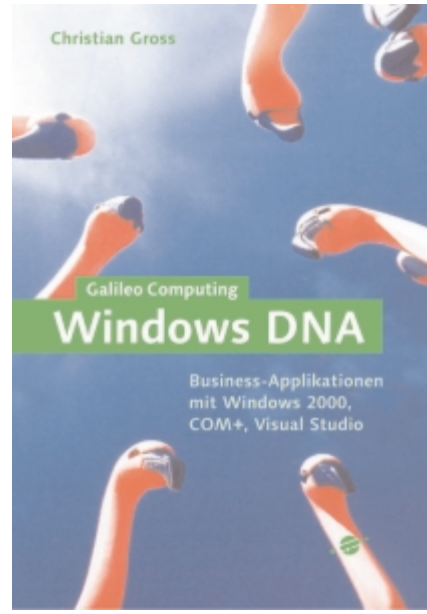

Christian Gross

Windows DNA

E-Business-Applikationen

dows 2000, COM+, Visual Studio

Galileo Press

Inhalt

1 Einführung in Windows DNA

Wenn Sie dieses Buch lesen, sind Sie wahrscheinlich ein Softwareentwickler, der Informationen zu Windows 2000 und Windows DNA benötigt. Sie suchen nach einer Lösung, die sämtliche Dienste und COM-Objekte (Component Object Model) beinhaltet, die für Ihre Arbeit erforderlich sind. Und genau dies werden Sie in diesem Buch finden. Sie werden in die Entwicklung verteilter Windows 2000-Anwendungen eingeführt. Und Sie lernen die Bedeutung von Windows DNA kennen.

Windows DNA steht für Windows Distributed Internet Applications Architecture, also die Architektur für verteilte Windows-Internetanwendungen. Die Marketingstrategie von Microsoft hat sich zwar geändert, aber Windows DNA ist weiterhin das, was Microsoft ursprünglich folgendermaßen beschrieben hat:

Ein Lösungsansatz, mit dem Unternehmensentwickler und unabhängige Softwarehändler unter Verwendung grundlegender Windows-Technologien verteilte Unternehmensanwendungen entwerfen und entwickeln können.

Dies bedeutet, dass Microsoft momentan und in Zukunft versucht, die besten Ansätze zur Lösungsentwicklung mit den Microsoft-Technologien zu definieren. Der Grund, aus dem Windows DNA- und Windows 2000-Lösungen sich gleichen, ist der, dass Windows 2000 sämtliche Dienste bereitstellt, die in einer vollständigen Windows DNA-Anwendung benötigt werden.

Ich werde das DNA-Modell besprechen, darüber hinaus jedoch auch die Verwendung der Technologien erläutern, die am besten zur Lösung spezieller Kundenprobleme geeignet sind. Ich habe über zehn Jahre lang in einer Unternehmensumgebung mit den Microsoft-Technologien gearbeitet, und das Wichtigste, was ich dabei gelernt habe, ist Folgendes: Die Arbeit mit den Microsoft-Technologien kann zu guten Ergebnissen führen.

Die Windows DNA, von der ich hier spreche, eignet sich optimal zur Lösung von Kundenproblemen. Viele Konzepte der Windows DNA stehen auch auf anderen Plattformen zur Verfügung, beispielsweise mit Java oder CORBA (Common Object Request Broker Architecture). Ich denke, es wird Ihnen gefallen.

1.1 Die Wichtigkeit des Kontextes

Wenn ich ein Problem angehe, versuche ich stets, den Kontext des Problems zu erfassen. Mit Kontext ist hierbei nicht nur die gegenwärtige Situation gemeint, sondern auch die Umstände, die zur aktuellen Situation geführt haben. Im Umgang mit vertikalen Anwendungen ist es häufig so, dass Entscheidungen nicht nur aufgrund technischer Aspekte, sondern auch auf der Grundlage persönlicher oder anderer Gründe getroffen werden. Und diese persönlichen Entscheidungen sind schwer einschätzbar. Manchmal sind sie logisch, manchmal jedoch auch nicht. Die Entscheidung sollte auf dem Kontext des Problems basieren.

Im Hinblick auf das vorliegende Buch gehört zu diesem Kontext u.a., ein Problem mit Hilfe von Windows DNA zu lösen.

Lassen Sie uns zusammen den Kontext erarbeiten, der Windows 2000 zu dem Betriebssystem gemacht hat, das es heute ist. Nehmen Sie folgende Aussage:

Windows 2000 ist stabil.

Haben Sie gelacht? Ich möchte wetten, dass viele von Ihnen gelacht haben. Wir haben alle die Horrorstories vom BSOD (Blue Screen of Death) von **Windows NT** gehört. Sie denken vielleicht, dass Windows 2000 aufgrund seiner Ablaufverfolgung niemals stabil sein kann. Wenn Sie sich letztere jedoch genauer ansehen und sich klarmachen, was Microsoft damit erreichen wollte, werden Sie erkennen, warum einige Dinge genau so umgesetzt wurden, wie sie umgesetzt wurden. Und dann werden Sie auch verstehen, wie es zu Windows DNA gekommen ist.

Windows und Microsoft gibt es schon einige Jahre. DOS war erfolgreich, aber erst mit Windows 3 kam der Durchbruch für Microsoft. Also lassen Sie uns betrachten, wie Windows zu dem wurde, was es heute ist.

1.1.1 DOS – Die erste Ära

Als DOS herausgebracht wurde, war der größte Vorteil die Tatsache, dass ein kostengünstiger Rechner bereitgestellt wurde, der durch eine Einzelperson programmiert werden konnte. Es waren weder Expertenteams noch das umfangreiche Budget großer Unternehmen erforderlich. DOS (Disk Operating System) konnte auf einem Gerät ausgeführt werden, der als Personal Computer (PC) bezeichnet wurde. Keine dieser Innovationen war jedoch wirklich neu. Apple entwickelte einen eigenen PC sowie das zugehörige Betriebssystem, und auch CP/M von Digital war eine Art Festplattenbetriebssystem. Die Neuheit war, dass DOS auf kostengünstiger, generischer Hardware ausgeführt werden konnte, und das DOS selbst ebenfalls nicht teuer war.

1.1.2 Windows 3.x – Die zweite Ära

Als die erste Ära PCs den Markt eroberte, wurde schnell klar, wo die Probleme lagen. Hardware gab es im Überfluss, generische Software mit den geeigneten Treibern dagegen war rar. Die Computerhersteller verbesserten ständig die Rechnergeschwindigkeit und statteten die PCs mit mehr Festplatten- und Arbeitsspeicher aus, aber um diese Hardwareinnovationen nutzen zu können, benötigten die Anwendungen benutzerdefinierte Treiber. Jede Anwendung hatte eigene Treiber, die in den meisten Fällen nicht mit anderen Anwendungen kompatibel waren.

So bestand der Kontext in einer größer werdenden Menge proprietärer Software, die keine ausreichende Kompatibilität mit anderen Produkten aufwies. Die Lösung für dieses Problem war Windows. Windows abstrahierte das Konzept eines Gerätetreibers, sodass das Betriebssystem dieses verstehen konnte. Auf diese Weise konnte der Benutzer Software unabhängiger Hersteller einsetzen. Jeder konnte beliebige Softwarekomponenten einsetzen, immer vorausgesetzt, es war ein entsprechender Windows-Treiber vorhanden. Die Hardwarehersteller brauchten lediglich einen Treiber zu entwickeln, der für alle Anwendungen verwendet werden konnte.

1.1.3 32-Bit-Windows – Die dritte Ära

Als die zweite Computer-Ära fortschritt, förderte dies neue Probleme zutage. Die zweite Computer-Ära basierte auf den technologischen Konzepten der ersten Ära. So verwendeten Anwendungen 16-Bit-Prozessorbefehle, es gab kein Multitasking und es gab keinen Anwendungsfehlerschutz. Anwendungsfehler verursachten für einen Anwendungsabstürze, die wiederum häufig zu einem Absturz des gesamten Windows-Betriebssystems führten.

Dieses Problem resultierte aus dem Treibermodell, das in der zweiten Ära eingeführt wurde. In der ersten Ära erstellte jeder Entwickler eigene Systeme. Diese übernahmen die vollständige Steuerung von Prozessor, Festplatte und Bildschirmanzeige. In der zweiten Ära mussten die Anwendungen diese Ressourcen gemeinsam nutzen, aber viele Entwickler schrieben keine Anwendungen oder Treiber, die dieses Konzept auch wirklich umsetzten. Selbst Windows legte die Schnittstellen nicht richtig offen, daher führte die gemeinsame Ressourcennutzung zu Fehlern. Das Entwickeln von Systemen für die gemeinsame Nutzung ist schwierig, da es ein gewisses Vertrauen in die anderen Anwendungen und deren ordnungsgemäße Funktion erfordert.

Die Antwort auf diesen Kontext der gemeinsamen Ressourcennutzung war die Einführung des Multitasking und der Schutz des Anwendungsbereichs. Der Prozessor gaukelte den Anwendungen vor, dass sie sich weiterhin in der ersten Ära

befanden und somit die exklusive Steuerung der Umgebung innehatten. Der Prozessor setzte diese gemeinsame Nutzung auf Chipebene um. Diese geschützte gemeinsame Ressourcennutzung machte das Betriebssystem stabil und gleichzeitig schneller und effektiver.

1.1.4 Windows DNA – Die vierte Ära

Damit befinden wir uns nun in der vierten Generation. Anwendungen können sicher in einem eigenen, geschützten Bereich ausgeführt werden. Was weiterhin benötigt wird, ist eine Möglichkeit, mit der Anwendungen einen gemeinsamen Knotenpunkt nutzen können, ohne dass dies zu Konflikten führt. Die Lösung findet sich in der gemeinsamen Nutzung von Diensten. Die gemeinsame Nutzung mit anderen Systemkomponenten ist extrem schwierig zu steuern, da bei einem so weit verbreiteten System wie Windows nicht jeder die gleiche Vorstellung davon hat, wie die Dinge funktionieren sollten.

Aus diesem Grund entwickelte jeder eine eigene Version der verschiedenen Dienste. Einige dieser Dienste basierten auf Datenbanken. Andere basierten auf dem Web. Wieder andere basierten auf Transaktionen. Und jeder dieser Dienste verfügte über eine eigene Schnittstelle. Sobald ein Benutzer seine gemeinsam genutzten Dienste aktualisierte, führte dies zu Konflikten und Systemabstürzen. Die Benutzer denken häufig, dass Windows instabil ist, weil es schlecht geschrieben ist. Tatsächlich kann Windows jedoch häufig aufgrund der Unmenge an Anwendungen instabil werden, die gemeinsam das Betriebssystem nutzen und hierbei gewisse Kompromisse erforderlich machen. Solange die gemeinsame Nutzung von Komponenten ordnungsgemäß erfolgt, gibt es auch keinerlei Probleme.

Zur Lösung dieses Problems sieht die Windows DNA-Strategie u.a. die Offenlegung eines gemeinsamen Satzes an Diensten und Hilfsprogrammen vor, durch die der zuvor genannte Knotenpunkt implementiert wird. Diesen Knotenpunkt stellen die Kerndienste dar, beispielsweise Transaktions- und Nachrichtendienste sowie weitere Komponenten, die in verschiedenen Anwendungen wieder verwendet werden können. So wird das Konzept der gemeinsamen Nutzung vereinfacht, denn es ist nur ein Dienstesatz vorhanden.

Das Ziel von Windows DNA ist die Bereitstellung einer generischen Lösung mit den folgenden Leistungsmerkmalen:

- Interaktion mit dem Benutzer über Tastatur, Bildschirm oder Maus
- Mögliche Verwendung andere Eingabegeräte, beispielsweise Touchscreen, Pen oder Sprachaktivierung
- Verbindung zu weiteren Rechnern mit Hilfe von Netzwerkkonzepten

- Zuverlässige und konsistente Datenfreigabe für mehrere Computer
- Eigene Verwaltung sowie einfache Umgebungsverwaltung

Diese Art Lösung erfordert verschiedene Voraussetzungen, die aus der Ära des Windows-APIs (Application Programming Interface) stammen. Bis vor einiger Zeit konzentrierten sich die Hersteller auf ihre eigenen, spezifischen Anwendungen. Dieser allgemeine Trend hat sich jedoch gewandelt, da der Computer zu einem Teil des täglichen Lebens geworden ist und nicht mehr nur Aufgaben lösen, sondern auch mit anderen Geräten und Umgebungen interagieren muss. Der Verbraucher möchte, dass sein Computer genauso einfach zu bedienen ist wie beispielsweise der Toaster oder das Radio.

1.2 Die Windows DNA-Strategie

In Anlehnung an die Microsoft-Literatur kann die Windows DNA folgendermaßen definiert werden:

> *Die Kombination aus einem Betriebssystem mit konsistentem Satz wieder verwendbarer Dienste und einem logischen Entwurfsprozess zur Entwicklung von Anwendungen*

Diese Definition umfasst drei wichtige Aspekte. Der erste Aspekt ist das Betriebssystem. Das Betriebssystem ist ein fundamentaler Aspekt des Computersystems, da mit diesem die Grundlage für das gesamte Framework bereitgestellt wird. Dieses Betriebssystem sollte schnell, stabil und skalierbar sein. Der zweite Aspekt ist der, dass das Betriebssystem über eine Reihe von Diensten und Hilfsprogrammen verfügt. Diese Dienste ermöglichen die Ausführung bestimmter Aufgaben, beispielsweise Transaktionsverarbeitung oder Messaging. Und als drittes sollten Betriebssystem und Dienste so miteinander kombiniert werden, dass Anwendungen logisch entwickelt werden können und gleichzeitig den größten Nutzen aus den Diensten und Hilfsprogrammen ziehen.

Eine Windows DNA-Lösung wird typischerweise von Organisationen eingesetzt, die Anwendungen zur Lösung spezieller Unternehmensprobleme bereitstellen. Microsoft stellt den Knotenpunkt bereits zur Verfügung. Die Größe des Unternehmens, die diese Lösung einsetzt, spielt keine Rolle – kleine und große Unternehmen haben häufig die gleichen Probleme, diese unterscheiden sich lediglich in der Dimension.

1.2.1 Die Grundlagen von Window sDNA

Windows DNA weist die folgenden Hauptmerkmale auf:

- **Internetfähigkeit:** Internettechnologien, wie beispielsweise TCP/IP, HTTP, FTP usw., sind in das Betriebssystem integriert. Auf diese Weise können Anwendungen die Internetfähigkeit voraussetzen.
- **Kurze Entwicklungszeiten:** Integrierte Dienste und Hilfsprogramme ermöglichen dem Entwickler, sich auf das Schreiben der Anwendung statt auf die Bereitstellung des bereits genannten »Knotenpunktes« zu konzentrieren, wodurch die Entwicklungszeiten erheblich verkürzt werden.
- **Interoperabilität:** Offene Protokolle und Standards ermöglichen die leichte Kommunikation und Interaktion von Anwendungen auf verschiedenen Plattformen, sodass eine Lösung mit Produkten verschiedener Hersteller eingesetzt werden kann.
- **Geringere Komplexität:** Gemeinsam genutzte Dienste und Hilfsprogramme sind in das Betriebssystem integriert, neue Versionen werden mit den Betriebssystemaktualisierungen bereitgestellt. Auf diese Weise wird vermieden, dass durch die Setup-Programme der verschiedenen Anwendungen inkompatible Versionen derselben Dienste installiert werden. Gleichzeitig erleichtert dies den Installationsvorgang für Betriebssystem und Programme.
- **Sprachunabhängigkeit:** Eine Stärke der Windows-Plattform ist die Möglichkeit, dass Entwickler von Fremdanbietern die bevorzugte Entwicklungsumgebung verwenden können. Dies bedeutet, dass Entwickler die Programmiersprache wählen können, die zur Lösung des gegebenen Problems eingesetzt wird.
- **Senkung der Gesamtbetriebskosten:** Die Anwendung sollte einfach bereitgestellt, verwaltet und erneut verwendet werden können. Dies kann durch eine Vereinfachung der Installation und Prozesse zur Versionssteuerung erreicht werden.

1.3 Die Windows DNA-Architektur

Windows DNA verfügt über eine Schichtenarchitektur. Eine Schicht ist eine logische Abtrennung einer funktionellen Komponente von den weiteren funktionellen Bestandteilen. Die Schicht kann ein abstraktes Konzept darstellen oder die tatsächlichen Komponenten eines Projekts reflektieren.

Abbildung 1.1 zeigt die Windows DNA-Architektur, die zur Entwicklung mehrschichtiger Anwendungen eingesetzt wird. Es sind drei Schichten vorhanden, die Darstellungsschicht, die Schicht mit der Anwendungslogik und die Datenschicht. Zwischen der Darstellungsschicht und der Schicht der Anwendungslogik befindet

sich eine Firewall. Diese Architektur stellt keine physische Architektur, sondern ein abstraktes Konzept dar. Innerhalb einer der abstrakten Schichten können mehrere physische Schichten enthalten sein.

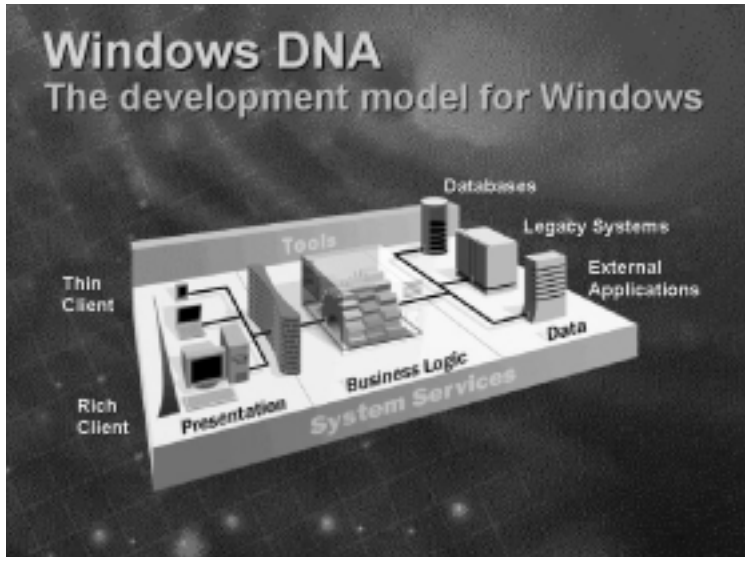

Abbildung 1.1 Die Windows DNA-Architektur

Damit die gesamte Architektur einer Windows DNA-Anwendung funktioniert, müssen die verschiedenen Schichten, aus denen sich die Anwendung zusammensetzt, miteinander kommunizieren können, ohne die individuellen Implementierungsdetails zu kennen. Diese Kommunikation findet anhand der COM-Technologie statt.

1.3.1 COM überall

Die Grundlage von COM ist ein binärer Standard, über den zwei Codeabschnitte Informationen austauschen können. Abbildung 1.2 zeigt eine so genannte VTable (Virtual Functions Table). Die VTable umfasst eine Reihe von Funktionszeigern auf Codeimplementierungen, die VTable-Definition kann auf einer beliebigen Plattform vorliegen, da sie plattformunabhängig generiert wird. Da die VTable generisch ist, wird sie als Schnittstelle bezeichnet. Durch diese Tabelle werden spezifische Intentionen definiert; anschließend verweist die Schnittstelle auf Implementierungen, mit deren Hilfe die beabsichtigten Aktionen umgesetzt werden.

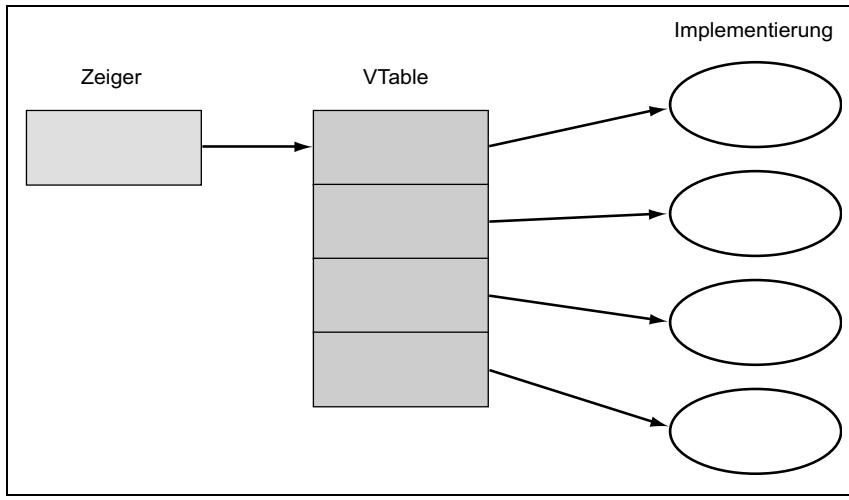

Abbildung 1.2 COM-Schnittstelle und Implementierungsverfahren

Die Implementierung erfolgte traditionell binär und plattformspezifisch, mit Visual J++ wird diese Einschränkung jedoch aufgehoben. Visual J++ erzeugt Java-Code, der auf einer beliebigen Plattform ausgeführt werden kann. Die einzige Bedingung ist, dass die COM-Laufzeitumgebung auf der Zielplattform vorhanden sein muss.

Bei der Entwicklung von Windows DNA-Anwendungen untersuchen Sie zunächst den Problemkontext. Anschließend formulieren Sie die für die Anwendung zu erreichenden Ziele. Mit anderen Worten, Sie definieren eine Reihe von Schnittstellen. Die Implementierungen können die angestrebten Ziele einer oder mehrerer Kontexte umsetzen. Mit Hilfe von COM wird abgefragt, ob eine Implementierung ein zu erreichendes Ziel in einem spezifischen Kontext unterstützt.

Windows 2000 im Kontext von Windows DNA erfüllt den gleichen Zweck. Das Betriebssystem legt eine Reihe von Diensten, z. B. Sicherheit, Transaktionsverarbeitung, Messaging und Hardwareunterstützung, als einen Satz COM-Objekte offen. Diese Tatsache steht im Gegensatz zu früheren Windows NT-Versionen, bei denen die Mehrzahl der Dienste als APIs offen gelegt wurde.

Warum also COM? Neben technischen Gründen stellt COM die derzeit umfangreichste Komponentenstrategie dar. Durch COM erhält Windows DNA mehr Kompatibilität. Bei Verwendung von COM können Softwarekomponenten entwickelt werden, die auf einer beliebigen Schicht ausgeführt und bereitgestellt werden können. Die COM-Laufzeitumgebung bietet Unterstützung für das Packen und Partitionieren sowie für weitere Aspekte mehrschichtiger Anwendungen.

1.3.2 Darstellungsschicht

Die Darstellungsschicht ist die Schicht, in der die Anwendungen mit dem Benutzer interagieren. Dieser Schritt erfolgt, um die Daten in einer Form zu präsentieren, die von uns verstanden werden kann. Aktuell basiert die Darstellungsschicht weitestgehend auf Tastatur, Bildschirm und Maus, in Zukunft muss sich dies jedoch ändern, um beispielsweise der Kommunikation per Spracheingabe oder über Bewegungen gerecht zu werden.

In der aktuellen Version von Windows DNA liegt das Hauptziel der Darstellungsschicht darin, überall und jederzeit jede Art von Zugriff zu bieten. Dies wird durch die Internetintegration erreicht, einschließlich Browser, Einwahl- und Netzwerkkomponenten. Der Entwurf dieser Schicht stellt eine Herausforderung dar, da die verschiedenen Clienttypen sich stark voneinander unterscheiden. In der Windows DNA-Architektur ist der Client nicht notwendigerweise Windows-basiert, kann jedoch UNIX-, Mac OS- und weitere Betriebssysteme umfassen.

Zur Unterstützung dieser verschiedenen Plattformen reicht die Darstellungsschicht von einem Rich Client bis zu einem Thin Client. Sie dürfen sich einen solchen Thin Client jedoch nicht wie im Rahmen der Downloadmenge in Byte vorstellen. Ein Thin Client kann mehrere Hundert Megabyte umfassen. Der Unterschied besteht im Funktionsumfang, der auf der Clientseite ausgeführt werden kann.

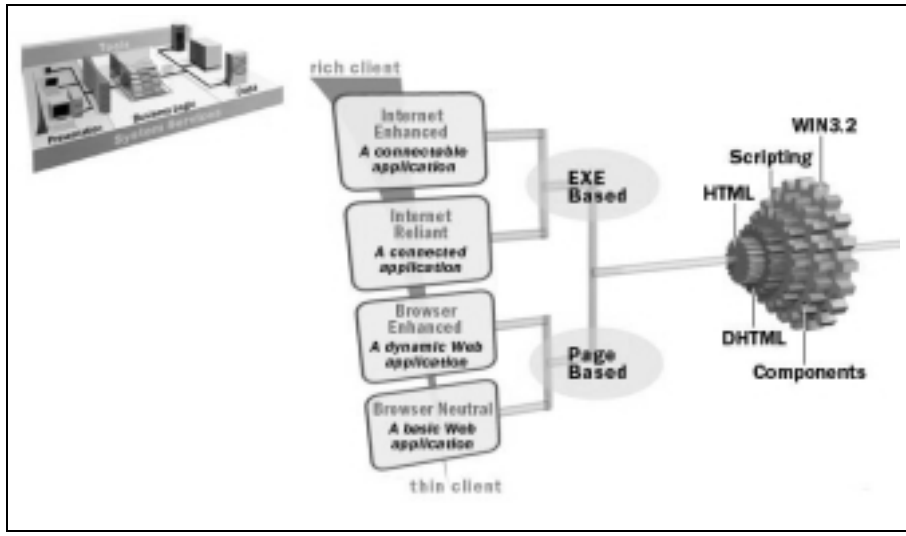

Abbildung 1.3 Die Darstellungsschicht im Detail

Abbildung 1.3 ist eine Abstraktion der Art und Weise, mit der die Darstellungsschicht erstellt wird. Auf der rechten Seite, dargestellt durch die Zahnräder, befinden sich die verschiedenen Technologien. Unter Verwendung dieser Technologien können zwei Arten von Clients erstellt werden, EXE-basierte und webseitenbasierte Clients. Diese Clienttypen können in vier weitere Typen unterteilt werden, die vom Thin Client bis zum Rich Client reichen.

Webseitenbasierte Lösungen

Der Thin Client ist eine Form des webseitenbasierten Clients. Dies bedeutet, dass der Clientcomputer an das Netzwerk angeschlossen ist und nach Bedarf Inhalte herunterlädt. Die Clientanwendung dient lediglich der Anzeige und dem Abruf von Informationen.

Beim Herunterladen spezieller Inhalte werden diese als statische Informationen angezeigt. Sie können anschließend in einigen Textfeldern Eingaben vornehmen oder ein paar Kontrollkästchen aktivieren und die Informationen zur Verarbeitung weiterleiten. Dieser Typ Funktionalität ähnelt der Stapelverarbeitung in Mainframesystemen. Der einzige Unterschied besteht darin, dass sich das Terminal an beliebiger Stelle im Internet befinden kann. Typischerweise handelt es sich bei diesem Clienttyp um einen Webbrowser, und diese Art Anwendung wird als browserneutrale Anwendung bezeichnet. Ein statischer Webbrowseransatz stellt sicher, dass Ihre Anwendung die größtmögliche Unterstützung aufweist.

Die Funktionalität auf Clientseite wird durch die Leistungsmerkmale der Laufzeitumgebung auf dem Clientcomputer bestimmt. Es gibt einfache Webbrowser, und es gibt leistungsstärkere Webbrowser. Der Lösungsansatz ist jedoch derselbe, denn in beiden Fällen wird die Anwendung heruntergeladen. In Abbildung 1.3 wird der erste Rich Client gezeigt, der browsererweiterte Client. Der Unterschied zwischen diesem Client und dem browserneutralen Client besteht darin, dass ein Teil der Verarbeitung auf dem Clientcomputer stattfindet.

Das Verwenden eines browsererweiterten Clients macht die Anwendung interaktiv, der Nachteil hierbei ist jedoch, dass weniger Webbrowser den Inhalt decodieren können. Es ist beispielsweise möglich, Inhalte zu entwickeln, die nur mit dem Internet Explorer oder ausschließlich mit dem Netscape Navigator verwendet werden können. Übliche Anwendungen sind webbrowserbasierte Intranet- oder Extranetanwendungen. In diesen Fällen ist der verwendete Browser häufig bekannt.

EXE-basierte Lösungen

Der zweite Lösungstyp erfordert mehr Funktionalität auf der Clientseite; es ist die Implementierung einer Fat Client-Anwendung erforderlich. Diese Clients basie-

ren auf ausführbaren Anwendungen, die auf dem Clientcomputer installiert und ausgeführt werden, daher können diese als Rich Clients bezeichnet werden.

Die erste Form des Rich Client ist eine internetbasierte Anwendung, die aus Gründen der Funktionalität das zugrunde liegende Betriebssystem und aus Gründen des Inhalts das Netzwerk miteinander kombiniert. Ein Beispiel für diesen Anwendungstyp ist eine Data Warehouse-Anwendung. Beim Sichten eines Data Warehouses erzeugen Sie üblicherweise komplexe Pivottabellen und benötigen einen Server zur Durchführung einiger Aufgaben, da das Verschieben aller Daten auf den Client nicht möglich ist. Zur Anzeige der Ergebnisse in einem Diagramm benötigen Sie jedoch einen Client, der ein grafisches Teilsystem umfasst. Daher benötigt die Anwendung ein dediziertes Netzwerk und umfangreiche Anzeigefunktionen.

Der vierte Typ Rich Client ist eine interneterweiterte Anwendung. Dieser Clienttyp ähnelt dem vorherigen, stellt jedoch immer dann eine Netzwerkverbindung her, wenn Informationen gemeinsam genutzt werden müssen. Nehmen Sie beispielsweise ein Textverarbeitungsprogramm. Die meisten Funktionen, die Sie benötigen, können auf der Clientseite ausgeführt werden. Das Netzwerk spielt sekundär ebenfalls eine Rolle, da über dieses Dokumente mit anderen Textverarbeitungsverwendern genutzt werden können.

Dieser Clienttyp bringt auch die neue Hybridclientanwendung hervor, bei der bei Bedarf das Internet genutzt wird, dies jedoch unsichtbar für den Benutzer. Ein Beispiel für diese Art Anwendung ist Microsoft Money. Diese Anwendung wird auf dem Clientcomputer installiert und hauptsächlich isoliert ausgeführt. Wenn Sie jedoch Ihre Kontoauszüge anzeigen möchten, müssen Sie online gehen. Hierbei werden Sie über einen eingebetteten Browser direkt mit Ihrer Bank verbunden. Anschließend werden Ihre Kontoauszüge abgerufen und angezeigt. Während dieses gesamten Vorgangs muss der Benutzer weder eine URL (Uniform Resource Locator) noch zusätzliche Netzwerkbefehle zur Verbindungsherstellung zum Internet eingeben. Mit anderen Worten, der Hybridclient stellt eine vollständig integrierte Form der Internetnutzung dar.

Die Technologien

Zum Verständnis der einzelnen Clienttypen müssen die verwendeten Kerntechnologien umrissen werden:

- **HTML:** Abkürzung für Hypertext Markup Language, der Websprache für Benutzerschnittstellen. Der Begriff HTML bezieht sich hier üblicherweise auf Version 3.2 oder Version 4.0 der HTML-Spezifikationen. Bei beiden Versionen handelt es sich um plain-vanilla HTML, durch das die formulargestützte Verar-

beitung unterstützt wird. Von allen Technologie verfügt diese über die größte Bandbreite und wird in der Regel für den Thin Client verwendet.

- **DHTML:** Dynamic HTML ist die erweiterte Form der Benutzerschnittstelle für Webbrowser. DHTML unterscheidet sich von HTML in der Unterstützung eines vollständigen Objektmodells. Jeder Bestandteil der Benutzerschnittstelle kann dynamisch adressiert, entfernt oder hinzugefügt werden. Auf diese Weise wird die Schnittstelle zu einer äußerst leistungsstarken Komponente. Diese Funktion wird jedoch nur von wenigen Browsern unterstützt.
- **Scripting:** Innerhalb des DHTML-Objektmodells wird eine Scriptingschnittstelle definiert. Das Scripting wird als separate Technologie betrachtet, da mit dieser Komponente jede Art von Client gesteuert werden kann. Beispielsweise kann mit Hilfe von Skripts Microsoft Excel zur Durchführung bestimmter Aufgaben angewiesen werden. Der Hauptvorteil der Skriptverwendung ist der, dass eine Anwendung angepasst werden kann, ohne dass hierzu umfangreiche Programmierkenntnisse erforderlich sind.
- **Komponenten:** Komponenten sind kompilierte Codeabschnitte, über die Funktionalität bereitgestellt wird. DHTML und Scriptingdienste stellen beispielsweise Komponenten dar. Unter Verwendung von COM können derartige Umgebungen miteinander kommunizieren.
- **Win32:** Das Win32-API stellt eine Möglichkeit zur Interaktion mit dem Windows-Betriebssystem dar. Diese Technologie eignet sich für Clients, die besonderen grafischen oder höheren Leistungsanforderungen gerecht werden müssen. Die Win32-API ist Windows-spezifisch. Es steht für HTML oder DHTML kein vergleichbares Programmiermodell zur Verfügung.

1.3.3 Firewall

Bei der Firewall handelt es sich zwar nicht um eine Schicht, dennoch muss sie erwähnt werden. Die Firewall befindet sich zwischen der Darstellungsschicht und der Schicht der Anwendungslogik. Eine Firewall bezeichnet eine Sicherheitsbarriere, mit der die Daten innerhalb der Firewall geschützt werden.

In einer allgemeinen Architektur ist eine Firewall nur ein Dienst innerhalb des gesamten Frameworks, dies ist bei Windows DNA jedoch nicht der Fall. Windows DNA unterscheidet sich von der Mehrzahl der Betriebssysteme dadurch, dass die Sicherheit in die Schicht der Anwendungslogik verlagert wird.

Dies bringt verschiedene Vorteile mit sich. Zunächst wird eine konsistente Sicherheitsprüfung etabliert. Stellen Sie sich Ihr privates Netzwerk als eine Burg vor. Bei den derzeit verwendeten Sicherheitsmechanismen könnte ein Reiter anonym in Ihre Burg gelangen. Anschließend, abhängig von den angeforderten Diensten, fin-

det eine Sicherheitsprüfung statt. Das Problem hierbei ist, dass sich der anonyme Reiter auf seinem Weg durch die Burg wiederholt einer ID-Prüfung unterziehen muss. Außerdem könnte ein Reiter mit bösen Absichten Schaden anrichten, wenn ein Dienst nicht ausreichend geschützt wurde. Darüber hinaus ist es nicht einfach, eine konsistente Sicherheitsrichtlinie aufrechtzuerhalten, wenn jeder Dienst eine eigene Sicherheitsprüfung durchführt. Sobald der Reiter aus einer Bar in der Burg hinausgeworfen wird, sollten auch alle weiteren Bars für diesen Reiter geschlossen bleiben. Ohne eine zentrale Sicherheitsautorität ist dies nicht zu erreichen.

Durch das Errichten einer Firewall einer zentralen Sicherheitsautorität ist jeder Zugriff auf jeden Dienst bekannt. Den Nutzern werden spezifische Rechte und Privilegien erteilt. Bei Missbrauch dieser Rechte können diese auf einfache und konsistente Weise wieder entzogen werden. Letztendlich führt dies zu einem schnelleren und einfacher strukturierten Sicherheitssystem.

1.3.4 Schicht der Anwendungslogik

Die Schicht der Anwendungslogik stellt das Herz Ihrer Anwendung dar. Hier werden die Daten verarbeitet und bearbeitet. Hierfür sind Geschäftsprozessregeln und Workflowprozeduren verantwortlich. Diese Schicht erfordert die meiste Programmierarbeit. In dieser Schicht werden die Verarbeitung großer Datenmengen, Transaktionsunterstützung, Bereitstellungen großen Umfangs, Messaging und ggf. die Webschnittstelle definiert. Daher ist die Schicht der Anwendungslogik äußerst komplex.

Wenn Sie Windows 2000 als Server für die Anwendungslogik einsetzen, stehen drei Hauptdienste zur Verfügung, COM+, MSMQ (Microsoft Message Queue) und IIS (Internet-Informationsdienste). Siehe hierzu Abbildung 1.4. Es stehen darüber hinaus weitere Dienste zur Verfügung, beispielsweise Microsoft Exchange und Microsoft SQL Server (Structured Query Language), aber diese stellen keinen Bestandteil des standardmäßigen Windows 2000-Pakets dar. Diese Komponenten müssen separat erworben werden.

Jeder Dienst führt eine spezielle Aufgabe aus. COM+ ist für die COM-Grundlagen und die Transaktionsfunktionalität verantwortlich. IIS sorgt für die Internetdienste wie SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol) und HTTP (Hypertext Transfer Protocol). MSMQ übernimmt das Messaging. Jeder dieser Dienste kann andere Dienste zur Erstellung neuer Dienste verwenden. In die Warteschlange gestellte Komponenten beispielsweise verwenden sowohl COM+ als auch MSMQ zur Erstellung neuer Dienste.

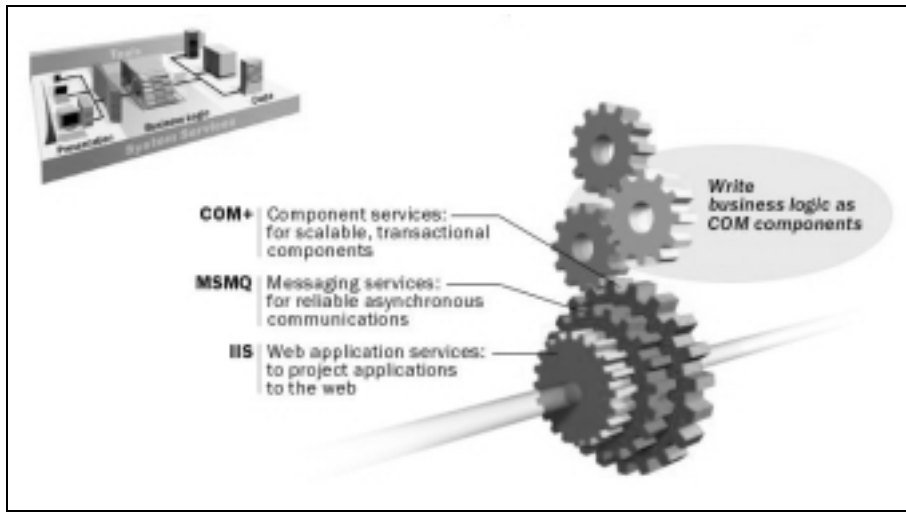

Abbildung 1.4 Aufbau der Schicht der Anwendungslogik

COM+

Die nächste Generation von COM ist COM+. Durch das Hinzufügen von Diensten werden die Einsatzmöglichkeiten von COM im Unternehmen erweitert. COM bietet nur das Framework für die Komponenten, COM+ fügt darüber hinaus Transaktionsfunktionen hinzu.

COM+ und Transaktionsdienste wurden erstmalig in SQL Server 6.5 vorgestellt, bei dem ein Dienst namens DTC (Distributed Transaction Coordinator) eingesetzt wurde. Dieser Dienst sorgte für die Verwaltung von SQL Server-Transaktionen. Anschließend wurde DTC mit einer einfacheren Programmierungs- und Bereitstellungsumgebung namens MTS (Microsoft Transaction Server) neu herausgebracht. MTS ermöglichte das Schreiben von COM-Objekten, durch die Geschäftsprozesse ausgeführt und Transaktionen gesteuert werden konnten. COM+ übernimmt diesen Transaktionsaspekt und integriert ihn in die COM-Laufzeit sowie das Betriebssystem. MTS wurde in das Betriebssystem eingebettet, aber ein MTS-Objekt konnte MTS umgehen. Mit COM+ wird nicht nur dieses Hintertürchen geschlossen, COM+ ist auch konsistenter und eleganter. Aus der Sicht des Programmierers können mit COM+ alle serverseitigen COM-Objekte die Vorteile der Transaktionsverarbeitung nutzen und dazu beitragen, Anwendungen zuverlässiger zu machen.

Die Schicht der Anwendungslogik enthält mit Abstand die größte Anzahl an COM-Objekten, über die spezifische Dienste bereitgestellt werden. Diese Flexibilität ist zur Entwicklung qualitativ hochwertiger Unternehmensanwendungen er-

forderlich, über die Daten von der Datenschicht empfangen, verarbeitet und zurückgesendet werden können. Verschiedene Erweiterungen basieren auf COM und COM+, beispielsweise die TIP-Integration (Transaction Internet Protocol), die erweiterte Sicherheit und der dynamische Lastenausgleich. Diese Erweiterungen werden in den folgenden Abschnitten beschrieben.

TIP-Integration COM+ verwendet OLE-Transaktionen (Objekt Linking and Embedding) zur Verwaltung der Umgebung für die Transaktionsverarbeitung. Dies gilt jedoch nur für die Windows 2000-Plattform. TIP ermöglicht COM-Objekten die Teilnahme an anderen TP-Umgebungen (Transaction-Processing). Diese Transaktionen können durch das TP-System verwaltet werden. Das TP-System unterscheidet sich von der vorherigen Version in MTS. In der damaligen Version mussten Transaktionen durch MTS und DTC verwaltet werden.

Erweiterte Sicherheit Im vorangegangenen Abschnitt zur Firewall wurde erläutert, dass Windows 2000 das Sicherheitsmodell erweitert. Ein Teil dieser Erweiterung erfolgt durch die COM+-Umgebung. In COM+ wird die Sicherheit entweder durch ein rollenbasiertes Sicherheitssystem oder durch Berechtigungen für den Prozesszugriff erreicht. Die rollenbasierte Sicherheit ist ein Verfahren, bei dem einer Rolle ein Satz Berechtigungen zugewiesen wird, die zur Durchführung einer bestimmten Aufgabe benötigt werden. Diese Rolle wird anschließend den Benutzern zugeordnet, die diese Aufgabe ausführen müssen. Diese Rollen können beispielsweise Manager, Angestellte, Bestellungsverarbeitung usw. sein. Das Verwenden von Rollen vereinfacht das domänenunabhängige Zuweisen verschiedener Sicherheitsprivilegien für Benutzer. Ein Angestellter der Buchhaltung kann beispielsweise über viele Zugriffsberechtigungen zu den Buchhaltungsprozessen verfügen, jedoch nur wenige Zugriffsrechte auf die Produktionsprozesse besitzen. Eine Erweiterung der rollenbasierten Sicherheit in COM+ stellt die Fähigkeit dar, Sicherheitsrichtlinien auf der Methodenebene von COM-Objekten anzuwenden.

Zentrale Verwaltung In der vorangegangenen Version von Windows NT wurden COM-Objekte unter Verwendung des MTS Explorers verwaltet. Zur Verwaltung der DCOM (Distributed Component Object Model)-Einstellungen wurde das Dienstprogramm DCOMCNFG eingesetzt. Unter Windows 2000 werden beide Tools durch den Komponentendienst ersetzt. Durch diesen wird die Verwaltung der COM+-Objekte kombiniert und erweitert. Zu den administrativen Aufgaben gehören Bereitstellung, Verwaltung und Überwachung der COM+-Anwendung.

In Warteschlange gestellte Komponenten DCOM ermöglicht den Aufruf von COM-Objekten auf unterschiedlichen Computern. DCOM ist ein synchrones Protokoll, d.h. wenn ein Aufruf durch einen Computer erfolgt, muss der Empfänger verfügbar sein. Steht der Empfänger nicht zur Verfügung, schlägt der Aufruf fehl, und jede zugehörige Operation schlägt ebenfalls fehl. Wurde diese Opera-

tion innerhalb des Kontextes einer Transaktion ausgeführt, schlägt auch die Transaktion fehl. Sie müssen demnach ein Verfahren zum erneuten Versuch nach einem Fehlschlag implementieren.

Unter Windows 2000 wird die einfachere Lösung der Verwendung in Warteschlange gestellter Komponenten angeboten. In Warteschlange befindliche Komponenten ermöglichen das asynchrone Aufrufen anderer Komponenten auf weiteren Computern. Die in Warteschlange befindlichen Komponenten verwenden als zugrunde liegende Messagingarchitektur MSMQ. Wenn daher die Verbindung oder der Aufruf fehlschlagen, kann so lange ein erneuter Versuch unternommen werden, bis die Operation erfolgreich ist.

Ereignisdienste Unter Windows NT 4.x war es nicht einfach, Ereignisse mit Komponenten zu verbinden. Es war eine Technologie vorhanden, die so genannten COM-Verbindungspunkte, aber diese wiesen viele Einschränkungen auf. COM+-Ereignisse dagegen ermöglichen die lockere Anordnung von Ereignissen in Paaren, wobei der Ereignisempfänger aktiv oder inaktiv sein kann. Bei Verwendung von COM+-Ereignissen können Unicast- oder Multicastaufrufe durchgeführt werden. Unicast bedeutet hierbei 1–1, Multicast 1–**n**. Der Empfänger des Ereignisses stellt mit Hilfe eines Abonnementverfahrens eine Verbindung zum Sender her.

Das Veröffentlichen und Abonnieren ermöglicht dem Verleger, den COM+-Ereignisdienst darüber zu informieren, dass er Informationen verteilen möchte. Ein Empfänger, der diese Informationen erhalten möchte, teilt dies dem COM+-Ereignisdienst durch ein Abonnement mit. Sobald auf dem Verleger Informationen geändert werden, werden diese Änderungen über den COM+-Ereignisdienst an die einzelnen Abonnenten weitergegeben.

Dynamischer Lastenausgleich Mit Windows 2000 ist es möglich geworden, einen dynamischen Lastenausgleich für COM+-Objekte bereitzustellen. Beim dynamischen Lastenausgleich wird die Prozessorlast gleichmäßig über mehrere Computer verteilt. Der Lastenausgleich stellt einen serverorientierten Prozess dar und erfolgt gegenüber dem Client, der den Serveraufruf initiiert, transparent.

Internetdienste

Über IIS werden verschiedene Internetdienste bereitgestellt.

- **HTTP:** Das Protokoll des Web. Mit HTTP werden Webseiten und jeglicher Inhalt abgerufen, der über den Webbrowser angefordert wird.
- **FTP:** Das Protokoll zur Dateiübertragung im Internet. Es ist speziell auf die Dateiübertragung ausgelegt und daher optimal für diese Aufgabe geeignet.

- **SMTP:** Das Protokoll, das zur Übertragung von E-Mail-Nachrichten von einem Computer zu einem anderen eingesetzt wird. Es kann zusammen mit POP verwendet werden.
- **Gopher:** Dieses Protokoll wurde in früheren Versionen von IIS verwendet, dann aber durch HTTP ersetzt.

Im Vergleich zu anderen Internetservern ist ein besonderes Merkmal von IIS die Transaktionsintegration. Bei Durchführung einer Anforderung aus IIS handelt es sich um eine Transaktionsanforderung. Durch die Verwendung von Transaktionen als Grundlage für die Anforderungsverarbeitung können alle Vorteile von COM+ genutzt werden. Gleichzeitig ist IIS so beim dynamischen Lastenausgleich oder bei der Optimierung sehr flexibel.

Zur Erstellung von IIS-Inhalten kann ISAPI (Internet Server API) oder – häufiger verwendet – ASP (Active Server Pages) herangezogen werden. ASP ist eine Skriptumgebung, bei der der Programmcode über Skripts ausgeführt wird. Die Skriptsprache ist neutral und voll dynamisch. Zusammen mit dem COM+-Objekt eingesetzt, können voll funktionsfähige Unternehmensanwendungen erstellt werden. Mit Hilfe von ASP kann DHTML oder XML (Extensible Markup Language) generiert werden. Über Microsoft Visual InterDev können ASP-, DHTML- und XML-Anwendungen erstellt werden (diese Funktion steht unter Windows 2000 nicht zur Verfügung).

Messagingdienste

Das asynchrone Messaging wird von MSMQ gehandhabt. Das Messaging und das Schreiben von Code unterscheiden sich sehr stark voneinander. Beim Messaging wird angenommen, dass die Verbindung zwischen Client und Server nicht vorhanden ist, die Operation aber dennoch ausgeführt wird. Das Eventualitätsmodell ermöglicht das Schreiben von Anwendungen, die auch in unzuverlässigen Netzwerken oder unter unzuverlässigen Bedingungen funktionieren.

MSMQ ist eine ereignisgesteuerte Architektur. Die Daten werden auf den Server gelegt, anschließend muss der Server auf das aufgerufene Ereignis reagieren. Das ereignisgesteuerte Modell ist nicht neu, denn Windows selbst ist ebenfalls ereignisgesteuert. Dennoch bedeutet dies, dass die Anwendungsprogrammierung von der allgemeinen Norm abweichen muss, da eine sofortige Antwort nicht zu erwarten ist.

Interoperabilität mit vorhandenen Systemen, Anwendungen und Daten

Windows 2000 internetfähig zu machen war eine Sache. Eine mindestens ebenso wichtige Aufgabe ist jedoch, eine Integration mit älteren Systemen zu ermöglichen, den so genannten Legacysystemen. Legacysysteme funktionieren in der Regel und reichen zur Ausführung der ihnen zugedachten Aufgabe vollständig aus. Daher stellt das Ersetzen eines solchen Systems keine Option dar. Stattdessen müssen derartige Systeme in die Gesamtarchitektur integriert werden.

Windows 2000 bietet verschiedene Optionen zur Integration unterschiedlicher Legacytechnologien:

- **MSMQ-Integration:** Die Messagingintegration für MSMQ wird für verschiedene Produkte angeboten. Zur Verbindung der Windows-Plattform mit MQSeries kann ein SNA Server eingesetzt werden. Die Verbindung von anderen Plattformen (UNIX, MVS, OS/2 usw.) zu MSMQ kann über FalconMQ-Clients ab Level8 erfolgen.
- **COM-Transaktionsintegration:** Mit Hilfe dieser Schnittstelle können CICS- und IMS-Transaktionen (Messaging- und Transaktionsprodukte von IBM) zu COM-Objekten erweitert werden. Mit der COM-Transaktionsintegration stehen eine Reihe von Entwicklungstools und eine Anwendungslogik zur Verfügung, mit der die IBM-Mainframetransaktion gekapselt werden kann. Die Kommunikation mit einer IBM-Transaktion wird durch Windows SNA Server ermöglicht.
- **UDA (Universal Data Access):** Diese Schicht wird für den Datenbankzugriff verwendet. Durch Verwendung systemeigener Treiber kann eine Verbindung zu Ressourcen hergestellt werden, die sich auf einer anderen Plattform befinden.
- **DCOM auf UNIX und Mainframe:** Eine Technologie, mit der COM-Objekte nicht nur auf einer Windows-Plattform geschrieben werden können. Die Software AG brachte den ersten Anschluss für Solaris heraus. Dieser Anschluss wurde zum Einschluss der Mainframeplattform erweitert und wird derzeit auf eine Gesamtarchitektur names EntireX iNTegrator ausgeweitet.
- **TIP/XA-Transaktionsintegration:** Es sind verschiedene Standards für Transaktionsprotokolle vorhanden. TIP wurde bereits beschrieben, ein weiterer Standard ist das XA-Protokoll. Die Integration dieses Protokolls wird durch einen Ressourcenverteiler/-manager erzielt.

1.3.5 Datenschicht

Die dritte Schicht ist die Datenschicht. Hier werden die Ressourcen verwaltet. Im Gegensatz zur Schicht der Anwendungslogik werden auf dieser Schicht keine Daten verarbeitet. Stattdessen wird auf dieser Schicht die Verwaltung der riesigen Datenmengen definiert. Datenbanken und Ressourcen wachsen in Ihrer Größe täglich, und die Aufgabe, diese Daten zu verwalten, gestaltet sich immer schwieriger.

Einige der Daten werden in Tabellen gespeichert, andere in Textverarbeitungsdokumenten, der Großteil befindet sich in relationalen Datenbanken. Es gibt zwei Methoden zur Speicherung dieser Vielfalt und Menge an Daten. Bei der ersten Methode erstellen Sie einen Speicher (Repository), in dem alle diese Datentypen gespeichert werden können. In der Praxis werden für diese Form der Datenspeicherung so genannte Blobs verwendet (Binary Large Objects).

Anhand von Blobs werden Daten jedoch nur als binäre Abbilder gespeichert. Die Richtigkeit der relationalen Informationen wird jedoch nicht gewährleistet. Wenn beispielsweise ein gespeicherter Blob auf ein anderes Dokument verweist, wird ein ungültiger Verweis nicht erfasst. In der Datenschicht haben diese ungültigen Verweise keine Bedeutung, in der Schicht der Anwendungslogik jedoch schon. Durch derartige Verweise könnten eine oder mehrere Transaktionen abgebrochen werden.

Sie können natürlich dafür sorgen, dass in der Schicht der Anwendungslogik richtige Blobs geschrieben werden. Dies erfordert jedoch zusätzliche Programmierschritte, die Teil der Datenschicht sein sollten, denn die Verweisverwaltung ist eine Datenoperation. Zur Bereitstellung der Verweisverwaltung müssen Sie dem Datenrepository ein externes Modul hinzufügen. Dies stellt jedoch nicht die optimale Lösung dar.

Besser ist es, wenn alle Ressourcen ihre eigenen Datenformate verwalten. Jede Ressource muss hierbei einem üblichen Objektmodell entsprechen. Auf diese Weise können Daten formatunabhängig bearbeitet, kopiert und referenziert werden. Genau diese Lösung wird mit der UDA-Strategie (Universal Data Access) geboten.

Das UDA-Objektmodell wird auch als OLE DB-Objektmodell bezeichnet (siehe Abbildung 1.5). Es handelt sich um ein Lowlevel-Format, das eine Spezifikation dazu enthält, wie Transaktionen und Rowsets definiert werden. Das OLE DB-Objekt ist ein schnelles Lowlevel-Objekt. Dies bedeutet, dass das Schreiben von OLE DB-Objekten etwas Arbeit erfordert. Im Gegensatz zu Windows 2000 bietet Visual C++ eine kleine, überlagerte Schicht mit Vorlagen, die so genannten

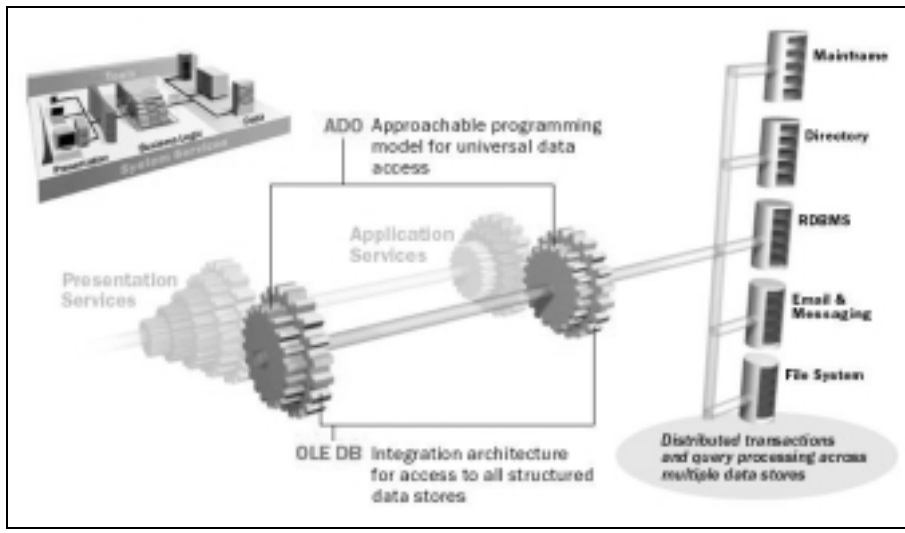

Abbildung 1.5 Universeller Datenzugriff (UDA, Universal Data Access)

OLE DB-Consumer Templates. Mit dieser Schicht wird OLE DB vereinfacht, ohne es langsamer oder weniger effizient zu machen.

Das OLE DB-Objektmodell hat den Vorteil, dass dieses weder durch eine bestimmte Form noch durch ein Format beschränkt wird. Im relationalen Modus beispielsweise muss das einzig mögliche Resultset eine bestimmte Anzahl Zeilen und Spalten umfassen. Für OLE DB muss eine kleine Anzahl COM-Schnittstellen implementiert werden, alles Weitere liegt beim OLE DB-Provider. Außer der Bedingung, dass es sich um ein COM-Objekt handeln muss, gelten keine weiteren Einschränkungen.

OLE DB ist eine ausgeprägte Lowlevel-Technologie und kann nur in compilerbasierten Entwicklungsumgebungen, z. B. Visual C++ oder Delphi, genutzt werden. Dem Visual Basic-Programmierer oder dem Skriptschreiber wird ein einfacheres Objektmodell namens ADO (Active Data Objects) zur Verfügung gestellt. ADO ist ebenso leistungsstark wie OLE DB, da es den Zugriff auf die meisten OLE DB-Funktionen ermöglicht.

Eine Frage muss noch gestellt werden: Wie legt OLE DB die Funktionalität der relationalen Datenbankjobs offen? Die Antwort ist – mit Hilfe von COM-Objekten. Aus diesem Grund ist es nicht erforderlich, dass mit ADO sämtliche der verschiedenen Datenformate verarbeitet werden können. Es liegt in der Verantwortung des Providers, die als Schnittstelle zwischen Daten und ADO-Aufrufen benötigten COM-Objekte zu generieren.

1.4 Schreiben von Anwendungen mit Hilfe von Visual Studio

Die Strategie von Windows DNA richtet sich nicht nur auf Dienste und Hilfsprogramme in Windows 2000. Sie umfasst auch das Erstellen von Anwendungen mit Hilfe dieser Dienste und Hilfsprogramme. Dies ist das Thema, das in diesem Buch behandelt werden soll.

Zum Entwickeln von Anwendungen müssen verschiedene Tools eingesetzt werden. Microsoft Visual Studio ist ein Paket, das sämtliche der erforderlichen Tools enthält. Es stehen verschiedene Versionen von Microsoft Visual Studio zur Verfügung. Interessant für Sie ist die Visual Studio Enterprise Edition, die auch Visual Studio DNA-Edition genannt wird. Dies rührt daher, dass diese Edition genau die Tools enthält, die zum Entwickeln einer Windows DNA-Anwendung benötigt werden.

Ein Problem von Visual Studio ist, dass es unterschiedliche Programmiersprachen enthält. Dies kann zu Problemen führen, da viele Entwickler eine Programmiersprache favorisieren. Eine Strategie bei der Implementierung von Windows DNA liegt jedoch darin, die beste Sprache für eine bestimmte Aufgabe zu wählen. Auf diese Weise wird die Diskussion um die beste Sprache an sich in die Frage »Wann ist welche Sprache die beste?« gewandelt.

Im vorliegenden Buch habe ich einen willkürlichen Ansatz gewählt und jeweils die Sprache zur Lösung einer Aufgabe gewählt, die mir als die geeignetste erschien. Bevor Sie mich also als verrückt abtun, sollten Sie daran denken, dass Sie ein Windows DNA-Ingenieur sind und daher immer auch den Kontext und die Vorgeschichte einer Entscheidung berücksichtigen sollten.

1.4.1 Der Kontext einer willkürlichen Sprachauswahl

Ein Freund von mir arbeitet als Consultant und entwickelt vertikale Anwendungen, die anschließend den jeweiligen Kundenwünschen angepasst werden. Dies stellt eine klassische Windows DNA-Anwendungsentwicklung dar.

Die von ihm entwickelten Anwendungen weisen drei Merkmale auf:

- Die Anwendungen umfassen einen vollständigen Entwicklungszyklus (Entwurf, Test, Dokumentation usw.)
- Die Anwendungen weisen einen vollständigen Lebenszyklus auf (Bug Fixes, Versionsnummern)
- Jeder Schritt im Produktentwicklungszyklus wird nach Möglichkeit auf die effektivste Weise umgesetzt

Mein Freund musste sich bei der Entwicklung seiner Anwendungen für eine Programmiersprache entscheiden. Für die erste Anwendungsversion verwendete er C++, da einige Legacyanwendungen vorhanden waren. Um jedoch ein optimales Kosten-Nutzen-Verhältnis zu gewährleisten, führte sein Team eine interne Studie durch, die sich auf den gesamten Produktentwicklungszyklus erstreckte und bei der verschiedene Projekte berücksichtigt wurden, in denen unterschiedliche Sprachen verwendet worden waren.

Und zu welchem Schluss führte diese Studie? Sie kam zu dem Ergebnis, dass die verwendete Sprache keine Rolle spielt. Worauf es ankommt, ist die Leistungsfähigkeit Ihres Teams, die Erfahrung der einzelnen Entwickler. Im Einzelnen konnten der Studie jedoch einige interessante Anhaltspunkte in Bezug auf die jeweiligen Sprachen entnommen werden.

Da wäre zunächst die Produktivität des Programmierers. Visual Basic ist eine Umgebung, in der eine Anwendung schnell implementiert werden kann. Visual C++ ist langsamer, da die Entwicklung einiger Bibliotheken und Objekte mehr Zeit erfordert.

Ein weiteres Maß der Produktivität ist der Faktor, wie schnell ein Programmierer eine Sprache erlernen kann. Auch hier ist Visual Basic die beste Wahl, da diese Sprache leicht erlernt werden kann. Visual C++ dagegen ist schwieriger zu erlernen und zu programmieren.

Ein drittes Produktivitätsmaß stellt die Anzahl der im geschriebenen Code erzeugten Bugs dar. In dieser Hinsicht schneidet Visual Basic sehr schlecht ab. Auf jeden Bug in Visual C++ kommen drei Bugs in Visual Basic.

Welche Sprache ist also die produktivste? Anscheinend Visual Basic. Obwohl hier mehr Bugs erzeugt werden, erfolgt das Beheben dieser Bugs schneller, da die Produktivität an sich sehr viel höher liegt. Unglücklicherweise ist diese Antwort falsch. Die richtige Antwort lautet, dass Visual Basic eine genauso schwierige Sprache ist wie Visual C++.

Und hierfür sorgt die Art der Bugs in Visual Basic. Die Bugs in Visual Basic sind logisch, in Visual C++ stellen Bugs ungültige Zeiger oder beschädigten Speicher dar. Logische Bugs sind schlecht für den Produktentwicklungszyklus, da sie Änderungen in der Dokumentation, im Design usw. verlangen. Solche Änderungen erfordern mehr Zeit und erzeugen zusätzliche Kosten. Bugs in Visual C++ dagegen sind einfacher zu beheben, da sie nur Änderungen am Code erfordern, nicht jedoch Änderungen an der Codelogik.

Warum dies so ist? Visual C++ ist eine schwierige Sprache, die einen erheblichen Lernaufwand erfordert. Sie müssen lernen, einen Punkt auf das **i** und einen Strich

durch das **t** zu machen, um es einmal so auszudrücken. Das heißt, ein Visual C++-Anfänger hat mehr Erfahrung als ein Visual Basic-Programmierer. Diese Erfahrung eines Visual C++-Programmierers umfasst beispielsweise die Fähigkeit, Klassen und Systeme richtig zu entwerfen.

Zu welcher Schlussfolgerung gelangte also mein Freund? Besorgen Sie sich einen geeigneten Programmierer! Beim Einsatz erfahrener Visual Basic-Programmierer sank die Anzahl der logischen Bugs dramatisch. Deshalb gibt es nicht **die** richtige Sprache für eine Aufgabe, es gibt nur den richtigen Entwickler für eine Aufgabe. Deshalb werden für die Beispiele in diesem Buch auch verschiedene Sprachen eingesetzt. Natürlich verwende ich keine Stapelverarbeitungsskripts zur Erstellung logischer Komponenten. Aber ich verwende gelegentlich Visual Basic oder Visual J++ oder sogar Visual C++. Hierbei wird jedoch auch immer das richtige Verfahren zum Einsatz einer bestimmten Technologie aufgezeigt. Diese Vorgehensweise stellt einen Bestandteil der Windows DNA-Anwendungsentwicklung mit Hilfe von Windows 2000 dar.

1.4.2 Programmiersprachen in Visual Studio

Die Basisedition von Visual Studio beinhaltet fünf Sprachprodukte für die Entwicklung.

Visual J++

Visual J++ ist ein Entwicklungstool, das auf der Java-Sprache basiert. Java ist eine Sprache, bei der Quellcode temporär in Bytecode kompiliert wird. Der Bytecode wird anschließend durch eine VM (Virtual Machine) interpretiert. Mit Hilfe dieses Ansatzes ist es möglich, Quellen zu kompilieren und anschließend auf einem beliebigen Rechner mit Java-VM auszuführen.

Sun Microsystems entwickelte die ursprüngliche Java-Sprache und definierte ebenfalls die Standardbibliotheken von Java. Visual J++ unterstützt die meisten Sun Java-Bibliotheksaufrufe, ist jedoch auf die Windows-Entwicklung zugeschnitten. Daher ist Visual J++ in gewisser Weise inkompatibel mit den Java-Standardbibliotheken.

Vorteile Java ist einfacher zu handhaben als C++, jedoch komplizierter als Visual Basic. Es handelt sich um eine rein objektorientierte Sprache, wodurch der Entwickler zur Verwendung von Objekten gezwungen ist. Java erkennt die in C++ verwendeten Zeiger nicht, die diese Sprache so komplex machen. Das Schnittstellenkonzept ist in die Sprache integriert, daher stellt die Komponentenentwicklung eine logische Schlussfolgerung dar. Die derzeit in der Visual Studio-Umge-

bung unterstützte Visual J++-Entwicklungsumgebung stellt gleichzeitig die produktivste dar. Hierbei wird die Dokumentation in das Produkt integriert.

Nachteile Der größte Nachteil sind Geschwindigkeit und Flexibilität. Java kann als einfach und doch leistungsstark oder als komplex und einschränkend betrachtet werden. Dies richtet sich ganz nach Ihrer Haltung gegenüber Java. Die aktuelle Implementierung von Visual J++ unterstützt den Bibliothekssatz von Sun Java, hinsichtlich der Produktivität ist es jedoch nicht mit Sun Java vergleichbar. Während plattformübergreifende Fähigkeiten eine der besten Funktionen von Java darstellen, werden genau diese im Visual J++-Kontext nicht besonders gut unterstützt.

Visual C++

Visual C++ ist ein Entwicklungstool, dass als Entwicklungssprache C++ verwendet. Visual C++ wird häufig für die Entwicklung von Systemen und Softwareanwendungen eingesetzt. Diese Sprache besitzt die Fähigkeit, C und COM miteinander zu kombinieren, und dies ohne Einschränkungen.

Vorteile C++ ist bei weitem die populärste Sprache zur Entwicklung von System- oder Softwareanwendungen. Es werden alle Aspekte der objektorientierten Entwicklung unterstützt, und gleichzeitig wird die Fähigkeit zur Integration von C-Legacyquellcode bereitgestellt. C++ ist eine Sprache, mit der beliebige Aufgaben ausgeführt werden können und stellt daher die beste Wahl zur Entwicklung von Bibliotheken, Objekten und Komponenten dar. Angewendet durch einen erfahrenen Entwickler, tendiert der C++-Code dazu, zuverlässiger zu sein, da Debugbibliotheken und -objekte eingebunden werden können. Diese Objekte können dazu eingesetzt werden, insbesondere Inkonsistenzen bei Laufzeit und Kompilierungszeit zu ermitteln.

Nachteile C++ weist zwar sehr viele Vorteile auf, besitzt jedoch auch einige Nachteile. Mit C++ werden dem Entwickler bei Entwicklung und Entwurf zwar viel Freiheit und Flexibilität gewährt, Programmierfehler können jedoch katastrophale Folgen haben. Darüber hinaus ist C++ schwer zu erlernen. Aufgrund der zahlreichen Leistungsmerkmale entwickeln viele C++-Entwickler eigene Entwicklungsstrategien, was die Teamarbeit negativ beeinflussen kann.

Visual Basic

Visual Basic ist eines der populärsten Entwicklungstools auf der Windows-Plattform. Es handelt sich um eine Sprache, die auf einer Microsoft-spezifischen Version von Basic basiert. Visual Basic 6.0 unterstützt objektorientierte Konstruktionen, wie beispielsweise Objekte, jedoch keine Vererbung. Die Komplexität des Betriebssystems wird als eine Reihe von COM-Objekten maskiert.

Vorteile Der größte Vorteil von Visual Basic ist seine Popularität. Es bietet hohe Produktivität, da Komponenten und Anwendungen leicht entwickelt werden können. Sprache und Umgebung sind einfach und leicht zu erlernen.

Nachteile Der größte Nachteil von Visual Basic ist seine Einfachheit. Diese führt zu Anwendungen, die praktisch »im Vorbeigehen entworfen« und geschrieben werden. Obwohl diese Art der Anwendungen sehr wahrscheinlich nicht so viele Entwicklungsfehler aufweist wie Visual C++ (Zeigerfehler), neigt Visual Basic dazu, mehr logische Fehler zu erzeugen. Diese Art Fehler kann aufwendig zu beheben sein, da logische Fehler eine Umgestalung des Logikdesigns erfordern. Da in Visual Basic das Betriebssystem sowie die Dienste als eine Reihe von COM-Objekten abstrahiert werden, ist die Verteilung einer Visual Basic-Anwendung aufgrund der Größe der Endversion einer Anwendung komplizierter.

Visual FoxPro

In den vorangegangenen Erläuterungen der Entwicklungsumgebungen stellte jedes der Tools ein Szenario zur Codierung und Kompilierung dar. Visual FoxPro unterscheidet sich in diesem Punkt, da eine Entwicklungsumgebung in eine Datenbankumgebung integriert wird. Auf diese Weise wird die Entwicklung von Datenbankanwendungen erheblich erleichtert. Visual FoxPro ist objektorientiert und extrem schnell beim Datenbankzugriff.

Vorteile Visual FoxPro-Entwickler sind häufig sehr talentiert und kennen ihr Entwicklungstool besonders gut. Das heißt, eine Visual FoxPro-Anwendung ist im Allgemeinen gut entworfen und entwickelt. Visual FoxPro ist eine so genannte All-In-One-Umgebung mit umfassenden objektorientierten Fähigkeiten.

Nachteile Der Entwicklerpool, obgleich konstant, ist im Vergleich zu den anderen Tools kleiner. FoxPro ist ein klassisches Client/Server-Entwicklungstool. Es stellt jedoch für die Komponentenentwicklung nicht das ideale Tool dar.

Visual InterDev

Das fünfte Entwicklungstool in der Visual Studio-Entwicklungsumgebung ist das DHTML- und ASP-Tool Visual InterDev. Visual InterDev beansprucht für sich nicht, ein All-In-One-Tool zu sein. Visual InterDev ist eine Tool zur Erstellung von Webanwendungen, bei dem auch Komponenten eingesetzt werden. Während DHTML (Dynamic HTML) einen Standard darstellt, ist Microsoft Internet Explorer der einzige Browser, von dem Visual InterDev unterstützt wird.

Vorteile Visual InterDev ist das Entwicklungstool für Dynamic HTML und ASP. Es eignet sich besonders für die Entwicklung vollständig skalierbarer Webanwendungen.

Nachteile Visual InterDev neigt dazu, einige Dinge zu verkomplizieren, die eigentlich einfacher ausgedrückt werden könnten. Während es sich gut dazu eignet, einen klassischen Programmierer an eine Webumgebung heranzuführen, wird es jedoch den Ansprüchen der Entwickler, die mit dem Web aufgewachsen sind, in Bezug auf Leistungsstärke und Vielfältigkeit nicht gerecht. Es kann zur Erstellung von Webseiten für Netscape Navigator oder andere Browser verwendet werden, bei der Erstellung clientseitiger Skripts ist es jedoch eher für den Microsoft Internet Explorer geeignet.

1.4.3 Funktionen der Enterprise Edition

In der Enterprise Edition von Visual Studio 6.0 stehen verschiedene Tools für die Unternehmensentwicklung zur Verfügung. Diese Tools vereinfachen die Entwicklung von Windows DNA-Anwendungen. Es handelt sich zwar nicht um objektorientierte Tools, aber es werden andere Aspekte des Entwicklungsprozesses unterstützt, beispielsweise Anwendungsversionen sowie Leistungsaspekte.

Visual Source Safe

Bei Visual Source Safe (VSS) handelt es sich um ein System zur Versionssteuerung für alle Entwicklungsaufgaben. Es ist in sämtliche Entwicklungstools der Microsoft-Plattform integriert. Mit diesem Tool wird die Arbeit mit Quellcode, Dokumentation und anderen Entwicklungstools im Team ermöglicht.

Application Performance Explorer

Wenn Sie eine Anwendung entwickeln, ist es häufig erforderlich, das beste Kosten-Nutzen-Verhältnis zu ermitteln. Ist es beispielsweise besser, einen Server mit 4 Prozessoren und 1 GB Speicher zu verwenden, oder ist es kosteneffektiver, zwei Systeme mit jeweils 2 Prozessoren und 512 MB RAM zu kaufen? Die Antwort auf diese Frage fällt nicht leicht, da zu einer Beantwortung u.a. Netzwerkverbindung, Festplattentyp, Bus- und Prozessorgeschwindigkeit berücksichtigt werden müssen. Die Anzahl der Hardwareparameter ist enorm. Des Weiteren können die Dienste des Betriebssystems so optimiert werden, dass die jeweiligen Hardwarevorteile besser genutzt werden, sodass noch mehr Parameter zum Tragen kommen. Auf diese Weise wird das Errechnen des tatsächlichen Kosten-Nutzen-Wertes nahezu unmöglich.

Eine Methode zur einfachen Berechnung des Kosten-Nutzen-Verhältnisses stellt der Application Performance Explorer dar. Es handelt sich hierbei um ein Tool, mit dem verschiedene Szenarien verteilter Anwendungen bereitgestellt werden, die zum Testen der Hardwareeffizienz verwendet werden können. Diese Szenarien können dann von einem Computer auf einen anderen verlagert und erneut aus-

geführt werden. Basierend auf den Ergebnissen für die Szenarien können mathematische Berechnungen zum besten Kosten-Nutzen-Wert durchgeführt werden.

Visual Studio Analyzer

Beim Testen verteilter Anwendungen ist es erforderlich, die jeweils zu erwartenden Engpässe zu kennen. Dies ist nicht mit der Profilerstellung vergleichbar, da diese Programme sprachspezifisch sind und einen Quellcodezugriff erfordern. Stattdessen wird ein Test des Laufzeitsystems mit Hilfe von Visual Studio Analyzer durchgeführt, bei dem kein solcher Eingriff erforderlich ist. Es können verschiedene Datenquellen eingerichtet werden, mit denen die Engpässe des gesamten Systems ermittelt werden.

Visual Modeler

Jede Architektur erfordert eine Entwurfsmethode für das gesamte System. Visual Modeler ist ein Tool, bei dem das System mit Hilfe von UML (Unified Modeling Language) entworfen wird. UML ist eine Notation, die zur Definition von Klassen, Methoden und Parametern eingesetzt wird. UML ist keine Sprache und dient ausschließlich zu Entwurfszwecken. Visual Modeler besitzt die Fähigkeit, das Modell aus dem Quellcode zu generieren. Anschließend kann das Modell in der Implementierungsphase des Produktzyklus verwendet werden.

Component Manager

Bei der Entwicklung vieler Systeme mit Komponenten ist es vorteilhaft, die Komponenten wieder zu verwenden. Diese Komponenten können Quellcode, Entwurfskonzepte oder binäre Komponenten sein. Der Component Manager unterstützt Sie bei diesem Prozess. Mit diesem Tool können Sie Informationen veröffentlichen, die durch eine andere Komponente wieder verwendet wird.

Der Component Manager verwendet das Repositoryframework von Microsoft. Mit dem Microsoft Repository können Sie, basierend auf Funktionalität und Funktionssätzen, Abfragen nach spezifischen Komponenten durchführen. Das Repository kann als Datenbank betrachtet werden, tatsächlich ist es jedoch mehr als das, da es auf die Anforderungen des Entwicklungszyklus zugeschnitten ist.

1.5 Resümee

Windows DNA stellt eine Form der Anwendungsentwicklung dar, die verschiedenste Funktionen unterstützt, beispielsweise Transaktionen, das Messaging und Datendienste. Windows DNA und Windows 2000 greifen hierbei ineinander. Windows DNA basiert auf den Diensten, die durch Windows 2000 bereitgestellt werden. Aus der Sicht von Microsoft ist Windows DNA ein Marketingtool, das

eine Möglichkeit bietet, Anwendungen zu entwickeln. Ich bin der Meinung, dass mit Windows DNA eine Vielzahl von Methoden und Verfahren für die Anwendungsentwicklung zur Verfügung gestellt wird.

Windows DNA kann als Modell zur Förderung effizienter Programmiertechniken verstanden werden, und hierbei stellt die Definition des Problemkontextes das erste effektive Verfahren dar.

Es wurde zunächst kurz der Kontext von Windows umrissen, um einen Eindruck der Windows-Plattformhintergründe zu vermitteln.

Das Anwenden des gleichen Verfahrens auf die eigene Problemsituation hilft Ihnen dabei, die Hintergründe des Problems zu erkennen.

Basierend auf diesem Wissen können Sie einen Lösungsansatz erarbeiten, der am ehesten zum gewünschten Ziel führt.

2 Alles über Muster

In diesem Kapitel soll die Anwendungsentwicklung eingehender betrachtet werden. Einen Ansatz zur Anwendungsentwicklung stellt die Verwendung von Mustern dar. In diesem Buch werden viele solcher Muster eingesetzt. Sie werden im Laufe Ihrer Karriere als Softwareentwickler feststellen, dass viele Probleme wiederholt auftreten. In diesen Situationen ist es sinnvoll, eine Lösung bzw. ein Muster anzuwenden, dessen Effektivität bereits erprobt wurde.

2.1 Einführung in Muster

Was ist ein Muster? Es stellt eine Lösung zu einem Problem in einem bestimmten Kontext dar. Zusätzlich zeichnet sich ein Muster dadurch aus, dass es wiederholt eingesetzt werden kann. Eine Lösung kann nur dann tatsächlich als Muster bezeichnet werden, wenn dieses in mindestens drei ähnlichen Situationen zur Problemlösung eingesetzt wurde. Aus dieser Definition ergibt sich gleichzeitig, dass sich ein Muster nicht ausschließlich auf den Bereich der Computeranwendungsentwicklung beschränkt. Der Architekt Christopher Alexander definierte das Muster (pattern) folgendermaßen: »Jedes Muster ist eine aus drei Elementen bestehende Regel, mit der eine Relation zwischen einem bestimmten Kontext, einem Problem und einer Lösung ausgedrückt wird.«

Das Muster verknüpft ein Problem und eine Lösung mit einem Kontext. Mit dieser Aussage wird an die Kontextdefinition von Kapitel 1 angeknüpft. Ein klar definierter Kontext stellt die Basis für ein gutes Muster dar. Wenn der Kontext nicht eindeutig klar ist, kann das Problemverständnis vielleicht unvollständig sein und dazu führen, dass die bereitgestellte Lösung dem Problem nicht angemessen ist.

Angenommen, es liegt eine Problemsituation vor, bei der der Kontext definiert und eine Lösung gefunden wurde. Bedeutet dies automatisch, dass die Lösung ausgereift ist? Unglücklicherweise nicht, da ein Muster ausschließlich zur Lösung eines spezifischen Problems eingesetzt werden kann, und eine Lösung wirkt sich immer auch auf die weiteren Bestandteile des Systems aus. Als Ergebnis erhält man einen neuen Kontext mit neuen Problemen. Es gibt nie eine absolut richtige Antwort, durch die das System nicht in irgendeiner Weise beeinflusst wird. Sie können immer einen neuen Kontext anwenden, der wieder andere Muster und Lösungen erfordert.

Die Auswirkungen der Verwendung eines Musters sind genauso wichtig wie der Kontext, in dem das Muster verwendet wird. Wenn Problem, Kontext und Muster

klar definiert sind, können Sie Ihre Muster wie kleine Puzzleteilchen zusammenfügen und so die beste Lösung für die Anwendung erarbeiten.

2.2 Ein Beispiel für ein einfaches Muster

Das folgende Beispiel hat nichts mit Computern zu tun, sagt aber trotzdem eine Menge über das Konzept von Mustern aus.

Das Problem besteht darin, ein Haus zu errichten. Der Kontext lautet, dass das Haus in den Wäldern von Kalifornien erbaut werden soll. Daraus ergibt sich die einzukalkulierende Gefahr eines Waldbrandes, d.h., für das Haus ist ein optimaler Brandschutz erforderlich. Dennoch ist die zu bewältigende Aufgabe nicht neu, unter diesen Voraussetzungen wurden bereits viele Häuser erbaut. Sie können sich beispielsweise erkundigen, wie bei anderen Häusern vorgegangen wurde, die in einem Wald erbaut wurden, und was zu deren Schutz vor einem Waldbrand getan wurde. Sie würden schnell feststellen, dass die Erbauung dieser Häuser stets einem bestimmten Muster folgt; z.B. werden im näheren Umkreis des Hauses die vorhandenen Bäume gefällt, und es wird nach Möglichkeit mit nicht entflammbaren Materialien gebaut. Wenn Sie also bei der Errichtung des Hauses diesem Muster folgen, minimieren Sie die Gefahr, Ihr Haus durch einen Brand zu verlieren.

Um ein Muster zu erkennen und in der richtigen Situation anzuwenden, müssen Sie Kontext, Problem und Lösung verstehen. Im vorangegangenen Beispiel besteht der Problemkontext in der Lage des Hauses. Als Nächstes wird das Problem definiert: Die Region, in der Sie das Haus errichten möchten, ist der Gefahr eines Waldbrandes ausgesetzt. Dies bedeutet, dass Ihr Haus abbrennen könnte.

Die Lösung für das Problem besteht darin, die Bäume um das Haus zu entfernen und beim Hausbau nicht entflammbare Materialien zu verwenden. Auf diese Weise erhöhen Sie die Chance, bei einem Waldbrand ungeschoren davonzukommen.

2.2.1 Wo Muster versagen

Muster stellen nicht immer den perfekten Lösungsansatz dar. Das häufigste Problem besteht darin, dass der Kontext nicht voll erfasst wird und bei der Verwendung von Mustern falsche Tatsachen vorausgesetzt werden. Nehmen wir wieder das Beispiel des Hauses, das in einem waldigen, trockenen Gebiet erbaut werden soll. Die Lösung bestand darin, die Bäume im näheren Umkreis des Hauses zu fällen und nicht entflammbares Material einzusetzen. Dabei geht man von der Annahme aus, dass für den Hausbau ein beliebiges Material eingesetzt werden kann, und dass es möglich ist, die Bäume um das Haus zu fällen.

Nehmen Sie aber einmal an, dass Sie Ihr Haus in einer Region mit strengen Bauvorschriften im Hinblick auf das Aussehen des Hauses erbauen möchten, dass sein Aussehen in Einklang mit der Natur stehen muss. Darüber hinaus stehen einige der Bäume im Umkreis Ihres Hauses unter Naturschutz und dürfen nicht gefällt werden. Das gefundene Muster greift hier also nicht. Im obigen Muster wurde davon ausgegangen, dass Sie auf Ihrem Grundstück und bezüglich des Aussehens Ihres Hauses freie Hand haben. Dies ist eine falsche Voraussetzung, das vorgeschlagene Muster würde zu einem Fehlschlagen des Projekts führen.

Ist deswegen das Muster falsch? Nein – das Muster ist richtig, aber nur in den Fällen, in denen die umliegenden Bäume nicht unter Naturschutz stehen und ein beliebiges Material für den Hausbau eingesetzt werden kann. Das Muster und der Kontext müssen lediglich klar definiert und aufeinander abgestimmt werden. Dies verdeutlicht, dass der Kontext sorgfältig und umsichtig definiert werden muss.

2.3 Grundlegendes zu Mustern

Der Hauptgrund für die Verwendung von Mustern stellt die Möglichkeit zur Entwicklung einer gemeinsamen Sprache dar. Mit einer solchen ist die Kommunikation mit anderen Systementwicklern ohne die Erläuterung des Kontextes möglich. Ein Muster ermöglicht Ihnen das Konvertieren eines Satzes aus Kontext/Problem/Lösung in Spezifikationen, die leicht verständlich sind.

2.3.1 Definieren eines guten Musters

Ein gutes Muster führt zur Lösung eines Problems, ist nicht linear und beruht auf Erfahrungswerten. Es werden drei Anforderungen erfüllt:

- **Das Muster führt zur Problemlösung:** Über das Muster werden Lösung und Problem definiert. Die Definition muss hierbei eindeutig sein, da viele Systeme sich nach dieser Kombination aus Lösung/Problem richten. Darüber hinaus müssen Problem und Lösung wiederholt anwendbar sein, da es sich ansonsten nicht um ein Muster handeln würde.
- **Die Lösung ist nicht linear:** Muster sind niemals ganz logisch oder rational. Wenn das Problem logisch wäre, wäre auch die Lösung einfach und jeder könnte zu der gleichen Antwort gelangen. Die Mathematik ist beispielsweise logisch. Wenn zwei Zahlen addiert werden, kann es nur ein richtiges Ergebnis geben. Das Addieren von Zahlen führt immer zum gleichen Ergebnis, egal, ob Sie im Wald stehen oder nicht. Es gibt keine Kontextabhängigkeit, die Definition eines Musters ist daher nicht erforderlich.
- **Erfahrungswerte dienen als Grundlage:** Da Muster nicht linear und zum Teil irrational sind, hängen Sie stark vom jeweiligen Erfahrungsschatz desjenigen

ab, der die Lösung bereitstellt. Ein Muster kann gut oder schlecht sein. Daher sollten Muster auf verschiedenen Systemen basieren. Einige Referenzen erfordern mindestens drei Systeme, hierbei gilt: je mehr, desto besser.

Mit einem guten Muster sammeln Sie Erfahrung, ohne zu experimentieren. Im Fachjargon wird dies als »intelligent guess« bezeichnet. »Guess«, da es keine richtige oder falsche Antwort gibt, »intelligent«, weil es sich trotzdem nicht um einen willkürlichen Versuch handelt – Sie setzen eine erprobte Methode zur Lösung des Problems ein.

2.3.2 Arten von Mustern

Der Prozess der Produktentwicklung umfasst verschiedene Schritte und unterschiedliche Arten von Mustern für die einzelnen Schritte. Es sind drei Arten von Softwaremustern vorhanden:

- **Konzeptuelle Muster:** In der anfänglichen Produktentwicklung werden diese Muster zur Beschreibung des Systems als Gruppe von Einheiten verwendet. Wenn Sie beispielsweise über die anfängliche Architektur nachdenken, tauchen bestimmte Probleme auf. Diese Probleme sind sehr abstrakt und beschäftigen sich mit dem Anwendungskonzept selbst. Es werden keine technischen Aspekte berücksichtigt, es werden ausschließlich Konzeptfragen geklärt.
- **Entwurfsmuster:** Bei der Implementierung des Anwendungsdomänenmodells stehen verschiedene allgemeine Softwaremodelle zur Verfügung. Ein Modell kann beispielsweise den besten Zugriff auf eine Datenbank definieren und dynamische Objekte generieren. Die auf dieser Ebene definierten Muster sind häufig abstrakte Konzepte, die über die Software in Form von Schnittstellen, Objekten usw. angewendet werden. Entwurfsmuster enthalten in der Regel keine Implementierungsdetails.
- **Programmierungsmuster:** Bei der Implementierung der Anwendungsschnittstellen und -entwürfe gibt es eine spezielle Methode zum Schreiben einiger der Codesegmente. Diese Muster sind sehr sprach-, plattform- und technologiespezifisch. Mit diesen Mustern wird versucht, unter Verwendung der verfügbaren Technologie ein optimales Programmsegment zu schreiben, beispielsweise einen Smartpointer (intelligenter Zeiger) in C++.

Die drei verschiedenen Muster dienen unterschiedlichen Zwecken. Es besteht eine klare Staffelung von der konzeptionellen Ebene bis hin zur Implementierung. Die Muster bleiben in den unterschiedlichen Phasen jedoch abstrakt und umreißen ein Grundprinzip. Daher stellen Muster kein Patentrezept zur Implementierung eines Algorithmus oder eines UML-Diagramms (Unified Modeling Language) dar.

Wann ist ein Muster kein Muster?

Die Welt besteht nicht nur aus Mustern. Es gibt fundamentale Richtlinien oder Regeln, die nicht kontextabhängig sind, beispielsweise die Regeln der Mathematik, und diese stellen keine Muster dar. Erweist sich ein Konzept in allen Fällen als richtig, handelt es sich nicht um ein Muster.

Muster können nicht geplant werden. Es ist nicht möglich, ein Projekt mit der Absicht anzugehen, ein Muster zur Lösung des Problems zu entwerfen. Muster entstehen im Verlaufe eines Prozesses aus Fehlern und Erfolgen. Durch Fehler lernen Sie die Dynamik eines Vorgangs kennen und können sich so ein Muster erarbeiten. Daher ist es nicht möglich, ein Muster im Vorfeld zu planen.

Bei der Erarbeitung eines Musters ist konstruktive Kritik gefragt. Durch konstruktive Kritik, beispielsweise durch Vergleiche, kann die Anwendungsfähigkeit eines Musters eingeschätzt werden.

2.3.3 Vorlage zum Beschreiben von Mustern

Das Anwenden von Mustern erfordert gleichzeitig das Verständnis von Mustern. Leider ist die Lektüre der Musterdefinitionen wenig spannend und gleicht dem Durcharbeiten eines Standardlehrbuches. Dennoch können Sie sich Informationen zu verschiedenen Mustern aneignen und anschließend Probleme mit Hilfe der erlernten Muster lösen. Oder Sie versuchen, Problem und Kontext zu definieren und nach einem geeigneten Muster zu suchen. Ich neige dazu, Probleme nach dem zweiten Prinzip anzugehen. Der Nachteil hierbei ist, dass Sie zunächst eine Vielzahl von Büchern nach einem geeigneten Muster durchsuchen, bevor Sie sich ein umfassendes Wissen über häufig verwendeten Muster angeeignet haben.

Bei der Suche nach einem Muster müssen Sie in der Lage sein, dieses zu kategorisieren und mit dem eigenen Problem und Kontext zu vergleichen. Dies kann schwierig sein, da unterschiedliche Musterbücher unterschiedliche Notationen verwenden. Viele dieser Notationen verwenden jedoch eine gemeinsame Technologie. Einige dieser Begriffe werden nachstehend beschrieben.

Muster – allgemeine Begriffe

Ein Muster wird durch seinen **Namen** identifiziert. Der Name kann aus einem kurzen Satz oder einem einzigen Wort bestehen. In beiden Fällen sollte der Name eindeutig und intuitiv sein. Es ist hilfreich, wenn der Name eines Musters dessen Funktion widerspiegelt.

In vielen Musterdefinitionen bezeichnet das nächste Element das **Problem**, auch **Intention** genannt. Hier werden die Ziele für das Muster definiert. Diese Beschreibung umfasst häufig einen kurzen Absatz.

Der nächste Schritt besteht in der Definition des Zwecks für das Entwurfsmuster. Diese Komponente wird auch **Motivation** oder **Beweggrund** genannt und beschreibt, aus welchem Grund das Muster erstellt wird. In vielen Fällen wird hier ein Szenario definiert. Falls möglich, sollte bei der Definition des Musters auch das philosophische Argument dargelegt werden, damit der Leser die Gedankengänge des Entwicklers während der Dokumentation des Musters nachvollziehen kann. Ein Verständnis der Denkweise des Entwicklers ermöglicht dem Leser, seine eigene Situation mit der durch das Muster beschriebenen Situation in Beziehung zu bringen. Die unterschiedlichen Beweggründe können Richtigkeit, Ressourcen, Struktur, Aufbau oder Verwendung sein.

Die **Anwendbarkeit** eines Musters definiert dessen Grenzen. Durch das Betrachten von Intention und Anwendbarkeit kann leicht bestimmt werden, ob sich das Muster für eine Situation eignet. Es wird eine Art Prüfliste definiert, die verwendet werden sollte, wenn die Intention sich als übereinstimmend erweist.

Der letzte Punkt definiert die **Auswirkungen** oder **Konsequenzen** des Musters. Nehmen Sie an, ein Entwickler implementiert ein Muster und muss dann feststellen, dass dieses mit einem anderen Muster inkompatibel ist. Der Entwickler wird nicht gerade erfreut sein, denn das entworfene Muster sollte ja eigentlich eine Erleichtung darstellen. Durch das Beschreiben der Auswirkungen eines Musters ist es dem Entwickler möglich, eventuell bei der Implementierung auftretende Konflikte mit anderen Zielen einzuschätzen. Ruft die Implementierung Konflikte mit anderen Mustern hervor, ist das Muster nicht anwendbar.

Ein weiterer Aspekt bei den Auswirkungen eines Musters ist der, die positiven Eigenschaften des Musters hervorzuheben. Angenommen, ein Entwickler muss die Verwendung seines Musters rechtfertigen. Die positiven Auswirkungen des Musters können den Entwickler bei der Entscheidung unterstützen, das Muster im Entwurf zu verwenden oder nicht. Auch wenn es sich nicht empfiehlt, können hier Schlagworte wie Skalierbarkeit, Stabilität, Erweiterbarkeit usw. eingesetzt werden.

Abschließend benötigt jedes Muster eine **Implementierung**. Die Implementierung ist einfach, da über sie die Verwendung des Musters definiert wird. Hier können Code oder UML-Diagramme eingefügt werden, je nach beschriebenem Mustertyp.

Lesen der Muster in diesem Buch

Nachdem nun die Begrifflichkeit erläutert wurde, sind Sie in der Lage, ein Muster zu lesen. Aber wie wenden Sie ein Muster auf ein Problem an? Hierzu vergleichen Sie Ihren Kontext mit den Beweggründen für das Muster. Nachdem Sie ein Mus-

ter zur Verfügung haben, sollten Sie sicherstellen, dass in anderen Teilen der Anwendung keine Probleme hervorgerufen werden.

Dieses Buch wendet Muster an, wenn es das Szenario erfordert. Bei der Beschreibung der Probleme, die durch COM (Component Object Model) gelöst werden, wird beispielsweise das Bridge-Muster verwendet. Innerhalb eines Kapitels, in dem COM beschrieben wird, wird demnach das Bridge-Muster referenziert und erläutert. In Anhang A finden Sie einfache Definitionen der Muster; diese werden mit den folgenden Begriffen beschrieben:

- **Name:** Hierbei handelt es sich um den Namen des Entwurfsmusters.
- **Intention:** Entspricht der zuvor definierten allgemeinen Zielsetzung.
- **Start- und Endkontext:** Das Verwenden von Begriffen wie Anwendbarkeit, Intention usw. kompliziert die Musterdefinition. Stattdessen kann ein Start- und ein Endkontext definiert werden. Dies ist einfacher, da nur zwei Elemente definiert werden müssen. Der Startkontext definiert das zu bewältigende Problem. Der Endkontext definiert die zu erreichenden Ziele oder Anforderungen an das System mit einfachen Worten. Sie können diesen Vorgang damit vergleichen, dass Sie den Startkontext in eine Blackbox geben und als Ergebnis den Endkontext erhalten.
- **Lösung:** Eine Implementierung der Blackbox, bei der als Eingabe der Startkontext und als Ergebnis der Endkontext verwendet wird.
- **Auswirkungen:** Alle Lösungen besitzen Auswirkungen, die kompliziert oder einfach sein können. Diese müssen definiert werden, da durch sie der Endkontext etabliert wird.

Diese Musterdefinition ist sehr einfach, eignet sich jedoch auch für eine Komponentenarchitektur. Komponenten sind wie Computerchips – es sind einige Drähte vorhanden, die verbunden und aktiviert werden. Eine Komponentenmethodik und eine Musterdefinition, um die dieser Ansatz zur Softwareentwicklung ergänzt wird, machen die Dinge verständlicher. Komponenten werden in den nächsten Kapiteln eingehender erläutert.

2.3.4 Muster und deren Beziehung zu anderen Konzepten

Neben den Mustern gibt es Frameworks, Handbücher, empfohlene Vorgehensweisen usw. Worin unterscheidet sich ein Muster von diesen Dingen? Zunächst ist ein Muster ein generisches Konzept. Mit einem Muster kann eine Implementierung definiert werden, aber in erster Linie ist es ein Konzept. Frameworks und Handbücher definieren Lösungen für spezifische Probleme. Diese Konzepte weisen keinen Kontext auf. Stattdessen werden Probleme auf allgemeine Weise gelöst. Beispiele hierfür ist das MFC-Framework (Microsoft Foundation Library, ver-

wendet in C++), das zur Erstellung einer objektorientierten Benutzerschnittstelle verwendet wird. MFC ist kein Entwurfsmuster.

Empfohlene Vorgehensweisen und Muster ähneln sich. Bei empfohlenen Vorgehensweisen handelt es sich um nicht validierte Muster – das Warum und der Kontext einer empfohlenen Vorgehensweise wurden nicht definiert oder ausreichend überprüft. Muster nehmen diese empfohlenen Vorgehensweisen auf und strukturieren diese. Eine empfohlene Vorgehensweise kann beispielsweise lauten: »Fasse nicht mit der Hand in kochendes Wasser.« Diese Empfehlung klingt sinnvoll. Tatsächlich ist sie aber nicht immer sinnvoll. Was geschieht, wenn die angesprochene Person Schutzhandschuhe trägt? Mit einer empfohlenen Vorgehensweise wird nicht der genaue Kontext definiert, ein Muster dagegen definiert einen Kontext und eine Lösung.

2.4 Antimuster

Muster werden zum Erreichen positiver Ziele entworfen. Einige Projekte schlagen jedoch fehl, da Fehler in der Programmierungslogik oder bei den Programmiermethoden gemacht werden. Auch in diesen Fällen kann das Konzept der Muster angewendet werden, denn Fehler wiederholen sich häufig. Dennoch kann diese Art der Analyse nicht als Muster bezeichnet werden und wird daher Antimuster genannt.

2.4.1 Warum treten Antimuster auf?

Ein Antimuster ist ein Muster, das sich wiederholt und negative Auswirkungen hervorruft. Eine offensichtliche Frage im Hinblick auf Antimuster ist, warum diese nicht von vornherein vermieden werden. Dies wäre die logische Konsequenz. Die Antwort ist, dass Menschen Fehler machen und bei der Anwendungsentwicklung nicht immer logische Entscheidungen treffen. Denken Sie an den Kontext – aufgrund der Geschäftspolitik, der Unternehmensprozesse, technischer Unerfahrenheit oder unzureichender technologischer Ressourcen usw. müssen Kompromisse gefunden werden.

Bei der Anwendungsentwicklung für ein Unternehmen sehen wir uns häufig einer Art isolierter Umgebung gegenüber. Entscheidungen zur Anwendungsentwicklung werden durch die Unternehmensrichtlinien diktiert, was nicht immer zu einem optimalen Entwurfskonzept führt. Der Entwurf wird außerdem durch persönliche Vorlieben der Entwickler, die Unternehmenspolitik und durch die persönlichen Ziele der beteiligten Personen beeinflusst. Nicht zu vergessen die engen Deadlines, die um jeden Preis eingehalten werden müssen. In diesem Kontext ist es nicht überraschend, das in einer Anwendung Antimuster auftauchen.

Ein weiterer Grund für das Auftreten von Antimustern sind Unternehmen, die an einem traditionellen Geschäftsprozess festhalten. Dies ist an sich eine gute Sache, da sich ein Unternehmen so von anderen abhebt. Mit dieser Einstellung gewinnt das Unternehmen einen Vorteil im allgemeinen Konkurrenzkampf. Da ein solcher Geschäftsprozess sich jedoch nicht über Nacht entwickelt, treten häufig Probleme auf. Es müssen Kompromisse eingegangen werden, die häufig unlogisch sind. Das Softwaresystem wiederum muss diesen unlogischen Entwurf widerspiegeln, was dann zu schlechten Softwaresystemen führt.

Das Gegenstück zu einem schlechten Geschäftsprozess ist schlechte Software. Ein Unternehmen verfügt häufig über einen Geschäftsprozess, dem die Software nicht gewachsen ist, daher muss sich das Unternehmen an die Software anpassen. Ein Beispiel hierfür ist das sehr umfangreiche ERP-Programm (Enterprise Resource Planner), bei dem genau dieser Umstand auftritt. Die Unternehmen fügen sich diesem Prozess der Softwareintegration, da dies als vorteilhaft für das Unternehmen angesehen wird. Dies ist jedoch nicht immer richtig.

Eine weitere Ursache für Antimuster ist die Unerfahrenheit von Softwareentwicklern. Das Entwickeln eines konkurrenzfähigen Produkts erfordert umfangreiche Forschungs- und Entwicklungsarbeit, die dann zur Entwicklung eines neuen Produkts führen sollte. Es treten Fehler auf, und mit der Zeit entwickeln sich Muster negativer Auswirkungen.

Daher treten selbst bei größter Anstrengung immer Fehler und daher auch immer Antimuster auf. Ihre Aufgabe besteht darin, die Zahl der Antimuster in einem Projekt zu minimieren und, falls möglich, vorhandene Antimuster zu beseitigen. Es wird immer Projekte geben, die nicht so funktionieren, wie sie sollten. Dies schmerzt, aber aus einer negativen Erfahrung kann man auch immer etwas lernen.

2.5 Resümee

Die Definition eines Musters bleibt weiterhin etwas nebulös, da nicht jeder in gleicher Weise über ein Muster denkt. Dennoch glaube ich an das Konzept der Muster. Muster stellen eine praktische Methode zum Verknüpfen von Kontext und Problem sowie der anschließenden Definition einer Lösung dar. Versuchen Sie nicht, Muster als eine Art Patentrezept für die Problemlösung zu betrachten. Stattdessen sollten Sie Muster als eine Möglichkeit zum Angehen eines Problems begreifen.

Der Vorteil bei der Verwendung von Mustern liegt darin, dass Sie nicht den Prozess durchlaufen müssen, den andere bereits getestet und erfolgreich erprobt haben. Stattdessen führen Sie den Entwicklungsprozess weiter fort, indem Sie einige Grundkonzepte anwenden und eigene Verfahren hinzufügen. Durch das Doku-

mentieren dieser eigenen Entwurfsverfahren unter Verwendung eines Musterentwicklungszyklus können Sie den Entwicklungsprozess erweitern. Dies führt letztendlich zu einer stabileren und verständlicheren Software, selbst wenn diese sehr komplex ist.

3 Entwickeln von Anwendungen

*In diesem Kapitel wird der Prozess der Anwendungsentwicklung definiert. Zunächst soll der Anwendungszyklus definiert werden, anschließend wird eine praktische Entwicklungsstrategie erläutert, die als **iterative, kontextuelle Entwicklung** bezeichnet wird. Als Nächstes werden die einzelnen Schritte der Anwendungsdefinition unter Verwendung des Anwendungsentwicklungszyklus untersucht.*

3.1 Entwickeln von Anwendungen

Die Anwendungsentwicklung stellte nie eine einfache Aufgabe dar. Die Definition eines Modells zur Anwendungsentwicklung ähnelt manchmal dem Klinkenputzen – es wird viel versprochen, aber nicht alles gehalten. Meine Erfahrung hat mich gelehrt, dass jeder ein Modell definieren würde, wenn diese Aufgabe tatsächlich so einfach wäre. Es gibt jedoch keine einfache Antwort, da der Prozess der Anwendungsentwicklung sehr komplex ist. Es müssen viele Abstriche gemacht, Kompromisse eingegangen und Anpassungen vorgenommen werden.

Bedeutet dies, dass alle Modelle zur Anwendungsentwicklung nutzlos sind? Nein. Es ist möglich, verschiedene generische Prozesse zu definieren und diese anschließend logisch zu optimieren. Bei der Entwicklung einer Anwendung beispielsweise wird immer ein bestimmter Zyklus durchlaufen, der so genannte Anwendungsentwicklungszyklus (Application Development Cycle, ADC). Es handelt sich um einen **Zyklus**, da Sie sich am Ende des Prozesses wieder dort befinden, wo Sie begonnen haben.

3.1.1 Der Anwendungsentwicklungszyklus

Der Anwendungsentwicklungszyklus folgt dem nachstehenden allgemeinen Rhythmus:

- **Definieren des Prozesses:** Das erste Konzept der Anwendung wird erarbeitet. Die Kundenwünsche werden zusammengetragen. In dieser Phase kommen Kunde und Anwendungsentwickler zusammen und treffen eine Übereinkunft hinsichtlich des idealen Geschäftsprozesses.
- **Modellentwurf:** Die technischen Details von Anwendungskonzept und Geschäftsprozess werden definiert, beispielsweise die Einzelheiten des Komponentenmodells, die Bearbeitungsweise der Transaktionen sowie die zu verwendende Datenbank. Darüber hinaus werden in dieser Phase Begrifflichkeiten und Konzepte definiert. Diese Terminologie wird im gesamten Zyklus zur Refe-

renzierung spezifischer Kontexte eingesetzt. Zu diesem Zeitpunkt beginnt auch die Dokumentation.

- **Entwickeln eines Prototyps:** Basierend auf dem Entwurfsmodell wird der erste Prototyp entwickelt. Durch dieses Modell werden die Kernfunktionen implementiert und bestimmte Annahmen getestet.
- **Implementieren des Geschäftsprozesses:** Basierend auf Entwurfsmodell und Prototyp wird eine detaillierte Implementierung des Geschäftsprozesses vorgenommen.
- **Testen der Implementierung:** Die Implementierung muss getestet werden, damit eine richtige und konsistente Implementierung des Geschäftsprozesses gewährleistet ist.
- **Abschließende Dokumentation von Code und Anwendung:** Nach dem Testlauf müssen Quellcode und Anwendung dokumentiert werden. Die Dokumentation zielt darauf ab, das Vorgehen und die Beweggründe der Anwendungsentwicklung zu erläutern.
- **Herausgeben der Anwendung:** Abschließend wird das System an den Kunden ausgeliefert. Der Kunde kann dann das System zur Vereinfachung seines Geschäftsbetriebes einsetzen. Zu diesem Zeitpunkt beginnt der gesamte Prozess von vorn, um die nächste Version der Anwendung zu entwickeln.
- **Beheben von Bugs:** Da kein System fehlerfrei ist, ist das Beheben von Bugs sowie das erneute Verteilen der verbesserten Anwendung erforderlich.

Der Anwendungsentwicklungszyklus definiert die allgemeine Reihenfolge der verschiedenen Schritte, es wird jedoch nicht definiert, wie diese Schritte umgesetzt werden. Müssen die Schritte beispielsweise seriell erfolgen, oder ist eine parallele Ausführung möglich? Das Wasserfallmodell stellt eine Form dar, diesen Zyklus näher zu verdeutlichen.

3.1.2 Das Wasserfallmodell

Die klassische Implementierung des Anwendungsentwicklungszyklus (ADC) ist das Wasserfallmodell, bei dem die vorzunehmenden Schritte nacheinander durchgeführt werden. Jeder Schritt muss abgeschlossen sein, bevor der nächste Schritt in Angriff genommen wird. Abbildung 3.1 zeigt dieses Modell.

In diesem Schaubild muss jeder der Behälter gefüllt werden, bevor der nächste Behälter gefüllt werden kann, genau wie im Entwicklungszyklus. Jeder Schritt muss abgeschlossen sein, bevor der nächste beginnen kann. Diese Form der Entwicklung erleichtert die Steuerung des Projekts, da jeder Schritt als Prüfpunkt dient, bei dem Faktoren wie Kosten, Zeit u.a. berücksichtigt werden können.

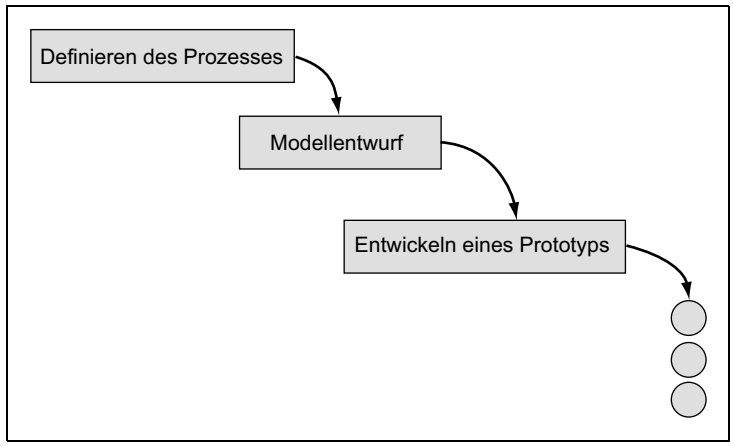

Abbildung 3.1 Wasserfallmodell für die Anwendungsentwicklung

Da jedoch keine optimale Ressourcennutzung gegeben ist, stellt diese Methode nicht das beste Modell dar. Ein typisches Entwicklerteam besteht aus Experten, die sich auf bestimmten Gebieten spezialisiert haben. Wenn der Programmierer beispielsweise auf das Entwurfsmodell warten muss, führt dies zu einer Leerlaufzeit für diesen Programmierer. Ein weiteres Problem beim Wasserfallmodell ist die klare Trennung zwischen Designer, Entwickler und Programmierer. Ein Programmierer oder Entwickler verfügt häufig über einen Einblick, den der Designer nicht besitzt, was bei unzureichender Kommunikation zu Mehrkosten durch Entwurfsänderungen führen kann.

Stellen Sie sich beispielsweise vor, dass eine bereits getroffene Entscheidung rückgängig gemacht werden muss. Abbildung 3.2 veranschaulicht die hierbei entstehenden Kosten.

Das obige Schaubild zeigt, dass sich bei der Zurücknahme einer Entscheidung die Kosten mit der Anzahl der neu durchzuführenden Schritte erhöhen. Nehmen Sie an, dass ein Fehler bei der Herausgabe der Anwendung dazu führt, dass der Zyklus in der Entwurfsphase neu begonnen werden muss. Dies bedeutet, dass fünf Schritte wiederholt werden müssen, und die Kosten für dieses Vorgehen sind enorm, da alle beteiligten Personen erneut in die Entwurfsphase eintreten müssen.

Das Wasserfallmodell ist bei der Entwicklung umfangreicher Anwendungen nicht sehr effektiv und wird daher in den meisten Fällen nicht verwendet.

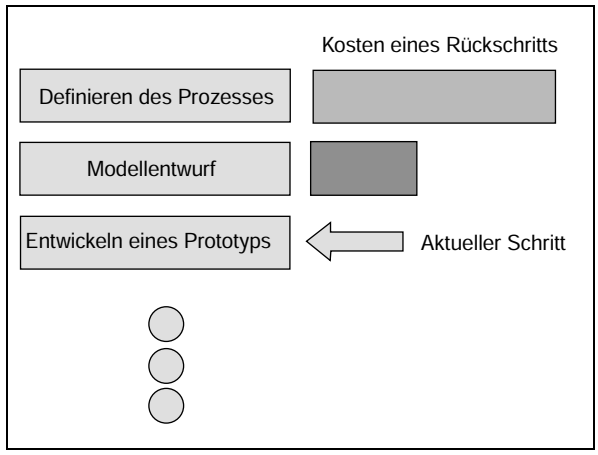

Abbildung 3.2 Mehrkosten durch das Zurücknehmen einer zuvor getroffenen Entscheidung

3.1.3 Gefahren bei der Implementierung einer Anwendung

Ein Merkmal der Computerindustrie ist ihre Unbeständigkeit. Vor weniger als zehn Jahren wurden DOS-Anwendungen noch als Nonplusultra betrachtet. Heute würde niemand auch nur daran denken, zur Entwicklung einer Schnittstelle DOS-Interrupts zu verwenden. Die Industrie unterliegt einem ständigen Wandel. Die einzige konstante Größe ist die, dass es keine konstante Größe gibt. Viele Entwickler versuchen jedoch, diese permanenten Veränderungen zu ignorieren und langfristig einsetzbare Anwendungen zu entwickeln.

Frameworks

Ein klassischer Ansatz bei der Anwendungsentwicklung ist die Erstellung einer isolierten Schicht in der Anwendung. Wenn eine Änderung auftritt, kann diese Isolationsschicht an die aufgetretenen Änderungen angepasst werden. Diese Isolationsschicht wird häufig als Framework bezeichnet. Ein Framework wird als der Versuch definiert, einen Satz funktioneller Dienstprogramme zu entwickeln, die der Erleichterung spezieller Probleme dienen. Bei diesen Dienstprogrammen kann es sich um Tools, Bibliotheken oder um eine Kombination aus beidem handeln.

Im Extremfall können Frameworks so allumfassend sein, dass der Eindruck entsteht, dass mit diesen die Entwicklung der Isolationsschicht erfolgt. So wird die gesamte Lösung in einen vollständigen Zyklus überführt. Die Lösung wird zu einem Problem, das anschließend zu einer Lösung wird. Dies ergibt nicht immer Sinn.

Der Schlüssel bei Entwurf und Entwicklung eines sinnvollen Frameworks besteht in dem Verständnis, dass über das Framework verschiedene Problemen gelöst werden müssen. Bei einer geschäftlichen Konferenz besteht beispielsweise das Problem darin, eine oder mehrere Konferenzen und deren Teilnehmer zu koordinieren, und genau dieses Problem muss mit einem Framework gelöst werden.

Herstellerabhängigkeit

Bei der Anwendungsentwicklung müssen verschiedene Technologien eingesetzt werden. Der Faktor des Herstellers kommt zum Tragen, wenn bei der Anwendungsentwicklung Technologien verwendet werden, die nur über einen Hersteller bezogen werden können. Je größer das Projekt, desto schwerer wiegt dieser Faktor und wird zu einem Problem, wenn der Geschäftsprozess vom Wohlwollen des Produktherstellers abhängt.

Dennoch wird ein gewisser Grad der Herstellerabhängigkeit häufig in Kauf genommen, um ein Projekt abschließen zu können. Diese Situation tritt natürlich nicht nur bei der Entwicklung von Computerlösungen auf. Wenn Sie beispielsweise einen Mercedes fahren, lassen Sie diesen nicht in einer BMW-Werkstatt reparieren. Der BMW-Händler verfügt nicht über die Ausstattung zur Reparatur eines Mercedes. Daher können Sie nur hoffen, dass sich Ihre Investition gelohnt hat und dass der Mercedes-Händler weiterhin den nötigen Service bereitstellt. Falls dies nicht der Fall ist, können Sie nur auf eine andere (bessere) Lösung warten. Der Punkt ist, dass Sie einem Hersteller nicht blind vertrauen sollten, sondern dass Sie Ihre Hausaufgaben machen und herausfinden, welche Lösung zum gegebenen Zeitpunkt und in naher Zukunft die beste Option darstellt.

3.1.4 Die Lösung – Der iterative, kontextuelle Entwurf

Derzeit stellt die beste Methode zur Implementierung des Anwendungsentwicklungszyklus das iterative, kontextuelle Entwurfsmuster dar. Dieses Muster wird nicht im Anhang erläutert, sondern nachstehend beschrieben.

Das iterative, kontextuelle Entwurfsmuster umfasst zwei Hauptkonzepte:

- **Iterativ:** Das schrittweise Lösen eines Problems. Gute Systeme werden unter Verwendung evolutionärer Verfahren entwickelt.
- **Kontextuell:** Zielt darauf ab, den Kundenwünschen besondere Aufmerksamkeit zu schenken und hierzu die Kommunikation zwischen allen Beteiligten Parteien zu verbessern.

Iterativer Entwurf

Ein iterativer Ansatz ist der Prozess, eine Anwendung auf evolutionäre Weise zu entwickeln. Der Vorteil eines schrittweisen Vorgehens ist der, dass nicht alle Probleme gleichzeitig gelöst bzw. alle Ziele gleichzeitig erreicht werden müssen. Beim iterativen Entwurf können neue Systeme aus alten Systemen entwickelt werden, indem das alte System aufgerüstet wird.

Der iterative Entwurf sieht vor, Lösungen immer wieder auf die aktuellen Probleme zu untersuchen. Während der Lösungsimplementierung werden bestimmte Feststellungen gemacht – einige Dinge sind optimal, andere nicht. Diese Inkonsistenzen werden notiert und in zukünftigen Softwareversionen berücksichtigt.

Die beste Methode, den iterativen Entwurf zu verstehen, besteht darin, sich die zu entwickelnde Anwendung als eine Reihe von Inseln vorzustellen, die miteinander kommunizieren. Siehe hierzu Abbildung 3.3.

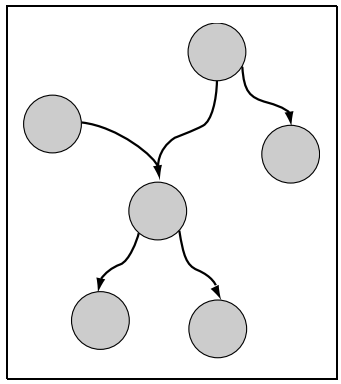

Abbildung 3.3 Technologieinseln

Jede Insel repräsentiert einen Teil Ihrer Anwendung. Eine Insel stellt eine Gruppe von Funktionen dar, die von den anderen Inseln isoliert ist. Einige Insel sind rein technologisch (beispielsweise ein Datenbank- oder Dateiserver), einige Inseln basieren auf einer gewissen Logik (beispielsweise ein Verwaltungssystem oder Produktionsstatistiken). Einige dieser Inseln können eine ähnliche Logik besitzen, jedoch eine andere Technologie verwenden, und umgekehrt.

Zur Übertragung von Daten von einer Insel zu einer anderen wird ein Protokoll eingesetzt. Das Protokoll stellt das gemeinsame Kommunikationsmittel zwischen zwei Inseln dar. Dieser Vorgang lässt sich mit zwei Personen vergleichen, die sich auf die Verständigung in einer bestimmten Sprache geeinigt haben. Beispiele für die verwendeten Protokolle sind HTTP (Hypertext Transfer Protocol), DCOM

(Distributed Component Object Model) und Kommunikationsprotokolle für den Datenbankzugriff.

Der iterative Ansatz Beim iterativen Ansatz werden die Inseln Schritt für Schritt erstellt. Inseln können hinzugefügt, entfernt oder aktualisiert und in Form von »Inselketten« angeordnet werden, durch die beispielsweise ein Verwaltungssystem entsteht. Anschließend können mehrere Inselketten zusammen eine Anwendung bilden.

Das Entwicklen der Anwendung ähnelt dem Erstellen einer eigenen Welt mit Hilfe von Inseln. Beim iterativen Ansatz beginnen Sie mit einigen wenigen Inseln und erstellen nach Bedarf weitere Inseln.

Zur Strukturierung der Anwendung können Sie einen Plan erstellen, in dem Sie festhalten, welche Funktionen die einzelnen Inseln und Inselketten übernehmen sollen. Die zentralen Inseln sollten die Kernfunktionalität bereitstellen.

Lassen Sie uns dieses Entwurfsmuster auf ein praktisches Beispiel anwenden. Angenommen, Sie benötigen ein Konferenzanmeldungssystem, mit dem ein Teilnehmer die Anmeldung an einer Konferenz selbst vornehmen kann.

Der erste Schritt besteht in der Definition der drei Hauptinselketten (Datenbank, Anwendungslogik und Webseiten). Die Datenbankinselkette ist verantwortlich für die Datenbankfunktionalität. Sie enthält die Datenbank, gespeicherte Prozeduren und Datenbanktabellen. Die Schicht der Anwendungslogik enthält die verschiedenen Geschäftsobjekte, die in den einzelnen Prozessen implementiert wurden. Die Inselkette mit den Webseiten definiert die verschiedenen Benutzerschnittstellenkomponenten.

Bei näherem Hinsehen enthält die Inselkette der Anwendungslogik drei Inseln – die Anmeldung, die Benutzerverwaltung und die Anwendungslogik für die Konferenzanmeldung, wie dargestellt in Abbildung 3.4.

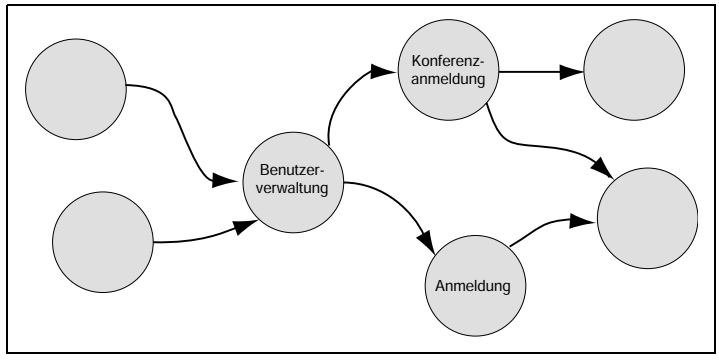

Abbildung 3.4 Plan der Inseln und Inselketten

Beim iterativen Ansatz werden zunächst einige wenige Inseln erstellt. Die Idee sieht vor, das Konzept der Anwendung basierend auf den funktionellen Komponenten zu verstehen. Anschließend, basierend auf diesen Inseln, wird die Kommunikation zwischen den Inseln definiert. So werden die Abhängigkeiten der einzelnen Anwendungselemente eingerichtet. In dieser Phase werden einige Inseln implementiert, damit das Konzept teilweise erprobt werden kann.

Wenn einige der Konzepte sich als falsch erweisen, können die Inseln neu geplant und strukturiert werden. Dies führt zu anderen Abhängigkeiten, und wieder müssen einige der Inseln implementiert werden, um die neuen Konzepte auf ihre Richtigkeit zu prüfen. Dieser Vorgang wird solange wiederholt, bis die Anwendung fertiggestellt ist.

Ein Nachteil dieses Ansatzes liegt darin, dass er langsam ist und nach dem Trial-and-Error-Prinzip verfährt. Der Grund für diese Kritik ist die Tatsache, dass ständig Verbesserungen und Aktualisierungen vorgenommen werden, d.h., es erfolgt eine immer neue Iteration. Man könnte dies mit einem Hauseigentümer vergleichen, der ständig Reparaturen an seinem Haus vornimmt. Das Haus unterliegt einem ständigen Wandel. Sie müssen in diesem Fall in der Lage sein, die Dinge so zu belassen, wie sie momentan sind und für einen späteren Zeitpunkt eine Reparatur anberaumen. Es gibt immer eine andere Version.

Stellen Sie sich nun die Situation vor, in der Sie einen Plan erstellt haben und jetzt eine neue Technologie oder einen neuen Geschäftsprozess einführen möchten. Das Problem besteht darin, dass neue Inseln nicht ohne weiteres hinzugefügt werden – es sind Legacykomponenten vorhanden. Und in einigen Fällen werden aufgrund der knappen Zeitbemessung für die Inselanpassung so genannte Hacks oder Shortcuts verwendet. (Tatsächlich werden immer Hacks und Shortcuts verwendet.) Beim Inselansatz kommt es darauf an, dass die Designer in der Entwurfsphase eine gute Entwurfsspezifikation vorlegen. Die Erfahrung zeigt, dass die Schnittstelle mit einer guten Entwurfsspezifikation viele Entwurfsiterationen überstehen und leichter an ein Legacyszenario angepasst werden kann.

Einen großer Vorteil des Inselansatzes stellt die Möglichkeit dar, individuelle Systeme zu entwickeln und zu testen und anschließend zusammenzuführen. Die Granularität des Systems richtet sich in diesem Fall nach den verschiedenen Teilsystemen. Die Granularität stellt die Codemenge einer einzelnen Insel dar. Ist der Code umfangreich, ist die Granularität hoch, ist der Code nicht umfangreich, liegt die Granularität niedrig und ist damit gut. Was Sie erreichen möchten, ist eine mittlere Granularität. Eine hohe Granularität erschwert das Testen, da zu viele Parameter geprüft werden müssen. Eine niedrige Granularität ist leichter zu testen, die Verfolgung aller Teilsysteme wird jedoch zu schwierig. Die exakte Definition einer mittleren Granularität wird in den folgenden Kapiteln definiert.

Der Cathedral-Ansatz Sie können Ihr System auch nach dem Cathedral-Ansatz gestalten. Bei diesem Ansatz wird ein System entwickelt und anschließend für eine bestimmte Zeit beibehalten. Die verwendete Technologie basiert auf den zum Entwicklungszeitpunkt verfügbaren Technologien, auch **Technologiesnapshot** genannt. Eine Technologie, die nach diesem Snapshot herausgebracht wird, wird ignoriert, da ansonsten der Entwicklungszyklus gestört würde. Der Zeitraum, für den eine Anwendung beibehalten wird, richtet sich nach dem Investitionsumfang. Häufig behalten Designer und Entwickler einen Lebenszyklus für **n** Jahre bei, und nach Ablauf dieser Zeitspanne wird ein nahezu komplett neues System entwickelt. Einige der Legacykomponenten werden hierbei beibehalten, dies jedoch lediglich aus Zeit- und Kostengründen.

Das Ersetzen einer Technologie wird im Allgemeinen durch veraltete Technologien erforderlich. Das neue System muss funktionieren, und dies bedeutet einen zusätzlichen Stressfaktor für das Entwicklungteam.

Beim Cathedral-Ansatz stellt sich das Problem, dass es nicht möglich ist, ein System basierend auf einem Technologiesnapshot zu entwickeln. Ein typisches Projekt nach dem Cathedral-Ansatz beinhaltet eine Definition der technologischen Anforderungen und benennt die technologischen Voraussetzungen. Das Projekt dauert etwa ein Jahr, und während dieses Jahres gibt es Änderungen im Technologiebereich. Im besten Fall handelt es sich lediglich um minimale Änderungen, im schlimmsten Fall umfassen die Änderungen grundlegende Neuerungen, beispielsweise die Einführung des Web. Auf diese Weise ist die ursprüngliche Technologie veraltet, bevor das Projekt abgeschlossen wurde. Eine neue technologische Definition ist erforderlich. Dies führt zu einem Mehraufwand bezüglich Zeit und Kosten.

Ein weiterer Nachteil des Cathedral-Ansatzes liegt darin, dass Sie nicht garantieren können, dass eine Anwendung auch in Zukunft effektiv einsetzbar ist. Wenn die Anwendung beispielsweise eine Lebensspanne von fünf Jahren aufweist, müssen verschiedene neue technologische Komponenten implementiert werden, um die Anwendung dauerhaft einsetzbar zu machen. Die neue Technologie ist jedoch noch nicht vollständig erprobt, d.h. es gibt keine Muster, die angewendet werden können, da es sich um eine völlig neue Domäne handelt. Daraus folgt, dass Entwurfsfehler entstehen, die wiederum zu einem erhöhtem Zeitaufwand, höheren Kosten usw. führen.

Iterativer und Cathedral-Ansatz im Vergleich Der iterative Ansatz beinhaltet ständige Änderungen, und ein solcher ständiger Wandel kann sehr kostenintensiv sein. Ein schlecht verwaltetes Projekt kommt nie zu einem richtigen Ende. Stattdessen wird es ständig weitergeführt. Daher wird häufig der Cathedral-Ansatz bevorzugt. Beim Cathedral-Ansatz gibt es eine Reihe von Zielen, die den

Anfang und das Ende eines Projekts markieren. Diese Ziele stellen jedoch gleichzeitig die Falle dar, denn sie ändern sich stetig.

Obwohl ich den Cathedral-Ansatz nicht befürworte, kann dieser gelegentlich nicht umgangen werden. Es gibt beispielsweise alte Systeme, die in einer eigenen Welt existieren, und diese Systeme bedürfen einer beständigen Justierung. Das Einbringen dieser Systeme in neue Entwicklungen stellt hierbei keine Option dar. Der einfachere Ansatz ist deshalb der, Ressourcen in das alte System fließen zu lassen und gleichzeitig ein ganz neues System zu entwickeln, mit dem die gleichen Ziele erreicht werden. Die Lösungen für das Jahr 2000-Problem waren Beispiele für dieses Vorgehen. Nur wenige Unternehmen integrierten hier alte und neue Technologien. Stattdessen wurde der zweigleisige Ansatz gewählt, der einfacher und weniger risikoreich ist.

Kontextueller Entwurf

Der zweite Teil des iterativen, kontextuellen Entwurfs besteht in der Definition des Kontextes. Bei Projekten mit iterativem Entwurf wird häufig zunächst ein Plan erstellt, aber verschiedene Teams verfahren unterschiedlich, sodass Inseln entwickelt werden, die nicht definiert wurden. Manchmal sind diese Änderungen richtig, doch häufig sind diese Zusätze das Ergebnis aus Lücken im anfänglichen Entwurf, die Bestandteil des gesamten Anwendungsentwurfs hätten sein müssen. Diese fehlenden Details erzeugen Inkonsistenzen, die wiederum zu Testproblemen führen.

Zur Einbringung dieser fehlenden Details in die Inselerstellung müssen zwischen den Projektteams Kommunikationskanäle eingerichtet werden. Die Kommunikation ist äußerst wichtig, da der iterative Entwurf ständige Änderungen beinhaltet. Alle Teams müssen über Änderungen informiert werden, sonst können Teilsysteme nicht richtig funktionieren.

Der kontextuelle Entwurf umfasst das Projektmanagement, dessen Erläuterung jedoch nicht im Rahmen dieses Buches erfolgen kann. Empfehlenswert ist hierzu das Buch **Contextual Design** (siehe Anhang), in dem mit aller Ausführlichkeit auf die zugrunde liegenden Konzepte eingegangen wird.

Resümee

Der Vorteil einer iterativen, auf Inseln basierenden Entwicklung ist der, dass Sie neue Technologien einbringen können, ohne das gesamte System umgestalten zu müssen. Sie können die Auswirkungen einer neu hinzugekommenen Insel auf die ganze Inselkette erfassen. Basierend auf diesen Auswirkungen kann die Insel erweitert oder abgelehnt werden.

Die sich ergebenden Auswirkungen sind wichtig, da Sie anzeigen, welche Fähigkeiten Ihre Anwendungen aufweist. Es gibt zwei Arten von Auswirkungen, technologische Auswirkungen und die Folgen der Interaktion von Technologien.

Die technologischen Auswirkungen müssen abgeschätzt werden, denn normalerweise ist die Technologie entweder noch neu und unerprobt oder wurde zumindest noch nicht zur Lösung des betreffenden Problems eingesetzt. Das Definieren der Auswirkungen ist schwierig, da keine direkten Erfahrungswerte herangezogen werden. Es gibt jedoch häufig indirekte Erfahrungswerte, die extrapoliert werden können.

Die Berücksichtigung von Auswirkungen ist besonders wichtig. Eine leistungsstarke Technologie, die allein betrachtet gut funktioniert, kann in der vorliegenden Situation möglicherweise nicht in andere Komponenten integriert werden. In diesem Fall kann die Technologie nicht eingesetzt werden.

Betrachten Sie beispielsweise die Auswirkungen bei der Adaptation von HTML (Hypertext Markup Language) als Benutzerschnittstelle. HTML unterscheidet sich stark von jedem anderen Benutzerschnittstellentyp, daher erscheint die Einschätzung der Folgen nahezu unmöglich. HTML stellt jedoch keine Neuentwicklung, sondern eine Weiterentwicklung von SGML (Standard Generalized Markup Language) dar. Daher sollten Sie bei der Betrachtung von HTML die Erfahrungen mit SGML berücksichtigen. Eine Überlegung ist, dass SGML zu komplex ist; HTML wurde aus diesem Grund vereinfacht. SGML ist ein Standard, der plattformübergreifend ist, und HTML weist diesen Vorteil ebenfalls auf. SGML ist jedoch dokumentbasiert, d.h., es weist keine vielfältige Funktionalität auf. In seiner reinen Form weist auch HTML diese Beschränkung auf. Daher ist als Auswirkungen der Verwendung von HTML eine plattformübergreifende Benutzerschnittstelle zu erwarten, die jedoch in ihrer Funktionalität eingeschränkt ist.

Stellen Sie sich vor, Sie kombinieren die HTML-Schnittstelle mit serverseitigen Daten. Hierzu stehen verschiedene Methoden zur Verfügung. Eine Methode wäre, alle HTML-Seiten mit Hilfe eines Stapelverarbeitungsprozesses zu generieren, der zu bestimmten Zeitpunkten ausgeführt wird. Die Auswirkung hierbei ist, dass die Aktualität der Anwendungsdaten davon abhängt, wann der Stapelverarbeitungsprozess das letzte Mal ausgeführt wurde. Eine weitere Methode besteht darin, einige Komponenten zu schreiben, mit denen die Seite dynamisch erzeugt wird. Dies führt jedoch zu der Frage, welche Technologie verwendet wird, und es sind weitere Folgen zu berücksichtigen. Sollte ASP (Active Server Pages) oder ISAPI (Internet Server Application Programming Interface) verwendet werden?

Der verbleibende Teil des Buches richtet sich auf die Anwendung der Technologie und darauf, die Auswirkungen der jeweiligen Technologie richtig einzuschätzen.

3.2 Starten eines Projekts

Jetzt, nachdem die Konzepte definiert sind, wollen wir uns der Definition eines Projekts und dessen Umsetzung zuwenden. Das verwendete Projekt wurde tatsächlich entwickelt und für Microsoft Deutschland implementiert. Auf Teile dieses Projekts wird im gesamten Buch verwiesen. Da durch ein Projekt jedoch niemals alle Funktionen implementiert werden können, werden darüber hinaus auch andere Implementierungsszenarien betrachtet.

3.2.1 Framework: ja oder nein?

Bei der Entwicklung einer Anwendung müssen Sie auch entscheiden, ob Sie zur Vereinfachung der Entwicklung weiterer Anwendungen ein Framework einsetzen oder nicht.

Die Antwort lautet, dass Sie entsprechend den Anwendungszielen unter Verwendung des iterativen Entwurfs entwickeln. Wenn das Ziel der Anwendung in der Entwicklung eines Frameworks besteht, dann sollten Sie dieses Ziel schrittweise umsetzen. Versuchen Sie nicht, den Cathedral-Ansatz zu verwenden, da dies lediglich zu Problemen führt. Stellt die Entwicklung eines Frameworks kein Ziel dar, sollten Sie auch keines entwickeln – das Entwickeln von Frameworks ist schwierig und erfordert umfassende Kenntnisse zu Domäne und verwendeter Technologie.

3.2.2 Einige Hintergrundinformationen

Beim Kombinieren von Anwendungsentwicklungszyklus, iterativem, kontextuellem Entwurf und Kontext stellt der ersten Schritt das Hinterfragen des Hintergrunds eines Projekts dar – Warum ist das Projekt erforderlich? Dieser Vorgang wird als Erarbeitung des Projektkontextes bezeichnet.

In unserem Beispiel lautet die Antwort auf diese Frage, dass Microsoft zu einem sehr großen Unternehmen geworden ist, und sich daher – wie bei allen großen Unternehmen – einige Prozesse schneller ändern als andere. Microsoft Deutschland benötigte ein System für die Konferenzanmeldung, und obwohl Microsoft ein solches System besitzt, genügte dieses System nicht den speziellen Anforderungen der deutschen Firmenniederlassung. Microsoft führte zur Lösung des Problems eine Untersuchung bei allen Zweigniederlassungen durch, die jedoch Zeit beanspruchte. Deshalb entschied sich Microsoft Deutschland zur Entwicklung einer Zwischenlösung, die die vorhandene Lücke schließen sollte, bis das unternehmensweite System verfügbar wurde.

Die Hintergrundinformationen zu diesem Projekt lauteten folgendermaßen:

- **Kurzfristigkeit:** Es handelte sich um ein zeitlich begrenztes Projekt. Es bestand keinerlei Notwendigkeit zur Entwicklung eines evolutionären Systems, da das System ein Jahr später verworfen werden sollte.
- **Legacydaten:** Die Anwendung würde Legacydaten erzeugen, die in das neue Unternehmenssystem von Microsoft eingebettet werden müssten. Darüber hinaus war eine Interaktion der vorhandenen Daten und Anwendungen anderer Microsoft-Zulieferer erforderlich.
- **Erfahrungswert:** Es musste geeignete Technologie eingesetzt werden, aber die Implementierung bestand auch im Testen anderer Microsoft-Technologien, um Erfahrung zu sammeln. Zum vollständigen Verständnis der Entwicklung des großen Unternehmenssystems mussten Erfahrungswerte durch kleine Systeme zusammengetragen werden, diese Erfahrung sollte anschließend als Muster für das große System verwendet werden.

Diese Hintergrundinformationen vermitteln den Kontext der Anwendung – Microsoft beabsichtigte, wenig zu investieren, aber möglichst viel Erfahrung zu sammeln.

Erarbeiten des Kontextes

Die Erarbeitung des Kontextes für das Konferenzanmeldungssystem war nicht einfach, da diese Lösung nur temporär das eigentliche System ersetzen sollte. Diese Situation macht das Entwickeln einer Anwendung besonders interessant, da Sie sowohl die Produkteinführungszeit als auch die Verwendungsdauer des Produkts berücksichtigen müssen.

Der Kontext wird aus den Informationen erarbeitet, die während der Meetings und im Gespräch mit den beteiligten Personen zusammengetragen werden können. Hierbei werden keine Projekteinzelheiten, sondern Hoffnungen und Erwartungen besprochen. In dieser Phase erfahren Sie, auf welche Punkte Wert gelegt wird und welche Punkte dem Kunden weniger wichtig sind. Beim Konferenzanmeldungssystem beispielsweise war ein sauberer, wieder verwendbarer Entwurf nicht wichtig, die Produkteinführungszeit und die Stabilität stellten die Hauptziele dar. Dies ist wichtig, da ein sauberer Entwurf Zeit und Kosten erfordert. Dies bedeutet nicht, dass ein Projekt keinen Entwurf benötigt. Es heißt lediglich, dass sich der Entwurf auf den Erhalt eines lauffähigen Systems unter Verwendung neuer Technologie konzentriert.

Das Verständnis der grundlegenden Ziele ermöglicht das Implementieren der Funktionen, mit denen der Kunde unter geringstmöglichem Zeit- und Kostenaufwand zufrieden gestellt werden kann. Aber genau dieser Punkt wird von vielen

Softwareentwicklern falsch verstanden. Softwareentwickler möchten häufig möglichst »reine« Anwendungen entwickeln, hierzu ist jedoch viel Zeit und Geld erforderlich. Worauf es ankommt, sind jedoch die Kundenwünsche.

Üblicherweise sollten Sie mit den Endbenutzern sprechen. Verlassen Sie sich nicht auf die gedruckte Dokumentation, da in dieser weder Emotionen noch Hoffnungen ausgedrückt werden, es werden hier ausschließlich die Unternehmensvorstellungen dargestellt. Emotionen spielen im Prozess der Anwendungsentwicklung eine sehr große Rolle. Missfällt dem Kunden eine Kleinigkeit, kann dies dem Gesamtbild schaden. Fällt dem Kunden jedoch eine Kleinigkeit positiv auf, kann dies dem Gesamtbild förderlich sein.

Durch das Gespräch mit den Endbenutzern wird außerdem sichergestellt, daß die Wünsche des Managements mit denen der Endbenutzer übereinstimmen. Besteht in diesem Punkt ein Konflikt, muss dieser umgehend gelöst werden. Dieser Schritt in der Phase der Kontexterarbeitung kann darüber entscheiden, ob die Anwendung akzeptiert wird oder nicht. Nach meiner Erfahrung wird bei Unstimmigkeiten zwischen den Erwartungen von Endbenutzer und Management häufig der Entwickler der Anwendung für daraus resultierende Probleme verantwortlich gemacht. Häufig wird in diesem Fall die Anwendung als inakzeptabel betrachtet, und da der Vertrag scheinbar nicht eingehalten wurde, kommt es vor, dass der Lieferant nicht voll bezahlt wird.

Wie viele Hintergrundinformationen sind erforderlich?

Das Problem beim Zusammentragen der Informationen besteht darin, dass Sie mit Informationen überschwemmt werden. Wann liegen zu wenig Informationen, wann zu viele Informationen vor?

Erfolgreiche Projekte folgen der nachstehenden Faustregel:

> *Wenn Sie für eine Insel einer Anwendung verantwortlich sind, dann tragen Sie die nötigen Informationen für diese Insel zusammen. Alle weiteren Informationen sind in der Regel irrelevant.*

Die einzige Ausnahme für diese Regel besteht darin, dass jeder ein Gesamtbild davon haben muss, welche Funktion die Anwendung übernimmt, damit das gemeinsame Ziel erreicht werden kann.

Dies ist ein scheuklappenartiger Ansatz, aber er ist erfolgreich. Wenn Sie nicht diesem Ansatz folgen, wird es häufig zu Problemen kommen, die eigentlich nicht vorhanden sind oder nicht in Ihrem Verantwortungsbereich liegen, was wiederum zu einem höheren Zeit- und Kostenaufwand führt.

Beispielsweise war einer der Punkte, den wir im Rahmen des Kontextes festgelegt haben, dass eine Insel dann richtig entworfen und implementiert wurde, wenn dies im Interesse des Kunden des Konferenzanmeldungssystem erfolgt. Im Beispiel mussten Entwurf und Implementierung lediglich »gut genug« sein. Wieder war dies ein Ergebnis der Tatsache, dass es sich bei dem Projekt um eine Zwischenlösung handelte. Eine Lösung nach dem Prinzip »gut genug« mag für einige Designer und Entwickler ein harter Brocken sein, aber die Situation war die, dass einige Bestandteile des Systems in absehbarer Zeit ausgemustert werden würden.

3.2.3 Anwendungsanforderungen

Nachdem der Kontext definiert ist, muss der erste Schritt im Anwendungsentwicklungszyklus implementiert werden. Hierbei wird der Prozess definiert. Bei diesem Schritt wird der allgemeine Entwurf der Insel und Inselketten geplant. In vielen Büchern wird dieser Vorgang als Entwurf der grundlegenden Architektur bezeichnet. Bei diesem Schritt wird die Dynamik der Anwendung beschrieben und die Intention der Anwendung definiert. Die Kombination dieser zwei Aspekte soll Sie dabei unterstützen, die Systemparameter für die Optimierung zu ermitteln. Die Optimierung bezeichnet das Ändern von Teilen eines Systems und das Ermitteln der Auswirkungen, die sich hieraus für die weiteren Bestandteile des Systems ergeben.

Bei der Definition der grundlegenden Architektur werden die Ziele der Anwendung zeitweise ignoriert. Sie überlegen, wie die verschiedenen Technologien und Ideen zur Lösung Ihres Problems eingesetzt werden könnten. In dieser Phase sollten Sie verschiedene Ideen und Konzepte ausprobieren. Sie möchten sicherstellen, dass die implementierte auch die optimale Lösung darstellt. Das Testen von Technologien zu einem späteren Zeitpunkt führt zu einer Störung des Entwicklungsprozesses.

Zur Definition des Prozesses müssen folgende Dinge vorliegen:

- Ein Domänenmodell, mit dem die Anwendung in einfachen Worten definiert wird
- Eine definierte Technologieplattform, die zur Umsetzung der Anwendung eingesetzt wird
- Eine Liste mit Anwendungsfällen (Use Cases)

Die Technologieplattform wird nicht behandelt, da vorausgesetzt wird, dass Sie Windows DNA verwenden. Wie die Dienste von Windows DNA angewendet werden, wird im weiteren Verlauf des Buches erläutert.

Domänenmodell

Das Domänenmodell dient der Anwendungsbeschreibung in Worten, die allgemein verständlich sind. Es stellt die Grundlage Ihrer Anwendung dar.

Idealerweise handelt es sich bei dem Domänenmodell um einen geschriebenen Text. Der Text sollte präzise und interessant sein. Das Problem bei einem langweiligen Domänenmodelltext liegt darin, dass die Aufmerksamkeit der Leser nachlässt und ihnen deshalb wichtige Punkte entgehen.

Das Domänenmodell enthält die Begriffe, die im verwendeten Kontext ausführlich beschrieben werden. Das Wort »Titel« beispielsweise hat mehrere Bedeutungen, es kann den Titel einer Person oder den Titel eines Buches bezeichnen. In einem Domänenmodell wird explizit definiert, was für eine Bedeutung Begriffe wie »Titel« besitzen.

Ein guter Domänenmodelltext weist folgende Merkmale auf:

- Es handelt sich nicht lediglich um eine Aufzählung, da Informationen in Form einer Auflistung nicht richtig vermittelt werden können. Das Schreiben einer guten und erklärenden Liste ist schwierig. Eine schlechte Liste deutet entweder auf Faulheit oder Zeitmangel hin.
- Der Text enthält nicht zu viele Schaubilder. Üblicherweise werden Schaubilder eingefügt, um dem Dokument ein wichtigeres Aussehen zu verleihen. Schaubilder unterbrechen jedoch den Lesefluss, und wenn das Diagramm nicht ordnungsgemäß platziert ist, lenkt es den Leser vom Thema ab oder, noch schlimmer, bringt den Leser dazu, nur die Diagramme zu lesen. Der Text wird ignoriert und wichtige Details werden übersehen, was bei der späteren Implementierung zu Problemen führen kann.
- Ein guter Domänenmodelltext ist nicht zu lang. Es sollte nicht mehr als Domänenmodell und Anwendung erläutert werden. Obwohl es eindrucksvoller erscheinen mag, mit einer 30seitigen Erklärung aufzuwarten, ist der Text deswegen nicht besser. Der Schlüssel liegt darin, präzise zu sein.

Schreiben eines Domänenmodelltextes Wie schreibt man einen Domänenmodelltext? Die Antwort ist, mit Übung und Geduld. In technischen Hochschulen und Computerschulen wird dem Berichteschreiben häufig keine besondere Aufmerksamkeit geschenkt. Die Hochschulen (und Sie vielleicht auch) setzen voraus, dass Ihnen dieses Wissen in der Schulzeit vermittelt wurde. Es besteht jedoch ein großer Unterschied zwischen der Fähigkeit, etwas in eigene Worte zu fassen, und der Fähigkeit, diese Gedanken präzise und strukturiert in einer Präsentation darzulegen, die allgemein verständlich ist. Richtig oder falsch, häufig wird die Qualität einer schriftlichen Präsentation als Indikator für die Qualität ei-

nes gesamten Projekts herangezogen. Die Fähigkeit, sich stilistisch gut und gleichzeitig klar auszudrücken, stellt einen unschätzbaren Wert dar.

Ich weiß dies nur zu gut, da Deutsch für mich immer das schlimmste Lehrfach darstellte. In der Universität musste ich beinahe ein Semester wiederholen, da mein Bericht nicht lesbar war. Mein damaliger Professor besaß jedoch die Freundlichkeit, mir zu erläutern, aus welchen Gründen der Bericht ihm nicht akzeptabel erschien. Daher konnte ich drei Berichtsversionen später das Semester doch noch erfolgreich abschließen.

Für meine Karriere als technischer Autor hat sich diese Lektion jedoch als sehr nützlich erwiesen, wie Sie sich vielleicht vorstellen können. Der Domänenmodelltext sollte in einem journalistischen Stil geschrieben werden, eher in der Form eines technischen Artikels als in Form eines Buches. Das Schreiben eines Buches und das Verfassen eines Artikels sind zwei völlig verschiedene Dinge.

In einem Buch erzählen Sie eine Geschichte, die am Anfang des Buches beginnt und auf der letzten Seite des Buches endet. Das vorliegende Buch ist eine Geschichte über Windows DNA. In einem Buch wiederholen Sie sich in der Regel nicht, obwohl Sie vielleicht von Zeit zu Zeit einzelne Passagen wieder aufgreifen.

Das Schreiben eines Buches führt zu einer detaillierten Geschichte, bei der eine Zusammenfassung nicht auf allen Ebenen erfolgt, die der Leser vielleicht erwartet. Ein Manager, der nur eine kurze Synopsis von Windows DNA benötigt, wird diese im vorliegenden Buch nicht finden. Dieser Manager sollte sich besser nach einer anderen Informationsquelle umsehen.

Technische Artikel beschäftigen sich häufig mit einem spezifischen Themenbereich und umfassen keine Geschichte, die im ersten Absatz beginnt und mit dem letzten endet. Stattdessen enthalten technische Artikel eine Reihe von Informationsschleifen, wie in der folgenden Abbildung dargestellt.

Im Schaubild ist ein kleiner Kreis dargestellt. Dieser repräsentiert den ersten Absatz des Artikels, in dem der Themenschwerpunkt erläutert wird. Der zweite Kreis ist größer – der zweite Absatz und vielleicht der dritte, in dem die Informationen des ersten Absatzes ausgeweitet werden. In diesen Absätzen werden die Informationen des ersten Absatzes erneut aufgegriffen und näher ausgeführt. Dieser Prozess setzt sich innerhalb des Artikels immer weiter fort.

Warum ist dies besser? Es ist einfach, den Artikel an einem bestimmten Punkt enden zu lassen. Enthielt der ursprüngliche Artikel beispielsweise 5.000 Wörter und drei Schleifen, so können Sie durch Auslassen der letzten Schleife einen kürzeren Artikel für eine höhere Managementebene erstellen, die ein weniger detailliertes

Gesamtbild wünschen. Der Artikel enthält hierbei die gleichen Informationen, jedoch mit weniger Details.

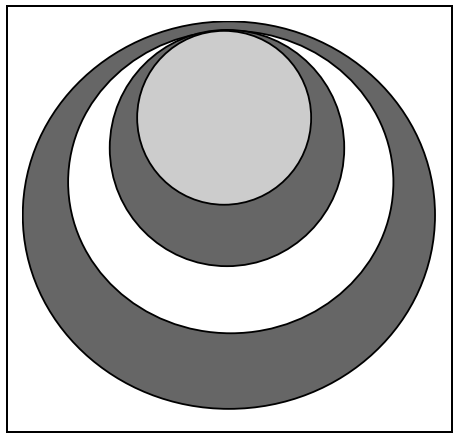

Abbildung 3.5 Informationsschleifen in technischen Artikeln

Mit diesem Ansatz können Sie ein einzelnes Dokument auf die verschiedenen Ebenen von Management, Programmierern, Entwicklern usw. anpassen. Das Management der höchsten Ebene wird nur die einführende Zusammenfassung lesen, die Programmierer interessieren sich für alle sie betreffenden Details. Es ist möglich, bestimmte Bereiche zu kürzen und dennoch das Gesamtkonzept zu verstehen.

Beispiel für einen Domänenmodelltext Als Beispiel möchte ich den Domänenmodelltext anführen, der für das Konferenzanmeldungsprojekt verwendet wurde:

> *Es ist erforderlich, eine webbasierte Anwendung zur Konferenzanmeldung zu entwickeln.*

Dies ist die erste Schleife erläuternder Informationen. Hier wird gesagt, was erreicht werden soll. In diesem Fall die Entwicklung eines webbasierten Konferenzanmeldungssystems. Kommen wir zu den Details:

> *Das Konferenzanmeldesystem ist eine webbasierte Anwendung. Unter Verwendung dieses Systems kann der Benutzer nähere Informationen zu einer Konferenz anzeigen. Ist die Konferenz von Interesse für den Benutzer, kann er sich durch Drücken auf eine Schaltfläche für diese Konferenz anmelden. Diese Anmeldung wird in Echtzeit ausgeführt, d.h., die Konferenzveranstalter verfügen stets über aktuelle Anmeldeinformationen.*

Dieser zweite Absatz beginnt mit der Erläuterung der Anwendungsdetails. Sie müssen den ersten Absatz nicht gelesen haben, da im zweiten Absatz die genannten Informationen wiederholt werden.

Es werden Informationen zur Systemdynamik bereitgestellt. Es gibt einen Benutzer – einen potenziellen Konferenzteilnehmer. Der Benutzer könnte durch ein Objekt (Insel) in der Anwendung repräsentiert werden. Es ist wichtig, das Objektnamen und Domänenmodelltext zueinander in Beziehung stehen. Andernfalls wird der Leser verwirrt.

Des Weiteren werden bestimmte Prozesse offensichtlich. Diese Prozesse stellen die Funktionen der Anwendung dar. Der Domänenmodelltext sollte so viele Funktionen wie möglich definieren.

Abschließend sollten die Anwendungsanforderungen genannt werden. Die Anmeldung soll in Echtzeit erfolgen, damit den Konferenzorganisatoren stets aktuelle Daten vorliegen. An diesem Punkt ist es interessant, zu hinterfragen, warum diese Echtzeitfunktion nicht bereits früher implementiert wurde. Das Sammeln von Hintergrundinformationen ist ein wichtiger Aspekt bei der Projektarbeit. Häufig wurden bestimmte Aufgaben nicht ausgeführt, da beispielsweise die entstehenden Kosten zu hoch lagen. Nach einiger Zeit geraten diese Gründe in Vergessenheit, ein zweiter Versuch wird gestartet und möglicherweise kommt es erneut zu einem Misserfolg. Wenn Sie herausfinden, warum die Echtzeitfunktion nicht bereits früher implementiert wurde, können Sie ermitteln, ob immer noch etwas gegen eine solche Implementierung spricht. Ist dies der Fall, können Sie diese Anforderung aus dem Domänenmodelltext streichen.

3.2.4 Definieren der Anwendung

Nachdem die Anwendung untersucht und beschrieben wurde und die zu implementierenden Funktionen festgelegt wurden, beginnt nun die Entwicklung der Anwendung. In der Begrifflichkeit des Anwendungsentwicklungszyklus handelt es sich hierbei um den Schritt des Modellentwurfs. Die beste Methode zur Entwurfsbeschreibung stellt die UML (Unified Modeling Language) dar. UML ermöglicht das Entwerfen der Anwendung, ohne dass das Schreiben von Code erforderlich ist. Bei UML handelt es sich um ein abstraktes Modellierungsverfahren, bei dem Elemente verwendet werden, die durch Pfeile und Linien miteinander verbunden werden.

Wie steht es mit CASE?

CASE (Computer Aided Software Engineering) war ein Schlagwort, das mit der Vision einer höheren Produktivität und einer gleichzeitigen Vereinfachung der An-

wendungsentwicklung verknüpft wurde. Der Gedanke war, mit Hilfe einer Art Notation eine vollständige Anwendung entwerfen zu können, und UML war ein Beispiel für dieses Verfahren. Sie klicken auf die Schaltfläche zum Generieren, und das CASE-Tool gibt eine fertige Anwendung aus.

Ja, dies ist möglich; aber, nein, Sie sollten diese Methode nicht einsetzen. Die Probleme von CASE können folgendermaßen zusammengefasst werden:

- **Aufgeblähter Code:** Es ist einfach, schöne Grafiken zu erstellen und anschließend durch das CASE-Tool automatisch die Klassen generieren zu lassen. Diese Art der Vorgehensweise führt jedoch zu einer Bequemlichkeit, bei der der Designer ein Diagramm nach dem anderen entwirft. Ich habe einmal an einem solchen Projekt mitgearbeitet – das Projekt umfasste 350 Geschäftsobjekte, was zu 15.000 Java-Klassendateien führte.
- **Kleinster gemeinsamer Nenner:** Die meisten Notationen unterstützen lediglich ein einfaches, objektorientiertes Framework. Einfache Konzepte, beispielsweise Vererbung und Verknüpfung, werden unterstützt, erweiterte Programmierungskonzepte, beispielsweise C++-Vorlagen, dagegen nicht. Daher muss bei Verwendung von CASE einfacher Code geschrieben werden. Dies ist an sich nicht schlimm, kann aber zu einem Problem werden, wenn viele Strukturen hinzugefügt werden, um den einfachen Code mit den Funktionen eines eleganteren, komplexen Codes zu versehen.

Ist die Verwendung von CASE demnach nicht zu empfehlen? Nein, es bedeutet lediglich, dass CASE nur in einem angemessenen Kontext eingesetzt werden sollte. Der Grund für die Entwicklung von CASE ist der, dass einige Programmierungsprojekte einfach zu umfangreich und komplex sind, um mit Hilfe einer gesprochenen Sprache wie beispielsweise Englisch umschrieben zu werden. Mit Hilfe von CASE können die Konzepte einiger höherwertiger Strukturen vereinfacht werden, da Sie zur Beschreibung des Softwareentwurfs eine Notation verwenden.

Sie sollten CASE als Hilfe zur Strukturierung Ihrer Gedanken einsetzen. UML und die Tools zur Anwendungsmodellierung ermöglichen Ihnen das Umsetzen von Konzepten und Ideen. Diese Konzepte und Ideen ermöglichen Ihnen ein Verständnis der Dynamik sowie der möglichen Problembereiche einer Anwendung. Das Verwenden eines solchen Tools gleicht dem Erstellen einer Miniaturversion Ihrer Anwendung. Es werden nicht alle Details gezeigt, aber Sie erhalten einen Eindruck davon, was Sie erwartet.

Steht der Nutzen eines solchen Diagramms nicht im Verhältnis zum erforderlichen Arbeitsaufwand, sollten Sie keines erstellen. In diesem Fall verschwenden Sie lediglich Ihre Zeit. Einige Anwendungen benötigen aufgrund Ihrer Einfachheit

keine solchen Modelltools. Andere Anwendungen erfordern erhebliche Arbeit für den Modellentwurf, da sie sehr umfangreich oder komplex sind. Dies hängt ganz von der jeweiligen Anwendung ab. Der Anhang enthält eine Liste mit weiterführender Literatur zu diesem Thema.

Abstriche bei der Funktionsimplementierung

Beim Modellentwurf möchten Sie üblicherweise alle der möglichen Funktionen in das Modell einarbeiten. Einige der angestrebten Funktionen sind hierbei nützlicher und wichtiger als andere.

Üblicherweise gibt es einige Funktionen, deren Implementierung unbedingt erforderlich ist. Dies stellt kein Problem dar, da es sich um die Grundlage der Anwendung handelt. Problematischer dagegen ist es, herauszufinden, welche der (vielen) gewünschten Extrafunktionen implementiert und welche Funktionen erst in der nächsten Anwendungsversion umgesetzt werden können.

Sie können zu diesem Zweck eine Gleichung aufstellen, bei der die folgenden Attribute berücksichtigt werden:

- **Zeit:** Wie viel Zeit wird zur Implementierung einer bestimmten Funktion benötigt? Dieser Faktor kann in Zeiteinheiten gemessen werden.
- **Vorteil:** Welche direkten Vorteile hat das Hinzufügen dieser Funktion? Der Vorteil einer Funktion kann in Begriffen verbesserter Verwendbarkeit, einfacherer Codeverwaltung oder reduziertem Testaufwand ausgedrückt werden. Je mehr Vorteile, desto höher die Wertung.
- **Eindruck:** Ist die Funktion »Zucker fürs Auge«? Etwas, das den Benutzer zum Lächeln bringt? Es kann sich um eine vollkommen triviale Funktion handeln, die nicht Teil der gesamten Anwendung ist, deren Implementierung jedoch die Benutzer freuen würde. Häufig sind die Argumente, die für Funktion dieser Art sprechen, irrational und hängen mit einer bestimmten Vorliebe zusammen, beispielsweise für Java, Client/Server, Middleware usw. Je besser der gewonnene Eindruck, desto höher die Wertung.
- **Auswirkungen:** Wie stark wirkt sich das Hinzufügen der Funktion auf das System aus? Die Auswirkungen stehen mit dem Testaufwand in Verbindung, der aufgebracht werden muss, bis eine gewisse Stabilität erreicht werden kann. Je höher die Auswirkungen auf die Anwendung, desto höher die Wertung.
- **Priorität:** Die Priorität, die den einzelnen Funktionen der Anwendung beigemessen wird. Einige Funktionen müssen implementiert werden, da sie zur Hauptfunktionalität des Systems gehören. Diese Funktionen werden von der Auswahl der zu implementierenden Funktionen ausgenommen. Andere Funk-

tionen weisen unterschiedliche Prioritäten auf. Je höher die Priorität, desto höher die Wertung.

Die Wertung sollte anhand einer gemeinsamen Skala für alle Attribute erfolgen. Die Skala kann von 1 bis 10 oder von 1 bis 100 reichen. Schließen Sie die 0 nicht ein, da diese zu mathematischen Fehlern führen kann. Schließen Sie den Zeitfaktor ein, indem Sie die Zeit für die Implementierung nehmen und durch die verbleibende Gesamtzeit des Projekts dividieren.

Wenden Sie dann folgende Formel an:

(Vorteil x Eindruck x Priorität) / (Zeit x Auswirkungen)

Das Ergebnis stellt keinen besonderen Wert dar. Es dient lediglich als Vergleichswert gegenüber anderen Funktionen. Je höher der Ergebniswert, desto wichtiger ist eine Funktion.

Ein mögliches Problem bei dieser Gleichung ist, dass den einzelnen Bewertungsfaktoren gleiche Bedeutung eingeräumt wird. Es kann jedoch vorkommen, dass bei einem Projekt ein Faktor wichtiger ist als beispielsweise der Eindruck oder die Priorität. In diesem Fall müssen Sie jedem der Attribute einen Gewichtungsfaktor zuordnen. Der einfachste Weg, eine Gewichtung vorzunehmen, stellt das Verwenden von prozentualen Werten dar, wobei sichergestellt werden muss, dass alle Faktoren zusammen den Wert 100 ergeben. Ist die Zeit der wichtigste Faktor, können Sie die folgende Gleichung verwenden:

((0,1 x Vorteil) x (0,1 x Eindruck) x (0,1 x Priorität)) /
((0,6 x Zeit) x (0,1 x Auswirkungen))

In Ihrer speziellen Entwicklungssituation können natürlich auch andere Attribute erforderlich sein. Zum Einschluss dieser Faktoren müssen Sie den Attributwert lediglich über oder unter dem Divisor einfügen. Ist ein Attribut erwünscht, wird der zugehörige Wert über dem Divisor platziert. Ist ein Attribut unerwünscht, wird der zugehörige Wert unter dem Divisor platziert.

Sie sollten nicht zu viele Attribute verwenden, da so das Ergebnis weniger klar ausfällt. Ein Beispiel hierfür ist der Kostenfaktor. Die Kosten scheinen einen eigenen Faktor darzustellen, tatsächlich handelt es sich jedoch um eine Auswirkung eines anderen Attributs. Höhere Kosten resultieren beispielsweise aus einem erhöhten Zeitaufwand, stärkeren Auswirkungen oder geringeren Vorteilen. Das Erhöhen des Zeitfaktors für eine Funktion führt automatisch zu einer Erhöhung der Kosten, und wenn die Kosten ebenfalls in die Gleichung aufgenommen werden und nicht anderweitig ausgeglichen werden, führt die Gleichung zu irreführenden Ergebnissen. Die Mathematik kann Ihnen ein beliebiges Ergebnis liefern, wichtig ist, wie Sie das Ergebnis interpretieren.

Wenn Sie sich unter Verwendung von Gleichungen für oder gegen die Implementierung einer Funktion entscheiden, bedeutet dies, dass diese Entscheidung von den Gesamtvorteilen für die Anwendung abhängig gemacht wird. Funktionen werden nicht auf der Grundlage eines einzelnen Attributs implementiert.

Anwendungsfälle und Sequenzdiagramme

Basierend auf der Einschätzung des Für und Wider einer Funktionsimplementierung wird ein Satz Funktionen ausgewählt, der anschließend entworfen werden muss. Funktionen werden unter Verwendung von UML-Anwendungsfällen (so genannte Use Cases) implementiert.

In der UML-Terminologie bezeichnet ein Anwendungsfall eine Beschreibung der unterschiedlichen Anwendungsverwendung durch verschiedene Benutzer. Der Anwendungsfall (Use Case) beschreibt eine Interaktion zwischen Benutzer und Anwendung und vermittelt Ihnen einen Eindruck davon, welche Möglichkeiten dem Benutzer bei der Anwendungsverwendung zur Verfügung stehen.

Der Domänenmodelltext bietet ein Gesamtbild sowie ein Verständnis der gesamten Anwendung, die Anwendungsfälle zeigen verschiedene Aspekte der Anwendung auf. Nachdem Sie sämtliche Anwendungsfälle für eine Anwendung kennen, können Sie ermitteln, welche Anwendungsfälle wieder verwendet werden können, und Sie können die Abhängigkeiten zwischen den verschiedenen Anwendungsfällen einrichten.

In UML stellen die Anwendungsfälle eine einfache Struktur dar, die jedoch nicht sehr selbsterklärend ist. Üblicherweise möchten Sie einen Anwendungsfall mit einem Sequenzdiagramm kombinieren. Ein Sequenzdiagramm verdeutlicht die verschiedenen Schritte, die zur Ausführung einer bestimmten Aufgabe erforderlich sind. Ein Sequenzdiagramm zeigt typischerweise die Implementierung eines Anwendungsfalles (Use Case). Ein Anwendungsfall kann mit mehreren Sequenzdiagrammen verknüpft sein.

Ein einfacher Anwendungsfall Lassen Sie uns zunächst den Anwendungsfall für einen Benutzer darstellen, der die Konferenzwebsite durchsucht. Als Tool zur Darstellung von Anwendungsfällen verwende ich Rational Rose, das von der Rational Software Corp. entwickelt wurde. Ich verwende dieses Tool, da ich es bereits häufig eingesetzt habe. Sie können natürlich auch ein anderes Tool verwenden. Das verwendete Tool spielt keine Rolle, da lediglich die Anwendungsfälle veranschaulicht werden sollen – hierzu kann auch Paint oder PowerPoint eingesetzt werden. Der zum Lieferumfang der Enterprise Edition von Visual Studio gehörende Visual Modeler stellt einen Teilsatz von Rational Rose dar, durch den

keine Anwendungsfälle oder Sequenzdiagramme unterstützt werden. Sie können Visual Modeler dennoch genau wie Paint oder PowerPoint einsetzen.

Das Anwendungsfalldiagramm für einen Benutzer, der die Konferenzsite durchsucht, wird in Abbildung 3.6 dargestellt.

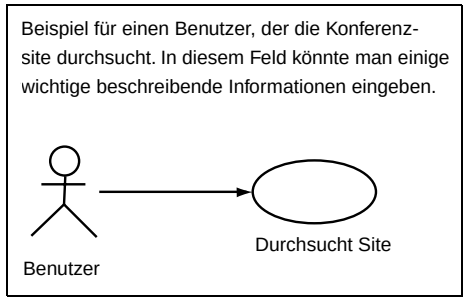

Abbildung 3.6 Einfacher Anwendungsfall für einen Benutzer, der eine Konferenzsite durchsucht

In diesem Diagramm sind drei Informationen enthalten. Es sind ein kleines Strichmännchen, ein Pfeil und ein Oval sichtbar. Das Strichmännchen wird als **Akteur** bezeichnet und repräsentiert eine Einzelperson mit einer spezifischen Rolle. Das Oval repräsentiert einen Anwendungsfall. Wenn der Zeiger (Pfeil) vom Akteur zum Anwendungsfall zeigt, wird dies als **Interaktion** bezeichnet.

Der Akteur Der Akteur ist eine Einheit, die extern zur Anwendung vorliegt und auf bestimmte Weise mit dem System interagiert. Ereignisse werden durch einen Akteur instanziiert, und der Akteur kann seiner Rolle entsprechend beschrieben werden.

Im Beispiel der Konferenzanmeldungsanwendung gehörten zu den Akteuren u. a.:

- **Benutzer:** Ein Akteur, der die Konferenzsite nach Informationen durchsucht, die ihm bei der Entscheidung helfen, an welcher Konferenz er teilnehmen möchte (oder nicht). Die Identität des Benutzers ist unbekannt.
- **Verwender:** Ein Verwender ist mehr als ein Benutzer. Der Verwender ist eine Person, die sich an der Konferenzsite angemeldet hat und vielleicht (oder auch nicht) an der Konferenz teilnimmt. Die Identität des Verwenders ist bekannt.
- **Konferenzveranstalter:** Der Konferenzveranstalter ist für den Inhalt der Konferenzwebsite verantwortlich. Der Konferenzveranstalter prüft in regelmäßigen Abständen die Datenbank, um die Konferenzstatistik anzuzeigen.

Sämtliche Akteure im Konferenzanmeldungssystem sind Personen, dies ist jedoch nicht immer der Fall. Ein Akteur kann eine anwendungsinterne oder -externe Einheit darstellen, beispielsweise eine andere Anwendung.

Der Konferenzanmeldungsanwendung wurde folgender Akteur hinzugefügt:

- System-Manager: Dieser prüft die Datenbank regelmäßig auf Ihre Datenkonsistenz.

Der System-Manager ist ein besonderer Akteur. Es handelt sich um einen Task-Manager, mit dem regelmäßig durchgeführte Aufgaben wie das Bereinigen von E-Mail-Adressen durchgeführt werden.

Warum ist ein System-Manager erforderlich? Viele Verwender geben bei der Anmeldung eine falsche E-Mail-Adresse an (in 25% der Fälle). Die Fehler waren sind besonders schwer wiegend (beispielsweise das Verwechseln von Bindestrichen und Unterstrichen), aber die Fehler verursachen E-Mail-Fehler. Der System-Manager durchsucht die Datenbank auf derartige Fehler.

In komplizierteren Fällen können mehrere Akteure an einem Anwendungsfall beteiligt sein. Der **initiierende Akteur** startet das erste Ereignis. Die weiteren Akteure werden als **teilnehmende Akteure** bezeichnet.

Der Anwendungsfall Der Anwendungsfall (Use Case) wird durch ein einfaches Oval dargestellt. Um dem Ganzen Sinn zu verleihen, muss eine Interaktion stattfinden. Ein guter Anwendungsfall beschreibt sämtliche Interaktionen zwischen Benutzer und Anwendung, es wird üblicherweise jedoch auf keine spezielle Technologie verwiesen.

Im Beispiel der Konferenzanmeldungsanwendung stellt ein Anwendungsfall die Anmeldung des Verwenders an einer Konferenz dar. Der Vorgang, bei dem der Verwender das Anmeldeformular ausfüllt, stellt hierbei keinen eigenen Anwendungsfall dar – dieser Vorgang ist Bestandteil des Anwendungsfalls »Anmeldung«. Die Granularität der Anwendungsfälle ist äußerst wichtig. Sie sollten alle der möglichen Szenarien beschreiben, diese jedoch nicht bis ins Detail ausführen.

Das Entwickeln der verschiedenen Anwendungsfälle stellt einen iterativen Prozess dar, und die Anzahl der Anwendungsfälle sollte mit der Anzahl der Iterationen steigen. In einer iterativen Lösung besteht das Ziel darin, sich bis zu einer spezifischen Lösung vorzuarbeiten.

Ein komplexerer Anwendungsfall

Das Konferenzanmeldungssystem weist neben dem Durchsuchen der Site weitere Anwendungsfälle auf. Weitere Anwendungsfälle umfassen das Verwalten der Konferenzinformationen, das Anmelden an der Site sowie das Teilnehmen an einer Konferenz. Abbildung 3.7 zeigt ein etwas komplexeres Anwendungsfalldiagramm.

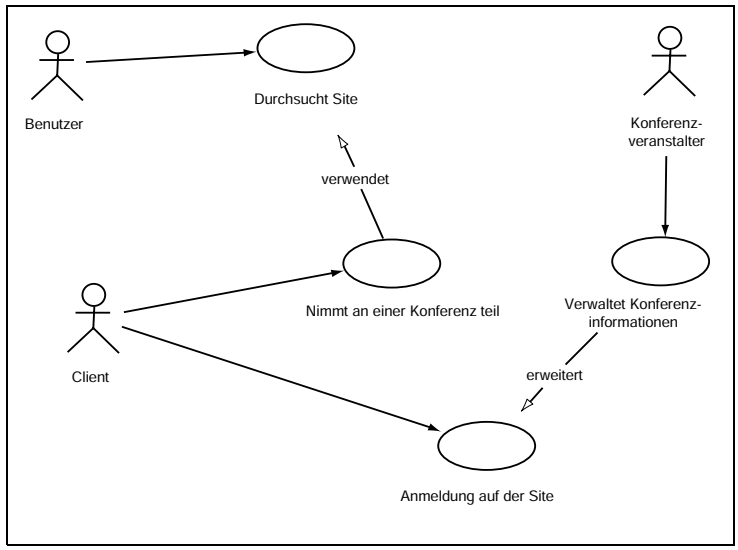

Abbildung 3.7 Anwendungsfalldiagramm zur Konferenzanmeldung

In diesem Diagramm sind vier Anwendungsfälle enthalten, das Durchsuchen der Site, das Teilnehmen an einer Konferenz, das Anmelden an der Site sowie das Verwalten der Konferenzinformationen. Im ursprünglichen Entwurf wurde kein Unterschied zwischen der Anmeldung an einer Konferenz und der Anmeldung an der Site gemacht. Beim Verfassen des Domänenmodelltextes konnte jedoch festgestellt werden, dass zwei Formen der Registrierung eingerichtet werden müssen, da die Anmeldung zu einer Konferenz einen umfangreicheren Anwendungsfall darstellt als die Siteanmeldung.

Interaktion In Abbildung 3.7 findet eine Interaktion zwischen den Akteuren und den Anwendungsfällen statt. Ein einfacher Pfeil repräsentiert die Benutzeraktion. Für eine Interaktion ist jedoch nicht zwangsläufig eine Benutzeraktion erforderlich. Bei den Akteuren könnte es sich auch um Programme auf Systemebene handeln.

Beim Konferenzanmeldungssystem werden die Datenbanken auf Stapelverarbeitungsbasis durch eine Access-Datenbank zusammengeführt. Wie wird diese Interaktion modelliert? Die einfachste Methode besteht darin, das Legacysystem zu einem Akteur zu machen. Der Legacysystemakteur wird **Systemakteur** genannt.

Anwendungsfälle und Erweiterungen Meldet sich ein Verwender für eine Konferenz an, erfordert das deutsche Recht, dass der Verwender die betreffende Website durchsucht, damit er weiß, was er kauft. Andernfalls könnte die über das Web durchgeführte Anmeldung als nicht bindend betrachtet werden. Daher muss

der Anwendungsfall »Teilnahme an einer Konferenz« die Funktionalität »Durchsuchen der Site« einschließen. Effizienter wäre es jedoch, wenn der Anwendungsfall »Teilnahme an einer Konferenz« die Funktionalität des Anwendungsfalls »Durchsuchen der Site« einschließen würde. Im Anwendungsfalldiagramm für die Konferenzanmeldung (Abbildung 3.7) wird der Anwendungsfall »Durchsuchen der Site« durch die UML-Notation eines Pfeils mit einer weißen Spitze und dem Wort **verwendet** dargestellt. So wird angezeigt, das ein Anwendungsfall die Funktionalität eines anderen Anwendungsfalls verwendet.

Eine weitere Möglichkeit zur Wiederverwendung von Code ist die Erweiterung der Funktionalität. Im Anwendungsfalldiagramm für die Konferenzanmeldung (Abbildung 3.7) muss der Konferenzveranstalter in der Lage sein, weitergehende Aufgaben für die Site durchzuführen. Aber wie der Verwender muss sich der Konferenzveranstalter an der Website anmelden, um sich selbst als Konferenzveranstalter zu identifizieren. Mit Hilfe des gleichen Pfeils, diesmal jedoch mit dem Schlüsselwort **erweitert**, wird die Funktionalität des Anwendungsfalls »Anmeldung« durch den Anwendungsfall »Konferenzveranstalter« erweitert.

Es ist möglich, über eine der beiden genannten Methoden Abhängigkeiten zwischen den verschiedenen Anwendungsfällen zu erzeugen, und so zu wieder verwendbarem Code zu gelangen.

3.2.5 Definieren der Modelldetails

Nachdem Kontext, Domänenmodelltext und Anwendungsfälle definiert sind, sollten Sie sich einen Überblick verschafft haben, was mit der Anwendung erreicht werden soll. Bis zu diesem Punkt gestaltet sich das Verständnis der Anwendung jedoch sehr abstrakt.

Das Spezifizieren der Details hängt von den zuvor durchgeführten Operationen ab. Wenn beispielsweise Anwendungsfälle, Domänenmodell oder Kontext falsch sind, sind die darauf basierenden Details ebenfalls falsch. Dies bedeutet, dass vor der Definition der Modelldetails sichergestellt werden muss, dass Kontext, Domänenmodelltext und Anwendungsfälle richtig sind. Sie sollten diese Phase nicht überspringen.

Es gibt vier Möglichkeiten, den Anwendungsentwurf weiter zu verfeinern:

- **Sequenzdiagramm:** Die Definition eines Anwendungsfalls in Form einer Schrittfolge
- **Kollaborationsdiagramm:** Ein dem Sequenzdiagramm ähnliches Diagramm, in diesem können jedoch auch Verknüpfungen zwischen Objekten definiert werden

- **Statusdiagramm:** Ein Diagramm, mit dem die verschiedenen Statuszustände eines Objekts dargestellt werden, die ein Objekt in verschiedenen Anwendungsfällen aufweisen kann
- **Klassendiagramm:** Ein Diagramm, in dem die Klassen der Anwendung und deren Beziehungen untereinander dargestellt werden

Bei diesen Diagrammen ist es wichtig, sich auf einige wenige Elemente zu beschränken. Obwohl es vier verschiedene Modelle gibt, ist es nicht nötig, für jedes ein Entwurfsmodell zu implementieren. Halten Sie die Anzahl der Modelle den Anforderungen entsprechend gering. Das Erstellen zu vieler Modell kompliziert den iterativen Prozess und führt zu einer Codeaufblähung.

Ein Klassendiagramm ist erforderlich, da mit diesem das Gesamtobjektmodell der Architektur definiert wird. Dieser Aspekt des Entwurfsprozesses wird in Kapitel 9 behandelt, »Implementieren von COM-Objekten«. Das Statusdiagramm ist in den meisten Systemen nicht erforderlich. Bei der Verwendung von Transaktionen kann es jedoch hilfreich sein. Sequenz- oder Kollaborationsdiagramme (siehe nächster Abschnitt) sind für den Entwurfsprozess ebenfalls nützlich. Es ist nicht notwendig, beide Diagrammtypen einzusetzen, da Sie auf verschiedene Weise die gleiche Funktion erfüllen.

Sequenzdiagramme

Durch das Sequenzdiagramm werden die Schritte definiert, die für einen Anwendungsfall erforderlich sind. Es können mehrere Sequenzdiagramme mit einem Anwendungsfall verknüpft werden. Abbildung 3.8 zeigt das Sequenzdiagramm für den Anwendungsfall »Teilnahme an einer Konferenz«.

In dieser Abbildung befinden sich im oberen Diagrammbereich ein Akteur und vier Felder, die Objekte. Unter jedem Objekt befinden sich verschiedene vertikale Linien. Beginnend beim Akteur »Verwender« ist eine horizontaler Pfeil eingezeichnet, der auf das zweite Objekt von links zeigt. Dies verdeutlicht das Senden einer Meldung vom Verwender zum **Account**-Objekt. Bei der gesendeten Meldung handelt es sich um die Anmeldung.

Von der Anmeldungsnachricht gehen zwei schmale, vertikale Felder ab. Diese Felder definieren den Steuerungsfokus. Wird ein Methodenaufruf durch einen anderen Methodenaufruf aufgerufen, wechselt der Steuerungsfokus vom ersten zum zweiten Methodenaufruf. In einem Sequenzdiagramm geschieht dasselbe. Innerhalb jedes Feldes im Diagramm verbleibt der Steuerungsfokus bei der Operation. Wenn ein Pfeil auf ein anderes Feld oder auf sich selbst zeigt, wechselt der Steuerungsfokus auf das Element, auf das der Pfeil zeigt.

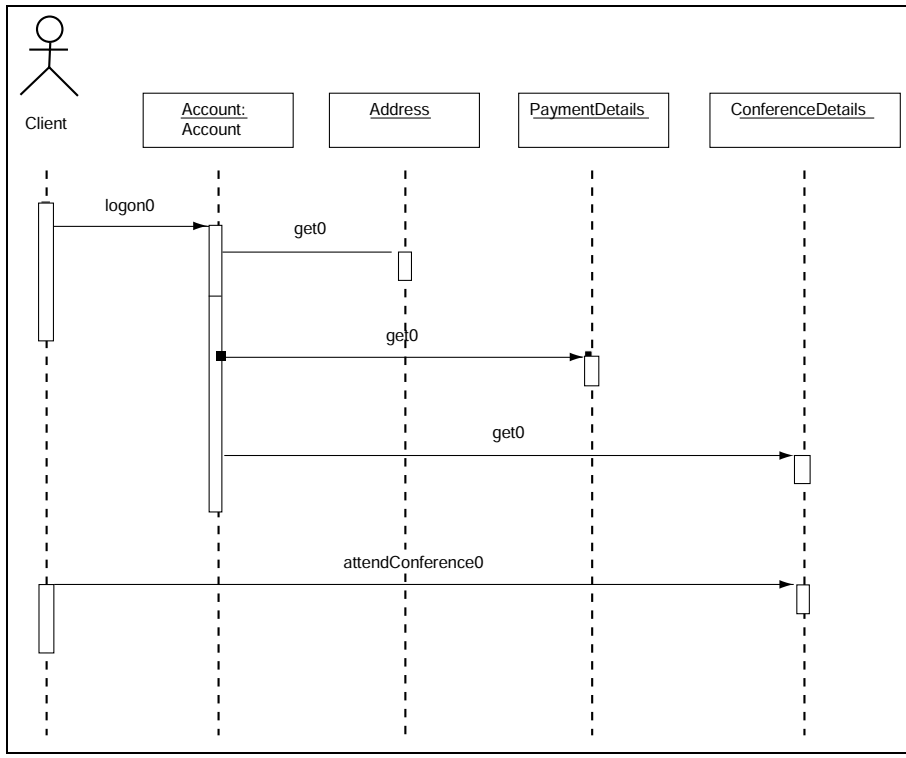

Abbildung 3.8 Sequenzdiagramm für den Anwendungsfall »Teilnahme an einer Konferenz«

Die Nachricht zur Anmeldung ist auf der rechten Seite mit einem Steuerungsfokus verbunden, der drei **Get**-Meldungen umfasst. Dies weist darauf hin, dass die Nachricht zur Anmeldung so lange warten muss, bis drei **Get**-Meldungen durchgeführt wurden. In der Programmierungsterminologie gleicht dies dem Aufruf einer Funktion, durch die eine weitere Funktion aufgerufen wird.

Meldungen in Sequenzdiagrammen werden üblicherweise als einfach betrachtet. Die Mehrzahl der UML-Tools ermöglicht die Definition anderer Meldungstypen, beispielsweise asynchrone Meldungen, Blockierung mit Zeitüberschreitung oder sogar periodische Meldungen.

Nachdem die Anmeldung abgeschlossen ist, wird die Meldung **attend-Conference( )** gesendet. Auf diese Weise wird der Verwender am **ConferenceDetails**-Objekt angemeldet.

Im Sequenzdiagramm verläuft der Meldungsfluss von links nach rechts. Es ist auch möglich, einen Meldungsaufruf von rechts nach links zu definieren, dies würde ei-

nen Rückruf anzeigen. Wie in Abbildung 3.9 gezeigt wird, ist es einem Objekt darüber hinaus möglich, ein Objekt an sich selbst zu senden.

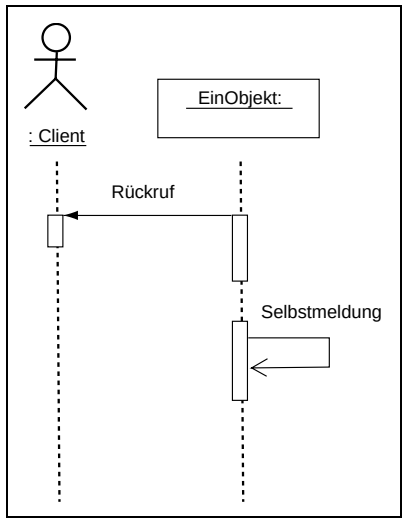

Abbildung 3.9 Sequenzdiagramm mit Rückruf und Selbstmeldung (Self-Message)

Beim Lesen eines Sequenzdiagramms sollten Sie den Steuerungsfokus betrachten, um zu sehen, welche Operationen in einem Kontext ausgeführt werden. Beachten Sie außerdem, dass die Aufrufsequenz oben im Diagramm beginnt und nach unten verläuft.

Kollaborationsdiagramm

Wie das Sequenzdiagramm zeigt auch das Kollaborationsdiagramm eine Schrittfolge. Es verfügt jedoch über die zusätzliche Fähigkeit, dass mit Hilfe von Objektverknüpfungen Objektinteraktionen dargestellt werden können. Der Anwendungsfall »Teilnahme an einer Konferenz« wird in Abbildung 3.10 als Kollaborationsdiagramm dargestellt.

In Abbildung 3.10 sind die verschiedenen Objekte (Akteure und Felder) über das Diagramm verstreut. Die Linien, mit denen die Objekte miteinander verbunden werden, sind Verknüpfungen und stellen eine programmatische Form der Verknüpfung zweier Objekte dar. Die Verknüpfung kann global, lokal, ein Parameter oder ein einfaches Feld sein.

Wie beim Sequenzdiagramm deuten die Pfeile auf Meldungen hin, die von einem Objekt zu einem anderen gesendet werden. Um jedoch die Reihenfolge zu verstehen, in der die Schritte ausgeführt werden, ist ein numerischer Bezeichner mit der Meldung verknüpft.

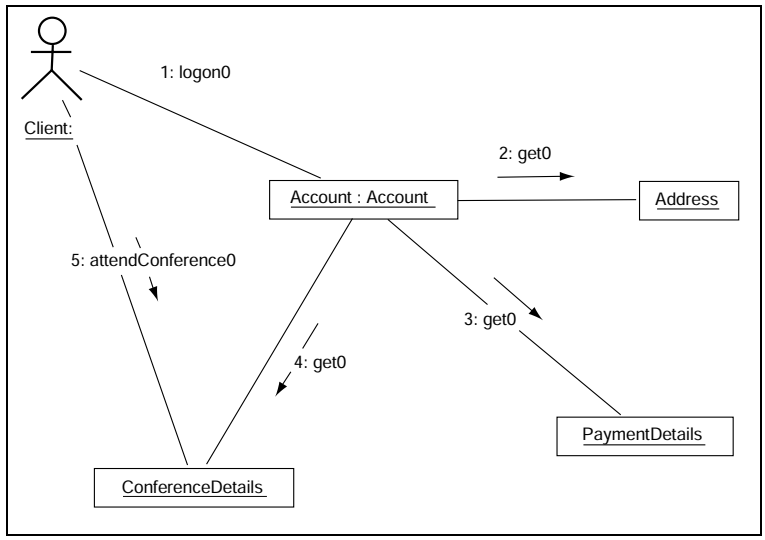

Abbildung 3.10 Kollaborationsdiagramm zum Anwendungsfall »Teilnahme an einer Konferenz«

Sequenzdiagramme ähneln den Kollaborationsdiagrammen sehr stark. In einem Sequenzdiagramm ist die Anordnung der Operationsschritte übersichtlicher, da die Schritte geordnet dargestellt werden. In einem Kollaborationsdiagramm können die Objekte nach Belieben angeordnet werden. Schlussendlich liegt die Auswahl des zu verwendenden Diagramms bei Ihnen.

3.3 Resümee

In diesem Kapitel wurde Ihnen ein grundlegendes Verständnis zur Definition einer Anwendung vermittelt. Durch das bloße Erstellen einiger Klassen und das willkürliche Zusammenfügen dieser Klassen wird noch keine Anwendung definiert. Stattdessen muss der Kontext einer Anwendung erarbeitet und ein Domänenmodelltext verfasst werden, mit dem die Anwendung definiert wird. Anschließend können Anwendungsfälle und Sequenz- oder Kollaborationsdiagramme erstellt werden, die dem Verständnis der Anwendung dienen und Ihnen dabei helfen, die Auswirkungen von möglicherweise erforderlichen Änderungen einzuschätzen.

Wozu all diese Schritte? Es mag so erscheinen, als wolle ich Ihnen zusätzliche Arbeit auferlegen oder eine Strukturierung um der Strukturierung willen durchführen. Meine Erfahrung hat jedoch gezeigt, dass Projekte mit umfassender vorheriger Planung sich als die besseren Projekte herausstellen. Geht diese Planungsphase mit einem iterativen Entwicklungsansatz einher, können die verschiedenen

Faktoren optimal aufeinander abgestimmt werden, und es ist die Optimierung, auf die es bei einer Anwendung ankommt.

Jeder kann eine Anwendung entwickeln. Wenn Sie jedoch eine stabile, skalierbare und langfristig einsetzbare Anwendung entwickeln möchten, erfordert dies einige Vorarbeit. Software und Technologie ändern sich, und obwohl eine Planung erforderlich ist, kann nichts »überplant« werden, denn niemand kann in die Zukunft sehen.

4 Entwickeln eines Prototyps

Der Anwendungsentwicklungszyklus (Application Development Cycle, ADC) erfordert nach Fertigstellung des Entwurfs das Entwickeln eines Prototyps. Zur Prototypentwicklung ist das Schreiben von Code erforderlich. Zum Verständnis der Entwicklung eines Prototyps muss zunächst das Konzept der Komponenten erläutert werden. Komponenten stellen wieder verwendbare Module dar, die dem Programmierer als Black Box erscheinen. Ein Hauptmerkmal der Komponenten besteht darin, die Schnittstelle von der Implementierung zu trennen.

4.1 Komponenten

Im Zusammenhang mit Komponenten denkt man üblicherweise an COM (Component Object Model), CORBA (Common Object Request Broker Architecture) oder Java Beans. Diese Beschreibung von Komponenten schließt jedoch den direkten Verweis auf eine spezifische Technologie ein. Eine angemessenere Beschreibung von Komponenten lautet, dass es sich um eine generische Entwicklungsstrategie handelt. Diese Beschreibung ist treffender, da Komponenten eine Möglichkeit darstellen, logische Abschnitte zu kapseln und anschließend mit einem bestimmten Protokolltyp offen zu legen. Dies bedeutet, dass Komponenten den im vorherigen Kapitel erwähnten Inseln entsprechen.

4.1.1 Ein Stück Komponentengeschichte

Zum Verständnis der Komponenten kann ein Vergleich mit der objektorientierten Entwicklung angestellt werden. Eine objektorientierte Sprache, beispielsweise C++ oder Smalltalk, ist eine Sprache, die auf Klassen basiert. Wenn Sie mit Hilfe einer dieser Sprachen eine Anwendung entwickeln, erstellen Sie ein Objekt, um ein spezifisches Problem zu lösen.

Das klassische Beispiel hierfür ist die **Shape**-Klasse. Mit dieser werden generische Operatoren zum Zeichnen und Beschreiben der Form (Shape) offen gelegt. Wenn Sie anschließend eine bestimmte **Shape**-Klasse, z. B. ein Rechteck, erstellen möchten, müssen Sie eine neue Klasse erstellen und diese von **Shape** ableiten. Die neue Rechteckklasse weist in diesem Fall zusätzliche Methoden für die zusätzliche Funktionalität auf. Wenn die Rechteckklasse jedoch generisch verwendet wird, kann nur die **Shape**-Klasse eingesetzt werden.

Komponenten unterscheiden sich hiervon dadurch, dass sie mit Schnittstellen und Implementierungen arbeiten. Bei den Komponenten sind die Schnittstellen generisch, die Implementierung ist spezifisch. Eine ausführlichere Erläuterung

wird in den folgenden Abschnitten bereitgestellt, zunächst müssen jedoch noch einige Konzepte der objektorientierten Entwicklung definiert werden.

Kapselung

Die Grundidee bei der Kapselung besteht darin, Codeabschnitte mit einer bestimmten Funktion zu verbergen, sodass nur über einen Methodenaufruf auf diesen zugegriffen werden kann. Hierbei handelt es sich um einen Black Box-Ansatz, bei dem der Benutzer die Implementierungsdetails nicht kennen muss. Der Benutzer muss lediglich wissen, wie der Methodenaufruf funktioniert.

Das einfachste Beispiel der objektorientierten Entwicklung ist die **foo**-Klasse, die zur Ausgabe »Hello world« führt:

```
class foo {
public:
   void sayHello() {
      printf("Hello world");
   }
};
```

In diesem Beispiel weist die Klasse **foo** die Methode **sayHello** auf, mit der die C-Bibliotheksfunktion **printf** aufgerufen wird. Die Funktion **printf** sendet den Text "Hello world" an die Standardausgabe.

Die Methode **sayHello** ist öffentlich und kann durch jeden Benutzer aufgerufen werden, der diese Klasse instanziiert:

```
int main() {
   foo ex;
   ex.sayHello();
}
```

Der Konsument, der so genannte Client, muss nicht wissen, wie die Begrüßung implementiert wird. Der Programmierer muss lediglich wissen, dass mit der Methode **sayHello** eine Begrüßung ausgegeben wird. In dieser Version von **sayHello** wird über die Methode ein statischerText ausgegeben.

Erweitern wir dieses Beispiel und definieren die Methode **setRepitition**, mit der festgelegt wird, wie häufig die Begrüßung **sayHello** erfolgt. Die erweiterte **foo**-Klasse lautet folgendermaßen:

```
class foo {
private:
   long m_repetition;
public:
   foo() { m_repetition = 1L; }
   void setRepetition( long value) {
      m_repetition = value;
}
void sayHello() {
      for( int c1 = 0; c1 < m_repetition; c1 ++) {
         printf("Hello world\n");
      }
   }
};
```

Die **foo**-Klase wurde von Version 1 nach Version 2 lediglich überarbeitet. Auch wenn die Benutzer von Version 1 keine Wiederholung einstellen möchten, funktioniert der Code weiterhin – der Constructor von **foo** setzt **m_repitition** auf den Wert 1. So wird sichergestellt, dass die Benutzer, die **foo::sayHello** aufrufen, nicht von der Änderung betroffen sind. Benutzer der zweiten Version von **foo** besitzen die Fähigkeit, den Wert für die Wiederholung einzustellen. Auch hier muss der Benutzer nicht wissen, wie die Wiederholung implementiert wird. Der Benutzer kennt lediglich den Unterschied zwischen den beiden Methodenaufrufen.

Vererbung und Polymorphismus

Die Mehrzahl der objektorientierten Sprachen weist die Fähigkeit zur Vererbungsimplementierung auf. Bei der Vererbung können neue Klassen basierend auf bereits vorhandenen Klassen erstellt werden. Die Vererbung bietet die Fähigkeit, die Funktionalität einer Klasse zu ändern, die vorhandene Basisfunktionalität jedoch beizubehalten. Im Fall der **foo**-Klasse könnte beispielsweise die Begrüßung auch in anderen Sprachen erfolgen, z.B. in französisch. Die neue Klasse würde folgendermaßen lauten:

```
class frenchFoo : public foo{
public:
   void sayHello() {
      printf("Salut tout le monde");
   }
};
```

Die Klasse **frenchFoo** wurde mit Hilfe der C++-Notation in der ersten Codezeile von der Klasse **foo** abgeleitet. Beachten Sie, dass **frenchFoo** die Methode **sayHello** aufweist, deren Methodensignatur mit der von **foo::sayHello** identisch ist. Dies bedeutet, dass bei der Instanziierung einer Klasse von **frenchFoo** alle Aufrufe für **sayHello** in **frenchFoo::sayHello** geändert werden. Sehen Sie sich den folgenden Quellcode an:

```
int main() {
foo ex;
  frenchFoo fex;
fex.sayHello();
   ex.sayHello();
}
```

Dieses Beispiel ergibt folgende Ausgabe:

```
Salut tout le monde
Hello world
```

Bei Instanziierung der Klasse **foo** führt das gleiche **sayHello** zu der Begrüßung, die wir im vorherigen Abschnitt gesehen haben.

Die Vererbung befähigt zur Erstellung von Klassen, die sich ähneln. Wenn beispielsweise die Klasse **frenchFoo** an ein Objekt übergeben wird, das den Typ **foo** erwartet, erzwingt der Compiler einen Typecast. Dies bedeutet, dass Benutzer der Klasse **foo** denken, dass sie über **foo** verfügen, tatsächlich wird jedoch **frenchFoo** verwendet. Als Ergebnis werden Benutzer, die über die **foo**-Klasse **sayHello** aufrufen, an **frenchFoo::sayHello** umgeleitet.

Das Vorliegen einer Klasse, die einer anderen Klasse ähnelt, wird **Polymorphismus** genannt. Wenn Sie beispielsweise eine generische Begrüßung erstellen möchten, kann eine **b**-Klasse als Basis verwendet werden. Von dieser Klasse ausgehend, würde eine beliebige Klasse zur Ausgabe einer Begrüßung die **sayHello**-Methode ableiten und implementieren. Verschiedene Sprachen verfügen über Techniken, mit denen Sie gezwungen werden, spezifische Methoden zu implementieren, wenn diese von einer Klasse abgeleitet werden. In C++ wird diese Technik als **rein virtuell** bezeichnet.

Die Probleme der objektorientierten Entwicklung

Die objektorientierte Entwicklung ist ein Fortschritt für den Entwicklungsprozess. Hierbei treten jedoch auch Probleme auf:

- **Verwendung einer einzelnen Sprache erforderlich:** Der vorangegangene Code wurde in C++ geschrieben, und dieser Code kann nicht mit Visual Basic ver-

mischt werden. Der Compiler versteht nur eine Sprache, das Verwenden zweier oder mehrerer Sprachen führt zu Kompilierungsfehlern.

- **Vollständige Klassenbeschreibungen erforderlich:** Verwendet man eine Sprache wie C++, muss die **foo**-Klasse gegenüber dem Compiler vollständig beschrieben werden. Andernfalls weiß der Compiler nicht, welche Funktion das Objekt besitzt oder wie das Objekt arbeitet. Müssen Vererbung und Polymorphismus aufgelöst werden, muss der Compiler die Definitionen aller involvierten Klassen kennen, um das Layout der Klasse zu bestimmen oder zu definieren. Dies ist problematisch, denn wenn Sie binären Code gemeinsam verwenden möchten, müssen Sie zusätzliche Informationen zu einer Klasse bereitstellen. Und da nicht alle Compiler gleich sind, muss vielleicht der gesamte Quellcode freigegeben werden.
- **Entwurf nicht einfach:** Das Entwerfen einer guten objektorientierten Anwendung ist nicht einfach. Da der objektorientierte Entwurf komplexer ist und große Anwendungen, die durch mehrere Personen geschrieben wurden, häufig verschiedene Programmierstile aufweisen, gestalten sich Integration, Test und Wartung schwieriger.

4.1.2 Komponenten – Das Entwurfskonzept

Bei der Entwicklung eines Computers wird die Hauptplatine mit einer Reihe von Chips ausgestattet. Diese Chips werden verdrahtet und mit Hilfe von Elektrizität funktionstüchtig gemacht. Die Entwickler einer solchen Hauptplatine kümmern sich nicht um die internen Details dieser Computerchips. Sie müssen lediglich die externen Verbindungen und deren Funktion kennen. Der Chip ist für die Entwickler der Hauptplatine eine Blac kBox.

Die Komponentenentwicklung folgt der gleichen Vorstellung. Die Komponenten stellen Teile eines Programmcodes dar, dessen Funktionen mit Hilfe einer Art Signatur offen gelegt wird. Die Schnittstellen, Methoden innerhalb der Schnittstellen, und Parameter innerhalb der Methode definieren eine Signatur. Unter Zuhilfenahme des Inselkonzeptes, das im vorangegangenen Kapitel besprochen wurde, stellen die Inseln individuelle Komponenten dar.

Es soll versucht werden, die Begriffe objektorientierte Programmierung oder Modulprogrammierung zu vermeiden. Dies geschieht aus einem wichtigen Grund. Die Programmierung unterlag in den letzten zwanzig Jahren erheblichen Änderungen. Vor zwanzig Jahren wurden BASIC (Beginner's All-Purpose Symbolic Instruction Code), COBOL (Common Business-Oriented Language) und FORTRAN (Formula Translation) als Nonplusultra betrachtet. Heute gibt es CORBA, COM, C++, Java usw. Das Beschränken der Komponenten auf eine Technologie ist kein guter Ansatz, da so auch das eigentliche Konzept eingeschränkt wird.

Im Laufe der Jahre war die alles bestimmende Idee die Entwicklung von wieder verwendbarer Software. Codeabschnitte mit dieser Funktionalität wurden zunächst als Bibliotheken, dann als Module und schließlich als Komponenten bezeichnet. Ziel all dieser Techniken war es, die Funktionalität in einer Form zu »verpacken«, dass die Details des Pakets in einer Black Box verborgen wurden. Der Benutzer des Pakets brauchte lediglich die Schnittstelle zu verstehen, bei der es sich um Strukturen, Methoden oder Objekte handeln konnte.

Bei den Komponenten ist das Ziel dasselbe, das Konzept von Schnittstelle und Implementierung ist jedoch sehr explizit. Eine Komponente erfordert eine Schnittstelle. Dem Konsumenten gegenüber stellt die Komponente einen statischen Block dar. Die Implementierung ist weder direkt zugänglich, noch kann der Konsument diese bearbeiten. Durch das Verwenden von Schnittstellen arbeitet der Konsument eine Abstraktion ein, die später zur Definition neuer Funktionalität ohne Änderungen für den Konsumenten verwendet werden kann.

Komponenten lassen sich am besten wie folgt zusammenfassen:

> *Eine Softwarekomponente ist eine Kompositionseinheit, die nur aus vertraglich spezifizierten Schnittstellen und expliziten Kontextabhängigkeiten besteht. Eine Softwarekomponente kann unabhängig bereitgestellt werden und unterliegt der Gestaltung von Drittanbietern.*

Diese Definition wurde zuerst auf der ECOOP '96 formuliert (European Conference on Object-Oriented Programming).

Middleware und Komponenten

Möchten ein Konsument und ein Server kommunizieren, ist die Überbrückung technologischer Unterschiede erforderlich. Diese Brücken (Bridges) werden als **Middleware** bezeichnet. Die Middleware ähnelt einem Übersetzungsdienst, über den zwei Personen unterschiedlicher Sprache miteinander kommunizieren können. COM ist ein Beispiel für die Middleware. COM ermöglicht einem Visual Basic-Objekt die Kommunikation mit einem Visual C++-Objekt.

Die COM-Middleware ist wie in Abbildung 4.1 aufgebaut.

In der Architektur der Middleware sind Konsument und Server zwei separate Einheiten, auf der abstrakten Ebene scheinen diese jedoch direkt miteinander verbunden zu sein. Über die Middleware wird dem Konsumenten ein Ghost des Servers und dem Provider ein Ghost des Benutzers zur Verfügung gestellt. Dieser Ghostingeffekt vermittelt jeder Partei den Eindruck, sie verfüge über eine direkte Verbindung. Tatsächlich findet jedoch bei der Kommunikation zwischen Konsument und Server eine Kommunikation zwischen Konsument und Proxy statt, wo-

bei der Proxy die Informationen sammelt und in einer Art Paket zusammenfasst. COM verwaltet das Paket und sendet es an den Stub. Anschließend wird das Paket durch den Stub entpackt, und es wird ein Aufruf an den Server ausgeführt.

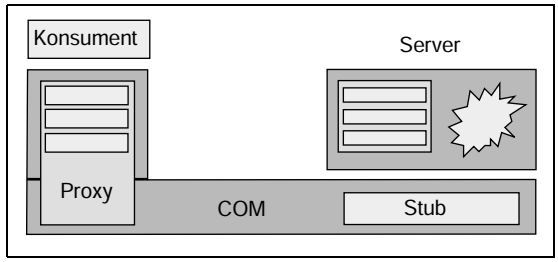

Abbildung 4.1 Die Architektur der COM-Middleware

Der Vorteil der Verwendung von Middleware besteht darin, dass Konsument und Server nichts voneinander wissen müssen. Der Konsument muss lediglich die Serverschnittstelle kennen.

Auswirkungen von Standards auf Komponenten

Der Grund für die Einrichtung von Standards ist die Vereinfachung des Informationsaustausches zwischen unterschiedlichen Parteien. Standards sind ein notwendiger Bestandteil unseres täglichen Lebens – ohne Standards würde die Welt im Chaos versinken. Die Softwareindustrie hat es jedoch wiederholt versäumt, derartige Standards einzusetzen. So sind einige Missverständnisse entstanden, die es zu beseitigen gilt:

- **»Das Etablieren von Standards erfolgt nur langsam.«**
  Dies ist nicht länger richtig. Das ETSI (European Telecommunications Standards Institute) räumt ein, dass es in der Vergangenheit oft Jahre dauerte, einen Standard zu etablieren. Mit Einführung des Webs können Standards jedoch in weniger als einem Jahr definiert werden.
- **»Ein Standard zwingt Produkte und Funktionen auf den kleinsten gemeinsamen Nenner.«**
  Dies ist keineswegs richtig, zu sehen beispielsweise am Internet. Die Protokolle des Internets stellen einen Standard dar und werden von Millionen von Menschen eingesetzt. Es wurden unter Verwendung dieser Standards viele innovative Produkte entwickelt. Die tatsächliche Innovation liegt darin, die Standards den eigenen Anforderungen gemäß anzupassen.

Obwohl das vorliegende Buch ein an Microsoft orientiertes Buch ist, in dem spezifische Microsoft-Technologien zum Einsatz kommen, können Sie ohne weiteres universelle Standards zum Erstellen von Anwendungen einsetzen. In diesem Buch

werden nach Möglichkeit derartige Standards verwendet, da diese Ihnen die Option zur Verwendung mehrerer Tools, Betriebssysteme und Umgebungen geben.

Wie sieht es bei COM mit Standards aus? Microsoft entwickelte COM, aber im Laufe der Zeit wurde COM auf verschiedenen Plattformen verfügbar. Die Mozilla-Gruppe (offener Netscape-Browser) unternimmt sogar den Versuch, eine offene, plattformübergreifende COM-Version mit Namen XPCOM zu entwickeln. Dies bedeutet, dass die COM-Technologie auch in den nächsten Jahren noch zum Einsatz kommen wird.

4.1.3 Schnittstellen und Implementierungen

Nachfolgend soll die Funktionsweise von Schnittstellen und Implementierungen eingehender betrachtet werden. Die Schnittstelle definiert ein zu erreichendes Ziel; mit der Implementierung wird dieses angestrebte Ziel umgesetzt. Der Konsument interagiert ausschließlich mit der Schnittstelle. Werden ein Konsument und ein Server zusammengebracht, bilden sie einen »Vertrag«, der nicht durch eine der beiden Parteien gebrochen werden darf.

Der Vorteil des Schnittstellen/Implementierungs-Ansatzes besteht darin, das **Bridge-Muster** umzusetzen. Ein Bridge-Muster ist ein komponentenartiges Muster. Durch Verwenden des Bridge-Musters kann der Server seine Implementierung ändern, ohne dass eine Änderung oder Neukompilierung des Konsumentencodes erforderlich wird.

Die Implementierung kann außerdem mehrere Schnittstellen unterstützen. Muss beispielsweise eine Schnittstelle geändert werden, kann eine neue Schnittstelle erstellt werden. Der neue Konsument verwendet diese Schnittstellte, die alte Schnittstelle bleibt jedoch zur Unterstützung der alten Konsumenten weiterhin verfügbar.

Granularität

Mit den Komponenten soll das Inselkonzept umgesetzt werden, das im vorherigen Kapitel definiert wurde. Die **Granularität** definiert die in einer Komponente implementierte Codemenge. Ist die Granularität niedrig, ist die Menge des enthaltenen Code ebenfalls niedrig. Liegt die Granularität hoch, ist der Umfang der in der Komponente enthaltenen Codemenge ebenfalls hoch.

Die Granularität spielt bei der Komponentenentwicklung eine wichtige Rolle. Betrachten Sie die Abbildungen 4.2 und 4.3.

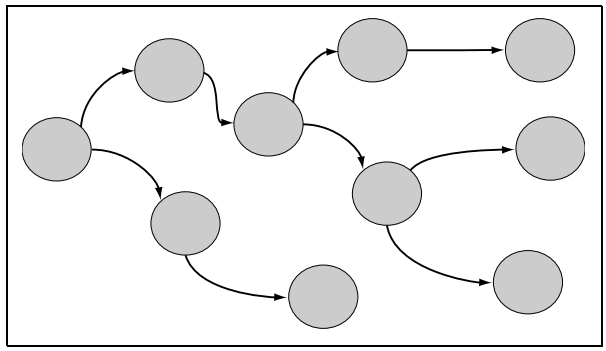

Abbildung 4.2 Ein komplexeres Inseldiagramm mit vielen unterschiedlichen Verträgen

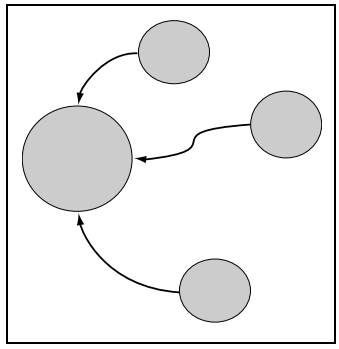

Abbildung 4.3 Ein einfacheres Inseldiagramm mit größeren Inseln, aber weniger Verträgen

Die Pfeile in den Abbildungen stellen die Verträge zwischen Konsument und Server dar. Aus diesen Diagrammen können zwei Beziehungen abgeleitet werden. Je höher die Granularität, desto weniger Verträge. Und je höher die Granularität, desto mehr Abhängigkeit von einer spezifischen Komponente.

Abbildung 4.2 weist viele Komponenten, aber nur wenige Verträge auf, die mit einer spezifischen Komponente verknüpft sind. In diesem System mit geringer Granularität hat eine Änderung einer einzelnen Komponente keine besonders große Auswirkung auf die Anwendung. Da jedoch mehr Verträge vorhanden sind, kann die Verwaltung dieser Verträge schwieriger sein. Die Probleme wachsen mit der Anzahl der Versionen und Schnittstellen. Dies wiederum wirkt sich auf die Anwendungsstabilität aus.

Abbildung 4.3 weist weniger Verträge zwischen den Komponenten auf, diese sind jedoch auch wichtiger. Es ist eine sehr große Komponente vorhanden, alle weiteren Komponenten besitzen Verträge mit dieser Komponente. Dies bedeutet, dass

die umfangreiche Komponente nicht ohne weiteres geändert werden kann, da von ihrer Funktionalität viele Konsumenten abhängen. Diese Komponente weist eine monolithische Architektur auf. Dennoch kann eine Anwendung so stabiler werden, da eine Versionsänderung ein umfangreicheres direktes Testen erfordert.

Welche stellt die beste Lösung dar? Sehen Sie sich Abbildung4.4 an.

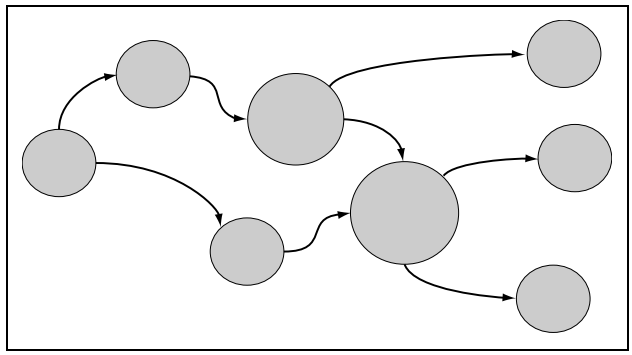

Abbildung 4.4 Eine Komponentenarchitektur mit Hybridgranularität

Bei der in Abbildung 4.4 gezeigten Architektur liegt die Komponentengranularität niedrig, es sind jedoch einige Schlüsselkomponenten vorhanden. Diese Schlüsselkomponenten enthalten mehr Funktionalität (in der Abbildung als große Kreise dargestellt). Der Grund für diesen Ansatz liegt in der Verringerung von Verständnisproblemen. Viele Verträge sind aus Sicht des Computers nicht negativ. Aus der Sicht der Person, die nur eine bestimmte Informationsmenge aufnehmen kann, ist das Verwenden vieler Verträge dagegen sehr wohl negativ. Wenn die Verträge in einer Hierarchie angeordnet werden, kann der Konsument jeden der vollzogenen Schritte verstehen.

Eine weitere Möglichkeit zur Verringerung der Komplexität stellt das Erstellen spezifischer Komponenten dar, die als Vermittler dienen. Diese Komponenten verbergen eine Gruppe von Komponenten, durch die eine bestimmte Funktionalität bereitgestellt wird. Auf diese Weise muss der Konsument nur die Vermittler verstehen, nicht jedoch die gesamte Architektur. Ändert sich die Technologie, kann durch einen Vermittler diese Änderung besser verborgen werden als durch eine einzelne Komponente.

Entwicklung in Schichten

Üblicherweise befinden sich in einer Anwendung mehrere Komponenten, und einige Komponenten rufen weitere Komponenten auf. In einem entstehenden System erhöht sich die Anzahl der Verträge, und es ist wichtig, diese Verträge und die

Komponenten zu strukturieren. Eine Möglichkeit der Strukturierung stellt eine Schichtenarchitektur dar. In einer Schichtenarchitektur weisen Komponenten ausschließlich einseitige Abhängigkeiten auf.

Eine einseitige Abhängigkeit tritt beispielsweise auf, wenn Komponente A von Komponente B abhängt, die wiederum von Komponente C abhängt; es bestehen jedoch keine weiteren Abhängigkeiten zwischen diesen Komponenten.

Gegenseitige Abhängigkeiten zwischen Komponenten In einer Schichtenarchitektur werden gegenseitige Abhängigkeiten (siehe Abbildung 4.5) nach Möglichkeit vermieden.

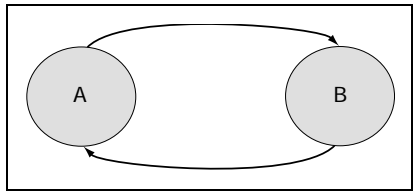

Abbildung 4.5 Beispiel einer gegenseitigen Abhängigkeit zweier Komponenten

Das Problem bei gegenseitigen Abhängigkeiten besteht darin, dass diese nicht leicht kompiliert oder getestet werden können, da Komponente A Komponente B als Bestandteil ihres Zustands erfordert. In einer sequenziellen Kompilierung muss Komponente B zunächst definiert werden. Dies macht eine Kompilierung nahezu unmöglich. Viele Sprachen lösen dieses Problem durch vorwärtsgerichtete Definitionen.

Das Testproblem gestaltet sich komplizierter. Nehmen Sie an, es soll sowohl Komponente A als auch Komponente B getestet werden, beginnend mit Komponente A. Da die Implementierung auf Komponente B beruht, können die Ergebnisse nicht als richtig betrachtet werden, denn Komponente B wurde noch nicht getestet und als richtig eingestuft.

In einigen Fällen ist jedoch eine Vermeidung von gegenseitigen Abhängigkeiten nicht möglich. Dann muss ein Verfahren zum Testen der gegenseitigen Abhängigkeiten vorhanden sein. Beim Testen von Komponente A kann Komponente B durch eine künstliche Implementierung ersetzt werden. Über diese künstliche Komponente B werden die erforderlichen Entwurfsbeschränkungen implementiert. Unabhängig davon, wie die Implementierung von Komponente A Komponente B aufruft, wird immer ein richtiges Ergebnis erzielt.

Dieser Ansatz kann unnötig sein, wenn die Implementierungen von Komponente A und B einfach sind. Sehr wichtig ist dieses Vorgehen, wenn die Implementierun-

gen von Komponente A und B weitere Komponenten einbeziehen, was zu einer komplexeren Implementierung führt.

Gegenseitige Abhängigkeiten müssen nicht, wie hier beschrieben, direkt sein. Sie könnten mehr als das Entfernen einer Komponente umfassen. Das Hauptproblem liegt darin, dass eine Komponente von einer anderen Komponente abhängt, die wiederum von der ursprünglichen Komponente abhängt.

Die verschiedenen Möglichkeiten bei der Schichtung Die Entwicklung einer Schichtenarchitektur zielt auf den Erhalt einer relativ strikt abgegrenzten Schichtenarchitektur ab. Das in Abbildung 4.6 gezeigte Schichtendiagramm besteht aus einem Satz Kernfunktionen, die durch die rechts dargestellten Komponenten definiert werden. Durch diese Komponenten wird die grundlegende Funktionalität des Systems bereitgestellt. In der nächsten Schicht, die sich links neben den Kernkomponenten befindet, wird eine weitere Funktionalitätsebene definiert. Links neben diesen Komponenten wiederum befindet sich eine weitere Schicht Komponenten. Die Verträge zwischen Konsument und Server verlaufen in eine Richtung von links nach rechts. Die Komponenten überspringen beim Aufruf einer weiteren Schicht keine der Schichten. Die Komponenten in der dritten Spalte von rechts beispielsweise rufen die Kernkomponenten nicht direkt auf. Dies ist ein Beispiel für eine strikte Komponentenschichtung.

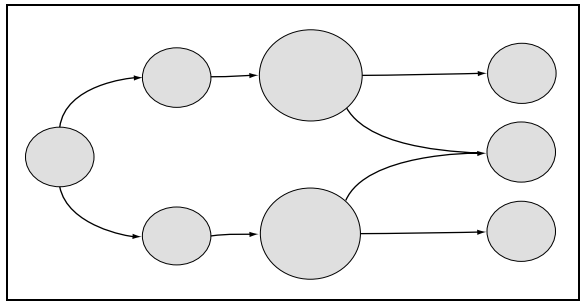

Abbildung 4.6 Beispiel einer Schichten- und einer Zwiebelarchitektur

Abbildung 4.7 zeigt einen nicht strikten Schichtenansatz. In diesem Beispiel wird über die ursprünglich aufrufende Komponente, die erste Komponente links im Diagramm, die nächstliegende und die darunter befindliche Schicht aufgerufen. Eine Komponente in der nächstliegenden Schicht kann außerdem die Kernkomponentenschicht aufrufen.

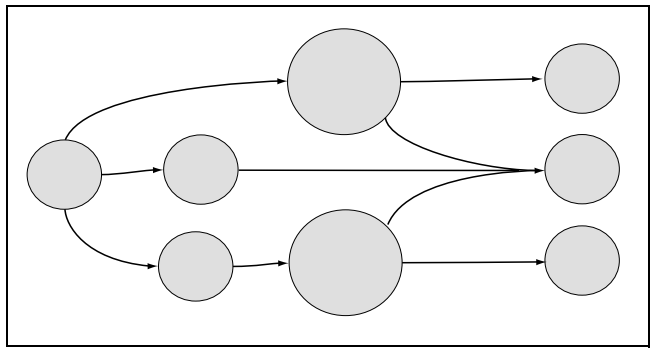

Abbildung 4.7 Beispiel einer nicht strikten Schichtenarchitektur

Der Grund für die Verwendung einer strikten Schichtenarchitektur liegt darin, dass Sie die Auswirkungen von Änderungen innerhalb einer Schicht einschätzen können. Bei einem nicht strikten Schichtenmodell können Sie nicht einschätzen, welche Auswirkungen eine Änderung in einer bestimmten Schicht hervorruft. Selbst wenn Sie wüssten, welche Schichten von der Änderung betroffen sind, wird eine Verwaltung auf längere Sicht sehr kompliziert. Das Ziel bei der Schichtenarchitektur besteht in der Verringerung der Auswirkungen, die sich aus einer Änderung ergeben.

Betrachten wir diese Situation nun in der Praxis. Ich habe bereits erwähnt, dass Sie eine relativ strikte Schichtenarchitektur anstreben sollten. Der Grund, warum ich »relativ strikt« sage, ist der, dass es gelegentlich nicht möglich ist, eine strikte Schichtenarchitektur zu implementieren. Oder es ist möglich, führt jedoch zu einer monolithischen Anwendung, die aus verschiedenen, stark voneinander abhängigen Komponenten besteht, bei der eine Ersetzung nicht mehr möglich ist. Der Schlüssel liegt in der Schichtung und dem Entwurf guter Schnittstellen. Dies kann gelegentlich bedeuten, andere Schichten zu umgehen, um bestimmte Kernfunktionen aufzurufen.

Rückrufe und Ereignisse

Die Schichtenarchitektur setzt voraus, dass eine Komponente der höheren Ebene eine Komponente niedrigerer Ebene aufruft – die Reihenfolge beim Aufruf ist sequenziell. Es gibt jedoch Situationen, in denen Sie einen Rückruf erzeugen möchten, bei dem ein Konsument einer Schnittstelle eine eigene Schnittstelle an den Provider weiterleitet. Ein Rückruf ähnelt einer gegenseitigen Abhängigkeit, der Unterschied liegt darin, dass die an den Konsumenten weitergeleitete Schnittstelle nicht Teil der Benutzerschnittstelle ist. Die Schnittstelle stellt eine separate Einheit dar, wie dargestellt in Abbildung 4.8.

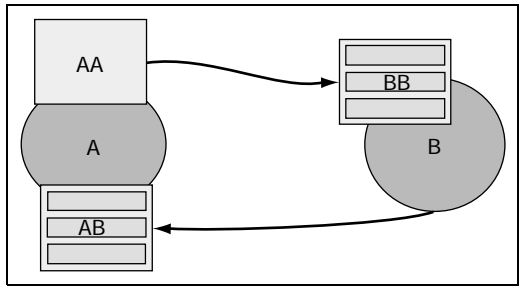

Abbildung 4.8 Rückrufimplementierung zwischen zwei Komponenten

In Abbildung 4.8 sind zwei Komponenten sichtbar, Komponente A und Komponente B. Komponente A implementiert Schnittstelle AB und verfügt über eine Implementierung mit Namen AA. Komponente B implementiert Schnittstelle BB. Sobald Komponente A Schnittstelle BB aufruft, wird Schnittstelle AB ein Verweis auf Komponente B übermittelt. Jetzt verfügt Komponente B über die Fähigkeit, jederzeit Komponente A aufzurufen. Dieses Verhalten wird auch als **asynchroner Aufruf** oder als **ereignisbasiertes Verhalten** bezeichnet.

Betrachtet man Komponente A genauer, wird deutlich, dass die Implementierungen der Schnittstellen AA und BB dieselbe Komponente, jedoch unterschiedliche Bereich der Komponente verwenden. Dies bedeutet, dass keine gegenseitige Abhängigkeit entsteht, da das Testen fortgesetzt werden kann. Es handelt sich hier jedoch um eine Gratwanderung, da die internen Strukturen der Komponente unter Umständen den gleichen Code verwenden, wodurch wiederum gegenseitige Abhängigkeiten im Code entstehen. Der Punkt ist, dass gelegentlich gegenseitige Abhängigkeiten implementiert werden müssen.

Testen von Komponenten

Testläufe werden häufig durchgeführt, in vielen Fällen erfolgt jedoch kein ordnungsgemäßer Test. Das Testen kann sehr langwierig sein, und wenn eine Sache langwierig wird, neigen viele Menschen dazu, sie entweder zu verschieben oder abzukürzen. In beiden Fällen führt dies dazu, dass kein richtiger Test erfolgt. Schlecht durchgeführte Tests führen zu schlechter Software.

Das Testen von Komponenten unterscheidet sich vom Testen einer Anwendung, da keine Benutzerschnittstelle vorhanden ist. Eine Benutzerschnittstelle erschwert das Testen von Komponenten, da es schwierig sein kann, eine Benutzerschnittstelle als richtig oder falsch einzuschätzen. Bei Komponenten ist eine eindeutige Schnittstelle vorhanden, die zu messbaren Ergebnissen führt.

Komponenten können entweder anhand eines Programms oder eines Skripts getestet werden, mit der die Schnittstelle aufgerufen wird. Das Testen von Komponenten kann nicht mit dem Testen eines Bibliotheksmoduls verglichen werden, da eine Komponente, im Gegensatz zu einer Bibliothek, einen Statuswert enthält, der nur teilweise durch die zugehörigen Parameter definiert wird. Zum Testen einer Komponente müssen die Parameter und der Status innerhalb der Komponente validiert werden.

4.1.4 Einige Komponentenbeispiele

Zum Verständnis der verschiedenen Vertragstypen müssen verschiedene Komponententypen beschrieben werden. Zu den Komponententypen gehören:

- **COM:** COM (Component Object Model) ist eine Technologie, mit der zwei separate Codeabschnitte einer Anwendung miteinander kommunizieren können. COM basiert auf dem binären Konzept der VTable (Virtual Functions Table). VTables werden in Kapitel 8, »COM-Schnittstellen«, im Rahmen der Besprechung von COM näher erläutert. Im COM-Modell repräsentiert die VTable eine Schnittstellensignatur von Methoden, die durch den Anwendungscode implementiert werden. COM besitzt die Fähigkeit zur Kommunikation mit weiteren COM-Komponenten auf demselben Computer oder einem Remotecomputer.
- **ADO/OLE DB:** ADO (Active Data Objects) und OLE DB (Object Linking and Embedding Database) sind Beispiele für COM-Schnittstellen mit spezifischen VTable-Signaturen. Diese Schnittstellen sind spezifisch für den jeweiligen Datenbankprovider. Mit Hilfe dieser vordefinierten Schnittstellen kann der Benutzer verschiedene Datenbanken auswählen, ohne dass der Code für eine spezielle Datenbank umgeschrieben werden muss.
- **ODBC:** ODBC (Open Database Connectivity) ähnelt ADO und OLE DB, bei ODBC handelt es sich jedoch um eine DLL-Spezifikation (Dynamic Link Library). Es sind keine Objekte vorhanden, nur Funktionsaufrufe zu einer generischen Bibliothek. Auch ODBC (wie ADO und OLE DB) ermöglicht die Auswahl verschiedener Datenbanken, ohne dass der Client umgeschrieben werden muss.
- **Gespeicherte SQL-Prozeduren:** Eine gespeicherte SQL-Prozedur (Structured Query Language) stellt eine Methode zur Interaktion mit der Datenbank dar, bei der keine Details zur Datenbank bekannt sein müssen. Eine gespeicherte Prozedur ähnelt einem Funktionsaufruf und ist in der Datenbank gespeichert.
- **HTTP/CGI:** Das HTTP-Protokoll (Hypertext Tranfer Protocol) wird zur Suche im Web verwendet. Über dieses Protokoll wird die Datenübertragung zwischen Konsument und Provider definiert. HTTP ist eine Art standardisierte Middleware, die global anerkannt wird. Während HTTP keine Komponenten im Sinne

von COM unterstützt, wird CGI (Common Gateway Interface) unterstützt, eine Methode zum Aufrufen serverseitiger Funktionen. HTTP wird üblicherweise nicht als Komponententechnologie klassifiziert, fällt aber eigentlich unter diese Kategorie, da HTTP Middleware einsetzt und eine Methode zur Erstellung von Konsument und Provider darstellt.

- **XML:** XML (Extensible Markup Language) stellt den Datenaspekt einer Komponententechnologie dar. XML ist ein offener Standard zur Datenspeicherung sowie für den Datenaustausch zwischen Konsument und Provider. In Kombination mit HTTP/CGI stellt XML eine vollständige, vielseitige Komponententechnologie dar, die als Standard betrachtet werden kann. Im Verlauf dieses Buches wird diese Komponententechnologie zur Anwendungsentwicklung eingesetzt.

4.2 Entwickeln von Prototypen

Beim UML-Entwurf für die Anwendung, der im letzten Kapitel besprochen wurde, lag das Hauptziel in der Erstellung einer Funktionsspezifikation ohne Berücksichtigung der involvierten Objekte. Jetzt besteht der nächste Schritt im Anwendungsentwicklungszyklus in der Entwicklung eines Prototyps.

Der Prototyp stellt eine vorläufige Realisierung der Komponenten dar, die zusammen die vollständige Anwendung bilden. Das Entwickeln eines guten Prototyps ist nicht so einfach, wie dies zunächst erscheinen mag, da die mit einem Prototyp zu erreichenden Ziele häufig falsch ausgelegt werden.

4.2.1 Was kennzeichnet einen guten Prototypen?

Wie der UML-Entwurf erfolgt der Prototypentwurf ohne besondere Berücksichtigung der Objekthierarchie. Dies ist wichtig, da Sie selten alle Details eines Projekts kennen werden. Sie haben vielleicht Kenntnisse über die Technologie, aber Sie kennen niemals alle Anwendungsdetails, selbst dann nicht, wenn Sie dies glauben.

Das Ziel bei der Entwicklung eines Prototyps besteht darin, bestimmte Konzepte auszuprobieren. In der Phase der Anwendungsdefinition mussten bei der Implementierung einige Bereiche besonders berücksichtigt werden, z. B. hinsichtlich Geschwindigkeit, Verfügbarkeit, Kosten, Wiederverwendbarkeit oder ähnlicher Faktoren. Diese Faktoren machen Sie vielleicht nervös, da sie die Deadlines und die Funktionalität der Anwendung betreffen könnten. Bei der Entwicklung des Prototyps sollte dies nebensächlich sein. Im Vordergrund steht die Erarbeitung eines umfassenden Verständnisses der Anwendungsfunktionalität. Unter Zuhilfenahme des Prototyps möchten Sie die Auswirkungen der Anwendung einer Technologie und eines Verfahrens auf bestimmte Anwendungsprobleme ermitteln.

Was Sie nicht tun sollten

Es gibt bestimmte Dinge, die Sie nicht tun sollten:

- **Denken Sie nicht zuviel nach.** Sie werden anfangen, über den Entwurf und die beste Entwurfsmöglichkeit für die Objekte nachzudenken. Widerstehen Sie dieser Versuchung. Die Objekte ergeben sich bei der Entwicklung des Prototyps von allein.
- **Verwenden Sie den Prototyp nicht wieder.** Sie geraten vielleicht in die Versuchung, den Prototyp als Basis für die Anwendung zu verwenden. Obwohl dies eine zeitsparende Option zu sein scheint, ist sie es normalerweise nicht. Sie können einzelne Codeabschnitte wieder verwenden, nicht jedoch ganze Komponenten. Sie sollten die funktionierenden Muster des Prototyps wieder verwenden.
- **Entwickeln Sie nicht die gesamte Benutzerschnittstelle.** Über viele Prototypen wird lediglich die Benutzerschnittstelle entwickelt. Das liegt daran, dass die Benutzerschnittstelle einfach nachzuvollziehen und für Demonstrationen gut geeignet ist. Der Zweck eines Prototyps ist jedoch keine Demonstration, sondern die Verfeinerung des Anwendungsentwurfs. Dies soll nicht heißen, dass keine Benutzerschnittstelle entwickelt werden soll.

4.2.2 Zu implementierende Funktionen

Wo sollten Sie beginnen? Der erste Schritt besteht darin, eine Liste der unterschiedlichen technischen Bestandteile der Anwendung aufzustellen. Hierbei geht es nicht darum, sämtliche der technischen Details zu verstehen. Nachfolgend sehen Sie die Liste für das in Kapitel 3 vorgestellte Konferenzanmeldungsprojekt:

- Webbasierte Benutzerschnittstelle
- Plattformübergreifende Benutzerschnittstelle
- COM-Komponenten ohne Benutzerschnittstellenattribute
- SQL Server-Datenbank
- Gespeicherte Prozeduren
- COM-Komponenten der mittleren Schicht
- Automatischer Import in Legacysysteme
- Faxerstellung

Einige der in der Liste aufgeführten Elemente wurden im Domänenmodelldokument nicht genannt. Dies geschah zur Vereinfachung des Domänenmodelldokuments. Meine Absicht lag darin, den Ansatz bei der Erstellung des Domänenmo-

delltextes aufzuzeigen. Jetzt, bei der Entwicklung der Prototypkriterien, müssen jedoch alle Aspekte definiert werden.

Gehen Sie mit dieser Liste zu Ihrem Kunden und erkundigen Sie sich, welche technischen Aspekte von Wichtigkeit sind. Die Schwierigkeit besteht darin, einen Kunden richtig zu verstehen. Im Beispiel verfügte der Kunde über ein abgeschlossenes Wirtschaftsstudium. Er kannte sich in Bezug auf Konferenzen und die Unterstützung der Entwicklergemeinschaft aus. Die Kenntnis der technischen Details dagegen war minimal. Dennoch war dem Kunden bekannt, welche Funktionen benötigt wurden. Ich umschrieb die technischen Details für den Kunden, und es entstand die folgende Liste:

- Jeder, der mit einem Webbrowser ausgestattet ist, soll auf die Site zugreifen und diese verwalten können. Auf der Site sollen alle »coolen« DHTML-Funktionen (Dynamic HTML) enthalten sein.
- Einigen Benutzern steht möglicherweise kein E-Mail-Konto zur Verfügung, daher müssen zur Bestätigung der Konferenzanmeldungen auch Faxe eingesetzt werden.
- Die Website dient als Datencenter für alle Lieferanten von Drittanbieterfirmen.

Diese Liste wird mit der ursprünglichen technischen Liste verglichen, in der folgende Aspekte berücksichtigt wurden: Ein Gesichtspunkt war die Forderung, mit einem beliebigen Browser auf alle Funktionen zugreifen zu können, auf der anderen Seite stand die Notwendigkeit zur Verwendung von DHTML-Funktionen. Die Bestätigung per Fax stellte einen weiteren Problembereich dar, da Microsoft in vorherigen Versionen von **Windows NT keine gute netzwerkbasierte Faxlösung** bereitstellte. Der dritte Aspekt stellte die Forderung eines Datencenters dar, wofür sämtliche Drittanbieter über einen Internetzugang verfügen müssten. Dies war nicht der Fall. Einige Drittanbieter verwendeten weiterhin DOS-Anwendungen auf Windows 3.1-Arbeitsstationen.

Anhand dieser Liste kann ein Prototyp entwickelt werden.

4.2.3 Festlegen der ersten Codezeile

Basierend auf dem Listenvergleich mag es so erscheinen, als ob die gesamte Anwendung als Prototyp geschrieben werden muss. Dies ist jedoch nicht der Fall. Vom UML-Anwendungsfallmodell ausgehend, können Sie alle Anwendungsfälle betrachten und die minimale Anzahl Anwendungsfälle herauspicken, die alle Bereiche der Liste betreffen. So entsteht ein unvollständiger, minimaler Prototyp, aber mit diesem Prototyp wird die Anwendung optimal untersucht.

Der gewählte Anwendungsfall bestand in der Registrierung eines potenziellen Konferenzteilnehmers für eine Konferenz. Dieser Anwendungsfall führt zur Implementierung aller Kernfunktionen, die bei der Anwendung berücksichtigt werden müssen. Darüber hinaus erhalten Sie ein repräsentatives Bild zum Kompliziertheitsgrad der gesamten Anwendung.

Verwenden Sie bei der Entwicklung des Prototyps keine UML-Tools, auf diese Weise vergeuden Sie lediglich Zeit. Sie möchten die Entwicklung des Prototyps so schnell wie möglich abschließen. Sie möchten aber auch keine technischen Shortcuts verwenden, darum entwickeln Sie den Prototyp. Sie möchten einige Theorien testen. Dies bedeutet, dass Sie Ihren gesunden Menschenverstand einsetzen und die Komponenten geschickt definieren. Sind die Komponenten nicht gut durchdacht, werden bestimmte Komponenteninteraktionen maskiert, die zu Problemen bei der Anwendungsimplementierung führen können.

4.2.4 Codierungsstandards

Der Prototyp erfordert das Schreiben von Code. Dieser Code wird wahrscheinlich nicht wieder verwendet, die im Hinblick auf die Prototypentwicklung gemachten Erfahrungen dagegen schon. Zur Beschleunigung der Prototypentwicklung sollte eine Dokumentation vermieden werden. Bei fehlender Dokumentation muss jedoch unbedingt darauf geachtet werden, dass der Code leicht lesbar ist. Der Code selbst stellt hier die Dokumentation dar. Wenn das Team zusammenkommt und den Erfolg oder das Fehlschlagen eines Prototyps einschätzt, dient der Code als Grundlage für diese Einschätzung.

Kommentierung

Das Erstellen von Kommentaren ist eine Kunst und erfordert einige Vorüberlegungen. Der Hauptgrund für einen Kommentar ist die Erläuterung der Codeabsicht, falls diese aus dem Code selbst nicht klar hervorgeht. Kommentare werden nicht angefügt, wenn die Funktion klar ersichtlich ist. Es wird ein gewisser gesunder Menschenverstand vorausgesetzt. Beim Prototyp für Microsoft Deutschland lag der folgende, schlecht kommentierte Code vor.

```
Public Function updateUser()
Dim objUser As UserSite
Dim status As Long

Set objUser = CreateObject("VB_PDCRegistration.UserSite")
objUser.user.id = user.id
status = objUser.getStatus()
```

```
   If status = 1 Then
   'Der Benutzer benötigt eine vollständige Adresse, daher leere Zeile einfügen
      Call addEmpty
   End If

   db.StoredProcedure "SQL_PDCRegistration", "clientUpdate200"
```

Der Code enthält nur einen Kommentar, »Der Benutzer benötigt eine vollständige Adresse, daher leere Zeile einfügen«. Was soll dieser Kommentar bedeuten? Bei Durchsicht des Codes können wir erkennen, was passiert, wenn **status** den Wert 1 erhält. Was wird erreicht, wenn **status** den Wert 1 annimmt? Jemand, der den Code nie zuvor gesehen hat, muss sich die Implementierung von **objUser.getStatus** ansehen, um den Zusammenhang zu verstehen. Der genannte Kommentar ist schlecht, da der Leser den Ablauf des Komponentenaufrufs nachvollziehen muss, um den Code zu verstehen.

Was muss kommentiert werden? Für Anfänger muss erläutert werden, was erreicht werden soll. Der Funktionsname gibt an, dass Sie versuchen, einen Benutzer zu aktualisieren, aber im Kommentar wird einfach angegeben, dass beim Statuswert 1 eine leere Zeile eingefügt werden muss. Da diese Funktionalität nicht offensichtlich ist, muss ein Kommentar angefügt werden. Eine bessere Kommentierung des Code lautet folgendermaßen:

```
Public Function updateUser()
   Dim objUser As UserSite
   Dim status As Long

   Set objUser = CreateObject("VB_PDCRegistration.UserSite")
   objUser.user.id = user.id
   ' Abrufen des Registrierungsstatus des Benutzers. Mögliche
   ' Werte sind "leer", "Site registriert" oder "Konferenz registriert"
   status = objUser.getStatus()

   If status = 1 Then
      ' Zur Einsparung von Funktionsaufrufen wird bei Benutzerstatus
      ' "leer" vorausgesetzt, dass ein gültiger, aber leerer Benutzereintrag
      ' hinzugefügt wird
      Call addEmpty
   End If
```

Beim Lesen des vorstehenden Kommentars wird die Funktionalität ersichtlich. Im ersten Kommentar wird erläutert, dass der Status die Art der Registrierung in der

Datenbank angibt. Im zweiten Kommentar wird, zur Einsparung von Funktionsaufrufen, ein leerer Eintrag hinzugefügt, wenn der Benutzer nicht vorhanden ist.

Noch etwas zu den Kommentaren. Der erste Kommentar ist nicht erforderlich, da das Statusflag auf Komponentenmethodenebene in der Methode **UserSite.getStatus** definiert werden könnte. So wäre das Hinzufügen eines Kommentars nicht nötig. Hier wurde der Kommentar eingefügt, da anhand des Statuswertes eine wichtige Entscheidung getroffen wird – in diesem Fall das Hinzufügen eines Eintrags zur Datenbank. Würde der Status zur späteren Verwendung auf eine Variable übertragen, wäre ein Kommentar nicht erforderlich. Sie benötigen Kommentare, um dem Leser eine Codesequenz zu erläutern, ohne dass dieser blättern muss, denn dies ist störend, unnötig und zeitaufwendig.

Benennungskonventionen

Kommentare erläutern den Code, aber Benennungskonventionen ermöglichen Ihnen, den Code zu lesen. Code mit expliziter Benennung erfordert weniger Kommentare.

Was zeichnet eine gute Benennungskonvention aus? Dies hängt von Ihrer Erfahrung ab. Bei einer guten Benennungskonvention muss zunächst sichergestellt werden, dass alle Teammitglieder die Benennungskonventionen verstehen. Hier einige Faustregeln:

- Verwenden Sie anstelle von kurzen Namen lange Namen. Der Name **.getStatus** ist besser als **.stat**, denn der erste Bestandteil des Namens gibt an, dass Sie den Status des Objekts abrufen. **Stat** alleine ist nicht aussagekräftig genug.
- Verwenden Sie keine Akronyme, da diese regions- und sprachenabhängig variieren können. Akronyme können zu einer Fehlinterpretation des Codes führen. Wenn Sie Akronyme verwenden, sollten Sie sich an eine kurze Liste vordefinierter Akronyme halten, die von allen anderen Entwicklern im Team verwendet wird. Die Verwendung von Akronymen sollte konsistent erfolgen – wenn Sie anstelle von »table« »tbl« verwenden, sollten Sie immer die Kurzform verwenden.
- Verwenden Sie Begriffe aus dem Domänentextmodell der Anwendung. Diese Begriffe werden im Text definiert, daher kann jeder, der dieses Dokument gelesen hat, den Code verstehen.
- Eine Variable oder Deklaration muss sich nicht nur aus einem Begriff zusammensetzen. Verschiedene Begriffe können zu einem einzelnen, besser verständlichen Wort zusammengefasst werden.

Bei der Verwendung zusammengesetzter Begriffe sollten Sie daran denken, dass diese das Lesen des Code erschweren. Verwenden Sie daher Großbuchstaben

oder Unterstriche, um den Beginn eines neuen Wortes kenntlich zu machen. Sehen Sie sich die folgenden vier Beispiele an:

```
SoMevAriAble
SomeVariable
someVariable
some_variable
```

Das erste Beispiel zeigt, wie verwirrend ein Wort dem Leser erscheinen kann. Im zweiten Beispiel werden Großbuchstaben verwendet. Sie würden diesen Stil üblicherweise zur Kennzeichnung eines Daten- oder Objekttyps bzw. eines Funktionsaufrufs einsetzen. Das dritte Beispiel ist eine Variable, bei der es sich um eine Objektinstanz handeln kann. Das letzte Beispiel wird in Sprachen ohne Berücksichtigung der Groß- und Kleinschreibung verwendet, z. B. SQL.

Die Groß- oder Kleinschreibung des ersten Buchstabens ist im Allgemeinen nicht definiert. In Java und Smalltalk beginnten Objektmethoden üblicherweise mit einem Kleinbuchstaben. In Visual C++ und Visual Basic beginnt der Methodenname mit einem Großbuchstaben. Jede dieser Konventionen ist akzeptabel, solange sie konsistent eingesetzt wird.

Häufig verwendet wird auch die ungarische Benennungskonvention. Ein Zeichenfolgenwert wird hierbei z. B. durch **strVar** dargestellt. Der Datentyp wird in ein Akronym konvertiert und erhält als Präfix den Namen der Variablen. Auf diese Weise ist erkennbar, welchen Datentyp die Variable trägt, ohne dass die Deklaration bekannt ist. Einige Programmierer befürworten diese Notation, andere bemängeln, dass der erste Buchstabensatz eher verwirrend und schwer zu verstehen ist. Auch hier gilt: wenn Sie sich für eine Notation entschieden haben, sollten Sie diese konsistent einsetzen.

Weitere Tipps für die Codierung

Beim Schreiben von Anwendungscode sollten Sie niemals Shortcuts verwenden, da diese den Code zu kryptisch werden lassen. In C++ beispielsweise können Sie statt einer **if**-Anweisung die Notation **? :** verwenden. Das Problem hierbei ist, dass diese Notation nicht sehr selbsterklärend ist. Sicher kennen die meisten C++-Programmierer die Bedeutung dieser Notation, aber mit einer **if**-Anweisung können auch Nicht-C++-Programmierer den Code problemlos lesen. Das **if** ist einfacher zu verstehen und führt zu einer besseren Trennung der verschiedenen Codeabschnitte.

4.3 Resümee

Komponenten stellen die wichtigsten Elemente aller modernen, mehrschichtigen Anwendungssysteme dar. Komponenten sind keiner spezifischen Technologie zugeordnet – sie stellen die Grundidee beim Anwendungsentwurf dar. Komponenten erfordern zur erfolgreichen Anwendungsentwicklung eine Trennung von Schnittstelle und Implementierung. So kann eine Komponente aktualisiert werden, ohne dass alle Clients aktualisiert werden müssen, die von der Komponente abhängen.

Bevor eine Anwendung geschrieben werden kann, muss der letzte Schritt im Anwendungsentwicklungszyklus durchgeführt werden, die Entwicklung eines Prototyps. Das Entwickeln eines Prototyps ist eine Herausforderung. Hierbei muss der Wunsch, möglichst alle Funktionen zu implementieren, zurückgestellt werden. Die Idee besteht darin, die wichtigsten Elemente zu implementieren und anschließend zu ermitteln, ob die Annahmen über die Anwendung sich als richtig erweisen. Fachleute nennen dies »einen Versuchsballon starten«.

Im nächsten Kapitel soll mit der Codierung begonnen werden. Dieser Eintritt in die Implementierungsphase des Anwendungsentwicklungszyklus kann beginnen, sobald die letzte Entwurfsphase des Anwendungsentwicklungszyklus abgeschlossen ist.

Das Kapitel fehlt in dieser PDF-Datei.

Sorry, aber wir wollen ja auch das eine oder andere Buch verkaufen :-)

7 Entwickeln einer Webanwendung

In Kapitel 5 wurde der Thin Client vorgestellt, Kapitel 6 enthielt eine Einführung in den Rich Client. Im vorliegenden Kapitel soll der Prozess der Webanwendungsentwicklung erläutert werden. Das Hauptziel besteht hierbei darin, die Informationen aus Kapitel 5 und 6 mit einigen neuen Verfahren zu kombinieren. Das Entwickeln von Webanwendungen unterscheidet sich von der Entwicklung traditioneller Anwendungen dahingehend, dass bei einer Webanwendung eine Server- und eine Clientseite vorhanden ist. Die Serverseite ist nicht vom ausgeführten Clientbrowser abhängig – es können Inhalte für einen beliebigen HTTP-basierten Browser generiert werden.

7.1 Erstellen der serverseitigen Anwendungslogik

Bei der einfachsten Methode der Webanwendungsentwicklung wird vorausgesetzt, dass die wichtigsten Verarbeitungsprozesse auf Serverseite stattfinden. Dies bedeutet, dass auf der Serverseite Harddaten gespeichert werden. Der Client arbeitet mit Softdaten. Im Falle des Thin Client können auf der Clientseite keine Softdaten gespeichert werden. Hier wird der Status schrittweise erstellt und temporär im ASP-Objekt **Session** gespeichert.

7.1.1 Cookies, ja oder nein?

Session-Variablen werden mit Hilfe von Cookies implementiert, über die sich die Geister scheiden. Es gibt viele Endbenutzer, die keine Cookies akzeptieren, und es gibt Proxys, die Cookies herausfiltern. Der Grund für die Verwendung von Cookies ist einfach. Bei der Entwicklung des ursprünglichen HTTP-Protokolls wurde dies als statuslos betrachtet. Dies ist jedoch nur scheinbar eine gute Eigenschaft eines Transaktionsverarbeitungssystems.

Das eigentliche Problem besteht darin, dass die Statuslosigkeit für jede einzelne Anforderung gilt. Beim Anfordern einer Seite wird diese aufgerufen, anschließend wird die Verbindung getrennt. D.h., obwohl die gesamte Transaktion in einer HTTP-Anforderung ausgeführt wird, wird der Vorgang sehr langwierig – es muss für jede Anforderung ein Kontext erstellt werden. Zur Umgehung dieser Einschränkung verwendet ASP (Active Server Pages) zur Identifizierung der Benutzer und für das Erstellen eines **Session**-Objekts Cookies. Ein Cookie ist ein einfaches Token zur Benutzeridentifizierung. Es ist textbasiert und führt zu keiner Codeausführung. Da ein Cookie nur zur Identifizierung des aktuellen Benutzers, nicht jedoch zur Ermittlung der Identität eines Benutzers eingesetzt wird, stellt es auch keinen schwerwiegenden Eingriff in die Privatsphäre dar. Wenn Ihnen Cookies

nicht geheuer sind, leeren Sie den Cookiepuffer einfach täglich. Cookies sind harmlos!

7.1.2 Verschieben der Daten

Als Nächstes müssen Sie ermitteln, wie Daten vom Client zum Server und umgekehrt verschoben werden. Es wurden bereits einige Verfahren hierzu vorgestellt, beispielsweise die Verwendung von CGI (Common Gateway Interface) und XML (Extensible Markup Language), die auch weiterhin verwendet werden sollen. Der Vorteil dieser Methoden liegt darin, dass sowohl CGI als auch XML offene Standards darstellen. Theoretisch könnte der Webserver also von Windows 2000/NT auf UNIX portiert werden und umgekehrt.

Beim Verschieben der Daten wird vorausgesetzt, dass die Datenübertragung nicht immer fehlerfrei verläuft. Die Programme werden daher in einem »defensiven« Stil geschrieben.

7.1.3 Noch etwas zu ASP

In Kapitel 5 wurden die zwei vordefinierten Objekte **Application** und **Session** erläutert. Es sind noch fünf weitere vordefinierte ASP-Objekte vorhanden. Mit diesen Objekten wird der Mechanismus zur Interaktion mit ASP bereitgestellt. In einem der vorgestellten Architekturdiagramme wurde beispielsweise gezeigt, dass ein Anforderungsstream und ein Antwortstream vorhanden sind. Dementsprechend stehen die zwei ASP-Objekte **Request** und **Response** zur Verfügung. Die Definitionen der verbleibenden fünf ASP-Objekte lauten folgendermaßen:

- **ASPError:** Ein neues Objekt, das mit IIS 5.0 (Internet Information Server) eingeführt wurde. Mit diesem Objekt können Fehler abgerufen werden, die innerhalb des ASP-Ausführungskontextes aufgetreten sind.
- **ObjectContext:** Dieses Objekt ermöglicht bei Ausführung der ASP-Seite die Interaktion mit dem Transaktionskontext.
- **Request:** Dieses Objekt repräsentiert den eingehenden Stream und enthält Informationen zur Anforderung. Zu diesen Informationen zählen beispielsweise Parameter, Cookies und clientseitige Zertifikate.
- **Response:** Dieses Objekt repräsentiert den ausgehenden Stream. Mit diesem Objekt ist es möglich, HTTP-Header (Hypertext Transfer Protocol) zu definieren, Puffer für das verzögerte Senden zu erstellen sowie auf weitere HTTP-Elemente zuzugreifen.
- **Server:** Ein einfaches Dienstprogrammobjekt zur Instanziierung weiterer COM-Objekte (Component Object Model). Diese COM-Objekte können zur Erweiterung der Funktionalität von ASP-Skripts eingesetzt werden.

Verwenden von COM-Objekten

Innerhalb einer HTML-(Hypertext Markup Language) oder ASP-Seite können COM-Komponenten verwendet werden. COM-Komponenten sind binäre Ausführungseinheiten, die Code enthalten und eine Schnittstelle zu ihrer Funktionalität offen legen. Die Schnittstelle wird innerhalb der Skriptumgebung verwendet.

Ein COM-Objekt wird entweder mit Hilfe einer PROG-ID (Programm-ID) oder einer CLSID (Klassen-ID) instanziiert. Der Unterschied zwischen diesen beiden Methoden besteht darin, dass erstere ID von Menschen lesbar, die zweite ID maschinenlesbar ist. Nachfolgend ein Beispiel einer PROG-ID:

```
COMSNAP.SnapinAboutImpl.1
```

Die PROG-ID setzt sich typischerweise aus drei Bestandteilen zusammen. Der erste Teil repräsentiert die COM-Komponente, in der das COM-Objekt gespeichert ist. Der zweite Teil stellt den COM-Objektverweis dar, beim dritten Teil handelt es sich um die Versionsnummer des COM-Objekts. Diese Unterteilung folgt einer Benennungskonvention und kann daher variieren.

Die PROG-ID wird in eine Zahl aufgelöst, die als CLSID oder COM-Coklasse bezeichnet wird. Der Computer verwendet die CLSID zur Referenzierung eines COM-Objekts. Eine CLSID sieht folgendermaßen aus:

```
{52938292-1D8B-11d3-955F-0080C700807A}
```

Diese Zahl muss nicht manuell generiert werden, sondern kann anhand der Microsoft Visual Studio-Tools erstellt werden.

7.2 Entwickeln der Webanwendung

Kehren wir zum ursprünglichen Websiteprojekt für die Konferenzanmeldung zurück. Die Forderung lautete, eine ansprechende und gleichzeitig einfache Website zu erstellen. Aufgrund dieser Anforderung muss die Website in eine coole, schicke Website und eine funktionelle Website unterteilt werden, mit der die größtmögliche Anzahl Endbenutzer unterstützt wird. Im ursprünglichen Domänenmodelltext wurde angemerkt, dass der Benutzer eine Anmeldung auch über Fax vornehmen können soll, falls kein Internetzugang verfügbar ist. Dies ermöglichte das Aufstellen einiger Regeln und minimaler Browseranforderungen, da jeder Benutzer, der diese Anforderungen nicht erfüllt, die Anmeldung per Fax vornehmen kann. Eine dieser Regeln besagt, dass der Browser Cookies akzeptieren muss.

Viele Computer verfügen jedoch nicht über diese Optionen. In diesem Fall besteht das Problem weiterhin darin, eine Website zu erstellen, die cool, schick und gleichzeitig funktionell ist. Hierzu müssen die verschiedenen Browser nach ihren

Fähigkeiten sortiert werden. Anhand dieser Fähigkeiten kann der Server entscheiden, welcher Inhalt gesendet wird.

7.2.1 Ermitteln der Browserfähigkeiten

Fordert ein Browser eine Webseite an, identifiziert sich der Browser gegenüber dem Server. Dieses Token wird als Benutzer-Agent-ID bezeichnet und in den HTTP-Headern gespeichert. Schreiben Sie den folgenden Code, damit es auf dem Server abgerufen wird:

```
Request.ServerVariables( "HTTP_USER_AGENT")
```

So wird der Browsertyp zurückgegeben, beispielsweise:

```
Mozilla/4.0 (compatible; MSIE 4.01; Windows NT)
```

Die Textzeichenfolge beginnt mit der Definition der unterstützten Kompatibilitätsebene. In diesem Beispiel wird übermittelt, dass der Browser mit Mozilla 4.0 kompatibel ist (Mozilla ist der Spitzname des Netscape-Browsers). Da es sich bei dem tatsächlichen Browser um Microsoft Internet Explorer (MSIE) handelt, ist eine zusätzliche Identifizierung erforderlich. In diesem Fall wird MSIE 4.0 auf der **Windows NT**-Plattform ausgeführt.

Unterstützt dieser Browser das Scripting? Werden Rahmen unterstützt? Dies kann anhand der obigen Beschreibung nicht ermittelt werden. Sie können diese Informationen jedoch im Internet abrufen und anschließend Skriptcode für MSIE 4.0 oder andere Browser erstellen. Was ist jedoch mit MSIE 4.0, MSIE 4.02, MSIE 3.**x** oder MSIE, wenn diese auf einem Windows CE-Gerät, unter Windows 95/98 oder auf einem Macintosh ausgeführt werden? Die Anzahl der möglichen Browser- und Plattformkombinationen für den Internet Explorer sind enorm, und das Schreiben einer **if**-Anweisung zur Handhabung sämtlicher Fälle würde sehr viel Zeit in Anspruch nehmen. Glücklicherweise kann dieses Problem einfacher gelöst werden.

Die eigentliche Aufgabe bestand darin, die vom Browser unterstützten Funktionen zu ermitteln. Der Browsertyp könnte ein Schlüssel zu einer Datenbank mit Browserfähigkeiten sein. Das ASP-Framework enthält ein COM-Objekt, mit dem Browsertyp und zugehörige Fähigkeiten abgerufen werden können. Diese Komponente wird als **Browserfähigkeitenkomponente** bezeichnet. Mit ihr wird nicht offen gelegt, um welchen Browser es sich handelt, sondern welchen Funktionssatz der Browser unterstützt. Der Browser erstellt hierzu eine Reihe von Variablen, durch die die verschiedenen Browserfunktionen repräsentiert werden. Sehen Sie sich den folgenden ASP-Quellcode an:

```
<%
var bc;
bc = Server.CreateObject("MSWC.BrowserType") %>
<table border=1>
<tr>
<td>Browser</td><td> <%=bc.browser%></td></tr>
<tr>
<td>Version</td><td><%=bc.version%></td></TR>
<tr>
<td>Frames</td><td><%=bc.Frames%></td></tr>
<tr>
<td>Tables</td><td><%=bc.Tables%></td></TR>
<tr>
<td>BackgroundSounds</td><td><%=bc.BackgroundSounds%></td></tr>
<tr>
<td>VBScript</td><td><%=bc.vbscript%></td></tr>
<tr>
<td>JScript</td><td><%=bc.javascript%></td></tr>
</tr></table>
```

Über diesen Code wird ein Objekt erstellt, mit dem versucht wird, eine Datei der Browserfähigkeiten zu laden. Im ersten Abschnitt des ASP-Code wird das Objekt mit **Server.CreateObject** und unter Verwendung der COM-PROG-ID **MSWC.BrowserType** instanziiert.

Nach der Objektinstanziierung wird versucht, die Datei namens **browscap.ini** zu laden. Diese Datei befindet sich im selben Verzeichnis wie **browscap.dll**, der Browserfähigkeitenkomponente. Während der Initialisierung ruft das Objekt die Zeichenfolge für den Browsertyp ab und versucht dann, für diese Zeichenfolge einen Querverweis in der Datei **browscap.ini** zu finden. Die Datei **browscap.ini** ist eine .ini-Datei, bei der die Zeichenfolge für den Browsertyp mit einem Abschnitt in der Datei verknüpft ist. Ein INI-Abschnitt wird definiert, indem Text von eckigen Klammern umschlossen wird.

Suchen wir nach dem zu Beginn des Kapitels erwähnten Benutzer-Agenten.

```
 [IE 4.0]
browser=IE
Version=4.0
majorver=4
minorver=0
frames=TRUE
```

```
tables=TRUE
cookies=TRUE
backgroundsounds=TRUE
vbscript=TRUE
javascript=TRUE
javaapplets=TRUE
ActiveXControls=TRUE
Win16=False
beta=False
AK=False
SK=False
AOL=False
crawler=False
cdf=True
[Mozilla/4.0 (compatible; MSIE 4.*)]
parent=IE 4.0
```

In dieser Beispieldatei befinden sich zwei INI-Abschnitte, **[IE 4.0]** und **[Mozilla/4.0 (compatible, MSIE 4.*9]**. Bei keinem der beiden Abschnitte handelt es sich um eine genaue Übereinstimmung mit der ursprünglichen Zeichenfolge für den Browsertyp. Im zweiten Abschnitt folgt auf das **4.** jedoch ein Sternchen (*). Dieses Sternchen steht für einen Platzhalter, d.h., alle 4.**xx**-Versionen des Browsers sind eingeschlossen, und hierzu zählt auch der zu Beginn des Abschnitts angegebene Browsertyp. Internet Explorer 4.0, 4.01 und 4.02 weisen im Grunde identische Funktionssätze auf. Der Unterschied zwischen den verschiedenen Versionen liegt in einigen Bug Fixes.

Liegt eine Abschnittsübereinstimmung vor, sind verknüpfte INI-Werte vorhanden. Im Falle der zuvor erwähnten Übereinstimmung gibt es den Wert **parent=IE 4.0**. Dies ist ein Verweis auf einen weiteren Abschnitt, der alle Einstellungen für **parent** enthält. Und tatsächlich, es gibt einen Abschnitt mit der Überschrift [IE 4.0]. Unter dieser befinden sich verschiedene Werte. Sehen Sie sich die folgenden Werte an: **Win16=False**, **cookies=TRUE**. Diese Werte werden als COM-Eigenschaften für das erstellte Objekt der Browserfähigkeiten offen gelegt. Die Werte dieser Eigenschaften sind entweder unwahr oder zutreffend (**false** oder **true**). Daher handelt es sich nicht um einen Windows 16-Bit-Browser, der Browser akzeptiert jedoch Cookies.

Hinzufügen benutzerdefinierter Eigenschaften

Durch das Installieren eines ASP- oder HTTP-Servers wird ein grundlegender Funktionssatz für jeden Browser installiert. Diese Datei **browsercap.ini** kann er-

weitert werden, d.h., die Browserfähigkeiten können auch erweitert werden. Diese Erweiterung wird durch das Definieren von Werten unter einer Abschnittsüberschrift erreicht.

Im Beispiel der Anwendung für die Konferenzanmeldung benötigen wir ein Flag, mit dem angegeben wird, ob ein Browser einen bestimmten minimalen Funktionssatz unterstützt. Dieser Abschnittswert **supportsMinimumFeatures** wird der Datei **browsercap.ini** hinzugefügt, wie nachstehend gezeigt:

```
[Mozilla/4.0 (compatible; MSIE 4.*)]
supportsMinimumFeatures=true
```

Der neue Abschnittswert kann auf einen beliebigen Wert eingestellt werden. Der Wert muss nicht unbedingt **true** oder **false** lauten, es können auch Zeichenwerte oder numerische Werte verwendet werden. Zur Referenzierung der Eigenschaft **supportsMinimumFeatures** in der ASP-Seite wird folgender Code benötigt:

```
<%
var bc;
bc = Server.CreateObject("MSWC.BrowserType") %>
<p>My property</p><p><%=bc.supportsMinimumFeatures %></p>
```

Der Vorteil einer benutzerdefinierten Eigenschaft besteht darin, dass für den Abruf nur eine Codezeile erforderlich ist. Andere Lösungen erfordern mehrere Codezeilen, d.h., es können mehr Fehler entstehen und der Code ist komplexer.

7.2.2 Zurückgeben der richtigen Webseite

Nach Ermittlung der Browserfähigkeiten kann es wünschenswert sein, den Endbenutzer an die geeignete HTML-Seite umzuleiten. Bei der Konferenzanmeldungsanwendung müssen die Benutzer eine Sicherheitsprüfung hinsichtlich Identität und Zugriffsebene durchlaufen. Versucht ein Benutzer, ohne Sicherheitsprüfung auf eine Seite zuzugreifen, muss der Benutzer an die Seite für die Sicherheitsprüfung umgeleitet werden. Für die Dokumentumleitung stehen verschiedene Methoden zur Verfügung.

Automatische clientseitige Umleitung

Die einfachste Methode zur Umleitung wird als clientseitige Umleitung bezeichnet. Der Name ist etwas irreführend, da die Umleitung tatsächlich durch den Server ausgelöst wird. Fordert der Client ein Dokument an, sendet der Server einen HTTP-Befehl zurück, mit dem angegeben wird, dass das Dokument seinen Standort geändert hat und dass der angegebene Inhalt geladen werden muss. Diese

Methode wurde bei der Webanwendung für die Konferenzanmeldung verwendet. Sehen Sie sich den folgenden Quellcode an:

```
<%@ Language=JavaScript %>
<%   if( Session("userId") == -1) {
       Response.Redirect( "../Logon/Default.asp");
   }%>
<HTML>
```

In diesem Beispiel wird der Anmeldestatus in der **Session**-Variablen **userId** gespeichert. Lautet der Status –1, ist der Benutzer nicht angemeldet. In diesem Fall wird die **Response.Redirect**-Methode aufgerufen, um den Benutzer an die Seite **»../Logon/Default.asp«** umzuleiten, der HTML-Seite für die Sicherheitsprüfung.

Automatische Umleitung innerhalb einer Seite Im zuvor genannten Beispiel funktioniert die Umleitung nur, wenn kein Text an den ASP-Stream gesendet wurde. Wie Sie wissen, kann der an den Browser gesendete Text nicht zurückgerufen werden; Text stellt hierbei alles dar, was sich nicht im Skriptcodeabschnitt befindet. Muss eine Umleitung in der Mitte oder am Ende einer Seite vorgenommen werden, muss der ASP-Stream gepuffert werden. Beim Puffern des ASP-Streams wird dieser temporär lokal gespeichert und kann anschließend entweder gesendet oder gelöscht werden. Das vorangegangene Beispiel könnte für diese Methode folgendermaßen umgeschrieben werden:

```
<%@ Language=JavaScript %>
<%Response.Buffer = true%>
<HTML>
<HEAD>
</HEAD>
<BODY>
<%   if( Session( "userId") == -1) {
       Response.Redirect("../logon/default.asp");
       Response.Flush();
       Response.End();
   }%>
</BODY>
</HTML>
<%Response.End()%>
```

Bevor Text in den Stream geschrieben wird, muss **Response.Buffer** auf **true** gesetzt werden. So wird der Puffer vorbereitet. Erfolgt diese Vorbereitung später, tritt ein Fehler auf, da kein Inhalt gepuffert werden kann, der bereits an den Client

gesendet wurde. Nach der Pufferaktivierung kann **Response.Redirect** an beliebiger Stelle innerhalb der HTML-Seite aufgerufen werden. Vergessen Sie jedoch nicht, **Response.Flush** und **Response.End** hinzuzufügen – so wird verhindert, dass die vom Proxy gesendete Meldung angezeigt wird, dass das Objekt verschoben wurde.

Gelegentlich kann es wünschenswert sein, eine gepufferte HTML-Seite neu zu erstellen, da der Inhalt geändert werden muss. Verwenden Sie hierzu die Methode **Response.Clear**. So wird der Puffer geleert und ein Neustart ermöglicht.

Wenn eine große Menge Daten an den Client gesendet werden muss und hierzu eine erhebliche Verarbeitungszeit erforderlich ist, können Sie den Inhalt in einzelnen Blöcken senden. Über die Methode **Response.Flush** wird der ASP-Streaminhalt gesendet, der bis zu diesem Zeitpunkt erstellt wurde. Nach dem Senden des Inhalts kann **Response.Redirect** jedoch nicht mehr verwendet werden.

Der letzte Schritt besteht darin, die Verarbeitung der ASP-Seite zu beenden und den Inhalt zu senden, der sich im Puffer befindet. Hierzu kann der Befehl **Response.End** eingesetzt werden. Seien Sie bei der Platzierung dieser Methode vorsichtig, da durch diesen Methodenaufruf die Verarbeitung der aktuellen Seite beendet wird.

Automatische Aktualisierung

Eine weitere Methode zur automatischen Umleitung einer Webseite stellt der HTTP-Befehl **refresh** dar:

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="GENERATOR" content="Microsoft FrontPage (Visual InterDev Edition)
2.0">
<META HTTP-EQUIV="REFRESH" CONTENT="5" URL="nextstep.asp">
<title>Document Title</title>
</head>
```

Das **<META>**-Tag weist das Attribut **HTTP-EQUIV** auf, mit dem angegeben wird, dass diese Seite nach dem Download aktualisiert wird. In diesem Fall wird die Zeit durch das **CONTENT**-Attribut repräsentiert und ist auf fünf Sekunden eingestellt. Das **URL**-Attribut (Uniform Resource Locator) definiert, welche Seite nach Ablauf der fünf Sekunden geladen wird.

Probleme treten bei dieser Methode auf, wenn der Benutzer zum Zeitpunkt der automatischen Browserumleitung noch nicht sämtliche Informationen in ein Formular eingegeben hat. In diesem Fall werden sämtliche Informationen gelöscht,

die der Benutzer bis dato eingegeben hat. Bei einigen Browsern führt das erneute Laden einer Seite dazu, dass der Browser für kurze Zeit blockiert wird.

Automatische serverseitige Umleitung

Mit Einführung von Windows 2000 wurden IIS-integrierte Objekte auf die Unterstützung der serverseitigen Umleitung erweitert. Im Gegensatz zur clientseitigen Umleitung erfordert die serverseitige Umleitung keine neue clientseitige Verbindung.

Die **Server.Transfer**-Methode ermöglicht Folgendes:

- **Session**- und **Application**-Informationen werden zwischen mehreren Webanwendungen übertragen. So kann ein Benutzer innerhalb verschiedener Webanwendungen verfolgt werden.
- Die Umleitung kann während der Skriptausführung erfolgen, ohne dass ein Puffer erstellt werden muss.

Das ursprüngliche Beispiel für die Konferenzanmeldung könnte unter Verwendung dieser Methode folgendermaßen abgeändert werden:

```
<%@ Language=JavaScript %>
<HTML>
<HEAD>
</HEAD>
<BODY>
Logon process
<%   if( Session("userId") == -1) {
       Server.Transfer( "../logon/default.asp");
   }%>
</BODY>
</HTML>
```

Einbetten einer Aktivität

Das Problem bei der Umleitung besteht darin, dass es sich um eine Aktion mit dauerhaften Auswirkungen handelt. Ohne einigen Zusatzaufwand ist es nicht möglich, zur ursprünglichen Seite zurückzukehren. Durch Windows 2000 wird mit dem IIS-Objekt **Server** ein Verfahren bereitgestellt, durch das ein Aufruf einer separaten Webseite wie ein Funktionsaufruf ausgeführt werden kann. So kann eine HTML-Seite in Fragmenten erstellt werden, ähnlich der Verwendung von Rahmen in einer Einzelseite.

Die hierzu verwendete Methode heißt **Server.Execute**. Beispiel:

```
<%@ Language=JavaScript %>
<HTML>
<HEAD>
</HEAD>
<BODY>
Example site
<%      // Banner content
Server.Execute( "../banner/default.asp");%>

</BODY></HTML>
```

In diesem Beispiel wird die Methode **Server.Execute** zum Hinzufügen eines Banners zur aktuellen Webseite verwendet. Die Methode **Server.Execute** führt zur Seitenverarbeitung. Das Verwenden des serverseitigen **include**-Elements zur Erstellung einer Seite in Fragmenten beinhaltet keine Verarbeitung und kann schneller erfolgen. **Include** wird üblicherweise verwendet, wenn der einzuschließende Inhalt auf der Webseite als Verweis verwendet wird. Der Einschluss kann sowohl auf der Client- als auch auf der Serverseite erfolgen.

Einsatzmöglichkeiten der einzelnen Umleitungsmethoden

Die Frage ist, wann welche Umleitungsmethode eingesetzt werden sollte. Die Antwort richtet sich nach den Einschränkungen jeder Methode.

Die Methode **Response.Redirect** ist sehr leistungsstark und leicht zu verwenden, aber viele Webbrowser und Proxys können die zugehörigen Meldungen nicht richtig interpretieren. Statt die clientseitige Umleitung automatisch auszuführen, wird eine Meldung angezeigt. Ein weiteres Problem bei dieser Methode besteht darin, dass die Browserschaltfläche **Zurück** nicht ordnungsgemäß funktioniert. Der Benutzer ist gezwungen, die Schaltfläche schnell zu klicken und darauf zu warten, dass die richtige Seite angezeigt wird.

Die Methoden **Server.Transfer** und **Server.Execute** stellen die besten Methoden dar, wenn die Umleitung auf demselben Rechner erfolgt. Diese neuen Methoden können in der Website für die Konferenzanmeldung verwendet werden. Problematisch bei diesen Methoden ist, dass sie weder einen anderen physischen Rechner aufrufen noch an diesen übermitteln können.

7.2.3 Sprachenunterstützung

Im Internet werden viele verschiedene Sprachen eingesetzt, und ebenso sind Websites mit mehreren Sprachen vorhanden. Je nach der im Browser angezeigten

Sprache möchten Sie vielleicht Textelemente hinzufügen. Mit wachsender Popularität des Webs steigt auch die Zahl der nicht englischsprachigen Webnutzer. Dies bedeutet, dass die Website lokalisiert werden muss, damit die Benutzer die Seiten in ihrer eigenen Sprache anzeigen können.

Der traditionelle Ansatz mehrsprachiger Websites besteht darin, Nationalflaggen hinzuzufügen, mit denen auf Links zu den jeweiligen Sprachen verwiesen wird. Der Benutzer kann dann die gewünschte Version auswählen. Dies ist eine akzeptable Lösung, die jedoch noch erweitert werden kann.

Bei einer optimalen Lösung wählt der Server – abhängig von den Informationen, die der Browser an den Server übermittelt – automatisch die richtige Sprache aus. Hierzu gehen Sie folgendermaßen vor:

```
Request.ServerVariables("HTTP_ACCEPT_LANGUAGE");
```

Der Browser gibt eine Einstellung wie **en-us** für Englisch (USA) oder **de** für Deutsch (Deutschland) zurück. Sowohl der Internet Explorer von Microsoft als auch der Netscape Navigator unterstützen diese Funktion. Die Spracheinstellung richtet sich nicht nach dem installierten Browser, sondern nach der Einstellung für den Computer. Wenn demnach die **Ländereinstellungen**, die sich in der Systemsteuerung befinden, auf die Option **Deutsch (Deutschland)** eingestellt sind, sendet der Client immer die Option **de** an den Server, wenn eine HTTP-Anforderung durchgeführt wird. Der Server erkennt die Spracheinstellung und erstellt den geeigneten Inhalt bzw. leitet den Browser an die Seite um, die den Inhalt in der angegebenen Sprache enthält.

Das Erstellen von Websites, die mehrere Sprachen, verschiedene Browser oder unterschiedliche Technologien unterstützen, ist nicht einfach. Der zusätzliche Aufwand ist jedoch gerechtfertigt, denn die Website erhält so ein professionelles Aussehen. Sie sollten bei der Entwicklung einer lokalisierten Site daran denken, dass Sie mit jemandem zusammenarbeiten, der die Zielsprache und -kultur kennt. Das Lokalisieren des Ausgangstextes umfasst nicht nur die sprachliche Übertragung in die Zielsprache, sondern auch das Übertragen von Bedeutungsinhalten, Farben oder Grafiken in eine andere Kultur.

7.2.4 Erstellen des Hauptteils

Wie bereits in Kapitel 5 erwähnt, erfordert der Hauptteil der Webanwendung die meiste Entwicklungszeit. Sämtliche der bisher genannten Verfahren finden auch hier Anwendung. Für diese Aufgabe ist jedoch mehr Aufwand erforderlich, da eine Interaktion mit der Datenbank, den Geschäftsobjekten und den vom Client gesendeten Daten stattfinden muss.

Bei der Entwicklung der serverseitigen Inhalte sieht sich der Programmierer einer neuen Form der Programmierung gegenüber, da der ASP-Code eine Mischung aus HTML und Skriptcode darstellt. Beim ersten Hinsehen scheint es sich um ein großes Durcheinander zu handeln. Haben Sie sich jedoch erst einmal an die Notation gewöhnt, werden Sie erkennen, dass der ASP-Code sich optimal für Webanwendungen eignet. Der Vorteil ist, dass Sie keine HTML-Tags in die Programmierumgebung einbetten müssen und damit das Lesen der Programmiersprache nicht erschwert wird. ASP ist eine Art Vorlagenansatz.

Der Skriptcode soll anhand von JavaScript und VBScript erläutert werden. Sehen Sie sich folgendes Skript an:

```
<%@ Language=VBScript %>
<HTML><HEAD></HEAD>
<BODY>
<h2>Example Loop</h2>
<%
dim c1
for c1 = 0 to 10
   Response.Write "<br> Count " & c1
%>
   <br>Count <%=c1%>
<%next%>
</BODY></HTML>
```

In diesem Codebeispiel ist eine Schleife (**for...next**) von 0 bis 10 enthalten. Innerhalb der Schleife kann Text auf zwei Arten an den Stream ausgegeben werden. Die erste Möglichkeit besteht darin, den Methodenaufruf **Response.Write** zu verwenden, mit dem Daten in kleinen Blöcken an den Stream gesendet werden. In diesem Beispiel wird der Text dynamisch erstellt. Die zweite Möglichkeit ist die, ohne explizite Methoden Text an den Stream zu senden. Zur Ausgabe einer spezifischen Variable wird die Notation **<%=c1%>** verwendet. In dieser Situation kann innerhalb des Ausgabeblocks kein weiterer Verarbeitungscode eingefügt werden, dies würde zu einem Fehler führen.

Beide Methoden führen zu einer Datenausgabe an den Stream. Welche Methode die bessere ist, lässt sich nicht ohne weiteres sagen, sondern richtet sich nach der jeweiligen Situation. Wenn Sie mit Hilfe des **Response**-Objekts Text an den Stream ausgeben, müssen Sie sicherstellen, dass keines der Elemente eine Formatierung benötigt. Es ist äußerst schwierig, auf diese Elemente eine Vorlage oder Formatierung anzuwenden. Ist dies erforderlich, muss die Codierung manuell er-

folgen. Dieses Szenario eignet sich besonders für die Entwicklung von Skriptobjekten.

Wenn Sie den zweiten Ansatz verwenden, sind HTML- und Skriptcode fragmentierter, aber Sie können einen Editor einsetzen, um die einzelnen HTML-Tags zu bearbeiten. Eine herausragende Eigenschaft dieser Methode stellt das Erzeugen von Sonderzeichen dar; das Anzeigen von Anführungszeichen mit Hilfe von **Response.Write** ist beispielsweise sehr viel schwieriger als mit der Notation **<%=c1%>**. Diese zweite Methode erleichtert die Feinabstimmung der HTML-Tags erheblich, ist jedoch auch besser für das direkte Bearbeiten der ASP-Seite geeignet.

Bei der Kombination von HTML mit Skriptcode funktioniert folgende Formatierung meiner Meinung nach am besten:

```
<%@ Language=VBScript %>
<HTML><HEAD></HEAD>
<BODY>
<h2>Square of number</h2>
<table>
   <tr>
      <td>Number</td><td>Square</td></tr>
<%
dim c1, c2

for c1 = 0 to 10
   c2 = c1 * c1
%>
   <tr>
      <td><%=c1%></td><td><%=c2%></td></tr>
<%next%>
</table>
</BODY></HTML>
```

Die Formatierung ist einfach. HTML-Tags und Code werden mit Hilfe von Tabulatoren eingerückt. Die Tabulatorensequenz im Codeabschnitt sowie in den HTML-Tagabschnitten werden jedoch voneinander getrennt. Die Notationen für Variablenausgabe folgen der Tabulatorennotation des HTML-Tagabschnitts, so sieht die ASP-Seite wie eine kontinuierliche Seite aus.

7.2.5 Die Skriptbibliothek von Visual InterDev

In Kapitel 6 wurden die JavaScript-Objekte vorgestellt. Microsoft Visual InterDev verfügt über eine neue Bibliothek mit dem Namen Skriptobjektbibliothek. Diese arbeitet mit Entwurfszeitsteuerelementen zusammen, um die Entwicklung von Webanwendungen durch Verwenden objektorientierter Methoden zu vereinfachen.

Auf der Clientseite befindet sich ein Ereignismodell. Die Skriptobjektbibliothek erstellt ebenfalls ein Ereignismodell auf Serverseite und verbirgt die Komplexität von Client und Server. So können auf einfache Weise Webanwendungen für Basisbrowser und für Browser erstellt werden, die das gesamte DOM-Modell (Document Object Model) unterstützen. Mit Hilfe der Skriptobjektbibliothek können Sie Methoden der Datenbindung für die Datennavigation nutzen. Die Skriptobjektbibliothek wird mit JavaScript geschrieben, der ASP-Skriptcode kann jedoch in VBScript vorliegen.

Entwurfszeitsteuerelemente

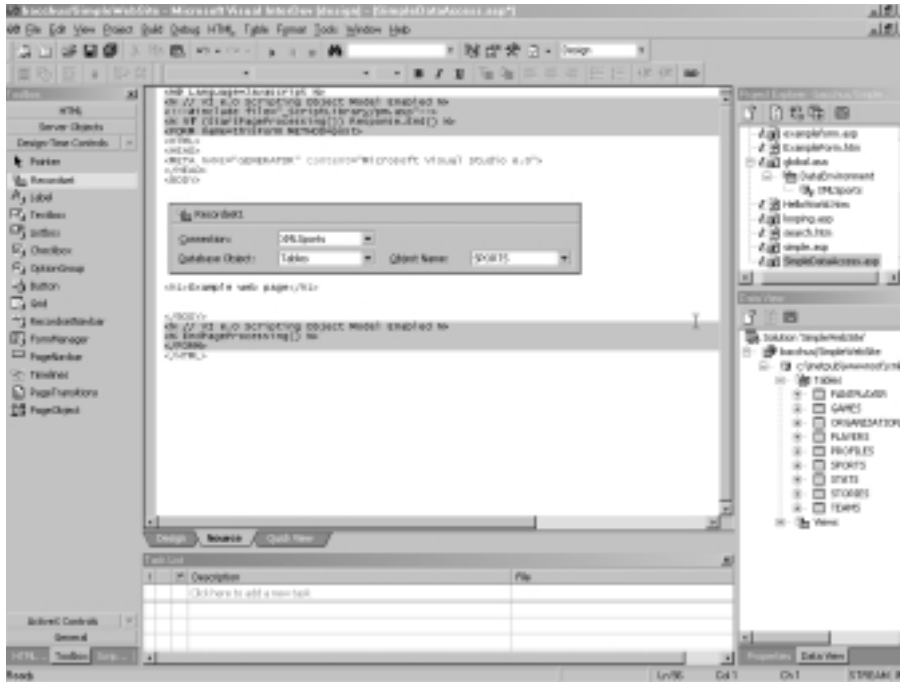

Abbildung 7.1 DTC-Schaubild

Entwurfszeitsteuerelemente (Design-Time Controls, DTC) gibt es seit Visual InterDev 1.0. Ein DTC unterscheidet sich von einem regulären COM-Steuerelement

dadurch, dass es nur zur Entwurfszeit eingesetzt werden kann. Am einfachsten kann man sich diese Steuerelemente als wieder verwendbare Assistenten vorstellen, über die Code generiert wird. Das DTC weist eine Reihe von Parametern auf, die durch den Programmierer eingestellt werden, der den Code generiert. Der Code wird beim Speichern der HTML-Seite erzeugt.

Das DTC ist das Element in der Mitte von Abbildung 7.1. Die Parameter lauten **Connection** (eine Datenverbindung), **Database object** (eine Tabelle, Sicht oder eine gespeicherte Prozedur) und **Object name**, der sich nach dem Datenbankobjekt richtet (SQL-Tabelle **Sports**). Beim Speichern der Webseite wird der folgende Code generiert:

```
<!--METADATA TYPE="DesignerControl" startspan
<OBJECT classid="clsid:9CF5D7C2-EC10-11D0-9862-0000F8027CA0" id=Recordset1
style="LEFT: 0px; TOP: 0px" VIEWASTEXT>
   <PARAM NAME="ExtentX" VALUE="12197">
   <PARAM NAME="ExtentY" VALUE="2090">
   <PARAM NAME="State"
VALUE="(TCConn=\qXMLSports\q,TCDBObject=\qTables\q,TCDBObjectName=\qSPORTS\
q,TCControlID_Unmatched=\qRecordset1\q,TCPPConn=\qXMLSports\q,RCDBObject=\q
RCDBObject\q,TCPPDBObject=\qTables\q,TCPPDBObjectName=\qSPORTS\q,TCCursor-
Type=\q3\s-\sStatic\q,TCCursorLocation=\q3\s-\sUse\sclient-side\scur-
sors\q,TCLockType=\q3\s-
\sOptimistic\q,TCCacheSize_Unmatched=\q10\q,TCCommTimeout_Unmatched=\q10\q,
CCPrepared=0,CCAllRecords=1,TCNRecords_Unmatched=\q10\q,TCODBCSyntax_Unmatc
hed=\q\q,TCHTargetPlatform=\q\q,TCHTargetBrowser_Unmatched=\qServer\s(ASP)\
q,TCTargetPlatform=\qInherit\sfrom\spage\q,RCCache=\qRCBookPage\q,CCO-
pen=1,GCParameters=(Rows=0))"></OBJECT>
-->
<!--#INCLUDE FILE="_ScriptLibrary/Recordset.ASP"-->
<SCRIPT LANGUAGE="Javascript" RUNAT="server">
…
function _Recordset1_dtor()
{
   Recordset1._preserveState();
   thisPage.setState('pb_Recordset1', Recordset1.getBookmark());
}
</SCRIPT>
<!--METADATA TYPE="DesignerControl" endspan-->
```

Der DTC-Code wird innerhalb einer HTML-Kommentierung gespeichert, **<!--METADATA TYPE="DesignerControl" startspan**. Diese Kommentierung sowie das schließende **<METADATA>**-Tag mit **endspan** werden vom DTC-Host dazu eingesetzt, die DTC-Grenzen zu ermitteln. Jeglicher Code innerhalb dieser zwei Grenzmarkierungen sollte nicht manuell bearbeitet werden. Wenn Sie den Code manuell bearbeiten, ändern Sie nur die DTC-Eigenschaften. Das Bearbeiten des generierten Abschnitts resultiert in Änderungen, die verloren gehen, wenn das DTC seinen Inhalt neu generiert.

Der Text zwischen den **<METADATA>**-Tags wird in zwei Abschnitte unterteilt. Der erste Teil umfasst die DTC-Objektidentifizierung sowie die verknüpften Parameter, der zweite Teil enthält den generierten Text. Beachten Sie, dass die DTC-Informationen in einer umfangreichen Kommentierung gespeichert werden. So wird sichergestellt, dass der Inhalt nach Erhalt durch den Client ignoriert und nicht verarbeitet wird. Zur Einsparung von Bandbreite filtert IIS diese Kommentarabschnitte heraus. Der Abschnitt zur Objektidentifizierung wird mit Hilfe der Tags **OBJECT** und **PARAM** definiert. Hier wurde der generierte Text abgekürzt, da es sich lediglich um ein Beispiel handelt.

Verwenden der Bibliothek

Denken Sie daran, dass ASP vom Seitenanfang zum Seitenende ausgeführt wird. Die Skriptobjektbibliothek enthält Ereignisse für das Aufrufen und Verlassen einer Seite. Das Ereignis für den Seitenaufruf wird aufgerufen, um zu kennzeichnen, dass die Seitenverarbeitung beginnt und somit erforderliche Initialisierungen auf Seitenebene durchgeführt werden sollten. Das Ereignis zum Verlassen der Seite wird aufgerufen, um zu kennzeichnen, dass die Seitenverarbeitung beendet ist. Dieses Ereignis ermöglicht das Schreiben von Bereinigungs- oder Zerstörungsroutinen.

Das folgende Beispiel zeigt die Implementierung dieser Ereignisse:

```
<%@ Language=JavaScript %>
<% // VI 6.0 Scripting Object Model Enabled %>
<!--#include file="../_ScriptLibrary/pm.asp"-->
<% if (StartPageProcessing()) Response.End() %>
<FORM name=thisForm METHOD=post>
<HTML>
<HEAD>
</HEAD>
<BODY>
<script runat=server language=javascript>
```

```
function thisPage_onenter() {
   // Do something
   Response.Write("<h1>Starting the page</h1>");
}
function thisPage_onexit() {
   Response.Write("<h1>Ending the page</h1>");
}
</script>
<b>Some page content</b>
</BODY>
<% // VI 6.0 Scripting Object Model Enabled %>
<% EndPageProcessing() %>
</FORM>
</HTML>
```

Abbildung 7.2 Einfache Skriptausgabe

Das Schaubild scheint die richtige Ausgabe anzuzeigen. Jeder Textbestandteil befindet sich dort, wo er sein sollte. Sehen Sie sich jedoch den folgenden HTML-Quellcode an, der durch die Funktion **thisPage_onenter** generiert wurde:

```
<h1>Starting the page</h1><SCRIPT LANGUAGE=JavaScript SRC="/SimpleWebSite/
_ScriptLibrary/pm.js"></SCRIPT>
<SCRIPT LANGUAGE=JavaScript>thisPage._location = "/SimpleWebSite/scripting-
lib/simple.asp";</SCRIPT>
<FORM name=thisForm METHOD=post>
```

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<b>Some page content</b>
</BODY>
<h1>Ending the page</h1>
<INPUT type=hidden name="_method">
<INPUT type=hidden name="_thisPage_state" value="">
</FORM>
</HTML>
```

Dieser HTML-Quellcode verletzt sämtliche HTML-Richtlinien. Normalerweise beginnt eine HTML-Seite mit dem **<HTML>**-Tag, diese Seite beginnt jedoch mit dem **<h1>**-Tag. Warum? Weil hier das **onenter**-Ereignis aufgerufen wird, bevor Text in den Stream geschrieben wurde. Der Zweck dieses Ereignisses besteht darin, die Daten und Objekte auf der Seite zu initialisieren.

Ein weiterer Fehler auf dieser Seite besteht darin, dass das **<FORM>**-Tag dem **<HTML>**-Tag vorangeht. Visual InterDev generiert den HTML-Code in dieser Form, wenn die Skriptobjektbibliothek auf der ASP-Seite aktiviert ist. Deaktivieren Sie (wie zuvor beschrieben) die abgeblendeten Bereiche und verschieben Sie das **<FORM>**-Tag, um diesen Fehler zu beheben. Zur erneuten Aktivierung der abgeblendeten Bereiche muss das **<HTML>**-Tag verschoben werden, durch das die abgeblendeten Bereiche deaktiviert werden.

Der erste Schritt umfasst die Deklaration der verwendeten Programmiersprache. In diesem Fall wurde JavaScript verwendet. Die folgenden Zeilen sind wichtig, da mit ihnen die Skriptobjektbibliothek initialisiert wird. Fügen Sie diese Zeilen mit Hilfe der Visual InterDev-Umgebung ein, werden die Zeilen abgeblendet angezeigt und können nicht bearbeitet werden.

Wenn Sie abgeblendete Zeilen bearbeiten möchten, müssen Sie lediglich in der zweiten Zeile der ASP-Seite Inhalt einfügen, wie nachfolgend gezeigt:

```
<%@Language=JavaScript%>
<HTML>
<%//VI 6.0 Scripting Object Model Enabled%>
```

Speichern Sie die Seite, und laden Sie sie erneut. Es sind keine abgeblendeten Bereiche mehr vorhanden. Der Nachteil ist, dass Sie keine DTCs verwenden können, die die Skriptobjektbibliothek erfordern, da Visual InterDev davon ausgeht, dass

die Skriptobjektbibliothek nicht aktiviert ist. Obwohl dies nicht der Fall ist, muss der gesamte Code manuell eingegeben werden.

Die **<!-- include**-Anweisung umfasst die Kernelemente der Skriptobjektbibliothek. Die nächste Zeile enthält die Funktion **StartPageProcessing**, durch die die Objektbibliotek initialisiert und ein **thisPage**-Objekt erstellt wird. Direkt danach folgt eine Zeile mit dem **<FORM>**-Tag; dieses wird später besprochen.

Das »thisPage«-Objekt Die Skriptobjektbibliothek erstellt anhand des **thisPage**-Objekts eine ereignisgesteuerte Architektur. Das **thisPage**-Objekt stellt eine Instanziierung des JavaScript-Objekts **_SOMObject** dar. Dieses Objekt verfügt über viele Fähigkeiten und ist für die Ereignisverwaltung sowie die Persistenz der serverseitigen HTML-Seite verantwortlich. Die Ereignisse **onenter** und **onexit** besitzen keine Implementierungen, können jedoch unter Verwendung der folgenden Benennungskonvention auf der ASP-Seite implementiert werden:

```
[NamedesObjekts]_[Ereignisname]
```

Im Falle **onenter** führt dies zu den Funktionen **thisPage_onenter** und **thisPage_onexit**. Bei Ausführung der Funktion **StartPageProcessing** werden **thisPage_onenter** und **thisPage_onexit** an die dynamisch referenzierte Funktionstabelle (Dispatchtabelle) gebunden und aufgerufen. Die Tabelle ist Teil des Ereignismechanismus im **thisPage**-Objekt.

Im vorangegangenen Beispiel wird über die Funktion **thisPage_onenter** unter Verwendung von **Response.Write** Text in den Stream geschrieben (vergleichbar mit **thisPage_onexit**). Bei Anzeige der Seite würde die generierte Quelle der in Abbildung 7.2 gezeigten Seite ähneln.

Persistenz der Daten In den vorangegangenen HTML-Quellen waren einige **FORM**- und **INPUT**-Tags vorhanden, deren **type**-Attribut auf **hidden** eingestellt war. Diese Felder werden durch die Skriptobjektbibliothek generiert. Obwohl Cookies nicht so problematisch sind, wie viele Benutzer glauben, gibt es Benutzer, die keine Cookies verwenden möchten oder können. Verborgene Formularvariablen ermöglichen eine Statuserstellung, beispielsweise über die **Session**-Variable, allerdings wird diese vom Server zum Client und zurückgesendet. All dies wird ohne die Verwendung von Cookies erreicht.

Verborgene Formularvariablen können mit Hilfe von **_SOMObject** erstellt werden. Das **_SOMObject** speichert oder lädt Variablen, die mit **thisPage**-Objekt verknüpft sind. Sehen Sie sich das folgende Codefragment an:

```
thisPage.setState( "myObject", "somevalue");
```

Beim Aufruf der **EndPageProcessing**-Funktion wird das folgende HTML-Textfragment generiert:

```
<INPUT type=hidden name="_thisPage_state" value="myObject=somevalue">
```

Wird das Formular auf der Seite an eine andere ASP-Seite übermittelt, wird der Status anhand der **StartPageProcessing**-Methode neu erstellt. Die Werte werden durch den folgenden Code abgerufen:

```
thisPage.getState( "myObject");
```

JavaScript-Objekte Bei der Erstellung von JavaScript-Objekten ruft die **StartPageProcessing**-Funktion sämtliche Erstellungsroutinen für JavaScript-Objekte auf, die Funktion **EndPageProcessing** sorgt für das Aufrufen aller Zerstörungsroutinen für JavaScript-Objekte. Dies wirft die Frage auf, ob derartige JavaScript-Erstellungs- und -Zerstörungsroutinen vorhanden sind. Es sind JavaScript-Erstellungsroutinen vorhanden, wenn die **new**-Anweisung aufgerufen wird, es gibt jedoch keine instanzenspezifische Erstellungsroutine, die generisch aufgerufen werden kann. Ebenso ist auch keine instanzenspezifische Zerstörungsroutine vorhanden. Die generischen Erstellungs- und Zerstörungsroutinen werden durch Verwenden einer Benennungskonvention in der Skriptobjektbibliothek emuliert.

Für die Erstellungsroutine gilt:

```
[Objektname]_ctor
```

Die Benennungskonvention für die Zerstörungsroutine lautet folgendermaßen:

```
[Objektname]_dtor
```

Werden die Funktionen **StartPageProcessing** und **EndPageProcessing** ausgeführt, wird nach Funktionen gesucht, die auf **_ctor** oder **_dtor** enden. Sind derartige Funktionen vorhanden, werden sie aufgerufen.

Zugreifen auf die Datenbank

Die Skriptobjektbibliothek umfasst eine erweiterte Datenbankunterstützung. Hierbei wird ADO (Active Data Objects) eingesetzt, das Modell wird jedoch um Funktionen wie die Navigation auf Eintragsbasis erweitert. Nachfolgend soll untersucht werden, wie mit Hilfe der Skriptobjektbibliothek auf die Datenbank zugegriffen wird.

Einrichten der Datenumgebung In der Skriptobjektbibliothek befindet sich ein Objekt mit dem Namen **Recordset**. Verwechseln Sie dieses nicht mit dem ADO-**Recordset**. Obwohl beide Recordsets die gleiche Funktion erfüllen, bezieht

sich die vorliegende Erläuterung auf das **Recordset**-Objekt der Skriptobjektbibliothek.

Zur Verwendung des **Recordset**-Objekts müssen Sie über eine geeignete Objekt- und Datenbankdefinition in der Datei **global.asa** verfügen, beispielsweise:

```
Application("XMLSports_ConnectionString") = "DSN=xmlsports;DBQ=c:\inet-
pub\wwwroot\xmlsports\database\xml_sports.mdb;DriverId=281;FIL=MS
Access;MaxBufferSize=2048;PageTimeout=5;"
Application("XMLSports_ConnectionTimeout") = 15
Application("XMLSports_CommandTimeout") = 30
Application("XMLSports_CursorLocation") = 3
Application("XMLSports_RuntimeUserName") = ""
Application("XMLSports_RuntimePassword") = ""
'-- Project Data Environment
Set DE = Server.CreateObject("DERuntime.DERuntime")
Application("DE") = DE.Load(Server.MapPath("Global.ASA"), "_private/DataEn-
vironment/DataEnvironment.asa")
```

Der erste Satz **Application**-Definitionen definiert eine OLE DB-Datenbankverbindung (Object Linking and Embedding Database). Das **DE**-Objekt ist dagegen komplexer. In Visual InterDev können Sie Datenverbindungen und -befehle durch Verwendung der kontextabhängigen Menüs erstellen, die mit der Datei **global.asa** verknüpft sind. Über diese Operationen wird die Datenumgebung erstellt, die in einer Datei gespeichert wird. Im vorangegangenen Beispiel findet die Speicherung in der Datei **DataEnvironment** statt. Bei Ausführung der Webanwendung liest das **DERuntime.DEruntime**-COM-Objekt die Datenumgebung und richtet (bei Bedarf) die Datenbefehle und Datenverbindungen ein.

Wurden die Datenbefehle aus der Datei mit der Datenumgebung gelesen, werden diese mit der Datei **global.asa** verknüpft. So wird ein zentraler Standort für die Referenzierung der verschiedenen Datenbefehle erstellt. Durch diese Zentralisierung können die Datenbefehle in verschiedenen HTML-Seiten wieder verwendet werden. Diese Eigenschaft gehört zur Komponentenerstellung.

Erstellen eines Recordsets Üblicherweise möchten Sie in einer ASP-Seite Daten anzeigen. Erstellen Sie hierzu **Recordset**-Objekte. Über die Objekte können die einzelnen Felder angezeigt werden. Sind mehrere Einträge vorhanden, muss durch das **Recordset** navigiert werden.

Das Erstellen eines **Recordset**-Objekts oder anderer Objekte in der Skriptobjektbibliothek ähnelt nicht dem Erstellen von COM-Objekten. In der Skriptobjektbibliothek muss zunächst die Erstellungsroutine aufgerufen werden. Zum Erstellen

von **Recordset1**, einer Instanz von **Recordset**, muss beispielweise die folgende Erstellungsroutine erstellt werden:

```
function _Recordset1_ctor() {
   CreateRecordset('Recordset1', _initRecordset1, null);
}
```

Der Name von **Recordset1_ctor** basiert auf einer Benennungskonvention. Die Funktion könnte auch die Erstellung mehrerer Objekte umfassen. Der Funktionsaufruf **CreateRecordset** führt zur Instanziierung des **Recordset**-Objekts. Der erste Parameter ist der mit dem **Recordset**-Objekt verknüpfte Name, der zweite Parameter gibt die Initialisierungsfunktion für das **Recordset1**-Objekt an. Der letzte Parameter wird aktuell nicht verwendet.

Die Initialisierungsroutine sorgt für das Verknüpfen von **Data Command** und **Recordset**. Nachfolgend ein Beispiel für die Initialisierungsroutine:

```
function _initRecordset1() {
   thisPage.createDE();
   var rsTmp = DE.Recordsets('tabGames');
   Recordset1.setRecordSource(rsTmp);
   Recordset1.open();
   if (thisPage.getState('pb_Recordset1') != null)
      Recordset1.setBookmark(thisPage.getState('pb_Recordset1'));
}
```

Über die Initialisierungsroutine wird eine Instanz der **DE**-Datenumgebung erstellt. Der Methodenaufruf **DE.Recordsets** ruft **Data Command tabGames** ab. Das zurückgegebene Objekt ist ein ADO-**recordset**-Objekt, das anschließend mit dem JavaScript-**Recordset**-Objekt verknüpft wird. Navigiert der Benutzer durch den Datensatz, wird der derzeitige Recordsetstandort abgerufen. Die aktuelle **Recordset**-Standortvariable wird mit Hilfe von **thisPage.getState** als verborgenes Formularelement gespeichert. Eine Variable ohne Nullwert gibt ein Lesezeichen (Bookmark) an, das zum Setzen des Cursors verwendet wird.

Die Zerstörungsroutine speichert die aktuelle Cursorposition in einer verborgenen Formularvariable. Ein Beispiel lautet folgendermaßen:

```
function _Recordset1_dtor(){
   Recordset1._preserveState();
   thisPage.setState('pb_Recordset1', Recordset1.getBookmark());
}
```

Anzeigen von Feldinformationen Es gibt zwei Möglichkeiten, Felder aus der Datenbank anzuzeigen.

Der klassische Ansatz besteht darin, das Feld aus dem **Recordset**-Objekt abzurufen:

```
Recordset1.fields.getValue("GAMEID")';
```

Die Skriptobjektbibliothek bietet eine zusätzliche Möglichkeit zur Kapselung der verschiedenen HTML-Elemente als Objekte – die gekapselten Objekte können an ein **Recordset** gebunden werden. Beim Generieren des HTML-Elements kann dieses über den kleinsten gemeinsamen Nenner, ein HTML-Element, oder ein DHTML-Element gebunden werden. Wie das **Recordset**-Objekt weisen die HTML-Elemente Erstellungs- und Initialisierungsroutinen auf. Zur Anzeige von Elementen müssen keine Daten gespeichert werden, daher ist eine Zerstörungsroutine nicht erforderlich.

Das Format der Erstellungsroutine gleicht dem **Recordset**-Format. Die Initialisierungsroutine für ein HTML-Textfeld lautet folgendermaßen:

```
function _initTextbox1() {
   Textbox1.setStyle(TXT_TEXTBOX);
   Textbox1.setDataSource(Recordset1);
   Textbox1.setDataField('GAMEID');
   Textbox1.setMaxLength(20);
   Textbox1.setColumnCount(20);
}
```

In HTML kann ein Textfeld als Bereichstextfeld, als Kennworttextfeld oder als normales Textfeld definiert werden. In diesem Beispiel wird das normale **TXT_TEXTBOX** verwendet. Anschließend werden Datenquelle und Feld definiert. Die Datenquelle muss ein **Recordset**-Objekt sein, das auf der Seite instanziiert wurde. In unserem Beispiel ist dies **Recordset1**. Das interessante Feld hierbei ist **GAMEID**. Abschließend werden Textfeldlänge (**setMaxLength**) und Spaltenanzahl (**setColumCount**) definiert.

Zur Anzeige des Textfeldes wird folgender Methodenaufruf ausgeführt:

```
<% Textbox1.display(); %>
```

Durchsuchen der Daten Zum Durchsuchen der Daten wird das **Recordset-Navbar**-Objekt eingesetzt. Mit diesem Objekt werden verschiedene Schaltflächen zur Definition des Navigationsschemas erstellt. Es können verschiedene Kombinationen von Navigationsschaltflächen erstellt werden, beispielsweise für die reine Textsuche, das Springen zum nächsten oder vorherigen Eintrag usw. Die er-

stellten Schaltflächen sind JavaScript-**Button**-Objekte. Zum Durchsuchen eines Datensatzes muss die Navigationsleiste mit einem **Recordset**-Objekt verknüpft sein. Die Vorlage wird in der Initialisierungsroutine **RecordsetNavbar** definiert. Eine Beispieldefinition lautet folgendermaßen:

```
function _initRecordsetNavbar1() {
   RecordsetNavbar1.setAlignment(RSNB_ALIGN_HORIZONTAL);
   RecordsetNavbar1.setButtonStyles(RSNB_MASK_FIRSTCAPTION |
      RSNB_MASK_PREVCAPTION | RSNB_MASK_NEXTCAPTION
      RSNB_MASK_LASTCAPTION);

   RecordsetNavbar1.updateOnMove = false;
   RecordsetNavbar1.setDataSource(Recordset1);
   RecordsetNavbar1.getButton(0).value = ' |< ';
   RecordsetNavbar1.getButton(1).value = ' < ';
   RecordsetNavbar1.getButton(2).value = ' > ';
   RecordsetNavbar1.getButton(3).value = ' >| ';
}
```

Der Methodenaufruf **setAlignment** definiert, ob die Schaltflächen horizontal (**RSNB_ALIGN_HORIZONTAL**) oder vertikal (**RSNB_ALIGN_VERTICAL**) ausgerichtet werden. Über den Methodenaufruf **setButtonStyle** wird festgelegt, welche Schaltflächen erstellt werden und welche Vorlage hierbei verwendet wird. Als Vorlage kann entweder eine Grafik oder ein Textelement dienen. Im vorliegenden Fall basieren alle Schaltflächen auf Grafiken und werden mit Hilfe einer **or**-Operation assembliert. Die numerischen Werte der Schaltflächenvorlage lauten:

```
RSNB_MASK_FIRSTIMAGE = 1;
RSNB_MASK_FIRSTCAPTION = 2;
RSNB_MASK_PREVIMAGE = 4;
RSNB_MASK_PREVCAPTION = 8;
RSNB_MASK_NEXTIMAGE = 16;
RSNB_MASK_NEXTCAPTION = 32;
RSNB_MASK_LASTIMAGE = 64;
RSNB_MASK_LASTCAPTION = 128;
```

Unzureichende Dokumentation der Skriptobjektbibliothek Zum Zeitpunkt der Enstehung dieses Buches lag nur eine unzureichende Dokumentation der Skriptobjektbibliothek vor. Viele Entwickler wussten noch nicht einmal etwas von der Existenz dieser Bibliothek. Die Frage lautet: Wie kann ein Entwickler die Verwendung der Bibliothek ohne Dokumentation erlernen? Es gibt zur Verwendung der Skriptobjektbibliothek einige Tipps.

- Sehen Sie sich den Quellcode an. Für die Skriptobjektbibliothek steht der gesamte Quellcode zur Verfügung, da dieser in JavaScript geschrieben wurde, auch wenn gelegentlich einige externe Objekte (z. B. DE-Datenbankobjekte) verwendet werden. Dies erfordert eine umfassende Erfahrung im Umgang mit JavaScript. Die Skriptobjektbibliothek wird unter dem **_ScriptLibrary**-Ordner eines Visual InterDev-Projekts installiert.
- Die Bibliothek ist sehr gut strukturiert und leicht verständlich. Die Entwickler der Bibliothek haben diesbezüglich gute Arbeit geleistet.
- Jede der Bibliotheksdateien stellt ein einzelnes Objekt dar.
- Die Erstellungsroutine bzw. **_Prototype**-Funktion des Objekts enthält die verschiedenen generierten Ereignisse, die vom Objekt unterstützten Methoden sowie die Konstanten, die in den verschiedenen Methodenaufrufen eingesetzt werden.

7.3 Umsetzung

Nachfolgend entwickeln Sie unter Berücksichtigung der in diesem und den letzten zwei Kapiteln vorgestellten Konzepte einen Teil einer Anwendung.

7.3.1 Wo beginnen?

Eine Webanwendung sollte die folgenden Merkmale aufweisen:

- Das Dokument wird in verschiedene Abschnitte unterteilt, z. B. hinsichtlich Aussehen und Funktionalität.
- Das Teilsystem sollte aus Gründen der Wiederverwendbarkeit objektorientiert sein, beispielsweise durch Verwenden von JavaScript-Objekten.
- Die Anwendung sollte den niedrigsten Browserstandard (auch DOM-fähige Browser) unterstützen.
- Für Standardelemente, z. B. Schaltflächen und Textfelder, sollten Skriptobjekte verwendet werden.
- Die Anwendung sollte eine eigenständige Einheit darstellen, die in jeder Situation einsetzbar ist.

Sehen Sie sich nun unter Berücksichtigung dieser Punkte den Domänenmodelltext an.

> *Sicherheit der Website für die Konferenzanmeldung*
>
> *Das Sicherheitsteilsystem muss Funktionen für die An- und Abmeldung bieten. Ein Sicherheitsteilsystem ist erforderlich, da einige der Informationen vertraulich sind und nicht jedem Benutzer zur Verfügung stehen sollten. Hierzu zählen beispiels-*

weise Kreditkarten- und Adressinformationen. Das Teilsystem muss daher auf bestimmte Seiten angewendet werden. Es darf nicht möglich sein, das Sicherheitssystem zu umgehen. Jede vertrauliche Seite muss mit einer Sicherheitsprüfung versehen sein. Die zwischen Client und Server gesendeten Daten sollten nur zeit- und sitzungsbegrenzt gespeichert werden. So kann der Benutzer kein Lesezeichen für die Seite einrichten, d.h., Kollegen oder andere Benutzer sind nicht in der Lage, die Sicherheitsinformationen anderer für den Websitezugriff einzusetzen. Nicht allen Benutzern sollten sämtliche Informationen zur Verfügung stehen. Die angezeigten Informationen richten sich nach dem Prinzip »nur so viele Informationen wie nötig«. Die Anmeldung erfolgt anhand eines Standardformulars, das übermittelt und validiert wird.

Welche Anwendungsfälle und Kollaborationsszenarien sind verfügbar? Die Antwort ist einfach. Stellen Sie sich eine einfache Seite zur Eingabe der Benutzerinformationen vor. Der gesamte Eingabevorgang kann über mehrere HTML-Seiten verteilt werden, und jede HTML-Seite benötigt eine Sicherheitsbeschreibung. Das Kollaborationsszenario kann folgendermaßen aussehen:

1. Anfordern eines Adressdokuments
2. Ist der Benutzer bekannt?
3. Falls nein, Abruf der Anmeldeseite und Rückgabe an Benutzer
4. Festhalten des aktuellen Prozessstatus
5. Überprüfen des Sicherheitsstatus nach Übertragung der Anmeldeseite
6. Falls Anmeldung nicht erfolgreich, zurück zu Schritt 3

7.3.2 Entwerfen und Implementieren der einzelnen Abschnitte

Sehen Sie sich zum besseren Verständnis des Vorgangs die folgende, als vertraulich eingestufte HTML-Seite an:

```
<%@ Language=JavaScript %>
<HTML><HEAD></HEAD>
<BODY>
<% // VI 6.0 Scripting Object Model Enabled %>
<!--#include file="../_ScriptLibrary/pm.asp"-->
<% if (StartPageProcessing()) Response.End() %>
<FORM name=thisForm METHOD=post>
<!-- Log on the user functionality here-->
<H1>Sensitive Content</h1>
If you can see this then you have been logged on
<% // VI 6.0 Scripting Object Model Enabled %>
```

```
<% EndPageProcessing() %>
</FORM>
</BODY></HTML>
```

Auf diese Seite wurde das Framework der Skriptobjektbibliothek angewendet. Des Weiteren ist ein Kommentar vorhanden, der angibt, wo die Anmeldefunktionalität eingefügt werden sollte. Diese Funktionalität sollte implementiert werden, bevor vertraulicher Inhalt angezeigt werden kann.

Als Verwender von C++ und Visual Basic war mein erster Gedanke, den folgenden Quellcode zu schreiben:

```
<%if( user.isLoggedOn() == false) {
Response.Redirect("../logon.asp");
} else {%>
<H1>Sensitive Content</h1>
If you can see this then you have been logged on
<%}%>
```

Es ist ein Benutzerobjekt vorhanden, über das geprüft wird, ob der aktuelle Benutzer angemeldet ist (**isLoggedOn**). Wird der Wert **false** zurückgegeben, wird die aktuelle Seite an die Anmeldeseite umgeleitet. Andernfalls könnte der vertrauliche Inhalt angezeigt werden.

Dieser Ansatz birgt viele Probleme:

- Warum ist **user** ein Objekt? In ASP verfügt jeder **user** über einen eigenen Arbeitsbereich. Es ist nicht erforderlich, ausdrücklich ein **user**-Objekt zu erstellen. Stattdessen ist es angemessener, ein **security**-Objekt zu erstellen. Die Sicherheit wird auf den Benutzer angewendet.
- Ist der Benutzer nicht angemeldet, wird die Seite umgeleitet. Auf diese Weise wird der Anmeldevorgang gestartet, aber wie kehren Sie zur ursprünglich angeforderten Seite zurück? Diese Methode erfordert, dass der angemeldete Benutzer einen Extraklick ausführen muss. Die Webanwendung sollte »wissen«, welche Seite ursprünglich aufgerufen wurde.
- Der hier geschriebene Code ist nicht wieder verwendbar und erfordert einige Tipparbeit. Es wäre einfacher, wenn die Lösung weniger Codezeilen umfassen würde. Damit würde auch die Wartung vereinfacht.

Die Lösung besteht darin, über die Funktionsweise von HTML und Skriptobjekten nachzudenken. Beim Entwickeln einer Benutzerschnittstelle können JavaScript-Objekte erstellt werden, die sämtlichen Code für die Benutzerschnittstelle enthalten. Dies bringt uns jedoch zum ursprünglichen Problem nicht dynamischer Be-

nutzerschnittstellen zurück. Eine Lösung sollte die Benutzerschnittstelle für das Formular von der Funktionalität trennen.

Skizzieren der Lösung

In der Lösung werden die Statusverwaltungsfunktionen der Skriptobjektbibliothek sowie die ASP-**include**-Anweisungen zur Durchführung einer Sicherheitsprüfung verwendet. Es ist nicht nötig, eine Umleitung zu einer anderen HTML-Seite durchzuführen, wodurch der zusätzliche Schritt der »Speicherung« der ursprünglichen Seite entfällt.

Jede Seite mit vertraulichen Informationen umfasst zwei Elemente. Das erste Element ist die Sicherheit, das zweite ist der Inhalt selbst. In ASP wird zunächst die Sicherheit einer Seite verarbeitet. Besagen die Sicherheitsrichtlinien, dass eine Anmelderoutine ausgeführt werden muss, werden die entsprechenden HTML-Elemente in die Seite eingefügt. Das zweite Element erkennt die vorgenommenen Sicherheitsmaßnahmen und reagiert entsprechend. Wurde eine Anmelderoutine hinzugefügt, wird kein Inhalt generiert.

Dem Endbenutzer wird ein Formular angezeigt. Der Benutzer füllt das Formular aus und übermittelt es an den Server. Die URL ist immer noch die ursprünglich eingegebene, und der Prozess der Sicherheitsprüfung beginnt von vorn. Die Sicherheitsprüfung erkennt, dass die in das Anmeldeformular eingegebenen Informationen verarbeitet werden müssen.

Verläuft die Verarbeitung der Formulardaten erfolgreich, werden über das Inhaltselement die HTML-Elemente in die Seite eingefügt. Scheitert die Verarbeitung, generiert das Sicherheitselement für einen weiteren Anmeldeversuch erneut die Anmelderoutine.

Benutzerschnittstelle

Wenden wir uns der Implementierung einer vertraulichen Seite zu. Es sollen globale, wieder verwendbare Informationen eingesetzt werden. Die globalen Informationen können auch zusätzliche Informationen enthalten, mit denen Workflowanwendungen umsetzbar sind oder ermittelt werden könnte, wann die Daten aktualisiert werden müssen. Nehmen Sie folgende Implementierung:

```
<%@ Language=JavaScript %>
<HTML><HEAD></HEAD>
<BODY>
<% // VI 6.0 Scripting Object Model Enabled %>
<!--#include file="../_ScriptLibrary/pm.asp"-->
<% if (StartPageProcessing()) Response.End() %>
```

```
<FORM name=thisForm METHOD=post>
<!-- Log on the user functionality here-->
<H1>Sensitive Content</h1>
If you can see this then you have been logged on
<% // VI 6.0 Scripting Object Model Enabled %>
<% EndPageProcessing() %>
</FORM>
</BODY></HTML>
```

Das Dokument ist eine einfache "Hello World"-HTML-Seite. Es enthält jedoch Informationen, die als vertraulich betrachtet werden und muss daher über Anmeldefähigkeiten verfügen. Die **generateDocument**-Variable ist eine globale Variable und kann durch ein beliebiges Teilsystem geändert werden, um anzuzeigen, dass der Inhalt nicht angezeigt werden sollte. Die Anmeldefunktionalität ist in der Datei **logon.asp** enthalten. Der Einschluss erfolgt über die **<!--#include...>**-Anweisung. Auf diese Weise wird die Funktionalität des Sicherheitsteilsystems von der der vertraulichen Seite getrennt.

Üblicherweise wird die **#include**-Anweisung zum Einschließen von Skripts verwendet, in diesem Fall werden jedoch HTML-Tags und Skript eingeschlossen. Zum dynamischen Einschließen von Inhalt werden die Erstellungs- und Zerstörungsroutinen der Skriptobjektbibliothek verwendet. Damit wird der Status des Sicherheitsteilsystems festgelegt, über den wiederum der Inhalt generiert wird.

Nehmen Sie folgende Implementierung:

```
<!--#INCLUDE FILE="../_ScriptLibrary/TextBox.ASP"-->
<!--#INCLUDE FILE="../logon/securitymgr.asp"-->
<script language="javascript" runat="server">
function _page_ctor() {
   scrMgr.logon();
}
</script>
<SCRIPT LANGUAGE=JavaScript RUNAT= Server>
function _inittxtUsername() {
   txtUsername.setStyle( TXT_TEXTBOX);
   txtUsername.setMaxLength(20);
   txtUsername.setColumnCount(20);
}
function _inittxtPassword() {
   txtPassword.setStyle( TXT_PASSWORD);
   txtPassword.setMaxLength(20);
```

```
   txtPassword.setColumnCount(20);
}
function _creators_ctor() {
   CreateTextbox( 'txtUsername', _inittxtUsername, null);
   CreateTextbox( 'txtPassword', _inittxtPassword, null);
}
function thisPage_onshow() {
   scrMgr.username = txtUsername.value;
   scrMgr.password = txtPassword.value;
}
</script>
<%
if( scrMgr.isLoggedOn() == false) {
%>
<SCRIPT LANGUAGE=JavaScript>
function butSubmit_onclick( form) {
   form.submit();
}
</script>
<table>
   <tr>
      <td>Username</td><td><%txtUsername.display();%></td>
   </tr>
      <td>Password</td><td><%txtPassword.display();%></td>
   </tr>
   <tr>
      <td><input type="button" value="Submit"
         name="cmdSubmit" onClick="butSubmit_onclick( this.form);">
      </td>
   </tr>
</table>
<%
   scrMgr.operationStep = STP_LOGGING_ON;
   generateDocument = false;
}
%>
```

Betrachten wir die einzelnen Codeabschnitte. Würde diese Seite in die vorangegangene ASP-Seite eingefügt, werden ggf. die HTML-Anmeldeelemente generiert. Die Seitenerstellungsfunktion **page_ctor** wird aufgerufen. Momentan ist diese

Funktion leer, aber sie kann zur Definition eines beliebigen Status auf Seitenebene eingesetzt werden.

Es ist eine weitere Erstellungsroutine auf dieser Seite vorhanden, **_creators_ctor**. Diese Erstellungsroutine dient der Erzeugung von zwei Textfeldern, **txtUsername** und **txtPassword**. Warum ein Skriptobjekt-Textfeld erstellen, wenn es einfacher sein könnte, das **<INPUT>**-Tag zu verwenden? Es ist zwar weniger aufwendig, das **<INPUT>**-Tag zu verwenden, das Formular muss jedoch verarbeitet werden. Durch das Verwenden eines Skriptobjekt-Textfeldes erfolgen Textgenerierung, Formularverarbeitung und Werteabruf in einem Arbeitsschritt. In einem traditionellen Formular ist dies nicht der Fall. Das Textfeld **txtPassword** ruft zur Definition eines Kennwortfeldes **setStyle** auf. Das Textfeld **txtUsername** ruft zur Definition eines normalen Textfeldes **setStyle** auf. Über beide Textfelder werden Länge und Spaltenanzahl festgelegt.

Die Skriptobjektbibliothek definiert nicht nur serverseitige Schaltflächen und Textfelder. Es können auch clientseitige Schaltflächen und Textfelder verwendet werden. Ein clientseitiges **Button**-Skriptobjekt wurde nicht verwendet, da dies zu komplex wäre. Die Schaltflächen sind einfache **<INPUT type="button" ...>**-Tags mit verknüpften Ereignisbehandlungsroutinen, über die das Formular (**<FORM>**) übermittelt wird.

In dieser Seite wird die Methode **form.submit** aufgerufen, obwohl die Seite kein **<FORM>**-Tag enthält. Das Formular ist Teil der Skriptobjektbibliothek. Die ursprünglich in **logon.asp** eingeschlossene vertrauliche Seite weist eine Zeile mit folgendem **<FORM>**-Tag auf.

```
<FORM name=thisForm METHOD=post>
```

Bei der Formularübermittlung wird auf das oben genannte Formular verwiesen. Es wird nicht empfohlen, eigene Formulare zu verwenden, da über diese die von der Skriptobjektbibliothek verwendeten verborgenen Textfelder nicht übertragen werden. Dies bedeutet, dass der in **Textbox**, **txtUsername** und **txtPassword** erworbene Status nicht weitergegeben wird. Der **<FORM>**-Attributname referenziert jedoch **thisForm**. Was ist **thisForm**? Es stellt einen Teil der clientseitigen Skriptobjektbibliothek dar und definiert die aktuelle URL als **thisForm**. Daher wird beim Aufruf von **form.submit** die aktuelle URL aufgerufen. So wird das Problem gelöst, sich die ursprünglich angeforderte Seite zu »merken«.

Nach Aufruf der Erstellungsroutine wird das Ereignis **thisPage_onshow** aufgerufen. Dieses Ereignis wird generiert, um zu kennzeichnen, dass die Seite angezeigt werden soll und dass Werte, die geändert werden müssen, jetzt geändert werden sollten. In diesem Fall werden die Werte **scrMgr.username** und **scrMgr.password** zugewiesen. Diese Zuweisung ist bei der Generierung einer Anmeldung nicht

wichtig. Bedeutung erhält die Wertezuweisung erst bei der Verarbeitung des Anmeldeformulars.

Abschließend wird die Methode **Mgr.logon** aufgerufen. Aber woher kommt **scrMgr**? Die Antwort finden Sie im nächsten Abschnitt. Für den Moment betrachten wird diese Methode lediglich als verfügbar. Das **scrMgr**-Objekt verwaltet die Sicherheitsattribute des aktuellen Endbenutzers. Anschließend wird **scrMgr.isLoggedOn** aufgerufen, um zu ermitteln, ob der Anmeldeversuch erfolgreich war. Ist dies der Fall, wird die Anmelderoutine generiert und das Flag **generateDocument** wird auf **false** gesetzt. So wird sichergestellt, dass der eingeschlossene Anmeldungsinhalt nicht generiert wird.

Implementieren der Anwendungslogik

Die Anwendungslogik ist in der Datei **../logon/securitymgr.asp** enthalten. Der **securityManager** ist ein globales, lokales Objekt. Auch wenn sich dies widersprüchlich anhört, gibt es dieses Objekt. Beim Einschluss der Datei **securitymgr.asp** enthält diese eine Erstellungs- und eine Zerstörungsroutine. Diese Funktionen erstellen das **scrMgr**-Objekt. Dieses Objekt ist global im Sinne der Anwendung, aber gleichzeitig lokal, da nur der aktuelle Benutzer es sieht. Das Objekt wird immer dann instanziiert, wenn auf eine vertrauliche Seite zugegriffen wird. Die Objektdefinition lautet wie folgt:

```
<SCRIPT RUNAT=SERVER LANGUAGE="JavaScript">
function SecurityMgr() {
   if (typeof(_SecurityMgr_Prototype_called) == 'undefined')
      _SecurityMgr_Prototype();
   this.username = null;
   this.password = null;
   this.securityLevel = "";
   this.operationStep = STP_NOTHING;
}
function _SecurityMgr_Prototype() {
   SecurityMgr.prototype.logon = _SM_logon;
   SecurityMgr.prototype.logoff = _SM_logoff;
   SecurityMgr.prototype.isLoggedOn = _SM_isLoggedOn;
   _SecurityMgr_Prototype_called = true;

   STP_NOTHING = 0;
   STP_LOGGING_ON = 1;
   STP_LOGGED = 2;
}
```

```
function _SM_logon() {
   // Do something to log on the user
   if( this.operationStep == STP_LOGGING_ON) {
      this.operationStep = STP_LOGGED;
   }
}
function _SM_logoff() {
   // Do something to log off the user
}
function _SM_isLoggedOn() {
   if( this.operationStep == STP_LOGGED) {
      return true;
   } else {
      return false;
   }
}
function _SM_ctor() {
   if (typeof( scrMgr) != 'object')
      scrMgr = new SecurityMgr;

   scrMgr.operationStep = thisPage.getState( "operationStep");
}
function _SM_dtor() {
   thisPage.setState( "operationStep", scrMgr.operationStep);
}
</script>
```

Die Funktion **_SM_ctor** weist ein neues **SecurityMgr**-Objekt zu. **scrMgr.operationStep** wird ein globaler Status zugewiesen, d.h., der momentane Schritt im Sicherheitsprozess wird angegeben. Zu Beginn weist **operationStep** den Wert **STP_NOTHING** auf. Beim Aufruf der ersten vertraulichen Seite und während der Verarbeitung des Anmeldeformulars erhält **operationStep** den Wert **STP_LOGGING_ON**. Bei erfolgreicher Anmeldung weist **operationStep** den Wert **STP_LOGGED_ON** auf.

Beachten Sie, dass der Status von **operationStep** unter Verwendung von **thisPage.setStatus** und **thisPage.getStatus** verwaltet wird. Diese Funktionen gehen davon aus, dass sich nur ein **<FORM>**-Tag auf der Seite befindet. Der Status könnte auch in der **Session**-Variablen verwaltet werden. Eine **Session**-Variable sendet keine Daten an den Client, sondern erfordert Cookies. Der Vorteil von

thisPage liegt darin, dass zwar Daten an den Client gesendet werden, Cookies jedoch nicht erforderlich sind.

Interessant ist auch die Verwendung der Funktion **_SecurityMgr_Prototype**, die in der Erstellungsroutine von **SecurityManager** aufgerufen wird. Beim Erstellen einer Methodentabelle in einem Objekt können Standardfunktionen vererbt werden. Hierzu verwenden Sie am besten das Schlüsselwort **Prototype** wie in der Funktion **_SecurityMgr_Prototype**.

7.4 Resümee

Nach diesem Kapitel soll die HTML-Entwicklung nicht weiter behandelt werden, daher einige Tipps aus meiner praktischen Erfahrung als Websiteentwickler:

- Die Anzahl und Vielfältigkeit der verschiedenen Browser ist immens. Es gibt einfache Browser und moderne, sehr leistungsstarke Browser. Das Problem besteht häufig darin, auch die Benutzer zu berücksichtigen, die über keinen leistungsstarken Browser verfügen. Die Website muss alle Browsertypen unterstützen.
  Lösung: Verwenden Sie die Visual InterDev-Skriptobjektbibliothek zum einfachen Schreiben von Komponenten, die auf einem beliebigen Browser eingesetzt werden können.
- Benutzer lesen häufig nicht, was sie eingeben. In früheren Websites wurden 20% der Fehler durch falsch eingegebene E-Mail-Adressen verursacht. Häufig mussten hierbei lediglich Bindestriche entfernt oder Punkte eingefügt werden.
  Lösung: In Kapitel 5 wurde die Erstellung einer Routine für die Formularprüfung erläutert. Verwenden Sie immer eine Routine für die Validierung. Fügen Sie ggf. ein Dialogfeld ein, über das der Benutzer aufgefordert wird, seine Eingaben erneut zu prüfen, da durch falsche Angaben unnötig CPU-Zeit verbraucht wird.
- Ist die Website zu langsam, klicken viele Benutzer ein zweites Mal auf eine Schaltfläche, um ganz sicher zu gehen, dass der Befehl ausgeführt wird.
  Lösung: Es gibt leider keine Lösung für dieses Problem.
- Proxys sind ein Alptraum und können für viele Probleme verantwortlich sein. Ein Proxy verbirgt darüber hinaus den internen Benutzer und macht die IP-Referenzierung sinnlos.
  Lösung: Verwenden Sie formularbasierte Variablen, die den Status zwischen Browser und Server weitergeben.
- Skript-, Java- und COM/ActiveX-Steuerelemente werden häufig herausgefiltert, dies kann sich nachteilig auf die Seitenfunktion auswirken.
  Lösung: Verwenden Sie möglichst viel serverseitigen ASP-Code.

- Benutzer drucken gerne die Einzelheiten zu Themen aus, die sie interessant finden.
  Lösung: Bieten Sie eine druckorientierte HTML-Seite an, bzw. nutzen Sie die browserspezifischen Funktionen.

Das Entwickeln von Webanwendungen stellt keine einfache Aufgabe dar, besonders dann nicht, wenn Sie ein Anfänger sind. Denken Sie daran, dass die Webentwicklung eine völlig andere Welt darstellt. HTML weist eine Vielzahl von Vorteilen auf, die traditionelle Benutzerschnittstellen und Anwendungen nicht bieten. Wenn Sie also mit der Anwendungsentwicklung beginnen, sollten Sie klein anfangen und aus der Erfahrung lernen.

8 Entwerfen von COM-Schnittstellen

Das vorliegende Kapitel beschäftigt sich mit der Komponentenentwicklung. Wie bereits in vorangegangenen Kapiteln erwähnt, sollte die Entwicklung mit Hilfe von Schnittstellen und Implementierungen erfolgen. In diesem Kapitel werden die COM-Schnittstellen (Component Object Model) und die Auswirkungen ihrer Verwendung erläutert. Zunächst erhalten Sie eine Einführung in COM und das Schnittstellenkonzept, anschließend werden die technischen Details von COM und der COM-IDL (Interface Definition Language) besprochen. Des Weiteren werden Sie an die Probleme herangeführt, die bei der Verwendung von COM-Schnittstellen in Visual C++, Visual J++ Visual Basic sowie bei der Skripterstellung auftreten können. Hierbei wird vor allem die Interaktion zwischen den verschiedenen Umgebungen betrachtet.

8.1 Schnittstellen

Das Entwerfen von Anwendungen mit COM-Schnittstellen unterscheidet sich ein wenig von der Erstellung regulärer Klassen in Visual Basic, Visual J++ oder Visual C++. Beim Entwerfen einer Klasse entwerfen Sie auch eine zugehörige Implementierung. Das klassische objektorientierte Beispiel ist die Implementierung von Formen. Bei der klassischen objektorientierten Programmierung (Object-Oriented Programming) wird die Basisklasse **shape** zur Funktionsimplementierung der Formen definiert, beispielsweise Kreise, Quadrate und Rechtecke. Die Basisklasse **shape** enthält eine Funktion, die in den Klassen **square** oder **circle** implementiert werden kann. Unter Verwendung dieser Methode muss ein Konsument von **shape** nichts über die jeweilige Funktionsweise von **square** oder **circle** wissen.

8.1.1 Entwickeln einer Architektur

Die Verfahren der objektorientierten Programmierung gelten für Schnittstellen, jedoch in einem anderen Kontext. Bei der Entwicklung eines Systems ist es wichtig, zunächst die Architektur zu entwerfen. Die Trennung von Schnittstelle und Implementierung ermöglicht dem Designerteam die Betrachtung des Gesamtbildes ohne Berücksichtigung sämtlicher Implementierungsdetails. Sherlock Holmes sagte einmal: »Watson, ich lese keine Nachrichten, da mein Gedächtnis einem Speicher von begrenzter Größe gleicht. Je größer das Durcheinander, desto größer ist auch die Gefahr, dass etwas Wichtiges verloren geht.« Das Gleiche trifft auf

die Entwicklung großer Anwendungen zu. Es kann jeweils nur eine begrenzte Anzahl von Zielen umgesetzt werden. Alles Weitere geht leicht unter.

Das Designerteam ist verantwortlich für das Entwerfen der Kernschnittstellen zur Repräsentation der verschiedenen Anwendungsfälle und Kollaborationsdiagramme in der Anwendung. Wenn Sie das Buch Kapitel für Kapitel gelesen haben, sollten Sie einen Domänenmodelltext verfasst sowie einen Prototyp entwickelt haben, mit dem die verschiedenen technischen Schwierigkeiten untersucht werden. Nun kommen wir zu der Phase, in der bestimmte Aspekte des Objekt- und Programmverhaltens extrahiert werden können.

Gehen Sie pragmatisch vor. Lässt ein Anwendungsfall beispielsweise die Verwendung einer COM-Schnittstelle zu, dann wenden Sie diese an. Versuchen Sie nicht, Schnittstellen zu entwerfen, die für den Anwendungsfall sekundär sind. Obwohl Sie vielleicht denken, dass diese wichtig sind, geraten Sie bei der Implementierungsphase in Schwierigkeiten, da vielleicht noch eine weitere Schnittstelle implementiert werden muss, d.h. es ist noch mehr Arbeit erforderlich. Verfallen Sie jedoch genauso wenig in das entgegengesetzte Extrem, indem Sie fordern, dass jede Schnittstelle mit einem Anwendungsfall verknüpft sein muss. Das Ziel besteht darin, einen Kompromiss zu finden, bei dem die Haupt-COM-Schnittstellen in direktem Zusammenhang mit den verschiedenen Anwendungsfällen und Kollaborationsdiagrammen stehen, einschließlich einiger sekundärer COM-Schnittstellen.

Der Entwicklungsprozess kann in das in Abbildung 8.1 gezeigte Diagramm übertragen werden.

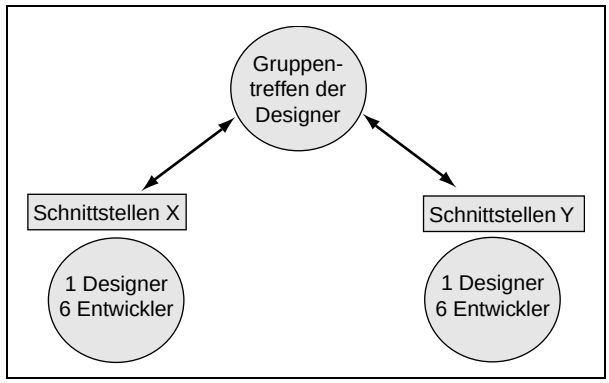

Abbildung 8.1 Teamstruktur mit Designerteam im Mittelpunkt

In Abbildung 8.1 wird ein Entwurfsteam gezeigt, dass sich als Gruppe trifft. Dieses Team entwirft die Schnittstellen für die gesamte Anwendung. Die Schnittstellen

werden dokumentiert und den zwei Teams ausgehändigt. Die einzelnen Teams implementieren die Schnittstellen, und wenn das Designerteam gute Arbeit geleistet hat, passen die einzelnen Elemente zusammen.

Komponenten und das Testen

Bei der Entwicklung von Schnittstellen und Komponenten ist die Erstellung einer Testumgebung unerlässlich. Dieses Thema wird in Kapitel 19 ausführlich erläutert. Das ordnungsgemäßige Testen ist jedoch nicht der einzige Faktor bei der Anwendungsstabilität.

In Abbildung 8.2 sehen Sie ein Entwurfsteam, das zwei Schnittstellen definiert hat, **IAccount** und **IUser**. Diese Schnittstellen werden durch zwei verschiedene Entwicklerteams implementiert. Da alle Teams Teil einer größeren Gruppe sind, kennen Sie einander und treffen sich gelegentlich. Während dieser Treffen spricht einer der Entwickler, der Schnittstelle **IAccount** implementiert hat, mit einem Mitglied des Teams, das Schnittstelle **User** implementiert. Sie stellen fest, dass beide Teams teilweise die gleiche Funktionalität implementiert haben. Zur Verringerung des Entwicklungsaufwands entscheiden beide Teams, eine Komponente **MyCommonObject** zu erstellen, mit denen die Probleme beider Teams behoben werden.

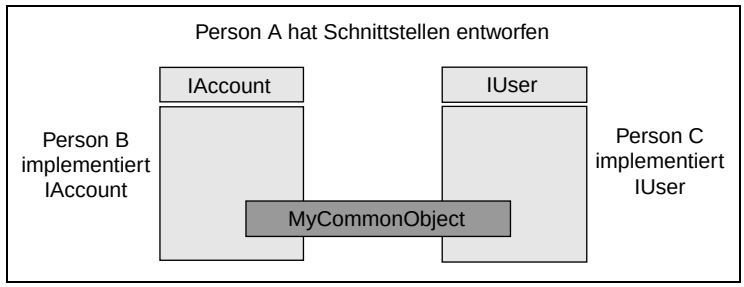

Abbildung 8.2 Entwicklungsdiagramm zur Implementierung der Schnittstellen

Obwohl dies ein guter Ansatz zu sein scheint, werden tatsächlich Codeabschnitte wieder verwendet, ohne dass die Auswirkungen dieses Vorgehens bekannt sind. Zunächst wissen nur die Teammmitglieder über die Codewiederverwendung Bescheid – Designer und Tester wurden nicht informiert. Beim Testen der Anwendung wird Schnittstelle **IAccount** unabhängig von Schnittstelle **IUser** getestet. Angenommen, Schnittstelle **IAccount** wird erfolgreich getestet, Schnittstelle **IUser** muss jedoch umgestaltet werden. Zur Behebung der Bugs wird die gemeinsame Komponente geändert. Die Tester führen lediglich einen erneuten Test für die Schnittstelle **IUser** durch, Schnittstelle **IAccount** muss jedoch auch neu getestet

werden. Demnach kann der zunächst als effektiv erscheinende Ansatz letztlich zu einem Misserfolg führen, und es muss ein Patch erstellt werden.

Die Lösung umfasst keine Wiederverwendung von Komponenten; stattdessen wird Code ausgeschnitten und wieder eingefügt, um zwei Quellcodes zu erhalten. Steht genügend Zeit zur Verfügung, können die Designer die Komponente untersuchen und die Auswirkungen der Komponentenverwendung ermitteln. Anschließend kann die Komponente eventuell wieder verwendet werden. Das Schreiben guter Komponenten ist nicht einfach, und Stabilität ist ein Ergebnis umfassender Testläufe, bei denen die Grenzen der Komponente ermittelt werden.

8.2 Entwickeln einer COM-Komponente

Die COM-Komponente setzt sich aus einer Reihe von Schnittstellen zusammen, die in Klassen und Bibliotheken gruppiert werden. Der Anspruch des Designerteams besteht darin, die Schnittstellen zu entwerfen. Hierzu gibt es keine einfache Lösung. Im Idealfall wird ein UML-Tool (Unified Modeling Language) verwendet, da jedoch die derzeitigen UML-Tools nur wenig Unterstützung für den COM-Schnittstellenentwurf bieten, wird die Verwendung von UML an anderer Stelle erläutert. In einigen Projekten wurden Microsoft Word-Dokumente zur Dokumentation und Erstellung von CORBA- (Common Object Request Broker Architecture, eine vergleichbare Komponententechnologie) und COM-Schnittstellen eingesetzt.

8.2.1 Funktionsweise von COM

Bei COM handelt es sich um eine binäre Technologie, die das Definieren von Komponenten ermöglicht. COM-Komponenten sind wieder verwendbare Black Boxes, die eine spezifische Funktionalität offen legen. COM ist eine leistungsstarke Komponentenarchitektur, da eine COM-Komponente sprachunabhängig arbeitet. Bei Verwendung von COM kann das Hauptaugenmerk auf die Schnittstellen und deren Verwendung gelegt werden.

Die COM-Technologie ist in Ebenen strukturiert. Auf der untersten Ebene befindet sich eine COM-Schnittstelle. Die COM-Schnittstelle stellt eine VTable (Virtual Table) dar, mit der eine Reihe von Funktionszeigern gruppiert wird, die auf die Implementierung der Schnittstelle verweisen. Dieser Bestandteil von COM folgt dem **Bridge-Muster** (siehe Muster im Anhang), da Schnittstelle und Implementierung unabhängig voneinander definiert werden können.

In der COM-Umgebung wird das COM-Objekt durch eine COM-Coklasse beschrieben, mit der die vom COM-Objekt unterstützten Schnittstellen angegeben

werden. Hierbei bezeichnet COM-Coklasse eine COM-Klasse, ein COM-Objekt ist eine Instanz einer COM-Klasse.

Mehrere COM-Klassen können in einer COM-Bibliothek zusammengefasst werden. Dies bezeichne ich als COM-Komponente. Sie können einwenden, dass eine Komponente eine einzelne physische COM-Klasse darstellt, und genau hier gehen die theoretische und die praktische Betrachtungsweise auseinander. Microsoft tendiert dazu, eine einzelne DLL als COM-Komponente zu bezeichnen, die eine COM-Bibliothek darstellt.

Jede COM-Schnittstelle, -Klasse und -Bibliothek muss eindeutig identifiziert werden. Dies wird mit einer **GUID** (Globally Unique Identifier, global eindeutige Kennung) erreicht. Es handelt sich hierbei um einen 128-Bit-Wert, der sich aus einer Gruppe von 8 Hexadezimalzeichen zusammensetzt, gefolgt von drei Gruppen á 4 Hexadezimalzeichen und 1 Gruppe aus 1–12 Hexadezimalzeichen. Eine Beispiel-GUID lautet folgendermaßen:

```
6B29FC40-CA47-1067-B31D-00DD010662DA
```

Diese Kennung wird verwendet, wenn der Konsument eine COM-Klasse, -Schnittstelle oder -Bibliothek aufruft.

8.2.2 IDL und seine Funktion

Die Schnittstelle ist ein Kernelement der COM-Technologie. Schnittstellen werden über IDL (Interface Definition Language) definiert. In der Vergangenheit verwendete Microsoft zwei Methoden zur Schnittstellendefinition, ODL (Object Definition Language) und IDL (Interface Definition Language). Die erste diente der OLE-Automatisierung, zweitere für RPCs (Remote Procedure Calls, Remoteprozeduraufrufe). Eine ausführlichere Beschreibung der OLE-Automatisierung erhalten Sie zu einem späteren Zeitpunkt. Mit der Zeit wurde ODL überflüssig. Es wurde entschieden, IDL sowohl für RPC- als auch COM-Beschreibungen zu verwenden. ODL wird weiterhin unterstützt, aber IDL stellt die empfohlene Methode zur Beschreibung von Schnittstellen dar.

IDL beinhaltet ein grundlegendes Konzept. Jedes bezeichnete Kennwort kann über Attribute verfügen und andere Schlüsselwörter enthalten. Im Folgenden ein Beispiel für die Notation:

```
 [attributes] keyword  label
{
   keyword member descriptions
};
```

oder

```
typedef [ type_attributes] type_keyword label
{
   keyword member descriptions
};
```

Das Schlüsselwort besteht aus einem einzelnen Befehl, beispielsweise **interfaces**, **coclass** oder **library**. In den eckigen Klammern befinden sich die mit dem Schlüsselwort verknüpften Attribute. Beispiele sind **object**, **uuid** und **helpstring**. Die Schlüsselwörter enthalten innerhalb der geschweiften Klammern untergeordnete Elemente.

Ein einfaches Beispiel

Bei der Entwicklung größerer Anwendungen mit vielen COM-Klassen kann man leicht durcheinander geraten. Diese Verwirrung ergibt sich nicht durch die Komplexität der COM-Klassen, sondern durch zu viele Informationen. Sie kombinieren vielleicht COM-Implementierungscode mit COM-Schnittstellenbeschreibungen; wir werden jedoch die COM-Schnittstellen von den Implementierungen trennen. Hierzu erstellen Sie eine IDL-Datei.

Es gibt verschiedene Möglichkeiten zur Verwendung einer IDL-Datei, die gewählte Methode richtet sich nach der verwendeten Programmierumgebung. Visual C++ verfügt über die Fähigkeit, IDL-Dateien systemeigen zu verwenden und in ein Projekt zu integrieren, Visual Basic und Visual J++ dagegen erfordern eine Typbibliothek für die Integration. Zur Erstellung einer Typbibliothek verwenden Sie das MIDL-Programm (Microsoft Interface Definition Language) zur Kompilierung der COM-Bibliothek in eine COM-Typbibliothek. Eine COM-Typbibliothek ist eine kompilierte, binäre Darstellung der IDL-Datei.

Sehen Sie sich die folgende IDL-Datei an:

```
import "oaidl.idl";
import "ocidl.idl";

   [
      object,
      uuid(8895EECD-0915-11D2-9C50-00A0247D759A),
      dual,
      helpstring("ISimpleInterface Interface"),
      pointer_default(unique)
   ]
```

```
   interface ISimpleInterface : IDispatch
   {
      [id(1), helpstring("method method1")] HRESULT method1( long param1);
   };

[
   uuid(8895EEC1-0915-11D2-9C50-00A0247D759A),
   version(1.0),
   helpstring("ServerPackage 1.0 Type Library")
]
library SERVERPACKAGELib
{
   importlib("stdole32.tlb");
   importlib("stdole2.tlb");

   [
      uuid(8895EECE-0915-11D2-9C50-00A0247D759A),
      helpstring("SimpleUser Class")
   ]
   coclass SimpleComponent
   {
      [default] interface ISimpleInterface;
   };
};
```

Bei der Visual Studio Enterprise Edition muss die Datei nicht unter Verwendung von MIDL manuell in eine COM-Typbibliothek kompiliert werden. Die Visual Studio C++-Shell ermöglicht Ihnen das Erstellen eines Dienstprogrammprojekts und das Hinzufügen der IDL-Datei zu diesem Projekt. Bei der Kompilierung des Dienstprogrammprojekts wird zur Kompilierung der IDL-Datei MIDL automatisch aufgerufen.

Für dieses Beispiel wurde nach der Erstellung des Dienstprogrammprojekts eine neue Datei mit dem Namen **InterfaceComponent.idl** erstellt. Fügen Sie den vorstehenden Quellcode hinzu, und führen Sie den Menübefehl **Erstellen** aus. Nach der Erstellung verfügen Sie über die Datei **InterfaceComponent.tlb**, bei der es sich um die Typbibliothek handelt.

IDL mag zunächst umständlich und schwierig erscheinen, dieser Eindruck täuscht jedoch. Sie müssen sich lediglich an die zuvor dargestellte Strukturierung erinnern. Eine COM-Komponente ist ein Paket, das über das Schlüsselwort **library** definiert wird und das Label **SERVERPACKAGELib** aufweist. In dem Paket können

mehrere COM-Klassen enthalten sein, die mit Hilfe der Schlüsselwörter **coclass** defininiert werden und das Label **SimpleComponent** aufweisen. Anhand einer COM-Klasse können Sie unter Verwendung des Schlüsselwortes **interface** mehrere Schnittstellen definieren, diese erhalten das Label **ISimpleInterface**. Alle genannten Schlüsselwörter verfügen durch Verwenden des **uuid**-Attributs über eine verknüpfte GUID.

Instanziieren einer COM-Schnittstelle

Bei Kompilierung der vorgenannten IDL erhalten Sie als Ergebnis eine **Typbibliothek**. Im vorangegangenen Beispiel nimmt der Client zur Verwendung der **ISimpleInterface**-Implementierung keine Instanziierung der Schnittstelle selbst vor. Stattdessen instanziiert der Konsument eine COM-Klasse (**coclass**), bei der die Schnittstelle implementiert wurde. Nach Instanziierung der **coclass** verwendet COM eine **QueryInterface** zum Auffinden der **ISimpleInterface**. Schlägt **QueryInterface** fehl, gibt COM einen Fehler aus, nach dem die Schnittstelle nicht existiert. Ist **ISimpleInterface** vorhanden, wird sie an den Konsumenten zurückgegeben, der anschließend verschiedene Methoden aufrufen kann.

Warum so kompliziert?

Es mag unsinnig erscheinen, IDL zu verwenden. Warum nicht Daten in einer Struktur speichern und anschließend Daten zwischen Betriebssystem und Programmiersprache austauschen? Die Antwort ist, dass unabhängig davon, wie ein Betriebssystem oder eine Programmiersprache geschrieben wurden, eine gemeinsame Sprache erforderlich ist. Jede Sprache, jedes Betriebssystem und jede Hardwarekomponente unterscheidet sich in der Form der Kommunikation. Auf einer 16-Bit-Plattform weist ein **integer**-Wert beispielsweise 16 Bit auf. Auf einer 32-Bit-Plattform umfasst ein **integer**-Wert 32 Bit. Wenn eine Struktur kompiliert und als Dateneinheit zwischen den zwei Plattformen ausgetauscht wird, werden die Werte möglicherweise nicht gleichartig gelesen. COM verwendet IDL, um eine neutrale Form bereitzustellen, mit der Größe und Ausrichtung der Daten angegeben werden. Das so genannte **Marshaling** und **Unmarsheling** sind Verfahren, bei denen die COM-Schicht die Daten von einem Betriebssystem oder einer Programmiersprache in COM und anschließend in das andere Betriebssystem bzw. die andere Programmiersprache zurück konvertiert.

8.2.3 Ein COM-Paket

Bei der Entwicklung einer COM-Anwendung ist kein umfassendes Verständnis von IDL erforderlich, aber Sie müssen das Verhalten von IDL kennen. Ein Verständnis von IDL ermöglicht eine Problemdiagnose im Hinblick auf die Komponentenverwaltung.

Die COM-Komponente

Die COM-Komponente wird durch das Schlüsselwort **library** definiert, wie im Beispielabschnitt zu sehen ist. Die Komponente enthält alle Klassen, die durch die COM-Komponente offen gelegt werden. Diese Attribute werden im Folgenden aufgeführt:

- **uuid:** Die ID zur Definition der Bibliothek. Dieser Parameter ist erforderlich.
- **version:** Gibt die Versionsnummer der Bibliothek an.
- **helpstring:** Ein Text zur Beschreibung der Bibliothek. Dieser ist hilfreich für Objektbrowser.
- **lcid:** Mit dieser ID wird angegeben, welche Sprache auf diese Bibliothek angewendet wird.
- **hidden:** Verbirgt das Objekt vor einem Objektbrowser.

Die COM-Klasse

Das Schlüsselwort **coclass** (Coklasse) wird zur Definition einer Implementierung in einer COM-Komponente eingesetzt. Beim Definieren einer IDL-Datei zur Kompilierung in eine Typbibliothek muss eine **coclass** definiert werden. Geschieht dies nicht, ist während der Kompilierungsphase in MIDL standardmäßig die Schnittstelle nicht enthalten. Die bei der Schnittstellendefinition verwendete **coclass** muss keine besondere Coklasse sein, da sie nicht zur Referenzierung einer Implementierung eingesetzt wird. Die typische Syntax für eine Coklasse lautet folgendermaßen:

```
[attributes]
coclass classname {
   [interface attributes] [interface | dispinterface] interfacename;
};
```

In der Implementierungsphase ist die **coclass** dagegen sehr wohl wichtig. Die **coclass** weist folgende Attribute auf:

- **uuid:** Gibt die CLSID (Class ID) an, die zur Identifizierung dieses Objekts eingesetzt wird.
- **version:** Gibt die Versionsnummer der Bibliothek an.
- **helpstring:** Ein Text zur Beschreibung der Klasse. Dieser ist hilfreich für Objektbrowser.

- **licensed:** Teilt der Klasse mit, dass das Objekt lizenziert ist und geprüft werden sollte. In diesem Fall sollte die Schnittstelle **IClassFactory2** als Class Factory verwendet werden.
- **hidden:** Verbirgt das Objekt vor einem COM-Objektbrowser.

Innerhalb der **coclass**-Definition können die verschiedenen Schnittstellen diese Attribute aufweisen:

- **source:** Gibt an, dass die Schnittstelle eine Ereignisquelle darstellt und zusammen mit dem Container **IConnectionPoint** verwendet wird.
- **default:** Wird von Makroprogrammierern (VBScript) zur Definition der standardmäßig aufzurufenden Schnittstelle verwendet, wenn keine Schnittstelle angegeben ist.
- **restricted:** Verhindert, dass die Schnittstelle von Makroprogrammierern verwendet wird.

8.2.4 Die COM-Schnittstelle

Die meiste Zeit muss für das Entwickeln guter Schnittstellen aufgebracht werden. Die Schnittstelle wird in COM folgendermaßen definiert:

```
[attributes]
interface interfacename [:baseinterface] {
   functionlist
};
```

Zu den gültigen Attributen gehören:

- **dual:** Gibt an, dass die definierte Schnittstelle sowohl benutzerdefinierte als auch **IDispatch**-Schnittstellen unterstützt.
- **object:** Ein besonderes Tag, das den MIDL-Compiler anweist, COM-kompatiblen Code zu generieren. Wird dieses Tag angegeben, muss es über eine verknüpfte **uuid** verfügen. Ohne dieses Tag wird die Schnittstelle als DCE RPC-Aufruf (Distributed Computing Environment) kompiliert.
- **uuid:** Gibt die CLSID an, mit der dieses Objekt identifiziert wird.
- **helpstring:** Ein Text zur Beschreibung der Schnittstelle. Dieser ist hilfreich für Objektbrowser.
- **pointer_default:** Wird verwendet, wenn innerhalb einer Schnittstelle Zeiger als Parameter verwendet werden. Dies schließt keine Zeiger der obersten Ebene ein, nur Elemente wie Doppelzeiger.
- **oleautomation:** Gibt an, dass die Schnittstelle nur Parameter unterstützt, die als Standardautomatisierungstypen betrachtet werden.

In der Methodenparameterliste befinden sich weitere Attribute, die angeben, wie der Parameterspeicher zugewiesen wird und wie Informationen gesendet werden. Die Attribute werden in der Liste durch Kommata voneinander getrennt.

Der »QueryInterface«-Prozess

COM unterscheidet sich von allen anderen Komponententechnologien dadurch, dass es mit unbekannten Elementen umgehen kann. Nehmen Sie an, ein Konsument instanziiert eine COM-Klasse. Wie erhält die COM-Schicht die Information, dass die COM-Klasse die COM-Schnittstelle implementiert hat? Die COM-Schicht fragt die COM-Klasse mit Hilfe des **QueryInterface**-Prozesses ab, ob die Schnittstelle implementiert wurde. Sehen wir uns an, welche Schnittstellen und Methoden implementiert werden müssen.

Jede COM-Klasse muss **IUnknown** implementieren, und jede COM-Schnittstelle muss von **IUnknown** erben. Die Schnittstellendefinition von **IUnknown** lautet folgendermaßen:

```
[
   local,
   object,
   uuid(00000000-0000-0000-C000-000000000046),
   pointer_default(unique)
]
interface IUnknown {
HRESULT QueryInterface( [in] REFIID riid, [out, iid_is(riid)] void **ppvOb-
ject);
ULONG AddRef();
ULONG Release();
}
```

Es sind drei Methoden in der **IUnknown**-Definition vorhanden, **QueryInterface**, **AddRef** und **Release**. COM ruft die erste Methode, **QueryInterface**, bei Instanziierung der COM-Klasse auf. Der erste Parameter, **riid**, ist die GUI der angeforderten Schnittstelle. In der Implementierung wird dieser Parameter mit der Liste der implementierten Schnittstellen verglichen. Liegt eine Übereinstimmung vor, gibt die Implementierung über den Zeiger **ppvObject** einen **vtable**-Zeiger zurück, der die Schnittstelle repräsentiert.

Zu diesem Zeitpunkt vertraut COM darauf, dass die **vtable**-Signatur der Schnittstellenimplementierung mit der **vtable**-Signatur der IDL-Schnittstelle übereinstimmt. Bei Verwendung der Sprachen Visual Basic, Visual C++ und Visual J++ stellt dies kein Problem dar, da eine Nichtübereinstimmung der **vtable**-Signaturen

zu einem Kompilierungsfehler führen würde. Bei anderen Sprachen ist dies in den meisten Fällen auch so, aber Sie sollten die COM-Implementierungsdetails der Sprache prüfen.

IDL unterstützt das Konzept der so genannten Schnittstellenvererbung, und eine Schnittstelle, die von **IUnkown** erbt, weist folgende IDL auf:

```
[
   object,
   uuid(E05034D2-8EB8-11d2-86CB-0000B45FCBCB),
   helpstring("ISimpleInterface2 Interface"),
   pointer_default(unique)
]
interface ISimpleInterface2: IUnknown {
   [helpstring("method1")] HRESULT method1();
};
```

Die **vtable**-Signatur für **ISimpleInterface2** verfügt über vier Funktionszeiger in der folgenden Reihenfolge:**QueryInterface**, **AddRef**, **Release** und **method1**. Die ersten drei Funktionszeiger sind das Ergebnis der Schnittstellenvererbung.

Verweiszählung für Implementierungen

Die zwei weiteren Methoden der **IUnknown**-Schnittstelle sind **AddRef** und **Release**. Diese zwei Methoden sorgen für die Verwaltung der Verweiszählung.

In Abbildung 8.3 instanziiert Konsument A die COM-Klasse **SimpleComponent**. Anschließend instanziiert Konsument B die COM-Klasse **SimpleComponent**. Die Instanz von **SimpleComponent** ist jedoch ein so genanntes Singleton und gibt einen Verweis auf sich selbst zurück. Ein Singleton ist ein besonderes Objekt, bei dem nur eine Instanz der Klasse im Speicher vorliegen kann. Konsument A beendet den Vorgang und löscht die Instanz **SimpleComponent**. Konsument B erfordert jedoch weiterhin die Ausführung der **SimpleComponent**-Instanz. Löscht Konsument A tatsächlich **SimpleComponent**? Die Antwort auf diese Frage liegt im Verweiszähler. Immer dann, wenn ein Konsument eine COM-Klasse instanziiert oder referenziert, wird der Verweiszähler mit Hilfe von **AddRef** erhöht. Benötigt der Konsument den Verweis nicht mehr, wird der Verweiszähler mit Hilfe von **Release** verringert. Nachdem die Verweiszählung den Wert Null erreicht hat, löscht die COM-Klasseninstanz sich selbst.

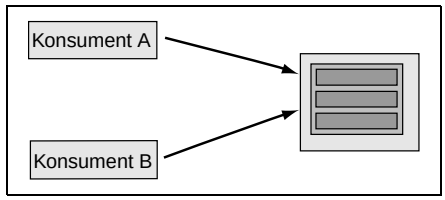

Abbildung 8.3 Zwei Konsumenten, die ein einzelnes COM-Objekt referenzieren

Dieses Vorgehen scheint geeignet, aber könnte es nicht zu Fehlzählungen kommen? Fehler treten in einer 32-Bit- oder 64-Bit-Multitaskingumgebung wie Windows 2000 nicht auf. Das Betriebssystem fängt Prozesse auf, die nicht nachvollziehbar gelöscht werden und weitere Probleme verursachen. Wurde das Programm für die Verweiszählung jedoch nicht geeignet geschrieben, kann eine falsche Verweiszählung erfolgen. Bei Sprachen wie Visual Basic oder Visual J++ wird die Zählung durch die Sprache verwaltet. Die Verweiszählung in Visual C++ wird durch die **Helper**-Klassen verwaltet.

Benutzerdefinierte Schnittstellen und frühe Bindung

Erbt eine Schnittstelle direkt von **IUnknown**, wird diese als benutzerdefinierte Schnittstelle bezeichnet, und benutzerdefinierte Schnittstellen nehmen eine **frühe Bindung** vor. Frühe Bindung bedeutet, dass die **vtable**-Signatur der Schnittstelle bei Konsumentenkompilierung bekannt ist. Bei der frühen Bindung besteht ein »Vertrag« zwischen Konsument und Implementierung, der durch die Typbibliothek definiert wird.

Die Laufzeitgeschwindigkeit liegt bei Verwendung der frühen Bindung höher, da der Konsument weiß, wo die Implementierungsfunktionalität zu finden ist. Das Verwenden der frühen Bindung führt jedoch zu Problemen, wenn die Implementierung den Vertrag ändert, ohne die Konsumentenanwendung hierüber zu informieren – derartige Änderungen der **vtable**-Signatur können zu Anwendungsfehlern führen.

Späte Bindung

Gelegentlich erstellen Sie COM-Klassen, die ihre zugehörigen Methoden dynamisch festlegen. Nehmen Sie die Erstellung einer COM-Schnittstelle zu einer Datenbank. Sie möchten die Fähigkeit besitzen, dynamisch Methoden zu generieren, mit denen Datenbankfunktionen repräsentiert werden. Sie können keine Typbibliothek erstellen, da diese überholt wäre, sobald sie erstellt wurde. In dieser Situation oder in anderen Fällen, in denen Sie keine Typbibliothek offen legen möchten, gilt die **späte Bindung**.

Die späte Bindung wird auch als OLE-Automatisierung bezeichnet. Die späte Bindung ist interessant, da der Konsument mit Hilfe einer als Vermittler fungierenden allgemeinen Funktion die Methode oder Methodeneigenschaft aufrufen kann. Der Vermittler akzeptiert Struktur und Parameter und versucht anschließend zu ermitteln, welche Methode in der Anwendung aufgerufen wird. Der Vermittler ruft dann die Methode auf und gibt die Daten in Form einer weiteren Struktur mit Informationen zurück. Dieser Vermittler wird als **IDispatch**-Schnittstelle bezeichnet, einem Bestandteil der Implementierung. ASP verwendet diese Bindungsmethode beim Aufruf von COM-Klassen.

Duale Schnittstellen

Bei der Objekterstellung muss entweder die frühe oder die späte Bindung unterstützt werden. Die frühe Bindung ist schneller, jedoch weniger flexibel. Liegt eine andere Version des Servers vor, und wurden bestimmte Methoden geändert oder entfernt, stürzt die Anwendung ab, da keine ordnungsgemäße VTabelle vorhanden ist.

Soll das Objekt flexibel bleiben und einen Softcrash oder einen wiederherstellbaren Absturz unterstützen, sollten Sie die späte Bindung einsetzen. Das Verwenden der späten Bindung geht auf Kosten der Geschwindigkeit. Bei der späten Bindung fragt der Konsument einer COM-Klasse, ob eine Methode mit spezifischer Signatur vorliegt. Die COM-Klasse durchsucht die Funktionstabelle und gibt eine ID zurück. Der Konsument erstellt anschließend eine Struktur mit allen Parametern und gibt diese an die COM-Klasse weiter, die die Struktur wiederum in einen Funktionsaufruf übersetzt. Da die Methoden allgemein sind und Parameter und Methoden in Arrays gespeichert werden, muss die Implementierung die Strukturen entschlüsseln. Aufgrund des Frage- und Antwortprozesses und der vorgenommenen Interpretation verlangsamt sich der Vorgang.

Beide Methoden weisen Vor- und Nachteile auf. Wenn Sie möchten, dass das Objekte beide Methoden unterstützt, können Sie sowohl die frühe als auch die späte Bindung implementieren. Diese Implementierung wird als duale Schnittstelle bezeichnet. Die Implementierung ist etwas umfangreicher, so wird jedoch sichergestellt, dass alle Clients das Objekt verwenden können. Die duale Methode wird bei den meisten Sprachen innerhalb der Programmierumgebung verborgen.

Welche Bindungsmethode ist besser?

Viele Programmierer glauben, dass benutzerdefinierte Schnittstellen besser sind, da durch sie schnellere Anwendungen und eine bessere Versionsnummerierung bereitgestellt werden. Im Allgemeinen ist die frühe Bindung vorzuziehen, aber es gibt besondere Situationen, in denen die späte Bindung eingesetzt werden sollte.

Im vorangegangenen Kapitel beispielsweise verwendete die Visual InterDev-Skriptobjektbibliothek die Datenumgebungs-COM-Komponente. Dieses Objekt übersetzt gespeicherte Prozeduraufrufe direkt in COM-Methodenaufrufe. Die Datenumgebung vollführt diesen Trick mit Hilfe der späten Bindung und durch das dynamische Offenlegen von Funktionen, die gespeicherte Prozeduraufrufe repräsentieren. Diese Technik erscheint auf den ersten Blick langsamer, aber sehen wir uns dies genauer an.

Wird die Datenumgebung mit Hilfe von benutzerdefinierten Schnittstellen implementiert, müssen Methoden für das Einstellen der verschiedenen Parameter sowie zum Aufrufen der gespeicherten Prozeduren geschrieben werden. Dies bedeutet, dass die Implementierung die Parameter und gespeicherten Prozeduraufrufe in spezifische gespeicherte Prozeduren übersetzen muss. Mit anderen Worten, die Implementierung muss zunächst einiges an Verarbeitung leisten, bevor die eigentliche gespeicherte Prozedur aufgerufen werden kann.

Das **Data Environment**-Objekt, auf der anderen Seite, ist ein Beispiel für ein Objekt zur Technologieüberbrückung. Es wird lediglich ein Methodenaufruf einer Technologie in eine andere (COM in SQL [Structured Query Language]) übersetzt. Das Verwenden der Datenumgebung ermöglicht eine einfachere Programmiersyntax und führt zu fast keinen Leistungseinbußen.

Die Faustregel bei Auswahl der Bindungsmethode lautet, immer eine duale Schnittstelle zu verwenden, wenn möglich jedoch die späte Bindung einzusetzen.

8.3 Einige Entwurfsmethoden für die COM-Schnittstelle

Nachdem Sie nun über ein Grundverständnis zu COM und Schnittstellen verfügen, müssen einige Entwurfsmethoden für die COM-Schnittstellen erläutert werden.

8.3.1 Schnittstellen sind unveränderlich

Wenn eine Schnittstelle veröffentlicht und der Öffentlichkeit zugänglich gemacht wurde, wird sie als unveränderlich betrachtet und kann nicht weiter bearbeitet werden. Das Veröffentlichen einer Schnittstelle bedeutet nicht zwangsläufig, dass diese dem allgemeinen Konsumenten zugänglich gemacht wird, sondern dem Entwicklungsteam.

Unveränderlichkeit einer Schnittstelle

Sie sollten eine Schnittstelle erst nach reiflicher Überlegung als unveränderlich festlegen. Diese Überlegung ähnelt der Entscheidung, sich einen Welpen anzuschaffen. Nachdem der Welpe ins Haus gekommen ist, braucht er Aufmerksamkeit und kann nicht einfach beiseite gelegt werden. Ebenso ist das Festlegen der

Unveränderlichkeit einer Schnittstelle eine Entscheidung, die nur einmal gefällt wird. Wenn Sie später Änderungen vornehmen möchten, ist dies nicht mehr möglich. Die Philosophie, die hinter der Anwendungsentwicklung in diesem Buch steckt, ist das Verwenden iterativer Entwicklungsmethoden. Eine unveränderliche Schnittstelle kann jedoch nicht schrittweise weiterentwickelt werden.

Im vorgestellten Entwicklungsprozess möchten wir mehrere COM-Schnittstellen einsetzen, um eine feine Granularität zu erzielen. Manchmal kann es aber auch zuviel des Guten sein. Eine feine Granularität kann die Dinge unnötig verkomplizieren. Mit dem **Façade-Muster** können die Dinge vereinfacht werden. Hierzu können Sie einen iterativen **Schichtenansatz** oder einen fließenden Ansatz verwenden.

Wenden wir nun an, was in den vorherigen Abschnitten zum Entwurfsteam gesagt wurde, das sich die Arbeit teilte. Und nehmen Sie an, die Anwendung wird anhand des Iterationskonzepts entwickelt. Wieder entwickelt das Entwurfsteam eine Reihe von COM-Schnittstellen für die Anwendung. Anschließend werden jedoch nur einige der Schnittstellen implementiert. Die Faktoren bei Auswahl der zu implementierenden Schnittstellen richten sich nach dem Team und den gesetzten Prioritäten. Nehmen Sie das in Abbildung 8.4 gezeigte Beispiel der Implementierung einer gesamten Anwendung mit einigen COM-Schnittstellen.

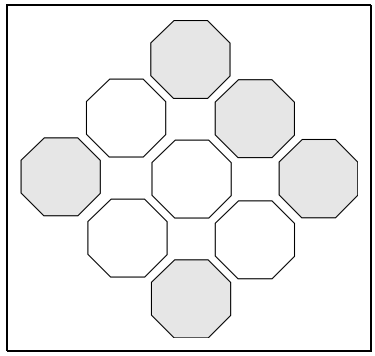

Abbildung 8.4 Gesamte Anwendung

Die gefüllten Blöcke repräsentieren implementierte COM-Schnittstellen, die leeren Blöcke stellen lediglich COM-Schnittstellendefinitionen dar. Bei diesem Ansatz werden die wichtigsten COM-Schnittstellen entwickelt und implementiert. Bei der weiteren Implementierung beginnt das Testteam mit dem Testen der Implementierungen. Das Testen wird schrittweise durchgeführt.

Sobald das Entwurfsteam den Schnittstellenentwurf vorläufig beendet hat, können die Testergebnisse verarbeitet werden. Einige Ergebnisse können auf Pro-

bleme hinweisen; das Entwurfsteam erkennt diese sofort und kann entsprechende Änderungen vornehmen. Diese Änderungen erfordern möglicherweise neue COM-Schnittstellen oder Änderungen an den Testspezifikationen. Diese inkrementellen Änderungen wirken sich zwar nicht auf die gesamte Anwendung aus, dennoch wird die gesamte Anwendung justiert.

Wo mit der Implementierung beginnen? Der iterative Ansatz klingt gut, bis Sie zum Implementieren und Testen der Objekte kommen. Der Grund hierfür ist der, dass Sie nicht wissen, welche COM-Schnittstellen implementiert werden müssen. Die Implementierung der COM-Schnittstellen ähnelt der Tunnelerstellung, wobei die Schicht der Anwendungslogik den Tunnel darstellt, mit dem Darstellungs- und Datenschicht miteinander verbunden werden. Beginnen Sie mit der Tunnelerstellung an der Darstellungsschicht und arbeiten sich zur Datenschicht vor, oder starten Sie an der Datenschicht und arbeiten sich zur Darstellungsschicht vor, oder beginnen Sie an beiden Seiten gleichzeitig und hoffen darauf, dass sie sich die Tunnelabschnitte in der Mitte treffen?

Die gewählte Methode richtet sich nach dem Entwicklerteam. Wenn Sie einen Ansatz wählen, bei dem der Tunnel von einer Schicht aus erstellt wird, entwickeln erstellen Sie einen Kernsatz, mit dem die verschiedenen Schichten nacheinander erstellt werden. Dies ist ein sequenzieller und iterativer Ansatz, da Planänderungen als Zwischenschritt vorgenommen werden. Das Problem besteht darin, dass die Schichten nicht signifikant geändert werden können, nachdem sie als unveränderlich festgelegt wurden – d. h. Änderungen erfordern ein erneutes Testen und Auswerten, was wiederum Zeit und Geld kostet. Dieser Ansatz ist jedoch einfacher umzusetzen als die gleichzeitige Tunnelerstellung von beiden Seiten.

Der Ansatz der Tunnelerstellung von beiden Seiten aus ist weitaus flexibler, da Schnittstellen erst als unveränderlich festgelegt werden, wenn Sie in verschiedenen Szenarien eingesetzt und für effektiv befunden wurden. Eine unzureichend entworfene Schnittstelle wird geändert, und jeder Benutzer dieser Schnittstelle muss entsprechende Änderungen vornehmen. Dieser Ansatz bedeutet ebenfalls, dass mehrere Änderungen auftreten können, und wenn einige Teams Änderungen an der Schnittstelle vornehmen, ohne die anderen Beteiligten zu informieren, treten Fehler auf. Es liegt in der Verantwortung des Entwurfsteams, diese Änderungen zu verfolgen und alle Beteiligten über Änderungen zu informieren.

Dieser Ansatz funktioniert nicht, wenn das Team nicht über die erforderliche Disziplin, Erfahrung in der Einschätzung von Implementierungsänderungen oder das Erstellen effektiver Kommunikationskanäle verfügt. Derartige Defizite führen zu Budgetüberschreitungen und Zeitverzögerungen und somit letztendlich zu einem Misserfolg bei der Anwendungsentwicklung. Die Erfahrung zeigt, dass der zweite

Ansatz besser einsetzbar ist. Es handelt sich um einen flexibleren Ansatz, der zu einer Anwendung führt, die eher den Wünschen des Endbenutzers entspricht.

8.3.2 Wann muss eine Schnittstelle erweitert werden?

Unveränderlich heißt soviel wie »nicht änderbar«. Wie im richtigen Leben gibt es jedoch Grauzonen. Diese Grauzonen werden besonders in der Entwicklungsphase erkennbar. Nehmen Sie beispielsweise an, dass eine Entwurfsänderung erforderlich ist, die Schnittstelle jedoch bereits getestet und praktisch als unveränderlich festgelegt wurde. Statt eine weitere Schnittstelle hinzuzufügen, durch die ein Mehraufwand hinsichtlich Dokumentation, Entwurf und Wartung erforderlich würde, ist es einfacher, die praktisch bereits als nicht änderbar geltende Schnittstelle zu bearbeiten. Sie können sich auch vorstellen, dass ein Patch für einen kleinen Bug bereitgestellt werden muss. Das Hinzufügen einer weiteren Schnittstelle ist zu kosten- und zeitintensiv, wenn das Hauptziel in der Minimierung von Kosten und Systemausfällen besteht.

Das Vornehmen derartiger Änderungen kann nicht als Entwurfsmethode bezeichnet werden. Gute Entwürfe und eine gute Planung können Situationen verhindern, in denen Änderungen an unveränderlichen Schnittstellen erforderlich werden.

Erweitern der Schnittstelle

Die erste Methode zur Bearbeitung der Schnittstelle stellt das einfache Hinzufügen der zusätzlichen Methode an das Ende der VTabelle dar, während die ursprüngliche VTabelle intakt bleibt. Diese Methode wird als Erweiterung bezeichnet.

Sehen Sie sich die folgende unveränderliche Schnittstelle an:

```
[
   uuid(BE1FFD3B-E489-11D1-B44D-00A0247D759A),
   version(1.0),
   helpstring("Math 1.0 Type Library")
]
interface IMath : IUnknown
{
   [id(1)] HRESULT add(long param1, long param2, long *retvalue);
};
```

Die obige Schnittstelle ist eine mathematische Schnittstelle mit der Fähigkeit, zwei Zahlen (**param1** und **param2**) zu addieren und das Ergebnis in der Variablen **retvalue** zurückzugeben.

Das folgende Beispiel fügt eine zusätzliche **subtract**-Methode an das Ende der ursprünglichen VTabelle an.

```
[
object, dual,
   uuid(BE1FFD3B-E489-11D1-B44D-00A0247D759A),
   version(2.0),
   helpstring("Math 2.0 Type Library")
]
interface IMath : IDispatch
{
   [id(1)] HRESULT add(long param1, long param2, long *retvalue);
   [id(2)] HRESULT subtract(long param1, long param2, long *retvalue);
};
```

In dieser erweiterten Schnittstellendefinition weist die **subtract**-Methode eine abweichende Funktionalität bereit. Im Vergleich zur Erstellung einer neuen Schnittstelle und dem damit verbundenen Overhead stellt diese Methode die weniger aufwendige und einfachere Lösung dar. Beim Veröffentlichen dieser Schnittstelle müssen Sie sicherstellen, dass die **version**-Nummer der Schnittstellenattribute geändert wird. Auf diese Weise kann der Kunde zwischen den verschiedenen Versionen unterscheiden.

Hinzufügen ergänzender Funktionalität

Im vorangegangenen Beispiel wurde die **subtract**-Methode hinzugefügt. Beide Methoden sind lediglich in der Lage, Zahlen vom Typ **integer** zu addieren (bzw. zu subtrahieren). Angenommen, Sie müssen die Funktionalität so abändern, das auch reelle Zahlen addiert werden können. Die Erweiterung wird auch hier verwendet, die Methode wird jedoch **add2** genannt, um die ähnliche Funktionalität zu verdeutlichen. Die Lösung lautet folgendermaßen:

```
interface IMath : IDispatch
{
   [id(1)] HRESULT add(long param1, long param2, long *retvalue);
   [id(2)] HRESULT add2(double param1, double param2, double *retvalue);
};
```

Die Zahl am Ende der **add2**-Methode gibt die abweichende Version an, was üblicherweise andere Parameter erfordert. Diese Methode wird angewendet, wenn nur wenige ähnliche Methoden vorhanden sind. Es sollte unter keinen Umständen **add100** verwendet werden, da dies bedeuten würde, dass Ihre Schnittstelle einhundert Mal geändert wurde.

Hinzufügen ergänzender Schnittstellen

Nehmen Sie an, Ihre Anwendungen müssen radikal geändert werden. Angenommen, Sie erfahren, dass kein Kostenunterschied zwischen der Verwendung von **long**- und **double**-Werten für die Addition vorhanden ist, und dass in einigen Situationen die Verwendung von **double**-Werten schneller zum Ziel führt. Tritt eine solche radikale Änderung ein, muss die Schnittstelle aktualisiert werden.

Eine radikale Änderung erfordert eine neue Schnittstellendefinition. Sehen Sie sich die folgende Schnittstellendefinition an:

```
interface IMath2 : IDispatch
{
   [id(1)] HRESULT add(double param1, double param2, double *retvalue);
   [id(2)] HRESULT subtract(double param1, double param2, double *retvalue);
};
```

In dieser Lösung wird nicht der Methode, sondern der Schnittstelle (**IMath2**) ein inkrementeller Wert hinzugefügt. Die Schnittstelle könnte auch **IMathDoubleVersion** heißen, beachten Sie jedoch die Situation, in der die Schnittstelle **long double**-Werte implementiert. Die Schnittstelle hieße nun **IMathLongDoubleVersion**. Erweitern Sie dies auf 100 Schnittstellen, und die Schnittstellennamen würden schnell äußerst lang und unübersichtlich. Dies ist eine Frage der Semantik, aber im Rahmen einer umfangreichen Skalierung von Vorteil. Der Vorteil des inkrementellen Ansatzes besteht darin, dass ein beliebiger Entwickler, der die Schnittstelle verwendet, weiß, dass die Schnittstellen ergänzend sind. Wenn bekannt ist, welche Funktion eine der Schnittstellen besitzt, sollte die Funktion der weiteren Schnittstellen erraten werden können.

8.3.3 Verwenden einer Programmiersprache zur Entwicklung von COM-Schnittstellen

Jetzt sollen COM-Schnittstellen definiert werden. Sie haben erfahren, dass IDL die Grundlage von COM darstellt, daher ist es sinnvoll, IDL einzusetzen. Da eine COM-kompatible Sprache COM-Typenbibliotheken generiert, sollte diese Sprache zur Definition der entsprechenden Schnittstellen eingesetzt werden. Dieser Vorgang wird als Erstellung einer leeren Implementierung bezeichnet. Verwenden wir diese Methode und ermitteln, welche Art von COM IDL generiert wird.

Eine einfache Komponente in den verschiedenen Sprachen

Zum Vergleich der generierten IDL verwenden wird die COM-Schnittstellendefinition aus dem Abschnitt »Ein einfaches Beispiel« weiter oben in diesem Kapitel. Hier versuchen wir vor allem, **SimpleComponent** zu definieren.

Definieren von »SimpleComponent« über Visual C++ Zur Entwicklung einer COM-Komponente, die als leere Implementierung verwendet wird, muss es sich bei dieser um eine ATL-Komponente (Active Template Library) handeln. Es ist möglich, MFCs (Microsoft Foundation Classes) zu verwenden, ATL ist jedoch für COM-Schnittstellen besser geeignet (sowohl ATL als auch MFC sind Bibliotheken, die den Prozess der Entwicklung von Windows-Anwendungen erleichtern). Bei der Erstellung der ATL-Komponente müssen Sie sicherstellen, dass es sich um eine DLL (Dynamic Link Library) handelt. Die Schnittstellen werden unter Verwendung des ATL-Objekt-Assistenten zum Hinzufügen eines neuen **Einfachen Objekts** entworfen. Es gibt zwei Dialogfelder, die sorgfältig ausgefüllt werden sollten, diese werden in den Abbildungen 8.5 und 8.6 gezeigt.

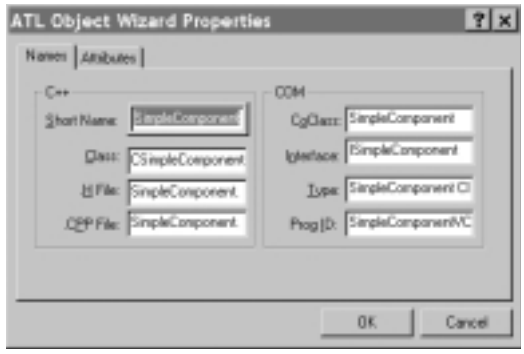

Abbildung 8.5 Registerkarte »Namen« im Eigenschaftenfenster des Assistenten für das Objekt »SimpleComponent«

Der kurze Name definiert den Komponentennamen, die COM-Coklasse trägt denselben Namen. Der Assistent erstellt nicht unabhängig von der COM-Schnittstelle eine COM-Coklasse – die zwei Elemente werden in einem Arbeitsschritt erstellt. Die Schnittstelle weist das Präfix »I« auf, um zu kennzeichnen, dass es sich um eine COM-Schnittstelle handelt. Die Programm-ID lautet **SimpleComponentVC.SimpleComponent**, eine Aneinanderreihung von COM-Bibliotheksname und COM-Coklasse.

Abbildung 8.6 definiert die Attribute der hinzuzufügenden Schnittstelle. Das einzige Attribut von Interesse ist das **Interface**-Attribut, das die Eigenschaft **Custom** oder **Dual** aufweisen kann. Lautet die Eigenschaft **Custom**, wird **ISimpleComponent** von **IUnknown** abgeleitet. **Dual** bedeutet, dass die Schnittstelle auf **IDispatch** basiert. In diesem Fall wurde das Schnittstellenattribut **Custom** ausgewählt.

Zur Vervollständigung der Entwicklung ist es erforderlich, mit Hilfe eines kleinen Assistenten **method1** als Teil der Schnittstelle hinzuzufügen. Dies ist alles, was zur

Erstellung einer Typbibliothek erforderlich ist, die eine Reihe von Schnittstellen enthält, die in anderen Entwicklungsumgebungen implementiert werden können.

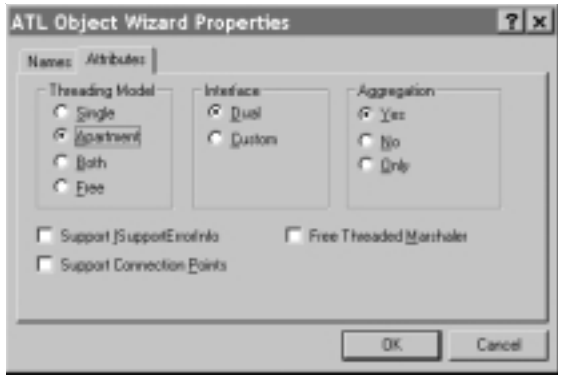

Abbildung 8.6 Registerkarte »Attribute« des Assistenten für das Objekt »SimpleComponent«

Was generiert der Assistent in der Typbibliothek? Dies muss nicht gezeigt werden, da die Ausgabe nahezu identisch mit der Anzeige in der ursprünglichen IDL-Datei ist. Der wichtige Punkt ist, dass Visual C++ die Fähigkeit zur Feinabstimmung der IDL bietet und die Erstellung einer wunschgemäßen Schnittstelle ermöglicht.

Definieren von »SimpleComponent« über Visual Basic In Visual Basic kann, wie in Visual C++, eine Typbibliothek durch das Definieren leerer Visual Basic-Klassen definiert werden. Erstellen Sie hierzu ein Visual Basic-ActiveX-DLL-Projekt (Dynamic Link Library). Nennen Sie das Projekt **SimpleComponentVB**, es wird anschließend eine einzige VB-Klasse mit Namen **SimpleComponent** enthalten. Die Definition der Klasse lautet:

```
Public Sub method1(ByVal param1 As Long)
End Sub
```

Nach der Kompilierung wird die Typbibliothek generiert.

```
 [
   uuid(FC81A421-2A94-11D3-957D-0080C700807A),
   version(4.0)
]
library SimpleComponentVB
{
importlib("STDOLE2.TLB");
```

```
   // Vorwärts gerichtete Deklaration aller in dieser typelib definierten
Typen
   interface _SimpleComponent;

   [
      odl,
      uuid(78C21322-2A95-11D3-957D-0080C700807A),
      version(1.0),
      hidden,
      dual,
      nonextensible,
      oleautomation
   ]
   interface _SimpleComponent : IDispatch {
      [id(0x60030000)] HRESULT method1([in] long param1);
   };
   [
      uuid(FC81A423-2A94-11D3-957D-0080C700807A),
      version(1.0),
      appobject
   ]
   coclass SimpleComponent {
      [default] interface _SimpleComponent;
   };
};
```

Ein Teil dieser Typbibliotheksgenerierung ist nicht richtig, da die Typbibliothek mit Hilfe der OLE-Ansicht angezeigt wurde. Die OLE-Ansicht übersetzt einige der IDL-Schlüsselwörter in eine eigene Darstellung, die nicht immer der ursprünglichen IDL entspricht. Im Folgenden liegt das Hauptaugenmerk auf der Benennungskonvention und den Schnittstellentypen. Im Gegensatz zu Visual C++ wird in Visual Basic der Schnittstellenname nicht mit dem Präfix »I« versehen, stattdessen wird ein Unterstrich verwendet. Wie in Visual C++ wird in Visual Basic die COM-Coklasse (**coclass**) mit der COM-Schnittstelle (**interface**) verknüpft. Im Gegensatz zu Visual C++ kann Visual Basic jedoch nur eine Schnittstelle aus einer spezifischen Klasse generieren – die von Visual Basic erstellte Schnittstelle ist dual. Mit Visual Basic können keine benutzerdefinierten Schnittstellen definiert werden. Des Weiteren ist eine Beschränkung hinsichtlich der Parameter und Parametertypen vorhanden.

Definieren von »SimpleComponent« über Visual J++ Zur Erstellung einer COM-Schnittstelle mit Visual J++ erstellen Sie ein Visual J++-Projekt, das eine COM-DLL repräsentiert. Benennen Sie anschließend die hinzugefügte Java-Klassendatei in **SimpleComponent** um, und definieren Sie die Quellen folgendermaßen:

```
/**
 * Diese Klasse wurde zur Paketerstellung im COM-DLL-Ausgabeformat entworfen.
 * Die Klasse verfügt mit Ausnahme der Erstellungsroutine über keine standard-
mäßigen Einsprungpunkte.
 * Öffentliche Methoden werden als Methoden der standardmäßigen COM-Schnitt-
stelle offen gelegt.
 * @com.register ( clsid=4D559511-2A97-11D3-957D-0080C700807A, type-
lib=4D559512-2A97-11D3-957D-0080C700807A )
 */
public class SimpleComponent
{
   public void method1( long param1) {
      return;
   }
}
```

Durch das Kompilieren der Klasse wird die Typbibliothek generiert.

```
[
   uuid(4D559512-2A97-11D3-957D-0080C700807A),
   version(1.0),
   helpstring("SimpleComponentVJ")
]
library SimpleComponentVJ
{
   // TLib :    // TLib : OLE Automation : {00020430-0000-0000-C000-000000000046}
   importlib("STDOLE2.TLB");

   // Vorwärts gerichtete Deklaration aller in dieser typelib definierten Typen
   dispinterface SimpleComponent_Dispatch;
   [
      uuid(72AA2BB7-2A97-11D3-957D-0080C700807A)
   ]
   dispinterface SimpleComponent_Dispatch {
      properties:
      methods:
         [id(0x00000064)]
         VARIANT wait(
```

```
            [in, out] VARIANT* Parameter0,
            [in, out] VARIANT* Parameter1);
         [id(0x00000065), helpstring("hashCode")]
         long hashCode();
         [id(0x00000066), helpstring("toString")]
         BSTR toString();
         [id(0x00000067), helpstring("equals")]
         VARIANT_BOOL equals([in] IDispatch* Parameter0);
         [id(0x00000068), helpstring("method1")]
         void method1([in] long Parameter0);
         [id(0x00000069), helpstring("notify")]
         void notify();
         [id(0x0000006a), helpstring("getClass")]
         IDispatch* getClass();
         [id(0x0000006b), helpstring("notifyAll")]
         void notifyAll();
   };

   [
      uuid(4D559511-2A97-11D3-957D-0080C700807A),
      helpstring("SimpleComponent")
   ]
   coclass SimpleComponent {
      [default] dispinterface SimpleComponent_Dispatch;
   };
};
```

Die generierte Ausgabe ist in einer Schnittstelle nicht erwünscht. Visual J++ fügt verschiedene Methoden hinzu, die Teil der Java-Objektklasse sind und betrachtet die Schnittstelle als **dispinterface**. Hierbei handelt es sich um eine COM-Spezifikation, nach der die Methoden der Schnittstelle nur mit Hilfe der späten Bindung ausgelöst werden können. Dies bedeutet, dass COM langsamer werden kann, und dies soll natürlich nicht passieren. Ein weiteres kleines Problem ist die Benennung der COM-Schnittstelle mit **SimpleComponent_Dispatch**. Wird dieser Client im Kontext eines Visual C++- oder Visual Basic-Clients verwendet, sind mehrere Benennungskonventionen vorhanden.

Und die Lösung ist? Die einfachste Methode zur Entwicklung einer Schnittstelle ist die Verwendung von Visual C++ oder das Schreiben eines Dienstprogrammprojekts und eine anschließende Kompilierung der IDL-Datei mit Hilfe von MIDL. Bei der manuellen Bearbeitung der IDL-Datei haben Sie umfassende Steuerungsmöglichkeiten hinsichtlich der Definition und Feinabstimmung der Schnittstellen und Methoden. Die IDL-Datei kann sich ändern und das Neuschreiben erfordern. Das Konzept der Schnittstellen und Implementierungen ändert sich jedoch nicht.

Parameter innerhalb von Methoden

COM bietet die Fähigkeit zur Definition von Parameterdatentypen. Sie können entweder allgemeine Datentypen oder feste Datentypen verwenden. Ein allgemeiner Datentyp ist beispielsweise **VARIANT**, der einen beliebigen Datentyp akzeptieren kann. Ein fester Datentyp, beispielsweise **long** oder **double**, kann nur einen bestimmten Wertetyp enthalten.

Feste Parameter weisen folgende Vorteile auf:

- Durch den spezifische Datentype kann leicht ermittelt werden, welcher Datentyp weitergegeben werden muss.
- Bei der Konvertierung von einem Datentyp in einen anderen gehen keine Daten verloren.
- Feste Datentypen sind schneller als **VARIANTS**.

Allgemeine Parameter weisen ebenfalls einige Vorteile auf:

- Implementierungen können leicht aktualisiert werden, ohne dass eine Schnittstellenänderung erforderlich ist, da die Parameter jeden Datentyp akzeptieren können.
- Die Parameter können verschiedene Datentypen handhaben, ohne dass der Client verschiedene Methoden oder Schnittstellen für die unterschiedlichen Datentypen aufrufen muss.
- Schnittstellen können einfacher entkoppelt werden. Mit allgemeinen Parametern hängt eine spezifische Schnittstelle nicht von einer anderen Schnittstelle oder Funktionalität ab.
- Allgemeine Parameter erfordern keine Skriptsprachenfunktionsaufruf zur Datenkonvertierung. Viele Skriptsprachen verwenden als Standarddatentyp **string**-Werte, was jedoch häufig eine falsche Annahme darstellt.

Die Entscheidung für allgemeine oder spezifische Datentypen ist keine einfache Entscheidung, da jede Option Stärken und Schwächen aufweist. Es gibt für den Einsatz der jeweiligen Datentypen keine Faustregel.

ADO-Objekte (Active Data Objects) beispielsweise, die in Visual Basic und bei der Skripterstellung verwendet werden, werden mit allgemeinen Datentypen geschrieben. Allgemeine Datentypen ermöglichen das Erstellen generischer Methoden für den Datenabruf. ADO arbeitet mit **Recordsets** von Datenbanken. Beim Werteabruf ist es unsinnig, Methoden wie beispielsweise **getDouble**, **getLong**, **getString** usw. zu schreiben. Die Schnittstelle würde zu lang und komplex. Es ist einfacher, mit Hilfe der so genannten **getValue**-Methode einen **VARIANT**-Wert

zurückzugeben. Und da der typische Datenprovider viele verschiedene Datentypen aufweist, ist die Verwendung von **VARIANT** angemessen.

Datentypen der OLE-Automatisierung Es besteht ein Unterschied zwischen den Parametertypen, die für benutzerdefinierte und für duale Schnittstellen verwendet werden können. Dies rührt daher, dass eine duale Schnittstelle in einer beliebigen Programmiersprache implementiert werden kann, aber nicht alle Konsumenten alle Datentypen unterstützen.

Es gibt eine Kategorie Datentypen, die als kompatibel mit der OLE-Automatisierung bezeichnet werden. Der Hauptdatentyp für die OLE-Automatisierung ist **VARIANT**, eine Struktur, die einen Referenzwert zum repräsentierten Datentyp enthält.

Die folgenden Datentypen werden am häufigsten eingesetzt:

- **long, BYTE, short, float, double, BOOL, DATE, char:** Diese Datentypen werden sehr oft bei der Darstellung von Zahlen eingesetzt.
- **BSTR:** Der **string**-Datentyp von COM.
- **IDispatch:** Ein Datentyp, der eine Referenz auf eine weitere COM-Schnittstelle enthält.
- **SafeArray:** Ein Datentyp, der ein Elementarray enthält, das entweder einem der obigen Typen oder **VARIANT** entspricht. Das Array ist »sicher« (safe), da es den Elementzähler, Start- und Endpunkt enthält.

Außerhalb der Datentypen für die OLE-Automatisierung befinden sich alle weiteren Datentypen. Obwohl diese im jeweiligen Kontext sinnvoll sind, können sie in vielen Programmierumgebungen nicht eingesetzt werden. Im Kontext von Windows DNA und Anwendungsentwicklung sollte jeder Datentyp ignoriert werden, der nicht überall verwendet werden kann, es sei denn, ein triftiger Grund spricht für dessen Verwendung.

Verwenden des Datentyps »Variant« Lassen Sie uns einen Schritt zurückgehen und untersuchen, wie der **VARIANT**-Datentyp in den verschiedenen Sprachen dargestellt wird. Sehen Sie sich die folgende IDL-Methodendeklaration an:

```
[ id(1)] HRESULT method2([in] VARIANT newVal);
```

Dieser **VARIANT**-Wert wird in Visual C++ folgendermaßen dargestellt:

```
STDMETHOD( method2)(/*[in]*/ VARIANT newVal);
```

Der **VARIANT**-Datentyp ist eine Rohstruktur, die Informationen zu den enthaltenen Daten aufweist:

In Visual J++ wird **VARIANT** wie folgt darstellt:

```
public void method2 (Variant newVal);
```

Der **Variant**-Datentyp ist eine Java-Klasse, die die Datenstruktur **VARIANT** kapselt. Visual J++ verwendet den COM-**VARIANT**-Datentyp nicht direkt, da es sich bei COM um ein Lowlevel-Modell handelt. Zur Vereinfachung der COM-Programmierung können Bibliotheken oder **Helper**-Elemente erstellt werden. Visual J++ hat das Helferobjekt **Variant** erstellt, mit dem die Details dieses Datentyps gehandhabt werden können.

In Visual Basic wird **VARIANT** wie folgt darstellt:

```
Public Sub method1(ByVal newVal As Variant)
```

In Visual Basic stellt der Datentyp **Variant** einen systemeigenen Datentyp dar. Dieser kann mit einer Reihe von Funktionen bearbeitet werden, aber da es sich um einen Datentyp handelt, kann er wie **integer**, **long** oder **double** direkt verwendet werden.

Unabhängig davon, welche Sprache der **VARIANT**-Datentyp verwendet, ist zusätzliche Arbeit erforderlich, und je nach Verwendung kann eine Verlangsamung die Folge sein. Dies rührt daher, dass die **VARIANT**-Datenstruktur einige Flags erfordert, damit die Struktur richtig erkannt wird.

Warum gibt es einen **VARIANT**-Datentyp? Über **VARIANT** wird eine Pseudostruktur erstellt und es werden flexible Parameter bereitgestellt. Im ADO-Beispiel wird das Problem durch den **VARIANT**-Datentypen perfekt gelöst. Werden Schnittstellen hauptsächlich in Skriptumgebungen verwendet, stellt das Verwenden des **VARIANT**-Datentyps die einfachere Option dar.

Verwenden des Objektdatentyps Sehen Sie sich die folgende Methodendeklaration an, die eine Objektreferenz enthält:

```
[ id(1)] HRESULT method3([in] IDispatch* newVal);
```

Das Objekt stellt einen Verweis auf die **IDispatch**-Schnittstelle dar. Es mag logischer erscheinen, **IUnknown** als Objekt zu referenzieren, aber das Problem hierbei ist, dass der Konsument bei einer Schnitstelle **IUnknown** wissen muss, welche Schnittstelle verwendet wird. Dies ist in vielen Skriptsprachen jedoch nicht möglich. Visual Basic hat Probleme, die **IUnknown**-Schnittstelle zu erkennen und erwartet, dass alle Objektverweise auf **IDispatch** basieren. Die **IDispatch**-Referenz ermöglicht der Skriptsprache das Abfragen verschiedener Methoden.

Visual C++ wird die Methode folgendermaßen deklariert:

```
STDMETHOD(method3)(/*[in]*/ IDispatch* newVal);
```

Hierbei handelt es sich um einen Zeiger, der auf die Schnittstelle selbst verweist.

Visual J++ wird die Methode folgendermaßen deklariert:

```
public void method3(Object newVal);
```

Der **Object**-Datentyp ist ein Java-Stammobjekt. Sie verwenden dieses, um einen Typecast für die gewünschte Schnittstelle durchzuführen.

Visual Basic wird die Methode folgendermaßen deklariert:

```
Public Sub method3(ByVal newVal As Object)
```

Wie in Visual J++ handelt es sich beim **Object**-Datentyp um ein Visual Basic-Stammobjekt. Im Gegensatz zu Visual J++ muss jedoch für das Objekt kein Typecast durchgeführt werden, um es zu verwenden. Visual Basic verwendet die **IDispatch**-Schnittstelle, wenn Methodenaufrufe für den **Object**-Datentyp durchgeführt werden.

Bei der Entwicklung der Schnittstelle kann es geschickter sein, feste Schnittstellendatentypen zu verwenden, wie nachfolgend gezeigt:

```
[ id(1)] HRESULT method3([in] ISimpleComponent* newVal);
```

Beide Methoden sind akzeptabel. Die einzige Tücke bei Verwenden einer festen Schnittstelle ist die, dass **ISimpleComponent** von **IDispatch** abgeleitet werden muss. Und obwohl es einfacher erscheinen mag, direkt die Schnittstellenzeiger zu verwenden, ist dies nur zutreffend, wenn Server und Konsument in Visual C++ geschrieben wurden. Visual C++ weist die Fähigkeit auf, Schnittstellenzeiger direkt zu verwenden. In anderen Sprachen kann eine **QueryInterface** vorhanden sein, in anderen Umgebungen, beispielsweise beim Scripting, kann nur **IDispatch** verwendet werden.

Verwenden des Datentyps »String« **String**-Werte stellen in einem C-Programmiersprachenkontext Puffer von Zeichen dar, die auf ein NULL-Zeichen enden. Die Zeichen können Einzel- oder Doppelbyte umfassen. Die Byteanzahl vor dem NULL-Zeichen definiert die Länge der Zeichenfolge. Bei COM und OLE-Automatisierung verwendet eine Zeichenfolge den **BSTR**-Datentyp. Ein **BSTR** unterscheidet sich von einer herkömmlichen Zeichenfolge darin, dass sie UNICODE-Daten enthält und die Länge der Zeichenfolge gespeichert wird. Sehen Sie sich diese Methodendeklaration an:

```
[helpstring("method method3")] HRESULT method3(BSTR param1);
```

In Visual C++ lautet die Deklaration:

```
STDMETHOD(method3)(BSTR param1);
```

In Visual J++ lautet die Deklaration folgendermaßen:

```
public void method3(String param1);
```

In Java handelt es sich bei dem **String**-Objekt um eine Java-Basisklasse.

In Visual Basic lautet die Deklaration folgendermaßen:

```
Public Sub method3( ByVal param1 as String)
```

Wie in Visual J++ stellt in Visual Basic das **String**-Objekt einen Bestandteil der Sprache dar.

Das Zeichenfolgenobjekt stellt weder in Visual J++ noch in Visual Basic ein Problem dar, da eine direkte Zuordnung vorhanden ist. In Visual C++ dagegen ist die Sache etwas komplizierter, da aufgrund einiger Legacyelemente viele Wrapperklassen für Zeichenfolgen vorhanden sind. In Kapitel 9 wird aufgezeigt, wie in Visual C++ Zeichenfolgen behandelt werden können.

Definieren von Rückgabewerten Bei der Deklaration von Funktionen möchten Sie üblicherweise Rückgabewerte für den Aufrufer deklarieren. In allen vorangegangenen IDL-Beispielen wiesen die Rückgabewerte die Form **HRESULT** auf. In COM-IDL ist es üblich, einen **HRESULT**-Wert zurückzugeben. **HRESULT** ist ein Behandlungsroutinenmechanismus.

COM behandelt Ausnahmen basierend auf dem Rückgabewert der aufgerufenen Methode. Diese Methode wird verwendet, da Ausnahmen nicht über COM-Schnittstellen hinweg erzeugt (**throw**-Anweisung) werden können. Hierbei handelt es sich um eine Entwurfsentscheidung für COM, die bereits früh getroffen wurde. Zu dieser Zeit war noch kein konsistenter Ausnahmemechanismus vorhanden, und nicht alle Sprachen unterstützten Ausnahmen. Daher müssen alle Methoden der COM-Schnittstellen einen Fehlercode zurückgeben, der angibt, ob der Methodenaufruf erfolgreich war. Obwohl mittlerweile fast alle Sprachen das Ausnahmekonzept unterstützen, kann in COM weiterhin keine Ausnahme erzeugt werden.

Zur Definition eines Rückgabewertes in COM muss die Funktion mit Hilfe dieser IDL deklariert werden:

```
[id(1)] HRESULT method2(long param1, [out,retval]long *retval);
```

In Visual Basic und Visual J++ benötigen Sie keinen Rückgabewert der Form **HRESULT** um anzugeben, ob eine Methode funktioniert, da die Laufzeit die Details von **HRESULT** maskiert. Hierbei wird die interne Fehlerbehandlung mit einem **HRESULT**-Wert verknüpft. Je nach Ergebnis wird ein Fehler oder **HRESULT** ausgegeben.

Die Implementierung der vorstehenden IDL-Methode in Visual Basic lautet folgendermaßen:

```
public function method2( ByVal param1 as long) as long
```

In Visual J++ lautet die Methode:

```
public int method2(int param1);
```

Da Visual C++ auf unterster Ebene arbeitet, ist die Methodendeklaration mit der IDL-Deklaration identisch und weist zwei Parameter auf.

Richtungsparameter Wenn ein COM-Konsument eine Implementierung aufruft, werden Daten vom Konsumenten an die Implementierung übergeben, und die Daten erfordern ein Marshaling und anschließendes Unmarshaling. Das Problem hierbei ist, dass die Daten einer kompilierten Programmierumgebung an eine weitere kompilierte Programmierumgebung gesendet werden müssen. Die Umgebungen können sich auf demselben Rechner oder in zwei völlig verschiedenen Netzwerken in verschiedenen Erdteilen befinden, daher müssen die Daten unter Umständen auf dem Rechner oder im Netzwerk gesendet werden. COM löst dieses Problem, da COM standortunabhängig ist. Der Preis hierfür ist jedoch, dass die Schnittstelle die Richtung des Parameters deklarieren muss.

Sehen Sie sich diese Funktionsdeklaration an:

```
HRESULT someMethod( long value);
```

Diese Methodendeklaration besagt, dass der Parameter vom Konsumenten an die Implementierung gesendet wird. Die COM-Marshalingroutine erstellt einen Puffer zur Speicherung des **long**-Wertes. Nehmen Sie jedoch einmal an, dass eine Zeichenfolge 40.000 Zeichen umfasst – ein derartiger Puffer ist nicht unbedeutend. Beim Verschieben des Puffers vom Konsumenten zur Implementierung und zurück wird Bandbreite benötigt. Beim vorangegangenen Methodenaufruf ist der Puffer nur erforderlich, wenn die Daten vom Konsumenten an die Implementierung gesendet werden. Für den Rückweg wird kein Puffer benötigt. Im Idealfall würde der Puffer lediglich an den Server gesendet und auf dem Rückweg ignoriert.

Sie können diese Optimierung erreichen, indem Sie die Richtung des Parameters angeben. Dies geschieht durch Anwenden der Richtungsattribute auf den Parameter. Der Standard lautet **in**, d.h. dass nur Daten vom Konsumenten zur Implementierung gesendet werden. Das Attribut **out** gibt an, dass der Datenfluss von der Implementierung zum Konsumenten verläuft. Das Kombinieren der Attribute **in** und **out** gibt an, dass die Daten vom Konsumenten zur Implementierung fließen, gelöscht und neu zugewiesen und anschließend von der Implementierung

zurück zum Konsumenten gesendet werden. Dieser Vorgang erscheint kompliziert, ist jedoch erforderlich, da COM nicht standortabhängig ist.

Nachfolgend eine Beispielschnittstelle, mit der alle drei Varianten implementiert werden:

```
interface ITest : IDispatch
{
   [id(1)] HRESULT onlyInput(long param1);
   [id(2)] HRESULT onlyOutput([out]long *output);
   [id(3)] HRESULT inputOutput([in, out]long *value);
};
```

Das Richtungsattribut ist bei der Implementierung einer Schnittstelle besonders wichtig, da die Sprachen die Schnittstelle anders behandeln, wenn ein Konsument die Schnittstelle beansprucht. In diesem Kapitel wird nur der Aspekt der Implementierung besprochen.

Wird Visual C++ zur Implementierung der Schnittstelle eingesetzt, ist die Methodendeklaration identisch mit der IDL. Die einzige Ausnahme besteht darin, dass die Deklaration auf der C++-Syntax basiert.

Bei der Implementierung mit Visual Basic sind jedoch Unterschiede vorhanden. Visual Basic kann die Methoden **onlyInput** und **inputOutput** problemlos verwenden, **onlyOutput** kann jedoch nicht eingesetzt werden. Wenn Sie versuchen, die **ITest**-Schnittstelle – so, wie sie ist – zu implementieren, erzeugt Visual Basic einen Kompilierungsfehler. Dieser Fehler entsteht, da keine **output**-Parameter verarbeitet werden können. Die Methode **inputOutput** weist in Visual Basic die folgende Signatur auf:

```
public sub inputOutput( value as long)
```

Diese Visual Basic-Methodendeklaration enthält in der Parameterliste kein **ByValue**. Dies rührt daher, dass bei einem **in/out**-COM-Parameter die Visual Basic-Laufzeitumgebung die Speicherverwaltung für den COM-Parameter übernimmt.

Visual J++ kann eine beliebige der in der Schnittstelle definierten Methoden implementieren. Es wird jedoch nicht zwischen **out** und **in/out** unterschieden. Die Methodendefinitionen lauten:

```
public void onlyOutput( int[] output);
public void inputOutput( int[] output);
```

Der Parameter wird als Array betrachtet. Auf diese Weise wird eine Referenz erzeugt, die praktisch einem Zeiger gleicht.

Eine spezielle Methode – die Eigenschaft

Bis zu diesem Punkt drehte sich alles um die Implementierung von Methoden, aber in COM können auch Eigenschaften definiert werden. Eigenschaften unterscheiden sich von Methoden darin, dass Sie einer Eigenschaft (wie einer Variablen) Werte zuordnen können. Obwohl Eigenschaften sehr effektiv sind, können Sie in einigen Sprachen nicht implementiert werden (beispielsweise in Visual C++ und Visual J++), da kein äquivalentes Konzept vorhanden ist. In Visual Basic und bei der Skripterstellung können Eigenschaften jedoch verwendet werden.

Sehen Sie sich folgenden JavaScript-Code an:

```
var temp = myObject.simpleProperty1;
```

Diese Syntax wird als **Property Get**-Anweisung bezeichnet, da der Wert der Eigenschaft vom Objekt abgerufen wird. Das Pendant hierzu ist die **Property Put**-Anweisung, die folgendermaßen lautet:

```
myObject.simpleProperty1 = temp;
```

MIDL behandelt Eigenschaften wie Funktionen mit den Attributen **propput** und **propget**, wie nachfolgend gezeigt wird:

```
[propget] HRESULT simpleProperty1([out, retval] VARIANT *pVal);
[propput] HRESULT simpleProperty1([in] VARIANT newVal);
```

In Visual C++ lautet die Methodendeklaration:

```
STDMETHOD(get_simpleProperty1)(/*[out, retval]*/ VARIANT *pVal);
STDMETHOD(put_simpleProperty1)(/*[in]*/ VARIANT newVal);
```

Die Methodendeklaration erhält die Präfixe **get_** und **put_**. Hierbei handelt es sich lediglich um ein Notationsdetail, das durch den MIDL-Compiler erzeugt wird. COM übersetzt die Eigenschaften für diese Methoden.

In Visual Basic lautet die Funktionsdeklaration folgendermaßen:

```
Public Property Let simpleProperty1(ByVal RHS As Variant)
Public Property Get simpleProperty1() As Variant
```

Visual Basic unterstützt die Notation von Eigenschaften, indem das **Property**-Attribut der Methodendeklaration vorangestellt wird. Das **Let**-Eigenschaftenschlüsselwort ist äquivalent mit dem IDL-Schlüsselwort **put**. Das **Get**-Eigenschaftenschlüsselwort stimmt mit dem IDL-Schlüsselwort **get** überein. Visual Basic weist ein weiteres Eigenschaftenschlüsselwort auf, **Set**, das dem **Let** entspricht, der Funktionsparameter weist hier jedoch kein **ByVal** auf.

In Visual J++ lautet die Funktionsdeklaration folgendermaßen:

```
public Variant getSimpleProperty1();
public void setSimpleProperty1(Variant pVal);
```

Der Schlüssel zur ordnungsgemäßen Funktion von Eigenschaften in Visual J++ liegt darin, den IDL-Methodennamen die Präfixe **get** und **set** voranzustellen. Zufälligerweise ist dies auch Bestandteil der Java Beans-Spezifikation. Untersucht man jedoch die Ergebnis-IDL, fällt etwas Interessantes auf:

```
[id(0x0000006c)]void setSimpleProperty1([in, out] VARIANT* Parameter0);
[id(0x0000006d)] VARIANT getSimpleProperty1();
[id(0x0000006e), propget] VARIANT simpleProperty1();
[id(0x0000006e), propput] void simpleProperty1([in] VARIANT rhs);
```

Der Visual J++-Compiler erzeugt zwei IDL-Deklarationen. Die erste (IDs 6C und 6D) behandelt jede der Funktionen, bei denen es sich um Eigenschaften handelt, wie Funktionen. Die zweite Deklaration (ID 6E) behandelt die Funktionen als Eigenschaften, indem die Attribute **propget** und **propput** hinzugefügt werden.

8.4 Resümee

In diesem Kapitel haben Sie eine Einführung in die COM-Schnittstellen erhalten. Sie haben gelernt, wie diese Schnittstellen in den verschiedenen Sprachen entwickelt und programmiert werden. Während dieses Prozesses haben Sie auch die jeweiligen IDL-Äquivalente kennen gelernt. Am einfachsten können Schnittstellen in IDL geschrieben werden. Das Verwenden anderer Sprachen ist weniger empfehlenswert, da durch jede Sprache zusätzliche Informationen hinzugefügt werden. Visual C++ fügt die wenigsten Zusatzinformationen hinzu, Visual Basic einige, Visual J++ weist so viele Extrainformationen auf, dass von der Verwendung von Visual J++ zur COM-Schnittstellendefinition abgeraten wird. Im nächsten Kapitel soll die Implementierung der verschiedenen Schnittstellen unter Verwendung unterschiedlicher Sprachen betrachtet werden.

Das Kapitel fehlt in dieser PDF-Datei.

Sorry, aber wir wollen ja auch das eine oder andere Buch verkaufen :-)

Index

C

D

E

F

G

H

I

J

P

Q

R

S

T

U

V

W

X

Z