1 Einleitung

Der universelle Transaktionsmonitor *open*UTM bietet als umfassende Middleware-Plattform alle Möglichkeiten, die Sie für das Design und den Einsatz transaktionsorientierter OLTP-Anwendungen benötigen. Darüber hinaus ist in *open*UTM die Funktionalität eines kompletten Message Queuing Systems integriert.

Durch optimale Performance, ausgereifte Security-Funktionen und höchste Verfügbarkeit ist *open*UTM auch für Anwendungsszenarien geeignet, bei denen herkömmliche OLTP-Systeme längst an ihre Grenzen stoßen.

*open*UTM bildet ein sicheres und effizientes Framework für moderne multi-tier Client/Server-Architekturen: *open*UTM sorgt für die Steuerung globaler Transaktionen, optimiert den Einsatz von System-Ressourcen (Arbeitsspeicher, CPU etc.), übernimmt das Management von parallelen Zugriffen, kümmert sich um Zugangs- und Zugriffskontrollen, den Aufbau von Netzverbindungen und vieles mehr.

Der Name „*open*UTM“ weist bereits auf wichtige Leistungsmerkmale hin:

open ... weil *open*UTM dem von X/Open definierten Referenzmodell für „Distributed Transaction Processing (DTP)“ entspricht und die von X/Open standardisierten offenen Schnittstellen unterstützt.

Universal ... weil *open*UTM unterschiedliche Welten verbindet und für die unterschiedlichsten Einsatzszenarien konzipiert ist: *open*UTM integriert heterogene Netze, Plattformen, Resource Manager und Anwendungen.

Transaction ... weil *open*UTM anwendungsübergreifend volle Transaktionssicherheit gewährleistet, entsprechend den klassischen ACID-Eigenschaften Atomicity, Consistency, Isolation und Durability, siehe auch Seite 12.

Monitor ... weil *open*UTM nicht „nur“ Transaktionsverarbeitung bietet, sondern das Management von verteilten, unternehmensweiten IT-Lösungen ermöglicht.

In Kapitel 2 dieses Handbuchs wird ausführlich auf die Leistungsmerkmale und Einsatzszenarien von *open*UTM eingegangen.

1.1 Konzept und Zielgruppen dieses Handbuchs

Das vorliegende Handbuch „Konzepte und Funktionen“ soll als Einstieg in die konkrete Arbeit mit *open*UTM dienen und richtet sich an alle, die mit *open*UTM noch nicht vertraut sind. Aber auch wenn Sie *open*UTM bereits kennen und einsetzen, können Sie dieses Handbuch verwenden, um sich einen Überblick über die Funktionsbreite und Leistungsfähigkeit des Produkts zu verschaffen.

In diesem Handbuch stehen nicht die syntaktischen Feinheiten einzelner Anweisungen oder die Details spezifischer Schnittstellen im Vordergrund. Vielmehr soll ein Überblick über die Leistungsfähigkeit und die Einsatzmöglichkeiten von *open*UTM gegeben werden. Mit diesem Überblick ausgestattet, werden Sie sich in den übrigen Handbüchern der *open*UTM-Reihe leicht zurechtfinden.

In Kapitel 2 werden Ihnen die Eigenschaften von *open*UTM kurz vorgestellt. Auf einige der dort angesprochenen Themen wird dann in den Kapiteln 3 bis 10 noch einmal etwas ausführlicher eingegangen.

*open*UTM ist für alle gängigen UNIX-Plattformen, für WindowsNT und für BS2000/OSD verfügbar. Die Funktionalität sowie die Anwenderschnittstellen gleichen sich dabei weitgehend. Deshalb gelten die Informationen in den ersten zehn Kapiteln dieses Handbuchs für alle Plattformen.

Die drei letzten Kapitel enthalten einige plattformspezifische Details, und zwar in Kapitel 11 für BS2000/OSD, Kapitel 12 für alle UNIX-Plattformen und Kapitel 13 für Windows NT. Die ausführlichen Verzeichnisse am Ende des Handbuchs - Fachwörter, Abkürzungen, Literatur und Stichwörter - sollen Ihnen den Umgang mit diesem Handbuch erleichtern.

Natürlich kann dieses Handbuch nicht alle Ihre Fragen beantworten. Aber es zeigt die Richtung, in der Sie die Lösungen auch für spezielle Probleme finden:

Alle Stellen, die mit dem nebenstehenden Symbol gekennzeichnet sind, verweisen auf umfassende und detaillierte Informationen zum jeweiligen Thema.

1.2 Wegweiser durch die Dokumentation zu *open*UTM

In diesem Abschnitt erhalten Sie einen Überblick über die Handbücher zu *open*UTM und zum Produktumfeld von *open*UTM. Die Abbildungen auf Seite 5 und Seite 6 geben einen aufgabenbezogenen Überblick, welche Handbücher Sie für welche Zwecke benötigen.

Es gibt Handbücher, die für alle Plattformen gültig sind sowie Handbücher, die jeweils für BS2000/OSD bzw. UNIX und Windows-NT gelten.

Zentrale Handbücher zu *open*UTM

Das Handbuch **Konzepte und Funktionen** enthält eine allgemeine Beschreibung aller Funktionen, Leistungen und Einsatzmöglichkeiten von *open*UTM. Sie erhalten alle Informationen, die Sie zum Planen des UTM-Einsatzes und zum Design einer UTM-Anwendung benötigen. Sie erfahren, was *open*UTM ist, wie man mit *open*UTM arbeitet und wie *open*UTM in die Betriebssysteme BS2000/OSD, UNIX und Windows NT eingebettet ist.

Zum Erstellen von Server-Anwendungen über die KDCS-Schnittstelle benötigen Sie das Handbuch **Anwendungen programmieren mit KDCS für COBOL, C und C++**, in dem die KDCS-Schnittstelle in der für COBOL, C und C++ gültigen Form beschrieben ist. Diese Schnittstelle umfaßt sowohl die Basisfunktionen des universellen Transaktionsmonitors als auch die Aufrufe für verteilte Verarbeitung. Es wird auch die Zusammenarbeit mit Datenbanken beschrieben.
Wenn Sie aus Ihrer UTM-Anwendung mit Transaktionssystemen von IBM kommunizieren wollen, benötigen Sie außerdem das Handbuch **Verteilte Transaktionsverarbeitung zwischen *open*UTM und CICS-, IMS- und LU6.2-Anwendungen**. Es beschreibt die CICS-Kommandos, IMS-Makros und UTM-Aufrufe, die für die Kopplung von UTM-Anwendungen mit CICS- und IMS-Anwendungen benötigt werden. Die Kopplungsmöglichkeiten werden anhand ausführlicher Konfigurations- und Generierungsbeispiele erläutert. Außerdem beschreibt es die Kommunikation über *open*UTM-LU62, sowie dessen Installation, Generierung und Administration.
Wollen Sie die X/Open-Schnittstellen nutzen, benötigen Sie das Handbuch **Anwendungen erstellen mit X/Open-Schnittstellen**. Es enthält die UTM-spezifischen Ergänzungen zu den X/Open-Programmschnittstellen TX, CPI-C und XATMI sowie Hinweise zu Konfiguration und Betrieb von UTM-Anwendungen, die X/Open-Schnittstellen nutzen. Ergänzend dazu benötigen Sie die X/Open-CAE-Specification für die jeweilige X/Open-Schnittstelle.

Zur Definition von Konfigurationen sowie zum Erzeugen und Betreiben von Anwendungen stehen Ihnen die Handbücher **Anwendungen generieren und betreiben** (jeweils für die Betriebssysteme UNIX/Windows NT und BS2000/OSD) zur Verfügung. Darin ist beschrieben, wie man eine UTM-Anwendung mit Hilfe des UTM-Tools KDCDEF generiert, ein UTM-Anwendungsprogramm bindet und startet, wie man sich bei einer UTM-Anwendung an-

und abmeldet und wie man Anwendungsprogramme strukturiert und im laufenden Betrieb austauscht. Außerdem enthält es die UTM-Kommandos, die dem Terminal-Benutzer zur Verfügung stehen.

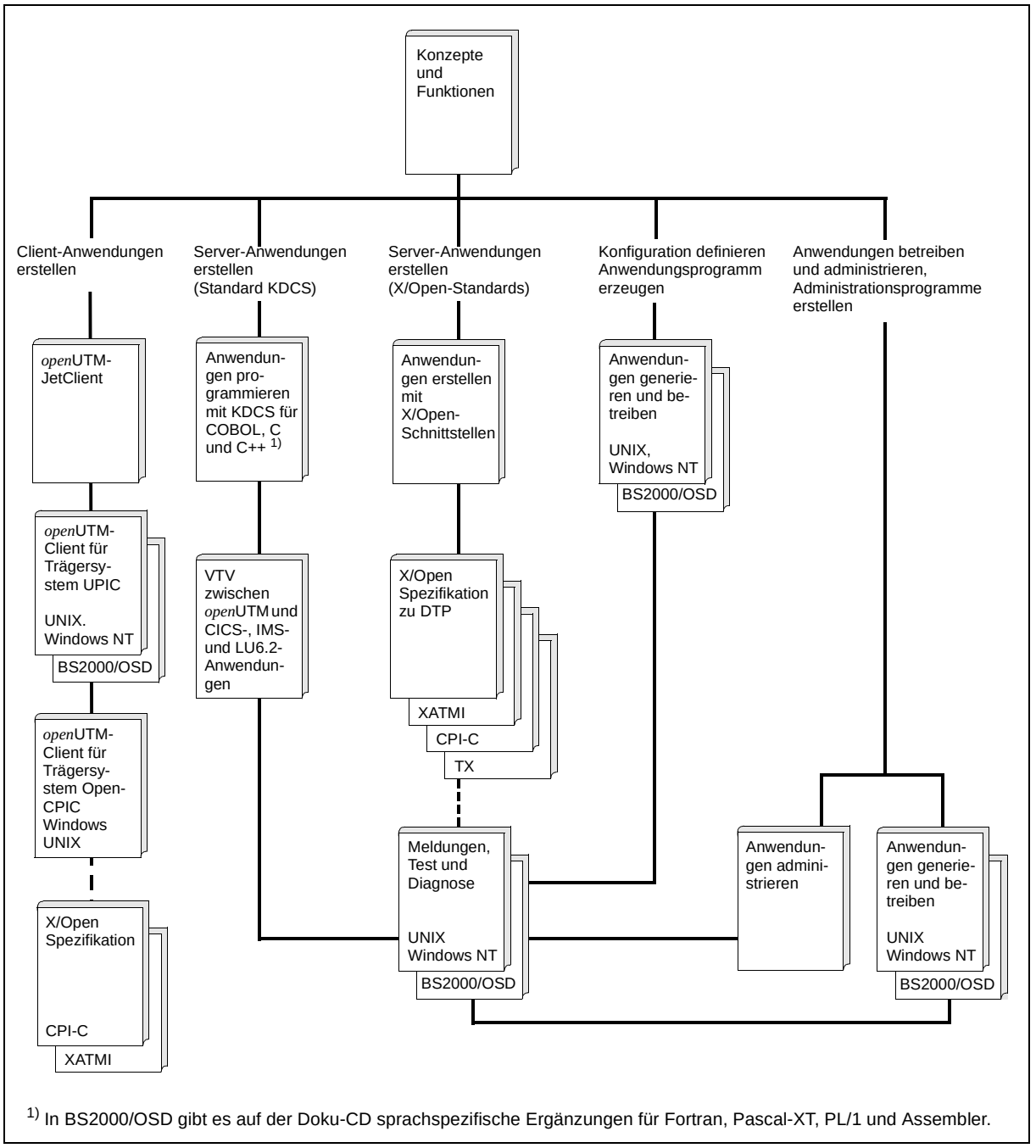
Für das Administrieren von Anwendungen finden Sie die Beschreibung der Programmschnittstelle zur Administration und die UTM-Administrationskommandos im Handbuch **Anwendungen administrieren**. Es informiert über die Erstellung eigener Administrationsprogramme und über die Möglichkeiten der zentralen Administration mehrerer Anwendungen. Darüber hinaus beschreibt es, wie Sie Message Queues und Drucker mit Hilfe der KDCS-Aufrufe DADM und PADM administrieren können.

Für die o.g. Aufgaben benötigen Sie außerdem die Handbücher **Meldungen, Test und Diagnose** (jeweils für die Betriebssysteme UNIX/Windows NT und BS2000/OSD). Sie beschreiben das Testen einer UTM-Anwendung, den Aufbau des UTM-Dumps, das Verhalten im Fehlerfall, das Meldungswesen von *open*UTM, sowie alle von *open*UTM ausgegebenen Meldungen und Returncodes.

Wenn Sie Client-Anwendungen für die Kommunikation mit UTM-Anwendungen erstellen wollen, stehen Ihnen folgende Handbücher zur Verfügung:

- Das Handbuch ***open*UTM-Client für Trägersystem UPIC** beschreibt Erstellung und Einsatz von Client-Anwendungen, die auf UPIC basieren. Neben der Beschreibung der Schnittstellen CPI-C und XATMI erhalten Sie Informationen, wie Sie ActiveX für die schnelle und einfache Programmerstellung nutzen können.
- Das Handbuch ***open*UTM-Client für Trägersystem OpenCPIC** beschreibt, wie man OpenCPIC installiert und eine OpenCPI-C-Anwendung konfiguriert. Es zeigt auf, was beim Programmieren einer CPI-C-Anwendung zu beachten ist und welche Einschränkungen es gegenüber der Programmschnittstelle X/Open CPI-C gibt.
- Das Handbuch ***open*UTM-Java Enterprise Technology Client** beschreibt Erstellung und Einsatz von Client-Anwendungen mit der Programmiersprache Java. Es erklärt, wie die JetClientClasses zur Kommunikation mit UTM-Anwendungen verwendet werden und erläutert den Einsatz des Verbindungsbausteins JetClientConnect auf Reliant UNIX.

Eine vollständige Liste aller Handbücher zu *open*UTM finden Sie im Literaturverzeichnis ab Seite 221.



1) In BS2000/OSD gibt es auf der Doku-CD sprachspezifische Ergänzungen für Fortran, Pascal-XT, PL/1 und Assembler.

Bild 1: Aufgabenbezogener Überblick über zentrale Handbücher zu *open*UTM

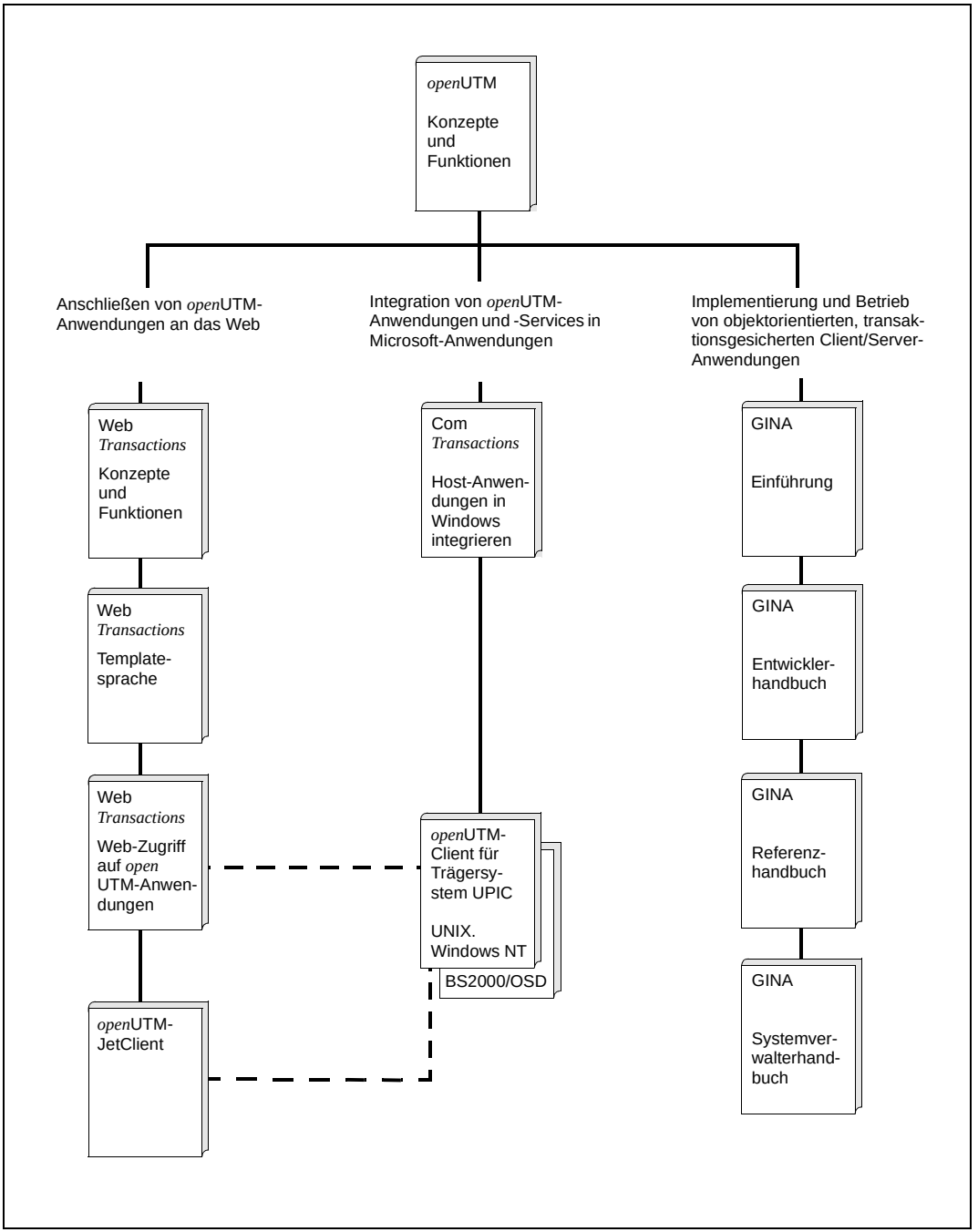


Bild 2: Aufgabenbezogener Überblick über die Handbücher zum Produktumfeld von *open*UTM

Handbücher zum Produktumfeld von *open*UTM

Zum Anschließen neuer und bestehender UTM-Anwendungen an das Web mit dem Produkt Web*Transactions* benötigen Sie die folgenden Handbücher:
Das einführende Handbuch **Web*Transactions* - Konzepte und Funktionen** gibt Ihnen einen Überblick über die Leistungsfähigkeit und Einsatzmöglichkeiten von Web*Transactions*. Es informiert Sie über die Eigenschaften und Funktionsweise von Web*Transactions*, erläutert das Objekt-Konzept und den dynamischen Ablauf einer Web*Transactions*-Sitzung.
Im Handbuch **Web*Transactions* - Templatesprache** sind alle Sprachmittel der Templatesprache WTML erläutert. Es enthält zahlreiche Beispiele, die die Sprachmittel veranschaulichen und Ihnen den Einsatz erleichtern.
Das Handbuch **Web*Transactions* - Web-Zugriff auf *open*UTM-Anwendungen** beschreibt die notwendigen Schritte für den Anschluß von UTM-Dialoganwendungen an das Web. An einem konkreten Beispiel werden diese Schritte nochmals verdeutlicht.

Für die Integration von *open*UTM-Anwendungen und -Services in Microsoft-Anwendungen mit dem Produkt Com*Transactions* steht Ihnen das Handbuch **Com*Transactions* - Host-Anwendungen in Windows integrieren** zur Verfügung. Es beschreibt neben der Installation und Konfiguration auch die Konzepte und Komponenten von Com*Transactions*. Außerdem wird die Administration und das Entwickeln von Client-Anwendungen erläutert.

Mit dem Produkt GINA (General Interface for Network Applications) können Sie objektorientierte, transaktionsgesicherte Client/Server-Anwendungen erstellen und betreiben.
GINA setzt auf *open*UTM und *open*UTM-Client als Basisprodukte auf. Zu GINA gibt es folgende Handbücher:
Das Handbuch **GINA - Einführung** gibt eine Übersicht über die Leistungsmerkmale von GINA und beschreibt die verschiedenen Komponenten.
Im **GINA - Entwicklerhandbuch** sind ausführlich die Konzepte für die Entwicklung von GINA-Applikationen sowie handlungsorientierte Anweisungen und Hilfen für den Einsatz beschrieben. Ein zusammenhängendes Beispiel für eine GINA-Applikation verdeutlicht das Prinzip und die Möglichkeiten von GINA.
Das **GINA - Referenzhandbuch** enthält eine formale Beschreibung sämtlicher Schnittstellen der GINA-Komponenten jeweils in alphabetischer Reihenfolge.
Im **GINA - Systemverwalterhandbuch** ist die Installation von GINA und von GINA-Applikationen erläutert. Es beschreibt außerdem, wie das Kommunikationssystem und GINA-Applikationen konfiguriert werden. Darüber hinaus ist der Betrieb von GINA-Applikationen in einer Client/Server-Umgebung dargestellt.

Readme-Dateien

Funktionelle Änderungen und Nachträge der aktuellen Produktversion zu diesem Handbuch entnehmen Sie bitte ggf. der produktspezifischen Readme-Datei.

- BS2000/OSD:

  Sie finden die Informationen auf Ihrem BS2000-Rechner entweder in der Freigabemitteilung (Dateiname SYSFGM.*produkt.version.sprache)* oder einer Readme-Datei (Dateiname SYSRME.*produkt.version.sprache)*. Die Benutzerkennung, unter der sich die Datei befindet, erfragen Sie bitte bei Ihrer zuständigen Systembetreuung. Die Datei können Sie mit dem Kommando /SHOW-FILE oder mit einem Editor ansehen oder auf einem Standarddrucker mit folgendem Kommando ausdrucken:

  ```
  /PRINT-DOCUMENT dateiname,LINE-SPACING=*BY-EBCDIC-CONTROL
  ```

  bei SPOOL -Versionen kleiner 3.0A:

  ```
  /PRINT-FILE FILE-NAME=dateiname,LAYOUT-CONTROL=
      PARAMETERS(CONTROL-CHARACTERS=EBCDIC)
  ```

- UNIX:

  Letzte Änderungen zum vorliegenden Handbuch können in einer sogenannten man-Datei (Handbuchergänzungsdatei) hinterlegt sein. In diesem Fall enthält die Lieferinformation einen Hinweis darauf.

  Falls das Readme-Package *SIreadmeM* installiert wurde, finden Sie auf Ihrem UNIX-System unter dem Dateiverzeichnis */opt/readme/produktname.sprachkürzel* readme- bzw. man-Dateien. Bei UNIX-Systemen, auf denen kein eigenes Readme-Package geliefert wird, stehen die Dateien im UTM-Installationsverzeichnis.

  Readme- und man-Dateien sind ASCII-Dateien, die Sie mit einem Editor ansehen oder auf einem Standarddrucker ausdrucken können.

- Windows NT:

  Die Readme-Datei und ggf. weitere Dateien wie z.B. eine Handbuchergänzungsdatei finden Sie im UTM-Installationsverzeichnis unter *\Docs\sprache*.

1.3 Neue Konzepte und Funktionen

In diesem Handbuch sind folgende neuen Konzepte und Funktionen von *open*UTM und dessen Umfeld beschrieben:

Windows NT als *open*UTM-Plattform

- *open*UTM ist mit seiner Server-Funktionalität auch auf Windows NT verfügbar. Einzelheiten finden Sie in Kapitel „openUTM in Windows NT“ auf Seite 173.

Server-Funktionen

- TCP/IP-Sockets-Anwendungen können sich als Transportsystem-Anwendung an *open*UTM anschließen, siehe Seite 64.
- UPIC-Clients können beim Anmelden den Event-Service SIGNON nutzen, siehe z.B. Seite 90.
- Transportsystem-Anwendungen können sich über den Anmeldevorgang mit einer UTM-Benutzerkennung an die UTM-Anwendung anmelden, siehe Seite 64.
- Verschlüsselung: Bei der Kommunikation mit Clients und Terminals (BS2000/OSD) können Berechtigungs- und Benutzerdaten verschlüsselt übertragen werden. Näheres finden Sie auf Seite 127.
- Die Accounting-Funktionen und der Meßmonitor KDCMON stehen auch für *open*UTM auf UNIX und Windows NT zur Verfügung, siehe Seite 28 und Seite 32.
- Die Prioritätensteuerung wurde wesentlich verfeinert, sowohl für Dialog- als auch für Hintergrund-Verarbeitung, siehe Seite 74.

Clients mit Trägersystem UPIC

- UPIC-Clients können die Verschlüsselungsfunktionen von *open*UTM nutzen.
- Wahlweise kann für UPIC-Clients beim Anmelden der SIGNON-Service durchlaufen werden.

Administrationsprogramm WinAdmin

- WinAdmin bietet neben Administrations-Funktionen auch eine Generierungsschnittstelle. Damit können Sie vom PC aus eine neue KDCFILE erzeugen oder mit KDCUPD eine Änderungsgenerierung vornehmen.
- Mit WinAdmin können UTM-Anwendungen auch gestartet werden, siehe Seite 112.

Web-Anbindung von *open*UTM

- Mit Web*Transactions* und *open*UTM-JetClient können UTM-Services vom Web aus genutzt oder Clients auf Java-Basis erstellt werden. Weitere Informationen finden Sie auf Seite 37 und Seite 53.

Integration in die Microsoft-Welt

- *open*UTM kann über die ActiveX-Schnittstelle des UPIC-Client oder über das Produkt Com*Transactions* in die Microsoft-Welt integriert werden, siehe Seite 39 und Seite 51. Com*Transactions* setzt auf UPIC auf.

Anbindung weiterer Systeme

- Die Kopplung mit R/3 ist erstmals ausführlich beschrieben, siehe Seite 40.
- Die Anbindung an CICS/IMS ist auch über das Protokoll LU6.2 und *open*UTM-LU62 möglich, siehe Seite 41 und Seite 62.
- *open*UTM kann mit MQSeries-Anwendungen gekoppelt werden, siehe Seite 42.
- Mit dem SNMP-Subagenten für *open*UTM kann *open*UTM in ein systemübergreifendes Anwendungsmanagement eingebunden werden, siehe Seite 43.

2 *open*UTM - Leistungsüberblick

Klassische Beispiele für Transaktionsmonitor-Anwendungen sind OLTP-Anwendungen (**O**n**L**ine **T**ransaction **P**rocessing) im Bankbereich oder Reisebuchungssysteme: Bankkunden veranlassen Überweisungen zwischen Konten unterschiedlicher Banken an unterschiedlichsten Orten - nicht nur am Schalter, sondern auch über Telefon oder World Wide Web. Reisebüros buchen international Flüge und Hotelzimmer mit Online-Zugriff auf Datenbanken von Fluglinien und Hotelketten. Immer häufiger bilden Transaktionsmonitore auch die Basis für unternehmensweite, integrierte IT-Lösungen auf der Grundlage von Client/Server-Konzepten.

Der universale Transaktionsmonitor *open*UTM wird aber auch in vielen anderen Bereichen eingesetzt, beispielsweise für Lagerhaltungs- und Produktionssteuerungssysteme.

*open*UTM eignet sich nicht nur für das Design und den Einsatz von dialogbasierten Anwendungen, sondern ermöglicht durch integrierte Message Queuing Funktionen auch Services, die entkoppelt vom Online-Dialog ablaufen. Workflow-Modellierung, Mobile Computing und Data Warehouse Lösungen können mit *open*UTM realisiert werden.

*open*UTM bildet somit eine optimale Middleware-Plattform für die verschiedensten Anwendungsszenarien: *open*UTM enthält Funktionen für die Erstellung von Services und Client-Programmen sowie mächtige und komfortable Schnittstellen für die Konfiguration und Administration von Anwendungen.

*open*UTM steuert den transaktionsgesicherten Informationsaustausch zwischen Clients, Anwendungen und Ressourcen und garantiert selbst bei komplexen, verteilten Strukturen Zuverlässigkeit, Verfügbarkeit und Performance.

*open*UTM läßt sich in das World Wide Web integrieren („Web-to-Host“), so daß UTM-Services z.B. über Web-Browser aufgerufen werden können.

2.1 Transaktionskonzept und Restart-Funktionen

Zu den wichtigsten Aufgaben von *open*UTM gehört es, die Konsistenz und Integrität der Anwendungsdaten sicherzustellen - auch dann, wenn Probleme auftreten, wie z.B. Netzstörungen oder Systemausfälle. Deshalb unterliegen alle Abläufe unter *open*UTM dem Transaktionskonzept.

Eine Transaktion ist eine Zusammenfassung von Arbeitsschritten, die ganz bestimmte Eigenschaften aufweist. Diese Eigenschaften werden üblicherweise entsprechend ihren Anfangsbuchstaben als ACID-Eigenschaften bezeichnet:

- **A**tomicity
  Die zu einer Transaktion zusammengefaßten Arbeitsschritte bilden eine atomare Einheit: Entweder werden alle Arbeitsschritte einer Transaktion ausgeführt oder gar keiner (Alles-oder-Nichts-Regel). Falls eine Transaktion aus irgendeinem Grund nicht vollständig durchgeführt werden kann, wird die Transaktion zurückgesetzt (rollback), d.h., alle Daten werden auf den Zustand vor Beginn der Transaktion zurückgesetzt.

- **C**onsistency
  Die Arbeitsschritte werden korrekt durchgeführt. Falls die Ressourcen (Datenbanken, Drucker, Message Queues etc.) sich vor Transaktionsbeginn in einem konsistenten Zustand befanden, befinden diese sich auch nach Ende der Transaktion in einem konsistenten Zustand.

- **I**solation
  Falls mehrere Transaktionen zeitlich überlappend stattfinden, werden konkurrierende Änderungen so ausgeführt, als wären die Transaktionen serialisiert, d.h. ohne Überlappung ausgeführt worden. Zwischenstände werden nicht für andere Transaktionen sichtbar. Die Transaktionen sind gegeneinander isoliert.

- **D**urability
  Ist eine Transaktion einmal erfolgreich abgeschlossen, so sind die vorgenommenen Änderungen dauerhaft, d.h., sie überstehen auch Systemausfälle. Beim Transaktionsende werden hierzu alle Änderungen gesichert. Das Transaktionsende stellt daher immer auch einen Sicherungspunkt dar.

So erfüllt z.B. eine Geldüberweisung, in der die Arbeitsschritte „Betrag abbuchen“ und „Betrag gutschreiben“ zusammengefaßt sind, die Atomicity-Bedingung, wenn nach jeder Abbuchung immer auch eine Gutschrift erfolgt. Die Überweisung erfüllt die Consistency-Bedingung, wenn sowohl Abbuchung als auch Gutschrift in der vorgeschriebenen Form und an der richtigen Stelle eingetragen werden. Die Isolation-Bedingung ist erfüllt, wenn keine Gefahr besteht, daß während der Bearbeitung parallele Buchungen die beteiligten Kontostände lesen und verändern. Die Durability-Bedingung schließlich ist erfüllt, wenn die Aktualisierung bei Systemstörungen, wie z.B. einem Stromausfall, nicht verlorengeht.

*open*UTM bezieht alle Ressourcen komplett in die Transaktionssteuerung ein und ist so in der Lage, umfassende Restart-Funktionen (automatischer Wiederanlauf) zu bieten.

Beim automatischen Wiederanlauf synchronisiert *open*UTM nicht nur die Daten in den Datenbanken mit den Daten in der Anwendung neu, sondern auch unterbrochene Services, lokale Betriebsmittel (z.B. Speicherbereiche), Queues, Kommunikationsverbindungen und alle Frontends wie Alpha-Terminals, PCs, Workstations und Drucker.

Diese Restart-Funktionalität ermöglicht es beispielsweise, daß selbst nach Systemausfällen (PCs, Netz, Server) alle Client-PCs mit dem jeweils letzten konsistenten PC-Bildschirm weiterarbeiten können.

Weitere Informationen zum Thema „Wiederanlauf“ finden Sie in diesem Handbuch im Abschnitt 10.5 auf Seite 135.

2.2 Zusammenarbeit mit Datenbanken und Resource Managern

Eine der wesentlichen Funktionen eines Transaktionsmonitors ist die koordinierte, gesicherte Zusammenarbeit mit Resource Managern (zum Begriff „Resource Manager“ siehe Abschnitt 2.6 auf Seite 25). *open*UTM unterstützt die von X/Open standardisierte Schnittstelle XA und ermöglicht so die Zusammenarbeit mit allen Resource Managern, die diese Schnittstelle anbieten, auch gemischt innerhalb einer Transaktion. An eine UTM-Anwendung können gleichzeitig mehrere Resource Manager angeschlossen werden.

Unter den Resource Managern nehmen Datenbank-Systeme die zentrale Rolle ein, daher wird die Zusammenarbeit mit Datenbank-Systemen im Mittelpunkt dieses Abschnitts stehen. Die Datenbank-Systeme, mit denen eine UTM-Anwendung zusammenarbeiten soll, werden bei der Generierung der UTM-Anwendung festgelegt.

Unterstützte Datenbank-Systeme

*open*UTM (BS2000/OSD) unterstützt die Koordination mit folgenden Datenbank-Systemen:

- UDS/SQL
- SESAM/SQL
- ORACLE
- CIS
- PRISMA
- LEASY (das Dateisystem LEASY verhält sich gegenüber *open*UTM wie ein Datenbank-System)

*open*UTM (UNIX) unterstützt die Koordination mit folgenden Datenbank-Systemen:

– INFORMIX
– ORACLE
– alle anderen Datenbank-, Datei- und Queuing-Systeme, die die XA-Schnittstelle (X/Open-Standard) unterstützen, z.B.: SYBASE, ADABAS, DB2, O_2, SQL-Server, ISAM/XA, MQSeries, ...

*open*UTM (Windows NT) unterstützt die Koordination mit folgenden Datenbank-Systemen:

– INFORMIX
– ORACLE
– Microsoft SQL Server

Koordination zwischen UTM-Transaktionen und Datenbank-Transaktionen

Die in einer Datenbank gespeicherten Datensätze unterliegen nicht den Sperr- und Sicherungsmechanismen von *open*UTM, sondern denen des jeweiligen Datenbank-Systems. Um die Konsistenz der Daten übergreifend sicherzustellen, müssen die UTM-Transaktionen mit den Transaktionen aller beteiligten Datenbank-Systeme synchronisiert werden. *open*UTM verwendet hierfür das **Two-Phase-Commit**-Verfahren: Ist die Bearbeitung einer lokalen Transaktion abgeschlossen, so wird diese Transaktion zunächst in den Zustand „vorläufiges Transaktionsende“ versetzt. Erst wenn alle beteiligten Transaktionen diesen Zustand erreicht haben, wird das globale Transaktionsende (Commit) gesetzt. Bild 3 verdeutlicht dieses Verfahren anhand eines einfachen Beispiels.

UTM-Transaktion und Datenbank-Transaktionen können daher als Teile einer gemeinsamen Transaktion aufgefaßt werden. Diese gemeinsame Transaktion kann nur dann erfolgreich beendet werden, wenn alle ihre Teile endgültig abgeschlossen werden können. Wenn also ein Teil - UTM- oder eine Datenbank-Transaktion - nicht beendet werden kann, dann wird die Gesamt-Transaktion zurückgesetzt. Das bedeutet, daß in der Transaktion durchgeführte Änderungen in den Datenbanken und in UTM-Speicherbereichen rückgängig gemacht und Aufrufe von Asynchron- und Ausgabeservices annulliert werden.

Für den Programmierer entsteht keinerlei Aufwand für die Koordination zwischen *open*UTM und Datenbank-Systemen.

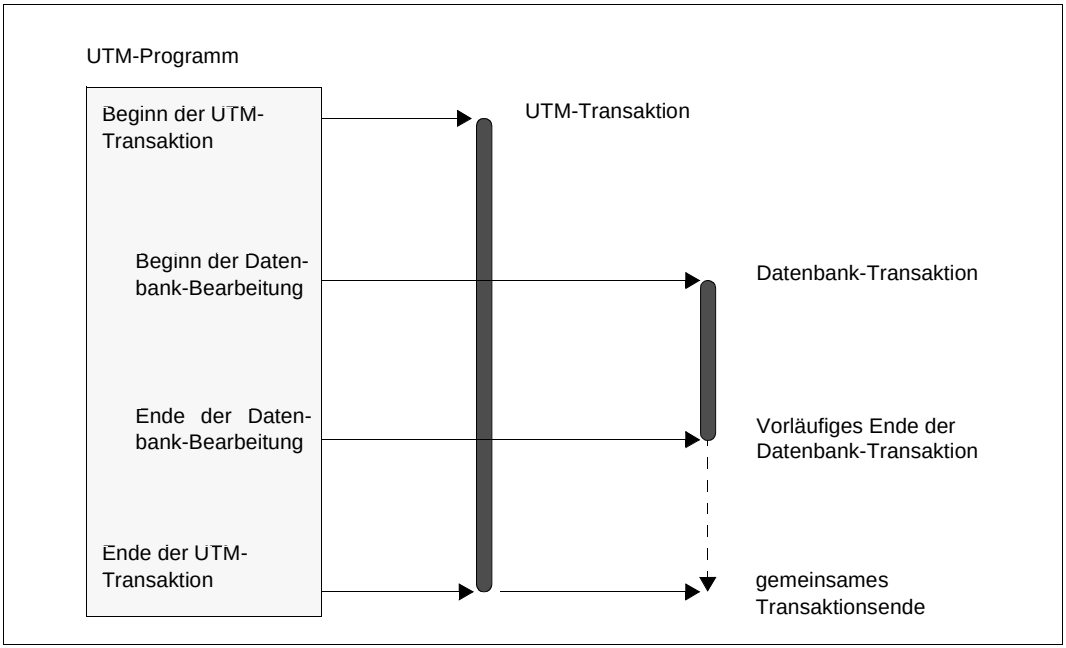


Bild 3: Koordination von UTM- und Datenbank-Transaktion über Two-Phase-Commit

Koordination mit verteilten und heterogenen Datenbank-Systemen

Die Koordination mit einem Resource Manager gehört wohl zum Funktionsumfang eines jeden Transaktionssystems. *open*UTM bietet Ihnen aber weitergehende Möglichkeiten:

Eine UTM-Anwendung kann mit mehreren Datenbank-Systemen koordiniert zusammenarbeiten. Das bedeutet, daß **ein** UTM-Serviceprogramm Aufrufe an **verschiedene** Datenbank-Systeme enthalten darf. Darüber hinaus ist es im Rahmen von verteilter Verarbeitung möglich, Daten von mehreren unterschiedlichen Datenbank-Systemen auf verschiedenen Rechnern innerhalb einer Transaktion zu bearbeiten. *open*UTM erlaubt somit die gesicherte Zusammenarbeit mit verteilten und heterogenen Datenbanken auf allen gängigen Hardware- und Betriebssystem-Plattformen - sogar innerhalb desselben Service-Programms.

Durch diese Leistungen bietet Ihnen *open*UTM eine höhere Flexibilität als andere Transaktionsmonitore: Bestehende Datenlogistik muß bei Integration in UTM-Anwendungen nicht modifiziert werden und bei Erweiterungen der IT-Landschaft ergibt sich ein wesentlich größerer Gestaltungsspielraum.

Fehlerbehandlung und Diagnose bei der Zusammenarbeit mit Datenbank-Systemen

Treten bei der Zusammenarbeit mit Datenbank-Systemen Fehler auf, so übernimmt *open*UTM die Fehlerbehandlung - der Programmierer muß keine speziellen Vorkehrungen treffen.

Das Datenbank-System informiert *open*UTM, ob es einen Aufruf erfolgreich durchführen konnte. Ist ein Fehler aufgetreten, so prüft *open*UTM den Fehlergrad und reagiert entsprechend: *open*UTM setzt die Transaktionen auf den letzten Sicherungspunkt zurück und gibt Meldungen aus, die Hinweise auf die Fehlerursachen enthalten.

Zur leichteren Diagnose wird für jeden Datenbank-Aufruf ein Eintrag in einem Bereich des UTM-Dumps erzeugt. Ein spezielles Tool ermöglicht es, den UTM-Dump gezielt und komfortabel auszuwerten.

Interne Schnittstelle zwischen *open*UTM und Datenbank-Systemen

*open*UTM steuert die Zusammenarbeit mit Datenbank-Systemen über eine einheitliche und neutrale Schnittstelle und ist damit entkoppelt von den implementierungsabhängigen Spezifika der unterschiedlichen Datenbank-Systeme.

Unter UNIX ist dies die von X/Open standardisierte XA-Schnittstelle, im BS2000/OSD die funktionell vergleichbare Schnittstelle IUTMDB.

Unterstützung weiterer Resource Manager

Datenbank-Systeme sind nur ein bestimmter Typ von Resource Managern.
*open*UTM bezieht neben Datenbanken auch transaktionsorientierte Dateisysteme (LEASY, ISAM/XA, ...), Message Queues, lokale Speicher, Logging-Dateien und Netzverbindungen in die Transaktionssicherung ein. Dabei ist es gleichgültig, ob es sich um einen *open*UTM Resource Manager handelt (*open*UTM Message Queues, lokale Speicher, Logging-Dateien) oder um einen externen Resource Manager, wie z.B. Message Queuing-Systeme anderer Hersteller: *open*UTM koordiniert in jedem Fall die Transaktionssicherung über die Grenzen von Anwendungen, Betriebssystemen und Hardware-Plattformen hinweg.

2.3 Framework für Client/Server-Computing

Client/Server-Computing hat sich zu einem der zentralen Einsatzkonzepte der modernen Informationstechnik entwickelt. Client/Server-Lösungen, die auf der Ebene von Arbeitsgruppen oder Einzelabteilungen entstanden sind, werden zunehmend ausgebaut und zusammengeführt. Andererseits werden monolithischen Mainframe-Anwendungen immer öfter offene Systeme an die Seite gestellt (Surrounding). In manchen Fällen ist es Ziel, Mainframe-Anwendungen ganz auf offene Systeme zu verlagern (Downsizing).

Client/Server-Computing bildet in vielen Fällen bereits die Grundlage für umfassende, unternehmensweite IT-Strategien.

Für die Attraktivität von Client/Server-Lösungen gibt es eine Reihe von Gründen:
– flexible Anpassungsmöglichkeiten an neue Unternehmensstrukturen
– Einsparungspotential bei den Hardwarekosten
– weitgehende Herstellerunabhängigkeit
– wachsendes Software-Angebot für offene Systeme
– Möglichkeit, komfortable grafische Oberflächen zu nutzen

Mit dem Vordringen von Client/Server-Lösungen in mission-critical Bereiche, d.h. in Bereiche, die das operative Geschäft eines Unternehmens unmittelbar betreffen, wachsen die Anforderungen an Verfügbarkeit, Datensicherheit und Performance. *open*UTM bildet eine sichere, homogene und hochperformante Basis für die unterschiedlichsten Client/Server-Architekturen.

Als klassisches Middleware-Produkt schließt *open*UTM die Kluft zwischen Mainframes und offenen Systemen. Es macht typische Mainframe-Qualitäten, wie kurze Antwortzeiten, hohen Durchsatz, Verläßlichkeit und Sicherheit, in der Welt offener Systeme verfügbar.

*open*UTM ist hervorragend geeignet für **Multi-Tier-Architekturen**, die eine Verteilung der Business-Logik (Verarbeitung) auf mehrere Server vorsehen (siehe auch Seite 45ff). Dabei können als Server-Plattformen sowohl Mainframes als auch offene Systeme benutzt werden.

*open*UTM übernimmt dabei die Funktion eines verteilten High-Level-Betriebssystems. Beim Design von Anwendungen kann auf ein Fundament mächtiger Funktionen aufgebaut werden: *open*UTM sorgt für die Steuerung globaler Transaktionen, optimiert den Einsatz von System-Ressourcen (Arbeitsspeicher, CPU etc.), übernimmt das Management von parallelen Zugriffen, kümmert sich um Zugangs- und Zugriffskontrollen, den Aufbau von Netzverbindungen und vieles mehr.

Da sich die Spezifika heterogener Netzwerke und unterschiedlicher Plattformen nicht in den Anwendungskomponenten niederschlagen, bleiben die einzelnen Komponenten umgebungsunabhängig, flexibel austauschbar und universell einsetzbar. Da sich mit *open*UTM die unterschiedlichsten Client/Server-Architekturvarianten realisieren lassen, ist dafür gesorgt, daß das Potential, das im Client/Server-Konzept steckt, auch in der Praxis ausgeschöpft werden kann.

Detaillierte Informationen über Client/Server-Computing mit *open*UTM finden sie in diesem Handbuch in einem eigenen Kapitel zu diesem Thema (Kapitel 3 auf Seite 45ff).

2.4 Message Queuing

Message Queuing (MQ) ist eine Form der Kommunikation, bei der Nachrichten (Messages) nicht unmittelbar, sondern über zwischengeschaltete Queues ausgetauscht werden. In *open*UTM sind mit dem Konzept der **asynchronen Verarbeitung** ausgereifte Message Queuing Funktionen integriert.

Der Begriff „asynchron“ wird oft für Programmierungsformen gebraucht, bei denen das Sender-Programm nach dem Verschicken einer Nachricht nicht auf die Antwort des Empfängers warten muß (non-blocking conversations). *open*UTM unterstützt auch diesen Programmierstil, bietet jedoch durch seinen transaktionsgesicherten Deferred Delivery Mechanismus darüber hinaus „echte“ asynchrone Verarbeitung: Sender und Empfänger können zeitlich und räumlich entkoppelt ablaufen, die Übermittlung der Nachricht wird garantiert, unabhängig davon, ob gerade eine Netzverbindung besteht oder nicht. Nachrichten werden in eine Queue eingetragen und solange zwischengespeichert, bis Leitung und Empfänger einsatzbereit sind. Zusätzliche Flexibilität bietet *open*UTM durch die Möglichkeit, beim Message Queuing mit Zeitsteuerung und Priority Scheduling zu arbeiten.

In UTM-Anwendungen stehen die MQ-Funktionen automatisch zur Verfügung, Queues oder Queuemanager müssen nicht eigens generiert und konfiguriert werden. Da *open*UTM mit service-spezifischen Queues arbeitet, müssen Sie für die Zuordnung von MQ-Nachrichten zu Empfängerprogrammen keine speziellen Trigger- oder Polling-Mechanismen konstruieren.

Dialogverarbeitung und Message Queuing sind frei kombinierbar: Innerhalb eines Dialogservices können MQ-Aufträge abgesetzt werden, ein asynchron gestartetes Serviceprogramm wiederum kann einen synchronen Dialog (conversation) mit einem fernen Dialog-Service führen. Besonders umfangreiche oder zeitunkritische Aufgaben - wie z.B. langsame Druckausgaben, langlaufende Statistikberechnungen, Sortierarbeiten, etc. - lassen sich so von Online-Dialogen entkoppeln, ohne daß auf Transaktionssicherheit verzichtet werden müßte.

Realisierung moderner Einsatzkonzepte

Mit seiner Message Queuing-Funktionalität ist *open*UTM hervorragend geeignet für moderne Einsatzkonzepte wie beispielsweise **Workflow-Strategien**: Betriebliche Arbeitsabläufe werden in Schritte gegliedert und Zwischenstände von Anwendung zu Anwendung weitergereicht. Das Absender-Programm erwartet dabei nicht unbedingt eine Antwort vom Empfänger und vielleicht muß der nächste Schritt nicht sofort begonnen werden. Absolut verläßlich muß jedoch sein, daß der Zwischenstand auch wirklich beim Empfänger eintrifft. Der transaktionsgesicherte Queuing-Mechanismus von *open*UTM gewährleistet dies. Auch wenn das Netz gerade nicht verfügbar oder die Empfänger-Anwendung offline ist: *open*UTM garantiert, daß keine Message verlorengeht oder verdoppelt wird.

Durch diese Unabhängigkeit von der Qualität und Verfügbarkeit der Verbindungsstrecke bildet *open*UTM eine sichere Basis-Middleware für **Mobile Computing**: Mobile Clients, z.B. Anwendungen, die auf Laptops laufen, können mit Servern zusammenarbeiten, ohne hierzu ständig mit ihnen verbunden sein zu müssen. Durch die transaktionsgesicherte lokale Sammlung von Messages wird eine geblockte Übertragung ermöglicht. Aufschaltzeiten werden hierdurch reduziert und damit Leitungskosten gesenkt.

Auch **Data Warehouse-Lösungen** und **Decision Support-Systeme** werden durch die Message Queuing-Funktionen von *open*UTM wirkungsvoll unterstützt. Solche Anwendungen arbeiten in der Regel mit sehr großen und heterogenen Informationspools: unternehmensinterne und oft auch externe Datenbestände aus verschiedensten IT-Systemen müssen verknüpft und ausgewertet werden. Die Informationen sollen einer großen Zahl von Benutzern zur Verfügung stehen. Da besonders zeitintensive Aufgaben über Message Queuing von den Dialogen entkoppelt werden können, sorgt *open*UTM dafür, daß die Antwortzeiten kurz bleiben. Falls es sich um reine Retrieval-Anwendungen handelt, bei denen Transaktionssicherheit und Wiederanlauf keine Rolle spielen, kann mit der Funktionsvariante *open*UTM-F (**F**ast) gearbeitet werden (siehe Abschnitt 10.5.3 auf Seite 137).

Weitere Informationen zum Thema „Message Queuing“ finden Sie in diesem Handbuch in Kapitel 4 auf Seite 69ff.

2.5 *open*UTM - der offene Transaktionsmonitor

Offenheit ist eine der Grundmaximen von *open*UTM. Sie manifestiert sich in vielen Eigenschaften und Funktionen. Die wichtigsten davon werden in den folgenden Abschnitten vorgestellt.

Plattformunabhängigkeit

Multivendor-Konfigurationen werden immer mehr zur Selbstverständlichkeit. *open*UTM steht zur Verfügung für alle gängigen UNIX-Plattformen, für Windows NT und für BS2000/OSD. Auch der Frontend-Baustein *open*UTM-Client ist auf allen diesen Plattformen verfügbar, zusätzlich auch noch auf Windows 9x und Windows 3.x.

Auf weiteren Client-Plattformen wie MAC OS und OS/2 können Client-Programme direkt per Transportschnittstelle mit UTM-Anwendungen kommunizieren.

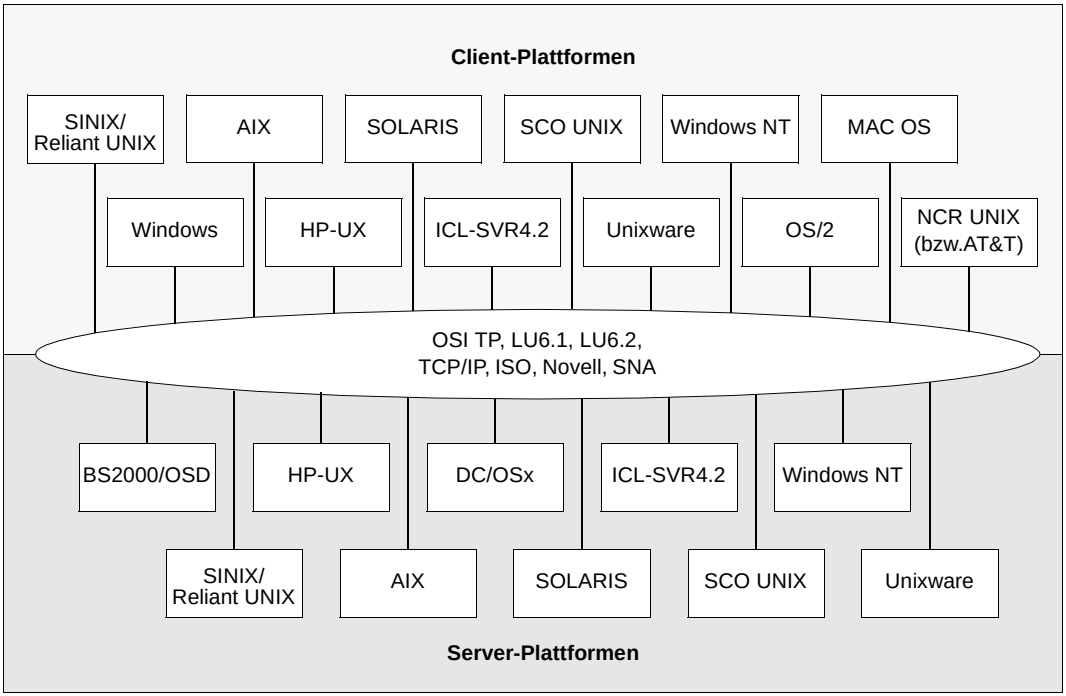


Bild 4: *open*UTM - verfügbar auf den verschiedensten Plattformen

Durch die hohe Portabilität von *open*UTM kann das Produkt für weitere UNIX-Plattformen innerhalb kürzester Zeit zur Verfügung gestellt werden.

Da *open*UTM auf Siemens-Mainframes mit dem Betriebssystem BS2000/OSD genauso ablauffähig ist wie unter UNIX- oder NT-Systemen, können bestehende Mainframe-Anwendungen problemlos mit neuen Anwendungen unter UNIX oder NT gekoppelt werden. So können die spezifischen Vorteile kleinerer Systeme, wie z.B. grafische Oberflächen und dezentrale, preiswerte CPU-Leistung, von bestehenden Mainframe-Anwendungen genutzt werden.

Integration in IBM-Mainframe-Umgebungen

*open*UTM unterstützt die in IBM-Umgebungen häufig verwendeten Kommunikationsprotokolle LU6.1 und LU6.2 (über *open*UTM-LU62). Daher können UTM-Anwendungen direkt und transaktionsgesichert mit CICS-IMS/TM- oder IMS/DC-Anwendungen zusammenarbeiten. Weitere Informationen siehe Abschnitte 2.19 auf Seite 41 und 3.4.5 auf Seite 62.

APPC/CPI-C Anwendungen in IBM-Umgebung können ebenfalls an UTM-Anwendungen angeschlossen werden. Ebenso können CICS-Anwendungen (ohne Transaktionssicherung) direkt über TCP/IP gekoppelt werden.

Für alle Downsizing, Rightsizing oder Surrounding-Projekte in IBM-Mainframe-Umgebungen ist *open*UTM damit optimal geeignet - sowohl in technischer Hinsicht als auch was die Einsparungsmöglichkeiten betrifft.

Kopplung mit anderen Transaktionsmonitoren über OSI TP und LU6.2

Über das standardisierte, offene Kommunikationsprotokoll OSI TP ist auch die unmittelbare Zusammenarbeit mit allen Transaktionsmonitoren möglich, die ebenfalls OSI TP unterstützen. Somit kann *open*UTM beispielsweise direkt und transaktionsgesichert mit Tuxedo-Anwendungen oder mit Anwendungen in UNISYS-Umgebungen kooperieren.

Bei Kopplung über *open*UTM-LU62 können auch Nicht-IBM-Systeme wie z.B. Encina erreicht werden.

Integration in das World Wide Web

Über die Produkte Web*Transactions* und *open*UTM-JetClient kann *open*UTM in das World Wide Web integriert werden. Die UTM-Anwendungen sind dann mit Web-Browsern wie z.B. Netscape oder InternetExplorer weltweit und von Millionen von Rechnern aus zu erreichen - über eine einheitliche und moderne grafische Oberfläche.

Weitere Einzelheiten finden Sie auf Seite 37 und Seite 53.

*open*UTM und Microsoft-Office

Wenn Sie die *open*UTM-Client-Software einsetzen, können UTM-Daten über die ActiveX-Technologie (COM/DCOM-Schnittstelle) auch direkt an Microsoft-Backoffice-Anwendungen, wie z.B. MS-Excel und MS-Word, übergeben werden. So ist es beispielsweise möglich, Daten aus Mainframe-Datenbanken direkt in Word-Serienbriefe einzusetzen, oder solche Daten mit den Mitteln von Excel optisch aufzubereiten. Ein *open*UTM-Client-Programm unter Windows kann als OLE-Server eingerichtet werden und läßt sich so mit den üblichen MS-OLE-Mitteln in jede beliebige Anwendung integrieren. Weitere Informationen siehe Abschnitt 2.17 auf Seite 39.

Wegen dieser Möglichkeiten und aufgrund seiner Ablauffähigkeit auf Windows NT hat *open*UTM von Microsoft das Backoffice-Logo erhalten.

Anschluß von Terminals

*open*UTM bietet Ihnen nicht nur Schnittstellen zum Anschluß von Client-Programmen, sondern unterstützt darüber hinaus den unmittelbaren Anschluß „dummer“ Terminals.

Diese Terminals können Sie im Zeilenmodus einsetzen, oder auch Bildschirm-Formate (Masken) verwenden. *open*UTM arbeitet mit Formatierungssystemen wie FORMANT (UNIX) oder FHS (BS2000/OSD) zusammen. Über Produkte wie WIN-DOORS oder FHS-DOORS lassen sich Bildschirm-Formate dynamisch in grafische Oberflächen umsetzen und so z.B. auch in Microsoft-Office-Umgebungen integrieren.

Für die direkte Kommunikation mit Terminal-Benutzern stellt *open*UTM spezielle Benutzerkommandos zur Verfügung.

Eine Beschreibung der Benutzerkommandos finden Sie im *open*UTM-Handbuch „Anwendungen generieren und betreiben“. Wie Sie die Terminalschnittstelle für Ihre Service-Routinen nutzen, erfahren Sie im *open*UTM-Handbuch „Anwendungen programmieren mit KDCS für COBOL, C und C++“. Tools für die Formatierung werden im vorliegenden Handbuch noch kurz vorgestellt, und zwar in Abschnitt 11.4 auf Seite 148 für BS2000/OSD und in Abschnitt 12.4 auf Seite 166 für UNIX.

Die Umsetzungstools WIN-DOORS und FHS-DOORS sind ausführlich in den Benutzerhandbüchern zu diesen Produkten beschrieben.

Anschluß von R/3-Anwendungen

R/3-Anwendungen können über ein Gateway an *open*UTM angeschlossen werden.
Die R/3-Anwendung verhält sich gegenüber der UTM-Anwendung wie ein Client und hat damit auch Zugriff auf die Datenbanken, die mit *open*UTM zusammenarbeiten. Weitere Informationen finden Sie in Abschnitt 2.18 auf Seite 40.

Zusammenarbeit mit beliebigen Applikationen

An *open*UTM können die verschiedensten Geräte, wie z.B. Waagen, Steuerungssysteme oder Industrieroboter als Clients angeschlossen werden, sofern diese über eine Schnittstelle auf Ebene des Transportsystems verfügen. Insbesondere können Anwendungen über die weit verbreitete **Sockets**-Schnittstelle an *open*UTM gekoppelt werden.

UTM-Services können somit von beliebigen Geräten angefordert werden, indem das Client-Gerät über die Transportschnittstelle den jeweiligen Service-Namen und gegebenenfalls auch Daten an *open*UTM übergibt (siehe auch Abschnitt 3.5 auf Seite 64).

Unterstützung verschiedenster Kommunikations- und Netzprotokolle

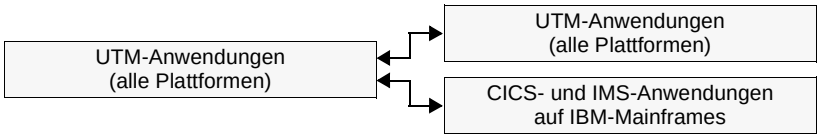
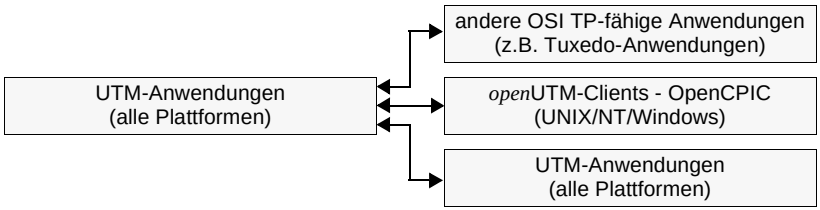
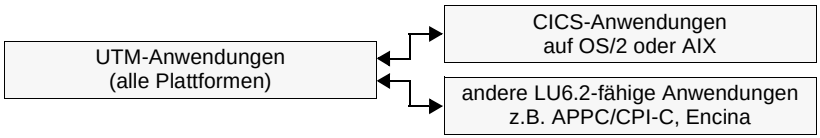
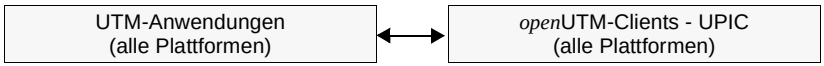
Grundlage für die Integration heterogener Welten bildet die Unterstützung unterschiedlichster Netz- und Kommunikationsprotokolle.

*open*UTM unterstützt alle gängigen Netzprotokolle, egal ob in LANs (Local Area Networks), z.B. TCP/IP, oder in WANs (Wide Area Networks), z.B. X.25. Weder Anwender noch Programmierer müssen sich um die unterschiedlichen Netztechniken kümmern: ihre UTM - Anwendungen sind in jeder Umgebung einsetzbar.

*open*UTM unterstützt aber nicht nur die verschiedensten Netzprotokolle, sondern auch unterschiedliche höhere Kommunikationsprotokolle. Diese Protokolle regeln Formen der Zusammenarbeit, die über den reinen Datenaustausch hinausgehen. So ist beispielsweise gewährleistet, daß Anwendungen unterschiedlicher Hersteller mit übergreifender Transaktionssicherung zusammenarbeiten können.

Die Tabelle auf der folgenden Seite gibt einen Überblick über die unterstützten höheren Kommunikationsprotokolle und die Kopplungsmöglichkeiten, die diese Protokolle eröffnen.

Übersicht:
Höhere Kommunikationsprotokolle und Kopplungsmöglichkeiten

LU6.1	(**L**ogical **U**nit **6.1**) Das LU6.1-Protokoll ist ein von IBM definiertes SNA-Protokoll. Es wurde laufend erweitert und hat sich zu einem Industriestandard entwickelt. Die Kommunikation erfolgt mit anwendungsübergreifender Transaktionssicherung. LU6.1 eignet sich besonders für folgende Kopplungen: 
OSI TP	(**O**pen **S**ystems **I**nterconnection **T**ransaction **P**rocessing) Von ISO definierter internationaler Standard für Distributed Transaction Processing. Ob mit anwendungsübergreifender Transaktionssicherung gearbeitet werden soll oder nicht, ist frei wählbar. OSI TP eignet sich besonders für folgende Kopplungen: 
LU6.2	(**L**ogical **U**nit **6.2**) Von IBM definiertes Protokoll, das von *open*UTM über *open*UTM-LU62 unterstützt wird. *open*UTM-LU62 ist aus Sicht von *open*UTM ein OSI TP-Partner. Damit kann mit und ohne anwendungsübergreifende Transaktionssicherung gearbeitet werden. *open*UTM-LU62 eignet sich für folgende Kopplungen: 
UPIC	(**U**niversal **P**rogramming **I**nterface for **C**ommunication) UTM-Schnittstelle und Protokoll für den Anschluß von Frontend-Clients: 

2.6 X/Open-Konformität von *open*UTM

Die Offenheit von *open*UTM spiegelt sich auch in der X/Open-Konformität wider: *open*UTM entspricht dem von X/Open definierten Referenzmodell für „Distributed Transaction Processing (DTP)“ und unterstützt die von X/Open standardisierten Schnittstellen.

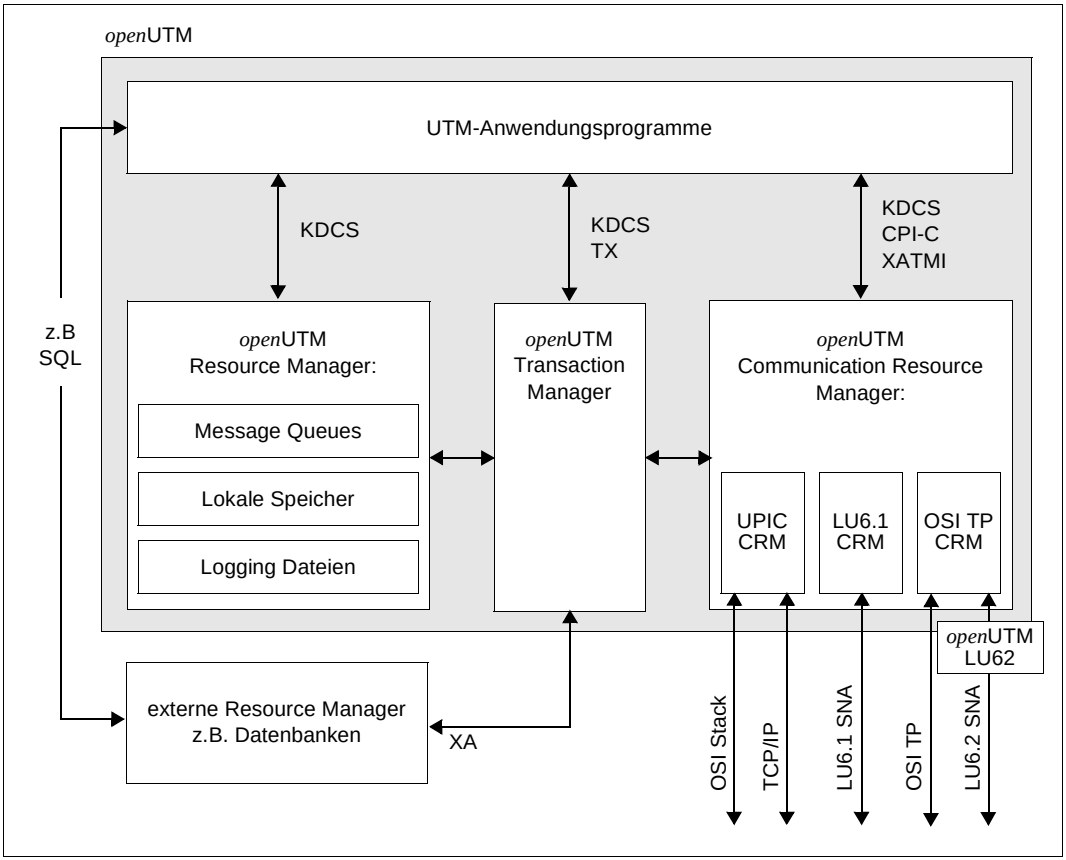


Bild 5: Architektur von *open*UTM entsprechend dem X/Open-Modell

X/Open unterscheidet bei den Komponenten transaktionsbasierter Systeme vier Typen: Anwendungsprogramme, Transaction Manager (TM), Resource Manager (RM) und Communication Resource Manager (CRM).

- Anwendungsprogramme:
  Anwendungsprogramme definieren die gewünschte Verarbeitung und nehmen über standardisierte Programmschnittstellen die Dienste der anderen Komponenten in Anspruch.

  Zusätzlich zu den von X/Open standardisierten Programmschnittstellen CPI-C, XATMI und TX unterstützt *open*UTM die universelle Schnittstelle KDCS (nationaler Standard).

- Transaction Manager (TM):
  Transaction Manager sind für die Steuerung und Überwachung von Transaktionen verantwortlich und stellen Recovery-Funktionen zur Verfügung. Transaction Manager koordinieren den Zugriff auf Daten- und Kommunikationsressourcen. Für externe Ressource-Manager, z.B. Datenbank-Systeme, erfolgt dies über die XA-Schnittstelle.

- Resource Manager (RM):
  Resource Manager verwalten Datenressourcen. Ein Beispiel für RMs sind Datenbank-Systeme. *open*UTM stellt aber auch selbst Resource Manager zur Verfügung, z.B. für den Zugriff auf Message Queues, lokale Speicherbereiche und Logging-Dateien. Anwendungsprogramme greifen auf RMs über RM-spezifische Schnittstellen zu. Für Datenbank-Systeme ist dies meist SQL, für die *open*UTM-RMs die Schnittstelle KDCS.

- Communication Resource Manager (CRM):
  Communication Resource Manager kontrollieren in verteilten Systemen die Kommunikation zwischen den Anwendungsprogrammen. *open*UTM stellt CRMs für den internationalen Standard OSI TP, für den Industrie-Standard LU6.1 und für das UTM-eigene Protokoll UPIC zur Verfügung (siehe auch Abschnitt 2.5 auf Seite 20 und die Abschnitte 3.3 auf Seite 49 bis 3.6 auf Seite 67). Der Anschluß von OSI TP ist in *open*UTM über das von X/Open standardisierte System Programming Interface XAP-TP realisiert. Der Anwendungsprogrammierer nutzt die Kommunikationsmöglichkeiten der CRMs über die Programmschnittstellen CPI-C, XATMI oder KDCS.

Im Anhang finden Sie eine Auflistung aller von *open*UTM unterstützen Normen und Standards.

Vorteile der X/Open-Konformität

Die X/Open-Konformität von *open*UTM eröffnet Ihnen eine Reihe von Vorteilen:

– Portabilität der Anwendungsprogramme durch standardisierte Programmschnittstellen (z.B. CPIC)

– Integration von heterogenen, verteilten Systemen durch standardisierte Kommunikationsprotokolle (z.B. OSI TP)

– Austauschbarkeit von Komponenten, beispielsweise von RMs, durch standardisierte Integrationsschnittstellen (z.B. XA)

Da *open*UTM zudem auf allen gängigen Hardwareplattformen eingesetzt werden kann und über ausgezeichnete Connectivity-Eigenschaften verfügt (siehe Abschnitt 2.5 auf Seite 20), können Sie Ihre Anwendung entsprechend den Arbeitsabläufen in Ihrem Unternehmen in einer heterogenen Umgebung verteilen. Bestehende Anwendungsteile - auch unter anderen Transaktionsmonitoren - können Sie problemlos integrieren und so Ihre Investitionen schützen.

Mit dem offenen und universellen Transaktionsmonitor *open*UTM haben Sie damit alle Möglichkeiten, in einer heterogenen IT-Welt die für Ihre Geschäftsabläufe passende Anwendungsarchitektur zu definieren und unter Verwendung der geeigneten Schnittstellen zu realisieren.

2.7 Performance, Durchsatz und Antwortzeiten

Höchste Performance ist eine der wesentlichen Stärken von *open*UTM. *open*UTM hat sich zum Ziel gesetzt, den höchsten Durchsatz und die kürzesten Antwortzeiten von allen derzeit verfügbaren Systemen zu bieten. Die hohe Effizienz wird erreicht durch ein ausgefeiltes Management der System-Ressourcen, z.B. durch Einsatz von Multithreading-Techniken und durch automatischen, dynamischen Lastausgleich. Multiprozessor-Hardware wird optimal genutzt.

Zeitintensive Aufgaben können durch Nutzung der Message Queuing Funktionalität von der Online-Verarbeitung entkoppelt werden.

Bei Online-Dialogen nutzt *open*UTM Wartezeiten, wie sie beispielsweise durch Denkpausen der Terminalbenutzer entstehen, optimal aus, indem es die Betriebsmittel anderen Aufträgen zuteilt. In solchen Fällen werden keine Prozesse blockiert. *open*UTM kann so die Last einer großen Zahl von simultanen Requests auf eine kleine Zahl von Prozessen verteilen.

Zur Performanceoptimierung bietet *open*UTM zudem ein ausgefeiltes Priority Scheduling-Konzept, das sowohl für Dialogverarbeitung als auch für Hintergrund-Aufträge (siehe Seite 74) genutzt werden kann.

Während bei herkömmlichen Lösungen der System-Overhead und damit die Antwortzeit mindestens proportional mit der Benutzerzahl zunimmt, bleibt die Antwortzeit von *open*UTM weit unter diesen Werten. Vergleichsmessungen belegen diese Tatsache schon bei relativ kleinen Benutzerzahlen.

*open*UTM ist aber ebenso geeignet für wirklich große Konfigurationen mit Tausenden von gleichzeitig arbeitenden Clients und Transaktionsraten von mehreren Millionen pro Tag.

Leistungskontrolle mit KDCMON

Zeichnen sich Leistungsengpässe ab, können Sie mit dem UTM-Meßmonitor **KDCMON** eine umfassende Analyse durchführen. KDCMON zeichnet vielfältige und detaillierte Informationen über den Ablauf von UTM-Anwendungen und Teilprogrammen auf. KDCMON liefert z.B. Informationen über Wartezeiten oder über den Ressourcen-Verbrauch einzelner Services.

Sie können KDCMON im laufenden Betrieb einschalten und nach der gewünschten Meßdauer wieder ausschalten. Dabei lassen sich Daten von einzelnen oder auch mehreren UTM-Anwendungen eines Rechners aufzeichnen.

Der Meßmonitor KDCMON ist ausführlich im *open*UTM-Handbuch „Anwendungen generieren und betreiben“ beschrieben.

2.8 Hochverfügbarkeit

Gerade dann, wenn Anwendungen in unternehmenskritischen Bereichen eingesetzt werden, können Ausfallzeiten nicht toleriert werden. *open*UTM gewährleistet, daß Ihre Anwendungen rund um die Uhr verfügbar sind (7∗24-Stunden-Betrieb):

- UTM-Anwendungen sind dynamisch konfigurierbar, d.h., Wartungsarbeiten wie Änderungen und Erweiterungen in der Konfiguration (etwa der Anschluß zusätzlicher Clients), sind im laufenden Betrieb möglich.
- Teile des Anwendungsprogramms oder sogar das gesamte Anwendungsprogramm lassen sich „on-the-fly“ austauschen, d.h., die Verfügbarkeit des Systems ist hierdurch zu keinem Zeitpunkt beeinträchtigt.
- Bei Fehlern in einem Serviceprogramm bleibt die Wirkung lokal. Es wird höchstens der betroffene Prozeß beendet (und danach automatisch ersetzt). Andere Services oder gar die gesamte Anwendung sind davon nicht betroffen. Es gibt also keinen „single point of failure“. Wenn gewünscht, wird im Fehlerfall ereignisgesteuert ein alternativer Service gestartet.
- Auch Hardware-Fehler in Peripheriegeräten, z.B. Ausfälle von Druckern oder Terminals, gefährden die Verfügbarkeit nicht, da *open*UTM Umschaltfunktionen bietet - per Administration oder auch automatisch.
- Selbst wenn ein Abbruch der Anwendung unvermeidlich ist, z.B. weil der Server-Rechner aufgrund eines Hardware-Fehlers ausfällt, ist die UTM-Anwendung sofort wieder einsatzbereit. Nach erneutem Start sichern die umfangreichen Wiederanlauf-Funktionen, daß die durch den Ausfall unterbrochenen Services und Dialoge unmittelbar und mit konsistenten Daten fortgesetzt werden können.
- Da das „Gedächtnis“ der UTM-Anwendung, die KDCFILE, doppelt geführt werden kann (hot standby), ist selbst nach physischer Zerstörung des Plattenspeichers die Einsatzbereitschaft der Anwendung nicht gefährdet.
- *open*UTM zusammen mit Hochverfügbarkeitshardware und -software wird auch extremen Anforderungen gerecht (siehe auch Seite 156 für BS2000/OSD, Seite 169 für UNIX und Seite 180 für Windows NT).

Weitere Informationen zum Thema Verfügbarkeit finden Sie in folgenden Handbüchern: *open*UTM-Handbuch „Anwendungen generieren und betreiben“ (doppelte KDCFILE-Führung, Programmaustausch) und *open*UTM-Handbuch „Anwendungen administrieren“ (dynamische Konfigurierung).

Auf die Fehlertoleranz und die Wiederanlauffunktionen von *open*UTM wird in diesem Handbuch in Kapitel 10 auf Seite 129ff näher eingegangen. Zum Themenkomplex „Hochverfügbarkeit“ gibt es auch ein eigenes Handbuch: „Hohe Verfügbarkeit von SNI UNIX-Systemen (Reliant UNIX V5.43)“ bzw. „Hochverfügbarkeit der RM-Systeme (Reliant UNIX V5.44)“.

2.9 Security-Funktionen

*open*UTM bietet umfassende, differenzierbare und klar strukturierte Konzepte zur Zugangs- und Zugriffskontrolle (Authentisierung und Autorisierung):

- Definition logischer Anschlußpunkte:
  Ein Client kann sich nur dann an eine UTM-Anwendung anschließen, wenn er einem logischen Anschlußpunkt (LTERM-Partner) dieser Anwendung zugeordnet ist. Der Client muß also der UTM-Anwendung bekannt sein.
- Definition von Benutzerkennungen:
  Für eine UTM-Anwendung können Benutzerkennungen definiert werden. Dies kann bei der Generierung oder auch dynamisch bei laufender Anwendung geschehen.
- Zuordnung von Paßwörtern zu Benutzerkennungen
  Den Benutzerkennungen lassen sich benutzer-spezifische Paßwörter zuordnen.
- Einsatz eines Ausweislesers zur Zugangskontrolle
- Stiller Alarm bei wiederholten Fehlversuchen
- Automatischer Verbindungsabbau bei Zeitüberschreitung
- Ereignisgesteuerte Routinen für selbstdefinierte Zugangskontrollen
- Lock-/Keycode-Konzept für differenzierte Zugriffsberechtigungen (Autorisierung):
  Mit dem Lock-/Keycode-Konzept können individuelle Berechtigungsprofile definiert werden. Services und Anschlußpunkte lassen sich durch Lockcodes vor unberechtigtem Zugriff schützen. Die Berechtigungen (Keycodes) können wahlweise an eine bestimmte Benutzerkennung, an einen bestimmten Anschlußpunkt oder an beides gleichzeitig gebunden werden. So kann beispielsweise die Berechtigung eines Benutzers, einen bestimmten Service aufzurufen, auf die Verwendung eines speziellen, besonders gesicherten Client-Rechners beschränkt werden.
- Verschlüsselung von Paßwörtern und Benutzerdaten für den Zugang von bestimmten Clients aus oder für den Zugriff auf bestimmte Services.
- Anwendungsübergreifendes Benutzerkonzept bei Nutzung von OSI TP

Detaillierte Informationen zum Thema Security finden Sie im Kapitel 9 auf Seite 117.

Die umfangreichen Zugangs- und Zugriffsschutzkonzepte, die *open*UTM bietet, stehen auch beim Anschluß von *open*UTM-Client-Programmen zur Verfügung sowie bei verteilter Verarbeitung mit mehreren UTM-Anwendungen.

Bei der Zusammenarbeit mit Datenbanken können selbstverständlich zusätzlich die speziellen Schutzmechanismen der Datenbank-Systeme genutzt werden. Diese Mechanismen sind in der Dokumentation der jeweiligen Datenbank-Systeme beschrieben.

Weitere Informationen zum Thema Security-Funktionen finden Sie unter den Stichwörtern LTERM, KEYSET, KEYS und LOCK in den *open*UTM-Handbüchern „Anwendungen generieren und betreiben“ und „Anwendungen administrieren“.

2.10 Internationalisierung / Anpassung von Meldungen

*open*UTM stellt umfassende Funktionen zur Internationalisierung zur Verfügung. Diese ermöglichen es, sprachübergreifende UTM-Anwendungen zu erstellen, die jedem Anwender in der von ihm gewohnten Landessprache zur Verfügung stehen. Datumsangaben, Uhrzeit, Maßangaben oder Währungssymbole können jeweils den ortsüblichen Konventionen entsprechend dargestellt werden.

Für die UTM-Systemmeldungen - standardmäßig in englischer und deutscher Sprache verfügbar - gibt es vielfältige und komfortable Anpassungsmöglichkeiten:
Meldungen in anderen Sprachen sind einfach integrierbar, die Standardtexte können ersetzt oder abgewandelt werden und auch andere Meldungseigenschaften, wie z.B. die jeweiligen Ausgabeziele, lassen sich neu festlegen.

Eine UTM-Anwendung kann mit mehreren Meldungsdateien arbeiten, so daß jeder Anwender mit individuell abgestimmten Meldungen versorgt werden kann. Dies eröffnet für das Anwendungsdesign einen großen Gestaltungsspielraum.

*open*UTM (UNIX) erfüllt die Richtlinien zur Internationalisierung, wie sie in den X/Open Portability Guides spezifiziert sind.

Detaillierte Informationen über den Aufbau von Meldungen (z.B. über die verschiedenen Meldungsziele) und über die Anpassungsmöglichkeiten erhalten Sie im *open*UTM-Handbuch „Meldungen, Test und Diagnose“.

2.11 Accounting

*open*UTM stellt Funktionen zur Verfügung, die es den Rechenzentren ermöglichen, den Benutzern einer UTM-Anwendung die in Anspruch genommene Rechnerleistung zu verrechnen. Unter Benutzer versteht man die UTM-Benutzerkennung, unter der sich ein Benutzer anmeldet. Bei Verteilter Verarbeitung tritt anstelle der UTM-Benutzerkennung die Session (LU6.1) bzw. die Association (OSI TP), so daß eine Verrechnung auch unter diesen Umständen möglich ist.

Dabei kann entweder per Festpreis oder verbrauchsabhängig abgerechnet werden.

Festpreis

Bei der Festpreisabrechnung verbucht *open*UTM für jeden Aufruf eines bestimmten Services eine feste Anzahl Verrechnungseinheiten auf das Benutzerkonto. Um den Festpreis eines Service zu ermitteln, kann *open*UTM den Betriebsmittelverbrauch von Services wie z.B. mittlerer CPU-Verbrauch aufzeichnen. Mit Hilfe dieser Aufzeichnungen wird dann per Generierung festgelegt, wieviele Einheiten für die einzelnen Services verrechnet werden sollen. Dabei können einzelne Services wie z.B. Auskünfte durchaus kostenlos sein.

Verbrauchsabrechung

Bei Verbrauchsabrechnung ermittelt *open*UTM den aktuellen Betriebsmittelverbrauch eines Benutzers (z.B. nach CPU-Sekunden) und verbucht die Einheiten in bestimmten Abständen auf dessen Konto. Je nach System können mehrere Betriebsmittel wie z.B. CPU, Druckausgaben, Ein -Ausgaben unterschieden und verschieden gewichtet werden.

Die Abrechnung kann sowohl per Generierung als auch im laufenden Betrieb ein- oder ausgeschaltet werden.

Das Accounting mit *open*UTM ist ausführlich im *open*UTM-Handbuch „Anwendungen generieren und betreiben“ beschrieben.

2.12 Diagnosemöglichkeiten in *open*UTM

*open*UTM bietet Ihnen bei der Diagnose und der Lokalisierung von Fehlern in der Anwendung umfassende Unterstützung.

Fehler bei Funktionsaufrufen werden von *open*UTM über entsprechende **Returncodes** und Meldungen signalisiert, wobei Sie zur weiteren Analyse der Fehlersituation einen **Speicherabzug (Dump)** anfordern können - entweder per Administrationskommando oder durch einen Aufruf im Programm. Bei Fehlern, die zum Abbruch eines Anwendungsprozesses oder der UTM-Anwendung führen, wird automatisch ein prozeßspezifischer Dump erstellt.

In der Dump-Datei werden auch alle Datenbankaufrufe protokolliert, so daß Fehler bei der Zusammenarbeit mit Datenbanken leicht lokalisiert werden können.

Für die Auswertung der Dump-Datei stellt Ihnen *open*UTM ein spezielles Tool (KDCDUMP) zur Verfügung, das die Dump-Datei in gut lesbarer Form aufbereitet. Dieses Tool bietet Ihnen außerdem die Möglichkeit, die Dump-Datei gezielt am Terminal zu bearbeiten und zum Beispiel nach bestimmten Tabelleneinträgen zu suchen und diese aufbereiten zu lassen. Natürlich können Sie auch die gesamte Datei aufbereiten lassen und sie anschließend ausdrucken oder editieren.

Als weitere Diagnosehilfe bietet Ihnen *open*UTM verschiedene **Trace-Funktionen** zur Ablaufverfolgung. Trace-Funktionen protokollieren beispielsweise alle verbindungsbezogenen Aktivitäten innerhalb einer UTM-Anwendung oder alle Aktivitäten über OSI-TP-Verbindungen in einer Trace-Datei. Auch zur Auswertung der Trace-Dateien stehen Ihnen UTM-Tools zur Verfügung. Die Trace-Funktionen können Sie beim Anwendungsstart, per Administrationskommando oder programmgesteuert über die Schnittstelle zur dynamischen Administration einschalten.

Wichtige Informationen für die Diagnose liefert auch die anwendungsspezifische **Protokolldatei SYSLOG** (SYStem LOGging), die *open*UTM beim Start einer Anwendung anlegt. In der SYSLOG-Datei protokolliert *open*UTM Meldungen, die für die laufende Überwachung oder für spätere Kontrollen nützlich sein können. Die Auswahl der zu protokollierenden Meldungen können Sie selbst beeinflussen, indem Sie bestimmten Meldungen das Meldungsziel SYSLOG zuordnen. Zur Aufbereitung und Auswertung der SYSLOG-Datei stehen Ihnen UTM-Tools zur Verfügung.

Eine detaillierte Beschreibung der Diagnosemöglichkeiten finden Sie im *open*UTM-Handbuch „Meldungen, Test und Diagnose“. Dort finden Sie auch Informationen zum Dump-Auswertungstool KDCDUMP. Die Tools zur Auswertung der Systemprotokoll-Datei SYSLOG sind im *open*UTM-Handbuch „Anwendungen generieren und betreiben“ beschrieben.

2.13 Einfache, komfortable Anwendungsentwicklung

Für die Programmierung der Service-Routinen können Sie die gewohnte Programmiersprache verwenden: Die UTM-Aufrufe lassen sich in C-, C++- oder COBOL-Programme integrieren. In BS2000/OSD steht die KDCS-Schnittstelle zusätzlich für Assembler, Fortran, PASCAL-XT und PL/I zur Verfügung. Auch wenn die Service-Routinen in verschiedenen Sprachen codiert sind, können sie beliebig kombiniert werden.

Bei der Programmierung sind Sie auch nicht an ein bestimmtes Kommunikationsmodell gebunden, da *open*UTM die unterschiedlichsten Modelle unterstützt (Conversations, Pseudo-Conversations, Request/Reply, Message Queuing).

Die Programmierung wird wesentlich auch dadurch vereinfacht, daß viele zentrale Aufgaben von *open*UTM selbsttätig erledigt werden.

*open*UTM übernimmt z.B.:

– die Steuerung globaler Transaktionen
– das Management von parallelen Zugriffen
– den Aufbau von Netzverbindungen
– Vorkehrungen für den Fehlerfall

Deshalb können Sie so programmieren, als würden Sie Ihre Anwendung nur für einen einzigen Anwender und für ein abgeschlossenes, homogenes System erstellen.

Effiziente Entwicklungsumgebungen

Für die Erstellung einer UTM-Anwendung können Sie alle vom Betriebssystem und von den Compilern zur Verfügung gestellten Hilfsmittel voll nutzen.

Darüber hinaus läßt sich *open*UTM in Programmiersysteme, Sprachen der 4. Generation und Design-Werkzeuge integrieren. Dies sind z.B.:

– DRIVE/Windows zusammen mit case/4/0 und DRIVE-Designer
– GINA für objektorientierte Anwendungserstellung (siehe auch Abschnitt 2.14 auf Seite 35)
– NATURAL der Software AG
– COOL:UTMGen von Sterling Software
– Score von Delta Software

2.14 Objektorientiertes Framework für *open*UTM auf Basis von GINA

Das Produkt GINA (General Interface for Network Applications) bietet ein Framework für Implementierung und Betrieb von objektorientierten, transaktionsgesicherten Client/Server-Applikationen. GINA ermöglicht Ihnen die Einführung von objekt-orientierter Technologie nicht nur für einzelne Workstations oder auf Abteilungsebene, sondern auch im business-critical Bereich unternehmensweiter IT-Lösungen.

Der Umstieg auf ein objektorientiertes Systemkonzept wird mit GINA erleichtert, da während der Umbau-Phase nicht auf die gewohnte Funktionalität bezüglich Transaktionsverarbeitung und Datenkonsistenz verzichtet werden muß.

GINA stellt objektorientierte Schnittstellen in der Programmiersprache C++ zur Verfügung und bietet folgende Leistungen:

– transaktionsgesicherte Objekt-Kommunikation in verteilten Systemen
– transaktionsgesicherte Speicherung von Objekten
– vollständige Einbettung in C++ (Vererbung, Polymorphie, u.a.)
– Support-Funktionen für den Einsatz in verteilten Umgebungen
– Wiederanlauf von verteilten Transaktionen
– Connectivity zu Fremd-Systemen

GINA besteht aus folgenden Komponenten:

– T-ORB
  transaktionsgesicherter Object Request Broker
– Persistency Service
  am Object Database Standard ODMG93 orientiert
– Support
  zur Integration verteilter Anwendungen
– CORBA-IDL-Compiler,
  der den Stub des Clients und die Schnittstelle des Servers erzeugt
– C++-Entwicklungsumgebung mit eigenen Tools
  die ausgehend von C++-Deklarationen das Objektmodell auf Relationen eines RDBMS abbilden und Stubs und Skeletons für plattformübergreifende Kommunikation erzeugen

Nähere Informationen finden Sie in den GINA-Handbüchern.

2.15 Grafische Administration mit WinAdmin

Mit der *open*UTM-Komponente WinAdmin können Sie vom PC aus über eine komfortable grafische Oberfläche administrieren und generieren:

- Administrieren
  Dabei können einzelne oder auch mehrere UTM-Anwendungen administrieren und dynamisch konfigurieren, d.h. Objekte neu aufnehmen oder löschen, denn WinAdmin nutzt den vollen Umfang der Administrations-Programmschnittstelle KDCADMI. Für Anwendungen mit älteren UTM-Versionen bietet WinAdmin auch die klassische Kommandoschnittstelle. Näheres finden Sie auf Seite 112.
- Generieren
  Sie können für eine ferne UTM-Anwendung KDCDEF-Eingabedateien erzeugen und daraus per Mausklick eine neue KDCFILE erzeugen oder die KDCFILE mit KDCUPD aktualisieren. Näheres finden Sie auf Seite 103.

Die UTM-Anwendungen können auch im Netz verteilt sein und auf unterschiedlichen Plattformen ablaufen. Die zu administrierenden UTM-Anwendungen lassen sich zu Kollektionen gruppieren und können so gemeinsam administriert werden. Beispielsweise ist es möglich, in **einem** Schritt Objekte **mehrerer** Anwendungen zu modifizieren

*open*UTM WinAdmin kommuniziert mit den UTM-Anwendungen über UPIC und ist eine echte 32-Bit-Anwendung, die unter Windows 9x und Windows NT läuft.

Security bei WinAdmin

Für die Administration über WinAdmin stehen Ihnen selbstverständlich sämtliche UTM-Security-Funktionen zur Verfügung, angefangen vom Zugangsschutz durch UTM-Benutzerkennung und -Paßwort bis zur Verschlüsselung von Paßwort und Daten.

Da jedoch an Administratoren besonders hohe Sicherheitsanforderungen gestellt werden, bietet WinAdmin zusätzlich ein eigenes Benutzerkonzept:

- Sie können mehrere Benutzer definieren und mit unterschiedlichen Rechten ausstatten, angefangen vom Benutzer mit reinem Leserecht bis zum „Master“, dem Administrator von WinAdmin.
- Für jeden Benutzer wird der Zugang zu WinAdmin durch ein Paßwort geschützt.

Über weitere Leistungsmerkmale von *open*UTM WinAdmin und über die Anwendung dieses Produkts informieren die Online-Hilfe sowie die Readme-Datei, die mit „*open*UTM WinAdmin“ ausgeliefert werden.

2.16 Web-Anbindung von *open*UTM

Mit den Produkten Web*Transactions* und *open*UTM-JetClient können Sie *open*UTM an das World Wide Web anbinden. Web*Transactions* stellt die Ein- und Ausgaben von *open*UTM als HTML-Dokumente bereit, während *open*UTM-JetClient die Möglichkeit bietet, eigene Clients auf Java-Basis zu erstellen und auf beliebigen Plattformen ablaufen zu lassen.

Web*Transactions*

Mit Web*Transactions* können Sie die Ein- und Ausgaben von UTM im Web als HTML-Dokumente bereitstellen und die Dialogabläufe anpassen bis hin zur Integration von UTM- und OSD- oder MVS-Anwendungen.

Die differenzierten Zugangs- und Zugriffsschutzmechanismen sowie die Wiederanlauf-Funktionen von *open*UTM stehen auch bei Nutzung über das Web zur Verfügung: *open*UTM und Web*Transactions* bieten mainframe-like Security im Web.

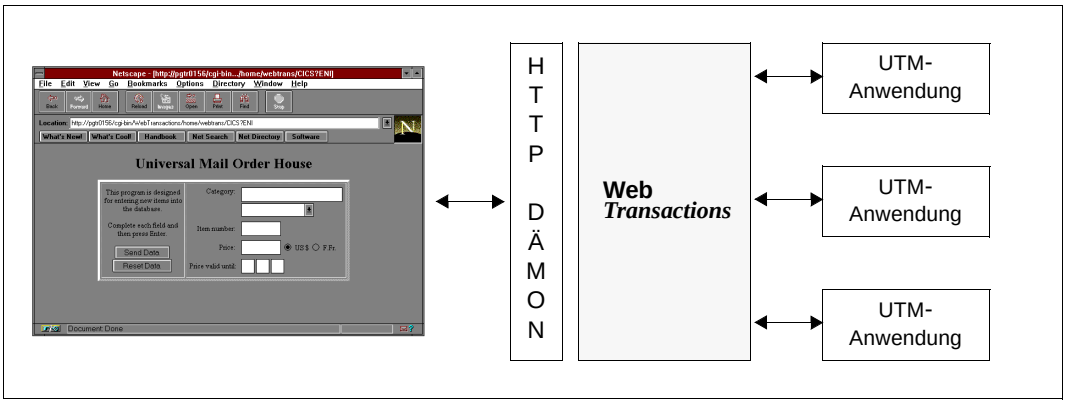


Bild 6: Web-Anbindung einer UTM-Anwendung über Web*Transactions*

Der Aufwand für die Anbindung von UTM-Anwendungen ist gering: die Anwendungslogik muß nicht angepaßt werden. Zudem bietet Web*Transactions* Tools für die Umwandlung bestehender Maskenformate in browserfähige HTML-Dokumente, siehe auch Abschnitt 3.3.2 auf Seite 53.

Mit Web*Transactions* lassen sich nicht nur UTM-Anwendungen ans Web anschließen, sondern auch beliebige andere Dialoganwendungen, z.B. OLTP-Anwendungen, die mit Transaktionsmonitoren anderer Hersteller (CICS, Tuxedo, ...) arbeiten, Datenbankanwendungen oder BTX-Anwendungen.

Nähere Informationen finden Sie im Web unter: *http://bs2www.mch.sni.de*. Die Dokumentation zu Web*Transactions* ist auch auf der *open* UTM-Doku-CD enthalten.

*open*UTM-JetClient

Mit *open*UTM-JetClient (*open*UTM Java Enterprise Technology Client) können Sie in Java geschriebene Clients an *open*UTM-Anwendungen anschließen. Damit können Sie von beliebigen Plattformen aus auf UTM-Anwendungen zugreifen, insbesondere auch von Network-Computern aus.

Die differenzierten Zugangs- und Zugriffsschutzmechanismen sowie die Wiederanlauf-Funktionen von *open*UTM stehen auch den Java-Clients zur Verfügung.

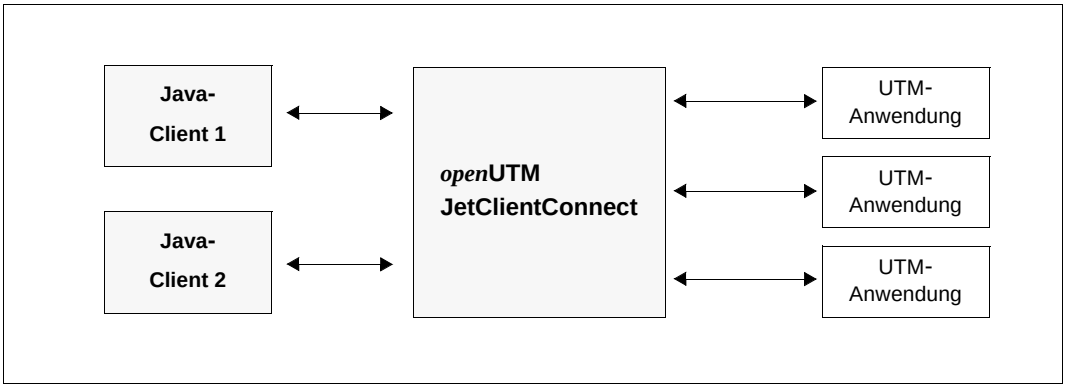


Bild 7: Anbindung von Java-Clients an *open*UTM

Die Java-Clients können sowohl Java-Anwendungen als auch Java-Applets sein. Die UTM-Anwendungen sehen nur den Verbindungsbaustein JetClientConnect, die Java-Clients selber werden nicht in *open*UTM konfiguriert.

Nähere Informationen zu Java-Clients für *open*UTM finden Sie im Handbuch *"open*UTM-Jet-Client (Reliant UNIX)“.

2.17 *open*UTM in der Microsoft -Welt

In einer Microsoft-Umgebung können Sie sowohl mit Hilfe von Programmiersystemen (Visual Basic, Visual C++) als auch über Standard-Anwendungen (Word, Excel, ...) auf UTM-Anwendungen zugreifen.

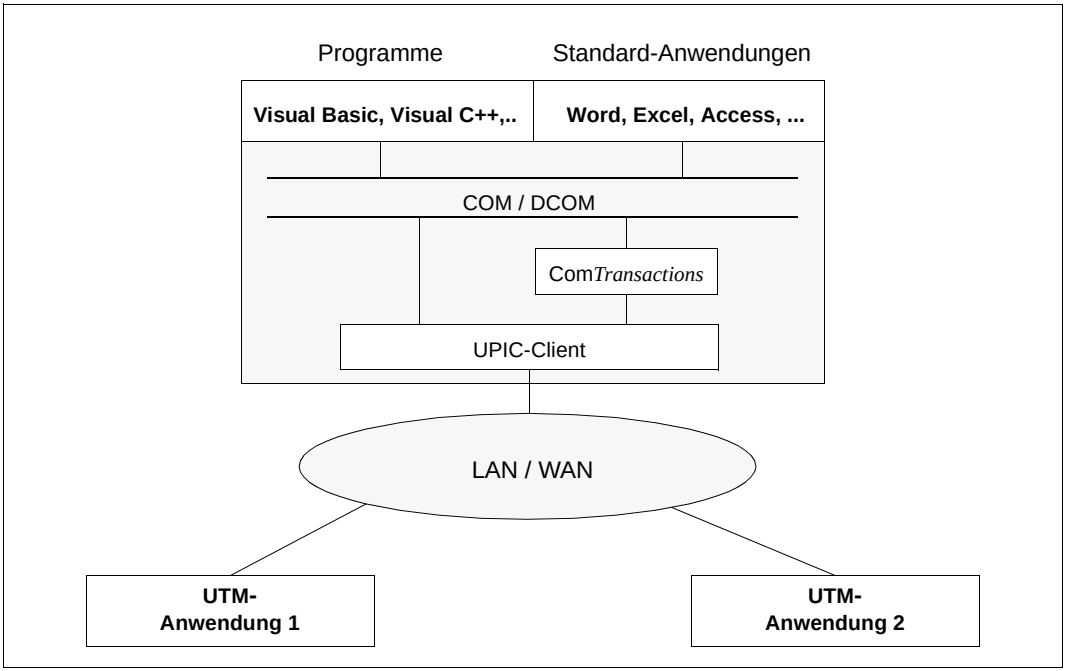


Bild 8: Zugriff auf *open*UTM aus der Microsoft-Umgebung

Als Basis dient der *open*UTM-Client mit dem Trägersystem UPIC. Für den Zugriff können Sie entweder direkt die ActiveX-Schnittstelle des *open*UTM-Client oder das Aufsatzprodukt Com*Transactions* einsetzen. Com*Transactions* bietet zusätzliche Administrations- und Entwicklungstools an, mit deren Hilfe auch Multi-Tier-Anwendungen erstellt werden können. Beide verwenden den Microsoft-Standard COM/DCOM (Component Object Model/Distributed Component Object Model), näheres siehe Abschnitt 3.3.1 auf Seite 51.

Die UTM-Anwendungen können auf jedem über das Netz erreichbaren Rechner liegen.

2.18 R/3-Anbindung von *open*UTM

Sie können *open*UTM-Anwendungen über das SAP-UTM-Gateway mit SAP-R/3-Anwendungen koppeln. Das SAP-UTM-Gateway setzt auf dem *open*UTM-Client auf.

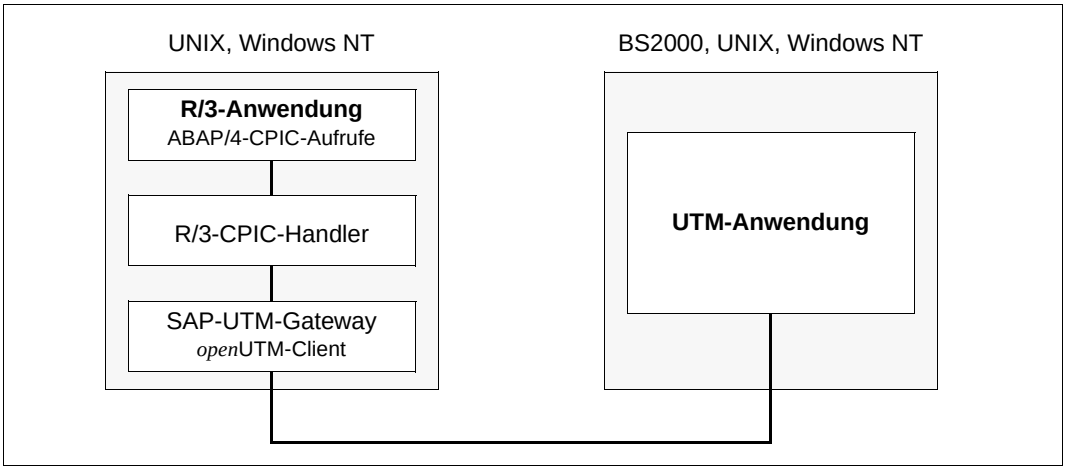


Bild 9: R/3-Anbindung von *open*UTM

Auf der UNIX-Plattform kann das SAP-UTM-Gateway sowohl über das Trägersystem UPIC als auch über das Trägersystem OpenCPIC mit der UTM-Anwendung kommunizieren. Auf Windows NT kann derzeit nur das Trägersystem UPIC verwendet werden.

- Anbindung über UPIC
  Hier wird die Kommunikation immer durch die R/3-Anwendung angestoßen, da UPIC reiner Client ist. Es können sich mehrere R/3-Benutzer gleichzeitig an die UTM-Anwendung anmelden, der UPIC-Client (SAP-Gateway) wird dann mehrfach gestartet. Auf UTM-Seite müssen dann entsprechend viele Clients definiert werden.
- Anbindung über OpenCPIC bei UNIX
  Diese Anbindung ist eine Kommunikation über OSI TP. Daher kann die Kommunikation sowohl durch R/3 als auch durch *open*UTM angestoßen werden. Auch sind echte Parallel-Verbindungen möglich.

Für die Kommunikation über das UTM-SAP-Gateway wird ein einfaches Protokoll verwendet, das als Vorspann vor der eigentlichen Nachricht steht. Dies wirkt sich auf den Auftraggeber aus. Dieser muß beim Senden einen Vorspann definieren und beim Empfangen den Vorspann auswerten.

Der Auftragnehmer hingegen braucht sich nicht um den Vorspann zu kümmern. Wenn also R/3 über UPIC an *open*UTM gekoppelt wird oder wenn *open*UTM Auftragnehmer ist (bei OpenCPIC-Anbindung), dann müssen die UTM-Teilprogramme diesbezüglich nicht geändert werden.

2.19 CICS-IMS-Anbindung von *open*UTM

Bei IBM-Systemen ist verteilte Transaktionsverarbeitung üblicherweise mit den Protokollen LU6.1 oder LU6.2 realisiert. Ältere Transaktionsmonitore sind häufig nur über LU6.1, neuere wie z.B. CICS/6000 meist nur über LU6.2 erreichbar.

*open*UTM kann mit Partneranwendungen über LU6.1 und LU6.2 kommunizieren. Für die Kommunikation über LU6.2 ist zusätzlich *open*UTM-LU62 notwendig. *open*UTM-LU62 ist aus UTM-Sicht ein OSI-TP-Partner und aus Sicht des IBM-Hosts ein LU6.2-Partner.

Für eine Kopplung zwischen *open*UTM und CICS / IMS gibt es drei Musterkonfigurationen.

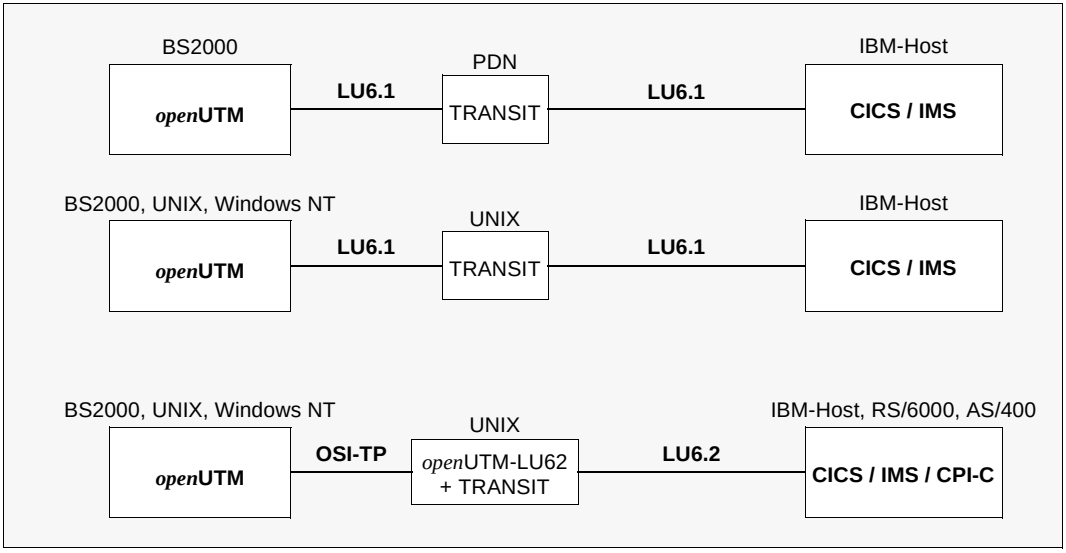


Bild 10: Anbindung von *open*UTM an IBM-Transaktionsmonitore

Im IBM-Host ist keine Zusatzsoftware notwendig, die SNA-Anbindung wird plattformspezifisch durch das Transportsystem-Gateway TRANSIT erbracht. Falls *open*UTM auf einem UNIX-Rechner läuft, können TRANSIT und *open*UTM-LU62 auch auf diesem UNIX-Rechner installiert und dieser direkt an das SNA-Netz angeschlossen werden.

Bei beiden Kopplungsarten mit den IBM-Transaktionsmonitoren gewährleistet *open*UTM volle, systemübergreifende Transaktionssicherheit und globale Wiederanlauffunktionen bis hinunter auf die Ebene von einzelnen Service-Routinen oder Ausgabe-Aufträgen.

CICS-Anwendungen können auch als Transportsystem-Anwendungen direkt über TCP/IP gekoppelt werden, allerdings ohne verteilte Transaktionssicherung.

2.20 *open*UTM-MQSeries-Gateway

Das *open*UTM-MQSeries-Gateway (UTM-MQS-GW) ermöglicht die Kopplung von UTM-Anwendungen unter BS2000/OSD oder UNIX mit MQSeries-Anwendungen auf anderen Plattformen. Unterstützt wird dabei sowohl asynchroner Nachrichtentransfer in beiden Richtungen als auch synchroner Dialog der Partneranwendungen. Durch Realisierung des Gateways als eigene *open*UTM-Anwendung kann eine durchgehende Transaktionssicherheit gewährleistet werden.

UTM-MQS-GW kommuniziert auf der einen Seite über UTM-Programmschnittstellen mit einer UTM-Anwendung und auf der anderen Seite über MQSeries-Programmschnittellen mit einem MQSeries Queuemanager, der auf dem Gateway-Rechner installiert sein muß. Der MQSeries Queuemanager leitet die Nachrichten an eine MQSeries-Anwendung weiter und empfängt sie von dort.

Die Partneranwendungen können, müssen aber nicht auf demselben Rechner wie das Gateway laufen. Falls die MQSeries-Partneranwendung auf demselben Rechner läuft, kann sie denselben Queue-Manager verwenden wie das Gateway.

Das Gateway ist auf allen Plattformen einsetzbar, auf denen sowohl MQSeries-Server als auch *open*UTM-Server freigegeben sind (z.B. Windows NT 4.0, UNIX-Plattformen).

2.21 SNMP-Subagent für *open*UTM

Das Managementprotokoll SNMP (Simple Network Management Protocol) gehört zur Protokollfamilie TCP/IP und ist heute außer für das Netzmanagement auch der De-facto-Standard für das System- und Anwendungsmanagement.

Der SNMP-Subagent für *open*UTM bindet den Transaktionsmonitor *open*UTM in einen systemübergreifenden Leitstand ein und

- ermöglicht für ausgewählte UTM-Anwendungen einen umfassenden Überblick über alle mit ihr in Zusammenhang stehenden Objekte wie Systemparameter, physikalische und logische Terminals, TACs, User usw.,
- integriert UTM-Anwendungen in die grafische Netzkarte eines SNMP-Managers und ermöglicht die farbliche Anzeige der Zustände,
- bietet Administationsfunktionen wie das Ändern von Eigenschaften einer Anwendung, Sperren und Entsperren von Clients, Starten und das Beenden einer Anwendung.

Der Subagent kommuniziert mit der überwachten UTM-Anwendung über die UPIC-Schnittstelle und kann zu einer Zeit immer nur mit einer Anwendung verbunden sein.

Der SNMP-Master-Agent mit seinen SNMP-Subagenten kann über SNMP im Prinzip an alle Management-Zentralen angeschlossen werden. In erster Priorität wird für die Management-Plattform das SNI-Produkt TransView Control Center mit TransView SNMP empfohlen. Diese Produkte bieten alle Möglichkeiten für die Integration von beliebigen Systemen mit privaten MIBs.

3 Client/Server-Computing mit *open*UTM

Das Thema Client/Server-Computing wurde bereits in Abschnitt 2.3 auf Seite 16 kurz angesprochen. In diesem Kapitel wird auf die unterschiedlichen Formen des Client/Server-Computing näher eingegangen.

3.1 Client/Server-Architekturvarianten

Allen Client/Server-Architekturen liegt eine Gliederung in einzelne Software-Komponenten, auch „Schichten“ oder „Tiers“ genannt, zugrunde (der Begriff „Tier“, übersetzt: „Stufe“, kommt aus dem Englischen, setzt sich aber auch im deutschen Sprachraum immer mehr durch). Man spricht von 1-Tier-, 2-Tier-, 3-Tier und auch von Multi-Tier-Modellen. Dabei wird oft nicht klar unterschieden, ob man die Aufgliederung auf der physischen oder auf der logischen Ebene betrachtet:

– Logische Software-Tiers liegen vor, wenn die Software in modulare Komponenten mit klaren Schnittstellen gegliedert ist - unabhängig davon, wie diese Komponenten im Netz verteilt sind.

– Physische Software-Tiers liegen dann vor, wenn die (logischen) Softwarekomponenten im Netz auf verschiedene Rechner verteilt sind.

In Bild 11 auf der folgenden Seite sind die fünf Client/Server-Basismodelle mit jeweils zwei physischen Software-Tiers dargestellt.

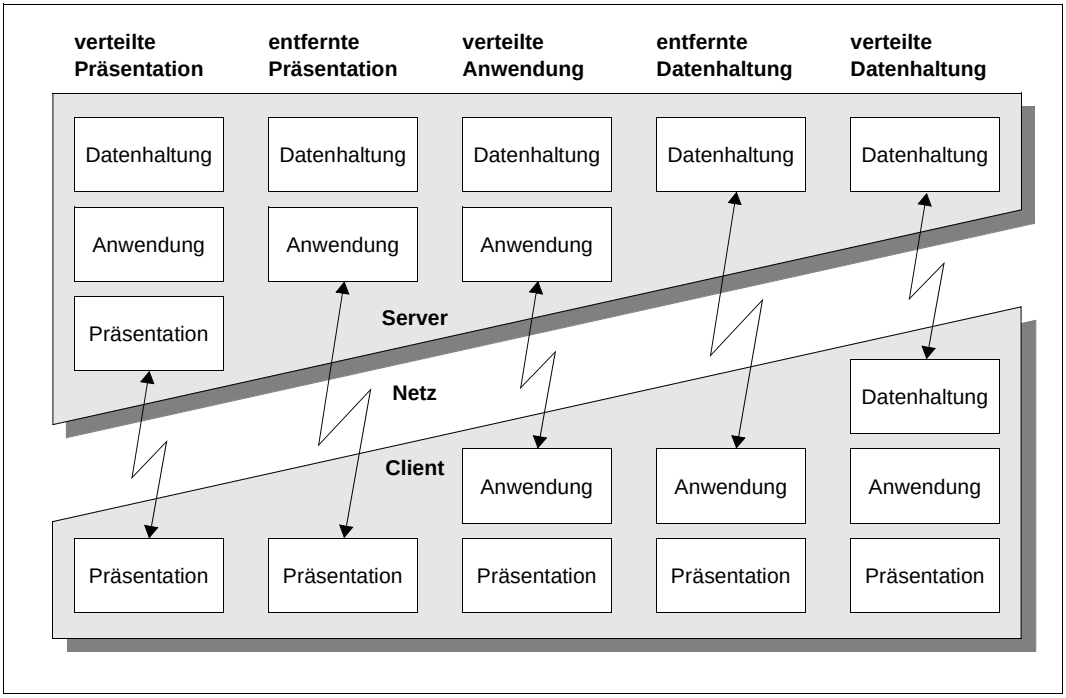


Bild 11: Basismodelle für Client/Server-Architekturen

*open*UTM unterstützt nicht nur die in Bild 11 dargestellten Grundschemata, sondern vielfältige Abwandlungen und Kombinationen. Mit *open*UTM sind komplexe Multi-Tier-Szenarien möglich - sowohl auf der Ebene physischer als auch auf der Ebene logischer Tiers.

Wie Bild 12 (auf der folgenden Seite) zeigt, erlaubt *open*UTM nicht nur die Verteilung von Daten oder Anwendungslogik, sondern auch beliebige Kombinationen. Dadurch können Daten an den Stellen abgelegt werden, an denen sie auch verarbeitet werden. Die Netzbelastung wird minimiert, da nicht mehr umfangreiche Datenmengen, sondern nur noch Aufträge und Ergebnisse über das Netz gehen. *open*UTM gewährleistet dabei, daß die Daten zu jedem Zeitpunkt global konsistent sind.

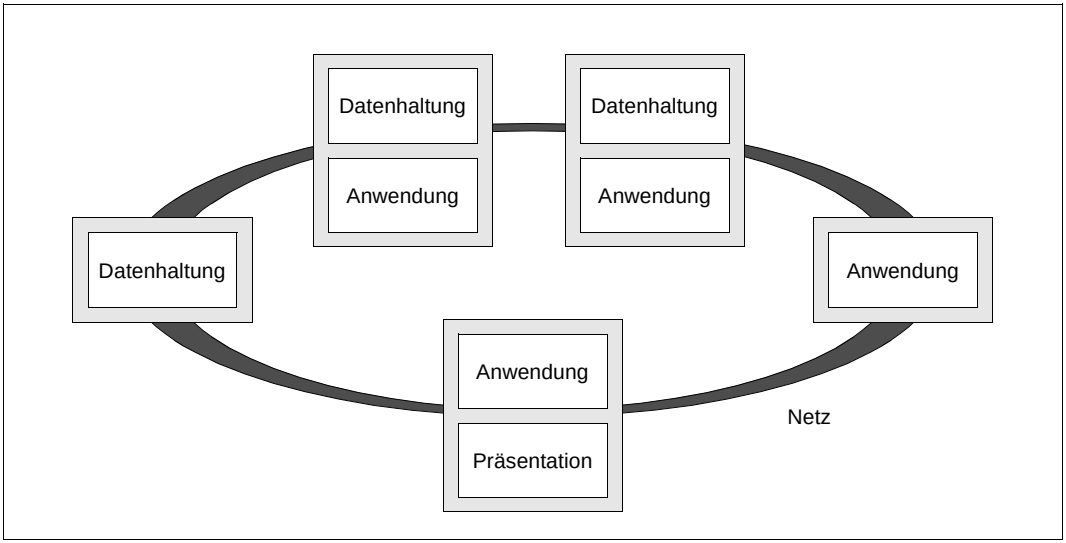


Bild 12: Multi-Tier-Architektur

Nachteile herkömmlicher 2-Tier-Architekturen / Vorteile von Multi-Tier-Verteilung

Viele herkömmliche Client-Server-Anwendungen basieren auf einer klassischen 2-Tier-Architektur: Der Server ist für die Datenhaltung zuständig, während der Client möglichst alle anderen Aufgaben übernimmt. Dieses Modell hat jedoch eine Reihe von Nachteilen:

- Da Clients und Server meist auf SQL-Ebene kommunizieren, gehen dabei oft unnötig große Datenmengen über das Netz (Round Trips). Gerade bei einer großen Benutzerzahl und bei Kommunikation über langsame WAN-Strecken werden die Antwortzeiten inakzeptabel.
- Auch ist bei solchen Architekturen die Systempflege zeitaufwendig und teuer, da die Anwendungslogik auf den Client-Rechnern implementiert ist. Dies führt gerade bei größeren Anwendungen in heterogenen Umgebungen zu nicht unerheblichen Redundanzen und Wartungsproblemen, da jede Änderung der Anwendungslogik eine Nachpflege eventuell mehrerer Quellprogramm-Basen nach sich zieht. Weil das Datenmodell - zumindest teilweise - auf den Clients sichtbar ist, führen Änderungen bei der Datenhaltung unweigerlich zu Anpassungen der Client-Software.
- Erweiterungen der Anwendungslogik führen zu einem Wachstum an Code und Ressourcenbedarf der Clients, es droht das **Fat Client Syndrom**: Die Clients müssen aufgerüstet werden, eventuell ist sogar der Umstieg auf eine leistungsfähigere Plattform notwendig.

- Um die geschilderten Probleme zu vermeiden bieten viele Datenbanksysteme die Möglichkeit, durch Stored Procedures oder Trigger-Funktionen zumindest Teile der Anwendungslogik auf den Server zu verlagern. Dabei werden aber oft proprietäre Script-Sprachen verwendet, die zur Laufzeit interpretiert werden müssen. Dies führt zu Performance-Einbußen. Zudem gibt es keine Möglichkeit, verschiedene Datenbanksysteme innerhalb einer Transaktion zu koordinieren.

Multi-Tier-Architekturen auf Basis von *open*UTM ermöglichen es, diese Einschränkungen und Engpässe zu umgehen. Die logischen Schichten: „Präsentation“, „Anwendung“ und „Datenhaltung“ sind auch physisch klar abgegrenzt und lassen sich beliebig verteilen. Die Netzbelastung wird minimiert, die System-Ressourcen werden geschont, heterogene Datenbanksysteme und Plattformen lassen sich koordinieren. Die Kosten für Systempflege bleiben gering. Bei Erweiterungen der Anwendung entsteht zusätzlicher Ressourcenbedarf nur an einer oder wenigen zentralen Stellen.

3.2 Zu den Begriffen „Client“ und „Server“

Obwohl „Client“ und „Server“ zu den meistgebrauchten Begriffen in der IT-Welt zählen, werden diese Begriffe oft sehr unterschiedlich verwendet:

Ganz allgemein geben diese Begriffe die Rolle an, die die Partner bei der Kommunikation einnehmen: Der **Client** fordert eine Dienstleistung (Service) an, der **Server** erbringt diese Dienstleistung.

Es hat sich eingebürgert, die Begriffe „Client“ und „Server“ auch für gesamte Anwendungen zu verwenden:

So werden UTM-Anwendungen als Server-Anwendungen bezeichnet, weil sie üblicherweise bei der Kommunikation die Server-Rolle einnehmen: Sie stellen Services zur Verfügung. Trotzdem können diese Anwendungen bei der Erfüllung bestimmter Service-Anforderungen selbst wiederum die Hilfe anderer Services nutzen, also die Client-Rolle einnehmen. Im Gegensatz zu den Server-Anwendungen, die die Rolle wechseln können, nehmen Client-Anwendungen bei der Kommunikation immer die Client-Rolle ein. Client-Anwendungen übernehmen in der Regel die Präsentationsaufgaben und bilden das Frontend zu den Benutzern.

Ein Service, oder genauer die Nutzung eines Services, wird bei *open*UTM auch **Vorgang** genannt.

Die Kommunikation zwischen zwei Server-Anwendungen wird auch als **Server-Server-Kommunikation** oder **Peer-to-Peer-Kommunikation** bezeichnet. Damit wird ausgedrückt, daß es sich um zwei gleichwertige Partner handelt, obwohl natürlich auch bei dieser Art der Kommunikation jeweils zwischen Client- und Server-Rolle unterschieden werden kann. Ein weiterer Ausdruck, der oft für diese Art der Zusammenarbeit gebraucht wird, ist **verteilte Verarbeitung**.

Häufig beziehen sich die Begriffe „Server“ und „Client“ auch auf die Hardware. Man spricht von Client-PCs oder Client-Workstations und meint damit Rechner, auf denen Client-Software installiert ist, oder nennt besonders leistungsfähige Rechner „Server“, um auszudrükken, daß sie für Server-Anwendungen besonders geeignet sind.

3.3 *open*UTM-Client-Anwendungen

Klassisches Einsatzgebiet von Client-Anwendungen ist die Präsentation: Sie ermöglichen die Verwendung komfortabler grafischer Benutzeroberflächen. Für die Entwicklung von Client-Anwendungen steht mit *open*UTM-Client ein eigenes Produkt zur Verfügung. Mit *open*UTM-Client können Sie Client-Programme erstellen, die Services von UTM-Anwendungen nutzen. Innerhalb dieser Client-Programme können Sie die X/Open-Programmschnittstellen CPI-C oder XATMI verwenden.

Sie können über diese Programmschnittstellen beispielsweise ein Visual C++- oder Visual Basic-Präsentationsprogramm auf einem PC mit einer UTM-Anwendung verbinden. Durch dieses Client-Programm können Sie z.B. Kommandos zur Administration einer UTM-Anwendung in einer grafischen Oberfläche auf einem PC bereitstellen. Bei Statistikauswertungen können Sie PC-Tools nutzen, indem Sie die von *open*UTM gelieferten Statistikwerte über OLE-Schnittstellen an andere PC-Programme übergeben und grafisch aufbereiten lassen und damit vollständig in Ihre Office-Umgebung integrieren.

Die Security-Funktionen und das komfortable Wiederanlaufverfahren von *open*UTM stehen Ihnen auch beim Anschluß von Client-Anwendungen zur Verfügung.

Der *open*UTM-Client setzt auf einem Trägersystem auf. Das Trägersystem hat die Aufgabe, die Verbindung zu anderen Komponenten wie z.B. dem Transportzugriffssystem herzustellen.

*open*UTM-Clients können entweder auf dem Trägersystem UPIC oder auf dem Trägersystem OpenCPIC aufsetzen. Welches Trägersystem Sie wählen, hängt vom Einsatzfall ab. Die wichtigsten Eigenschaften der beiden Trägersysteme sind im folgenden beschrieben.

Clients mit Trägersystem UPIC

UPIC ist schlankes, sehr performantes und einfach einsetzbares Trägersystem, das auf *open*UTM als Server zugeschnitten ist. Bei UPIC liegt die Initiative zur Kommunikation immer beim Client-Programm. Zur Kommunikation wird das UPIC-Protokoll benutzt.

Ein UPIC-Client kann die UTM-Funktionen optimal nutzen, denn UPIC
- unterstützt z.B. Funktionstasten,
- die Verschlüsselung von Zugangs- und Benutzerdaten
- und kann den Service SIGNON nutzen.

Ein UPIC-Client kann nach einem Störfall den Status der letzten Transaktion anfordern und ist dadurch in das Wiederanlaufkonzept von *open*UTM mit einbezogen.

Ein UPIC-Client kann innerhalb eines Programmlaufs mehrere Conversations gleichzeitig unterhalten („Multi-Conversations“), sofern das entsprechende System „Multi-Threading“ unterstützt.

UPIC gibt es auf allen gängigen UNIX-Plattformen, auf MS-Windows, Windows NT oder unter BS2000/OSD. Auf Windows bietet UPIC ein ActiveX Control und ist daher besonders gut in die Windows-Umgebung integriert.

Clients mit Trägersystem OpenCPIC

Das Trägersystem OpenCPIC ist mächtiger und damit auch komplexer als UPIC. Bei OpenCPIC kann die Initiative zur Kommunikation auch beim Server-Programm liegen. Zur Kommunikation wird das OSI TP-Protokoll benutzt.

Ein OpenCPIC-Client kann auch mit OpenCPIC-Anwendungen, CPI-C-Anwendungen und anderen Nicht-UTM-Anwendungen kommunizieren, sofern diese ebenfalls OSI TP verwenden. Über die XA-Schnittstelle kann ein OpenCPIC-Client auch mit einem Resource-Manager zusammenarbeiten.

Ein OpenCPIC-Client kann Beginn und Ende einer globalen Transaktion bestimmen und damit in die globale Transaktionsklammer mit einbezogen werden. Die Transaktionssteuerung erfolgt über die X/Open-Schnittstelle TX, die Kommunikation über die Schnittstelle CPI-C.

Die Verschlüsselung von Zugangs- und Benutzerdaten ist nicht möglich.

Clients mit dem Trägersystem OpenCPIC gibt es auf UNIX- und Windows-Plattformen.

Wie beim Anschluß von Client-Anwendungen die Security-Funktionen von *open*UTM genutzt werden können, ist in diesem Handbuch in Kapitel 9 beschrieben. Nähere Informationen zum Wiederanlauf-Verhalten finden Sie ebenfalls in diesem Handbuch in Kapitel 10. Zum Produkt *open*UTM-Client gibt es für die Trägersysteme UPIC und OpenCPIC jeweils ein eigenes Handbuch. Dort erfahren Sie alle Einzelheiten, die Sie für die Programmierung und den Anschluß von Client-Anwendungen benötigen.

3.3.1 ActiveX und Com*Transactions*

*open*UTM-Client mit Trägersystem UPIC bietet eine ActiveX-Schnittstelle, um UTM-Anwendungen über den COM/DCOM-Standard in die Microsoft-Welt zu integrieren. Wer weiteren Komfort wünscht, kann das Aufsatzprodukt Com*Transactions* einsetzen.

ActiveX-Schnittstelle

Der UPIC-Client besitzt das ActiveX Control Upic.ocx, mit dem Sie schnell und einfach *open*UTM-Client-Anwendungen auf Basis von Visual Basic, Visual C++ oder Borland Delphi erstellen können.

Upic.ocx setzt auf der CPI-C-Schnittstelle von UPIC auf und hat gegenüber dieser wesentliche Vorteile:

- Sie können mit Standard-Microsoft-Tools eine graphische Oberfläche für Ihre UTM-Anwendung erstellen
- Upic.ocx ist wesentlich einfacher zu handhaben, da die CPI-C-Verwaltungsaufrufe in die Schnittstelle integriert sind. Zudem werden Eigenschaften über eine graphische Oberfläche und nicht per Funktionsaufruf gesetzt.

Zusätzlich enthält der Client einen externen Automation Server, der auf Upic.ocx aufbaut. Damit können Sie aus Windows-Standard-Anwendungen wie Word oder Excel auf die UTM-Anwendung zugreifen. Der Automation Server ist in Visual Basic implementiert und wird auch in Source-Form zur Verfügung gestellt, so daß Sie ihn als Vorlage für eigene Automation-Bausteine verwenden können.

Com*Transactions*

Com*Transactions* ist ein eigenes Produkt und bietet ebenso wie der UPIC-Client eine Programm-Schnittstelle sowie einen Automation Server, um UTM-Services in Standard-Microsoft-Anwendungen zu integrieren.

Im Vergleich zur ActiveX-Schnittstelle des *open*UTM-Client bietet Com*Transactions* folgende Zusatzfunktionen an:

- Katalog und Analysewerkzeuge:
  Sie können die UTM-Services zentral in einem Katalog verwalten und mehreren unterschiedlichen Clients zugänglich machen. Dazu bietet Com*Transactions* Analyse-Werkzeuge, mit denen die Struktur von UTM-Eingaben und -Ausgaben (z.B. FHS-Formate) in das Objektmodell integriert werden können.

  Damit kann die Client-Anwendung auf die Daten der UTM-Services per Feldnamen zugreifen, was die Anbindung an bestehende UTM-Anwendungen wesentlich erleichtert.

- Automatisierung eines Sitzungsablaufs:
  Sie können eine Anwendungssitzung mit dem Dialog-Rekorder als Visual Basic-Programm aufzeichnen. Damit lassen sich komplexe Programmbausteine für den Service-Zugriff erstellen.
- Multi-Tier-Anwendungen:
  Com*Transactions* besitzt einen Klassen-Generator, der aus den Aufzeichnungen des Dialog-Rekorders Klassen erzeugen kann. Mit Hilfe dieser Klassen sowie DCOM können Sie Multi-Tier-Anwendungen erstellen, indem Sie die Kommunikationskomponente auf einen eigenen Windows NT-Rechner legen:

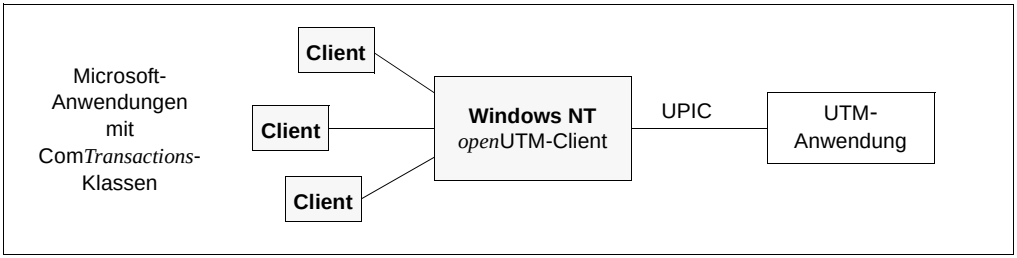


Bild 13: Multi-Tier-Anwendungen mit Com*Transactions*

  Diese Multi-Tier-Lösung ist skalierbar, d.h. es können sehr einfach weitere Clients an den NT-Server angeschlossen werden und die UTM-Services nutzen. Zudem ist diese Client-Anbindung an *open*UTM performanter als ein 2-Tier-Client.

Mit Com*Transactions* lassen sich auch OSD-Anwendungen integrieren, die Daten über das 9750-Protokoll austauschen.

3.3.2 Web*Transactions* und *open*UTM-JetClient

Diese beiden Produkte dienen dazu, neue oder bestehende UTM-Anwendungen an das Web anzuschließen.

Bei beiden Produkten wird ein *open*UTM-Client mit Trägersystem UPIC verwendet. Dieser UPIC-Client spielt den Ansprechpartner für die Benutzer im Web. Daher muß die UTM-Anwendung nur diesen Client kennen, nicht aber die Benutzer im Web oder deren Rechner.

Web*Transactions* for *open*UTM

Mit Web*Transactions* können Sie die Ein- und Ausgaben von *open*UTM im Web als HTML-Dokumente bereitstellen und die Dialogabläufe anpassen bis hin zur Anwendungsintegration. Damit sind Sie in der Lage, Geschäftsprozesse mit einer modernen Oberfläche zu versehen, diese zu optimieren und die Prozesse bei Bedarf mit anderen Prozessen zu verknüpfen, ohne die zugrundeliegenden Anwendungen ändern zu müssen.

Web*Transactions* hat damit fein abgestufte Einsatzmöglichkeiten:

- Automatische 1:1-Umsetzung
  Hier konvertiert Web*Transactions* vorhandene Formate in HTML und zurück. Dies ist die einfachste und schnellste Möglichkeit, UTM-Daten im Web bereitzustellen.
- Gestaltung der Oberfläche
  Sie können der 1:1-Umsetzung ein modernes Outfit geben und die Oberfläche dem Corporate Design Ihres Unternehmens anpassen. Dabei stehen Ihnen alle HTML- und JavaScript-Möglichkeiten offen.
- Gestaltung der Dialogabläufe
  Sie können Zwischendialoge einfügen oder Ein-/Ausgabe-Elemente herausfiltern und dadurch die 1:1-Zuordnung zwischen HTML-Seite und UTM-Format aufheben.
- Anwendungsintegration
  Sie können eine UTM-Anwendung und eine oder mehrere OSD- oder MVS-Anwendungen in einer Web*Transactions*-Anwendung vereinigen. Diese Anwendungen können auf unterschiedlichen Plattformen laufen. Dabei muß die Logik der beteiligten Anwendungen nicht geändert werden.

Alle Konvertierungs- und Integrationsleistungen werden zentral von Web*Transactions* erbracht.

*open*UTM-JetClient

Mit *open*UTM-JetClient (*open*UTM Java Enterprise Technology Client) können Sie in Java geschriebene Clients an *open*UTM-Anwendungen anschließen. Dabei kann der Client entweder als Java-Anwendungsprogramm oder als Java-Applet erstellt werden.

Der *open*UTM-JetClient besteht aus dem Koppelbaustein JetClientConnect sowie den Klassenbibliotheken, die für die Client-Programme benötigt werden. JetClientConnect läuft auf einem UNIX-Rechner (künftig auch auf Windows NT) und stellt die Verbindung zur UTM-Anwendung her. Die UTM-Anwendung kann sich auf jedem beliebigen Rechner im Netz befinden, auch auf dem gleichen Rechner wie JetClientConnect.

Für Anwendungen wie Electronic Commerce kann der Datenverkehr zwischen dem Client und dem Koppelbaustein JetClientConnect verschlüsselt werden. Dabei wird das SSL-Protokoll verwendet.

Java-Anwendungsprogramm

Das Java-Anwendungsprogramm kann auf einem beliebigen Rechner im Netz laufen, da es zur Kommunikation mit JetClientConnect eine Sockets-Verbindung verwendet.

Bild 14: *open*UTM-Client als Java-Anwendungsprogramm

Java-Applet

Wenn Sie den *open*UTM-Client als Java-Applet realisieren, dann können Sie sowohl die Vorteile der Web-Anbindung als auch der Java-Technologie nutzen. In diesem Fall wird das Java-Applet über einen Web-Browser in den Client-Rechner geladen. Dabei müssen sich sowohl der Koppelbaustein als auch das Java-Applet auf dem Web-Server befinden, denn das Java-Applet darf nur zu *dem* Rechner eine Verbindung aufbauen, von dem es heruntergeladen wurde.

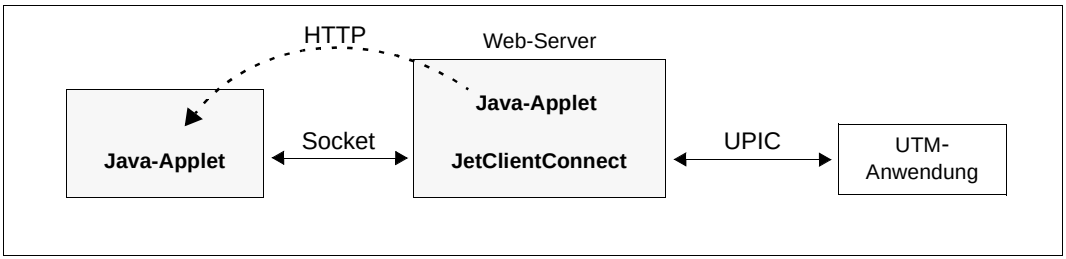


Bild 15: *open*UTM-Client als Java-Applet

Nachdem das Applet gestartet wurde, verwendet es ebenfalls eine Socket-Verbindung zum JetClientConnect.

3.4 Server-Server-Kommunikation

Zur Erfüllung von Service-Anforderungen kann eine Server-Anwendung selbst wiederum Services anderer Anwendungen in Anspruch nehmen, d.h., sie kann selbst Aufträge an andere Anwendungen richten. Diese Form der Verarbeitung wird auch als „Verteilte Verarbeitung“ bezeichnet. Bei der Server-Server-Kommunikation arbeiten also Services in zwei oder mehreren Anwendungen zusammen, um einen Auftrag zu bearbeiten, den ein Client gestellt hat. Ein Service, der eine Dienstleistung von einem anderen Service anfordert, wird **Auftraggeber-Service** genannt, der beauftragte Service, der diese Serviceleistung erbringt, **Auftragnehmer-Service**.

Die Server-Server-Kommunikation bietet auch die Möglichkeit, zwischen zwei Anwendungen mehrere parallele Verbindungen aufzubauen und mehrere Aufträge gleichzeitig zu bearbeiten.

Eine Anwendung kann nicht nur im Dialog mit anderen Anwendungen zusammenarbeiten, bei der Server-Server-Kommunikation kann auch die Message Queuing Funktionalität von *open*UTM genutzt werden (siehe Abschnitt 4.2.2 auf Seite 73).

Da die Server-Server-Kommunikation über den reinen Austausch von Nachrichten hinausgeht, sind hierfür spezielle höhere Kommunikationsprotokolle notwendig: *open*UTM unterstützt das LU6.1-Protokoll und das international standardisierte Protokoll OSI TP. Die Nutzung dieser weit verbreiteten Kommunikationsprotokolle hat den Vorteil, daß eine UTM-Anwendung nicht nur mit anderen UTM-Anwendungen zusammenarbeiten kann, sondern auch mit Anwendungen anderer Hersteller, z.B. mit CICS/IMS- oder Tuxedo-Anwendungen, selbst dann, wenn diese auf anderen Plattformen laufen.

3.4.1 Anwendungsübergreifende Dialoge

Bei einem anwendungsübergreifenden Dialog laufen Auftraggeber-Service und Auftragnehmer-Service synchron ab und nicht entkoppelt wie beim Message Queuing (siehe Seite 69ff). Dabei ist es gleichgültig, ob der Auftraggeber-Service selbst im Dialog oder über Message-Queuing gestartet wurde.

Bei anwendungsübergreifenden Dialogen sind auch komplexe Strukturen möglich:

– Ein Auftraggeber-Service kann innerhalb einer Transaktion mit mehreren Auftragnehmer-Services kommunizieren.

– Ein Auftragnehmer-Service, der im Dialog eine Teilaufgabe für einen anderen Auftraggeber-Service bearbeitet, kann selbst wieder einen dritten Dialog-Service in einer weiteren UTM-Anwendung beauftragen.

Durch solche parallelen und geschachtelten Strukturen entstehen mehrstufige Hierarchiebeziehungen, die sich durch Baumdiagramme darstellen lassen. Eine solche Hierarchie wird als **Service-Hierarchie** bezeichnet.

Aus dem folgenden Bild ist ersichtlich, daß ein Service gleichzeitig die Rolle des Auftragnehmer-Service wie auch die eines Auftraggeber-Service einnehmen kann. Die Services B, C und D sind Auftragnehmer-Services gegenüber Service A, die Services B und C sind zugleich Auftraggeber-Services gegenüber den Services E bzw. F und G auf der untersten Ebene.

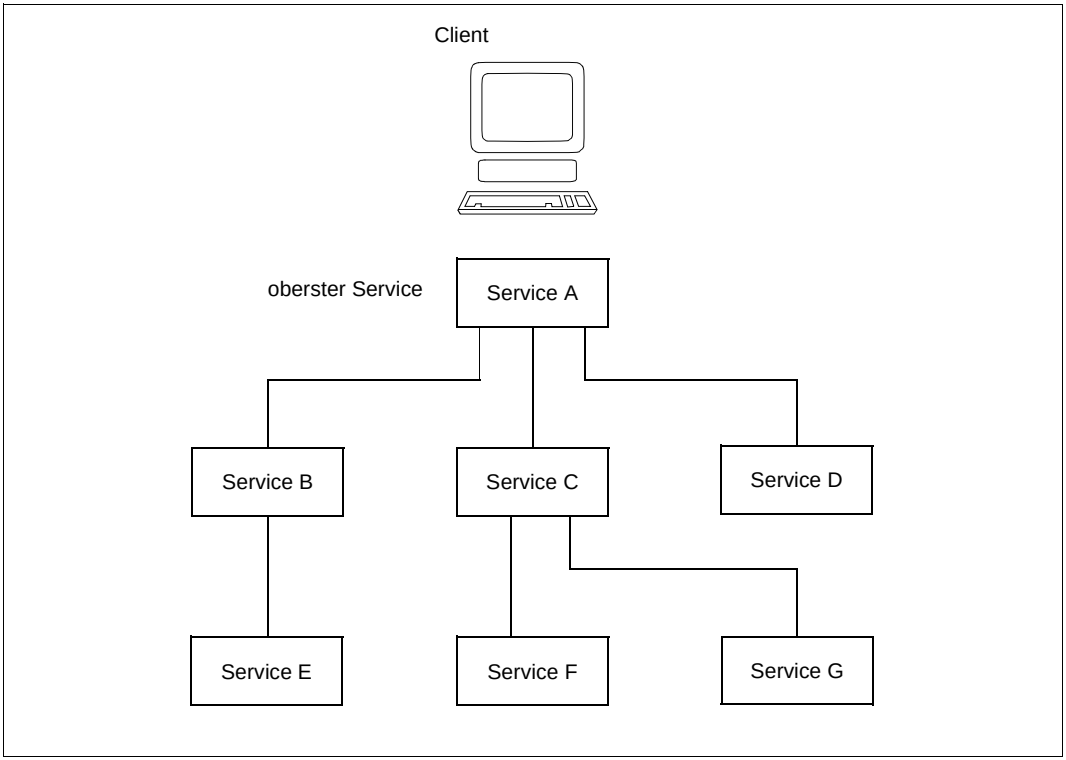


Bild 16: Service-Hierarchie bei anwendungsübergreifenden Dialogen

Jede Service-Hierarchie besitzt einen obersten Service. Eine Service-Hierarchie kann nur so lange bestehen, wie der oberste Service aktiv ist. Durch Adressierung neuer Auftragnehmer-Services und durch Beendigung bestehender Auftragnehmer-Services ändert sich eine Service-Hierarchie im Laufe der Bearbeitung eines Dialogauftrags.

Programmierung anwendungsübergreifender Dialoge

Bei anwendungsübergreifenden Dialogen wird eine Aufgabe von mehreren Teilprogrammen in verschiedenen Anwendungen bearbeitet. Damit kann ein Teilprogramm von Terminals, von Client-Programmen, von Programmen der eigenen Anwendung und von fremden

Anwendungen aus angesprochen werden. Es muß daher abhängig vom Partner entscheiden, welche Aufgabe es bearbeitet, an wen es die Nachricht schickt und ob es eine globale Transaktion beenden muß oder nicht.

Auch solche komplexen Formen der Programm-Programm-Kommunikation lassen sich mit *open*UTM einfach und sicher erstellen, da *open*UTM zur Steuerung anwendungsübergreifender Dialoge komfortable Kontrollmöglichkeiten zur Verfügung stellt.

Ausführliche Informationen über die Programmierung anwendungsübergreifender Dialoge und über die Kontrollmöglichkeiten, die Ihnen hierfür zur Verfügung stehen, finden Sie in den *open*UTM-Programmierhandbüchern: „Anwendungen programmieren mit KDCS für COBOL, C und C++“ und „Anwendungen erstellen mit X/Open-Schnittstellen“.

3.4.2 Transaktionssicherung bei Server-Server-Kommunikation

UTM-Services bearbeiten Aufträge transaktionsorientiert, d.h., ein UTM-Service besteht immer aus einer Transaktion oder aus mehreren Transaktionen. Kommuniziert ein UTM-Service mit einer anderen Anwendung über OSI TP, kann festgelegt werden, ob die Verarbeitung in der fernen Anwendung mit in die Transaktion eingeschlossen werden soll oder ob sie unabhängig davon erfolgen soll. Im ersten Fall spricht man von einer globalen Transaktion, im zweiten Fall von unabhängigen Transaktionen. Über das LU6.1-Protokoll wird immer mit globalen Transaktionen gearbeitet.

Globale Transaktionen

Bei der Server-Server-Kommunikation sind an der Bearbeitung eines Auftrags mehrere lokale Transaktionen in verschiedenen Anwendungen beteiligt. Falls mit globaler Transaktionssicherung gearbeitet wird, garantiert *open*UTM, daß sich die Daten auch anwendungsübergreifend zu jeder Zeit in einem konsistenten Zustand befinden. Hierzu synchronisiert *open*UTM das Ende dieser Transaktionen: In allen beteiligten Anwendungen werden bei erfolgreichem Abschluß die Sicherungspunkte gleichzeitig gesetzt. Im Fehlerfall sorgt *open*UTM dafür, daß alle beteiligten Transaktionen zurückgesetzt werden. Die synchronisierten Transaktionen bilden also eine Einheit, die auch als „verteilte Transaktion“ oder „globale Transaktion“ bezeichnet wird. Für diese Synchronisation verwendet *open*UTM das Two-Phase-Commit-Verfahren: Ist die Bearbeitung einer lokalen Transaktion abgeschlossen, so wird diese Transaktion zunächst in den Zustand „Prepare-to-Commit“ versetzt. Erst wenn alle beteiligten Transaktionen diesen Zustand erreicht haben, wird das globale Transaktionsende (Commit) gesetzt. Ein Beispiel hierfür finden Sie in Abschnitt 3.4.3 auf Seite 58.

Asynchronaufträge (MQ-Aufträge) werden bei Server-Server-Kommunikation mit globaler Transaktionssicherung genau einmal übertragen. Das bedeutet, daß auch bei Netzstörungen oder Abbruch einer Anwendung der Asynchronauftrag weder verloren geht noch die Nachricht verdoppelt wird.

Globale Transaktionssicherung ist immer dann notwendig, wenn hohe Anforderungen an die Datenkonsistenz und Datensicherheit gestellt werden.

Unabhängige Transaktionen

Im Gegensatz zur globalen Transaktion setzt bei der Zusammenarbeit zweier unabhängiger Transaktionen jede Anwendung ihre lokale Transaktion selbständig vor oder zurück. Dies kann, z.B. bei Fehlern in der Kommunikation, zu inkonsistenten Datenbeständen in den verschiedenen Anwendungen führen. Bei dieser Form der Kommunikation ist auch nicht gewährleistet, daß Asynchron-Aufträge genau einmal übertragen werden.

Diese Form der Zusammenarbeit von Anwendungen ist z.B. dann sinnvoll, wenn bei der Bearbeitung eines Auftrags nur die Datenbestände **einer** Anwendung verändert werden, wie dies z.B. der Fall ist, wenn die andere Anwendung eine reine Retrieval-Anwendung ist. Die Kommunikation wird dabei effizienter abgewickelt, da die Transaktionen in den beteiligten Anwendungen nicht synchronisiert werden müssen.

3.4.3 Beispiel: Anwendungsübergreifender Dialog mit globaler Transaktion

Ein Kunde einer Bank möchte in einer Zweigstelle einen Geldbetrag abheben. Sein Konto wird jedoch von einer anderen Zweigstelle geführt. Jede Zweigstelle setzt zur Verwaltung ihrer Datenbestände eine eigene UTM-Anwendung ein.

Wenn der Kunde Geld abhebt, dann müssen u.a. die Datenbestände beider UTM-Anwendungen geändert werden, d.h., der Kassenstand in der einen und der Kundenkontostand in der anderen Anwendung. Realisiert wird dieser Geschäftsvorgang mit einer einzigen Dialog-Transaktion, welche auf beide Anwendungen verteilt wird (globale Transaktion). Die Buchungen (Kasse und Kundenkonto) werden in lokalen Dialog-Services durchgeführt. Wie diese lokalen Dialog-Services miteinander kommunizieren, wird im folgenden Bild veranschaulicht.

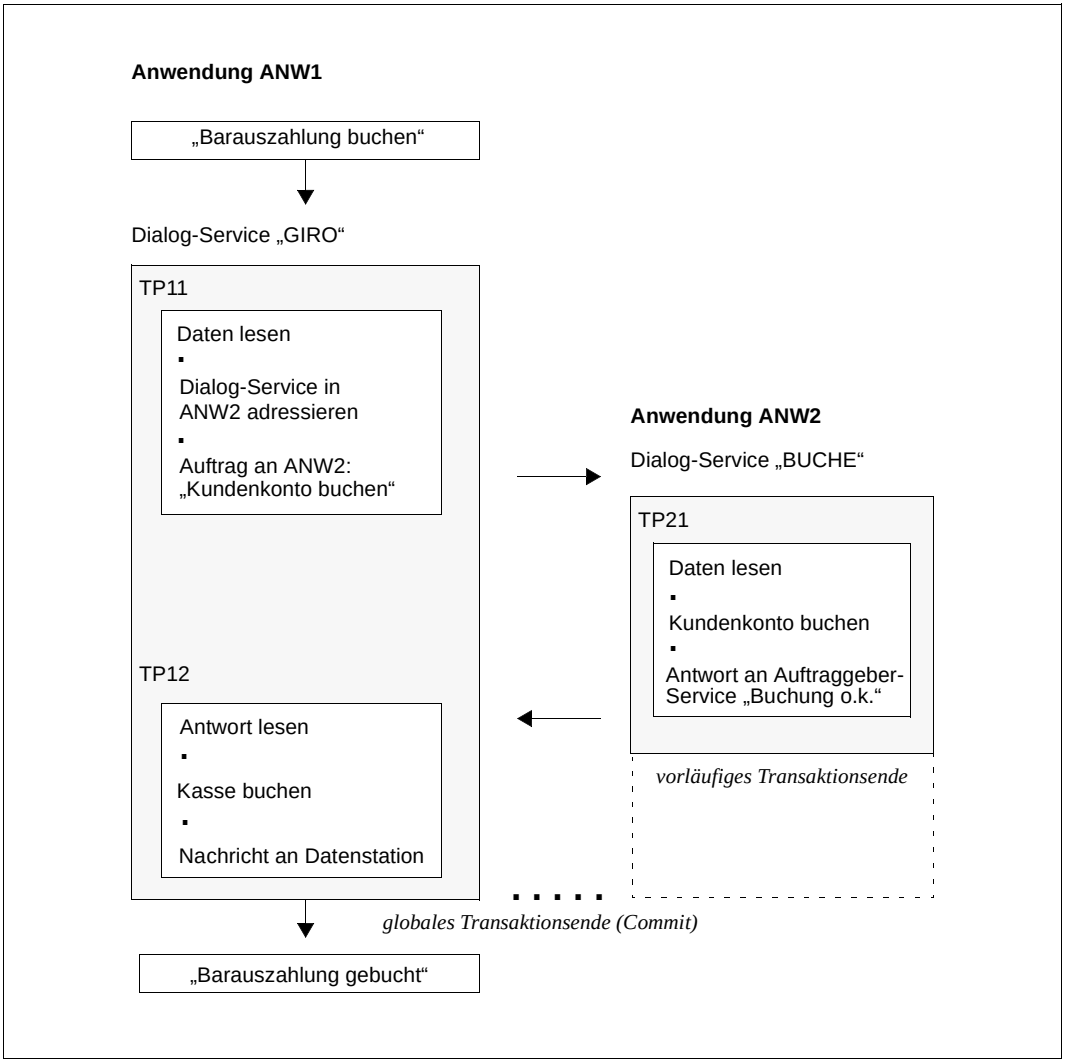


Bild 17: Dialogauftrag mit globaler Transaktionssicherung

Der Service „GIRO“ in der UTM-Anwendung ANW1 fungiert hier als **Auftraggeber-Service**. Dieser Service adressiert im Dialogteilprogramm TP11 den **Auftragnehmer-Service**, d.h. den Service „BUCHE“ in der UTM-Anwendung ANW2.

Das Teilprogramm TP11 beendet den Verarbeitungsschritt und hält seine Transaktion offen. Das Teilprogramm TP21 im Service „BUCHE“ erhält die Daten per Dialognachricht. TP21 schickt eine Dialogantwort an den Auftraggeber-Service zurück und fordert anschließend von *open*UTM Transaktionsende an.

Diese Dialogantwort startet im Service „GIRO“ das Folgeteilprogramm TP12, welches die Buchungsaufgabe beendet und dann ebenfalls Transaktionsende anfordert. *open*UTM synchronisiert nun die beiden lokalen Sicherungspunkte (= Transaktionsende) und setzt einen gemeinsamen Sicherungspunkt. Die beiden lokalen Transaktionen (in Anwendung ANW1 und Anwendung ANW2) bilden somit eine Sicherungseinheit, eine globale Transaktion.

Tritt ein Fehler auf, bevor der gemeinsame Sicherungspunkt gesetzt wurde, dann wird die gesamte Sicherungseinheit, d.h. jede der beiden lokalen Transaktionen, zurückgesetzt.

3.4.4 Adressierung ferner Services

Bevor ein ferner Service angefordert werden kann, muß er adressiert werden. Hierzu dient ein Adressierungsaufruf in der Serviceroutine des Auftraggebers. In diesem Aufruf gibt der Auftraggeber-Service den logischen Namen des fernen Service an und ordnet dem fernen Service eine Identifikation zu. Diese Service-Identifikation wird im Auftraggeber-Service bei allen Aufträgen an den fernen Service angegeben sowie jeweils beim Lesen der Ergebnisse.

Der ferne Service und die ferne Anwendung werden immer mit ihren logischen Namen angesprochen. Bei der Generierung werden der logische Name für eine ferne Anwendung (LPAP) und der logische Name für den fernen Service (LTAC) definiert und mit dem tatsächlichen Namen in der Partneranwendung verknüpft. Der logische Servicename entspricht in seiner Funktion dem Transaktionscode des Service. Er kann auf zwei Arten mit einer Partneranwendung verknüpft werden:

- Per Generierung
  In diesem Fall spricht man von **einstufiger Adressierung**, denn die Partneranwendung muß nicht im Adressierungsaufruf angegeben werden.
- Im Programm beim Adressierungsaufruf
  In diesem Fall spricht man von **zweistufiger Adressierung**. Diese Art der Adressierung ist dann sinnvoll, wenn der gleiche Service in mehreren Anwendungen gestartet werden kann.

Die beiden Arten der Adressierung eines fernen Services werden in den folgenden Bildern veranschaulicht. Das genaue Format der Adressierungsaufrufe finden Sie in den *open*UTM-Programmierhandbüchern: „Anwendungen programmieren mit KDCS für COBOL, C und C++“ und „Anwendungen erstellen mit X/Open-Schnittstellen“

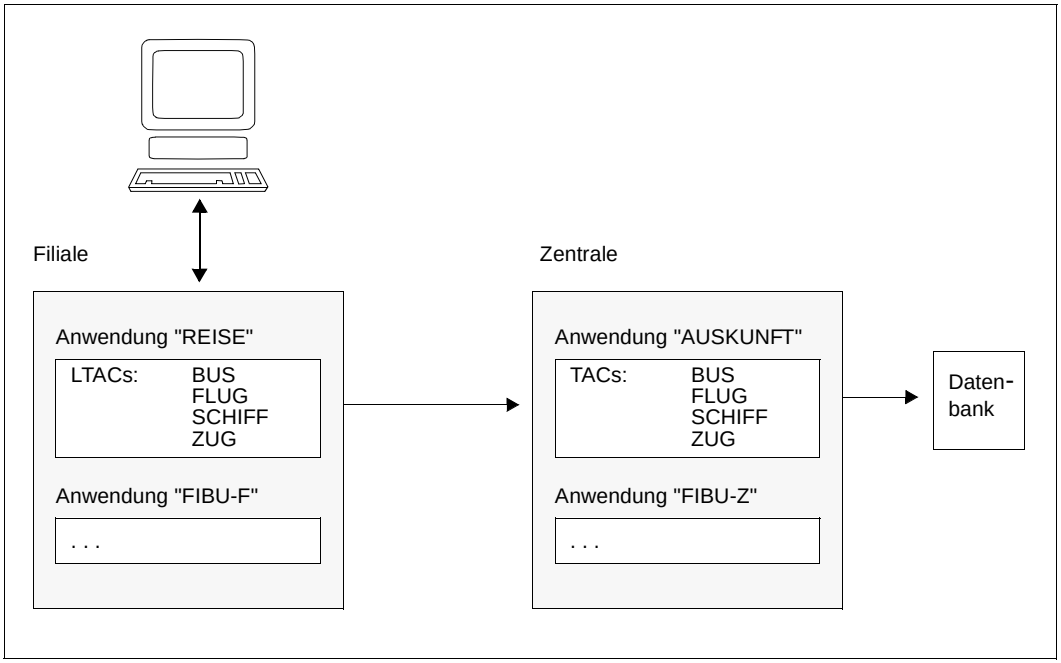


Bild 18: Einstufige Adressierung

In der Filiale eines Reiseunternehmens läuft die UTM-Anwendung „REISE“, die sich die nötigen Informationen von der UTM-Anwendung „AUSKUNFT“ in der Zentrale des Unternehmens holt. Die einzelnen Services, bezeichnet mit den logischen Namen „BUS“, „FLUG“, „SCHIFF“ und „ZUG“, sind per Generierung fest mit der Anwendung „AUSKUNFT“ verknüpft. Daher genügt es, im Programm den logischen Service-Namen anzugeben. Der logische Name der Partner-Anwendung muß im Programm nicht angegeben werden.

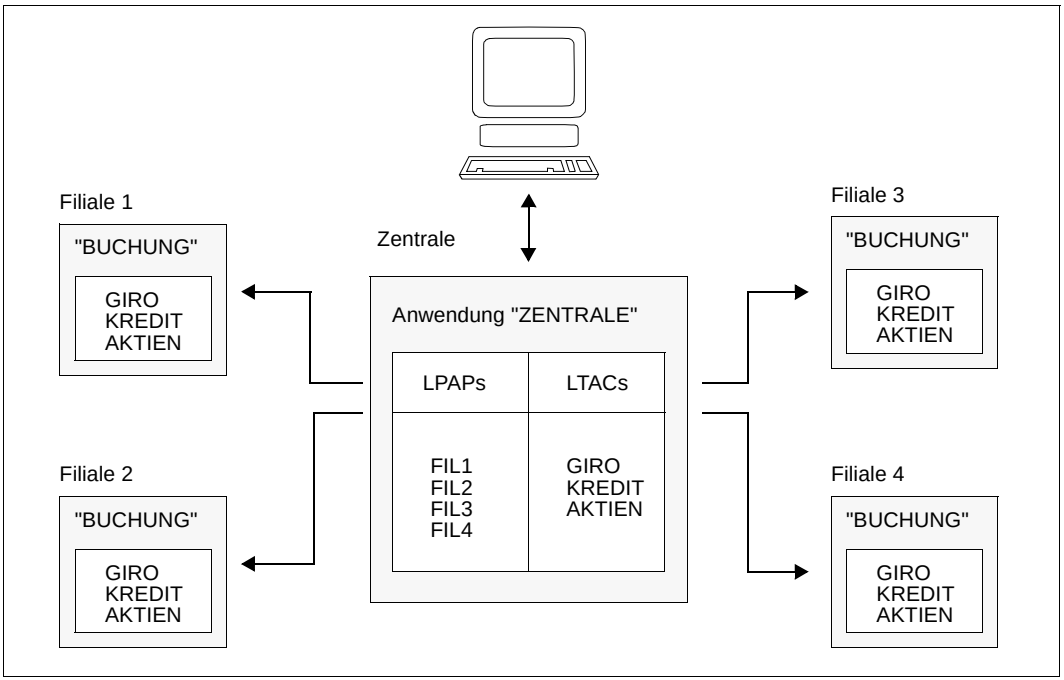


Bild 19: Zweistufige Adressierung

In jeder der vier Filialen einer Bank läuft die gleiche Anwendung „BUCHUNG“. Die Anwendung „ZENTRALE“ in der Bankzentrale darf auf verschiedene, in allen Filialen identische Services zugreifen. Der Auftraggeber-Service in der Bankzentrale muß daher im Programm sowohl den Service als auch die Partneranwendung auswählen.

3.4.5 Kommunikation mit CICS- und IMS-Anwendungen

*open*UTM kann über die SNA-Protokolle LU6.1 und LU6.2 mit CICS- und IMS-Anwendungen gekoppelt werden. Der Anschluß an das SNA-Netz läuft über das Transportsystem-Gateway TRANSIT.

- Mit LU6.1-Partnern kann *open*UTM direkt kommunizieren. Dabei wird immer mit globaler Transaktionssicherung gearbeitet. Aus Sicht von *open*UTM ist CICS oder IMS ein UTM-Partner. Aus Sicht von CICS oder IMS ist *open*UTM ein CICS- bzw. IMS-Partner.
- Einen LU6.2-Partner spricht *open*UTM über *open*UTM-LU62 an. Aus Sicht von *open*UTM ist CICS oder IMS ein OSI TP-Partner. Aus Sicht von CICS oder IMS ist *open*UTM ein LU6.2-Partner. Bei einer solchen Kopplung können die Partner sowohl mit als auch ohne globale Transaktionssicherung arbeiten.

Auftraggeber und Auftragnehmer

Eine UTM-Anwendung kann bei der Kopplung mit CICS oder IMS sowohl Auftraggeber- als auch Auftragnehmer-Services enthalten.

Bei *open*UTM sind diese beiden Rollen nicht gleichberechtigt, d.h. beim Beenden einer globalen Transaktion gelten bestimmte Regeln für Auftraggeber- und Auftragnehmer-Services. Diese Regeln müssen von den CICS- und IMS-Anwendungsprogrammen eingehalten werden, auch wenn diese an dieser Stelle ein anderes Vorgehen erlauben würden.

openUTM als Auftrageber

Ein UTM-Auftraggeber-Service adressiert einen CICS- oder IMS-Auftragnehmer-Service genauso wie einen UTM-Auftragnehmer-Service. Dabei kann *open*UTM sowohl Dialog- als auch Asynchronaufträge erteilen (Message Queuing).

CICS als Auftraggeber

CICS kann sowohl Dialog- als auch Asynchron-Services in *open*UTM starten.

IMS als Auftraggeber

Hier hängen die Möglichkeiten von der Kopplungsart ab:

- Bei einer LU6.2-Kopplung kann IMS sowohl Dialog- als auch Asynchron-Services in *open*UTM starten.
- Bei einer LU6.1-Kopplung kann IMS dagegen nur Asynchron-Services in *open*UTM starten. Damit IMS dennoch Dialoge führen kann, bietet LU6.1 die Möglichkeit, Pseudo-Dialoge zwischen Asynchronvorgängen zu bilden. Bei einem Pseudo-Dialog liefert der Auftraggeber zusätzliche Informationen, mit deren Hilfe der Auftragnehmer die Antwort an den richtigen Partner zurückschicken kann. *open*UTM besitzt zu diesem Zweck eine spezielle Erweiterung an der Programm-Schnittstelle.

  Pseudo-Dialoge sind auch mit CICS-Partnern möglich.

Wie Sie eine Kopplung mit CICS und IMS generieren und programmieren, ist ausführlich im Handbuch „Verteilte Transaktionsverarbeitung zwischen *open*UTM- und CICS, IMS- und LU6.2-Anwendungen“ beschrieben.

3.5 Kommunikation mit Transportsystem-Anwendungen

Neben der transaktionsorientierten verteilten Verarbeitung, die höhere Kommunikationsprotokolle voraussetzt, kann eine UTM-Anwendung ohne globale Transaktionssicherung auch mit Anwendungen kommunizieren, die direkt auf der Transportsystemschnittstelle aufsetzen. Beispiele für solche Anwendungen sind CMX-Anwendungen in UNIX, DCAM- oder PDN-Anwendungen oder beliebige TCP/IP-Sockets-Anwendungen.

Da keine höheren Protokolle verwendet werden, ist bei der Kommunikation mit Transportsystem-Anwendungen eine automatische Unterstützung globaler Transaktionen nicht möglich. *open*UTM kann in diesem Fall nur lokale Transaktionssicherheit gewährleisten.

Bei Störungen im Transportsystem oder bei abnormalem Ende der UTM-Anwendung ist nicht sichergestellt, daß eine Nachricht, die an eine andere Anwendung gesendet wurde, von dieser auch empfangen und gesichert werden konnte. In diesem Fall ist sowohl ein Verlust als auch eine Verdoppelung der Nachricht möglich.

Im folgenden ist beschrieben, was bei der Zusammenarbeit einer UTM-Anwendung mit Transportsystemanwendungen zu beachten ist.

Verbindungsaufbau

Die Initiative zum Verbindungsaufbau kann von der anderen Anwendung oder von der UTM-Anwendung ausgehen.

Anmelden an die UTM-Anwendung

Baut die Transportsystem-Anwendung die Verbindung auf, dann kann sie sich - ähnlich wie Terminalbenutzer oder Clients - unter einer Benutzerkennung an eine UTM-Anwendung anmelden.

Besitzt die UTM-Anwendung einen selbsterstellten Anmeldedialog (Event-Service SIGNON, siehe Seite 122), dann wird dieser auch beim Anmelden von Transportsystem-Anwendungen durchlaufen. Dieser Anmeldedialog kann z.B. dazu genutzt werden, der Transportsystem-Anwendung für die nachfolgende Verarbeitung eine (andere) Benutzerkennung zuzuweisen.

Transportsystem-Anwendungen können die „Multi-Signon“-Funktion nutzen, d.h. es können sich gleichzeitig mehrere Transportsystem-Anwendungen unter derselben UTM-Benutzerkennung anmelden.

Eine Transportsystem-Anwendung kann unter demselben Anwendungsnamen mehrere parallele Transportverbindungen zu einer UTM-Anwendung aufbauen („Multi-Connect“).

Bearbeitung von Aufträgen

Eine Transportsystem-Anwendung kann sowohl Asynchron- als auch Dialog-Aufträge an die UTM-Anwendung richten. Dabei muß folgendes beachtet werden:

– Die Aufträge müssen in der von *open*UTM erwarteten Weise formuliert sein; d.h. die ersten acht Zeichen der Nachricht müssen den Transaktionscode enthalten, unter dem der zu startende Service bei *open*UTM generiert wurde. Ob es sich bei diesem Service um einen Dialog- oder einen Asynchron-Service handelt, erkennt *open*UTM anhand des Transaktionscodes.

– Sockets-Anwendungen schicken einen Byte-Strom, während *open*UTM nachrichtenorientiert arbeitet. Damit die UTM-Anwendung Nachrichtengrenzen erkennen kann, muß eine Sockets-Anwendung der Nachricht ein Protokoll-Feld voranstellen.

– Bei Dialog-Aufträgen muß für die Einhaltung des von *open*UTM geforderten strengen Dialogs gesorgt werden; d.h. die Partneranwendung muß den Empfang einer Antwort von der UTM-Anwendung abwarten, bevor sie die nächste Nachricht senden darf.

– *open*UTM sendet keine Nachrichten der Länge 0 an Transportsystem-Anwendungen. Obwohl eine Nachricht der Länge 0 nicht gesendet wird, wechselt bei einer solchen Nachricht das Senderecht und *open*UTM wartet danach auf eine Nachricht der Transportsystemanwendung. Deshalb ist es notwendig, bei Mehrschritt-Dialogen auf die Logik des Dialogablaufs zu achten.

– Die Ausgabenachrichten werden nicht fragmentiert. Damit ist die Länge einer Gesamtnachricht durch die Größe des Ausgabepuffers der UTM-Anwendung beschränkt.

– *open*UTM führt bei Nachrichten, die für andere Anwendungen bestimmt sind, keine Formatierung durch, d.h. die andere Anwendung erhält die Nachricht so, wie sie vom Teilprogramm im Nachrichtenbereich bereitgestellt wurde.

– Restart-Fähigkeit: Gegebenenfalls wird ein Service-Wiederanlauf für Dialog-Services durchgeführt und dabei die letzte Ausgabenachricht wiederholt.

Während des Betriebs einer UTM-Anwendung können Meldungen erzeugt werden, die den Kommunikationspartner der UTM-Anwendung betreffen. Dabei werden den Kommunikationspartnern nur diejenigen UTM-Meldungen zugestellt, für die in dem UTM-Meldungsmodul das Ziel PARTNER angegeben wurde. Durch Erzeugen eines eigenen Meldungsmoduls kann dieses Meldungsziel für weitere Meldungen hinzu- oder aber auch von Meldungen weggenommen werden (siehe *open*UTM-Handbuch „Meldungen, Test und Diagnose“). Diese UTM-Meldungen können innerhalb oder außerhalb eines Dialogs auftreten. Der Kommunikationspartner muß auf die Meldungen entsprechend reagieren können.

Werden von einer anderen Anwendung UTM-Benutzerkommandos gesendet, so werden diese von *open*UTM nicht als solche interpretiert.

Verbindungsabbau

Den Anstoß zum Verbindungsabbau kann die andere Anwendung oder die UTM-Anwendung geben. Soll der Verbindungsabbau durch die UTM-Anwendung erfolgen, so kann dies entweder durch direkte Eingabe eines Administrationskommandos geschehen, oder aus einem Teilprogrammlauf mittels eines Administrationsaufrufs oder eines KDCS-Aufrufs (SIGN OF). Der Verbindungsabbau erfolgt in diesem Fall unmittelbar am Ende des Teilprogrammlaufs und nicht erst beim Eintreffen der nächsten Dialognachricht.

Mehr über Kopplungen mit Sockets-Anwendungen finden Sie im *open*UTM-Handbuch „Anwendungen generieren und betreiben“.

3.6 Übersicht: Partner, Protokolle,Transaktionssicherung

In folgender Tabelle sind Partner aufgelistet, mit denen eine UTM-Anwendung im Rahmen des Client/Server-Computings zusammenarbeiten kann. Dabei handelt es sich nur um Beispiele, die Liste erhebt also keinen Anspruch auf Vollständigkeit.

Partner	**Protokoll**	**ohne globale Transaktionen**	**mit globalen Transaktionen**
UTM-Anwendungen im selben oder auf einem fernen Rechner	LU6.1, OSI TP, Transportsystem	– x x	x x –
*open*UTM-Client-Anwendungen mit Trägersystem UPIC	UPIC	x	–
*open*UTM-Client-Anwendungen (ab Version V4.0) mit Trägersystem OpenCPIC (für *open*UTM-Client < V4.0 nur ohne globale Transaktionen)	OSI TP	x	x
CICS-/IMS-Anwendungen auf einem IBM-Mainframe	LU6.1	–	x
	*open*UTM: OSI TP IBM: LU6.2	x	x
CICS-/IMS-Anwendungen auf einem OS/2-, AIX- oder RS6000-Rechner	*open*UTM: OSI TP IBM: LU6.2	x	x
Encina-Anwendungen	*open*UTM: OSI TP Encina: LU6.2	x	–
Tuxedo-Anwendungen, Anwendungen in UNISYS-Umgebungen und alle anderen Anwendungen, die das OSI TP-Protokoll unterstützen	OSI TP (Bei Tuxedo auch LU6.2 möglich)	x	x
DCAM-Anwendungen im selben oder einem fernen BS2000-Rechner	Transportsystem	x	–
PDN-Anwendungen	Transportsystem	x	–
CMX-Anwendungen im selben oder einem fernen Rechner	Transportsystem	x	–
andere Transportsystem-Anwendungen, die auf OSI-Transportschichten oder auf RFC1006 über TCP/IP aufsetzen	Transportsystem	x	–
Sockets-Anwendungen	Transportsystem	x	–

4 Message Queuing

Vorzüge und Einsatzmöglichkeiten der in *open*UTM integrierten Message Queuing-Funktionalität wurden Ihnen bereits in Abschnitt 2.4 auf Seite 18ff vorgestellt.

Hier noch einmal kurz die Hauptvorteile:
– zeitliche und räumliche Unabhängigkeit der kommunizierenden Komponenten
– absolute Verläßlichkeit der Nachrichtenübertragung durch den transaktionsgesicherten Deferred Delivery Mechanismus
– Unabhängigkeit von der Qualität und Verfügbarkeit der Verbindungsstrecke

Dieses Kapitel geht genauer darauf ein, wie diese Funktionalität in *open*UTM realisiert ist.

Message Queuing (MQ) ist eine Form der Kommunikation, bei der die Nachrichten (Messages) nicht unmittelbar, sondern über zwischengeschaltete Queues ausgetauscht werden (store and forward). Deshalb werden diese Nachrichten auch **Asynchron-Nachrichten** genannt.

Mit *open*UTM muß beim Senden einer MQ-Nachricht nicht der Name oder die Adresse einer bestimmten Queue angegeben werden, sondern nur der Name des letztendlichen Empfängers. Der zwischengeschaltete Queuing-Mechanismus wird von *open*UTM automatisch zur Verfügung gestellt. Somit wird bereits beim Senden einer asynchronen Nachricht festgelegt, wie die Nachricht weiterverarbeitet werden soll. Weil die Folgeverarbeitung jeweils bereits feststeht, spricht man auch von **Asynchron-Aufträgen**.

*open*UTM unterscheidet - in Abhängigkeit vom jeweiligen Empfänger - zwei verschiedene Typen von Asynchron-Aufträgen:

- Ausgabe-Aufträge (Output Queuing), siehe Abschnitt 4.1 auf Seite 70
- Hintergrund-Aufträge, siehe Abschnitt 4.2 auf Seite 70

  Bei den Hintergrund-Aufträgen kann weiter differenziert werden:
  – lokale Hintergrund-Aufträge (Local Queuing)
    d.h. Hintergrund-Aufträge, die Services der eigenen Anwendung anfordern
  – Hintergrund-Aufträge, die ferne Services anfordern (Remote Queuing)

Die in Abschnitt 4.4 auf Seite 75 beschriebenen Steuerungsmöglichkeiten stehen sowohl für Ausgabe- als auch für Hintergrund-Aufträge zur Verfügung:
– Zeitsteuerung
– Rückmeldungen über Quittungsaufträge
– Administration von Queues

Wie die Steuerungsmöglichkeiten können Sie auch die MQ-Aufrufe der KDCS-Schnittstelle, die in Abschnitt 4.5 auf Seite 76 vorgestellt werden, für beide Typen von Asynchron-Aufträgen einsetzen.

4.1 Ausgabe-Aufträge (Output Queuing)

Ausgabe-Aufträge sind Asynchron-Aufträge, die die Aufgabe haben, eine Nachricht, z.B. ein Dokument, an einen Drucker oder an ein Terminal auszugeben. Ausgabeziel kann aber auch eine andere Anwendung sein, die über die Transportsystem-Schnittstelle angeschlossen wurde (siehe Abschnitt 3.5 auf Seite 64).

Ausgabe-Aufträge setzen sich zusammen aus der Angabe des Ausgabeziels und der asynchronen Nachricht, die ausgegeben werden soll.

Ausgabe-Aufträge werden ohne Mitwirkung des Anwendungsprogrammes automatisch von den UTM-Systemfunktionen abgearbeitet.

Ausgabe-Aufträge können ausgelöst werden durch:
– entsprechenden MQ-Aufruf aus einem Dialog- oder Hintergrund-Auftrag der UTM-Anwendung
– UTM-Meldungen (also ereignisgesteuert)

4.2 Hintergrund-Aufträge

Hintergrund-Aufträge sind Asynchron-Aufträge, die an einen Asynchron-Service der eigenen oder einer fernen Anwendung gerichtet sind. Hintergrund-Aufträge eignen sich besonders für zeitintensive oder zeitunkritische Verarbeitungen, deren Ergebnis keinen direkten Einfluß auf den aktuellen Dialog hat.

Hintergrund-Aufträge setzen sich zusammen aus dem Transaktionscode (TAC) des Teilprogramms, mit dem der Hintergrund-Auftrag beginnt, und ggf. einer asynchronen Nachricht. Dabei bestimmt der Typ des Transaktionscodes, daß der Auftrag asynchron - und nicht als Dialog-Auftrag - verarbeitet wird.

Hintergrund-Aufträge können ausgelöst werden durch:
- Eingabe von einem Terminal
- Aufruf aus einem *open*UTM-Client-Programm mit Trägersystem OpenCPIC
- Eingabe von einer anderen Anwendung, die über die Transportsystemschnittstelle angeschlossen ist
- MQ-Aufruf aus einem Service derselben oder einer fernen UTM-Anwendung
- UTM-Meldungen (also ereignisgesteuert)

4.2.1 Bearbeitung von Hintergrund-Aufträgen

Zur Bearbeitung eines Hintergrund-Auftrags startet die UTM-Anwendung zeitlich entkoppelt den entsprechenden **Asynchron-Service**. Dieser führt alle Schritte durch, die für den gestellten Auftrag notwendig sind.

Ein Asynchron-Service kann in mehrere Verarbeitungsschritte und Transaktionen strukturiert werden. Bei Verwendung der KDCS-Schnittstelle kann er auch mehrere Teilprogramme umfassen.

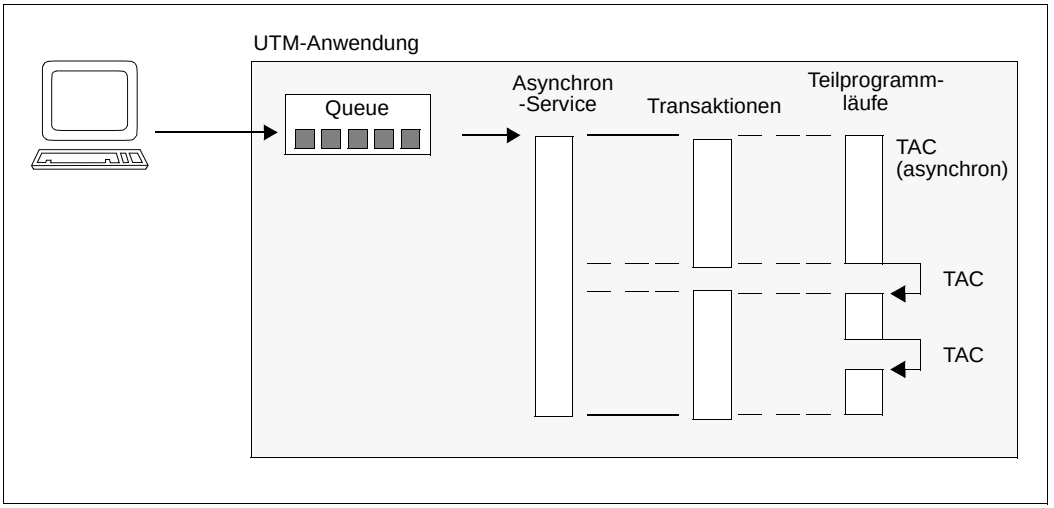


Bild 20: Struktur eines Asynchron-Services, von einem Terminal erteilt

In dem in Bild 20 dargestellten Beispiel wird von einem Terminal aus ein Hintergrund-Auftrag an einen Asynchron-Service der eigenen Anwendung gestellt. Hierzu wird am Terminal der Transaktionscode dieses Services und ggf. eine Nachricht eingegeben. *open*UTM reiht den Auftrag automatisch in die entsprechende Queue ein und startet den Asynchron-Service entkoppelt vom Auftraggeber, sobald die notwendigen Betriebsmittel zur Verfügung stehen.

Ein Asynchron-Service kann weitere Asynchron-Aufträge erzeugen. Dies können Ausgabe-Aufträge oder weitere Hintergrund-Aufträge sein. Bei verteilter Verarbeitung können aus einem Asynchron-Service auch Dialog-Aufträge an Partner-Anwendungen gestellt werden, d.h., der Asynchron-Service startet seinerseits ferne Dialog-Services.

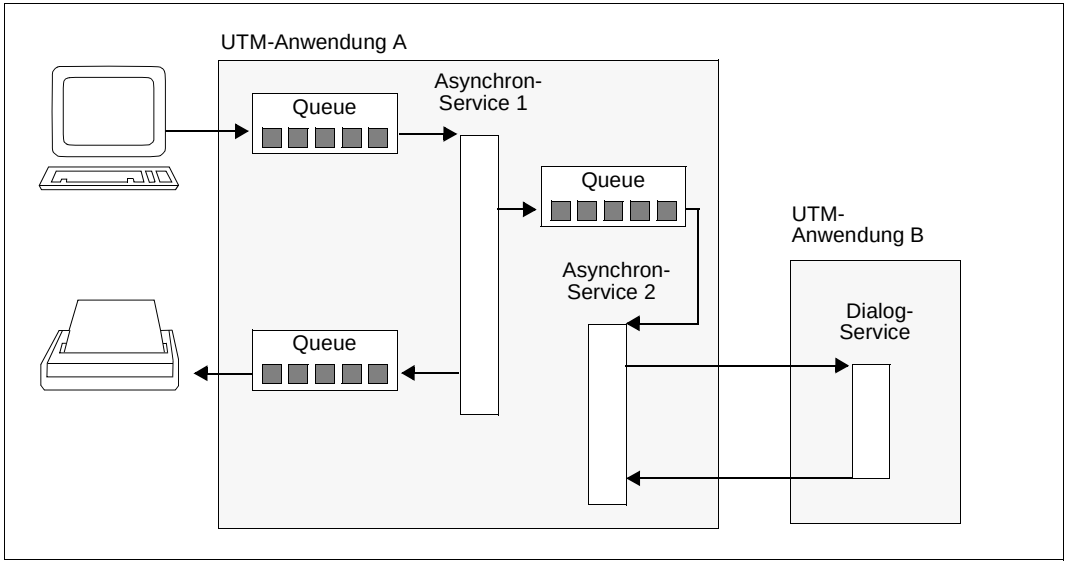


Bild 21: Asynchron-Services, die ihrerseits Aufträge absetzen

In Bild 21 wird - wie in Bild 20 - vom Terminal aus ein Hintergrund-Auftrag an einen lokalen Asynchron-Service gestellt. Innerhalb dieses Services wird ein weiterer Hintergrund-Auftrag abgesetzt sowie ein Ausgabe-Auftrag an einen Drucker. Der Asynchron-Service 2 seinerseits startet einen fernen Dialog-Service in der UTM-Anwendung B. Dabei kann die UTM-Anwendung B auf dem gleichen Rechner liegen wie UTM-Anwendung A oder auch auf einem anderen Rechner. Auch bei komplexeren Strukturen mit vielen unterschiedlichen Queues brauchen Sie sich um den Queuing-Mechanismus nicht zu kümmern: er wird von *open*UTM automatisch zur Verfügung gestellt.

4.2.2 Hintergrund-Aufträge an ferne Services (Remote Queuing)

Hintergrund-Aufträge können nicht nur an Asynchron-Services der lokalen Anwendung gestellt werden, sondern auch an Asynchron-Services ferner Anwendungen.

Hier zeigt sich eine der Hauptstärken der MQ-Funktionalität von *open*UTM, d.h., *open*UTM arbeitet bei Hintergrund-Aufträgen an ferne Anwendungen mit zwei jeweils lokalen Queues: eine Queue liegt dabei in der Sender-Anwendung, die andere Queue beim Empfänger. Durch dieses Deferred Delivery-Prinzip ist das rechnerübergreifende Message Queuing mit *open*UTM vollständig unabhängig davon, ob gerade eine Verbindung möglich ist oder nicht: Falls keine Verbindung aufgebaut werden kann, bleibt der Auftrag solange in der lokalen Sender-Queue, bis eine Verbindung hergestellt ist.

Auch beim Remote Queuing müssen Sie sich nicht um den Queuing-Mechanismus kümmern. Sie geben lediglich an, für welchen Asynchron-Service die MQ-Nachricht bestimmt ist. Ferne Asynchron-Services werden auf die gleiche Weise adressiert wie auch ferne Dialog-Services, was das Design verteilter Anwendungen erleichtert.

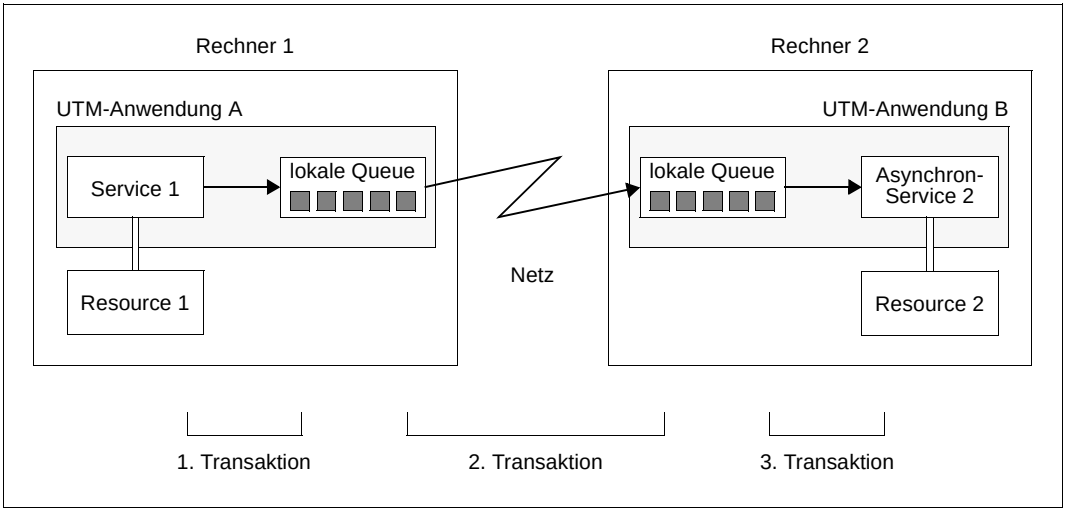


Bild 22: Remote Queuing mit *open*UTM

Bild 22 zeigt, wie ein Hintergrund-Auftrag an eine ferne Anwendung von *open*UTM behandelt wird. Das Bild macht auch deutlich, daß Remote Queuing in vielen Fällen eine sinnvolle Alternative zur verteilten Dialogverarbeitung darstellt: Die Kommunikation zwischen den beiden Services gliedert sich in drei transaktionsgesicherte Schritte. Sobald der Auftrag in die lokale Queue der Anwendung A eingetragen ist, können die benötigten Betriebsmittel in Anwendung A freigegeben und eventuell gesetzte Sperren in Resource 1 wieder aufgehoben werden - auch wenn das Netz gerade nicht verfügbar ist oder die Anwendung B nicht läuft.

Würde eine solche Kommunikation innerhalb einer Dialog-Transaktion abgearbeitet werden, bestünde die Gefahr langanhaltender Sperren und „hängender“ Transaktionen. Durch die Kommunikation über Remote Queuing kann vermieden werden, daß lokale Störungen oder Ausfälle zu übergreifenden, länger andauernden Blockaden führen.

4.3 Priority Scheduling bei Hintergrund-Aufträgen

Hintergrund-Aufträge werden oft dazu verwendet, besonders zeitintensive Aufgaben auszulagern. Dabei muß darauf geachtet werden, daß nicht zuviele Prozesse einer Anwendung gleichzeitig von Hintergrund-Aufträgen belegt werden, da sich dies nachteilig auf die Antwortzeiten für die Dialogverarbeitung auswirken könnte.

*open*UTM bietet deshalb ein zweistufiges Priority-Scheduling-Konzept:

- Es kann festgelegt werden, wieviele Prozesse einer UTM-Anwendung gleichzeitig Hintergrund-Aufträge ausführen dürfen. So ist gewährleistet, daß zu jedem Zeitpunkt eine ausreichende Zahl von Prozessen zur Bearbeitung von Dialog-Aufträgen zur Verfügung bleibt.
- Zusätzlich kann innerhalb der Hintergrund-Aufträge differenziert werden. Dazu sind die Hintergrund-Services zu TAC-Klassen zusammengefaßt. Hier gibt es zwei Alternativen:
  - Prozeßbeschränkung:
    Für jede TAC-Klasse kann die maximale Anzahl an Prozessen angegeben werden, die gleichzeitig für diese Klasse eingesetzt werden darf.
  - Prioritätensteuerung:
    Aufträge aus einer TAC-Klasse mit höherer Priorität werden vorrangig bearbeitet. Dabei kann zwischen absoluter, relativer und gleicher Priorität gewählt werden. Absolut heißt hier, daß erst *alle* Aufträge einer höherprioren TAC-Klasse abgearbeitet sein müssen, bevor die nächste TAC-Klasse zum Zuge kommt. Bei relativer Priorität werden TAC-Klassen höherer Priorität häufiger bearbeitet als TAC-Klassen niedrigerer Priorität.

  Sie können in einer Anwendung immer nur eine der beiden Alternativen verwenden.

Wie Sie die maximale Anzahl der Prozesse festlegen können, die für Hintergrund-Aufträge zur Verfügung stehen sollen, oder wie Sie TAC-Klassen definieren und priorisieren können, erfahren Sie im *open*UTM-Handbuch „Anwendungen generieren und betreiben“ unter den Stichwörtern ASYNTASKS, TACCLASS und TAC-PRIORITIES.

4.4 Steuerungsmöglichkeiten für Asynchron-Aufträge

Für Asynchron-Aufträge bietet *open*UTM verschiedene Steuerungsmöglichkeiten, die in die MQ-Aufrufe der KDCS-Schnittstelle integriert sind.

Quittungsaufträge

Zusammen mit dem eigentlichen Asynchron-Auftrag („Basisauftrag“) können bis zu zwei **Quittungsaufträge** formuliert werden, die an das positive bzw. negative Ergebnis der Auftragsdurchführung gebunden sind. Sie werden ausgeführt, wenn der Basisauftrag abgewikkelt ist. Mit den Quittungsaufträgen hat der Auftraggeber die Möglichkeit, auf ein positives oder negatives Auftragsergebnis zu reagieren. Ein Quittungsauftrag, der nicht zur Wirkung kommt - z.B. der negative Quittungsauftrag bei einem positiven Ergebnis - verfällt. Basisauftrag und Quittungsaufträge werden zusammen auch als **Auftragskomplex** bezeichnet.

Bei einem lokalen Hintergrund-Auftrag wird der Quittungsauftrag, der dem Ergebnis entspricht, gestartet, nachdem der Asynchron-Service beendet ist. Als positives Auftragsergebnis gilt ein ordnungsgemäßer Abschluß des Asynchron-Services. Als negatives Auftragsergebnis gilt ein definitiv fehlerhaftes Ende des Asynchron-Services, wobei dieses in der Regel vom Asynchron-Service herbeigeführt wird: Entweder absichtlich durch einen UTM-Aufruf im Programm (PEND ER/FR) oder durch einen schwerwiegenden Programmfehler.

Bei einem Hintergrund-Auftrag an einen fernen Service (Remote Queuing) wird der positive Quittungsauftrag gestartet, wenn der Auftrag erfolgreich in die Queue des fernen Services übertragen wurde, der negative Quittungsauftrag dann, wenn eine solche Übertragung dauerhaft nicht möglich ist.

Sind einem Ausgabe-Auftrag Quittungsaufträge zugeordnet, wird der Quittungsauftrag, der dem Ergebnis entspricht, gestartet, nachdem die Ausgabe beendet ist. Das Auftragsergebnis ist positiv, wenn aus Sicht von *open*UTM die Ausgabe der Nachricht bzw. des Dokuments abgeschlossen ist. Kann die Nachricht dauerhaft nicht ausgegeben werden, ist das Auftragsergebnis negativ.

Zeitsteuerung

Für einen Asynchron-Auftrag kann angegeben werden, wann er frühestens ausgeführt werden soll. Diese Zeit kann relativ zum Zeitpunkt der Auftragserteilung oder absolut angegeben werden. Man bezeichnet einen solchen Auftrag als **zeitgesteuerten Asynchron-Auftrag.** Bei zeitgesteuerten Asynchron-Aufträgen wird die Bearbeitung von *open*UTM gestartet, nachdem der gewünschte Zeitpunkt erreicht ist und die notwendigen Betriebsmittel zur Durchführung des Auftrags von *open*UTM bereitgestellt werden können.

Administration von Asynchron-Aufträgen

Die Ausführung von Asynchron-Aufträgen kann über die UTM-Administration beeinflußt werden. Ein Asynchron-Auftrag kann z.B. in der Bearbeitungsreihenfolge vorgezogen oder auch storniert werden. Für die Administration von Asynchron-Aufträgen steht der KDCS-Aufruf DADM zur Verfügung (siehe folgender Abschnitt).

4.5 Message Queue-Aufrufe der KDCS-Schnittstelle

*open*UTM stellt mächtige, aber einfache MQ-Aufrufe zur Verfügung. Der Zusatz „free“ in den Aufrufnamen soll widerspiegeln, daß es sich beim Message Queuing um eine vom Sender entkoppelte und von der Verfügbarkeit des Empfängers unabhängige Art der Kommunikation handelt.

- FPUT (**F**ree message **PUT**)
  FPUT-Aufrufe dienen zum Senden von Asynchron-Nachrichten an Ausgabegeräte (Ausgabe-Aufträge) oder an Asynchron-Services (Hintergrund-Aufträge).
  Eine Asynchron-Nachricht kann auch aus mehreren Teilnachrichten bestehen. Für jede Teilnachricht ist dann ein eigener FPUT-Aufruf notwendig.
- DPUT (**D**elayed free message **PUT**)
  Auch mit dem DPUT-Aufruf wird eine Asynchron-Nachricht oder -Teilnachricht an ein Ausgabegerät oder einen Asynchron-Service gesendet. Der DPUT-Aufruf bietet aber gegenüber dem FPUT-Aufruf zusätzlich die Möglichkeit der Zeitsteuerung und der Verwendung von Quittungsaufträgen.
- FGET (**F**ree message **GET**)
  Der Aufruf FGET dient zum Lesen von Asynchron-Nachrichten oder -Teilnachrichten innerhalb eines Asynchron-Services.
- MCOM (**M**essage **COM**plex)
  Der Aufruf MCOM dient dazu, einem Asynchron-Auftrag Quittungsaufträge zuzuordnen.
- DADM (**D**elayed free message **ADM**inistration)
  Mit DADM können Übersichtsinformationen über den gesamten Inhalt einer Queue oder gezielt über einzelne Elemente angefordert werden. Außerdem läßt sich mit DADM die Bearbeitungsreihenfolge steuern: Sie können Aufträge vorziehen, einzelne Aufträge stornieren oder auch die gesamten Aufträge in der Queue löschen.

Das genaue Format der Aufrufe FPUT, DPUT, FGET und MCOM sowie weitere Informationen zu diesen Aufrufen finden Sie im *open*UTM-Handbuch: „Anwendungen programmieren mit KDCS für COBOL, C und C++“. Der Aufruf DADM ist im *open*UTM-Handbuch „Anwendungen administrieren“ beschrieben.

5 Struktur einer UTM-Anwendung

Eine UTM-Anwendung stellt Services zur Verfügung: Sie erledigt Service-Requests (Aufträge), die von Terminal-Benutzern, von Client-Programmen oder auch von anderen Anwendungen an sie gestellt werden.

In einem Rechnersystem können mehrere UTM-Anwendungen existieren. Die Anwendungen sind voneinander unabhängig, lassen sich individuell generieren und administrieren. Wenn gewünscht, können so organisatorisch getrennte Aufgabenkomplexe in getrennten Anwendungen realisiert werden.

Eine UTM-Anwendung besteht aus dem UTM-Anwendungsprogramm, das in einer individuell festlegbaren Anzahl von Prozessen gestartet wird, der KDCFILE, die als „System Memory“ von allen Prozessen genutzt wird, und einem UTM-Cache-Speicher, der die Zugriffe auf die KDCFILE optimiert.

Technisch gesehen ist eine UTM-Anwendung eine Prozeßgruppe, die zur Laufzeit eine logische Server-Einheit bildet.

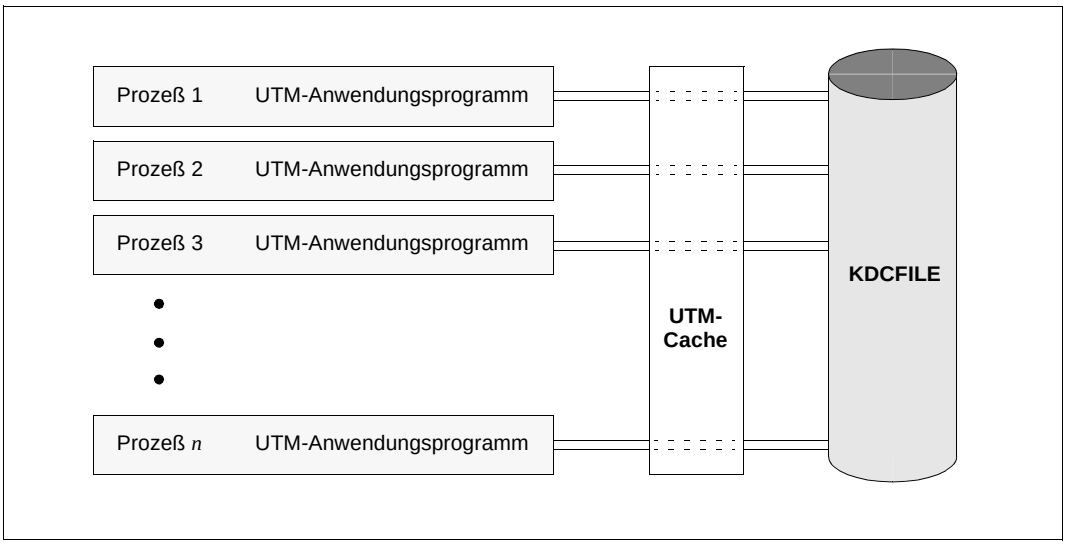


Bild 23: Struktur einer UTM-Anwendung

5.1 UTM-Anwendungsprogramm

Beim Design einer Anwendung programmieren Sie **Service-Routinen**, auch **Teilprogramme** genannt, und legen damit die Services fest, die Ihre Anwendung zur Verfügung stellen soll. Diese Service-Routinen können in einer gängigen Programmiersprache (z.B. C/C++/ COBOL) erstellt werden.

Durch integrierte **UTM-Aufrufe** nutzen die Service-Routinen die UTM-Systemfunktionen, z.B. für Transaktionssicherung, Aufbau von Verbindungen etc. (siehe auch Kapitel „Programmschnittstellen“ auf Seite 83ff).

Den Service-Routinen ordnen Sie **Transaktionscodes** (abgekürzt: TACs) zu - entweder bereits bei der Generierung mit der KDCDEF-Anweisung TAC oder auch im laufenden Betrieb mit dem Aufruf KC_CREATE_OBJECT für den Objekttyp KC_TAC. Diese Transaktionscodes sind frei wählbare Namen, über die die Service-Routinen von Terminal-Benutzern, Clients oder anderen Programmen gestartet werden können.

Damit die Service-Routinen unter dem Management von *open*UTM ablaufen können, binden Sie die übersetzten Service-Routinen, zusammen mit weiteren Modulen (Zuordnungstabellen, Meldungen, benutzte Bibliotheken etc.), zum **UTM-Anwendungsprogramm** (siehe Seite 100).

Als Teil des Anwendungsprogramms entsteht beim Binden die **Main Routine KDCROOT**, die als steuerndes Hauptprogramm fungiert und für die Ablaufkoordination zuständig ist.

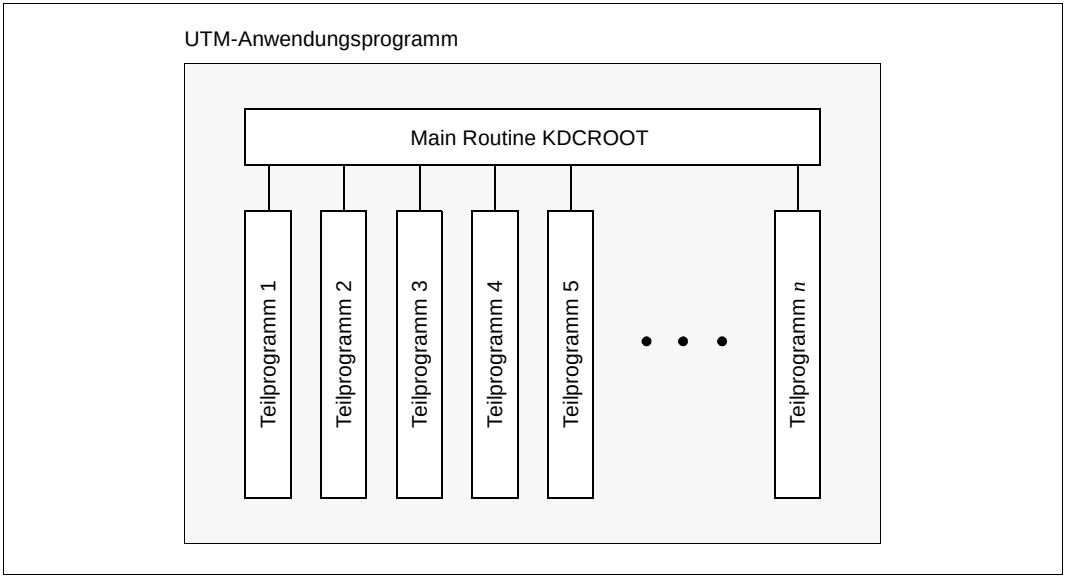


Bild 24: Struktur eines UTM-Anwendungsprogramms

5.2 Prozeßkonzept

Beim Start der Anwendung wird das Anwendungsprogramm in einer von Ihnen festgelegten Anzahl von Prozessen gestartet.

Da in der Regel viele Clients gleichzeitig mit einer UTM-Anwendung arbeiten, wird nicht jedem angeschlossenen Client exklusiv ein Prozeß zugeordnet. Die Last einer großen Zahl von simultanen Requests wird von *open*UTM alternierend auf eine kleine Zahl von Prozessen verteilt. Der System-Overhead - und damit die Antwortzeit - wächst also **nicht** proportional zur Benutzerzahl.

Stehen zu einem bestimmten Zeitpunkt mehr Aufträge zur Bearbeitung an, als Prozesse frei sind, ordnet *open*UTM die Aufträge in eine Warteschlange ein. Umgekehrt werden, falls mehr Prozesse frei sind, als Aufträge anstehen, die freien Prozesse in eine Prozeßwarteschlange eingereiht.

Das Konzept von *open*UTM, mit mehreren gleichwertigen, homogenen Prozessen zu arbeiten, hat eine Reihe weiterer Vorteile:

- Mehrere Aufträge können gleichzeitig bearbeitet werden.
- Wenn mehrere Aufträge den gleichen Service anfordern, kann dieser gleichzeitig in unterschiedlichen Prozessen zur Verfügung gestellt werden
- Auf schwankende Lastsituationen kann eine UTM-Anwendung flexibel reagieren, da per Administration im laufenden Betrieb Prozesse hinzugefügt oder weggenommen werden können.
- Durchsatzhemmende Engpässe bei bestimmten Services werden ausgeschlossen, da die Prozesse gleichwertig (homogen) sind, und jeder Prozeß für jede Aufgabe eingesetzt werden kann.
- Ein schwerwiegender Fehler in einem Anwendungsprogramm kann höchstens zum Abbruch des betroffenen Prozesses führen. Die Wirkung bleibt also lokal, d.h. es ist nur der Auftrag betroffen, den der Prozeß im Augenblick des Ausfalls bearbeitet. Die gesamte Anwendung oder aber andere Anwendungen, die ebenfalls unter Kontrolle von *open*UTM im gleichen Rechner ablaufen, sind davon nicht betroffen. Der abgebrochene Prozeß wird von *open*UTM automatisch durch einen neuen ersetzt.

Optimale Prozeßauslastung durch „Pseudo-Conversations“

Service-Routinen lassen sich so programmieren, daß bei Wartesituationen, z.B. während der „Denkzeit“ eines Terminal-Benutzers, kein Prozeß durch diesen Benutzer belegt wird. Der Prozeß wird automatisch sofort freigegeben und steht für andere Aufträge bereit. Wenn der Terminal-Benutzer dann seine Eingabe-Aktivitäten abgeschlossen hat, übernimmt - ohne daß der Benutzer dies bemerkt - u.U. ein anderer Prozeß die Fortsetzung des Dialogs.

*open*UTM sorgt so für eine optimale Auslastung der Prozesse, was sich positiv auf die Performance auswirkt.
Dieses Dialog-Konzept, auch „pseudo-conversational“ genannt, können Sie nicht nur für den Dialog mit Terminal-Benutzern, sondern auch für die Programm-Programm-Kommunikation einsetzen.

Nähere Informationen zum Thema „Pseudo Conversations“ finden Sie im *open*UTM-Handbuch: „Anwendungen programmieren mit KDCS für COBOL, C und C++“.

Auf einige betriebssystemspezifische Aspekte des Prozeßkonzepts wird in den Kapiteln 11, 12 und 13 eingegangen.

5.3 Die KDCFILE - das „Gedächtnis der Anwendung“

Die KDCFILE besteht aus einer oder mehreren Dateien und enthält die für den Ablauf einer UTM-Anwendung notwendigen Daten. Sie wird mit dem Generierungstool KDCDEF erzeugt (siehe Seite 97). Beim Betrieb einer UTM-Anwendung wird die KDCFILE von allen Prozessen der Anwendung gemeinsam benutzt.

Die KDCFILE ist logisch in die folgenden drei Bereiche untergliedert:

– Verwaltungsdaten
– Pagepool
– Wiederanlaufbereich

Aus Gründen der Sicherheit kann die KDCFILE doppelt auf unterschiedlichen Plattenlaufwerken geführt werden.

Um Engpässe auf Platten zu vermeiden und Zugriffszeiten zu verbessern, ist es möglich, Pagepool und Wiederanlaufbereich auf mehrere Dateien zu verteilen.

Eine detaillierte Beschreibung der KDCFILE finden Sie im *open*UTM-Handbuch „Anwendungen generieren und betreiben“.

Verwaltungsdaten

Der Bereich der Verwaltungsdaten enthält die Konfigurationsinformationen wie z.B. Parameter der UTM-Anwendung, Inhaltsverzeichnisse für alle über Namen ansprechbare Objekte, Verwaltungsinformationen für Pagepool und Wiederanlaufbereich sowie Tabellen für Benutzerkennungen, Clients, Transaktionscodes, Lock-Codes und Funktionstasten.

Beim Start einer UTM-Anwendung werden die Verwaltungsdaten, mit denen die Prozesse der Anwendung arbeiten und über die sie untereinander Informationen austauschen, in einen gemeinsam benutzbaren Hauptspeicherbereich gebracht. Änderungen der Verwaltungsdaten schreibt *open*UTM während des Laufs der Anwendung und bei Beendigung der Anwendung auf KDCFILE zurück, um beim nächsten Lauf der Anwendung auf dem Stand des vorherigen Laufs aufsetzen zu können und um bei abnormaler Beendigung der UTM-Anwendung den automatischen Wiederanlauf zu ermöglichen.

Pagepool

Im Pagepool werden alle während des Laufes der UTM-Anwendung anfallenden Benutzerdaten gespeichert. Das sind z.B.:

- verschiedene Sekundär-Speicherbereiche
- Information für den Bildschirmwiederanlauf
- Kommunikationsbereiche
- Queues mit Ausgabe-Aufträgen (auch zeitgesteuerten) und Hintergrund-Aufträgen
- zwischengespeicherte Sätze der Benutzer-Protokolldatei

Die laufende UTM-Anwendung greift auf den Pagepool über den UTM-Cache zu. Die Größe des Pagepools (Anzahl der UTM-Seiten) legen Sie bei der Generierung fest.

Wiederanlaufbereich

UTM-Aufrufe in einem Teilprogramm haben Änderungen in den Verwaltungsdaten und den Benutzerdaten zur Folge. *open*UTM sammelt Information über alle Änderungen, die innerhalb einer Transaktion anfallen.

Bei Transaktionsende bildet *open*UTM bei einer UTM-S-Anwendung (siehe Seite 135) aus diesen Informationen einen Datensatz mit Wiederanlauf-Informationen und schreibt ihn in den Wiederanlaufbereich der KDCFILE. Der Datensatz beschreibt, welche Änderungen in den Verwaltungsdaten als Folge der Transaktion vorzunehmen sind. Die Größe des Wiederanlaufbereichs bestimmt, in welchen Zeitabständen Änderungen der Verwaltungsdaten in den Bereich Verwaltungsdaten auf der KDCFILE übernommen werden müssen. Über die Häufigkeit dieser Vorgänge kann man sich mit einem Administrationskommando informieren.

In einer UTM-F-Anwendung (siehe Seite 135) werden nur in solchen Transaktionen Datensätze für den Wiederanlauf geschrieben, in denen Paßwörter geändert oder per dynamischer Konfigurierung Änderungen an den Verwaltungsdaten vorgenommen wurden.

5.4 UTM-Cache-Speicher

Der UTM-eigene Cache-Speicher ist ein Bereich im virtuellen Speicher, der generierungsabhängig in Einheiten von 2KB bzw. 4KB verwaltet wird. *open*UTM verwendet ihn als anwendungsglobalen Pufferbereich für Zugriffe auf den Pagepool, d.h. die Prozesse einer UTM-Anwendung wickeln alle Zugriffe auf Pagepool-Daten über diesen Speicherbereich ab. Ein Cache-Speicher von ausreichender Größe ermöglicht *open*UTM, die Schreib- und Lesezugriffe auf den Pagepool zu optimieren. Der Hauptvorteil liegt darin, daß Lesezugriffe eingespart werden können, wenn Seiten mit Pagepool-Daten, die von einer vorhergehenden Transaktion geschrieben wurden (eventuell von einem anderen Prozeß), noch im Cache-Speicher liegen.

Bei der UTM-Generierung wird festgelegt:

– die Größe des Cache-Speichers

– wieviel Prozent der Seiten des Cache-Speichers auf Pagepool geschrieben werden sollen, wenn für neue Daten Platz geschaffen werden muß. Im laufenden Betrieb ist diese Größe per Administration veränderbar.

Einen Einblick in die Nutzung des Cache-Speichers in einer laufenden Anwendung erlaubt Ihnen der Wert CACHE HIT RATE, den Sie z.B. mit dem Administrationskommando KDCINF STAT abfragen können. Er zeigt an, wie hoch die Cache-Trefferrate ist, d.h. in wievielen Fällen *open*UTM zu lesende Pagepool-Seiten noch im Cache-Speicher vorfindet.

Die optimale Einstellung dieser Parameter können Sie durch Vergleiche und Performance-Messungen ermitteln; eine allgemein gültige Universalformel gibt es nicht. Einige Faustregeln und Anhaltspunkte finden Sie jedoch im *open*UTM-Handbuch „Anwendungen generieren und betreiben“.

6 Programmschnittstellen

Mit den Programmschnittstellen von *open*UTM lassen sich die verschiedensten Formen der Kommunikation für die unterschiedlichsten Einsatzszenarien schnell und einfach programmieren.

*open*UTM unterstützt die universale Programmschnittstelle KDCS (nationaler Standard) und die von X/Open standardisierten Schnittstellen CPI-C, XATMI und TX.

Die Aufrufe aller Programmschnittstellen können Sie in ganz normale C-, C++- oder auch COBOL-Programme integrieren. In BS2000/OSD steht die KDCS-Schnittstelle zusätzlich für Assembler, Fortran, PASCAL-XT und PL/I zur Verfügung. Auch wenn die Service-Routinen in verschiedenen Sprachen codiert sind, können sie beliebig kombiniert werden.

6.1 Die universale Programmschnittstelle KDCS

Die Schnittstelle KDCS (Kompatibles Datenkommunikationssystem) wurde als herstellerneutrale Schnittstelle für transaktionsorientierte Anwendungen definiert und genormt (DIN 66 265).

KDCS zeichnet sich durch folgende Charakteristika aus:

– umfassende Palette von Funktionsaufrufen für universalen Einsatz
(z.B. auch für Pseudo Conversations, Message Queuing oder die unmittelbare Kommunikation mit Terminals)

– KDCS-spezifische Speicherbereiche für einfache und sichere Programmierung

– Event-Funktionen für Ereignissteuerung

KDCS steht für die Programmiersprachen C, C++ und COBOL zur Verfügung, in BS2000/OSD zusätzlich für Assembler, Fortran, PL/I und Pascal-XT.

Ausführliche Informationen über die KDCS-Programmschnittstelle finden Sie im *open*UTM-Handbuch „Anwendungen programmieren mit KDCS für COBOL, C und C++“. Für die in BS2000/OSD zusätzlich unterstützten Programmiersprachen gibt es auf der *open*UTM-Doku-CD jeweils einen eigenen Ergänzungsband.

6.1.1 KDCS-Aufrufe

Die Funktionsaufrufe von *open*UTM können in folgende Funktionsgruppen zusammengefaßt werden:

- Programm- und Transaktionsverwaltung
- Nachrichtenkommunikation im Dialog
- Nachrichtenkommunikation über Message Queuing
- Verwalten von Message Queues und Druckern
- Speicherverwaltung
- Informationsdienste
- Protokollierung
- An-/Abmelden und Paßwortänderung

Programmverwaltung und Transaktionssteuerung:

Aufruf	**Funktion**
INIT	Anmelden eines Programms bei UTM
PEND	Beenden des Programms
PGWT	Wartepunkt in einem Teilprogrammlauf setzen
RSET	Rücksetzen angeforderter Änderungen und Operationen

Nachrichtenkommunikation im Dialog:

Aufruf	**Funktion**
APRO	Adressieren eines Auftragnehmervorgangs (für verteilte Verarbeitung)
CTRL	Steuern eines OSI-TP-Dialogs (für verteilte Verarbeitung)
MGET	Empfangen einer Dialognachricht
MPUT	Senden einer Dialognachricht

Nachrichtenkommunikation über Message Queuing:

Aufruf	**Funktion**
APRO	Adressieren eines Auftragnehmervorgangs (bei verteilter Verarbeitung)
DPUT	Erzeugen von zeitgesteuerten Asynchronaufträgen und von Quittungsaufträgen
FGET	Empfangen von Asynchronnachrichten
FPUT	Erzeugen von Asynchronaufträgen
MCOM	Definieren eines Auftragskomplexes

Verwalten von Message Queues und Druckern:

Aufruf	**Funktion**
DADM	Administration von Asynchronaufträgen
PADM	Steuern von Druckern und Druckausgaben

Speicherverwaltung:

Aufruf	**Funktion**
GTDA	Lesen aus einem TLS
PTDA	Schreiben in einen TLS
SGET	Lesen aus einem Sekundärspeicherbereich
SPUT	Schreiben in einen Sekundärspeicherbereich
SREL	Freigeben eines Sekundärspeicherbereichs
UNLK	Entsperren eines TLS, ULS oder GSSB

Informationsdienste:

Aufruf	**Funktion**
INFO	Informationen abrufen

Protokollierung:

Aufruf	**Funktion**
LPUT	Schreiben in die Protokolldatei

An-/Abmelden und Paßwortänderung:

Aufruf	**Funktion**
SIGN	An- und Abmelden, Paßwortänderung

Returncodes der KDCS-Aufrufe

Nach jedem KDCS-Aufruf gibt *open*UTM einen genormten KDCS-Returncode und gegebenenfalls zusätzlich einen UTM-spezifischen Returncode zurück.

Diese Returncodes geben u.a. Aufschluß darüber, ob die gewünschte Operation erfolgreich war oder nicht. Es sind folgende Kategorien zu unterscheiden:

– Kein Fehler.

– Leichte Fehler, die vom Programm behebbar sind.

– Codes für Funktionstasten, die während des Vorgangs zur Übermittlung einer Eingabenachricht gedrückt wurden.

– Fehler, die es noch ermöglichen, den Dialogschritt oder Vorgang sinnvoll zu beenden.

– Schwere Fehler, bei denen *open*UTM die Transaktion zurücksetzt und den Vorgang mit Speicherabzug beendet. Der Returncode kann in diesem Fall dem Speicherabzug entnommen werden. Wenn möglich, schickt *open*UTM dem Kommunikationspartner eine Fehlermeldung.

Eine Übersicht über alle Returncodes finden Sie im *open*UTM-Handbuch „Meldungen, Test und Diagnose“.

6.1.2 UTM-Speicherbereiche

*open*UTM bietet Teilprogrammen zum Schreiben und Lesen von Benutzerdaten verschiedene Speicherbereiche. Durch diese Speicherbereiche wird eine klare Trennung von Programm- und Datenbereichen gewährleistet und die Reentrant-Fähigkeit der Programme sichergestellt. Außerdem ermöglichen die Bereiche einen hochperformanten und transaktionsgesicherten Austausch von Informationen zwischen Programmen und sorgen für effektiven Arbeitsspeichereinsatz. Einige Speicherbereiche sind speziell für statistische Zwecke und für die Protokollierung konzipiert.

Primärspeicherbereiche

Dies sind Speicherbereiche, die den Teilprogrammen im Arbeitsspeicher zur Verfügung stehen:

- Kommunikationsbereich (KB)

  *open*UTM legt den KB an, wenn ein neuer Service gestartet wird. Sein Inhalt wird dem jeweils aktuell ausgeführten Teilprogramm übergeben. Im KB stellt *open*UTM den Teilprogrammen aktuelle Informationen zur Verfügung. Der KB dient außerdem zum Informationsaustausch zwischen den Teilprogrammen eines Services. Er kann in seiner Größe den zu übergebenden Daten angepaßt werden. Der KB unterliegt der Transaktionssicherung und bleibt solange erhalten, bis der Service abgeschlossen ist.

- Standardprimärer Arbeitsbereich (SPAB)

  *open*UTM ordnet jedem Teilprogramm standardmäßig einen SPAB zu. Er steht dem Teilprogramm vom Programmstart bis zum Programmende (PEND-Aufruf) zur Verfügung. In diesem Bereich können deshalb keine Daten über das Ende eines Teilprogrammlaufs hinaus aufbewahrt oder weitergegeben werden. Der SPAB kann für den Parameterbereich genutzt werden, in dem das Teilprogramm bei KDCS-Aufrufen die Parameter bereitstellt. Außerdem läßt sich der SPAB z.B. zur Pufferung von Nachrichten verwenden. Da der SPAB nur als teilprogramm-spezifischer Arbeitsbereich dient, ist er nicht in die Transaktionssicherung einbezogen und wird von einem RSET-Aufruf nicht zurückgesetzt. Ein Vorteil des SPABs gegenüber anderen Programmspeichern (z.B. Stacks) ist die Tatsache, daß der SPAB im UTM-Dump ausgegeben wird.

- Weitere Datenbereiche (AREAs)

  Ein Teilprogramm kann weitere Bereiche zur Speicherung von Daten verwenden. Diese Bereiche werden bei der Generierung mit der KDCDEF-Anweisung AREA vereinbart und müssen vom Anwender selbst verwaltet werden, sie unterliegen nicht der Transaktionssicherung. Die Struktur dieser Bereiche ist von *open*UTM nicht vorgegeben, sondern frei definierbar.

Sekundärspeicherbereiche

Diese Speicherbereiche werden von *open*UTM auf Hintergrund-Speicher realisiert und unterliegen der Transaktionssicherung. Sie werden mit speziellen KDCS-Aufrufen geschrieben und gelesen:

- Lokaler Sekundärspeicherbereich (LSSB)

  Der LSSB ist ein service-spezifischer Hintergrundspeicher, der zur Weitergabe von Daten zwischen Teilprogrammen innerhalb eines Service dient. Während *open*UTM den Kommunikationsbereich jedem Teilprogramm automatisch zur Verfügung stellt und danach sichert, wird auf den LSSB nur bei Bedarf zugegriffen. Er eignet sich deshalb besonders für Daten, auf die nur lesend zugegriffen wird, oder wenn zwischen Schreiben und Lesen der Daten mehrere Teilprogramme liegen, in denen die Daten nicht benötigt werden. Der LSSB bleibt solange erhalten, bis er explizit freigegeben wird oder der Service abgeschlossen ist.

- Globaler Sekundärspeicherbereich (GSSB)

  Der GSSB ist ein Hintergrundspeicher, der die beliebige Übergabe von Daten zwischen Services einer UTM-Anwendung ermöglicht. Er bleibt, falls er nicht explizit freigegeben wird, auch über das Anwendungsende hinaus erhalten.

- Terminalspezifischer Langzeitspeicher (TLS)

  Der TLS ist einem Anschlußpunkt (LTERM-, LPAP oder OSI-LPAP-Partner) zugeordnet und dient zur Aufnahme von Informationen, die unabhängig von der Dauer eines Services und der Betriebszeit der Anwendung zur Verfügung stehen sollen. Er kann z.B. verwendet werden, um eine Statistik für einen LTERM-Partner zu erstellen.

- Userspezifischer Langzeitspeicher (ULS)

  Jeder Benutzerkennung kann bei der Konfigurierung ein ULS zugewiesen werden. Er dient zur Aufnahme von Informationen, die unabhängig von der Lebensdauer der Vorgänge und der Betriebszeit der Anwendung zur Verfügung stehen sollen. Er kann z.B. verwendet werden, um Statistiken für jede Benutzerkennung zu führen.

  Ein ULS wird auch einer Session (LU6.1) und einer Association (OSI TP) zugeordnet.

Benutzer-Protokolldatei (User-Logging-Datei USLOG)

Die Benutzer-Protokolldatei ist eine von *open*UTM verwaltete Protokolldatei, in die mit dem KDCS-Aufruf LPUT benutzerspezifische Informationen geschrieben werden können. Sie ist als Datei-Generationsgruppe realisiert (siehe *open*UTM-Handbuch „Anwendungen generieren und betreiben“). In der Benutzer-Protokolldatei lassen sich beispielsweise versuchte Zugriffsschutzverletzungen protokollieren.

Die Benutzer-Protokolldatei unterliegt der Transaktionssicherung und ist daher nicht für eine allgemeine Protokollfunktion geeignet, da beim Rücksetzen einer Transaktion die LPUT-Informationen verworfen werden.

Übersicht: UTM-Speicherbereiche

Kurz-name	Bereichstyp	Lebensdauer	Funktion	Transaktions-sicherung
KB	Kommunikations-bereich	Start des Service bis Ende des Service	Zugriff auf aktuelle, von *open*UTM bereitgestellte Information; Datenaustausch zwischen Teilprogrammen eines Service	ja
SPAB	Standard Primärer Arbeitsbereich	Start des Teilprogramms bis Ende des Teilprogramms	Parameterübergabe bei KDCS-Aufrufen; Nachrichtenpuffer	nein
AREA	Per Generierung vereinbarter weiterer Speicherbereich	Dauer der Anwendung	Aufnahme von anwendungsglobalen Daten, die vorzugsweise nur gelesen werden	nein
LSSB	Lokaler Sekundärer Arbeitsbereich	vom ersten Schreibaufruf bis zur expliziten Freigabe oder bis Service-Ende	Datenaustausch zwischen Teilprogrammen eines Service	ja
GSSB	Globaler Sekundärer Arbeitsbereich	vom ersten Schreibaufruf bis zur expliziten Freigabe oder bis zum Löschen der Anwendungsinformation	Austausch von Daten über Service-Grenzen hinweg	ja
ULS	Userspezifischer Langzeitspeicher	Generierung bis Änderung der Generierung	z.B. Statistiken spezifisch für bestimmte Benutzerkennungen, Sessions oder Associations	ja
TLS	Terminalspezifischer Langzeitspeicher	Generierung bis Änderung der Generierung	Statistiken spezifisch für bestimmte Anschlußpunkte (LTERMs, LPAPs, OSI-LPAPs)	ja
USLOG	Benutzer-Protokolldatei (User-Logging)	individuell festlegbar	Protokollierung	ja

6.1.3 Event-Funktionen

Um auf bestimmte Ereignisse im Ablauf programmiert reagieren zu können, bietet *open*UTM die Möglichkeit, Event-Funktionen zu verwenden. Im Gegensatz zu „normalen“ Services, die durch Angabe eines Transaktionscodes aufgerufen werden, startet *open*UTM diese Services automatisch beim Auftreten bestimmter Ereignisse.

Es gibt zwei unterschiedliche Arten von Event-Funktionen:
– Event-Services, die **KDCS-Aufrufe** enthalten **müssen**
– Event-Exits, welche **keine KDCS-Aufrufe** enthalten **dürfen**

Event-Services

BADTACS ist ein Dialog-Service. Er wird von *open*UTM gestartet, wenn ein Terminal-Benutzer oder ein Transportsystem-Client einen ungültigen Transaktionscode angibt.

Über BADTACS kann z.B. eine HELP-Ausgabe bzw. Benutzerführung realisiert werden, die den betreffenden Benutzer informiert, welche Transaktionscodes ihm zum Start von Services zur Verfügung stehen.

MSGTAC ist ein Asynchron-Service. Er wird von *open*UTM gestartet, wenn in der Anwendung UTM-Meldungen ausgelöst werden, denen Sie in der Meldungsdatei das Meldungsziel MSGTAC zugeordnet haben.

Der Event-Service MSGTAC läßt sich für die Automatisierung der Administration verwenden (siehe auch Seite 115). So kann z.B. bei Mißbrauch automatisch das betreffende Terminal gesperrt werden oder ein Transaktionscode, dessen Aufruf wiederholt zu Fehlern führte.

SIGNON ist ein Dialog-Service. SIGNON wird von *open*UTM gestartet, wenn sich ein Terminal-Benutzer, eine Transportsystem-Anwendung oder ein UPIC-Client an die UTM-Anwendung anschließt.

Ist ein SIGNON-Service generiert, dann wird er für Terminals und Transportsystem-Anwendungen immer durchlaufen. Für UPIC-Clients wird er hingegen nur dann aktiviert, wenn dies explizit so generiert wurde.

Mit dem Dialog-Service SIGNON können Sie den Anmeldungsdialog für Ihre Anwendung individuell gestalten. Mit dem SIGNON-Service lassen sich beispielsweise zusätzlich zu den Berechtigungsprüfungen von *open*UTM eigene Berechtigungsprüfungen durchführen (siehe auch Seite 122), oder man kann im Anmeldevorgang einer Transportsystem-Anwendungen explizit eine UTM-Benutzerkennung zuweisen.

Event-Exits

START Der Event-Exit START wird von *open*UTM beim Start und beim Neuladen des Anwendungsprogramms in jedem Prozeß aufgerufen.

Er kann z.B. verwendet werden, um eigene Dateien zu öffnen.

SHUT Der Event-Exit SHUT wird von *open*UTM bei Beendigung des Anwendungsprogramms in jedem Prozeß aufgerufen.

Er kann z.B. verwendet werden, um eigene Dateien zu schließen.

VORGANG In der Konfiguration einer UTM-Anwendung können Sie einzelnen Services jeweils einen Event-Exit VORGANG zuordnen. Beim Start und beim Beenden des Services wird von *open*UTM dann der entsprechende VORGANG-Exit aufgerufen - auch bei fehlerhaftem Beenden und beim Wiederanlauf. In diesem Event-Exit kann auf den KB-Kopf und den SPAB zugegriffen werden.

Der VORGANG-Exit ermöglicht service-spezifische Aktionen, z.B. das Öffnen und Schließen spezieller Ressourcen für bestimmte Services.

INPUT Der Event-Exit INPUT wird bei jeder Eingabe von einem Terminal aufgerufen, außer bei Eingaben im Event-Service SIGNON.

Mit dem INPUT-Exit können Sie bestimmen, welche Aktionen eine Eingabe vom Terminal jeweils auslösen soll, z.B. Starten eines Services oder Aufruf eines Benutzerkommandos. Er ermöglicht zusätzliche Freiheiten bei der Gestaltung der Benutzeroberfläche.

In BS2000/OSD steht zusätzlich der Event-Exit FORMAT zur Verfügung, mit dem sich eine selbstprogrammierte Formatierung realisieren läßt.

6.2 Die X/Open-Schnittstelle CPI-C

CPI-C (**C**ommon **P**rogramming **I**nterface for **C**ommunication) ist eine von X/Open und dem CIW (**C**PI-C **I**mplementor's **W**orkshop) standardisierte Programmschnittstelle für die Programm-Programm-Kommunikation über Rechnergrenzen hinweg.

*open*UTM stellt die CPI-C-Programmschnittstelle für die Programmiersprachen COBOL, C und C++ zur Verfügung. Unter *open*UTM kann CPI-C nicht nur über das X/Open-Protokoll OSI TP, sondern auch über die Protokolle LU6.1 und UPIC kommunizieren.

Da CPI-C nur die Programm-Programm-Kommunikation unterstützt, bietet CPI-C keine Funktionen zur Kommunikation mit Terminals. CPI-C-Services in *open*UTM können aus diesem Grund nicht direkt von einem Terminal (durch Eingabe eines Transaktionscodes) gestartet werden. CPI-C-Services einer UTM-Anwendung können nur durch Service-Anforderungen von anderen Programmen gestartet werden, z.B.

– von *open*UTM-Client-Programmen
– von anderen UTM-Anwendungen (Server-Server-Kommunikation)
– von Fremd-Anwendungen (z.B. von CICS-Anwendungen)

Die folgenden Tabellen geben eine Übersicht über die CPI-C-Aufrufe, die in *open*UTM zur Verfügung stehen. Eine umfassende Beschreibung der einzelnen Aufrufe finden Sie in der X/Open CAE Specification zu CPI-C (Version 2) vom Oktober 1994 (auf der *open*UTM-Doku-CD enthalten).

Alle UTM-spezifischen Details sind im *open*UTM-Handbuch „Anwendungen erstellen mit X/Open-Schnittstellen“ beschrieben.

Übersicht: CPI-C-Aufrufe in *open*UTM

Die Aufrufnamen sind für C/C++ und COBOL identisch.

Aufrufe des Starter-Sets:

Funktion	Aufruf	Beschreibung
Accept_Conversation	CMACCP	Incoming-Conversation annehmen
Allocate	CMALLC	Outgoing-Conversation aufbauen
Deallocate	CMDEAL	Conversation (normal) beenden
Initialize_Conversation	CMINIT	Outgoing-Conversation etablieren, Conversation-Charakteristika initialisieren
Receive	CMRCV	Daten empfangen
Send_Data	CMSEND	Daten senden

Aufrufe für Fehler- und Quittungsbehandlung:

Funktion	Aufruf	Beschreibung
Cancel_Conversation	CMCANC	eine Conversation abbrechen
Confirmed	CMCFMD	eine positive Quittung an den Partner senden
Send_Error	CMSERR	Fehlernachricht, negative Quittung senden

Aufrufe für die Konvertierung:

Funktion	Aufruf	Beschreibung
Convert_Incoming	CMCNVI	empfangene Daten von EBCDIC in den am lokalen System verwendeten Zeichensatz umsetzen
Convert_Outgoing	CMCNVO	zu sendende Daten von dem im lokalen System verwendeten Zeichensatz nach EBCDIC umsetzen

6.3 Die X/Open-Schnittstelle XATMI

XATMI ist eine von X/Open standardisierte Programmschnittstelle für die Programm-Programm-Kommunikation über Rechnergrenzen hinweg.

*open*UTM stellt die XATMI-Programmschnittstelle für die Programmiersprachen COBOL, C und C++ zur Verfügung. Unter *open*UTM kann XATMI nicht nur über das X/Open-Protokoll OSI TP, sondern auch über die Protokolle LU6.1 und UPIC kommunizieren.

XATMI-Services können nur Partnern gestartet werden, die ebenfalls die XATMI-Schnittstelle benutzen.

XATMI unterscheidet drei Kommunikationsmodelle:

– Synchrones Request-Response Modell:
  Der Client ist nach dem Senden der Service-Anforderung bis zum Eintreffen der Antwort blockiert.
– Asynchrones Request-Response Modell:
  Der Client ist nach dem Senden der Service-Anforderung nicht blockiert.
– Conversational Modell:
  Client und Server können beliebig Daten austauschen.

Die folgenden Tabellen geben eine Übersicht über alle XATMI-Aufrufe, die in *open*UTM zur Verfügung stehen. Eine umfassende Beschreibung der einzelnen Aufrufe finden Sie in der X/Open CAE Specification zu XATMI vom November 1995 (auf der *open*UTM-Doku-CD enthalten).

Alle UTM-spezifischen Details sind im *open*UTM-Handbuch „Anwendungen erstellen mit X/Open-Schnittstellen“ beschrieben.

Übersicht: XATMI-Aufrufe in *open*UTM

Aufrufe für das Request/Response-Modell:

C/C++-Aufruf	COBOL-Aufruf	Beschreibung
tpcall	TPCALL	Service-Anforderung im synchronen Request/Response-Modell
tpacall	TPACALL	Service-Anforderung im asynchronen Request/Response-Modell
tpgetrply	TPGETRPLY	Response im asynchronen Request/Response-Modell anfordern
tpcancel	TPCANCEL	löscht eine asynchrone Service-Anforderung, bevor die angeforderte Response eingetroffen ist

Aufrufe für das Conversational-Modell:

C/C++-Aufruf	COBOL-Aufruf	Beschreibung
tpconnect	TPCONNECT	baut eine Verbindung für den Nachrichtenaustausch auf
tpsend	TPSEND	sendet eine Nachricht
tprecv	TPRECV	empfängt eine Nachricht
tpdiscon	TPDISCON	baut eine Verbindung für den Nachrichtenaustausch ab

Aufrufe für typisierte Puffer:

C/C++-Aufruf	COBOL-Aufruf	Beschreibung
tpalloc	--	reserviert Speicherplatz für einen typisierten Puffer
tprealloc	--	verändert die Größe eines typisierten Puffers
tpfree	--	gibt einen typisierten Puffer frei
tptypes	--	fragt Typ eines typisierten Puffer an

Allgemeine Aufrufe für Service-Routinen:

C/C++-Aufruf	COBOL-Aufruf	Beschreibung
tpservice	TPSVCSTART	stellt ein Template für Service-Routinen zur Verfügung
tpreturn	TPRETURN	beendet eine Service-Routine
tpadvertise tpunadvertise	TPADVERTISE TPUNADVERTISE	wird nur syntaktisch unterstützt in *open*UTM (gibt den Namen einer Service-Routine bekannt)

6.4 Die X/Open-Schnittstelle TX

TX (Transaction Demarcation) ist eine von X/Open standardisierte Programmschnittstelle zur Festlegung von Transaktionen über Rechnergrenzen hinweg.

*open*UTM stellt die TX-Programmschnittstelle für die Programmiersprachen COBOL, C und C++ zur Verfügung.

TX-Aufrufe sind unter *open*UTM nur in CPI-C-Services sinnvoll, XATMI-Services werden implizit in globale Transaktionen eingeschlossen. Mit dem XATMI-Aufruf *tpreturn()* wird gesteuert, ob eine Transaktion erfolgreich beendet oder zurückgesetzt wird.

Transaktionen können mit TX entweder „Chained“ oder „Unchained“ ausgeführt werden.
Im Modus „Chained“ muß nur die erste Transaktion explizit begonnen werden:
das Transaktionsende markiert implizit den Beginn der nächsten Transaktion.
Im Modus „Unchained“ muß der Beginn jeder Transaktion explizit markiert werden.

*open*UTM arbeitet immer im Modus „Chained“. Beim Start eines Service unter *open*UTM wird automatisch eine Transaktion begonnen, daher entfällt auch die Notwendigkeit, die erste Transaktion zu markieren.

Das Trägersystem OpenCPIC ermöglicht es *open*UTM-Client-Anwendungen, Transaktionen über TX zu steuern.

Übersicht: TX-Aufrufe in *open*UTM

C/C++-Aufruf	**COBOL-Aufruf**	**Beschreibung**
tx_commit	TXCOMMIT	globale Transaktion erfolgreich beenden
tx_rollback	TXROLLBACK	globale Transaktion zurücksetzen
tx_info	TXINFORM	globale Transaktions-Informationen abfragen
tx_set_commit_return	TXSETCOMMITRET	Charakteristik *commit_return* setzen
tx_set_transaction_control	TXSETTRANCTL	Charakteristik *transaction_control* setzen
tx_set_transaction_timeout	TXSETTIMEOUT	Charakteristik *transaction_timeout* setzen
tx_open	TXOPEN	Liefert unter *open*UTM immer TX_OK zurück (Set von Resource Managern öffnen)
(Die Aufrufe *tx_begin()* (TXBEGIN) und *tx_close()* (TXCLOSE) liefern unter openUTM immer TX_PROTOCOL_ERROR zurück).		

Eine umfassende Beschreibung der einzelnen Aufrufe finden Sie in der X/Open CAE Specification zu TX vom April 1995 (auf der *open*UTM-Doku-CD enthalten).

Alle UTM-spezifischen Details sind im *open*UTM-Handbuch „Anwendungen erstellen mit X/Open-Schnittstellen“ beschrieben.

7 Generieren von UTM-Anwendungen

Damit Sie eine UTM-Anwendung einsetzen können, müssen Sie die Konfiguration definieren und das Anwendungsprogramm erzeugen. Die hierzu notwendigen Schritte werden unter dem Begriff „Generierung“ zusammengefaßt.

Die Konfiguration können Sie entweder lokal erstellen (siehe unten) oder von einem PC aus mit Hilfe der *open*UTM-Komponente WinAdmin (siehe Seite 103).

Bild 25 auf Seite 101 gibt eine Übersicht über die Arbeitsschritte bei der Generierung. Eine genaue Beschreibung finden Sie im *open*UTM-Handbuch „Anwendungen generieren und betreiben“.

7.1 Definieren der Konfiguration

Das Anwendungsprogramm benötigt zum Ablauf eine Reihe von Informationen, z.B. über:

- Anwendungsparameter (Maximalwerte, Timer etc.)
- Benutzerkennungen
- Zugangs- und Zugriffsschutz
- Eigenschaften von Clients und Partner-Servern
- Eigenschaften von Transaktionscodes und Teilprogrammen
- Reservierungen für die dynamische Konfigurierung

Die Summe dieser Eigenschaften heißt Konfiguration. Die Konfigurationsinformationen werden in der KDCFILE gespeichert, die aus einer oder auch aus mehreren Dateien besteht (siehe Seite 80). Die KDCFILE wird mit dem Generierungstool KDCDEF erzeugt.

Neben der KDCFILE erzeugt KDCDEF den Source-Text für die ROOT-Tabellen. Diese ROOT-Tabellen enthalten Zuordnungsinformationen, die beim Einsatz der Anwendung intern benötigt werden.

Die KDCFILE und die ROOT-Tabellen-Sourcen können wahlweise in einem KDCDEF-Lauf oder auch jeweils separat in unterschiedlichen KDCDEF-Läufen erzeugt werden. Wenn zwei KDCDEF-Läufe verwendet werden, müssen sie die gleichen Eingabedaten erhalten.

Als Input stellen Sie KDCDEF eine Eingabe-Datei zur Verfügung, die KDCDEF-Steueranweisungen enthält, mit denen Sie die von Ihnen gewünschte Konfiguration beschreiben.

Übersicht: KDCDEF-Steueranweisungen

Anweisung	Funktion
ABSTRACT-SYNTAX	Abstrakte Syntax (OSI TP) definieren
ACCESS-POINT	OSI-TP-Zugriffspunkt für lokale UTM-Anwendung einrichten
ACCOUNT	Abrechnungsparameter festlegen
APPLICATION-CONTEXT	Application Context (OSI TP) definieren
AREA	Namen zusätzlicher Datenbereiche festlegen
BCAMAPPL	weitere Anwendungsnamen definieren
CON	logische Verbindung (LU6.1) zu UTM-Partneranwendungen definieren
CREATE-CONTROL-STATEMENTS	aus Konfigurationsinformationen der vorhandenen KDCFILE Steueranweisungen für erneuten KDCDEF-Lauf erzeugen
EXIT	Event Exits definieren
FORMSYS	Formatierungssystem beschreiben
KSET	Keyset definieren
LPAP	LPAP-Namen für UTM-Partneranwendungen (LU6.1) vergeben
LSES	Sessionname für die Verbindung zweier UTM-Anwendungen über LU6.1 festlegen
LTAC	lokale Namen für TACs in UTM-Partneranwendungen vergeben
LTERM	LTERM-Partner definieren
MAX	Namen der UTM-Anwendung und Ablaufparameter festlegen
MESSAGE	Meldungsmodule beschreiben
OSI-CON	Adresse der Partneranwendung (OSI TP) definieren
OSI-LPAP	OSI-LPAP-Partner definieren
PROGRAM	Namen und Eigenschaften der Teilprogramme festlegen
PTERM	Clients und Drucker definieren
RESERVE	Plätze für die dynamische Konfigurierung reservieren
ROOT	Namen für ROOT-Tabellen-Sourcen vergeben
SESCHA	Sessioneigenschaften (LU6.1) definieren
SFUNC	Sonderfunktionen der Funktionstasten festlegen
SIGNON	Anmeldeverfahren steuern
TAC	Namen und Eigenschaften von Transaktionscodes festlegen
TACCLASS	Prozeßzahl für TAC-Klasse festlegen

Anweisung	**Funktion**
TAC-PRIORITIES	Prioritäten für TAC-Klassen festlegen
TLS	Namen von TLS-Blöcken festlegen
TPOOL	Terminalpools definieren
TRANSFER-SYNTAX	Transfer Syntax (OSI TP) definieren
ULS	Namen von ULS-Blöcken festlegen
USER	Benutzer in Anwendung aufnehmen
UTMD	Anwendungsglobale Werte für verteilte Verarbeitung festlegen
folgende Anweisungen nur für BS2000/OSD:	
DATABASE	Datenbanksystem beschreiben
DEFAULT	Standardwerte definieren
EDIT	Edit-Optionen definieren
ENTRY	Sekundäre Entries in gemeinsam benutzbaren Modulen beschreiben
LOAD-MODULE	Lademodule für den Programmaustausch mit BLS beschreiben
MODULE	INCLUDE-Anweisungen bzw. KDCSHARE-Makros erzeugen
MPOOL	Common Memory Pool beschreiben
MUX	Multiplexanschluß definieren
SATSEL	SAT-Protokollierung und zu protokollierende Ereignisse festlegen
TCBENTRY	Gruppe von TCB-Entries definieren
folgende Anweisung nur für UNIX und Windows NT:	
RMXA	Namen für Resource Manager angeben (Datenbank-Koppelung über die X/Open XA-Schnittstelle)
SHARED-OBJECT	Shared Objects für den Programmaustausch definieren

Neben den oben aufgeführten Anweisungen gibt es noch Anweisungen, um den KDCDEF-Lauf zu steuern oder Kommentare einzufügen.

7.2 Erzeugen des Anwendungsprogramms

Vor dem Erzeugen des Anwendungsprogramms müssen Teilprogramme geschrieben und übersetzt werden. Die Teilprogramme definieren die Anwendungslogik.

Damit die Teilprogramme unter dem Management von *open*UTM ablaufen können, muß ein Anwendungsprogramm erzeugt werden; dazu dienen folgende Schritte:

– den Source-Text für die ROOT-Tabellen, den KDCDEF erzeugt hat, übersetzen

– die übersetzten ROOT-Tabellen, Teilprogramme und eventuell weitere Module wie z.B. Format-Bibliotheken, eigene Meldungsmodule oder sprachspezifische Laufzeitsysteme zusammen mit UTM-Modulen zum Anwendungsprogramm binden. Dies kann statisch (d.h. vor Anwendungsstart) oder dynamisch (d.h. beim Anwendungsstart) erfolgen.

Welche Module Sie genau für Ihre Anwendung benötigen, hängt von Ihrer speziellen Anwendungsarchitektur und der Betriebssystem-Plattform ab. Detaillierte Informationen für UTM-Anwendungen in BS2000/OSD, UNIX und Windows NT finden Sie jeweils im *open*UTM-Handbuch „Anwendungen generieren und betreiben“.

Main Routine KDCROOT

Als Teil des Anwendungsprogramms entsteht beim Binden die Main Routine KDCROOT. KDCROOT fungiert beim Ablauf der Anwendung als steuerndes Hauptprogramm und übernimmt u. a. folgende Aufgaben:

– Verbindung zwischen den Teilprogrammen und den UTM-Systemfunktionen
– Ablauf-Koordination von Teilprogrammen unterschiedlicher Programmiersprachen
– Zusammenarbeit mit Formatierungssystemen
– Kopplung zu Datenbanken

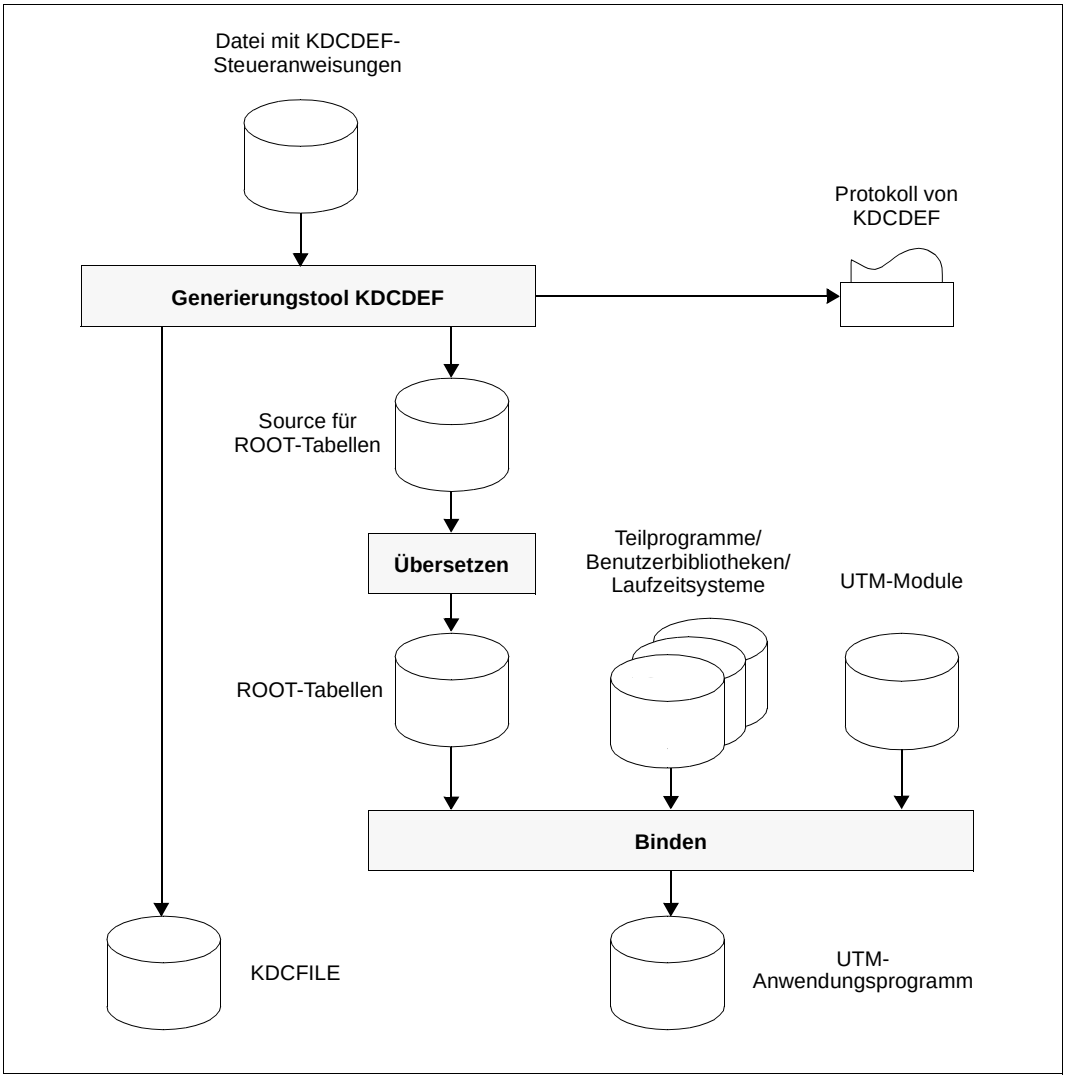


Bild 25: Generierung einer UTM-Anwendung

7.3 Änderungsgenerierung mit dem UTM-Tool KDCUPD

Für grundlegende Änderungen der Konfiguration, die dynamisch nicht durchführbar sind (z.B. Vergrößerung des Pagepools), steht Ihnen das UTM-Tool KDCUPD (**KDC**FILE **UPD**ate) zur Verfügung.

KDCUPD ermöglicht es, nach Beendigung der Anwendung die KDCFILE zu modifizieren, ohne daß die aktuellen Anwenderdaten verloren gehen. KDCUPD überträgt wahlweise entweder alle, oder gezielt nur ganz bestimmte Anwenderdaten aus der bisher benutzten Konfiguration in die neue KDCFILE.

Nachdem die Anwendung neu gestartet wurde, können die Benutzer an den Terminals unmittelbar weiterarbeiten:

- unterbrochene Services werden fortgesetzt
- Message Queues werden weiter bearbeitet:
  - alle Asynchronnachrichten werden ausgegeben
  - alle Hintergrund-Aufträge werden ausgeführt
  - alle zeitgesteuerten Aufträge werden zur festgelegten Zeit von *open*UTM bearbeitet

Konsistenzprüfung mit KDCUPD

KDCUPD bietet Ihnen zusätzlich eine CHECK-Option, mit der Sie Ihre KDCFILE-Dateien auf Konsistenz prüfen können - unabhängig davon, ob Sie die KDCFILE austauschen wollen oder nicht.

Versionswechsel mit KDCUPD

Das UTM-Tool KDCUPD läßt sich auch für den Versionswechsel nutzen. Sie können die Benutzerdaten aus der KDCFILE der Vorgängerversion in die neu erzeugte KDCFILE von *open*UTM V5.0 übertragen. Mit dieser KDCFILE starten Sie die Anwendung unter *open*UTM V5.0, die Benutzer können danach die aktuellen Arbeiten weiterführen. So wird beispielsweise ein zeitgesteuerter Auftrag, den Sie noch mit *open*UTM 4.0 abgesetzt haben, nach Versionswechsel automatisch von *open*UTM 5.0 angestoßen - genau zum gewünschten Zeitpunkt.

7.4 Generieren mit WinAdmin

WinAdmin bietet eine grafische Oberfläche zum Konfigurieren von UTM-Anwendungen. Über diese können Sie auch eine Neugenerierung erstellen oder eine Änderungsgenerierung per KDCUPD anstoßen. Dazu besitzt WinAdmin eine zentrale Generierungs-Datenbasis.

Generierungs-Datenbasis

Die Generierungs-Datenbasis enthält die Generierungsinformationen aller UTM-Anwendungen, die mit WinAdmin verwaltet werden. Damit können sämtliche Daten, die für die Erstellung einer KDCFILE notwendig sind, in der Datenbasis gehalten werden. WinAdmin bietet zum Erstellen und Verwalten dieser Datenbasis eine Reihe von Funktionen.

Generierungsdaten importieren

Für eine vorhandene UTM-Anwendung kann WinAdmin alle Parameter, die über die Administrationsprogrammschnittstelle KDCADMI ermittelt werden können, direkt in die Datenbasis einbringen.

Grafische Oberfläche

Sie können die Daten über die grafische Oberfläche anschauen, neu eintragen oder ändern. Die Oberfläche hat die gleiche Struktur wie für das Administrieren und bietet die gleichen komfortablen Editier-, Listen- und Sortier-Funktionen, siehe Seite 112.

Informationen, die nicht über die Administrations-Programmschnittstelle gelesen werden können (z.B. die ACCOUNT-Einstellungen), lassen sich über Dialogmasken in die Datenbasis einbringen.

Verbindungen konfigurieren

Mit Hilfe der grafischen Oberfläche können Sie auch einen Anwendungsverbund über OSI TP oder LU6.1 generieren. Dabei prüft WinAdmin, ob die notwendigen Parameter für die beteiligten Anwendungen aufeinander abgestimmt sind. Damit werden Generierungsfehler schon im Vorfeld vermieden.

Automatischer Update

Wenn Sie mit WinAdmin dynamisch administrieren und z.B. neue User aufnehmen, dann gibt es die Option, daß diese Änderungen automatisch in der Datenbasis nachgezogen werden. Damit steht Ihnen in WinAdmin immer der aktuelle Generierungsstand einer Anwendung zur Verfügung.

Neue KDCFILE erzeugen und KDCUPD aufrufen

Mit WinAdmin können Sie für jede der administrierten UTM-Anwendungen eine neue KDCFILE erzeugen. Dazu entnimmt WinAdmin die Generierungsdaten aus der Datenbasis, erzeugt daraus eine Datei mit KDCDEF-Anweisungen (siehe Seite 98), überträgt diese auf den Zielrechner und ruft dort das Generierungstool KDCDEF auf.

Diesen Mechanismus können Sie zum einen dafür verwenden, um UTM-Anwendungen komplett neu zu generieren. Zum anderen können Sie für vorhandene Anwendungen auch eine Änderungsgenerierung durchführen, z.B. wenn Sie Parameter ändern müssen, die nicht über die dynamische Administration modifiziert werden können.

Dazu bietet WinAdmin über die grafische Oberfläche die Möglichkeit, mit der neuen KDCFILE eine Änderungsgenerierung mit KDCUPD anzustoßen. KDCUPD wird dann genauso ausgeführt wie bei lokalem Aufruf, siehe Seite 102.

Zum Erzeugen der KDCFILE oder zum Aufruf von KDCUPD muß sowohl auf dem WinAdmin-Rechner als auch auf dem Zielrechner das Produkt *open*FT installiert sein.

8 Administrieren von UTM-Anwendungen

*open*UTM bietet umfassende und einfach einzusetzende Konzepte zur Administration, die den flexiblen und performanten Einsatz der UTM-Anwendungen ermöglichen und dauernde Verfügbarkeit sicherstellen.

Unter dem Begriff „Administration“ sind alle Aktivitäten zusammengefaßt, die zur Steuerung und Verwaltung laufender UTM-Anwendungen dienen, z.B.:

- Cache-Eigenschaften, Anzahl der parallelen Prozesse, Timer, Scheduling etc. festlegen und verändern (Tuning der Anwendung)
- Transaktionscodes sperren oder wieder zulassen
- neue Services integrieren
- Teile des Anwendungsprogramms oder sogar das gesamte Anwendungsprogramm im laufenden Betrieb austauschen
- Clients, Drucker oder Benutzerkennungen in die Konfiguration aufnehmen oder aus der Konfiguration löschen (dynamische Konfigurierung)
- Anzahl der für einen LTERM-Pool zugelassenen und gesperrten Clients oder Drucker festlegen
- logische Verbindungen zu Clients oder fernen Server-Anwendungen auf- oder abbauen
- Meldungen an die angeschlossenen Clients senden
- Diagnosehilfen steuern
- Betriebsdaten der UTM-Anwendung anzeigen lassen
- Aktuelle IP-Adressen ermitteln und der UTM-Anwendung zur Verfügung stellen
- die UTM-Anwendung beenden

Programmschnittstelle KDCADMI

Mit der Programmschnittstelle KDCADMI können individuelle Administrationsprogramme erstellt werden, die auf die jeweilige Anwendung zugeschnitten sind. Über die Programmschnittstelle KDCADMI sind alle Administrationsfunktionen zugänglich, siehe Seite 109.

Kommandoschnittstelle KDCADM

Das Standard-Administrationsprogramm KDCADM bietet eine Kommandoschnittstelle und enthält bereits vordefinierte Basis-Administrationsfunktionen, siehe Seite 107.

Grafisches Administrationsprogramm WinAdmin

Mit der *open*UTM-Komponente WinAdmin können Sie vom PC aus über eine komfortable grafische Oberfläche einzelne oder auch mehrere UTM-Anwendungen administrieren - auch dann, wenn diese UTM-Anwendungen im Netz verteilt sind und auf unterschiedlichen Plattformen ablaufen, siehe auch Seite 112.

Administrieren mit CALLUTM

CALLUTM ist ein vielseitig einsetzbares *open*UTM-Client-Programm im BS2000, mit dem Sie über eine SDF-Oberfläche eine oder mehrere lokale oder ferne UTM-Anwendungen administrieren können. Diese Anwendungen können sich auf unterschiedlichen Plattformen befinden. Näheres zu den Einsatzmöglichkeiten von CALLUTM siehe auch Seite 156.

Die Programmschnittstelle der Administration KDCADMI, das Standard-Administrationsprogramm KDCADM sowie das Programm CALLUTM sind im *open*UTM-Handbuch „Anwendungen administrieren“ im Detail beschrieben. Nähere Informationen zu *open*UTM WinAdmin finden Sie in der ausführlichen Online-Hilfe sowie in einer Readme-Datei. Beides wird zusammen mit WinAdmin ausgeliefert.

8.1 Kommandoschnittstelle zur Administration

Die Basis-Administrationsfunktionen werden über vorgegebene Transaktionscodes aufgerufen, die bei der Generierung dem Standard-Administrationsprogramm (KDCADM) zugeordnet werden. Diese vorgegebenen Transaktionscodes des Standard-Administrationsprogramms werden **Administrationskommandos** genannt.

Für jede Basisfunktion gibt es jeweils einen Dialog-Transaktionscode und einen Asynchron-Transaktionscode. Die Basisfunktionen können also synchron im Dialog genutzt werden oder asynchron über Message Queuing.

Administration im Dialog

Synchron im Dialog werden die Basis-Administrationsfunktionen entweder von einem Administrator am Terminal oder von einem Client-Programm angestoßen. Das gewünschte Administrationskommando wird in beiden Fällen als Dialog-Transaktionscode angegeben. *open*UTM führt die angeforderten Administrationsaufgaben unmittelbar aus und gibt eine entsprechende Antwort zurück.

Mehrere Administratoren und mehrere Administrations-Clients können gleichzeitig arbeiten.

Administration über Message Queuing

Auch bei dieser Variante können die Basis-Administrationsfunktionen vom Administrator am Terminal angestoßen werden. Der Administrator gibt hierfür das gewünschte Kommando als Asynchron-Transaktionscode ein.

Eine zweite Möglichkeit ist, die Basis-Administrationsfunktionen von UTM-Teilprogrammen aus zu nutzen. Das Teilprogramm richtet hierzu einen MQ-Aufruf (FPUT oder DPUT) an den entsprechenden Asynchron-Transaktionscode.

In beiden Fällen reiht *open*UTM den Administrationsauftrag in die entsprechende Queue ein, führt ihn entkoppelt vom Administrator oder vom Teilprogramm aus und informiert über das Ergebnis durch eine Asynchron-Meldung an ein festgelegtes Ziel. Als Ziel kann dabei z.B. das Terminal des Administrators dienen, aber auch ein anderes Terminal, ein Drucker oder ein Asynchron-Programm.

Auch bei der Administration über Message Queuing können mehrere Administratoren bzw. UTM-Teilprogramme gleichzeitig die Administrationsfunktionen nutzen.

Wird die Administration über MQ-Aufrufe aus UTM-Teilprogrammen angestoßen, kann Zeitsteuerung genutzt werden.

Übersicht: Transaktionscodes des Standard-Administrationsprogramms

Dialog-TAC	Asynchron-TAC	Administrationsfunktion
KDCAPPL	KDCAPPLA	Anzahl der Prozesse, Timer-Einstellungen und Maximalwerte ändern; Anwendungsprogramm austauschen; in *open*UTM(BS2000/OSD) Accounting-Funktionen ein- und ausschalten
KDCDIAG	KDCDIAGA	Diagnosehilfen anfordern: Testmodus, Trace- und KDCMON-Funktionen ein- und ausschalten, Dumps anfordern
KDCHELP	KDCHELPA	Auskunft über die Syntax der TACs von KDCADM abfragen
KDCINF	KDCINFA	Aktuelle Einstellungen von Systemparametern, Statistik über Auslastung der Anwendung und Objekteigenschaften abfragen
KDCLOG	KDCLOGA	Benutzer-Protokolldatei auf die nächste Dateigeneration umschalten
KDCLPAP	KDCLPAPA	Zur Administration von UTM-Anwendungen für verteilte Verarbeitung: logische Verbindungen zu Partneranwendungen auf- und abbauen, Ersatzverbindungen zu OSI-TP-Partnern schalten, Partner (ent-)sperren, Timer zur Überwachung der Sessions/Associations ändern
KDCLSES	KDCLSESA	Zur Administration von UTM-Anwendungen für verteilte Verarbeitung: Logische Verbindungen für eine Session auf- und abbauen
KDCLTAC	KDCLTACA	Zur Administration von UTM-Anwendungen für verteilte Verarbeitung: fernen Service (LTAC) für die lokale Anwendung (ent-)sperren, Timer zur Überwachung des Session/Association-Aufbaus und der Antwortzeiten vom Partner-Service einstellen.
KDCLTERM	KDCLTRMA	LTERM-Partner (ent-)sperren, Verbindungen auf- und abbauen
KDCPOOL	KDCPOOLA	Anzahl der für einen Terminalpool zugelassenen Clients ändern
KDCPROG	KDCPROGA	Lademodule des Anwendungsprogramms austauschen
KDCPTERM	KDCPTRMA	Clients/Drucker (ent-)sperren, Verbindungen auf- und abbauen
KDCSHUT	KDCSHUTA	Anwendung beenden
KDCSLOG	KDCSLOGA	System-Protokolldatei (SYSLOG) der Anwendung umschalten, Größenüberwachung ein- und ausschalten, Schwellwert für die Größenüberwachung festlegen, Informationen über SYSLOG abfragen
KDCSWTCH	KDCSWCHA	Zuordnungen zwischen Client/Drucker und LTERM-Partner ändern
KDCTAC	KDCTACA	Transaktionscodes (lokale Services) (ent-)sperren
KDCTCL	KDCTCLA	Anzahl der Prozesse einer TAC-Klasse ändern
KDCUSER	KDCUSERA	Benutzerkennungen (ent-)sperren, Paßwörter ändern
Folgende TACs stehen nur in *open*UTM(BS2000/OSD) zur Verfügung:		
KDCMUX	KDCMUXA	Multiplexanschlüsse (ent-)sperren, Verbindungen auf- und abbauen
KDCSEND	KDCSENDA	Meldungen an Terminal-Benutzer senden

8.2 Programmschnittstelle zur Administration

Mit der Programmschnittstelle KDCADMI, die *open*UTM zur Verfügung stellt, können Sie Administrationsprogramme selbst erstellen, die speziell auf Ihre Anwendung zugeschnitten sind. Da die Programmschnittstelle der Administration für die Programmierung eigener Administrationsprogramme mächtige Aufrufe zur Verfügung stellt, die zudem individuell eingesetzt werden können, bieten selbst erstellte Administrationsprogramme mehr Möglichkeiten als die Basis-Administrationsfunktionen:

- Als Informationsbasis steht nahezu die gesamte Generierungsinformation zur Verfügung.
- Aus dieser Basis können genau die Informationen abgerufen, ausgewertet und weiterverarbeitet werden, die im konkreten Anwendungsfall von Interesse sind.
- In selbst erstellten Administrationsprogrammen lassen sich die Aufrufe zur dynamischen Konfigurierung nutzen (siehe unten).
- Für Administrationsdialoge können Formate verwendet werden.

Die Aufrufe der Programmschnittstelle sind plattform-neutral. So ist es z.B. möglich, von einer UTM-Windows-Anwendung aus eine oder auch mehrere UTM-Anwendungen zu administrieren, die in UNIX oder BS2000/OSD ablaufen, und umgekehrt. Da zudem in Hinblick auf zukünftige UTM-Versionen Source-Kompatibilität garantiert wird, müssen selbst erstellte Administrationsprogramme weder bei einem Plattformwechsel noch beim Umstieg auf neue *open*UTM-Versionen angepaßt werden.

Der Aufwand für die Programmierung eigener Administrationsprogramme ist gering:
Die Aufrufe der Programmschnittstelle können in C-, C++- oder COBOL-Teilprogramme integriert werden. Sowohl Dialog- als auch Asynchron-Programme sind möglich.
Ein Programm kann beliebig viele Administrationsaufrufe enthalten. Die jeweils benötigten Datenstrukturen sind bereits vordefiniert und stehen als Include-Dateien bzw. COPY-Elemente zur Verfügung.

Dynamische Konfigurierung

Die Programmschnittstelle KDCADMI stellt Aufrufe bereit, mit denen Sie die Konfigurierung der Anwendung „on-the-fly“ ändern können: Clients, Drucker, Benutzerkennungen und Services können im laufenden Betrieb in die Konfiguration neu aufgenommen oder aus der Konfiguration gelöscht werden, die Verfügbarkeit des Systems wird dadurch in keiner Weise beeinträchtigt.

Für alle dynamisch konfigurierbaren Objekte können - online oder offline - entsprechende KDCDEF-Anweisungen erzeugt werden (inverser KDCDEF). Da diese Anweisungen dann als Input für das Generierungstool KDCDEF zur Verfügung stehen, lassen sich alle dynamischen Änderungen der Konfiguration bei einer Neu-Generierung problemlos einbeziehen.

Übersicht: Administrationsfunktionen der Programmschnittstelle KDCADMI

Operationscode	Funktion
KC_CHANGE_APPLICATION	gesamtes Anwendungsprogramm im laufenden Betrieb austauschen
KC_CREATE_DUMP	UTM-Dump erzeugen
KC_CREATE_OBJECT	dynamisch neue Objekte (Teilprogramme, Terminals, Benutzer usw.) in die Konfiguration aufnehmen
KC_CREATE_STATEMENTS	im laufenden Betrieb (online) KDCDEF-Steueranweisung für dynamisch konfigurierbare Objekte erzeugen
KC_DELETE_OBJECT	Objekte der Anwendung löschen, d.h. aus der Konfiguration der Anwendung herausnehmen
KC_ENCRYPT	RSA-Schlüsselpaar erzeugen, löschen, auslesen
KC_GET_OBJECT	Information über Objekte und Parameter der Anwendung abfragen
KC_MODIFY_OBJECT	Eigenschaften von Objekten oder Anwendungsparameter ändern
KC_SHUTDOWN	Anwendung beenden
KC_SPOOLOUT	Verbindung zu Druckern automatisch aufbauen, für die Nachrichten vorliegen
KC_SYSLOG	System-Protokolldatei SYSLOG administrieren
KC_UPDATE_IPADDR	IP-Adresse aktualisieren
KC_USLOG	im laufenden Betrieb die Benutzer-Protokolldatei(en) auf die nächste Dateigeneration umschalten
folgenden Operationscode gibt es nur in *open*UTM(BS2000/OSD)	
KC_SEND_MESSAGE	Systemzeilen-Nachricht an ein oder mehrere Dialog-Terminals senden

Beispielprogramme

*open*UTM bietet Ihnen die C-Beispielprogramme HNDLUSR (handle user data) und SUSRMAX (show users and modify MAX values), in denen die Nutzung der KDCADMI-Schnittstelle demonstriert wird. Für COBOL gibt es das Beispielprogramm COBUSER. Da die Beispiele auch als Source-Code ausgeliefert werden, können Sie sie individuell anpassen oder als Vorlage für eigene Administrationsprogramme verwenden. Die Beispielprogramme lassen sich jedoch auch unverändert einsetzen, z.B. um Informationen zu Benutzerkennungen und MAX-Werten abzufragen, aktuelle Einstellungen zu ändern oder um Benutzerkennungen dynamisch zu konfigurieren.

In UNIX sind die Beispielprogramme HNDLUSR und SUSRMAX in die Beispielanwendung integriert, das Beispiel COBUSER finden Sie im Installationsverzeichnis unter *utm/sample/src*. In BS2000/OSD sind alle Beispiele in der Bibliothek SYSLIB.UTM.050.EXAMPLE enthalten.

Was Sie für den Einsatz der Programme wissen müssen, ist am Anfang des jeweiligen Source-Codes in Kommentaren beschrieben.

8.3 Grafisches Administrationsprogramm WinAdmin

WinAdmin ist ein Administrationsprogramm auf einem Windows PC und basiert auf dem UPIC-Client. WinAdmin ist ein grafischer Administrationsarbeitsplatz, von dem aus Sie gleichzeitig mehrere UTM-Anwendungen administrieren können. Dabei kann zwischen der klassischen Anwendungssicht und der Objektsicht gewählt werden.

Die UTM-Anwendungen können auf allen freigegebenen Plattformen laufen und unterschiedliche Versionsstände besitzen. Welche Administrationsfunktionen für die einzelnen UTM-Anwendungen möglich sind, hängt von der Version ab:

- Für UTM-Anwendungen ab V4.0 kann der volle Funktionsumfang der Programmschnittstelle genutzt werden, den die jeweilige Version bietet. Damit können Sie z.B. auch Objekte dynamisch in eine Konfiguration aufnehmen oder Objekte löschen. Für *open*UTM V5.0 sind die Funktionen auf Seite 109 ff aufgelistet.

  Darüberhinaus können Sie mit WinAdmin UTM-Anwendungen starten. Dies setzt voraus, daß auf den beteiligten Rechnern *open*FT im Einsatz ist.

- Für UTM-Anwendungen ab Version 3.4 können alle Funktionen der Kommandoschnittstelle genutzt werden, siehe auch Seite 107.

Grafische Oberfläche

Insbesondere für komplexe Anwendungen bringt der Komfort einer Windows Oberfläche erhebliche Vorteile beim Administrieren wie z.B.:

– Einfaches Navigieren:
  Ein gewünschtes Objekt oder ein bestimmter Anwendungsparameter kann schnell per Mausklick gefunden werden.

– Übersichtlichkeit:
  Die Parameter eines Objekts wie z.B. eines Client oder User sind übersichtlich in einem Fenster aufgelistet und können dort geändert werden.

– Tabellen
  Gleichartige Objekte wie z.B. Drucker oder Lterms werden in übersichtlichen Tabellen aufgelistet und können per Mausklick sortiert werden.

– Diagramme:
  Statistiken können grafisch dargestellt werden, z.B. die Anzahl der Transaktionen pro Sekunde in einem bestimmten Zeitraum. Die Statistikwerte lassen sich einfach in eine Datei sichern und für spätere Analysen heranziehen.

– Plausibilitätsprüfungen:
  Beziehungen zwischen unterschiedlichen Objekten, z.B. zwischen Benutzerkennungen und Keysets werden kontextsensitiv berücksichtigt.

Mehrere Anwendungen administrieren

WinAdmin erlaubt einen single-point-of-view auf mehrere Anwendungen, indem Sie diese Anwendungen zu sogenannten „Kollektionen“ zusammenfassen. Sie können z.B. eine Kollektion aus allen Filial-Anwendungen bilden und diese gemeinsam administrieren, während Sie die Anwendungen in der Zentrale separat betrachten.

Anwendungssicht

Bei dieser klassischen Sichtweise sehen Sie eine bestimmte Anwendung auf einem bestimmten Rechner mit deren Einstellungen und Objekten. Damit können Sie von „oben nach unten“ arbeiten und gezielt einzelne Parameter anschauen und ändern, oder Objekte wie z.B. User und Clients löschen oder neu erzeugen.

Objektsicht

Hierbei wählen Sie Objekte „querbeet“ über mehrere Anwendungen hinweg, z.B. alle User einer Kollektion. Anschließend können Sie weiter nach Server oder Anwendung differenzieren und sich alle gesperrten User mehrerer Anwendungen ausgeben lassen. Diese können Sie dann mit einer Aktion entsperren.

Für die schnelle und umfassende Information können Sie Objektlisten nach vielfältigen Kriterien zusammenstellen und sortieren.

8.4 Berechtigungskonzept

Speziell für die Administration stellt *open*UTM - zusätzlich zu den allgemeinen Security-Funktionen - ein zweistufiges Berechtigungskonzept zur Verfügung.

- Stufe 1: Lesender Zugriff auf alle Informationen der Administration

  Wenn einem Transaktionscode eines Administrationsprogramms bei der Konfiguration ADMIN=READ zugeordnet wird, darf das entsprechende Programm auf sämtliche Informationen lesend zugreifen. Um diesen Transaktionscode aufzurufen, braucht der Benutzer keine Administrationsberechtigung zu haben. Die Informationen sind also für alle Benutzer zugänglich.

- Stufe 2: Volle Administrationsberechtigung

  Um alle Administrationsfunktionen (Kommandos und selbst erstellte Administrationsprogramme) uneingeschränkt nutzen zu können, müssen folgende Voraussetzungen erfüllt sein:

  – Für einen Transaktionscode, der ein Administrationsprogramm aufruft, muß bei der Konfigurierung ADMIN=Y vereinbart werden.

  – Der Benutzer, der diesen Transaktionscode aufruft, muß administrationsberechtigt sein. Hierzu muß für die Benutzerkennung bzw. Partner-Anwendung bei der Konfigurierung PERMIT=ADMIN vereinbart werden.

Zusätzlich lassen sich feinere Differenzierungen mit dem Zugriffsschutzmechanismus des Lock-/Keycode-Konzepts realisieren (siehe Seite 123).

8.5 Automatische Administration

Wenn Sie die Möglichkeiten der Ereignissteuerung nutzen, die *open*UTM Ihnen zur Verfügung stellt, können Sie Administrationsaufgaben automatisieren.

Dies funktioniert nach folgendem Schema:
*open*UTM signalisiert ein bestimmtes Ereignis (z.B. Verbindungsverlust) durch eine UTM-Meldung, der Sie als Meldungsziel MSGTAC zugeordnet haben. Diese Meldung aktiviert automatisch den Event-Service MSGTAC. Die MSGTAC-Routine liest die Meldung ein, wertet sie aus, und stößt die entsprechende Administrationsfunktion an.

Folgende Vorbereitungen sind hierfür notwendig:

– Den für die Administration relevanten UTM-Meldungen ordnen Sie das Meldungsziel MSGTAC zu. Hierfür steht Ihnen das UTM-Tool KDCMMOD (**KDC M**essage **MOD**ify) zur Verfügung.

– Die MSGTAC-Routine realisieren Sie als KDCS-Teilprogramm, in dem die Meldungen mit dem KDCS-Aufruf FGET eingelesen werden. Die Auswertung der eingelesenen Meldungen ist leicht zu programmieren, da *open*UTM Datenstrukturen zur Verfügung stellt, die der Struktur der Meldungen entsprechen (für COBOL im COPY-Element KCMSGC, für C/C++ in der Include-Datei *kcmsg.h*). Um die entsprechenden Administrationsfunktionen anzustoßen verwenden Sie FPUT- oder DPUT-Aufrufe.

– Bei der Generierung nehmen Sie das angepaßte Meldungsmodul mit der KDCDEF-Anweisung MESSAGE in die Konfiguration auf und ordnen der MSGTAC-Routine mit der KDCDEF-Anweisung TAC den Transaktionscode KDCMSGTC zu.

  Das übersetzte Meldungsmodul und die übersetzte MSGTAC-Routine binden Sie ins Anwendungsprogramm ein.

Im *open*UTM-Handbuch „Anwendungen programmieren mit KDCS“ finden Sie weitere Informationen zum Event-Service MSGTAC und Beispiele für MSGTAC-Routinen in C und COBOL.

8.6 Administration von Message Queues und Druckern

*open*UTM bietet zur Administration von Message Queues den KDCS-Aufruf DADM (**D**elayed free message **ADM**inistration). Mit diesem Aufruf können Sie:

- Übersichtsinformationen über den gesamten Inhalt einer Queue anfordern
- gezielt Informationen zu einzelnen Elementen der Queue anfordern
- Asynchron-Aufträge vorziehen
- Asynchron-Aufträge stornieren
- die gesamte Queue löschen

Zur Administration von Druckern dient der KDCS-Aufruf PADM (**P**rinter **ADM**inistration), der folgende Möglichkeiten bietet:

- Druckausgaben bestätigen oder wiederholen
- zwischen Quittungs- und Automatikmodus umschalten
- Informationen über einen Drucker oder eine Druckausgabe anfordern
- Zuordnung eines Druckers ändern
- Drucker sperren und entsperren, Verbindungen zu Druckern aufbauen oder abbauen

Diese Aufrufe sind über die KDCS-Programmschnittstelle und nicht über die Administrationsschnittstelle realisiert. Für so alltägliche Aufgaben wie das Stornieren von eigenen Druckaufträgen oder das explizite Bestätigen von wichtigen Drucken wie z.B. Scheckformularen ist deshalb volle Administrationsberechtigung nicht zwingend erforderlich:
Bei der Generierung können Sie Drucksteuer-LTERMs definieren, um zu ermöglichen, daß Anwender die von ihnen standardmäßig genutzten Drucker und Druckauftrags-Queues auch ohne Administrationsberechtigung selbst administrieren können. Für die Administration „fremder“ Drucker und Queues ist die Administrationsberechtigung jedoch erforderlich.

*open*UTM stellt Ihnen die Beispielprogramme KDCDADM und KDCPADM zur Verfügung: Diese Beispielprogramme machen alle Leistungen der Aufrufe DADM und PADM unmittelbar zugänglich.

Nähere Informationen zur Administration von Message Queues und Druckern sowie zu den Beispielprogrammen finden Sie im *open*UTM-Handbuch „Anwendungen administrieren“.

9 Security-Funktionen

Ohne geeignete Security-Funktionen wären unternehmensweite, integrierte IT-Lösungen undenkbar. *open*UTM stellt Ihnen umfassende, differenzierbare und klar strukturierte Security-Konzepte zur Verfügung. Diese ermöglichen selbst dort offene Lösungen, wo es aus Gründen der Sicherheit bisher nicht möglich war.

*open*UTM bietet Ihnen:

- Funktionen zur Zugangskontrolle (Identifizierung, Authentisierung), auch bei Client/Server-Kommunikation und bei verteilter Verarbeitung über OSI TP
- Funktionen zur Zugriffskontrolle (Autorisierung)
- Zugangs- und Zugriffskontrolle auch bei verteilter Verarbeitung
- Verschlüsselung von Paßwörtern und Benutzerdaten bei Client/Server-Kommunikation
- zusätzliches zweistufiges Berechtigungskonzept speziell für die Administration (siehe Abschnitt 8.4 auf Seite 114)
- Nutzungsmöglichkeit der Security-Mechanismen externer Resource Manager

9.1 Zugangskontrolle (Identifizierung und Authentisierung)

Für die Realisierung der Zugangskontrolle bietet Ihnen *open*UTM folgende Möglichkeiten:

- Definition von logischen Anschlußpunkten für Clients und Partner-Server
- Benutzerkennungen und Paßwörter beim Einsatz von Terminals
- Benutzerkennungen und Paßwörter beim Einsatz von Client-Programmen
- Zugangsschutz durch Ausweisprüfung
- Stiller Alarm bei wiederholten Fehlversuchen
- Automatischer Verbindungsabbau bei wiederholten Fehlversuchen
- Selbst programmierte Berechtigungsprüfungen - Event-Service SIGNON

In den folgenden Abschnitten werden diese Möglichkeiten kurz erläutert. Das genaue Format der entsprechenden Generierungsanweisungen finden Sie im *open*UTM-Handbuch „Anwendungen generieren und betreiben“. Die KDCADMI-Aufrufe sind im Detail im *open*UTM-Handbuch „Anwendungen administrieren“ beschrieben

Definition von logischen Anschlußpunkten für Clients und Partner-Server

Jeder Client und jeder Partner-Server, der eine UTM-Anwendung nutzen will, muß dieser UTM-Anwendung bekannt sein. Ein Client oder Partner-Server ist einer Anwendung dann bekannt, wenn er einem in der Konfiguration der UTM-Anwendung definierten logischen Anschlußpunkt zugeordnet ist.

LTERM-Partner - Anschlußpunkte für Terminals und Client-Programme

Die logischen Anschlußpunkte für Clients heißen LTERM-Partner (**L**ogical **TERM**inal) und werden mit der KDCDEF-Anweisung LTERM generiert. Mit der KDCDEF-Anweisung PTERM (**P**hysical **TERM**inal) wird dem LTERM-Anschlußpunkt ein „realer“ Client zugeordnet. LTERM-Partner und PTERM-Zuordnungen lassen sich auch dynamisch bei laufender Anwendung definieren (KCADMI-Aufruf KC_CREATE_OBJECT).

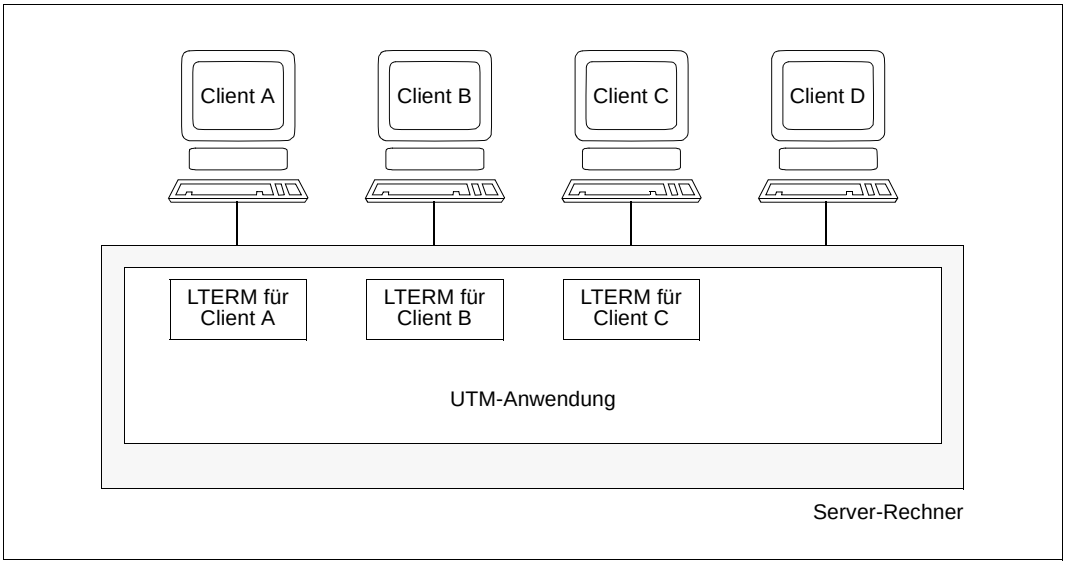


Bild 26: Anschluß von Clients über LTERM-Partner

Bei der in Bild 26 dargestellten Beispiel-Konfiguration könnten die Clients A, B und C mit der UTM-Anwendung arbeiten. Obwohl auch Client D über eine Leitung zum Server-Rechner verfügt, hat er keinen Zugang zur UTM-Anwendung, weil ihm in der Konfiguration der Anwendung kein LTERM-Partner zugeordnet ist.

Falls gewünscht, können Sie dieses strenge Zuordnungsschema durch die Nutzung von LTERM-Pools lockern. Diese ermöglichen einer festlegbaren Anzahl von Clients den Zugang zur UTM-Anwendung, ohne daß jeder Client explizit einem LTERM-Partner zugeordnet werden müßte. Falls Sie einen LTERM-Pool einsetzen, wird jedem Client, der sich an die UTM-Anwendung anschließen will und nicht explizit generiert ist, automatisch ein LTERM-Partner aus dem Pool zugeordnet.

(OSI-)LPAP-Partner - Anschlußpunkte für Partner-Server

Die logischen Anschlußpunkte für Partner-Server heißen LPAP-Partner (**L**ogical **P**artner **AP**plication) oder, falls über das OSI-TP-Protokoll gearbeitet werden soll, OSI-LPAP-Partner. Sie werden entsprechend mit den KDCDEF-Anweisungen LPAP bzw. OSI-LPAP definiert. Für die Zuordnung der „realen“ Partner-Anwendung steht die KDCDEF-Anweisung CON (**CON**nection) bzw. OSI-CON zur Verfügung.

Definition von Benutzerkennungen und Paßwörtern

Für UTM-Anwendungen können Benutzerkennungen definiert werden, entweder bereits bei der Generierung mit der KDCDEF-Anweisung USER oder dynamisch mit dem KDCADMI-Aufruf KC_CREATE_OBJECT.

Wird eine UTM-Anwendung mit UTM-Benutzerkennungen generiert, dann können benutzerspezifische Paßwörter vereinbart werden.

Für die Paßwortvergabe kann eine bestimmte Mindestlänge und eine bestimmte Komplexitätsstufe zur Bedingung gemacht werden. Ebenso läßt sich für jeden Benutzer die minimale und maximale Gültigkeitsdauer seines Paßwortes festlegen.

Die Paßwörter der Benutzerkennungen werden von *open*UTM verschlüsselt abgespeichert, d.h., sie sind in einem Dump nicht erkennbar. Paßwörter werden bei Kommunikation mit Clients und Terminal-Emulationen verschlüsselt, sofern diese die Verschlüsselung unterstützen.

Benutzer haben im laufenden Betrieb die Möglichkeit, das eigene Paßwort zu ändern, sofern für die UTM-Anwendung ein entsprechender Service erstellt wurde.

Benutzerkennungen und Paßwörter beim Einsatz von Terminals

Alle Terminal-Benutzer, die mit einer UTM-Anwendung arbeiten wollen, müssen sich dann gegenüber der UTM-Anwendung durch Angabe ihrer Benutzerkennung identifizieren. Diese Berechtigungsprüfung wird auch KDCSIGN genannt. Sofern ein Paßwort generiert wurde, muß dies ebenfalls angegeben werden.

Terminal-Benutzer haben bei der Anmeldung die Möglichkeit, das eigene Paßwort selbst zu ändern (in UTM (BS2000) nur bei dunkelgesteuertem Paßwort).

Benutzerkennungen und Paßwörter beim Einsatz von Client-Programmen

Das Identifizierungs- und Authentisierungskonzept von *open*UTM steht Ihnen nicht nur beim Einsatz von Terminals zur Verfügung. Auch wenn Sie *open*UTM-Client-Programme einsetzen, können Sie Benutzerkennungen und Paßwörter nutzen. Die Client-Programme übergeben der UTM-Anwendung Benutzerkennung und Paßwort über spezielle Aufrufe:

– Für CPI-C sind das die Aufrufe *Set_Conversation_Security_User_ID* und *Set_Conversation_Security_Password*.

– Bei XATMI dienen hierfür die Parameter *usrname* und *passwd* des Aufrufs *tpinit.*

Vor dem Start eines Services validiert *open*UTM die Berechtigungsdaten, die vom Client übergeben werden, und ordnet die jeweilige Benutzerkennung und das entsprechende Berechtigungsprofil zu (siehe Seite 123). Dies entspricht dem KDCSIGN eines Terminal-Benutzers.

Ein und dasselbe Client-Programm kann also unter verschiedenen Benutzerkennungen mit jeweils individuellen Berechtigungsprofilen arbeiten.

Wenn ein Client-Programm **keine** Berechtigungsdaten übergibt, wird eine Standard-Benutzerkennung zugeordnet. Somit können auch solche Clients mit UTM-Anwendungen arbeiten, denen keine Protokolle und Schnittstellen zur Übergabe von Berechtigungsdaten zur Verfügung stehen - allerdings nur mit einem feststehenden Standard-Berechtigungsprofil.

Genaue Informationen zum Security-Konzept beim Anschluß von Client-Programmen finden Sie in den *open*UTM-Client-Handbüchern.

Benutzerkonzept bei Server-Server-Kommunikation über OSI TP

Das Benutzerkonzept von *open*UTM steht Ihnen bei der Server-Server-Kommunikation über OSI TP anwendungsübergreifend zur Verfügung. Bei der Adressierung des Partners können Sie im APRO-Aufruf einen Security-Typ wählen:

N (none)
Es werden keine Berechtigungsdaten an den Auftragnehmer übergeben.

S (same)
Es wird die Benutzerkennung an den Auftragnehmer übergeben, unter der der lokale Service läuft.

P (program)
Es werden im Programm explizit spezifizierte Werte als Benutzerkennung und Paßwort an den Auftragnehmer übergeben.

Zugangsschutz bei Terminals durch Ausweisprüfung

Benutzerkennungen für eine UTM-Anwendung können so konfiguriert werden, daß der Zugang zu der Anwendung über eine Benutzerkennung nur mit einem speziellen Ausweis möglich ist. Dazu muß am Terminal ein entsprechender Leser vorhanden sein. Wird während des Arbeitens mit der UTM-Anwendung der Ausweis aus dem Leser entfernt, bricht *open*UTM die Verbindung zum Terminal ab. Bei Terminals mit entsprechenden Lesern kann somit der Zugang zur UTM-Anwendung abhängig gemacht werden von einer Benutzerkennung, einem Paßwort und vom Besitz eines gültigen Ausweises.

Stiller Alarm bei wiederholten Fehlversuchen

Werden von einem Terminal oder Client-Programm aus nacheinander mehrere erfolglose (fehlerhafte) Anmeldeversuche unternommen, dann erzeugt *open*UTM intern eine Meldung mit dem Standardziel SYSLOG (System-Protokolldatei) und wahlweise auch MSGTAC, um auf mögliche Eindringversuche hinzuweisen. Dadurch haben Sie die Möglichkeit, in sol-

chen Fällen gezielte Maßnahmen einzuleiten, z.B. mittels automatischer Administration (siehe Seite 115). Sie können in der Konfiguration festlegen, nach wieviel fehlerhaften Anmeldeversuchen diese Meldung erzeugt werden soll.

Automatischer Verbindungsabbau

Bei der Generierung können Sie festlegen, wie lange die UTM-Anwendung nach Transaktionsende - oder auch nach Dialogausgabe innerhalb einer Transaktion - maximal auf eine Eingabe von einem Client warten soll. Erfolgt in dieser Zeit keine Eingabe, dann wird die Verbindung zum Client abgebaut. Ist der Client ein Terminal, dann wird dabei zusätzlich eine Meldung ausgegeben. Wenn beispielsweise ein Benutzer nach Beendigung seiner Arbeit vergessen hat, sich bei der UTM-Anwendung abzumelden, wird durch diesen automatischen Verbindungsabbau die Wahrscheinlichkeit für einen unberechtigten Zugang zur UTM-Anwendung verringert.

Selbst programmierte Berechtigungsprüfungen - Event-Service SIGNON

Im Standardfall werden die Terminal-Benutzer bei der Anmeldung an eine UTM-Anwendung durch vorgegebene UTM-Meldungen aufgefordert, Benutzerkennung und Paßwort anzugeben und ggf. den Ausweis einzulegen.

Mit dem Event-Service SIGNON können Sie den Anmeldungsdialog für Ihre Anwendung jedoch auch individuell gestalten und zusätzlich zu den Berechtigungsprüfungen von *open*UTM eigene Berechtigungsprüfungen durchführen. Dieser Service kann auch von Clients mit Trägersystem UPIC oder von Transportsystem-Anwendungen genutzt werden.

Mit *open*UTM werden Beispiel-Programme für einen SIGNON-Service ausgeliefert - sowohl die übersetzten Objekte als auch die COBOL-Sourcen. Diesen SIGNON-Service, der einen Anmeldungsdialog mit Formaten realisiert, können Sie als Vorlage verwenden und individuell abändern, oder auch unverändert einsetzen.

Informationen über Programmierung und Einsatzmöglichkeiten von SIGNON-Services finden Sie im *open*UTM-Handbuch „Anwendungen generieren und betreiben“.

9.2 Zugriffskontrolle (Autorisierung): Lock-/Keycode-Konzept

UTM-Anwendungen umfassen meist eine Vielzahl von Services. Darunter sind in der Regel einige, die allen Benutzern zur Verfügung stehen sollen. Andere dagegen sollen nur von bestimmten Benutzern gestartet werden können. Bei Services, die Zugriff auf sicherheitsrelevante Daten haben, ist es sinnvoll, den Zugriff auf einige wenige Benutzer zu beschränken. Deshalb bietet Ihnen *open*UTM mit dem Lock-/Keycode-Konzept die Möglichkeit, in der Konfiguration einer UTM-Anwendung die Zugriffsrechte mehrstufig differenziert festzulegen. Im Prinzip kann für jeden einzelnen Benutzer ein persönliches Berechtigungsprofil vereinbart werden.

Mit dem Lock-/Keycode-Konzept können Sie z.B. erreichen, daß nur besonders autorisierte Benutzer oder Client-Programme bestimmte Services der UTM-Anwendung verwenden dürfen. Sie können auch vereinbaren, daß die Anmeldung unter einer Benutzerkennung nur über bestimmte LTERM-Partner (Anschlußpunkte) möglich ist oder daß bestimmte Services nur über spezielle LTERM-Partner gestartet werden können. So ist es möglich, die Berechtigung eines Benutzers, einen bestimmten Service zu starten, auf die Verwendung eines speziellen, besonders gesicherten Terminals oder Client-Rechners zu beschränken.

*open*UTM verwendet zu diesem Zweck ein **Lock-/Keycode-Konzept**. Die zu schützenden Objekte - das sind zum Beispiel LTERM-Partner und den Services zugeordnete Transaktionscodes - können mit einem Lockcode versehen werden. Als Lockcode wird eine Zahl vergeben, die ein logisches Zahlenschloß darstellt. Für Benutzerkennungen und LTERM-Partner werden Keycodes definiert. Wenn ein Keycode mit dem Lockcode eines gesicherten Objekts übereinstimmt, ist der Zugriff auf dieses Objekt erlaubt.
In der Regel hat eine Benutzerkennung oder ein LTERM-Partner Zugriff auf mehrere Services und verfügt deshalb über mehrere Keycodes. Die einzelnen Keycodes sind daher jeweils zu Keysets zusammengefaßt.

Das Lock-/Keycode-Konzept hat folgende Effekte:

– Die Anmeldung eines Terminals oder eines Client-Programms ist nur möglich, wenn der angegeben Benutzerkennung ein Keycode zugeordnet ist, der mit dem Lockcode des zugeordneten LTERM-Partners übereinstimmt.

– Ein Terminal-Benutzer oder ein Client-Programm kann einen Service nur dann aufrufen, wenn **sowohl** das Keyset der jeweiligen Benutzerkennung **als auch** das des LTERM-Partners einen Keycode enthalten, der mit dem Lockcode des Transaktionscodes übereinstimmt.

Nachstehend ein Beispiel für die Nutzung des Lock-/Keycode-Konzeptes.

Beispiel für die Nutzung des Lock-/Keycode-Konzepts

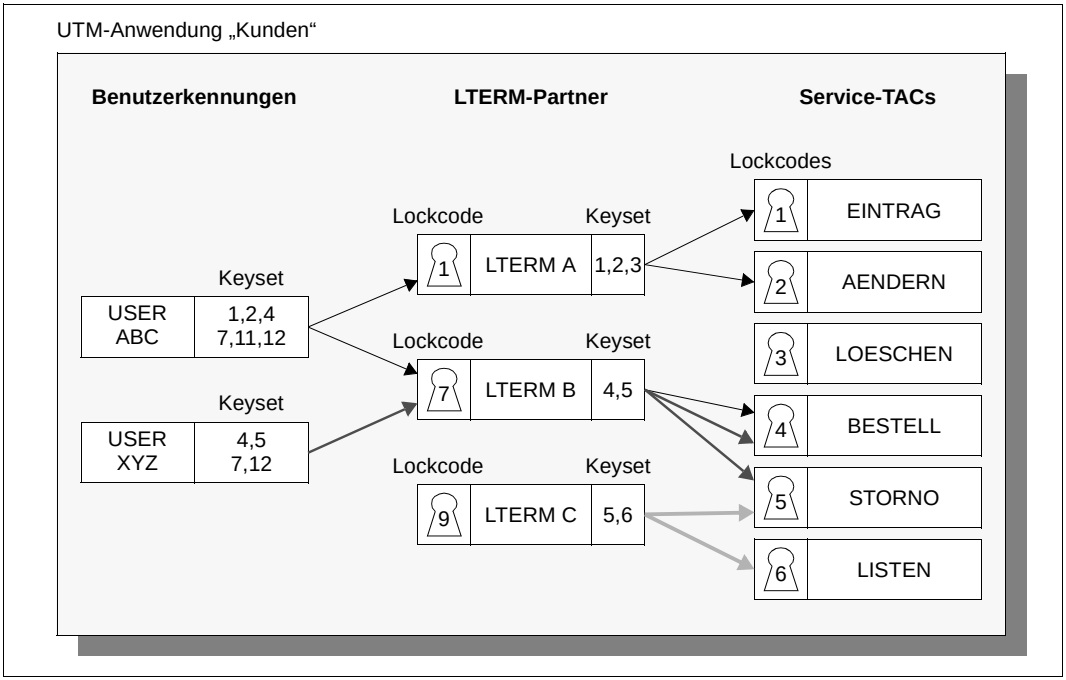


Bild 27: Zugriffsschutz mit dem Lock-/Keycode-Konzept

Ein Benutzer mit der Benutzerkennung ABC möchte mit der UTM-Anwendung „Kunden“ arbeiten. Hierzu muß er sich gegenüber der UTM-Anwendung durch Angabe seiner Kennung identifizieren und ggf. ein Paßwort eingeben.

Das Keyset der Benutzerkennung ABC umfaßt die Keycodes 1,2,4,7,11 und 12. Damit darf sich der Benutzer sowohl über den Client anmelden, der dem LTERM-Partner A (Lockcode 1) zugeordnet ist, als auch über den Client, der dem LTERM-Partner B (Lockcode 7) zugeordnet ist. Der LTERM-Partner C ist mit dem Lockcode 9 gesichert. Da die Benutzerkennung ABC aber keinen entsprechenden Keycode besitzt, würde ein Versuch des Benutzers, sich über den zugeordneten Client anzumelden, von der UTM-Anwendung abgewiesen.

Ein Benutzer kann einen Service nur dann starten, wenn **sowohl** seine Benutzerkennung **als auch** der LTERM-Partner über einen Keycode verfügen, der zum Lockcode des entsprechenden Transaktionscodes (Service-TACs) paßt.

Der LTERM-Partner A besitzt die Keycodes 1,2,3. Da der Benutzerkennung ABC jedoch der Keycode 3 fehlt, könnte der Benutzer mit der Benutzerkennung ABC über diesen LTERM-Partner nur die Services „EINTRAG“ (Lockcode 1) und „AENDERN“ (Lockcode 2) starten.

9.3 Zugangs- und Zugriffskontrolle bei verteilter Verarbeitung

Die umfangreichen Security-Funktionen, die *open*UTM bietet, stehen Ihnen auch dann zur Verfügung, wenn bei verteilter Verarbeitung mehrere UTM-Anwendungen über Server-Server-Kommunikation zusammenarbeiten.

Zugangsschutz durch logische Anschlußpunkte

UTM-Anwendungen können nur dann zusammenarbeiten, wenn in jeder Anwendung für die jeweils (aus Sicht dieser Anwendung) fernen Partner-Anwendungen logische Anschlußpunkte definiert sind. Diese Anschlußpunkte heißen (OSI-)LPAP-Partner (siehe auch Seite 119).

Zusätzlich können Sie bei verteilter Verarbeitung über OSI TP das UTM-Benutzerkonzept auch anwendungsübergreifend nutzen (siehe Seite 121).

Lock-/Keycode-Konzept bei verteilter Verarbeitung

Auch bei verteilter Verarbeitung steht Ihnen das Lock-/Keycode-Konzept von *open*UTM zur Verfügung. Die Schutzmaßnahmen werden bei der Generierung der Anwendungen festgelegt.

- Schutzmaßnahmen in der Auftraggeber-Anwendung:

  Bei der Generierung einer Anwendung legen Sie zunächst generell fest, welche Services einer fernen Partner-Anwendung aufrufbar sein sollen: Für jeden fernen Service, der genutzt werden soll, vereinbaren Sie einen lokalen Transaktionscode (LTAC). Der Zugriff auf ferne Services, für die keine LTACs vereinbart sind, bleibt der Anwendung grundsätzlich verwehrt.

  Um den Zugriffsschutz weiter zu differenzieren, können Sie die einzelnen LTACs mit Lockcodes versehen. Ein Service der lokalen Anwendung kann nur dann einen fernen Service anfordern, wenn der Benutzer, der den lokalen Service gestartet hat, über entsprechende Keycodes verfügt.

- Schutzmaßnahmen in der Partner-Server-Anwendung:

  In der Partner-Server-Anwendung wird der Auftraggeber-Anwendung ein Keyset zugeordnet. Nur wenn dieses Keyset einen Keycode enthält, der zum Lockcode des angeforderten Services paßt, wird der von der Auftraggeber-Anwendung angeforderte Vorgang gestartet.

Damit auf einen fernen Service zugegriffen werden kann, muß also in der lokalen Anwendung der Lockcode des LTACS überwunden werden. Außerdem muß das Keyset, das die Partner-Server-Anwendung allen von der Auftraggeber-Anwendung eingehenden Anforderungen zuordnet, einen Keycode enthalten, der zu dem in der Partner-Server-Anwendung definierten Lockcode des Services paßt.

Beispiel: Lock-/Keycode-Konzept bei verteilter Verarbeitung

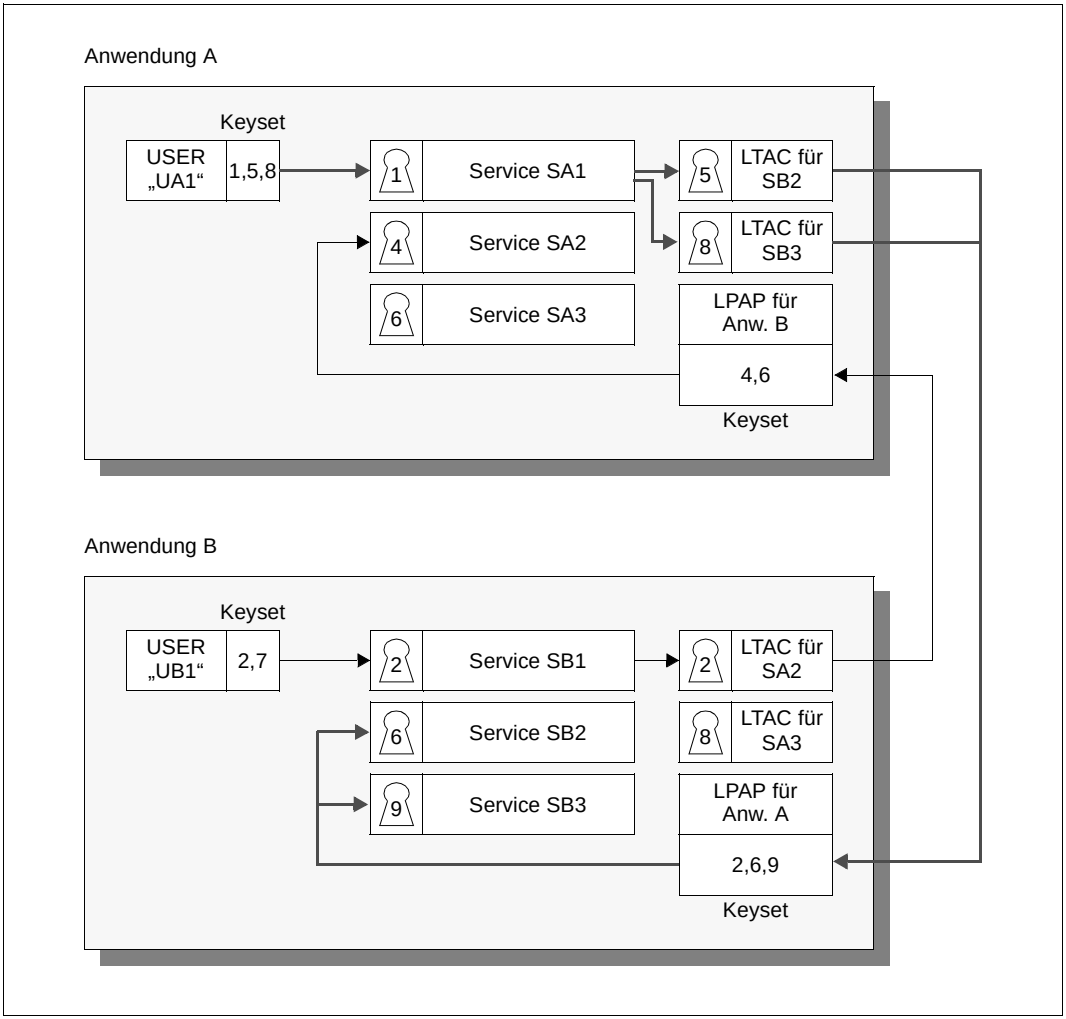


Bild 28: Zugriffsschutz mit dem Lock-/Keycode-Konzept bei verteilter Verarbeitung

In dem in Bild 28 gezeigten Beispiel ist eine Server-Server-Kommunikation zwischen Anwendung A und Anwendung B grundsätzlich möglich, da in jeder Anwendung für die jeweils ferne Anwendung ein LPAP-Partner generiert ist. In der Anwendung A sind LTACs nur für die fernen Services SB2 und SB3 generiert. Der Service SB1 kann also von Anwendung A aus auf keinen Fall genutzt werden. Ein Benutzer, der sich an Anwendung A unter der Benutzerkennung „UA1“ anmeldet, verfügt über den Keycode 1 und kann damit in der Anwendung A den Service SA1 nutzen.

Da der Benutzer auch passende Keycodes für die beiden LTACs besitzt, kann Service SA1 über diese LTACs die Dienste der fernen Services SB2 und SB3 anfordern. Auch in dem Keyset, das Anwendung B allen von Anwendung A eingehenden Anforderungen zuordnet, sind passende Keycodes für die Services SB2 und SB3 enthalten. Somit sind sowohl in der Konfiguration von Anwendung A als auch in der Konfiguration von Anwendung B alle Voraussetzungen erfüllt: Der unter der Benutzerkennung UA1 in der Anwendung A gestartete Service SA1 darf auf die Services SB2 und SB3 der Anwendung B zugreifen.

9.4 Verschlüsselung

Clients greifen häufig über offene Netze auf UTM-Services zu. Damit besteht die Möglichkeit, daß Unbefugte auf der Leitung mitlesen und z.B. Paßwörter für UTM-Benutzerkennungen oder sensible Benutzerdaten ermitteln. Um dies zu verhindern, unterstützt *open*UTM die Verschlüsselung von Paßwörtern und Benutzerdaten für Client-Verbindungen.

*open*UTM verwendet zum Verschlüsseln eine Kombination aus DES-Verfahren (Data Encryption Standard)- und RSA-Verfahren, benannt nach den Autoren Rivest, Shamir und Adleman. Dieser Mechanismus kann bei allen Clients sinnvoll genutzt werden, die diese Verfahren ebenfalls beherrschen, wie z.B. UPIC-Clients ab *open*UTM-Client V5.0. Der DES-Schlüssel wird vom Client erzeugt, während das RSA-Schlüsselpaar (*öffentlicher Schlüssel (= public key)* und *privater Schlüssel (= private key)*) von *open*UTM erzeugt wird. Die RSA-Schlüssel können in *open*UTM per Administration geändert werden. Zusätzlich besteht für den UPIC-Client die Möglichkeit, den *öffentlichen Schlüssel* vorab lokal zu hinterlegen. In diesem Fall kann der Client beim Verbindungsaufbau verifizieren, ob der empfangene *öffentliche Schlüssel* mit dem lokal hinterlegten Schlüssel übereinstimmt.

Aus rechtlichen Gründen werden die Verschlüsselungs-Funktionen von *open*UTM als eigenes Produkt ausgeliefert, das separat installiert werden muß.

Die Verschlüsselung kann bei *open*UTM dazu verwendet werden, sowohl den *Zugang* durch Clients als auch den *Zugriff* auf bestimmte Services zu kontrollieren.

Zugangsschutz

In der UTM-Konfiguration kann man für jeden Client und jede Client-Gruppe festgelegen, ob und inwieweit sie Nachrichten und Paßwort verschlüsseln müssen. Dabei gibt es drei Fälle:

1. Der Client muß auf jeden Fall verschlüsseln, sonst bekommt er keinen Zugang zur UTM-Anwendung.
2. Der Client wird zwar ohne Verschlüsselung zugelassen, er muß jedoch verschlüsseln, wenn es ein Service explizit verlangt (siehe unten unter „Zugriffsschutz“).
3. Der Client gilt als sicher (*trusted*) und muß nicht verschlüsseln.

Zugriffsschutz

*open*UTM kann einzelne Services schützen. Ein Client darf auf solche Services nur dann zugreifen, wenn er entweder als sicher eingestuft ist oder wenn er verschlüsseln kann.

Versucht ein Client, der nicht als sicher gilt, auf einen derart geschützten Service zuzugreifen, dann muß er eine eventuelle Eingabenachricht verschlüsselt senden. *open*UTM schickt die Ausgabenachricht verschlüsselt zurück, auch dann, wenn der Client den Service ohne Eingabenachricht gestartet hat oder der Service durch Vorgangskettung gestartet wurde.

Informationen über Verschlüsselung für Clients und Services finden Sie im *open*UTM-Handbuch „Anwendungen generieren und betreiben“ unter dem Stichwort ENCRYPTION-LEVEL.

Auch Java-Clients auf Basis von *open*UTM-JetClient können den Datenverkehr verschlüsseln, siehe Seite 53.

9.5 Security-Funktionen externer Resource Manager

Wenn Ihre UTM-Anwendungen mit externen Resource-Managern, wie z.B. Datenbanksystemen, zusammenarbeiten, können Sie selbstverständlich zusätzlich die speziellen Schutzmechanismen dieser Resource Manager nutzen. Die Schutzmechanismen von *open*UTM und die des Datenbanksystems bleiben dabei klar getrennt: Es findet keine Delegation statt, d.h., die einzelnen Benutzer werden nicht „durchgereicht“. Es ist die UTM-Anwendung, die gegenüber der Datenbank als Benutzer agiert.

Dies hat eine Reihe positiver Effekte:

– klare Trennung der Komponenten (Tiers) „Anwendung“ und „Datenhaltung“

– Zugriffsschutz bereits auf Service-Ebene und nicht erst auf Daten-Ebene

– keine unnötigen Redundanzen in den Berechtigungsprofilen, wodurch der Administrationsaufwand verringert wird und Inkonsistenzen, wie z.B. Änderungsanomalien, ausgeschlossen werden

– schnellere Zugriffszeiten und effizienteren Ressourcen-Einsatz im Datenbanksystem (Verwaltungstabellen, Caches)

10 Fehlertoleranz und Wiederanlauf

*open*UTM schützt Ihre Daten und Anwendungen nicht nur durch ausgefeilte Zugangs- und Zugriffsschutzmechanismen, wie sie im vorhergehenden Kapitel beschrieben wurden, sondern garantiert Sicherheit und Konsistenz selbst dann, wenn Fehler und Störungen auftreten: Die Auswirkung von Fehlern im Anwendungsprogramm bleibt lokal, Benutzerfehler werden abgefangen. Interne Prüfroutinen erkennen Systemfehler oder Inkonsistenzen und reagieren automatisch.

Selbst wenn der Serverrechner oder das Betriebssystem ausfallen, gehen keine Daten verloren. Bei schwerwiegenden Fehlern beendet sich die UTM-Anwendung, bevor weitere Schäden entstehen können. Universale Wiederanlauf-Funktionen (universal restart) stellen sicher, daß nach erneutem Start alle Terminals und Client-Programme ihre Arbeit mit konsistenten Daten fortsetzen können. Näheres siehe Abschnitt 10.5 auf Seite 135.

Damit ist *open*UTM voll **cluster-fähig**, d.h. die Verarbeitung kann nach dem Wiederanlauf auf einem anderen System fortgesetzt werden, siehe auch die Abschnitte zum Thema „Hochverfügbarkeit“ auf Seite 156, Seite 169 und Seite 180.

10.1 Eingrenzung von Teilprogramm- und Formatierungsfehlern

Bei Fehlern in Teilprogrammen oder bei Formatierungsfehlern sorgt *open*UTM dafür, daß die Auswirkungen begrenzt bleiben: Andere Transaktionen oder die Arbeit der gesamten Anwendung werden nicht beeinträchtigt.

Dafür stellt *open*UTM die folgenden Funktionen zur Verfügung:

- Bei leichteren Fehlern und Störungen gibt *open*UTM Returncodes zurück, die den Fehler beschreiben. Das betroffene Teilprogramm hat so die Möglichkeit, auf den Fehler oder die Störung entsprechend zu reagieren.
- Da UTM-Anwendungen parallel mit mehreren Prozessen arbeiten, kann selbst ein gravierender Fehler in einem Anwendungsprogramm höchstens zum Abbruch des betroffenen Prozesses führen. Es gibt also in UTM-Anwendungen keinen „Single Point of Failure“. Ein abgebrochener Prozeß wird automatisch neu gestartet. Da alle offenen Transaktionen - auch die offenen Transaktionen externer Resource Manager - koordiniert zurückgesetzt werden, bleibt die Datenkonsistenz anwendungsübergreifend erhalten.

- Jedem Prozeß einer UTM-Anwendung wird automatisch ein eigener prozeßlokaler Speicherbereich zugeordnet, in dem die für einen Auftrag bzw. einen Programmlauf aktuellen Systemdaten gespeichert werden. Der Datenbereich ist nur von dem eigenen Prozeß unter Kontrolle des UTM-Systemcodes erreichbar. Das heißt, die Prozesse einer UTM-Anwendung sind voreinander geschützt.
- Falls aufgrund eines gravierenden Fehlers Dialog-Services abgebrochen werden, informiert *open*UTM den Client über den Fehler. Der Client kann danach weitere Services starten. Beim Abbruch eines Asynchron-Service wird der Auftrag aus der Queue entfernt, so daß nachfolgende Aufträge gestartet werden können. Falls gewünscht, kann durch Quittungsaufträge auch beim Abbruch von Asynchron-Services der Auftraggeber über den Fehler informiert werden.
- Wenn während der Formatierung von Nachrichten Fehler auftreten, erkennt *open*UTM dies und reagiert entsprechend:
  - Dialog-Services werden abgebrochen. Der betroffene Client wird durch eine Meldung informiert und kann danach andere Services starten.
  - Bei Asynchron-Services entfernt *open*UTM nach Abbruch des Service den Auftrag aus der Queue. Nachfolgende Aufträge können so ungehindert gestartet werden.
- Jedes UTM-Teilprogramm muß mit einem speziellen UTM-Aufruf verlassen werden. Wenn Sie mit der KDCS-Programmschnittstelle arbeiten, ist dies der PEND-Aufruf. Damit ist u.a. gewährleistet, daß lokale Speicherbereiche freigegeben werden. Wird ein Fehlerfall nicht schon im Teilprogramm abgefangen (bei KDCS mit PEND ER), setzt *open*UTM intern einen PEND ER-Aufruf ab: Der Prozeß wird in jedem Fall „geordnet“ abgebrochen. Zugleich wird in diesem Fall zur Diagnose ein Dump erzeugt.
- Die KDCS-Speicherbereiche KBPROG und SPAB können am Ende eines Dialogschritts mit einem beliebigen, per Generierung festzulegenden Zeichen beschrieben werden. Zusätzlich wird von *open*UTM überprüft, ob in einem Teilprogramm ein größerer Speicherbereich (KBPROG / SPAB) angefordert wird, als bei der Generierung angefordert.

10.2 Automatische Prüfungen

Erkennen von Benutzerfehlern

Falls ein Terminal-Benutzer ein Benutzerkommando eingibt, das zum Eingabezeitpunkt nicht erlaubt ist, reagiert *open*UTM mit einer entsprechenden Meldung.

Wenn ein Terminal-Benutzer beim Ausfüllen eines Eingabe-Formats unzulässige Tasten verwendet, erkennt *open*UTM dies und gibt das fehlerhaft ausgefüllte Eingabe-Format erneut aus. Der Terminal-Benutzer hat so die Möglichkeit, die fehlerhafte Eingabe zu korrigieren.

Konsistenzprüfungen beim Anwendungsstart

*open*UTM führt beim Anwendungsstart Konsistenzprüfungen durch. Falls z.B. versucht wird, eine Anwendung mit Komponenten unterschiedlicher UTM-Versionen zu starten, oder falls die Konfigurationsdateien (KDCFILE) nicht zusammenpassen oder in sich inkonsistent sind, bricht *open*UTM den Start ab und gibt eine Meldung aus, die die Art des Fehlers beschreibt.

Interne Prüfroutinen

Im UTM-Code sind eine Reihe interner Prüfroutinen implementiert. Werden Fehler oder Inkonsistenzen entdeckt, durch welche die Anwendungsdaten verfälscht werden könnten, beendet *open*UTM sofort die UTM-Anwendung, schließt alle offenen Dateien und erzeugt für jeden Prozeß der Anwendung einen spezifischen UTM-Dump. Danach kann die UTM-Anwendung - mit logisch konsistenten Daten - wieder gestartet werden. Beim automatischen Wiederanlauf wird jedem einzelnen Client mitgeteilt, wie weit seine Aufträge bearbeitet worden sind.

10.3 Schutz bei Störungen oder Ausfall lokaler Betriebsmittel

*open*UTM gewährleistet, daß selbst beim Ausfall lokaler Betriebsmittel keine Daten verlorengehen. *open*UTM bietet z.B. Schutz bei den in den folgenden Abschnitten geschilderten Problemen.

Betriebssystemfehler

Meldet eine Betriebssystemfunktion einen Fehler an *open*UTM, dann wird dieser Fehler wie ein interner Fehler von *open*UTM behandelt, d.h. *open*UTM beendet - bevor Schäden entstehen können - sofort die Anwendung mit einem entsprechenden Fehlercode.

Plattenausfall

Um eine erhöhte Datensicherheit zu erreichen, ist es möglich, die KDCFILE doppelt, auf unterschiedlichen Laufwerken zu führen (hot standby). Die Zeiten für Ein-/Ausgabe-Operationen erhöhen sich bei doppelter KDCFILE-Führung nur unwesentlich (sie verdoppeln sich keinesfalls) - die Performance wird kaum beeinträchtigt.

Hardwarefehler bei Terminals

Fällt ein Terminal aus, so kann der Benutzer an ein anderes Terminal wechseln, sich dort erneut unter seiner Benutzerkennung anmelden, und den begonnenen Service fortsetzen.

Selbst wenn die Anwendung ohne Benutzerkennungen arbeitet, kann ein Terminal-Benutzer nach Ausfall seines Terminals die Arbeit an einem anderen Terminal nahtlos fortsetzen. Allerdings muß in diesem Fall dem Benutzer per Administration ein anderes Terminal zugewiesen werden.

Falls Asynchron-Nachrichten aufgrund eines defekten Terminals nicht von diesem empfangen werden, kann per Administration dem betreffenden logischen Anschlußpunkt (LTERM-Partner) ein intaktes Terminal zugewiesen werden. Die Asynchron-Nachrichten werden anschließend auf dieses Terminal ausgegeben.

Netzausfall oder schwerwiegende Netz-Störungen

Bei Netzausfall gehen die betroffenen Netzverbindungen zu openUTM verloren.

Verbindungsverlust bei Terminals

Wenn das (Teil-)Netz wieder zur Verfügung steht, können sich Terminal-Benutzer erneut bei der UTM-Anwendung anmelden und ihre Arbeit fortsetzen.

Verbindungsverlust bei Druckern

Ein Netzausfall oder eine schwerwiegende Netzstörung kann einen Verbindungsverlust zu Druckern zur Folge haben. Ein weiterer Grund für einen solchen Verbindungsverlust kann das Ausbleiben einer logischen Abdruckquittung sein: Logische Abdruckquittungen verwendet *open*UTM, um das Drucken der Nachrichten zu überwachen. Falls eine angeforderte Abdruckquittung nicht in einem festgelegten Zeitraum eintrifft, baut *open*UTM die Verbindung zu diesem Drucker ab.

In allen diesen Fällen versucht *open*UTM, die Verbindung wieder aufzubauen. Der Aufbauversuch wird in einem festgelegten Zeitabstand wiederholt. Kommt längere Zeit keine Verbindung zustande, hat der Administrator die Möglichkeit, einen anderen Drucker zuzuordnen. Bei einem Druckerbündel werden in solchen Fällen die Nachrichten automatisch auf einem anderen Drucker ausgegeben.

In folgenden Situationen wird nach Wiederaufbau der Verbindung der gesamte Druckvorgang wiederholt:

– Die zu druckende Nachricht wurde vollständig gesendet, aber im festgelegten Zeitraum kam keine Abdruckquittung zurück.

– Mehrere Teilnachrichten einer Gesamtnachricht wurden bereits gesendet, die Gesamtnachricht aber noch nicht vollständig gedruckt.

Um auf die hier gegebene Möglichkeit einer doppelten Ausgabe dieser Nachricht hinzuweisen, sendet *open*UTM bei einem erneuten Ausgabeversuch eine entsprechende Meldung.

Störungen, die nicht zum Verbindungsverlust führen

Gestörte Nachrichten können sowohl bei der Ein- als auch bei der Ausgabe von Nachrichten auftreten. Ursache können entweder Hardwarefehler der Terminals bzw. Drucker oder Netzstörungen sein. Bei der Eingabe werden solche Fehler vom Formatierungssystem erkannt. *open*UTM gibt in diesen Fällen die letzte Ausgabenachricht nochmals aus, so daß der Terminal-Benutzer seine Eingabe wiederholen kann. Wenn der Terminal-Benutzer gestörte Ausgabenachrichten erkennt (z.B. Schmierzeichen), kann er sich die letzte Ausgabe nochmals ausgeben lassen (Benutzerkommando KDCDISP).

Wird bei Druckern die Störung erkannt, bevor bei *open*UTM der Druckvorgang für die gesamte Drucknachricht abgeschlossen ist, wird die Verbindung von *open*UTM abgebaut. Nach erneutem Verbindungsaufbau wird die Nachricht nochmals ausgegeben.

10.4 Abnormale Beendigung einer UTM-Anwendung

Wie in den vorangegangenen Abschnitten beschrieben, wird in bestimmten Ausnahmesituationen der Lauf einer UTM-Anwendung durch *open*UTM abgebrochen, um die Sicherheit von Programmen und Daten nicht zu gefährden (abnormale Beendigung). Beim nächsten Start führt *open*UTM automatisch einen Wiederanlauf durch (siehe Abschnitt 10.5 auf Seite 135).

Wird eine UTM-Anwendung abnormal beendet, führt *open*UTM - soweit möglich - vorher noch folgende Aktionen durch:

– alle Verbindungen zu den Kommunikationspartnern der UTM-Anwendung abbauen

– Verbindungen zu externen Resource Managern (z.B. zu Datenbanksystemen) abbauen

– alle Dateien schließen

– für jeden Prozeß der UTM-Anwendung einen UTM-Dump zur Fehleranalyse erzeugen

– alle Prozesse beenden

Aus Sicherheitsgründen wird bei einer abnormalen Beendigung einer UTM-Anwendung nicht mehr versucht, die KDCFILE, welche alle für den Ablauf einer UTM-Anwendung notwendigen Daten enthält, wieder in einen konsistenten Zustand zu bringen. Dies geschieht erst während des erneuten Starts.

10.5 Wiederanlauf-Funktionen von *open*UTM

Die Wiederanlauf-Funktionen von *open*UTM stellen sicher, daß Services und Aufträge, die durch Störungen oder Ausfälle unterbrochen wurden, unmittelbar und mit konsistenten Daten fortgesetzt oder wiederholt werden können. Dabei ist es gleichgültig, ob es sich um den Abbruch einzelner Services handelt, z.B. infolge Verbindungsverlust, oder um die abnormale Beendigung der gesamten UTM-Anwendung, z.B. verursacht durch einen Ausfall des Server-Rechners.

Als Voraussetzung hierfür schreibt *open*UTM während des Anwendungslaufs Sätze mit Wiederanlauf-Informationen in die KDCFILE (Transaktions-Logging). *open*UTM bietet hierzu zwei Varianten, die sich in ihrem Wiederanlaufverhalten unterscheiden.

10.5.1 Die Varianten UTM-S und UTM-F

*open*UTM bietet für den Betrieb einer Anwendung die beiden Varianten UTM-S und UTM-F, die sich wie folgt charakterisieren lassen.

– **UTM-S** (UTM-Secure): Hohe Sicherheit durch universelle Wiederanlauf-Funktionen

– **UTM-F** (UTM-Fast): Hohe Performance bei eingeschränkten Wiederanlauf-Funktionen

UTM-S arbeitet mit einem umfassenden Transaktions-Logging aller Benutzer- und Verwaltungsdaten. UTM-F führt das Transaktions-Logging dagegen nur für Verwaltungsdaten durch und benötigt dadurch weniger I/Os. Deshalb hat UTM-F eine etwas bessere Performance, dafür aber nicht die volle Wiederanlauffähigkeit. Bezüglich der Verwaltungsdaten unterscheiden sich die beiden Varianten nur geringfügig, Einzelheiten hierzu sind im Handbuch „Anwendungen administrieren“ bei den zugehörigen Administrations-Aufrufen beschrieben.

Mit welcher Variante eine UTM-Anwendung betrieben wird, legen Sie bei der Generierung der UTM-Anwendung fest.

Im laufenden Betrieb verhalten sich die Varianten funktionell gleich. Auch UTM-F führt die Transaktionsverarbeitung nach dem „Alles-oder-Nichts“-Prinzip durch und gewährleistet die Konsistenz der Daten bei der Zusammenarbeit mit *einem* Datenbanksystem.

Unterschiede zwischen den beiden Varianten werden erst nach Ende des Anwendungslaufs (normal oder durch Anwendungsabbruch) sichtbar, da UTM-F keine Benutzerdaten für den folgenden Anwendungswiederanlauf sichert. Wie sich die beiden Varianten beim Wiederanlauf verhalten, ist in den folgenden Abschnitten genauer beschrieben.

10.5.2 Anwendungswiederanlauf mit UTM-S

Wird nach abnormaler Beendigung einer UTM-S-Anwendung (UTM-Secure) die Anwendung erneut gestartet, stellt *open*UTM innerhalb kürzester Zeit die Einsatzbereitschaft wieder her: Alle offenen Transaktionen werden auf einen konsistenten Stand gesetzt, unterbrochene Services werden erneut gestartet, die Benutzer können mit dem Bildschirm weiterarbeiten, der zu diesem konsistenten Stand gehört (Bildschirmwiederanlauf).

Um diesen umfassenden Wiederanlauf zu ermöglichen, schreibt *open*UTM während des Betriebs einer UTM-S-Anwendung bei *jeder* Transaktion einen Satz mit Wiederanlauf-Informationen in die KDCFILE.

Konsistenz bei unterbrochenen Transaktionen

*open*UTM setzt beim Wiederanlauf nicht einfach alle offenen Transaktionen zurück, sondern geht differenzierter vor:

- Transaktionen, die sich zum Zeitpunkt des Ausfalls in der Transaktionsende-Bearbeitung befanden, werden durch *open*UTM abhängig davon behandelt, ob die beteiligten Transaktionen externer Resource Manager / Datenbanksysteme zum Zeitpunkt des Ausfalls beendet waren oder nicht.
  - Waren die Transaktionen der Resource Manager nicht beendet und wurden sie beim Wiederanlauf der Resource Manager zurückgesetzt, so setzt *open*UTM auch die UTM-Transaktion zurück (Rollback).
  - Waren die Transaktionen der Resource Manager bereits beendet, so beendet auch *open*UTM die zum Zeitpunkt des Ausfalls offenen UTM-Transaktionen (Commit).

  Entsprechend verfährt *open*UTM bei verteilten Transaktionen.
- Alle Transaktionen, die sich zum Zeitpunkt des Ausfalls noch nicht in der Transaktionsende-Bearbeitung befanden, werden zurückgesetzt.

Wiederanlauf bei Dialog-Services

Unterbrochene Dialog-Services werden nach einem Wiederanlauf fortgesetzt, falls sie vor der Unterbrechung bereits einen (vor dem Service-Ende liegenden) Sicherungspunkt erreicht hatten. Beim Wiederanlauf entscheidet *open*UTM, ob es notwendig ist, den Benutzer (Terminalbenutzer oder Client-Programm) über den Wiederanlauf in Kenntnis zu setzen:

- Dies ist nicht notwendig, wenn ausgeschlossen werden kann, daß seit dem letzten Sicherungspunkt eine Interaktion mit dem Benutzer stattgefunden hat. Daher ist es z.B. möglich, daß ein Client-Programm den Service-Wiederanlauf gar nicht bemerkt.
- Falls eine Benutzer-Interaktion stattgefunden hatte, muß der Benutzer nach Wiederanlauf u.U. die seit dem letzten Sicherungspunkt gemachten Eingaben wiederholen.

Deshalb informiert *open*UTM in diesen Fällen Terminal-Benutzer durch entsprechende UTM-Meldungen. Client-Programme werden in solchen Fällen automatisch diskonnektiert und können nach erneutem Verbindungsaufbau den Wiederanlauf explizit anfordern.

Detaillierte Informationen zum Wiederanlauf beim Anschluß von Client-Programmen finden Sie in den *open*UTM-Client-Handbüchern.

Wiederanlauf bei Hintergrund- und Ausgabe-Aufträgen

Der transaktionsorientierte Queuing-Mechanismus von *open*UTM gewährleistet, daß auch bei einem Systemausfall alle Hintergrund- und Ausgabe-Aufträge erhalten bleiben, die *open*UTM zur Bearbeitung angenommen und bis dahin noch nicht vollständig abgearbeitet hat.

Beim Wiederanlauf geht *open*UTM differenziert vor:

- Hintergrund-Aufträge, mit deren Bearbeitung noch nicht begonnen wurde oder die noch keinen Sicherungspunkt erreicht hatten, werden nach dem Wiederanlauf neu gestartet.
- Bei Hintergrund-Aufträgen, die bereits einen Sicherungspunkt erreicht hatten, wird die Bearbeitung an diesem Sicherungspunkt fortgesetzt.
- Alle Ausgabe-Aufträge, mit deren Bearbeitung noch nicht begonnen wurde oder die zum Abbruchzeitpunkt noch nicht vollständig abgearbeitet waren, stehen nach dem Wiederanlauf erneut zur Ausgabe an.

10.5.3 Anwendungswiederanlauf mit UTM-F

Beim Betrieb einer UTM-F-Anwendung sichert *open*UTM einen Großteil der administrativen Änderungen in den Verwaltungsdaten für den Wiederanlauf. Dazu gehören z.B. geänderte Paßwörter, neu eingetragene Benutzer oder das Sperren von TACs, siehe auch Handbuch „Anwendungen administrieren“. Diese Änderungen stehen also auch bei einer UTM-F-Anwendung nach einen Neustart zur Verfügung.

Dagegen werden bei UTM-F keine Benutzerdaten auf KDCFILE gesichert. Dementsprechend werden bei UTM-F sowohl bei normaler als auch abnormaler Beendigung der Anwendung alle Benutzerdaten „vergessen“. Dazu gehören z.B. Sekundärspeicherbereiche wie GSSBs, Informationen über offene Dialoge oder Hintergrund-Aufträge.

Daher läßt sich bei einer UTM-F-Anwendung die Datenkonsistenz nach dem Wiederanlauf nur dann sicherstellen, wenn man ausschließlich lokale Transaktionen verwendet und wenn sämtliche Benutzerdaten in einer einzigen Datenbank gehalten werden.

10.6 Fehlerbehandlung bei verteilter Verarbeitung

Bei der verteilten Verarbeitung kann es folgende Fehlersituationen geben:

– Fehler in einer der Anwendungen
– Verbindungsverlust zwischen Anwendungen
– Beendigung einer der beteiligten Anwendungen bei offenen Transaktionen

*open*UTM reagiert auf diese Fehler mit Meldungen, die auf die Ursache des Fehlers hinweisen. Bei Fehlern innerhalb einer Anwendung werden Fehlercodes und meist auch ein Dump erzeugt. Die Partneranwendung wird so bald wie möglich über den Fehler informiert.

Das Rollback- und Wiederanlaufverhalten unterscheidet sich, je nachdem ob mit globaler Transaktionssicherung oder mit unabhängigen, lokalen Transaktionen gearbeitet wird.

10.6.1 Rollback und Wiederanlauf bei globaler Transaktionssicherung

Verteilte Verarbeitung mit globaler Transaktionssicherung kann über die Protokolle LU6.1 oder OSI TP betrieben werden. Bei LU6.1 arbeitet *open*UTM grundsätzlich mit globaler Transaktionssicherung, während bei OSI TP die globale Transaktionssicherung nur gilt, wenn die Funktionseinheit „Commit“ gewählt ist.

Rücksetzen von globalen Transaktionen

*open*UTM kann eine verteilte Transaktion aus verschiedenen Gründen zurücksetzen. Dazu zählen z.B. Verstöße gegen Programmierregeln, Verbindungsverlust, Zeitüberschreitung (Timeout) bei der Überwachung der Antwortzeit oder Beendigung einer Anwendung.

Je nach Rücksetzgrund werden entweder Auftraggeber- und/oder Auftragnehmer-Service beendet, weil eine Fortsetzung nicht möglich oder nicht sinnvoll ist, oder Auftraggeber- und Auftragnehmer-Service werden auf den letzten Sicherungspunkt zurückgesetzt und die Kommunikation wird wieder aufgenommen (Service-Wiederanlauf).

Detaillierte Informationen hierzu finden Sie im *open*UTM-Handbuch „Anwendungen programmieren mit KDCS für COBOL, C und C++“.

Verbindungsverlust

Ein Verbindungsverlust zwischen zwei Anwendungen kann auftreten bei Störung der Übertragungsstrecke oder durch Beenden einer der beteiligten Anwendungen. Der Verlust der Verbindung zwischen den Partnern führt zum Rücksetzen der Transaktion.

Bei Kommunikation über das OSI TP Protokoll wird bei Verbindungsverlust der Auftragnehmer-Service in jedem Fall beendet. Bei Nutzung des LU6.1-Protokolls wird der Auftragnehmer-Service nur dann beendet, wenn er noch keinen Sicherungspunkt erreicht hat.

Wird der Auftragnehmer-Service beendet, erhält der Auftraggeber-Service eine Fehlernachricht.

Hat der Auftraggeber-Service noch keinen Sicherungspunkt erreicht, so wird er auch beendet. Er kann dann natürlich auch keine Fehlernachricht mehr erhalten. *open*UTM schickt jedoch eine Meldung an das Terminal oder das Client-Programm, das den Auftraggeber-Service gestartet hat.

Wiederanlauf unterbrochener Services

Wird die globale Transaktion zurückgesetzt, z.B. wegen Verbindungsverlust, Timeout oder Beendigung einer Anwendung, ohne daß der Auftraggeber-Service beendet wird, so findet ein Service-Wiederanlauf statt. Dabei wird die Kommunikation zwischen dem Client und dem Auftraggeber-Service sowie (bei LU6.1) zwischen Auftraggeber- und Auftragnehmer-Service am letzten Sicherungspunkt wieder aufgenommen.

Der Wiederanlauf findet sofort statt, wenn der Client an der Auftraggeber-Anwendung angeschlossen bleibt. Ist der Client nach dem Rücksetzen nicht mehr an der Auftraggeber-Anwendung angeschlossen, etwa weil die Verbindung zum Client abgebaut wurde, so findet der Wiederanlauf statt, sobald sich der Client erneut anmeldet.

Sessionwiederanlauf (nur bei Kommunikation über LU6.1)

„Session“ ist die Bezeichnung für eine Kommunikationsbeziehung zwischen Anwendungen über das LU6.1-Protokoll. Jede Transportverbindung zwischen den Anwendungen ist genau einer Session zugeordnet. Das Session-Konzept ermöglicht es, bei unterbrochener Kommunikation Informationen über die zuletzt gesichert gesendeten Nachrichten zu speichern. Dadurch kann eine unterbrochene Kommunikation wieder aufgenommen werden.

Nach einem Verbindungsverlust versucht *open*UTM, die Kommunikation an einem Sicherungspunkt wieder aufzunehmen. Dazu muß zunächst wieder eine Transportverbindung verfügbar sein.

Anlässe für einen Sessionwiederanlauf können sein:

– automatische Wiederaufnahme der Kommunikation beim Wiederanlauf von Auftraggeber- und Auftragnehmer-Service

– Neustart einer Anwendung, wenn für die ferne Anwendung automatischer Verbindungsaufbau generiert ist

– Administrationskommando

– erneutes Senden einer Nachricht über die Session

Der Sessionwiederanlauf kann vom fernen Partner abgelehnt werden, weil er zum betreffenden Zeitpunkt den Sessionwiederanlauf noch nicht bearbeiten kann. Dieser Partner kann dann den Sessionwiederanlauf zu einem späteren Zeitpunkt erneut anstoßen.

10.6.2 Rollback und Wiederanlauf bei unabhängigen Transaktionen

Mit unabhängigen, lokalen Transaktionen arbeitet *open*UTM z.B. dann, wenn bei der Kommunikation direkt auf die Transportschnittstelle aufgesetzt wird, oder wenn bei der Kommunikation über OSI TP auf die Funktionseinheit „Commit“ verzichtet wird.

Bezüglich Rollback und Wiederanlauf lassen sich folgende Fälle unterscheiden:

- Fehler in der Auftragnehmer-Anwendung oder Verbindungsverlust

  *open*UTM setzt die Transaktion im Auftragnehmer-Service zurück und beendet den Service. Die Transaktion im Auftraggeber-Service wird nicht automatisch zurückgesetzt. Der Auftraggeber-Service erhält jedoch eine Fehlernachricht, falls er auf ein Ergebnis vom Auftragnehmer-Service wartet, und kann ebenfalls mit Rücksetzen reagieren.

- Fehler in der Auftraggeber-Anwendung

  *open*UTM setzt die Transaktionen in Auftraggeber- und Auftragnehmer-Service zurück und beendet den Auftragnehmer-Service. Hat der Auftraggeber-Service bereits einen Sicherungspunkt erreicht, wird nach dem Rücksetzen der Transaktion ein Vorgangswiederanlauf durchgeführt.

11 *open*UTM in BS2000/OSD

In diesem Kapitel werden einige plattformspezifische Details beschrieben, die sich speziell auf den Einsatz von *open*UTM in BS2000/OSD beziehen:

- Systemeinbettung
- UTM-Prozesse in BS2000/OSD
- Adreßraumkonzept
- BS2000/OSD-spezifische Funktionen
- Hochverfügbarkeit

11.1 Systemeinbettung

Die Architektur von *open*UTM in BS2000/OSD ist abgestimmt auf die Architektur des BS2000/OSD und des Datenkommunikationssystems TRANSDATA.

UTM-Systemcode

Der Systemcode einer UTM-Anwendung (der Monitor) läuft überwiegend im privilegierten Zustand (TPR) und nutzt zentrale BS2000/OSD- und TRANSDATA-Funktionen.

Der Systemcode unterliegt dadurch dem Schutz des Betriebssystems und ist u.a. gegen das Überschreiben durch Teilprogramme gesichert.

Der Systemcode von *open*UTM ist als eigenes Subsystem UTM des Betriebssystems realisiert. Der Systemverwalter kann das Subsystem mit DSSM in den Systemspeicher laden.

Der UTM-Systemcode ruft die Funktionen des Betriebssystems über ein Anpaßmodul auf. In einer Version von *open*UTM sind Anpaßmodule für mehrere BS2000/OSD-Versionen verfügbar: Beim Start des Subsystems wird das passende Modul geladen. Mit dieser Technik ist eine *open*UTM-Version in mehreren BS2000/OSD-Versionen ablauffähig.
Dies ermöglicht es Ihnen, mit Ihren UTM-Anwendungen ohne Umstellungen in eine höhere BS2000/OSD-Version zu migrieren.

Existieren in einem BS2000/OSD-System mehrere UTM-Anwendungen, ist der UTM-Systemcode nur einmal im BS2000 geladen. Der gesamte UTM-Systemcode wird als ein Subsystem des BS2000 geladen. Der Systemverwalter legt durch Anweisungen für das Subsystem *open*UTM fest, zu welchem Zeitpunkt der UTM-Systemcode geladen werden soll. Werden weitere UTM-Anwendungen in BS2000/OSD gestartet, verwenden diese ebenfalls den zuvor einmal geladenen Systemcode. Jede UTM-Anwendung wird allerdings von *open*UTM über eigene Anwendungstabellen verwaltet, so daß die UTM-Anwendungen völlig unabhängig voneinander ablaufen.

Vom UTM-Systemcode benutzte Subsysteme und Systemfunktionen

Der UTM-Systemcode nutzt eine Reihe systeminterner Schnittstellen und Subsysteme:

- Funktionen zum Anfordern und Freigeben der systeminternen Speicherbereiche (Memory Management)
- Funktionen zur Auftragsverwaltung und zur Serialisierung (Börsen)
- Funktionen zur Verwaltung von Ressourcen einer Anwendung (Name Manager)
- Timer-Funktionen zur Überwachung von Betriebsmittelbelegungen und zeitgesteuerten Nachrichten
- Funktionen der BS2000-Subsystemverwaltung (DSSM) zum Laden und Entladen des UTM-Systemcodes
- Datenverwaltungssystem mit den Zugriffsmethoden UPAM und SAM
- BCAM zur Kommunikation mit Anwendungen und Kommunikationspartnern
- VTSU zur Aufbereitung der Nachrichten von und zu Terminals
- RSO (Remote SPOOL Output) zur Unterstützung von RSO-Druckern (siehe Seite 151)
- OSS und CMX bei verteilter Verarbeitung über OSI-TP
- SAT zur Protokollierung sicherheitsrelevanter UTM-Ereignisse.

Ablauffähigkeit auf Hardware-Plattform SR2000

*open*UTM V5.0 ist im vollen Umfang auf den RISC-basierten Business Servern der Systemlinie SR2000 ablauffähig. Nähere Informationen zur Installation von *open*UTM auf dieser Hardware finden Sie im Anhang des *open*UTM-Handbuchs „Anwendungen generieren und betreiben (BS2000/OSD)“.

Paralleler Betrieb mehrerer *open*UTM-Versionen

Es ist möglich, im selben BS2000/OSD-System mehrere *open*UTM-Versionen nebeneinander zu laden und gleichzeitig einzusetzen. Der Parallelbetrieb ist für *open*UTM-Versionen ab V3.2A möglich.

Diese Funktion ist besonders beim Übergang auf eine neue *open*UTM-Version von Vorteil. Sie können dann während der Nutzung einer UTM-Version einzelne UTM-Anwendungen versuchsweise mit der *open*UTM-Folgeversion ablaufen lassen. Die Umstellung mehrerer UTM-Anwendungen von einer *open*UTM-Version auf eine andere innerhalb eines Rechners kann damit schrittweise getestet und durchgeführt werden.

Von der Main Routine benutzte Schnittstellen und Komponenten

Neben den systeminternen Schnittstellen hat eine UTM-Anwendung im nicht-privilegierten Funktionszustand (TU) Schnittstellen zum Format-Handling-System FHS, zu externen Resource Managern wie Datenhaltungs- oder Datenbanksystemen und zu den Laufzeitsystemen der verwendeten Programmiersprachen. Die Anbindungen an diese Produkte sind ebenfalls technisch entkoppelt und werden über neutrale Schnittstellen realisiert:

IUTMDB	Schnittstelle für die koordinierte Zusammenarbeit mit externen Resource Managern, wie Datenhaltungs- oder Datenbanksystemen. Über diese Schnittstelle werden alle Resource Manager, die über ein IUTMDB-Anschlußmodul verfügen, in einer einheitlichen Weise bedient (z.B. LEASY, SESAM, PRISMA, UDS oder ORACLE).
IUTMFORM	Als Schnittstelle zu Formatierungssystemen, zur Zeit nur FHS.
VTSU	Für den Aufbau der terminalspezifischen Nachrichten benutzt *open*UTM neben FHS die Dienste der Komponente VTSU (Virtual Terminal Support).
IUTMHLL	Einheitliche Schnittstelle zu den Laufzeitsystemen der einzelnen Programmiersprachen.
ILCS	(Inter Language Communication Services) Schnittstelle zum sprachunabhängigen Laufzeitsystem CRTE (Common Runtime Environment).

Die Teilprogramme nutzen die Funktionen über die Programm-Schnittstellen von *open*UTM, also über die X/Open-Schnittstellen CPI-C und XATMI oder über die Schnittstelle KDCS (nationaler Standard).

Übersicht: Schnittstellen von *open*UTM

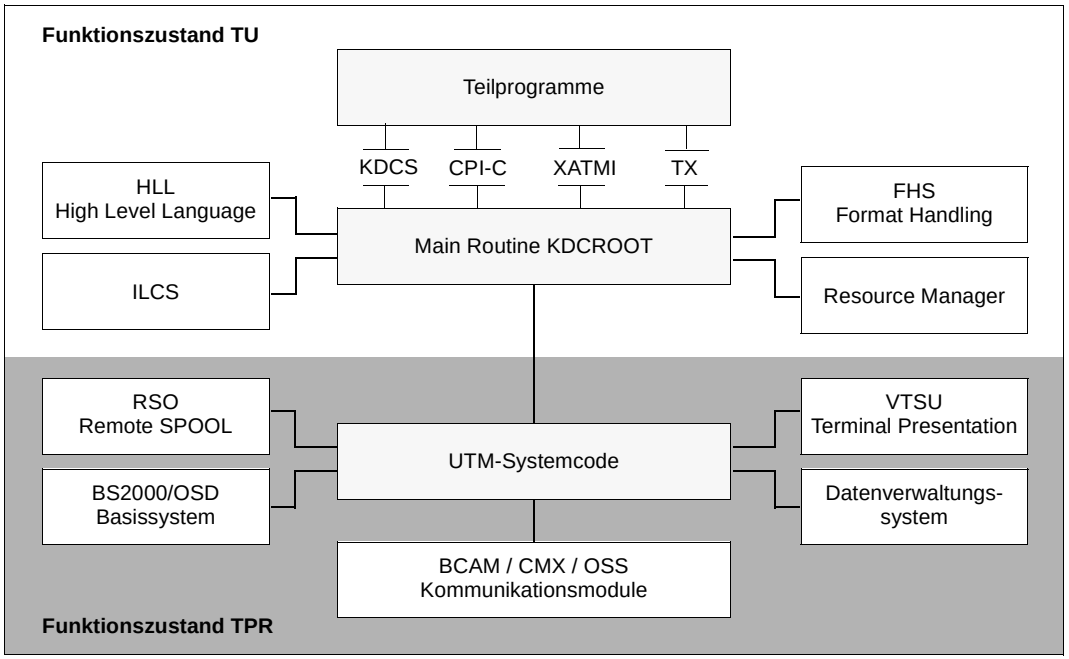


Bild 29: Schnittstellen von *open*UTM zu anderen Systemkomponenten

11.2 Prozesse in BS2000/OSD

In einer UTM-Anwendung, die in BS2000/OSD abläuft, gibt es nicht wie bei UTM-Anwendungen in UNIX unterschiedliche Prozesse mit unterschiedlichen Aufgaben, sondern nur Prozesse eines Prozeßtyps. Diese Prozesse sind als BS2000/OSD-Tasks realisiert und entsprechen ihrer Aufgabe nach den UNIX-Workprozessen. Die Aufgaben, die in UNIX von den übrigen Prozeßtypen übernommen werden, sind in BS2000/OSD von den Basisfunktionen des Betriebssystem abgedeckt.

Das gebundene Anwendungsprogramm wird über eine BS2000-Prozedur als Batch Job gestartet. Der so gestartete Prozeß ist der erste Prozeß der UTM-Anwendung, der zunächst die Anwendung einrichtet. Danach aktiviert dieser Prozeß soviele Folgeprozesse, wie bei der Generierung der Anwendung bzw. in der Startprozedur angegeben wurde. Die Folgeprozesse schließen sich an die bestehende Anwendung an.

Nach Anwendungsstart warten alle Prozesse der UTM-Anwendung in einer gemeinsamen Prozeßwarteschlange auf Aufträge. Trifft ein Auftrag ein, so wird er einem wartenden Prozeß in der Prozeßwarteschlange zugeordnet. Dieser Prozeß bearbeitet den Auftrag und reiht sich anschließend wieder in die Prozeßwarteschlange ein.

Sind zu einer Zeit mehr Aufträge als Workprozesse vorhanden, dann wird eine Auftragswarteschlange aufgebaut. Sowohl Auftrags- als auch Prozeßwarteschlangen sind anwendungsbezogen; das bedeutet, daß unterschiedliche Anwendungen auch jeweils eine eigene Prozeß- und Auftragswarteschlange haben.

11.3 Adreßraumkonzept

Die von einer UTM-Anwendung benutzten Speicherbereiche sind je nach Art der Daten in unterschiedlichen Speicherklassen des BS2000/OSD abgelegt.

- Der UTM-Systemcode liegt im Klasse-4-Speicher und kann von allen Prozessen aller UTM-Anwendungen eines BS2000/OSD-Systems gemeinsam benutzt werden.
- Anwendungslokale Systemspeicherbereiche mit den Konfigurationsdaten, den Verwaltungsdaten und einem Pufferbereich zur Reduzierung der Dateizugriffe (Cache) werden in Common Memory Pools im Klasse-5-Speicher abgelegt. Die Systemspeicherbereiche einer UTM-Anwendung sind nur von den Prozessen der eigenen Anwendung und nicht von Prozessen fremder UTM-Anwendungen erreichbar.
- Jedem Prozeß einer UTM-Anwendung ist ein eigener prozeßlokaler Klasse-5-Speicherbereich zugeordnet, in dem die für einen Auftrag bzw. einen Programmlauf aktuellen Systemdaten gespeichert werden. Auf diesen Datenbereich kann nur von einem Prozeß und ausschließlich von den UTM-Systemfunktionen zugegriffen werden. Diese Daten sind von anderen Prozessen auch innerhalb der eigenen Anwendung nicht erreichbar, d.h. auch die Prozesse innerhalb derselben Anwendung sind voreinander geschützt. Benutzerprogramme können den Bereich nicht beschreiben und so die Abläufe der UTM-Systemfunktionen nicht stören.
- Jedem Prozeß einer UTM-Anwendung ist vom Betriebssystem ein Benutzeradreßraum im Klasse-6-Speicher zugeordnet. Dieser Benutzeradreßraum kann je nach Verwendung prozeßlokal sein oder als Common Memory Pool gemeinsam von allen Prozessen einer Anwendung (anwendungslokal) oder auch von Prozessen mehrerer Anwendungen (anwendungsglobal) benutzt werden. Er enthält in der Regel:
  - Die Main Routine KDCROOT, die den Kontakt zwischen Teilprogrammen und den UTM-Systemfunktionen herstellt, sowie Daten- und Pufferbereiche, die gemeinsam von den UTM-Systemfunktionen und dem Anschlußprogramm benutzt werden. Die Main Routine wird bei der Anwendungsgenerierung anwendungsspezifisch parametrisiert.
  - Die Teilprogramme und deren Datenbereiche. Falls Teilprogramme mehrfach benutzbar (shareable) sind, können sie in gemeinsamen Benutzerspeichern anwendungslokal oder anwendungsglobal in Common Memory Pools bzw. nichtprivilegierten Subsystemen abgelegt werden. Dadurch wird der von den Prozessen der Anwendung belegte externe Speicher (paging area) kleiner und die Anzahl der Seitenwechsel (paging) wird reduziert, was sich günstig auf die Performance auswirkt.
  - Die Formatierungsroutine zur Aufbereitung von Bildschirmmasken und Druckerformularen sowie die zur Anwendung gehörenden Formatdefinitionen. Wird das Formatierungssystem FHS (Format Handling System) eingesetzt, so können diese Teile ebenfalls shareable in Common Memory Pools abgelegt werden.

– Dateisysteme (wie z.B. LEASY) mit zugehörigem Anschlußmodul oder Anschlußmodule für die Datenbanksysteme wie z.B. UDS und SESAM.
– Laufzeitsysteme der Programmiersprachen, in denen die Teilprogramme codiert sind.
– Administrationsteilprogramme für die UTM-Anwendung.

Das folgende Bild gibt eine Übersicht über die Speicherstruktur einer UTM-Anwendung in BS2000/OSD.

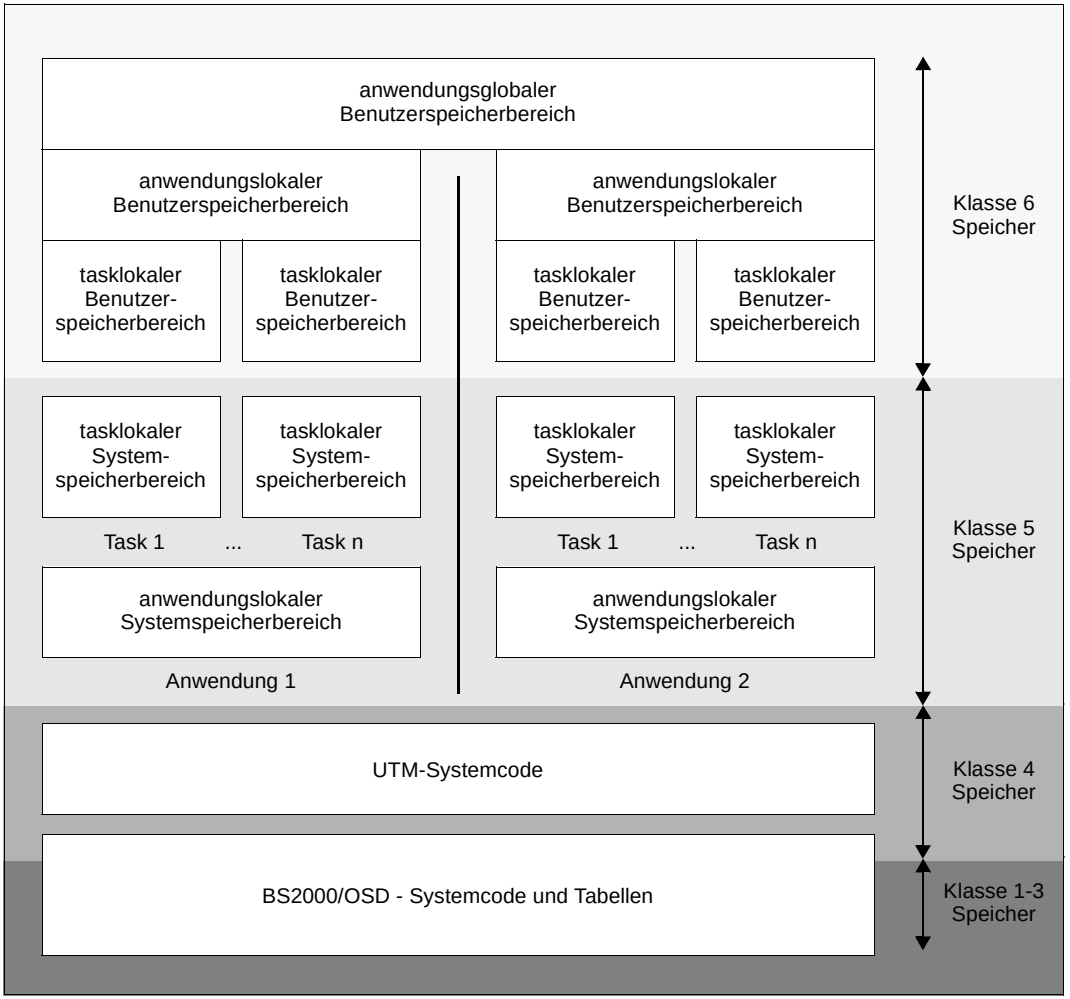


Bild 30: Speicherstruktur von UTM-Anwendungen in BS2000/OSD

11.4 Formatierung

Falls Sie für Ihre UTM-Anwendung Terminals im Formatmodus einsetzen wollen, können Sie entweder die BS2000/OSD-Softwareprodukte IFG (Interaktiver Formatgenerator) und FHS (Format Handling System) nutzen oder eine eigene Formatierungsroutine erstellen (Event-Exit FORMAT). *open*UTM (BS2000/OSD) unterscheidet die Formattypen *Formate, +Formate, #Formate und -Formate mit jeweils unterschiedlichen Funktionen und Einsatzmöglichkeiten. Der Formattyp ist am ersten Zeichen des Formatnamens zu erkennen und wird beim Übertragen einer Nachricht mitgeliefert. Für die Formatierung von *, + und #Formaten verwendet *open*UTM die Formatsteuerung FHS, für die Formatierung von -Formaten den Event-Exit FORMAT.

Formatgenerierung mit IFG

Mit dem interaktiven Formatgenerator IFG lassen sich im geführten Dialog schnell und einfach Formate erstellen. IFG erzeugt dabei automatisch entsprechende Datenstrukturen (=Adressierungshilfen), die Sie in Ihre Teilprogramme integrieren können. IFG unterstützt Sie auch bei der Verwaltung und Wartung Ihrer Format-Bibliotheken.

Der interaktive Formatgenerator IFG ist in einem eigenen Handbuch mit dem Titel „IFG“ beschrieben.

Formatsteuerung mit FHS

Die Formatsteuerung FHS (Format Handling System) unterstützt den Einsatz von Formaten, die mit IFG erstellt wurden. Sie bietet eine Vielzahl von Funktionen wie:

- Auffüllen der Nachrichtenbereiche mit frei wählbaren Zeichen
- Kennzeichnen der vom Terminal-Benutzer ausgewählten Felder
- Übertragen der ungeschützten Felder
- Übertragen nur der vom Terminal-Benutzer modifizierten Felder
- Cursor positionieren
- automatisches Hardcopy
- Verändern der Anzeigeeigenschaften
- Sicherung der Nachrichten zur Wiederherstellung eines zerstörten Formats

Für die Kommunikation mit dem Formatierungssystem FHS verwendet *open*UTM die neutrale Schnittstelle IUTMFORM.

IUTMFORM bietet folgende Vorteile:

– Die Produkte *open*UTM und FHS werden durch diese Schnittstelle entkoppelt. Dadurch können Sie z.B. die Funktionen einer neuen FHS-Version bei gleichbleibender *open*UTM-Version nutzen und umgekehrt.

– Bei der Generierung müssen keine FHS-Makros übersetzt werden.

– UTM-Teilprogramme können auch direkt mit FHS kommunizieren. Hierzu dient der Aufruf CALL KDCFHS.

– Die Schnittstelle IUTMFORM ist so konzipiert, daß sie im Prinzip den Anschluß beliebiger Formatierungssysteme erlaubt, und ist damit offen für künftige Entwicklungen. Zur Zeit wird in BS2000/OSD nur die Formatsteuerung FHS unterstützt.

Über Produkte wie WIN-DOORS oder FHS-DOORS lassen sich Bildschirm-Formate dynamisch in grafische Oberflächen umsetzen und so z.B. auch in Microsoft-Office-Umgebungen integrieren.

Zur Formatsteuerung FHS gibt es ein eigenes Handbuch mit dem Titel „FHS - Formatierungssystem für UTM, TIAM, DCAM“. Auch die Produkte WIN-DOORS und FHS-DOORS sind ausführlich in eigenen Benutzerhandbüchern beschrieben.

Event-Exit FORMAT

Der Event-Exit FORMAT ist eine vom Anwender erstellte eigene Formatierungsroutine. Sie muß, wie das von *open*UTM standardmäßig eingesetzte Formatierungssystem FHS, sowohl physische Eingabenachrichten verarbeiten, als auch physische Ausgabenachrichten - auch für Bildschirmwiederanlauf - erzeugen können. Eine eigene Formatierungsroutine ist in folgenden Fällen sinnvoll:

– wenn Sie Funktionen benötigen, die über den von *open*UTM unterstützten Funktionsumfang von FHS hinausgehen

– wenn Sie ein anderes Formatierungssystem als FHS verwenden

– wenn Sie Terminals auf der physischen Ebene bedienen möchten

Der Event-Exit FORMAT ist im *open*UTM-Handbuch „Anwendungen programmieren mit KDCS für COBOL, C und C++“ beschrieben.

11.5 BS2000/OSD-spezifische Funktionen

Leistungskontrolle mit dem Software-Monitor SM2

Der **Software Monitor SM2** des BS2000/OSD liefert statistische Daten über die Leistungen und die Auslastung der Betriebsmittel. Mit SM2 können Sie im laufenden Betrieb das Verhalten einer UTM-Anwendung überwachen und Leistungsengpässe aufdecken. Sie können sich die Daten, die *open*UTM liefert, auf einem SM2-Bildschirm online anzeigen lassen. Zusätzlich speichert SM2 die Daten in einer Datei zur späteren Auswertung. Die Auswertung der Meßdaten zeigt das Verhalten der gesamten Anwendung auf, z.B. Mittelwerte zu Transaktionsrate, Durchsatz und Bearbeitungszeit.

Der Software Monitor SM2 ist in einem eigenen Handbuch mit dem Titel „SM2 (BS2000/OSD) Software Monitor 2“ dokumentiert. Speziell auf das Zusammenspiel zwischen SM2 und *open*UTM wird im *open*UTM-Handbuch „Anwendungen generieren und betreiben (BS2000/OSD)“ eingegangen.

Lokaler und zentraler Anschluß von Druckern

Drucker können auf zwei verschiedene Arten angeschlossen werden:

– „lokal“ an einem Terminal (wobei das Terminal die Funktion der Druckersteuerung übernimmt)
– „zentral“ an einer Druckersteuerung oder über eine im Drucker integrierte Steuerung

Die gewünschte Anschlußform ist sowohl bei der UTM- als auch bei der PDN-Generierung zu berücksichtigen, wirkt sich jedoch nicht an der Programmschnittstelle aus. Unabhängig von der Anschlußform unterstützt *open*UTM die Betriebsarten:

– Hardcopy-Betrieb und
– Spool-Betrieb.

Bei einem lokal angeschlossenen Drucker spricht man statt von Spool-Betrieb auch von Bypass-Betrieb. Bei Bypass-Betrieb kann das Terminal unabhängig von der Druckausgabe einen Dialog führen. Bypass-Betrieb ist nur zu realisieren:

– für bestimmte Terminaltypen (z.B. für 9763-Terminals) und
– wenn das betreffende Terminal nicht an einer MSN-Steuerung (Mehrfachsteuerung für Nahanschluß) angeschlossen ist.

Nutzung von RSO-Druckern

UTM-Anwendungen in BS2000/OSD können auch RSO-Drucker nutzen: Über die OLTP-Schnittstelle von RSO (Remote Spool Output) erhält *open*UTM Zugang zu allen Druckern, die RSO unterstützt, d.h. auch zu Druckern, die über LAN oder PC angeschlossen sind. Zu diesen Druckern baut *open*UTM keine Transportverbindung auf, sondern bedient sie über die OLTP-Schnittstelle von RSO, d.h. *open*UTM reserviert den Drucker bei RSO und übergibt den Druckauftrag an RSO.

An den Programm-Schnittstellen werden RSO-Drucker genauso behandelt wie auf andere Art angeschlossene Drucker.

Zu dem von BS2000/OSD zur Verfügung gestellten Software-Produkt RSO gibt es eine eigene Handbuch-Reihe. Was Sie bei der UTM-Generierung beachten müssen, ist im *open*UTM-Handbuch „Anwendungen generieren und betreiben (BS2000/OSD)“ unter dem Stichwort „RSO“ beschrieben.

Drucker Sharing

Über einen Generierungsparameter (PLEV-Parameter der KDCDEF-Anweisung LTERM) läßt sich erreichen, daß ein Drucker von mehreren UTM-Anwendungen benutzt werden kann. Dies wird dadurch möglich, daß die Verbindung zwischen einer UTM-Anwendung und dem Drucker nur für kurze Zeiträume aufrechterhalten wird, um damit anderen UTM-Anwendungen die Möglichkeit zum Verbindungsaufbau zu geben. Wird dieser Parameter verwendet, so wird von *open*UTM die Verbindung zu einem Drucker erst dann aufgebaut, wenn die Anzahl der zum Druck anstehenden Nachrichten einen Schwellwert überschreitet, welcher druckerspezifisch generiert werden kann. Die Verbindung wird wieder abgebaut, wenn keine Nachrichten mehr zum Druck anstehen.

Run Prioritäten

Bei der Generierung können Sie jedem Transaktionscode eine individuelle Run Priorität des BS2000/OSD zuordnen. Diese Run Priorität wird dem UTM-Prozeß zugeordnet, in dem das Teilprogramm abläuft. So können Sie die Scheduling Mechanismen des BS2000/OSD zur Ablaufsteuerung von UTM-Teilprogrammen einsetzen.

Spezifische Security-Funktionen

SAT-Protokollierung

Mit der BS2000/OSD-Funktion SAT (Security Audit Trail) können Sie sicherheitsrelevante UTM-Ereignisse protokollieren lassen. Diese Protokollierung ermöglicht die Beweissicherung, die nach den F2/Q3-Kriterien des ITS-Katalogs gefordert wird.

Zusätzliche Zugangskontrolle durch Verbindungspaßwörter

Jede UTM-Anwendung kann in BS2000/OSD durch ein Verbindungspaßwort, das beim Start der Anwendung vereinbart werden kann, gegen mißbräuchliche Benutzung geschützt werden. Jeder Terminalbenutzer, der mit dieser UTM-Anwendung arbeiten will, muß dieses Paßwort angeben.

Zugangsschutz durch Chipkarte

Für UTM-Anwendungen in BS2000/OSD können Benutzerkennungen so generiert werden, daß der Zugang zu einer UTM-Anwendung über diese Benutzerkennung nur mit Hilfe einer Chipkarte möglich ist. Dafür muß ein Chipkartenterminal vorhanden sein. Der Terminalbenutzer weist durch Angabe seiner persönlichen Identifikationsnummer (PIN) am Chipkartenterminal seine Berechtigung nach, diese Chipkarte verwenden zu dürfen. Die Chipkarteninformationen werden sowohl gegenüber der UTM-Anwendung als auch gegenüber den an einem zentralen Autorisierungsterminal hinterlegten Daten geprüft.

Datenbank-Schlüssel

Für UTM-Anwendungen in BS2000/OSD kann bei der Generierung einem Transaktionscode ein Datenbank-Schlüssel zugeordnet werden. Damit besteht die Möglichkeit, dem TAC bestimmte Zugriffsrechte auf der Datenbankseite zuzuordnen. *open*UTM übergibt den Schlüssel an das Datenbank-System. Dort wird er von einigen Datenbank-Systemen ausgewertet um die Zugriffsrechte auf Datenbank-Sätze zu prüfen.

Verschlüsselung bei Terminal-Anbindung

Für UTM-Anwendungen in BS2000/OSD kann die Verschlüsselung auch für Terminal-Emulationen genutzt werden. Dabei wird die Verschlüsselung auf der Host-Seite durch das BS2000/OSD-Produkt VTSU-B durchgeführt. Somit kann jede Emulation mit Verschlüsselung betrieben werden, sofern sie (wie z.B. DESK2000) die entsprechenden Funktionen unterstützt.

Nähere Informationen zu den genannten BS2000/OSD-spezifischen Security-Funktionen finden Sie im *open*UTM-Handbuch „Anwendungen generieren und betreiben (BS2000/OSD)“.

Internationalisierung / XHCS-Unterstützung

*open*UTM unterstützt die Internationalisierung:

Eine UTM-Anwendung kann so erstellt werden, daß verschiedensprachige Kommunikationspartner jeweils in ihrer Landessprache bedient werden. Selbst regional bedingte Unterschiede innerhalb einer Sprache lassen sich dabei berücksichtigen. Datumsangaben, Uhrzeit, Maßangaben oder Währungssymbole können jeweils den ortsüblichen Konventionen entsprechend dargestellt werden.

In BS2000/OSD haben Sie die Möglichkeit, bei der Konfigurierung einer UTM-Anwendung den einzelnen Benutzerkennungen, Anschlußpunkten (LTERM-Partnern) oder der gesamten Anwendung jeweils eine bestimmte Sprachumgebung - auch „Locale“ genannt - zuzuordnen. Dieses Locale geben Sie dabei jeweils wie folgt an:

LOCALE=(*sprachkennzeichen*, *territorialkennzeichen*, *zeichensatzname*)

Die Teilprogramme der UTM-Anwendung können auf die Locale-Informationen zugreifen und Eingaben des Kommunikationspartners entsprechend interpretieren bzw. Nachrichten an den Kommunikationspartner entsprechend aufbauen.

Zusätzlich haben Sie in BS2000/OSD die Möglichkeit, mehrere Meldungsmodule für eine UTM-Anwendung zu generieren und so Mehrsprachigkeit auch bei den UTM-Meldungen zu realisieren.

Damit bietet *open*UTM in BS2000/OSD Internationalisierung in vollem Umfang. In UNIX wird eine vergleichbare Funktionalität durch Nutzung des NLS (Native Language Support) zur Verfügung gestellt.

Zur Darstellung der Schrift- und Sonderzeichen der einzelnen Sprachen an Terminals oder Druckern werden u.U. verschiedene erweiterte Zeichensätze (8-Bit-Codes) benötigt. Mit Hilfe des BS2000-Softwareprodukts XHCS (eXtended Host Code Support) können in einem BS2000/OSD-System gleichzeitig mehrere erweiterte Zeichensätze verwendet werden. *open*UTM unterstützt die Funktionen von XHCS. Damit können Sie den einzelnen Benutzerkennungen, den einzelnen Anschlußpunkten (LTERM-Partnern) und auch der gesamten Anwendung eigene erweiterte Zeichensätze zuordnen, die jeweils bei der Aufbereitung der Nachrichten verwendet werden.

Nähere Informationen zu XHCS finden Sie im Benutzerhandbuch „XHCS 8-bit-Code-Verarbeitung im BS2000/OSD - Internationalisierung“. Die UTM-spezifischen Aspekte der Internationalisierung sind im *open*UTM-Handbuch „Anwendungen generieren und betreiben (BS2000/OSD)“ beschrieben. Details zum Einsatz von mehreren Meldungsmodulen enthält das *open*UTM-Handbuch „Meldungen, Test und Diagnose (BS2000/OSD)“.

Codeumsetzung bei Kommunikation mit Sockets-Anwendungen

Sockets-Anwendungen schicken ihre Nachrichten meist im ASCII- bzw. ISO 8859-1-Code, während im BS2000/OSD der EBCDIC-Code verwendet wird. Daher bietet *open*UTM in BS2000/OSD eine automatische Codeumsetzung für Sockets-Partner an.

Zur Umsetzung können eine Standardtabelle mit einer 7-Bit ASCII-EBCDIC-Konvertierung sowie drei Tabellen für eine 8-Bit-Konvertierung ausgewählt werden. Die Standardtabelle wird mit Vorbelegung ausgeliefert, die anderen Tabellen müssen vom Anwender erstellt werden.

Nähere Informationen zur Codeumsetzung finden Sie im *open*UTM-Handbuch „Anwendungen generieren und betreiben (BS2000/OSD)“, Anweisungen PTERM und TPOOL.

Einsatz des Session Managers OMNIS

Für UTM-Anwendungen in BS2000/OSD lassen sich die Dienste des BS2000/OSD-Softwareprodukts OMNIS nutzen. OMNIS ist ein Session Manager, der es ermöglicht, daß ein Terminalbenutzer Services verschiedener UTM-Anwendungen direkt aufrufen kann - selbst dann, wenn die UTM-Anwendungen im Netz verteilt sind. Dabei muß der Terminal-Benutzer nicht wissen, auf welchem Rechner und in welcher UTM-Anwendung der Service liegt: OMNIS baut automatisch eine Verbindung zur „richtigen“ UTM-Anwendung auf und regelt die Zuordnung der Nachrichten (Nachrichtenverteilung).

Zusätzlich können Sie beim Einsatz von OMNIS die Multiplex-Funktion nutzen, die *open*UTM in BS2000/OSD zur Verfügung stellt: Eine große Zahl von Terminals kann über eine kleine Zahl von Transportverbindungen mit einer UTM-Anwendung in Verbindung stehen.

Für die menügeführte Nutzung von OMNIS steht Ihnen das Zusatzprodukt OMNIS-MENU zur Verfügung.

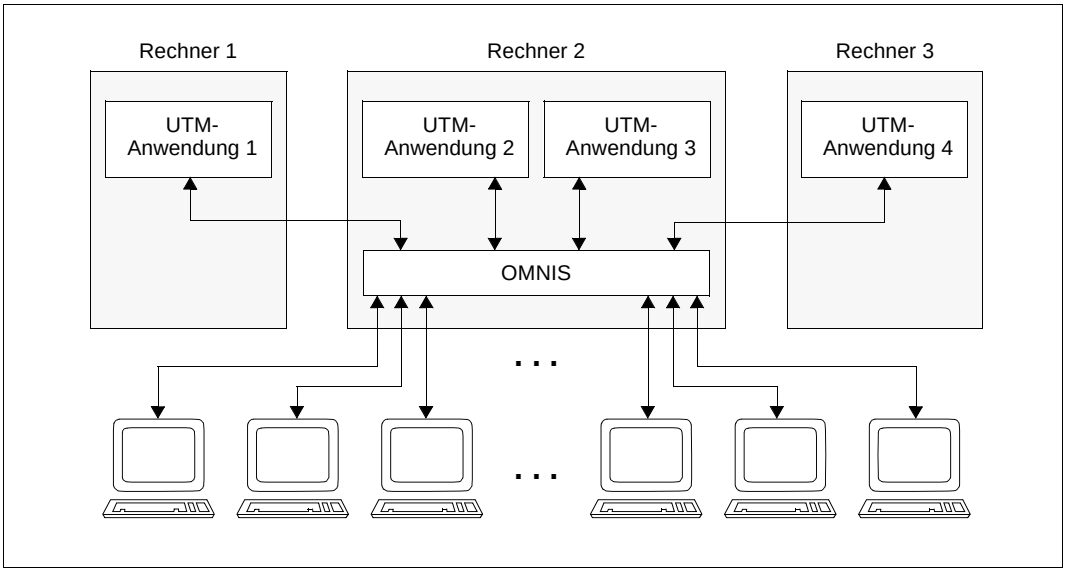


Bild 31: Nachrichtenverteilung und Multiplexing mit OMNIS

Zum Session-Manager OMNIS und zur OMNIS-Menüführung gibt es eigene Handbücher: „OMNIS/OMNIS-MENU Funktionen und Kommandos“ und „OMNIS/OMNIS-MENU Administration und Programmierung“. Wie man bei der UTM-Generierung Multiplex-Anschlüsse definiert, erfahren Sie im *open*UTM-Handbuch „Anwendungen generieren und betreiben (BS2000/OSD)“.

Dynamisches Nachladen des ROOT-Tabellenmoduls

Falls Sie in BS2000/OSD mit BLS (Binder-Lader-Starter) und LLMs (Link and Load Modules) arbeiten, haben Sie die Möglichkeit, das ROOT-Tabellenmodul beim Start der Anwendung dynamisch nachzuladen. Damit müssen Sie selbst bei tiefgehenden Änderungen in der Konfiguration, wie z.B. beim Hinzufügen neuer Lade-Module, das Anwendungsprogramm nicht neu binden.

Nähere Informationen zum dynamischen Nachladen des ROOT-Tabellenmoduls finden Sie im *open*UTM-Handbuch „Anwendungen generieren und betreiben (BS2000/OSD)“.

Aufruf von UTM-Services mit CALLUTM

Mit *open*UTM(BS2000/OSD) wird das Programm CALLUTM ausgeliefert, das es erlaubt, aus einer beliebigen BS2000-Task heraus UTM-Services aufzurufen. Das Programm bietet eine SDF-Oberfläche und kann aus dem BS2000-Kommandomodus aufgerufen werden. Eine Anwendungsmöglichkeit ist der Aufruf von UTM-Administrationskommandos, z.B. um prozedurgesteuert alle UTM-Anwendungen mit KDCSHUT zu beenden. Auf diese Weise können eine oder mehrere UTM-Anwendungen administriert werden, unabhängig davon, auf welchem Rechner oder unter welchem Betriebssystem sie laufen.

CALLUTM basiert auf dem UPIC-Client im BS2000/OSD und kommuniziert mit den UTM-Anwendungen über die UPIC-Schnittstelle. Damit kann CALLUTM das UTM-Benutzerkonzept nutzen, d.h. sich über eine UTM-Benutzerkennung anmelden, die auch durch Paßwort geschützt sein kann.

Nähere Informationen zu CALLUTM finden Sie im *open*UTM-Handbuch „Anwendungen administrieren“.

11.6 Hochverfügbarkeit durch HIPLEX AF

Eine der herausragenden Anforderungen an ein Rechenzentrum ist ein unterbrechungsfreier Betrieb, also die Verfügbarkeit des Systems rund um die Uhr, an 365 Tagen im Jahr. Eine Verfügbarkeit von 100% ist von keinem Betriebssystem erreichbar. BS2000/OSD bietet jedoch eine Vielzahl von Funktionen, mit denen ein weitgehend unterbrechungsfreier Betrieb ermöglicht wird.

Zu den Funktionen, die hierfür von BS2000/OSD angeboten werden, zählen u.a. eine dynamische Rekonfiguration der Hardware und der Software, eine dynamische Netz-Rekonfiguration, eine unterbrechungsfreie Zeitumstellung und eine Minimierung der Ausfallzeiten.

Die Minimierung von nicht-geplanten Ausfallzeiten kann dabei mit HIPLEX AF erreicht werden. HIPLEX steht für Highly Integrated Systems Complex und ist das Konzept zur Unterstützung eines Verfügbarkeits- und Lastverbundes von mehreren BS2000/OSD Business Servern.

Mit dem Produkt HIPLEX AF können über ein Mehrrechnerkonzept Anwendungen mit hoher Verfügbarkeit realisiert werden. Bei Ausfall eines Rechners werden automatisch die von HIPLEX AF überwachten Anwendungen mit ihren Betriebsmitteln auf einen intakten Rechner umgeschaltet. Die Ausfallzeit wird dabei durch eine automatische Ausfallerkennung und durch die Einsparung des Systemwiederanlaufs minimiert.

Zusammen mit *open*UTM kann HIPLEX seine volle Wirkung entfalten, denn *open*UTM bietet u.a. globale Transaktionssicherung und umfassende Wiederanlaufähigkeit bis zum Client. Damit wird die Verarbeitung nach dem Wiederanlauf auf dem anderen Rechner fortgesetzt, ohne daß dabei Daten verloren gehen oder Inkonsistenzen auftreten können.

Neben der Beherrschung des Systemausfalls kann HIPLEX AF auch die Verfügbarkeit nach Anwendungsausfall erhöhen sowie einen Lastverbund ermöglichen, indem mit HIPLEX AF Anwendungen gezielt auf einen anderen Rechner verlagert werden.

Der Einsatz von HIPLEX AF bietet Ihnen folgende Vorteile:

– Die Fehlererkennung und -behandlung erfolgt automatisch.

– Bei Einsatz von DCAM ab V12 wird bei Verlagerung der Anwendung auf einen anderen Rechner die Netzadresse der Anwendung übernommen. D.h. die Kommunikationspartner müssen lediglich die Verbindung neu aufbauen. (Voraussetzung ist die Angabe von MAX HOSTNAME in der KDCDEF-Generierung.)

– Wird der Zugang der Endbenutzer zur UTM-Anwendung durch OMNIS realisiert, so bleibt bei Anwendungsverlagerung die Verbindung zu den Endbenutzern trotz Beendigung der Anwendung erhalten (Voraussetzung ist OMNIS V7.1).

– HIPLEX AF basiert auf Standardprodukten und kann ohne kundenspezifische Programmierung oder Eingriff in die Anwendungen eingesetzt werden.

Nähere Informationen finden Sie im Handbuch:
„HIPLEX AF - Umschalten von Anwendungen“

12 *open*UTM in UNIX

In diesem Kapitel werden einige plattformspezifische Details beschrieben, die sich speziell auf den Einsatz von *open*UTM in UNIX beziehen:

- Systemeinbettung
- UTM-Prozesse in UNIX
- Adreßraumkonzept
- Formatierung
- Hochverfügbarkeit
- Vereinfachtes Konfigurieren der Netzanbindung

12.1 Systemeinbettung

Um größtmögliche Portabilität zu erreichen, nutzt der UTM-Systemcode nur die Systemaufrufe und Bibliotheksfunktionen, die UNIX im Rahmen des X/Open-Universums (System V) zur Verfügung stellt:

- Funktionen zum Anfordern und Freigeben der systeminternen Speicherbereiche (prozeßlokaler Speicher und Shared Memories)
- Funktionen zur Serialisierung (Semaphore)
- Funktionen zum Erzeugen und Beenden von Prozessen
- Funktionen zur Zeitüberwachung von Betriebsmitteln und zeitgesteuerten Nachrichten
- Funktionen zur Datei- und Datenbankbearbeitung

Neben den Schnittstellen zum UNIX-Betriebssystem verfügt eine UTM-Anwendung intern über eine Reihe weiterer Schnittstellen:

– XA-Schnittstelle (X/Open-Standard) zum Anschluß externer Resource Manager (wie z.B. INFORMIX, ORACLE, SYBASE, ADABAS, DB2, O2, MQS, ISAM/XA,...)

– Schnittstelle zum Formatierungssystem FORMANT

– UPIC-L-Schnittstelle, die es ermöglicht, *open*UTM-Client-Programme mit Trägersystem UPIC lokal anzuschließen (d.h. die Client-Programme können im selben UNIX-System ablaufen wie die UTM-Anwendung)

– Schnittstellen zu den Laufzeitsystemen der verwendeten Programmiersprachen

– Schnittstellen zu den Kommunikationskomponenten OSS und CMX bzw. PCMX

Die Teilprogramme nutzen die Funktionen über die Programm-Schnittstellen von *open*UTM, also über die X/Open-Schnittstellen CPI-C und XATMI + TX oder über die Schnittstelle KDCS (nationaler Standard).

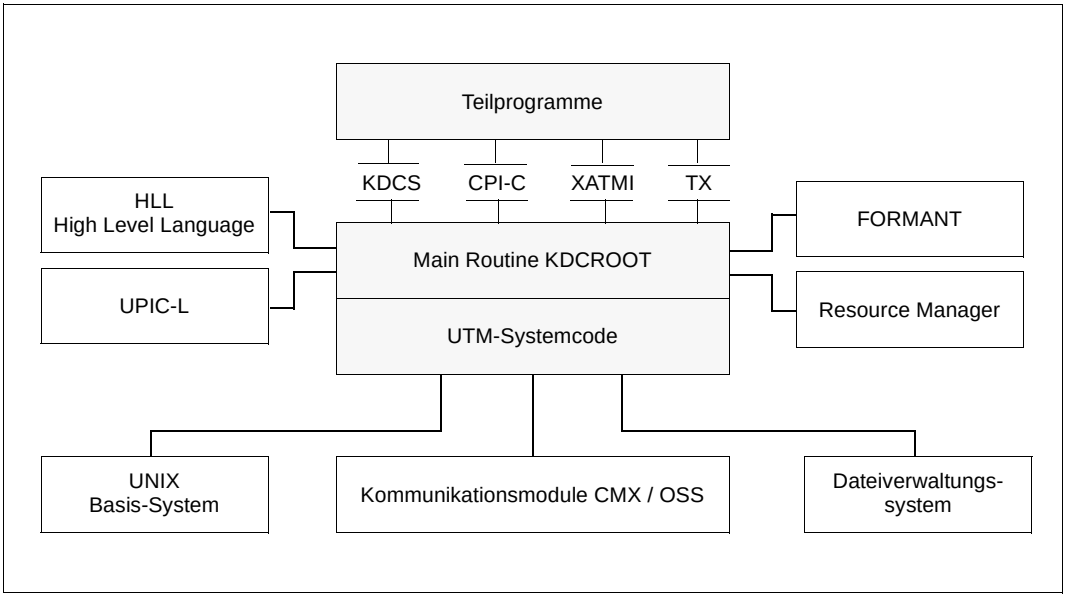


Bild 32: Schnittstellen von *open*UTM zu anderen Systemkomponenten

12.2 UTM-Prozesse in UNIX

Bei der Ausführung eines UTM-Anwendungsprogamms, das unter UNIX abläuft, arbeiten unterschiedliche Prozesse mit jeweils spezifischen Aufgaben zusammen. Diese Prozeßtypen werden in den folgenden Absätzen beschrieben.
Eine Übersicht gibt Bild 33 auf Seite 163.

Mainprozeß und Workprozesse

Gestartet wird eine UTM-Anwendung durch das Programm *utmmain*. Dieses Programm wird im allgemeinen als Hintergrundprozeß - **Mainprozeß** genannt - gestartet. Über den Mainprozeß werden so viele **Workprozesse** gestartet, wie in den Startparametern angegeben ist. In allen diesen Workprozessen wird das vom Anwender erzeugte Anwendungsprogramm geladen und gestartet.

Die Workprozesse leisten die eigentliche Arbeit: sie erledigen die Service-Anforderungen, die an die UTM-Anwendung gerichtet werden. Der Mainprozeß überwacht diese produktiv arbeitenden Prozesse. Er erzeugt während des Anwendungslaufs automatisch dann weitere Workprozesse, wenn sich ein Workprozeß wegen eines Fehlers beendet oder per Administration der Anwendung explizit weitere Workprozesse zugeteilt werden.

Nach Anwendungsstart warten alle Workprozesse der UTM-Anwendung in einer gemeinsamen Prozeßwarteschlange auf Aufträge. Trifft ein Auftrag ein, so wird er einem wartenden Workprozeß in der Prozeßwarteschlange zugeordnet. Dieser Prozeß bearbeitet den Auftrag und reiht sich anschließend wieder in die Prozeßwarteschlange ein.

Sind zur gleichen Zeit mehr Aufträge als Workprozesse vorhanden, dann wird eine Auftragswarteschlange aufgebaut. Sowohl Auftrags- als auch Prozeßwarteschlangen sind anwendungsbezogen; das bedeutet, daß unterschiedliche Anwendungen jeweils eine eigene Prozeß- und Auftragswarteschlange haben. Die Warteschlangen für Prozesse werden über Semaphore und die Warteschlangen für Aufträge über Shared Memory realisiert.

Timerprozeß

Der Mainprozeß richtet außer den Workprozessen einen der Anwendung zugeordneten **Timerprozeß** (Zeitgeberprozeß) ein. Der Timerprozeß nimmt zur Zeitüberwachung von Wartezuständen Aufträge von den Workprozessen entgegen und ordnet sie in ein Auftragsbuch ein. Nach Ablauf einer der im Auftragsbuch vermerkten Zeiten wird dies den Workprozessen zur Bearbeitung mitgeteilt.

Dialogterminalprozesse (DTPs)

Für jedes Terminal, das mit der UTM-Anwendung arbeitet, existiert ein eigener Dialogprozeß, genannt **Dialogterminalprozeß**. Er wird weder vom Main- noch von einem Workprozeß erzeugt, sondern entweder aus der Shell durch Starten des Programms *utmdtp* etabliert oder nach erfolgreichem Anmelden des Benutzers an das UNIX-System automatisch gestartet.

Der Terminal-Benutzer wählt die UTM-Anwendung aus. Dadurch wird eine Verbindung zwischen dem Dialogterminalprozeß und der UTM-Anwendung etabliert. Anschließend kann der Dialogterminalprozeß Aufträge an diese UTM-Anwendung senden bzw. Nachrichten von dieser UTM-Anwendung empfangen.

Lokale Client-Prozesse

Für jeden *open*UTM-Client mit Trägersystem UPIC, der mit der UTM-Anwendung arbeitet, existiert ein eigener lokaler Client-Prozeß. Er wird weder vom Main- noch von einem Workprozeß erzeugt, sondern aus der Shell gestartet.

Der lokale Client-Prozeß baut die Verbindung zur UTM-Anwendung auf. Anschließend kann der lokale Client-Prozeß Aufträge an diese UTM-Anwendung senden bzw. Nachrichten von dieser UTM-Anwendung empfangen.

Druckerprozesse

Asynchrone Nachrichten an Drucker werden von der UTM-Anwendung über eigene Prozesse, **Druckerprozesse** genannt, ausgegeben. Für jeden angeschlossenen Drucker wird vom Mainprozeß der UTM-Anwendung ein Druckerprozeß eingerichtet. Der Druckerprozeß für einen Drucker existiert solange, wie dieser an die UTM-Anwendung angeschlossen ist.

Netzprozesse

Bei verteilter Verarbeitung werden UTM-Anwendungen über **Netzprozesse** ans Netz angebunden. Diese Prozesse haben die Aufgabe, Verbindungsanforderungen zu bearbeiten und den Datentransfer auf dieser Verbindung zu verwalten.

Die Netzanbindung kann über CMX oder direkt über die Sockets-Schnittstelle laufen. Die Anzahl der Netzprozesse und deren Art („Single-Threading oder „Multi-Threading“) ist generierungsabhängig.

Weitere Informationen über Netzprozesse sowie über Details der Generierung erhalten Sie im *open*UTM-Handbuch „Anwendungen generieren und betreiben (UNIX, Windows NT)“.

Übersicht: Prozesse einer UTM-Anwendung in UNIX

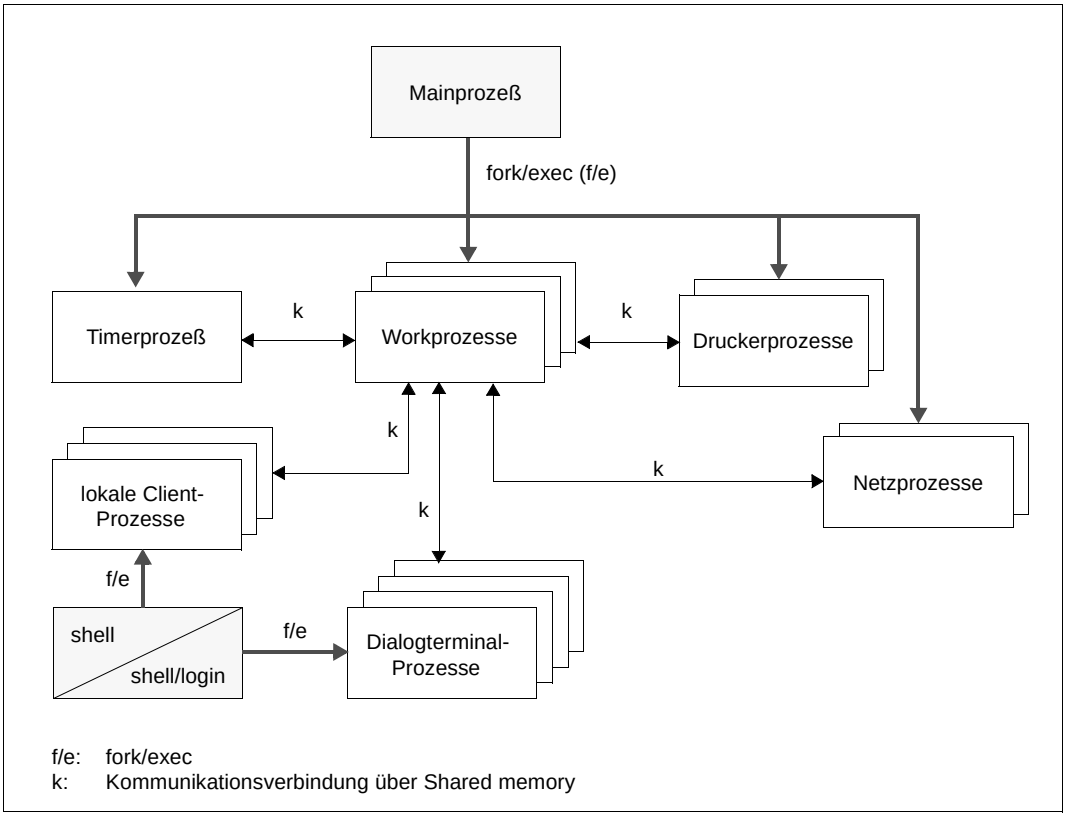


Bild 33: Prozeßinteraktion in einer UNIX-UTM-Anwendung (Netzanbindung single-threaded)

12.3 Adreßraumkonzept

In einer UTM-Anwendung verfügt jeder Workprozeß über einen prozeßspezifischen Speicherbereich. In ihm sind enthalten:

- der Datenbereich ROOTDATA zur Verständigung zwischen KDCROOT und den Systemfunktionen
- die Bereiche KB und SPAB
- Pufferbereiche für MPUT-Nachrichten
- ein Tracebereich für KDCS-Aufrufe zu Diagnosezwecken
- Tabellen zum Ansprung der Teilprogramme
- der Datenbereich KTA (KDCS Task Area), der nur von den UTM-Systemfunktionen benutzt wird. Er enthält weitere Pufferbereiche, einen UTM-internen Tracebereich und verschiedene prozeßspezifische Kontrolldaten.

Alle Workprozesse einer UTM-Anwendung verfügen gemeinsam über ein Shared Memory, das die konfigurations- und anwendungsglobalen Verwaltungsdaten enthält (KAA = KDCS Application Area), sowie über ein Shared Memory für einen Cachebereich zur Optimierung der Dateizugriffe.

Workprozesse und externe Prozesse (Dialogterminal-, Drucker-, Netzprozesse, Timerprozeß sowie lokaler Client-Prozeß) verwenden zusammen einen Shared Memory-Bereich zur Prozeßkommunikation (IPC = Inter Process Communication) und Auftragsabwicklung.

Da das Betriebssystem UNIX für Anwendungsprogramme über keine speziellen Schutzmechanismen verfügt, ist zu beachten, daß Fehler in den vom Anwender erstellten Teilprogrammen auch UTM-System-Bereiche zerstören können.

Das folgende Bild zeigt die genannten Beziehungen zwischen den Shared Memories und den Prozessen einer UTM-Anwendung.

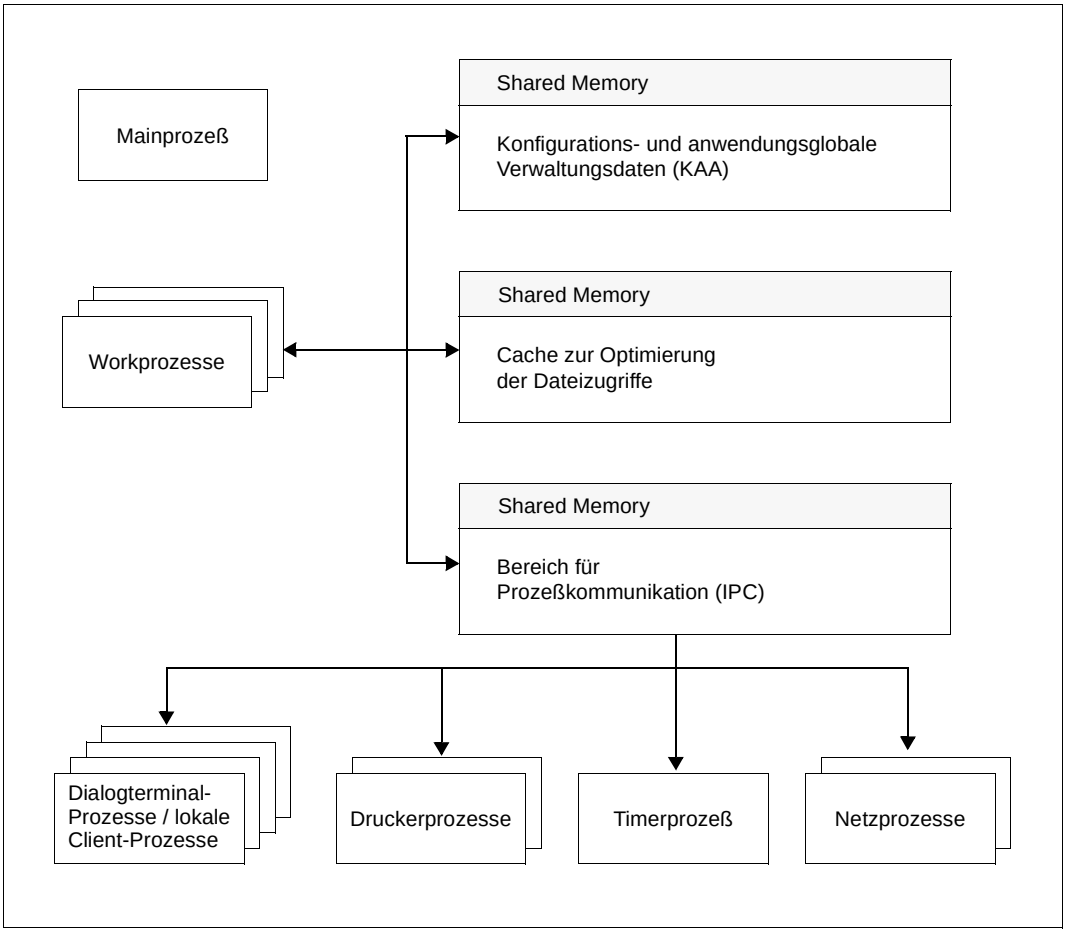


Bild 34: Shared Memories und Prozesse in UTM-Anwendungen

12.4 Formatierung

Falls Sie für Ihre UTM-Anwendung Terminals im Formatmodus einsetzen wollen, können Sie die Standard-Software FORMANT (FORmat MANager für Terminals) nutzen.

FORMANT besteht aus zwei Komponenten:

1. Formatgenerator FORMANTGEN
2. FORMANT-Formatsteuerung

Formatgenerator FORMANTGEN

Mit dem interaktiven Formatgenerator FORMANTGEN lassen sich im geführten Dialog schnell und einfach Formate erstellen. FORMANTGEN erzeugt dabei automatisch entsprechende Datenstrukturen (=Adressierungshilfen), die Sie in Ihre Teilprogramme integrieren können. FORMANTGEN unterstützt Sie auch bei der Verwaltung und Wartung Ihrer Format-Bibliotheken.

FORMANT-Formatsteuerung

Die FORMANT-Formatsteuerung ermöglicht den Einsatz der erzeugten Formate in den Teilprogrammen: Nachrichten im Formatmodus, die von Teilprogrammen an Terminals und Drucker, bzw. von Terminals an Teilprogramme gesendet werden, werden in *open*UTM über die FORMANT-Formatsteuerung formatiert. *open*UTM (UNIX) unterscheidet die Formattypen *Formate, +Formate und #Formate, mit jeweils unterschiedlichen Funktionen und Einsatzmöglichkeiten. Der Formattyp ist am ersten Zeichen des Formatkennzeichens zu erkennen und wird beim Übertragen einer Nachricht mitgeliefert.

Die Unterstützung der Schnittstellen zur Formatsteuerung FORMANT ist in *open*UTM integriert. FORMANT unterstützt den Einsatz von Formaten, die mit FORMANTGEN erstellt wurden sowie von IFG-Formaten, wenn diese zuvor von den Migrationsprogrammen (BS2000/UNIX) umgesetzt wurden. Somit können Sie auch Formate, die für BS2000/OSD-Anwendungen erstellt wurden, für Ihre UNIX-UTM-Anwendung übernehmen.

FORMANT bietet eine Vielzahl von Funktionen, wie z.B.:

- Auffüllen der Nachrichtenbereiche mit frei wählbaren Zeichen
- Kennzeichnen der vom Terminal-Benutzer ausgewählten bzw. nicht ausgewählten Felder
- Übertragen aller variablen Felder
- Positionieren des Cursors
- Manuelles Hardcopy mittels PRINT-Taste
- Verändern der Anzeigeeigenschaften
- Wiederherstellen eines zerstörten Bildschirminhalts mit der WAZ-Taste
- Sichern der Nachrichten zur Wiederherstellung eines zerstörten Formats

Die Formatierung von Nachrichten für Terminals und Drucker läuft zu einem Teil in den Workprozessen und zum anderen Teil in den Dialogterminal- bzw. Druckerprozessen ab. Die Funktionen des Formatierungssystems FORMANT sind auf unterschiedliche Prozesse verteilt.

Den Nachrichtenfluß und die beteiligten Prozesse bei UTM-Anwendungen mit FORMANT zeigt Bild 35 auf der folgenden Seite.

Nähere Informationen zur Formatierung finden Sie im Handbuch „FORMANT“. Speziell über die Nutzung von FORMANT für *open*UTM gibt es ein eigenes Handbuch mit dem Titel „UTM(UNIX) Formatierungssystem“.

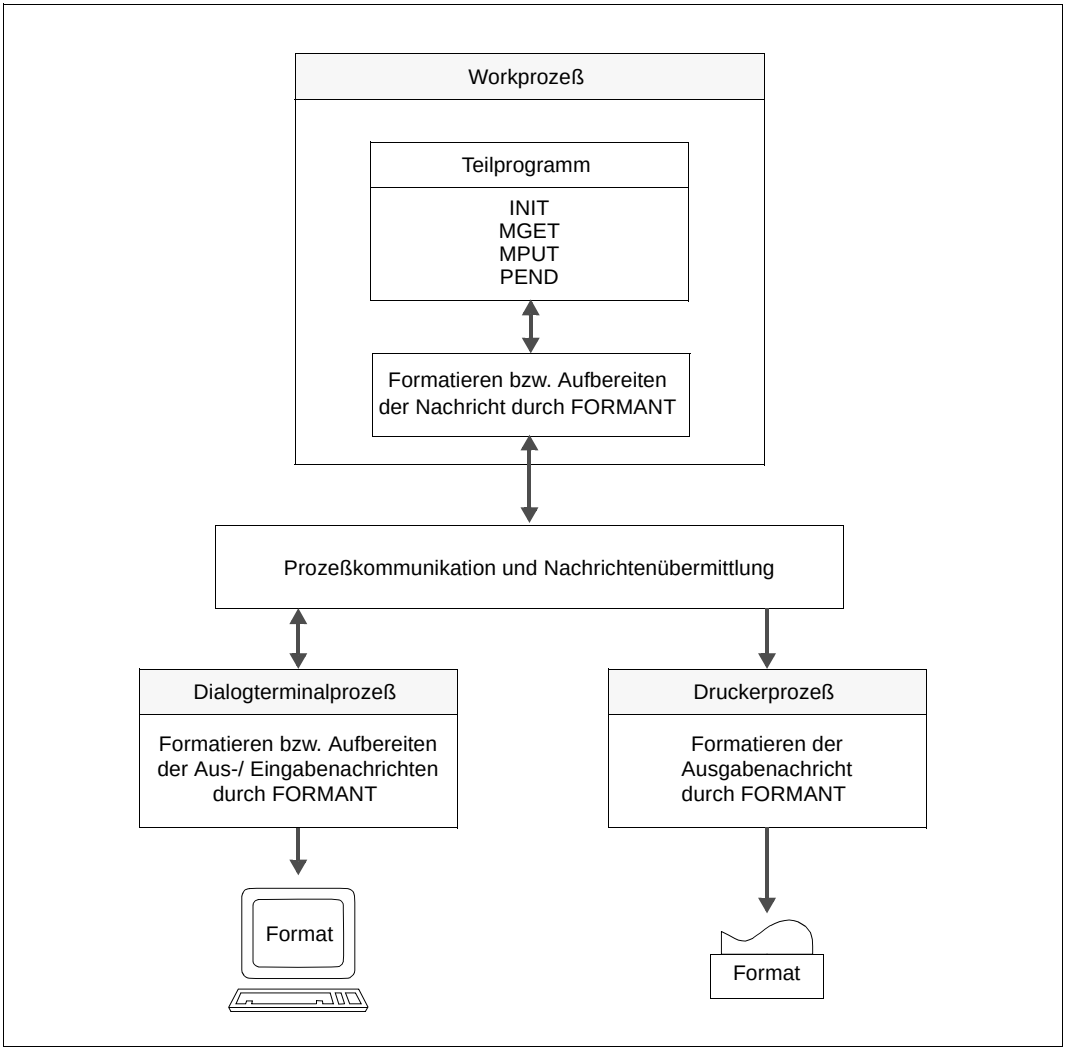


Bild 35: Nachrichtenfluß bei UTM-Anwendungen mit dem Formatierungssystem FORMANT

12.5 Hochverfügbarkeit in Reliant UNIX

Ausfälle von Rechnersystemen, Hardware oder Software können in manchen Anwendungen hohe Kosten verursachen. Daher gelten vorbeugende Maßnahmen in solchen Fällen als grundlegende Bedingungen für den Betrieb.

Hierzu gehören u.a.:

– der Einsatz von Cluster-Konfigurationen
– spezielle Platten-Peripherie, wie z.B. RAID-Systeme
– Software, die Defekte erkennt und ein rasches Umschalten von defekter Hardware auf Ersatzsysteme erlaubt
– Entfernte Aufstellung von Peripherien
– Aufteilung eines Failover-Clusters auf entfernte Rechenzentren

Diese Maßnahmen können zusammen mit *open*UTM ihre optimale Wirkung entfalten, denn *open*UTM bietet u.a. globale Transaktionssicherung und umfassende Wiederanlaufähigkeit bis zum Client. Damit ist *open*UTM insbesondere für den Einsatz auf Clustern prädestiniert, da die Verarbeitung nach dem Wiederanlauf auf dem anderen System fortgesetzt wird, ohne daß dabei Daten verloren gehen oder Inkonsistenzen auftreten können.

Reliant UNIX- bzw. SINIX-(RM-)Systeme können wie folgt als Cluster eingesetzt werden:

- Zwei Systeme (1:1 Konfiguration) mit umschaltbaren Peripherieschränken:
  Durch Umschalten der Peripheriegeräte vom defekten Rechner auf den aktiven redundanten Rechner kann die UTM-Anwendung nach einer kurzen Unterbrechung weitergeführt werden. Die Umschaltung der Festplatten oder der externen SCSI-Geräte erfolgt über den SCSI Bus-Umschalter (SCU1 bzw. SCU2). Terminals und Drucker können umgeschaltet werden, wenn sie über Terminalserver TACLAN angeschlossen sind.

  Terminals und Drucker, die direkt am LAN hängen, können mit IP-Aliasing umgeschaltet werden. Je nach Konfiguration kann die Plattenperipherie auch über Multihosted SCSI oder über Fibre Channel angeschlossen werden, so daß dafür keine Umschalter mehr notwendig sind.

- Zwei Systeme (1:1 Konfiguration) mit umschaltbaren RAID-Systemen:
  Wenn die beiden Systeme über ein RAID-System (mit zwei RAID-Controllern) gekoppelt werden, ist SCSI-Umschalter SCU1 bzw. SCU2 nicht mehr erforderlich.

- Mehrrechnersysteme (1:n) mit umschaltbaren Plattenschränken bzw. RAID-Systemen:
  Bei einer 1:n-Konfiguration kann ein aktiver redundanter Rechner die übrigen n Rechner überwachen und ggf. deren Aufgaben übernehmen. Zu einem bestimmten Zeitpunkt kann allerdings nur von **einem** (defekten) Rechner auf den aktiven Rechner umgeschaltet werden. Zur Zeit können Konfigurationen bis 1:4 unterstützt werden. Hierzu ist SINIX bzw. Reliant UNIX ab Version 5.42 erforderlich.

- Mehrrechnersysteme (*n*:*m*) mit von mehreren Rechnern zugreifbaren Plattenschränken oder RAID-Systemen, die sinnvollerweise gespiegelt sind. Bei Ausfall eines Rechners können die Anwendungen auf die anderen Rechner des Clusters verteilt werden. Dabei sind Konfigurationen bis zu 8 Rechnern möglich.

Als Überwachungs- und Umschalt-Software kann OBSERVE für Cluster mit 2 Knoten und RMS für Cluster mit bis zu 8 Knoten eingesetzt werden. Diese Software dient dazu, Ausfälle zu erkennen und automatisch von defekten auf intakte Systeme umzuschalten. Eine entsprechend vorbereitete UTM-Anwendung kann den Betrieb auf dem intakten System sofort oder nach einem Restart weiterführen.

Nähere Informationen zu diesem Themenkomplex finden Sie im Handbuch: „Hohe Verfügbarkeit von SNI UNIX-Systemen (Reliant UNIX V5.43)“ bzw. „Hochverfügbarkeit der RM-Systeme (Reliant UNIX V5.44)“.

12.6 Vereinfachtes Konfigurieren der Netzanbindung

Die Parameter für die Netzanbindung (Rechnername, Adresse,...) werden auf UNIX normalerweise im Transport Name Server (TNS) gehalten. Der TNS ist Bestandteil des Transportzugriffssystems CMX. Die Einträge im TNS müssen mit der Konfiguration des jeweiligen Aufsatzproduktes abgeglichen werden.

Abgleich zwischen UTM-Generierung und TNS

*open*UTM bietet in UNIX die Möglichkeit, eine Reihe der notwendigen Parameter in der UTM-Konfiguration zu hinterlegen, so daß nur noch vereinfachte Referenzeinträge im TNS benötigt werden. Beim Generierungslauf werden diese Parameter auf Konsistenz geprüft und mit dem TNS abgeglichen. Die vollständigen TNS-Einträge werden anschließend mit UTM-spezifischen Tools erzeugt. Damit werden viele Fehler schon beim Konfigurieren entdeckt und müssen nicht erst zur Laufzeit der Anwendung mühsam analysiert werden.

Besonders einfach kann eine TCP/IP-Kopplung konfiguriert werden.

TCP/IP-Kopplung ohne TNS

Bei Kopplungen über TCP/IP-RFC1006 kann der Anwender ganz auf TNS-Einträge verzichten. Dazu muß er die notwendigen Parameter in die UTM-Generierung eintragen. Die Parameter werden dann zur Laufzeit aus der UTM-Generierung entnommen; die IP-Adressen werden beim Anwendungsstart aus der jeweiligen Hosts-Datei bzw. Host-Datenbank ermittelt.

Damit eine UTM-Anwendung auch dann ohne Unterbrechung betrieben werden kann, wenn sich im Netz eine Adresse ändert, besitzt *open*UTM eine entsprechende Administrations-Funktion. Mit dieser Funktion können IP-Adressen aus der Host-Datenbank gelesen und zur Laufzeit an die UTM-Anwendung übergeben werden.

Diese Funktion wird auch für reine TCP/IP-Verbindungen über Sockets genutzt.

Wie Sie die Netzanbindung konfigurieren, ist detailliert im *open*UTM-Handbuch „Anwendungen generieren und betreiben (UNIX, Windows NT)“ beschrieben.

13 *open*UTM in Windows NT

In diesem Kapitel werden einige plattformspezifische Details beschrieben, die sich speziell auf den Einsatz von *open*UTM in Windows NT beziehen:

– Systemeinbettung
– UTM-Prozesse in Windows NT
– Adreßraumkonzept
– Hochverfügbarkeit
– Konfigurieren der Netzanbindung

Einschränkungen

Formatierung und transaktionsgesicherte Druckausgaben werden auf Windows NT nicht unterstützt.

13.1 Systemeinbettung

Der UTM-Systemcode wurde von UNIX auf Windows NT portiert. Neben den Schnittstellen zum Windows NT-Betriebssystem verfügt eine UTM-Anwendung intern über eine Reihe weiterer Schnittstellen:

– XA-Schnittstelle (X/Open-Standard) zum Anschluß externer Resource Manager (wie z.B. INFORMIX, ORACLE, MS-SQL Server,...)

– UPIC-L-Schnittstelle, die es ermöglicht, *open*UTM-Client-Programme mit Trägersystem UPIC lokal anzuschließen (d.h. die Client-Programme können im selben Windows NT-System ablaufen wie die UTM-Anwendung)

– Schnittstellen zu den Laufzeitsystemen der verwendeten Programmiersprachen

– Schnittstellen zu den Kommunikationskomponenten OSS und PCMX

Die Teilprogramme nutzen die Funktionen über die Programm-Schnittstellen von *open*UTM, also über die X/Open-Schnittstellen CPI-C und XATMI + TX oder über die Schnittstelle KDCS (nationaler Standard).

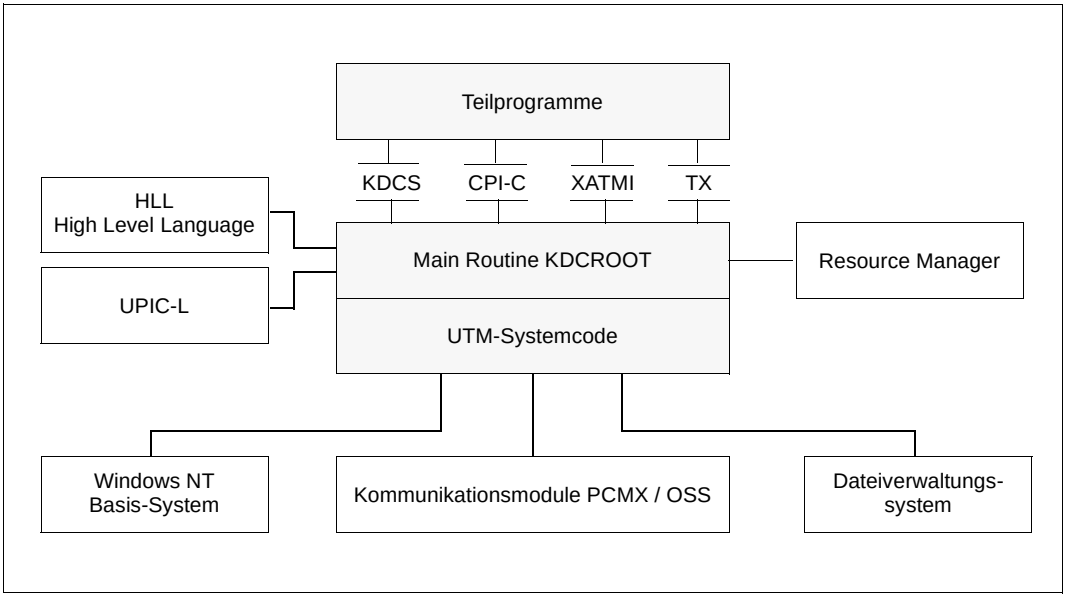


Bild 36: Schnittstellen von *open*UTM zu anderen Systemkomponenten

13.2 UTM-Prozesse in Windows NT

Ein UTM-Anwendung wird in Windows NT als Win32 Console-Application erstellt. Bei der Ausführung eines UTM-Anwendungsprogamms arbeiten unterschiedliche Prozesse mit jeweils spezifischen Aufgaben zusammen. Einige dieser Prozesse werden in einem DOS-Fenster per Programmaufruf gestartet. Für diese Programmaufrufe können Shortcuts erstellt werden, so daß die Prozesse per Mausklick oder Tastaturbefehl gestartet werden können.

Die verschiedenen Prozeßtypen werden in den folgenden Absätzen beschrieben. Eine Übersicht gibt Bild 37 auf Seite 177.

Mainprozeß und Serviceprozeß

Eine UTM-Anwendung wird gestartet, indem der **Mainprozeß** eingerichtet wird. Der Mainprozeß kann entweder im Vordergrund oder im Hintergrund ablaufen.

Durch Aufruf des Programms *utmmain* wird der Mainprozeß im Vordergrund gestartet. *utmmain* wird in einem DOS-Fenster aufgerufen. Diese Art des Anwendungsstarts kann insbesondere während der Anwendungsentwicklung verwendet werden.

Für den Produktivbetrieb kann eine UTM-Anwendung auch über das Programm *utmmains* als Dienst (Service) gestartet werden. Dieser Prozeß wird daher **Serviceprozeß** genannt. Er startet wiederum den Mainprozeß, der im Hintergrund abläuft. Dabei werden alle Ausgaben auf Datei umgelenkt. Ist die Anwendung als Dienst eingerichtet, dann kann sie nach dem Start von Windows NT automatisch mit gestartet werden.

Workprozesse

Über den Mainprozess werden so viele **Workprozesse** gestartet, wie in den Startparametern angegeben ist. In allen diesen Workprozessen wird das vom Anwender erzeugte Anwendungsprogramm geladen und gestartet.

Die Workprozesse leisten die eigentliche Arbeit: sie erledigen die Service-Anforderungen, die an die UTM-Anwendung gerichtet werden. Der Mainprozeß überwacht diese produktiv arbeitenden Prozesse. Er erzeugt während des Anwendungslaufs automatisch dann weitere Workprozesse, wenn sich ein Workprozeß wegen eines Fehlers beendet oder per Administration der Anwendung explizit weitere Workprozesse zugeteilt werden.

Nach Anwendungsstart warten alle Workprozesse der UTM-Anwendung in einer gemeinsamen Prozeßwarteschlange auf Aufträge. Trifft ein Auftrag ein, so wird er einem wartenden Workprozeß in der Prozeßwarteschlange zugeordnet. Dieser Prozeß bearbeitet den Auftrag und reiht sich anschließend wieder in die Prozeßwarteschlange ein.

Sind zur gleichen Zeit mehr Aufträge als Workprozesse vorhanden, dann wird eine Auftragswarteschlange aufgebaut. Sowohl Auftrags- als auch Prozeßwarteschlangen sind anwendungsbezogen; das bedeutet, daß unterschiedliche Anwendungen jeweils eine eigene Prozeß- und Auftragswarteschlange haben. Die Warteschlangen für Prozesse werden über Semaphore und die Warteschlangen für Aufträge über Shared Memory realisiert.

Timerprozeß

Der Mainprozeß richtet außer den Workprozessen einen der Anwendung zugeordneten **Timerprozeß** (Zeitgeberprozeß) ein. Der Timerprozeß nimmt zur Zeitüberwachung von Wartezuständen Aufträge von den Workprozessen entgegen und ordnet sie in ein Auftragsbuch ein. Nach Ablauf einer der im Auftragsbuch vermerkten Zeiten wird dies den Workprozessen zur Bearbeitung mitgeteilt.

Dialogterminalprozeß (DTP)

Für jedes Console-Fenster, das mit der UTM-Anwendung arbeitet, existiert ein eigener Dialogprozeß, genannt Dialogterminalprozeß. Dieser wird aus der DOS-Shell durch Starten des Programms *utmdtp* etabliert, er kann aber auch automatisch nach Anmelden des Benutzers an das Windows NT-System gestartet werden, z.B. durch einen entsprechenden Eintrag in der Autostart-Gruppe.

Der Benutzer wählt die UTM-Anwendung aus. Dadurch wird eine Verbindung zwischen dem Dialogterminalprozeß und der UTM-Anwendung etabliert. Anschließend kann der Dialogterminalprozeß Aufträge an diese UTM-Anwendung senden bzw. Nachrichten von dieser UTM-Anwendung empfangen.

Ein Dialogterminalprozeß kann nur auf dem Rechner gestartet werden, auf dem die UTM-Anwendung läuft.

Shutdownprozeß

Bereits beim Anwendungsstart wird der Shutdownprozeß *utmshut* eingerichtet. Dieser Prozeß sorgt beim System-Shutdown dafür, daß die UTM-Anwendung ordnungsgemäß beendet wird.

Lokale Client-Prozesse

Für jeden *open*UTM-Client mit Trägersystem UPIC, der mit der UTM-Anwendung arbeitet, existiert ein eigener lokaler Client-Prozeß. Er wird aus der DOS-Shell gestartet.

Der lokale Client-Prozeß baut die Verbindung zur UTM-Anwendung auf. Anschließend kann der lokale Client-Prozeß Aufträge an diese UTM-Anwendung senden bzw. Nachrichten von dieser UTM-Anwendung empfangen.

Netzprozesse

Bei verteilter Verarbeitung werden UTM-Anwendungen über **Netzprozesse** ans Netz angebunden. Diese Prozesse werden vom Mainprozeß eingerichtet und haben die Aufgabe, Verbindungsanforderungen zu bearbeiten und den Datentransfer auf dieser Verbindung zu verwalten.

Die Netzanbindung kann über PCMX oder direkt über die Sockets-Schnittstelle laufen. Die Anzahl der Netzprozesse ist generierungsabhängig. Die Geschwindigkeit des Netzzugriffs kann mit Umgebungsvariablen gesteuert werden.

Weitere Informationen über Netzprozesse sowie über Details der Generierung erhalten Sie im *open*UTM-Handbuch „Anwendungen generieren und betreiben (UNIX, Windows NT)“.

Übersicht: Prozesse einer UTM-Anwendung in Windows NT

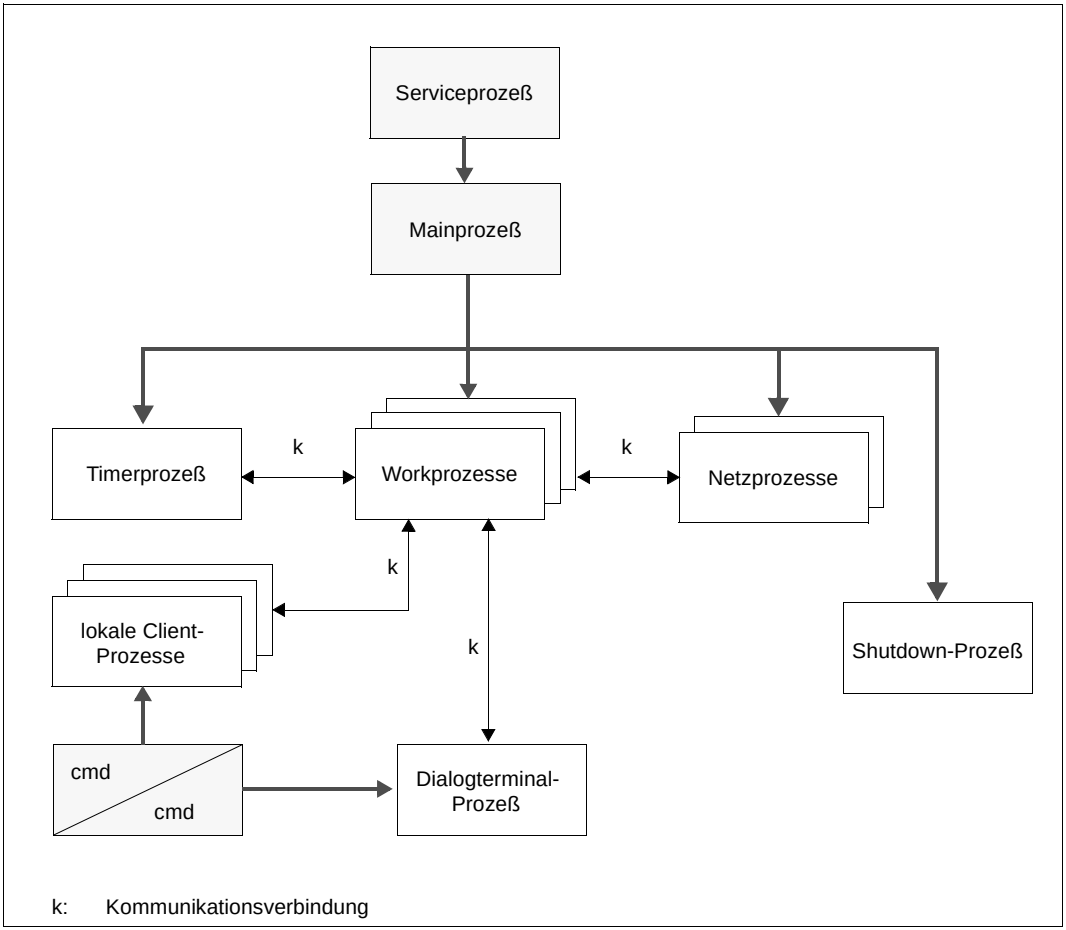


Bild 37: Prozeßinteraktion in einer UTM-Anwendung auf Windows NT

13.3 Adreßraumkonzept

In einer UTM-Anwendung verfügt jeder Workprozeß über einen prozeßspezifischen Speicherbereich. In ihm sind enthalten:

– der Datenbereich ROOTDATA zur Verständigung zwischen KDCROOT und den Systemfunktionen

– die Bereiche KB und SPAB

– Pufferbereiche für MPUT-Nachrichten

– ein Tracebereich für KDCS-Aufrufe zu Diagnosezwecken

– Tabellen zum Ansprung der Teilprogramme

– der Datenbereich KTA (KDCS Task Area), der nur von den UTM-Systemfunktionen benutzt wird. Er enthält weitere Pufferbereiche, einen UTM-internen Tracebereich und verschiedene prozeßspezifische Kontrolldaten.

Alle Workprozesse einer UTM-Anwendung verfügen gemeinsam über ein *memory mapped file*, das die Konfigurations- und anwendungsglobalen Verwaltungsdaten enthält (KAA = KDCS Application Area), sowie über ein *memory mapped file* für einen Cachebereich zur Optimierung der Dateizugriffe.

Workprozesse und externe Prozesse (Dialogterminalprozesse, Netzprozesse, Timerprozeß sowie lokaler Client-Prozeß) verwenden zusammen einen *memory mapped file*-Bereich zur Prozeßkommunikation (IPC = Inter Process Communication) und Auftragsabwicklung.

Da das Betriebssystem Windows NT für Anwendungsprogramme über keine speziellen Schutzmechanismen verfügt, ist zu beachten, daß Fehler in den vom Anwender erstellten Teilprogrammen auch UTM-System-Bereiche zerstören können.

Das folgende Bild zeigt die genannten Beziehungen zwischen den *memory mapped files* und den Prozessen einer UTM-Anwendung.

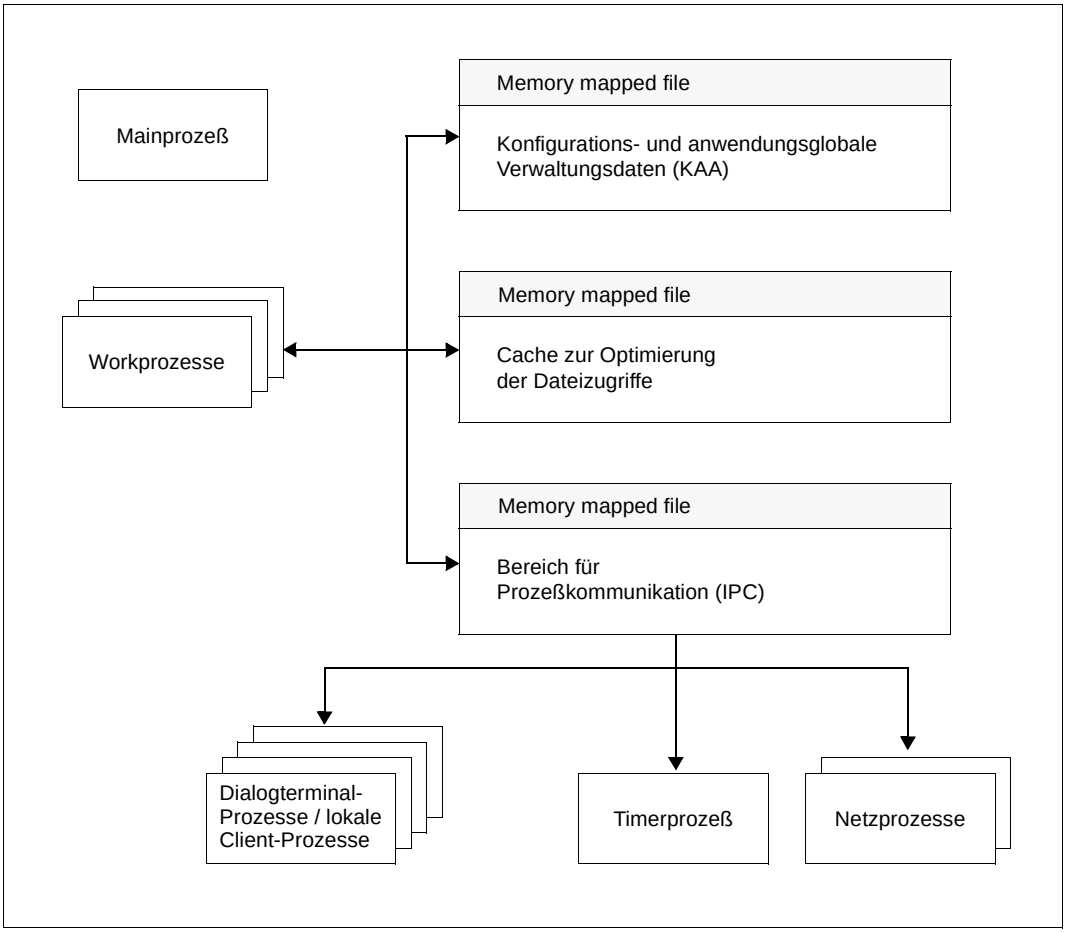


Bild 38: Memory mapped files und Prozesse in UTM-Anwendungen auf Windows NT

13.4 Hochverfügbarkeit in Windows NT

Ausfälle von Rechnersystemen, Hardware oder Software können in manchen Anwendungen hohe Kosten verursachen. Daher gelten vorbeugende Maßnahmen in solchen Fällen als grundlegende Bedingungen für den Betrieb. Hierzu gehören u.a.:

– der Einsatz von Cluster-Konfigurationen
– spezielle Platten-Peripherie, wie z.B. RAID-Systeme
– Software, die Defekte erkennt und ein rasches Umschalten von defekter Hardware auf Ersatzsysteme erlaubt

Diese Maßnahmen können zusammen mit *open*UTM ihre optimale Wirkung entfalten, denn *open*UTM bietet u.a. globale Transaktionssicherung und umfassende Wiederanlaufähigkeit bis zum Client. Damit ist *open*UTM insbesondere für den Einsatz auf Clustern prädestiniert, da die Verarbeitung nach dem Wiederanlauf auf dem anderen System fortgesetzt wird, ohne daß dabei Daten verloren gehen oder Inkonsistenzen auftreten können.

Windows NT-Systeme mit PRIMERGY-Hardware können wie folgt als Cluster eingesetzt werden:

- Symmetrischer Cluster mit Microsoft Cluster Server:
  Eine Microsoft Cluster Server Konfiguration besteht heute aus zwei unabhängigen Windows NT-basierten Servern, die über eine Hochgeschwindigkeitsverbindung miteinander gekoppelt sind und sich dem Benutzer als ein einziges Serversystem darstellen. Die Clients im Verbund können transparent auf alle Ressourcen des Clusters zugreifen. Fällt ein Server aus, dann übernimmt der andere Server automatisch dessen Funktionen. Beide Server können im "mission-critical"-Betrieb laufen, d.h. sie arbeiten produktiv an eigenen Anwendungen und überwachen sich zusätzlich gegenseitig.

- Datenbank-Cluster
  Datenbank-Cluster, allen voran ORACLE Parallel Server (OPS), haben sich in den letzten Jahren durch viele Installationen weltweit etabliert. Durch Portierung dieser Lösungen auf Microsoft Windows NT wird die aus diesen Installationen erwachsene Erfahrung und Kompetenz nun auch NT-Nutzern zur Verfügung gestellt. ORACLE Parallel Server ist für Windows NT derzeit für die Kopplung von bis zu 4 PRIMERGY-Systemen freigegeben und verfügbar.

- Asymmetrischer Cluster mit PRIMERGY ServerShield und SCSI-Switch:
  Der Cluster besteht aus Primär- und Sekundär-Server. Der Primär-Server bearbeitet standardmäßig die unternehmenskritischen Aufgaben, während der Sekundär-Server weniger kritische Aufgaben übernimmt. Der Sekundär-Server überwacht den Primär-Server. Erkennt er einen Fehler des Primär-Servers, so beendet er ordnungsgemäß seine eigenen Anwendungen und übernimmt die Datenbestände des Primär-Servers. Der anschließende Neustart des Sekundär-Servers mit Daten und Anwendungen des Primär-Servers stellt die Verfügbarkeit des Gesamtsystems in kürzester Zeit wieder her.

13.5 Vereinfachtes Konfigurieren der Netzanbindung

Die Parameter für die Netzanbindung (Rechnername, Adresse,...) werden auf Windows NT normalerweise im Transport Name Server (TNS) gehalten. Der TNS ist Bestandteil des Transportzugriffssystems PCMX. Die Einträge im TNS müssen mit der Konfiguration des jeweiligen Aufsatzproduktes abgeglichen werden.

Abgleich zwischen UTM-Generierung und TNS

*open*UTM bietet in Windows NT die Möglichkeit, eine Reihe der notwendigen Parameter in der UTM-Konfiguration zu hinterlegen, so daß nur noch vereinfachte Referenzeinträge im TNS benötigt werden. Beim Generierungslauf werden diese Parameter auf Konsistenz geprüft und mit dem TNS abgeglichen. Die vollständigen TNS-Einträge werden anschließend mit UTM-spezifischen Tools erzeugt. Damit werden viele Fehler schon beim Konfigurieren entdeckt und müssen nicht erst zur Laufzeit der Anwendung mühsam analysiert werden.

Besonders einfach kann eine TCP/IP-Kopplung konfiguriert werden.

TCP/IP-Kopplung ohne TNS

Bei Kopplungen über TCP/IP-RFC1006 kann der Anwender ganz auf TNS-Einträge verzichten, denn *open*UTM entnimmt die Parameter zur Laufzeit aus der UTM-Generierung; die IP-Adressen werden beim Anwendungsstart aus der jeweiligen Hosts-Datei bzw. Host-Datenbank ermittelt.

Damit eine UTM-Anwendung auch dann ohne Unterbrechung betrieben werden kann, wenn sich im Netz eine Adresse ändert, besitzt *open*UTM eine entsprechende Administrations-Funktion. Mit dieser Funktion können IP-Adressen aus der Host-Datenbank gelesen und zur Laufzeit an die UTM-Anwendung übergeben werden.

Diese Funktion wird auch für reine TCP/IP-Verbindungen über Sockets genutzt.

Wie Sie die Netzanbindung konfigurieren, ist detailliert im *open*UTM-Handbuch „Anwendungen generieren und betreiben (UNIX, Windows NT)“ beschrieben.

14 Anhang: Unterstützte Standards und Normen

Programmschnittstellen

ISO/IEC 9805-1:1994 (CCR Protocol)

ISO/IEC 10026-3:1996 (OSI TP Protocol, Second Edition)

ISO/IEC ISP 12061-1:1995 (OSI TP Taxonomy)

ISO/IEC ISP 12061-2:1995 (OSI TP Support of OSI TP APDUs)

ISO/IEC ISP 12061-3:1995 (OSI TP Support of CCR APDUs)

ISO/IEC ISP 12061-4:1995 (OSI TP Support of Session, Presentation and ACSE PDUs)

ISO/IEC ISP 12061-5:1995 (OSI TP Profile ATP11)

ISO/IEC ISP 12061-7:1995 (OSI TP Profile ATP21)

ISO/IEC ISP 12061-9:1995 (OSI TP Profile ATP31)

X/Open Distributed TP: Reference Model, Version 3, G504 2/96 (X/Open Guide)

X/Open Distributed TP: The XA Specification, C193 2/92

X/Open Distributed TP: The TX (Transaction Demarcation) Specification, C504 4/95

X/Open Distributed TP: The XATMI Specification, C506 11/95

X/Open Distributed TP: The XCPI-C Specification, Version 2, C419 12/95

X/Open ACSE/Presentation: Transaction Processing API (XAP-TP), C409 4/95

DIN-Norm 66 265: Schnittstellen eines Kerns für transaktionsorientierte Anwendungssysteme (KDCS-TAS-Kern)

XAP-TP Schnittstelle

ISO/IEC 9805-1:1994 (CCR Protocol)

ISO/IEC 10026-3:1996 (OSI TP Protocol, Second Edition)

ISO/IEC ISP 12061-1:1995 (OSI TP Taxonomy)

ISO/IEC ISP 12061-2:1995 (OSI TP Support of OSI TP APDUs)

ISO/IEC ISP 12061-3:1995 (OSI TP Support of CCR APDUs)

ISO/IEC ISP 12061-4:1995 (OSI TP Support of Session, Presentation and ACSE PDUs)

ISO/IEC ISP 12061-5:1995 (OSI TP Profile ATP11)

ISO/IEC ISP 12061-6:1995 (OSI TP Profile ATP12)

ISO/IEC ISP 12061-7:1995 (OSI TP Profile ATP21)

ISO/IEC ISP 12061-8:1995 (OSI TP Profile ATP22)

ISO/IEC ISP 12061-9:1995 (OSI TP Profile ATP31)

ISO/IEC ISP 12061-10:1995 (OSI TP Profile ATP32)

X/Open ACSE/Presentation: Transaction Processing API (XAP-TP), C409 4/95

Fachwörter

Ablaufinvariantes Programm
reentrant program
siehe *reentrant-fähiges Programm*.

Abnormale Beendigung einer UTM-Anwendung
abnormal termination of a UTM application
Beendigung einer *UTM-Anwendung*, bei der die *KDCFILE* nicht mehr aktualisiert wird. Eine abnormale Beendigung wird ausgelöst durch einen schwerwiegenden Fehler, z.B. Rechnerausfall, Fehler in der Systemsoftware. Wird die Anwendung erneut gestartet, führt *open*UTM einen *Warmstart* durch.

abstrakte Syntax (OSI)
abstract syntax
Eine abstrakte Syntax ist die Menge der formal beschriebenen Datentypen, die zwischen Anwendungen über *OSI TP* ausgetauscht werden sollen. Eine abstrakte Syntax ist unabhängig von der eingesetzten Hardware und der jeweiligen Programmiersprache.

Access Point (OSI)
siehe *Dienstzugriffspunkt.*

ACID-Eigenschaften
ACID properties
Abkürzende Bezeichnung für die grundlegenden Eigenschaften von *Transaktionen*: Atomicity, Consistency, Isolation und Durability.

Administration
administration
Verwaltung und Steuerung einer *UTM-Anwendung* durch einen *Administrator* oder ein *Administrationsprogramm*.

Administrationskommando
administration command
Kommandos, mit denen der *Administrator* einer *UTM-Anwendung* Administrationsfunktionen für diese Anwendung durchführt. Die Administrationskommandos sind als *Transaktionscodes* realisiert.

Administrationsprogramm
administration program
Teilprogramm, das Aufrufe der *Programmschnittstelle für die Administration* enthält. Dies kann das Standard-Administrationsprogramm *KDCADM* sein, das mit *open*UTM ausgeliefert wird, oder ein vom Anwender selbst erstelltes Programm.

Administrator
administrator
Benutzer mit Administrationsberechtigung.

Akzeptor (CPI-C)
acceptor
Die Kommunikationspartner einer *Conversation* werden *Initiator* und Akzeptor genannt. Der Akzeptor nimmt die vom Initiator eingeleitete Conversation mit Accept_Conversation entgegen.

Anmeldevorgang (KDCS)
sign-on service
Spezieller *Dialog-Vorgang*, bei dem die Anmeldung eines Clients an eine UTM-Anwendung durch *Teilprogramme* gesteuert wird.

Anschlußprogramm
linkage program
siehe *KDCROOT*.

Anwendungsinformation
application information
Sie stellt die Gesamtmenge der von der *UTM-Anwendung* benutzten Daten dar. Dabei handelt es sich um Speicherbereiche und Nachrichten der UTM-Anwendung, einschließlich der aktuell auf dem Bildschirm angezeigten Daten. Arbeitet die UTM-Anwendung koordiniert mit einem Datenbanksystem, so gehören die in der Datenbank gespeicherten Daten ebenfalls zur Anwendungsinformation.

Anwendungsprogramm
application program
Ein Anwendungsprogramm bildet den Hauptbestandteil einer *UTM-Anwendung*. Es besteht aus der Main Routine *KDCROOT* und den *Teilprogrammen*. Es bearbeitet alle Aufträge, die an eine *UTM-Anwendung* gerichtet werden.

Anwendungswiederanlauf
application restart

Start einer *UTM-S*-Anwendung nach einer vorhergehenden abnormalen Beendigung. Dabei wird die *Anwendungsinformation* auf den zuletzt erreichten konsistenten Zustand gesetzt. Unterbrochene *Dialog-Vorgänge* werden dabei auf den zuletzt erreichten *Sicherungspunkt* zurückgesetzt, so daß die Verarbeitung an dieser Stelle wieder konsistent aufgenommen werden kann (*Vorgangswiederanlauf*). Unterbrochene *Asynchron-Vorgänge* werden zurückgesetzt und neu gestartet oder auf den zuletzt erreichten *Sicherungspunkt* zurückgesetzt. Bei UTM-F-Anwendungen werden beim Start nach einer vorhergehenden abnormalen Beendigung lediglich die dynamisch geänderten Konfigurationsdaten auf den zuletzt erreichten konsistenten Zustand gesetzt.

Application Context (OSI)
application context

Der Application Context ist die Menge der Regeln, die für die Kommunikation zwischen zwei Anwendungen gelten sollen. Dazu gehören z.B. die abstrakten Syntaxen und die zugeordneten Transfer-Syntaxen.

Application Entity (OSI)
application entity

Eine Application Entity (AE) repräsentiert alle für die Kommunikation relevanten Aspekte einer realen Anwendung. Eine Application Entity wird durch einen global (d.h. weltweit) eindeutigen Namen identifiziert, den *Application Entity Title* (AET). Jede Application Entity repräsentiert genau einen *Application Process*. Ein Application Process kann mehrere Application Entities umfassen.

Application Entity Title (OSI)
application entity title

Ein Application Entity Title ist ein global (d.h. weltweit) eindeutiger Name für eine *Application Entity.* Er setzt sich zusammen aus dem *Application Process Title* des jeweiligen *Application Process* und dem *Application Entity Qualifier.*

Application Entity Qualifier (OSI)
application entity qualifier

Bestandteil des *Application Entity Titles*. Der Application Qualifier identifiziert einen *Dienstzugriffspunkt* innerhalb der Anwendung. Ein Application Entity Qualifier kann unterschiedlich aufgebaut sein. *open*UTM unterstützt den Typ „Zahl“.

Application Process (OSI)
application process
Der Application Process repräsentiert im *OSI-Referenzmodell* eine Anwendung. Er wird durch den *Application Process Title* global (d.h. weltweit) eindeutig identifiziert.

Application Process Title (OSI)
application process title
Gemäß der OSI-Norm dient der Application Process Title (APT) zur global (d.h. weltweit) eindeutigen Identifizierung von Anwendungen. Er kann unterschiedlich aufgebaut sein. *open*UTM unterstützt den Typ *Object Identifier.*

Application Service Element (OSI)
application service element
Ein Application Service Element (ASE) repräsentiert eine Funktionsgruppe der Anwendungsschicht (Schicht 7) des *OSI-Referenzmodells*.

ASECO (BS2000/OSD)
ASECO (Advanced Security Control) ermöglicht eine erweiterte Zugangskontrolle durch den Einsatz von Chipkarten.

Association (OSI)
association
Eine Association ist eine Kommunikationsbeziehung zwischen zwei *Application Entities*. Dem Begriff Association entspricht der *LU6.1*-Begriff *Session*.

Asynchron-Auftrag
queued job
Auftrag, der vom Auftraggeber zeitlich entkoppelt durchgeführt wird. Zur Bearbeitung von Asynchron-Aufträgen sind in *open*UTM *Message Queuing* Funktionen integriert. Ein Asynchron-Auftrag wird durch die *Asynchron-Nachricht*, den Empfänger und ggf. den gewünschten Ausführungszeitpunkt beschrieben.
Ist der Empfänger ein Terminal oder Drucker, so ist der Asynchronauftrag ein *Ausgabeauftrag*, ist der Empfänger ein Asynchron-Vorgang derselben oder einer fernen Anwendung, so handelt es sich um einen *Hintergrundauftrag*.
Asynchron-Aufträge können *zeitgesteuerte Aufträge* sein oder auch in einen *Auftragskomplex* integriert sein.

Asynchron-Conversation
asynchronous conversation
CPI-C-Conversation, bei der nur der *Initiator* senden darf. Für den *Akzeptor* muß in der *UTM-Anwendung* ein asynchroner Transaktionscode generiert sein.

Asynchron-Nachricht
queued message

Es gibt zwei Arten von Asynchron-Nachrichten:
- Nachricht, die einen lokalen oder fernen *Asynchron-Vorgang* startet (vgl. auch *Hintergrundauftrag*)
- Nachricht, die zur Ausgabe an ein Terminal oder einen Drucker geschickt wird (vgl. auch *Ausgabeauftrag*)

Asynchrone Nachrichten werden von der lokalen *UTM-Anwendung* zunächst in einer *Message Queue* zwischengespeichert. Ihre weitere Verarbeitung erfolgt dann unabhängig vom Auftraggeber.

Asynchron-Programm
asynchronous program

Teilprogramm, das von einem *Hintergrundauftrag* gestartet wird.

Asynchron-Vorgang (KDCS)
asynchronous service

Vorgang, der einen *Hintergrundauftrag* bearbeitet. Die Verarbeitung erfolgt entkoppelt vom Auftraggeber. Ein Asynchron-Vorgang kann aus einem oder mehreren Teilprogrammen/Transaktionen bestehen. Er wird über einen asynchronen *Transaktionscode* gestartet.

ATAC-Auftrag
ATAC job

Begriff ersetzt durch *Hintergrundauftrag*.

Auftrag
job

Anforderung eines *Services*, der von einer *UTM-Anwendung* zur Verfügung gestellt wird, durch Angabe eines Transaktionscodes. Siehe auch: *Ausgabeauftrag*, *Dialogauftrag*, *Hintergrundauftrag*, *Auftragskomplex*.

Auftraggeber-Vorgang
job-submitting service

Ein Auftraggeber-Vorgang ist ein *Vorgang*, der zur Bearbeitung eines Auftrags einen Service von einer anderen Server-Anwendung (*Auftragnehmer-Vorgang*) anfordert.

Auftragnehmer-Vorgang
job-receiving service

Ein Auftragnehmer-Vorgang ist ein *Vorgang*, der von einem *Auftraggeber-Vorgang* einer anderen Server-Anwendung gestartet wird.

Auftragskomplex
job complex

Auftragskomplexe dienen dazu, *Asynchronaufträgen Quittungsaufträge* zuzuordnen. Ein Asynchronauftrag innerhalb eines Auftragkomplexes wird *Basisauftrag* genannt.

Ausgabeauftrag
queued output job

Ausgabeaufträge sind *Asynchronaufträge*, die die Aufgabe haben, eine Nachricht, z.B. ein Dokument, an einen Drucker oder an ein Terminal auszugeben. Ausgabeziel kann aber auch eine andere Anwendung sein, die über die Transportsystem-Schnittstelle angeschlossen wurde.
Ausgabeaufträge werden ausschließlich von UTM-Systemfunktionen bearbeitet, d.h. für die Bearbeitung müssen keine Teilprogramme erstellt werden.

Authentisierung
authentication

siehe *Zugangskontrolle*.

Autorisierung
authorization

siehe *Zugriffskontrolle*.

Basisauftrag
basic job

Asynchronauftrag in einem *Auftragskomplex*.

Basisformat
basic format

Format, in das der Terminal-Benutzer alle Angaben eintragen kann, die notwendig sind, um einen Vorgang zu starten.

Benutzerausgang
user exit

Begriff ersetzt durch *Event-Exit*.

Benutzerkennung
user ID

Bezeichner für einen Benutzer, der in der *Konfiguration* der *UTM-Anwendung* festgelegt ist (optional mit Paßwort zur *Zugangskontrolle*) und dem spezielle Zugriffsrechte (*Zugriffskontrolle*) zugeordnet sind. Ein Terminal-Benutzer oder ein *Client* muß bei der Anmeldung an die UTM-Anwendung diesen Bezeichner (und ggf. das zugeordnete Paßwort) angeben. UTM-Anwendungen können auch ohne Benutzerkennungen generiert werden.

Benutzer-Protokolldatei
user log file

Datei oder Dateigeneration, in die der Benutzer mit dem LPUT-Aufruf Sätze variabler Länge schreibt. Jedem Satz werden die Daten aus dem KB-Kopf des *KDCS-Kommunikationsbereichs* vorangestellt. Die Benutzerprotokolldatei unterliegt der Transaktionssicherung von *open*UTM.

Berechtigungsprüfung
sign-on check

siehe *Zugangskontrolle*.

Beweissicherung (BS2000/OSD)
audit

Im Betrieb einer *UTM-Anwendung* werden zur Beweissicherung sicherheitsrelevante UTM-Ereignisse von *SAT* protokolliert.

Bildschirm-Wiederanlauf
screen restart

Wird ein *Dialog-Vorgang* unterbrochen, gibt *open*UTM beim *Vorgangswiederanlauf* die *Dialog-Nachricht* der letzten abgeschlossenen *Transaktion* erneut auf dem Bildschirm aus.

Bypass-Betrieb (BS2000/OSD)
bypass mode

Betriebsart eines Druckers, der lokal an ein Terminal angeschlossen ist. Im Bypass-Betrieb wird eine an den Drucker gerichtete *Asynchron-Nachricht* an das Terminal gesendet und von diesem auf den Drucker umgeleitet, ohne auf dem Bildschirm angezeigt zu werden.

Cache-Speicher
cache

Pufferbereich zur Zwischenspeicherung von Anwenderdaten für alle Prozesse einer *UTM-Anwendung*. Der Cache-Speicher dient zur Optimierung der Zugriffe auf den *Pagepool*.

CCS-Name (BS2000/OSD)
CCS name

siehe *Coded-Character-Set-Name*.

Client
client

Clients einer *UTM-Anwendung* können sein:
- Terminals
- UPIC-Client-Programme
- Transportsystem-Anwendungen (z.B. DCAM-, PDN-, CMX-, Socket-Anwendungen oder UTM-Anwendungen, die als *Transportsystem-Anwendung* generiert sind)

Clients werden über LTERM-Partner an die UTM-Anwendung angeschlossen.

*open*UTM-Clients mit Trägersystem OpenCPIC werden wie *OSI-TP-Partner* behandelt.

Client-Seite einer Conversation
client side of a conversation

Begriff ersetzt durch *Initiator*.

Coded-Character-Set-Name (BS2000/OSD)
coded character set name

Bei Verwendung des Produkts *XHCS* (e**X**tended **H**ost **C**ode **S**upport) wird jeder verwendete Zeichensatz durch einen Coded-Character-Set-Namen (abgekürzt: „CCS-Name“ oder „CCSN“) eindeutig identifiziert.

Communication Resource Manager
communication resource manager

Communication Resource Manager (CRMs) kontrollieren in verteilten Systemen die Kommunikation zwischen den Anwendungsprogrammen. *open*UTM stellt CRMs für den internationalen Standard OSI TP, für den Industrie-Standard *LU6.1* und für das *open*UTM-eigene Protokoll UPIC zur Verfügung.

Contention Loser
contention loser

Bei der Kommunikation zwischen zwei Server-Anwendungen über LU6.1 ist der Contention Loser der Partner des *Contention Winner*.

Contention Winner
contention winner

Bei der Kommunikation zwischen zwei Server-Anwendungen über LU6.1 ist diejenige Anwendung der Contention Winner, die die Verbindung verwaltet.

Conversation
conversation

Bei CPI-C nennt man die Kommunikation zwischen zwei CPI-C-Anwendungsprogrammen Conversation. Die Kommunikationspartner einer Conversation werden *Initiator* und *Akzeptor* genannt.

Conversation-ID
conversation ID

Jeder *Conversation* wird von CPI-C lokal eine Conversation-ID zugeordnet, d.h., *Initiator* und *Akzeptor* haben jeweils eine eigene Conversation-ID. Mit der Conversation-ID wird jeder CPI-C-Aufruf innerhalb eines Programms eindeutig einer Conversation zugeordnet.

CPI-C

CPI-C (Common Programming Interface for Communication) ist eine von X/Open und dem CIW (**C**PI-C **I**mplementor's **W**orkshop) normierte Programmschnittstelle für die Programm-Programm-Kommunikation in offenen Netzen. Das in *open*UTM implementierte CPI-C genügt der CPI-C V2.0 CAE Specification von X/Open. Die Schnittstelle steht in COBOL und C zur Verfügung. CPI-C in *open*UTM kann über die Protokolle OSI-TP, *LU6.1*, UPIC und mit *open*UTM-LU62 kommunizieren.

Datenstation
terminal

Der Begriff „Datenstation" wird ab *open*UTM V4.0 ersetzt durch „Terminal", „*Client*", „Drucker", „Terminal/Drucker" bzw. „Client/Drucker" - abhängig vom jeweiligen Kontext.

DES

DES (Data Encryption Standard) ist eine internationale Norm zur Verschlüsselung von Daten. Bei diesem Verfahren wird ein Schlüssel zum Ver- und Entschlüsseln verwendet. Bei UTM wird für jede Sitzung vom UPIC-Client ein DES-Schlüssel erzeugt.

Dialog-Auftrag
dialog job, interactive job

Auftrag, der einen *Dialog-Vorgang* startet. Der Auftrag kann von einem *Client* oder - bei *Server-Server-Kommunikation* - von einer anderen Anwendung (Client-Vorgang) erteilt werden.

Dialog-Conversation
dialog conversation

CPI-C-Conversation, bei der sowohl der *Initiator* als auch der *Akzeptor* senden darf. Für den *Akzeptor* muß in der *UTM-Anwendung* ein Dialog-Transaktionscode generiert sein.

Dialog-Nachricht
dialog message

Nachricht, die eine Antwort erfordert oder selbst eine Antwort auf eine Anfrage ist. Dabei liegen Anfrage und Antwort innerhalb eines Vorgangs.

Dialogprogramm
dialog program

Teilprogramm, das einen *Dialogschritt* teilweise oder vollständig bearbeitet.

Dialogschritt
dialog step

Ein Dialogschritt beginnt mit dem Empfang einer *Dialog-Nachricht* durch die *UTM-Anwendung*. Er endet mit der Antwort der UTM-Anwendung.

Dialogterminalprozeß (UNIX/Windows NT)
dialog terminal process

Ein Dialogterminalprozeß verbindet ein UNIX/Windows NT - Terminal mit den *Workprozessen* der *UTM-Anwendung*. Dialogterminalprozesse werden entweder vom Benutzer durch Eingabe von utmdtp oder über die LOGIN-Shell gestartet. Für jedes Terminal, das an eine UTM-Anwendung angeschlossen werden soll, ist ein eigener Dialogterminalprozeß erforderlich.

Dialog-Vorgang
dialog service

Vorgang, der einen *Auftrag* im Dialog (zeitlich gekoppelt) mit dem Auftraggeber bearbeitet. Ein Dialog-Vorgang verarbeitet *Dialog-Nachrichten* vom Auftraggeber und erzeugt Dialog-Nachrichten für diesen. Ein Dialog-Vorgang besteht aus mindestens einer *Transaktion*. Ein Dialog-Vorgang umfaßt in der Regel mindestens einen *Dialogschritt*. Ausnahme: Bei *Vorgangskettung* können auch mehrere Vorgänge einen Dialogschritt bilden.

Dienst
service

Programm, das im Hintergrund unabhängig von angemeldeten Benutzern oder Fenstern abläuft.

Dienstzugriffspunkt
service access point

Im OSI-Referenzmodell stehen einer Schicht am Dienstzugriffspunkt die Leistungen der darunterliegenden Schicht zur Verfügung. Der Dienstzugriffspunkt wird im lokalen System durch einen *Selektor* identifiziert. Bei der Kommunikation bindet sich die *UTM-Anwendung* an einen Dienstzugriffspunkt. Eine Verbindung wird zwischen zwei Dienstzugriffspunkten aufgebaut.

Distributed Transaction Processing

X/Open-Architekturmodell für die transaktionsorientierte *verteilte Verarbeitung*.

Druckadministration
print administration

Funktionen zur *Drucksteuerung* und Administration von *Ausgabeaufträgen*, die an einen Drucker gerichtet sind.

Druckerbündel
printer pool

Mehrere Drucker, die demselben *LTERM-Partner* zugeordnet sind.

Druckergruppe (UNIX)
printer group

UNIX richtet für jeden Drucker standardmäßig eine Druckergruppe ein, die genau diesen Drucker enthält. Darüber hinaus lassen sich mehrere Drucker einer Druckergruppe, aber auch ein Drucker mehreren Druckergruppen zuordnen.

Druckerprozeß (UNIX)
printer process

Prozeß, der vom *Main-Prozeß* zur Ausgabe von *Asynchron-Nachrichten* an eine *Druckergruppe* eingerichtet wird. Er existiert, solange die Druckergruppe an die *UTM-Anwendung* angeschlossen ist. Pro angeschlossener Druckergruppe gibt es einen Druckerprozeß.

Druckersteuerstation
printer control terminal

Begriff ersetzt durch *Druckersteuer-LTERM*

Druckersteuer-LTERM
printer control LTERM
Über ein Druckersteuer-LTERM kann sich ein *Client* oder ein Terminal-Benutzer an eine *UTM-Anwendung* anschließen. Von dem Client-Programm oder Terminal aus kann dann die *Administration* der Drucker erfolgen, die dem Druckersteuer-LTERM zugeordnet sind. Hierfür ist keine Administrationsberechtigung notwendig.

Drucksteuerung
print control
*open*UTM-Funktionen zur Steuerung von Druckausgaben.

Dynamische Konfigurierung
dynamic configuration
Änderung der *Konfiguration* durch die Administration. Im laufenden Betrieb der Anwendung können *Teilprogramme*, *Transaktionscodes*, *Clients*, Drucker und *Benutzerkennungen* in die Konfiguration aufgenommen oder gelöscht werden. Hierzu kann das Administrationsprogramm WinAdmin verwendet werden, oder es müssen eigene *Administrationsprogramme* erstellt werden, die die Funktionen der *Programmschnittstelle der Administration* nutzen.

Einschritt-Transaktion
single-step transaction
Transaktion, die genau einen *Dialogschritt* umfaßt.

Einschritt-Vorgang
single-step service
Dialog-Vorgang, der genau einen *Dialogschritt* umfaßt.

Ereignisgesteuerter Vorgang
event-driven service
Begriff ersetzt durch *Event-Service*.

Event-Exit
event exit
Routine des *Anwendungsprogramms*, das bei bestimmten Ereignissen (z.B. Start eines Prozesses, Ende eines Vorgangs) automatisch gestartet wird. Diese darf - im Gegensatz zu den *Event-Services* - keine KDCS-, CPI-C- und XATMI-Aufrufe enthalten.

Event-Funktion
event function
Oberbegriff für *Event-Exits* und *Event-Services*.

Event-Service
event service
Vorgang, der beim Auftreten bestimmter Ereignisse gestartet wird, z.B. bei bestimmten *open*UTM-Meldungen. Die *Teilprogramme* ereignisgesteuerter Vorgänge müssen KDCS-Aufrufe enthalten.

Generierung
generation
Statische Konfigurierung einer *UTM-Anwendung* mit dem UTM-Tool KDCDEF und Erzeugen des Anwendungsprogramms.
Bei der Generierung entstehen die *KDCFILE* und die Source für das Tabellenmodul der Main-Routine *KDCROOT.*

Globaler Name
Symbolischer Name, den *TNS* zur Adressierung einer Anwendung benötigt. Unter dem globalen Namen werden die Adressen der lokalen als auch fernen Anwendungen in der *TNS*-Datenbasis abgelegt. Der globale Name der lokalen Anwendung ist der Anwendungsname (BCAMAPPL-Name, Name des OSI-TP-Zugriffspunkts). Der globale Name einer fernen Anwendung setzt sich zusammen aus Stationsname und Prozessorname.

Globaler Sekundärer Speicherbereich/GSSB
global secondary storage area
siehe *Sekundärspeicherbereich.*

Globale Transaktion
global transaction
Transaktion, die sich über mehr als eine Anwendung erstreckt.

Hardcopy-Betrieb
hardcopy mode
Betriebsart eines Druckers, der lokal an ein Terminal angeschlossen ist. Dabei wird eine Nachricht, die auf dem Bildschirm angezeigt wird, zusätzlich auf dem Drucker abgedruckt.

Heterogene Kopplung
heterogeneous link
Bei *Server-Server-Kommunikation:* Kopplung einer *UTM-Anwendung* mit einer Nicht-UTM-Anwendung, z.B. einer CICS- oder TUXEDO-Anwendung.

Hintergrundauftrag
background job

Hintergrund-Aufträge sind *Asynchron-Aufträge*, die an einen *Asynchron-Vorgang* der eigenen oder einer fernen Anwendung gerichtet sind. Hintergrund-Aufträge eignen sich besonders für zeitintensive oder zeitunkritische Verarbeitungen, deren Ergebnis keinen direkten Einfluß auf den aktuellen Dialog hat.

Homogene Kopplung
homogeneous link

Bei *Server-Server-Kommunikation*: Kopplung von *UTM-Anwendungen*. Dabei spielt es keine Rolle, ob die Anwendungen auf der gleichen oder auf unterschiedlichen Betriebssystem-Plattformen ablaufen.

Inbound-Conversation (CPI-C)
inbound conversation

siehe *Incoming-Conversation*.

Incoming-Conversation (CPI-C)
incoming conversation

Eine *Conversation*, bei der das lokale CPI-C-Programm *Akzeptor* ist, heißt Incoming-Conversation. In der X/Open-Specification wird für Incoming-Conversation auch das Synonym Inbound-Conversation verwendet.

Initiator (CPI-C)
initiator

Die Kommunikationspartner einer *Conversation* werden Initiator und *Akzeptor* genannt. Der Initiator baut die Conversation mit den CPI-C-Aufrufen Initialize_Conversation und Allocate auf.

Insert
insert

Feld in einem Meldungstext, in das *open*UTM aktuelle Werte einträgt.

Inverser KDCDEF
invers KDCDEF

Funktion, die aus den Konfigurationsdaten der *KDCFILE*, die im laufenden Betrieb dynamisch angepaßt wurde, Steueranweisungen für einen *KDCDEF*-Lauf erzeugt. Der inverse KDCDEF kann „offline“ unter KDCDEF oder „online“ über die *Programmschnittstelle zur Administration* gestartet werden.

Kaltstart
cold start

Starten einer *UTM-Anwendung* nach einer *normalen Beendigung* der Anwendung oder nach einer Neugenerierung (vgl. auch *Warmstart*).

KDCADM

Standard-Administrationsprogramm, das zusammen mit *open*UTM ausgeliefert wird. KDCADM stellt Administrationsfunktionen zur Verfügung, die über Transaktionscodes (*Administrationskommandos*) aufgerufen werden.

KDCDEF

UTM-Tool für die *Generierung* von *UTM-Anwendungen*. KDCDEF erstellt anhand der Konfigurationsinformationen in den KDCDEF-Steueranweisungen die UTM-Objekte *KDCFILE* und die ROOT-Tabellen-Source für die Main Routine *KDCROOT*.

KDCFILE

Eine oder mehrere Dateien, die für den Ablauf einer *UTM-Anwendung* notwendige Daten enthalten. Die KDCFILE wird mit dem UTM-Generierungstool *KDCDEF* erstellt. Die KDCFILE enthält unter anderem die *Konfiguration* der Anwendung.

KDCROOT

Main Routine eines *Anwendungsprogramms*, die das Bindeglied zwischen *Teilprogrammen* und UTM-Systemcode bildet. KDCROOT wird zusammen mit den *Teilprogrammen* zum *Anwendungsprogramm* gebunden.

KDCSHARE-Tabellen (BS2000/OSD)

KDCSHARE tables

KDCSHARE-Tabellen sind eine Methode, die *open*UTM bietet, um Teile des Anwendungscodes shareable zu laden. Die Assembler-Quelldateien für die KDCSHARE-Tabellen werden vom UTM-Tool KDCDEF erzeugt.

KDCS-Parameterbereich

KDCS parameter area

siehe *Parameterbereich*.

KDCS-Programmschnittstelle

KDCS program interface

Universelle UTM-Programmschnittstelle, die den nationalen Standard DIN 66 265 erfüllt und Erweiterungen enthält. Mit KDCS (Kompatible Datenkommunikationsschnittstelle) lassen sich z.B. Dialog-Services erstellen und *Message Queuing* Funktionen nutzen. Außerdem stellt KDCS Aufrufe zur *verteilten Verarbeitung* zur Verfügung.

Kommunikationsbereich/KB (KDCS)
communication area

Transaktionsgesicherter KDCS-*Primärspeicherbereich*, der vorgangsspezifische Daten enthält. Der Kommunikationsbereich besteht aus 3 Teilen:

– dem KB-Kopf mit allgemeinen Vorgangsdaten,
– dem KB-Rückgabebereich für Rückgaben nach KDCS-Aufrufen
– dem KB-Programmbereich zur Datenübergabe zwischen UTM-Teilprogrammen innerhalb eines *Vorgangs*.

Konfiguration
configuration

Summe aller Eigenschaften einer *UTM-Anwendung*. Die Konfiguration beschreibt:

– Anwendungs- und Betriebsparameter
– die Objekte der Anwendung und die Eigenschaften dieser Objekte. Objekte sind z.B. *Teilprogramme* und *Transaktionscodes*, Kommunikationspartner, Drucker, *Benutzerkennungen*
– definierte Zugriffsschutz- und Zugangsschutzmaßnahmen

Die Konfiguration einer UTM-Anwendung wird bei der Generierung festgelegt und kann per *Administration* dynamisch (während des Anwendungslaufs) geändert werden. Die Konfiguration ist in der *KDCFILE* abgelegt.

Konfigurierung
configuration

Festlegen der *Konfiguration* der UTM-Anwendung. Es wird unterschieden zwischen *statischer* und *dynamischer Konfigurierung*.

Logische Verbindung
virtual connection

Zuordnung zweier Kommunikationspartner.

Lokaler Sekundärer Speicherbereich/LSSB
local secondary storage area

siehe *Sekundärspeicherbereich.*

LPAP-Partner
LPAP partner

Für die *verteilte Verarbeitung* über das *LU6.1*-Protokoll muß in der lokalen Anwendung für jede Partneranwendung ein LPAP-Partner konfiguriert werden. Der LPAP-Partner spiegelt in der lokalen Anwendung die Partneranwendung wider. Bei der Kommunikation wird die Partneranwendung nicht über ihren Anwendungsnamen oder ihre Adresse, sondern über den Namen des zugeordneten LPAP-Partners angesprochen.

LTERM-Partner
LTERM partner

Um *Clients* oder Drucker an eine *UTM-Anwendung* anschließen zu können, müssen in der Anwendung LTERM-Partner konfiguriert werden. Ein Client oder Drucker kann nur angeschlossen werden, wenn ihm ein LTERM-Partner mit entsprechenden Eigenschaften zugeordnet ist. Diese Zuordnung wird i.a. bei der Konfigurierung festgelegt, sie kann aber auch dynamisch über Terminal-Pools erfolgen.

LTERM-Pool
LTERM pool

Statt für jeden *Client* eine LTERM- und eine PTERM-Anweisung anzugeben, kann mit der Anweisung TPOOL ein Pool von LTERM-Partnern definiert werden. Schließt sich ein Client über einen LTERM-Pool an, wird ihm dynamisch ein LTERM-Partner aus dem Pool zugeordnet.

LU6.1

Geräteunabhängiges Datenaustauschprotokoll (Industrie-Standard) für die transaktionsgesicherte *Server-Server-Kommunikation*.

Main-Prozeß (UNIX/Windows NT)
main process

Prozeß, der die *UTM-Anwendung* startet. Er startet außerdem die *Workprozesse*, die *Druckerprozesse, Netzprozesse* und den *Timerprozeß* und überwacht und steuert die *UTM-Anwendung*.

Main Routine KDCROOT
main routine KDCROOT

siehe *KDCROOT*.

Meldung
UTM message

Meldungen werden von der Transaktionsmonitor *open*UTM oder von UTM-Tools (wie z.B. *KDCDEF*) an *Meldungsziele* ausgegeben. Eine Meldung besteht aus einer Meldungsnummer und dem Meldungstext, der ggf. *Inserts* mit aktuellen Werten enthält. Je nach Meldungsziel werden entweder die gesamte Meldung oder nur Teile der Meldung (z.B. nur die Inserts) ausgegeben.

Meldungsdefinitionsdatei
UTM message definition file

Die Meldungsdefinitionsdatei wird mit *open*UTM ausgeliefert und enthält standardmäßig die UTM-Meldungstexte in deutscher und englischer Sprache und die Definitionen der Meldungseigenschaften. Aufbauend auf diese Datei kann der Anwender auch eigene, individuelle Meldungsmodule erzeugen.

Meldungsziel
UTM message destination
: Ausgabemedium für eine *Meldung*. Mögliche Meldungsziele einer Meldung sind z.B. Terminals, der *Event-Service* MSGTAC oder die *System-Protokolldatei* SYSLOG.

Mehrschritt-Transaktion
multi-step transaction
: *Transaktion*, die aus mehr als einem *Verarbeitungsschritt* besteht.

Mehrschritt-Vorgang (KDCS)
multi-step service
: *Vorgang*, der in mehreren *Dialogschritten* ausgeführt wird.

Message Queuing
message queuing
: Message Queuing (MQ) ist eine Form der Kommunikation, bei der die Nachrichten (Messages) nicht unmittelbar, sondern über zwischengeschaltete Queues ausgetauscht werden. Sender und Empfänger können zeitlich und räumlich entkoppelt ablaufen, die Übermittlung der Nachricht wird garantiert, unabhängig davon, ob gerade eine Netzverbindung besteht oder nicht.
In *open*UTM sind mit dem Konzept der asynchronen Verarbeitung Message Queuing Funktionen integriert. Vgl. auch *Asynchron-Auftrag*, *Asynchron-Nachricht*.

Message Queue
message queue
: Warteschlange für *Asynchron-Nachrichten.*

Multiplexanschluß (BS2000/OSD)
multiplex connection
: Spezielle Möglichkeit, Terminals an eine *UTM-Anwendung* anzuschließen. Ein Multiplexanschluß ermöglicht es, daß sich mehrere Terminals eine *Transportverbindung* teilen.

Nachrichtenbereich/NB (KDCS)
KDCS message area
: Bei KDCS-Aufrufen: Puffer-Bereich, in dem Nachrichten oder Daten für *open*UTM oder für das *Teilprogramm* bereitgestellt werden.

Nachrichtenverteiler (BS2000/OSD)
message router

Einrichtung in einem zentralen Rechner oder Kommunikationsrechner zur Verteilung von Eingabenachrichten an unterschiedliche *UTM-Anwendungen*, die auf unterschiedlichen Rechnern liegen können. Der Nachrichtenverteiler ermöglicht außerdem, mit *Multiplexanschlüssen* zu arbeiten.

Netz-Prozeß (UNIX/Windows NT)
net process

Prozeß einer *UTM-Anwendung* zur Netzanbindung.

Netzwerk-Selektor
network selector

Der Netzwerk-Selektor identifiziert im lokalen System einen *Dienstzugriffspunkt* zur Vermittlungsschicht des *OSI-Referenzmodells*.

Normale Beendigung einer UTM-Anwendung
normal termination of a UTM application

Kontrollierte Beendigung einer *UTM-Anwendung*; das bedeutet u.a., daß die Verwaltungsdaten auf der *KDCFILE* aktualisiert werden. Eine normale Beendigung veranlaßt der *Administrator* (z.B. mit KDCSHUT N). Den Start nach einer normalen Beendigung führt *open*UTM als *Kaltstart* durch.

Object Identifier
object identifier

Ein Object Identifier ist ein global (d.h. weltweit) eindeutiger Bezeichner für Objekte im OSI-Umfeld. Ein Object Identifier besteht aus einer Folge von ganzen Zahlen, die einen Pfad in einer Baumstruktur repräsentiert.

Offener Terminalpool
open terminal pool

Terminalpool, der nicht auf *Clients* eines Rechners oder eines bestimmten Typs beschränkt ist. An diesen Terminalpool können sich alle Clients anschließen, für die kein rechner- oder typspezifischer Terminalpool generiert ist.

***open*UTM-Anwendung**
*open*UTM application

siehe *UTM-Anwendung*.

***open*UTM-D**

*open*UTM-D (*open*UTM-Distributed) ist eine *open*UTM-Komponente, die *verteilte Verarbeitung* ermöglicht. *open*UTM-D ist integraler Bestandteil von *open*UTM.

OSI-LPAP-Partner
OSI-LPAP partner

OSI-LPAP-Partner sind die bei *open*UTM generierten Adressen der *OSI-TP-Partner*. Für die *verteilte Verarbeitung* über das Protokoll *OSI TP* muß in der lokalen Anwendung für jede Partneranwendung ein OSI-LPAP-Partner konfiguriert werden. Der OSI-LPAP-Partner spiegelt in der lokalen Anwendung die Partneranwendung wider. Bei der Kommunikation wird die Partneranwendung nicht über ihren Anwendungsnamen oder ihre Adresse, sondern über den Namen des zugeordneten OSI-LPAP-Partners angesprochen.

OSI-Referenzmodell
OSI reference model

Das OSI-Referenzmodell stellt einen Rahmen für die Standardisierung der Kommunikation von offenen Systemen dar. ISO, die Internationale Organisation für Standardisierung, hat dieses Modell im internationalen Standard ISO IS7498 beschrieben. Das OSI-Referenzmodell unterteilt die für die Kommunikation von Systemen notwendigen Funktionen in sieben logische Schichten. Diese Schichten haben jeweils klar definierte Schnittstellen zu den benachbarten Schichten.

OSI TP

Von der ISO definiertes Kommunikationsprotokoll für die verteilte Transaktionsverarbeitung. OSI TP steht für Open System Interconnection Transaction Processing.

OSI-TP-Partner
OSI-TP partner

Partner der *open*UTM-Anwendung, der mit der *open*UTM-Anwendung über das OSI-TP-Protokoll kommuniziert.
Beispiele für solche Partner sind:

- eine *open*UTM-Anwendung, die über OSI TP kommuniziert
- eine Anwendung im IBM-Umfeld (z.B. CICS), die über *open*UTM-LU62 angeschlossen ist
- eine Anwendung des Trägersystems OpenCPIC des *open*UTM-Client
- Anwendungen anderer TP-Monitore, die OSI TP unterstützen

Outbound-Conversation (CPI-C)
outbound conversation

siehe *Outgoing-Conversation*.

Outgoing-Conversation (CPI-C)
outgoing conversation

Eine Conversation, bei der das lokale CPI-C-Programm der *Initiator* ist, heißt Outgoing-Conversation. In der X/Open-Specification wird für Outgoing-Conversation auch das Synonym Outbound-Conversation verwendet.

Pagepool
page pool

Teil der *KDCFILE*, in dem Anwenderdaten gespeichert werden, z.B. *Dialog-Nachrichten*, *Asynchron-Nachrichten*, *Sekundärspeicherbereiche*.

Parameterbereich
parameter area

Datenstruktur, in der ein *Teilprogramm* bei einem UTM-Aufruf die für diesen Aufruf notwendigen Operanden an *open*UTM übergibt.

Postselection (BS2000/OSD)
postselection

Auswahl der protokollierten UTM-Ereignisse aus der SAT-Protokolldatei, die ausgewertet werden sollen. Die Auswahl erfolgt mit Hilfe des Tools SATUT.

Prädialog (BS2000/OSD)
predialog

Aufforderung des Terminal-Benutzers an das Datenkommunikationssystem zum Aufbau einer *logischen Verbindung* zur *Anwendung*. Der Prädialog entfällt, wenn die logische Verbindung auf Initiative der Anwendung aufgebaut wird.

Preselection (BS2000/OSD)
preselection

Festlegung der für die *SAT-Beweissicherung* zu protokollierenden UTM-Ereignisse. Die Preselection erfolgt durch die UTM-SAT-Administration. Man unterscheidet ereignisspezifische, benutzerspezifische und auftrags-(TAC-)spezifische Preselection.

Presentation-Selektor
presentation selector

Der Presentation-Selektor identifiziert im lokalen System einen *Dienstzugriffspunkt* zur Darstellungsschicht des *OSI-Referenzmodells*.

Primärspeicherbereich
primary storage area

Bereich im Arbeitsspeicher, auf den das *KDCS-Teilprogramm* direkt zugreifen kann, z.B. *Standard Primärer Arbeitsbereich*, *Kommunikationsbereich*.

Printerprozeß (UNIX)
printer process
siehe *Druckerprozeß*.

Programmschnittstelle zur Administration
program interface for administration
UTM-Programmschnittstelle, mit deren Hilfe der Anwender eigene *Administrationsprogramme* erstellen kann. Die Programmschnittstelle zur Administration bietet u.a. Funktionen zur *dynamischen Konfigurierung*, zur Modifikation von Eigenschaften und Anwendungsparametern und zur Abfrage von Informationen zur *Konfiguration* und zur aktuellen Auslastung der Anwendung.

Prozeß
process
In den *open*UTM-Handbüchern wird der Begriff „Prozeß“ als Oberbegriff für Prozeß (UNIX/Windows NT) und Task (BS2000/OSD) verwendet.

QuickStart Kit
Beispielanwendung, die mit *open*UTM (Windows NT) ausgeliefert wird.

Quittungsauftrag
confirmation job
Bestandteil eines *Auftragskomplexes*, worin der Quittungsauftrag dem *Basisauftrag* zugeordnet ist. Es gibt positive und negative Quittungsaufträge. Bei positivem Ergebnis des *Basisauftrags* wird der positive Quittungsauftrag wirksam, sonst der negative.

Reentrant-fähiges Programm
reentrant program
Programm, dessen Code durch die Ausführung nicht verändert wird.
Im BS2000/OSD ist dies Voraussetzung dafür, *Shared Code* zu nutzen.

Request
request
Anforderung einer *Service-Funktion* durch einen *Client* oder einen anderen Server.

Requestor
requestor
In XATMI steht der Begriff Requestor für eine Anwendung, die einen Service aufruft.

Resource Manager
resource manager

Resource Manager (RMs) verwalten Datenressourcen. Ein Beispiel für RMs sind Datenbank-Systeme. *open*UTM stellt aber auch selbst Resource Manager zur Verfügung, z.B. für den Zugriff auf *Message Queues*, lokale Speicherbereiche und Logging-Dateien. Anwendungsprogramme greifen auf RMs über RM-spezifische Schnittstellen zu. Für Datenbank-Systeme ist dies meist SQL, für die *open*UTM-RMs die Schnittstelle KDCS.

RFC1006

Von IETF (Internet Engeneering Task Force) definiertes Protokoll der TCP/IP-Familie zur Realisierung der ISO-Transportdienste (Transportklasse 0) auf TCP/IP-Basis.

RSA

Abkürzung für die Erfinder dieses Verschlüsselungsverfahrens Rivest, Shamir und Adleman. Bei diesem Verfahren wird ein Schlüsselpaar verwendet, das aus einem öffentlichen und einem privaten Schlüssel besteht. Eine Nachricht wird mit dem öffentlichen Schlüssel verschlüsselt und kann nur mit dem privaten Schlüssel entschlüsselt werden. Das RSA-Schlüsselpaar wird von der UTM-Anwendung erzeugt.

SAT-Beweissicherung (BS2000/OSD)
SAT audit

Beweissicherung durch die Komponente SAT (Security Audit Trail) des BS2000-Softwareproduktes SECOS.

Sekundärspeicherbereich
secondary storage area

Transaktionsgesicherter Speicherbereich auf der *KDCFILE*, auf den das KDCS-*Teilprogramm* mit speziellen Aufrufen zugreifen kann. Lokale Sekundärspeicherbereiche (LSSB) sind einem *Vorgang* zugeordnet, auf globale Sekundärspeicherbereiche (GSSB) kann von allen Vorgängen einer *UTM-Anwendung* zugegriffen werden. Weitere Sekundärspeicherbereiche sind der *Terminalspezifische Langzeitspeicher (TLS)* und der *Userspezifische Langzeitspeicher (ULS)*.

Selektor
selector

Ein Selektor identifiziert im lokalen System einen *Zugriffspunkt* auf die Dienste einer Schicht des *OSI-Referenzmodells*. Jeder Selektor ist Bestandteil der Adresse des Zugriffspunktes.

Semaphor (UNIX/Windows NT)
semaphore
: UNIX- und Windows NT-Betriebsmittel, das zur Steuerung und Synchronisation von Prozessen dient.

Server
server
: Ein Server ist eine *Anwendung*, die *Services* zur Verfügung stellt. Oft bezeichnet man auch den Rechner, auf dem Server-Anwendungen laufen, als Server.

Server-Seite einer Conversation (CPI-C)
server side of a conversation
: Begriff ersetzt durch *Akzeptor*.

Server-Server-Kommunikation
server-server communication
: siehe *verteilte Verarbeitung*.

Service Access Point
: siehe *Dienstzugriffspunkt.*

Service
service
: Services sind Dienstleistungen, die von *Servern* erbracht werden. Diese Services können von Clients oder anderen Servern aufgerufen werden.

Service Routine
service routine
: siehe *Teilprogramm*.

Session
session
: Kommunikationsbeziehung zweier adressierbarer Einheiten im Netz über das SNA-Protokoll *LU6.1*.

Session-Selektor
session selector
: Der Session-Selektor identifiziert im lokalen System einen *Zugriffspunkt* zu den Diensten der Kommunikationssteuerschicht (Session-Layer) des *OSI-Referenzmodells*.

Shared Code (BS2000/OSD)
shared code
: Code, der von mehreren Prozessen gleichzeitig benutzt werden kann.

Shared Memory
shared memory
: Virtueller Speicherbereich, auf den mehrere Prozesse gleichzeitig zugreifen können.

Shared Objects (UNIX/Windows NT)
shared objects
: Teile des *Anwendungsprogramms* können als Shared Objects erzeugt werden. Diese werden dynamisch zur Anwendung dazugebunden und können im laufenden Betrieb ausgetauscht werden. Shared Objects werden mit der KDCDEF-Anweisung SHARED-OBJECT definiert.

Sicherungspunkt
synchronization point, consistency point
: Ende einer *Transaktion*. Zu diesem Zeitpunkt werden alle in der Transaktion vorgenommenen Änderungen der *Anwendungsinformation* gegen Systemausfall gesichert und für andere sichtbar gemacht. Während der Transaktion gesetzte Sperren werden wieder aufgehoben.

Standard Primärer Arbeitsbereich/SPAB (KDCS)
standard primary working area
: Bereich im Arbeitsspeicher, der jedem KDCS-*Teilprogramm* zur Verfügung steht. Sein Inhalt ist zu Beginn des Teilprogrammlaufs undefiniert oder mit einem Füllzeichen vorbelegt.

Startformat
start format
: Format, das *open*UTM am Terminal ausgibt, wenn sich ein Benutzer erfolgreich bei der *UTM-Anwendung* angemeldet hat (ausgenommen nach *Vorgangswiederanlauf* und beim *Anmeldevorgang*).

statische Konfigurierung
static configuration
: Festlegen der *Konfiguration* bei der Generierung mit Hilfe des UTM-Tools *KDCDEF*.

SYSLOG-Datei
SYLOG file
: siehe *System-Protokolldatei*.

System-Protokolldatei
system log file

Datei oder Dateigeneration, in die *open*UTM während des Laufs einer *UTM-Anwendung* alle UTM-Meldungen protokolliert, für die das *Meldungsziel* SYSLOG definiert ist.

TAC
TAC

siehe *Transaktionscode*.

Teilprogramm
program unit

UTM-*Services* werden durch ein oder mehrere Teilprogramme realisiert. Die Teilprogramme sind Bestandteile des *Anwendungsprogramms*. Sie enthalten UTM-Funktionsaufrufe und sind über *Transaktionscodes* ansprechbar. Einem Teilprogramm können mehrere Transaktionscodes zugeordnet werden.

Terminalspezifischer Langzeitspeicher/TLS (KDCS)
terminal-specific long-term storage

Sekundärspeicher, der einem *LTERM-, LPAP- oder OSI-LPAP-Partner* zugeordnet ist und über das Anwendungsende hinaus erhalten bleibt.

Timerprozeß (UNIX/Windows NT)
timer process

Prozeß, der Aufträge zur Zeitüberwachung von *Workprozessen* entgegennimmt, sie in ein Auftragsbuch einordnet und nach einer im Auftragsbuch festgelegten Zeit den Workprozessen zur Bearbeitung wieder zustellt.

TNS

Abkürzung für den Transport Name Service, der einem Anwendungsnamen einen Transport-Selektor und das Transportsystem zuordnet, über das die Anwendung erreichbar ist.

Transaktion
transaction

Verarbeitungsabschnitt innerhalb eines *Services*, der die *ACID-Eigenschaften* aufweist. Von den in einer Transaktion beabsichtigten Änderungen der *Anwendungsinformation* werden entweder alle konsistent durchgeführt oder es wird keine durchgeführt (Alles-oder-Nichts Regel). Das Transaktionsende bildet einen *Sicherungspunkt*.

Transaktionscode/TAC
transaction code
: Name, über den ein *Teilprogramm* aufgerufen werden kann. Der Transaktionscode wird dem Teilprogramm bei der *statischen* oder *dynamischen Konfigurierung* zugeordnet. Einem Teilprogramm können auch mehrere Transaktionscodes zugeordnet werden.

Transaktionsrate
transaction rate
: Anzahl der erfolgreich beendeten *Transaktionen* pro Zeiteinheit.

Transfer-Syntax
transfer syntax
: Bei *OSI TP* werden die Daten zur Übertragung zwischen zwei Rechnersystemen von der lokalen Darstellung in die Transfersyntax umgewandelt. Die Transfersyntax beschreibt die Daten in einem neutralen Format, das von allen beteiligten Partnern verstanden wird. Jeder Transfersyntax muß ein *Object Identifier* zugeordnet sein.

Transport-Selektor
transport selector
: Der Transport-Selektor identifiziert im lokalen System einen *Dienstzugriffspunkt* zur Transportschicht des *OSI-Referenzmodells*.

Transportsystem-Anwendung
transport system application
: Anwendung, die direkt auf der Transportsystem-Schnittstelle (z.B. CMX) aufsetzt. Für den Anschluß von Transportsystem-Anwendungen muß bei der Konfigurierung als Partnertyp APPLI angegeben werden. Beim Anschluß von Transportsystem-Anwendungen ist keine globale Transaktionssicherung möglich.

TS-Anwendung
TS application
: siehe *Transportsystem-Anwendung*.

Typisierter Puffer (XATMI)
typed buffer
: Puffer für den Austausch von typisierten und strukturierten Daten zwischen Kommunikationspartnern. Durch diese typisierten Puffer ist die Struktur der ausgetauschten Daten den Partnern implizit bekannt.

UPIC

Trägersystem für *open*UTM-Clients. UPIC steht für Universal Programming Interface for Communication.

UPIC-Client

Bezeichnung für *open*UTM-Clients mit Trägersystem UPIC

Userspezifischer Langzeitspeicher/ULS

user-specific long-term storage

Sekundärspeicher, der einer *Benutzerkennung*, einer *Session* oder einer *Association* zugeordnet ist und über das Anwendungsende hinaus erhalten bleibt.

USLOG-Datei

USLOG file

siehe *Benutzer-Protokolldatei*.

UTM-Anwendung

UTM application

Eine UTM-Anwendung stellt *Services* zur Verfügung, die Aufträge von *Clients* oder anderen *Server*-Anwendungen bearbeiten. *open*UTM übernimmt dabei u.a. die Transaktionssicherung und das Management der Kommunikations- und Systemressourcen. Technisch gesehen ist eine UTM-Anwendung eine Prozeßgruppe, die zur Laufzeit eine logische Server-Einheit bildet.

UTM-D

siehe *openUTM-D*.

UTM-Datenstation

UTM terminal

Begriff ersetzt durch *LTERM-Partner*.

UTM-F

UTM-Anwendungen können als UTM-F-Anwendungen (UTM-Fast) generiert werden. Bei UTM-F wird zugunsten der Performance auf Platteneingaben/-ausgaben verzichtet, mit denen bei *UTM-S* die Sicherung von Benutzer- und Transaktionsdaten durchgeführt wird. Gesichert werden lediglich Änderungen der Verwaltungsdaten.

UTM-S

Bei UTM-S-Anwendungen sichert *open*UTM neben den Verwaltungsdaten auch alle Benutzerdaten über ein Anwendungsende und einen Systemausfall hinaus. Außerdem garantiert UTM-S bei allen Störungen die Sicherheit und Konsistenz der Anwendungsdaten. Im Standardfall werden UTM-Anwendungen als UTM-S-Anwendungen (UTM-Secure) generiert.

UTM-SAT-Administration (BS2000/OSD)
UTM SAT administration

Durch die UTM-SAT-Administration wird gesteuert, welche sicherheitsrelevanten UTM-Ereignisse, die im Betrieb der *UTM-Anwendung* auftreten, von *SAT* protokolliert werden sollen. Für die UTM-SAT-Administration wird eine besondere Berechtigung benötigt.

UTM-Seite
UTM page

Ist eine Speichereinheit, die entweder 2K oder 4K umfaßt. Die Größe wird bei der Generierung der *UTM-Anwendung* festgelegt. *Pagepool* und Wiederanlauf-Bereich der KDCFILE werden in Einheiten der Größe einer UTM-Seite unterteilt.

Verarbeitungsschritt
processing step

Ein Verarbeitungsschritt beginnt mit dem Empfangen einer *Dialog-Nachricht*, die von einem *Client* oder einer anderen Server-Anwendung an die *UTM-Anwendung* gesendet wird. Der Verarbeitungsschritt endet entweder mit dem Senden einer Antwort und beendet damit auch den *Dialogschritt* oder er endet mit dem Senden einer Dialog-Nachricht an einen Dritten.

Verschlüsselungsstufe
encryption level

Die Verschlüsselungsstufe legt fest, ob und inwieweit ein Client Nachrichten und Paßwort verschlüsseln muß.

Verteilte Transaktion
distributed transaction

siehe *globale Transaktion*.

Verteilte Transaktionsverarbeitung
Distributed Transaction Processing

Verteilte Verarbeitung mit *globalen Transaktionen*.

Verteilte Verarbeitung
distributed processing

Bearbeitung von *Dialogaufträgen* durch mehrere Anwendungen oder Übermittlung von *Hintergrundaufträgen* an eine andere Anwendung. Für die verteilte Verarbeitung werden die höheren Kommunikationsprotokolle *LU6.1* und *OSI TP* verwendet. Über *open*UTM-LU62 ist verteilte Verarbeitung auch mit LU6.2 Partnern möglich. Man unterscheidet verteilte Verarbeitung mit *globalen Transak-*

tionen (anwendungsübergreifende Transaktionssicherung) und verteilte Verarbeitung ohne globale Transaktionen (nur lokale Transaktionssicherung). Die verteilte Verarbeitung wird auch Server-Server-Kommunikation genannt.

Vorgang (KDCS)
service

Ein Vorgang dient zur Bearbeitung eines *Auftrags* in einer *UTM-Anwendung*. Er setzt sich aus einer oder mehreren *Transaktionen* zusammen. Die erste Transaktion wird über den *Vorgangs-TAC* aufgerufen. Es gibt *Dialog-Vorgänge* und *Asynchron-Vorgänge*. *open*UTM stellt den Teilprogrammen eines Vorgangs gemeinsame Datenbereiche zur Verfügung. Anstelle des Begriffs Vorgang wird häufig auch der allgemeinere Begriff *Service* gebraucht.

Vorgangskellerung (KDCS)
service stacking

Ein Terminal-Benutzer kann einen laufenden *Dialog-Vorgang* unterbrechen und einen neuen Dialog-Vorgang einschieben. Nach Beendigung des eingeschobenen *Vorgangs* wird der unterbrochene Vorgang fortgesetzt.

Vorgangskettung (KDCS)
service chaining

Bei Vorgangskettung wird nach Beendigung eines *Dialog-Vorgangs* ohne Angabe einer *Dialog-Nachricht* ein Folgevorgang gestartet.

Vorgangs-TAC (KDCS)
service TAC

Transaktionscode, mit dem ein *Vorgang* gestartet wird.

Vorgangswiederanlauf (KDCS)
service restart

Nach einem Vorgangsabbruch, z.B. infolge Abmeldens des Terminal-Benutzers oder Beendigung der *UTM-Anwendung,* führt *open*UTM einen Vorgangswiederanlauf durch. Ein *Asynchron-Vorgang* wird neu gestartet oder beim zuletzt erreichten *Sicherungspunkt* fortgesetzt, ein *Dialog-Vorgang* wird beim zuletzt erreichten *Sicherungspunkt* fortgesetzt. Für den Terminal-Benutzer wird der Vorgangswiederanlauf eines Dialog-Vorgangs als *Bildschirm-Wiederanlauf* sichtbar.

Warmstart
warm start

Starten einer *UTM-Anwendung* nach einer *abnormalen Beendigung*. Beim Warmstart wird die *KDCFILE* wieder in einen konsistenten Zustand gebracht.

Wiederanlauf
restart

siehe *Anwendungswiederanlauf*,
siehe *Bildschirm-Wiederanlauf*,
siehe *Vorgangswiederanlauf*.

Workprozeß (UNIX/Windows NT)
work process

Prozeß, in dem die *Services* der *UTM-Anwendung* ablaufen.

XATMI

XATMI (X/Open Application Transaction Manager Interface) ist eine von X/Open standardisierte Programmschnittstelle für die Programm-Programm-Kommunikation in offenen Netzen.
Das in *open*UTM implementierte XATMI genügt der XATMI CAE Specification von X/Open. Die Schnittstelle steht in COBOL und C zur Verfügung. XATMI in *open*UTM kann über die Protokolle OSI-TP, *LU6.1* und UPIC kommunizieren.

XHCS (BS2000/OSD)

XHCS (Extended Host Code Support) ist ein BS2000/OSD-Softwareprodukt für die Unterstützung internationaler Zeichensätze.

Zeitgesteuerter Auftrag
time-driven job

Asynchron-Auftrag, der von *open*UTM bis zu einem definierten Zeitpunkt in einer *Message Queue* zwischengespeichert und dann an den Empfänger weitergeleitet wird. Empfänger kann sein: ein *Asynchron-Vorgang* derselben Anwendung, eine Partneranwendung, ein Terminal oder ein Drucker.
Zeitgesteuerte Aufträge können nur von KDCS-*Teilprogrammen* erteilt werden.

Zugangskontrolle
system access control

Prüfung durch *open*UTM, ob eine bestimmte *Benutzerkennung* berechtigt ist, mit der *UTM-Anwendung* zu arbeiten. Die Berechtigungsprüfung entfällt, wenn die UTM-Anwendung ohne Benutzerkennungen generiert wurde.

Zugriffskontrolle
data access control

Überwachung der Zugriffe auf die Daten und Objekte einer Anwendung. Die Zugriffsrechte werden als Bestandteil der Konfiguration festgelegt.

Zugriffspunkt
access point

siehe *Dienstzugriffspunkt*.

Abkürzungen

ACSE	Association Control Service Element
AEQ	Application Entity Qualifier
AET	Application Entity Title
APT	Application Process Title
ASCII	American Standard Code for Information Interchange
ASE	Application Service Element
ASECO	Advanced Security Control
BCAM	Basic Communication Access Method
BER	Basic Encoding Rules
BLS	Binder-Lader-Starter (BS2000/OSD)
BSI	Bundesamt für Sicherheit in der Informationstechnik
CCP	Communication Control Program
CCR	Commitment, Concurrency and Recovery
CCS	Codierter Zeichensatz (Coded Character Set)
CCSN	Name des codierten Zeichensatzes (Coded Character Set Name)
CID	Control Identification
CIW	CPI-C Implementor’s Workshop
CMX	Communication Manager in UNIX
COM	Component Object Model
CPI-C	Common Programming Interface for Communication
CRM	Communication Resource Manager
CRTE	Common Runtime Environment (BS2000/OSD)
DB	Datenbank
DC	Data Communication
DCAM	Data Communication Access Method

DCOM	Distributed Component Object Mode
DES	Data Encryption Standard
DMS	Data Management System
DNS	Domain Name Service
DSR	Datenstationsrechner
DSS	Datensichtstation
DTP	Dialogterminalprozeß (UNIX, Windows NT)
DVR	Datenübertragungsvorrechner
DVS	Datenverwaltungssystem
EBCDIC	Binärcode für die stellenweise Verschlüsselung von Dezimalziffern (Extended Binary-Coded Decimal Interchange Code)
FGG	File Generation Group
FHS	Format Handling System
FORMANT	Format-Manager für Terminals
FORMANTGEN	Formatgenerator
FORSCHAB	Formatschablonenprogramm
FT	File Transfer
GSSB	Globaler Sekundärer Speicherbereich
HLL	High-Level Language
IFG	Interaktiver Format-Generator
ILCS	Inter Language Communication Services (BS2000/OSD)
IPC	Inter-Process-Communication
IRV	Internationale Referenzversion
ISO	International Organization for Standardization
IT	Informationstechnik
ITS	Informationstechnische Systeme
KAA	KDCS Application Area
KB	Kommunikationsbereich
KBPRG	KB-Programmbereich
KDBS	Kompatible Datenbankschnittstelle
KDCS	Kompatible Datenkommunikationsschnittstelle
KOGS	Konfigurationorientierte Generatorsprache

KTA	KDCS Task Area
LCF	Local Configuration File
LLM	Link and Load Module (BS2000/OSD)
LSSB	Lokaler Sekundärer Speicherbereich
MIGRAT	Migrationsprogramm
MQ	Message Queuing
MSCF	Multiple System Control Facility (BS2000/OSD)
MSF	Mehrfachsteuerung für Fernanschluß
MSN	Mehrfachsteuerung für Nahbereich
NEA	Netzwerkarchitektur bei TRANSDATA-Systemen
NB	Nachrichtenbereich
NLS	Unterstützung der Landessprache (Native Language Support)
OCX	OLE Control Extension
OML	Object Modul Library
OSI	Open System Interconnection
OSI-TP	Open System Interconnection Transaction Processing
OSS	OSI Session Service
PCMX	Portable Communication Manager
PDN	Programmsystem für Datenfernverarbeitung und Netzsteuerung
PIN	Persönliche Identifikationsnummer
PLU	Primary Logical Unit
RAV	Rechenzentrums-Abrechnungs-Verfahren
RDF	Ressource Definition File
RM	Resource Manager
RSA	Encryption-Algorithmus nach Rivest, Shamir, Adleman
RSO	Remote SPOOL Output (BS2000/OSD)
SAT	Security Audit Trail (BS2000/OSD)
SECOS	Security Control System
SLU	Secondary Logical Unit
SM2	Software Monitor 2 (BS2000/OSD)
SNA	Systems Network Architecture
SPAB	Standard Primärer Arbeitsbereich

SSB	Sekundärer Speicherbereich
TAC	Transaktionscode
TCP/IP	Transmission Control Protocol / Internet Protocol
TIAM	Terminal Interactive Access Method
TLS	Terminalspezifischer Langzeitspeicher
TM	Transaction Manager
TNS	Transport Name Service
TNSX	Transport Name Service in UNIX / Windows NT
TP	Transaction Processing (Transaktions-Betrieb)
TPR	privilegierter Funktionszustand des BS2000 (Task priviledged)
TSN	Task Sequence Number
TU	nicht privilegierter Funktionszustand des BS2000 (Task user)
TX	Transaction Demarcation (X/Open)
UDS	Universelles Datenbanksystem
UDT	Unstructured Data Transfer
ULS	Userspezifischer Langzeitspeicher
UPIC	Universal Programming Interface for Communication
UTM	Universelle Transaktionsmonitor
UTM-D	UTM-Variante für verteilte Verarbeitung im BS2000
UTM-F	Schnelle ("Fast") UTM-Variante
UTM-S	UTM-Sicherheitsvariante
VAR	Verarbeitungsrechner
VGID	Vorgangsidentifikation
VTSU	Virtual Terminal Support
VTV	Verteilte Transaktionsverarbeitung
VV	Verteilte Verarbeitung
XATMI	X/Open Application Transaction Manager Interface
XHCS	eXtended Host Code Support
ZSI	Zentralstelle für Sicherheit in der Informationstechnik

Literatur

Zu *open*UTM gibt es auch eine CD-ROM, die alle *open*UTM-Handbücher als PDF-Dateien enthält.
Die PDF-Dateien der Handbücher zu *open*UTM sind auch über das Internet als Download kostenlos erhältlich. Unter *http://www.siemens.de/servers/man* finden Sie die Übersichtsseite der im Internet verfügbaren Online-Dokumentation.

***open*UTM**
Konzepte und Funktionen
Benutzerhandbuch

Zielgruppe
Alle, die sich einen Überblick über die Funktionsbreite und Leistungsfähigkeit von *open*UTM verschaffen wollen

***open*UTM** (BS2000/OSD, UNIX, Windows NT)
Anwendungen programmieren mit KDCS für COBOL, C und C++
Basishandbuch

Zielgruppe
Programmierer, die für die Programmierung von UTM-Anwendungen die Programmschnittstelle KDCS nutzen wollen

***open*UTM** (BS2000/OSD, UNIX, Windows NT)
Anwendungen erstellen mit X/Open-Schnittstellen
Basishandbuch

Zielgruppe
Programmierer, die für die Programmierung von UTM-Anwendungen die X/Open-Schnittstellen nutzen wollen

***open*UTM** (UNIX, Windows NT)
Anwendungen generieren und betreiben
Benutzerhandbuch

Zielgruppe
Anwendungsplaner, Organisatoren, Anwender und Betreuer von UTM-Anwendungen

***open*UTM** (BS2000/OSD)
Anwendungen generieren und betreiben
Benutzerhandbuch

Zielgruppe
Anwendungsplaner, Organisatoren, Anwender und Betreuer von UTM-Anwendungen

***open*UTM** (BS2000/OSD, UNIX, Windows NT)
Anwendungen administrieren
Benutzerhandbuch

Zielgruppe
Administratoren und Programmierer von Administrationsprogrammen

***open*UTM** (UNIX, Windows NT**)**
Meldungen, Test und Diagnose
Benutzerhandbuch

Zielgruppe
Anwender, Administratoren und Programmierer von UTM-Anwendungen

***open*UTM** (BS2000/OSD)
Meldungen, Test und Diagnose
Benutzerhandbuch

Zielgruppe
Anwender, Administratoren und Programmierer von UTM-Anwendungen

***open*UTM-Client** (UNIX)
für Trägersystem OpenCPIC
Client-Server-Kommunikation mit *open*UTM
Benutzerhandbuch

Zielgruppe
Organisatoren, Einsatzplaner, Programmierer von CPI-C- und XATMI-Programmen

***open*UTM-Client V5.0** (UNIX, Windows)
für Trägersystem UPIC
Client-Server-Kommunikation mit *open*UTM
Benutzerhandbuch

Zielgruppe
Organisatoren, Einsatzplaner, Programmierer von CPI-C- und XATMI-Programmen

***open*UTM-Client V4.0** (BS2000/OSD, UNIX, Windows)
für Trägersystem UPIC
Client-Server-Kommunikation mit *open*UTM
Benutzerhandbuch

Zielgruppe
Organisatoren, Einsatzplaner, Programmierer von CPI-C- und XATMI-Programmen

Für BS2000/OSD ist dieses Handbuch zu *open*UTM-Client V4.0 gültig.

***open*UTM-JetClient** (Reliant UNIX)
***open*UTM Java Enterprise Technology Client**
Benutzerhandbuch

Zielgruppe
Programmierer und Betreiber von *open*UTM-Client-Anwendungen auf Windows 3.1x/95/NT, UNIX und BS2000/OSD.

***open*UTM**, ***open*UTM-LU62** (BS2000/OSD, UNIX, Windows NT)
Verteilte Transaktionsverarbeitung
zwischen *open*UTM und CICS-, IMS- und LU6.2-Anwendungen

Zielgruppe
Organisatoren, Einsatzplaner, Programmierer und Administratoren

***open*UTM**
***open*UTM Connectivity**
Ready Reference

Zielgruppe
Alle, die sich einen Überblick über die Kopplungsmöglichkeiten mit *open*UTM oder *open*UTM-Client verschaffen wollen

open*UTM Web*Transactions
Konzepte und Funktionen
nur online verfügbar: Internet und *open*UTM-Handbuch-CD

Zielgruppe
Alle, die UTM-, MVS- oder OSD-Anwendungen ans Web anbinden wollen

open*UTM Web*Transactions
Templatesprache
nur online verfügbar: Internet und *open*UTM-Handbuch-CD

Zielgruppe
Alle, die UTM-, MVS- oder OSD-Anwendungen ans Web anbinden wollen

open*UTM Web*Transactions
Web-Zugriff auf *open*UTM-Anwendungen
nur online verfügbar: Internet und *open*UTM-Handbuch-CD

Zielgruppe
Alle, die UTM-Anwendungen ans Web anbinden wollen

open*UTM Web*Transactions
Web-Zugriff auf MVS-Anwendungen
nur online verfügbar: Internet und *open*UTM-Handbuch-CD

Zielgruppe
Alle, die UTM-Anwendungen ans Web anbinden wollen

open*UTM Web*Transactions
Web-Zugriff auf OSD-Anwendungen
nur online verfügbar: Internet und *open*UTM-Handbuch-CD

Zielgruppe
Alle, die UTM-Anwendungen ans Web anbinden wollen

Com*Transactions* V2.0 (Windows 95, Windows NT)
Host-Anwendungen in Windows integrieren
Benutzerhandbuch

Zielgruppe
Administratoren, Anwendungsentwickler und Endbenutzer, die die Daten von Host-Anwendungen in Windows-Ungewbung bearbeiten und nutzen wollen.

***open*UTM** (BS2000/OSD)
Anwendungen programmieren mit KDCS für Assembler
Ergänzung zum Basishandbuch
nur online verfügbar: Internet und *open*UTM-Handbuch-CD

Zielgruppe
Programmierer von UTM-Assembler-Anwendungen

***open*UTM** (BS2000/OSD)
Anwendungen programmieren mit KDCS für Fortran
Ergänzung zum Basishandbuch
nur online verfügbar: Internet und *open*UTM-Handbuch-CD

Zielgruppe
Programmierer von UTM-Fortran-Anwendungen

***open*UTM** (BS2000/OSD)
Anwendungen programmieren mit KDCS für Pascal-XT
Ergänzung zum Basishandbuch
nur online verfügbar: Internet und *open*UTM-Handbuch-CD

Zielgruppe
Programmierer von UTM-Pascal-XT-Anwendungen

***open*UTM** (BS2000/OSD)
Anwendungen programmieren mit KDCS für PL/I
Ergänzung zum Basishandbuch
nur online verfügbar: Internet und *open*UTM-Handbuch-CD

Zielgruppe
Programmierer von UTM-PL/I-Anwendungen

UTM (SINIX)
Beispielanwendung

Zielgruppe
Anwender und Programmierer von UTM-Anwendungen

UTM (SINIX)
Formatierungssystem
Benutzerhandbuch

Zielgruppe
UTM(SINIX)-Anwender, die mit Formaten arbeiten wollen sowie C-Programmierer und COBOL-Programmierer

AID (BS2000/OSD)
Advanced Interactive Debugger
Basishandbuch
Benutzerhandbuch

Zielgruppe
Programmierer im BS2000

AID-FE (SINIX)
Grafische Testoberfläche auf Basis von AID V2.1B (BS2000)
Benutzerhandbuch

Zielgruppe
C-, C++- und COBOL85-Programmierer in einer BS2000-Umgebung mit SINIX-Vernetzung und grafikfähigem Terminal

ASECO (BS2000/OSD)
Advanced Security Control
Benutzerhandbuch

Zielgruppe
BS2000-Systemverwaltungen, die den Zugangsschutz mit Chipkarten realisieren

BCAM (BS2000/OSD)
BCAM Band 1
Benutzerhandbuch

Zielgruppe
Netzplaner, -generierer und -verwalter, die in BS2000-Systemen BCAM betreiben

BCAM (BS2000/OSD)
BCAM Band 2
Referenzhandbuch

Zielgruppe
Netzoperateure, -generierer und -verwalter, die in BS2000-Systemen BCAM betreiben

BINDER (BS2000/OSD)
Benutzerhandbuch

Zielgruppe
Software-Entwickler

BINDER (BS2000/OSD)
Tabellenheft

Zielgruppe
Software-Entwickler, die im Umgang mit dem BINDER geübt sind

BS2000/OSD
Makroaufrufe an den Ablaufteil
Benutzerhandbuch

Zielgruppe
BS2000/OSD-Assembler-Programmierer

BS2000/OSD-BC
Bindelader-Starter
Benutzerhandbuch

Zielgruppe
Software-Entwickler und geübte BS2000/OSD-Benutzer

CMX (SINIX)
Communication Manager in SINIX
Betrieb und Administration
Benutzerhandbuch

Zielgruppe
Systemverwalter

CMX (UNIX)
Benutzerhandbuch

Zielgruppe
Systemverwalter

DCAM (BS2000/OSD, TRANSDATA)
COBOL-Aufrufe
Benutzerhandbuch

Zielgruppe
Programmierer von DCAM-COBOL-Programmen

DCAM (BS2000/OSD, TRANSDATA)
Makroaufrufe
Benutzerhandbuch

Zielgruppe
Programmierer von DCAM-Assembler-Programmen

DCAM (BS2000/OSD, TRANSDATA)
Programmschnittstellen
Beschreibung

Zielgruppe
Organisatoren, Einsatzplaner, Programmierer, Systemverwalter und Netzadministratoren

FHS (BS2000/OSD, TRANSDATA)
Formatierungssystem für UTM, TIAM, DCAM
Benutzerhandbuch

Zielgruppe
Programmierer

FHS-DOORS (Windows, BS2000/OSD)
Grafische Oberfläche für BS2000/OSD-Anwendungen
Benutzerhandbuch

Zielgruppe
End-Anwender von FHS-DOORS und Administratoren, die Masken konvertieren und diese bereitstellen

FORMANT (SINIX)
Beschreibung

Zielgruppe
C-Programmierer, COBOL-Programmierer und Anwendungsplaner

GINA - General Interface for Network Applications
Einführung

Zielgruppe
Entscheider, Anwender

GINA - General Interface for Network Applications
Entwicklerhandbuch

Zielgruppe
Entwickler von GINA-Applikationen

GINA - General Interface for Network Applications
Referenzhandbuch

Zielgruppe
Programmierer von GINA-Applikationen

GINA - General Interface for Network Applications
Systemverwaltung

Zielgruppe
Systemverwalter

HIPLEX AF (BS2000/OSD)
Automatisches Umschalten von Anwendungen zwischen BS2000-Systemen
Produkthandbuch

Zielgruppe
Systemverwalter und Organisatoren in Rechenzentren

Hohe Verfügbarkeit von SNI UNIX-Systemen
(RM-Modelle)

Zielgruppe
EDV-Verantwortliche, Systemorganisatoren, Systemverwalter und Benutzer

IFG für FHS (TRANSDATA)
Benutzerhandbuch

Zielgruppe
Datenstationsbenutzer, Anwendungsdesigner und Programmierer

OBSERVE (SINIX)
Benutzerhandbuch

Zielgruppe
OBSERVE-Systemverwalter und OBSERVE-Programmierer

OMNIS/OMNIS-MENU (TRANSDATA, BS2000)
Funktionen und Kommandos
Benutzerhandbuch

Zielgruppe
OMNIS-Administrator und OMNIS-Anwender

OMNIS/OMNIS-MENU (TRANSDATA, BS2000)
Administration und Programmierung
Benutzerhandbuch

Zielgruppe
OMNIS-Administrator und Programmierer

OMNIS-MENU (TRANSDATA, BS2000/OSD)
Benutzerhandbuch

Zielgruppe
OMNIS-MENU-Benutzer und OMNIS-MENU-Administrator

OSS (BS2000/OSD)
OSI Session Service
User Guide

Zielgruppe
OSI TP-Anwender

OSS (SINIX)
OSI Session Service
User Guide

Zielgruppe
OSI TP-Anwender

SECOS (BS2000/OSD)
Security Control System
Benutzerhandbuch

Zielgruppe
BS2000-Systemverwalter und BS2000-Anwender, die den erweiterten Zugriffsschutz für Dateien nutzen

SECOS (BS2000/OSD)
Security Control System
Tabellenheft

Zielgruppe
Erfahrene SECOS-Anwender

SM2 (BS2000/OSD)
Software Monitor
Benutzerhandbuch

Zielgruppe
Anwender und Systembetreuer

TIAM (BS2000/OSD, TRANSDATA)
Benutzerhandbuch

Zielgruppe
nicht privilegierte BS2000-Anwender und Programmierer

VTSU (BS2000/OSD, TRANSDATA)
Virtual Terminal Support
Benutzerhandbuch

Zielgruppe
Anwender der Zugriffsmethoden DCAM, TIAM und UTM sowie System- und Netzverwalter

XHCS (BS2000/OSD)
8-bit-Code-Verarbeitung im BS2000/OSD
Benutzerhandbuch

Zielgruppe
Anwender der Zugriffsmethoden DCAM, TIAM und UTM sowie Systemverwalter, Anwender, die von EHCS auf XHCS umstellen

Sonstige Literatur

BS2000/OSD Technische Beschreibung
Hrsg.: Peter Jilek
Transaktionssystem
v. Rudolf Koch
ISBN 3 89578 017 0

CPI-C (X/Open)
Distributed Transaction Processing
X/Open CAE Specification, Version 2
ISBN 1 85912 135 7

Reference Model Version 2 (X/Open)
Distributed Transaction Processing
X/Open Guide
ISBN 1 85912 019 9

TX (Transaction Demarcation) (X/Open)
Distributed Transaction Processing
X/Open CAE Specification
ISBN 1 85912 094 6

XATMI (X/Open)
Distributed Transaction Processing
X/Open CAE Specification
ISBN 1 85912 130 6

Stichwörter

Inhalt

*open*UTM V5.0

Konzepte und Funktionen

Zielgruppe
Alle, die sich über die Funktionsbreite und Leistungsfähigkeit von *open*UTM informieren wollen.

Inhalt

Das Handbuch enthält eine allgemeine Beschreibung aller Funktionen und Leistungen von *open*UTM sowie einführende Informationen, die als Einstieg in die Arbeit mit *open*UTM dienen sollen.

Ausgabe: Oktober 1998

Datei: utm_kon.pdf