - Attribut 220
- Transaction Manager 216
- Transact-SQL 206
- Transaktionen 206
 - Atomar 209
 - ausführen 309
 - Bedingungen 209
 - Beständig 209
 - deklarative 220
 - externe 305
 - gespeicherte Prozeduren 235
 - interne 305
 - Isoliert 209
 - Konsistent 209
 - lokale oder COM+ 238
 - Multiobjekt- 230
 - programmieren 213
 - Root-Objekt 222
 - rückgängig machen 225
 - verteilte 215

- Transaktions-Manager 216
- Transaktionsmanager
 - koordinierender 217
- Transaktionssysteme 208
- Transaktionsunterstützung 220
- Two-Phase-Commit 216
 - Protokoll 216
- Two-Tier
 - Anwendungen
 - Nachteile 2
 - Systeme 2
- Typbibliotheken 51
 - Attribute 87
 - eigenständige 85
 - kompilieren 135
 - konfigurieren 135
 - verteilen 135
- Typbibliothek-Marshaler 75
- Typ-Casting 66

U

- übergeordnete Klasse 29
- Überwachungscode
 - hinzufügen 365
- UDT-Parameter 93
- UDTs 81, 93
 - verwenden 133
- Universal-Marshaler 75, 390
- unsicheres Superklassen-Szenario 33
- untergeordnete Klasse 29
- UPN 337
- User Defined Types 81
- User Principal Name 337
- UsesTransaction 221
- UUID-Eigenschaft 57

V

- Vererbung 35
- Vererbungshierarchie 30
- Verleger 325
 - erstellen 330
 - ID 328
- Verschlüsselung
 - asymmetrische 370
 - symmetrische 370
- Versender-Applikation 286
- version compatibility 114
- versionskompatibel 120
- Versionskompatibilitätseinstellungen 117
- Vertrauenspfad 339
- Verweiszähler 65
- Verzeichnis
 - virtuelles 252
- virtuelle Tabelle 50
- Visual Basic
 - verteilte Anwendungen 22

- Visual Basic-Komponenten
 - debuggen 261
- Visual Basic-Objekte
 - erstellen 256
- Visual InterDev 18
- vTable 50
- vTable-Bindung 50
 - Bedingungen 72

W

- WAM 252
- WAM-Komponenten 369
- Wartbarkeit 25
- Warteschlangen 286
 - Komponenten 20, 311
 - Dienst 311
 - Einschränkungen 315
 - Fehlerbehandlung 322
 - konfigurieren 317
 - programmieren 319
 - Listener-Anwendung 19
 - lokalisieren 297
 - Manager 291
 - private 296
- Web Application Manager 252
- Web Service 389
- Webapplikationen
 - skalieren 273
 - unterteilen 259
- Web-Farmen 274
- Webgruppen 274
- Webserver-Arrays 274
- White-Box-Wiederverwendbarkeit 33
- Wiederholungswarteschlangen 323
- Wiederverwendbarkeit 25
 - Black-Box 34
- Win32-Threading 173
- Windows 2000 16
 - Site 291
- Windows Distributed interNet Applications
 - Architecture 10
- Windows DNA 10
- Windows Scripting Host 70
- Windows-Authentifizierung 374
- Windows-Registrierung
 - CLSID 60
- WINERROR.H 106
- WriteProperties 395
- WriteProperty-Methode 394
- WSH 70

X

- X/Open 218
- XML 1, 248
- XSL 404

Z

- Zertifikate 371
 - Authentifizierung 374
- Zirkelverweise 66
- Zugriff
 - anonym 341
- Zugriffsmodus
 - anonym 373
- Zugriffssteuerungsliste
 - wahlweise 294
- Zugriffssteuerungslisten 262
- Zuordnung
 - automatische 234

