Deutsch

FUJITSU Software

openUTM V6.3

Konzepte und Funktionen

Benutzerhandbuch

Ausgabe Januar 2015

Kritik… Anregungen… Korrekturen…

Die Redaktion ist interessiert an Ihren Kommentaren zu diesem Handbuch. Ihre Rückmeldungen helfen uns, die Dokumentation zu optimieren und auf Ihre Wünsche und Bedürfnisse abzustimmen.

Sie können uns Ihre Kommentare per E-Mail an manuals@ts.fujitsu.com senden.

Zertifizierte Dokumentation nach DIN EN ISO 9001:2008

Um eine gleichbleibend hohe Qualität und Anwenderfreundlichkeit zu gewährleisten, wurde diese Dokumentation nach den Vorgaben eines Qualitätsmanagementsystems erstellt, welches die Forderungen der DIN EN ISO 9001:2008 erfüllt.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright und Handelsmarken

Inhalt

1 Einleitung 11

1.1 Konzept und Zielgruppen dieses Handbuchs 13

1.2 Wegweiser durch die Dokumentation zu openUTM 14
1.2.1 openUTM-Dokumentation 14
1.2.2 Dokumentation zum openSEAS-Produktumfeld 19
1.2.3 Readme-Dateien 20

1.3 Neuerungen in openUTM V6.3 21
1.3.1 Neue Server-Funktionen 21
1.3.2 Last-Simulation mit "Workload Capture & Replay" 24
1.3.3 Neue Client-Funktion 25
1.3.4 Neue und geänderte Funktionen für openUTM WinAdmin 25
1.3.5 Neue Funktionen für openUTM WebAdmin 25

2 openUTM - Leistungsüberblick 27

2.1 openUTM – die „high-end Transaction Processing Platform“ 27

2.2 Transaktionskonzept und Restart-Funktionen 29

2.3 Zusammenarbeit mit Datenbanken und Resource Managern 30

2.4 UTM-Cluster-Anwendung 34
2.4.1 UTM-Cluster-Dateien 37
2.4.2 Systemvoraussetzungen für den Einsatz von UTM-Cluster-Anwendungen 39
2.4.3 Einsatz von SESAM/SQL- und UDS/SQL-Datenbanken im Cluster 40

2.5 Message Queuing 42

2.6 openUTM - offen für unterschiedliche Plattformen und Protokolle 45

2.7 X/Open-Konformität von openUTM 50

2.8 Performance, Durchsatz und Antwortzeiten 53

2.9 Workload Capture & Replay 54

2.10 Hochverfügbarkeit . . . 55
2.11 Security-Funktionen . . . 56
2.12 Dynamisierung . . . 58
2.13 Internationalisierung / Anpassung von Meldungen . . . 59
2.14 openUTM in der Unicode-Umgebung . . . 59
2.15 Accounting . . . 60
2.16 Leistungskontrolle mit dem Software Monitor openSM2 . . . 61
2.17 Diagnosemöglichkeiten in openUTM . . . 62
2.18 Einfache, komfortable Anwendungsentwicklung . . . 63
2.19 Grafische Administration mit WinAdmin . . . 64
2.20 Grafische Administration mit WebAdmin . . . 66
2.21 SNMP-Subagent für openUTM . . . 68

3 Integrationsszenarien mit openUTM . . . 69

3.1 Unterschiedliche Anwendungen integrieren . . . 69
3.2 openUTM in Java Enterprise Umgebung integrieren . . . 70
3.2.1 openUTM als Server für Java EE Application Server . . . 70
3.2.2 openUTM als Client von Java EE Application Servern . . . 72
3.2.3 UTM-Cluster-Anwendung als Client oder Server . . . 73
3.3 openUTM über Web Services ansprechen . . . 74
3.4 Bestehende Anwendungen in das Web stellen . . . 75

4 Verteilte Verarbeitung mit openUTM . . . 77

4.1 Client/Server-Architekturvarianten . . . 77
4.2 Zu den Begriffen „Client“ und „Server“ . . . 81
4.2.1 Kommunikation mit openUTM-Client-Anwendungen . . . 82
4.2.1.1 Clients mit Trägersystem UPIC . . . 82
4.2.1.2 Clients mit Trägersystem OpenCPIC . . . 83
4.2.2 Java-Clients . . . 84
4.3 Server-Server-Kommunikation . . . 87
4.3.1 Anwendungs-übergreifende Dialoge . . . 87

4.3.2 Transaktionssicherung bei Server-Server-Kommunikation . . . 90
4.3.3 Beispiel: Anwendungs-übergreifender Dialog mit verteilter Transaktion . . . 91
4.3.4 Adressierung ferner Services . . . 93
4.3.5 Kommunikation mit CICS-, IMS- und TXSeries-Anwendungen . . . 96

4.4 Kommunikation mit Transportsystem-Anwendungen . . . 98

4.5 Übersicht: Partner, Protokolle,Transaktionssicherung . . . 101

5 Message Queuing . . . 103

5.1 UTM-gesteuerte Queues . . . 104
5.1.1 Ausgabeaufträge (Output Queuing) . . . 104
5.1.2 Hintergrundaufträge . . . 104
5.1.2.1 Bearbeitung von Hintergrundaufträgen . . . 105
5.1.2.2 Hintergrundaufträge an ferne Services (Remote Queuing) . . . 107
5.1.3 Priority Scheduling bei Hintergrundaufträgen . . . 108

5.2 Service-gesteuerte Queues . . . 109
5.2.1 USER-Queues . . . 110
5.2.2 TAC-Queues . . . 111
5.2.3 Temporäre Queues . . . 112

5.3 Steuerungsmöglichkeiten für Message Queues . . . 115

5.4 Message Queue-Aufrufe der KDCS-Schnittstelle . . . 119

6 Struktur einer UTM-Anwendung . . . 121

6.1 UTM-Anwendungsprogramm . . . 122

6.2 Prozesskonzept . . . 124

6.3 Die KDCFILE - das „Gedächtnis der Anwendung“ . . . 126
6.3.1 KDCFILE einer stand-alone-Anwendung . . . 126
6.3.2 KDCFILEs in einer UTM-Cluster-Anwendung . . . 128

6.4 UTM-Cache-Speicher . . . 129

7 Programmschnittstellen . . . 131

7.1 Überblick über die Programmschnittstellen . . . 132

7.2 Die universale Programmschnittstelle KDCS . . . 134
7.2.1 KDCS-Aufrufe . . . 134

7.2.2 UTM-Speicherbereiche 138
7.2.3 Event-Funktionen 142
7.3 Die X/Open-Schnittstelle CPI-C 145
7.4 Die X/Open-Schnittstelle XATMI 147
7.5 Die X/Open-Schnittstelle TX 149
7.6 Die XML-Schnittstelle von openUTM 150

8 Generieren von UTM-Anwendungen 155

8.1 Definieren der Konfiguration 156
Übersicht: KDCDEF-Steueranweisungen 157
8.2 Erzeugen des Anwendungsprogramms 159
8.3 Konfiguration aktualisieren mit dem Tool KDCUPD 161

9 Administrieren von UTM-Anwendungen 165

9.1 Kommandoschnittstelle zur Administration 167
9.2 Programmschnittstelle zur Administration 170
9.3 Grafisches Administrationsprogramm WinAdmin 173
9.4 Grafisches Administrationsprogramm WebAdmin 176
9.5 Berechtigungskonzept 178
9.6 Dynamische Erweiterung der Generierung 179
9.7 Automatische Administration 181
9.8 Administration von Message Queues und Druckern 182

10 Security-Funktionen 183

10.1 Zugangskontrolle (Identifizierung und Authentisierung) 184
10.2 Zugriffskontrolle (Autorisierung) 190
10.2.1 Lock-/Keycode-Konzept 190
10.2.2 Access-List-Konzept 192
10.3 Zugangs- und Zugriffskontrolle bei verteilter Verarbeitung 195

10.4 Verschlüsselung 200
10.5 Security-Funktionen externer Resource Manager 202

11 Hochverfügbarkeit mit stand-alone UTM-Anwendungen 205

11.1 Hochverfügbarkeit in BS2000-Systemen 205
11.2 Hochverfügbarkeit in Unix-Systemen 207
11.3 Hochverfügbarkeit in Windows-Systemen 208

12 Hochverfügbarkeit und Lastverteilung mit UTM-Cluster-Anwendungen 209

12.1 Hochverfügbarkeit mit UTM-Cluster-Anwendungen 209
12.2 Lastverteilung 215
12.2.1 Lastverteilung bei verteilter Verarbeitung 215
12.2.2 Lastverteilung bei UPIC-Clients 216
12.2.3 Lastverteilung mit Oracle® RAC 216

13 Fehlertoleranz und Wiederanlauf 217

13.1 Eingrenzung von Teilprogramm- und Formatierungsfehlern 217
13.2 Automatische Prüfungen 219
13.3 Schutz bei Störungen oder Ausfall lokaler Betriebsmittel 220
13.4 Abnormale Beendigung einer UTM-Anwendung 222
13.5 Wiederanlauf-Funktionen von openUTM 223
13.5.1 Die Varianten UTM-S und UTM-F 223
13.5.2 Wiederanlauf mit UTM-S 224
13.5.3 Wiederanlauf mit UTM-F 226
13.6 Fehlerbehandlung bei verteilter Verarbeitung 227
13.6.1 Rollback und Wiederanlauf bei globaler Transaktionssicherung 227
13.6.2 Rollback und Wiederanlauf bei unabhängigen Transaktionen 229

14 openUTM in BS2000-Systemen 231

14.1 Systemeinbettung 231

14.2 UTM-Prozesse 235
14.3 Adressraumkonzept 236
14.4 Formatierung 238
14.5 Codeumsetzung 240
14.6 BS2000-spezifische Funktionen 241

15 openUTM in Unix-Systemen 247

15.1 Systemeinbettung 247
15.2 UTM-Prozesse 249
15.3 Adressraumkonzept 253
15.4 Konfigurieren der Netzanbindung 255
15.5 Codeumsetzung 255
15.6 Ablauf auf 64-Bit-Plattformen 255

16 openUTM in Windows-Systemen 257

16.1 Systemeinbettung 257
16.2 UTM-Prozesse 258
16.3 Adressraumkonzept 263
16.4 Konfigurieren der Netzanbindung 265
16.5 Codeumsetzung 265

17 Anhang: Unterstützte Standards und Normen 267

Fachwörter 269

Abkürzungen 309

Literatur 315

Stichwörter 325

Inhalt

1 Einleitung

Moderne unternehmensweite IT-Umgebungen unterliegen zahlreichen Herausforderungen von zunehmender Brisanz. Dies wird verursacht durch

- heterogene Systemlandschaften
- unterschiedliche HW-Plattformen
- unterschiedliche Netze und Netzzugriffe (TCP/IP, SNA, ...)
- Verflechtung der Anwendungen mit den Unternehmen

Dadurch entwickeln sich Problemfelder, sei es bei Fusionen, durch Kooperationen oder auch nur durch Rationalisierungsmaßnahmen. Die Unternehmen fordern flexible und skalierbare Anwendungen, gleichzeitig soll die Transaktionssicherheit für Prozesse und Daten gewährleistet bleiben, obwohl die Geschäftsprozesse immer komplexer werden. Die wachsende Globalisierung geht selbstverständlich davon aus, dass Anwendungen im 7x24-Stunden-Betrieb laufen und hochverfügbar sind, um beispielsweise Internetzugriffe auf bestehende Anwendungen über Zeitzonen hinweg zu ermöglichen.

Die High-End-Plattform für Transaktionsverarbeitung openUTM bietet eine Ablaufumgebung, die all diesen Anforderungen moderner unternehmenskritischer Anwendungen gewachsen ist, denn openUTM verbindet alle Standards und Vorteile von transaktionsorientierten Middleware-Plattformen und Message Queuing Systemen:

- Konsistenz der Daten und der Verarbeitung
- Hohe Verfügbarkeit der Anwendungen (nicht nur der Hardware)
- Hohen Durchsatz auch bei großen Benutzerzahlen, d.h. höchste Skalierbarkeit
- Flexibilität bezüglich Änderungen und Anpassungen des IT-Systems

Eine UTM-Anwendung kann auf einem einzelnen Rechner als stand-alone UTM-Anwendung oder auf mehreren Rechnern gleichzeitig als UTM-Cluster-Anwendung betrieben werden.

openUTM ist Teil des umfassenden Angebots von **openSEAS**. Gemeinsam mit der Oracle Fusion Middleware bietet openSEAS die komplette Funktionalität für Anwendungsinnovation und moderne Anwendungsentwicklung. Im Rahmen des Produktangebots **openSEAS** nutzen innovative Produkte die ausgereifte Technologie von openUTM:

- BeanConnect ist ein Adapter gemäß der Java EE Connector Architecture (JCA) von Oracle/Sun und bietet den standardisierten Anschluss von UTM-Anwendungen an Java EE Application Server. Dadurch können bewährte Legacy-Anwendungen in neue Geschäftsprozesse integriert werden.
- Mit WebTransactions steht in openSEAS ein Produkt zur Verfügung, welches es ermöglicht, bewährte Host-Anwendungen flexibel in neuen Geschäftsprozessen und modernen Einsatzszenarien zu nutzen. Bestehende UTM-Anwendungen können unverändert ins Web übernommen werden.

Die Produkte BeanConnect und WebTransactions werden im Leistungsüberblick kurz dargestellt. Für diese Produkte gibt es eigene Handbücher.

1.1 Konzept und Zielgruppen dieses Handbuchs

Das vorliegende Handbuch „Konzepte und Funktionen“ dient dazu, die Produktfamilie openUTM kennen zu lernen und in die konkrete Arbeit mit openUTM einzusteigen. Es richtet sich insbesondere an diejenigen, die mit openUTM noch nicht vertraut sind. Aber auch wenn Sie openUTM 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 openUTM gegeben werden. Mit diesem Überblick ausgestattet, werden Sie sich in den übrigen Handbüchern der openUTM-Reihe leicht zurechtfinden.

In Kapitel 2 werden Ihnen die Eigenschaften von openUTM kurz vorgestellt. Auf einige der dort angesprochenen Themen wird dann in den Kapiteln 3 bis 13 noch einmal etwas ausführlicher eingegangen.

openUTM ist für alle gängigen Unix- und Windows-Plattformen sowie für BS2000-Plattformen ausführlichen verfügbar. Die Funktionalität sowie die Anwenderschnittstellen gleichen sich dabei weitgehend. Deshalb gelten die Informationen in den ersten Kapiteln dieses Handbuchs für alle Plattformen.

Die drei letzten Kapitel enthalten einige Plattform-spezifische Details, und zwar in Kapitel 14 für BS2000-Plattformen, in Kapitel 15 für alle Unix-Plattformen und in Kapitel 16 für Windows-Plattformen. 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.

Wenn im Folgenden allgemein von Unix-System bzw. Unix-Plattform die Rede ist, dann ist darunter sowohl ein Unix-basiertes Betriebssystem wie z.B. Solaris oder HP-UX als auch eine Linux-Distribution wie z.B. SUSE oder Red Hat zu verstehen.

Wenn im Folgenden von Windows-System bzw. Windows-Plattform die Rede ist, dann sind damit alle Windows-Varianten gemeint, auf denen openUTM zum Ablauf kommt.

1.2 Wegweiser durch die Dokumentation zu openUTM

In diesem Abschnitt erhalten Sie einen Überblick über die Handbücher zu openUTM und zum Produktumfeld von openUTM.

1.2.1 openUTM-Dokumentation

Die openUTM-Dokumentation besteht aus Handbüchern, den Online-Hilfen für den grafischen Administrationsarbeitsplatz openUTM WinAdmin und das grafische Administrationstool WebAdmin sowie einer Freigabemitteilung für jede Plattform, auf der openUTM freigegeben wird.

Es gibt Handbücher, die für alle Plattformen gültig sind, sowie Handbücher, die jeweils für BS2000-Systeme bzw. für Unix-Systeme und Windows-Systeme gelten.

Sämtliche Handbücher sind als PDF-Datei im Internet verfügbar unter der Adresse

http://manuals.ts.fujitsu.com

Geben Sie dort in das Feld **Produktsuche** den Suchbegriff "openUTM V6.3" ein, um sich alle openUTM-Handbücher der Version 6.3 anzeigen zu lassen.

Die Handbücher sind auf offenen Plattformen auf der Enterprise DVD enthalten und stehen für BS2000-Systeme auf der WinAdmin DVD zur Verfügung.

Die folgenden Abschnitte geben einen Aufgaben-bezogenen Überblick über die Dokumentation zu openUTM V6.3. Eine vollständige Liste der Dokumentation zu openUTM finden Sie im Literaturverzeichnis auf Seite 315.

Einführung und Überblick

Das Handbuch **Konzepte und Funktionen** gibt einen zusammenhängenden Überblick über die wesentlichen Funktionen, Leistungen und Einsatzmöglichkeiten von openUTM. Es enthält alle Informationen, die Sie zum Planen des UTM-Einsatzes und zum Design einer UTM-Anwendung benötigen. Sie erfahren, was openUTM ist, wie man mit openUTM arbeitet und wie openUTM in die BS2000-, Unix- und Windows-Plattformen eingebettet ist.

Programmieren

- 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.
- 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.
- Wenn Sie Daten auf Basis von XML austauschen wollen, benötigen Sie das Dokument **XML für openUTM**. Darin werden die C- und COBOL-Aufrufe beschrieben, die zum Bearbeiten von XML-Dokumenten benötigt werden.
- Für BS2000-Systeme gibt es Ergänzungsbände für die Programmiersprachen Assembler, Fortran, Pascal-XT und PL/1.

Konfigurieren

Zur Definition von Konfigurationen steht Ihnen das Handbuch **Anwendungen generieren** zur Verfügung. Darin ist beschrieben, wie Sie mit Hilfe des UTM-Tools KDCDEF sowohl für eine stand-alone UTM-Anwendung als auch für eine UTM-Cluster-Anwendung

- die Konfiguration definieren
- die KDCFILE erzeugen
- und im Falle einer UTM-Cluster-Anwendung die UTM-Cluster-Dateien erzeugen.

Zusätzlich wird gezeigt, wie Sie wichtige Verwaltungs- und Benutzerdaten mit Hilfe des Tools KDCUPD in eine neue KDCFILE übertragen, z.B. beim Umstieg auf eine neue Version von openUTM oder nach Änderungen in der Konfiguration. Für eine UTM-Cluster-Anwendung wird außerdem gezeigt, wie Sie diese Daten mit Hilfe des Tools KDCUPD in die neuen UTM-Cluster-Dateien übertragen.

Binden, Starten und Einsetzen

Um UTM-Anwendungen einsetzen zu können, benötigen Sie für das betreffende Betriebssystem (BS2000- bzw. Unix-/Windows-Systeme) das Handbuch **Einsatz von openUTM-Anwendungen**.
Dort ist beschrieben, wie man 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. Zudem wird ausführlich auf die Punkte eingegangen, die beim Betrieb von UTM-Cluster-Anwendungen zu beachten sind.

Administrieren und Konfiguration dynamisch ändern

- 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 für den Betrieb einer stand-alone UTM-Anwendung oder einer UTM-Cluster-Anwendung sowie über die Möglichkeiten, mehrere UTM-Anwendungen zentral zu administrieren. Darüber hinaus beschreibt es, wie Sie Message Queues und Drucker mit Hilfe der KDCS-Aufrufe DADM und PADM administrieren können.
- Wenn Sie den grafischen Administrationsarbeitsplatz **openUTM WinAdmin** oder die funktional vergleichbare Web-Anwendung **openUTM WebAdmin** einsetzen, dann steht Ihnen folgende Dokumentation zur Verfügung:
  - Die **WinAdmin-Beschreibung** und die **WebAdmin-Beschreibung** bieten einen umfassenden Überblick über den Funktionsumfang und das Handling von WinAdmin/WebAdmin. Die Dokumente werden jeweils mit der Software ausgeliefert und sind zusätzlich auch online als PDF-Datei verfügbar.
  - Das jeweilige **Online-Hilfesystem** beschreibt kontextsensitiv alle Dialogfelder und die zugehörigen Parameter, die die grafische Oberfläche bietet. Außerdem wird dargestellt, wie man WinAdmin bzw. WebAdmin konfiguriert, um stand-alone UTM-Anwendungen und UTM-Cluster-Anwendungen administrieren zu können.

Details zur Integration von openUTM WebAdmin in den SE Manager des SE Servers finden Sie im SE Server Handbuch **Bedienen und Verwalten**.

Testen und Fehler diagnostizieren

Für die o.g. Aufgaben benötigen Sie außerdem die Handbücher **Meldungen, Test und Diagnose** (jeweils ein Handbuch für Unix-/Windows-Systeme und für BS2000-Systeme). Sie beschreiben das Testen einer UTM-Anwendung, den Inhalt und die Auswertung eines UTM-Dumps, das Verhalten im Fehlerfall, das Meldungswesen von openUTM, sowie alle von openUTM ausgegebenen Meldungen und Returncodes.

openUTM-Clients erstellen

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

- Das Handbuch **openUTM-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 die C++-Klassen für die schnelle und einfache Programmerstellung nutzen können.
- Das Handbuch **openUTM-Client für Trägersystem OpenCPIC** beschreibt, wie man OpenCPIC installiert und 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.
- Für die mit **BeanConnect** ausgelieferten **JUpic-Java-Klassen** wird die Dokumentation mit der Software ausgeliefert. Diese Dokumentation besteht aus Word- und PDF-Dateien, die die Einführung und die Installation beschreiben, sowie aus einer Java-Dokumentation mit der Beschreibung der Java-Klassen.
- Das Handbuch **BizXML2Cobol** beschreibt, wie Sie bestehende Cobol-Programme einer UTM-Anwendung so erweitern können, dass sie als Standard-Web-Service auf XML-Basis genutzt werden können. Die Arbeit mit der grafischen Bedienoberfläche ist in der zugehörigen **Online-Hilfe** beschrieben.
- Wenn Sie UTM-Services auf einfache Weise ins Web stellen möchten, benötigen Sie das Handbuch **Web-Services für openUTM**. Das Handbuch beschreibt, wie Sie mit dem Software-Produkt WS4UTM (WebServices for openUTM) Services von UTM-Anwendungen als Web Services verfügbar machen. Die Arbeit mit der grafischen Bedienoberfläche ist in der zugehörigen **Online-Hilfe** beschrieben.

Kopplung mit der IBM-Welt

Wenn Sie aus Ihrer UTM-Anwendung mit Transaktionssystemen von IBM kommunizieren wollen, benötigen Sie außerdem das Handbuch **Verteilte Transaktionsverarbeitung zwischen openUTM 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 openUTM-LU62, sowie dessen Installation, Generierung und Administration.

Dokumentation zu PCMX

Mit openUTM auf Unix- und Windows-Systemen wird die Kommunikationskomponente PCMX ausgeliefert. Die Funktionen von PCMX sind in folgenden Dokumenten beschrieben:

- Handbuch CMX (Unix-Systeme) "Betrieb und Administration" für Unix-Systeme
- Online-Hilfe zu PCMX für Windows-Systeme

1.2.2 Dokumentation zum openSEAS-Produktumfeld

Die Verbindung von openUTM zum openSEAS-Produktumfeld wird im openUTM-Handbuch **Konzepte und Funktionen** kurz dargestellt. Die folgenden Abschnitte zeigen, welche der openSEAS-Dokumentationen für openUTM von Bedeutung sind.

Integration von Java EE Application Servern und UTM-Anwendungen

Der Adapter BeanConnect gehört zur Produkt-Suite openSEAS. Der BeanConnect-Adapter realisiert die Verknüpfung zwischen klassischen Transaktionsmonitoren und Java EE Application Servern und ermöglicht damit die effiziente Integration von Legacy-Anwendungen in Java-Anwendungen.

- Das Handbuch **BeanConnect** beschreibt das Produkt BeanConnect, das einen JCA 1.5- und JCA 1.6-konformen Adapter bietet, der UTM-Anwendungen mit Anwendungen auf Basis von Java EE , z.B. mit dem Application Server von Oracle, verbindet.

  Die Handbücher zum Application Server von Oracle sind bei Oracle beziehbar.

Web-Anbindung und Anwendungsintegration

Zum Anschließen neuer und bestehender UTM-Anwendungen an das Web mit dem Produkt WebTransactions benötigen Sie die Handbücher zu **WebTransactions**.

Die Dokumentation wird durch JavaDocs ergänzt.

1.2.3 Readme-Dateien

Funktionelle Änderungen und Nachträge der aktuellen Produktversion zu diesem Handbuch entnehmen Sie bitte ggf. den Produkt-spezifischen Readme-Dateien.

Readme-Dateien stehen Ihnen online bei dem jeweiligen Produkt zusätzlich zu den Produkthandbüchern unter *http://manuals.ts.fujitsu.com* zur Verfügung. Für die Plattform BS2000 finden Sie Readme-Dateien auch auf der Softbook-DVD.

Informationen unter BS2000-Systemen

Wenn für eine Produktversion eine Readme-Datei existiert, finden Sie im BS2000-System die folgende Datei:

`SYSRME.<product>.<version>.<lang>`

Diese Datei enthält eine kurze Information zur Readme-Datei in deutscher oder englischer Sprache (<lang>=D/E). Die Information können Sie am Bildschirm mit dem Kommando `/SHOW-FILE` oder mit einem Editor ansehen.
Das Kommando `/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product>` zeigt, unter welcher Benutzerkennung die Dateien des Produkts abgelegt sind.

Ergänzende Produkt-Informationen

Aktuelle Informationen, Versions-, Hardware-Abhängigkeiten und Hinweise für Installation und Einsatz einer Produktversion enthält die zugehörige Freigabemitteilung. Solche Freigabemitteilungen finden Sie online unter *http://manuals.ts.fujitsu.com*.

Readme-Datei unter Unix-Systemen

Die Readme-Datei und ggf. weitere Dateien wie z.B. eine Handbuchergänzungsdatei finden Sie im *utmpfad* unter `/docs/`*sprache*.

Readme-Datei unter Windows-Systemen

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

1.3 Neuerungen in openUTM V6.3

Die folgenden Abschnitte gehen näher auf die Neuerungen in den einzelnen Bereichen ein.

1.3.1 Neue Server-Funktionen

Zusätzliche UTM-System-Prozesse für interne Aufgaben

UTM startet zusätzlich zu den per Startparameter angegebenen Prozessen bis zu drei weitere Prozesse, die für interne Aufgaben von openUTM oder privilegierte Aufträge des Administrators freigehalten werden.

Dazu wurden die Generierungs- und die Administrationschnittstelle erweitert:

- Generierung, KDCDEF-Anweisung MAX
  - Neuer Operand PRIVILEGED-LTERM, um ein bestimmtes LTERM als privilegiert auszuzeichnen. Durch die Anmeldung eines Benutzer mit Administrationsberechtigung werden alle Aufträge des Benutzers zu privilegierten Aufträgen.
  - Operand TASKS: Der Maximalwert wurde wegen der zusätzlichen System-Prozesse auf 240 reduziert.
- Administrationsschnittstelle KDCADMI
  - Datenstruktur *kc_max_par_str*: Neues Feld *privileged_lterm* für das generierte privilegierte LTERM.
  - Datenstruktur *kc_tasks_par_str*: Neue Felder *gen_system_tasks* und *curr_system_tasks* für die System-Prozesse.
  - Datenstruktur *kc_curr_par_str*: Neues Feld *curr_system_tasks* für die System-Prozesse.

Höhere Auflösung der verbrauchten CPU-Zeit

Die verbrauchte CPU-Zeit wird für TACs jetzt in Mikrosekunden und für USERs in Millisekunden ausgegeben. Dazu wurden folgende Schnittstellen geändert:

- KDCADMI
  - Datenstruktur *kc_tac_str*: Neues Feld *taccpu_micro_sec* für die durchschnittlich verbrauchte CPU-Zeit in Mikrosekunden.
  - Datenstrukturen *kc_user_str* und *kc_user_dyn1_str*: Neues Feld *cputime_msec* für die verbrauchte CPU-Zeit in Millisekunden.

- Kommando-Schnittstelle KDCADM
  - KDCINF type=TAC: TACCPU gibt die durchschnittlich verbrauchte CPU-Zeit in Mikrosekunden aus.
  - KDCINF type=USER: CPUTIME gibt die verbrauchte CPU-Zeit in Millisekunden aus.
- KDCEVAL-Listen
  - In den KDCEVAL-Listen werden einige Zeiten jetzt in Mikrosekunden ausgegeben.

Neue Trace-Funktionen

Im laufenden Betrieb können zusätzliche Traces ein- und ausgeschaltet werden:

- ADMI Trace, d.h. Trace der Programmschnittstelle zur Administration (KDCADMI)
- X/Open Traces (CPI-C, TX, XATMI)

Dazu wurden folgende Schnittstellen erweitert:

- Startparameter:

  Neue Startparameter ADMI-TRACE, CPIC-TRACE, TX-TRACE und XATMI-TRACE zum Einschalten der Traces.

- KDCADMI:

  Datenstruktur *kc_diag_and_account_par_str*: Neue Felder *admi_trace*, *cpic_trace*, *tx_trace* und *xatmi_trace* zum Ein- und Ausschalten der Traces.

KDCDEF-Ein-/Ausgabe über LMS-Bibliothekselemente

In BS2000-Systemen können KDCDEF-Anweisungen aus LMS-Bibliothekselementen gelesen und beim inversen KDCDEF in LMS-Bibliothekselemente ausgegeben werden. Dazu wurden folgende Schnittstellen erweitert:

- Generierung
  - KDCDEF-Anweisung OPTION: Neuer Operandenwert LIBRARY-ELEMENT(...) beim Operanden DATA.
  - KDCDEF-Anweisung CREATE-CONTROL-STATEMENTS: Neuer Operandenwert LIBRARY-ELEMENT(...) beim Operanden TO-FILE.
- KDCADMI

  Datenstruktur *kc_create_statements_str*: Neue Felder *lib_name*, *elem_name*, *vers*, *type*, *stmt_type* und *file_error_code*.

- Meldungen

  Neue Meldungen K234, K519 und K520 beim Lesen von KDCDEF-Anweisungen aus LMS-Bibliothekselementen und beim Ausgeben von KDCDEF-Anweisungen in LMS-Bibliothekselemente.

Performanceverbesserungen

- UTM-Cache

  Der UTM-Cache wurde optimiert, um die Performance bei intensiver Nutzung des UTM-Cache (z.B. bei sehr umfangreichen Vorgangsdaten) zu verbessern.

- UTM-Lock Algorithmus

  Für konkurrierende Zugriffe auf UTM-interne Verwaltungsdaten wird auf offenen Plattformen durchgängig die vom Betriebssystem angebotene Compare&Swap Funktionalität verwendet.

- UTM-Netzanbindung

  Die Netzanbindung auf offenen Plattformen wurde dahingehend verbessert, dass insbesondere bei Niederlast beim Senden von Daten an UTM-Partneranwendungen keine Verzögerungen mehr auftreten.

Sonstige Änderungen

- Meldungen
  - Der Meldungsbereich für Systemmeldungen wurde vergrößert und umfasst jetzt den Bereich von K001 bis K399 (bisher bis K249). Damit haben sich folgende Meldungsbereiche verschoben:
    - Die Meldungsnummern für Meldungen, die ausschließlich von KDCUPD ausgegeben werden, belegen jetzt den Bereich von K800 bis K899 statt K250 bis K322.

      Meldungen, die sowohl von KDCUPD als auch vom Online-Import ausgegeben werden, gelten als Systemmeldungen und bleiben unverändert.
    - Die Meldungsnummern für KDCCSYSL- und KDCPSYSL-Meldungen belegen jetzt den Bereich K600 bis K649 statt K550 bis K599.
  - Neue Meldung K235, falls die Namensauflösung für einen Rechner zu lange dauert.
  - Bei den Meldungen K162 und K163 wurden die Standard-Meldungsziele geändert.

- KDCADMI
  - Die Felder *auto_connect* bei *kc_lpap_str* und *auto_connect_number* bei *kc_osi_lpap_str* besitzen die Eigenschaft GPD statt PD, d.h. Änderungen für diese Felder wirken nun Anwendungs-global. Eine administrative Änderung der Eigenschaften "Automatischer Verbindungsaufbau" bei LPAP und "Anzahl der Verbindungen" bei OSI-LPAP wirkt über das Anwendungsende hinaus.
  - Neues Feld *max_btrace_lth* bei *kc_diag_and_account_par_str* für die maximale Länge der Aufzeichnungsdaten bei eingeschalteter BCAM-Trace-Funktion.
- Für Plattformen, auf denen UTM im 64-Bit Mode laufen kann, ermöglicht KDCUPD den Umstieg von einer 32-Bit-Anwendungsumgebung auf eine 64-Bit-Anwendungsumgebung. Derzeit unterstützt UTM den 64-Bit-Modus nur auf Unix-Plattformen.
- Die Oracle User-Id kann bei den KDCDEF-Anweisungen DATABASE und RMXA auch in Kleinbuchstaben angegeben werden.
- Auf Windows-Systemen wird das Installationsverfahren InstallAware verwendet. Daher wird openUTM auf Windows-Systemen in Form von MSI-Dateien ausgeliefert.
- Neues Beispielprogramm ADJTCLT (ADJust Tac-CLass Table)

  Mit dem C-Teilprogramm ADJTCLT kann der Anwender steuern, wie die Prozesse auf die TAC-Klassen aufgeteilt werden, und zwar abhängig von der aktuellen Anzahl aller Prozesse und der aktuellen Anzahl der Asynchron-Prozesse. Dazu erstellt der Anwender eine Tabelle mit den gewünschten Einstellungen. Die Einstellungen müssen so gewählt werden, dass immer mindestens ein Prozess frei ist, um andere Aufgaben, z.B. Transaktionsende-Verarbeitung von verteilten Transaktionen, zu erledigen.

1.3.2 Last-Simulation mit "Workload Capture & Replay"

Mit der neuen Funktion Workload Capture & Replay kann die Kommunikation von UTM-Anwendungen mit UPIC-Clients mitgeschnitten und anschließend mit einstellbaren Lastprofilen abgespielt werden. Damit lässt sich das Verhalten der UTM-Anwendung bei hoher Last unter realen Bedingungen testen.

Workload Capture & Replay besteht aus folgenden Komponenten:

- *UPIC Capture*: schneidet die Kommunikation mit dem UPIC-Client mit.

  Zum Mitschneiden einer UPIC-Session (Capture) wird die Trace-Funktion BTRACE (BCAM-Trace) verwendet, die auf allen Server-Plattformen vorhanden ist.
- *UPIC Analyzer*: dient zur Analyse der mitgeschnittenen Kommunikation.
- *UPIC Replay*: dient zum Abspielen der mitgeschnittenen UPIC-Session mit unterschiedlichen Lastparametern (Geschwindigkeit, Client-Anzahl).

UPIC Analyzer und *UPIC Replay* stehen nur auf 64-Bit-Linux-Systemen zur Verfügung und sind Lieferbestandteil des openUTM-Client (UPIC).

Zusätzlich wird mit openUTM auf Unix- und Windows-Systemen das Dienstprogramm *kdcsort* ausgeliefert. Mit *kdcsort* können Sie die von BTRACE mitgeschnittene Kommunikation zeitlich sortieren, wenn die UTM-Anwendung beim Mitschneiden mit mehr als einem Prozess gelaufen ist und deshalb mehrere prozess-spezifische Dateien erzeugt wurden.

1.3.3 Neue Client-Funktion

Der UPIC-Client steht auf Windows-Systemen neben der 32-Bit Variante zusätzlich in einer 64-Bit Variante zur Verfügung.

1.3.4 Neue und geänderte Funktionen für openUTM WinAdmin

- WinAdmin unterstützt alle Neuerungen der UTM V6.3 bzgl. der Programmschnittstelle zur Administration. Dazu gehören z.B. die neuen Trace-Funktionen, das Schreiben von KDCDEF-Anweisungen in Bibliothekselemente beim Ablauf des inversen KDCDEF im BS2000 oder die Anzeige der verbrauchten CPU-Zeit eines Users in Millisekunden.
- Einführung einer Lebensdauer für Statistikwerte, um die Anzahl der in der Konfigurationsdatenbank gespeicherten Statistikwerte zu beschränken.

1.3.5 Neue Funktionen für openUTM WebAdmin

Zusatzfunktionen

WebAdmin bietet weitere Zusatzfunktionen, die über die Funktionalität der Administrationsschnittstelle KDCADMI hinausgehen und bisher nur in WinAdmin zur Verfügung standen:

- Message Queues anzeigen (DADM-Funktionalität)
- Statistikkollektoren verwalten und deren Werte tabellarisch anzeigen (einschließlich der neuen Funktion "Lebensdauer für Statistikwerte")
- Statistiken in grafischer Form darstellen (Verlaufsgrafik)
- Schwellwert-Aktionen für Statistikkollektoren ausführen

Unterstützung der Neuerungen in openUTM V6.3

WebAdmin unterstützt alle Neuerungen von UTM V6.3 bzgl. der Programmschnittstelle zur Administration. Dazu gehören z.B. die neuen Trace-Funktionen, das Schreiben von KDCDEF-Anweisungen in Bibliothekselemente beim Ablauf des Inversen KDCDEF im BS2000 oder die Anzeige der verbrauchten CPU-Zeit eines Users in Millisekunden.

Integration in den SE Server

WebAdmin kann auf der Management Unit (SE Manager) eines SE Servers als Add-on installiert werden und bietet dann im Wesentlichen den selben Funktionsumfang wie bei einem Betrieb außerhalb des SE Managers.

2 openUTM - Leistungsüberblick

Dieses Kapitel gibt einen Überblick über die wichtigsten Funktionen des Anwendungsservers openUTM und geht auf die IT-Umgebung ein, in der openUTM typischerweise eingesetzt wird.

2.1 openUTM – die „high-end Transaction Processing Platform“

Als high-end Transaction Processing Platform wird openUTM für OLTP-Anwendungen eingesetzt (OLTP=Online Transaction Processing). Dies sind z.B. Anwendungen 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. openUTM ist auch die Basis für unternehmensweite, integrierte IT-Lösungen auf der Grundlage von Client/Server-Konzepten oder für Lagerhaltungs- und Produktionssteuerungssysteme.

openUTM beherrscht sowohl dialog-gesteuerte als auch asynchrone, vom Dialog entkoppelte Services. Seine Message Queuing-Eigenschaften unterstützen Workflow-Konzepte, Mobile Computing und ähnliche Anwendungen.

openUTM übernimmt als Anwendungsserver die folgenden Aufgaben:

- openUTM bildet die Ablaufumgebung für Service-Routinen.
- Die Service-Routinen können statisch generiert oder dynamisch im laufenden Betrieb zur Anwendung hinzugefügt werden.
- Die Service-Routinen nutzen standardisierte Schnittstellen zum UTM-Anwendungsserver (X/Open: CPI-C, TX, XATMI; DIN: KDCS).
- Die Service-Routinen können direkt auf Datenbanken zugreifen (mittels SQL).
- Der UTM-Anwendungsserver koordiniert die Transaktionen mit dem Datenbanksystem.
- openUTM steuert den transaktionsgesicherten Informationsaustausch zwischen Clients, Anwendungen und Ressourcen. Dadurch garantiert openUTM Zuverlässigkeit, Verfügbarkeit und Performance - selbst bei komplexen, verteilten Strukturen.

Für die Administration einer UTM-Anwendung bietet openUTM die Programmschnittstelle KDCADMI.

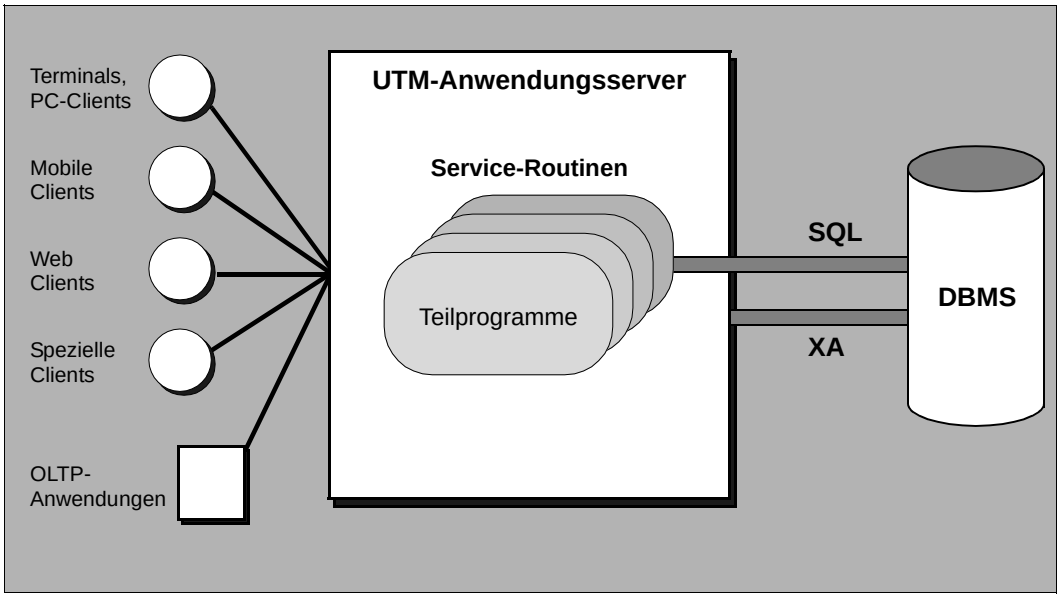


Bild 1: Der openUTM-Anwendungsserver als high-end Transaction Processing Platform

openUTM unterstützt Multi-Tier-Architekturen, die eine Verteilung der Business-Logik (Verarbeitung) auf mehrere Server vorsehen. openUTM übernimmt dabei die Funktion eines verteilten High-Level-Betriebssystems.

Beim Design von Anwendungen kann auf ein Fundament mächtiger Funktionen aufgebaut werden: openUTM 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, um 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.

Detaillierte Informationen über Client/Server-Computing und Server/Server-Kommunikation mit openUTM finden Sie in diesem Handbuch im Kapitel „Integrationsszenarien mit openUTM“ auf Seite 69.

2.2 Transaktionskonzept und Restart-Funktionen

Zu den wichtigsten Aufgaben von openUTM 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 openUTM 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.

openUTM 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 openUTM 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 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 „Wiederanlauf-Funktionen von openUTM“ auf Seite 223.

2.3 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 „X/Open-Konformität von openUTM“ auf Seite 50). openUTM 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

openUTM auf BS2000-Systemen unterstützt die Koordination mit folgenden Datenbank-Systemen:

- UDS/SQL
- SESAM/SQL
- Oracle
- LEASY (das Dateisystem LEASY verhält sich gegenüber openUTM wie ein Datenbank-System)

openUTM für Unix- und Windows-Systeme unterstützt die Koordination mit folgenden Datenbank-Systemen:

– Oracle
– INFORMIX

Koordination zwischen UTM-Transaktionen und Datenbank-Transaktionen

Die in einer Datenbank gespeicherten Datensätze unterliegen nicht den Sperr- und Sicherungsmechanismen von openUTM, 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. openUTM 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 2 auf Seite 32 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 openUTM und Datenbank-Systemen.

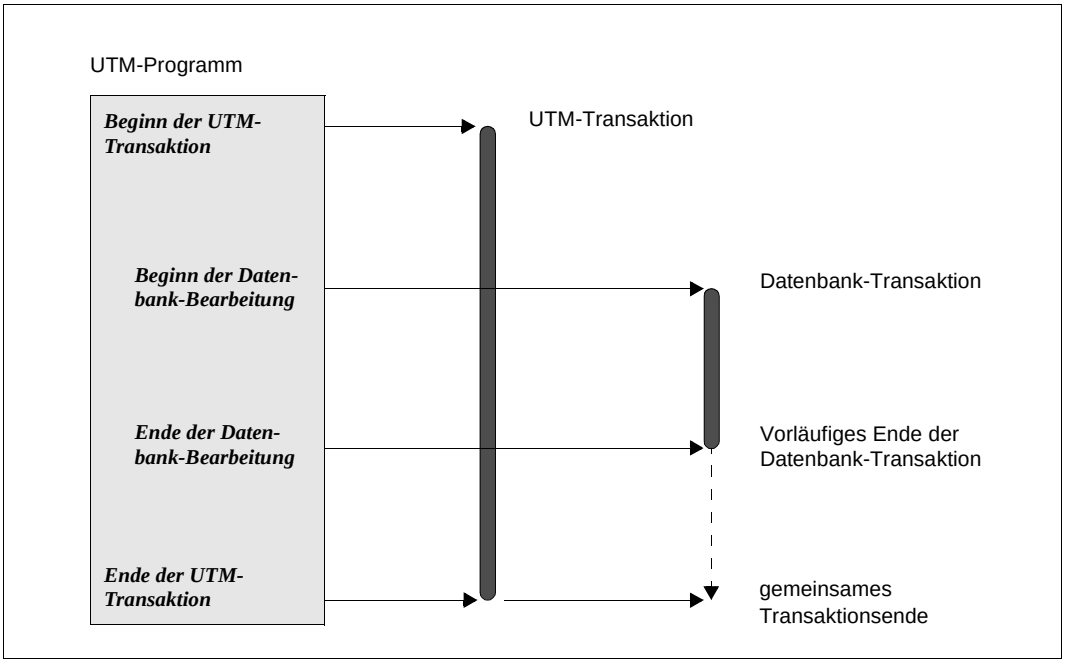


Bild 2: Koordination von UTM- und Datenbank-Transaktion über Two-Phase-Commit

Koordination mit verteilten und heterogenen Datenbank-Systemen

Die Koordination mit einem Resource Manager gehört zum Funktionsumfang eines jeden Transaktionssystems. openUTM 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. openUTM 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 openUTM 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 openUTM die Fehlerbehandlung - der Programmierer muss keine speziellen Vorkehrungen treffen.

Das Datenbank-System informiert openUTM, ob es einen Aufruf erfolgreich durchführen konnte. Ist ein Fehler aufgetreten, so prüft openUTM den Fehlergrad und reagiert entsprechend: openUTM 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.

Unterstützung von Failover mit Oracle Real Application Clusters

Bei Zusammenarbeit mit Oracle Real Application Clusters ist openUTM im Failover-Fall in der Lage, den Anwendungslauf normal fortzusetzen. Transaktionen, die durch den Failover unterbrochen wurden, werden wenn möglich ordnungsgemäß beendet. Ist dies nicht möglich, dann werden sie entweder durch openUTM oder durch das Datenbank-System zurückgesetzt, so dass die Datenbank auch nach dem Failover in einem konsistenten Zustand ist.

Interne Schnittstelle zwischen openUTM und Datenbank-Systemen

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

openUTM verwendet die von X/Open standardisierte XA-Schnittstelle, in BS2000-Systemen wird zusätzlich die funktional gleichwertige Schnittstelle IUTMDB unterstützt.

Unterstützung weiterer Resource Manager

Datenbank-Systeme sind nur ein bestimmter Typ von Resource Managern.
openUTM 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 UTM Resource Manager handelt (UTM Message Queues, lokale Speicher, Logging-Dateien) oder um einen externen Resource Manager, wie z.B. Message Queuing-Systeme anderer Hersteller: openUTM koordiniert in jedem Fall die Transaktionssicherung über die Grenzen von Anwendungen, Betriebssystemen und Hardware-Plattformen hinweg.

2.4 UTM-Cluster-Anwendung

Eine wesentliche Funktion von openUTM ist die Unterstützung von Clustern.

Ein Cluster ist eine Anzahl von Rechnern (Knoten), die über ein schnelles Netzwerk verbunden sind, und die sich eine gemeinsame Peripherie teilen. Auf einem Cluster kann eine UTM-Anwendung in Form einer UTM-Cluster-Anwendung betrieben werden.

In einer UTM-Cluster-Anwendung steht Ihnen wie bei einer stand-alone Anwendung der volle Funktionsumfang von openUTM zur Verfügung.

UTM-Cluster-Anwendung

Im Unterschied zu einer einzelnen (stand-alone) UTM-Anwendung besteht eine UTM-Cluster-Anwendung aus mehreren identisch generierten UTM-Anwendungen, die auf den Knoten eines Clusters laufen, und die man logisch als eine Anwendung betrachten kann. Jede dieser sogenannten Knoten-Anwendungen läuft auf einem eigenen Knoten.

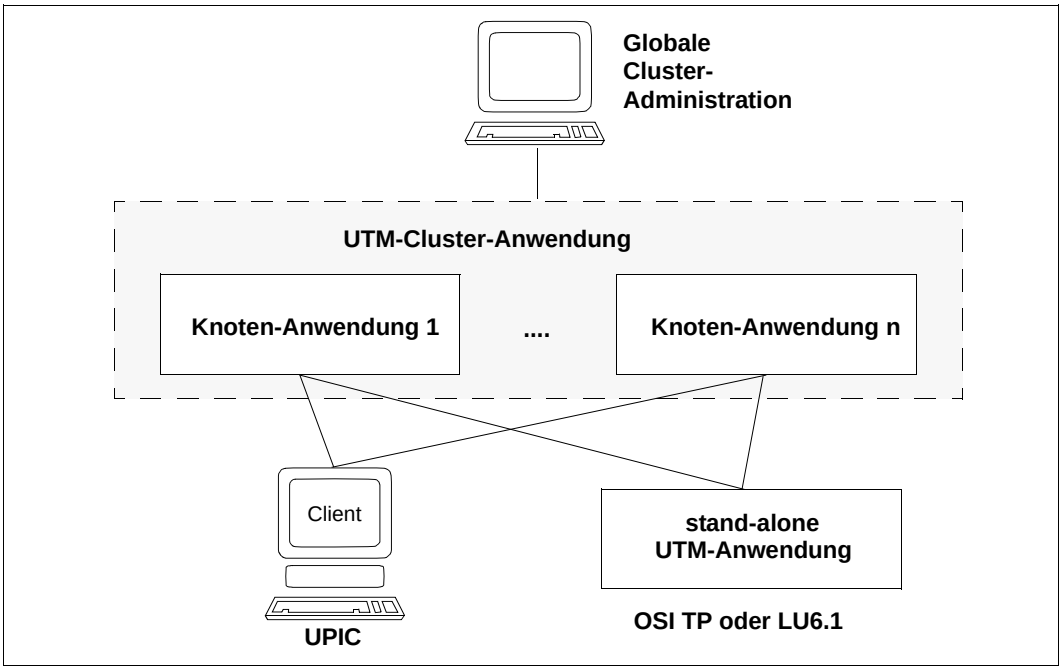


Bild 3: UTM-Cluster-Anwendung

Eine UTM-Cluster-Anwendung kann in BS2000-Systemen auf bis zu 16 Knoten und in Unix- oder Windows-Systemen auf bis zu 32 Knoten verteilt werden.

Eine UTM-Cluster-Anwendung bietet Vorteile bei der Lastverteilung und der Hochverfügbarkeit:

- Zentrale Hochverfügbarkeitsfunktionen wie Anwendungsüberwachung, Online-Import von Anwendungsdaten und Online-Update von Anwendungsprogrammen und UTM-Korrekturstufen gewährleisten die Hochverfügbarkeit der UTM-Cluster-Anwendungen im 7x24-Stunden-Betrieb. Wird eine Knoten-Anwendung normal beendet, können Benutzer offene Vorgänge an einer beliebigen anderen Knoten-Anwendung fortsetzen.
- Für die Kommunikation einer UTM-Anwendung mit einer UTM-Cluster-Anwendung bietet openUTM außerdem für LU6.1- und OSI TP-Kommunikation die Möglichkeit einer automatischen Lastverteilung über LPAP-Bündel.
- Für die Kommunikation von Clients mit einer UTM-Cluster-Anwendung besteht die Möglichkeit einer Lastverteilung auf die einzelnen Knoten-Anwendungen mit Hilfe eines externen Lastverteilers. Für die UPIC-Kommunikation bietet openUTM den UPIC-Lastverteiler für UPIC-Clients.
- Mit einer UTM-Cluster-Anwendung ist – im Gegensatz zu einer stand-alone UTM-Anwendung – eine optimale Lastverteilung mit Oracle® RAC möglich.

Detaillierte Information zu den Hochverfügbarkeitsfunktionen und zur Lastverteilung mit UTM-Cluster-Anwendungen entnehmen Sie dem Kapitel „Hochverfügbarkeit und Lastverteilung mit UTM-Cluster-Anwendungen“ auf Seite 209.

Administration einer UTM-Cluster-Anwendung

Sie haben folgende Möglichkeiten, eine UTM-Cluster-Anwendung zu administrieren:

- über eigene Administrationsprogramme, die Sie mit der Programmschnittstelle der Administration (KDCADMI) erstellen,
- über die Kommando-Schnittstelle der Administration,
- über die grafischen Administrationstools WinAdmin und WebAdmin (siehe Seite 64 bzw. Seite 66), die ebenfalls die Programmschnittstelle der Administration nutzen.

Je nach Administrationsauftrag wirken diese Änderungen entweder nur lokal auf der einzelnen Knoten-Anwendung, an der Sie angemeldet sind, oder global in allen Knoten-Anwendungen.

Detaillierte Information zur Administration einer UTM-Cluster-Anwendung über die Programmschnittstelle entnehmen Sie dem openUTM-Handbuch „Anwendungen administrieren“.

Administration einer UTM-Cluster-Anwendung über WinAdmin und WebAdmin

Mit WinAdmin und WebAdmin stehen Ihnen komfortable Funktionen zur Administration einer UTM-Cluster-Anwendung zur Verfügung. Beide Tools bieten sowohl eine Cluster-globale Sicht als auch eine Knoten-lokale Sicht an, wobei administrative Änderungen, die Cluster-global wirken, nur im Cluster-globalen Kontext und administrative Änderungen, die Knoten-lokal wirken, nur im Knoten-lokalen Kontext angeboten werden.

WinAdmin bzw. WebAdmin sammeln die Daten von allen aktiven Knoten-Anwendungen und zeigen sie zusammengefasst an der Oberfläche an.

Neben administrativen Änderungen der Knoten-Anwendungen können Sie sich auch Statistikdaten einer Knoten-Anwendung und über WinAdmin auch aller Knoten-Anwendungen der UTM-Cluster-Anwendung anzeigen lassen.

Außerdem können Sie sich einfach über die Verfügbarkeit der Knoten-Anwendungen informieren.

Detaillierte Information zur Administration einer UTM-Cluster-Anwendung über WinAdmin und WebAdmin entnehmen Sie der jeweiligen Online-Hilfe.

2.4.1 UTM-Cluster-Dateien

Die Konfiguration einer UTM-Cluster-Anwendung beinhaltet folgende UTM-Cluster-Dateien, auf die alle Knoten-Anwendungen gemeinsam zugreifen:

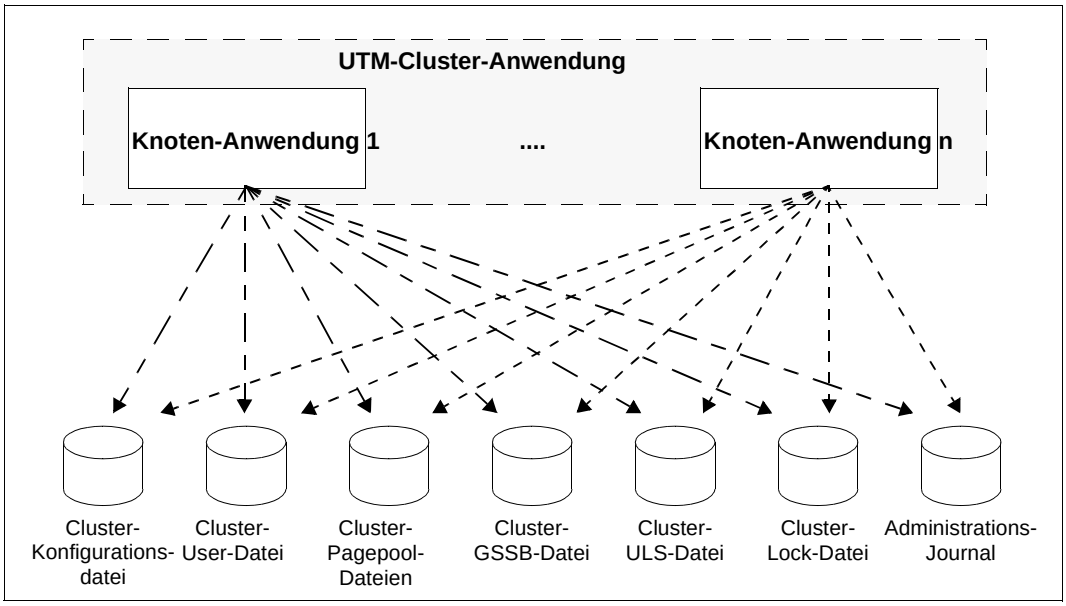


Bild 4: Gemeinsamer synchronisierter Zugriff der Knoten-Anwendungen auf die UTM-Cluster-Dateien

Die Cluster-Konfigurationsdatei, die Cluster-User-Datei, die Cluster-Pagepool-Dateien sowie die Cluster-GSSB- und die Cluster-ULS-Datei werden bei der Erst-Generierung einer UTM-Cluster-Anwendung in einem Basis-Generierungslauf erzeugt. Die Cluster-Lock-Datei und die Dateien des Administrations-Journals werden von der Anwendung beim ersten Startup erzeugt.

Alle Knoten-Anwendungen müssen Zugriff auf die UTM-Cluster-Dateien haben. Um diesen gemeinsamen Zugriff zu ermöglichen, müssen diese Dateien auf Shared Public Volume Sets in BS2000-Systemen, Network File System/Service (NFS) auf Unix-Systemen oder Netz-Laufwerken auf Windows-Systemen abgelegt werden.

Rechnerübergreifende Sperrmechanismen synchronisieren die Zugriffe auf diese Dateien.

Die globalen Cluster-Dateien werden in einem gemeinsamen Verzeichnis bzw. mit identischem Dateinamens-Präfix abgelegt. Im Einzelnen sind dies folgende Dateien:

- Zur Verwaltung des Clusters gibt es die zentrale **Cluster-Konfigurationsdatei**, die die Konfiguration der UTM-Cluster-Anwendung enthält.

- Zur Verwaltung der Benutzer einer UTM-Cluster-Anwendung dient die **Cluster-User-Datei**. Diese Datei kann während des Betriebs einer Anwendung erweitert werden, wenn der Administrator neue Benutzer für die UTM-Cluster-Anwendung definiert, oder wenn ein neuer KDCDEF-Lauf für die Anwendung durchgeführt wird.
- Die Cluster-weit gültigen Anwenderdaten einer UTM-Cluster-Anwendung werden in **Cluster-Pagepool-Dateien** gehalten. Dies sind z.B. Inhalte von GSSBs und ULS oder Vorgangsdaten von Benutzern, damit Benutzer offene Vorgänge an einem anderen Knoten fortsetzen können.
- Die Globalen Speicherbereiche GSSB und ULS werden in einer UTM-Cluster-Anwendung über die **Cluster-GSSB-Datei** und die **Cluster-ULS-Datei** verwaltet.
- Die **Cluster-Lock-Datei** dient dazu, die Sperren für die Anwenderdaten zu verwalten.
- Um Cluster-globale administrative Änderungen in allen Knoten-Anwendungen durchzuführen, werden diese Aktionen in das **Administrations-Journal** geschrieben. Anhand des Administrations-Journals vollziehen alle Knoten-Anwendungen diese Änderungen nach.

  Global wirkende Aktionen sind für jede Knoten-Anwendung der UTM-Cluster-Anwendung gültig, unabhängig davon, ob eine Knoten-Anwendung gerade aktiv ist oder nicht. Beispiele für solche Änderungen sind das dynamische Erzeugen von Objekten, z.B. von neuen Benutzern (USER), oder eine Änderung des Status eines Objekts. Global wirkende Aktionen kann ein Administrator von jeder Knoten-Anwendung der UTM-Cluster-Anwendung aus veranlassen.

Neben den von allen Knoten-Anwendungen gemeinsam verwendeten **globalen** Cluster-Dateien gibt es pro Knoten **lokale** Dateien, z.B KDCFILE (siehe Seite 126), Protokoll- und Diagnosedateien. Die KDCFILE wird nur einmal erzeugt und dann für jede Knoten-Anwendung einmal kopiert. Beim Ablauf verwendet jede Knoten-Anwendung ihre Kopie exklusiv. Auch die KDCFILEs der einzelnen Knoten-Anwendungen müssen so angelegt werden, dass sie von allen Knoten-Anwendungen zugreifbar sind. Dies ist für die Anwendungsüberwachung und für den Online-Import von Anwendungsdaten notwendig.

Detaillierte Information zum Administrations-Journal und zur Dateistruktur einer UTM-Cluster-Anwendung finden Sie in

- openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter BS2000-Systemen“ bzw.
- openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter Unix- und Windows-Systemen“.

2.4.2 Systemvoraussetzungen für den Einsatz von UTM-Cluster-Anwendungen

Folgende Systemeigenschaften sind Voraussetzung für den Einsatz von UTM-Cluster-Anwendungen:

- Auf den BS2000-Systemen muss die Software HIPLEX® MSCF für rechnerübergreifende Dateizugriffe und zur Synchronisation von Zugriffen auf gemeinsame Dateien (Shared Pubset-Betrieb) installiert sein.
- Auf den Unix-Systemen muss ein Network File System/Service (NFS) zur Verfügung stehen. Dazu muss keine zusätzliche Software auf dem Knoten-Rechner installiert werden.
- Auf Windows-Systemen werden keine zusätzlichen Systemvoraussetzungen benötigt. Der Zugriff auf gemeinsame Dateien wird über die üblichen Windows-Netzlaufwerke realisiert.

Detaillierte Informationen zu den aufgeführten Software-Voraussetzungen entnehmen Sie der Freigabemitteilung.

- Die Knoten eines Clusters müssen alle auf gleichartigen Betriebssystemen laufen, eine Mischkonfiguration, z.B. BS2000- und Unix-Rechner, ist nicht möglich.
- Die Knoten-Rechner eines Clusters müssen alle die gleiche "Systemplattform-Klasse" haben. Folgende "Systemplattform-Klassen" werden unterstützt:
  - BS2000-Systeme (/390 und x86, wobei auch Misch-Konfigurationen möglich sind)
  - Solaris (Sparc): Es sind nur Cluster möglich, bei denen alle Knoten-Anwendungen einheitlich entweder im 32-Bit- oder 64-Bit-Modus ablaufen.
  - Linux (x86): Es muss, wie auf Solaris (Sparc), der gleiche Bit-Modus in allen Knoten verwendet werden.
  - Windows: Keine besonderen Anforderungen.

Detaillierte Informationen dazu entnehmen Sie der Freigabemitteilung.

Die Verwendung von unterschiedlichen Betriebssystem-Versionen auf den Knoten ist prinzipiell möglich. Beachten Sie dabei Details zur Binärkompatibilität der Systeme. Meistens ist Aufwärtskompatibilität möglich.

Mischungen aus den verschiedenen Systemplattform-Klassen, z.B. Linux und Solaris, sind technisch nicht möglich.

- Wenn die Anwendung mit Datenbanken arbeitet, muss auf dem System eine Software installiert sein, mit der der rechnerübergreifende Zugriff auf die gemeinsame Datenbank möglich ist.

2.4.3 Einsatz von SESAM/SQL- und UDS/SQL-Datenbanken im Cluster

SESAM/SQL- und UDS-SQL-Datenbanken werden in (/390 und x86, in der Regel so konfiguriert, dass der Database Handler (DBH) auf demselben Rechner läuft wie die UTM-Anwendung. Wird die Datenbank in einem Cluster genutzt, dann muss diese Konfiguration modifiziert werden, da mindestens eine Knoten-Anwendung remote auf die Datenbank zugreift:

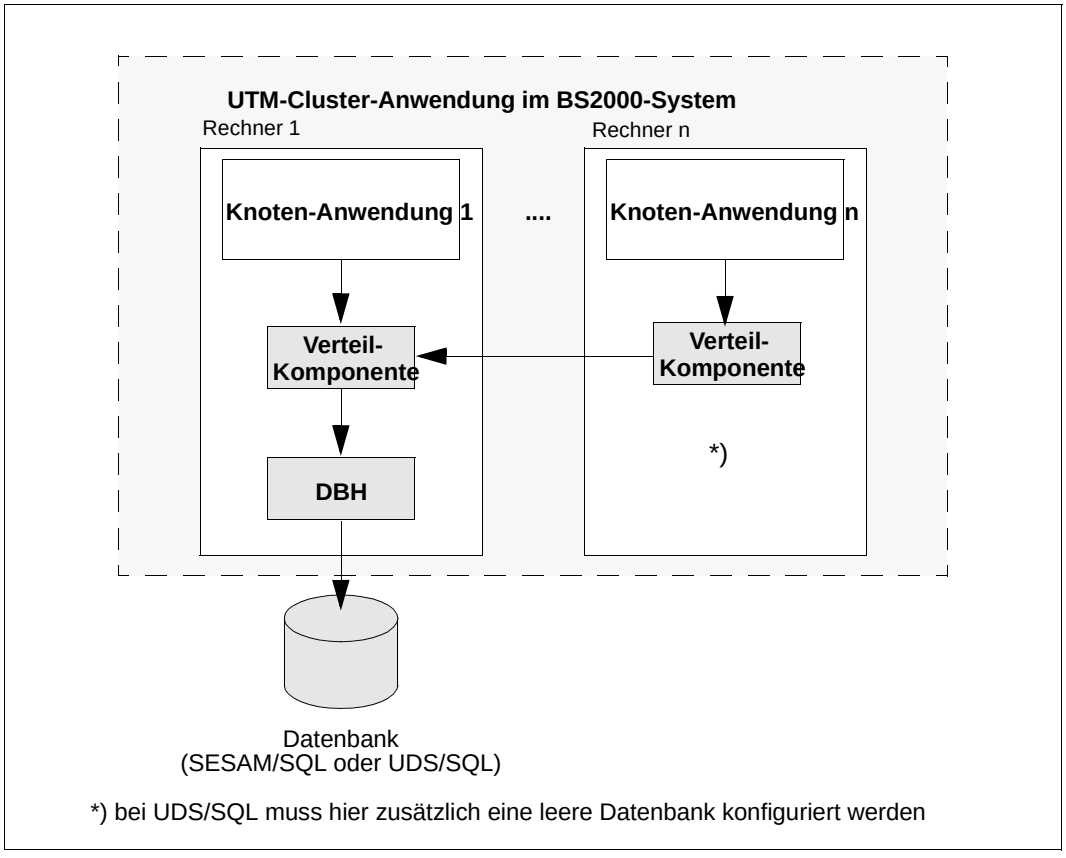


Bild 5: Datenbank-Einsatz im Cluster auf (/390 und x86,

Die Verteil-Komponente ist datenbankspezifisch (SESDCN bei SESAM/SQL bzw.UDS-D bei UDS/SQL). Sie wird benötigt, um modifizierende Remote-Zugriffe mit Transaktionssicherung abzuwickeln und muss daher auf jedem Rechner installiert werden, von dem aus remote auf den DBH zugegriffen wird.

2.5 Message Queuing

Message Queuing (MQ) ist eine Form der Kommunikation, bei der Nachrichten (Messages) nicht unmittelbar, sondern über zwischengeschaltete Queues ausgetauscht werden. In openUTM 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). openUTM unterstützt natürlich auch diesen Programmierstil, bietet aber darüber hinaus ein breites und fein abgestuftes Spektrum an Steuerungsmöglichkeiten. Je nachdem, wer für die Verarbeitung der Nachrichten an einer Queue verantwortlich ist, unterscheidet man UTM-gesteuerte Queues und Service-gesteuerte Queues.

UTM-gesteuerte Queues

Eine UTM-gesteuerte Queue ist eine Message Queue, bei der der Abruf und die Weiterverarbeitung der Nachrichten vollständig durch openUTM gesteuert werden. UTM-gesteuerte Queues werden für Hintergrundaufträge und Ausgabeaufträge verwendet. Sämtliche MQ-Funktionen stehen sowohl sendeseitig als auch empfangsseitig zur Verfügung, daher müssen Sie derartige Queues und deren Queue-Manager nicht eigens generieren oder konfigurieren und auch keine speziellen Trigger- oder Polling-Mechanismen konstruieren.

Service-gesteuerte Queues

Eine Service-gesteuerte Queue ist eine Message Queue, bei der der Abruf und die Weiterverarbeitung der Nachrichten durch Services gesteuert werden. D.h. der Empfänger der Nachricht ist dafür verantwortlich, dass diese gelesen und verarbeitet wird. Service-gesteuerte Queues gibt es in den Varianten USER-Queue, TAC-Queue und Temporäre Queue:

- Eine USER-Queue ist eine Benutzer-spezifische Message Queue. Jedem UTM-Benutzer steht automatisch eine eigene USER-Queue zur Verfügung. Mit USER-Queues können Sie z.B. Mailbox-Mechanismen für UTM-Benutzer realisieren.
- Eine TAC-Queue muss explizit per KDCDEF generiert werden (Ausnahme: Dead-Letter-Queue). TAC-Queues können von jedem Service unter ihrem generierten Namen angesprochen werden. TAC-Queues werden z.B. dazu verwendet, um UTM-Meldungen an den grafischen Administrationsplatz WinAdmin oder die Web-Anwendung Web-Admin weiterzuleiten und dort zu archivieren.
- Eine Temporäre Queue wird dynamisch per Programm erzeugt und kann auch per Programm gelöscht werden. Mit Hilfe von Temporären Queues kann z.B. ein Dialog zwischen zwei voneinander unabhängigen Services realisiert werden.

Transaktionsgesicherter Deferred Delivery Mechanismus

openUTM arbeitet bei allen Message Queues mit dem transaktionsgesicherten Deferred Delivery Mechanismus: 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 openUTM durch die Möglichkeit, beim Message Queuing mit Zeitsteuerung und Priority Scheduling zu arbeiten oder maximale Wartezeiten für Service-gesteuerte Queues zu definieren.

Dialogverarbeitung und Message Queuing sind frei kombinierbar: Innerhalb eines Dialog-services 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 muss.

Realisierung moderner Einsatzkonzepte

Mit seiner Message Queuing-Funktionalität ist openUTM 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 openUTM gewährleistet dies. Auch wenn das Netz gerade nicht verfügbar oder die Empfänger-Anwendung offline ist: openUTM garantiert, dass keine Message verloren geht oder verdoppelt wird.

Durch diese Unabhängigkeit von der Qualität und Verfügbarkeit der Verbindungsstrecke bildet openUTM 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.

Durch den Einsatz von USER-Queues und TAC-Queues lassen sich fein abgestimmte Konzepte für **Mailbox-Systeme** und **Alarm-Mechanismen** entwickeln, in die auch die Clients mit einbezogen sind. Für die Benutzer der Anwendung steigert sich der Komfort erheblich - sowohl für Kunden als auch z.B. für Administratoren. Dies erhöht die Akzeptanz und die Zukunftssicherheit der Anwendung.

Auch **Data Warehouse-Lösungen** und **Decision Support-Systeme** werden durch die Message Queuing-Funktionen von openUTM 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 openUTM dafür, dass die Antwortzeiten kurz bleiben. Falls es sich um reine Retrieval-Anwendungen handelt, bei denen Wiederanlauf eine weniger wichtige Rolle spielt, kann mit der Funktionsvariante UTM-F (**F**ast) gearbeitet werden (siehe Abschnitt „Wiederanlauf mit UTM-F“ auf Seite 226).

Weitere Informationen zum Thema „Message Queuing“ finden Sie in diesem Handbuch im Kapitel „Message Queuing“ auf Seite 103ff.

2.6 openUTM - offen für unterschiedliche Plattformen und Protokolle

Offenheit ist eine der Grundmaximen von openUTM. 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. openUTM steht zur Verfügung

- für Linux-Distributionen wie z.B. SUSE oder Red Hat
- für alle gängigen Unix-Plattformen wie z.B. Solaris, HP-UX oder AIX
- für alle gängigen Windows-Plattformen, z.B. Windows 7
- für BS2000-Systeme, für Business Server mit /390 oder X86-Architektur

Auch der Frontend-Baustein openUTM-Client ist auf diesen Plattformen verfügbar.

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

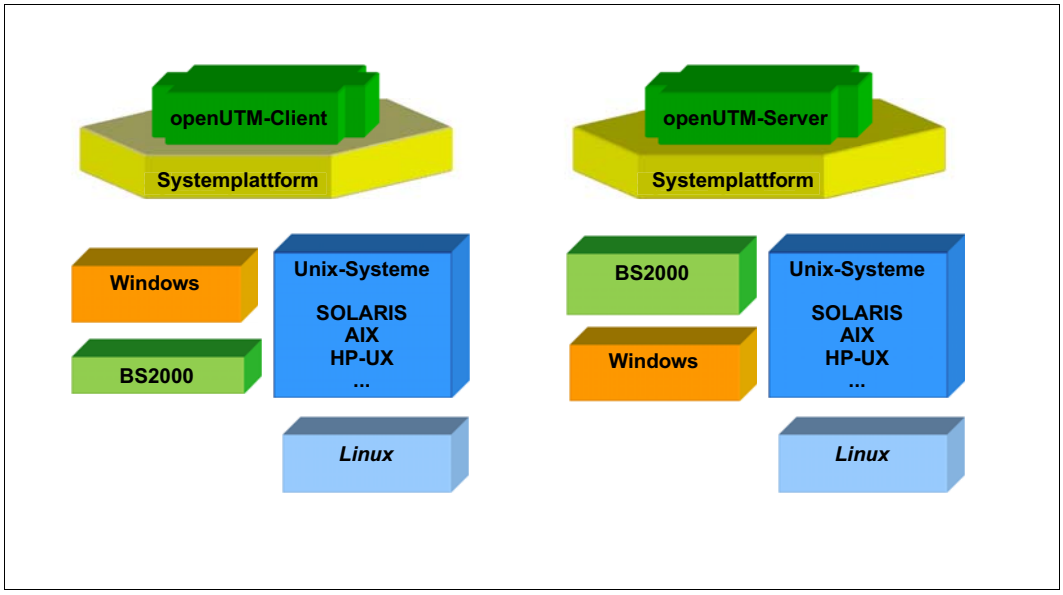


Bild 6: openUTM - verfügbar auf den verschiedensten Plattformen

Durch die hohe Portabilität von openUTM kann das Produkt auf Anfrage auch für weitere Unix-Plattformen zur Verfügung gestellt werden.

Da openUTM auf Mainframes mit BS2000-Betriebssystem genauso ablauffähig ist wie auf Unix- oder Windows-Systemen, können bestehende Mainframe-Anwendungen problemlos mit neuen Anwendungen auf Unix- oder Windows-Systemen gekoppelt werden. So können z.B. kleine Abteilungsserver schnell und flexibel in ein System bestehender Mainframe-Anwendungen eingebunden werden.

Integration in IBM-Mainframe-Umgebungen

openUTM unterstützt die in IBM-Umgebungen häufig verwendeten Kommunikationsprotokolle LU6.1 und LU6.2. Daher können UTM-Anwendungen transaktionsgesichert mit CICS-, IMS/TM- oder IMS/DC-Anwendungen zusammenarbeiten. Weitere Informationen siehe Abschnitt „Kommunikation mit CICS-, IMS- und TXSeries-Anwendungen“ auf Seite 96.

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 Connectivity-Projekte in IBM-Mainframe-Umgebungen ist openUTM 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 openUTM beispielsweise direkt und transaktionsgesichert mit Tuxedo-Anwendungen oder mit Anwendungen in UNISYS-Umgebungen kooperieren.

Integration in das World Wide Web

Über das Produkt WebServices for openUTM, das Produkt WebTransactions oder durch die JUpic-Java-Klassen in BeanConnect kann openUTM in das World Wide Web integriert werden. Die UTM-Anwendungen sind dann mit Web-Browsern wie z.B. Firefox oder InternetExplorer weltweit und von Millionen von Rechnern aus zu erreichen - über eine einheitliche und moderne grafische Oberfläche.

Weitere Einzelheiten finden Sie im Abschnitt „openUTM über Web Services ansprechen“ auf Seite 74 und im Abschnitt „Java-Clients“ auf Seite 84.

Anschluss von Terminals

openUTM bietet neben Schnittstellen zum Anschluss von Client-Programmen den Anschluss von Terminals. Dazu zählen z.B.:

- ältere Geräte wie Alpha-Terminals
- Programme, die solche Terminals emulieren wie z.B. 9750-Emulationen
- Network-Terminals, die von einer PC-Software gesteuert werden, die nicht alle Funktionen eines PCs zulässt.

Diese Terminals können Sie im Zeilenmodus betreiben. In BS2000-Systemen können Sie durch die Zusammenarbeit von openUTM mit Formatierungssystemen wie z.B. FHS auch Bildschirm-Formate (Masken) verwenden. Über Produkte wie WebTransactions lassen sich Bildschirm-Formate dynamisch in grafische Oberflächen umsetzen und so z.B. in Microsoft-Office-Umgebungen bzw. Web-Anwendungen integrieren.

Für die direkte Kommunikation mit Terminal-Benutzern stellt openUTM spezielle Benutzerkommandos zur Verfügung.

Eine Beschreibung der Benutzerkommandos finden Sie im openUTM-Handbuch „Einsatz von openUTM-Anwendungen“. Wie Sie die Terminalschnittstelle für Ihre Service-Routinen nutzen, erfahren Sie im openUTM-Handbuch „Anwendungen programmieren mit KDCS“. Tools für die Formatierung werden im vorliegenden Handbuch noch kurz vorgestellt, siehe Abschnitt „Formatierung“ auf Seite 238 für BS2000-Systeme.

Zusammenarbeit mit beliebigen Applikationen

An openUTM können die verschiedensten Geräte, wie z.B. Sensoren, Steuerungssysteme oder Industrieroboter als Clients angeschlossen werden, sofern diese über eine Schnittstelle auf Ebene des Transportsystems verfügen. Insbesondere können Anwendungen über die weit verbreitete **Socket**-Schnittstelle an openUTM 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 openUTM übergibt (siehe auch Abschnitt „Kommunikation mit Transportsystem-Anwendungen“ auf Seite 98).

Unterstützung verschiedenster Kommunikations- und Netzprotokolle

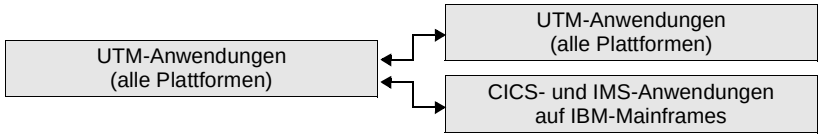
Grundlage für die Integration heterogener Welten bildet die Unterstützung unterschiedlichster Netz- und Kommunikationsprotokolle.

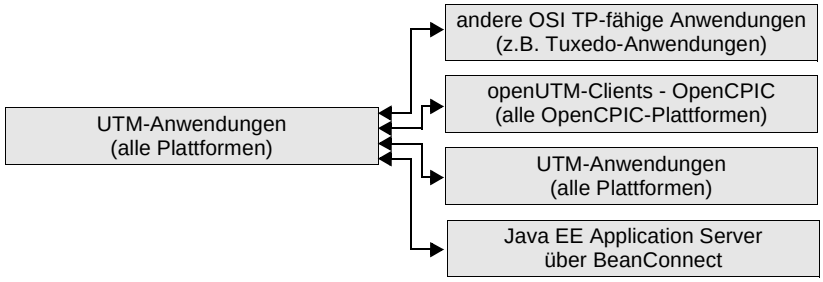
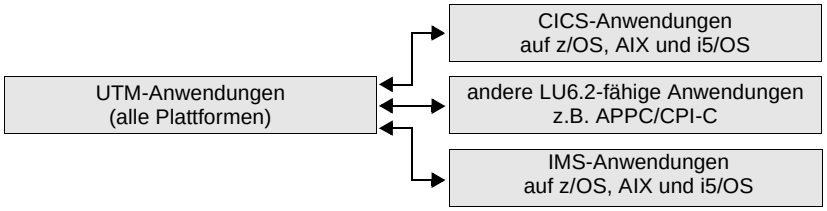
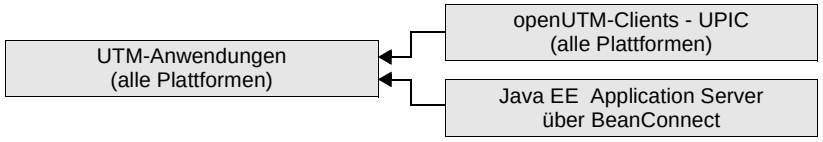
openUTM unterstützt verschiedene Netzprotokolle, z.B. TCP/IP über RFC1006 oder TCP/IP native über Socket-Schnittstelle. Weder Anwender noch Programmierer müssen sich um die unterschiedlichen Netztechniken kümmern: ihre UTM-Anwendungen sind in jeder Umgebung einsetzbar.

openUTM 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 folgende Tabelle 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 Anwendungs-übergreifender 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 Anwendungs-übergreifender 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 openUTM über openUTM-LU62 unterstützt wird. openUTM-LU62 ist aus Sicht von openUTM ein OSI TP-Partner. Damit kann mit und ohne Anwendungs-übergreifende Transaktionssicherung gearbeitet werden. openUTM-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: 

2.7 X/Open-Konformität von openUTM

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

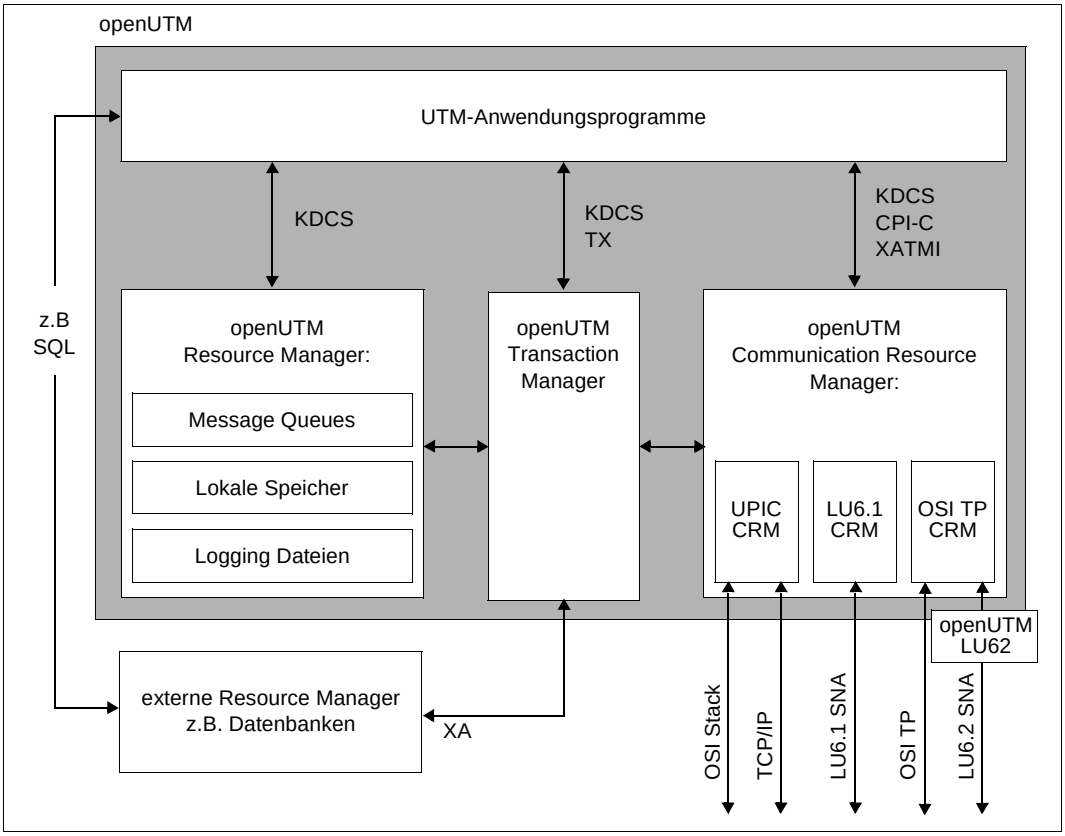


Bild 7: Architektur von openUTM entsprechend dem X/Open-Modell

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

- Anwendungsprogramme:
  Anwendungsprogramme implementieren 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 openUTM 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 Resource 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. openUTM 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 UTM-RMs die Schnittstelle KDCS.

- Communication Resource Manager (CRM):
  Communication Resource Manager kontrollieren in verteilten Systemen die Kommunikation zwischen den Anwendungsprogrammen.
  openUTM stellt CRMs für den internationalen Standard OSI TP, für die Industrie-Standards LU6.1 und LU6.2 sowie für das UTM-eigene Protokoll UPIC zur Verfügung. Näheres siehe auch Abschnitt „openUTM - offen für unterschiedliche Plattformen und Protokolle“ auf Seite 45 und Abschnitt „Kommunikation mit openUTM-Client-Anwendungen“ auf Seite 82 bis Abschnitt „Übersicht: Partner, Protokolle,Transaktionssicherung“ auf Seite 101).

  Der Anschluss von OSI TP ist in openUTM 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 openUTM unterstützten Normen und Standards.

Vorteile der X/Open-Konformität

Die X/Open-Konformität von openUTM eröffnet Ihnen eine Reihe von Vorteilen:

- Portabilität der Anwendungsprogramme durch standardisierte Programmschnittstellen (z.B. CPI-C)
- 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 openUTM zudem auf allen gängigen Hardwareplattformen eingesetzt werden kann und über ausgezeichnete Connectivity-Eigenschaften verfügt (siehe Abschnitt „openUTM - offen für unterschiedliche Plattformen und Protokolle“ auf Seite 45), 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 openUTM 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.

2.8 Performance, Durchsatz und Antwortzeiten

Höchste Performance ist eine der wesentlichen Stärken von openUTM. openUTM 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 openUTM 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. openUTM kann so die Last einer großen Zahl von simultanen Requests auf eine kleine Zahl von Prozessen verteilen.

Zur Performanceoptimierung bietet openUTM zudem ein ausgefeiltes Priority Scheduling-Konzept, das sowohl für Dialogverarbeitung als auch für Hintergrundaufträge (siehe Seite 108) genutzt werden kann.

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

openUTM 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. Die Messdaten werden mit dem Programm KDCEVAL ausgewertet und können anschließend auf einen PC übertragen und dort in Diagrammform dargestellt werden.

Der Messmonitor KDCMON ist ausführlich im jeweiligen openUTM-Handbuch „Einsatz von openUTM-Anwendungen“ beschrieben.

2.9 Workload Capture & Replay

Die Simulation von Last-Situationen unter realen Bedingungen ist ein wichtiger Baustein, um Anwendungen zu testen, zu optimieren und die Performance zu steigern.

"Workload Capture & Replay" stellt Funktionen zur Verfügung, mit denen Sie z.B. identische UTM-Anwendungen auf Basis von unterschiedlichen UTM-Versionen vergleichen oder Vergleichsmessungen auf unterschiedlichen Rechner-Hardware-Varianten durchführen können. Sogar der Vergleich von unterschiedlichen System-Plattformen ist per Replay Vorgang möglich. Insbesondere ist auch die Simulation von zukünftig prognostizierter höherer Last einfach per Skalierungs-Parameter beim Replay Vorgang steuerbar.

Workload Capture & Replay steht für Anwendungslast zur Verfügung, die von UPIC-Clients ohne Verschlüsselung erzeugt wurde.

Prozess-Schritte beim Workload Capture & Replay

Workload Capture & Replay besteht aus folgenden Schritten:

1. UPIC Capture

   Während des realen Anwendungslaufs wird die Kommunikation der UTM-Anwendung mit UPIC-Clients mitgeschnitten. Dabei werden Trace-Funktionen von UTM verwendet, der Mitschnitt steht in Form von prozesslokalen Trace-Dateien zur Verfügung.

2. Trace Merging

   Der Mitschnitt aller UTM-Prozesse wird nun entsprechend der genauen zeitlichen Reihenfolge in eine Datei sortiert.

3. UPIC Analyzer

   Der sortierte Mitschnitt wird mit dem Programm **UpicAnalyzer** analysiert. Als Ergebnis wird eine Datei im UPIC Replay Format erzeugt.

4. UPIC Replay

   Die Datei im UPIC Replay Format dient als Input für das auf UPIC basierende Lasttreiber-Programm **UpicReplay**. Beim Ablauf des Lasttests werden die mitgeschnittenen UPIC-Conversations in der zeitlichen Abfolge automatisch wiederholt. Über Skalierungs-Parameter kann die Anzahl der parallel agierenden UPIC-Clients und die Denkzeit pro Client variiert werden, um unterschiedliche Lasten zu simulieren.

   Die Last und damit der Durchsatz kann beim Replay-Schritt auch vielfach höher als die mitgeschnittene Last eingestellt werden. Dieser Replay-Vorgang kann mehrfach reproduzierbar wiederholt werden, um z.B. das Verhalten der UTM-Anwendung bei verschiedenen Lastfaktoren nacheinander zu untersuchen.

Die Programme **UpicAnalyzer** und **UpicReplay** laufen auf 64-Bit-Linux-Systemen..

2.10 Hochverfügbarkeit

Gerade dann, wenn Anwendungen in unternehmenskritischen Bereichen eingesetzt werden, können Ausfallzeiten nicht toleriert werden. openUTM 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.
- Sämtliche Protokolldateien können im laufenden Betrieb umgeschaltet werden, um sie z.B. anschließend auszuwerten und zu sichern oder zu löschen.
- 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 openUTM Umschaltfunktionen bietet - per Administration oder auch automatisch.
- Selbst wenn ein Abbruch der Anwendung unvermeidlich ist, z.B. weil der Server-Rechner auf Grund 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 und auf unterschiedlichen Plattenspeichern geführt werden kann, ist selbst nach physischer Zerstörung eines der Plattenspeicher die Einsatzbereitschaft der Anwendung nicht gefährdet.
- openUTM zusammen mit Hochverfügbarkeitshardware und -software wird auch extremen Anforderungen gerecht (siehe auch Seite 205 für BS2000-Systeme sowie Seite 208 für Windows-Systeme).
- openUTM bietet verbesserte Hochverfügbarkeit durch die Unterstützung von Clustern. Auf einem Cluster kann eine UTM-Anwendung in Form einer UTM-Cluster-Anwendung betrieben werden. Besonders die Möglichkeit, bei laufender UTM-Cluster-Anwendung

die Konfiguration oder das Anwendungsprogramm zu ändern, oder eine neue UTM-Korrekturstufe einzusetzen, gewährleistet die Hochverfügbarkeit der UTM-Cluster-Anwendungen. Diese Funktionalität wird auch als Online-Update bezeichnet. Weitere zentrale Hochverfügbarkeitsfunktionen wie Anwendungsüberwachung und der Online-Import von Anwendungsdaten runden die Funktionalität ab (siehe Kapitel „Hochverfügbarkeit und Lastverteilung mit UTM-Cluster-Anwendungen“ auf Seite 209).

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

2.11 Security-Funktionen

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

- Definition logischer Anschlusspunkte:
  Eine UTM-Anwendung kann so konfiguriert werden, dass sich ein Client nur dann an die UTM-Anwendung anschließen kann, wenn für ihn ein logischer Anschlusspunkt (LTERM-Partner) in dieser Anwendung konfiguriert ist. Der Client muss in diesem Fall 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 spezifische Passwörter zuordnen. Zudem kann man die Komplexität und die Gültigkeitsdauer der Passwörter vorgeben und es kann eine Historie der letzten Passwörter geführt werden.
- Einsatz eines Ausweislesers zur Zugangskontrolle
- Stiller Alarm bei wiederholten erfolglosen Anmeldeversuchen eines Benutzers
- Automatischer Verbindungsabbau bei Zeitüberschreitung
- Ereignisgesteuerte Routinen für selbstdefinierte Zugangskontrollen
- Differenzierte Zugriffsberechtigungen (Autorisierung):
  Hierzu bietet openUTM zwei Konzepte mit unterschiedlicher Blickrichtung:
  - Das Lock-/Keycode-Konzept, das benutzerorientiert ist.

  - Das Access-List-Konzept, das rollenorientiert ist.

  Beide Konzepte lassen sich innerhalb einer Anwendung parallel verwenden.

- Verschlüsselung von Passwörtern sowie Verschlüsselung der Nachrichten auf dem Weg zwischen Client und Server. Dazu wird die Komponente openUTM-Crypt benötigt.
- Anwendungs-übergreifendes Benutzerkonzept bei Nutzung von OSI TP
- Unterstützung von Kerberos auf BS2000-Systemen (für den Terminalbetrieb)

Detaillierte Informationen zum Thema Security finden Sie im Kapitel „Security-Funktionen“ auf Seite 183.

Die umfangreichen Zugangs- und Zugriffsschutzkonzepte, die openUTM bietet, stehen auch beim Anschluss von openUTM-Client-Programmen zur Verfügung sowie bei verteilter Verarbeitung mit anderen Partner-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 ACCESS-LIST, LTERM, KEYSET, KEYS und LOCK in den openUTM-Handbüchern „Anwendungen generieren“ und „Anwendungen administrieren“.

2.12 Dynamisierung

Von OLTP-Anwendungen wird eine dynamische Anpassung der Konfigurationsdaten gefordert, um z.B. neue Benutzer oder Services in eine Anwendung aufnehmen zu können, ohne dass der Anwendungslauf unterbrochen werden muss.

In einer UTM-Anwendung werden die Konfigurationsdaten wie z.B. Informationen über Benutzerkennungen, Kommunikationspartner, Puffergrößen etc. zunächst statisch bei der Generierung der Anwendung festgelegt. Für die Dynamisierung bietet openUTM mächtige Tools und Schnittstellen an, mit denen die Konfiguration während des Anwendungslaufs geändert werden kann.

Das grafische Administrationsprogramm WinAdmin bietet eine vollständige, Java-basierte Lösung für die Dynamisierung. Mit WinAdmin können Sie von jeder Java-Plattform aus dynamisch die generierten Anwendungsparameter ändern oder Generierungsobjekte neu aufnehmen oder löschen (z.B. Benutzerkennungen oder Verbindungen zu Partnern). Weitere Informationen zu WinAdmin finden Sie auf Seite 64 und Seite 173.

Die Komponente WebAdmin stellt eine web-basierte Benutzeroberfläche zum Administrieren von UTM-Anwendungen auf allen Plattformen zur Verfügung. WebAdmin bietet einen zu WinAdmin vergleichbaren Funktionsumfang. Weitere Informationen zu WebAdmin finden Sie auf Seite 66 und Seite 176.

Mit der Programmschnittstelle KDCADMI können Sie eigene, auf Ihren Bedarf zugeschnittene Programme zur dynamischen Administration erstellen, um z.B. bestimmte Änderungen zeitgesteuert und automatisch ablaufen zu lassen. Näheres finden Sie auf Seite 170.

Das UTM-Tool KDCUPD erlaubt es, nach einer Neugenerierung Ihrer UTM-Anwendung (stand-alone oder im Cluster) Benutzerdaten und Verwaltungsinformationen der Anwendung in die neue Konfiguration zu übernehmen. KDCUPD wird dann eingesetzt, wenn grundlegende Modifikationen an der Generierung vorgenommen werden müssen oder wenn auf eine andere UTM-Version gewechselt wird. Beim Einsatz von KDCUPD gilt:

- Der Anwendungslauf einer stand-alone-Anwendung wird kurz unterbrochen, danach können alle Benutzer wieder ihre Arbeit an der Stelle aufnehmen, an der sie vorher unterbrochen wurden.
- Eine UTM-Cluster-Anwendung kann bei den meisten Änderungen ohne Unterbrechung weiterlaufen, da sukzessive nur einzelne Knoten-Anwendungen unterbrochen werden müssen.

Weitere Informationen finden Sie auf Seite 161.

2.13 Internationalisierung / Anpassung von Meldungen

openUTM stellt umfassende Funktionen zur Internationalisierung zur Verfügung. Diese ermöglichen es, Sprach-übergreifende 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.

openUTM für Unix-Systeme erfüllt die Richtlinien zur Internationalisierung, wie sie in den X/Open Portability Guides spezifiziert sind.

Näheres zur Internationalisierung auf BS2000-Systemen finden Sie in Abschnitt „Internationalisierung / XHCS-Unterstützung“ auf Seite 243.

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

2.14 openUTM in der Unicode-Umgebung

Eine UTM-Anwendung kann problemlos auf eine Unicode-Umgebung umgestellt werden und lässt sich in einer Unicode-Umgebung betreiben. openUTM kann z.B. mit Datenbanken zusammenarbeiten, deren Datenbestand im Unicode-Format vorliegt, und Unicode-Daten mit anderen Anwendungen austauschen. openUTM selbst verhält sich gegenüber Unicode-Daten transparent.

Wenn Sie Anwendungsprogramme für die Verarbeitung von Unicode-Daten umstellen, dann muss der zugehörige Compiler Unicode-fähig sein.

Die UTM-Verwaltungsdaten wie z.B. Benutzerkennungen, Transaktionscodes etc. werden auch in einer Unicode-Umgebung im 7-Bit Code definiert.

2.15 Accounting

openUTM 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 an Stelle der UTM-Benutzerkennung die Session (LU6.1) bzw. die Association (OSI TP ohne Anwendungs-übergreifendes Benutzerkonzept), so dass eine Verrechnung auch unter diesen Umständen möglich ist.

Dabei kann entweder per Festpreis oder verbrauchsabhängig abgerechnet werden.

Festpreis

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

Verbrauchsabrechnung

Bei der Verbrauchsabrechnung ermittelt openUTM 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 openUTM auf den verschiedenen Plattformen ist ausführlich im jeweiligen openUTM-Handbuch „Einsatz von openUTM-Anwendungen“ beschrieben.

2.16 Leistungskontrolle mit dem Software Monitor openSM2

Der **Software Monitor openSM2** liefert statistische Daten über die Leistungen und die Auslastung der Betriebsmittel. Mit openSM2 können Sie im laufenden Betrieb das Verhalten einer UTM-Anwendung überwachen und Leistungsengpässe aufdecken. Sie können sich die Daten, die openUTM liefert, auf einopenSM2-Bildschirm online anzeigen lassen. Zusätzlich speichert openSM2 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 openSM2 ist in einem eigenen Handbuch mit dem Titel „openSM2 Software Monitor “ dokumentiert. Speziell auf das Zusammenspiel zwischen openSM2 und openUTM wird im openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter BS2000-Systemen“ bzw. im openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter Unix- und Windows-Systemen“ eingegangen.

2.17 Diagnosemöglichkeiten in openUTM

openUTM bietet Ihnen bei der Diagnose und der Lokalisierung von Fehlern in der Anwendung umfassende Unterstützung.

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

Die Dump-Datei enthält alle KDCS- und Administrationsaufrufe sowie Datenbankaufrufe durch openUTM. In BS2000-Systemen sind zusätzlich die Aufrufe an das Formatierungssystem und, bei Einsatz von IUTMDB, auch Datenbankaufrufe durch das Programm enthalten. Damit lassen sich z.B. Fehler bei der Zusammenarbeit mit Datenbanken leicht lokalisieren.

Für die Auswertung der Dump-Datei stellt Ihnen openUTM ein spezielles Tool (KDCDUMP) zur Verfügung, das die Dump-Datei in abdruckbarer Form aufbereitet. Dieses Tool bietet Ihnen außerdem die Möglichkeit, die Dump-Datei gezielt am Terminal auszuwerten 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.

Als weitere Diagnosehilfe bietet Ihnen openUTM 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 über die Programmschnittstelle zur Administration einschalten.

Wichtige Informationen für die Diagnose liefert auch die anwendungsspezifische System-Protokolldatei **SYSLOG** (SYStem LOGging), die openUTM beim Start einer Anwendung anlegt. In der SYSLOG-Datei protokolliert openUTM Meldungen, die für die laufende Überwachung oder für spätere Auswertungen 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 jeweiligen openUTM-Handbuch „Meldungen, Test und Diagnose“. Dort finden Sie auch Informationen zum Dump-Auswertungstool KDCDUMP und zu den Tools zur Auswertung der Systemprotokoll-Datei SYSLOG.

2.18 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-Systemen 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 openUTM die unterschiedlichsten Modelle unterstützt (Conversations, Pseudo-Conversations, Request/Reply, Message Queuing).

Die Programmierung wird wesentlich auch dadurch vereinfacht, dass viele zentrale Aufgaben von openUTM selbsttätig erledigt werden.

openUTM 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.

Für COBOL-Anwender auf offenen Plattformen sind NetCOBOL von Fujitsu und Visual-Cobol von Micro Focus die geeigneten Entwicklungsumgebungen.

C- und C++-Anwender werden durch Oracle Solaris Studio von Oracle Corporation und das Microsoft Visual Studio wirkungsvoll unterstützt.

BS2IDE - Integrierte Entwicklungsumgebung für BS2000-Systeme auf Basis von Eclipse

Für die Erstellung von UTM-Anwendungen auf BS2000-Systemen können Sie auch BS2IDE verwenden. Die BS2IDE wird als Plug-In zur offenen Entwicklungsumgebung Eclipse angeboten. Dieses Plug-In unterstützt den Entwickler bei typischen Aufgaben und integriert dabei die Vorzüge moderner Entwicklungsumgebungen (engl. IDE - Integrated Development Environment).

Die BS2IDE vereint die wichtigsten Tools des Software-Entwicklungsprozesses in einer Oberfläche und unterstützt den Entwickler bei der Fehlersuche.

2.19 Grafische Administration mit WinAdmin

Mit der openUTM-Komponente WinAdmin können Sie von Windows- oder Unix-Systemen aus über eine komfortable grafische Oberfläche UTM-Anwendungen auf allen Plattformen administrieren:

- Dabei können Sie sowohl einzelne oder mehrere UTM-Anwendungen als auch einzelne oder mehrere UTM-Cluster-Anwendungen administrieren und dynamisch konfigurieren, d.h. Objekte neu aufnehmen oder löschen, denn WinAdmin unterstützt den vollen Umfang der Administrations-Programmschnittstelle KDCADMI.
- Neben der Administrations-Programmschnittstelle bietet WinAdmin auch die klassische Kommandoschnittstelle an. Näheres finden Sie auf Seite 173.
- Zusätzlich kann WinAdmin Statistiken erzeugen und als Diagramme darstellen.
- Bei der Administration von UTM-Cluster-Anwendungen stellt Ihnen WinAdmin sowohl Administrationsfunktionen zur Verfügung, die nur auf eine Knoten-Anwendung wirken, als auch Cluster-globale Administrationsfunktionen, die auf alle Knoten-Anwendungen der UTM-Cluster-Anwendung wirken. Näheres finden Sie auf Seite 173.

Die UTM-Anwendungen können dabei beliebig im Netz auf unterschiedliche Plattformen verteilt sein. 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.

openUTM WinAdmin wird als UPIC-Client an die UTM-Anwendung angebunden und läuft auf allen gängigen Unix-, Linux- und Windows-Systemen.

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. Die Zugriffsrechte lassen sich auch gezielt auf einzelne Anwendungen oder Objekttypen beschränken.
- Für jeden Benutzer kann der Zugang zu WinAdmin durch ein Passwort geschützt werden.

Über weitere Leistungsmerkmale von openUTM WinAdmin und über die Anwendung dieses Produkts informiert Sie die Dokumentation zu WinAdmin (Online-Hilfe und WinAdmin-Beschreibung, die online als PDF-Datei zur Verfügung steht).

2.20 Grafische Administration mit WebAdmin

Mit der Komponente WebAdmin bietet Ihnen openUTM eine web-basierte Benutzeroberfläche zum Administrieren von UTM-Anwendungen auf allen Plattformen. WebAdmin läuft auf einem Web-Server und kann von einem beliebigen Client über den Browser aufgerufen werden:

- Dabei können Sie UTM-Anwendungen und UTM-Cluster-Anwendungen administrieren und dynamisch konfigurieren, d.h. Objekte neu aufnehmen oder löschen, denn WebAdmin unterstützt den vollen Umfang der Administrations-Programmschnittstelle KDCADMI.
- Zusätzlich kann WebAdmin Statistiken erzeugen und tabellarisch darstellen.
- Bei der Administration von UTM-Cluster-Anwendungen stellt Ihnen WebAdmin sowohl Administrationsfunktionen zur Verfügung, die nur auf eine Knoten-Anwendung wirken, als auch Cluster-globale Administrationsfunktionen, die auf alle Knoten-Anwendungen der UTM-Cluster-Anwendung wirken. Näheres finden Sie auf Seite 176.

Die UTM-Anwendungen können dabei beliebig im Netz auf unterschiedliche Plattformen verteilt sein. Die zu administrierenden UTM-Anwendungen lassen sich zu Kollektionen gruppieren.

openUTM WebAdmin wird als UPIC-Client an die UTM-Anwendung angebunden und läuft auf allen gängigen Unix-, Linux- und Windows-Systemen sowie als Add-on auf der Management Unit eines SE Servers.

Security bei WebAdmin

Für die Administration über WebAdmin 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 WebAdmin 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 WebAdmin.
- Für jeden Benutzer kann der Zugang zu WebAdmin durch ein Passwort geschützt werden.

Über weitere Leistungsmerkmale von openUTM WebAdmin und über die Anwendung dieses Produkts informiert Sie die Dokumentation zu WebAdmin (Online-Hilfe und WebAdmin-Beschreibung, die online als PDF-Datei zur Verfügung steht).

2.21 SNMP-Subagent für openUTM

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 openUTM bindet den Transaktionsmonitor openUTM in einen System-übergreifenden Leitstand ein und

- ermöglicht für ausgewählte UTM-Anwendungen einen umfassenden Überblick über alle mit ihnen 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 Administrationsfunktionen 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, z.B. an das CA Unicenter. Dieses Produkt bietet alle Möglichkeiten für die Integration von beliebigen Systemen mit privaten MIBs (Management Informations Bases).

3 Integrationsszenarien mit openUTM

openUTM wird immer häufiger zusammen mit anderen Anwendungen eingesetzt, besonders im Zusammenspiel mit Java Enterprise Edition Technologie. Dabei spielt die so genannte Service Oriented Architecture (SOA) eine wichtige Rolle. openUTM lässt sich leicht in diese Architekturen integrieren, wobei openUTM sowohl als gerufener Service agieren als auch die Rolle des rufenden „Client“ übernehmen kann.

openUTM als Teil von openSEAS

openSEAS ist eine Suite aufeinander abgestimmter Produkte, die den besonderen Anforderungen entsprechen, die aktuell an IT-Systeme gestellt werden, wie z.B. Web-Anbindung oder die Integration in andere Systeme. openUTM ist ein grundlegender Bestandteil des openSEAS-Konzeptes.

3.1 Unterschiedliche Anwendungen integrieren

Die Gestaltung von durchgängigen Geschäftsprozessen aus mehreren Anwendungen wird heute mehr und mehr zur zentralen Aufgabe von IT-Abteilungen. Hier sind zwei unterschiedliche Szenarien zu betrachten:

– entweder repräsentiert die UTM-Anwendung die Logik des Geschäftsprozesses (oder eines Teilprozesses) und aus der UTM-Anwendung heraus soll eine andere Anwendung als Service aufgerufen werden. In diesem Fall kann die UTM-Anwendung z.B. auf eine Java Enterprise Edition Anwendung über BeanConnect zugreifen.

– oder die Teilprogramme der UTM-Anwendung stellen Services für die Geschäftsprozess-Anwendung zur Verfügung.

Diese Möglichkeiten werden in den folgenden Kapiteln genauer dargestellt.

3.2 openUTM in Java Enterprise Umgebung integrieren

Die Java Enterprise Edition Technologie (Java EE), die von Oracle definiert worden ist, gewinnt stetig wachsende Bedeutung für die Erstellung von Server-Anwendungen. Das liegt zum Einen an der Durchgängigkeit der Programmiersprache Java und zum Anderen an der Komponententechnologie. Diese erlaubt es, Programmteile wie Bausteine zusammenzufügen, ganz gleich, ob es sich um selbst entwickelte oder um Standardbausteine vom Markt handelt. Ein Beispiel hierfür sind E-Commerce-Lösungen, die teilweise aus eigenen Services und teilweise aus allgemeinen Services bestehen (z.B. die Warenkorb-Verwaltung).

openUTM kann besonders gut in der Umgebung solcher Java EE Application Server eingesetzt werden, denn mit dem Produkt **BeanConnect** lassen sich Java EE Lösungen optimal mit Lösungen auf Basis openUTM integrieren.

BeanConnect ist ein Adapter gemäß der Java Connector Architecture (JCA) von Oracle und bietet den standardisierten Anschluss von UTM-Anwendungen an Java EE Application Server, insbesondere an Oracle's Application Server. Andere Applikations-Server wie z.B. WebSphere von IBM lassen sich jedoch ebenso einsetzen, nur muss dann wegen der Unterschiede bei den Applikations-Servern ein eigenes Servicepaket erworben werden.

3.2.1 openUTM als Server für Java EE Application Server

openUTM kann bei einer Kopplung als Server fungieren, wobei eine Anwendung (Enterprise Java Bean, EJB) im Java EE Application Server die Kommunikation durch Senden einer Nachricht an die UTM-Anwendung startet. Aus Sicht des Java EE Servers ist das eine Outbound-Kommunikation. Dabei ist die Kopplung auf zwei Arten möglich:

- Als reine Client-Anbindung über das UPIC-Protokoll
- Als Server-Server-Kopplung über OSI TP

Diese beiden Kopplungen werden über Komponenten des Produkts BeanConnect realisiert, siehe folgendes Bild:

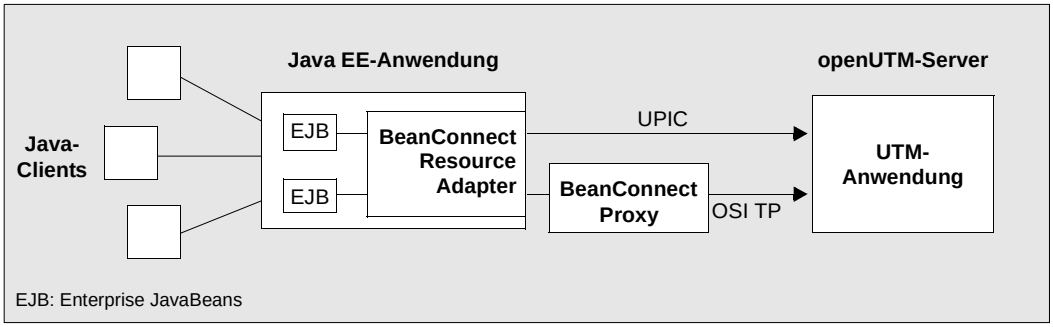


Bild 8: openUTM als Server eines Java EE Application Servers

Kopplung über UPIC

Diese Kopplungsart entspricht aus UTM-Sicht einer klassischen Anbindung eines Client über das UPIC-Protokoll, siehe Abschnitt „Clients mit Trägersystem UPIC“ auf Seite 82. Dadurch kann die Java EE Anwendung jeden UTM-Dialog-TAC aufrufen, jedoch keine Asynchron-TACs.

Durch das UPIC-Protokoll ist die Java EE Anwendung in das Wiederanlaufkonzept von openUTM eingebunden, d.h. die Java EE Seite kann z.B. den Status der letzten Transaktion anfordern. Die Initiative muss dabei immer von der Java EE Anwendung ausgehen, eine globale Transaktionssicherung ist nicht möglich.

Diese Kopplungsart eignet sich für einfache Integrationsszenarien wie z.B. die bloße Anforderung von Information oder Einschritt-Transaktionen, die keine verteilte Transaktionen erfordern. Für solche Szenarien ist dieser einfache „Einstiegs“- Adapter eine attraktive Lösung.

Kopplung über OSI TP

Diese Kopplungsart entspricht aus UTM-Sicht einer Server-Server-Kopplung, siehe Seite 87. Dabei wird die Komponente BeanConnect Proxy dazwischengeschaltet. Die Kopplung über OSI TP bietet im Vergleich zur Anbindung über UPIC zusätzliche Möglichkeiten, mit denen sich auch anspruchsvolle Integrationsszenarien umsetzen lassen:

– Die Anwendungen können wahlweise mit oder ohne verteilte Transaktionssicherung arbeiten

– Der Java EE Application Server kann in der UTM-Anwendung sowohl Dialog- als auch Asynchron-TACs aufrufen

Um eine Kopplung herzustellen, müssen Sie im Falle einer OSI TP Kopplung die Generierung der UTM-Anwendung anpassen den BeanConnect Proxy entsprechend konfigurieren.

3.2.2 openUTM als Client von Java EE Application Servern

openUTM kann als Client eines Java EE Application Servers fungieren, indem die UTM-Anwendung die Kommunikation durch Senden einer Nachricht an Java EE Application Server startet. Aus Sicht des Java EE Servers ist das eine Inbound-Kommunikation. Für die Kommunikation kann das Protokoll OSI TP oder ein Transportsystem-Protokoll verwendet werden, siehe folgendes Bild:

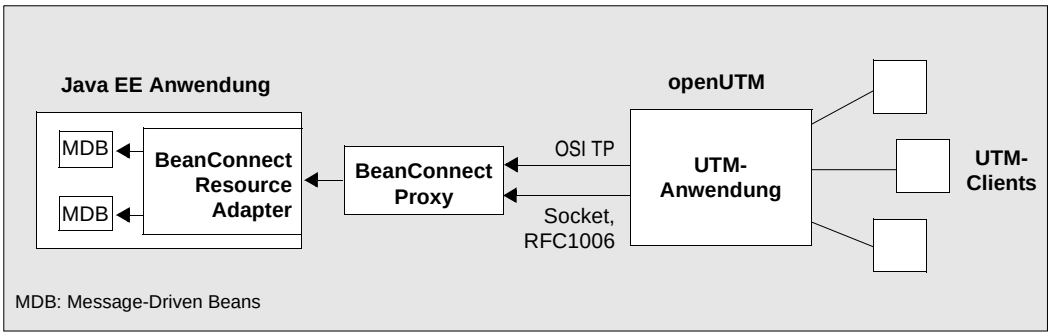


Bild 9: openUTM als Client eines Java EE Application Servers

Diese Kopplung zwischen openUTM und der Java EE Anwendung läuft über die Komponente BeanConnect Proxy. Die von der UTM-Anwendung gesendeten Nachrichten sind an Message Endpoint Anwendungen (MessageDriven Beans) im Java EE Application Server gerichtet.

Die Kommunikation kann über OSI TP oder Transportsystem-Protokoll stattfinden:

– Wenn das Protokoll OSI TP verwendet wird, dann kann wahlweise mit oder ohne Transaktionssicherung gearbeitet werden. Die Kommunikation ist sowohl im Dialog als auch asynchron möglich.

– Wenn ein Transportsystem-Protokoll wie Socket oder RFC1006 verwendet wird, dann ist die Kommunikation immer nicht-transaktional und asynchron.

Um eine Kopplung herzustellen, müssen Sie die Generierung der UTM-Anwendung anpassen und den BeanConnect Proxy entsprechend konfigurieren.

Für die Konfigurierung des BeanConnect Proxy verwenden Sie die Management Console von BeanConnect. Weitere Einzelheiten finden Sie im Handbuch „BeanConnect“.

3.2.3 UTM-Cluster-Anwendung als Client oder Server

Eine UTM-Cluster-Anwendung kann bei einer Kopplung als Server für Outbound-Kommunikation und als Client für Inbound-Kommunikation fungieren, wobei eine Anwendung (Enterprise Java Bean, EJB) im Java EE Application Server die Kommunikation durch Senden einer Nachricht an die UTM-Cluster-Anwendung startet. Aus Sicht des Java EE-Servers ist das eine Outbound-Kommunikation. Bei Outbound-Kommunikation ist es möglich, die einzelnen Knoten der UTM-Cluster-Anwendung im Rundlauf-Verfahren (Round Robin) anzusprechen. Die Kopplung kann sowohl als Server-Server-Kopplung über OSI TP (siehe auch Abschnitt „Kopplung über OSI TP“ auf Seite 71) als auch als Client-Anbindung über das UPIC-Protokoll realisiert werden.

Diese Kopplung wird über Komponenten des Produkts BeanConnect realisiert, siehe folgendes Bild:

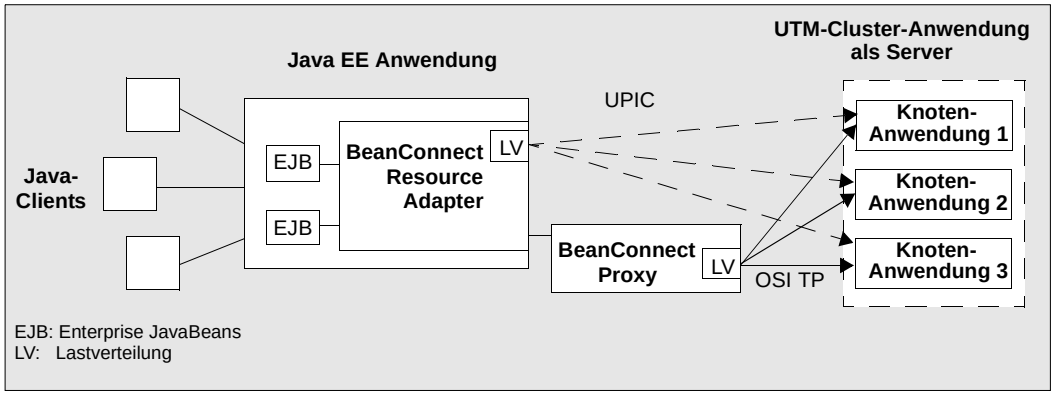


Bild 10: UTM-Cluster-Anwendung als Server eines Java EE Application Servers

Die Lastverteilung (LV) wird bei der Kopplung über OSI TP im BeanConnect Proxy durchgeführt, dabei wird das Round-Robin-Verfahren verwendet.

Bei der UPIC-Anbindung wählt der Lastverteiler im BeanConnect Resource Adapter die Verbindungen zufallsgesteuert aus.

3.3 openUTM über Web Services ansprechen

Das Consulting Projektpaket WebServices for openUTM (WS4UTM) ermöglicht es, auf komfortable Weise eine UTM-Anwendung als Web Service verfügbar zu machen.

Web Services sind Anwendungen, auf die plattformunabhängig über Web-Server zugegriffen wird, und die über eine standardisierte Schnittstelle (öffentlich) verfügbar sind. Durch den Einsatz der Web Service-Technologie steht einem Anwender von openUTM eine einfache, unabhängige, XML-basierte, standardisierte Schnittstelle für den Zugriff aus anderen Systemen zur Verfügung. Mit Hilfe von WS4UTM ist es möglich, Einschritt-Dialog-Programme einer UTM-Anwendung als Web Service (über HTTP/SOAP) verfügbar zu machen.

Als Web Service Server wird der Axis-Server von Apache verwendet. Voraussetzung für den Einsatz von WS4UTM ist die Installation von Tomcat und Axis(Java).

WS4UTM muss auf demselben Rechner installiert werden wie die Tomcat/Axis-Instanz, über die die WS4UTM-Kommunikation erfolgt.

WS4UTM ist auf den gängigen Unix- und Windows-Plattformen ablauffähig und besteht aus den Komponenten WS4UTMDeploy und WS4UTMAxis:

– **WS4UTMDeploy** ist ein grafisches Tool zum Konfigurieren und Deployen von Web Services.

– **WS4UTMAxis** ist ein Paket von Klassen, die von Axis geladen werden und die Funktion eines Providers erfüllen. WS4UTMAxis dient u.a. dazu, den Service bei der entsprechenden UTM-Anwendung über die JConnect Schnittstelle aufzurufen. JConnect verwendet das UPIC-Protokoll.

Die folgende Abbildung zeigt die Architektur von WS4UTM zur Laufzeit:

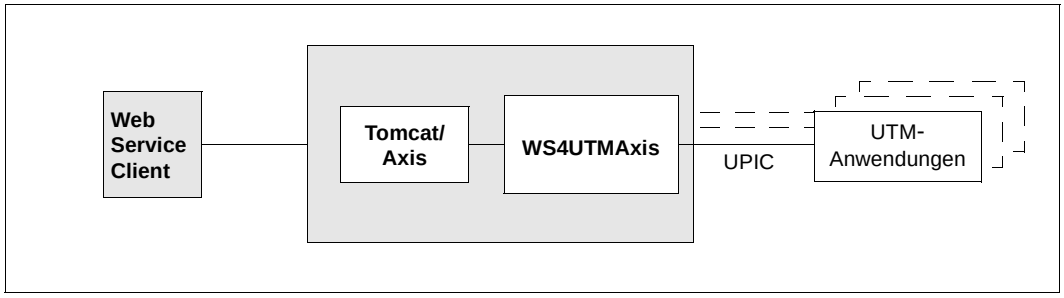


Bild 11: UTM Services mit WS4UTM als Web Services aufrufen

Einen Gesamtüberblick über WS4UTM finden Sie im zugehörigen Handbuch.

3.4 Bestehende Anwendungen in das Web stellen

Bestehende UTM-Anwendungen können unverändert mit Hilfe des Produkts WebTransactions an das World Wide Web angeschlossen werden. Der Akzent liegt hier auf „unverändert“, denn die gesamte Server-Anwendung bleibt wie sie ist, aber die Präsentation im Web kann in großer Bandbreite gestaltet werden. Das Web-Hosting kann in BS2000-Systemen selbst oder auf einem eigenständigen Web-Server liegen.

Mit diesen Lösungen kann der Internet/Intranet-Nutzer auf die Services des Anwendungsservers openUTM zugreifen.

WebTransactions for openUTM realisiert die Web-Anbindung von UTM-Anwendungen, die für FHS oder FORMANT entwickelt wurden oder über die Client-Server-Schnittstelle UPIC arbeiten. WebTransactions for openUTM steht auf den BS2000-, Unix- und Windows-Plattformen zur Verfügung.

Unter Unix- und Windows-Systemen kann mit beliebigen Web-Servern zusammengearbeitet werden. Unter BS2000-Systemen ist für WebTransactions der Web-Server von Apache erforderlich.

Auch die UTM-Host-Anwendung kann auf jeder dieser Plattformen ablaufen. Der Host-Adapter für die Kommunikation mit der Host-Anwendung basiert auf openUTM-Client (UPIC), d.h., WebTransactions und Host-Anwendung können jeweils auf unterschiedlichen Plattformen ablaufen.

Mit WebTransactions for openUTM wird das BS2000-Programm `IFG2FLD` ausgeliefert, mit dem die Beschreibungen der Host-Formate aus der IFG-Bibliothek in Formatbeschreibungsquellen umgesetzt werden. Aus den Formatbeschreibungsquellen können mit Web*Lab* automatisch Templates generiert werden. Diese generierten Templates bilden die Basis für die individuelle Gestaltung einzelner Formate.

Zusätzlich bietet WebTransactions weitere Integrationsmöglichkeiten:

- Gestaltung der Oberfläche (GUIfication)
- Gestaltung der Dialogabläufe (Interface Reengineering)
- Anwendungsintegration (Business Process Reengineering)

Parallel zum Zugriff über WebTransactions kann weiterhin auch über „gewöhnliche“ Terminals oder Clients auf die Host-Anwendungen oder dynamische Web-Inhalte zugegriffen werden. So können Sie eine Host-Anwendung schrittweise ans Web anschließen und die Wünsche und Bedürfnisse unterschiedlicher Nutzergruppen berücksichtigen.

Einen Gesamtüberblick über WebTransactions finden Sie im Handbuch „WebTransactions - Konzepte und Funktionen“.

Details zum Anschluss der verschiedenen Host-Anwendungen sind in eigenen WebTransactions-Handbüchern beschrieben. Alle WebTransactions-Handbücher sind online im PDF-Format verfügbar.

4 Verteilte Verarbeitung mit openUTM

In diesem Kapitel wird auf die unterschiedlichen Formen der Verteilten Verarbeitung mit openUTM eingegangen. Dabei wird die Client/Server-Kommunikation, die Server/Server-Kommunikation (mit und ohne Transaktionssicherung) und die Kommunikation mit Transportsystem-Anwendungen behandelt.

4.1 Client/Server-Architekturvarianten

Allen Client/Server-Architekturen liegt eine Gliederung in einzelne Software-Komponenten, auch „Schichten“ oder „Tiers“ genannt, zu Grunde (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 12 auf der folgenden Seite sind die fünf Client/Server-Basismodelle mit jeweils zwei physischen Software-Tiers dargestellt.

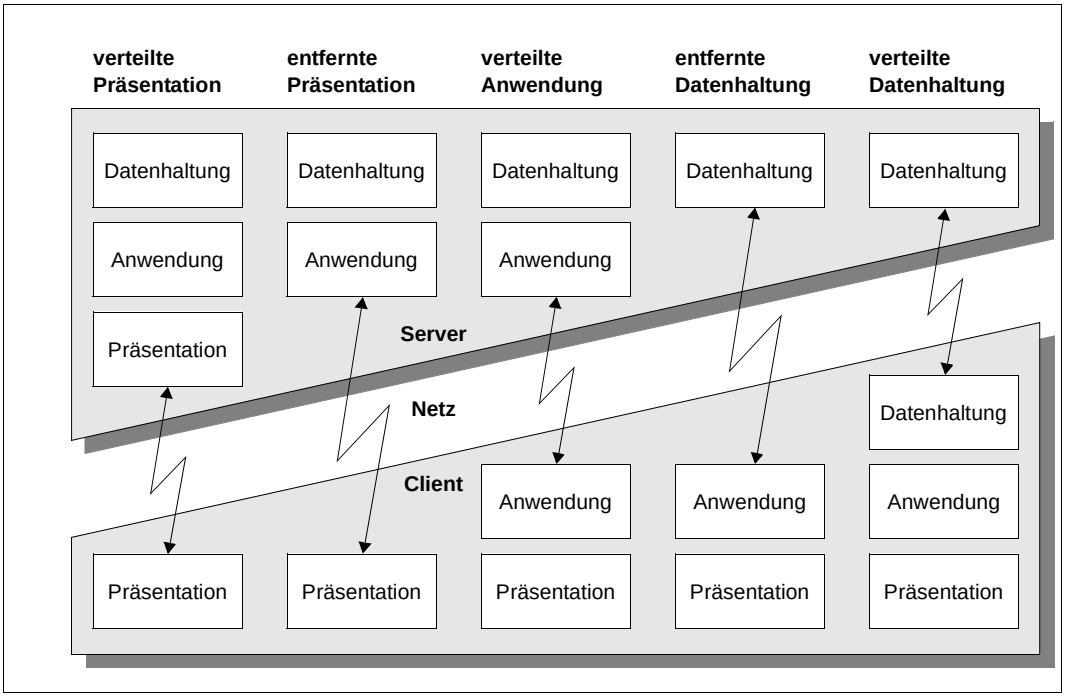


Bild 12: Basismodelle für Client/Server-Architekturen

openUTM unterstützt nicht nur die in Bild 12 dargestellten Grundschemata, sondern vielfältige Abwandlungen und Kombinationen. Mit openUTM sind komplexe Multi-Tier-Szenarien möglich - sowohl auf der Ebene physischer als auch auf der Ebene logischer Tiers.

Wie Bild 13 (auf der folgenden Seite) zeigt, erlaubt openUTM 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. openUTM gewährleistet dabei, dass die Daten zu jedem Zeitpunkt global konsistent sind.

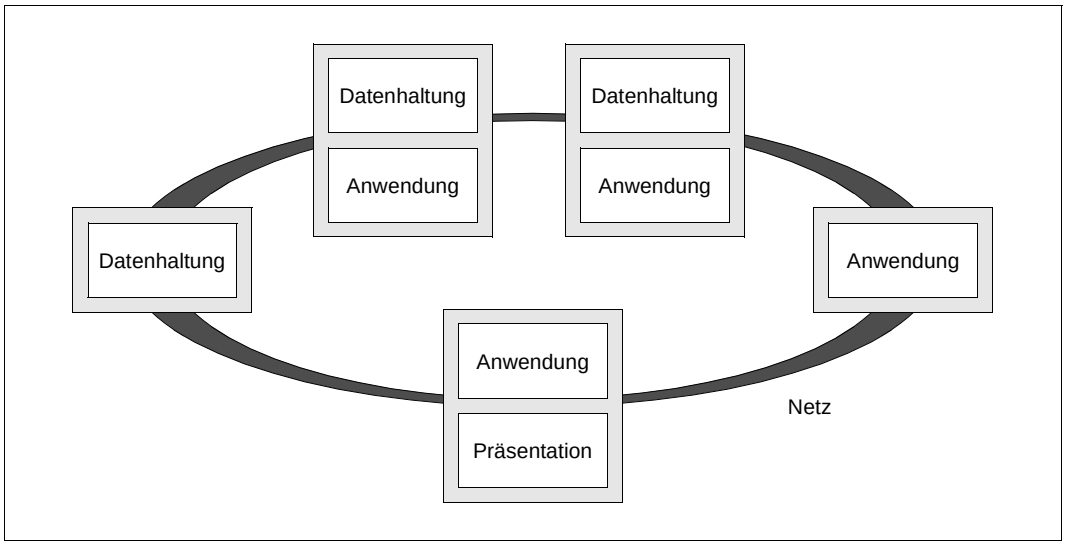


Bild 13: Multi-Tier-Architektur

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 openUTM 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.

4.2 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.

Clients einer UTM-Anwendung können sein:

– Terminals oder Terminalemulationen
– UPIC-Clients (siehe Seite 82)
– OpenCPIC-Clients (siehe Seite 83)
– Transportsystem-Anwendungen (siehe Seite 98)

UPIC-Clients und OpenCPIC-Clients werrden auch als openUTM-Clients bezeichnet. 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 openUTM 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 auszudrücken, dass sie für Server-Anwendungen besonders geeignet sind.

4.2.1 Kommunikation mit openUTM-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 openUTM-Client ein eigenes Produkt zur Verfügung. Mit openUTM-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 openUTM gelieferten Statistikwerte 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 openUTM stehen Ihnen auch beim Anschluss von Client-Anwendungen zur Verfügung.

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

openUTM-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.

4.2.1.1 Clients mit Trägersystem UPIC

UPIC ist ein schlankes, sehr performantes und einfach einsetzbares Trägersystem, das auf openUTM 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:

- UPIC unterstützt z.B. die Übertragung von Formatnamen und Funktionstasten.
- UPIC unterstützt die Verschlüsselung von Zugangs- und Benutzerdaten.
- UPIC kann den Event-Service SIGNON nutzen, näheres siehe Seite 189.
- Ein UPIC-Client kann nach einem Störfall den Status der letzten Transaktion anfordern und ist dadurch in das Wiederanlaufkonzept von openUTM mit einbezogen.

- Ein UPIC-Client kann innerhalb eines Programmlaufs mehrere Conversations gleichzeitig unterhalten („Multi-Conversations“), sofern das entsprechende System „Multi-Threading“ unterstützt.
- UPIC-Clients können die „Multi-Signon“-Funktion nutzen, d.h. es können sich gleichzeitig mehrere UPIC-Clients unter derselben UTM-Benutzerkennung anmelden, wenn für diese Benutzerkennung auf den Vorgangswiederanlauf verzichtet wird.
- Ein UPIC-Client kann unter demselben Anwendungsnamen mehrere parallele Transportverbindungen zu einer UTM-Anwendung aufbauen („Multi-Connect“).
- UPIC bietet für die Kommunikation mit einer UTM-Cluster-Anwendung eine einfache Lastverteilungs-Funktionalität. Der UPIC-Client kommuniziert mit einer der zugehörigen Knoten-Anwendungen, die zufällig ausgewählt wurde. Näheres siehe Seite 209.
- Für UPIC-Clients können mit Workload Capture & Replay Last-Situationen simuliert werden, siehe Abschnitt „Workload Capture & Replay“ auf Seite 54.

UPIC gibt es auf allen gängigen Unix- und Windows-Plattformen oder unter BS2000-Systemen.

4.2.1.2 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.

Bei Kommunikation mit OpenCPIC-Clients kann ausgewählt werden, ob mit oder ohne globale Transaktionen gearbeitet wird:

- Bei globaler Transaktionssicherung kann ein OpenCPIC-Client Beginn und Ende der 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.
- Beim Arbeiten ohne globale Transaktionen wird der OpenCPIC-Client in das Wiederanlaufkonzept von openUTM mit einbezogen.

OpenCPIC-Clients können die „Multi-Signon“-Funktion nutzen, d.h. es können sich gleichzeitig mehrere OpenCPIC-Clients unter derselben UTM-Benutzerkennung anmelden, wenn für diese Benutzerkennung auf den Vorgangswiederanlauf verzichtet wird oder wenn mit globaler Transaktionssicherung gearbeitet wird.

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

Wie beim Anschluss von Client-Anwendungen die Security-Funktionen von openUTM genutzt werden können, ist im Kapitel „Security-Funktionen“ auf Seite 183 beschrieben. Nähere Informationen zum Wiederanlauf-Verhalten finden Sie im Kapitel „Fehlertoleranz und Wiederanlauf“ auf Seite 217. Zum Produkt openUTM-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.

4.2.2 Java-Clients

Das Produkt BeanConnect stellt mit der Komponente JConnect Java-Klassen bereit, mit denen Sie in Java geschriebene Clients erstellen können. Diese Clients können Sie als UPIC-Clients an UTM-Anwendungen anbinden, siehe auch Abschnitt „openUTM in Java Enterprise Umgebung integrieren“ auf Seite 70.

Dabei gibt es die drei folgenden Anschlussmöglichkeiten:

– Über Servlets, die auf dem Web-Server laufen
– In Browser-Umgebung über Applets
– Direktanschluss

Die folgenden Bilder veranschaulichen diese drei Möglichkeiten:

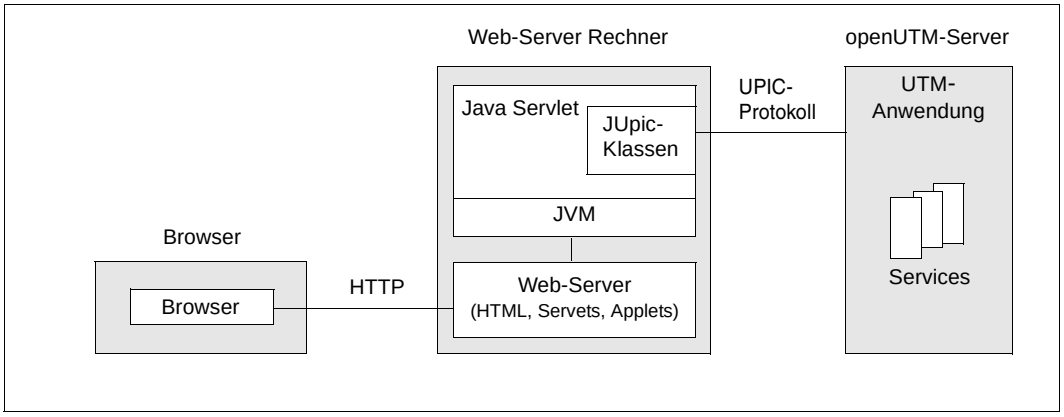


Bild 14: Anschluss über Servlets, die auf dem Web-Server laufen

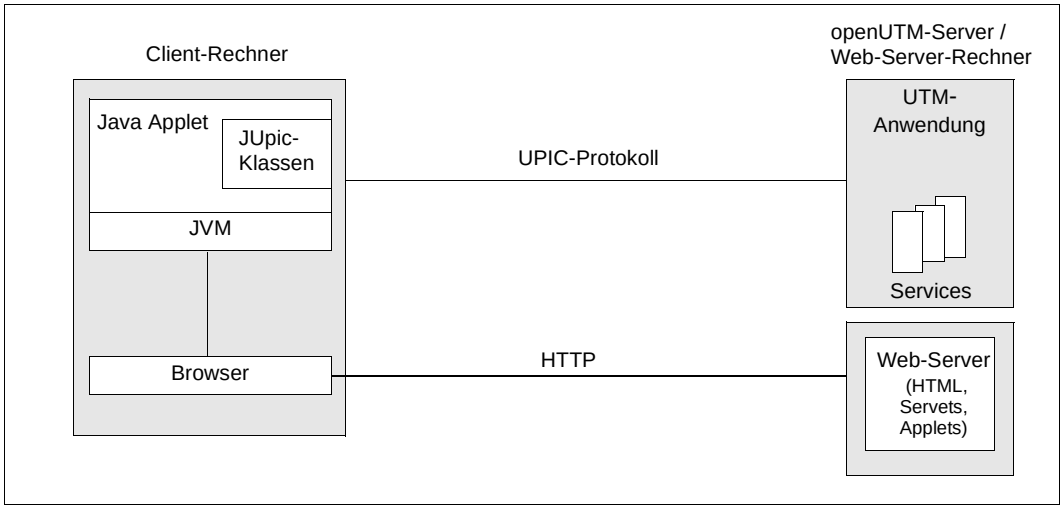


Bild 15: Anschluss in Browser-Umgebung über Applets

Über den Browser wird ein Applet vom Web-Server geladen und auf dem Client-Rechner gestartet. Dieses Applet kommuniziert dann direkt mit der UTM-Anwendung über das UPIC-Protokoll. Der Web-Server läuft in diesem Fall (Bild 15) aus Sicherheitsgründen auf dem gleichen Rechner wie die UTM-Anwendung, d.h. Applets dürfen nur mit dem Rechner kommunizieren, von dem sie heruntergeladen wurden.

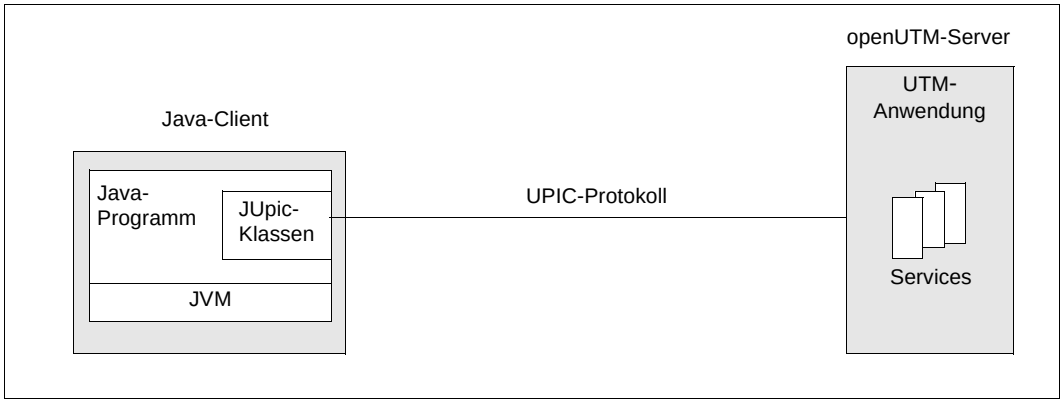


Bild 16: Direkter Anschluss von Java-Clients

Die JUpic-Klassen liefern die bewährte Funktionalität von UPIC wie z.B.:

- Conversation mit UTM-Anwendungen
- Multi-Threading, d.h. mehrere parallele Conversations
- Unterstützung der Wiederanlauffunktionen
- Unterstützung des UTM-Zugangs- und Zugriffsschutzes
- Passwortänderung durch den Benutzer möglich
- Anpassung der ASCII-EBCDIC-Konvertierung

Damit steht dem Java-Programmierer ein leistungsfähiger Zugang zu UTM-Anwendungen zur Verfügung, der zudem einen hohen Sicherheitslevel bietet.

Die JUpic-Klassen sind in den Java-Docs der Komponente JConnect beschrieben.

4.3 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 openUTM genutzt werden (siehe Abschnitt „Hintergrundaufträge an ferne Services (Remote Queuing)“ auf Seite 107).

Da die Server-Server-Kommunikation über den reinen Austausch von Nachrichten hinausgeht, sind hierfür spezielle höhere Kommunikationsprotokolle notwendig: openUTM 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.

Mit Hilfe des Produkts openUTM-LU62 ist auch ein Dialog mit Anwendungen möglich, die das Protokoll LU6.2 verwenden. openUTM-LU62 setzt das OSI TP-Protokoll in das LU6.2-Protokoll um, denn openUTM kommuniziert in diesem Fall über OSI TP, siehe auch Abschnitt „Kommunikation mit CICS-, IMS- und TXSeries-Anwendungen“ auf Seite 96.

Die Server-Server-Kommunikation ist in vollem Umfang Cluster-fähig, d.h. eine oder auch beide Server-Anwendungen können als UTM-Cluster-Anwendung realisiert werden, siehe auch Abschnitt „Lastverteilung bei verteilter Verarbeitung“ auf Seite 215.

4.3.1 Anwendungs-übergreifende Dialoge

Bei einem Anwendungs-übergreifenden Dialog laufen Auftraggeber-Service und Auftragnehmer-Service synchron ab und nicht entkoppelt wie beim Message Queuing (siehe Seite 103ff). Dabei ist es gleichgültig, ob der Auftraggeber-Service selbst im Dialog oder über Message-Queuing gestartet wurde.

Bei Anwendungs-übergreifenden 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 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.

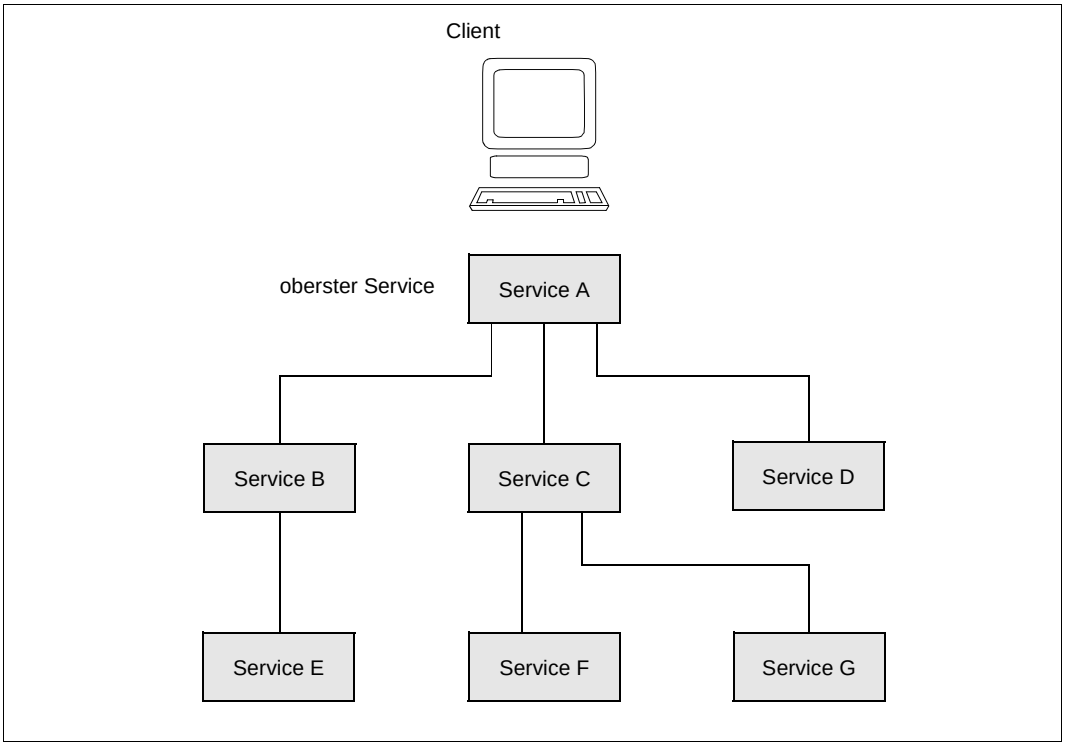


Bild 17: Service-Hierarchie bei Anwendungs-übergreifenden Dialogen

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 Anwendungs-übergreifender Dialoge

Bei Anwendungs-übergreifenden 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 openUTM einfach und sicher erstellen, da openUTM zur Steuerung Anwendungs-übergreifender Dialoge komfortable Kontrollmöglichkeiten zur Verfügung stellt.

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

4.3.2 Transaktionssicherung bei Server-Server-Kommunikation

UTM-Services bearbeiten Aufträge transaktionsorientiert, d.h. ein UTM-Service besteht aus einer oder 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 verteilten Transaktion, im zweiten Fall von unabhängigen Transaktionen. Über das LU6.1-Protokoll wird immer mit verteilten Transaktionen gearbeitet.

Verteilte Transaktionen

Bei der Server-Server-Kommunikation sind an der Bearbeitung eines Auftrags mehrere lokale Transaktionen in verschiedenen Anwendungen beteiligt. Falls mit globaler (=anwendungsübergreifender) Transaktionssicherung gearbeitet wird, garantiert openUTM, dass sich die Daten auch Anwendungs-übergreifend zu jeder Zeit in einem konsistenten Zustand befinden. Hierzu synchronisiert openUTM das Ende dieser Transaktionen: In allen beteiligten Anwendungen werden bei erfolgreichem Abschluss die Sicherungspunkte gleichzeitig gesetzt. Im Fehlerfall sorgt openUTM dafür, dass alle beteiligten Transaktionen zurückgesetzt werden. Die synchronisierten Transaktionen bilden also eine Einheit, die auch als „verteilte Transaktion“ bezeichnet wird. Für diese Synchronisation verwendet openUTM 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 im Abschnitt „Beispiel: Anwendungs-übergreifender Dialog mit verteilter Transaktion“ auf Seite 91.

Asynchron-Aufträge (MQ-Aufträge) werden bei Server-Server-Kommunikation mit globaler Transaktionssicherung genau einmal übertragen. Das bedeutet, dass auch bei Netzstörungen oder Abbruch einer Anwendung der Asynchron-Auftrag 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 verteilten Transaktion setzt bei der Zusammenarbeit zweier unabhängiger Transaktionen jede Anwendung ihre lokale Transaktion selbstständig 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.

4.3.3 Beispiel: Anwendungs-übergreifender Dialog mit verteilter 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 (verteilte Transaktion). Die Buchungen (Kasse und Kundenkonto) werden in lokalen Dialog-Services durchgeführt. Wie diese lokalen Dialog-Services miteinander kommunizieren, wird im folgenden Bild veranschaulicht.

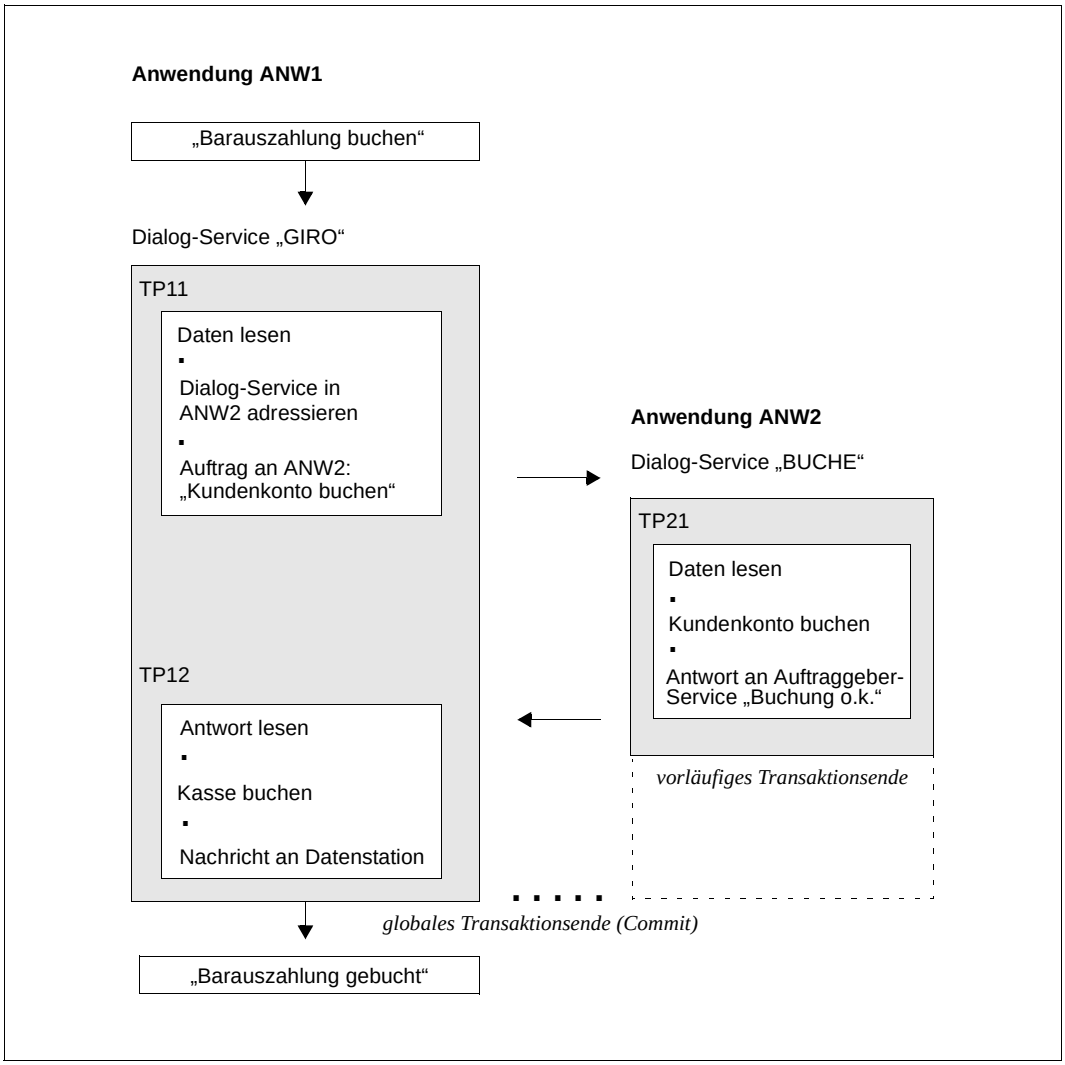


Bild 18: Dialogauftrag mit globaler Transaktionssicherung

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 Dialog-Nachricht. TP21 schickt eine Dialogantwort an den Auftraggeber-Service zurück und fordert anschließend von openUTM Transaktionsende an.

Diese Dialogantwort startet im Service „GIRO“ das Folgeteilprogramm TP12, welches die Buchungsaufgabe beendet und dann ebenfalls Transaktionsende anfordert. openUTM 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 verteilte Transaktion.

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

4.3.4 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 bzw. OSI-LPAP) und der logische Name für den fernen Service (LTAC) definiert und mit dem tatsächlichen Namen in der Partner-Anwendung verknüpft. Der logische Servicename entspricht in seiner Funktion dem Transaktionscode des Service. Er kann auf zwei Arten mit einer Partner-Anwendung verknüpft werden:

- Per Generierung
  In diesem Fall spricht man von **einstufiger Adressierung**, denn die Partner-Anwendung 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 UTM-Programmierhandbüchern: „Anwendungen programmieren mit KDCS für COBOL, C und C++“ und „Anwendungen erstellen mit X/Open-Schnittstellen“

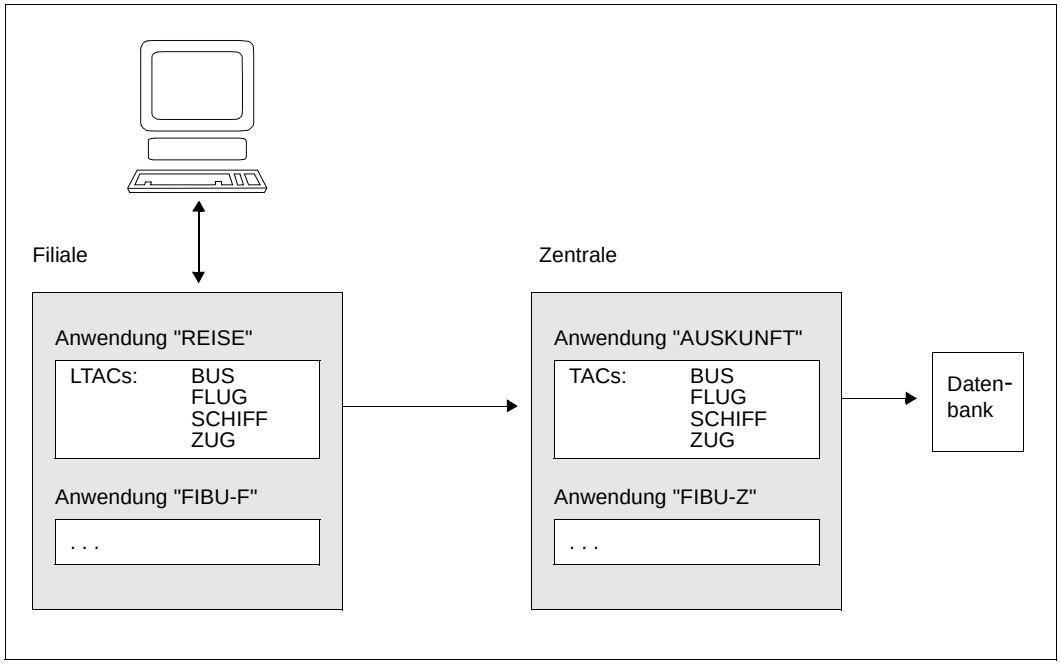


Bild 19: Einstufige Adressierung

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.

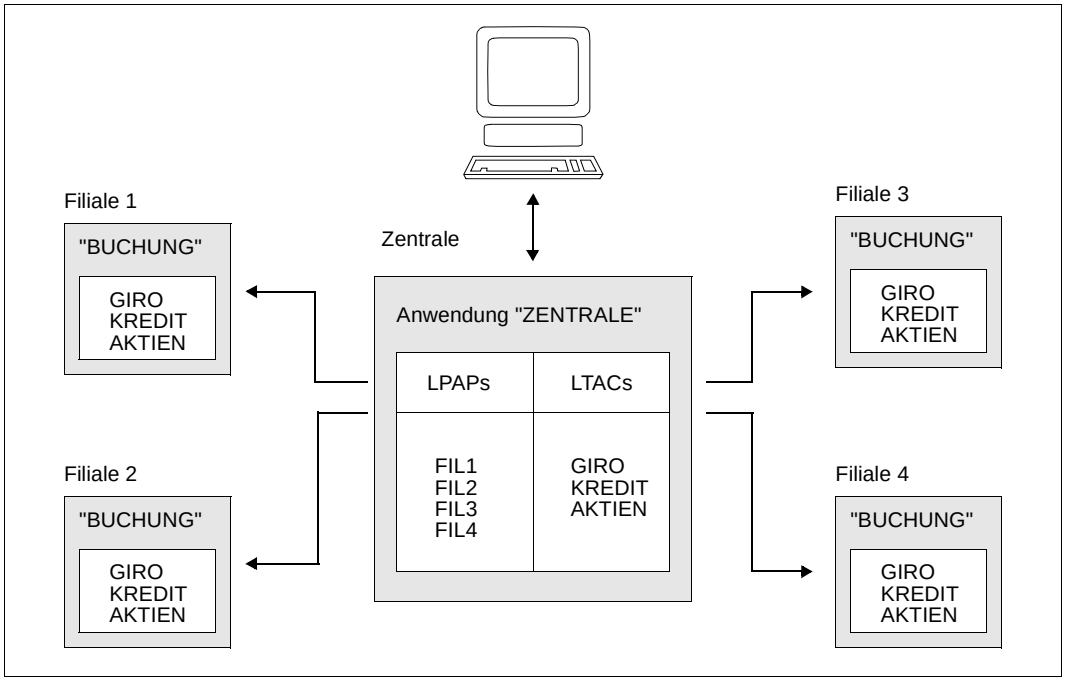


Bild 20: Zweistufige Adressierung

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 Partner-Anwendung auswählen.

4.3.5 Kommunikation mit CICS-, IMS- und TXSeries-Anwendungen

openUTM kann über die SNA-Protokolle LU6.1 und LU6.2 mit CICS-, IMS- und TXSeries-Anwendungen gekoppelt werden. Für neue Kopplungen sollte immer das Protokoll LU6.2 verwendet werden.

Einen LU6.2-Partner spricht openUTM über openUTM-LU62 an. Aus Sicht von openUTM ist CICS oder IMS ein OSI TP-Partner. Aus Sicht von CICS oder IMS ist openUTM ein LU6.2-Partner. Bei einer solchen Kopplung können die Partner sowohl mit als auch ohne globale Transaktionssicherung arbeiten.

Die wichtigsten Kopplungsmöglichkeiten entnehmen Sie bitte folgender Grafik.

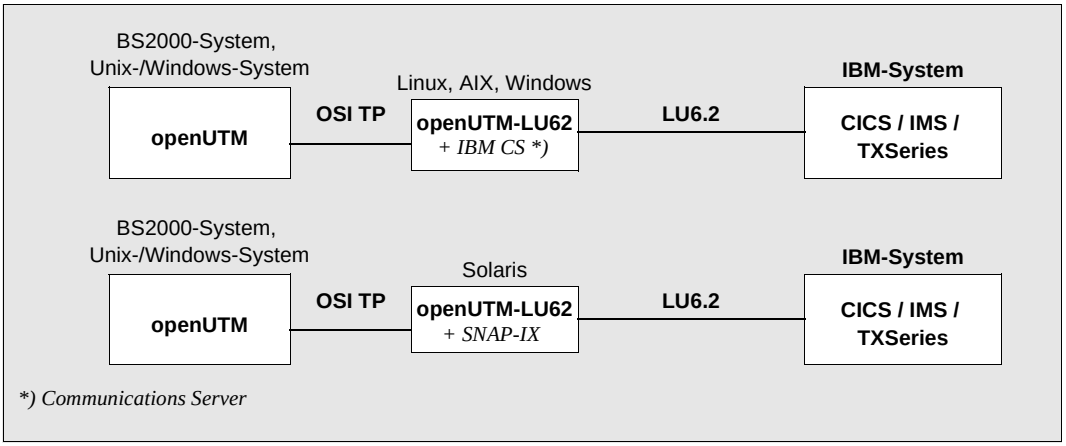


Bild 21: Anbindung von openUTM an IBM-Transaktionsmonitore

Im IBM-Host ist keine Zusatzsoftware notwendig, die SNA-Anbindung wird durch das Transportsystem-Gateway openUTM-LU62 zusammen mit den plattformspezifischen Komponenten SNAP-IX bzw. IBM Communications Server erbracht. Falls openUTM auf einem Unix- oder Windows- System läuft, können openUTM-LU62 und SNAP-IX bzw. IBM Communications Server auch auf diesem System installiert und dieses direkt an das SNA-Netz angeschlossen werden.

Auftraggeber und Auftragnehmer

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

Bei openUTM 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 Auftraggeber

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

CICS als Auftraggeber

CICS kann sowohl Dialog- als auch Asynchron-Services in openUTM 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 openUTM starten.
- Bei einer LU6.1-Kopplung kann IMS dagegen nur Asynchron-Services in openUTM starten. Damit IMS dennoch Dialoge führen kann, bietet LU6.1 die Möglichkeit, Pseudo-Dialoge zwischen Asynchron-Vorgängen zu bilden. Bei einem Pseudo-Dialog liefert der Auftraggeber zusätzliche Informationen, mit deren Hilfe der Auftragnehmer die Antwort an den richtigen Partner zurückschicken kann. openUTM 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 openUTM- und CICS, IMS- und LU6.2-Anwendungen“ beschrieben.

4.4 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- oder Windows-Systemen, DCAM-Anwendungen in BS2000-Systemen oder oder beliebige TCP/IP-Socket-Anwendungen.

Da keine höheren Protokolle verwendet werden, ist bei der Kommunikation mit Transportsystem-Anwendungen eine Unterstützung globaler Transaktionen nicht möglich. openUTM 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 verarbeitet 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 wird sie unter einer fest dem Partner zugeordneten Benutzerkennung, der so genannten Verbindungs-Benutzerkennung, bei der UTM-Anwendung angemeldet.

Besitzt die UTM-Anwendung einen selbsterstellten Anmeldedialog (Event-Service SIGNON, siehe Seite 189), 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 echte Benutzerkennung zuzuweisen.

Transportsystem-Anwendungen können die „Multi-Signon“-Funktion nutzen, d.h. es können sich gleichzeitig mehrere Transportsystem-Anwendungen unter derselben echten UTM-Benutzerkennung anmelden, wenn für diese Benutzerkennung auf den Vorgangswiederanlauf verzichtet wird.

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 openUTM erwarteten Weise formuliert sein; d.h. die ersten acht Zeichen der Nachricht müssen den Transaktionscode enthalten, unter dem der zu startende Service bei openUTM generiert wurde. Ob es sich bei diesem Service um einen Dialog- oder einen Asynchron-Service handelt, erkennt openUTM anhand des Transaktionscodes.
- Socket-Anwendungen schicken einen Byte-Strom, während openUTM nachrichtenorientiert arbeitet. Damit die UTM-Anwendung Nachrichtengrenzen erkennen kann, muss eine Socket-Anwendung der Nachricht ein Protokollfeld, genannt USP (UTM Socket Protokoll), voranstellen. Mit Hilfe des USP lassen sich auch Nachrichten in mehreren Teilen schicken, so dass die Gesamtnachricht größer sein kann als die maximale Größe des Eingabepuffers.
  Das USP kann wahlweise auch für die Ausgabe verwendet werden.
- Bei Dialog-Aufträgen muss für die Einhaltung des von openUTM geforderten strengen Dialogs gesorgt werden; d.h. die Partner-Anwendung muss den Empfang einer Antwort von der UTM-Anwendung abwarten, bevor sie die nächste Nachricht senden darf.
- openUTM 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 openUTM wartet danach auf eine Nachricht der Transportsystemanwendung. Deshalb ist es notwendig, bei Mehrschritt-Dialogen auf die Logik des Dialogablaufs zu achten.
- Die Nachrichtenlängen bei Kommunikation mit Nicht-Socket-Partnern sind durch die Größe des Ein-/Ausgabepuffers beschränkt.
- Bei Kommunikation mit Socket-Partnern können die Nachrichten fragmentiert werden, siehe obiger Hinweis zu USP. Dabei wird nur die Länge eines Fragments beschränkt, die Länge der Gesamtnachricht ist hingegen unbeschränkt.
- openUTM 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.
- Es kann per Generierung eine automatische Code-Umsetzung (ASCII-EBCDIC) veranlasst werden. In BS2000-Systemen ist dies nur für Socket-Partner möglich, auf anderen Plattformen für alle TS-Anwendungen.
- Restart-Fähigkeit: Gegebenenfalls wird ein Service-Wiederanlauf für Dialog-Services durchgeführt und dabei die letzte Ausgabe-Nachricht 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 jeweiliges openUTM-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 openUTM 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).

Mehr über Kopplungen mit Socket-Anwendungen finden Sie im openUTM-Handbuch „Anwendungen generieren“.

4.5 Übersicht: Partner, Protokolle,Transaktionssicherung

In folgender Tabelle sind Partner aufgelistet, mit denen eine UTM-Anwendung 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 –
openUTM-Client-Anwendungen mit Trägersystem UPIC	UPIC	x	–
Java-Client mit JUpic-Klassen	UPIC	x	–
openUTM-Client-Anwendungen mit Trägersystem OpenCPIC	OSI TP	x	x
Java EE Anwendungen	UPIC, Transportsystem	x	–
	OSI TP	x	x
CICS-/IMS-/TXSeries-Anwendungen auf einem IBM-Mainframe	LU6.1	–	x
	openUTM: OSI TP IBM: LU6.2	x	x
CICS-/IMS-Anwendungen auf anderen IBM-Systemen wie z.B: AIX-Rechner	openUTM: OSI TP IBM: LU6.2	x	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	–
PCMX-Anwendungen in einem fernen Unix- oder Windows-System	Transportsystem	x	–
andere Transportsystem-Anwendungen, die auf OSI-Transportschichten oder auf RFC1006 über TCP/IP aufsetzen	Transportsystem	x	–
Socket-Anwendungen	Transportsystem	x	–

5 Message Queuing

Vorzüge und Einsatzmöglichkeiten der in openUTM integrierten Message Queuing-Funktionalität wurden Ihnen bereits in Abschnitt „Message Queuing“ auf Seite 42ff kurz vorgestellt.

Hier noch einmal die wichtigsten Eigenschaften:
– zeitliche und räumliche Unabhängigkeit der kommunizierenden Komponenten
– absolute Verlässlichkeit der Nachrichtenübertragung durch die transaktionsgesicherte entkoppelte Zustellung (Deferred Delivery Mechanismus)
– Unabhängigkeit von der augenblicklichen Verbindung

Message Queuing (MQ) ist eine Form der Kommunikation, bei der die Nachrichten (Messages) über zwischengeschaltete Queues ausgetauscht werden (store and forward). Deshalb werden diese Nachrichten auch **Asynchron-Nachrichten** genannt. Die Kommunikation wird in Form von **Asynchron-Aufträgen** abgewickelt. Ein Asynchron-Auftrag besteht aus der Asynchron-Nachricht, dem Empfänger der Nachricht und ggf. dem gewünschten Ausführungszeitpunkt.

Je nachdem, wer für die Verarbeitung der an die Queue gerichteten Nachrichten verantwortlich ist, unterscheidet man UTM-gesteuerte Queues und Service-gesteuerte Queues.

- Bei **UTM-gesteuerten Queues** wird der zwischengeschaltete Queuing-Mechanismus komplett von openUTM zur Verfügung gestellt, d.h openUTM übernimmt neben der reinen Queuing-Funktionalität auch Trigger-Funktionen.
- Bei **Service-gesteuerten Queues** ist ein Service verantwortlich für die Weiterverarbeitung der Nachricht, d.h. openUTM führt eine reine Queuing-Funktionalität aus.

Für beide Arten von Message Queues stehen folgende Steuerungsmöglichkeiten zur Verfügung, siehe auch Seite 115:

– Zeitsteuerung
– Rückmeldungen über Quittungsaufträge
– Administration von Queues

Für das Message Queuing gibt es MQ-Aufrufe der KDCS-Schnittstelle, die im Abschnitt „Message Queue-Aufrufe der KDCS-Schnittstelle“ auf Seite 119 vorgestellt werden.

5.1 UTM-gesteuerte Queues

Bei UTM-gesteuerten Queues übernimmt openUTM die komplette Folgeverarbeitung, sobald die Nachricht in die Queue gestellt wird. Abhängig vom Empfänger unterscheidet man **Ausgabeaufträge** (Output Queuing) und **Hintergrundaufträge**.

5.1.1 Ausgabeaufträge (Output Queuing)

Ausgabeaufträge 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 „Kommunikation mit Transportsystem-Anwendungen“ auf Seite 98).

Ausgabeaufträge setzen sich zusammen aus der Angabe des Ausgabeziels und der Nachricht, die asynchron ausgegeben werden soll. Sie werden ohne Mitwirkung des Anwendungsprogrammes automatisch von den UTM-Systemfunktionen abgearbeitet.

Ausgabeaufträge können durch MQ-Aufrufe aus einem Dialog- oder Asynchron-Service der UTM-Anwendung ausgelöst werden.

5.1.2 Hintergrundaufträge

Hintergrundaufträge sind Asynchron-Aufträge, die an einen Asynchron-Service (= Asynchron-Vorgang) der eigenen oder einer fernen Anwendung gerichtet sind. Daher spricht man auch von Local Queuing bzw. Remote Queuing. Hintergrundaufträge eignen sich besonders für zeitintensive oder zeitunkritische Verarbeitungen, deren Ergebnis keinen direkten Einfluss auf den aktuellen Dialog hat.

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

Hintergrundaufträge können ausgelöst werden durch:

- Eingabe von einem Terminal
- Aufruf aus einem openUTM-Client-Programm mit Trägersystem OpenCPIC
- eine Nachricht von einer anderen Anwendung, die mit der UTM-Anwendung über das LU6.1- oder OSI TP-Protokoll kommuniziert
- Eingabe von einer anderen Anwendung, die über die Transportsystemschnittstelle angeschlossen ist
- MQ-Aufruf aus einem Service der lokalen Anwendung oder einem Service einer fernen UTM-Anwendung
- UTM-Meldungen (also ereignisgesteuert)

Hintergrundaufträge sind Redelivery-fähig, d.h. sie können nach abnormalem Beenden eines Asynchron-Vorgangs erneut gestartet werden; die asynchronen Nachrichten werden dann erneut zugestellt.

Alternativ dazu bzw. nach letztmaliger Redelivery kann openUTM eine fehlerhaft verarbeitete FGET-Nachricht eines Asynchron-Vorgangs in eine eigene Queue stellen, die so genannte Dead Letter Queue.

5.1.2.1 Bearbeitung von Hintergrundaufträgen

Zur Bearbeitung eines Hintergrundauftrags 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.

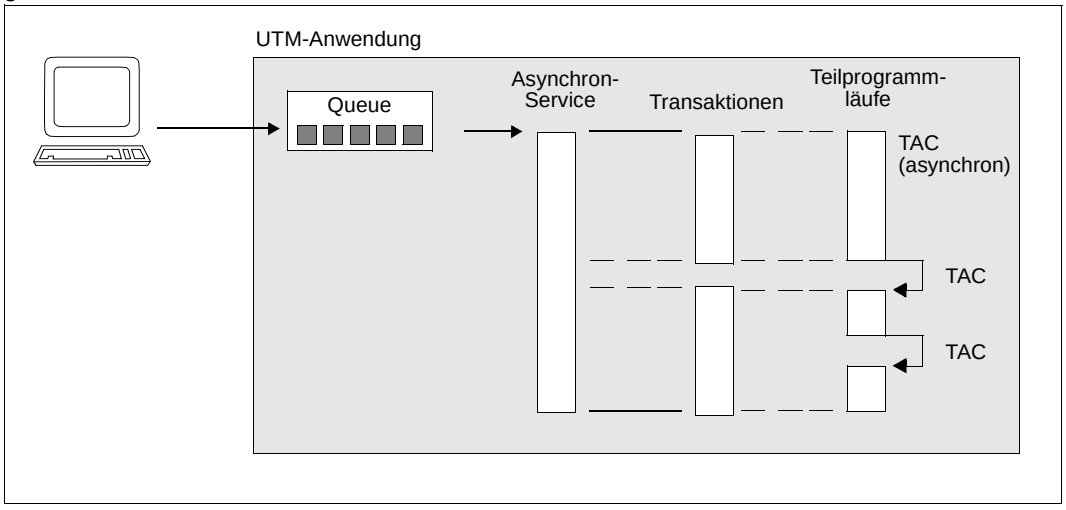


Bild 22: Struktur eines Asynchron-Services, von einem Terminal erteilt

In dem in Bild 22 dargestellten Beispiel wird von einem Terminal aus ein Hintergrundauftrag an einen Asynchron-Service der eigenen Anwendung gestellt. Hierzu wird am Terminal der Transaktionscode dieses Services und ggf. eine Nachricht eingegeben. openUTM 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 Ausgabeaufträge oder weitere Hintergrundaufträge 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.

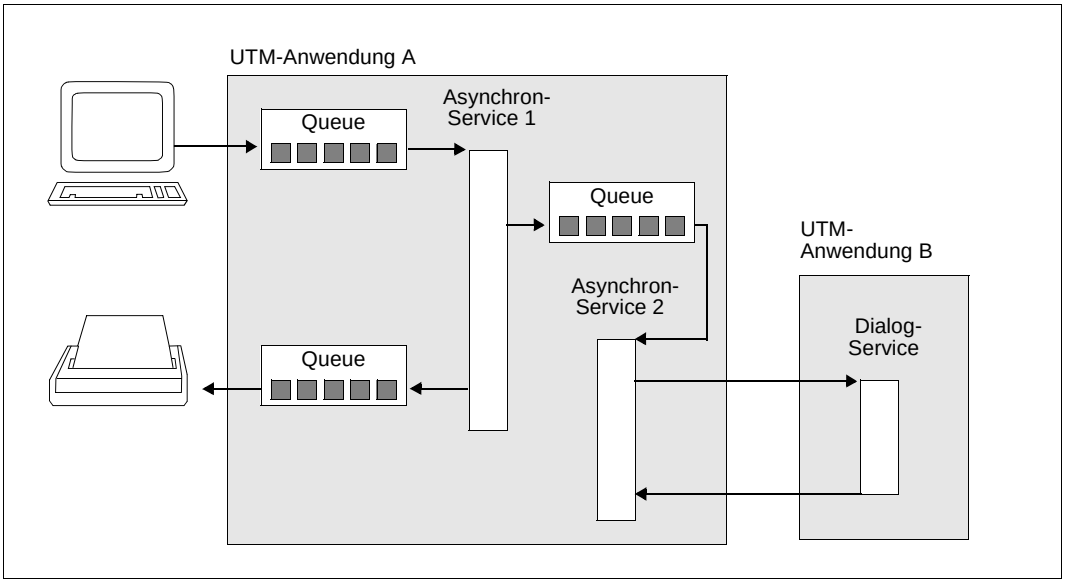


Bild 23: Asynchron-Services, die ihrerseits Aufträge absetzen

In Bild 23 wird - wie in Bild 22 - vom Terminal aus ein Hintergrundauftrag an einen lokalen Asynchron-Service gestellt. Innerhalb dieses Services wird ein weiterer Hintergrundauftrag abgesetzt sowie ein Ausgabeauftrag 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 openUTM automatisch zur Verfügung gestellt.

5.1.2.2 Hintergrundaufträge an ferne Services (Remote Queuing)

Hintergrundaufträge 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 openUTM, d.h. openUTM arbeitet bei Hintergrundaufträgen 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 Rechner-übergreifende Message Queuing mit openUTM 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.

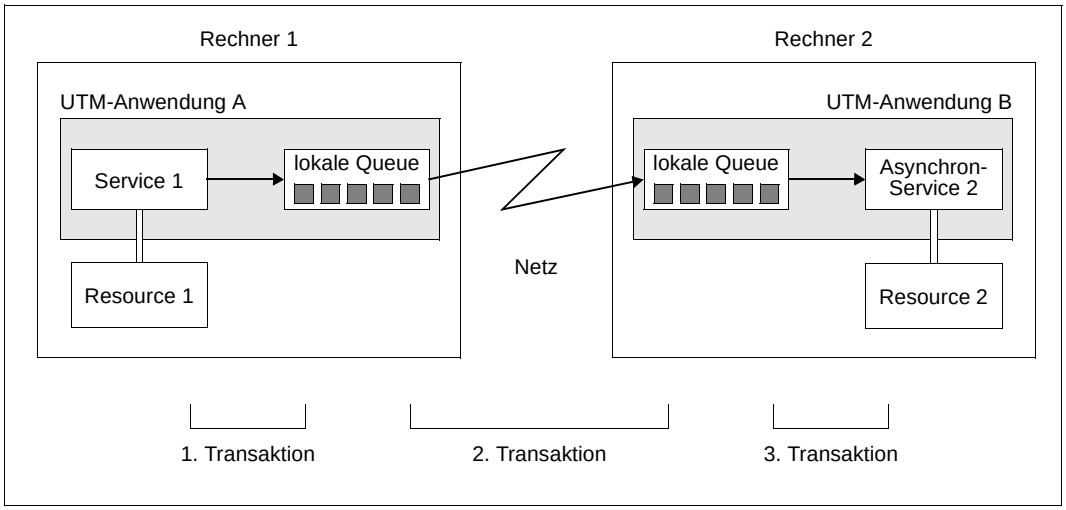


Bild 24: Remote Queuing mit openUTM

Bild 24 zeigt, wie ein Hintergrundauftrag an eine ferne Anwendung von openUTM 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 lang anhaltender Sperren und „hängender“ Transaktionen. Durch die Kommunikation über Remote Queuing kann vermieden werden, dass lokale Störungen oder Ausfälle zu übergreifenden, länger andauernden Blockaden führen.

5.1.3 Priority Scheduling bei Hintergrundaufträgen

Hintergrundaufträge werden oft dazu verwendet, besonders zeitintensive Aufgaben auszulagern. Dabei muss darauf geachtet werden, dass nicht zu viele Prozesse einer Anwendung gleichzeitig von Hintergrundaufträgen belegt werden, da sich dies nachteilig auf die Antwortzeiten für die Dialogverarbeitung auswirken könnte.

openUTM bietet deshalb ein zweistufiges Priority-Scheduling-Konzept:

- Es kann festgelegt werden, wie viele Prozesse einer UTM-Anwendung gleichzeitig Hintergrundaufträge 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 Hintergrundaufträge differenziert werden. Dazu werden 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 Hintergrundaufträge zur Verfügung stehen sollen, oder wie Sie TAC-Klassen definieren und priorisieren können, erfahren Sie im openUTM-Handbuch „Anwendungen generieren“ unter den Stichwörtern ASYNTASKS, TACCLASS und TAC-PRIORITIES.

5.2 Service-gesteuerte Queues

Bei einer Service-gesteuerten Queue müssen die Nachrichten, die in der Queue stehen, durch einen Service der Anwendung abgeholt werden, um sie anschließend zu verarbeiten. D.h. die Initiative muss immer von einem Service der Anwendung ausgehen, da openUTM für diese Queues keine Trigger-Funktionalität übernimmt.

openUTM stellt dazu drei Arten von Service-gesteuerten Queues zur Verfügung:

- USER-Queues:
  Eine USER-Queue ist Benutzer-spezifisch und steht jedem UTM-Benutzer automatisch zur Verfügung.
- TAC-Queues:
  Diese Art Queues stehen im Prinzip jedem Service zur Verfügung. Sie haben einen festen Namen, der explizit generiert werden muss. Eine Ausnahme ist die Dead Letter Queue, die den festen Namen KDCDLETQ trägt und die nicht generiert werden muss.
- Temporäre Queues:
  Eine Temporäre Queue wird dynamisch durch einen Service per Programmaufruf erzeugt und auch wieder gelöscht.

openUTM unterstützt für alle Service-gesteuerten Queues sowohl die Browse- als auch die Verarbeitungsfunktion:

- Browsen
  Die Nachricht kann gleichzeitig von mehreren Vorgängen gelesen werden und bleibt nach dem Lesen in der Queue.
- Verarbeiten
  Die Nachricht kann zu einer Zeit nur von einem Vorgang gelesen werden und wird nach dem Lesen gelöscht.

Die Service-gesteuerten Queues unterliegen ebenso wie die UTM-gesteuerten Queues dem Transaktionskonzept von openUTM:

- Die Nachrichten werden transaktionsgesichert in die Queue eingereiht und so lange dort ausfallsicher gespeichert, bis sie (transaktionsgesichert) durch die Services abgeholt und weiterverarbeitet werden.
- Wird eine Transaktion zurückgesetzt, dann werden die in dieser Transaktion verarbeiteten Nachrichten wieder in die Queue gestellt und können erneut gelesen werden (Redelivery). Alternativ dazu bzw. nach letztmaliger Redelivery kann openUTM eine fehlerhaft verarbeitete Nachricht einer TAC-Queue in die Dead Letter Queue stellen.

Die maximale Anzahl der erneuten Zustellungen nach Rücksetzen der Transaktion lässt sich einstellen. Details finden Sie im openUTM-Handbuch „Anwendungen generieren“ unter dem Stichwort REDELIVERY.

5.2.1 USER-Queues

USER-Queues sind permanente Service-gesteuerte Message Queues. Jedem generierten UTM-Benutzer steht jederzeit eine USER-Queue zur Verfügung. Auf USER-Queues kann im Prinzip jeder Service per Programmaufruf zugreifen, sofern er den Namen des Users kennt. Um unkontrollierte Zugriffe zu vermeiden, sind USER-Queues in das Autorisierungskonzept von openUTM integriert. Damit können Sie die Zugriffsrechte rollenspezifisch regeln.

Eine USER-Queue existiert so lange wie die zugehörige UTM-Benutzerkennung.

Mit USER-Queues können Sie z.B. Mailbox-Anwendungen für UTM-Benutzer realisieren. Eine weitere Anwendung für USER-Queues zeigt das Beispiel im folgenden Bild. Dort dienen USER-Queues dazu, Meldungen oder Warnungen asynchron an UPIC-Clients zu schicken.

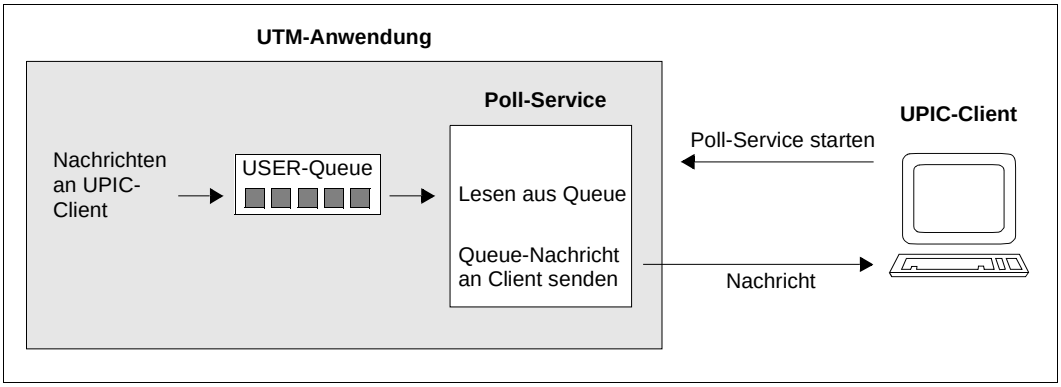


Bild 25: Beispiel: Nutzung einer USER-Queue für Asynchron-Nachrichten an UPIC-Client

Bei diesem Beispiel wird die Tatsache ausgenutzt, dass ein UPIC-Client sich mit einer UTM-Benutzerkennung bei openUTM anmeldet. Asynchrone Nachrichten an den UPIC-Client gehen über die USER-Queue dieser Benutzerkennung.

Der Poll-Service wird bei Dialogbeginn durch den UPIC-Client gestartet. Er liest die Nachricht aus der USER-Queue der Benutzerkennung und schickt sie sofort als Dialog-Nachricht an den UPIC-Client. Ist keine Nachricht in der Queue, dann wartet der Poll-Service, bis eine Nachricht eintrifft. Wenn das UPIC-Clientprogramm multi-threaded programmiert wird, dann können Poll-Service und normaler Dialog parallel laufen. Damit kann der Benutzer am Client automatisch über das Eintreffen asynchroner Nachrichten informiert werden, z.B. durch einen Icon oder eine Dialogbox.

Wie Sie USER-Queues definieren und wie Sie deren Eigenschaften festlegen können, erfahren Sie im openUTM-Handbuch „Anwendungen generieren“ unter den Stichwörtern USER, Q-READ-ACL, Q-WRITE-ACL und USER-Queue.

5.2.2 TAC-Queues

TAC-Queues sind permanente Service-gesteuerte Message Queues. Sie besitzen einen festen, per Generierung festgelegten Namen. Auf TAC-Queues kann im Prinzip jeder Service per Programmaufruf zugreifen, sofern er den Namen der Queue kennt. Um unkontrollierte Zugriffe zu vermeiden, sind TAC-Queues in das Autorisierungskonzept von openUTM integriert. Damit können Sie die Zugriffsrechte rollenspezifisch regeln.

Eine TAC-Queue existiert für einen unbegrenzten Zeitraum, falls sie nicht explizit per Administration gelöscht wird.

Die Dead Letter Queue ist eine TAC-Queue mit dem festen Namen KDCDLETQ. Sie steht immer zur Verfügung, um Asynchron-Nachrichten an Transaktionscodes oder TAC-Queues zu sichern, die nicht verarbeitet werden konnten. Um diese Nachrichten nach einer Fehlerbehebung noch verarbeiten zu können, können sie in andere Message Queues verschoben, d.h. entweder ihrem ursprünglichen Ziel oder einem neuen Ziel zugeordnet werden. Das Nachrichten-Aufkommen in der Dead Letter Queue kann mit der Meldung `K134` überwacht werden.

Das folgende Bild zeigt ein Beispiel, wie Sie TAC-Queues verwenden können, um Meldungen an den grafischen Administrationsplatz WinAdmin weiterleiten und dort archivieren zu können. Das Beispiel lässt sich analog auch für WebAdmin realisieren.

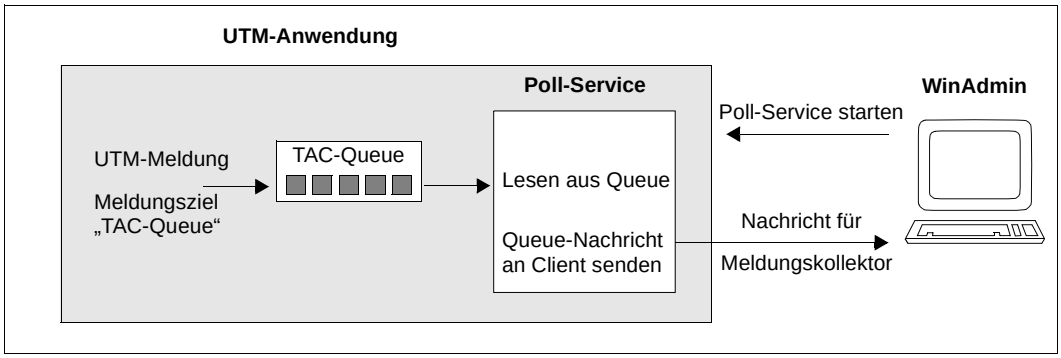


Bild 26: Beispiel: Nutzung einer TAC-Queue für Meldungskollektoren von WinAdmin

Der Meldungskollektor wird in WinAdmin definiert. Er stellt einen Polling-Mechanismus dar, der die UTM-Meldungen - je nach Einstellung in WinAdmin- entweder zyklisch oder auf explizite Anforderung aus der TAC-Queue abholt. Auf UTM-Seite muss in diesem Fall nur die TAC-Queue generiert und ein Meldungsmodul erzeugt werden, in dem festgelegt ist, welche Meldungen an die TAC-Queue gesendet werden sollen.

Der Poll-Service für WinAdmin und WebAdmin wird mit openUTM ausgeliefert (Teilprogramm KDCWADM).

Wie Sie TAC-Queues definieren und wie Sie deren Eigenschaften festlegen können, erfahren Sie im openUTM-Handbuch „Anwendungen generieren“ unter den Stichwörtern TAC, Q-READ-ACL, Q-WRITE-ACL und TAC-Queue.

5.2.3 Temporäre Queues

Temporäre Queues werden per Programmaufruf erzeugt und können per Programm auch wieder gelöscht werden. Der Name der Queue wird beim Erzeugen einer Queue vergeben, d.h. er muss nicht generiert werden. Der Name kann wahlweise durch das Programm oder durch openUTM erzeugt werden.

Bei UTM-S bleiben Temporäre Queues über ein Anwendungsende hinaus erhalten, falls sie nicht zuvor explizit gelöscht wurden. Bei UTM-F gehen alle Temporären Queues verloren, nachdem die Anwendung beendet wurde.

Temporäre Queues eignen sich besonders gut zur freien Kommunikation zwischen voneinander unabhängigen Services („freier“ Dialog). Temporäre Queues verkörpern daher im besonderen Maße das Konzept des **synchronen Wartens auf asynchrone Ereignisse**. Sie stehen damit zwischen den beiden Konzepten „Strenger Dialog“ und „Hintergrundaufträge“.

Die Services können innerhalb einer Anwendung residieren oder sich auf unterschiedlichen Rechnern befinden. Zwei wichtige Anwendungsfälle für dieses Konzept sind:

- Der Dialog zweier Services innerhalb einer UTM-Anwendung
- Der Dialog einer UTM-Anwendung mit einer fernen Transportsystem-Anwendung

Beide Möglichkeiten werden im Folgenden anhand von Grafiken erläutert.

Im ersten Beispiel sucht der Service „Kunde“ einer Call-Center-Anwendung zusätzliche Kundendaten mit Hilfe des asynchronen Services „Suchen“. Dieser schreibt die Daten in eine Temporäre Queue, die in diesem Fall als reine „Reply-Queue“ verwendet wird, d.h. der Datenaustausch geht nur in eine Richtung.

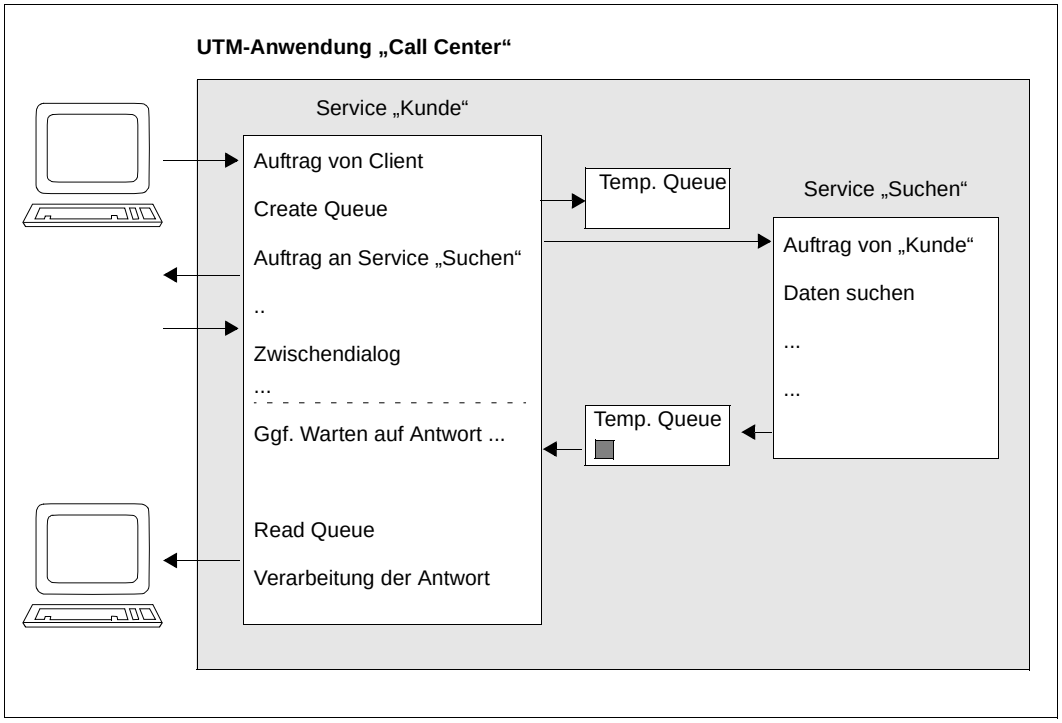


Bild 27: Kommunikation zweier Services über eine Temporäre Queue

Der Service „Kunde“ erzeugt die Temporäre Queue und gibt den Namen der Queue beim Auftrag an den Service „Suchen“ weiter. Danach führt der Service „Kunde“ weitere Dialogabfragen mit dem Client aus, während der Service „Suchen“ im Hintergrund eine Datenbankabfrage durchführt. Wenn nötig wartet der Service „Kunde“ auf die Antwort vom Service „Suchen“ (synchrones Warten). Hat dieser die gewünschten Daten erhalten, schreibt er sie in die Queue und beendet sich anschließend. Sobald die Nachricht in der Queue steht, wird „Kunde“ aktiviert, übernimmt die weitere Verarbeitung und löscht anschließend die Queue.

Im zweiten Beispiel möchte eine UTM-Anwendung auf die Daten einer fernen Transportsystem-Anwendung zugreifen. Als reine Transportsystem-Anwendung kann diese keine höheren Protokolle wie. z.B. OSI TP oder LU6.1 für die Kommunikation einsetzen. Um dennoch einen Dialog führen zu können, werden auf UTM-Seite Temporäre Queues verwendet.

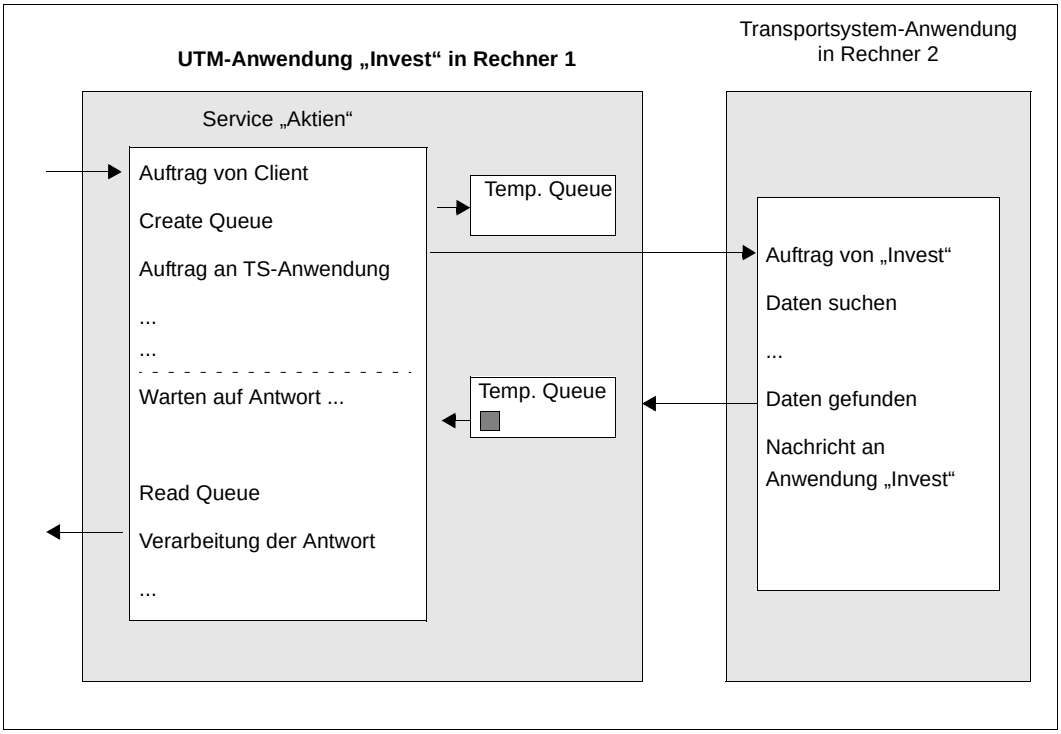


Bild 28: Kommunikation mit einer Transportsystem-Anwendung über eine Temporäre Queue

Der Service „Aktien“ erzeugt die Temporäre Queue und gibt den Namen der Queue beim Auftrag an die Transportsystem-Anwendung weiter. Danach wartet der Service „Aktien“ auf die Antwort (synchrones Warten). Inzwischen ermittelt die Transportsystem-Anwendung die gewünschten Daten und sendet sie an die Anwendung „Invest“; der Name der Queue muss dabei am Anfang der Nachricht stehen. Der Service „Aktien“ wird aktiviert, sobald die Nachricht in der Queue steht. Nach der Verarbeitung wird die Queue gelöscht.

Wie Sie Temporäre Queues erzeugen und löschen, ist im openUTM-Handbuch „Anwendungen programmieren mit KDCS“ beschrieben. Wie Sie die maximale Anzahl und die Eigenschaften von Temporären Queues festlegen können, erfahren Sie im openUTM-Handbuch „Anwendungen generieren“ unter den Stichwörtern QUEUE und Temporäre Queue.

5.3 Steuerungsmöglichkeiten für Message Queues

Für UTM-gesteuerte Queues und Service-gesteuerte Queues bietet openUTM verschiedene Steuerungsmöglichkeiten, die in die MQ-Aufrufe der KDCS-Schnittstelle integriert sind.

Quittungsaufträge

Zusammen mit der Nachricht an die Queue („Basisauftrag“) können bis zu zwei **Quittungsaufträge** formuliert werden, die an das positive bzw. negative Ergebnis der Auftragsdurchführung gebunden sind. Quittungsaufträge sind für folgende Arten von Basisaufträgen möglich:

– Hintergrundaufträge
– Ausgabeaufträge an Terminals, Drucker oder Transportsystem-Anwendungen
– Aufträge an TAC-Queues

Quittungsaufträge werden ausgeführt, wenn der Basisauftrag abgewickelt 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 **Auftrags-Komplex** bezeichnet. Das Verhalten ist je nach Art der Queue unterschiedlich.

Lokaler Hintergrundauftrag

Der positive Quittungsauftrag wird gestartet, nachdem der Asynchron-Service ordnungsgemäß beendet wurde.

Der negative Quittungsauftrag wird gestartet, wenn der Asynchron-Service abnormal beendet wurde und wenn kein Redelivery aktiviert ist (siehe Seite 117). Das abnormale Beenden kann durch einen UTM-Aufruf im Programm (PEND ER/FR) oder durch einen schwerwiegenden Programmfehler verursacht werden.

Hintergrundauftrag an einen fernen Service (Remote Queuing)

Der positive Quittungsauftrag wird gestartet, wenn der Auftrag erfolgreich in die Queue des fernen Services übertragen wurde.

Der negative Quittungsauftrag wird gestartet, wenn eine solche Übertragung dauerhaft nicht möglich ist.

Ausgabeauftrag

Der positive Quittungsauftrag wird gestartet, nachdem die positive Abdruckquittung vom Drucker oder der Druckeradministration eingetroffen ist.

Der negative Quittungsauftrag wird gestartet:

- nach Fehlern beim Aufbereiten der Nachricht durch VTSU-B oder beim Formatieren,
- beim Löschen des Auftrags beim Verbindungsaufbau-/abbau von Clients, die mit RESTART=NO generiert sind,
- falls ein Auftrag per Administration gelöscht wurde.

Der negative Quittungsauftrag wird jedoch nicht gestartet, wenn eine negative Abdruckquittung vorliegt, bei einer Zeitüberschreitung beim Warten auf die Quittung oder wenn der Druckauftrag wiederholt wird.

Auftrag an eine TAC-Queue

Der positive Quittungsauftrag wird gestartet, nachdem die Nachricht in der TAC-Queue erfolgreich gelesen und die Transaktion ordnungsgemäß beendet wurde.

Der negative Quittungsauftrag wird gestartet,

- wenn die Nachricht mit DADM DL gelöscht wird und dabei explizit angegeben wird, dass der negative Quittungsauftrag gestartet werden soll (KCMOD=N),
- wenn die Transaktion bei der Verarbeitung der Nachricht zurückgesetzt wurde und die maximale Anzahl wiederholter Zustellungen noch nicht erreicht ist, siehe Abschnitt „Erneute Zustellung (Redelivery)“ auf Seite 117“.

Zeitsteuerung

Zeitsteuerung ist für Hintergrundaufträge, Ausgabeaufträge an LTERM-Partner und LPAP-Partner sowie Nachrichten an TAC-Queues möglich. Dabei kann angegeben werden, wann der Auftrag frühestens ausgeführt werden soll bzw. wann die Nachricht frühestens gelesen werden kann. 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 openUTM gestartet, nachdem der gewünschte Zeitpunkt erreicht ist und die notwendigen Betriebsmittel zur Durchführung des Auftrags von openUTM bereitgestellt werden können.

Erneute Zustellung (Redelivery)

Für Hintergrundaufträge und Nachrichten an Service-gesteuerte Queues lässt sich die Neuzustellung von Nachrichten steuern. Per Generierung kann man einstellen:

- ob eine Nachricht an einen Asynchron-Vorgang nach dem abnormalen Vorgangsende erneut zugestellt wird,
- ob die Nachricht an eine Service-gesteuerte Queue nach Rücksetzen der Transaktion erneut zugestellt wird,
- und wie oft eine solche Neuzustellung maximal wiederholt werden kann. Für Hintergrundaufträge und Service-gesteuerte Queues sind unterschiedliche Werte möglich. Mit einer Obergrenze lassen sich z.B. Endlosschleifen verhindern.

Mit der Redelivery-Funktion kann man sicherstellen, dass die Nachrichten nach dem Vorgangsabbruch/Rücksetzen der Transaktion erhalten bleiben und nicht sofort gelöscht werden. openUTM stellt zudem einen Wiederholungszähler zur Verfügung, der vom Teilprogramm gelesen werden kann. Damit kann das Programm „Schleifensituationen“ erkennen und entsprechend darauf reagieren.

Wenn eine Nachricht erneut zugestellt wird, dann werden keine negativen Quittungsaufträge aktiviert.

Sicherung fehlerhaft verarbeiteter Aufträge in der Dead Letter Queue

In der Dead Letter Queue werden Asynchron-Nachrichten an Transaktionscodes oder TAC-Queues gesichert, die nicht verarbeitet werden konnten. Auf diese Weise können Sie einen Nachrichtenverlust bei dauerhaften Fehlern vermeiden, ohne dass es zu Endlosschleifen kommt.

Für Asynchron-Transaktionscodes mit `CALL=BOTH/FIRST` und TAC-Queues kann per Generierung festgelegt werden, ob ihre Nachrichten im Fehlerfall in der globalen Dead Letter Queue gespeichert werden sollen. Dies kann alternativ zu Redelivery durchgeführt werden oder auch als letzte Rückfallstufe dienen, nachdem die maximale Anzahl von Redelivery-Versuchen erreicht wurde.

Administration von Message Queues

Die Bearbeitung von Nachrichten und Aufträgen in Message Queues kann über die UTM-Administration beeinflusst werden. Dies gilt gleichermaßen für UTM-gesteuerte als auch Service-gesteuerte Queues. Ein Asynchron-Auftrag oder eine Nachricht in einer USER-Queue kann z.B. in der Bearbeitungsreihenfolge vorgezogen oder auch storniert werden. Nachrichten der Dead Letter Queue können außerdem auch anderen Message Queues (TAC-Queues und Asynchron-TACs) zugewiesen werden. Für die Administration von

Message Queues steht der KDCS-Aufruf DADM zur Verfügung (siehe folgender Abschnitt). Alternativ dazu können Sie auch den grafischen Administrationsplatz WinAdmin oder WebAdmin verwenden.

5.4 Message Queue-Aufrufe der KDCS-Schnittstelle

openUTM stellt an der Programmschnittstelle funktionell mächtige, aber einfach zu versorgende Aufrufe für Message Queuing Funktionen 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 (Ausgabeaufträge), an Asynchron-Services (Hintergrundaufträge) oder an TAC-Queues. 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.

  Darüber hinaus können mit dem DPUT-Aufruf auch USER-Queues, TAC-Queues oder Temporäre Queues adressiert werden.

- FGET (**F**ree message **GET**)

  Der Aufruf FGET dient zum Lesen von Asynchron-Nachrichten oder -Teilnachrichten innerhalb eines Asynchron-Services.

- DGET (**D**ata **GET**)

  Mit dem Aufruf DGET werden Nachrichten aus USER-Queues, TAC-Queues oder Temporären Queues gelesen.

- QCRE (**Q**ueue **CRE**ate)

  Mit dem Aufruf QCRE wird dynamisch eine Temporäre Nachrichten-Queue erzeugt.

- QREL (**Q**ueue **REL**ease)

  Mit dem Aufruf QREL wird dynamisch eine Temporäre Nachrichten-Queue gelöscht.

- MCOM (**M**essage **COM**plex)

  Mit dem Aufruf MCOM werden einem Asynchron-Auftrag Quittungsaufträge zugeordnet.

- 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, DGET, MCOM, QCRE, QREL und DADM sowie weitere Informationen zu diesen Aufrufen finden Sie im openUTM-Handbuch „Anwendungen programmieren mit KDCS“. Der Aufruf DADM ist auch im openUTM-Handbuch „Anwendungen administrieren“ beschrieben.

6 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.

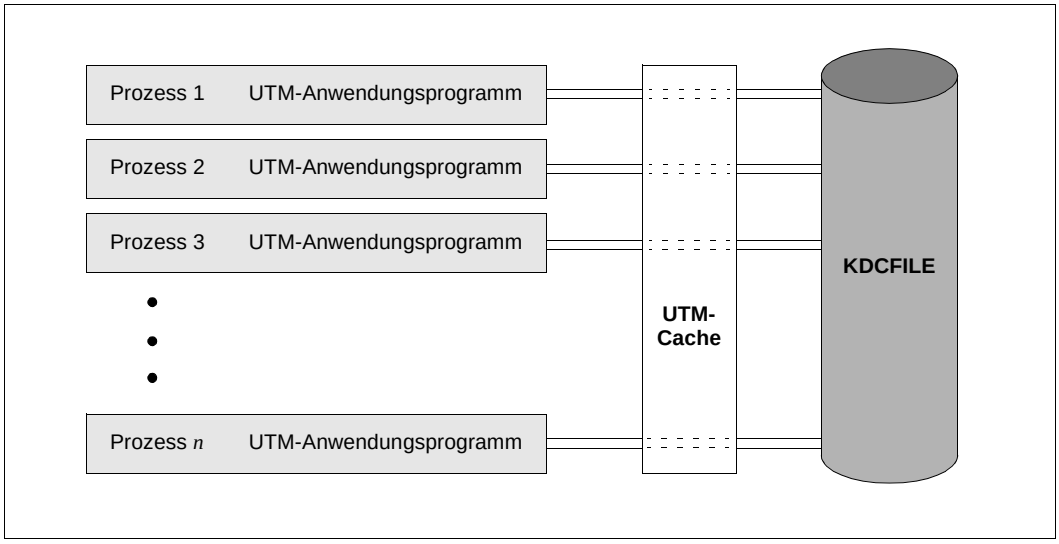


Bild 29: Struktur einer UTM-Anwendung

6.1 UTM-Anwendungsprogramm

Beim Design einer Anwendung definieren 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 die Integration von **UTM-Aufrufen** nutzen die Service-Routinen die UTM-Systemfunktionen, z.B. für Transaktionssicherung, zum Senden und Empfangen von Nachrichten etc. (siehe auch Kapitel „Programmschnittstellen“ auf Seite 131ff).

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 openUTM ablaufen können, binden Sie die übersetzten Service-Routinen, zusammen mit weiteren Modulen (Zuordnungstabellen, Meldungen, benutzte Bibliotheken etc.), zum **UTM-Anwendungsprogramm**
(siehe Seite 159). Alternativ zum statischen Binden der Service-Routinen können diese auch erst beim Start eines Prozesses der Anwendung oder beim ersten Aufruf eines Services dynamisch gebunden werden.

Einen Teil des Anwendungsprogramms bildet die **Main Routine KDCROOT**, die als steuerndes Hauptprogramm fungiert und für die Ablaufkoordination zuständig ist.

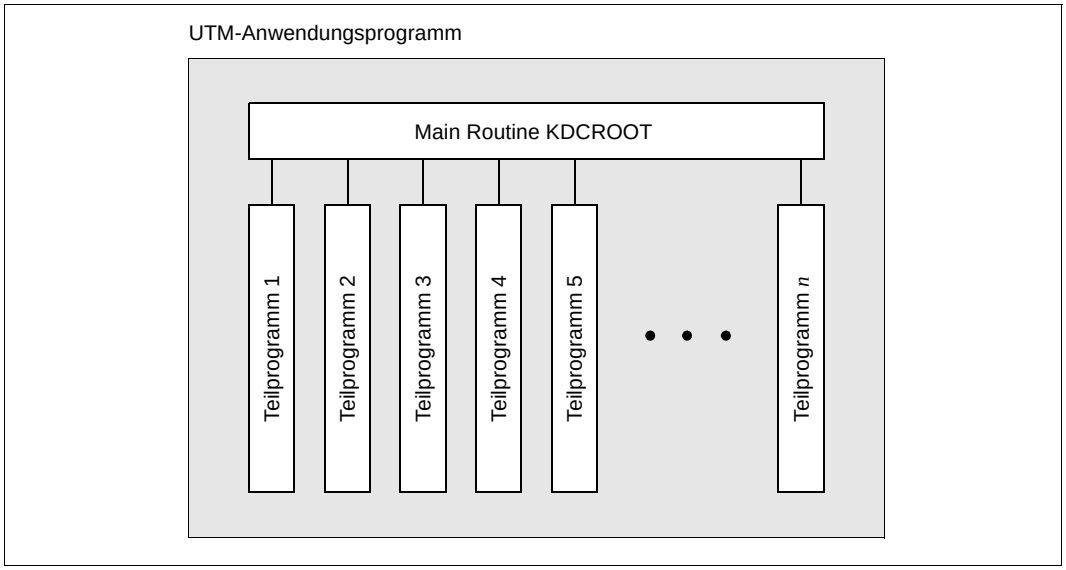


Bild 30: Struktur eines UTM-Anwendungsprogramms

6.2 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 openUTM 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 openUTM die Aufträge in eine Warteschlange ein. Umgekehrt werden, falls mehr Prozesse frei sind, als Aufträge anstehen, die freien Prozesse in eine Prozesswarteschlange eingereiht.

Zusätzlich zu der von Ihnen festgelegten Anzahl von Prozessen werden für eine UTM-Anwendung weitere Prozesse gestartet, die als **UTM-System-Prozesse** bezeichnet werden, siehe unter, Abschnitt „UTM-System-Prozesse“.

Das Konzept von openUTM, 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 openUTM im gleichen Rechner ablaufen, sind davon nicht betroffen. Der abgebrochene Prozess wird von openUTM automatisch durch einen neuen ersetzt.

UTM-System-Prozesse

Die UTM-System-Prozesse sollen Anwendungen, die unter hoher Last laufen, reaktionsfähig halten. UTM-System-Prozesse bearbeiten nur ausgewählte Aufträge, die in erster Linie durch kurze Laufzeiten gekennzeichnet sind. Ausgewählte Aufträge sind zum Beispiel Verbindungsaufbauten oder -abbauten, Timeout-Anzeigen, Transaktionsende-Anforderungen bei UTM-D und Antworten für Programme, die auf einem PGWT-Aufruf warten.

Ausgewählte Aufträge können auch die Teilprogrammläufe eines Administrators sein. Dazu kann bei der Generierung ein so genannter privilegierter LTERM-Partner angegeben werden. Für die Verbindung, die zu diesem LTERM-Partner gehört, gilt Folgendes:

- Wird für diese Verbindung ein Anmelde-Vorgang gestartet, dann wird der Anmelde-Vorgang auch von den UTM-System-Prozessen bearbeitet.
- Meldet sich ein Administrator über diese Verbindung an, dann werden Teilprogrammläufe für diese Verbindung auch von den UTM-System-Prozessen bearbeitet.
- Meldet sich ein normaler Benutzer über diese Verbindung an, dann wird diese Verbindung bis zum Abmelden dieses Benutzers ausschließlich von „normalen“ Prozessen bearbeitet.

UTM startet die UTM-System-Prozesse implizit beim Start einer Anwendung. Die Anzahl der gestarteten UTM-System-Prozesse hängt von der Angabe im Startparameter TASKS ab, siehe folgende Tabelle.

Startparameter TASKS=	Anzahl zusätzlich gestarteter UTM-System-Prozesse	Summe gestarteter Prozesse
1	0	1
2	1	3
3	2	5
4	2	6
5	3	8
n > 5	3	n + 3

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 dann sofort freigegeben und steht für andere Aufträge bereit. Wenn der Terminal-Benutzer seine Eingabe-Aktivitäten abgeschlossen hat, übernimmt - ohne dass der Benutzer dies bemerkt - u.U. ein anderer Prozess die Fortsetzung des Dialogs.

openUTM 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 openUTM-Handbuch: „Anwendungen programmieren mit KDCS für COBOL, C und C++“.

Auf einige betriebssystemspezifische Aspekte des Prozesskonzepts wird in den Kapiteln „openUTM in BS2000-Systemen“ auf Seite 231, „openUTM in Unix-Systemen“ auf Seite 247 und „openUTM in Windows-Systemen“ auf Seite 257 eingegangen.

6.3 Die KDCFILE - das „Gedächtnis der Anwendung“

Der Inhalt und die Rolle einer KDCFILE hängt davon ab, ob es sich um eine stand-alone-Anwendung (siehe unten) oder eine UTM-Cluster-Anwendung (siehe Seite 128) handelt.

6.3.1 KDCFILE einer stand-alone-Anwendung

Die KDCFILE einer stand-alone-Anwendung 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 156). 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 openUTM-Handbuch „Anwendungen generieren“.

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 Services, 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 openUTM während des Laufs der Anwendung und bei Beendigung der Anwendung auf die 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 Ausgabeaufträgen (auch zeitgesteuerten) und Hintergrundaufträgen
– Nachrichten von Service-gesteuerten Queues
– 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. openUTM sammelt Information über alle Änderungen, die innerhalb einer Transaktion anfallen.

Bei Transaktionsende bildet openUTM bei einer UTM-S-Anwendung (siehe Seite 224) 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 226) werden nur in solchen Transaktionen Datensätze für den Wiederanlauf geschrieben, in denen Passwörter geändert oder per dynamischer Konfigurierung Änderungen an den Verwaltungsdaten vorgenommen wurden.

6.3.2 KDCFILEs in einer UTM-Cluster-Anwendung

In einer UTM-Cluster-Anwendung besitzt jede Knoten-Anwendung eine eigene KDCFILE. Das es Knoten-lokale Daten und Cluster-weit gültige Daten gibt, ergeben sich im Vergleich zu einer stand-alone-Anwendung einige Besonderheiten:

- Knoten-lokale Daten werden ausschließlich in der KDCFILE der betreffenden Knoten-Anwendung gehalten. Knoten-lokale Daten sind z.B. TLS, Asynchron-Nachrichten, Hintergrund-Aufträge und Daten zu anderen knotengebundenen Vorgängen. Die KDCFILEs der einzelnen Knoten-Anwendungen sind zur Laufzeit nicht identisch.
- Cluster-weit gültige Daten werden in den UTM-Cluster-Dateien gehalten, siehe Abschnitt „UTM-Cluster-Dateien“ auf Seite 37. Dazu gehören Anwenderdaten wie z.B. GSSB, ULS, die Vorgangsdaten von Benutzern sowie Passwörter.

Das Gedächtnis einer UTM-Cluster-Anwendung besteht damit aus der Summe der lokalen KDCFILEs und den UTM-Cluster-Dateien.

Bitte beachten Sie, dass in einer UTM-Cluster-Anwendung keine doppelte KDCFILE-Führung möglich ist, d.h. die KDCFILE einer Knoten-Anwendung wird immer nur einfach geführt.

6.4 UTM-Cache-Speicher

Der UTM-eigene Cache-Speicher ist ein Bereich im virtuellen Speicher, der generierungsabhängig in Einheiten von 2KB, 4KB oder 8KB verwaltet wird. openUTM 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 openUTM, 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.

Die Nutzung des Cache-Speichers in einer laufenden Anwendung kann per Administration abgefragt werden

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 openUTM-Handbuch „Anwendungen generieren“.

7 Programmschnittstellen

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

openUTM unterstützt die Schnittstelle KDCS (nationaler Standard), die von X/Open standardisierten Schnittstellen CPI-C, XATMI und TX sowie die Schnittstelle UTM-XML:

– KDCS ist eine universale Programmschnittstelle für transaktionsorientierte Anwendungen und ist genormt nach DIN 66 265

– CPI-C und XATMI sind Schnittstellen für Programm-Programm-Kommunikation

– TX ist eine Progranmschnittstelle zur Festlegung von Transaktionen

– UTM-XML ist eine Schnittstelle zur Bearbeitung von XML-Dokumenten

7.1 Überblick über die Programmschnittstellen

Dieser Abschnitt beschreibt, welche Programmschnittstellen in welchen Programmiersprachen für Server und für Clients verfügbar sind und wie sie miteinander kombiniert werden können.

Programmschnittstellen für openUTM-Server

Die folgende Tabelle gibt einen Überblick darüber, in welchen Sprachumgebungen die vier Programmschnittstellen auf den Server-Plattformen verfügbar sind.

Server-Plattform	Verfügbare Sprachumgebungen für die Schnittstellen				
	KDCS	**CPI-C (X/Open)**	**XATMI (X/Open)**	**TX (X/Open)**	**UTM-XML (W3C)**
BS2000-Systeme	COBOL, C, C++, Assembler, Fortran, PL/I, Pascal-XT	C, C++	C, C++	C, C++	C, C++ COBOL
Unix- und Windows-Systeme	COBOL, C, C++	C, C++	C, C++	C, C++	C, C++ COBOL

TX ist eine Programmschnittstelle zur Festlegung von Transaktionen. Sie wird immer zusammen mit einer der Kommunikationsschnittstellen CPI-C (explizit) oder XATMI (implizit) verwendet und kann nicht alleine eingesetzt werden.

Die Kombination von KDCS und TX ist nicht erlaubt.

Innerhalb einer Anwendung sind folgende Kombinationen mit UTM-XML sinnvoll:

– KDCS + UTM-XML

– CPI-C + TX + UTM-XML

– XATMI + UTM-XML

Programmschnittstellen für openUTM-Clients

Die openUTM-Clients gibt es mit den Trägersystemen UPIC und OpenCPIC. Beide Trägersysteme bieten eine Reihe von Programmschnittstellen an.

Die folgende Tabelle gibt einen Überblick darüber, ob und in welchen Sprachumgebungen die einzelnen Programmschnittstellen auf den Trägersystemen verfügbar sind.

Client-Schnittstelle	Sprachumgebung für UPIC	Sprachumgebung für OpenCPIC
CPI-C (UPIC-Aufrufe)	C, C++, COBOL, MS Visual Basic, Borland Delphi,...	COBOL, C, C++
CPI-C (kompletter X/Open-Umfang)	--	
TX (X/Open)	--	COBOL, C, C++
C++-Klasse CUpic	C++, MS Visual Basic, Borland Delphi,...	--
XATMI (X/Open)	C, C++ COBOL	C, C++ COBOL
UTM-XML (W3C)	C, C++ COBOL	C, C++ COBOL

Schnittstellen-Kombinationen für die Kommunikation

Für der Kommunikation zwischen Client und Server oder zwischen zwei Servern gelten für die Schnittstellen-Kombination folgende Regeln:

- Wenn keiner der Partner die XATMI-Schnittstelle verwendet, dann ist jede Kombination der in den Tabellen aufgeführten Schnittstellen ≠XATMI erlaubt.
- Eine Kommunikation über die XATMI-Schnittstelle ist nur möglich, wenn **beide** Partner die XATMI-Schnittstelle verwenden.
- Globale Transaktionssicherung ist nur dann möglich, wenn beide Partner eine Schnittstelle mit Transaktionssicherung verwenden, z.B.:
  - KDCS mit KDCS
  - KDCS mit CPI-C + TX
  - XATMI mit XATMI

7.2 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-Systemen zusätzlich für Assembler, Fortran, PL/I und Pascal-XT.

Ausführliche Informationen über die KDCS-Programmschnittstelle finden Sie im openUTM-Handbuch „Anwendungen programmieren mit KDCS für COBOL, C und C++“. Für die in BS2000-Systemen zusätzlich unterstützten Programmiersprachen steht online jeweils ein Ergänzungsband zur Verfügung.

7.2.1 KDCS-Aufrufe

Die Funktionsaufrufe von openUTM 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 openUTM
PEND	Beenden des Programms
PGWT	Wartepunkt in einem Teilprogrammlauf setzen
RSET	Rücksetzen angeforderter Änderungen und Operationen

Nachrichtenkommunikation im Dialog:

Aufruf	**Funktion**
APRO	Adressieren eines Auftragnehmer-Vorgangs (für verteilte Verarbeitung)
CTRL	Steuern eines OSI TP-Dialogs (für verteilte Verarbeitung)
MGET	Empfangen einer Dialog-Nachricht
MPUT	Senden einer Dialog-Nachricht

Nachrichtenkommunikation über Message Queuing:

Aufruf	**Funktion**
APRO	Adressieren eines Auftragnehmer-Vorgangs (bei verteilter Verarbeitung)
DGET	Lesen einer Nachricht aus einer Service-gesteuerten Queue
DPUT	Erzeugen von zeitgesteuerten Asynchron-Aufträgen, Nachrichten an Service-gesteuerten Queues und von Quittungsaufträgen
FGET	Empfangen von Asynchron-Nachrichten
FPUT	Erzeugen von Asynchron-Aufträgen und von Nachrichten an TAC-Queues
MCOM	Definieren eines Auftrags-Komplexes
QCRE	Erzeugen einer Temporären Queue
QREL	Löschen einer Temporären Queue

Verwalten von Message Queues und Druckern:

Aufruf	**Funktion**
DADM	Administration von Nachrichten an UTM-gesteuerte und Service-gesteuerte Queues sowie Administration der Dead Letter Queue
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 openUTM 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 Eingabe-Nachricht gedrückt wurden.
- Fehler, die es noch ermöglichen, den Dialogschritt oder Vorgang sinnvoll zu beenden.
- Schwere Fehler, bei denen openUTM die Transaktion zurücksetzt und den Vorgang mit Speicherabzug beendet. Der Returncode kann in diesem Fall dem Speicherabzug entnommen werden. Wenn möglich, schickt openUTM dem Kommunikationspartner eine Fehlermeldung.

Eine Übersicht über alle Returncodes finden Sie im jeweiligen openUTM-Handbuch „Meldungen, Test und Diagnose“.

7.2.2 UTM-Speicherbereiche

openUTM 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.

Service-gesteuerte Queues können in diesem Sinne ebenfalls als UTM-Speicherbereiche betrachtet werden. Da Service-gesteuerte Queues speziell für das Message Queuing konzipiert sind, werden sie gesondert im Kapitel „Message Queuing“ ab Seite 109 beschrieben.

Primärspeicherbereiche

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

- Kommunikationsbereich (KB)

  openUTM legt den KB an, wenn ein neuer Service gestartet wird. Sein Inhalt wird dem jeweils aktuell ausgeführten Teilprogramm übergeben. Im KB stellt openUTM 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)

  openUTM 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 openUTM nicht vorgegeben, sondern frei definierbar.

Sekundärspeicherbereiche

Diese Speicherbereiche werden von openUTM 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 Services dient. Während openUTM 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.

  In UTM-Cluster-Anwendungen sind GSSBs Cluster-weit nutzbar, alle Knoten-Anwendungen haben dieselbe Sicht auf die Daten eines GSSB.

- Terminal-spezifischer 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.

  In UTM-Cluster-Anwendungen werden TLS nur Knoten-lokal unterstützt.

- User-spezifischer 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.

  In UTM-Cluster-Anwendungen sind ULS Cluster-weit nutzbar, alle Knoten-Anwendungen haben dieselbe Sicht auf die Daten eines ULS.

Benutzer-Protokolldatei (User-Logging-Datei USLOG)

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

In UTM-Cluster-Anwendungen hat jede Knoten-Anwendung ihre eigene Benutzer-Protokolldatei.

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 openUTM bereitgestellte Information; Datenaustausch zwischen Teilpro-grammen eines Service	ja
SPAB	Standard Primärer Arbeitsbereich	Start des Teilpro-gramms bis Ende des Teilprogramms	Parameterübergabe bei KDCS-Aufrufen; Nachrichtenpuffer	nein

Kurz-name	Bereichstyp	Lebensdauer	Funktion	Transaktions-sicherung
AREA	Per Generierung vereinbarter weiterer Speicher-bereich	Dauer der Anwendung	Aufnahme von anwen-dungsglobalen Daten, die vorzugsweise nur gelesen werden; In UTM-Cluster-Anwen-dungen sind AREAs nur Knoten-lokal.	nein
LSSB	Lokaler Sekundärer Arbeitsbereich	vom ersten Schreib-aufruf bis zur expliziten Freigabe oder bis Service-Ende	Datenaustausch zwischen Teilpro-grammen eines Service	ja
GSSB	Globaler Sekundärer Arbeitsbereich	vom ersten Schreib-aufruf bis zur expliziten Freigabe oder bis zum Löschen der Anwen-dungsinformation	Austausch von Daten über Service-Grenzen hinweg	ja
ULS	User-spezifischer Langzeitspeicher	Generierung bis Änderung der Generierung	z.B. Statistiken spezifisch für bestimmte Benutzer-kennungen, Sessions oder Associations	ja
TLS	Terminal-spezifi-scher Langzeit-speicher	Generierung bis Änderung der Generierung	Statistiken spezifisch für bestimmte Anschluss-punkte (LTERMs, LPAPs, OSI-LPAPs)	ja
USLOG	Benutzer-Protokolldatei (User-Logging)	individuell festlegbar	Protokollierung	ja

7.2.3 Event-Funktionen

Um auf bestimmte Ereignisse im Ablauf per Programm reagieren zu können, bietet openUTM die Möglichkeit, Event-Funktionen zu verwenden. Im Gegensatz zu „normalen“ Services, die durch Angabe eines Transaktionscodes aufgerufen werden, startet openUTM 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**

Der Einsatz von allen Event-Funktionen ist optional. Per Generierung legen Sie fest, welche Event-Funktionen in einer UTM-Anwendung aktiviert werden.

Event-Services

BADTACS ist ein Dialog-Service. Er wird von openUTM 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 openUTM 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 181). 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 openUTM gestartet, wenn sich ein Terminal-Benutzer, eine Transportsystem-Anwendung oder ein UPIC-Client an die UTM-Anwendung anschließt.

Es sind mehrere SIGNON-Services definierbar, und zwar einer pro Transportsystem-Zugangspunkt. Damit lassen sich diese Services abhängig vom Partnertyp gestalten.

Ist ein SIGNON-Service für einen Transportssystem-Zugangspunkt generiert, dann wird er für die Terminals und Transportsystem-Anwendungen, die die Verbindung zur Anwendung über diesen Zugangspunkt aufbauen, 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 Anmelde-Dialog für Ihre Anwendung individuell gestalten. Mit dem SIGNON-Service lassen sich beispielsweise zusätzlich zu den Berechtigungsprüfungen von openUTM eigene Berechtigungsprüfungen durchführen (siehe auch Seite 189), oder man kann im Anmelde-Vorgang einer Transportsystem-Anwendungen explizit eine UTM-Benutzerkennung zuweisen.

Event-Exits

START Der Event-Exit START wird von openUTM beim Start und beim Neuladen des Anwendungsprogramms in jedem Prozess aufgerufen.

Er kann z.B. verwendet werden, um eigene Dateien zu öffnen.
Es können bis zu 8 verschiedene START-Exits definiert werden.

SHUT Der Event-Exit SHUT wird von openUTM bei Beendigung des Anwendungsprogramms in jedem Prozess aufgerufen.

Er kann z.B. verwendet werden, um eigene Dateien zu schließen.
Es können bis zu 8 verschiedene SHUT-Exits definiert werden.

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 openUTM 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-Systemen stehen zusätzlich der Event-Exit FORMAT zur Verfügung, mit dem sich eine selbstprogrammierte Formatierung realisieren lässt.

7.3 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.

openUTM stellt die CPI-C-Programmschnittstelle für die Programmiersprachen COBOL, C und C++ zur Verfügung. Unter openUTM 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 openUTM 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 openUTM-Client-Programmen
– von anderen UTM-Anwendungen (Server-Server-Kommunikation)
– von Fremd-Anwendungen (z.B. von CICS-Anwendungen bei Server-Server-Kommunikation)

Die folgenden Tabellen geben eine Übersicht über die CPI-C-Aufrufe, die in openUTM 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.

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

Übersicht: CPI-C-Aufrufe in openUTM

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

7.4 Die X/Open-Schnittstelle XATMI

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

openUTM stellt die XATMI-Programmschnittstelle für die Programmiersprachen COBOL, C und C++ zur Verfügung. Unter openUTM 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 von 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 openUTM zur Verfügung stehen. Eine umfassende Beschreibung der einzelnen Aufrufe finden Sie in der X/Open CAE Specification zu XATMI vom November 1995.

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

Übersicht: XATMI-Aufrufe in openUTM

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 openUTM (gibt den Namen einer Service-Routine bekannt)

7.5 Die X/Open-Schnittstelle TX

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

openUTM stellt die TX-Programmschnittstelle für die Programmiersprachen COBOL, C und C++ zur Verfügung.

TX-Aufrufe sind unter openUTM 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.

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

Das Trägersystem OpenCPIC ermöglicht es openUTM-Client-Anwendungen, Transaktionen über TX zu steuern.

Übersicht: TX-Aufrufe in openUTM

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 openUTM 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.

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

7.6 Die XML-Schnittstelle von openUTM

XML (Extensible Markup Language) ist eine logische Auszeichnungssprache, mit der Dokumente in Elemente strukturiert werden können. XML ist von SGML abgeleitet (Standard Generalized Markup Language). Mit XML kann der Anwender eine eigene Sprachsyntax definieren, z.B mit Hilfe einer DTD (Document Type Definition) oder eines XML-Schemas (XSD).

Die XML-Schnittstelle von openUTM (kurz UTM-XML) ist eine Programmschnittstelle, mit der sich XML-Dokumente lesen, bearbeiten und neu erstellen lassen. Außerdem können XML-Dokumente gegenüber XML-Schemata validiert werden.

Diese Schnittstelle wird innerhalb von UTM-Teilprogrammen aufgerufen und steht für die Programmiersprachen C, C++ und COBOL zur Verfügung.

Als Kommunikationsschnittstelle kann jede der in diesem Kapitel beschriebenen Schnittstellen verwendet werden (KDCS, CPI-C oder XATMI). Außerdem ist UTM-XML auch in openUTM-Clients einsetzbar.

Ein einfacher Einsatzfall wie z.B. das Bearbeiten eines XML-Dokuments im KDCS-Teilprogramm sieht wie folgt aus:

1. Lesen der Daten mit MGET

   Das XML-Dokument wird in den Nachrichtenbereich eingelesen.

2. Bearbeiten der Daten über die UTM-XML-Schnittstelle

   Das XML-Dokument wird in einen Objektbaum konvertiert und die einzelnen Elemente werden bearbeitet. Nach vollständiger Bearbeitung wird das XML-Objekt wieder in ein XML-Dokument überführt.

   Für die Datentyp-Konvertierung wird eine eigene Struktur verwendet (t_value-Struktur).

3. Senden des XML-Dokuments mit MPUT

   Das bearbeitete XML-Dokument wird im Nachrichtenbereich bereitgestellt und an den Auftraggeber zurückgeschickt.

Wird ein XML-Dokument interaktiv in mehreren Dialogschritten (= Teilprogrammen) bearbeitet, dann muss der Programmierer dafür sorgen, dass der Kontext erhalten bleibt. Dazu gibt es 2 Möglichkeiten:

- Das XML-Objekt wird in ein XML-Dokument konvertiert und zwischenzeitlich in einem geeigneten UTM-Speicherbereich gesichert, z.B. KB, LSSB oder GSSB, siehe Seite 138.
- Das Programm verwendet den KDCS-Aufruf PGWT. PGWT verhindert einen Prozesswechsel und erhält den gesamten Programmkontext.

Übersicht: C/C++-Aufrufe der UTM-XML-Schnittstelle

C/C++-Aufruf	Beschreibung
KXLInitEnv	Initialisierung der UTM-XML-Umgebung
KXLFrom*datentyp*	Konvertierung von *datentyp* in die t_value-Struktur (*datentyp*= Short, Int, Long, Float, Double, Char, String, Struct, Array)
KXLTo*datentyp*	Konvertierung von der t_value-Struktur in *datentyp* (*datentyp*= Short, Int, Long, Float, Double, Char)
KXLCreateNewObj KXLWrite	Erzeugen eines XML-Objekts
KXLSetSubObject KXLSetRootNode KXLSetParentNode	Navigieren im XML-Objekt
KXLRead KXLReadNode KXLReadNextSib KXLReadChild KXLReadNextSingleNode KXLReadAttr	Lesen eines XML-Objekts
KXLConvDocToObj KXLConvObjToDoc KXLFreeObj	Konvertieren zwischen XML-Objekt und XML-Dokument, Freigeben eines Objekts
KXLGetHomeEnc KXLGetDocEnc KXLSetDocEnc KXLStringFromUTF8 KXLStringToUTF8	Zeichensatz ermitteln/setzen und Konvertieren von/in UTF8
KXLWriteNS KXLDelNS KXLReadNSList KXLSearchNS	Namensräume verwalten
KXLGetSizeOfNodeList	Größe einer Node List abfragen
KXLTSENV KXLGetLastParserError	Diagnosefunktionen
KXLConvDocToObjAndValid KXLParseSchema KXLParseSchemaFile KXLValidDoc KXLValidDocBuf KXLFreeSchema	Unterstützung der Schema-Funktionalität (konvertieren, parsen, validieren, freigeben)

COBOL-Schnittstelle

Die COBOL-Schnittstelle wird über einen Adaptermodul aufgerufen, beim Aufruf wird die gewünschte Funktion in Form von Parametern übergeben. Dazu wird ein COPY-Element verwendet, das den Parameterbereich beschreibt. Dieser Bereich muss vor dem Aufruf versorgt werden, die Parameter entsprechen den Parametern der zugehörigen C-Funktion.

Die COBOL-Schnittstelle enthält gegenüber C/C++ die zusätzliche Funktion *KXLSchemaGetRoot*, mit der der Root-Knoten des Schema-Objekts angefordert werden kann. In C/C++ kann dies direkt per Sprachmittel realisiert werden.

Weitere Informationen zur UTM-XML-Schnittstelle finden Sie im Handbuch „XML für openUTM“.

8 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.

Bild 31 auf Seite 160 gibt eine Übersicht über die Arbeitsschritte bei der Generierung.

Beim Generieren einer UTM-Cluster-Anwendung sind einige Besonderheiten zu berücksichtigen. Weitere Informationen zu UTM-Cluster-Anwendungen finden Sie im Abschnitt „UTM-Cluster-Anwendung“ auf Seite 34ff sowie im Kapitel „Hochverfügbarkeit und Lastverteilung mit UTM-Cluster-Anwendungen“ auf Seite 209ff.

Eine genaue Beschreibung der Arbeitsschritte bei der Generierung finden Sie im openUTM-Handbuch „Anwendungen generieren“.

8.1 Definieren der Konfiguration

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

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

Die Summe dieser Eigenschaften heißt Konfiguration. Die Konfigurationsinformationen werden in der KDCFILE gespeichert, die aus einer oder auch aus mehreren Dateien besteht (siehe Seite 126). 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-Source 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
CLUSTER	Globale Eigenschaften einer UTM-Cluster-Anwendung definieren
CLUSTER-NODE	Knoten-Anwendung eines Clusters definieren
CON	logische Verbindung (LU6.1) zu UTM-Partner-Anwendungen definieren
CREATE-CONTROL-STATEMENTS	aus Konfigurationsinformationen der vorhandenen KDCFILE Steueranweisungen für erneuten KDCDEF-Lauf erzeugen
EXIT	Event Exits definieren
KSET	Keyset definieren
LPAP	LPAP-Namen als logischen Anschlusspunkt für Partner-Anwendungen (LU6.1) vergeben
LSES	Sessionname für die Verteilte Verarbeitung über LU6.1 festlegen
LTAC	lokale Namen für TACs in UTM-Partner-Anwendungen vergeben
LTERM	LTERM-Partner als logischen Anschlusspunkt für Clients und Drucker definieren
MASTER-LU61-LPAP	Master-LPAP eines LU6.1-LPAP-Bündels definieren
MASTER-OSI-LPAP	Master-LPAP eines OSI-LPAP-Bündels festlegen
MAX	Namen der UTM-Anwendung und Ablaufparameter festlegen
MESSAGE	Meldungsmodule beschreiben
MSG-DEST	Benutzer-spezifische Meldungsziele definieren
OSI-CON	Logische Verbindung zur Partner-Anwendung (OSI TP) definieren
OSI-LPAP	OSI-LPAP-Namen als logischen Anschlusspunkt für Partner-Anwendungen (OSI TP) definieren
PROGRAM	Namen und Eigenschaften der Teilprogramme festlegen
PTERM	Clients und Drucker definieren
RESERVE	Plätze für die dynamische Konfigurierung reservieren
QUEUE	Tabelleneinträge für temporäre Queues definieren
ROOT	Namen für ROOT-Tabellen-Source vergeben

Anweisung	Funktion
SESCHA	Sessioneigenschaften (LU6.1) definieren
SFUNC	Sonderfunktionen der Funktionstasten festlegen
SIGNON	Anmeldeverfahren steuern
TAC	Namen und Eigenschaften von Transaktionscodes oder von TAC-Queues festlegen
TACCLASS	Prozesszahl für TAC-Klasse festlegen
TAC-PRIORITIES	Prioritäten für TAC-Klassen festlegen
TLS	Namen von TLS-Blöcken festlegen
TPOOL	Terminal-Pools definieren
TRANSFER-SYNTAX	Transfer Syntax (OSI TP) definieren
ULS	Namen von ULS-Blöcken festlegen
USER	Benutzer in die Konfiguration aufnehmen
UTMD	Anwendungsglobale Werte für Verteilte Verarbeitung festlegen
folgende Anweisungen nur für BS2000-Systeme:	
DATABASE	Datenbanksystem und Resource Manager beschreiben
DEFAULT	Standardwerte definieren
EDIT	Edit-Optionen definieren
FORMSYS	Formatierungssystem beschreiben
LOAD-MODULE	Lademodule für den Programmaustausch mit BLS beschreiben
MPOOL	Common Memory Pool beschreiben
MUX	Multiplex-Anschluss definieren
SATSEL	SAT-Protokollierung und zu protokollierende Ereignisse festlegen
TCBENTRY	Gruppe von TCB-Entries definieren
folgende Anweisungen nur für Unix-Systeme und Windows-Systeme:	
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.

8.2 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 openUTM ablaufen können, muss ein Anwendungsprogramm erzeugt werden; dazu dienen folgende Schritte:

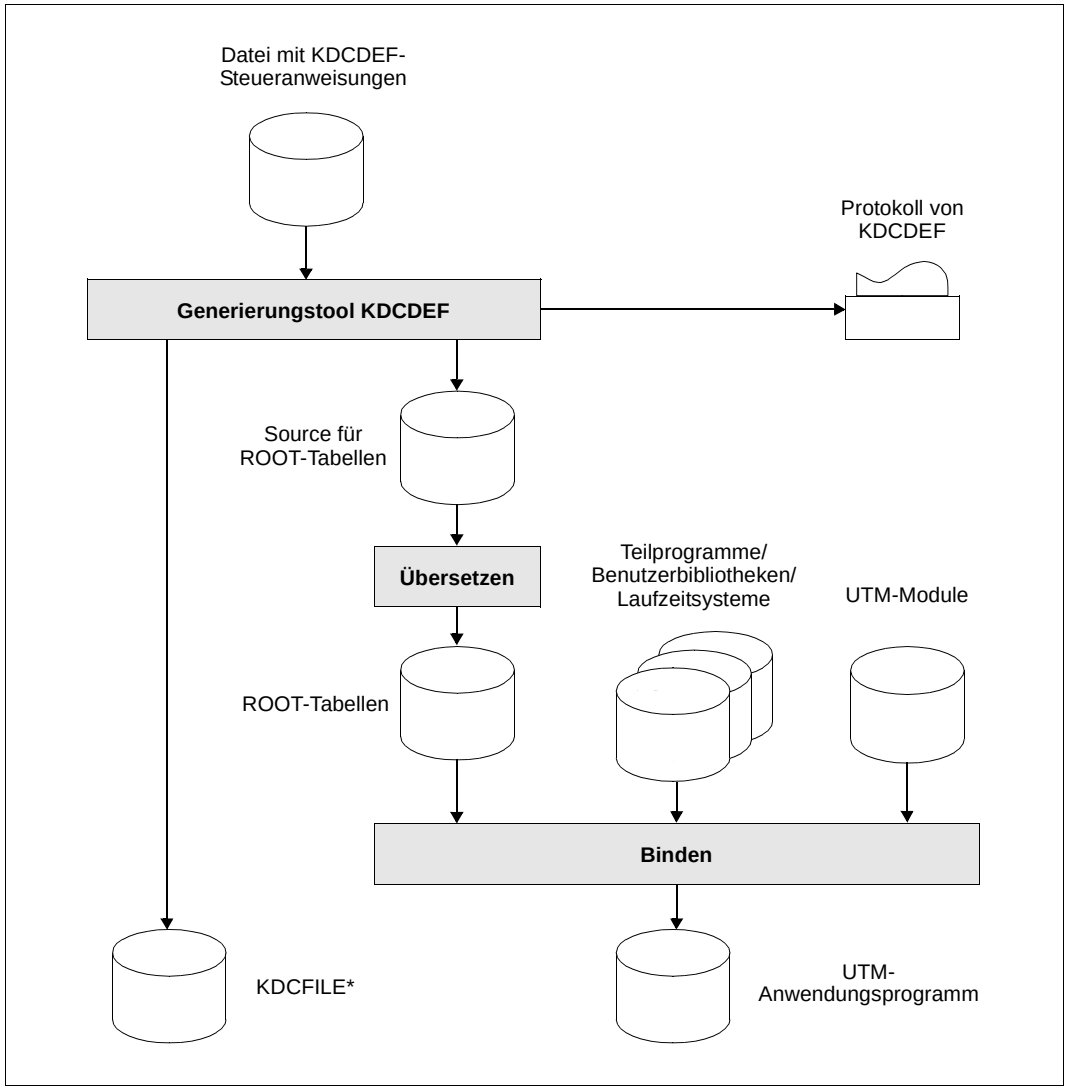
- den Source-Text für die ROOT-Tabellen, den KDCDEF erzeugt hat, übersetzen
- die übersetzten ROOT-Tabellen, Teilprogramme und eventuell weitere Module wie z.B. Format-Bibliotheken, eigene Meldungsmodule oder Sprach-spezifische Laufzeitsysteme zusammen mit UTM-Modulen zum Anwendungsprogramm binden. Dies 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 finden Sie jeweils im openUTM-Handbuch „Einsatz von openUTM-Anwendungen“.

Main Routine KDCROOT

Einen Teil des Anwendungsprogramms bildet 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 (nur BS2000-Systeme)
– Kopplung zu Datenbanken und Resource Managern



*Bei einer UTM-Cluster-Anwendung werden zusätzlich zur KDCFILE noch weitere Cluster-globale Dateien erzeugt (siehe Seite 37).

Bild 31: Generierung einer UTM-Anwendung

8.3 Konfiguration aktualisieren mit dem Tool KDCUPD

Mit dem Tool KDCUPD (**KDC**FILE **UPD**ate) können Sie nach einer Neugenerierung Ihrer UTM-Anwendung wichtige Benutzerdaten und Verwaltungsinformationen der Anwendung übernehmen:

- In stand-alone UTM-Anwendungen übernimmt KDCUPD die Daten von der alten in die neue KDCFILE.
- In UTM-Cluster-Anwendungen wird unterschieden zwischen Knoten-Update (KDCFILE einer Knoten-Anwendung aktualisieren) und Cluster-Update (UTM-Cluster-Dateien aktualisieren).

Darüber hinaus können Sie mit Hilfe von KDCUPD von einer älteren openUTM-Version zur aktuellen openUTM-Version wechseln, ohne die Daten in der KDCFILE der bisherigen Produktiv-Anwendung zu verlieren.

KDCUPD in einer stand-alone-Anwendung

In einer stand-alone-UTM-Anwendung kann KDCUPD sowohl bei UTM-S als auch bei UTM-F eingesetzt werden.

Bei der Variante UTM-S ermöglicht es KDCUPD, nach Beendigung des Anwendungslaufs die Anwenderdaten aus der bisherigen KDCFILE in eine neu erzeugte KDCFILE zu übertragen. Dabei kann KDCUPD wahlweise alle oder nur bestimmte Anwenderdaten übertragen.

Nach dem Transfer aller Anwenderdaten mit KDCUPD und dem erneuten Start der Anwendung können die Benutzer weiterarbeiten:

- unterbrochene Services werden fortgesetzt
- UTM-gesteuerte Message Queues werden weiter bearbeitet:
  - alle Asynchron-Nachrichten werden ausgegeben
  - alle Hintergrundaufträge werden ausgeführt
  - alle zeitgesteuerten Aufträge werden zur festgelegten Zeit von openUTM bearbeitet
- Alle Nachrichten in Service-gesteuerten Queues stehen zur weiteren Bearbeitung bereit.

KDCUPD in einer UTM-Cluster-Anwendung

In einer UTM-Cluster-Anwendung kann KDCUPD sowohl bei UTM-S als auch bei UTM-F eingesetzt werden. KDCUPD ermöglicht eine Knoten-Update und einen Cluster-Update.

Knoten-Update

Beim Knoten-Update überträgt KDCUPD Verwaltungs- und Anwenderdaten aus der alten KDCFILE einer Knoten-Anwendung in die neue KDCFILE einer Knoten-Anwendung. Dabei werden Knoten-lokale Daten wie z.B. TLS, Asynchron-Nachrichten oder Hintergrund-Aufträge übertragen.

Bei vielen Generierungsänderungen (z.B. neue Verbindungen zu Partner-Anwendungen kann) kann ein Knoten-Update bei laufender UTM-Cluster-Anwendung ausgeführt werden (Online-Update). Dabei wird der Knoten-Update der Reihe nach für die einzelnen Knoten durchgeführt, d.h. die Knoten-Anwendung wird beendet, die Daten werden mit KDCUPD übertragen und Knoten-Anwendung wird wieder gestartet. Die restlichen Knoten-Anwendungen laufen in dieser Zeit weiter.

Bei einigen Änderungen muss die UTM-Cluster-Anwendung beendet werden.

Cluster-Update

Beim Cluster-Update überträgt KDCUPD Verwaltungs- und Anwenderdaten aus den alten UTM-Cluster-Dateien in die neuen UTM-Cluster-Dateien. Dazu muss die UTM-Cluster-Anwendung beendet werden.

Ein Cluster-Update ist nur dann notwendig, wenn grundlegende Änderungen der Cluster-Konfiguration vorgenommen werden, z.B. wenn die Anzahl der generierten Knoten-Anwendungen oder die Anzahl der Cluster-Pagepool-Dateien erhöht wird.

Beim Cluster-Update werden Cluster-weit gültige Daten (ULS, GSSB, Vorgangsdaten, Passworte, Locale) übertragen.

Nach einem Cluster-Update muss in der Regel für jede Knoten-Anwendung ein Knoten-Update durchgeführt werden.

Weitere Informationen zu KDCUPD in einer UTM-Cluster-Anwendung finden Sie im plattformspezifischen openUTM-Handbuch „Einsatz von openUTM-Anwendungen“ unter „Änderungsgenerierung im Cluster“.

KDCFILE auf Konsistenz prüfen

KDCUPD bietet Ihnen zusätzlich eine CHECK-Option, mit der Sie die Dateien einer KDCFILE auf Konsistenz prüfen können, ohne Anwenderdaten in eine neue KDCFILE zu übertragen.

Versionswechsel mit KDCUPD

Das UTM-Tool KDCUPD lässt sich auch für den Versionswechsel nutzen. Sie können die Benutzerdaten aus der KDCFILE der Vorgängerversionen in die neu erzeugte KDCFILE von openUTM V6.3 übertragen. Mit dieser KDCFILE starten Sie die Anwendung unter openUTM V6.3, die Benutzer können danach die aktuellen Arbeiten weiterführen. So wird beispielsweise ein zeitgesteuerter Auftrag, den Sie noch mit openUTM < V6.3 abgesetzt haben, nach Versionswechsel automatisch von openUTM V6.3 angestoßen - genau zum gewünschten Zeitpunkt.

Umstieg auf 64-Bit-Umgebung mit KDCUPD

Mit KDCUPD können Informationen aus einer 32-Bit-Anwendungsumgebung in eine 64-Bit-Anwendungsumgebung übertragen werden. Dadurch können Sie einfach und ohne Datenverlust auf eine 64-Bit-Umgebung wechseln.

Weitere Informationen zu KDCUPD finden Sie im openUTM-Handbuch „Anwendungen generieren“.

9 Administrieren von UTM-Anwendungen

openUTM 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.:

- 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 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 festlegen
- logische Verbindungen zu Clients, Druckern oder fernen Server-Anwendungen auf- oder abbauen
- Protokolldateien umschalten
- Diagnosehilfen einstellen
- Betriebsdaten der UTM-Anwendung anzeigen lassen
- Aktuelle IP-Adressen ermitteln und der UTM-Anwendung zur Verfügung stellen
- RSA-Schlüssel erzeugen, auslesen oder löschen
- die UTM-Anwendung beenden
- Cache-Eigenschaften ändern

In einer UTM-Cluster-Anwendung wirkt jede dieser Aktivitäten entweder nur lokal auf der einzelnen Knoten-Anwendung oder global in allen Knoten-Anwendungen.

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 170.

Kommandoschnittstelle KDCADM

Das Standard-Administrationsprogramm KDCADM bietet eine Kommandoschnittstelle und enthält bereits vordefinierte Basis-Administrationsfunktionen, siehe Seite 167.

Grafisches Administrationsprogramm WinAdmin

Mit der openUTM-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 173.

WinAdmin deckt den gesamten über KDCADMI verfügbaren Funktionsumfang ab und bietet darüber hinaus noch weitere Funktionen.

Grafisches Administrationsprogramm WebAdmin

WebAdmin ist wie WinAdmin ein Programm zur Administration von UTM-Anwendungen mit einer grafischen Oberfläche. Im Gegensatz zu WinAdmin bietet WebAdmin jedoch eine Internet-basierte Administration. WebAdmin wird einmal zentral installiert und stellt eine Web-Anwendung zur Verfügung, an die man sich dann von einem beliebigen Web-Browser aus anmelden kann, siehe auch Seite 176.

WebAdmin deckt den gesamten über KDCADMI verfügbaren Funktionsumfang ab und bietet darüber hinaus noch weitere Funktionen. WebAdmin steht auch als Add-on auf einer Management Unit eines SE Servers zur Verfügung.

Administrieren mit CALLUTM in BS2000-Systemen

CALLUTM ist ein vielseitig einsetzbares openUTM-Client-Programm in BS2000-Systemen, 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 246.

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

9.1 Kommandoschnittstelle zur Administration

Die Basis-Administrationsfunktionen werden über vorgegebene Transaktionscodes aufgerufen, die bei der Generierung dem Standard-Administrationsprogramm (KDCADM) zugeordnet werden. Diese vorgegebenen Transaktionscodes 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. openUTM führt die angeforderten Administrationsaufgaben unmittelbar aus und gibt eine entsprechende Antwort zurück.

Mehrere Administratoren und mehrere Administrations-Clients können gleichzeitig arbeiten.

Administration über UTM-gesteuertes 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 openUTM 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; Protokoll-Dateien umschalten; in openUTM auf BS2000-Systemen Accounting-Funktionen ein- und ausschalten.
KDCBNDL	KDCBNDLA	Master-LTERMs von zwei LTERM-Bündeln austauschen.
KDCDIAG	KDCDIAGA	Diagnosehilfen anfordern: Testmodus, Trace- und KDCMON-Funktionen ein- und ausschalten, Dumps anfordern, Debug-Modus für den XA-Datenbankanschluss ein- und ausschalten. In openUTM auf BS2000-Systemen kann zusätzlich das STXIT-Logging ein- und ausgeschaltet werden.
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 Partner-Anwendungen 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, LTERM einer LTERM-Gruppe zuordnen.
KDCPOOL	KDCPOOLA	Anzahl der für einen Terminal-Pool zugelassenen Clients ändern.
KDCPROG	KDCPROGA	Lademodule des Anwendungsprogramms austauschen.
KDCPTERM	KDCPTRMA	Clients/Drucker (ent-)sperren, Verbindungen auf- und abbauen.
KDCSHUT	KDCSHUTA	Anwendung beenden. Bei UTM-Cluster-Anwendung: Wahlweise Wirkung auf Knoten-Anwendung oder ganze UTM-Cluster-Anwendung möglich.
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.

Dialog-TAC	Asynchron-TAC	Administrationsfunktion
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 openUTM auf BS2000-Systemen zur Verfügung:		
KDCMUX	KDCMUXA	Multiplex-Anschlüsse (ent-)sperren, Verbindungen auf- und abbauen.
KDCSEND	KDCSENDA	Meldungen an Terminal-Benutzer senden.

9.2 Programmschnittstelle zur Administration

Mit der Programmschnittstelle KDCADMI, die openUTM 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 unabhängig von der Plattform, auf der das Administrationsprogramm abläuft. So ist es z.B. möglich, von einer UTM-Anwendung auf einem Windows-System aus eine oder auch mehrere UTM-Anwendungen zu administrieren, die in Unix-Systemen oder in BS2000-Systemen 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 openUTM-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 Konfiguration der Anwendung „on-the-fly“ ändern können: Clients, Drucker, Benutzerkennungen, Services, usw. 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_LOCK_MGMT	Sperre der Cluster-User-Datei aufheben
KC_MODIFY_OBJECT	Eigenschaften von Objekten oder Anwendungsparametern ändern
KC_ONLINE_IMPORT	Anwendungsdaten online importieren
KC_PTC_TA	Transaktion im Zustand PTC zurücksetzen
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 openUTM auf BS2000-Systemen:	
KC_SEND_MESSAGE	Systemzeilen-Nachricht an ein oder mehrere Dialog-Terminals senden

Beispielprogramme

openUTM bietet Ihnen folgende C-Beispielprogramme, in denen die Nutzung der KDCADMI-Schnittstelle demonstriert wird:

- HNDLUSR (handle user data, nur in BS2000-Systemen verfügbar)
- SUSRMAX (show users and modify MAX values)
- ENCRADM (encryption administration)
- ADJTCLT (adjust tacclass tasks)

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- und Windows-Systemen sind die Beispielprogramme ENCRADM, SUSRMAX und ADJTCLT in die Beispielanwendung bzw. das Quick Start Kit integriert, das Beispiel COBUSER finden Sie im *utmpfad* unter *sample/src* bzw. *sample\scr*. In BS2000-Systemen sind alle Beispiele in der Bibliothek SYSLIB.UTM.063.EXAMPLE enthalten.

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

9.3 Grafisches Administrationsprogramm WinAdmin

WinAdmin ist ein grafischer Administrationsarbeitsplatz, von dem aus Sie gleichzeitig mehrere UTM-Anwendungen administrieren können.

WinAdmin basiert auf Java-Technologie und kann damit sowohl auf Windows-Systemen als auch auf Unix- und Linux-Plattformen ablaufen.

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 openUTM-Version ab:

- Für UTM-Anwendungen kann der volle Funktionsumfang der Programmschnittstelle KDCADMI genutzt werden, den die jeweilige UTM-Version bietet, siehe Abschnitt „Programmschnittstelle zur Administration“ auf Seite 170. Damit können Sie z.B. auch Objekte dynamisch in eine Konfiguration aufnehmen oder Objekte löschen, siehe auch Seite 179. Die Funktionen sind auf Seite 171 aufgelistet.

  Darüber hinaus können Sie mit WinAdmin UTM-Anwendungen starten. Dies setzt voraus, dass auf den beteiligten Rechnern openFT im Einsatz ist.

- Außerdem bietet WinAdmin folgende Funktionen, die nicht über KDCADMI zugänglich sind:

  – Definition von Meldungskollektoren. Damit ist es WinAdmin möglich, UTM-Meldungen von den laufenden UTM-Anwendungen abzufragen, anzuzeigen und zu archivieren.
  – Administration von Message Queues
  – Druckeradministration und Druckersteuerung
  – Anzeigen von GSSB-Inhalten und Löschen von GSSBs
  – Erzeugen und Löschen von Temporären Queues
  – Zusammenfassen mehrerer Administrationsschritte in einer Transaktion
  – Sehr weitgehende Unterstützung des UTM-Securitykonzeptes über Rollen und Access Lists, siehe Seite 192
  – Definition von Aktionen, z.B. das zeitgesteuerte Speichern von Objekteigenschaften in Dateien oder Reaktionen auf Schwellwertüberschreitungen bzw. Unterschreitungen.
  – Sammeln und Archivieren von Statistikdaten der UTM-Anwendungen

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 so genannten „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.

UTM-Cluster-Anwendungen administrieren

WinAdmin bietet Administrationsfunktionen, die Sie auf alle Knoten-Anwendungen in der UTM-Cluster-Anwendung global anwenden können. Außerdem können Sie mit WinAdmin z.B. auch zusammenfassende Statistiken anzeigen, die alle laufenden Knoten-Anwendungen einbeziehen.

Detaillierte Information zur Administration einer UTM-Cluster-Anwendung entnehmen Sie dem Abschnitt „UTM-Cluster-Anwendung“ auf Seite 34 sowie der „WinAdmin Online-Hilfe“.

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.

9.4 Grafisches Administrationsprogramm WebAdmin

WebAdmin ist eine web-basierte Anwendung zum Administrieren von UTM-Anwendungen auf allen Plattformen.

Zusammen mit WebAdmin wird der Web-Server Apache Tomcat ausgeliefert, in den die Web-Anwendung beim ersten Start automatisch deployt wird. WebAdmin kann sowohl als eigenständiges Tool installiert werden als auch als Add-on auf einer Management Unit eines SE Servers. In beiden Varianten steht im Wesentlichen derselbe Funktionsumfang zur Verfügung.

Nachdem Sie WebAdmin zentral installiert haben, stellt es eine Web-Anwendung zur Verfügung, auf die Sie von beliebigen Client-Rechnern zugreifen können. Auf den Client-Rechnern muss lediglich ein Web-Browser zur Verfügung stehen.

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 openUTM-Version ab:

- Für UTM-Anwendungen kann der volle Funktionsumfang der Programmschnittstelle KDCADMI genutzt werden, den die jeweilige UTM-Version bietet, siehe Abschnitt „Programmschnittstelle zur Administration“ auf Seite 170. Damit können Sie z.B. auch Objekte dynamisch in eine Konfiguration aufnehmen oder Objekte löschen, siehe auch Seite 179. Die Funktionen sind auf Seite 171 aufgelistet.

  Wenn Sie WebAdmin als eigenständiges Tool einsetzen, können Sie mit WebAdmin auch UTM-Anwendungen starten. Dies setzt voraus, dass auf den beteiligten Rechnern openFT im Einsatz ist.

- Außerdem bietet WebAdmin folgende Funktionen, die nicht über KDCADMI zugänglich sind:

  – Definition von Meldungskollektoren. Damit ist es WebAdmin möglich, UTM-Meldungen von den laufenden UTM-Anwendungen abzufragen, anzuzeigen und zu archivieren.
  – Administration von Message Queues
  – Druckeradministration und Druckersteuerung
  – Erzeugen und Löschen von Temporären Queues
  – Anzeigen von GSSB-Inhalten und Löschen von GSSBs
  – Sehr weitgehende Unterstützung des UTM-Securitykonzeptes über Rollen und Access Lists, siehe Seite 192
  – Definition von Aktionen, z.B. das zeitgesteuerte Speichern von Objekteigenschaften in Dateien oder Reaktionen auf Schwellwertüberschreitungen bzw. Unterschreitungen.
  – Sammeln und Archivieren von Statistikdaten der UTM-Anwendungen

Im Gegensatz zu WinAdmin bietet WebAdmin eine „Rund um die Uhr“ Überwachung von UTM-Anwendungen mittels Statistikkollektoren und ggf. Schwellwertaktionen. Dazu muss kein Client mit der Web-Anwendung verbunden sein. WebAdmin überprüft periodisch die Verfügbarkeit der überwachten UTM-Anwendungen.

Grafische Oberfläche

Insbesondere für komplexe Anwendungen bringt der Komfort einer web-basierten, grafischen 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 auf einer Registerkarte 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.

UTM-Cluster-Anwendungen administrieren

WebAdmin bietet Administrationsfunktionen, die Sie auf alle Knoten-Anwendungen in der UTM-Cluster-Anwendung global anwenden können.

Detaillierte Information zur Administration einer UTM-Cluster-Anwendung entnehmen Sie dem Abschnitt „UTM-Cluster-Anwendung“ auf Seite 34 sowie der „WebAdmin Online-Hilfe“.

9.5 Berechtigungskonzept

Speziell für die Administration stellt openUTM - 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, benötigt der Benutzer keine Administrationsberechtigung. 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 190).

9.6 Dynamische Erweiterung der Generierung

Sie können die mit KDCDEF erzeugte statische UTM-Generierung dynamisch ändern, ohne den Anwendungslauf zu unterbrechen. Dazu stehen Ihnen die grafischen Administrationsprogramme WinAdmin und WebAdmin zur Verfügung, mit denen Sie Objekte der Generierung modifizieren, neue aufnehmen oder löschen können. Sie können aber auch eigene Administrationsprogramme auf Basis der Programmschnittstelle KDCADMI schreiben.

Die folgende Tabelle gibt einen Überblick darüber, welche Objekttypen einer UTM-Generierung sich in der aktuellen UTM-Version erzeugen, ändern und löschen lassen:

KDCDEF-Anweisung und Operand für Objekttyp	**Bedeutung des Objekttyps**	**erzeugen**	**ändern**	**löschen**
CLUSTER	Parameter für eine UTM-Cluster-Anwendung	nein	ja	nein
CLUSTER-NODE	Adress-Komponenten und Basisnamen einer Knoten-Anwendung	nein	ja	nein
CON	LU6.1-Verbindungen	ja	nein	ja
KSET (+ LOCK-Operanden)	Zugriffsberechtigungen (Keysets) (+ Zugriffsschutz)	ja	ja	ja
MAX	Anwendungsparameter und Maximalwerte	--*	ja*	--*
LOAD-MODULE (BS2000-Systeme)	Lade-Module	nein	ja	nein
LPAP	LPAP-Partner für LU6.1	nein	ja	nein
LSES	Sessions für LU6.1	ja	ja	ja
LTAC	TACs für LU6.1-Partner und OSI TP-Partner	ja	ja	ja
LTERM	LTERM-Partner von Clients und Druckern	ja	ja	ja
MUX (BS2000-Systeme)	Multiplex-Anschluss	nein	ja	nein
OSI-CON	Verbindung für OSI TP-Partner	nein	ja	nein
OSI-LPAP	OSI-LPAP-Partner für OSI TP	nein	ja	nein
PROGRAM	Teilprogramme	ja	nein	ja
PTERM	Clients und Drucker	ja	ja	ja
SHARED-OBJECT (Unix-Systeme, Windows-Systeme)	Shared Objects	nein	ja	nein

KDCDEF-Anweisung und Operand für Objekttyp	Bedeutung des Objekttyps	erzeugen	ändern	löschen
TAC	Transaktionscode und TAC-Queues	ja	ja	ja
TACCLASS	TAC-Klassen	nein	ja	nein
TPOOL	LTERM-Pools	nein	ja	nein
USER	Benutzerkennungen	ja	ja	ja

* Das Erzeugen und Löschen der Anwendungsparameter ist nicht sinnvoll. Welche Parameter sich im Einzelnen verändern lassen, ist im openUTM-Handbuch „Anwendungen administrieren“ oder in der jeweiligen Online-Hilfe von WinAdmin bzw. WebAdmin beschrieben.

Außerdem können RSA-Schlüssel gelöscht und neu erzeugt werden. RSA-Schlüssel werden standardmäßig beim KDCDEF-Lauf erzeugt, entsprechend den Angaben bei PTERM, TAC und TPOOL.

Jede einzelne dynamische Veränderung der UTM-Generierung ist transaktionsgesichert, d.h. die Veränderung wird entweder vollständig oder gar nicht durchgeführt. Wenn Sie WinAdmin einsetzen, können Sie auch mehrere Schritte zu einer Transaktion zusammenfassen, z.B. Eigenschaften mehrerer Benutzer ändern.

Für Objekte, die dynamisch erzeugt werden sollen, muss bei der Generierung mit KDCDEF entsprechender Speicherplatz reserviert werden.

Welche Objekte und Parameter sich auf welche Weise dynamisch ändern lassen, ist im openUTM-Handbuch „Anwendungen administrieren“ unter dem Stichwort Programmschnittstelle KDCADMI und Transaktionssicherung beschrieben.

9.7 Automatische Administration

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

Dazu gibt es folgende zwei Möglichkeiten:

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

- Genauso wie über MSGTAC ist eine automatische Administration auch über die Meldungsziele USER-DEST-n (n=1,2,3,4) möglich.

  – Ordnen Sie dazu einem Meldungsziel USER-DEST-n mit der KDCDEF-Anweisung MSG-DEST einen Asynchron-TAC zu und ordnen Sie den relevanten UTM-Meldungen per Dienstprogramm KDCMMOD das Meldungsziel USER-DEST-n zu.

  – In dem Programm, das dem Asynchron-TAC zugeordnet ist, können Sie dann – wie in der MSGTAC-Routine – die Meldungen lesen und Administrationsaufrufe absetzen.

9.8 Administration von Message Queues und Druckern

openUTM bietet zur Administration von UTM-gesteuerten und Service-gesteuerten 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 Aufträgen der Queue anfordern
– Einzelne Aufträge vorziehen
– Einzelne Aufträge löschen
– Alle Aufträge in einer Queue löschen
– Nachrichten der Dead Letter Queue verschieben

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.

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

Message Queues und Drucker können auch über WinAdmin (Seite 173) oder WebAdmin (Seite 176) administriert werden.

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

10 Security-Funktionen

Ohne geeignete Security-Funktionen wären unternehmensweite, integrierte IT-Lösungen undenkbar. openUTM 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.

openUTM 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 „Berechtigungskonzept“ auf Seite 178)
- Nutzungsmöglichkeit der Security-Mechanismen externer Resource Manager

10.1 Zugangskontrolle (Identifizierung und Authentisierung)

Für die Realisierung der Zugangskontrolle bietet Ihnen openUTM 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
- Benutzerkennungen und Passwörter beim Einsatz von OSI TP-Partner-Anwendungen (Anwendungs-übergreifendes Benutzerkonzept)
- Zugangsschutz durch Ausweisprüfung beim Einsatz von Terminals
- Verwendung von Kerberos auf BS2000-Systemen beim Einsatz von Terminals
- 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 openUTM-Handbuch „Anwendungen generieren“. Die KDCADMI-Aufrufe sind im Detail im openUTM-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 Clients

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 (KDCADMI-Aufruf KC_CREATE_OBJECT).

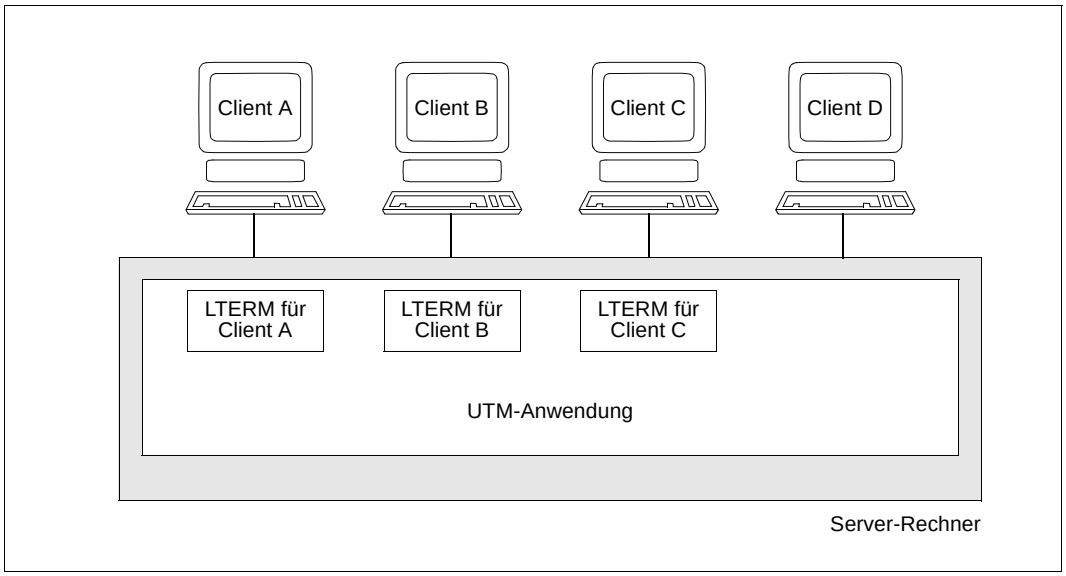


Bild 32: Anschluss von Clients über LTERM-Partner

Bei der in Bild 32 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. LTERM-Pools 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 Benutzer-spezifische 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 openUTM verschlüsselt abgespeichert, d.h. sie sind in einem Dump nicht erkennbar. Passwörter werden bei der 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 openUTM auf BS2000-Systemen nur bei dunkelgesteuertem Passwort).

Benutzerkennungen und Passwörter beim Einsatz von Client-Programmen

Das Identifizierungs- und Authentisierungskonzept von openUTM steht Ihnen auch beim Einsatz von openUTM-Client-Programmen zur Verfügung.

UPIC-Clients und OpenCPIC-Clients

UPIC- und OpenCPIC-Clients 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*.

Bei Einsatz des openUTM-Client mit Trägersystem UPIC können Sie das Passwort auch ändern (Aufruf *Set_Conversation_Security_New_Password*). Der Client kann anhand eines Returncodes feststellen, dass die Gültigkeitsdauer eines Passwortes abgelaufen ist.

Vor dem Start eines Services validiert openUTM die Berechtigungsdaten, die vom Client übergeben werden, und ordnet die jeweilige Benutzerkennung und das entsprechende Berechtigungsprofil zu (siehe Seite 190). Dies entspricht dem KDCSIGN eines Terminal-Benutzers.

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

TS-Anwendungen

Das Identifizierungs- und Authentisierungskonzept von openUTM steht Ihnen beim Einsatz von Transportsystem-Anwendungen nur dann zur Verfügung, wenn Sie für diesen Transportsystem-Zugangspunkt einen Anmelde-Vorgang definiert haben, siehe Seite 189.

Bei TS-Anwendungen gilt die Anmeldung immer für die Dauer der Verbindung. Beim Aufbau der Verbindung startet openUTM den Anmelde-Vorgang, validiert die Berechtigungsdaten, die vom Anmelde-Vorgang übergeben werden, und ordnet die jeweilige Benutzerkennung und das entsprechende Berechtigungsprofil für die Dauer der Verbindung zu.

Wenn ein Client-Programm keine Berechtigungsdaten übergibt, wird eine fest dem LTERM-Partner zugeordnete Verbindungs-Benutzerkennung angemeldet. 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 Berechtigungsprofil der Verbindungs-Benutzerkennung.

Genaue Informationen zum Security-Konzept beim Anschluss von Client-Programmen finden Sie in den openUTM-Client-Handbüchern.

Benutzerkonzept bei Server-Server-Kommunikation über OSI TP

Das Benutzerkonzept von openUTM steht Ihnen bei der Server-Server-Kommunikation über OSI TP Anwendungs-übergreifend 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 openUTM 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.

Einsatz von Kerberos mit openUTM auf BS2000-Systemen

Für Terminals ermöglicht openUTM auf BS2000-Systemen zusammen mit SECOS den Einsatz von Kerberos (RFC1510). Kerberos ist ein Netzwerk-Authentisierungsprotokoll, das am Massachussets Institute of Technology (MIT) entwickelt wurde. Es handelt sich um ein Sicherheitssystem, das auf kryptographischen Verschlüsselungsverfahren basiert. Bei einer Authentisierung mit Kerberos werden keine Kennwörter im Klartext über das Netzwerk gesendet. Dadurch wird das Abfangen von Kennwörtern im Netzwerk verhindert. Außerdem entfällt damit die Verwaltung von Kennwörtern eines Benutzers für verschiedene Anwendungen, d.h. Kerberos ermöglicht zugleich einen Single-Sign-On zu verschiedenen Anwendungen.

Kerberos arbeitet mit symmetrischer Verschlüsselung, d.h. alle Schlüssel liegen an zwei Stellen vor, beim Eigentümer eines Schlüssels (Principal) und beim KDC (Key Distribution Center).

Stiller Alarm bei wiederholten Fehlversuchen

Werden von einem Client oder einem OSI TP-Partner aus nacheinander mehrere erfolglose (fehlerhafte) Anmeldeversuche unternommen, dann erzeugt openUTM intern eine Meldung mit dem Standardziel SYSLOG (System-Protokolldatei) und wahlweise auch MSGTAC, um auf mögliche Eindringversuche hinzuweisen. Die Anmeldeversuche müssen nicht von demselben Client aus erfolgen. Erfolglose Anmeldeversuche eines Benutzers können damit auch beim Zugang über Terminal-Pool überwacht werden.

Dadurch haben Sie die Möglichkeit, in solchen Fällen gezielte Maßnahmen einzuleiten, z.B. mittels automatischer Administration (siehe Seite 181). 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 openUTM eigene Berechtigungsprüfungen durchführen. Sie können mehrere SIGNON-Services definieren und damit die Anmelde-Vorgänge Typ-spezifisch gestalten (Terminal, Clients mit Trägersystem UPIC, Transportsystem-Anwendungen, ...).

Mit openUTM 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 openUTM-Handbuch „Anwendungen programmieren mit KDCS“.

10.2 Zugriffskontrolle (Autorisierung)

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. Zusätzlich kann der Zugriff weiter eingeschränkt werden, indem sicherheitsrelevante Zugriffe nur über bestimmte LTERM-Partner (Anschlusspunkte) zugelassen werden. Deshalb bietet Ihnen openUTM die Möglichkeit, in der Konfiguration einer UTM-Anwendung die Zugriffsrechte mehrstufig differenziert festzulegen.

openUTM bietet Ihnen dazu zwei Zugriffskontrollverfahren, die gleiche Differenzierungsmöglichkeiten bieten, sich aber in der Sichtweise auf die UTM-Objekte unterscheiden:

- das benutzerorientierte Lock-/Keycode-Konzept
- das rollenorientierte Access-List-Konzept

Beide Verfahren können innerhalb einer Anwendung gemischt werden, für ein einzelnes Objekt müssen Sie sich jedoch für eine Methode entscheiden.

10.2.1 Lock-/Keycode-Konzept

Mit dem Lock-/Keycode-Konzept können Sie z.B. erreichen, dass nur besonders autorisierte Benutzer 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.

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 Benutzers ist nur möglich, wenn der angegebenen Benutzerkennung ein Keycode zugeordnet ist, der mit dem Lockcode des zugeordneten LTERM-Partners übereinstimmt.

- Ein Benutzer 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 finden Sie ein Beispiel für die Nutzung des Lock-/Keycode-Konzeptes.

Beispiel für die Nutzung des Lock-/Keycode-Konzepts

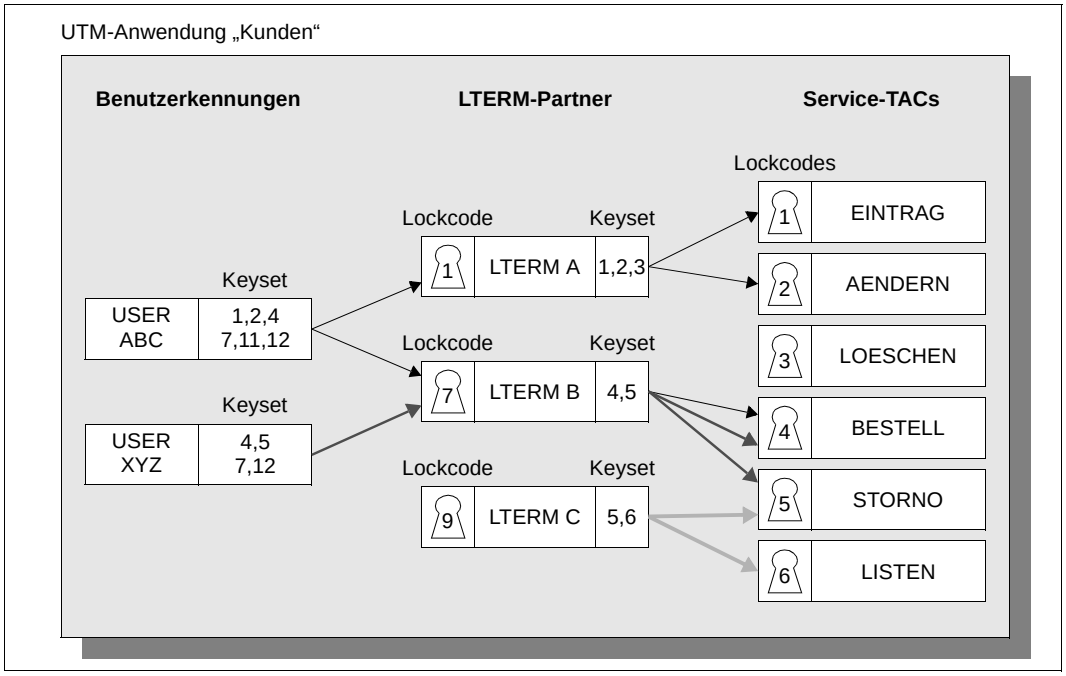


Bild 33: Zugriffsschutz mit dem Lock-/Keycode-Konzept

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.

10.2.2 Access-List-Konzept

Bei der Nutzung des **Access-List-Konzepts** werden Benutzer nach Rollen oder Funktionen im Unternehmen zusammengefasst (Pförtner, Sachbearbeiter, Personalbearbeiter, Abteilungsleiter, Administrator, Controller, Geschäftsführer,...), wobei ein Benutzer natürlich mehrere „Rollen“ haben kann. Jede „Rolle“ wird auf einen Keycode abgebildet.

- Jedem Benutzer einer UTM-Anwendung ordnet der Administrator eine oder mehrere Rollen zu (z.B. Sachbearbeiter, Abteilungsleiter, ...).
- Für die zu schützenden Objekte - Services und TAC-Queues - wird dann anhand der Access-List festgelegt, welche Benutzergruppen (Sachbearbeiter, Controller...) Zugriff haben.
- Haben Sie zum Beispiel „Personalbearbeiter“ als Rolle 2 definiert, und Geschäftsführer als Rolle 1, so können Sie festlegen, dass nur diese Benutzergruppen Zugriff auf den Service „Personal“ haben sollen, indem Sie dem Service eine Access-List zuweisen, die die Codes 1 und 2 enthält.
- Den betreffenden Benutzern wiederum wird ein Keyset zugewiesen, das alle Rollen (Keycodes) enthält, die der Benutzer wahrnehmen darf.

Wenn Sie das Administrationstool WinAdmin oder WebAdmin für die Festlegung von Access-Lists und Keysets verwenden, können Sie statt der numerischen Codes auch sprechende Rollennamen verwenden (UTM-intern werden diese symbolischen Namen dann in numerische Codes umgesetzt).

LTERM-Partner können nur mit einem Lockcode geschützt werden. Bei der Verwendung von Access-Lists sollten Sie jedoch auf den zusätzlichen Zugriffsschutz der LTERM-Partner durch Lockcodes verzichten, d.h. den Operanden LOCK der LTERM- bzw. TPOOL-Anweisung sollten Sie nicht angeben. Durch die Zuweisung eines geeigneten Keysets an den LTERM-Partner können Sie trotzdem sicherstellen, dass nur über bestimmte LTERM-Partner auf sicherheitsrelevante Daten zugegriffen wird.

Diese Variante hat zudem den Vorteil, dass die im Keyset des LTERM-Partners vorhandenen Keycodes nicht im Keyset des Benutzers vorhanden sein müssen. Damit kann zum Beispiel in einer Anwendung eine Reihe von Keycodes für den Zugang über LTERM-Partner reserviert werden, da es genügt, wenn beim Zugriff auf einen bestimmten Service in der entsprechenden Access-List eine der „Rollen“ des zugreifenden Benutzers enthalten ist und einer der Keycodes des LTERM-Partners.

Wenn Sie einen Service oder eine Queue mit einer Access-List schützen wollen, müssen Sie:

– die Access-List mit der Anweisung KSET definieren und
– die Access-List mit dem Parameter ACCESS-LIST der TAC-Anweisung dem Service oder der Queue zuweisen
– die Benutzer-spezifischen Keysets mit der Anweisung KSET definieren und
– den gewünschten Keyset mit dem Parameter KSET der Anweisung USER dem Benutzer zuweisen

Wollen Sie außerdem festlegen, dass der Zugriff auf sicherheitsrelevante Daten nur über bestimmte LTERM-Partner erfolgen darf, weisen Sie den LTERM-Partnern mit dem Operanden KSET der LTERM- oder TPOOL-Anweisung geeignete Keysets zu.

Der Zugriff auf einen Service oder eine TAC-Queue setzt dann voraus, dass sowohl für den Benutzer als auch für den LTERM-Partner, über den der Benutzer angemeldet ist, mindestens jeweils eine Rolle definiert ist, die in der Access-List des Services/der Queue enthalten ist.

Beispiel Personalverwaltung

Bei diesem Beispiel gibt es für Bearbeiter und LTERMs folgende Rollen:

1: Geschäftsführer

2: Sachbearbeiter

3: Auskunft

10: LTERM mit hoher Sicherheitsstufe

11: LTERM mit normaler Sicherheitsstufe

Die Anwendung besitzt die Services GEHALT (Gehaltsabrechnung), PERSDATA (Mitarbeiterdaten bearbeiten) und TELEFON (Telefonlisten abrufen).

Damit ergibt sich folgendes Bild:

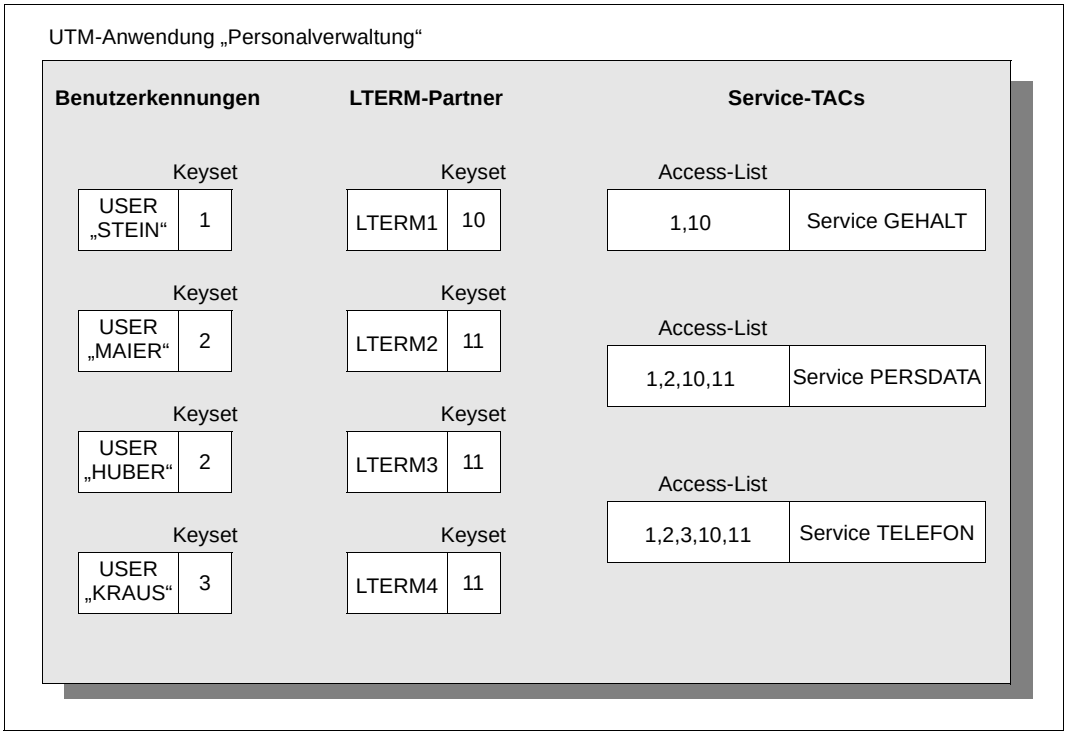


Bild 34: Zugriffsschutz mit dem Access-List-Konzept

Ein Benutzer darf einen Service nur dann starten, wenn das Keyset der Benutzerkennung und das Keyset des LTERM-Partners, über den er sich anmeldet, einen Key enthalten, der in der Access-List des Services enthalten ist:

- Benutzer STEIN ist Geschäftsführer und darf als einziger alle Services aufrufen. Für den Service GEHALT muss er sich aus Sicherheitsgründen über den Client anmelden, der dem LTERM-Partner LTERM1 zugeordnet ist.
- Die Benutzer MAIER und HUBER sind Personalsachbearbeiter und dürfen die Services PERSDATA und TELEFON aufrufen (über jedes LTERM).
- Benutzer KRAUS arbeitet in der Telefonvermittlung und darf nur auf TELEFON zugreifen.

10.3 Zugangs- und Zugriffskontrolle bei verteilter Verarbeitung

Die umfangreichen Security-Funktionen, die openUTM 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 185).

Zusätzlich können Sie bei verteilter Verarbeitung über OSI TP das UTM-Benutzerkonzept auch Anwendungs-übergreifend nutzen (siehe Seite 187).

Zugriffsschutz bei verteilter Verarbeitung

Auch bei verteilter Verarbeitung steht Ihnen das Lock-/Keycode-Konzept und das Access-List-Konzept von openUTM 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 bzw. mit Access-Lists 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 bzw. der Access-List 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 bzw. die Access-List des LTACs überwunden werden. Außerdem muss das Keyset, das die Partner-Server-Anwendung allen von der Auftrag-

geber-Anwendung eingehenden Anforderungen zuordnet, einen Keycode enthalten, der zu dem in der Partner-Server-Anwendung definierten Lockcode bzw. zu der Access-List des Services passt.

Beispiel: Lock-/Keycode-Konzept bei verteilter Verarbeitung

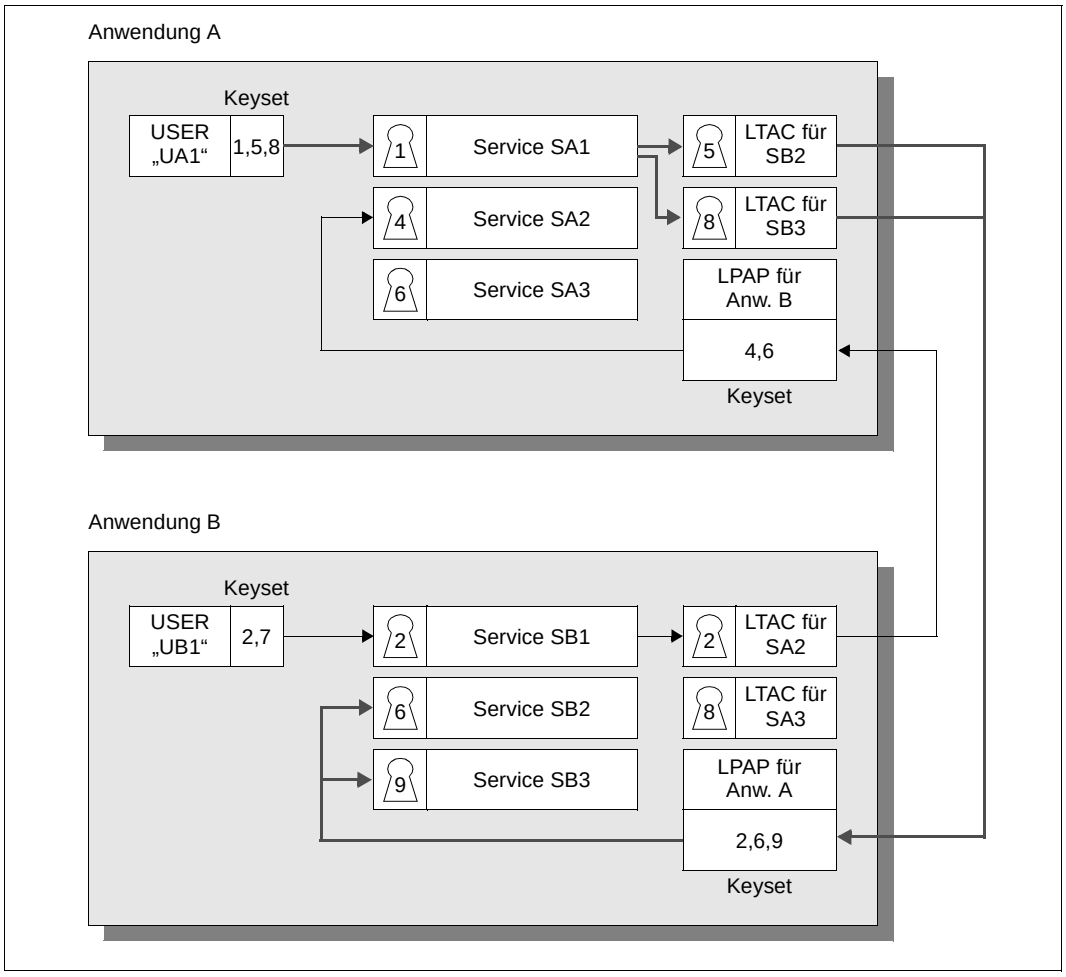


Bild 35: Zugriffsschutz mit dem Lock-/Keycode-Konzept bei verteilter Verarbeitung

In dem in Bild 35 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.

Beispiel: Access-List-Konzept bei Verteilter Verarbeitung

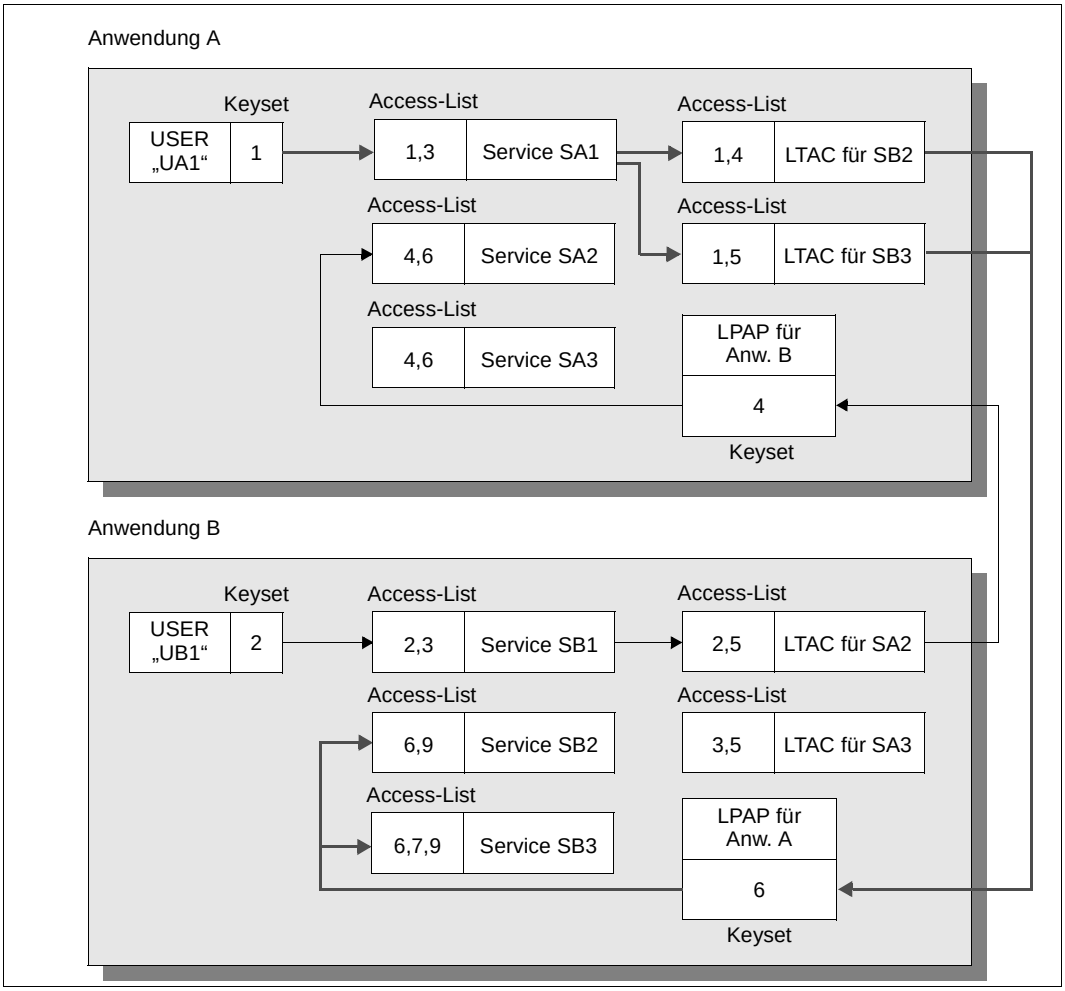


Bild 36: Zugriffsschutz mit dem Access-List-Konzept bei verteilter Verarbeitung

In dem in Bild 36 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 nicht genutzt werden.

Ein Benutzer, der sich an Anwendung A unter der Benutzerkennung „UA1“ anmeldet, verfügt über die Rolle „1“. Damit kann er in der Anwendung A den Service SA1 nutzen. Da die Rolle „1“ auch den Zugriff auf die beiden LTACs erlaubt, kann Service SA1 über diese LTACs die Dienste der fernen Services SB2 und SB3 anfordern. In der Anwendung B besitzt das LPAP für die Anwendung A den Key „6“, der dazu berechtigt, die Services SB2 und SB3 aufzurufen.

Daher sind alle Voraussetzungen erfüllt, damit der unter der Benutzerkennung UA1 in der Anwendung A gestartete Service SA1 auf die Services SB2 und SB3 der Anwendung B zugreifen darf. Entsprechendes gilt, wenn der User UB1 über den Service SB1 auf den Service SA2 in Anwendung A zugreifen möchte.

10.4 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 das zu verhindern, unterstützt openUTM die Verschlüsselung von Passwörtern und Benutzerdaten für Verbindungen zu UPIC-Clients.

openUTM verwendet zum Verschlüsseln eine Kombination aus AES-Verfahren (Advanced Encryption Standard) und RSA-Verfahren (benannt nach den Autoren Rivest, Shamir und Adleman). Der AES-Schlüssel wird vom UPIC-Client für jede Sitzung erzeugt.

Das RSA-Schlüsselpaar wird von openUTM erzeugt und besteht aus einem *öffentlichen Schlüssel* (= *public key*) und einem *privaten Schlüssel* (= *private key*). Es können in einer Anwendung mehrere Schlüsselpaare mit unterschiedlichen Schlüssellängen generiert werden. Damit lassen sich mehrere Verschlüsselungsebenen realisieren.

Aus rechtlichen Gründen werden die Verschlüsselungs-Funktionen von openUTM als eigenes Produkt openUTM-Crypt ausgeliefert, das separat installiert werden muss.

Nutzung des Verschlüsselungsverfahrens

Die Daten werden in folgenden Schritten verschlüsselt und entschlüsselt:

- Beim Verbindungsaufbau wird der öffentliche RSA-Schlüssel, der zur generierten Verschlüsselungsebene passt, an den UPIC-Client übertragen. Ist keine Verschlüsselungsebene generiert (NONE), dann wird der längste aktuell verfügbare RSA-Schlüssel übertragen.
- Der UPIC-Client erzeugt einen Verbindungs-spezifischen AES-Schlüssel. Dieser wird mit dem öffentlichen Schlüssel verschlüsselt und an die UTM-Anwendung gesendet.
- Nachdem die UTM-Anwendung den AES-Schlüssel mithilfe des RSA-Schlüssels richtig entschlüsselt hat, versendet der Client die verschlüsselten Daten.
- openUTM entschlüsselt die Daten mit dem AES-Schlüssel.
- Alle weiteren Daten, die auf dieser Verbindung zwischen Client und openUTM ausgetauscht werden, werden ebenfalls mit den AES-Schlüssel verschlüsselt.

Zusätzlich besteht die Möglichkeit, den öffentlichen RSA-Schlüssel auszulesen und auf sicherem Wege bei den UPIC-Clients lokal zu hinterlegen. In diesem Fall kann der Client beim Verbindungsaufbau überprüfen, ob der empfangene öffentliche Schlüssel mit dem lokal hinterlegten Schlüssel übereinstimmt („Man-in-the-middle“-Problem).

Ein RSA-Schlüsselpaar einer UTM-Anwendung sollte aus Sicherheitsgründen in regelmäßigen Abständen durch ein neues Paar ersetzt werden (per programmierter Administration oder über WinAdmin/WebAdmin). Per Administration können RSA-Schlüssel aktiviert,

deaktiviert und gelöscht werden. Damit kann man einen neu erzeugten RSA-Schlüssel zuerst an alle UPIC-Clients verteilen, bevor man ihn aktiviert und anschließend das alte Schlüsselpaar deaktiviert oder löscht.

Die Verschlüsselung kann bei openUTM 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 festlegen, 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. Dabei lassen sich mehrere Verschlüsselungsebenen festlegen. Der Client muss mindestens die Anforderung der jeweiligen Ebene erfüllen.
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

openUTM kann einzelne Services schützen. Ein Client darf auf solche Services nur dann zugreifen, wenn er entweder als sicher eingestuft ist oder wenn er mindestens mit der geforderten Verschlüsselungsebene verschlüsseln kann.

Möchte ein Client, der nicht als sicher gilt, auf einen derart geschützten Service zugreifen, dann muss er eine eventuelle Eingabe-Nachricht verschlüsselt senden. openUTM schickt die Ausgabe-Nachricht verschlüsselt zurück, auch dann, wenn der Client den Service ohne Eingabe-Nachricht gestartet hat oder der Service durch Vorgangskettung gestartet wurde.

Informationen über Verschlüsselung für Clients und Services finden Sie im openUTM-Handbuch „Anwendungen generieren“ unter dem Stichwort ENCRYPTION-LEVEL und im openUTM-Handbuch „Anwendungen administrieren“ unter dem Stichwort KC_ENCRYPT.

10.5 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 openUTM 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)

Die Zugangsdaten für Oracle-Datenbanken, die als XA Resource Manager an openUTM angebunden sind, können Sie wahlweise im Klartext in den Startparametern angeben oder bei der Generierung der UTM-Anwendung definieren.

11 Hochverfügbarkeit mit stand-alone UTM-Anwendungen

Eine der herausragenden Anforderungen an unternehmenskritische Anwendungen ist ein unterbrechungsfreier Betrieb, also die Verfügbarkeit der Anwendung rund um die Uhr, an 365 Tagen im Jahr. Rechnerausfälle und damit Ausfälle der Anwendung können hohe Kosten verursachen, ganz zu schweigen vom Imageschaden für den Betreiber der Anwendung.

Eine wesentliche Rolle bei der Hochverfügbarkeit spielt dabei - zusammen mit der Rechnerkonfiguration - das Betriebssystem. Eine Verfügbarkeit von 100% ist von keinem Betriebssystem erreichbar. BS2000-Systemen, die Unix-Systeme und die Windows-Systeme bieten jedoch eine Vielzahl von Funktionen, mit denen ein weitgehend unterbrechungsfreier Betrieb ermöglicht wird.

openUTM unterstützt die Hochverfügbarkeit der jeweiligen Betriebssysteme durch globale Transaktionssicherung und Wiederanlauf. Damit ergänzen sich openUTM und Betriebssystem auf optimale Art und Weise und können so einen nahezu ausfallsicheren Betrieb der Anwendung gewährleisten.

11.1 Hochverfügbarkeit in BS2000-Systemen

Zu den Hochverfügbarkeits-Funktionen, die von BS2000-Systemen 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-Systemen.

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 openUTM kann HIPLEX seine volle Wirkung entfalten, denn openUTM 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 Verlagerung der Anwendung auf einen anderen Rechner wird 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.
- HIPLEX AF basiert auf Standardprodukten und kann ohne Kunden-spezifische Programmierung oder Eingriff in die Anwendungen eingesetzt werden.

Nähere Informationen finden Sie im Handbuch „HIPLEX AF - Umschalten von Anwendungen“

11.2 Hochverfügbarkeit in Unix-Systemen

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 reibungslosen Betrieb:

– 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
– Aufteilung eines Failover-Clusters auf entfernte Rechenzentren

Diese Maßnahmen können zusammen mit openUTM ihre optimale Wirkung entfalten, denn openUTM bietet u.a. globale Transaktionssicherung und umfassende Wiederanlaufähigkeit bis zum Client. Damit ist openUTM 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.

Solaris und Linux-Systeme können z.B. mit PRIMECLUSTER RMS als Cluster eingesetzt werden. Diese Software dient dazu, Ausfälle zu erkennen, Auto-Recover durchzuführen und/oder automatisch von defekten auf intakte Systeme umzuschalten (Auto-Switchover). Eine entsprechend vorbereitete UTM-Anwendung kann den Betrieb auf dem intakten System sofort oder nach einem Restart weiterführen.

Weitere Informationen zu diesem Themenkomplex finden Sie in den Handbüchern zu PRIMECLUSTER auf Solaris bzw. PRIMECLUSTER auf Linux.

11.3 Hochverfügbarkeit in Windows-Systemen

Windows-Systeme unterstützen die Hochverfügbarkeit durch

– den 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 openUTM ihre optimale Wirkung entfalten, denn openUTM bietet u.a. globale Transaktionssicherung und umfassende Wiederanlaufähigkeit bis zum Client. Damit ist openUTM 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-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-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.
- 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.

12 Hochverfügbarkeit und Lastverteilung mit UTM-Cluster-Anwendungen

Gerade dann, wenn Anwendungen in unternehmenskritischen Bereichen eingesetzt werden, können Ausfallzeiten nicht toleriert werden. Hochverfügbarkeit der Anwendungen sowie zu Hochlastzeiten die Möglichkeit, die anfallende Last automatisch zu verteilen, haben höchste Priorität.

openUTM unterstützt die Hochverfügbarkeit durch die UTM-Cluster-Funktionalität. Außerdem bietet openUTM auch für den Cluster die Möglichkeit einer Lastverteilung über LPAP-Bündel für LU6.1- und OSI TP-Verbindungen, sowie den UPIC-Lastverteiler für UPIC-Clients.

Grundlegende Information zur Cluster-Funktion von openUTM entnehmen Sie dem Abschnitt „UTM-Cluster-Anwendung“ auf Seite 34.

12.1 Hochverfügbarkeit mit UTM-Cluster-Anwendungen

Die folgende Abbildung gibt einen Überblick über die zentralen Hochverfügbarkeitsfunktionen, die eine UTM-Cluster-Anwendung bietet:

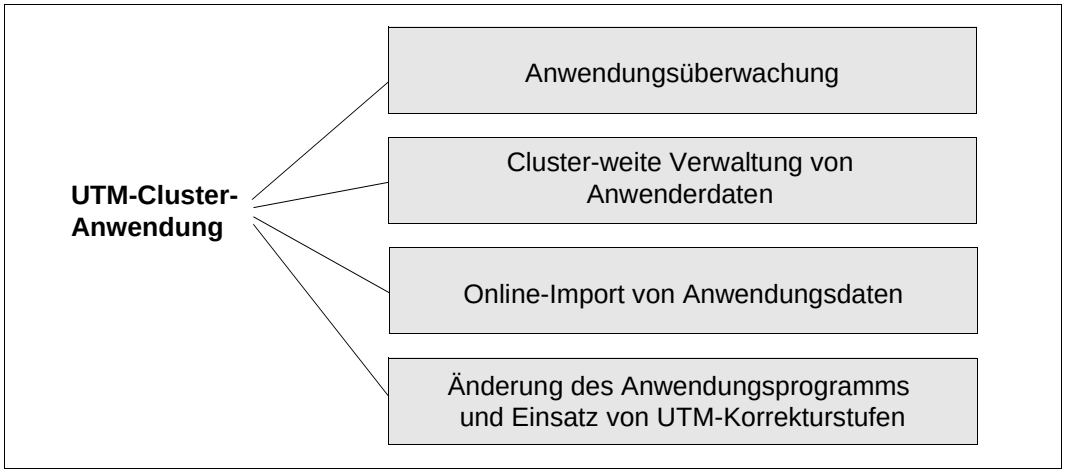


Bild 37: Zentrale Hochverfügbarkeitsfunktionen einer UTM-Cluster-Anwendung

Anwendungsüberwachung und Maßnahmen bei Ausfallerkennung

Die Überwachung der UTM-Cluster-Anwendungen ist ohne zusätzliche Software möglich. Die Knoten-Anwendungen überwachen sich gegenseitig. Bei Ausfall einer Knoten-Anwendung ist es möglich, Folgeaktionen auszulösen.

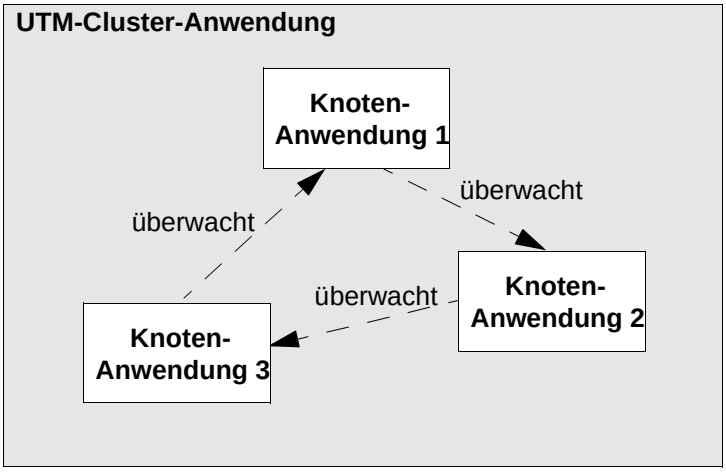


Bild 38: Ringüberwachung zwischen drei Knoten-Anwendungen

Beim Start einer Knoten-Anwendung wird dynamisch bestimmt, welche andere Knoten-Anwendung von dieser Anwendung überwacht werden soll, und welche andere Knoten-Anwendung diese Anwendung überwacht. Diese Überwachungsbeziehung wird in die Cluster-Konfigurationsdatei eingetragen. Beim Beenden der Anwendung wird diese Beziehung wieder aufgelöst. Nachdem eine Überwachungsbeziehung eingerichtet wurde, wird in einem festgelegten, generierbaren Intervall die Existenz der überwachten Knoten-Anwendung geprüft.

Überwacht wird die Verfügbarkeit einer Knoten-Anwendung in einem mehrstufigen Verfahren, wobei die jeweils nächste Stufe zum Tragen kommt, wenn die vorherige Stufe einen Ausfall der überwachten Anwendung nicht ausschließen kann:

- Die erste Stufe der Überwachung wird über den Austausch von Nachrichten realisiert.
- In BS2000-Systemen wird als zweite Stufe versucht, den Status der Knoten-Anwendung durch Zugriff auf deren Jobvariable (mit dem Basisnamen der KDCFILE) zu ermitteln.
- Als letzte Stufe wird versucht, auf die KDCFILE der überwachten Anwendung zuzugreifen, um so zu überprüfen, ob die überwachte Anwendung noch aktiv ist.

Für den Fall, dass der Ausfall einer Knoten-Anwendung erkannt wird, können Sie über eine Prozedur oder ein Skript Folgeaktionen veranlassen. Der Inhalt der Prozedur bzw. des Skripts ist abhängig von den Aktionen und der Plattform, auf der die Knoten-Anwendungen ablaufen, und wird vom Anwender festgelegt.

Detaillierte Informationen zur Anwendungsüberwachung im Cluster finden Sie in folgenden Handbüchern:

- openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter BS2000-Systemen“
- openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter Unix- und Windows-Systemen“

Cluster-weite Verwaltung von Anwendungsdaten

Bestimmte Anwendungsdaten wie Globale Speicherbereiche GSSB und ULS sowie vorgangsspezifische Daten von Benutzern werden Cluster-weit verwaltet. Das bedeutet:

- GSSB und ULS sind Cluster-weit nutzbar. Damit steht in jeder Knoten-Anwendung stets der aktuelle Inhalt der GSSB und ULS zur Verfügung. Änderungen an diesen Bereichen, die von einer Knoten-Anwendung vorgenommen werden, sind sofort in allen anderen Knoten-Anwendungen sichtbar.
- Der Vorgangs-Wiederanlauf ist knotenunabhängig. Nach dem normalen Beenden einer Knoten-Anwendung kann ein Benutzer einen offenen Dialog-Vorgang an einer anderen Knoten-Anwendung fortsetzen, sofern es sich nicht um einen knotengebundenen Vorgang handelt. Der Benutzer kann unverzüglich weiterarbeiten und muss nicht darauf warten, bis die beendete Knoten-Anwendung wieder zur Verfügung steht.

Die Anwendungsdaten werden in UTM-Cluster-Dateien verwaltet, auf die alle Knoten-Anwendungen gemeinsam zugreifen. Eine kurze Beschreibung dieser Dateien finden Sie im Abschnitt „UTM-Cluster-Dateien“ auf Seite 37.

Detaillierte Informationen zu UTM-Cluster-Dateien in UTM-Cluster-Anwendungen finden Sie im openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter BS2000-Systemen“ bzw. im openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter Unix- und Windows-Systemen“.

Besonderheiten bei der LU6.1-Kopplung

Das LU6.1-Protokoll benutzt für die Kommunikation so genannte Sessions. Die Sessions besitzen zweiteilige Namen, die jeweils in beiden Partnern bekannt sein müssen. Der Session-Name ist damit knotenspezifisch. Da in UTM-Cluster-Anwendungen alle Knoten-Anwendungen identisch generiert sind, sind in einer Knoten-Anwendung alle Session-Namen vorhanden, auch die für die anderen Knoten-Anwendungen. Damit die UTM-

Anwendung beim Verbindungsaufbau zu einer Partneranwendung eine passende Session auswählen kann, muss einer Session per Generierung zusätzlich der logische Name der Knoten-Anwendung zugeordnet werden.

Detaillierte Informationen zur Generierung einer LU6.1-Kopplung für eine UTM-Cluster-Anwendung finden Sie im openUTM-Handbuch „Anwendungen generieren“ unter „Steueranweisungen von KDCDEF - Abschnitte CLUSTER-NODE und LSES“ sowie unter „Verteilte Verarbeitung über das LU6.1-Protokoll“.

Online-Import von Anwendungsdaten

Nachdem eine Knoten-Anwendung normal beendet wurde, kann eine andere, laufende Knoten-Anwendung Nachrichten an (OSI-)LPAPs, LTERMs, Asynchron-TACs oder TAC-Queues und offene Asynchron-Vorgänge aus der beendeten Knoten-Anwendung importieren. Importierte Daten werden in der beendeten Knoten-Anwendung gelöscht.

Der Online-Import muss administrativ angestoßen werden.

Detaillierte Informationen zum Online-Import von Anwendungsdaten im Cluster finden Sie im openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter BS2000-Systemen“ bzw. im openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter Unix- und Windows-Systemen“.

Online-Update der Anwendung

Der Einsatz einer UTM-Cluster-Anwendung ermöglicht einen echten 7x24-Stunden-Betrieb. Die Änderung der (statischen) Konfiguration ist ein wichtiger Aspekt der Hochverfügbarkeit.

Während des Betriebs einer UTM-Cluster-Anwendung ist es möglich, einen neuen Stand des Anwendungsprogramms in Betrieb zu nehmen, Änderungen an der Konfiguration vorzunehmen oder eine neue UTM-Korrekturstufe einzusetzen.

Wie Sie dabei vorgehen müssen, ist im Folgenden beschrieben.

Neues Anwendungsprogramm

Wenn Sie zu einer UTM-Cluster-Anwendung neue Anwendungsprogramme hinzufügen oder bestehende Programme ändern wollen, müssen Sie dazu nicht die gesamte UTM-Cluster-Anwendung beenden.

Zusätzlich zu den Möglichkeiten des Programmaustauschs im laufenden Betrieb, z.B. Programm per Administration hinzufügen, Lademodul neu binden, Lademodul austauschen, sind im Cluster im laufenden Betrieb auch Änderungen an dem Anwendungsprogramm möglich, für die Sie die Knoten-Anwendungen nacheinander kurz beenden

müssen. Beispielsweise können Sie so einen neuen Lademodul zur Anwendung hinzufügen, oder mehr Programme hinzufügen, als reservierte Plätze zur Verfügung stehen (siehe openUTM-Handbuch „Anwendungen generieren“, RESERVE-Anweisung).

Nachdem Sie das Anwendungsprogramm hinzugefügt haben, können Sie die jeweilige Knoten-Anwendung mit dem neuen Anwendungsprogramm wieder starten.

Neue Konfiguration

Wenn Sie Änderungen an der Konfiguration einer UTM-Cluster-Anwendung vornehmen wollen, die nicht über die dynamische Administration möglich sind, dann ist es nicht in allen Fällen notwendig, die gesamte UTM-Cluster-Anwendung beenden.

Sie müssen dazu Ihre UTM-Cluster-Anwendung neu generieren. Nach dem Generieren müssen Sie die KDCFILE für jede Knoten-Anwendung übernehmen.

Eine detaillierte Beschreibung, wie Sie dabei vorgehen müssen, und welche Aktionen für die einzelnen Knoten-Anwendungen erforderlich sind, entnehmen Sie dem openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter BS2000-Systemen“ bzw. dem openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter Unix- und Windows-Systemen“.

UTM-Korrekturstufen

Sie können UTM-Korrekturstufen grundsätzlich bei laufendem Betrieb einsetzen, d.h. es kann immer ein Teil der Knoten-Anwendungen laufen, während für andere Knoten-Anwendungen die Korrekturstufe eingespielt wird.

Hierzu müssen Sie eine Knoten-Anwendung nach der anderen herunterfahren und anschließend mit der neuen Korrekturstufe wieder starten.

Detaillierte Informationen zum Online-Update einer UTM-Cluster-Anwendung finden Sie in folgenden Handbüchern:

- openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter BS2000-Systemen“
- openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter Unix- und Windows-Systemen“

Knoten-Recovery auf einem andren Knoten-Rechner

Wenn eine Knoten-Anwendung durch einen Rechnerausfall abnormal beendet wurde und nicht zeitnah auf dem ausgefallenen Rechner neu gestartet werden kann, dann kann für diese Knoten-Anwendung auf einem beliebigen anderen Knoten-Rechner des UTM-Clusters eine Knoten-Recovery durchgeführt werden.

Damit können Sie die Folgen des abnormalen Endes einer Knoten-Anwendung beheben, wie beispielsweise Sperren auf die Cluster-globalen Speicherbereiche ULS und GSSB, die zum Zeitpunkt des Abbruchs der Knoten-Anwendung gehalten wurden, aufheben, oder Benutzer, die zu diesem Zeitpunkt an der Knoten-Anwendung exklusiv angemeldet waren, abmelden.

Nach dem Knoten-Recovery sind auch Knoten- oder Cluster-Updates und Online-Import wieder möglich.

openUTM unterstützt ein Knoten-Recovery auch bei Zusammenarbeit mit Datenbanken. Voraussetzung ist, dass die jeweilige Datenbank die benötigten Funktionen bereitstellt. Details finden Sie in der Freigabemitteilung zu openUTM.

Detaillierte Informationen zu Voraussetzungen und Konfiguration eines Knoten-Recovery finden Sie in folgenden Handbüchern:

- openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter BS2000-Systemen“
- openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter Unix- und Windows-Systemen“

12.2 Lastverteilung

Für einen Cluster kann einerseits die automatische Lastverteilung bei verteilter Verarbeitung über die Protokolle LU6.1 und OSI TP genutzt werden. Andererseits besteht die Möglichkeit der Lastverteilung bei UPIC-Clients, um die anfallende Last zu verteilen.

12.2.1 Lastverteilung bei verteilter Verarbeitung

Bei der Kommunikation zwischen einer stand-alone UTM-Anwendung und einer UTM-Cluster-Anwendung über das OSI TP- oder das LU6.1-Protokoll bietet openUTM die Funktionalität der LPAP-Bündel an.

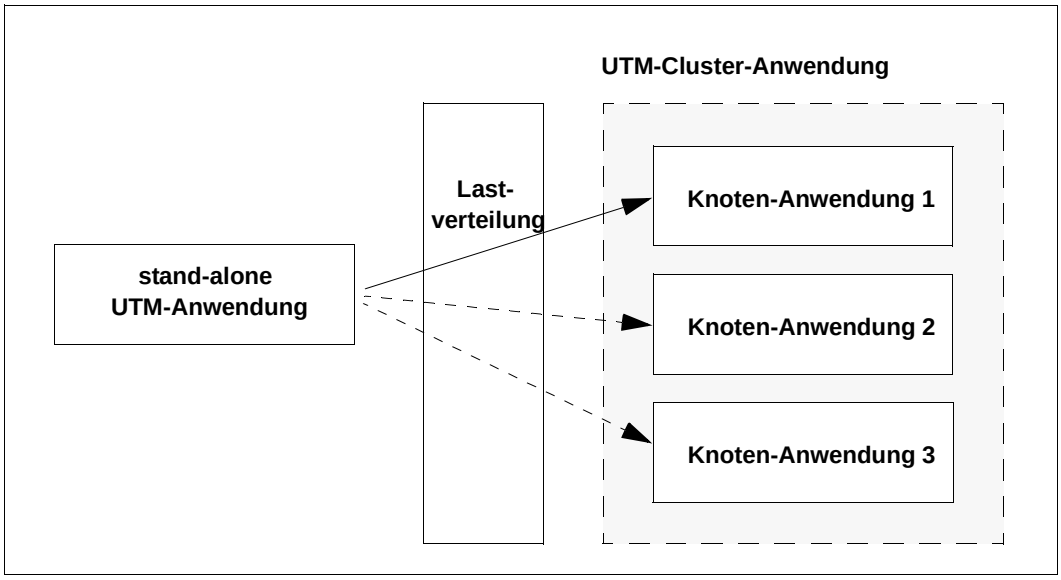


Bild 39: Kommunikation zwischen stand-alone UTM-Anwendung und UTM-Cluster-Anwendung mit Lastverteilung

Durch diese Funktionalität werden Aufträge, die von der stand-alone Anwendung an die UTM-Cluster-Anwendung gesendet werden, auf die Knoten-Anwendungen, die verfügbar sind, verteilt.

Die LPAP-Bündel müssen in der stand-alone UTM-Anwendung generiert werden.

Detaillierte Informationen zur automatischen Lastverteilung über (OSI-)LPAP-Bündel finden Sie im openUTM-Handbuch „Anwendungen generieren“.

Lastverteilung zwischen zwei UTM-Cluster-Anwendungen

Das Konzept der LPAP-Bündel lässt sich auch auf die Situation übertragen, wenn auf beiden Seiten UTM-Cluster-Anwendungen stehen. Stellt nur eine der beiden UTM-Cluster-Anwendungen Anfragen an die Gegenseite stellt, dann müssen nur in dieser UTM-Cluster-Anwendung LPAP-Bündel generiert werden. Bei gegenseitigen Anfragen werden die LPAP-Bündel auf beiden Seiten benötigt.

12.2.2 Lastverteilung bei UPIC-Clients

Aufträge von UPIC-Clients an die UTM-Cluster-Anwendung werden auf die einzelnen Knoten-Anwendungen einer UTM-Cluster-Anwendung verteilt. Aus einer Liste von Knoten-Anwendungen wird zufällig eine Knoten-Anwendung ausgewählt, mit der die nächste UPIC-Kommunikation erfolgt.

Detaillierte Informationen zur Lastverteilung bei UPIC-Clients entnehmen Sie dem Handbuch „openUTM-Client für das Trägersystem UPIC“.

12.2.3 Lastverteilung mit Oracle® RAC

Mit einer UTM-Cluster-Anwendung ist eine optimale Lastverteilung auf die Oracle® RAC-Knoten möglich.

Detaillierte Informationen zur Lastverteilung mit Oracle® RAC finden Sie im openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter BS2000-Systemen“ bzw. im openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter Unix- und Windows-Systemen“.

13 Fehlertoleranz und Wiederanlauf

openUTM schützt Ihre Daten und Anwendungen nicht nur durch ausgefeilte Zugangs- und Zugriffsschutzmechanismen, wie sie im Kapitel „Security-Funktionen“ 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 Benutzer ihre Arbeit mit konsistenten Daten fortsetzen können. Näheres siehe Abschnitt „Wiederanlauf-Funktionen von openUTM“ auf Seite 223.

Damit ist openUTM voll **cluster-fähig**, d.h. die Verarbeitung kann nach dem Wiederanlauf auf einem anderen System fortgesetzt werden, siehe Kapitel „Hochverfügbarkeit mit standalone UTM-Anwendungen“ auf Seite 205ff.

13.1 Eingrenzung von Teilprogramm- und Formatierungsfehlern

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

Dafür stellt openUTM die folgenden Funktionen zur Verfügung:

- Bei leichteren Fehlern und Störungen gibt openUTM 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 Anwendungs-übergreifend 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 auf Grund eines gravierenden Fehlers Dialog-Services abgebrochen werden, informiert openUTM den Client über den Fehler. Der Client kann danach weitere Services starten. Beim Abbruch eines Asynchron-Services 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 openUTM 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 openUTM 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 openUTM 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 openUTM überprüft, ob in einem Teilprogramm ein größerer Speicherbereich (KBPROG / SPAB) angefordert wird, als bei der Generierung angefordert.

13.2 Automatische Prüfungen

Erkennen von Benutzerfehlern

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

Wenn ein Terminal-Benutzer beim Ausfüllen eines Eingabe-Formats unzulässige Tasten verwendet, erkennt openUTM 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

openUTM führt beim Anwendungsstart Konsistenzprüfungen durch. Falls z.B. versucht wird, eine Anwendung mit Komponenten unterschiedlicher UTM-Versionen zu starten, oder falls die Konfigurationsdateien (KDCFILE und ggf. UTM-Cluster-Dateien) nicht zusammenpassen oder in sich inkonsistent sind, bricht openUTM 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 openUTM 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.

13.3 Schutz bei Störungen oder Ausfall lokaler Betriebsmittel

openUTM gewährleistet, dass selbst beim Ausfall lokaler Betriebsmittel keine Daten verloren gehen. openUTM bietet z.B. Schutz bei den in den folgenden Abschnitten geschilderten Problemen.

Betriebssystemfehler

Meldet eine Betriebssystemfunktion einen Fehler an openUTM, dann wird dieser Fehler wie ein interner Fehler von openUTM behandelt, d.h. openUTM 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 auf Grund 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 openUTM, um das Drucken der Nachrichten zu überwachen. Falls eine angeforderte Abdruckquittung nicht in einem festgelegten Zeitraum eintrifft, baut openUTM die Verbindung zu diesem Drucker ab.

In allen diesen Fällen versucht openUTM, 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 openUTM 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. openUTM gibt in diesen Fällen die letzte Ausgabe-Nachricht nochmals aus, so dass der Terminal-Benutzer seine Eingabe wiederholen kann. Wenn der Terminal-Benutzer gestörte Ausgabe-Nachrichten erkennt (z.B. Schmierzeichen), kann er sich die letzte Ausgabe nochmals ausgeben lassen (Benutzerkommando KDCDISP).

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

13.4 Abnormale Beendigung einer UTM-Anwendung

Wie in den vorangegangenen Abschnitten beschrieben, wird in bestimmten Ausnahmesituationen der Lauf einer UTM-Anwendung durch openUTM abgebrochen, um die Sicherheit von Programmen und Daten nicht zu gefährden (abnormale Beendigung). Beim nächsten Start führt openUTM automatisch einen Wiederanlauf durch (siehe Abschnitt „Wiederanlauf-Funktionen von openUTM“ auf Seite 223).

Wenn sich eine UTM-Anwendung abnormal beendet, führt openUTM - soweit möglich - vorher noch folgende Aktionen durch:

- 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.

13.5 Wiederanlauf-Funktionen von openUTM

Die Wiederanlauf-Funktionen von openUTM 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 openUTM während des Anwendungslaufs Sätze mit Wiederanlauf-Informationen in die KDCFILE (Transaktions-Logging). openUTM bietet hierzu zwei Varianten, die sich in ihrem Wiederanlaufverhalten unterscheiden.

13.5.1 Die Varianten UTM-S und UTM-F

openUTM 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 openUTM-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:

In stand-alone-Anwendungen sichert UTM-F z.B. keine Benutzerdaten für den folgenden Wiederanlauf (Anwendungs-Warmstart). In UTM-Cluster-Anwendungen sichert UTM-F Benutzerdaten erst beim Abmelden des Benutzers.

Wie sich die beiden Varianten beim Wiederanlauf verhalten, ist in den folgenden Abschnitten genauer beschrieben.

13.5.2 Wiederanlauf mit UTM-S

Wird nach abnormaler Beendigung einer UTM-S-Anwendung (UTM-Secure) die Anwendung erneut gestartet (= Warmstart), stellt openUTM 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 openUTM während des Betriebs einer UTM-S-Anwendung bei *jeder* Transaktion einen Satz mit Wiederanlauf-Informationen in die KDCFILE.

Konsistenz bei unterbrochenen Transaktionen

openUTM 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 openUTM 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 openUTM auch die UTM-Transaktion zurück (Rollback).
  - Waren die Transaktionen der Resource Manager bereits **beendet**, so beendet auch openUTM die zum Zeitpunkt des Ausfalls offenen UTM-Transaktionen (Commit).

  Entsprechend verfährt openUTM 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 Anwendungswiederanlauf unter bestimmten Umständen fortgesetzt (Vorgangswiederanlauf):

- Wenn ein Benutzer einen offenen Vorgang hat und sich durch einen Anmelde-Vorgang angemeldet hat, dann entscheidet der Anmelde-Vorgang, ob ein Vorgangswiederlauf durchgeführt wird oder der Vorgang abnormal beendet wird.
- Wenn ein Benutzer einen offenen Vorgang hat und sich über ein Client-Programm angemeldet hat, muss ein Vorgangswiederlauf explizit durch das Client-Programm angefordert werden, ansonsten beendet openUTM den Vorgang abnormal.

- Wenn ein Benutzer einen offenen Vorgang hat und sich über ein Terminal anmeldet, führt openUTM immer einen Vorgangswiederlauf durch.

Das Verhalten von openUTM ist unabhängig davon, ob die letzte Aktion mit einem Sicherungspunkt abgeschlossen wurde oder nicht. Bei einem Vorgangswiederanlauf wird immer auf den letzten Sicherungspunkt zurückgesetzt.

Detaillierte Informationen zum Wiederanlauf beim Anschluss von Client-Programmen finden Sie in den openUTM-Client-Handbüchern.

Wiederanlauf bei Hintergrund- und Ausgabeaufträgen

Der transaktionsorientierte Queuing-Mechanismus von openUTM gewährleistet, dass auch bei einem Systemausfall alle Hintergrund- und Ausgabeaufträge erhalten bleiben, die openUTM zur Bearbeitung angenommen und bis dahin noch nicht vollständig abgearbeitet hat.

Beim Wiederanlauf geht openUTM differenziert vor:

- Hintergrundaufträge, mit deren Bearbeitung noch nicht begonnen wurde oder die noch keinen Sicherungspunkt erreicht hatten, werden nach dem Wiederanlauf neu gestartet.
- Bei Hintergrundaufträgen, die bereits einen Sicherungspunkt erreicht hatten, wird die Bearbeitung an diesem Sicherungspunkt fortgesetzt.
- Alle Ausgabeaufträge, mit deren Bearbeitung noch nicht begonnen wurde oder die zum Abbruchzeitpunkt noch nicht vollständig abgearbeitet waren, stehen nach dem Wiederanlauf erneut zur Ausgabe an.
- Alle Nachrichten aus Service-gesteuerten Queues, die noch nicht in beendeten Transaktionen gelesen wurden, stehen nach dem Wiederanlauf erneut zum Lesen bereit.

13.5.3 Wiederanlauf mit UTM-F

Beim Betrieb einer UTM-F-Anwendung sichert openUTM 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 openUTM-Handbuch „Anwendungen administrieren“. Diese Änderungen stehen also auch bei einer UTM-F-Anwendung nach einem Neustart zur Verfügung.

Dagegen werden bei UTM-F

- in stand-alone-Anwendungen 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 Hintergrundaufträge.
- in UTM-Cluster-Anwendungen die Vorgangsdaten von Benutzern erst beim Abmelden im Cluster-Pagepool gesichert. War z.B. ein Benutzer bei einem abnormalen Ende einer Knoten-Anwendung noch an die Knoten-Anwendung angemeldet, dann werden seine Benutzerdaten „vergessen“. GSSB und ULS hingegen werden bei UTM-F-Cluster-Anwendungen bei Transaktionsende in UTM-Cluster-Dateien gesichert - im Gegensatz zu stand-alone-Anwendungen.

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.

13.6 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

openUTM 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 Partner-Anwendung 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.

13.6.1 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 openUTM grundsätzlich mit globaler Transaktionssicherung, während bei OSI TP die globale Transaktionssicherung nur gilt, wenn die Funktionseinheit „Commit“ gewählt ist.

Rücksetzen von verteilten Transaktionen

openUTM kann eine verteilte Transaktion aus verschiedenen Gründen zurücksetzen. Dazu zählen z.B. Verstöße gegen Programmierregeln, Verbindungsverlust, Zeitüberschreitung (Timeout) bei der Überwachung der Antwortzeit, Beendigung einer Anwendung oder auch bestimmte Situationen nach einem Failover einer Datenbank bei offenen Transaktionen, siehe Seite 33.

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 openUTM-Handbuch „Anwendungen programmieren mit KDCS“.

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. openUTM 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 openUTM, 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.

13.6.2 Rollback und Wiederanlauf bei unabhängigen Transaktionen

Mit unabhängigen, lokalen Transaktionen arbeitet openUTM 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

  openUTM 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

  openUTM setzt die Transaktionen im Auftraggeber- und im 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.

14 openUTM in BS2000-Systemen

In diesem Kapitel werden einige Plattform-spezifische Details beschrieben, die sich speziell auf den Einsatz von openUTM in BS2000-Systemen beziehen:

- Systemeinbettung
- UTM-Prozesse
- Adressraumkonzept
- Formatierung
- BS2000-spezifische Funktionen

14.1 Systemeinbettung

Die Architektur von openUTM in BS2000-Systemen ist abgestimmt auf die Architektur des BS2000-Betriebssystems und des Kommunikationssystems openNetworking.

UTM-Systemcode

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

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

Der Systemcode von openUTM 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 openUTM sind Anpassmodule für mehrere BS2000-Betriebssystemversionen verfügbar: Beim Start des Subsystems wird das passende Modul geladen. Mit dieser Technik ist eine openUTM-Version in mehreren BS2000-Betriebssystemversionen ablauffähig. Dies ermöglicht es Ihnen, mit Ihren UTM-Anwendungen ohne Umstellungen in eine höhere BS2000-Betriebssystemversion zu migrieren.

Existieren in einem BS2000-System mehrere UTM-Anwendungen, ist der UTM-Systemcode nur einmal im BS2000-System geladen. Der gesamte UTM-Systemcode wird als ein Subsystem des BS2000-Betriebssystems geladen. Der Systemverwalter legt durch Anweisungen für das Subsystem openUTM fest, zu welchem Zeitpunkt der UTM-Systemcode geladen werden soll. Werden weitere UTM-Anwendungen im BS2000-System gestartet, verwenden diese ebenfalls den zuvor einmal geladenen Systemcode. Jede UTM-Anwendung wird allerdings von openUTM 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 sowie zum etwaigen Starten des Subsystems UTM-SM2
- Datenverwaltungssystem mit den Zugriffsmethoden UPAM und SAM
- BCAM zur Kommunikation mit Anwendungen und Kommunikationspartnern
- VTSU zur Aufbereitung der Nachrichten von und an Terminals
- SOC-TP, wenn openUTM mit Socket-Partnern kommunizieren soll
- RSO (Remote SPOOL Output) zur Unterstützung von RSO-Druckern (siehe Seite 241)
- OSS und CMX bei verteilter Verarbeitung über OSI TP
- SAT zur Protokollierung sicherheitsrelevanter UTM-Ereignisse
- MSCF zum synchronisierten, Rechner-übergreifenden Zugriff auf Cluster-globale Dateien (für UTM-Cluster-Anwendungen)

Ablauffähigkeit auf x86-Hardware-Plattform

openUTM V6.3 ist im vollen Umfang auf den X86-basierten Business Servern mit der Systemlinie SQ und auf den Server Units x86 des SE Servers ablauffähig. Nähere Informationen zur Installation von openUTM auf dieser Hardware finden Sie im Anhang des openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter BS2000-Systemen“.

Paralleler Betrieb mehrerer openUTM-Versionen

Es ist möglich, im selben BS2000-System mehrere openUTM-Versionen nebeneinander zu laden und gleichzeitig einzusetzen.

Diese Funktion ist besonders beim Übergang auf eine neue openUTM-Version von Vorteil. Sie können dann während der Nutzung einer openUTM-Version einzelne UTM-Anwendungen versuchsweise mit der openUTM-Folgeversion ablaufen lassen. Die Umstellung mehrerer UTM-Anwendungen von einer openUTM-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 oder UDS).
XA	Schnittstelle zum Anschluss externer Resource Manager wie z.B. Oracle. Die XA-Schnittstelle ist ein X/Open-Standard.
IUTMFORM	Als Schnittstelle zum Formatierungssystem FHS.
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 openUTM, also über die X/Open-Schnittstellen CPI-C und XATMI oder über die Schnittstelle KDCS (nationaler Standard).

UTM-Module und Dienstprogramme als LLMs

Folgende Bestandteile von openUTM werden als LLMs ausgeliefert:

- UTM-Systemmodule
- UTM-ROOT-Code
- Administrationsprogramm
- alle Dienstprogramme
- KDCUPD-Module

Die LLMs ermöglichen es, im Fehlerfall für einzelne Module Source-Korrekturen zu liefern, sodass Sie diese Module selbst austauschen können.

Die UTM-Dienstprogramme werden beim Aufruf aus einer Bibliothek nachgeladen.

Nähere Informationen zu UTM-Modulen und Dienstprogrammen finden Sie im openUTM-Handbuch „Anwendungen generieren“ und im openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter BS2000-Systemen“.

Übersicht: Schnittstellen von openUTM

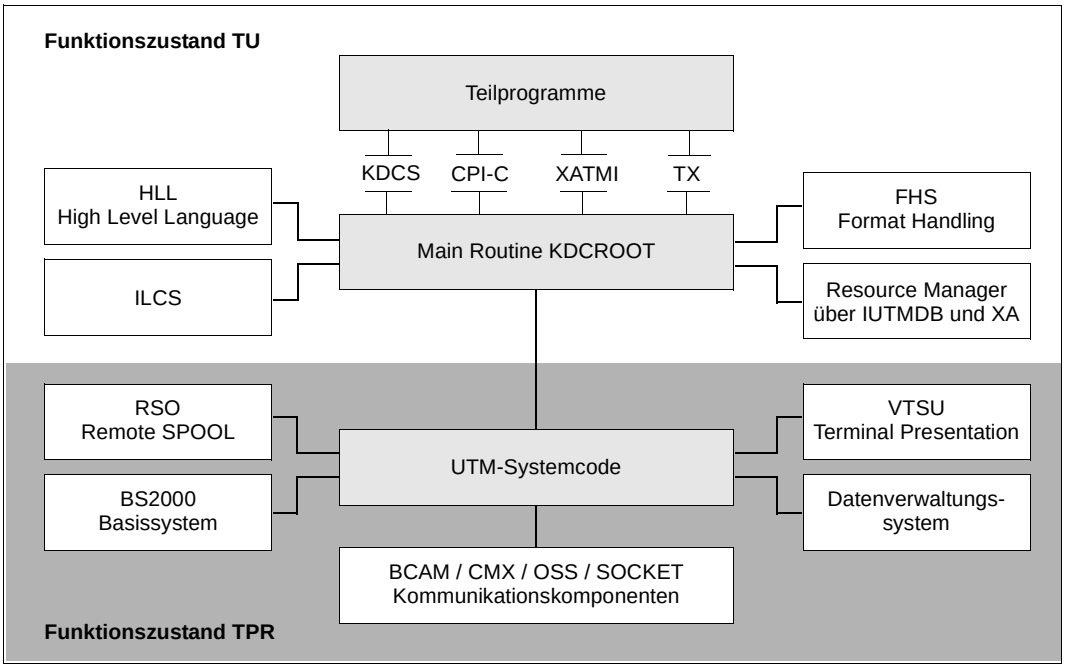


Bild 40: Schnittstellen von openUTM zu anderen Systemkomponenten

14.2 UTM-Prozesse

Für jede UTM-Anwendung, die im BS2000-System abläuft, gibt es eine homogene Prozess-Familie, d.h. alle Prozesse der Anwendung sind gleich ausgestattet und jeder Prozess kann jede Aufgabe übernehmen. Die Prozesse sind als BS2000-Tasks realisiert.

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 so viele 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 Prozesse 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.

14.3 Adressraumkonzept

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

- Der UTM-Systemcode liegt im Klasse-4-Speicher und wird von allen Prozessen aller UTM-Anwendungen eines BS2000-Systems gemeinsam benutzt.
- 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 Anwendungs-spezifisch 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. nicht-privilegierten 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/SQL und SESAM/SQL.
– 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 im BS2000-System.

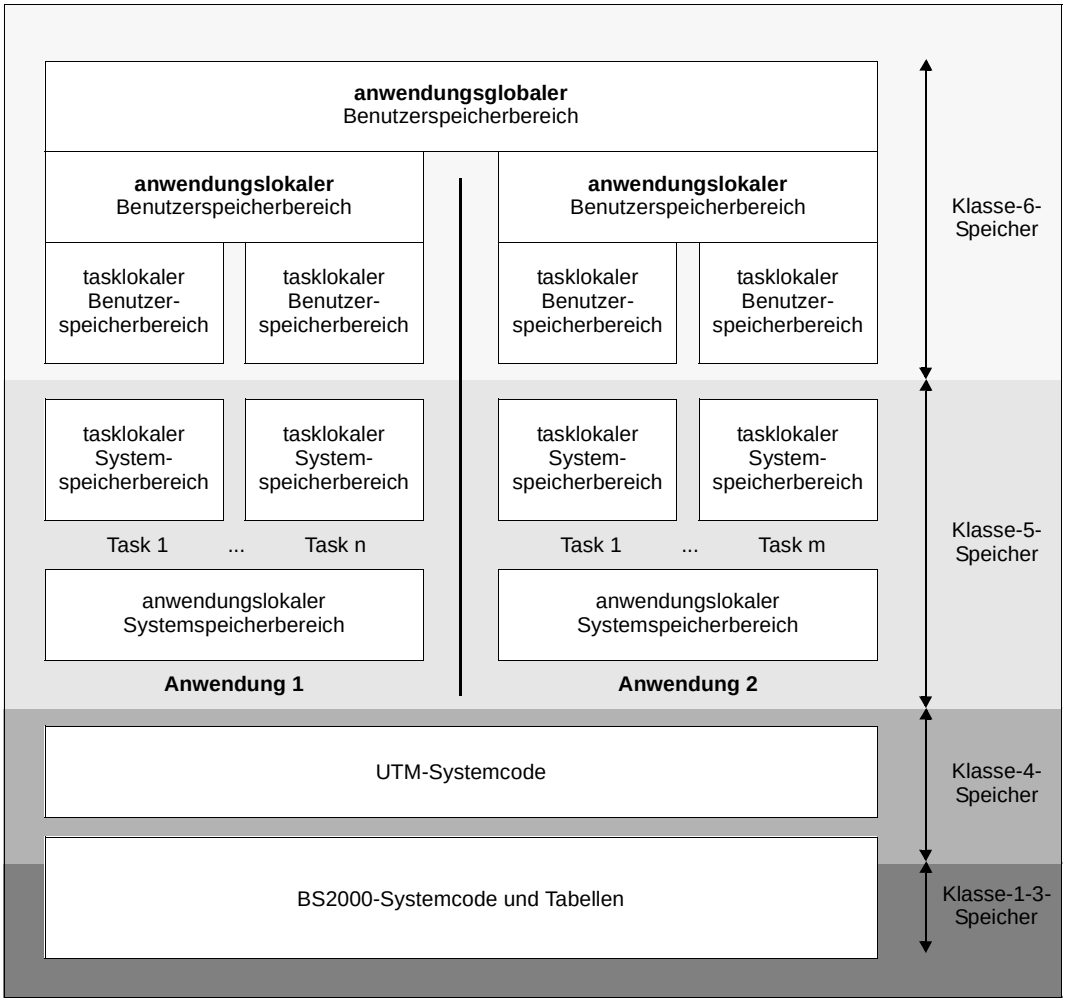


Bild 41: Speicherstruktur von UTM-Anwendungen in BS2000-Systemen

14.4 Formatierung

Falls Sie für Ihre UTM-Anwendung Terminals im Formatmodus einsetzen wollen, können Sie entweder die BS2000-Softwareprodukte IFG (Interaktiver Formatgenerator) und FHS (Format Handling System) nutzen oder eine eigene Formatierungsroutine erstellen (Event-Exit FORMAT).

openUTM auf BS2000-Systemen unterscheidet *Formate, +Formate, #Formate und -Formate mit jeweils unterschiedlichen Funktionen und Einsatzmöglichkeiten. Ein Format wird beim Senden und Empfangen einer Nachricht durch das Formatkennzeichen spezifiziert. Das Formatkennzeichen besteht aus dem Formattyp als erstem Zeichen und dem Formatnamen. Für die Formatierung von *Formaten, +Formaten und #Formaten verwendet openUTM 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. Die Datenstrukturen können auch für Daten im Unicode-Format erstellt werden. 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
- Verarbeitung von Daten in 7-Bit-Code, 8-Bit-Code oder Unicode

Für die Kommunikation mit dem Formatierungssystem FHS verwendet openUTM die neutrale Schnittstelle IUTMFORM.

IUTMFORM bietet folgende Vorteile:

- Die Produkte openUTM und FHS werden durch diese Schnittstelle entkoppelt. Dadurch können Sie z.B. die Funktionen einer neuen FHS-Version bei gleichbleibender openUTM-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-Systemen nur die Formatsteuerung FHS unterstützt.

Über Produkte wie WIN-DOORS und FHS-DOORS lassen sich Bildschirm-Formate dynamisch in grafische Oberflächen umsetzen und so z.B. auch in Microsoft-Office-Umgebungen integrieren. Mit WebTransactions können Sie FHS-Formate auch in HTML umwandeln und damit in Web-Oberflächen einbetten.

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 openUTM standardmäßig eingesetzte Formatierungssystem FHS, sowohl physische Eingabe-Nachrichten verarbeiten, als auch physische Ausgabe-Nachrichten - auch für Bildschirmwiederanlauf - erzeugen können. Eine eigene Formatierungsroutine ist in folgenden Fällen sinnvoll:

- wenn Sie Funktionen benötigen, die über den von openUTM 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 openUTM-Handbuch „Anwendungen programmieren mit KDCS“ beschrieben.

14.5 Codeumsetzung

Socket-Anwendungen schicken ihre Nachrichten meist im ASCII- bzw. ISO 8859-1-Code, während in BS2000-Systemen der EBCDIC-Code verwendet wird. Daher bietet openUTM in BS2000-Systemen eine automatische Codeumsetzung für Socket-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 openUTM-Handbuch „Anwendungen generieren“ bei den Anweisungen PTERM und TPOOL sowie im Anhang.

14.6 BS2000-spezifische Funktionen

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 openUTM 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 auf BS2000-Systemen können auch RSO-Drucker nutzen: Über die OLTP-Schnittstelle von RSO (Remote Spool Output) erhält openUTM Zugang zu allen Druckern, die RSO unterstützt, d.h. auch zu Druckern, die über LAN oder PC angeschlossen sind. Zu diesen Druckern baut openUTM keine Transportverbindung auf, sondern bedient sie über die OLTP-Schnittstelle von RSO, d.h. openUTM 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. Zusätzlich können Druckoptionen in Form einer Parameterliste an den Drucker übergeben werden.

Zu dem vom BS2000-System zur Verfügung gestellten Software-Produkt RSO gibt es eine eigene Handbuch-Reihe. Was Sie bei der UTM-Generierung beachten müssen, ist im openUTM-Handbuch „Anwendungen generieren“ 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 openUTM die Verbindung zu einem Drucker erst dann aufgebaut, wenn die Anzahl der zum Druck anstehenden Nachrichten einen Schwellwert überschreitet, welcher Drucker-spezifisch 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-Systems zuordnen. Diese Run Priorität wird dem UTM-Prozess zugeordnet, in dem das Teilprogramm abläuft. So können Sie die Scheduling Mechanismen des BS2000-Systems zur Ablaufsteuerung von UTM-Teilprogrammen einsetzen.

Spezifische Security-Funktionen

SAT-Protokollierung

Mit der BS2000-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 im BS2000-System 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.

Unterstützung von Kerberos und Zugangskontrolle durch Kerberos

Beim Verbindungsaufbau von Terminals kann im BS2000-System ein Kerberos-Dialog durchgeführt werden, dessen Ergebnis im Teilprogramm gelesen werden kann. Diese Funktionalität kann kann mit den Anweisungen LTERM und TPOOL generiert werden.

Die Zugangskontrolle mit dem verteilten Authentifizierungsdienst Kerberos kann mit der Anweisung USER generiert werden.

Zugangsschutz durch Zertifikatsprüfung

Für UTM-Anwendungen auf BS2000-Systemen können Benutzerkennungen so generiert werden, dass der Zugang zu einer UTM-Anwendung über diese Benutzerkennung nur mit Hilfe eines Zertifikats möglich ist. Ein Zertifikat wird von einer autorisierten Zertifizierungsstelle vergeben. Es besteht aus der Zertifikatsnummer sowie der Nummer der Zertifizierungsstelle und wird auf einer Chipkarte gespeichert. Diese Chipkarte verwendet der Benutzer, wenn er sich über einen Client an die UTM-Anwendung anmeldet.

Die Zertifikatsinformationen werden beim Anmelden an die UTM-Anwendung übergeben und mit den generierten Daten verglichen. Der Benutzer kann sich nur dann erfolgreich anmelden, wenn die übergebenen Daten mit den generierten Daten übereinstimmen.

Datenbank-Schlüssel

Für UTM-Anwendungen auf BS2000-Systemen kann bei der Generierung einem Transaktionscode ein Datenbank-Schlüssel zugeordnet werden, falls sie die IUTMDB-Schnittstelle verwenden, siehe Seite 233. Damit besteht die Möglichkeit, dem TAC bestimmte Zugriffsrechte auf der Datenbankseite zuzuordnen. openUTM 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 auf BS2000-Systemen kann die Verschlüsselung auch für Terminal-Emulationen genutzt werden. Dabei wird die Verschlüsselung auf der Host-Seite durch das BS2000-Produkt VTSU-B durchgeführt. Somit kann jede Emulation mit Verschlüsselung betrieben werden, sofern sie die entsprechenden Funktionen unterstützt.

Nähere Informationen zu den genannten auf BS2000-spezifischen Security-Funktionen finden Sie im openUTM-Handbuch „Anwendungen generieren“.

Internationalisierung / XHCS-Unterstützung

openUTM 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-Systemen 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-Systemen die Möglichkeit, mehrere Meldungsmodule für eine UTM-Anwendung zu generieren und so Mehrsprachigkeit auch bei den UTM-Meldungen zu realisieren.

Damit bietet openUTM auf BS2000-Systemen Internationalisierung in vollem Umfang. In Unix-Systemen 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 Zeichensätze benötigt (8-Bit-Zeichensätze oder Unicode-Zeichensätze wie UTFE). Mit Hilfe des BS2000-Softwareprodukts XHCS (eXtended Host Code Support) können in einem BS2000-System gleichzeitig mehrere Zeichensätze verwendet werden. openUTM 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. Insbesondere können Unicode-Daten im Anwendungsprogramm verarbeitet und FHS-Formate mit Unicode-Daten eingelesen und ausgegeben werden.

Nähere Informationen zu XHCS finden Sie im Benutzerhandbuch „XHCS V2.0 - 8-bit-Code- und Unicode-Unterstützung im BS2000/OSD“. Die UTM-spezifischen Aspekte der Internationalisierung sind im openUTM-Handbuch „Anwendungen generieren“ beschrieben. Details zum Einsatz von mehreren Meldungsmodulen enthält das openUTM-Handbuch „Meldungen, Test und Diagnose in BS2000-Systemen“.

Einsatz des Session Managers OMNIS

Für UTM-Anwendungen auf BS2000-Systemen lassen sich die Dienste des BS2000-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 openUTM auf BS2000-Systemen 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.

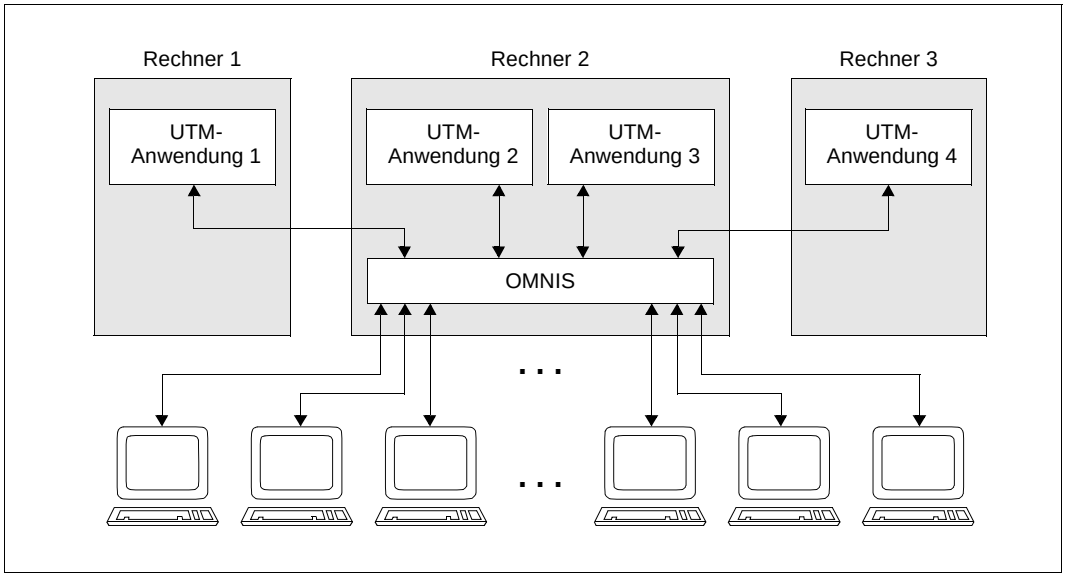


Bild 42: Nachrichtenverteilung und Multiplexing mit OMNIS

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 openUTM-Handbuch „Anwendungen generieren“.

Aufruf von UTM-Services mit CALLUTM

Mit openUTM auf BS2000-Systemen wird das Programm CALLUTM ausgeliefert, das es erlaubt, aus einer beliebigen BS2000-Batch- oder Dialog-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-System 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 openUTM-Handbuch „Anwendungen administrieren“.

SDF-Schnittstelle für UTM-Tools

UTM-Tools, wie beispielsweise KDCDEF, können mit eigenen SDF-Kommandos gestartet werden. Die Kommandos sind im SDF-Anwendungsbereich UTM abgelegt.

Eine Beschreibung dieser SDF-Kommandos finden Sie im openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter BS2000-Systemen“.

15 openUTM in Unix-Systemen

In diesem Kapitel werden einige Plattform-spezifische Details beschrieben, die sich speziell auf den Einsatz von openUTM in Unix-Systemen beziehen:

- Systemeinbettung
- UTM-Prozesse
- Adressraumkonzept
- Vereinfachtes Konfigurieren der Netzanbindung
- Ablauf auf 64-Bit-Systemen

Wenn im Folgenden allgemein von Unix-System die Rede ist, dann ist darunter sowohl ein Unix-basiertes Betriebssystem wie Solaris, HP-UX oder AIX als auch eine Linux-Distribution wie SUSE oder Red Hat zu verstehen.

15.1 Systemeinbettung

Um größtmögliche Portabilität zu erreichen, nutzt der UTM-Systemcode nur die Systemaufrufe und Bibliotheksfunktionen, die Unix-Systeme im Rahmen des X/Open-Universums (System V) zur Verfügung stellen:

- 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. Oracle, INFORMIX)
- UPIC-L-Schnittstelle, die es ermöglicht, openUTM-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 zur Kommunikationskomponente PCMX

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

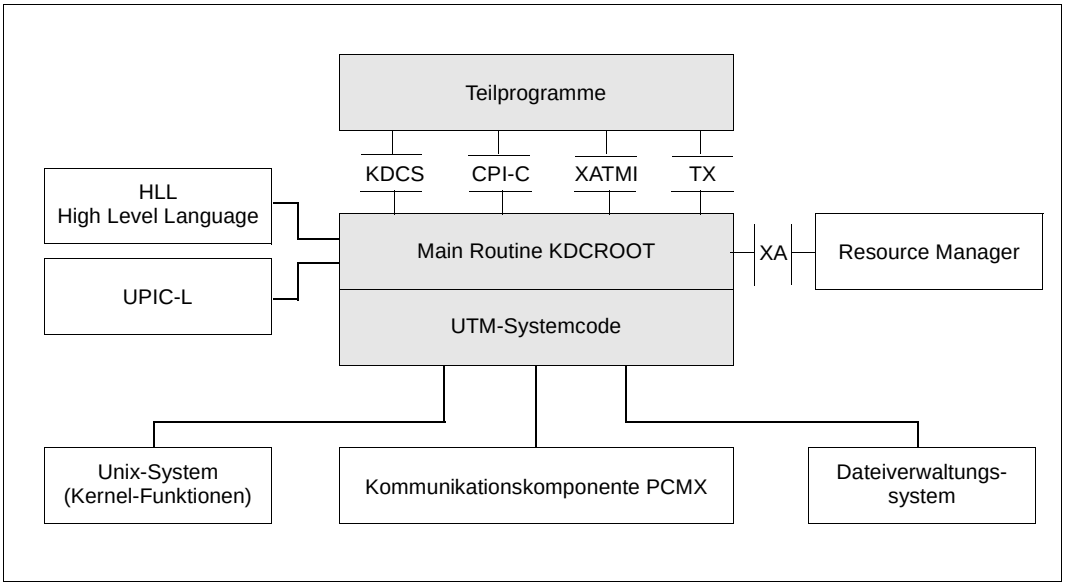


Bild 43: Schnittstellen von openUTM zu anderen Systemkomponenten

15.2 UTM-Prozesse

Bei der Ausführung eines UTM-Anwendungsprogramms, das unter einem Unix-System abläuft, arbeiten unterschiedliche Prozesse mit jeweils spezifischen Aufgaben zusammen. Diese Prozesstypen werden in den folgenden Absätzen beschrieben.
Eine Übersicht gibt Bild 44 auf Seite 252.

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 Anwendungs-bezogen; 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.

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 PCMX oder direkt über die Socket-Schnittstelle laufen. Die Anzahl der Netzprozesse ist generierungsabhängig.

Weitere Informationen über Netzprozesse sowie über Details der Generierung erhalten Sie im openUTM-Handbuch „Anwendungen generieren“.

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 openUTM-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.

Logging-Prozess

openUTM kann bei laufender Anwendung bestimmte Daten protokollieren wie z.B. Abrechnugssätze oder Messdaten. Diese Protokollierung wird über den **Logging-Prozess** gesteuert (*utmlog*).

- Falls generiert, stellt openUTM während des Anwendungslaufes Accounting-Informationen bereit. Diese Information wird in sogenannten Abrechnungssätzen von openUTM erfasst und an den Logging-Prozess weitergeleitet. Der Logging-Prozess schreibt diese Sätze in eine Datei im Unterverzeichnis ACCNT.
- Zur Leistungskontrolle einer UTM-Anwendung steht die in openUTM integrierte Funktion des UTM-Messmonitors KDCMON zur Verfügung. Nach Starten der KDCMON-Messung erfassen die Workprozesse Messdaten und übermitteln diese an den Logging-Prozess, der die Datensätze in eine Datei im Unterverzeichnis KDCMON schreibt. Dadurch wird die Verwaltung dieser Daten von den Workprozessen auf eine einzige Instanz, den Logging-Prozess, verlagert.

Übersicht: Prozesse einer UTM-Anwendung in Unix-Systemen

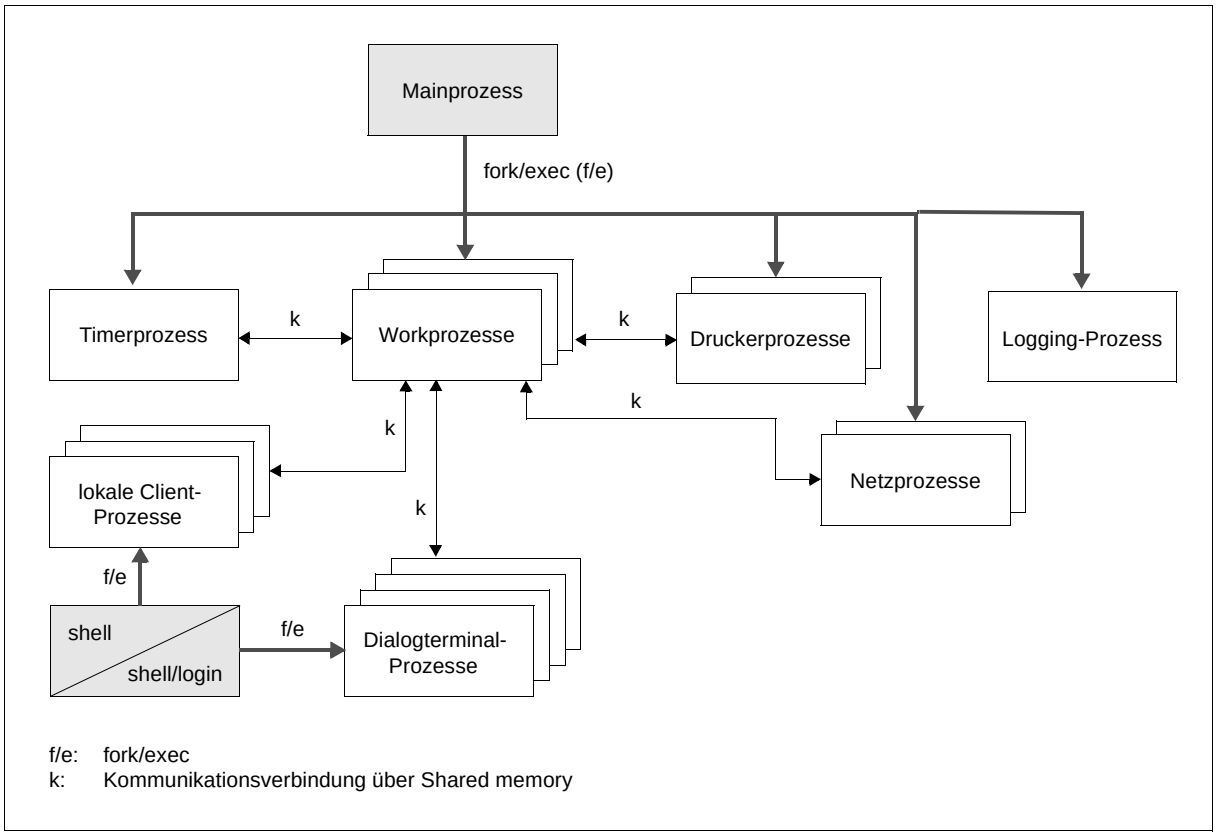


Bild 44: Prozessinteraktion in einer UTM-Anwendung auf Unix-Systemen

15.3 Adressraumkonzept

In einer UTM-Anwendung verfügt jeder Workprozess über einen Prozess-spezifischen 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 Prozess-spezifische 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 Unix-System 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.

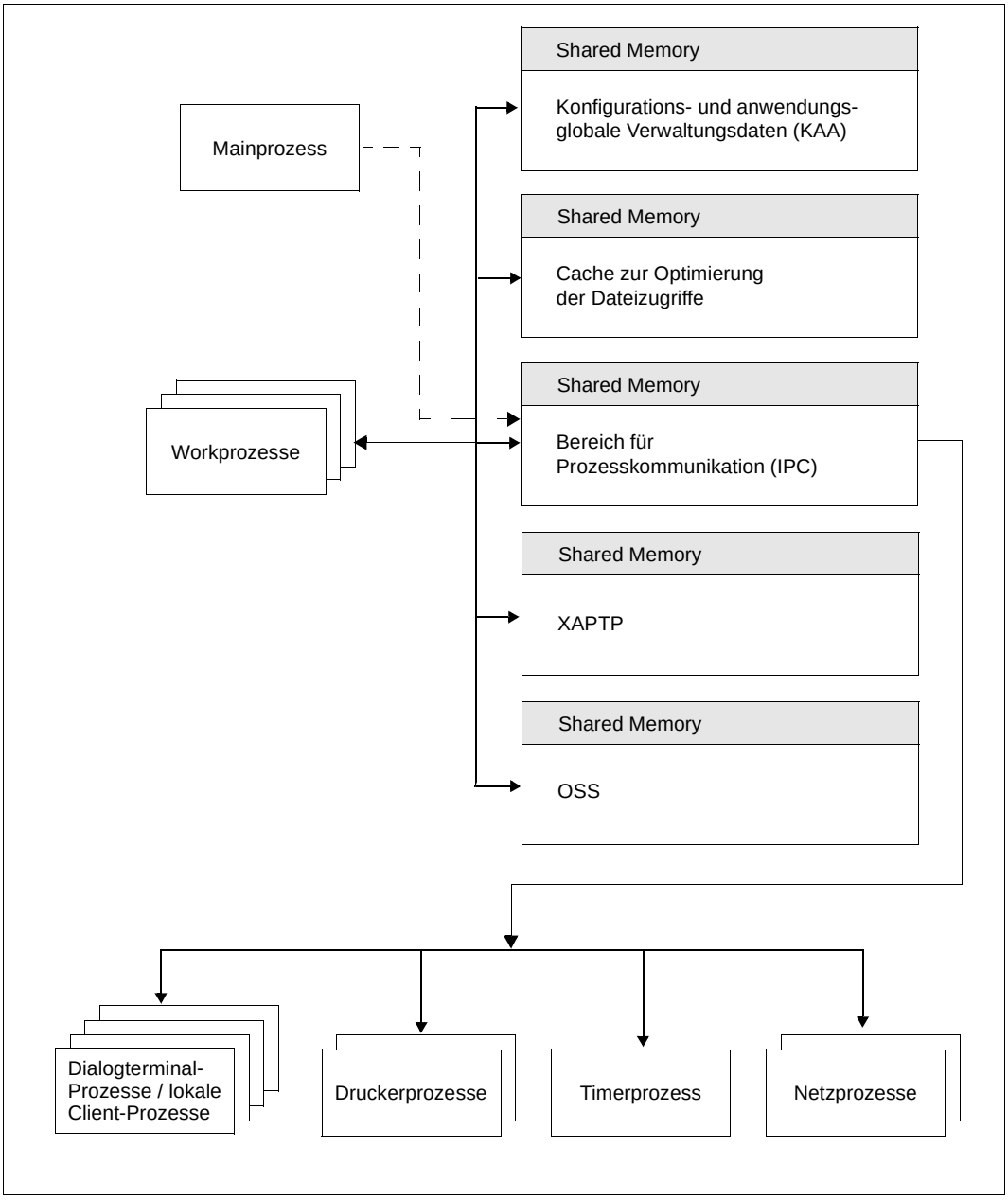


Bild 45: Shared Memories und Prozesse in UTM-Anwendungen auf Unix-Systemen

15.4 Konfigurieren der Netzanbindung

Bei Kopplungen über TCP/IP-RFC1006 und über Socket trägt der Anwender die notwendigen Parameter in die UTM-Generierung ein. 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. Zusätzlich werden bei jedem Verbindungsaufbau die aktuell gültigen Adressen ermittelt und ab diesem Zeitpunkt verwendet.

Damit eine UTM-Anwendung auch dann ohne Unterbrechung betrieben werden kann, wenn sich im Netz eine Adresse ändert, besitzt openUTM eine entsprechende Administrations-Funktion. Mit dieser Funktion können IP-Adressen aus der Host-Datenbank gelesen und zur Laufzeit an die UTM-Anwendung übergeben werden.

Wie Sie die Netzanbindung konfigurieren, ist detailliert im openUTM-Handbuch „Anwendungen generieren“ beschrieben.

15.5 Codeumsetzung

Beim Austausch von Nachrichten einer UTM-Anwendung mit einer Partner-Anwendung gibt es die Möglichkeit, dass openUTM eine automatische ASCII-EBCDIC Code-Konvertierung durchführt. Bei der Konvertierung verwendet openUTM eine Standard-Tabelle. Sie können diese Standardtabelle modifizieren.

Nähere Informationen zur Codeumsetzung finden Sie im openUTM-Handbuch „Anwendungen generieren“ bei den Anweisungen PTERM und TPOOL sowie im Anhang.

15.6 Ablauf auf 64-Bit-Plattformen

openUTM wird auf allen Plattformen auch als 64-Bit-Variante ausgeliefert. Damit lassen sich 64-Bit-fähige UTM-Anwendungen erstellen. Bitte beachten Sie jedoch, dass in diesem Fall alle Komponenten, die zur Anwendung dazugebunden werden, ebenfalls 64-Bit-fähig sein müssen. Dazu gehören z.B. Anwenderprogramme, die Main Routine KDCROOT, Datenbankanschluss, PCMX, openUTM-Encryption-Bibliotheken, openUTM-Systembibliotheken.

Ein Mischbetrieb von 32-Bit und 64-Bit in einer UTM-Anwendung ist nicht möglich.

Beim Anwendungsstart und beim Starten eines Dienstprogramms wird geprüft, ob die verwendeten Komponenten bzgl. der Plattform und dem Bit-Modus untereinander verträglich sind.

16 openUTM in Windows-Systemen

In diesem Kapitel werden einige Plattform-spezifische Details beschrieben, die sich speziell auf den Einsatz von openUTM in Windows-Systemen beziehen:

– Systemeinbettung
– UTM-Prozesse
– Adressraumkonzept
– Konfigurieren der Netzanbindung

Einschränkungen

Formatierung und transaktionsgesicherte Druckausgaben werden auf Windows-Systemen nicht unterstützt.

16.1 Systemeinbettung

Neben den Schnittstellen zum Windows-Betriebssystem verfügt eine UTM-Anwendung intern über eine Reihe weiterer Schnittstellen:

– XA-Schnittstelle (X/Open-Standard) zum Anschluss externer Resource Manager (wie z.B. Oracle,...)

– UPIC-L-Schnittstelle, die es ermöglicht, openUTM-Client-Programme mit Trägersystem UPIC lokal anzuschließen (d.h. die Client-Programme können im selben Windows-System ablaufen wie die UTM-Anwendung)

– Schnittstellen zu den Laufzeitsystemen der verwendeten Programmiersprachen

– Schnittstellen zur Kommunikationskomponente PCMX32

Außerdem wird beim Betrieb mit openUTM das **Windows-Event-Logging** unterstützt, d.h. die Installation, Deinstallation und der Betrieb von openUTM-Services werden in der Ereignisanzeige (Event Viewer) in Form von Ereignissen (Events) protokolliert. Die openUTM-Ereignisse befinden sich im Event Viewer im Bereich Application, die Quelle ist openUTM.

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

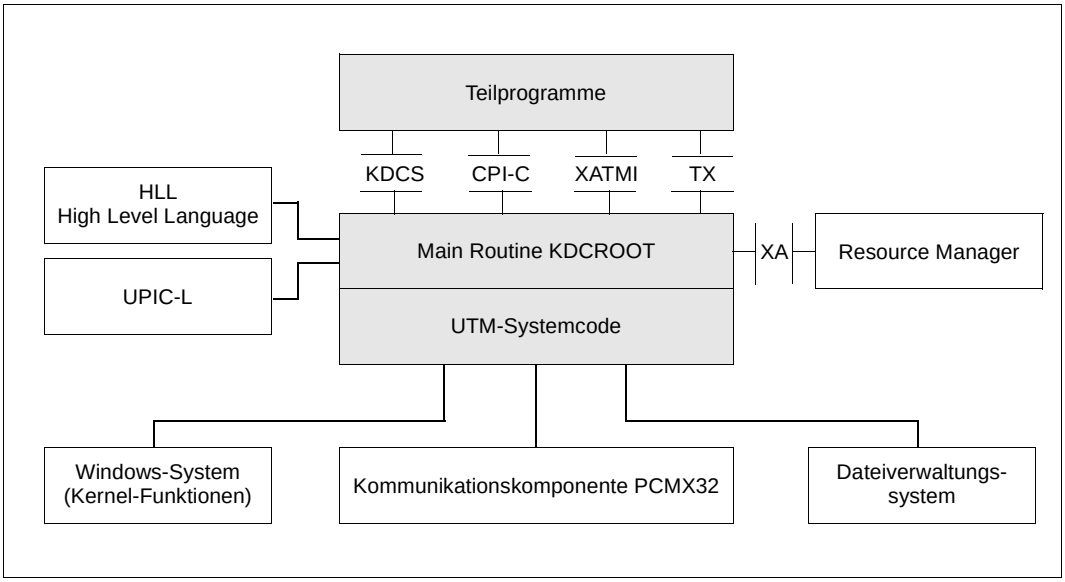


Bild 46: Schnittstellen von openUTM zu anderen Systemkomponenten

16.2 UTM-Prozesse

Eine UTM-Anwendung wird in Windows-Systemen als Win32 Console-Application erstellt. Bei der Ausführung eines UTM-Anwendungsprogramms arbeiten unterschiedliche Prozesse mit jeweils spezifischen Aufgaben zusammen. Einige dieser Prozesse werden in einem DOS-Fenster (Eingabeaufforderung) 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 47 auf Seite 262.

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 CMD-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 des Windows-Systems 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 Anwendungs-bezogen; 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.

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 CMX oder direkt über die Socket-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 openUTM-Handbuch „Anwendungen generieren“.

Wie Sie die Netzprozesse über Umgebungsvariablen steuern können, ist im openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter Unix- und Windows-Systemen“ beschrieben.

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-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 openUTM-Client mit Trägersystem UPIC, der mit der UTM-Anwendung arbeitet, existiert ein eigener lokaler Client-Prozess. Er wird beispielsweise aus der Windows-Eingabeaufforderung 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.

Logging-Prozess

openUTM kann bei laufender Anwendung bestimmte Daten protokollieren wie z.B. Abrechnugssätze oder Messdaten. Diese Protokollierung wird über den **Logging-Prozess** gesteuert (*utmlog*).

- Falls generiert, stellt openUTM während des Anwendungslaufes Accounting-Informationen bereit. Diese Information wird in sogenannten Abrechnungssätzen von openUTM erfasst und an den Logging-Prozess weitergeleitet. Der Logging-Prozess schreibt diese Sätze in eine Datei im Unterverzeichnis ACCNT.
- Zur Leistungskontrolle einer UTM-Anwendung steht die in openUTM integrierte Funktion des UTM-Messmonitors KDCMON zur Verfügung. Nach Starten der KDCMON-Messung erfassen die Workprozesse Messdaten und übermitteln diese an den Logging-Prozess, der die Datensätze in eine Datei im Unterverzeichnis KDCMON schreibt. Dadurch wird die Verwaltung dieser Daten von den Workprozessen auf eine einzige Instanz, den Logging-Prozess, verlagert.

Übersicht: Prozesse einer UTM-Anwendung in Windows-Systemen

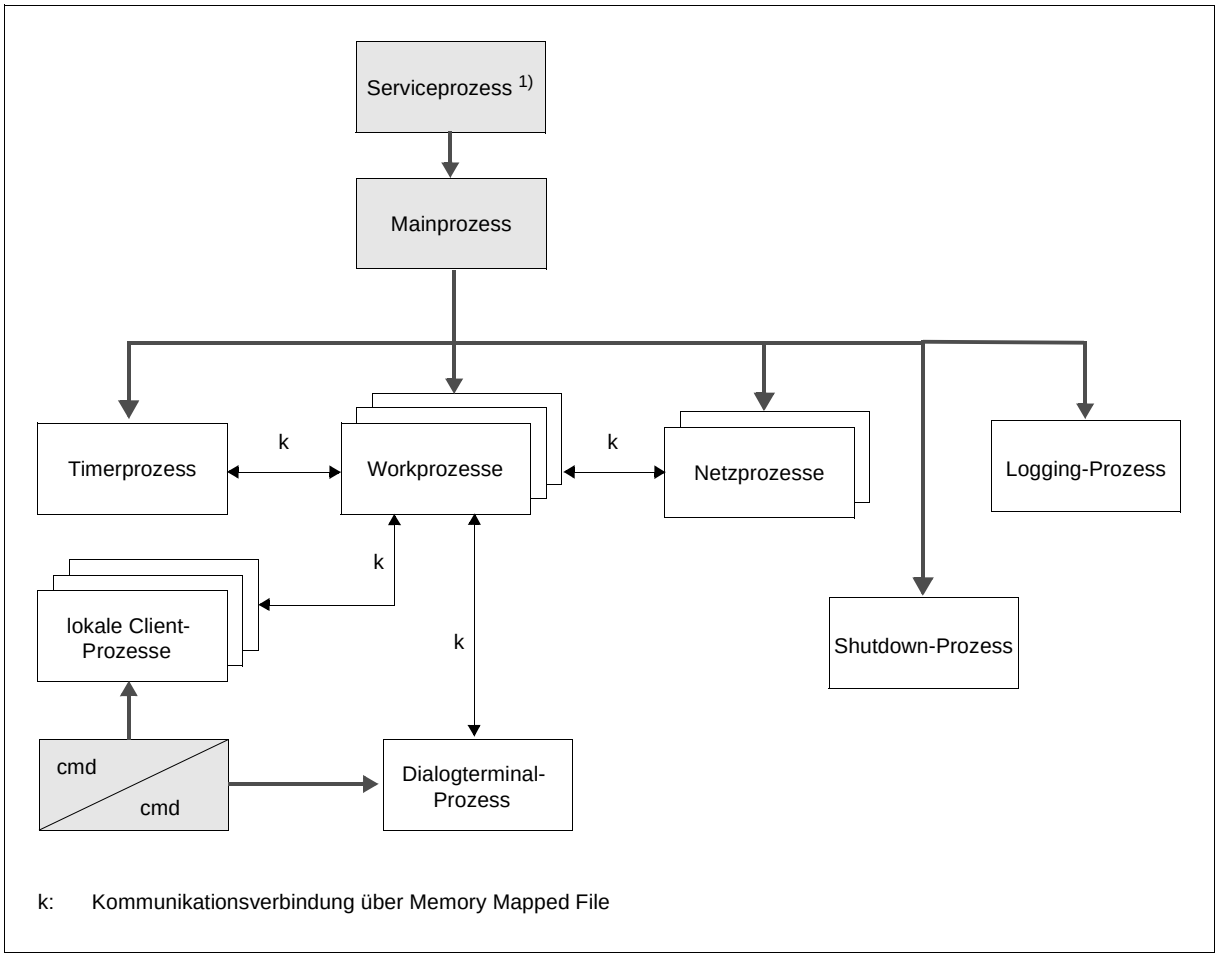


Bild 47: Prozessinteraktion in einer UTM-Anwendung auf Windows-Systemen

1) Der Serviceprozess ist optional. Er exisitiert nur, wenn die Anwendung als Dienst gestartet wurde.

16.3 Adressraumkonzept

In einer UTM-Anwendung verfügt jeder Workprozess über einen Prozess-spezifischen 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 Prozess-spezifische 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 Windows-System 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.

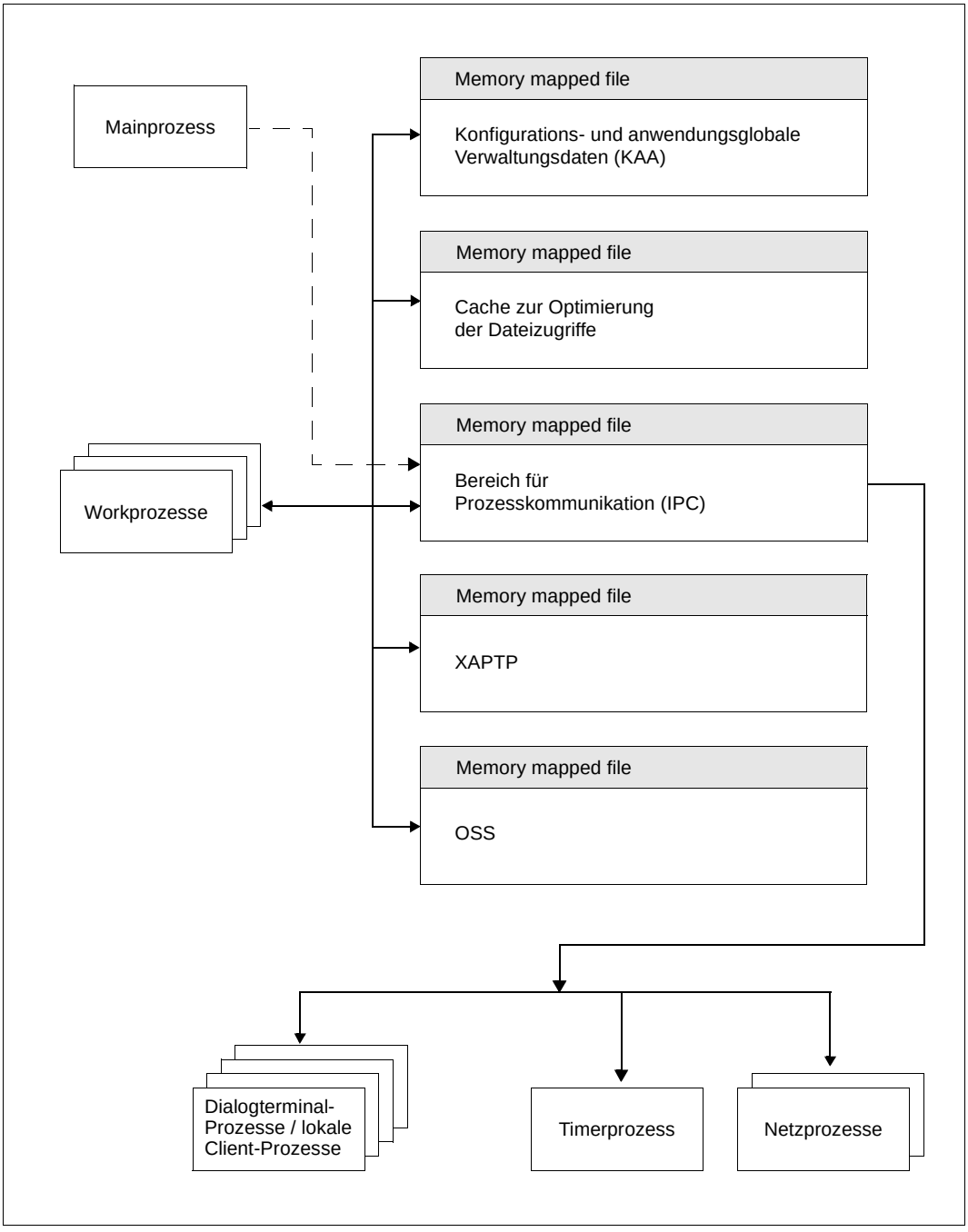


Bild 48: Memory mapped files und Prozesse in UTM-Anwendungen auf Windows-Systemen

16.4 Konfigurieren der Netzanbindung

Bei Kopplungen über TCP/IP-RFC1006 und über Socket trägt der Anwender die notwendigen Parameter in die UTM-Generierung ein. 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. Zusätzlich werden bei jedem Verbindungsaufbau die aktuell gültigen Adressen ermittelt und ab diesem Zeitpunkt verwendet.

Damit eine UTM-Anwendung auch dann ohne Unterbrechung betrieben werden kann, wenn sich im Netz eine Adresse ändert, besitzt openUTM eine entsprechende Administrations-Funktion. Mit dieser Funktion können IP-Adressen aus der Host-Datenbank gelesen und zur Laufzeit an die UTM-Anwendung übergeben werden.

Wie Sie die Netzanbindung konfigurieren, ist detailliert im openUTM-Handbuch „Anwendungen generieren“ beschrieben.

16.5 Codeumsetzung

Beim Austausch von Nachrichten einer UTM-Anwendung mit einer Partner-Anwendung gibt es die Möglichkeit, dass openUTM eine automatische ASCII-EBCDIC Code-Konvertierung durchführt. Bei der Konvertierung verwendet openUTM eine Standard-Tabelle. Sie können diese Standardtabelle modifizieren.

Nähere Informationen zur Codeumsetzung finden Sie im openUTM-Handbuch „Anwendungen generieren“ bei den Anweisungen PTERM und TPOOL sowie im Anhang.

17 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

Fachwörter, die an anderer Stelle erklärt werden, sind mit *kursiver* Schrift ausgezeichnet.

Ablaufinvariantes Programm
siehe *reentrant-fähiges Programm*.

Abnormale Beendigung einer UTM-Anwendung
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 openUTM einen *Warmstart* durch.

abstrakte Syntax (OSI)
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-List
Eine Access-List definiert die Berechtigung für den Zugriff auf einen bestimmten *Service*, auf eine bestimmte *TAC-Queue* oder auf eine bestimmte *USER-Queue*. Eine Access-List ist als *Keyset* definiert und enthält einen oder mehrere *Keycodes*, die jeweils eine Rolle in der Anwendung repräsentieren. Benutzer, LTERMs oder (OSI-)LPAPs dürfen nur dann auf den Service oder die *TAC-Queue*/*USER-Queue* zugreifen, wenn ihnen die entsprechenden Rollen zugeteilt wurden, d.h. wenn ihr *Keyset* und die Access-List mindestens einen gemeinsamen *Keycode* enthalten.

Access Point (OSI)
siehe *Dienstzugriffspunkt.*

ACID-Eigenschaften
Abkürzende Bezeichnung für die grundlegenden Eigenschaften von *Transaktionen*: Atomicity, Consistency, Isolation und Durability.

Administration
Verwaltung und Steuerung einer *UTM-Anwendung* durch einen *Administrator* oder ein *Administrationsprogramm*.

Administrations-Journal
siehe *Cluster-Administrations-Journal*.

Administrationskommando
Kommandos, mit denen der *Administrator* einer *UTM-Anwendung* Administrationsfunktionen für diese Anwendung durchführt. Die Administrationskommandos sind als *Transaktionscodes* realisiert.

Administrationsprogramm
Teilprogramm, das Aufrufe der *Programmschnittstelle für die Administration* enthält. Dies kann das Standard-Administrationsprogramm *KDCADM* sein, das mit openUTM ausgeliefert wird, oder ein vom Anwender selbst erstelltes Programm.

Administrator
Benutzer mit Administrationsberechtigung.

AES
AES (Advanced Encryption Standard) ist der aktuelle symmetrische Verschlüsselungsstandard, festgelegt vom NIST (National Institute of Standards and Technology), basierend auf dem an der Universität Leuven (B) entwickelten Rijndael-Algorithmus. Wird das AES-Verfahren verwendet, dann erzeugt der UPIC-Client für jede Sitzung einen AES-Schlüssel.

Akzeptor (CPI-C)
Die Kommunikationspartner einer *Conversation* werden *Initiator* und Akzeptor genannt. Der Akzeptor nimmt die vom Initiator eingeleitete Conversation mit Accept_Conversation entgegen.

Anmelde-Vorgang (KDCS)
Spezieller *Dialog-Vorgang*, bei dem die Anmeldung eines Benutzers an eine UTM-Anwendung durch *Teilprogramme* gesteuert wird.

Anschlussprogramm
siehe *KDCROOT.*

Anwendungsinformation

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.

Anwendungs-Kaltstart

siehe *Kaltstart*.

Anwendungsprogramm

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.

Anwendungs-Warmstart

siehe *Warmstart*.

Apache Axis

Apache Axis (Apache eXtensible Interaction System) ist eine SOAP-Engine zur Konstruktion von darauf basierenden Web Services und Client-Anwendungen. Es existiert eine Implementierung in C++ und Java.

Apache Tomcat

Apache Tomcat stellt eine Umgebung zur Ausführung von Java-Code auf Web-Servern bereit, die im Rahmen des Jakarta-Projekts der Apache Software Foundation entwickelt wird. Es handelt sich um einen in Java geschriebenen Servlet-Container, der mithilfe des JSP-Compilers Jasper auch JavaServer Pages in Servlets übersetzen und ausführen kann. Dazu kommt ein kompletter HTTP-Server.

Application Context (OSI)

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)

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)

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)

Bestandteil des *Application Entity Titles*. Der Application Entity Qualifier identifiziert einen *Dienstzugriffspunkt* innerhalb der Anwendung. Ein Application Entity Qualifier kann unterschiedlich aufgebaut sein. openUTM unterstützt den Typ "Zahl".

Application Process (OSI)

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)

Gemäß der OSI-Norm dient der Application Process Title (APT) zur global (d.h. weltweit) eindeutigen Identifizierung von Anwendungen. Er kann unterschiedlich aufgebaut sein. openUTM unterstützt den Typ *Object Identifier*.

Application Service Element (OSI)

Ein Application Service Element (ASE) repräsentiert eine Funktionsgruppe der Anwendungsschicht (Schicht 7) des *OSI-Referenzmodells*.

Association (OSI)

Eine Association ist eine Kommunikationsbeziehung zwischen zwei *Application Entities*. Dem Begriff Association entspricht der *LU6.1*-Begriff *Session*.

Asynchron-Auftrag

Auftrag, der vom Auftraggeber zeitlich entkoppelt durchgeführt wird. Zur Bearbeitung von Asynchron-Aufträgen sind in openUTM *Message Queuing* Funktionen integriert, vgl. *UTM-gesteuerte Queue* und *Service-gesteuerte Queue*.
Ein Asynchron-Auftrag wird durch die *Asynchron-Nachricht*, den Empfänger und ggf. den gewünschten Ausführungszeitpunkt beschrieben.
Ist der Empfänger ein Terminal, ein Drucker oder eine Transportsystem-Anwendung, so ist der Asynchron-Auftrag ein *Ausgabe-Auftrag*; ist der Empfänger ein Asynchron-Vorgang derselben oder einer fernen Anwendung, so handelt es sich um einen *Hintergrund-Auftrag*.
Asynchron-Aufträge können *zeitgesteuerte Aufträge* sein oder auch in einen *Auftrags-Komplex* integriert sein.

Asynchron-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

Asynchron-Nachrichten sind Nachrichten, die an eine *Message Queue* gerichtet sind. Sie werden von der lokalen *UTM-Anwendung* zunächst zwischengespeichert und dann unabhängig vom Auftraggeber weiter verarbeitet. Je nach Empfänger unterscheidet man folgende Typen von Asynchron-Nachrichten:

– Bei Asynchron-Nachrichten an eine *UTM-gesteuerte Queue* wird die Weiterverarbeitung komplett durch openUTM gesteuert. Zu diesem Typ gehören Nachrichten, die einen lokalen oder fernen *Asynchron-Vorgang* starten (vgl. auch *Hintergrund-Auftrag*) und Nachrichten, die zur Ausgabe an ein Terminal, einen Drucker oder eine Transportsystem-Anwendung geschickt werden (vgl. auch *Ausgabe-Auftrag*).

– Bei Asynchron-Nachrichten an eine *Service-gesteuerte Queue* wird die Weiterverarbeitung durch einen *Service* der Anwendung gesteuert. Zu diesem Typ gehören Nachrichten an eine *TAC-Queue*, Nachrichten an eine *USER-Queue* und Nachrichten an eine *Temporäre Queue*. Die User-Queue und die Temporäre Queue müssen dabei zur lokalen Anwendung gehören, die TAC-Queue kann sowohl in der lokalen als auch in einer fernen Anwendung liegen.

Asynchron-Programm

Teilprogramm, das von einem *Hintergrund-Auftrag* gestartet wird.

Asynchron-Vorgang (KDCS)

Vorgang, der einen *Hintergrund-Auftrag* 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.

Auftrag

Anforderung eines *Services*, der von einer *UTM-Anwendung* zur Verfügung gestellt wird, durch Angabe eines *Transaktionscodes*. Siehe auch: *Ausgabe-Auftrag*, *Dialog-Auftrag*, *Hintergrund-Auftrag*, *Auftrags-Komplex*.

Auftraggeber-Vorgang

Ein Auftraggeber-Vorgang ist ein *Vorgang*, der zur Bearbeitung eines Auftrags einen Service von einer anderen Server-Anwendung (*Auftragnehmer-Vorgang*) anfordert.

Auftragnehmer-Vorgang
Ein Auftragnehmer-Vorgang ist ein *Vorgang*, der von einem *Auftraggeber-Vorgang* einer anderen Server-Anwendung gestartet wird.

Auftrags-Komplex
Auftrags-Komplexe dienen dazu, *Asynchron-Aufträgen Quittungsaufträge* zuzuordnen. Ein Asynchron-Auftrag innerhalb eines Auftrags-Komplexes wird *Basis-Auftrag* genannt.

Ausgabe-Auftrag
Ausgabeaufträge sind *Asynchron-Aufträge*, die die Aufgabe haben, eine Nachricht, z.B. ein Dokument, an einen Drucker, ein Terminal oder eine Transportsystem-Anwendung auszugeben.
Ausgabeaufträge werden ausschließlich von UTM-Systemfunktionen bearbeitet, d.h. für die Bearbeitung müssen keine Teilprogramme erstellt werden.

Authentisierung
siehe *Zugangskontrolle*.

Autorisierung
siehe *Zugriffskontrolle*.

Axis
siehe *Apache Axis*.

Basis-Auftrag
Asynchron-Auftrag in einem *Auftrags-Komplex*.

Basisformat
Format, in das der Terminal-Benutzer alle Angaben eintragen kann, die notwendig sind, um einen Vorgang zu starten.

Basisname
Basisname UTM-Anwendung.
In BS2000-Systemen ist Basisname das Präfix für die *KDCFILE*, die *Benutzerprotokoll-Datei* USLOG und die *System-Protokolldatei* SYSLOG.
In Unix- und Windows-Systemen ist Basisname der Name des Verzeichnisses, unter dem die KDCFILE, die Benutzerprofokoll-Datei USLOG, die System-Protokolldatei SYSLOG und weitere Dateien der UTM-Anwendung abgelegt sind.

Basisname der Knoten-Anwendung

Dateinamens-Präfix bzw. Verzeichnisname für die *KDCFILE*, *Benutzerprotokoll-Datei* und *Systemprotokoll-Datei* der *Knoten-Anwendung*.

Basisname der UTM-Cluster-Anwendung

Dateinamens-Präfix bzw. Verzeichnisname für die *UTM-Cluster-Dateien*.

Benutzerausgang

Begriff ersetzt durch *Event-Exit*.

Benutzerkennung

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 muss bei der Anmeldung an die UTM-Anwendung diesen Bezeichner (und ggf. das zugeordnete Passwort) angeben. In BS2000-Systemen ist außerdem eine Zugangskontrolle über *Kerberos* möglich.
Für andere Clients ist die Angabe der Benutzerkennung optional, siehe auch *Verbindungs-Benutzerkennung*.
UTM-Anwendungen können auch ohne Benutzerkennungen generiert werden.

Benutzer-Protokolldatei

Datei oder Dateigeneration, in die der Benutzer mit dem KDCS-Aufruf LPUT Sätze variabler Länge schreibt. Jedem Satz werden die Daten aus dem KB-Kopf des *KDCS-Kommunikationsbereichs* vorangestellt. Die Benutzerprotokolldatei unterliegt der Transaktionssicherung von openUTM.

Berechtigungsprüfung

siehe *Zugangskontrolle*.

Beweissicherung (BS2000-Systeme)

Im Betrieb einer *UTM-Anwendung* können zur Beweissicherung sicherheitsrelevante UTM-Ereignisse von *SAT* protokolliert werden.

Bildschirm-Wiederanlauf

Wird ein *Dialog-Vorgang* unterbrochen, gibt openUTM beim *Vorgangswiederanlauf* die *Dialog-Nachricht* der letzten abgeschlossenen *Transaktion* erneut auf dem Bildschirm aus, sofern die letzte Transaktion eine Nachricht auf den Bildschirm ausgegeben hat.

Browsen von Asynchron-Nachrichten

Ein *Vorgang* liest nacheinander die *Asynchron-Nachrichten*, die sich in einer *Service-gesteuerten Queue* befinden. Die Nachrichten werden während des Lesens nicht gesperrt und verbleiben nach dem Lesen in der Queue. Dadurch ist gleichzeitiges Lesen durch unterschiedliche Vorgänge möglich.

Bypass-Betrieb (BS2000-Systeme)

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

Pufferbereich zur Zwischenspeicherung von Anwenderdaten für alle Prozesse einer *UTM-Anwendung*. Der Cache-Speicher dient zur Optimierung der Zugriffe auf den *Pagepool* und für UTM-Cluster-Anwendungen zusätzlich auf den *Cluster-Pagepool*.

CCS-Name (BS2000-Systeme)

siehe *Coded-Character-Set-Name*.

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. openUTM-Clients mit Trägersystem OpenCPIC werden wie *OSI TP-Partner* behandelt.

Client-Seite einer Conversation

Begriff ersetzt durch *Initiator.*

Cluster

Eine Anzahl von Rechnern, die über ein schnelles Netzwerk verbunden sind und die von außen in vielen Fällen als ein Rechner gesehen werden können. Das Ziel des "Clustering" ist meist die Erhöhung der Rechenkapazität oder der Verfügbarkeit gegenüber einem einzelnen Rechner.

Cluster-Administrations-Journal

Das Cluster-Administrations-Journal besteht aus:

– zwei Protokolldateien mit Endungen JRN1 und JRN2 für globale Administrationsaktionen,
– der JKAA-Datei, die eine Kopie der KDCS Application Area (KAA) enthält. Aus dieser Kopie werden administrative Änderungen übernommen, die nicht mehr in den beiden Protokolldateien enthalten sind.

Die Administrations-Journal-Dateien dienen dazu, administrative Aktionen, die in einer UTM-Cluster-Anwendung Cluster-weit auf alle Knoten-Anwendungen wirken sollen, an die anderen Knoten-Anwendungen weiterzugeben.

Cluster-GSSB-Datei

Datei zur Verwaltung von GSSBs in einer *UTM-Cluster-Anwendung*. Die Cluster-GSSB-Datei wird mit dem UTM-Generierungstool *KDCDEF* erstellt.

Cluster-Konfigurationsdatei

Datei, die die zentralen Konfigurationsdaten einer *UTM-Cluster-Anwendung* enthält. Die Cluster-Konfigurationsdatei wird mit dem UTM-Generierungstool *KDCDEF* erstellt.

Cluster-Lock-Datei

Datei einer *UTM-Cluster-Anwendung*, die dazu dient, Knoten-übergreifende Sperren auf Anwenderdatenbereiche zu verwalten.

Cluster-Pagepool

Der Cluster-Pagepool besteht aus einer Verwaltungsdatei und bis zu 10 Dateien, in denen die Cluster-weit verfügbaren Anwenderdaten (Vorgangsdaten inklusive LSSB, GSSB und ULS) einer *UTM-Cluster-Anwendung* gespeichert werden. Der Cluster-Pagepool wird mit dem UTM-Generierungstool *KDCDEF* erstellt.

Cluster-Startserialisierungs-Datei

Lock-Datei, mit der die Starts einzelner Knoten-Anwendungen serialisiert werden (nur bei Unix- und Windows-Systemen).

Cluster-ULS-Datei

Datei zur Verwaltung von ULS-Bereichen einer *UTM-Cluster-Anwendung*. Die Cluster-ULS-Datei wird mit dem UTM-Generierungstool *KDCDEF* erstellt.

Cluster-User-Datei

Datei, die die Verwaltungsdaten der Benutzer einer *UTM-Cluster-Anwendung* enthält. Die Cluster-User-Datei wird mit dem UTM-Generierungstool *KDCDEF* erstellt.

Coded-Character-Set-Name (BS2000-Systeme)

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 (CRMs) kontrollieren in verteilten Systemen die Kommunikation zwischen den Anwendungsprogrammen. openUTM stellt CRMs für den internationalen Standard OSI TP, für den Industrie-Standard *LU6.1* und für das openUTM-eigene Protokoll UPIC zur Verfügung.

Contention Loser

Jede Verbindung zwischen zwei Partnern wird von einem der Partner verwaltet. Der Partner, der die Verbindung verwaltet, heißt *Contention Winner*. Der andere Partner ist der Contention Loser.

Contention Winner

Der Contention Winner einer Verbindung übernimmt die Verwaltung der Verbindung. Aufträge können sowohl vom Contention Winner als auch vom *Contention Loser* gestartet werden. Im Konfliktfall, wenn beide Kommunikationspartner gleichzeitig einen Auftrag starten wollen, wird die Verbindung vom Auftrag des Contention Winner belegt.

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

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 openUTM 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 openUTM kann über die Protokolle OSI TP, LU6.1, UPIC und mit openUTM-LU6.2 kommunizieren.

Cross Coupled System / XCS

Verbund von BS2000-Rechnern mit *Highly Integrated System Complex* Multiple System Control Facility (HIPLEX® MSCF).

Dead Letter Queue

Die Dead Letter Queue ist eine *TAC-Queue* mit dem festen Namen KDCDLETQ. Sie steht immer zur Verfügung, um Asynchron-Nachrichten an *Transaktionscodes* oder TAC-Queues zu sichern, die nicht verarbeitet werden konnten. Die Sicherung von Asynchron-Nachrichten in der Dead Letter Queue kann durch den Parameter DEAD-LETTER-Q der TAC-Anweisung für jedes Nachrichtenziel einzeln ein- und ausgeschaltet werden.

DES

DES (Data Encryption Standard) ist eine internationale Norm zur Verschlüsselung von Daten. Bei diesem Verfahren wird ein Schlüssel zum Ver- und Entschlüsseln verwendet. Wird das DES-Verfahren verwendet, dann erzeugt der UPIC-Client für jede Sitzung einen DES-Schlüssel.

Dialog-Auftrag

Auftrag, der einen *Dialog-Vorgang* startet. Der Auftrag kann von einem *Client* oder - bei *Server-Server-Kommunikation* - von einer anderen Anwendung erteilt werden.

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

Nachricht, die eine Antwort erfordert oder selbst eine Antwort auf eine Anfrage ist. Dabei bilden Anfrage und Antwort einen *Dialog-Schritt*.

Dialog-Programm

Teilprogramm, das einen *Dialog-Schritt* teilweise oder vollständig bearbeitet.

Dialog-Schritt

Ein Dialog-Schritt beginnt mit dem Empfang einer *Dialog-Nachricht* durch die *UTM-Anwendung*. Er endet mit der Antwort der UTM-Anwendung.

Dialog-Terminalprozess (Unix-/Windows-Systeme)

Ein Dialog-Terminalprozess verbindet ein Unix-/Windows-Terminal mit den *Workprozessen* der *UTM-Anwendung*. Dialog-Terminalprozesse 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 Dialog-Terminalprozess erforderlich.

Dialog-Vorgang

Vorgang, der einen *Auftrag* im Dialog (zeitlich gekoppelt) mit dem Auftraggeber (*Client* oder eine andere Server-Anwendung) 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 *Dialog-Schritt*. Ausnahme: Bei *Vorgangskettung* können auch mehrere Vorgänge einen Dialog-Schritt bilden.

Dienst

Programm auf Windows-Systemen, das im Hintergrund unabhängig von angemeldeten Benutzern oder Fenstern abläuft.

Dienstzugriffspunkt

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 Lock Manager / DLM (BS2000-Systeme)

Konkurrierende, Rechner-übergreifende Dateizugriffe können über den Distributed Lock Manager synchronisiert werden.
DLM ist eine Basisfunktion von HIPLEX® MSCF.

Distributed Transaction Processing

X/Open-Architekturmodell für die transaktionsorientierte *verteilte Verarbeitung*.

Druckadministration

Funktionen zur *Drucksteuerung* und Administration von *Ausgabeaufträgen*, die an einen Drucker gerichtet sind.

Druckerbündel

Mehrere Drucker, die demselben *LTERM-Partner* zugeordnet sind.

Druckergruppe (Unix-Systeme)
Die Unix-Plattform 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-Systeme)
Prozess, der vom *Mainprozess* 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
Begriff wurde ersetzt durch *Druckersteuer-LTERM*.

Druckersteuer-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
openUTM-Funktionen zur Steuerung von Druckausgaben.

Dynamische Konfiguration
Änderung der *Konfiguration* durch die Administration. Im laufenden Betrieb der Anwendung können UTM-Objekte wie z.B. *Teilprogramme*, *Transaktionscodes*, *Clients*, *LU6.1-Verbindungen*, Drucker oder *Benutzerkennungen* in die Konfiguration aufgenommen, modifiziert oder teilweise auch gelöscht werden. Hierzu können die Administrationsprogramme WinAdmin oder WebAdmin verwendet werden, oder es müssen eigene *Administrationsprogramme* erstellt werden, die die Funktionen der *Programmschnittstelle der Administration* nutzen.

Einschritt-Transaktion
Transaktion, die genau einen *Dialog-Schritt* umfasst.

Einschritt-Vorgang
Dialog-Vorgang, der genau einen *Dialog-Schritt* umfasst.

Ereignisgesteuerter Vorgang
Begriff ersetzt durch *Event-Service*.

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

Oberbegriff für *Event-Exits* und *Event-Services*.

Event-Service

Vorgang, der beim Auftreten bestimmter Ereignisse gestartet wird, z.B. bei bestimmten UTM-Meldungen. Die *Teilprogramme* ereignisgesteuerter Vorgänge müssen KDCS-Aufrufe enthalten.

Generierung

Statische Konfiguration einer *UTM-Anwendung* mit dem UTM-Tool *KDCDEF* und Erzeugen des *Anwendungsprogramms*.

Globaler Sekundärer Speicherbereich/GSSB

siehe *Sekundärspeicherbereich.*

Hardcopy-Betrieb

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

Bei *Server-Server-Kommunikation:* Kopplung einer *UTM-Anwendung* mit einer Nicht-UTM-Anwendung, z.B. einer CICS- oder TUXEDO-Anwendung.

Highly Integrated System Complex / HIPLEX®

Produktfamilie zur Realisierung eines Bedien-, Last- und Verfügbarkeitsverbunds mit mehreren BS2000-Servern.

Hintergrund-Auftrag

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.

HIPLEX® MSCF

(MSCF = **M**ultiple **S**ystem **C**ontrol **F**acility)
stellt bei HIPLEX® die Infrastruktur sowie Basisfunktionen für verteilte Anwendungen bereit.

Homogene Kopplung

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)

siehe *Incoming-Conversation*.

Incoming-Conversation (CPI-C)

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.

Initiale KDCFILE

In einer *UTM-Cluster-Anwendung* die *KDCFILE*, die von *KDCDEF* erzeugt wurde und vor dem Start der Knoten-Anwendungen für jeden Knoten kopiert werden muss.

Initiator (CPI-C)

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

Feld in einem Meldungstext, in das openUTM aktuelle Werte einträgt.

Inverser 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.

JDK

Java Development Kit
Standard-Entwicklungsumgebung von Sun Microsystems für die Entwicklung von Java-Anwendungen.

Kaltstart

Starten einer *UTM-Anwendung* nach einer *normalen Beendigung* der Anwendung oder nach einer Neugenerierung (vgl. auch *Warmstart*).

KDCADM

Standard-Administrationsprogramm, das zusammen mit openUTM 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*.
In UTM-Cluster-Anwendungen erstellt KDCDEF zusätzlich die *Cluster-Konfigurationsdatei*, die *Cluster-User-Datei*, den *Cluster-Pagepool*, die *Cluster-GSSB-Datei* und die *Cluster-ULS-Datei*.

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.

KDCS-Parameterbereich

siehe *Parameterbereich*.

KDCS-Programmschnittstelle

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.

Kerberos

Kerberos ist ein standardisiertes Netzwerk-Authentisierungsprotokoll (RFC1510), das auf kryptographischen Verschlüsselungsverfahren basiert, wobei keine Kennwörter im Klartext über das Netzwerk gesendet werden.

Kerberos-Principal

Eigentümer eines Schlüssels.
Kerberos arbeitet mit symmetrischer Verschlüsselung, d.h. alle Schlüssel liegen an zwei Stellen vor, beim Eigentümer eines Schlüssels (Principal) und beim KDC (Key Distribution Center).

Keycode

Code, der in einer Anwendung eine bestimmte Zugriffsberechtigung oder eine bestimmte Rolle repräsentiert. Mehrere Keycodes werden zu einem *Keyset* zusammengefasst.

Keyset

Zusammenfassung von einem oder mehrerer *Keycodes* unter einem bestimmten Namen. Ein Keyset definiert Berechtigungen im Rahmen des verwendeten Berechtigungskonzepts (Lock-/Keycode-Konzept oder *Access-List*-Konzept). Ein Keyset kann einer *Benutzerkennung*, einem *LTERM-Partner*, einem (OSI-)*LPAP-Partner*, einem *Service* oder einer *TAC-Queue* zugeordnet werden.

Knoten

Einzelner Rechner eines *Clusters*.

Knoten-Anwendung

UTM-Anwendung, die als Teil einer *UTM-Cluster-Anwendung* auf einem einzelnen *Knoten* zum Ablauf kommt.

Knoten-Recovery

Wenn für eine abnormal beendete Knoten-Anwendung zeitnah kein Wartstart auf ihrem eigenen *Knoten-Rechner* möglich ist, kann man für diesen Knoten auf einem anderen Knoten des UTM-Clusters eine Knoten-Recovery (Wiederherstellung) durchführen. Dadurch können Sperren, die von der ausgefallenen Knoten-Anwendung gehalten werden, freigegeben werden, um die laufende *UTM-Cluster-Anwendung* nicht unnötig zu beeinträchtigen.

Knotengebundener Vorgang

Ein knotengebundener Vorgang eines Benutzers kann nur an der Knoten-Anwendung fortgesetzt werden, an der der Benutzer zuletzt angemeldet war. Folgende Vorgänge sind immer knotengebunden:

– Vorgänge, die eine Kommunikation mit einem Auftragnehmer über LU6.1 oder OSI TP begonnen haben und bei denen der Auftragnehmervorgang noch nicht beendet wurde
– eingeschobene Vorgänge einer Vorgangskellerung
– Vorgänge, die eine SESAM-Transaktion abgeschlossen haben

Außerdem ist der Vorgang eines Benutzers knotengebunden, solange der Benutzer an eine Knoten-Anwendung angemeldet ist.

Kommunikationsbereich/KB (KDCS)

Transaktionsgesicherter KDCS-*Primärspeicherbereich*, der Vorgangs-spezifische 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

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 (*statische Konfiguration*) und kann per *Administration* dynamisch (während des Anwendungslaufs) geändert werden (*dynamische Konfiguration*). Die Konfiguration ist in der *KDCFILE* abgelegt.

Logging-Prozess

Prozess in Unix- und Windows-Systemen, der die Protokollierung von Abrechnungssätzen oder Messdaten steuert.

Logische Verbindung

Zuordnung zweier Kommunikationspartner.

Log4j

Log4j ist ein Teil des Apache Jakarta Projekts. Log4j bietet Schnittstellen zum Protokollieren von Informationen (Ablauf-Informationen, Trace-Records,...) und zum Konfigurieren der Protokoll-Ausgabe. *WS4UTM* verwendet das Softwareprodukt Log4j für die Trace- und Logging-Funktionalität.

Lockcode

Code, um einen LTERM-Partner oder einen Transaktionscode vor unberechtigtem Zugriff zu schützen. Damit ist ein Zugriff nur möglich, wenn das *Keyset* des Zugreifenden den passenden *Keycode* enthält (Lock-/Keycode-Konzept).

Lokaler Sekundärer Speicherbereich/LSSB

siehe *Sekundärspeicherbereich.*

LPAP-Bündel

LPAP-Bündel ermöglichen die Verteilung von Nachrichten an LPAP-Partner auf mehrere Partner-Anwendungen. Soll eine UTM-Anwendung sehr viele Nachrichten mit einer Partner-Anwendung austauschen, kann es für die Lastverteilung sinnvoll sein, mehrere Instanzen der Partner-Anwendung zu starten und die Nachrichten auf die einzelnen Instanzen zu verteilen. In einem LPAP-Bündel übernimmt *open*UTM die Verteilung der Nachrichten an die Instanzen der Partner-Anwendung. Ein LPAP-Bündel besteht aus einem Master-LPAP und mehreren Slave-LPAPs. Die Slave-LPAPs werden dem Master-LPAP bei der Generierung zugeordnet. LPAP-Bündel gibt es sowohl für das OSI TP-Protokoll als auch für das LU6.1-Protokoll.

LPAP-Partner

Für die *verteilte Verarbeitung* über das *LU6.1*-Protokoll muss in der lokalen Anwendung für jede Partner-Anwendung ein LPAP-Partner konfiguriert werden. Der LPAP-Partner spiegelt in der lokalen Anwendung die Partner-Anwendung wider. Bei der Kommunikation wird die Partner-Anwendung nicht über ihren Anwendungsnamen oder ihre Adresse, sondern über den Namen des zugeordneten LPAP-Partners angesprochen.

LTERM-Bündel

Ein LTERM-Bündel (Verbindungsbündel) besteht aus einem Master-LTERM und mehreren Slave-LTERMs. Mit einem LTERM-Bündel (Verbindungsbündel) verteilen Sie asynchrone Nachrichten an eine logische Partner-Anwendung gleichmäßig auf mehrere parallele Verbindungen.

LTERM-Gruppe

Eine LTERM-Gruppe besteht aus einem oder mehreren Alias-LTERMs, den Gruppen-LTERMs, und einem Primary-LTERM. In einer LTERM-Gruppe ordnen Sie mehrere LTERMs einer Verbindung zu.

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. in der *Konfiguration* festgelegt, sie kann aber auch dynamisch über Terminal-Pools erfolgen.

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*.

LU6.1-LPAP-Bündel

LPAP-Bündel für *LU6.1*-Partner-Anwendungen.

LU6.1-Partner

Partner der *UTM-Anwendung*, der mit der UTM-Anwendung über das Protokoll *LU6.1* kommuniziert.
Beispiele für solche Partner sind:

– eine UTM-Anwendung, die über LU6.1 kommuniziert
– eine Anwendung im IBM-Umfeld (z.B. CICS, IMS oder TXSeries), die über LU6.1 kommuniziert

Mainprozess (Unix-/Windows-Systeme)

Prozess, der die *UTM-Anwendung* startet. Er startet die *Workprozesse*, die *UTM-System-Prozesse*, *Druckerprozesse, Netzprozesse, Logging-Prozess* und den *Timer-prozess* und überwacht die *UTM-Anwendung*.

Main Routine KDCROOT

siehe *KDCROOT.*

Management Unit

Komponente des *SE Servers*; ermöglicht mit Hilfe des *SE Managers* ein zentrales, web-basiertes Management aller Units eines SE Servers.

Mapped Hostname

Abbildung des UTM-Hostnamen der Partner-Anwendung in einen realen Hostnamen oder umgekehrt.

Meldung / UTM-Meldung

Meldungen werden vom Transaktionsmonitor openUTM 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

Die Meldungsdefinitionsdatei wird mit openUTM 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

Ausgabemedium für eine *Meldung*. Mögliche Meldungsziele von Meldungen des Transaktionsmonitors openUTM sind z.B. Terminals, *TS-Anwendungen*, der *Event-Service* MSGTAC, die *System-Protokolldatei* SYSLOG oder *TAC-Queues, Asynchron-TACs, USER-Queues*, SYSOUT/SYSLST bzw. stderr/stdout. Meldungsziele von Meldungen der UTM-Tools sind SYSOUT/SYSLST bzw. stderr/stdout.

Mehrschritt-Transaktion

Transaktion, die aus mehr als einem *Verarbeitungsschritt* besteht.

Mehrschritt-Vorgang (KDCS)

Vorgang, der in mehreren *Dialog-Schritten* ausgeführt wird.

Message Queuing

Message Queuing (MQ) ist eine Form der Kommunikation, bei der die Nachrichten (Messages) nicht unmittelbar, sondern über zwischengeschaltete *Message Queues* ausgetauscht werden. Sender und Empfänger können zeitlich und räumlich entkoppelt ablaufen. Die Übermittlung der Nachricht hängt nicht davon ab, ob gerade eine Netzverbindung besteht oder nicht. Bei openUTM gibt es *UTM-gesteuerte Queues* und *Service-gesteuerte Queues*.

Message Queue

Warteschlange, in der bestimmte Nachrichten transaktionsgesichert bis zur Weiterverarbeitung eingereiht werden. Je nachdem, wer die Weiterverarbeitung kontrolliert, unterscheidet man *Service-gesteuerte Queues* und *UTM-gesteuerte Queues*.

MSGTAC

Spezieller Event-Service, der Meldungen mit dem Meldungsziel MSGTAC per Programm verarbeitet. MSGTAC ist ein Asynchron-Vorgang und wird vom Betreiber der Anwendung erstellt.

Multiplexanschluss (BS2000-Systeme)

Spezielle Möglichkeit, Terminals an eine *UTM-Anwendung* anzuschließen. Ein Multiplexanschluss ermöglicht es, dass sich mehrere Terminals eine *Transportverbindung* teilen.

Nachrichten-Bereich/NB (KDCS)

Bei KDCS-Aufrufen: Puffer-Bereich, in dem Nachrichten oder Daten für openUTM oder für das *Teilprogramm* bereitgestellt werden.

Nachrichten-Verteiler (BS2000-Systeme)
Einrichtung in einem zentralen Rechner oder Kommunikationsrechner zur Verteilung von Eingabe-Nachrichten an unterschiedliche *UTM-Anwendungen*, die auf unterschiedlichen Rechnern liegen können. Der Nachrichten-Verteiler ermöglicht außerdem, mit *Multiplexanschlüssen* zu arbeiten.

Network File System/Service / NFS
Ermöglicht den Zugriff von Unix-Rechnern auf Dateisysteme über das Netzwerk.

Netzprozess (Unix-/Windows-Systeme)
Prozess einer *UTM-Anwendung* zur Netzanbindung.

Netzwerk-Selektor
Der Netzwerk-Selektor identifiziert im lokalen System einen *Dienstzugriffspunkt* zur Vermittlungsschicht des *OSI-Referenzmodells*.

Normale Beendigung einer UTM-Anwendung
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 openUTM als *Kaltstart* durch.

Object Identifier
Ein Object Identifier ist ein 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
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 Typ-spezifischer Terminalpool generiert ist.

Online-Import
Als Online-Import wird in einer *UTM-Cluster-Anwendung* das Importieren von Anwendungsdaten aus einer normal beendeten Knoten-Anwendung in eine laufende Knoten-Anwendung bezeichnet.

Online-Update
Als Online-Update wird in einer *UTM-Cluster-Anwendung* die Änderung der Konfiguration der Anwendung oder des Anwendungsprogramms oder der Einsatz einer neuen UTM-Korrekturstufe bei laufender *UTM-Cluster-Anwendung* bezeichnet.

OpenCPIC
Trägersystem für UTM-Clients, die das *OSI TP* Protokoll verwenden.

OpenCPIC-Client
OSI TP Partner-Anwendungen mit Trägersystem *OpenCPIC*.

openSM2
Die Produktlinie openSM2 ist eine einheitliche Lösung für das unternehmensweite Performance Management von Server- und Speichersystemen. openSM2 bietet eine Messdatenerfassung, Online-Überwachung und Offline-Auswertung.

openUTM-Anwendung
siehe *UTM-Anwendung*.

openUTM-Cluster
aus der Sicht von UPIC-Clients, **nicht** aus Server-Sicht:
Zusammenfassung mehrerer Knoten-Anwendungen einer UTM-Cluster-Anwendung zu einer logischen Anwendung, die über einen gemeinsamen Symbolic Destination Name adressiert wird.

openUTM-D
openUTM-D (openUTM-Distributed) ist eine openUTM-Komponente, die *verteilte Verarbeitung* ermöglicht. openUTM-D ist integraler Bestandteil von openUTM.

OSI-LPAP-Bündel
LPAP-Bündel für *OSI TP*-Partner-Anwendungen.

OSI-LPAP-Partner
OSI-LPAP-Partner sind die bei openUTM generierten Adressen der *OSI TP-Partner*. Für die *verteilte Verarbeitung* über das Protokoll *OSI TP* muss in der lokalen Anwendung für jede Partner-Anwendung ein OSI-LPAP-Partner konfiguriert werden. Der OSI-LPAP-Partner spiegelt in der lokalen Anwendung die Partner-Anwendung wider. Bei der Kommunikation wird die Partner-Anwendung nicht über ihren Anwendungsnamen oder ihre Adresse, sondern über den Namen des zugeordneten OSI-LPAP-Partners angesprochen.

OSI-Referenzmodell
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

Partner der UTM-Anwendung, der mit der UTM-Anwendung über das OSI TP-Protokoll kommuniziert.
Beispiele für solche Partner sind:

– eine UTM-Anwendung, die über OSI TP kommuniziert
– eine Anwendung im IBM-Umfeld (z.B. CICS), die über openUTM-LU62 angeschlossen ist
– eine Anwendung des Trägersystems OpenCPIC des openUTM-Client
– Anwendungen anderer TP-Monitore, die OSI TP unterstützen

Outbound-Conversation (CPI-C)

siehe *Outgoing-Conversation*.

Outgoing-Conversation (CPI-C)

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

Teil der *KDCFILE*, in dem Anwenderdaten gespeichert werden.
In einer *stand-alone-Anwendung* sind dies z.B. *Dialog-Nachrichten*, Nachrichten an *Message Queues*, *Sekundärspeicherbereiche*.
In einer *UTM-Cluster-Anwendung* sind dies z.B. Nachrichten an *Message Queues, TLS*.

Parameterbereich

Datenstruktur, in der ein *Teilprogramm* bei einem UTM-Aufruf die für diesen Aufruf notwendigen Operanden an openUTM übergibt.

Partner-Anwendung

Partner einer UTM-Anwendung bei *verteilter Verarbeitung*. Für die verteilte Verarbeitung werden höhere Kommunikationsprotokolle verwendet (*LU6.1*, *OSI TP* oder LU6.2 über das Gateway openUTM-LU62).

Postselection (BS2000-Systeme)
Auswahl der protokollierten UTM-Ereignisse aus der SAT-Protokolldatei, die ausgewertet werden sollen. Die Auswahl erfolgt mit Hilfe des Tools SATUT.

Prepare to commit (PTC)
Bestimmter Zustand einer verteilten Transaktion:
Das Transaktionsende der verteilten Transaktion wurde eingeleitet, es wird jedoch noch auf die Bestätigung des Transaktionsendes durch den Partner gewartet.

Preselection (BS2000-Systeme)
Festlegung der für die *SAT-Beweissicherung* zu protokollierenden UTM-Ereignisse. Die Preselection erfolgt durch die UTM-SAT-Administration. Man unterscheidet Ereignis-spezifische, Benutzer-spezifische und Auftrags-(TAC-)spezifische Preselection.

Presentation-Selektor
Der Presentation-Selektor identifiziert im lokalen System einen *Dienstzugriffspunkt* zur Darstellungsschicht des *OSI-Referenzmodells*.

Primärspeicherbereich
Bereich im Arbeitsspeicher, auf den das *KDCS-Teilprogramm* direkt zugreifen kann, z.B. *Standard Primärer Arbeitsbereich*, *Kommunikationsbereich*.

Printerprozess (Unix-Systeme)
siehe *Druckerprozess*.

Programmschnittstelle zur Administration
UTM-Programmschnittstelle, mit deren Hilfe der Anwender eigene *Administrationsprogramme* erstellen kann. Die Programmschnittstelle zur Administration bietet u.a. Funktionen zur *dynamischen Konfiguration*, zur Modifikation von Eigenschaften und Anwendungsparametern und zur Abfrage von Informationen zur *Konfiguration* und zur aktuellen Auslastung der Anwendung.

Prozess
In den openUTM-Handbüchern wird der Begriff "Prozess" als Oberbegriff für Prozess (Unix-/Windows-Systeme) und Task (BS2000-Systeme) verwendet.

Queue
siehe *Message Queue*

Quick Start Kit
Beispielanwendung, die mit openUTM (Windows-Systeme) ausgeliefert wird.

Quittungs-Auftrag

Bestandteil eines *Auftrags-Komplexes*, worin der Quittungs-Auftrag dem *Basis-Auftrag* zugeordnet ist. Es gibt positive und negative Quittungsaufträge. Bei positivem Ergebnis des *Basis-Auftrags* wird der positive Quittungs-Auftrag wirksam, sonst der negative.

Redelivery

Erneutes Zustellen einer *Asynchron-Nachricht*, nachdem diese nicht ordnungsgemäß verarbeitet werden konnte, z.B. weil die *Transaktion* zurückgesetzt oder der *Asynchron-Vorgang* abnormal beendet wurde. Die Nachricht wird wieder in die Message Queue eingereiht und lässt sich damit erneut lesen und/oder verarbeiten.

Reentrant-fähiges Programm

Programm, dessen Code durch die Ausführung nicht verändert wird.
In BS2000-Systemen ist dies Voraussetzung dafür, *Shared Code* zu nutzen.

Request

Anforderung einer *Service-Funktion* durch einen *Client* oder einen anderen Server.

Requestor

In XATMI steht der Begriff Requestor für eine Anwendung, die einen Service aufruft.

Resource Manager

Resource Manager (RMs) verwalten Datenressourcen. Ein Beispiel für RMs sind Datenbank-Systeme. openUTM 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 openUTM-RMs die Schnittstelle KDCS.

RFC1006

Von IETF (Internet Engineering Task Force) definiertes Protokoll der TCP/IP-Familie zur Realisierung der ISO-Transportdienste (Transportklasse 0) auf TCP/IP-Basis.

RSA

Abkürzung für die Erfinder des RSA-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-Systeme)
Beweissicherung durch die Komponente SAT (Security Audit Trail) des BS2000-Softwareproduktes SECOS.

SE Manager
Web-basierte Benutzeroberfläche (GUI) für Business Server der SE Serie. Der SE Manager läuft auf der *Management Unit* und ermöglicht die zentrale Bedienung und Verwaltung von Server Units (mit /390-Architektur und/oder x86-Architektur), Application Units (x86-Architektur), Net Unit und der Peripherie.

SE Server
Ein Business Server der SE Serie von Fujitsu.

Sekundärspeicherbereich
Transaktionsgesicherter Speicherbereich, 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 *Terminal-spezifische Langzeitspeicher (TLS)* und der *User-spezifische Langzeitspeicher (ULS)*.

Selektor
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-Systeme)
Betriebsmittel auf Unix- und Windows-Systemen, das zur Steuerung und Synchronisation von Prozessen dient.

Server
Ein Server ist eine *Anwendung*, die *Services* zur Verfügung stellt. Oft bezeichnet man auch den Rechner, auf dem Anwendungen laufen, als Server.

Server-Seite einer Conversation (CPI-C)
Begriff ersetzt durch *Akzeptor*.

Server-Server-Kommunikation
siehe *verteilte Verarbeitung*.

Service Access Point
siehe *Dienstzugriffspunkt.*

Service
Services bearbeiten die *Aufträge*, die an eine Server-Anwendung geschickt werden. Ein Service in einer UTM-Anwendung wird auch *Vorgang* genannt und setzt sich aus einer oder mehreren *Transaktionen* zusammen. Ein Service wird über den *Vorgangs-TAC* aufgerufen. Services können von *Clients* oder anderen Services angefordert werden.

Service-gesteuerte Queue
Message Queue, bei der der Abruf und die Weiterverarbeitung der Nachrichten durch *Services* gesteuert werden. Ein Service muss zum Lesen der Nachricht explizit einen KDCS-Aufruf (DGET) absetzen.
Service-gesteuerte Queues gibt es bei openUTM in den Varianten *USER-Queue*, *TAC-Queue* und *Temporäre Queue*.

Service Routine
siehe *Teilprogramm*.

Session
Kommunikationsbeziehung zweier adressierbarer Einheiten im Netz über das SNA-Protokoll *LU6.1*.

Session-Selektor
Der Session-Selektor identifiziert im lokalen System einen *Zugriffspunkt* zu den Diensten der Kommunikationssteuerschicht (Session-Layer) des *OSI-Referenzmodells*.

Shared Code (BS2000-Systeme)
Code, der von mehreren Prozessen gemeinsam benutzt werden kann.

Shared Memory
Virtueller Speicherbereich, auf den mehrere Prozesse gleichzeitig zugreifen können.

Shared Objects (Unix-/Windows-Systeme)
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

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.

single system image

Unter single system image versteht man die Eigenschaft eines *Clusters*, nach außen hin als ein einziges, in sich geschlossenes System zu erscheinen. Die heterogene Natur des Clusters und die interne Verteilung der Ressourcen im Cluster ist für die Benutzer des Clusters und die Anwendungen, die mit dem Cluster kommunizieren, nicht sichtbar.

SOA

SOA (Service-oriented architecture).
SOA ist ein Konzept für eine Systemarchitektur, in dem Funktionen in Form von wieder verwendbaren, technisch voneinander unabhängigen und fachlich lose gekoppelten *Services* implementiert werden. Services können unabhängig von zugrunde liegenden Implementierungen über Schnittstellen aufgerufen werden, deren Spezifikationen öffentlich und damit vertrauenswürdig sein können. Service-Interaktion findet über eine dafür vorgesehene Kommunikationsinfrastruktur statt.

SOAP

SOAP (Simple Object Access Protocol) ist ein Protokoll, mit dessen Hilfe Daten zwischen Systemen ausgetauscht und Remote Procedure Calls durchgeführt werden können. SOAP stützt sich auf die Dienste anderer Standards, XML zur Repräsentation der Daten und Internet-Protokolle der Transport- und Anwendungsschicht zur Übertragung der Nachrichten.

Socket-Verbindung

Transportsystem-Verbindung, die die Socket-Schnittstelle verwendet. Die Socket-Schnittstelle ist eine Standard-Programmschnittstelle für die Kommunikation über TCP/IP.

stand-alone Anwendung

siehe *stand-alone UTM-Anwendung*.

stand-alone UTM-Anwendung

Herkömmliche *UTM-Anwendung*, die nicht Bestandteil einer *UTM-Cluster-Anwendung* ist.

Standard Primärer Arbeitsbereich/SPAB (KDCS)
Bereich im Arbeitsspeicher, der jedem KDCS-*Teilprogramm* zur Verfügung steht. Sein Inhalt ist zu Beginn des Teilprogrammlaufs undefiniert oder mit einem Füllzeichen vorbelegt.

Startformat
Format, das openUTM am Terminal ausgibt, wenn sich ein Benutzer erfolgreich bei der *UTM-Anwendung* angemeldet hat (ausgenommen nach *Vorgangs-Wiederanlauf* und beim Anmelden über *Anmelde-Vorgang*).

statische Konfiguration
Festlegen der *Konfiguration* bei der Generierung mit Hilfe des UTM-Tools *KDCDEF*.

SYSLOG-Datei
siehe *System-Protokolldatei*.

System-Protokolldatei
Datei oder Dateigeneration, in die openUTM während des Laufs einer *UTM-Anwendung* alle UTM-Meldungen protokolliert, für die das *Meldungsziel* SYSLOG definiert ist.

TAC
siehe *Transaktionscode*.

TAC-Queue
Message Queue, die explizit per KDCDEF-Anweisung generiert wird. Eine TAC-Queue ist eine *Service-gesteuerte Queue* und kann unter dem generierten Namen von jedem Service aus angesprochen werden.

Teilprogramm
UTM-*Services* werden durch ein oder mehrere Teilprogramme realisiert. Die Teilprogramme sind Bestandteile des *Anwendungsprogramms*. Abhängig vom verwendeten API müssen sie KDCS-, XATMI- oder CPIC-Aufrufe enthalten. Sie sind über *Transaktionscodes* ansprechbar. Einem Teilprogramm können mehrere Transaktionscodes zugeordnet werden.

Temporäre Queue
Message Queue, die dynamisch per Programm erzeugt wird und auch wieder per Programm gelöscht werden kann, vgl. *Service-gesteuerte Queue*.

Terminal-spezifischer Langzeitspeicher/TLS (KDCS)
Sekundärspeicher, der einem *LTERM-*, *LPAP-* oder *OSI-LPAP-Partner* zugeordnet ist und über das Anwendungsende hinaus erhalten bleibt.

Timerprozess (Unix-/Windows-Systeme)
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 (Unix-/Windows-Systeme)
Abkürzung für den Transport Name Service, der einem Anwendungsnamen einen Transport-Selektor und das Transportsystem zuordnet, über das die Anwendung erreichbar ist.

Tomcat
siehe *Apache Tomcat*

Transaktion
Verarbeitungsabschnitt innerhalb eines *Services*, für den die Einhaltung der *ACID-Eigenschaften* garantiert wird. 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
Name, über den ein *Teilprogramm* aufgerufen werden kann. Der Transaktionscode wird dem Teilprogramm bei der *statischen* oder *dynamischen Konfiguration* zugeordnet. Einem Teilprogramm können auch mehrere Transaktionscodes zugeordnet werden.

Transaktionsrate
Anzahl der erfolgreich beendeten *Transaktionen* pro Zeiteinheit.

Transfer-Syntax
Bei *OSI TP* werden die Daten zur Übertragung zwischen zwei Rechnersystemen von der lokalen Darstellung in die Transfer-Syntax umgewandelt. Die Transfer-Syntax beschreibt die Daten in einem neutralen Format, das von allen beteiligten Partnern verstanden wird. Jeder Transfer-Syntax muss ein *Object Identifier* zugeordnet sein.

Transport-Selektor
Der Transport-Selektor identifiziert im lokalen System einen *Dienstzugriffspunkt* zur Transportschicht des *OSI-Referenzmodells*.

Transportsystem-Anwendung
Anwendung, die direkt auf einer Transportsystem-Schnittstelle wie z.B. CMX, DCAM oder Socket aufsetzt. Für den Anschluss von Transportsystem-Anwendungen muss bei der *Konfiguration* als Partnertyp APPLI oder SOCKET angegeben werden. Eine Transportsystem-Anwendung kann nicht in eine *Verteilte Transaktion* eingebunden werden.

TS-Anwendung
siehe *Transportsystem-Anwendung.*

Typisierter Puffer (XATMI)
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 UTM-Clients. UPIC steht für Universal Programming Interface for Communication.

UPIC-Client
Bezeichnung für UTM-Clients mit Trägersystem UPIC.

UPIC Analyzer
Komponente zur Analyse der mit *UPIC Capture* mitgeschnittenen UPIC-Kommunikation. Dieser Schritt dient dazu, den Mitschnitt für das Abspielen mit *UPIC Replay* aufzubereiten.

UPIC Capture
Mitschneiden der Kommunikation zwischen UPIC-Clients und UTM-Anwendungen, um sie zu einem späteren Zeitpunkt abspielen zu können (*UPIC Replay*).

UPIC Replay
Komponente zum Abspielen der mit *UPIC Capture* mitgeschnittenen und mit *UPIC Analyzer* aufbereiteten UPIC-Kommunikation.

USER-Queue
Message Queue, die openUTM jeder Benutzerkennung zur Verfügung stellt. Eine USER-Queue zählt zu den *Service-gesteuerten Queues* und ist immer der jeweiligen Benutzerkennung zugeordnet. Der Zugriff von fremden UTM-Benutzern auf die eigene USER-Queue kann eingeschränkt werden.

User-spezifischer Langzeitspeicher/ULS

Sekundärspeicher, der einer *Benutzerkennung*, einer *Session* oder einer *Association* zugeordnet ist und über das Anwendungsende hinaus erhalten bleibt.

USLOG-Datei

siehe *Benutzer-Protokolldatei*.

UTM-Anwendung

Eine UTM-Anwendung stellt *Services* zur Verfügung, die Aufträge von *Clients* oder anderen Anwendungen bearbeiten. openUTM 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-Cluster-Anwendung

UTM-Anwendung, die für den Einsatz in einem *Cluster* generiert ist und die man logisch als **eine** Anwendung betrachten kann.
Physikalisch gesehen besteht eine UTM-Cluster-Anwendung aus mehreren, identisch generierten UTM-Anwendungen, die auf den einzelnen *Knoten* laufen.

UTM-Cluster-Dateien

Oberbegriff für alle Dateien, die für den Ablauf einer UTM-Cluster-Anwendung benötigt werden. Dazu gehören folgende Dateien:

- *Cluster-Konfigurationsdatei*
- *Cluster-User-Datei*
- Dateien des *Cluster-Pagepool*
- *Cluster-GSSB-Datei*
- *Cluster-ULS-Datei*
- Dateien des *Cluster-Administrations-Journals**
- *Cluster-Lock-Datei**
- Lock-Datei zur Start-Serialisierung* (nur bei Unix- und Windows-Systemen)

Die mit * gekennzeichneten Dateien werden beim Start der ersten Knoten-Anwendung angelegt, alle anderen Dateien werden bei der Generierung mit KDCDEF erzeugt.

UTM-D

siehe *openUTM-D*.

UTM-Datenstation

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.
In UTM-Cluster-Anwendungen, die als UTM-F-Anwendung generiert sind (APPLIMODE=FAST), werden Cluster-weit gültige Anwenderdaten auch gesichert. Dabei werden GSSB- und ULS-Daten genauso behandelt wie in UTM-Cluster-Anwendungen, die mit UTM-S generiert sind. Vorgangs-Daten von Benutzern mit RESTART=YES werden jedoch nur beim Abmelden des Benutzers anstatt bei jedem Transaktionsende geschrieben.

UTM-gesteuerte Queues

Message Queues, bei denen der Abruf und die Weiterverarbeitung der Nachrichten vollständig durch openUTM gesteuert werden. Siehe auch *Asynchron-Auftrag, Hintergrund-Auftrag* und *Asynchron-Nachricht*.

UTM-S

Bei UTM-S-Anwendungen sichert openUTM 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-Systeme)

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

Ist eine Speichereinheit, die entweder 2K, 4K oder 8K umfasst. In *stand-alone UTM-Anwendungen* kann die Größe einer UTM-Seite bei der Generierung der UTM-Anwendung auf 2K, 4K oder 8K gesetzt werden. In einer *UTM-Cluster-Anwendung* ist die Größe einer UTM-Seite immer 4K oder 8K. *Pagepool* und Wiederanlauf-Bereich der KDCFILE sowie *UTM-Cluster-Dateien* werden in Einheiten der Größe einer UTM-Seite unterteilt.

UTM-System-Prozess

UTM-Prozess, der zusätzlich zu den per Startparameter angegebenen Prozessen gestartet wird und nur ausgewählte Aufträge bearbeitet. UTM-System-Prozesse dienen dazu, eine UTM-Anwendung auch bei sehr hoher Last reaktionsfähig zu halten.

utmpfad (Unix-/Windows-Systeme)

Das Dateiverzeichnis unter dem die Komponenten von openUTM installiert sind, wird in diesem Handbuch als *utmpfad* bezeichnet.
Um einen korrekten Ablauf von openUTM zu garantieren, muss die Umgebungsvariable UTMPATH auf den Wert von *utmpfad* gesetzt werden. Auf Unix-Systemen müssen Sie UTMPATH vor dem Starten einer UTM-Anwendung setzen, auf Windows-Systemen wird UTMPATH bei der Installation gesetzt.

Verarbeitungsschritt

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 *Dialog-Schritt* oder er endet mit dem Senden einer Dialog-Nachricht an einen Dritten.

Verbindungs-Benutzerkennung

Benutzerkennung, unter der eine *TS-Anwendung* oder ein *UPIC-Client* direkt nach dem Verbindungsaufbau bei der *UTM-Anwendung* angemeldet wird.
Abhängig von der Generierung des Clients (= LTERM-Partner) gilt:

- Die Verbindungs-Benutzerkennung ist gleich dem USER der LTERM-Anweisung (explizite Verbindungs-Benutzerkennung). Eine explizite Verbindungs-Benutzerkennung muss mit einer USER-Anweisung generiert sein und kann nicht als "echte" *Benutzerkennung* verwendet werden.
- Die Verbindungs-Benutzerkennung ist gleich dem LTERM-Partner (implizite Verbindungs-Benutzerkennung), wenn bei der LTERM-Anweisung kein USER angegeben wurde oder wenn ein LTERM-Pool generiert wurde.

In einer *UTM-Cluster-Anwendung* ist der Vorgang einer Verbindungs-Benutzerkennung (RESTART=YES bei LTERM oder USER) an die Verbindung gebunden und damit Knoten-lokal.
Eine Verbindungs-Benutzerkennung, die mit RESTART=YES generiert ist, kann in jeder *Knoten-Anwendung* einen eigenen Vorgang haben.

Verbindungsbündel

siehe *LTERM-Bündel*.

Verschlüsselungsstufe

Die Verschlüsselungsstufe legt fest, ob und inwieweit ein Client Nachrichten und Passwort verschlüsseln muss.

Verteilte Transaktion

Transaktion, die sich über mehr als eine Anwendung erstreckt und in mehreren (Teil)-Transaktionen in verteilten Systemen ausgeführt wird.

Verteilte Transaktionsverarbeitung

Verteilte Verarbeitung mit *verteilten Transaktionen*.

Verteilte Verarbeitung

Bearbeitung von *Dialog-Aufträgen* 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 openUTM-LU62 ist verteilte Verarbeitung auch mit LU6.2 Partnern möglich. Man unterscheidet verteilte Verarbeitung mit *verteilten Transaktionen* (Anwendungs-übergreifende Transaktionssicherung) und verteilte Verarbeitung ohne verteilte Transaktionen (nur lokale Transaktionssicherung). Die verteilte Verarbeitung wird auch Server-Server-Kommunikation genannt.

Vorgang (KDCS)

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*. openUTM stellt den Teilprogrammen eines Vorgangs gemeinsame Datenbereiche zur Verfügung. Anstelle des Begriffs Vorgang wird häufig auch der allgemeinere Begriff *Service* gebraucht.

Vorgangs-Kellerung (KDCS)

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.

Vorgangs-Kettung (KDCS)

Bei Vorgangs-Kettung wird nach Beendigung eines *Dialog-Vorgangs* ohne Angabe einer *Dialog-Nachricht* ein Folgevorgang gestartet.

Vorgangs-TAC (KDCS)

Transaktionscode, mit dem ein *Vorgang* gestartet wird.

Vorgangs-Wiederanlauf (KDCS)

Wird ein Vorgang unterbrochen, z.B. infolge Abmeldens des Terminal-Benutzers oder Beendigung der *UTM-Anwendung*, führt openUTM einen Vorgangs-Wiederanlauf 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 Vorgangs-Wiederanlauf eines Dialog-Vorgangs als *Bildschirm-Wiederanlauf* sichtbar, sofern am letzten Sicherungspunkt eine Dialog-Nachricht an den Terminal-Benutzer gesendet wurde.

Warmstart

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 (*Vorgangs-Wiederanlauf*). Unterbrochene *Asynchron-Vorgänge* werden zurückgesetzt und neu gestartet oder beim zuletzt erreichten *Sicherungspunkt* fortgesetzt. Bei UTM-F-Anwendungen werden beim Start nach einer vorhergehenden abnormalen Beendigung lediglich die dynamisch geänderten Konfigurationsdaten auf den zuletzt erreichten konsistenten Zustand gesetzt.
In UTM-Cluster-Anwendungen werden die globalen Sperren auf GSSB und ULS, die bei der abnormalen Beendigung von dieser Knoten-Anwendung gehalten wurden, aufgehoben. Außerdem werden Benutzer, die zum Zeitpunkt der abnormalen Beendigung an dieser Knoten-Anwendung angemeldet waren, abgemeldet.

Web Service

Anwendung, die auf einem Web-Server läuft und über eine standardisierte und programmatische Schnittstelle (öffentlich) verfügbar ist. Die Web Services-Technologie ermöglicht es, UTM-Teilprogramme für moderne Web-Client-Anwendungen verfügbar zu machen, unabhängig davon, in welcher Programmiersprache sie entwickelt wurden.

WebAdmin

Web-basiertes Tool zur Administration von openUTM-Anwendungen über Web-Browser. WebAdmin enthält neben den kompletten Funktionsumfang der *Programmschnittstelle zur Administration* noch zusätzliche Funktionen.

Wiederanlauf

siehe *Bildschirm-Wiederanlauf*,
siehe *Vorgangs-Wiederanlauf*.

WinAdmin

Java-basiertes Tool zur Administration von openUTM-Anwendungen über eine grafische Oberfläche. WinAdmin enthält neben dem kompletten Funktionsumfang der *Programmschnittstelle zur Administration* noch zusätzliche Funktionen.

Workload Capture & Replay

Programmfamilie zur Simulation von Lastsituationen, bestehend aus den Haupt-Komponenten *UPIC Capture*, *UPIC Analyzer* und *Upic Replay* und auf Unix- und Windows-Systemen) dem Dienstprogramm *kdcsort*. Mit Workload Capture & Replay lassen sich UPIC-Sessions mit UTM-Anwendungen aufzeichnen, analysieren und mit veränderten Lastparametern wieder abspielen.

Workprozess (Unix-/Windows-Systeme)

Prozess, in dem die *Services* der *UTM-Anwendung* ablaufen.

WS4UTM

WS4UTM (**W**eb**S**ervices for open**UTM**) ermöglicht es Ihnen, auf komfortable Weise einen Service einer UTM-Anwendung als Web Service zur Verfügung zu stellen.

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 openUTM implementierte XATMI genügt der XATMI CAE Specification von X/Open. Die Schnittstelle steht in COBOL und C zur Verfügung. XATMI in openUTM kann über die Protokolle *OSI TP*, *LU6.1* und *UPIC* kommunizieren.

XHCS (BS2000-Systeme)

XHCS (Extended Host Code Support) ist ein BS2000-Softwareprodukt für die Unterstützung internationaler Zeichensätze.

XML

XML (eXtensible Markup Language) ist eine vom W3C (WWW-Konsortium) genormte Metasprache, in der Austauschformate für Daten und zugehörige Informationen definiert werden können.

Zeitgesteuerter Auftrag

Auftrag, der von openUTM bis zu einem definierten Zeitpunkt in einer *Message Queue* zwischengespeichert und dann an den Empfänger weitergeleitet wird. Empfänger kann sein: ein *Asynchron-Vorgang* der selben Anwendung, eine *TAC-Queue*, eine Partner-Anwendung, ein Terminal oder ein Drucker.
Zeitgesteuerte Aufträge können nur von KDCS-*Teilprogrammen* erteilt werden.

Zugangskontrolle

Prüfung durch openUTM, ob eine bestimmte *Benutzerkennung* berechtigt ist, mit der *UTM-Anwendung* zu arbeiten. Die Berechtigungsprüfung entfällt, wenn die UTM-Anwendung ohne Benutzerkennungen generiert wurde.

Zugriffskontrolle

Prüfung durch openUTM, ob der Kommunikationspartner berechtigt ist, auf ein bestimmtes Objekt der Anwendung zuzugreifen. Die Zugriffsrechte werden als Bestandteil der Konfiguration festgelegt.

Zugriffspunkt

siehe *Dienstzugriffspunkt*.

Abkürzungen

ACSE	Association Control Service Element
AEQ	Application Entity Qualifier
AES	Advanced Encryption Standard
AET	Application Entity Title
APT	Application Process Title
ASCII	American Standard Code for Information Interchange
ASE	Application Service Element
Axis	Apache eXtensible Interaction System
BCAM	Basic Communication Access Method
BER	Basic Encoding Rules
BLS	Binder-Lader-Starter (BS2000)
CCP	Communication Control Program
CCR	Commitment, Concurrency and Recovery
CCS	Codierter Zeichensatz (Coded Character Set)
CCSN	Name des codierten Zeichensatzes (Coded Character Set Name)
CICS	Customer Information Control System (IBM)
CID	Control Identification
CMX	Communication Manager in Unix-Systemen
COM	Component Object Model
CPI-C	Common Programming Interface for Communication
CRM	Communication Resource Manager
CRTE	Common Runtime Environment (BS2000)
DB	Datenbank
DC	Data Communication
DCAM	Data Communication Access Method

DES	Data Encryption Standard
DLM	Distributed Lock Manager (BS2000)
DMS	Data Management System
DNS	Domain Name Service
DSS	Datensichtstation (=Terminal)
DTD	Document Type Definition
DTP	Distributed Transaction Processing
DVS	Datenverwaltungssystem
EBCDIC	Extended Binary-Coded Decimal Interchange Code
EJB	Enterprise JavaBeans™
FGG	File Generation Group
FHS	Format Handling System
FT	File Transfer
GSSB	Globaler Sekundärer Speicherbereich
HIPLEX®	Highly Integrated System Complex (BS2000)
HLL	High-Level Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IFG	Interaktiver Format-Generator
ILCS	Inter Language Communication Services (BS2000)
IMS	Information Management System (IBM)
IPC	Inter-Process-Communication
IRV	Internationale Referenzversion
ISO	International Organization for Standardization
Java EE	Java Platform, Enterprise Edition
JCA	Java EE Connector Architecture
JDK	Java Development Kit
KAA	KDCS Application Area
KB	Kommunikationsbereich
KBPROG	KB-Programmbereich
KDCADMI	KDC Administration Interface
KDCS	Kompatible Datenkommunikationsschnittstelle

KTA	KDCS Task Area
LAN	Local Area Network
LCF	Local Configuration File
LLM	Link and Load Module (BS2000)
LSSB	Lokaler Sekundärer Speicherbereich
LU	Logical Unit
MQ	Message Queuing
MSCF	Multiple System Control Facility (BS2000)
NB	Nachrichtenbereich
NEA	Netzwerkarchitektur bei BS2000-Systemen
NFS	Network File System/Service
NLS	Unterstützung der Landessprache (Native Language Support)
OLTP	Online Transaction Processing
OML	Object Modul Library
OSI	Open System Interconnection
OSI TP	Open System Interconnection Transaction Processing
OSS	OSI Session Service
PCMX	Portable Communication Manager
PID	Prozess-Identifikation
PIN	Persönliche Identifikationsnummer
PLU	Primary Logical Unit
PTC	Prepare to commit
RAV	Rechenzentrums-Abrechnungs-Verfahren
RDF	Resource Definition File
RM	Resource Manager
RSA	Encryption-Algorithmus nach Rivest, Shamir, Adleman
RSO	Remote SPOOL Output (BS2000)
RTS	Runtime System (Laufzeitsystem)
SAT	Security Audit Trail (BS2000)
SECOS	Security Control System
SEM	SE Manager
SGML	Standard Generalized Markup Language

SLU	Secondary Logical Unit
SM2	Software Monitor 2
SNA	Systems Network Architecture
SOA	Service-oriented Architecture
SOAP	Simple Object Access Protocol
SPAB	Standard Primärer Arbeitsbereich
SQL	Structured Query Language
SSB	Sekundärer Speicherbereich
SSO	Single-Sign-On
TAC	Transaktionscode
TCEP	Transport Connection End Point
TCP/IP	Transport Control Protocol / Internet Protocol
TIAM	Terminal Interactive Access Method
TLS	Terminal-spezifischer Langzeitspeicher
TM	Transaction Manager
TNS	Transport Name Service
TP	Transaction Processing (Transaktions-Betrieb)
TPR	Task privileged (privilegierter Funktionszustand des BS2000)
TPSU	Transaction Protocol Service User
TSAP	Transport Service Access Point
TSN	Task Sequence Number
TU	Task user (nicht privilegierter Funktionszustand des BS2000)
TX	Transaction Demarcation (X/Open)
UDDI	Universal Description, Discovery and Integration
UDS	Universelles Datenbanksystem
UDT	Unstructured Data Transfer
ULS	User-spezifischer Langzeitspeicher
UPIC	Universal Programming Interface for Communication
USP	UTM-Socket-Protokoll
UTM	Universeller Transaktionsmonitor
UTM-D	UTM-Funktionen für verteilte Verarbeitung („Distributed“)
UTM-F	Schnelle UTM-Variante („Fast“)
UTM-S	UTM-Sicherheitsvariante

UTM-XML	XML-Schnittstelle von openUTM
VGID	Vorgangs-Identifikation
VTSU	Virtual Terminal Support
VTV	Verteilte Transaktionsverarbeitung
VV	Verteilte Verarbeitung
WAN	Wide Area Network
WS4UTM	WebServices for openUTM
WSDD	Web Service Deployment Descriptor
WSDL	Web Services Description Language
XA	X/Open Access Interface (Schnittstelle von X/Open zum Zugriff auf Resource Manager)
XAP	X/OPEN ACSE/Presentation programming interface
XAP-TP	X/OPEN ACSE/Presentation programming interface Transaction Processing extension
XATMI	X/Open Application Transaction Manager Interface
XCS	Cross Coupled System
XHCS	eXtended Host Code Support
XML	eXtensible Markup Language

Literatur

Die Handbücher finden Sie im Internet unter *http://manuals.ts.fujitsu.com*. Handbücher, die mit einer Bestellnummer angezeigt werden, können Sie auch in gedruckter Form bestellen.

PDF-Dateien von allen openUTM-Handbüchern sind sowohl auf der openUTM Enterprise Edition DVD für die offenen Plattformen als auch für BS2000-Systeme auf der openUTM WinAdmin-DVD enthalten.

Dokumentation zu openUTM

openUTM
Konzepte und Funktionen
Benutzerhandbuch

openUTM
Anwendungen programmieren mit KDCS für COBOL, C und C++
Basishandbuch

openUTM
Anwendungen generieren
Benutzerhandbuch

openUTM
Einsatz von openUTM-Anwendungen unter BS2000-Systemen
Benutzerhandbuch

openUTM
Einsatz von openUTM-Anwendungen unter Unix- und Windows-Systemen
Benutzerhandbuch

openUTM
Anwendungen administrieren
Benutzerhandbuch

openUTM
Meldungen, Test und Diagnose in BS2000-Systemen
Benutzerhandbuch

openUTM
Meldungen, Test und Diagnose in Unix- und Windows-Systemen
Benutzerhandbuch

openUTM
Anwendungen erstellen mit X/Open-Schnittstellen
Benutzerhandbuch

openUTM
XML für openUTM

openUTM-Client (Unix-Systeme)
für Trägersystem OpenCPIC
Client-Server-Kommunikation mit openUTM
Benutzerhandbuch

openUTM-Client
für Trägersystem UPIC
Client-Server-Kommunikation mit openUTM
Benutzerhandbuch

openUTM WinAdmin
Grafischer Administrationsarbeitsplatz für openUTM
Beschreibung und Online-Hilfe

openUTM WebAdmin
Web-Oberfläche zur Administration von openUTM
Beschreibung und Online-Hilfe

openUTM, **openUTM-LU62**
Verteilte Transaktionsverarbeitung
zwischen openUTM und CICS-, IMS- und LU6.2-Anwendungen
Benutzerhandbuch

openUTM (BS2000)
Anwendungen programmieren mit KDCS für Assembler
Ergänzung zum Basishandbuch

openUTM (BS2000)
Anwendungen programmieren mit KDCS für Fortran
Ergänzung zum Basishandbuch

openUTM (BS2000)
Anwendungen programmieren mit KDCS für Pascal-XT
Ergänzung zum Basishandbuch

openUTM (BS2000)
Anwendungen programmieren mit KDCS für PL/I
Ergänzung zum Basishandbuch

WS4UTM (Unix- und Windows-Systeme**)**
Web-Services für openUTM

openUTM
Masterindex

Dokumentation zum openSEAS-Produktumfeld

BeanConnect
Benutzerhandbuch

JConnect
Verbindung von Java-Clients zu openUTM
Benutzerdokumentation und Java-Docs

WebTransactions
Konzepte und Funktionen

WebTransactions
Template-Sprache

WebTransactions
Anschluss an openUTM-Anwendungen über UPIC

WebTransactions
Anschluss an MVS-Anwendungen

WebTransactions
Anschluss an OSD-Anwendungen

Dokumentation zum BS2000-Umfeld

AID
Advanced Interactive Debugger
Basishandbuch
Benutzerhandbuch

BCAM
BCAM Band 1/2
Benutzerhandbuch

BINDER
Benutzerhandbuch

BS2000 OSD/BC
Makroaufrufe an den Ablaufteil
Benutzerhandbuch

BS2000
BLSSERV
Bindelader-Starter
Benutzerhandbuch

DCAM
COBOL-Aufrufe
Benutzerhandbuch

DCAM
Makroaufrufe
Benutzerhandbuch

DCAM
Programmschnittstellen
Beschreibung

FHS
Formatierungssystem für openUTM, TIAM, DCAM
Benutzerhandbuch

IFG für FHS
Benutzerhandbuch

HIPLEX AF
Hochverfügbarkeit von Anwendungen in BS2000/OSD
Produkthandbuch

HIPLEX MSCF
BS2000-Rechner im Verbund
Benutzerhandbuch

IMON
Installationsmonitor
Benutzerhandbuch

LMS
SDF-Format
Benutzerhandbuch

MT9750 (MS Windows)
9750-Emulation unter Windows
Produkthandbuch

OMNIS/OMNIS-MENU (BS2000)
Funktionen und Kommandos
Benutzerhandbuch

OMNIS/OMNIS-MENU (BS2000)
Administration und Programmierung
Benutzerhandbuch

OSS (BS2000)
OSI Session Service
User Guide

RSO
Remote SPOOL Output
Benutzerhandbuch

SECOS
Security Control System
Benutzerhandbuch

SECOS
Security Control System
Tabellenheft

SESAM/SQL
Datenbankbetrieb
Benutzerhandbuch

openSM2
Software Monitor
Band 1: Verwaltung und Bedienung

TIAM
Benutzerhandbuch

UDS/SQL
Datenbankbetrieb
Benutzerhandbuch

Unicode im BS2000/OSD
Übersichtshandbuch

VTSU
Virtual Terminal Support
Benutzerhandbuch

XHCS
8-bit-Code- und Unicode-Unterstützung im BS2000/OSD
Benutzerhandbuch

Dokumentation zum Umfeld von Unix-Systemen

CMX V6.0 (Unix-Systeme)
Betrieb und Administration
Benutzerhandbuch

CMX V6.0
CMX-Anwendungen programmieren
Programmierhandbuch

OSS (UNIX)
OSI Session Service
User Guide

PRIMECLUSTER*TM*
Konzept (Solaris, Linux)
Benutzerhandbuch

openSM2
Die Dokumentation zu openSM2 wird in Form von ausführlichen Online-Hilfen bereitgestellt, die mit dem Produkt ausgeliefert werden.

Sonstige Literatur

XCPI-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

XML
Spezifikation des W3C (www – Konsortium)
Webseite: *http://www.w3.org/XML*

Stichwörter

64-bit-Plattform 255
64-Bit-Umgebung
  Umstieg 163
8-Bit-Code 244

A
Abnormale Beendigung 222
Abrechnung 60
Access-List-Konzept 192
  bei verteilter Verarbeitung 195
Accounting 60
ACID-Eigenschaften 29
  Atomicity 29
  Consistency 29
  Durability 29
  Isolation 29
adjtclt.c 172
Administration 165
  Asynchron-Auftrag 117
  automatisch 181
  Cluster-Anwendung über WinAdmin 36
  im Dialog 167
  mit WebAdmin 176
  mit WinAdmin 173
  Security-Funktionen 178
  über Message Queuing 167
  von Druckern 182
  von Message Queues 182
Administrations-Journal 38, 276
Administrationsfunktionen
  Message Queuing 167
  Programmschnittstelle KDCADMI 171
Administrationskommando 167
Administrationsprogramm
  KDCADM 166
  Transaktionscodes 168
Adressierung
  ferner Services 93
Adressraumkonzept
  BS2000-System 236
  Unix-Systeme 253
  Windows-Systeme 263
AES-Verschlüsselung 200
Alarm-Mechanismen 43
Anmelden
  Transportsystem-Anwendung 98
Anmeldevorgang 189
Anpassen der Konfiguration
  dynamisch 58
Anschlusspunkt
  logisch 184, 195
Antwortzeit 53
Anwendungen
  integrieren 69
Anwendungs-übergreifende Dialoge 87
Anwendungsentwicklung 63
Anwendungsprogramm 51
  erzeugen 159
Anwendungsüberwachung
  im Cluster 210
AREA 139
Assembler 132
Asynchron-Auftrag 103
  Administration 117
  Basisauftrag 115
  Quittungsauftrag 115
  Steuerungsmöglichkeiten 115
  Zeitsteuerung 116
Asynchron-Nachricht 103

Asynchron-Service 104, 105
Asynchron-Vorgang 104
Asynchrone Verarbeitung 42
Asynchrones Request-Response-Modell 147
Atomicity 29
Auftraggeber-Service 87
Auftragnehmer-Service 87
Auftrags-Komplex 115
Ausfallerkennung im Cluster 210
Ausgabeauftrag 104
Auslastung ermitteln 61
Ausweisprüfung 188
Authentisierung 56, 184
Automatischer Verbindungsabbau 189
Autorisierung 56, 190
Axis 74

B

BADTACS 142
Basisauftrag 115
BeanConnect 70, 71, 73
BeanConnect Proxy 71, 72
Beendigung, abnormal 222
Benutzer-Protokolldatei 140
Benutzeradressraum 236
Benutzerkennung 186
  bei Client-Programmen 186
  bei Terminals 186
Berechtigungskonzept 178
  Administration 178
Berechtigungsprüfung
  selbst programmiert 189
Betriebsmittel
  Leistung ermitteln 61
Betriebssystemfehler 220
Borland Delphi 133
Browsen, Queue-Nachrichten 109
BS2000-Task 235
Business-Logik 28
Bypass-Betrieb 241

C

C 132
C++ 132
Cache-Speicher 129
CALLUTM 246
Chipkarte 243
CICS 46, 96
CICS/6000 96
Client
  Begriffsklärung 81
  sicherer 201
Client-Anwendung 81
  Anschluss 82
Client-Programm 82
Client-Prozess
  Unix-Systeme 250
  Windows-Systeme 260
Client/Server-Computing 77
  Architekturvarianten 77
  Basismodelle 78
  Begriffsklärungen 81
Client/Server-Kommunikation
  Benutzerkonzept 186
Cluster
  Administrations-Journal 38
  Anwendungsprogramme ändern 212
  Cluster-Konfigurationsdatei 37
  Cluster-User-Datei 38
  Hochverfügbarkeit 209
  Knoten-Anwendung 34
  Lastverteilung 215
  Online-Import von Anwendungsdaten 211, 212
  Online-Update 212
  openUTM 209
  Überwachung 210
  UTM-Cluster-Anwendung 34
  UTM-Cluster-Dateien 37
  UTM-Korrekturstufen 213
  Windows-Systeme 208
Cluster-Administrations-Journal 276
Cluster-Anwendung
  Knoten-lokale Dateien 38
  Online-Update 212
Cluster-Konfigurationsdatei 37
Cluster-Pagepool 38
Cluster-Update 162

Cluster-User-Datei 38
CMX-Anwendung 98
COBOL 132
Code-Umsetzung
  BS2000-System 240
  Unix-Systeme 255, 265
Common Memory Pool
  BS2000-System 236
Communication Resource Manager 51
Consistency 29
Conversation
  non-blocking 42
  Pseudo 125
Conversational Modell 147
CPI-C 132, 145
CPU-Verbrauch
  abrechnen 60
CUpic 133

D

DADM 120
Data Warehouse-Lösung 44
Datenbank-System 30
  Diagnose 33
  Fehlerbehandlung 33
  heterogen 32
  koordinierte Zusammenarbeit 31
  Schnittstelle 33
  unterstützt 30
  verteilt 32
Datenbankschlüssel 243
DBH im Cluster 40
DCAM-Anwendung 98
Dead Letter Queue 105, 111, 117
Decision Support-System 44
Deferred Delivery-Prinzip 107
Diagnosemöglichkeit 62
Dialog
  Anwendungs-übergreifend 87
  beim Administrieren 167
Dialogterminalprozess
  Unix-Systeme 250
  Windows-Systeme 260
DIN-Norm 66 265 267
Distributed Transaction Processing (DTP) 50
Document Type Definition 150
Dokumentation, Wegweiser 14
DPUT 119
Drucker
  Anschluss im BS2000 241
  Drucker Sharing 242
  RSO 241
Druckerbündel
  Störungen 221
Druckerprozess 250
DTD 150
Dump-Dateien 62
Durability 29
Durchsatz 53
Dynamische Administration siehe Dynamische Konfigurierung
Dynamische Erweiterung
  Konfiguration 179

E

Einstufige Adressierung 93
encradm.c 172
Erneutes Zustellen
  Nachricht an Message Queue 117
Erweiterter Zeichensatz 244
Event-Exit
  FORMAT 144, 239
  INPUT 143
  SHUT 143
  START 143
  VORGANG 143
Event-Funktion 142
Event-Service 142
  BADTACS 142
  MSGTAC 142, 181
  SIGNON 142, 189

F

Fehler
  automatische Prüfungen 219
  Eingrenzung 217
Fehlerbehandlung
  bei verteilter Verarbeitung 227

Fehlertoleranz 217
Festpreis, für Service 60
FGET 119
FHS 238
FORMAT 144, 239
Formatgenerator IFG 238
Formatierung
  BS2000-System 238
Formatierungsfehler
  Eingrenzung 217
Formatierungssystem FHS 238
Formatsteuerung FHS 238
Fortran 132
FPUT 119
Funktionen zur Druckausgabe 182

G
Generierung 155
  dynamisch anpassen 58
Generierungsschnittstelle 155
Generierungstool KDCDEF 156
Globale Transaktion
  Rücksetzen 227
  Wiederanlauf 227
Globale Transaktionssicherung
  CICS und IMS 96
  unterschiedliche Schnittstellen 133
Globaler Sekundär-Speicherbereich 139
Grafische Oberfläche
  zur Administration 173, 177
Grafisches Administrationsprogramm
  WebAdmin 176
GSSB 139

H
Hardcopy-Betrieb 241
Hardwarefehler bei Terminals 220
Hintergrund-Auftrag
  an fernen Service 107
  Priority Scheduling 108
HIPLEX AF 205
hndlusr.c 172
Hochverfügbarkeit 55
  BS2000 205
  mit Cluster 209
  stand-alone openUTM 205
  Unix-Systeme 207
  Windows-Systeme 208
HP-UX 13

I
IBM-Mainframe
  Integration 46
Identifizierung 184
IFG 238
ILCS 233
IMS 46, 96
INPUT 143
Integration von Anwendungen 69
Inter Process Communication 253, 263
Internationalisierung 59
  BS2000-System 243
Inverser KDCDEF 170
IP-Adresse 255, 265
IPC 253, 263
ISO/IEC 267
Isolation 29
IUTMDB 33, 233
IUTMFORM 233, 239
IUTMHLL 233

J
J2EE Application Server 70
Java EE 70
Java EE Application Server 70
  Inbound-Kommunikation 72
Java EE Server
  Kommunikation mit UTM-Cluster-Anwendung 73
Java Enterprise Edition 70
Java-Clients 84
JCA-Adapter
  JConnect 71
JConnect-Adapter 71

K
KAA 253, 263
KB 138

KDCADM 166
KDCADMI 165, 171
  Beispielprogramme 172
KDCDADM 182
KDCDEF 126
  invers 170
KDCDEF-Steueranweisungen
  Übersicht 157
KDCDUMP 62
KDCFILE 126, 156
KDCMMOD 181
KDCMON 53
KDCPADM 182
KDCROOT 123, 159
KDCS 134
  Aufrufe 134
  Returncode 137
  Speicherbereiche 138
KDCS-Schnittstelle
  MQ-Aufrufe 119
KDCUPD 161
Kerberos 188, 242
Klasse-4-Speicher 236
Klasse-5-Speicher 236
Klasse-6-Speicher 236
Knoten-Anwendung 34
Knoten-lokale Daten 128
Knoten-Recovery 213
Knoten-Update 162
Kollektionen 174
Kombination
  Programmschnittstellen 133
Kommunikation
  Peer-to-Peer-Kommunikation 81
  Server-Server-Kommunikation 81
Kommunikationsbereich 138
Kommunikationsmodell
  XATMI 147
Kommunikationsprotokoll 48
  Übersicht 48, 101
Komponententechnologie 70
Konfiguration 156
  definieren 156
  dynamisch ändern 170
  dynamisch erweitern 179
Kopplungsmöglichkeiten 48
KTA 253, 263

L

Last-Situation
  simulieren 54
Lastverteilung
  Cluster 215
  Oracle® RAC (Cluster) 216
  UPIC-Clients (Cluster) 216
  verteilte Verarbeitung (Cluster) 215
Leistungsüberblick 27
Linux-Distribution 13
LLM (BS2000-System) 234
Local Queuing 104
Locale 243
Lock-/Keycode-Konzept 56
  bei verteilter Verarbeitung 195
  Beispiel 191
Lokaler Sekundärspeicherbereich 139
LPAP-Partner 185
LSSB 139
LTERM-Partner 184
  privilegiert 125
LTERM-Pool 185
LU6.1 48, 96, 101
LU6.2 49, 96
LU6.2-Gateway 46

M

Mailbox-Systeme 43
Main Routine KDCROOT 123, 159
Main-Prozess
  Unix-Systeme 249
  Windows-Systeme 258
Man-in-the-middle 200
MCOM 119
Mehrfach benutzbar 236
Meldungen anpassen 59
Memory Mapped File 263
Message Queuing 42, 103
  für Administration 167
  KDCS-Aufrufe 119

MessageDriven Beans 72
MIB 68
Microsoft Cluster Server 208
Mobile Computing 43
MQ-Aufruf 119
  DADM 120
  DPUT 119
  FGET 119
  FPUT 119
  MCOM 119
MSCF 279
MSGTAC 142, 181
Multi-Connect 98
  UPIC 83
Multi-Conversations
  UPIC 83
Multi-Signon 98
  OpenCPIC 83
  UPIC 83
Multi-Tier-Architektur 80
Multithreading 53
Multivendor-Konfiguration 45

N
Netzanbindung
  Unix-Systeme 255
  Windows-Systeme 265
Netzprotokoll 48
Netzprozess
  Unix-Systeme 250
  Windows-Systeme 259
Netzstörung 220
Non-blocking conversation 42
Normen 267

O
Objektsicht beim Administrieren 175
Öffentlicher Schlüssel 200
OLTP-Anwendungen 27
OMNIS 245
Online-Import
  von Anwendungsdaten (Cluster) 211, 212
Online-Update
  Cluster-Anwendung 212
OpenCPIC 83
OpenCPIC-Client 83
openSM2 61
openUTM
  Leistungsüberblick 27
  Plattformunabhängigkeit 45
  X/Open-Konformität 50
openUTM WinAdmin 64, 66
openUTM-Anwendung, siehe UTM-Anwendung
openUTM-Client 81
openUTM-LU62 46, 96
openUTM-Systemschnittstellen
  BS2000 233
  Unix-Systeme 248
Oracle WebLogic Server 70
Oracle® RAC
  Lastverteilung (Cluster) 216
OSI TP 46, 49, 101
OSI TP-Kommunikation
  mit Java EE Server 72
OSI-LPAP-Partner 185
Output Queuing 104

P
Pagepool 127
Paging area 236
Parallelbetrieb
  BS2000-System 233
Pascal XT 132
Passwort 186
  bei Client-Programmen 186
  bei Terminals 186
  Wiederanlauf 226
PCMX 18
Performance 53
PL/1 132
Plattenausfall 220
Plattformen
  unterstützte 45
Plattformunabhängigkeit 45
Portabilität 46
Primärspeicherbereich 138
PRIMECLUSTER 207
PRIMERGY ServerShield 208

Prioritätensteuerung 108
Priority Scheduling 108
Private key 200
Privater Schlüssel 200
Privilegiert
  LTERM-Partner 125
Programmfehler
  Eingrenzung 217
Programmschnittstelle
  Clients 133
  CPI-C 145
  KDCADMI 170
  KDCS 131
  TX 149
  XATMI 147
  zur Administration 170
Prozessbeschränkung 108
Prozesse
  BS2000-System 235
  im BS2000 235
  Windows 258
Prozesskommunikation 253, 263
Prozesskonzept 124
Pseudo-Conversation 125
Pseudo-Dialog
  IMS und CICS 97
Public key 200
Puffer
  typisierte 148

Q

Quittungsauftrag 115

R

Red Hat 13
Redelivery 117
  Hintergrundaufträge 105
  Service-gesteuerte Queues 109
Remote Queuing 104, 107
Remote Spool Output 241
Reply-Queue 112
Request-Response-Modell
  asynchron 147
  synchron 147
Resource Manager 30, 51
  Dateisystem 33
  Logging-Datei 33
  lokaler Speicher 33
  Message Queue 33
Restart, siehe Wiederanlauf
Returncode, KDCS-Aufruf 137
RFC1006-Kommunikation
  mit Java EE Server 72
RFC1510 188
Rollback, siehe Rücksetzen
ROOT-Tabelle 156
ROOTDATA 253, 263
RSA 200
RSO-Drucker 241
Rücksetzen
  globaleTransaktion 227
  lokale Transaktion 229
Run Priorität 242

S

SAT-Protokollierung 242
Schnittstellen-Kombinationen 133
SDF-Schnittstelle
  für UTM-Tools 246
SECOS 188
Security-Funktionen 183
  Administration 178
  bei verteilter Verarbeitung 195
  BS2000-System 242
  externe Resource Manager 202
  Überblick 56
Sekundärspeicherbereich 139
Server
  Begriffsklärung 81
Server-Anwendung 81
Server-Server-Kommunikation 87
  Benutzerkonzept 187
ServerShield 208
Service
  Adressierung 93
  asynchron 105
  ereignisgesteuert 142
  Verschlüsselung 201

Wiederanlauf 224
Service Oriented Architecture
SOA 69
Service-gesteuerte Queue 42, 103, 109
Service-Hierarchie 88
Service-Prozess 259
Service-Routine 122
SESAM/SQL im Cluster 40
SESDCN
im Cluster 41
Session 228
Wiederanlauf 228
SGML 150
Shareable 236
Shared Memory 253
SHUT 143
Shutdownprozess 260
Sicherer Client 201
Sicherungspunkt 29
SIGNON 142, 189
Transportsystem-Anwendungen 98
SM2 61
SNA-Anbindung 96
SNA-Protokoll 96
SNMP-Subagent
für openUTM 68
SOAP 74
Socket-Anwendung 98
Socket-Kommunikation
mit Java EE Server 72
Software Monitor SM2 61
Solaris 13
Source-Korrekturverfahren 234
SPAB 138
Speicherbereich
Übersicht 140
Speicherklassen im BS2000 236
Speicherstruktur im BS2000 237
Spool-Betrieb 241
Stand-alone UTM-Anwendung 11
Hochverfügbarkeit 205
Standard-Administrationsprogramm 166
Standardprimärer Arbeitsbereich 138
Standards 267
START 143
Startparameter TASKS 125
Steuerung der Druckausgabe 182
Stiller Alarm 188
Störung
ohne Verbindungsverlust 221
Subsystem 232
Sun Solaris 255
64-bit 255
SUSE 13
susrmax.c 172
Synchrones Request-Response-Modell 147
Synchrones Warten 112
SYSLOG-Datei 62
Systemeinbettung
BS2000-System 231
Unix-Systeme 247
Windows-Systeme 257
Systemfunktion 232
Systemschnittstellen
BS2000 233
Unix-Systeme 248
Windows-Systeme 258

T

TAC-Klassen 108
TAC-Queues 42, 109, 111
Tasks 235
Teilprogramm 122
Temporäre Queues 42, 109, 112
Terminal
Benutzerkonzept 186
unmittelbarer Anschluss 47
Terminal-spezifischer Langzeitspeicher 139
Tier 77
Timerprozess
Unix-Systeme 249
Windows-Systeme 259
TLS 139
Tomcat 74
Trace Merging 54
Trace-Datei 62
Trägersystem 82
OpenCPIC 83

UPIC 82
Transaction Manager 51
Transaktion
Ende 90
global 90
lokal 90
rücksetzen 224
Synchronisierung 90
unabhängig 90
Transaktions-Logging 223
Transaktionscode 122
Standard-Administrationsprogramm 168
Transaktionskonzept 29
Transaktionssicherung 90
Transportsystem-Anwendung 98
Trusted Client 201
Tuxedo 46
Two-Phase-Commit 31, 90
TX 83, 132, 149
Typisierte Puffer
Aufrufe 148

U

UDS-D im Cluster 41
UDS/SQL im Cluster 40
ULS 140
Unabhängige Transaktion 90
siehe auch lokale Transaktion
Unicode 59
Unicode-Zeichensätze 244
UNISYS 46
Unix-Plattform 13
Unix-Systeme 247
UPIC 49, 82, 101
UPIC Capture 54
UPIC-Client 82
Lastverteilung (Cluster) 216
UpicAnalyzer 54
UpicReplay 54
User-Logging-Datei USLOG 140
USER-Queues 42, 109, 110
User-spezifischer Langzeitspeicher 140
USLOG 140
USP 99
UTFE 244
UTM Socket Protokoll 99
UTM-Anwendung
generieren 155
Struktur 121
Wiederanlauf 223
UTM-Anwendungsprogramm 122
erzeugen 159
UTM-Aufruf 122
UTM-Cache 129
UTM-Cluster-Anwendung 11, 34, 209
Administration über WinAdmin 36
als Server 72
Cluster-Administrations-Journal 276
Überwachung 210
UTM-Cluster-Dateien 37
UTM-Dump 33
UTM-F 223
Wiederanlauf 226
UTM-gesteuerte Queues 42, 103, 104
UTM-Messmonitor 53
UTM-S 223
Wiederanlauf 224
UTM-System-Prozesse 125
UTM-Systemcode 231
UTM-Tools
SDF-Schnittstelle 246
UTM-XML 132, 150

V

Verarbeiten
Queue-Nachrichten 109
Verarbeitung
asynchron 42
Verbindungsabbau
automatisch 189
Verbindungspasswort 242
Verbindungsverlust
bei Terminals 220
bei verteilter Verarbeitung 228
Verschlüsselung 200
Terminalemulation 243
UPIC-Clients 200
Verschlüsselungsebene 200

für Clients 201
für Services 201
Verteilte Verarbeitung 81
Lastverteilung mit Cluster 215
VerteilteTransaktion 90
Visual Basic 82, 133
Visual C++ 82
VORGANG 143
Vorgang 81
VTSU-B
Verschlüsselung 243

W

Warmstart 224
UTM-F 226
UTM-S 224
Web 75
Web Services 74
WebAdmin 66
administrieren mit 176
WebSphere 70
WebTransactions 75
Wiederanlauf 223
bei verteilter Verarbeitung 228
globale Transaktion 227
lokale Transaktion 229
Service 224
Session 228
UTM-F 226
UTM-S 224
Wiederanlaufbereich 127
WinAdmin 64
Administration einer UTM-Cluster-Anwendung 36
administrieren mit 173
Windows-System 13
Workflow-Strategie 43
Workload Capture & Replay 54
Workprozess
Unix-Systeme 249
Windows-Systeme 259
World Wide Web 46
WS4UTM 74

X

X/Open 132, 267
X/Open-Konformität 50
Vorteile 52
X/Open-Modell 50
X/Open-Schnittstelle
CPI-C 145
TX 149
XATMI 147
x86-Hardware 232
XA-Schnittstelle 33
XAP-TP 268
XATMI 132, 147
XHCS-Unterstützung 244
XML 132, 150
C/C++-Aufrufe 151
COBOL-Schnittstelle 152
XML-Schema 150

Z

Zeichensatz
erweitert 244
Zeitgeberprozess
Unix-Systeme 249
Windows-Systeme 259
Zeitsteuerung 116
Zertifikatsnummer 243
Zertifikatsprüfung 243
Zugangskontrolle 56, 184
durch Kerberos 242
Zugangsschutz
Zertifikatsprüfung 243
Zugriffskontrolle 56, 190
Lock-/Keycode-Konzept 190
Zweistufige Adressierung 93