Deutsch

FUJITSU Software

openUTM V6.3

Anwendungen erstellen mit X/Open-Schnittstellen

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

Copyright © 2015 Fujitsu Technology Solutions GmbH.

Alle Rechte vorbehalten.
Liefermöglichkeiten und technische Änderungen vorbehalten.

Alle verwendeten Hard- und Softwarenamen sind Handelsnamen und/oder Warenzeichen der jeweiligen Hersteller.

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.

1.1 Konzept und Zielgruppen dieses Handbuchs

Das vorliegende Handbuch „Anwendungen erstellen mit X/Open-Schnittstellen“ beschreibt die Besonderheiten der X/Open-Schnittstellen CPI-C, TX und XATMI in openUTM-Anwendungen. Die Beschreibung richtet sich an Programmierer, die für openUTM-Anwendungen X/Open-Schnittstellen verwenden wollen.

Dieses Handbuch ist als Ergänzung der folgenden vier X/Open-Spezifikationen zum Thema „Distributed Transaction Processing“ konzipiert (die genauen Titelangaben finden Sie im nächsten Abschnitt):

– „Reference Model Version 2“

– „The CPI-C Specification, Version 2“

– „The XATMI-Specification“

– „The TX (Transaction Demarcation) Specification“

Die Kenntnis dieser X/Open-Spezifikationen und Kenntnisse von openUTM werden in diesem Handbuch vorausgesetzt; die Syntax der einzelnen Aufrufe wird nicht beschrieben.

In Kapitel 2, das auf diese Einleitung folgt, wird das von X/Open entwickelte Referenzmodell mit seinen Schnittstellen kurz vorgestellt und in das openUTM-Umfeld eingeordnet.

In den Kapiteln 3, 4 und 5 werden die unter openUTM zu beachtenden Besonderheiten der Schnittstellen CPI-C, XATMI und TX erläutert.

Die ausführlichen Verzeichnisse am Ende des Handbuchs - Fachwörter, Abkürzungen, Literatur und Stichwörter - sollen Ihnen den Umgang mit diesem Handbuch erleichtern.

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 203.

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.

1.4 Darstellungsmittel

B
B
BS2000-spezifische Teile der Beschreibung sind am linken Rand mit dem nebenstehenden Symbol gekennzeichnet.

X
X
Unix-System-spezifische Teile der Beschreibung sind am linken Rand mit dem nebenstehenden Symbol gekennzeichnet.

W
W
Windows-System-spezifische Teile der Beschreibung sind am linken Rand mit dem nebenstehenden Symbol gekennzeichnet.

B/X
B/X
Teile der Beschreibung, die nur für openUTM in BS2000- und Unix-Systemen von Bedeutung sind, sind am linken Rand mit dem nebenstehenden Symbol gekennzeichnet.

B/W
B/W
Teile der Beschreibung, die nur für openUTM in BS2000- und in Windows-Systemen von Bedeutung sind, sind am linken Rand mit dem nebenstehenden Symbol gekennzeichnet.

X/W
X/W
Teile der Beschreibung, die nur für openUTM in Unix- und Windows-Systemen von Bedeutung sind, sind am linken Rand mit dem nebenstehenden Symbol gekennzeichnet.


für Verweise auf umfassende und detaillierte Informationen zum jeweiligen Thema.


für Hinweistexte.


für Warnhinweise.

X/W
X/W
X/W
utmpfad
bezeichnet unter Unix- und Windows-Systemen das Verzeichnis, unter dem openUTM installiert wurde.

B
B
$userid bezeichnet unter BS2000-Systemen für die Kennung, unter der openUTM installiert wurde

2 Kommunikation über X/Open-Schnittstellen

Die fortschreitende Vernetzung von Computersystemen bildete den Ausgangspunkt dafür, dass auf der Grundlage des Client-/Server-Modells neue, heterogene Anwendungsarchitekturen entstanden, die die Integration unterschiedlicher Hardwareplattformen und Software-Komponenten in offenen Netzen voraussetzen.

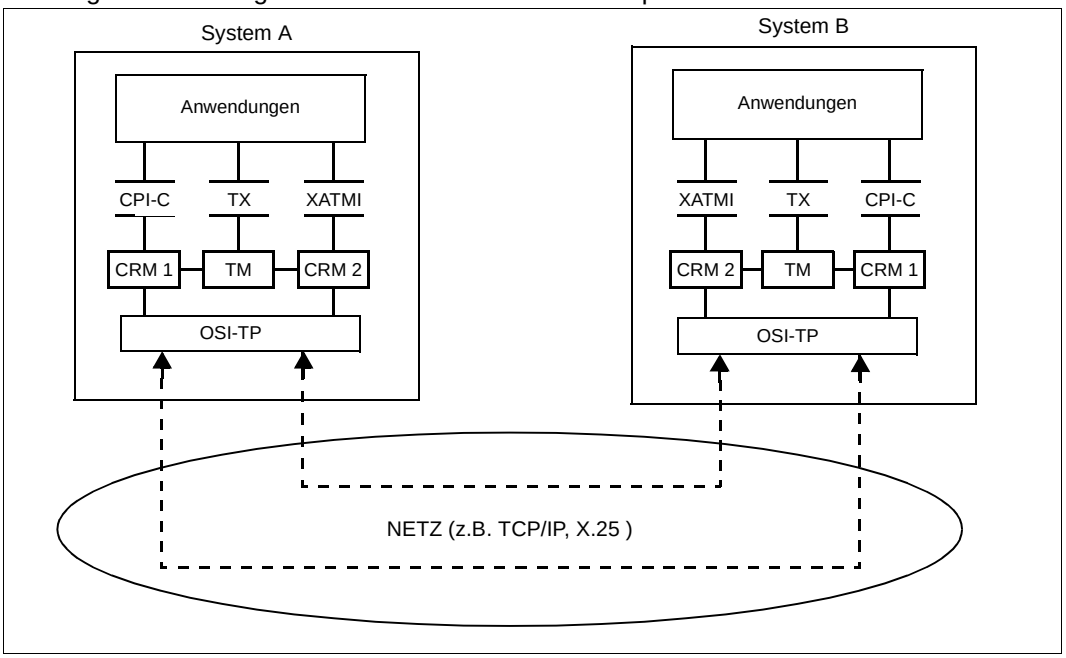
X/Open hat eine Entwicklungsumgebung definiert, die "Common Applications Environment (CAE)", die Standards und Schnittstellen für die Implementierung offener Systeme zur Verfügung stellt.

Für den Bereich der Transaktionsverarbeitung ist die Grundlage für offene Systeme das von X/Open im Rahmen der "Common Applications Environment" entwickelte Referenzmodell „Distributed Transaction Processing“ (DTP).
Den für openUTM relevanten Ausschnitt aus dem Referenzmodell finden Sie im nächsten Abschnitt . Eine vollständige Beschreibung des DTP-Referenzmodells finden Sie in dem X/Open-Handbuch „Distributed Transaction Processing: Reference Model Version 2“ .

openUTM unterstützt die im Rahmen des „Distributed Transaction Processing“ definierten Schnittstellen und bietet darüber hinaus optimale Möglichkeiten zur Integration bestehender Software-Komponenten wie zum Beispiel IBM- oder BS2000-Mainframe-Services. Eine Übersicht zur Connectivity von openUTM finden Sie im openUTM-Handbuch „Konzepte und Funktionen“.

2.1 Das Referenzmodell "Distributed Transaction Processing“

Das folgende Bild zeigt einen Ausschnitt aus dem X/Open-DTP-Referenzmodell:



Nach dem „Distributed Transaction Processing“-Modell steuert der Transaction Manager (TM) globale Transaktionen. Der Transaction Manager ist zuständig für den Start von Transaktionen und die Koordination von Commit- bzw. Rollback-Entscheidungen sowie für den Wiederanlauf nach Störungen oder Systemausfällen.

Die Kommunikation zwischen zwei Transaktionsanwendungen wird über Communication Resource Manager (CRM) abgewickelt. Die Communication Resource Manager kommunizieren über das von ISO definierte Kommunikationsprotokoll OSI-TP (Open Systems Interconnection - distributed Transaction Processing).
OSI-TP können unterschiedliche Netzprotokolle, wie z.B. TCP/IP oder X.25 zugrundeliegen.
Der OSI TP Communication Resource Manager ermöglicht einer UTM-Anwendung die Kommunikation mit anderen Anwendungen, die ebenfalls das Kommunikationsprotokoll OSI-TP verwenden. Insbesondere kann eine openUTM-Client-Anwendung mit Trägersystem OpenCPIC oder UPIC angeschlossen werden, die unter einem Unix- oder Windows-System läuft. Damit haben Sie beispielsweise die Möglichkeit, den Benutzern grafische Oberflächen anzubieten.
Darüberhinaus bietet Ihnen der OSI TP CRM in openUTM die Möglichkeit, sich über ein LU62-Gateway an Anwendungen zu koppeln, die das Kommunikationsprotokoll LU 6.2 verwenden.

Zusätzlich zu der von X/Open geforderten Kommunikation über OSI-TP können CPI-C-Programme und XATMI-Programme in openUTM auch über die Protokolle LU6.1 und UPIC kommunizieren.

Schnittstelle Anwendungsprogramm - Transaction Manager

Als Schnittstelle zwischen Anwendungsprogramm und Transaction Manager wurde von X/Open die Schnittstelle TX definiert, die in der Common Applications Environment (CAE) Spezifikation „Distributed Transaction Processing: The TX (Transaction Demarcation) Specification," beschrieben ist.

Schnittstellen Anwendungsprogramm - Communication Resource Manager

Als Schnittstellen zwischen Anwendungsprogramm und CRM wurden von X/Open die Schnittstellen CPI-C und XATMI definiert, die in den Common Applications Environment (CAE) Spezifikationen „Distributed Transaction Processing: The CPI-C Specification, Version 2" und „Distributed Transaction Processing: The XATMI Specification" beschrieben sind.

Beschreibung der X/Open-Schnittstellen in diesem Handbuch

In diesem Handbuch wird beschrieben, wie Sie die X/Open-Schnittstellen in openUTM einsetzen können, die Aufrufe selbst sind nicht beschrieben.

Zur Erstellung von Services unter openUTM, die die Aufrufe von CPI-C, XATMI oder TX verwenden, ist deshalb die Kenntnis der jeweiligen X/Open-Spezifikation unbedingt erforderlich.
Die genauen Titel der X/Open-Spezifikationen finden Sie im Literaturverzeichnis.

2.2 Einordnung der X/Open-Schnittstellen

Zusätzlich zu den offenen Schnittstellen unterstützt openUTM mit der Schnittstelle KDCS einen nationalen Standard, der sämtliche Aufrufe zur Programmierung einer Transaktionsanwendung im Rahmen einer umfassenden Schnittstelle zur Verfügung stellt.

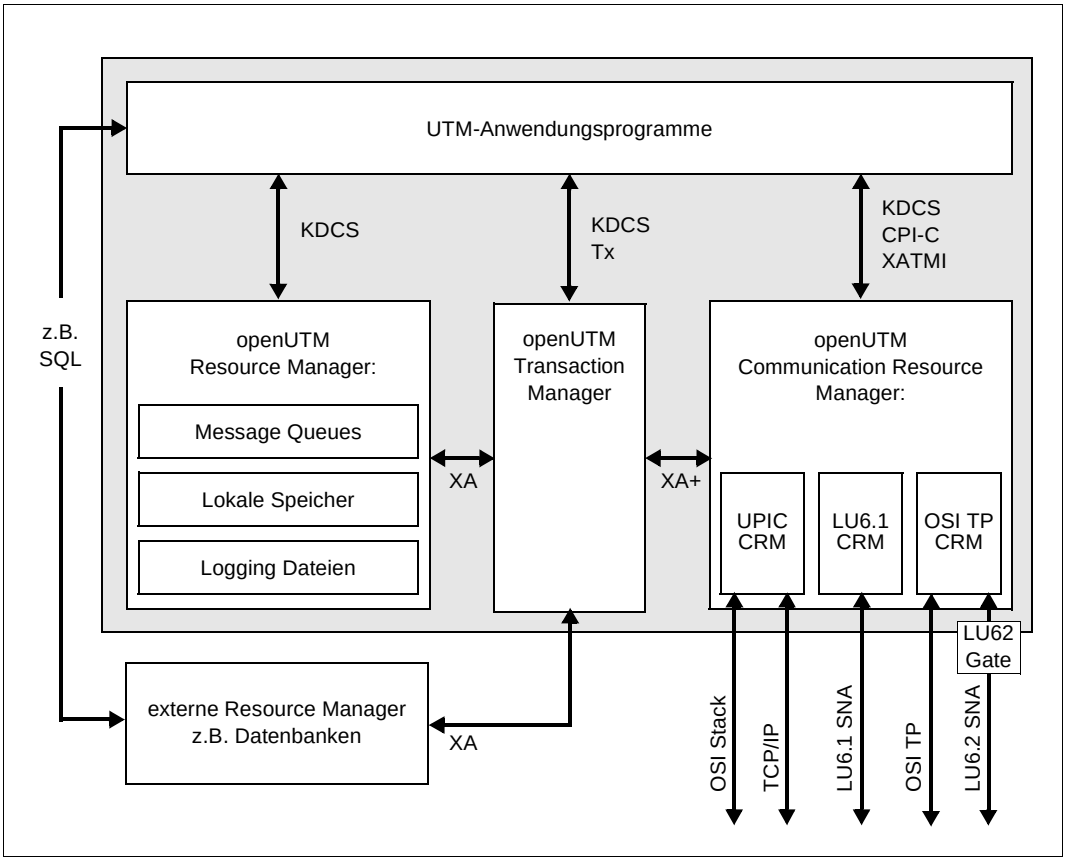
KDCS-Programmschnittstelle

Die Schnittstelle KDCS stellt Ihnen umfassende Möglichkeiten zur Transaktionssteuerung und zur Programm-Programm-Kommunikation zur Verfügung. Darüberhinaus bietet Ihnen KDCS ein transaktionsgesichertes Message Queueing mit der Möglichkeit, Nachrichten (Messages) zeitgesteuert zu übermitteln.
Zusätzlich können Sie mit KDCS-Aufrufen transaktionsgesichert auf openUTM-Ressourcen wie Speicherbereiche und Logging-Dateien zugreifen.

Die Schnittstelle KDCS ist im *open*UTM-Handbuch „Anwendungen programmieren mit KDCS“ beschrieben.

Das folgende Bild zeigt, welche Schnittstellen Ihnen openUTM zur Verfügung stellt:

.



Da openUTM auf allen gängigen Hardwareplattformen eingesetzt werden kann und über ausgezeichnete Connectivity-Eigenschaften verfügt, 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.

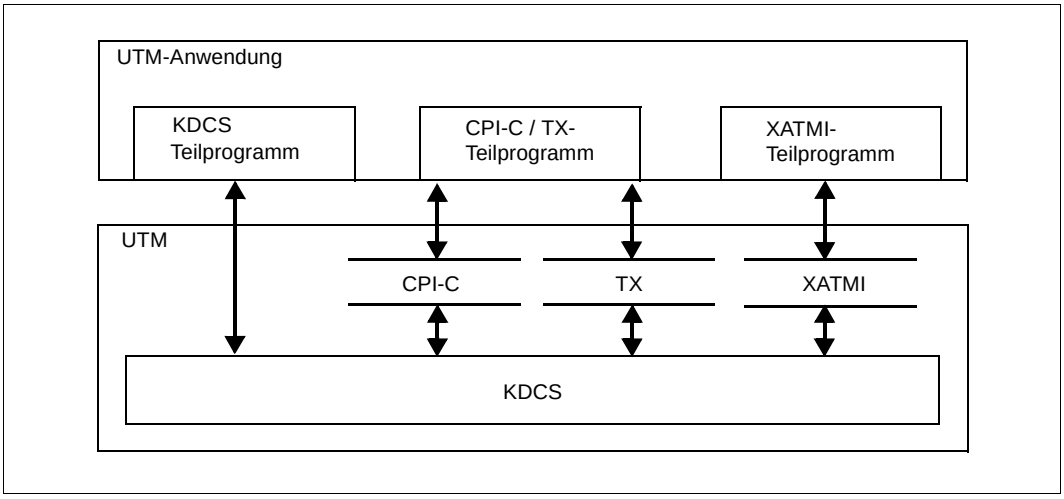
2.3 CPI-C, XATMI und TX unter openUTM

Die Schnittstellen CPI-C, XATMI und TX in openUTM genügen den CAE Specifications von X/Open zu CPI-C V2.0, XATMI und TX.

CPI-C und XATMI
: sind Programmschnittstellen für die Programm-Programm-Kommunikation über Rechnergrenzen hinweg. Sie stehen in COBOL und C zur Verfügung.
CPI-C und XATMI unter openUTM können zusätzlich zu OSI-TP auch über die Protokolle LU6.1 und UPIC kommunizieren.
CPI-C ist sowohl für synchrone als auch asynchrone Kommunikation im Conversational Mode geeignet, XATMI für synchrone und asynchrone Kommunikation im Request/Response Mode und im Conversational Mode.

TX
: ist eine Programmschnittstelle zur Festlegung von Transaktionsgrenzen über Rechnergrenzen hinweg. Sie steht in COBOL und C zur Verfügung.
TX-Aufrufe sind unter openUTM nur in CPI-C-Teilprogrammen sinnvoll, in XATMI-Teilprogrammen werden die Transaktionsgrenzen implizit über den XATMI-Aufruf tpreturn() festgelegt (vgl. Seite 142).

Das folgende Bild zeigt die Einbettung der X/Open-Programmschnittstellen in openUTM.

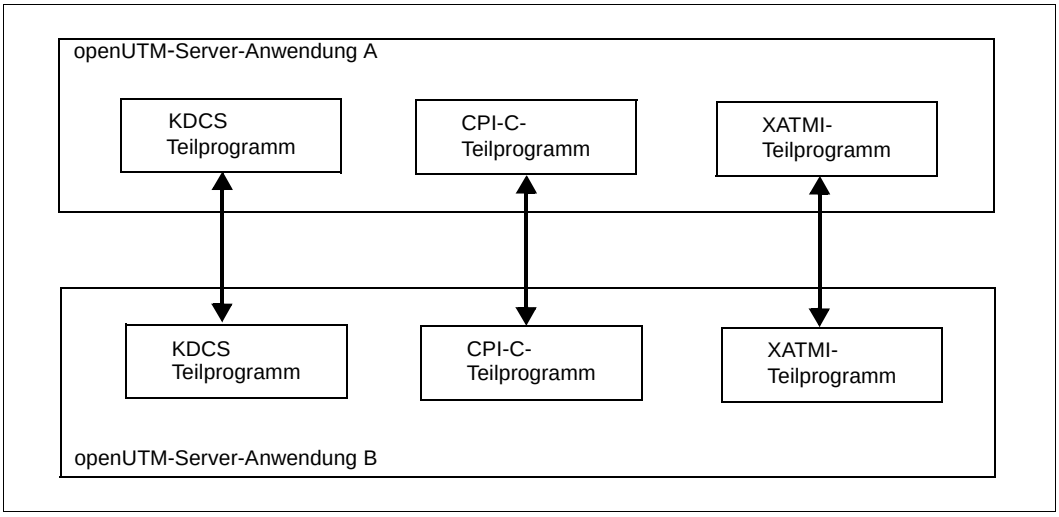


Eine Anwendung, die unter openUTM abläuft, kann Teilprogramme enthalten, die die Funktionen der Programmschnittstellen KDCS, CPI-C oder XATMI nutzen. Innerhalb eines Teilprogramms dürfen nur die Aufrufe einer dieser Schnittstellen verwendet werden. CPI-C-Teilprogramme können auch TX-Aufrufe zur Transaktionssteuerung enthalten. Für die Verarbeitung innerhalb eines Teilprogramms können selbstverständlich Datenbankaufrufe u.ä. verwendet werden.

Innerhalb eines Vorgangs dürfen Sie für die Kommunikation zwischen UTM-Teilprogrammen entweder nur KDCS-Aufrufe oder nur CPI-C-Aufrufe bzw. nur XATMI-Aufrufe verwenden, d.h. Sie dürfen nur eine dieser Schnittstellen in einem Vorgang verwenden.

2.4 Server/Server-Verbund mit CPI-C und XATMI unter openUTM

Bei der Kommunikation zwischen openUTM-Server-Anwendungen müssen die jeweils miteinander kommunizierenden Services (Teilprogramme) dieselbe Schnittstelle verwenden.

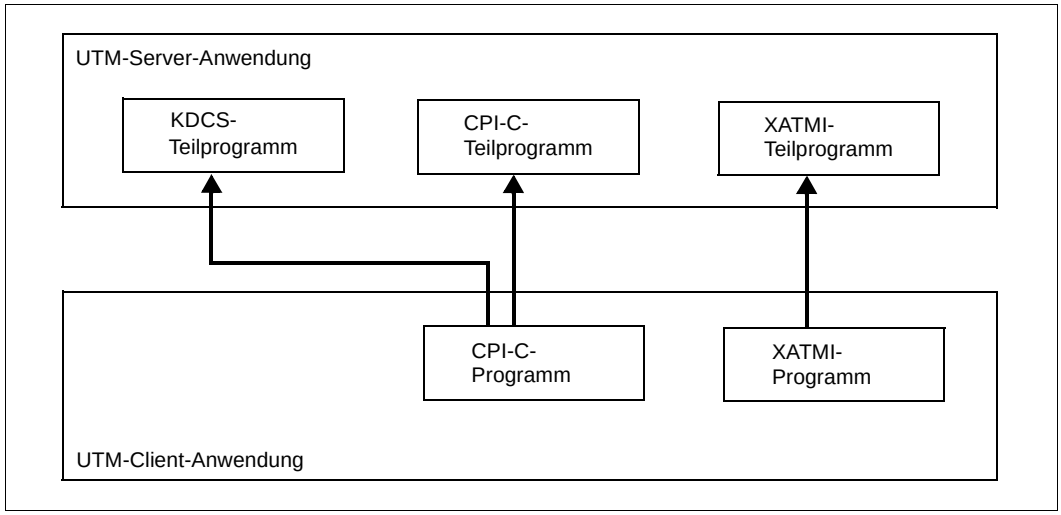


2.5 Client/Server-Verbund mit CPI-C und XATMI unter openUTM

Beim Client/Server-Verbund, d.h. bei der Kommunikation zwischen openUTM-Client-Anwendung und openUTM-Server-Anwendungen, sind folgende Kommunikationsmöglichkeiten gegeben:

– CPI-C-Client-Programme können sowohl mit KDCS- als auch mit CPI-C-Services (Teilprogrammen) der openUTM-Server-Anwendung kommunizieren.

– XATMI-Client-Programme können nur mit XATMI-Services (Teilprogrammen) der openUTM-Server-Anwendung kommunizieren.



Client/Server-Verbund

Für den Client/Server-Verbund mit CPI-C- und XATMI unter openUTM stehen folgende Produkte der openUTM-Produktfamilie zur Verfügung:

- Für Server-Anwendungen:
  openUTM (BS2000)
  openUTM Enterprise Edition (Unix-, Linux- und Windows-Systeme)

  Mit diesen Produkten können Sie Services auf Rechnern mit BS2000-, Unix-, Linux- und Windows-Systemen erstellen.

- Für Client-Anwendungen:
  openUTM-Client (BS2000)
  openUTM-Client (Unix- und Linux-Systeme)
  openUTM-Client (Windows-Systeme)

Mit diesen Produkten können Sie Client-Anwendungen auf BS2000-, Unix-, Linux und Windows-Systemen erstellen.

2.6 Einsatzgebiete von CPI-C, XATMI, TX und KDCS

In der folgenden Tabelle ist dargestellt, auf welchen Gebieten Sie die Schnittstellen schwerpunktmäßig einsetzen können:

Schnittstelle	**Einsatzgebiete**
KDCS	Ist eine vollständige Transaktionsmonitor-Schnittstelle, die alle wichtigen Transaktionsfunktionen enthält, wie z.B. Funktionen für die Programm-Programm-Kommunikation, Kommunikation mit Terminals und Druckern, Funktionen zum Erzeugen von Asynchronaufträgen (Hintergrund- und Ausgabeaufräge) etc. KDCS eignet sich für die Kommunikation von UTM-Anwendungen mit IMS und CICS über LU6.1.
CPI-C	Ist eine Schnittstelle zur Programm-Programm-Kommunikation. Sie ist geeignet zum Anschluss von Präsentations-Clients (PC, Workstation) an UTM-Anwendungen und zur Kommunikation zwischen Server-Anwendungen. Da CPI-C auch im Rahmen von IBM SAA definiert ist, eignet sich CPI-C besonders für Anwendungen, die in IBM-Umgebung eingesetzt werden sollen.
XATMI	Ist eine Schnittstelle zur Programm-Programm-Kommunikation. Da XATMI auch von anderen Transaktionsmonitoren unterstützt wird, eignet sich XATMI besonders für Anwendungen, die mit Anwendungen anderer Transaktionsmonitore, wie z.B. TUXEDO, kommunizieren sollen.
TX	Ist eine Schnittstelle zwischen Programm und Transaktionsmonitor, die der Steuerung von globalen Transaktionen dient. Sie eignet sich besonders für die Kopplung von OpenCPIC-Anwendungen mit UTM-Anwendungen.

2.7 Dateien und Bibliotheken für die X/Open-Schnittstellen

Für die Erstellung von openUTM-Server-Programmen, die die X/Open-Schnittstellen verwenden, werden Include-Dateien für C und COPY-Elemente für COBOL ausgeliefert.

Beim Binden dieser Programme muss die UTM-X/Open-Bibliothek eingebunden werden.

Für XATMI-Programme wird außerdem das Tool `xatmigen` ausgeliefert, das die Local Configuration File (LCF) aufbaut und Sie bei der KDCDEF-Generierung unterstützt.

X/W **Unix- und Windows-Systeme**

X/W Unter Unix- und Windows-Systemen sind die Dateien und Bibliotheken an folgenden X/W Stellen zu finden:

X/W
- Die Include-Dateien jeweils im Dateiverzeichnis

  X W *utmpfad*/*interface*/`include` (Unix-Systeme)
  *utmpfad**interface*\`include` (Windows-Systeme)

X/W
- Die COPY-Elemente im Dateiverzeichnis

  X W *utmpfad*/*interface*/`copy-cobol85` bzw. *utmpfad*/*interface*/`netcobol` (Unix-Systeme)
  utmpfad*interface*\`copy-cobol85` bzw. *utmpfad**interface*\`netcobol` (Windows-Systeme)

  X/W Dabei steht *interface* für die jeweilige X/Open-Schnittstelle (`cpic`, `tx` oder `xatmi`).

X/W
- Die X/Open-Bibliothek heißt

  X *utmpfad*/`sys/libxopen` (Unix-Systeme)

X/W
- Das Tool `xatmigen` finden Sie im Dateiverzeichnis

  X W *utmpfad*/`xatmi/ex` (Unix-Systeme)
  utmpfad\`xatmi\ex` (Windows-Systeme)

B **BS2000-Systeme**

B Unter BS2000-Systemen finden Sie die Include-Dateien und die COPY-Elemente als B Bibliothekselemente vom Typ S in der X/Open-Bibliothek

B $*userid*.SYSLIB.UTM.063.XOPEN

B Das Tool `xatmigen` finden Sie als Bibliothekselement vom Typ L in der Bibliothek

B $*userid*.SYSLNK.UTM.063.UTIL

3 X/Open-Schnittstelle CPI-C

In diesem Kapitel werden die Besonderheiten von CPI-C unter openUTM beschrieben. Diese Beschreibung ist als openUTM-spezifische Ergänzung zur X/Open Dokumentation „Distributed Transaction Processing: The CPI-C Specification, Version 2“ zu verstehen. Zum Verständnis und für die Erstellung von CPI-C-Programmen unter openUTM ist die Kenntnis der X/Open-Specification deshalb unbedingt erforderlich.

3.1 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) normierte Programmschnittstelle für die Programm-Programm-Kommunikation über Rechnergrenzen hinweg.

Da CPI-C nur die Programm-Programm-Kommunikation unterstützt, bietet CPI-C keine Funktionen zur Kommunikation mit Terminals. CPI-C-Teilprogramme in openUTM können aus diesem Grunde nicht direkt von einem Terminal (durch Eingabe eines Transaktionscodes) gestartet werden.

Daher sind CPI-C-Anwendungen unter openUTM immer Server-Anwendungen, d.h. CPI-C-Teilprogramme einer UTM-Anwendung können nur durch Service-Anforderungen von Partneranwendungen gestartet werden.

Service-Anforderungen können z.B. von openUTM-Client-Anwendungen oder von anderen Server-Anwendungen kommen.
Zur Bearbeitung der Anforderungen können die CPI-C-Teilprogramme ihrerseits auch Services von anderen UTM-Anwendungen und von Fremdanwendungen anfordern. In diesem Fall spricht man von einem Server/Server-Verbund, in dem der anfordernde Partner jeweils die Rolle des Client übernimmt.

3.1.1 Begriffsdefinitionen

Conversation

Bei CPI-C nennt man die Kommunikation zwischen zwei CPI-C-Anwendungsprogrammen *Conversation.* Eine Conversation entspricht in der openUTM/KDCS-Terminologie einem Vorgang.

Eine Conversation wird auf einer bereits bestehenden logischen Verbindung zwischen den Partneranwendungen, die die CPI-C-Programme enthalten, aufgebaut. Logische Verbindungen werden im Zusammenhang mit OSI-TP *Associations* und im Zusammenhang mit LU6.1 *Sessions* genannt.

Die beiden Partner einer Conversation nehmen verschiedene Rollen innerhalb der Conversation ein. Ein Partner initiiert die Conversation. Dieser Partner wird im Folgenden als *Initiator* der Conversation bezeichnet. Der andere Partner akzeptiert den Conversation-Aufbau und wird im Folgenden als *Akzeptor* bezeichnet. Bezüglich der Client/Server-Rollenverteilung ist der Initiator der Conversation der Client, der Akzeptor der Server.

Initiator einer Conversation / Outgoing-Conversation

Der *Initiator* einer Conversation baut eine Conversation durch die CPI-C-Aufrufe Initialize_Conversation und Allocate aktiv auf; er initiiert die Conversation. Dem Begriff *Initiator* einer Conversation entspricht in der openUTM / KDCS-Terminologie der Begriff Auftraggeber.
Eine Conversation, bei der das lokale CPI-C-Programm der Initiator ist, heißt *Outgoing-Conversation*. In der X/Open-Specification werden die Begriffe Outgoing-Conversation und Outbound-Conversation synonym verwendet.

Akzeptor einer Conversation / Incoming-Conversation

Der *Akzeptor* einer Conversation nimmt die Conversation an. Dazu verwendet er den CPI-C-Aufruf Accept_Conversation. Dem Begriff *Akzeptor* einer Conversation entspricht in der openUTM / KDCS-Terminologie der Begriff Auftragnehmer.
Eine Conversation, bei der das lokale CPI-C-Programm der Akzeptor ist, heißt *Incoming-Conversation*. In der X/Open-Specification werden die Begriffe Incoming-Conversation und Inbound-Conversation synonym verwendet.

Zu einem Zeitpunkt kann ein CPI-C-Programm unter openUTM genau eine *Incoming-* und mehrere *Outgoing-Conversations* unterhalten, d.h., das CPI-C-Programm unter openUTM kann innerhalb eines Programmlaufs auf genau einer Conversation Akzeptor sein und kann auf mehreren Conversations Initiator sein. Durch die Anforderung der *Incoming-Conversation* wird das CPI-C-Programm unter openUTM gestartet.

Dialog- und Asynchron-Conversation

In openUTM wird zwischen Dialog- und Asynchron-Conversations unterschieden, abhängig davon, ob der Akzeptor ein asynchroner Service ist oder nicht.

Auf *Dialog-Conversations* können beide Seiten der Conversation senden, d.h. sowohl der Initiator (Client) als auch der Akzeptor (Server). Der Akzeptor einer Dialog-Conversation muss in der UTM-Anwendung mit einem Dialog-Transaktionscode definiert werden (Angabe TYPE=D in der KDCDEF-Steueranweisung TAC oder LTAC).

Auf *Asynchron-Conversations* kann nur der Initiator der Conversation senden. Der Akzeptor einer Asynchron-Conversation muss in der UTM-Anwendung mit einem Asynchron-Transaktionscode definiert werden (Angabe TYPE=A in der KDCDEF-Steueranweisung TAC oder LTAC).

Conversation-ID

Jeder Conversation wird von CPI-C lokal eine eindeutige *Conversation-ID* zugeordnet, d.h. Initiator und Akzeptor haben jeweils eine eigene Conversation-ID, die nicht mit der des Partners übereinzustimmen braucht. Die Conversation-ID dient dazu, jeden CPI-C-Aufruf innerhalb eines Programms eindeutig einer Conversation zuzuordnen.

Conversation-Charakteristika

Jede Conversation besitzt eine Reihe von *Conversation-Charakteristika*. Die Werte der Charakteristika beeinflussen die Abläufe der Conversation und die bei der Conversation zur Verfügung stehenden Funktionen von CPI-C. Einige der Charakteristika bestimmen nur die Abläufe auf einer Seite der Conversation, andere gelten für beide Seiten.
Bei der Initialisierung der Conversation werden die Charakteristika mit Standardwerten vorbelegt. Die Initialisierungswerte der Charakteristika sind abhängig von der Rolle des CPI-C-Programms beim Start der Conversation, d.h. sie hängen davon ab, ob das Programm die Conversation initiiert oder akzeptiert. Durch die Abfolge der CPI-C-Aufrufe innerhalb der Conversation werden die Werte der Charakteristika verändert.

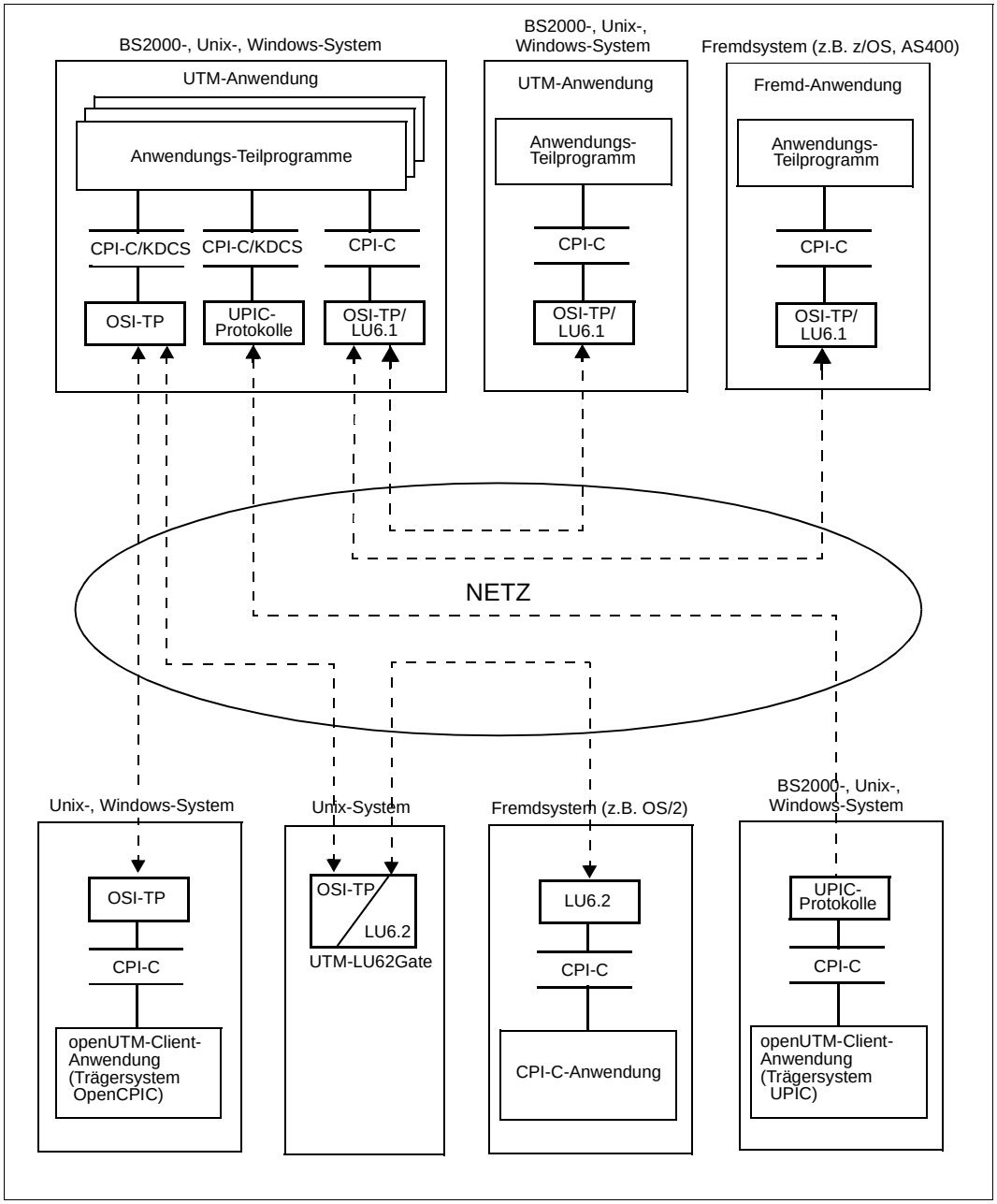
Die aktuellen Werte der Charakteristika können mit Extract-Aufrufen abgefragt und durch Set-Aufrufe explizit geändert werden.

Ab Seite 44 sind die Conversation-Charakteristika aufgeführt, die für CPI-C-Programme unter openUTM von Bedeutung sind.

3.1.2 Kommunikationspartner einer CPI-C-Anwendung unter openUTM

Das folgende Bild zeigt die möglichen Kommunikationspartner einer CPI-C-Anwendung unter openUTM. Das sind:

- CPI-C-Programme in anderen UTM-Anwendungen in BS2000-, Unix- oder Windows-Systemen. Dabei kann das lokale CPI-C-Programm sowohl Initiator als auch Akzeptor der Conversation sein.
- CPI-C-Programme in Anwendungen auf Fremdsystemen, die OSI-TP oder LU6.1 unterstützen. Auch hier kann das lokale CPI-C-Programm sowohl Initiator als auch Akzeptor der Conversation sein.
- openUTM-Client-Anwendung mit Trägersystem OpenCPIC.
  Bezüglich openUTM-Client-Anwendungen mit Trägersystem OpenCPIC kann die UTM-Anwendung sowohl Initiator als auch Akzeptor der Conversation sein.
- openUTM-Client-Anwendung mit Trägersystem UPIC.
  Mit openUTM-Client-Anwendungen mit Trägersystem UPIC kann eine UTM-Anwendung nur Incoming-Conversations unterhalten, d.h., Conversations, die von der openUTM-Client-Anwendung initiiert werden. Die UTM-Anwendung ist bei Conversations über das UPIC-Protokoll immer Akzeptor.
- LU6.2-Anwendungen in Fremdsystemen.
  UTM-Anwendungen können z.B. Conversations zu CPI-C-Anwendungen in OS/2-Systemen unterhalten, die das LU6.2-Protokoll verwenden. In diesem Fall muss die Verbindung zwischen der UTM-Anwendung und dem Fremdsystem über ein Unix- oder Windows-System aufgebaut werden, an dem OpenCPIC und das Gateway openUTM-LU62Gate installiert sind (siehe folgendes Bild).
  Für die Verbindung muss im Unix- oder Windows-System keine OpenCPIC-Anwendung erstellt werden.



Kommunikationspartner einer CPI-C-Anwendung unter openUTM

3.1.3 Server/Server-Verbund mit CPI-C

Damit zwei CPI-C-Partner eine Conversation aufbauen können, muss zwischen den Anwendungsinstanzen eine logische Verbindung aufgebaut werden. Auf dieser Verbindung bauen die CPI-C-Partneranwendungen die Conversation auf.

CPI-C-Programme unter openUTM können auf OSI-TP-Associations und LU6.1-Sessions Conversations aufbauen. Dabei kann ein CPI-C-Programm gleichzeitig Conversations über OSI-TP-Protokoll und LU6.1-Protokoll unterhalten.
Ein CPI-C-Programm unter openUTM kann auch über das UPIC-Protokoll eine Conversation von openUTM-Clients mit Trägersystem UPIC akzeptieren (=Incoming Conversation) und gleichzeitig Outgoing-Conversations zu anderen Server-Anwendungen über OSI-TP- und LU6.1-Protokoll unterhalten.

Sobald eine Anforderung für eine Conversation vorliegt, baut openUTM die notwendige Association bzw. Session entweder auf, oder den Partnern wird für die Conversation eine bereits bestehende Association/Session zur Verfügung gestellt, die von einer anderen Conversation freigegeben wurde.

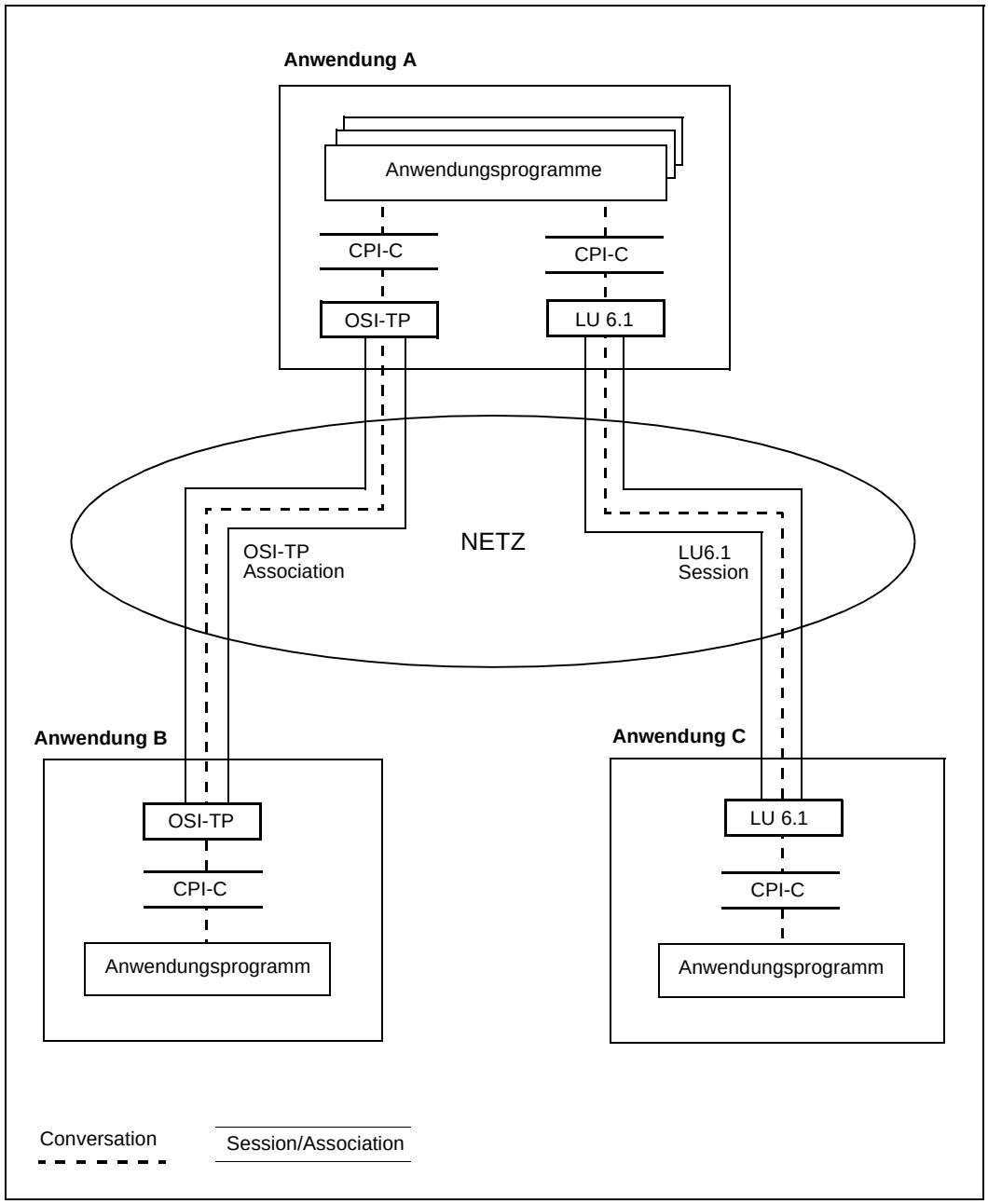
KDCDEF-Anweisungen für den Aufbau einer Association bzw. Session

Damit openUTM eine Association bzw. Session zum Partner aufbauen kann, muss eine entsprechende KDCDEF-Generierung vorliegen.
Was Sie bei der KDCDEF-Generierung beachten müssen, ist detailliert im openUTM-Handbuch „Anwendungen generieren“ beschrieben. Im Folgenden sind nur die Anweisungen genannt, die Sie für den Aufbau von Conversations benötigen.

Zum Aufbau einer Association müssen Sie die Partneranwendung mit Hilfe einer OSI-LPAP-Anweisung und die Verbindung mit einer OSI-CON-Anweisung definieren.

Zum Aufbau einer LU6.1-Session müssen Sie den Partner mit einer LPAP-Anweisung und die Verbindung mit CON-, SESCHA- und LSES-Anweisungen definieren.

Für einen openUTM-Client mit Trägersystem UPIC muss in der KDCDEF-Generierung eine LTERM- und eine PTERM-Anweisung angegeben werden oder mit einer TPOOL-Anweisung ein Terminalpool generiert werden.



Anwendungsverbund mit CPI-C

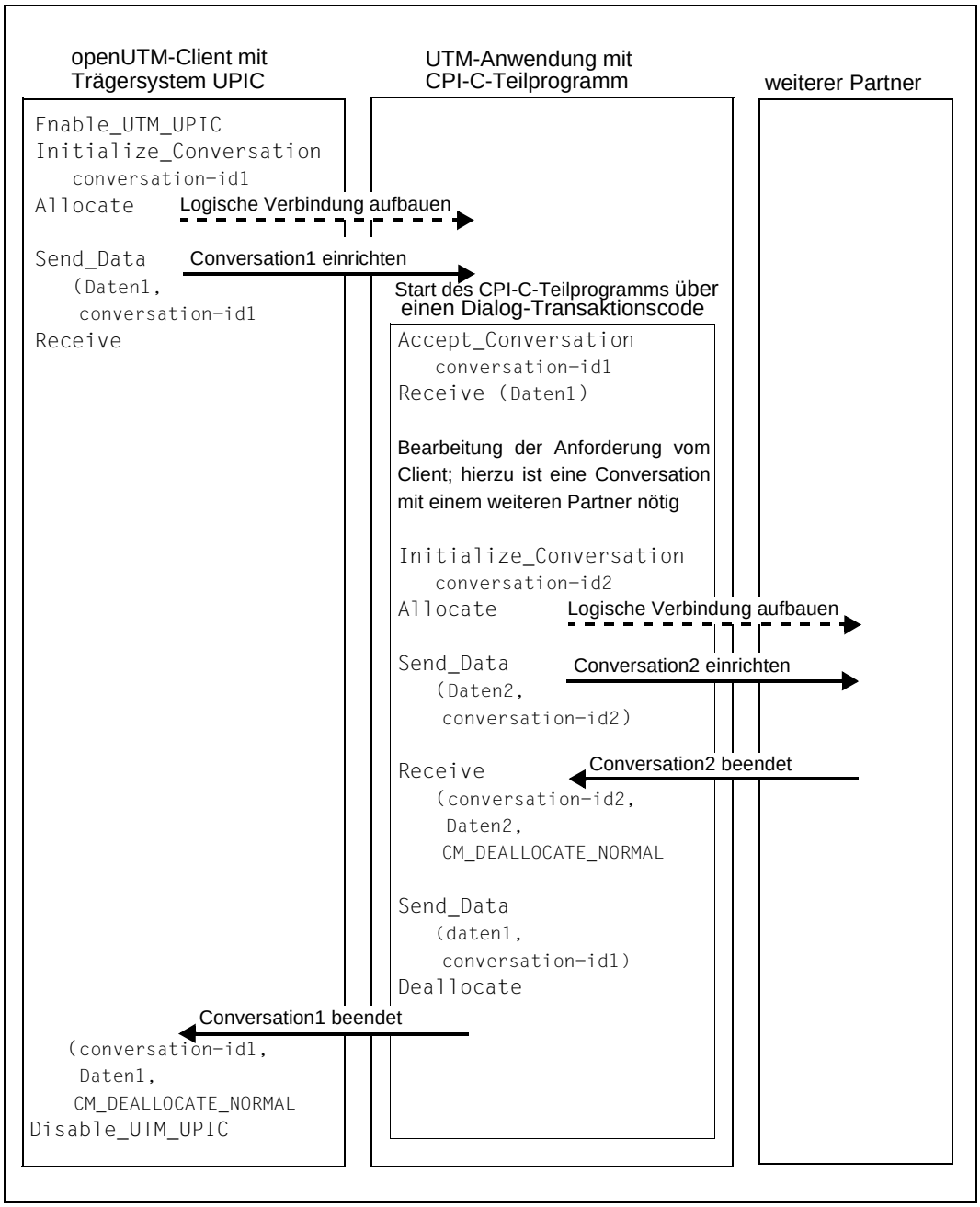
3.1.4 Anwendungsbeispiel - Ablaufdiagramm

Im Folgenden wird ein typisches Beispiel für einen Server/Server-Verbund mit CPI-C-Programmen unter openUTM beschrieben.

Das CPI-C-Teilprogramm einer UTM-Anwendung wird durch die Anforderung einer Conversation gestartet (es akzeptiert die Conversation). Das anfordernde Programm, das die Conversation initiiert, kann z.B. ein openUTM-Client sein, aber auch ein CPI-C-Programm einer anderen Anwendung, das selbst durch eine Conversation-Anforderung gestartet wurde und in der Conversation zum nächsten Service die Rolle des Initiators übernimmt.

Im folgenden Ablaufdiagramm wird das CPI-C-Teilprogramm unter openUTM durch einen openUTM-Client mit Trägersystem UPIC gestartet und akzeptiert die Conversation (Conversation1).
Conversation1 ist aus der Sicht des CPI-C-Teilprogramms unter openUTM eine Incoming-Conversation.

Das CPI-C-Programm unter openUTM bearbeitet die Anforderung des Initiators und führt zum Beispiel den Zugriff auf eine Datenbank durch. Zur Bearbeitung der Service-Anforderung kann es auch nötig sein, dass dieses CPI-C-Programm seinerseits eine Conversation mit einem weiteren CPI-C-Programm in einer anderen Anwendung initiiert (UTM-Anwendung oder Anwendung an einem Fremdsystem), z.B. weil die Datenbank, auf die zugegriffen werden muss, sich an diesem System befindet (Conversation2).
Conversation2 ist aus der Sicht des CPI-C-Teilprogramms unter openUTM eine Outgoing-Conversation.


openUTM-Client mit
Trägersystem UPIC
UTM-Anwendung mit
CPI-C-Teilprogramm
weiterer Partner
Enable_UTM_UPIC
Initialize_Conversation
conversation-id1
Allocate
Logische Verbindung aufbauen
Send_Data
(Daten1,
conversation-id1
Conversation1 einrichten
Receive
Start des CPI-C-Teilprogramms über
einen Dialog-Transaktionscode
Accept_Conversation
conversation-id1
Receive (Daten1)
Bearbeitung der Anforderung vom
Client; hierzu ist eine Conversation
mit einem weiteren Partner nötig
Initialize_Conversation
conversation-id2
Allocate
Logische Verbindung aufbauen
Send_Data
(Daten2,
conversation-id2)
Conversation2 einrichten
Receive
(conversation-id2,
Daten2,
CM_DEALLOCATE_NORMAL
Conversation2 beendet
Send_Data
(daten1,
conversation-id1)
Deallocate
Conversation1 beendet
(conversation-id1,
Daten1,
CM_DEALLOCATE_NORMAL
Disable_UTM_UPIC

3.2 CPI-C-Charakteristika und Funktionen in openUTM

Dieser Abschnitt soll Ihnen einen Überblick darüber geben, welche Funktionen Sie mit CPI-C unter openUTM nutzen können. Es werden die für CPI-C unter openUTM relevanten Conversation-Charakteristika, Zustände einer Conversation und Grenzwerte aufgeführt.

3.2.1 Conversation-Charakteristik Conversation-Type

CPI-C in openUTM unterstützt bei der Kommunikation über LU6.1 und OSI-TP den Conversation-Typ *Mapped Conversation*. Eine *Mapped Conversation* erlaubt den Austausch von Datensätzen in einem beliebigen Datenformat, das von den beiden Partnern der Conversation vereinbart wird.
Bei jedem Send_Data-Aufruf wird genau ein Datensatz übergeben.

Der Conversation-Charakteristik „conversation_type“ darf deshalb nur der Wert CM_MAPPED_CONVERSATION zugeordnet werden.

Der Wert CM_BASIC_CONVERSATION wird mit CM_PRODUCT_SPECIFIC_ERROR abgewiesen.

3.2.2 Conversation-Charakteristika für die Adressierung

Für den Aufbau einer Conversation wird Adressinformation benötigt. In CPI-C wird die Adressinformation in der *Side Information* verwaltet. Die in der *Side Information* verwaltete Adressinformation ist über einen symbolischen Namen vom Programm aus abrufbar. Diesen Namen übergibt das Programm beim Initialize_Conversation-Aufruf im Parameter *sym_dest_name* (symbolic destination name). Die Adressinformation wird gelesen und die Conversation-Charakteristika *TP_name*, *partner_LU_name*, *AE_qualifier*, *AP_title* und *application_context_name* werden mit den Daten aus der *Side Information* versorgt.

Bei openUTM ist die *Side Information* in der KDCFILE enthalten. Die Adressinformation wird bei der KDCDEF-Generierung festgelegt.

Einbringen der Adressinformation in die KDCFILE

Für jede Partneranwendung, mit der über das LU6.1-Protokoll kommuniziert werden soll, muss mindestens eine LPAP-Anweisung abgesetzt werden. Die LPAP-Anweisung beschreibt die ferne Partneranwendung. Den LPAP-Anweisungen müssen Sie bei der KDCDEF-Generierung CON-, SESCHA- und LSES-Anweisungen zuordnen. Die Information, die in diesen Anweisungen definiert wird, benötigt openUTM, um die Session zur Partneranwendung aufzubauen und die Session-Eigenschaften festzulegen.

Für jede Partneranwendung, mit der über das OSI-TP-Protokoll kommuniziert werden soll, muss mindestens eine OSI-LPAP-Anweisung abgesetzt werden. Die OSI-LPAP-Anweisung beschreibt die ferne Partneranwendung. Der OSI-LPAP-Anweisung müssen Sie bei der KDCDEF-Generierung eine OSI-CON-Anweisung zuordnen. Die Information, die in
OSI-LPAP- und OSI-CON-Anweisungen definiert wird, benötigt openUTM, um die Associations zur Partneranwendung aufzubauen und die Eigenschaften der Associations festzulegen

Ein openUTM-Client-Partner mit Trägersystem UPIC muss bei der KDCDEF-Generierung mit einer LTERM- und einer PTERM-Anweisung bzw. mit einer TPOOL-Anweisung definiert werden.

Für jede Partneranwendung, zu der ein CPI-C-Programm unter openUTM eine Outgoing-Conversation aufbauen will, muss mindestens eine LTAC-Anweisung abgesetzt werden. Die LTAC-Anweisung definiert in der lokalen Anwendung einen Transaktionscode für einen Service der fernen Server-Anwendung. Der in dieser LTAC-Anweisung angegebene Name (*ltac-name*) ist der symbolische Name, über den das CPI-C-Programm auf die Adressinformation zugreifen kann.

LTAC-Anweisung zusammen mit LPAP- oder OSI-LPAP-Anweisung enthalten die Information, die das CPI-C-Programm benötigt, um eine Conversation zur Partneranwendung aufbauen zu können.

Mehrere Partneranwendungen

Mehrere Partneranwendungen können mit der MASTER-OSI-LPAP- oder MASTER-LU61-LPAP-Anweisung zu einem Bündel zusammengefasst werden.

Wenn das Programm ein MASTER-OSI-LPAP oder MASTER-LU61-LPAP verwendet, wählt UTM ein OSI-LPAP bzw. LPAP aus dem Bündel aus. Welches OSI-LPAP bzw. LPAP eines Bündels UTM auswählt, finden Sie im openUTM-Handbuch „Anwendungen generieren“ im Kapitel „Verteilte Anwendungen generieren“.

Übergabe der Adressinformation

Um die zum Aufbau der Conversation benötigte Adressinformation zu spezifizieren, übergibt das Programm an openUTM entweder nur den *ltac-namen* aus der LTAC-Anweisung oder den *ltac*-und den *lpap*-Namen aus der LPAP- bzw. OSI-LPAP-Anweisung oder MASTER-LU61-LPAP bzw. MASTER-OSI-LPAP-Anweisung. Ob nur der *ltac-name* oder auch der *lpap-name* übergeben werden muss, ist abhängig davon, wie der Partner bei der KDCDEF-Generierung definiert wurde (siehe unten).

Dem Programmierer stehen folgende Möglichkeiten zur Verfügung, Adressinformationen zu spezifizieren:

- Bei der KDCDEF-Generierung wird in der LTAC-Anweisung der Operand LPAP=*lpap-name* angegeben *(lpap-name* aus LPAP- bzw. OSI-LPAP-Anweisung oder aus MASTER-LU61-LPAP- bzw. MASTER-OSI-LPAP-Anweisung) und somit dem LTAC direkt eine LPAP- bzw. OSI-LPAP-Anweisung oder MASTER-LU61-LPAP- bzw. MASTER-OSI-LPAP-Anweisung zugeordnet, die die Partneranwendung beschreibt. In diesem Fall reicht ein Initialize_Conversation-Aufruf aus, wobei der darin übergebene *sym_dest_name* gleich dem ltac-Namen aus der LTAC-Anweisung sein muss. Die Conversation Charakteristika *TP_name*, *partner_LU_name*, *AE_qualifier*, *AP_title* und *application_context_name* werden mit den Werten versorgt, die in der LTAC- und LPAP- bzw. OSI-LPAP-Anweisung definiert sind.

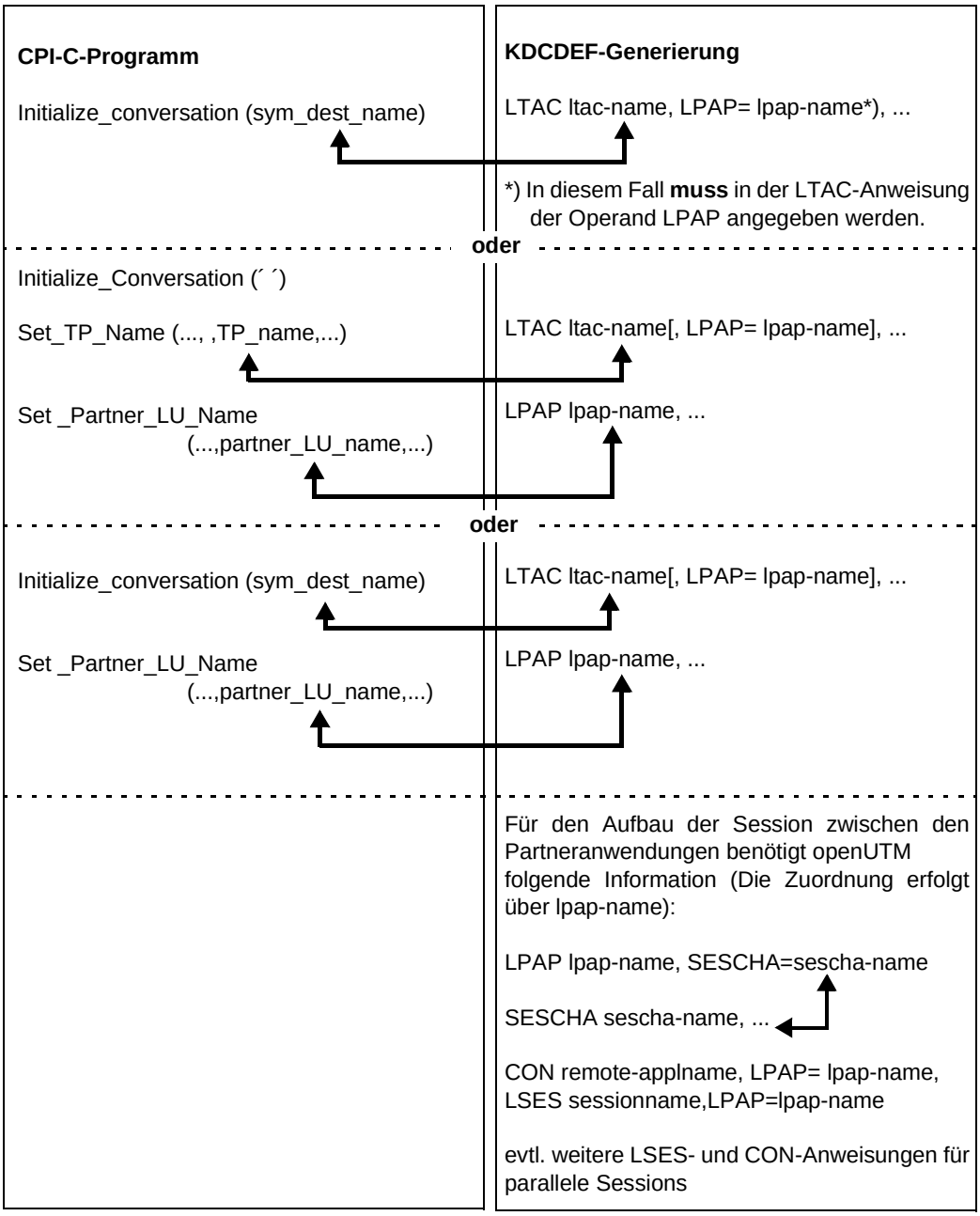
  Die Übergabe des *ltac*- und *lpap*-Namens kann optional auch durch die Aufrufe Set_TP_Name und Set_Partner_LU_Name erfolgen. Die bei diesen Set_-Aufrufen übergebenen Werte für die Conversation Charakteristika überschreiben die Werte, die durch den Initialize_Conversation-Aufruf gesetzt werden.

- Bei der KDCDEF-Generierung wird in der LTAC-Anweisung der Operand LPAP nicht angegeben, d.h. dem LTAC wird durch die Generierung keine ferne Anwendung zugeordnet. Die Zuordnung von LTAC zu LPAP bzw. OSI-LPAP oder MASTER-LU61-LPAP bzw. MASTER-OSI-LPAP muss dann im CPI-C-Programm erfolgen. In diesem Fall gibt es zwei Möglichkeiten, die Conversation Charakteristika mit der Adressinformation zu versorgen:

- Sie übergeben beim Aufruf Initialize-Conversation für *sym_dest_name* Leerzeichen (Blanks) und versorgen vor dem Allocate-Aufruf die Conversation Charakteristika durch den Aufruf Set_TP_Name (mit *TP_name*=*ltac-name*) und den Aufruf Set_Partner_LU_Name (mit *partner_LU_name*= *lpap-name*);
- oder Sie geben beim Aufruf Initialize_Conversation im Parameter *sym_dest_name* den ltac-Namen aus der LTAC-Anweisung an und versorgen *partner_LU_name* durch den Aufruf Set_Partner_LU_Name (mit *partner_LU_name*=*lpap-name*), um die noch fehlende Information über die ferne Partneranwendung zu spezifizieren. Der Aufruf Set_Partner_LU_Name muss vor dem Allocate-Aufruf erfolgen.

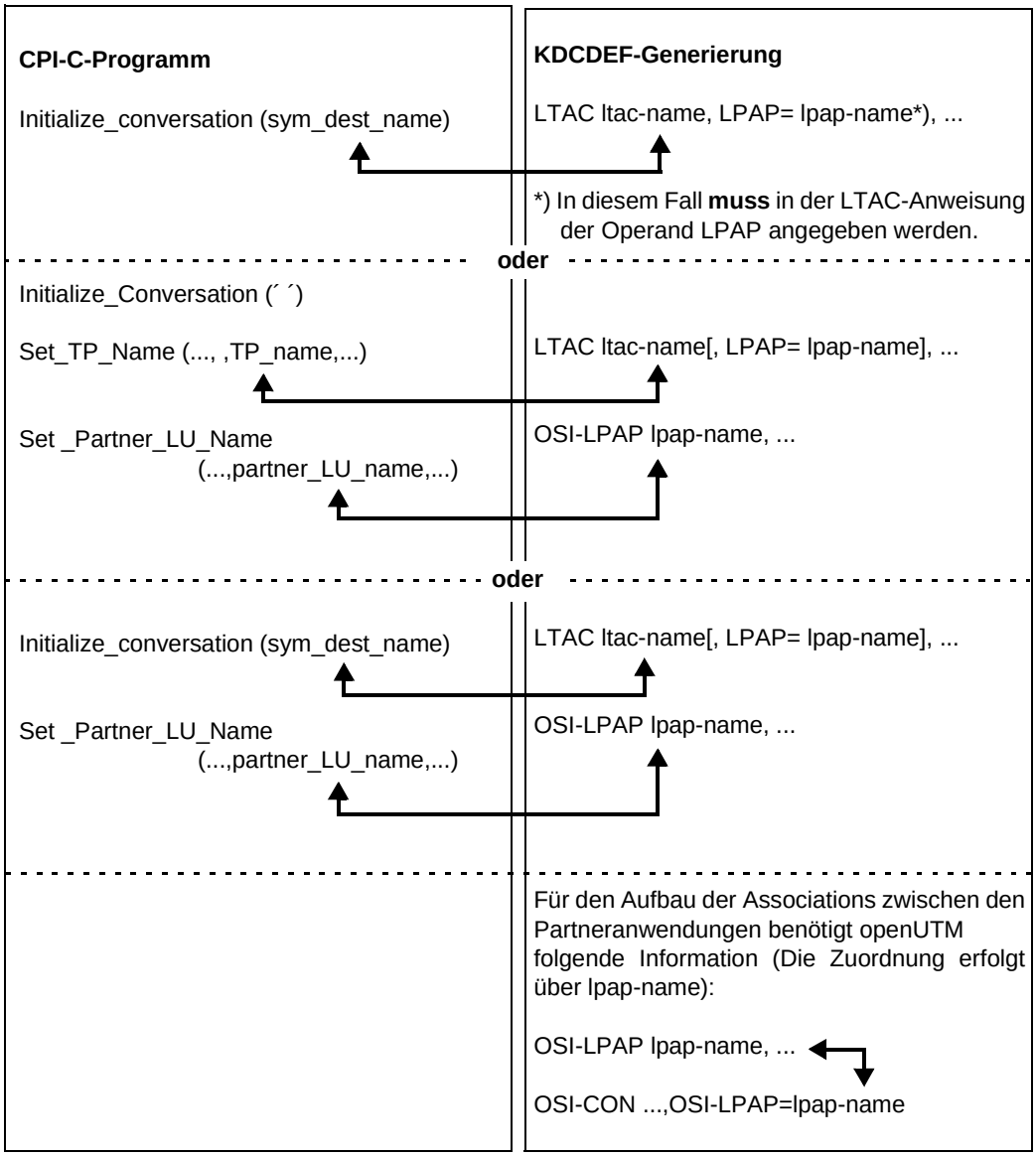
In den folgenden Bildern ist für Outgoing-Conversations zu OSI-TP- und LU6.1-Partnern dargestellt, wie die Angaben im CPI-C-Programm und die Definitionen bei der KDCDEF-Generierung aufeinander abgestimmt werden müssen. Die in den Bildern durch Pfeile verbundenen Namen müssen jeweils übereinstimmen.

Abstimmung CPI-C-Programm-Aufrufe mit KDCDEF-Generierung für Conversations über LU6.1 (mit einzelnen LPAPs)*



* Zu LPAP-Bündeln siehe Abschnitt „Mehrere Partneranwendungen“ auf Seite 46.

*Abstimmung CPI-C-Programm-Aufrufe mit KDCDEF-Generierung für Conversations über OSI-TP (mit einzelnen OSI-LPAPs *)*



* Zu OSI-LPAP-Bündeln siehe Abschnitt „Mehrere Partneranwendungen“ auf Seite 46.

Belegung der Conversation Charakteristika beim Einrichten einer Conversation

In der folgenden Tabelle ist beschrieben, wie die Conversation Charakteristika für die Adressierung der Partner beim Einrichten der Conversation belegt werden.

	Charakteristik	**Wert der Charakteristik bei Outgoing-Conversations (Initiator)**	**Wert der Charakteristik bei Incoming-Conversations (Akzeptor)**
LU6.1	partner_LU_name	lpap-name der LPAP-Anweisung, die die ferne Anwendung beschreibt. Aufruf Extract_Partner_LU_Name liefert nur dann einen Wert zurück, wenn *partner_LU_name* explizit mittels Aufruf Set_Partner_LU_Name gesetzt wurde.	lpap-name der LPAP-Anweisung, die die ferne Anwendung beschreibt
	TP_Name	der ltac-name der LTAC-Anweisung	Name des Transaktionscodes, unter dem das lokale Programm aufgerufen wurde (TAC tac-name)
OSI-TP	partner_LU_name	osi-lpap-name der OSI-LPAP-Anweisung, die die ferne Partneranwendung beschreibt. Aufruf Extract_Partner_LU_Name liefert nur dann einen Wert zurück, wenn *partner_LU_name* explizit mittels Aufruf Set_Partner_LU_Name gesetzt wurde.	osi-lpap-name der OSI-LPAP-Anweisung, die die ferne Partneranwendung beschreibt
	TP_Name	ltac-name der LTAC-Anweisung	TAC-Name des Transaktionscodes, unter dem das lokale Programm aufgerufen wurde
	AE_Qualifier	AEQ aus der OSI-LPAP-Anweisung, die die Partneranwendung beschreibt.	AEQ aus der OSI-LPAP-Anweisung, die die Partneranwendung beschreibt.
	AP-Title	APT aus der OSI-LPAP-Anweisung, die die Partneranwendung beschreibt.	APT aus der OSI-LPAP-Anweisung, die die Partneranwendung beschreibt.
	application_context	Wert des Operanden APPLICATION-CONTEXT der OSI-LPAP-Anweisung	Wert des Operanden APPLICATION-CONTEXT der OSI-LPAP-Anweisung

	Charakteristik	Wert der Charakteristik bei Outgoing-Conversations (Initiator)	Wert der Charakteristik bei Incoming-Conversations (Akzeptor)
UPIC-Anwendung	partner_LU_name	Zu einem openUTM-Client mit Trägersystem UPIC kann openUTM keine Outgoing Conversations aufbauen	Name des LTERM-Partners, über den die Verbindung aufgebaut wurde
	TP_Name		Name des Transaktionscodes, unter dem das lokale Programm aufgerufen wurde (TAC tac-name)

Maximale Namenslänge

Anders als in X/Open-CPI-C beträgt bei CPI-C unter openUTM die Maximallänge der bei *partner_LU_name* bzw. bei *TP_name* angegebenen Namen 8 Byte. Wird bei einem CPI-C-Aufruf im Parameter *partner_LU_name_length* oder *TP_name_length* ein Wert größer 8 byte angegeben, dann weist openUTM den Aufruf mit dem Returncode CM_PRODUCT_SPECIFIC_ERROR ab.

3.2.3 Sende-/Empfangsmodus und Senderecht

CPI-C unter openUTM unterstützt halbduplex Conversations, d.h. zu einem Zeitpunkt hat nur einer der beiden Partner das Senderecht. Das Senderecht kann vom sendenden Partner auf den anderen übertragen werden, indem
entweder die Charakteristik *send_type* auf CM_SEND_AND_PREP_TO_RECEIVE gesetzt wird und anschließend der Receive-Aufruf ausgeführt wird
oder der Receive-Aufruf im Zustand *Send* ausgeführt wird. Der nicht blockierende Aufruf Prepare_to_Receive wird in openUTM nicht unterstützt.

Das Senderecht kann nur von dem Partner, der das Senderecht hat, an den anderen übertragen werden. Der Aufruf Request_to_Send, mit dem ein Partner im Zustand *Receive* das Senderecht anfordern kann, wird von openUTM nicht unterstützt. Fordert ein ferner Nicht-UTM-Partner mit dem Aufruf Request_to_Send das Senderecht an, dann wird die Anforderung bereits von openUTM verworfen und nicht an das lokale CPI-C-Programm weitergereicht.

Der Sende- und Empfangsmodus wird in der Charakteristik *send_receive_mode* festgelegt. Die Festlegung muss bei der Initialisierung erfolgen. Die Standardeinstellung von *send_receive_mode* ist CM_HALF_DUPLEX und kann von CPI-C-Programmen unter openUTM nicht geändert werden.

3.2.4 Mehrere Conversations in einem CPI-C-Programmlauf

Innerhalb eines Programmlaufs kann ein CPI-C-Programm mehrere Conversations gleichzeitig unterhalten. Mit Hilfe der verschiedenen Conversation-IDs werden die CPI-C-Aufrufe im Programmablauf den einzelnen Conversations zugeordnet. Bei der maximal möglichen Anzahl gleichzeitig offener Conversations eines CPI-C-Programms wird unterschieden, ob es sich um Incoming-Conversations oder Outgoing-Conversations handelt.

Incoming-Conversation

Unter openUTM wird bei jeder neuen Service-Anforderung das entsprechende Teilprogramm neu gestartet, d.h. jedesmal, wenn ein dem Teilprogramm zugeordneter Transaktionscode (TAC tac-name) von einem fernen Partner empfangen wird, startet openUTM das Teilprogramm als Akzeptor dieser Incoming-Conversation neu.
Damit kann es nicht vorkommen, dass ein CPI-C-Programm eine zweite Incoming-Conversation zu bearbeiten hat.

Ein CPI-C-Programm kann daher maximal eine Incoming-Conversation mit dem Aufruf Accept_Conversation annehmen. Ein zweiter Accept_Conversation-Aufruf innerhalb eines Programmlaufs wird mit CM_PROGRAM_STATE_CHECK beantwortet, was die Bedeutung "No incoming conversation exists" hat.

Outgoing-Conversation

Die Maximalzahl der Outgoing-Conversations ist durch folgende Faktoren begrenzt:

1. durch die maximale Anzahl von Outgoing-Conversations, die in einem Programmlauf eines CPI-C-Programms gleichzeitig bestehen können.

   Es dürfen gleichzeitig sechs Outgoing-Conversations existieren neben der Incoming-Conversation, durch die das Programm gestartet wurde.
   Wird versucht eine siebente (offene) Outgoing-Conversation aufzubauen, dann wird der Aufruf Initialize_Conversation mit CM_PRODUCT_SPECIFIC_ERROR abgewiesen.
   Unterhält das CPI-C-Programm sechs Outgoing-Conversations und wird eine der Outgoing-Conversations beendet, dann kann das Programm eine neue Outgoing-Conversation aufbauen.

2. durch die maximale Anzahl der Outgoing-Conversations, die die lokale Anwendung gleichzeitig zu einer bestimmten Partneranwendung unterhalten kann.

   Diese Anzahl ist begrenzt durch die maximal erlaubte Anzahl paralleler OSI-TP-Associations bzw. LU6.1-Sessions, die gleichzeitig zu einer Partneranwendung existieren dürfen.
   Die Maximalzahl paralleler Associations zu einem OSI-TP-Partner wird im Operanden ASSOCIATIONS der OSI-LPAP-Anweisung festgelegt. Die Maximalzahl paralleler Sessions zu einem LU6.1-Partner ist begrenzt durch die Anzahl der LSES-Anweisungen, die der LPAP-Anweisung des Partners zugeordnet sind.

3. durch die maximale Anzahl der Outgoing-Conversations, die die lokale Anwendung gleichzeitig unterhalten kann.

   Die Anzahl ist begrenzt durch die Gesamtzahl der Associations und Sessions, die die lokale UTM-Anwendung maximal gleichzeitig unterhalten darf.
   Diese Anzahl ergibt sich aus der Summe aller generierten OSI-TP-Associations und LU6.1-Sessions (Anzahl der LSES-Anweisungen) multipliziert mit dem Wert von MAXJR (angegeben in %).
   (Anzahl Associations + Anzahl Sessions) ***** MAXJR
   Der Wert von MAXJR wird bei der KDCDEF-Generierung in der UTMD-Anweisung definiert.

Wird einer dieser Werte durch eine angeforderte Outgoing-Conversation überschritten, dann wird der zugehörige Allocate-Aufruf mit dem Returncode CM_ALLOCATE_FAILURE_NO_RETRY abgewiesen (die Liste der möglichen Returncodes finden Sie bei den Hinweisen zum Allocate-Aufruf auf Seite 69).

3.2.5 Conversation-Charakteristik sync_level

In der Conversation-Charakteristik *sync_level* wird der Grad der Synchronisation bei der Verarbeitung zwischen den beiden CPI-C-Programmen einer Conversation festgelegt. Der Wert von *sync_level* muss vor dem Einrichten der Conversation eingestellt werden.

CPI-C-Programme unter openUTM können für *sync_level* den Wert CM_NONE oder CM_SYNC_POINT_NO_CONFIRM einstellen.

CM_NONE bedeutet, dass für die Conversation keine Synchronisation angefordert wird - weder Synchronisation durch Confirmation (Anforderung von Quittungen) noch Sync Pointing (anwendungsübergreifende Transaktionssicherung).

CM_SYNC_POINT_NO_CONFIRM bedeutet, dass die Conversation in eine globale Transaktion eingeschlossen wird.

Versucht ein CPI-C-Programm unter openUTM mit dem Aufruf Set_Sync_level einen anderen Wert als CM_NONE oder CM_SYNC_POINT_NO_CONFIRM einzustellen, dann weist openUTM den Aufruf wie folgt zurück:

– sync_level=CM_CONFIRM
  mit dem Returncode CM_PRODUCT_SPECIFIC_ERROR

– sync_level= CM_SYNC_POINT
  mit dem Returncode CM_PARM_VALUE_NOT_SUPPORTED.

Conversation mit einer Nicht-UTM-Anwendung

Unterhält ein CPI-C-Programm unter openUTM eine Conversation zu einem CPI-C-Programm einer Nicht-UTM-Anwendung, dann kann das Partnerprogramm die Synchronisation der Conversation mit *sync_level* CM_CONFIRM oder CM_SYNC_POINT anfordern.

Das unter openUTM ablaufende Programm muss mit Extract_Sync_Level den eingestellten Wert abfragen und entsprechend reagieren.

– Das Nicht-UTM-Programm hat CM_CONFIRM eingestellt und fordert eine Quittung (Empfangsbestätigung) an, z.B. durch Aufruf Set_Prepare_To_Receive _Type mit *prepare_to_receive_type* = CM_PREP_TO_RECEIVE_CONFIRM und anschließendem Aufruf Prepare_To_Receive.

  In CPI-C unter openUTM sind der Confirmed-Aufruf und der Aufruf Send_Error verfügbar, mit denen das CPI-C-Programm unter openUTM eine positive bzw. negative Quittung an das Partnerprogramm senden kann.

– Das Nicht-UTM-Programm hat für *sync_leve*l CM_SYNC_POINT.

  Das CPI-C-Programm unter openUTM darf die Conversation nicht von sich aus

beenden sondern muss warten, bis der Client das Transaktionsende anfordert und die Conversation beendet (CM_TAKE_COMMIT_DEALLOCATE bzw. CM_TAKE_BACKOUT in *status_received*).

3.2.6 Maximale Nachrichtenlänge

Die Länge der Nachrichten ist begrenzt durch die maximale Puffergröße CM_MAXIMUM_BUFFER_SIZE. Diese ist bei openUTM auf 32767 Bytes eingestellt.

Die maximale Puffergröße ist bei den Parametern *buffer_length*, *send_length* und *requested_length* relevant. Wird für einen solchen Parameter ein Wert außerhalb des Bereiches 0 ≤ ..._length ≤ CM_MAXIMUM_BUFFER_SIZE angegeben, dann wird der Aufruf mit CM_PROGRAM_PARAMETER_CHECK abgewiesen.

3.2.7 Konvertierung von Characteristika und Benutzerdaten

Konvertierung von Charakteristika

Einige der Conversation-Charakteristika bestimmen die Conversation nicht nur lokal. Diese Charakteristika müssen an den Partner der Conversation übertragen werden. Sie müssen beim Partner in einer Codierung ankommen, die dieser „versteht“. Je nachdem, welche Zeichensätze an den Partnersystemen verwendet werden, müssen die Charakteristika konvertiert werden. Die Konvertierung erfolgt automatisch, der Programmierer muss sich darum nicht kümmern.

Bei der Kommunikation über LU6.1 werden die Charakteristik-Daten, sofern nötig, vom lokal verwendeten Systemcode in EBCDIC umgewandelt. Empfängt CPI-C Charakteristik-Daten vom Partner, dann geht CPI-C davon aus, dass die Daten in EBCDIC vorliegen und konvertiert sie in den am lokalen System verwendeten Zeichensatz.

Bei der Kommunikation über OSI-TP werden die Charakteristik-Daten in die Transfersyntax umgesetzt, die bei der Generierung der UTM-Anwendung für Associations zwischen den Partneranwendungen festgelegt wurde (OSI-LPAP-Anweisung Operand APPLICATION-CONTEXT). Die Charakteristik-Daten werden automatisch in diese Transfersyntax umgesetzt.

Konvertierung von Benutzerdaten

openUTM bietet zwar die Möglichkeit, die automatische Codekonvertierung für die ausgetauschten Benutzerdaten per Konfiguration einzuschalten, indem bei der KDCDEF-Generierung der Operand MAP=SYSTEM in der SESCHA- bzw. OSI-CON-Anweisung angegeben wird.

In der CPI-C-Schnittstelle ist die automatische Konvertierung der mit dem Aufruf Send_Data gesendeten bzw. der mit dem Receive-Aufruf empfangenen Benutzerdaten jedoch nicht enthalten.

CPI-C stellt dem Programmierer für die Konvertierung der Benutzerdaten den Aufruf Convert_Outgoing und den Aufruf Convert_Incoming zur Verfügung:

Convert_Outgoing (CMCNVO)

Mit dem Aufruf Convert_Outgoing können Sie Benutzerdaten vor dem Senden umsetzen.

B CPI-C in BS2000-Systemen geht davon aus, dass die im Parameter *buffer* übergebenen Daten im Zeichensatz EBCDIC.DF.04 codiert sind und setzt sie um in den Zeichensatz „EBCDIC Multilingual 697/1 Code Page 500/1“

X/W CPI-C in Unix- oder Windows-Systemen geht davon aus, dass die im Parameter *buffer* übergebenen Daten im ISO-8-Bit-Code ISO 8859-1 (ASCII) codiert sind und setzt sie um in einen modifizierten „EBCDIC Multilingual 697/1 Code Page 500/1“-Zeichensatz.

X/W Der Zeichensatz EBCDIC Multilingual 697/1 Code Page 500/1 (im weiteren kurz EBCDIC Code Page 500/1 genannt) ist in „X/Open CAE Specification CPI-C“ im Anhang A „Character Sets“ beschrieben.
X/W Dieser Zeichensatz wurde für openUTM auf Unix- und Windows-Systemen so modifiziert, dass das New Line-Zeichen (\n) nach der Umsetzung mit den Konvertierungsaufrufen in Unix- oder Windows-Systemen und BS2000-Systemen gleich wirkt. Dazu wurden die Plätze der Zeichen 0x15 und 0x25 in der Codetabelle vertauscht.

Convert_Incoming (CMCNVI)

Mit dem Aufruf Convert_Incoming können Sie die empfangenen Daten umsetzen. CPI-C geht davon aus, dass die Daten im Zeichensatz EBCDIC Code Page 500/1 codiert sind.

B Unter BS2000-Systemen konvertiert der Aufruf die Daten in den Zeichensatz EBCDIC.DF.04.

X Unter Unix- oder Windows-Systemen konvertiert der Aufruf die Daten in den Zeichensatz in ISO 8859-1 (ASCII).

Homogene Kopplung

Bei einer homogenen Kopplung von zwei BS2000-Rechnern oder zwei Unix- oder Windows-Rechnern ist eine Konvertierung zwar zulässig, aber nicht nötig. In diesem Fall müssen jedoch beide Partner dasselbe tun, d.h. beide konvertieren oder beide nicht konvertieren. Wenn z.B. bei der Kopplung zweier Unix- oder Windows-Partner eine Seite konvertiert und die andere Seite nicht, kann ein nicht verständlicher Code entstehen.

Heterogene Kopplung

Bei einer heterogenen Kopplung, z.B. BS2000- mit Unix- oder Windows-Systemen, sollte der Absender die Daten mit dem Aufruf Convert_Outgoing umcodieren und der Empfänger mit dem Aufruf Convert_Incoming ebenfalls.

Die EBCDIC-Varianten EBCDIC.DF.04 und EBCDIC Code Page 500/1 weichen in 17 Zeichen voneinander ab (z.B. in [ ] { } \ | !, siehe die Code-Konvertierungstabellen auf Seite 58). Kommen diese Zeichen in den Benutzerdaten nicht vor, so kann z.B. bei einer Kommunikation zwischen einem BS2000- und einem Unix-System die Konvertierung entfallen.

Achtung:

Werden für die Datenkonvertierung die Aufrufe Convert_Incoming und Convert_Outgoing verwendet, dann darf keine automatische Konvertierung durch openUTM generiert werden, d.h. es darf bei der Generierung der Verbindungen zur Partneranwendung *nicht* MAP=SYSTEM in der OSI-CON-Anweisung (bei OSI-TP-Partnern) bzw. in der SESCHA-Anweisung (bei LU6.1-Partnern) angegeben werden.

Bei der Verwendung der Konvertierungsaufrufe müssen Sie auf die korrekte Abfolge der Aufrufe achten, da mehrmals mit Convert_Incoming bzw. mehrmals mit Convert_Outgoing konvertierte Daten im Zielsystem nicht mehr interpretiert werden können. Ob bei den Partnern einer Conversation die Aufrufe Convert_Incoming und Convert_Outgoing richtig aufeinander abgestimmt sind, können Sie dem CPI-C-Trace entnehmen.

Code-Konvertierungstabellen

In diesem Abschnitt finden Sie die von CPI-C bei den Aufrufen Convert_Incoming und Convert_Outgoing verwendeten Code-Konvertierungstabellen. Es sind die Konvertierungstabellen dargestellt, die in openUTM auf BS2000-Systemenund openUTM auf Unix- und Windows-Systemen verwendet werden.

B **Konvertierungstabellen openUTM auf BS2000-Systemen**

B In den folgenden Tabelle werden nur die 17 Zeichen aufgelistet, in denen sich die Codes
B EBCDIC.DF.04 und EBCDIC Code Page 500/1 unterscheiden. Beim Convert_Incoming
B bzw. Convert_Outgoing werden nur diese Zeichen umgesetzt.

B *Code-Konvertierung bei Convert_Outgoing (BS2000-Systeme)*

B Code-Umsetztabelle EBCDIC.DF.04 → EBCDIC Code Page 500/1

B B	**EBCDIC.DF.04**	**EBCDIC Code Page 500/1**
B	0x4A	0x79
B	0x4F	0xBB
B	0x5A	0x4F
B	0x5F	0xFF
B	0x6A	0x5F
B	0x79	0xBD
B	0xA1	0xBC
B	0xBB	0x4A
B	0xBC	0xE0
B	0xBD	0x5A
B	0xC0	0xDD
B	0xD0	0x6A
B	0xDD	0xFB
B	0xE0	0xFD
B	0xFB	0xC0
B	0xFD	0xD0
B	0xFF	0xA1

B Code-Konvertierung bei Convert_Outgoing

Code-Konvertierung bei Convert_Incoming (BS2000-Systeme)

Code-Umsetztabelle EBCDIC Code Page 500/1 (EBCDIC) → EBCDIC.DF.04

EBCDIC Code Page 500/1	**EBCDIC.DF.04**
0x4A	0xBB
0x4F	0x5A
0x5A	0xBD
0x5F	0x6A
0x6A	0xD0
0x79	0x4A
0xA1	0xFF
0xBB	0x4F
0xBC	0xA1
0xBD	0x79
0xC0	0xFB
0xD0	0xFD
0xDD	0xC0
0xE0	0xBC
0xFB	0xDD
0xFD	0xE0
0xFF	0x5F

Code-Konvertierung bei Convert_Incoming

Konvertierungstabellen openUTM (Unix- und Windows-Systeme)

Code-Konvertierung bei Convert_Outgoing (Unix- und Windows-Systeme)

Code-Umsetztabelle ISO8859-1 → modifizierter EBCDIC Code Page 500/1
(die grau hinterlegten Felder zeigen die Modifikation gegenüber EBCDIC Code Page 500/1)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	37	2D	2E	2F	16	05	15	0B	0C	0D	0E	0F
1	10	11	12	13	3C	3D	32	26	18	19	3F	27	1C	1D	1E	1F
2	40	4F	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
3	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
4	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
5	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	4A	E0	5A	5F	6D
6	79	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
7	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	C0	6A	D0	A1	07
8	20	21	22	23	24	25	06	17	28	29	2A	2B	2C	09	0A	1B
9	30	31	1A	33	34	35	36	08	38	39	3A	3B	04	14	3E	FF
A	41	AA	B0	B1	9F	B2	BB	B5	BD	B4	9A	8A	BA	CA	AF	BC
B	90	8F	EA	FA	BE	A0	B6	B3	9D	DA	9B	8B	B7	B8	B9	AB
C	64	65	62	66	63	67	9E	68	74	71	72	73	78	75	76	77
D	AC	69	ED	EE	EB	EF	EC	BF	80	FD	FE	FB	FC	AD	AE	59
E	44	45	42	46	43	47	9C	48	54	51	52	53	58	55	56	57
F	8C	49	CD	CE	CB	CF	CC	E1	70	DD	DE	DB	DC	8D	8E	DF

Code-Konvertierung bei Convert_Incoming (Unix- und Windows-Systeme)

Code-Umsetztabelle modifizierter EBCDIC Code Page 500/1 (EBCDIC) → ISO 8859-1 (die grau hinterlegten Felder zeigen die Modifikation gegenüber EBCDIC Code Page 500/1)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	9C	09	86	7F	97	8D	8E	0B	0C	0D	0E	0F
1	10	11	12	13	9D	0A	08	87	18	19	92	8F	1C	1D	1E	1F
2	80	81	82	83	84	85	17	1B	88	89	8A	8B	8C	05	06	07
3	90	91	16	93	94	95	96	04	98	99	9A	9B	14	15	9E	1A
4	20	A0	E2	E4	E0	E1	E3	E5	E7	F1	5B	2E	3C	28	2B	21
5	26	E9	EA	EB	E8	ED	EE	EF	EC	DF	5D	24	2A	29	3B	5E
6	2D	2F	C2	C4	C0	C1	C3	C5	C7	D1	7C	2C	25	5F	3E	3F
7	F8	C9	CA	CB	C8	CD	CE	CF	CC	60	3A	23	40	27	3D	22
8	D8	61	62	63	64	65	66	67	68	69	AB	BB	F0	FD	FE	B1
9	B0	6A	6B	6C	6D	6E	6F	70	71	72	AA	BA	E6	B8	C6	A4
A	B5	7E	73	74	75	76	77	78	79	7A	A1	BF	D0	DD	DE	AE
B	A2	A3	A5	B7	A9	A7	B6	BC	BD	BE	AC	A6	AF	A8	B4	D7
C	7B	41	42	43	44	45	46	47	48	49	AD	F4	F6	F2	F3	F5
D	7D	4A	4B	4C	4D	4E	4F	50	51	52	B9	FB	FC	F9	FA	FF
E	5C	F7	53	54	55	56	57	58	59	5A	B2	D4	D6	D2	D3	D5
F	30	31	32	33	34	35	36	37	38	39	B3	DB	DC	D9	DA	9F

3.2.8 Zustände einer Conversation unter openUTM

Auf einer Conversation müssen die Funktionsaufrufe der beiden Partner-Programme aufeinander abgestimmt sein. Ein CPI-C-Programm führt eine Funktion aus, wissend, dass das Partnerprogramm als Antwort eine weitere Funktion für dieselbe Conversation ausführt. Um die Programmierregeln auf beiden Seiten der Conversation besser verstehen zu können, wird für die Conversation ein *Zustand* eingeführt. Die Menge der Folgeaktionen innerhalb einer Conversation ist abhängig vom jeweiligen Zustand der Conversation. Durch Funktionsaufrufe geht die Conversation von dem aktuellen Zustand in einen anderen Zustand über. Diese Übergänge nennt man *Transition*.

Eine Conversation kann unter openUTM folgende Zustände einnehmen:

Reset Der *conversation_ID* ist keine Conversation zugeordnet.

Initialize
: Der Aufruf Initialize_Conversation wurde erfolgreich beendet und der Conversation wurde eine *conversation_ID* zugeordnet.

Send Das Programm hat das Recht, Daten über die Conversation zu senden (Senderecht).

Receive
: Das Programm kann über die Conversation Daten vom Partner empfangen.

Send-Pending
: Das Programm hat beim letzten Receive-Aufruf vom Partner Daten und gleichzeitig das Senderecht erhalten.

Confirm-Send
: Das lokale Programm hat vom Partner eine Quittungsanforderung und das Senderecht empfangen. Das lokale Programm muss mit einem Confirmed-Aufruf antworten und geht dann in den Zustand *Send* über.

  Dieser Zustand kann bei CPI-C-Programmen unter openUTM nur in Conversations zu Nicht-UTM-Anwendungen auftreten.

Confirm-Deallocate
: Das lokale Programm hat vom Partner zusammen mit der Deallocate-Mitteilung eine Aufforderung zur Empfangsbestätigung empfangen. Das lokale Programm muss mit
  einem Confirmed-Aufruf antworten und geht dann in den Zustand *Reset* über.

  Dieser Zustand kann bei CPI-C-Programmen unter openUTM nur in Conversations zu Nicht-UTM-Anwendungen auftreten.

Defer-Deallocate

Das lokale Programm hat vom Partner zusammen mit der Deallocate-Mitteilung die Anforderung des Transaktionsendes empfangen. Das lokale Programm muss mit dem entsprechenden TX-Aufruf antworten und geht dann in den Zustand *Reset* über.

Zu Beginn befindet sich eine Conversation im Zustand *Reset*. Sie nimmt danach verschiedene Folgezustände ein, abhängig von den eigenen Funktionsaufrufen und von den Informationen, die vom Partner empfangen werden.
Der aktuelle Zustand einer Conversation kann mit dem Aufruf Extract_Conversation_State abgefragt werden. Die Zustandsübergänge sind in der X/Open-Specification im Anhang beschrieben.

Wird eine Funktion aufgerufen, die im aktuell gültigen Zustand nicht erlaubt ist, erhält das aufrufende Programm den Returncode CM_PROGRAM_STATE_CHECK. Die Funktion wird nicht ausgeführt.

3.3 CPI-C in openUTM

In diesem Abschnitt finden Sie:

- unter „Unterstützte CPI-C-Aufrufe“ eine Liste der in openUTM verfügbaren CPI-C-Aufrufe.
  Von den in der X/Open-Spezifikation zu CPI-C definierten Aufrufen sind in openUTM nur die wichtigsten verfügbar. Von diesen sind einige nur für die Kommunikation über das
  OSI-TP-Protokoll verwendbar.
- unter „Einschränkungen bei Conversations über das LU6.1- und UPIC-Protokoll“ die Aufrufe, die auf Conversations zu LU6.1-Partneranwendungen und zu openUTM-Clients mit Trägersystem UPIC nicht verwendet werden dürfen.
- unter „openUTM-spezifische Besonderheiten der Aufrufe“ die Besonderheiten, die Sie beim Aufruf der CPI-C-Funktionen unter openUTM beachten müssen.
- unter „Verhalten bei Verwendung nicht unterstützter CPI-C-Aufrufe“ die Returncodes, die openUTM zurückliefert, wenn Sie in einem CPI-C-Programm einen Aufruf verwenden, der in openUTM nicht verfügbar ist.
- unter „Programmierregeln“ die Regeln, die bei der Erstellung von CPI-C-Programmen zu beachten sind.

3.3.1 Unterstützte CPI-C-Aufrufe

Dieser Abschnitt gibt einen Überblick über die in openUTM verfügbaren Aufrufe der CPI-C-Programmschnittstelle in Form einer tabellarischen Übersicht, in der die Aufrufe nach Aufgabengebieten geordnet sind. Die Aufrufe sind alphabetisch nach den Funktionsnamen aufgeführt.

Den vollen hier beschriebenen Funktionsumfang können Sie nur auf Conversations zu OSI-TP-Partnern nutzen. Auf Conversations zu LU6.1-Partnern und openUTM-Clients mit Trägersystem UPIC können die Aufrufe Confirmed und Send_Error nicht verwendet werden (siehe Abschnitt „Einschränkungen bei Conversations über LU6.1- und UPIC-Protokoll“ auf Seite 67).

In den folgenden Tabellen finden Sie die in openUTM verfügbaren CPI-C-Aufrufe.

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

Aufrufe zur Abfrage von Information über Conversation-Charakteristika

Funktion	Aufruf	Beschreibung
Extract_Conversation_State	CMECS	aktuellen Zustand der Conversation abfragen
Extract_Conversation_Type	CMECT	aktuellen Wert der Charakteristik conversation_type abfragen
Extract_Maximum_Buffer_Size	CMEMBS	Maximallänge der Daten, die gesendet bzw. empfangen werden kann, abfragen
Extract_Partner_LU_Name	CMEPLN	Name des aktuellen Partners (Wert der Charakteristik partner_LU_name) abfragen
Extract_Sync_Level	CMESL	aktuellen Wert der Charakteristik sync-level abfragen
Extract_TP_Name	CMETPN	aktuellen Wert der Charakteristik TP_name (Name des aktuellen Partnerprogramms/Transaktionscodes) abfragen

Aufrufe zur Modifikation der Conversation-Charakteristika

Funktion	Aufruf	Beschreibung
Set_Conversation_Type	CMSCT	einen Wert für die Charakteristik conversation_type setzen
Set_Deallocate_Type	CMSDT	einen Wert für die Charakteristik deallocate_type setzen
Set_Partner_LU_Name	CMSPLN	einen Wert für die Charakteristik partner_LU_name setzen
Set_Receive_Type	CMSRT	einen Wert für die Charakteristik receive_type setzen
Set_Send_Type	CMSST	einen Wert für die Charakteristik send_type setzen
Set_Sync_Level	CMSSL	einen Wert für die Charakteristik sync_level setzen
Set_TP_Name	CMSTPN	einen Wert für die Charakteristik TP_name setzen

3.3.2 Einschränkungen bei Conversations über LU6.1- und UPIC-Protokoll

Auf Conversations zu LU6.1-Partnern und openUTM-Clients mit Trägersystem UPIC dürfen die Aufrufe CMCFMD (Confirmed) und CMSERR (Send_Error) nicht verwendet werden:

Confirmed (CMCFMD)

Der Aufruf dient dem Senden einer Empfangsbestätigung. Er ist nur erlaubt, wenn sich die Conversation im Zustand *Confirm-Send* oder *Confirm-Deallocate* befindet, d.h. wenn eine Quittungsanforderung vom fernen Partner empfangen wurde. Das kann auf Conversations zu LU6.1-Partnern und openUTM-Client-Anwendungen mit Trägersystem UPIC nicht vorkommmen. Der Confirmed-Aufruf wird in diesem Fall mit Returncode CM_PROGRAM_STATE_CHECK abgewiesen.

Send_Error (CMSERR)

Der Aufruf dient dazu, die Anforderung einer Empfangsbestätigung negativ zu beantworten (Senden einer negativen Quittung).
Auf Conversations zu einem LU6.1-Partner oder einem openUTM-Client mit Trägersystem UPIC führt der Aufruf Send_Error zum Programmabbruch und dem Verlust aller Conversations, die zu diesem Zeitpunkt vom Programm unterhalten werden. Der Aufruf Send_Error wirkt in diesem Fall wie der Aufruf Cancel_Conversation.

3.3.3 openUTM-spezifische Besonderheiten der CPI-C-Aufrufe

In diesem Abschnitt werden die openUTM-spezifischen Besonderheiten der in openUTM verfügbaren CPI-C-Aufrufe beschrieben. Die Aufrufe selbst sind mit den zugehörigen Error- und Returncodes in „X/Open Preliminary Specification: The CPI-C Specification Version 2 (1994)“ beschrieben. Die Kenntnis dieser Spezifikation ist Voraussetzung für das Erstellen von CPI-C-Programmen.

Accept_Conversation (CMACCP)

Akzeptiert eine Incoming-Conversation, d.h. eine von einem Partner initiierte Conversation.

CMACCP darf innerhalb eines Programmlaufs des CPI-C-Programms unter openUTM höchstens einmal aufgerufen werden. Enthält der Programmcode des CPI-C-Programms mehrere CMACCP-Aufrufe, dann muss die Ablauflogik des Programms dafür sorgen, dass der Aufruf CMACCP in einem Programmlauf höchstens einmal ausgeführt wird.

Wird das CPIC-Programm von openUTM über einen Dialog-Transaktionscode gestartet, dann muss in diesem Programmlauf genau einmal CMACCP aufgerufen werden.

Wird das CPIC-Programm von openUTM über einen Asynchron-Transaktionscode gestartet, dann kann der Aufruf CMACCP entfallen. In diesem Falle kann jedoch nicht auf die Conversation zugegriffen werden, d.h es können keine Daten vom Initiator empfangen werden.

Wird in einem Programm CMACCP ein zweites Mal aufgerufen, dann wird der Aufruf mit dem Returncode CM_PROGRAM_STATE_CHECK abgewiesen. Siehe dazu auch den Abschnitt „Mehrere Conversations in einem CPI-C-Programmlauf“ auf Seite 52.

Allocate (CMALLC)

Baut eine Outgoing-Conversation zu einem Partner auf. Für diesen Aufruf ist Folgendes zu beachten:

– Innerhalb eines Programmlaufs sind zu einem Zeitpunkt maximal 6 offene Outgoing-Conversations erlaubt.

– Die maximale Anzahl von parallelen OSI-TP-Associations/LU6.1-Sessions, die zwischen der lokalen UTM-Anwendung und einer fernen Partneranwendung gleichzeitig bestehen dürfen, ist begrenzt. Sie wird bei der KDCDEF-Generierung festgelegt. Wenn zur Zeit des Allocate-Aufrufs alle Associations bzw. Sessions durch andere Conversations belegt sind, wird der Aufruf mit dem Returncode CM_ALLOCATE_FAILURE_NO_RETRY abgewiesen.

– Die maximale Anzahl der Associations und Sessions, die die lokale UTM-Anwendung gleichzeitig belegen kann, ist generierungsabbhängig.
Wenn zur Zeit des Allocate-Aufrufs die maximale Anzahl an Associations und Sessions bereits durch andere Conversations belegt ist, wird der Aufruf mit dem Returncode CM_ALLOCATE_FAILURE_NO_RETRY abgewiesen.

Der Aufruf CMALLC wird auf den KDCS-Aufruf APRO abgebildet. Returncodes werden umgesetzt wie folgt:

KDCS-Returncode	**CPI-C-Returncode**
APRO 40Z KD10	CM_ALLOCATE_FAILURE_RETRY
APRO 40Z restliche DC-Codes	CM_ALLOCATE_FAILURE_NO_RETRY
APRO 44Z KD04	CM_PRODUCT_SPECIFIC_ERROR
APRO 44Z restliche DC-Codes	CM_PARAMETER_ERROR
APRO 46Z	CM_PARAMETER_ERROR
APRO restliche CC-Codes	CM_PRODUCT_SPECIFIC_ERROR

Cancel_Conversation (CMCANC)

Bricht eine Conversation ab. Bei openUTM wird der Programmlauf mit allen zu diesem Zeitpunkt bestehenden Conversations abgebrochen.

Der Aufruf wird auf PEND FR abgebildet, d.h. das Programm wird mit Fehler beendet und die lokale Transaktion wird zurückgesetzt. Ist der CPI-C-Trace mit Level DUMP oder ALL eingeschaltet, wird der Aufruf intern mit PEND ER abgebrochen und es wird ein Dump erzeugt. Bei schweren Fehlern kann es auch vorkommen, dass openUTM von sich aus einen PEND ER Dump zieht, obwohl die Job- bzw. Umgebungsvariable nicht gesetzt ist. Siehe auch Abschnitt „Fehlerdiagnose in CPI-C-Programmen“ auf Seite 99.

Durch den Aufruf Cancel_Conversation können Nachrichten, die bereits an den Initiator geschickt wurden, verlorengehen.

Confirmed (CMCFMD)

Führt zum Senden einer Empfangsbestätigung (positive Quittung).

Der Aufruf ist in einer Conversation mit einem LU6.1- oder openUTM-Client-Partner mit Trägersystem UPIC nicht erlaubt.

Er wird in openUTM unterstützt, um die Quittungsanforderung eines fernen Partners an einem Fremdsystem zu beantworten. CPI-C-Programme unter openUTM können keine Quittungen anfordern (die Angabe *sync_level*=CM_CONFIRM ist unter openUTM nicht erlaubt).

Laufen beide Partnerprogramme der Conversation unter openUTM, hat der Confirmed-Aufruf keine Wirkung.

Convert_Incoming (CMCNVI)

Konvertiert mit CMRCV (Receive) empfangene Daten in den lokalen Zeichensatz.
CMCNVI geht davon aus, dass die im Parameter *buffer* übergebenen Daten im Zeichensatz EBCDIC Code Page 500/1 vorliegen, und wandelt sie in den vom lokalen System verwendeten Zeichensatz um.

B
B
Unter BS2000-Systemen geht CPI-C davon aus, dass der lokal verwendete Zeichensatz EBCDIC.DF.04 ist.

X/W
X/W
Unter Unix- und Windows-Systemen geht CPI-C davon aus, dass der lokal verwendete Zeichensatz ISO 8859-1 ist, d.h. der ISO-8-Bit-Code (ASCII).

Bei einer heterogenen Kopplung sollten Sender und Empfänger konvertieren.
Bei einer homogenen Kopplung kann die Konvertierung entfallen; sie muss jedoch bei Sender und Empfänger entfallen. Siehe Abschnitt „Konvertierung von Characteristika und Benutzerdaten“ auf Seite 55.

Convert_Outgoing (CMCNVO)

Konvertiert Daten in den Zeichensatz EBCDIC Code Page 500/1, die mit CMSEND (Send_Data) gesendet werden sollen.
CMCNVO geht davon aus, dass die im Parameter *buffer* übergebenen Daten in dem vom lokalen System verwendeten Zeichensatz vorliegen, und wandelt sie in den Zeichensatz EBCDIC Code Page 500/1 um.

B
B
Unter BS2000-Systemen geht CPI-C davon aus, dass der lokal verwendete Zeichensatz EBCDIC.DF.04 ist.

X/W
X/W

Unter Unix- und Windows-Systemen geht CPI-C davon aus, dass der lokal verwendete Zeichensatz ISO 8859-1 ist, d.h. der ISO-8-Bit-Code (ASCII).

Bei einer heterogenen Kopplung sollten Sender und Empfänger konvertieren.
Bei einer homogenen Kopplung kann die Konvertierung entfallen; sie muss jedoch bei Sender und Empfänger entfallen. Siehe Abschnitt „Konvertierung von Characteristika und Benutzerdaten“ auf Seite 55.

Deallocate (CMDEAL)

Beenden einer Conversation und freigeben der Conversation_ID.

Wenn die Conversation-Charakteristik *deallocate_type* den Wert CM_DEALLOCATE_SYNC_LEVEL oder CM_DEALLOCATE_FLUSH hat, gilt Folgendes:

– In einer Asynchron-Conversation darf nur der Initiator (Client) die Conversation normal beenden und CMDEAL aufrufen.

– In einer Dialog-Conversation darf nur der Akzeptor (Server) die Conversation normal beenden und CMDEAL aufrufen. Außerdem muss der Akzeptor zwischen dem Übergang der Conversation in den Zustand *Send* bzw. *Send-Pending* und dem Aufruf von CMDEAL mindestens eine Nachricht mit dem Aufruf Send_Data an den Initiator gesendet haben.

Siehe auch Abschnitt „Programmierregeln“ auf Seite 81.

Wenn die Conversation-Charakteristik *deallocate_type* den Wert CM_DEALLOCATE_ABEND hat, wird durch den Deallocate-Aufruf das aufrufende Programm mit allen Conversations, die es zu diesem Zeitpunkt unterhält, abgebrochen. Der Aufruf Deallocate wirkt dann wie der Aufruf Cancel_Conversation.

Extract_Conversation_State (CMECS)

Aktuellen Zustand der Conversation abfragen.

Es gibt keine Einschränkungen zu diesem Aufruf in openUTM.

Extract_Conversation_Type (CMECT)

Abfragen des aktuellen Werts der Conversation-Charakteristik *conversation_type*.

Ergebnis dieses Aufrufs kann bei CPI-C unter openUTM immer nur der Wert CM_MAPPED_CONVERSATION sein.

Extract_Maximum_Buffer_Size (CMEMBS)

Abfragen der Maximallänge der Daten, die mit dem Aufruf Send_Data gesendet bzw. mit dem Receive-Aufruf empfangen werden können.

Bei openUTM ist die maximale Puffergröße auf 32767 Byte eingestellt (wie bei X/Open vorgeschlagen). Wird bei CPI-C-Aufrufen in den Parametern *buffer_length*, *request_length*, *send_length* ein Wert angegeben, der größer als die durch den Aufruf erhaltene Puffergröße ist, dann wird der entsprechende CPI-C-Aufruf mit CM_PROGRAM_PARAMETER_CHECK abgewiesen.

Es muss also gelten:

0 ≤ ..._length ≤ CM_MAXIMUM_BUFFER_SIZE

Extract_Partner_LU_Name (CMEPLN)

Abfragen des aktuellen Werts der Conversation-Charakteristik *partner_LU_name*.

Bei Incoming-Conversations liefert der Aufruf den Namen der Partneranwendung zurück. Bei openUTM wird bei Conversations über das UPIC-Protokoll der Name des LTERM-Partners, bei der Kommunikation über LU6.1 der Name des LPAP-Partners und bei der Kommunikation über OSI-TP der Name des OSI-LPAP-Partners zurückgeliefert (siehe Seite 45f).

Bei Outgoing-Conversations liefert der Aufruf den Namen der Partneranwendung nur dann zurück, wenn die Conversation-Charakteristik *partner_LU_name* zuvor mit dem Aufruf Set_Partner_LU_Name gesetzt worden ist. Sonst werden Leerzeichen zurückgeliefert.

Extract_Sync_Level (CMESL)

Abfragen des aktuellen Werts der Conversation-Charakteristik *sync_level*.

Es gibt keine Einschränkungen zu diesem Aufruf bei openUTM.

Extract_TP_Name (CMETPN)

Abfragen des aktuellen Werts der Conversation-Charakteristik *TP_name*.

Bei einer Incoming-Conversation liefert der Aufruf den Transaktionscode zurück, mit dem das lokale Programm gestartet wurde. Haben Sie für ein CPI-C-Programm mehrere Transaktionscodes definiert, dann können Sie mit CMETPN den Transaktionscode ermitteln, mit dem das Programm gestartet wurde.

Bei Outgoing-Conversations wird der LTAC-Name des Partners zurückgeliefert, der mit dem Aufruf CMSTPN (Set_TP_Name) gesetzt wurde. Wurde für die Conversation *TP_name* nicht explizit gesetzt, dann wird der beim Aufruf CMINIT (Initialize_Conversation) im Parameter *sym_dest_name* übergebene Wert zurückgeliefert.

Initialize_Conversation (CMINIT)

Initialisieren der Conversation-Charakteristika einer Outgoing-Conversation.

Der beim Aufruf mitgegebene symbolische Partnername *sym_dest_name* muss bei openUTM als LTAC-Name generiert sein (siehe auch Abschnitt „Conversation-Charakteristika für die Adressierung“ auf Seite 45). Erst beim Allocate-Aufruf wird überprüft, ob *sym_dest_name* ein gültiger LTAC-Name ist.

Innerhalb eines Programmlaufs sind zu einem Zeitpunkt maximal sechs offene Outgoing-Conversations erlaubt. Unterhält das CPI-C-Programm sechs offene Outgoing-Conversations und wird mit CMINIT versucht, eine siebente zu initialisieren, dann wird der Aufruf mit dem Returncode CM_PRODUCT_SPECIFIC_ERROR abgewiesen (siehe auch Abschnitt „Mehrere Conversations in einem CPI-C-Programmlauf“ auf Seite 52).

Receive (CMRCV)

Daten vom Partner empfangen.

Bei Conversations mit *sync_level* = CM_SYNC_POINT oder s*ync_level* = CM_SYNC_POINT_NO_CONFIRM wird für den Parameter *status_received* der Wert CM_TAKE_COMMIT oder CM_TAKE_COMMIT_DEALLOCATE bzw. CM_TAKE_BACKOUT oder CM_TAKE_BACKOUT_DEALLOCATE zurückgeliefert.

Bei openUTM ist der Receive-Aufruf im Zustand *Send-Pending* nicht erlaubt. Im Zustand *Send* ist der Aufruf nur erlaubt, wenn in diesem Zustand vorher schon CMSEND (Send_Data) aufgerufen wurde oder wenn der Zustand *Send* aus dem Zustand *Send_Pending* hervorgegangen ist.

Überträgt ein CPI-C-Server-Programm unter openUTM das Senderecht auf den fernen Partner, d.h ruft es CMRCV im Zustand *Send* oder nach einem CMSEND mit *send_type* = CM_SEND_AND_PREP_TO_RECEIVE auf, dann kann es vorkommen, dass das Programm von diesem Receive-Aufruf nicht mehr zurückkehrt und von openUTM abgebrochen wird.

Das kann folgende Ursachen haben:

– Die logische Verbindung zum Partner, über die die Conversation aufgebaut ist, ist verloren gegangen.

– Der Partner hat die Conversation abnormal beendet.

– Die Zeit, die das lokale CPI-C-Teilprogramm maximal auf eine Nachricht vom Partner warten darf, ist abgelaufen, ohne dass der Partner eine Nachricht gesendet hat. Diese maximale Wartezeit wird definiert mit dem Generierungsparameter MAX PGWTTIME=.

openUTM schreibt eine Meldung mit dem Grund für den Abbruch des Programmlaufs in die SYSLOG-Datei.

Der Receive-Aufruf wird auf den KDCS-Aufruf MGET abgebildet. Tritt beim MGET ein KDCS-Returncode ungleich 000, 01Z, 03Z, 10Z oder 12Z auf, wird das Programm abnormal mit PEND ER beendet.

Send_Data (CMSEND)

Daten an den Partner senden.

Der Aufruf CMSEND wird auf den KDCS-Aufruf FPUT abgebildet. Tritt beim FPUT ein KDCS-Returncode ungleich 000 auf, wird der CPI-C-Returncode CM_PRODUCT_SPECIFIC_ERROR erzeugt und im CPI-C-Trace wird der UTM-Fehler 40Z mit K704 protokolliert.

Send_Error (CMSERR)

Senden einer Fehlernachricht an den fernen Partner.

Der Aufruf darf nur auf einer Dialog-Conversation verwendet werden.

Bei Conversations über das LU6.1- und UPIC-Protokoll ist der Aufruf nicht erlaubt.

Set_Conversation_Type (CMSCT)

Setzen der Conversation-Charakteristik *conversation_type*.

Der Aufrufparameter *conversation_type* darf nur den Wert CM_MAPPED_CONVERSATION haben.
Der Wert CM_BASIC_CONVERSATION wird von openUTM nicht unterstützt und mit dem Returncode CM_PRODUCT_SPECIFIC_ERROR abgewiesen.

Set_Deallocate_Type (CMSDT)

Setzen der Conversation-Charakteristik *deallocate_type*.

Für den Parameter *deallocate_type* darf bei openUTM nur einer der folgenden Werte angegeben werden:

– CM_DEALLOCTE_SYNC_LEVEL,

– CM_DEALLOCATE_FLUSH oder

– CM_DEALLOCATE_ABEND

Wird CM_DEALLOCATE_ABEND angegeben, dann führt der nachfolgende Aufruf Deallocate dazu, dass das Programm mit allen Conversations abgebrochen wird. Die Wirkung dieses Deallocate-Aufrufs entspricht dann der Wirkung des Aufrufs Cancel_Conversation.

Der Parameterwert CM_DEALLOCATE_CONFIRM wird von openUTM mit dem Returncode CM_PRODUCT_SPECIFIC_ERROR abgewiesen.

Set_Partner_LU_Name (CMSPLN)

Setzen der Conversation-Charakteristik *partner_LU_name*.

Der im Parameter *partner_LU_name* angegebene Wert darf maximal 8 byte lang sein. Er muss bei der KDCDEF-Generierung der Anwendung als LPAP-Name oder OSI-LPAP-Name definiert sein.
Wird im Parameter *partner_LU_name_length* ein Wert größer 8 byte angegeben, dann wird der Aufruf von openUTM mit dem Returncode CM_PRODUCT_SPECIFIC_ERROR abgewiesen.

Set_Receive_Type (CMSRT)

Setzen der Conversation-Charakteristik *receive_type*.

Befindet sich die Conversation im Zustand *Receive*, dann hat der Wert von *receive_type* keine Bedeutung, da ein CPI-C-Programm von openUTM erst aufgerufen wird, wenn bereits alle Nachrichten eingetroffen sind.

Die Einstellung des *receive_type* wirkt sich bei openUTM nur aus, wenn der Receive-Aufruf im Zustand *Send* oder *Send-Pending* erfolgt. In diesem Fall wird ein Aufruf mit *receive_type*=CM_RECEIVE_IMMEDIATE mit dem Returncode CM_PROGRAM_STATE_CHECK abgewiesen.

Set_Send_Type (CMSST)

Setzen der Conversation-Charakteristik *send_type*.

Für den Aufrufparameter *send_type* dürfen bei openUTM nur folgende Werte angegeben werden:

– CM_BUFFER_DATA,
Daten ohne zusätzliche Information an den Partner senden

– CM_SEND_AND_PREP_TO_RECEIVE,
die übergebenen Daten werden sofort an das Partnerprogramm gesendet, das Senderecht wird an den Partner übertragen.

– CM_SEND_AND_DEALLOCATE,
Daten senden und Conversation normal beenden.

Die Werte CM_SEND_AND_FLUSH und CM_SEND_AND_CONFIRM sind nicht erlaubt und werden mit dem Returncode CM_PRODUCT_SPECIFIC_ERROR abgewiesen.

Set_Sync_Level (CMSSL)

Setzen der Conversation-Charakteristik *sync_level*.

Für den Aufrufparameter *sync_level* dürfen bei openUTM nur folgende Werte angegeben werden:

– CM_NONE
für die Conversation wird keine Synchronisation angefordert.

– CM_SYNC_POINT_NO_CONFIRM
die Conversation wird in eine globale Transaktion eingeschlossen.

Der Wert CM_CONFIRM wird mit dem Returncode CM_PRODUCT_SPECIFIC_ERROR abgewiesen; der Wert CM_SYNC_POINT wird mit dem Returncode CM_PARM_VALUE_NOT_SUPPORTED abgewiesen.

Set_TP_Name (CMSTPN)

Setzen der Conversation-Charakteristik *TP_Name*.

Der für den Aufrufparameter *TP_Name* angegebene Wert muss in openUTM als LTAC-Name definiert sein und darf maximal 8 byte lang sein.

Bei einer Längenangabe größer 8 byte in *TP_name_length* wird der Aufruf von openUTM mit dem Returncode CM_PRODUCT_SPECIFIC_ERROR abgewiesen.

3.3.4 Zusammenarbeit mit der TX-Schnittstelle

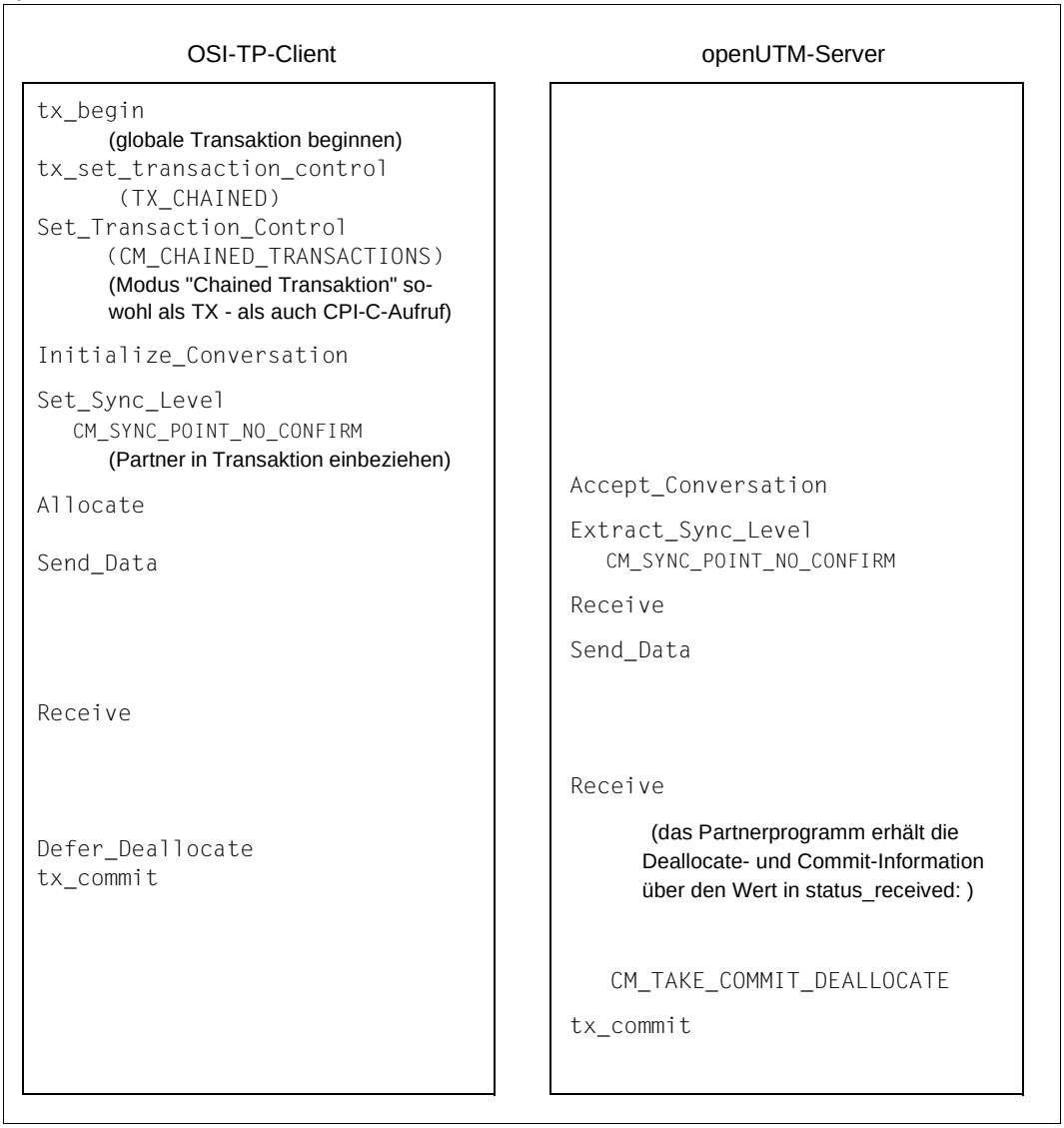
openUTM unterstützt globale Transaktionen in Conversations über die TX-Schnittstelle. Eine Beschreibung der TX-Schnittstelle finden Sie im Kapitel 5.

Bitte beachten Sie: Wenn Sie die TX-Schnittstelle verwenden und UTM-Server-Programme in eine globale Transaktion einbeziehen, verhalten sich bestehende UTM-Server-Programme anders, als ohne Einbeziehung in eine globale Transaktion.

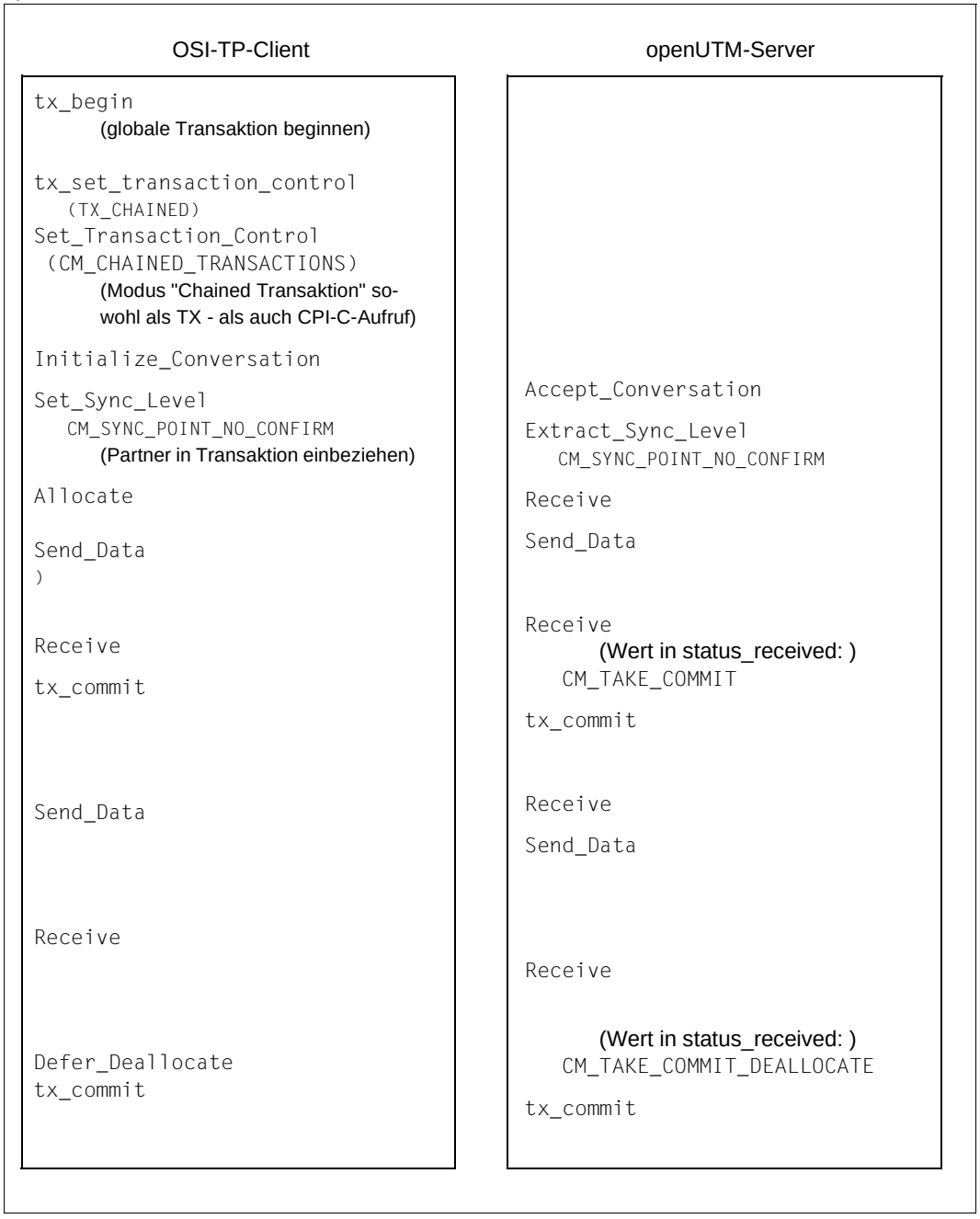
Insbesondere darf das Transaktionsende nur vom Client-Programm angefordert werden!

Im Folgenden ist schematisch der Ablauf für eine Einschritt- und eine Mehrschritt-Transaktion dargestellt.

1) Einschritt-Transaktion



2) Mehrschritt-Transaktion



3.3.5 Verhalten bei Verwendung nicht unterstützter CPI-C-Aufrufe

Enthält ein CPI-C-Programm unter openUTM CPI-C-Aufrufe, die openUTM nicht unterstützt, dann kann es zwar mit UTM gebunden werden, ist unter openUTM jedoch nicht ablauffähig.

Folgende Aufrufe werden mit dem Returncode CM_PRODUCT_SPECIFIC_ERROR abgewiesen:

Name der Funktion	**Aufruf**
Confirm	CMCFM
Extract_Mode_Name	CMEMN
Flush	CMFLUS
Prepare_To_Receive	CMPTR
Request_To_Send	CMRTS
Set_Log_Data	CMSLD
Set_Mode_Name	CMSMN
Set_Return_Control	CMSRC
Test_Request_To_Send_Received	CMTRTS

Der folgende Aufruf wird mit dem Returncode CM_PROGRAM_PARAMETER_CHECK abgewiesen, da er nur bei Basic Conversation erlaubt ist und openUTM keine Basic Conversation unterstützt.

Name der Funktion	**Aufruf**
Set_Fill	CMSF

Alle anderen optionalen CPI-C-Aufrufe, die openUTM nicht unterstützt , werden mit dem Returncode CM_CALL_NOT_SUPPORTED abgewiesen.

3.3.6 Prozess- bzw. Taskwechsel

X/W
X/W Innerhalb eines CPI-C-Programmlaufs findet kein Prozesswechsel statt. Eine Conversation ist an einen Prozess gebunden.

B
B Innerhalb eines CPI-C-Programmlaufs findet kein Taskwechsel statt. Eine Conversation ist an eine Task gebunden.

3.3.7 Programmierregeln

Bei der Erstellung von CPI-C-Teilprogrammen unter openUTM müssen Sie die im Folgenden beschriebenen Programmierregeln beachten. Die Regeln beziehen sich auf Conversations zu OSI-TP-Partnern. Bei Conversations zu LU6.1-Partnern und zu openUTM-Clients mit Trägersystem UPIC gelten zusätzliche Einschränkungen. Diese Einschränkungen sind auf Seite 67 zusammengefasst.

Bei der Erstellung von CPI-C-Teilprogrammen muss zwischen *Dialog-Conversations* und *Asynchron-Conversations* unterschieden werden.

Bei *Dialog-Conversations* können beide Seiten der Conversations senden, d.h. sowohl der Initiator (Client) als auch der Akzeptor (Server). Dem Akzeptor einer Dialog-Conversation muss in der UTM-Anwendung ein Dialog-Transaktionscode zugeordnet werden. Dialog-Transaktionscodes sind alle generierten Transaktionscodes, denen der Parameter TYPE=D zugeordnet ist (Standardeinstellung).

Bei der KDCDEF-Generierung müssen Sie also Folgendes angeben:

– `TAC tac-name,...[,TYPE=D]`
  wenn das lokale CPI-C-Programm Akzeptor (Server) der Dialog-Conversation ist.

– `LTAC ltac-name,...[,TYPE=D]`
  wenn das Partner-Programm Akzeptor (Server) der Dialog-Conversation ist.

Bei *Asynchron-Conversations* kann nur der Initiator (Client) der Conversation senden. Dem Akzeptor (Server) einer Asynchron-Conversation muss in der UTM-Anwendung ein Asynchron-Transaktionscode zugeordnet werden. Asynchron-Transaktionscodes sind alle generierten Transaktionscodes, denen der Parameter TYPE=A zugeordnet ist.

Bei der KDCDEF-Generierung müssen Sie also Folgendes angeben:

– `TAC tac-name,...,TYPE=A`
  wenn das lokale CPI-C-Programm Akzeptor (Server) der Asynchron-Conversation ist.

– `LTAC ltac-name,...,TYPE=A`
  wenn das Partner-Programm Akzeptor (Server) der Asynchron-Conversation ist.

Der Aufbau eines CPI-C-Programms innerhalb einer Conversation ist davon abhängig, ob es sich um eine Asynchron- oder Dialog-Conversation handelt und ob das CPI-C-Programm Initiator oder Akzeptor der Conversation ist.

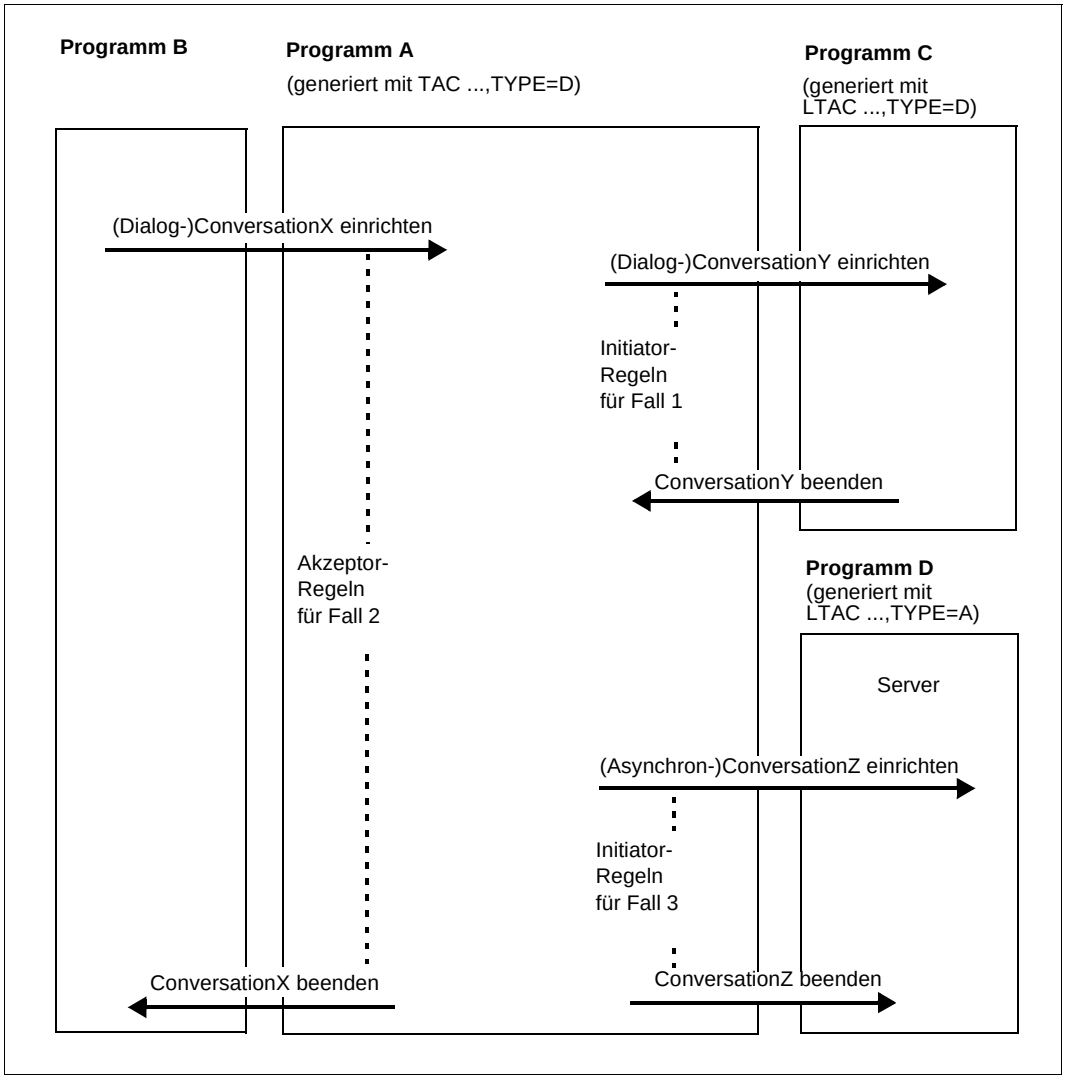
Es müssen folgende Fälle unterschieden werden:

1. Das CPI-C-Programm ist Initiator einer Dialog-Conversation (Outgoing-Conversation).
2. Das CPI-C-Programm ist Akzeptor einer Dialog-Conversation (Incoming-Conversation).
3. Das CPI-C-Programm ist Initiator einer Asynchron-Conversation (Outgoing-Conversation).
4. Das CPI-C-Programm ist Akzeptor einer Asynchron-Conversation (Incoming-Conversation).

Im Folgenden sind die Programmierregeln für diese vier Fälle aufgeführt. Dabei wird unterschieden zwischen den Regeln für CPI-C-Programme, die unter openUTM ablaufen, und CPI-C-Programme, die zwar nicht unter openUTM ablaufen, aber Conversations zu CPI-C-Programmen unter openUTM unterhalten (z.B. openUTM-Clients, die die Schnittstelle CPI-C nutzen).

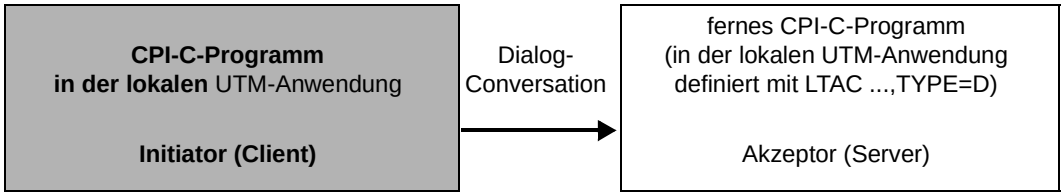
Die Programmierregeln gelten jeweils für eine Conversation. Jedes Programm kann mehrere Conversations unterhalten. Für jede dieser Conversations kann ein anderer der obigen Fälle zutreffen.

Im folgenden Bild ist dieser Sachverhalt beispielhaft dargestellt.
Programm A ist ein CPI-C-Programm unter openUTM, das über einen Dialog-Transaktionscode generiert ist. Es wird gestartet durch die Anforderung von Programm B. Programm A ist also Akzeptor (Server) einer Dialog-Conversation. Programm A baut weitere Conversations auf, eine Dialog-Conversation zum Programm C und eine Asynchron-Conversation zum Programm D. Für diese Conversations ist Programm A der Initiator (Client).


Programm B
Programm A
(generiert mit TAC ...,TYPE=D)
Programm C
(generiert mit
LTAC ...,TYPE=D)
(Dialog-)ConversationX einrichten
(Dialog-)ConversationY einrichten
Initiator-
Regeln
für Fall 1
ConversationY beenden
Akzeptor-
Regeln
für Fall 2
Programm D
(generiert mit
LTAC ...,TYPE=A)
Server
(Asynchron-)ConversationZ einrichten
Initiator-
Regeln
für Fall 3
ConversationX beenden
ConversationZ beenden

Fall 1: Initiator einer Dialog-Conversation

Die folgenden Programmierregeln gelten für die Seite der Conversation, die im Bild gerastert dargestellt ist.



Für den Initiator einer Dialog-Conversation gelten unter openUTM folgende Regeln:

Regel 1:

Nach dem Initialisieren und Einrichten der Conversation muss der Initiator mindestens eine Nachricht (evtl. der Länge 0) mit dem Aufruf Send_Data an das Partnerprogramm senden, bevor er den Zustand *Send* verlassen darf (im Zustand *Send* befindet sich der Initiator nach dem erfolgreichen Allocate-Aufruf.

Somit ist die folgende Aufrufreihenfolge **nicht** erlaubt.

```
Initialize_Conversation
Allocate
Receive
```

Regel 2:

Die Conversation kann in diesem Fall nur vom Akzeptor (Server) normal beendet werden, d.h. der Initiator kann die Conversation nur abnormal beenden mit:

```
Cancel_conversation
```

oder

```
Set_Deallocate_Type (deallocate_type=CM_DEALLOCATE_ABEND)
Deallocate
```

Es werden unter openUTM dann allerdings alle Conversations abgebaut, die das Programm zu diesem Zeitpunkt unterhält, und der Programmlauf beendet.

Der Initiator einer Dialog-Conversation muss also folgenden Programmaufbau haben:

```
Initialize_Conversation
........eventuell Set_Partner_LU_Name
........eventuell Set_TP_Name
Allocate
.......eventuell Convert_Outgoing
Send_Data
......eventuell weitere Send_Data
Receive
.....eventuell Convert_Incoming
.....eventuell weitere Receive zum Empfangen von  Daten
.....eventuell weitere Abfolgen von Send_Data und Receive
Receive zum Warten auf den return_code CM_DEALLOCATED_NORMAL
```

Zur Abfrage von Information können an beliebiger Stelle innerhalb der Conversation zusätzlich folgende CPI-C-Aufrufe aufgerufen werden:

– Extract_Conversation_State
– Extract_Maximum_Buffer_Size

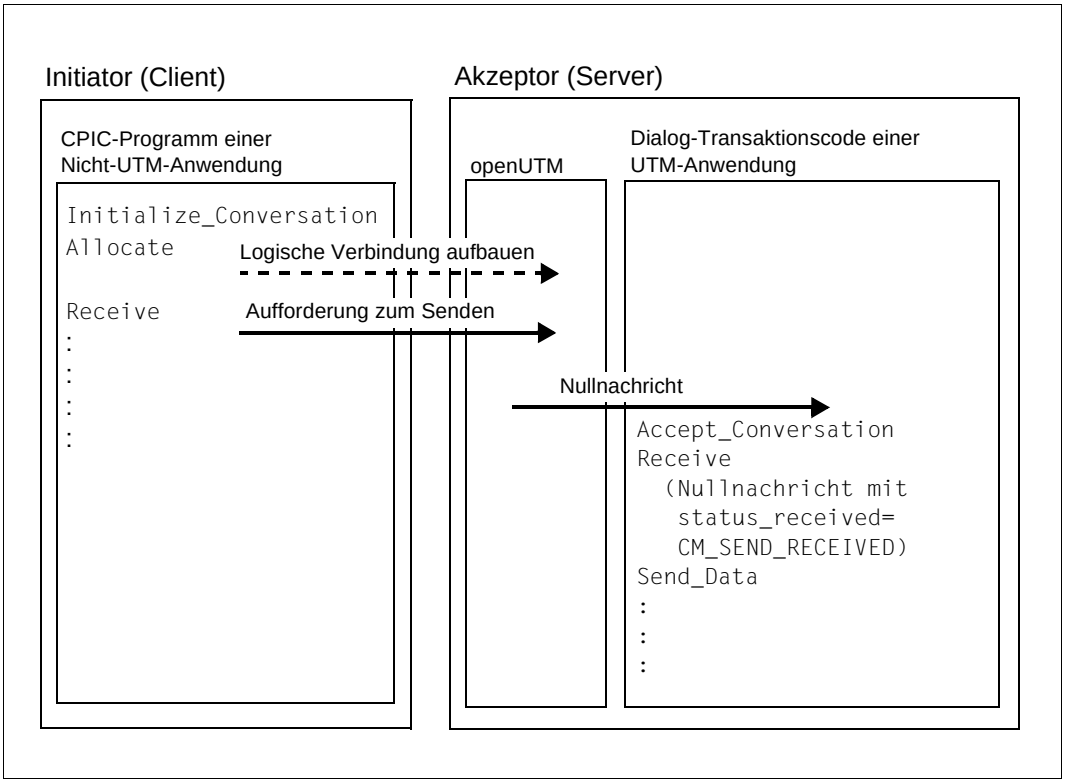
Im Fehlerfall können folgende CPI-C-Aufrufe verwendet werden:

– Send_Error
  (Ist bei Conversations zu LU6.1-Partnern und zu einem openUTM-Client mit Trägersystem UPIC nicht erlaubt. Der Aufruf führt in diesem Fall zum Programmabbruch und dem Verlust aller Conversations.)
– Cancel_Conversation
– Set_Deallocate_Type mit deallocate_type=CM_DEALLOCATE_ABEND gefolgt von Deallocate

Läuft der Initiator der Conversation nicht unter openUTM ab (der Akzeptor (Server) ist jedoch ein CPI-C-Programm unter openUTM), dann gilt die Regel 1 nicht.

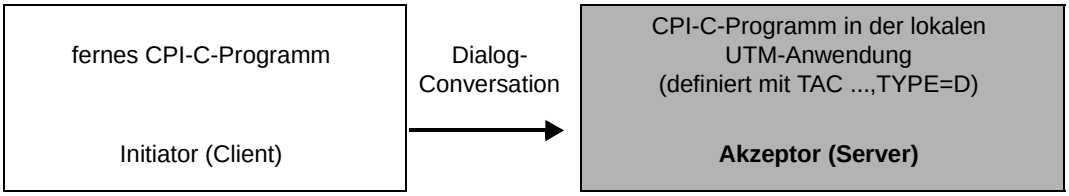
Verlässt der Initiator nach dem Allocate-Aufruf den Zustand *Send*, ohne zuvor mit dem Aufruf Send_Data eine Nachricht an den unter openUTM laufenden Akzeptor gesendet zu haben, dann erhält der Akzeptor von openUTM eine Nachricht der Länge Null.

Es ergibt sich folgender Programmablauf:



Fall 2: Akzeptor einer Dialog-Conversation

Die folgenden Programmierregeln gelten für die Seite der Conversation, die im Bild gerastert dargestellt ist.



Für den Akzeptor einer Dialog-Conversation gelten unter openUTM folgende Regeln:

Regel 1:

Als erste Funktion muss Accept_Conversation (CMACCP) aufgerufen werden. Mit diesem Aufruf wird die Conversation zum Initiator eröffnet.

Regel 2:

Nach dem Aufruf Accept_Conversation muss der Akzeptor mit dem Aufruf Receive (CMRCV) solange lesen, bis er das Senderecht erhält. Danach darf das Senderecht mit Hilfe der Aufrufe Send_Data (CMSEND) und Receive (CMRCV) beliebig oft gewechselt werden.

Regel 3:

Der Akzeptor muss die Conversation irgendwann mit dem Aufruf Deallocate (CMDEAL) oder Cancel_Conversation (CMCANC) aktiv abbauen, sofern die Conversation nicht abnormal vom Initiator oder vom Transportsystem beendet wird. Ein abnormaler Abbau von Seiten des Initiators oder vom Transportsystem liegt dann vor, wenn ein CPI-C-Aufruf für diese Conversation mit einem Returncode zurückkehrt, der die Conversation in den Reset-Zustand überführt. (Zustandsübergänge siehe „X/Open Preliminary Specification CPI-C The Specification Version 2 (1994)“; State Tables im Appendix C).

Regel 4:

Zwischen dem Eintreten in den Zustand Send und dem Deallocate-Aufruf muss das Programm mittels Send_Data mindestens eine Nachricht (evt. der Länge Null) an den Initiator senden, sonst wird der Deallocate-Aufruf mit CM_PRODUCT_SPECIFIC_ERROR abgewiesen.

Das heißt, folgende Aufrufreihenfolge ist **nicht** erlaubt:

```
Receive
        CM_SEND_RECEIVED
Deallocate
```

Der Akzeptor einer Dialog-Conversation muss also folgenden Programmaufbau haben:

```
Accept_Conversation
........eventuell Extract_Partner_LU_Name
........eventuell Extract_TP_Name
........eventuell Extract_Sync_Level
Receive
........eventuell Convert_Incoming
........eventuell weitere Receive zum Empfangen von  Daten
Receive zum Warten auf status_received=CM_SEND_RECEIVED
........eventuell Convert_Outgoing
Send_Data
........eventuell weitere Abfolgen von Receive und Send_Data

Deallocate

oder

Set_Send_Type (send_type=CM_SEND_AND_DEALLOCATE)
Send_Data
```

Zur Abfrage von Information können an beliebiger Stelle innerhalb der Conversation zusätzlich folgende CPI-C-Aufrufe aufgerufen werden:

– Extract_Conversation_State
– Extract_Maximum_Buffer_Size

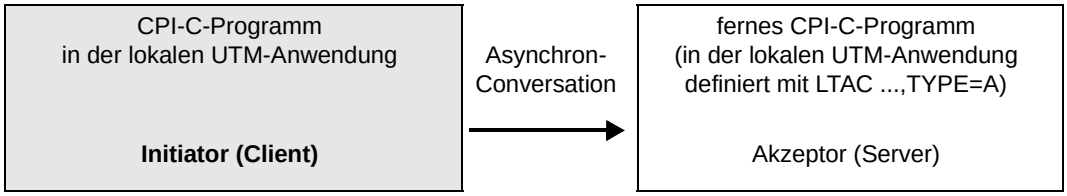
Im Fehlerfall können folgende CPI-C-Aufrufe verwendet werden:

– Send_Error
  (Ist bei Conversations zu LU6.1-Partnern und zu einem openUTM-Client mit Trägersystem UPIC nicht erlaubt. Der Aufruf führt in diesem Fall zum Programmabbruch und dem Verlust aller Conversations.)
– Cancel_Conversation
– Set_Deallocate_Type mit *deallocate_type*=CM_DEALLOCATE_ABEND gefolgt von Deallocate

Läuft der Akzeptor der Conversation nicht unter openUTM ab, dann gilt die Regel 4 nicht. Sendet der Akzeptor zwischen Eintreten in den Zustand *Send* und dem Deallocate-Aufruf keine Nachricht an den unter openUTM laufenden Initiator, dann erhält der Initiator von openUTM eine Nachricht der Länge Null.

Fall 3: Initiator einer Asynchron-Conversation

Die folgenden Programmierregeln gelten für die Seite der Conversation, die im Bild gerastert dargestellt ist.



Für den Initiator einer Asynchron-Conversation gelten unter openUTM folgende Regeln:

Regel 1:

Der Initiator darf die folgenden CPI-C-Aufrufe **nicht** verwenden:

CMCFMD (Confirmed)

Confirmed ist nur in den Zuständen *Confirm-Send* und *Confirm-Deallocate* erlaubt. Diese Zustände können beim Initiator einer Asynchron-Conversation nicht auftreten, da der Akzeptor keine Quittungsanforderung an den Initiator senden kann.

Läuft der Initiator der Conversation nicht unter openUTM ab, dann darf dieses Programm zwar den Deallocate-Aufruf mit *deallocate_type*= CM_DEALLOCATE_CONFIRM enthalten, aber openUTM stellt dem unter seiner Regie ablaufenden Akzeptor diese Quittungsanforderung nicht zu. openUTM sendet stattdessen selbst eine positive Quittung an den Initiator, sobald der Akzeptor (Server) der Conversation alle Nachrichten entgegengenommen hat.

CMRCV (Receive)

Der Akzeptor (Server) einer Asynchron-Conversation sendet keine Nachrichten an den Initiator. Ein Receive-Aufruf im Initiator ist somit sinnlos und wird mit CM_PRODUCT_SPECIFIC_ERROR abgewiesen.

CMSERR (Send_Error)

Der Aufruf Send_Error wird mit dem Returncode CM_PRODUCT_SPECIFIC_ERROR abgewiesen

CMSST (Set_Send_Type) mit *send_type*=CM_SEND_AND_PREP_TO_RECEIVE.

Der Akzeptor sendet keine Nachrichten an den Initiator der Conversation. Deshalb wird der Versuch, dem Akzeptor (Server) das Senderecht zu übertragen, mit dem Returncode CM_PRODUCT_SPECIFIC_ERROR abgewiesen.

Regel 2:

Laufen sowohl Initiator als auch Akzeptor unter openUTM ab, dann muss der Initiator mindestens eine Nachricht (evt. mit der Länge 0) mit dem Aufruf Send_Data an den Akzeptor senden.
Somit kann der Akzeptor nicht mit der folgenden einfachen Aufrufreihenfolge gestartet werden:

```
Initialize_Conversation
Allocate
Deallocate
```

Läuft der Akzeptor nicht unter openUTM ab, entfällt die Regel.

Die Client-Seite einer Asynchron-Conversation muss also folgenden Programmaufbau haben:

```
Initialize_Conversation
........eventuell Set_Partner_LU_Name
........eventuell Set_TP_Name
Allocate
........eventuell Convert_Outgoing
Send_Data
........eventuell weitere Send_Data

Deallocate
oder
Set_Send_Type (send_type=CM_SEND_AND_DEALLOCATE)
Send_Data
```

Zur Abfrage von Information können an beliebiger Stelle innerhalb der Conversation zusätzlich folgende CPI-C-Aufrufe aufgerufen werden:

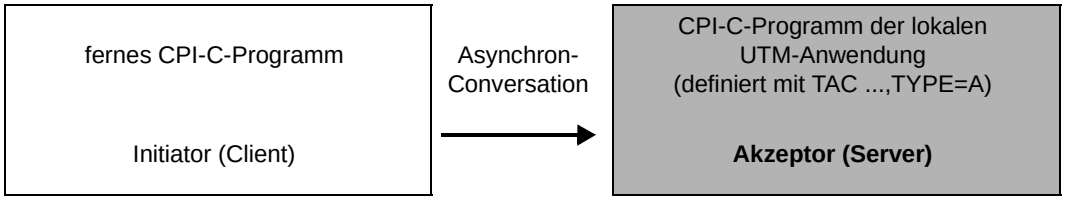
– Extract_Conversation_State
– Extract_Maximum_Buffer_Size

Im Fehlerfall können folgende CPI-C-Aufrufe verwendet werden:

– Cancel_Conversation
– Set_Deallocate_Type mit *deallocate_type*=CM_DEALLOCATE_ABEND gefolgt von Deallocate

Fall 4: Akzeptor der Asynchron-Conversation

Die folgenden Programmierregeln gelten für die Seite der Conversation, die im Bild gerastert dargestellt ist.



Für den Akzeptor einer Asynchron-Conversation gelten unter openUTM folgende Regeln:

Regel 1:

Der Akzeptor darf die folgenden CPI-C-Aufrufe **nicht** verwenden:

Confirmed (CMCFMD)

Confirmed ist nur in den Zuständen *Confirm-Send* und *Confirm-Deallocate* erlaubt. Diese Zustände können beim Akzeptor einer Asynchron-Conversation, der unter openUTM abläuft, nicht auftreten.

Schickt ein über das Protokoll OSI-TP angeschlossener Initiator, der nicht unter openUTM abläuft, dem unter openUTM ablaufenden Akzeptor einer Asynchronconversation eine Deallocate-Aufforderung mit Bestätigungsanforderung, so wird dies dem CPI-C-Programm unter openUTM nicht wie üblich beim Receive-Aufruf mit *status_received* = CM_CONFIRM_DEALLOC_RECEIVED gemeldet, sondern mit return_code = CM_DEALLOCATED_NORMAL.
openUTM schickt die positive Quittung selbst an den Initiator, sobald der Lauf des lokalen CPI-C-Programms beendet ist.

Send_Data (CMSEND)

Der Aufruf wird mit dem Returncode CM_PRODUCT_SPECIFIC_ERROR abgewiesen.

Send_Error (CMSERR)

Der Aufruf wird mit dem Returncode CM_PRODUCT_SPECIFIC_ERROR abgewiesen.

Set_Conversation_Type (CMSCT)

Der Aufruf ist nur im Zustand *Initialize* erlaubt. Dieser Zustand kann in diesem Fall nicht auftreten.

Regel 2:

Eine Asynchron-Conversation kann nur der Initiator der Conversation normal beenden. Ein unter openUTM ablaufender Akzeptor darf Deallocate (CMDEAL) nur aufrufen, wenn zuvor mit dem Aufruf Set_Deallocate_Type die Conversation-Charakteristik *deallocate_type* auf CM_DEALLOCATE_ABEND gesetzt wurde.

Der Akzeptor (Server) einer Asynchron-Conversation muss also folgenden Programmaufbau haben:

```
Accept_Conversation
........eventuell Extract_Partner_LU_Name
........eventuell Extract_TP_Name
........eventuell Extract_Sync_Level
Receive
........eventuell Convert_Incoming
........eventuell weitere Receive zum Empfangen von Daten
Receive zum Warten auf den return_code CM_DEALLOCATED_NORMAL
```

Zur Abfrage von Information können an beliebiger Stelle innerhalb der Conversation zusätzlich folgende CPI-C-Aufrufe aufgerufen werden:

– Extract_Conversation_State
– Extract_Maximum_Buffer_Size

Im Fehlerfall können folgende CPI-C-Aufrufe verwendet werden:

– Cancel_Conversation
– Set_Deallocate_Type mit *deallocate_type*=CM_DEALLOCATE_ABEND gefolgt von Deallocate

Läuft der Akzeptor nicht unter openUTM ab, dann erhält er zum Schluss keinen Returncode CM_DEALLOCATED_NORMAL, sondern im Parameter *status_received* den Wert CM_CONFIRM_DEALLOC_RECEIVED. Er muss danach mit Confirmed (CMCFMD) eine positive oder mit Send_Error (CMSERR) eine negative Quittung geben. Eine negative Quittung wird allerdings dem Initiator (d.h. dem CPI-C-Programm unter openUTM) nicht zugestellt, sondern führt dazu, dass der Auftrag wiederholt wird.

3.4 Erstellen einer CPI-C-Anwendung

CPI-C-Teilprogramme unter openUTM müssen immer Unterprogramme sein. openUTM übergibt dem CPI-C-Teilprogramm keine Parameter, d.h. die bei KDCS-Teilprogrammen unter openUTM üblichen Übergabeparameter KB, SPAB und eventuelle SPAB-Erweiterungen dürfen von CPI-C-Programmen nicht verwendet werden.

openUTM startet ein CPI-C-Programm dann, wenn für dieses Programm eine Incoming-Conversation von einem Partnerprogramm initiiert wird.

3.4.1 Übersetzen und Binden einer CPI-C-Anwendung unter Unix- und Windows-Systemen

X/W CPI-C-Teilprogramme können in beliebigen Programmiersprachen erstellt werden.

X/W X/W
- Für das Erstellen von CPI-C-Teilprogrammen in C wird mit openUTM eine Include-Datei ausgeliefert:

  X – *utmpfad*`/cpic/include/cpic.h` (Unix-Systeme)

  W – *utmpfad*`\cpic\include\cpic.h` (Windows-Systeme)

X/W
- Für COBOL wird ein COPY-Element ausgeliefert:

  X X – *utmpfad*`/cpic/copy-cobol85/CMCOBOL` bzw. *utmpfad*`/cpic/netcobol/CMCOBOL` (Unix-Systeme)

  W W – *utmpfad*`\cpic\copy-cobol85\CMCOBOL` bzw. *utmpfad*`\cpic\netcobol\CMCOBOL` (Windows-Systeme)

X X
- Zum Binden der CPI-C-Programme auf Unix-Systemen steht folgende XOPEN-Bibliothek zur Verfügung.

  X – *utmpfad*`/sys/libxopen`

X/W X/W
- Beim Binden einer CPI-C-Anwendung müssen folgende Bibliotheken dazugebunden werden:

  X/W – die Benutzerbibliothek mit den CPI-C-Teilprogrammen und -Modulen

  X/W – die UTM-Bibliothek

  X – die XOPEN-Bibliothek (Unix-Systeme)

X/W X/W X/W Wenn im Betrieb der Anwendung ein CPIC-Trace erzeugt werden soll, dann können Sie den Trace per Startparameter einschalten oder per Adminstration ein- und ausschalten (siehe Abschnitt „Steuerung des Trace“ auf Seite 99).

3.4.2 Übersetzen und Binden einer CPI-C-Anwendung unter BS2000-Systemen

CPI-C-Teilprogramme können in beliebigen Programmiersprachen erstellt werden.

Für das Erstellen von CPI-C-Teilprogrammen in C wird mit openUTM eine Include-Datei ausgeliefert, für COBOL ein Copy-Element.

Zum Binden der CPI-C-Programme steht die Bibliothek $*userid*.SYSLIB.UTM.063.XOPEN zur Verfügung.

Als Elemente vom Typ S finden Sie in dieser Bibliothek auch die Include-Datei für C (CPIC.H) und das Copy-Element für COBOL (CMCOBOL).

Beim Binden einer CPI-C-Anwendung müssen folgende Bibliotheken dazugebunden werden:

- die Benutzerbibliothek mit den CPI-C-Teilprogrammen und -Modulen
- die UTM-Bibliothek
- die XOPEN-Bibliothek $*userid*.SYSLIB.UTM.063.XOPEN
- die Bibliotheken SYSLNK.CRTE bzw. SYSLNK.CRTE.PARTIAL-BIND für Sprachumgebung und Laufzeitsystem

Wenn im Betrieb der Anwendung ein CPIC-Trace erzeugt werden soll, dann können Sie den Trace per Startparameter einschalten oder per Adminstration ein- und ausschalten (siehe Abschnitt „Steuerung des Trace“ auf Seite 99).

3.4.3 Generieren einer CPI-C-Anwendung

Bei der Generierung einer UTM-Anwendung mit CPI-C-Teilprogrammen müssen Sie die folgenden Besonderheiten beachten:

Lokale CPI-C-Teilprogramme und zugehörige Transaktionscodes generieren

Jedes Teilprogramm einer UTM-Anwendung muss mit einer PROGRAM-Anweisung definiert werden.

B Unter BS2000-Systemen muss als Compiler ILCS angegeben werden:

B
```
PROGRAM programmname,COMP=ILCS,...
```

Transaktionscodes, die in den CPI-C-Teilprogrammen realisiert sind, müssen mit

```
TAC  tacname, PROGRAM=programmname, API=(XOPEN,CPIC), ...
```

generiert werden (*programmname*=Name des CPI-C-Programms aus der PROGRAM-Anweisung).

Einem CPI-C-Teilprogramm können Sie mehrere Transaktionscodes zuordnen. Der Programmablauf kann dann anhand des Transaktionscodes gesteuert werden, mit dem das Programm gestartet wurde. Der Transaktionscode, mit dem das CPI-C-Programm gestartet wurde, ist in der Conversation-Charakteristik *TP_Name* enthalten.

PGWT-TAC-Klasse generieren

Soll ein CPI-C-Teilprogramm Dialog-Conversations unterhalten, bei denen das Senderecht durch den Aufruf Set_Send_Type mit *send_type*=CM_SEND_AND_PREP_TO_RECEIVE oder durch Aufruf von Receive im Zustand *Send* auf den Conversation-Partner übertragen wird, so muss in Anwendungen, die mit Auftragssteuerung durch Prozessbeschränkung arbeiten, der Transaktionscode dieses CPI-C-Teilprogramms einer TAC-Klasse zugeordnet werden, die mit PGWT=YES generiert ist, also z.B.:

```
MAX TASKS=2
MAX TASKS-IN-PGWT=1
TACCLASS 1,TASKS=1,PGWT=YES
TAC CPIC1,PROGRAM=xyz,API=(XOPEN,CPIC),TACCLASS=1
```

In Anwendungen, die mit Auftragssteuerung durch Prioritäten arbeiten, müssen Sie den Transaktionscode dieses CPI-C-Teilprogramms mit PGWT =YES generieren:

```
MAX TASKS=2
MAX TASKS-IN-PGWT=1
TAC CPIC1,PROGRAM=xyz,API=(XOPEN,CPIC), PGWT=YES
```

Angaben für Associations mit OSI-TP-Partnern und openUTM-Client-Partnern mit Trägersystem OpenCPIC

Die für den Aufbau von OSI-TP-Associations zwischen lokaler Anwendung und Partneranwendung benötigte Information müssen Sie in ACCESS-POINT-, OSI-LPAP- und OSI-CON-Anweisungen angeben, z.B. wie folgt:

```
* Name des lokalen Zugriffspunktes
ACCESS-POINT  cpicap,\
              P-SEL = NONE, S-SEL = NONE, T-SEL = C‘tsel‘,

* lokaler Name für fernen  Partner
OSI-LPAP    cpiclpap,\
            ASSOCIATION-NAMES = assoname,\
            ASSOCIATIONS = 2, CONTWIN = 1, CONNECT = 1,\
            APPLICATION-CONTEXT = UDTAC oder UDTDISAC

* lokaler Name für die Connection
OSI-CON     cpiccon,\
            P-SEL=NONE, S-SEL=NONE,\
            T-SEL=C'tsel', N-SEL=C'nsel',\
            LOCAL-ACCESS-POINT = cpicap,\
            OSI-LPAP = cpiclpap
```

Bei der Kommunikation mit Partnern an Fremdsystemen kann es sein, dass für lokale und ferne Anwendung zusätzlich der Application Entity Title definiert werden muss (siehe Handbuch openUTM „Anwendungen generieren und betreiben“)..

Angaben für Sessions mit LU6.1-Partnern

Die für den Aufbau von LU6.1 Sessions zwischen lokaler Anwendung (BCAMAPPL) und Partneranwendung benötigte Information müssen Sie in BCAMAPPL-, LPAP-, CON-, SESCHA- und LSES-Anweisungen angeben.

X/W *Hinweise zur Datenkonvertierung*

X/W Werden für die Datenkonvertierung die Aufrufe Convert_Incoming (CMCNVI) und
X/W Convert_Outgoing (CMCNVO) verwendet, dann darf bei der Generierung der Verbin-
X/W dungen zur Partneranwendung in der OSI-CON-Anweisung *nicht* MAP=SYSTEM
X/W angegeben werden.

Angaben für Verbindungen zu openUTM-Client-Partnern mit Trägersystem UPIC

Für jeden openUTM-Client-Partner mit Trägersystem UPIC müssen Sie eine PTERM- und eine LTERM-Anweisung angeben, z.B.:

```
PTERM UPICTTY, PTYPE=UPIC-R, LTERM=UPICLT, BCAMAPPL=CPICBA,  PRONAM=PGTRO175
LTERM UPICLT
```

Alternativ ist auch ein Anschluss über TPOOL möglich.

Falls ein openUTM-Client-Partner mit Trägersystem UPIC das Benutzerkonzept von openUTM nutzt, muss noch eine USER-Anweisung angegeben werden.

Definition der TP-Namen ferner CPI-C-Programme

Für jedes CPI-C-Programm einer fernen Anwendung, zu dem ein lokales CPI-C-Programm eine Outgoing-Conversation aufbauen will, müssen Sie eine LTAC-Anweisung angeben.

In der LTAC-Anweisung müssen Sie den Operanden LPAP= und ggf. weitere Operanden wie z.B. RTAC= oder TYPE= angeben. Der ltac-name der LTAC-Anweisung muss beim Initialize_Conversation-Aufruf im Parameter *sym_dest_name* angegeben werden.

Die oben genannten KDCDEF-Anweisungen sind openUTM-Handbuch „Anwendungen generieren“ beschrieben.

3.5 Fehlerdiagnose in CPI-C-Programmen

CPI-C-Aufrufe werden auf KDCS-Aufrufe abgebildet. Der UTM-Dump enthält deshalb auch bei CPI-C-Teilprogrammen nur die KDCS-Aufrufe. Sie können für die Diagnose von CPI-C-Programmen jedoch zusätzlich zum UTM-Dump einen Trace der CPI-C-Aufrufe erzeugen.

3.5.1 Steuerung des Trace

Der CPI-C-Trace kann wie folgt gesteuert werden:

- Durch den UTM-Startparameter CPIC-TRACE kann der Trace beim Start der Anwendung eingeschaltet werden, siehe jeweiliges openUTM-Handbuch „Einsatz von openUTM-Anwendungen“.
- Über WinAdmin oder WebAdmin kann der Trace im laufenden Betrieb ein- und ausgeschaltet werden. Dazu verwenden Sie den Dialog *Eigenschaften der UTM-Anwendung*, Registerkarte *Diagnose und Abrechnung*, Feld *CPIC Trace*.
- Über die Programmschnittstelle zur Administration KDCADMI kann der Trace im laufenden Betrieb ein- und ausgeschaltet werden. Dazu dient das Feld *cpic_trace* in der Datenstruktur *kc_diag_and_account_par_str*, siehe openUTM-Handbuch „Anwendungen administrieren“.

Sie können folgende Trace-Level einstellen:

Level	**Bedeutung**
TRACE	Zu jedem CPI-C-Funktionsaufruf wird der Inhalt der Input- und Output-Parameter ausgegeben. Von den Datenpuffern werden nur die ersten 16 Byte ausgegeben. Die Returncodes der KDCS-Aufrufe, auf die die CPI-C-Aufrufe abgebildet werden, werden ausgegeben.
BUFFER	Umfasst den Level TRACE, die Datenpuffer werden jedoch in voller Länge protokolliert.
DUMP	Umfasst den Level TRACE, zusätzlich werden Diagnose-Informationen in die Trace-Datei geschrieben.
ALL	Umfasst die Level BUFFER, DUMP und TRACE.
OFF	Trace ist ausgeschaltet.

3.5.2 Name der Trace-Datei

X/W **Trace-Datei unter Unix- und Windows-Systemen**

X/W Die Trace-Sätze werden in die Datei KDC.TRC.CPIC.*appliname.hostname.pid* im
X/W Verzeichnis *filebase* geschrieben. Dabei bedeuten:

X/W appliname
X/W Name der Anwendung

X/W hostname
X/W Name des Rechners, auf dem die Anwendung läuft

X/W pid PID des Prozesses

B **Trace-Datei unter BS2000-Systemen**

B Die Trace-Sätze werden standardmäßig in die Datei
B KDC.TRC.CPIC.*appliname.hostname.tsn* geschrieben. Dabei bedeuten:

B appliname
B Name der Anwendung

B hostname
B Name des Rechners, auf dem die Anwendung läuft

B tsn TSN der UTM-Task

B Sie können auch in der UTM-Startprozedur für jede Task eine andere Trace-Datei
B einrichten und mit dem Kommando SET-FILE-LINK den Linknamen KDCCPIC zuweisen.

3.5.3 Inhalt der Trace-Datei

Die CPI-C-Aufrufe werden wie folgt protokolliert:

--->IN funktionsname: Input-Parameter

Die Rückgaben der Funktionsaufrufe werden wie folgt protokolliert:

<---OUT funktionsname: return_code = returncode weitere Output-Parameter

returncode ist der Returncode des Aufrufs. Die weiteren Output-Parameter werden nur ausgegeben, wenn return_code = CM_OK ist.

Dazu werden die Rückmeldungen der abgesetzten KDCS-Aufrufe in die Trace-Datei geschrieben.

4 X/Open-Schnittstelle XATMI

XATMI ist eine von X/Open standardisierte Programmschnittstelle für einen Communication Resource Manager, der Client-Server-Kommunikation mit Transaktionssicherung ermöglicht.

Grundlage der XATMI-Programmschnittstelle ist die X/Open CAE Specification „Distributed Transaction Processing: The XATMI Specification“ vom November 1995. Die Kenntnis dieser Spezifikation wird im Folgenden vorausgesetzt.

Dieses Kapitel beschreibt die XATMI-Schnittstelle für Programme unter openUTM. XATMI-Programme unter openUTM sind immer Server.

Unterstützte Protokolle

Unter openUTM können XATMI-Anwendungen sowohl über OSI-TP als auch über das LU6.1-Protokoll kommunizieren.

Begriffe

In der folgenden Beschreibung werden folgende Begriffe verwendet:

Service	Eine Service-Funktion, die entsprechend der XATMI-Spezifikation in C oder COBOL programmiert ist. XATMI unterscheidet zwei Arten von Services: End-Services und Intermediate-Services. Ein End-Service ist nur mit seinem Client verbunden und ruft keine anderen Services auf. Ein Intermediate-Service ruft einen oder mehrere weitere Services auf.
Client	Eine Anwendung, die Service-Funktionen aufruft.
Server	Eine UTM-Anwendung, die Service-Funktionen in C und/oder in COBOL enthält. Die Service-Funktionen können aus mehreren Teilprogrammen bestehen.
Reques	Ein Request ist ein Aufruf an einen Service. Der Aufruf kann entweder von einem Client oder von einem Intermediate-Service aus erfolgen.

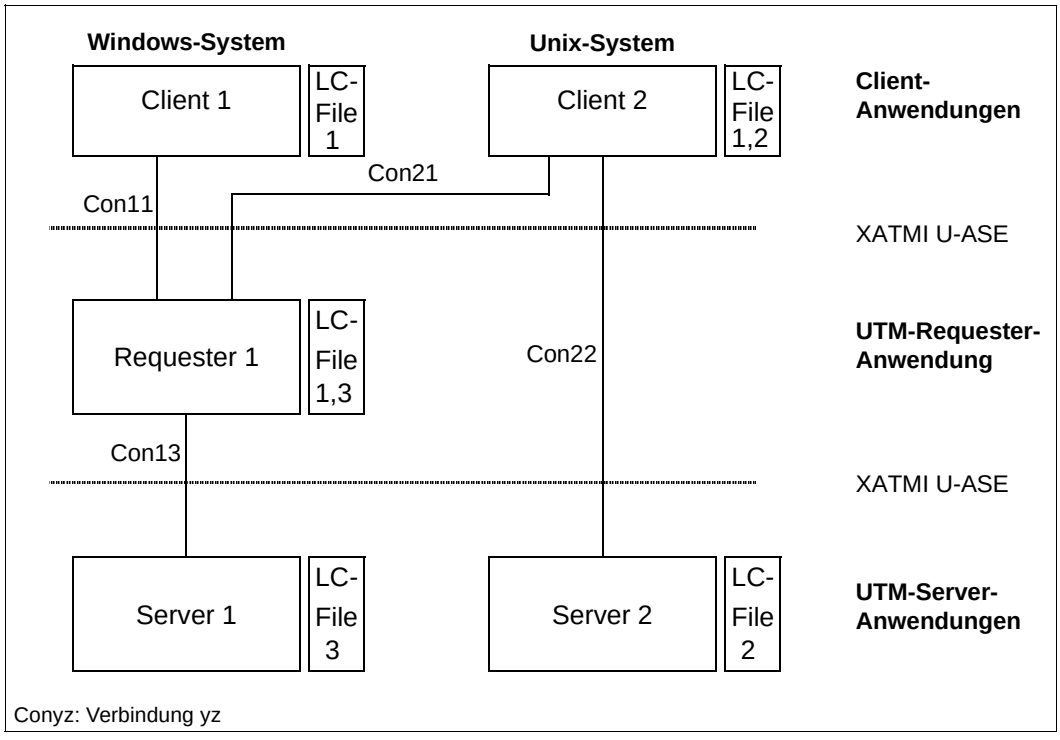
Requester Die XATMI-Specification verwendet den Begriff "Requester" als Bezeichnung für jegliche Anwendung, die einen Service aufruft. Ein Requester kann sowohl Client wie auch Server sein.

Typisierte Puffer
Puffer für den Austausch von typisierten und strukturierten Daten zwischen Kommunikationspartnern. Durch diese typisierten Puffer ist die Struktur der ausgetauschten Daten dem Trägersystem und der Anwendung implizit bekannt und werden auch im heterogenen Verbund automatisch angepasst (encodiert, decodiert).

4.1 Client-Server-Verbund

Das folgende Bild zeigt einen Client-Server Anwendungsverbund, bei dem Clients, Server und Requester miteinander kommunizieren. Clients können auch auch BS2000-Systemen residieren.

Die einzelnen Anwendungen enthalten XATMI-Aufrufe und tauschen typisierte Datenstrukturen (**Typisierte Puffer**) nach dem Protokoll der "XATMI U-ASE Definition" aus.



In einem beliebigen, heterogenen Anwendungsverbund muss sowohl den Servern wie auch den Clients eine Local Configuration beigestellt sein, die jeweils in der Local Configuration File (LCF) definiert ist. Die Local Configuration beschreibt jeweils die Services und ihre zugehörigen Datenstrukturen, d.h.:

– bei einem Server alle aufrufbaren Services
– bei einem Client die Services aller Server, mit denen der Client in Verbindung steht.
– bei einem Requester (Intermediate-Service) sowohl alle bereitgestellten Services als auch alle benutzten Services.

Die Local Configurations aller beteiligten Anwendungen müssen aufeinander abgestimmt sein.

Für die Client-Server-Kommunikation stehen mehrere Kommunikationsmodelle zur Verfügung (siehe Abschnitt „Kommunikationsmodelle“ auf Seite 106.

4.1.1 Default-Server

Zur Vereinfachung der Client-Server-Konfiguration bietet Ihnen openUTM die Möglichkeit, mit der Angabe DEST=.DEFAULT in der SVCU-Anweisung der Local Configuration File einen Default-Server zu vereinbaren (siehe Seite 121).

Falls bei den Aufrufen `tpcall`, `tpacall` oder `tpconnect` ein Service *svcname2* verwendet wird, der keinen SVCU-Eintrag in der Local Configuration File besitzt, wird automatisch folgender Eintrag verwendet:

```
SVCU svcname2, RSN=svcname2, TAC=scvname2, DEST=.DEFAULT, MODE=RR
```

openUTM erwartet dann in der KDCDEF-Input-File einen Default-Server-Eintrag, z.B.

```
LTAC svcname2, ... , LPAP=BRANCH9
```

Der Partner, in diesem Fall BRANCH9, muss openUTM natürlich weiterhin bekannt sein.

4.2 Kommunikationsmodelle

Für die Client-Server-Kommunikation stehen dem Programmierer drei Kommunikationsmodelle zur Auswahl:

– Synchrones Request-Response Modell: Einschritt- Dialog.
Der Client ist nach dem Senden der Service-Anforderung bis zum Eintreffen der Antwort blockiert.

– Asynchrones Request- Response Modell: Einschritt-Dialog.
Der Client ist nach dem Senden der Service-Anforderung nicht blockiert.

Spezialfall:
Single Request Modell: Kommunikation nur in eine Richtung.
Der Client fordert einen Service an, erwartet aber keine Antwort.

– Conversational Modell: Mehrschritt-Dialog.
Client und Server können beliebig Daten austauschen.

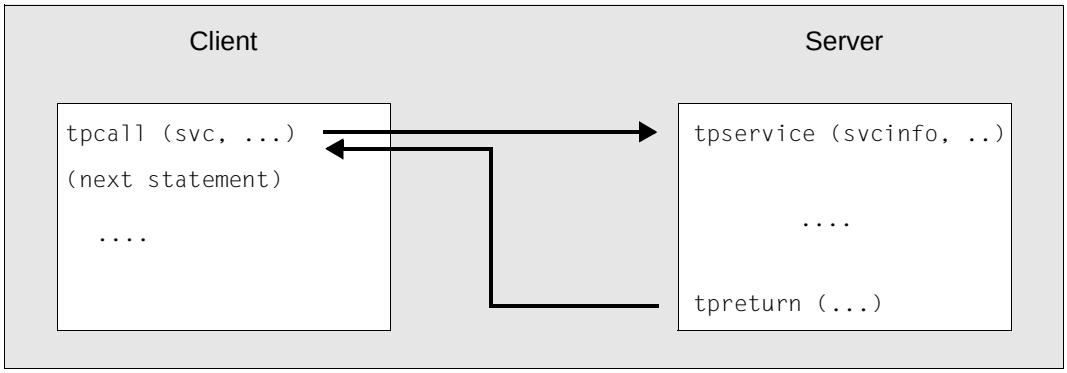
Die für diese Kommunikationsmodelle notwendigen XATMI-Funktionen werden im Folgenden nur skizziert, dabei wird die C-Notation verwendet. Die genaue Beschreibung der XATMI-Funktionen finden Sie in der X/Open-Spezifikation „Distributed Transaction Processing: The XATMI Specification“.

Je nach Modell wird die Kommunikation durch einen der XATMI-Aufrufe *tpcall*(), *tpacall*() und *tpconnect*() eröffnet.

Bei jedem XATMI-Aufruf *tpcall*(), *tpacall*() und *tpconnect*() in der UTM-Anwendung adressiert UTM einen Vorgang (KDCS-Aufruf APRO) und belegt für die Dauer der Kommunikation eine Session bzw. Association. Es müssen so viele Sessions bzw. Associations generiert sein wie maximal gleichzeitig Vorgänge adressiert sind.

Synchrones Request-Response Modell

Für die Kommunikation mit einem solchen Service wird nur ein einziger *tpcall()*-Aufruf benötigt.
Der *tpcall()* -Aufruf addressiert den Service, schickt genau eine Nachricht an diesen ab und wartet solange, bis ihn die Antwort erreicht, d.h. *tpcall()* wirkt blockierend.



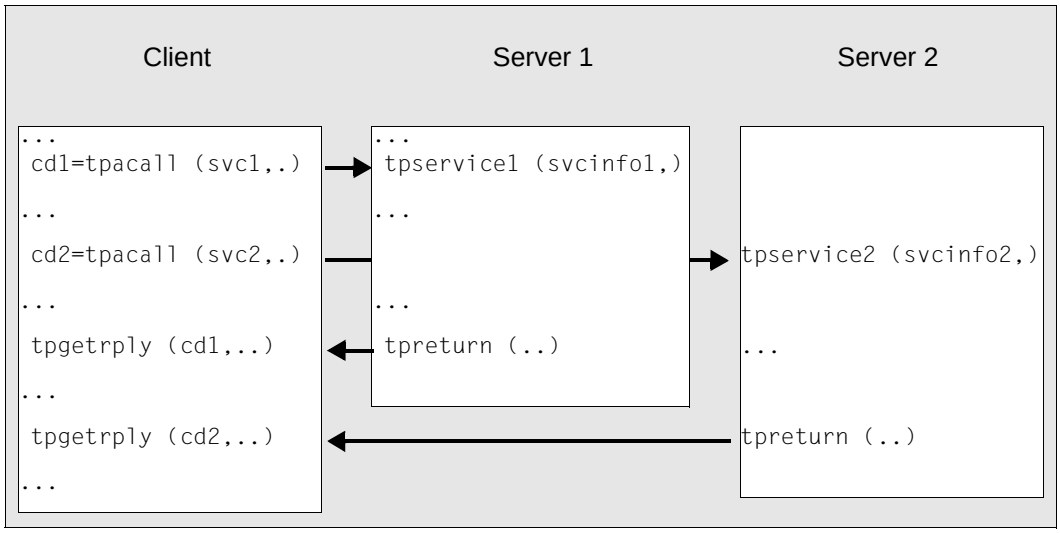
In diesem Bild bezeichnet *svc* den intern verwendeten Namen des Services, *svcinfo* die Service-Info-Struktur mit dem Service-Namen und *tpservice* den Programmnamen der Service-Routine. Für die Programmiersprache C steht dafür das Template *tpservice()* zur Verfügung. Bei COBOL muss der XATMI-Aufruf TPSVCSTART verwendet werden, siehe Seite 116. Die Service-Info-Struktur ist in der XATMI-Schnittstelle definiert und wird intern durch XATMI versorgt.

Bei diesem Modell muss auf der openUTM-Server-Seite ein Dialog-TAC für den angeforderten Service generiert sein.

Asynchrones Request-Response Modell

Bei diesem Modell wird die Kommunikation in zwei Schritten abgewickelt. Im ersten Schritt wird der Service mit dem Aufruf *tpacall()* addressiert und die Nachricht abgeschickt. Zu einem späteren Zeitpunkt wird im zweiten Schritt mit dem Aufruf *tpgetrply()* die Antwort abgeholt. Der Aufruf *tpacall ()* wirkt nicht blockierend, d.h. der Client kann nach dem Aufruf weitere lokale Verarbeitungen durchführen. Es sind auch weitere (bis zu 64) Verbindungen möglich.

Bei diesem Modell können parallel mehrere Services beauftragt werden, siehe folgendes Bild.



In dem Bild bezeichnen *svc*1, *svc2* die intern verwendeten Namen der Services , *cd1*, *cd2* die Prozesslokalen Communication Descriptoren, *svcinfo1*, *svcinfo2* die Service-Info-Strukturen mit den Service-Namen und *tpservice1, tpservice2* die Programmnamen der Service-Routinen.

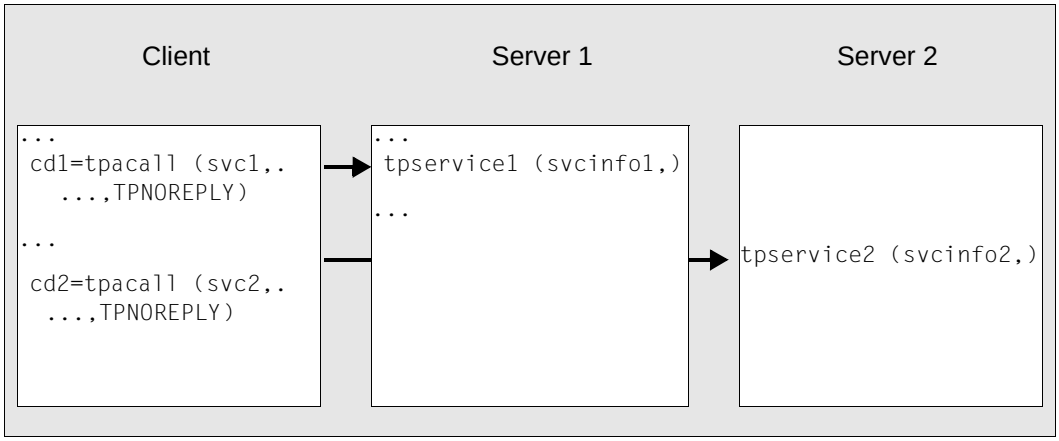
Im Gegensatz zu *tpacall()* wirkt der Aufruf *tpgetrply()* blockierend, d.h. der Client wartet solange, bis die Antwort eingetroffen ist.

Bei diesem Modell muss auf der openUTM-Server-Seite ein Dialog-TAC für den angeforderten Service generiert sein (wie beim synchronen Request-Response).

Ein Spezialfall dieses Modells ist der **Single Request** oder auch **Request with no Response**.

Single Request Modell

Bei diesem Modell wird mit *tpacall()* ein Service angefordert, es wird jedoch keine Antwort erwartet (signalisiert durch das Flag TPNOREPLY).



In dem Bild bezeichnen *svc*1, *svc2* die intern verwendeten Namen der Services , *cd1*, *cd2* die Prozesslokalen Communication Descriptoren, *svcinfo1*, *svcinfo2* die Service-Info-Strukturen mit den Service-Namen und *tpservice1, tpservice2* die Programmnamen der Service-Routinen.

Bei diesem Modell muss auf der openUTM-Server-Seite ein Asynchron-TAC für den angeforderten Service generiert sein.

Dieses Modell kann nur verwendet werden, wenn der Client eine UTM-Requester-Anwendung ist.

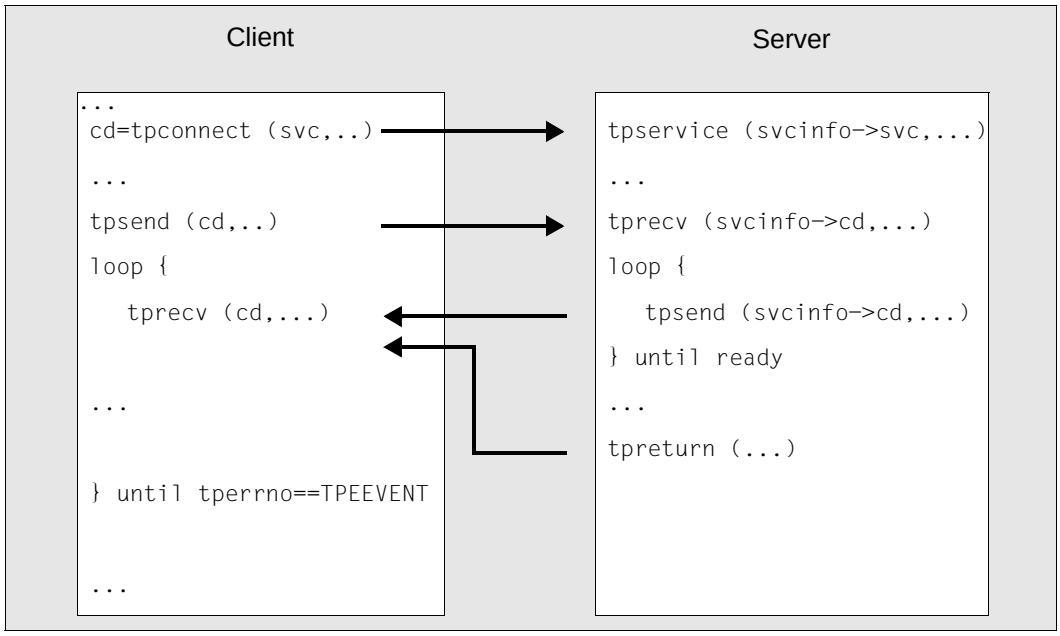
Conversational Modell

Für verbindungsorientiertes Arbeiten ("Conversation") bietet XATMI das Conversational Modell an.

Dieses Modell kann z.B. verwendet werden, um große Datenmengen in mehreren Teilschritten zu übertragen. Damit können Probleme vermieden werden, die beim synchronen Request-Respone Modell (Aufruf *tpcall()*) wegen der Größenbegrenzung der lokalen Datenpuffer auftreten könnten.

Beim Conversational Modell wird die Conversation zu einem Service explizit mit dem Aufruf *tpconnect()* aufgebaut. Solange sie besteht, können Client und Server mit *tpsend()* und *tprecv()* Daten austauschen. Innerhalb eines Dialogs kann nur eine Transaktion abgewickelt werden.

Beendet wird die Conversation, wenn der Server mit *tpreturn()* das Ende signalisiert; der Client erhält dann beim *tprecv()* in der Variable *tperrno* einen entsprechenden Code. Daher muss das Client-Programm mindestens einen *tprecv()*-Aufruf enthalten.



In dem Bild bezeichnet *svc* den lokalen Namen des Services, *cd* den Prozesslokalen Communication Descriptor, *tpservice* den Programmnamen der Service-Routine und *svcinfo* die Service-Info-Struktur mit dem Service-Namen und dem Communication Descriptor.

Bei diesem Modell muss auf der openUTM-Server-Seite ein Dialog-TAC für den angeforderten Service generiert sein.

In Fehlerfällen kann der Client den Abbruch einer Conversation mit dem Aufruf *tpdiscon()* erzwingen.

4.3 Typisierte Puffer

XATMI-Anwendungen tauschen Nachrichten mit Hilfe von "typisierten Datenpuffern" aus. Dadurch werden die über das Netz gehenden Daten korrekt zwischen den Anwendungen ausgetauscht, d.h. gemäß der über den Puffernamen identifizierten Datenstruktur mit ihren Datentypen.

Dies hat den Vorteil, dass die Anwendungen keine Maschinenabhängigkeiten berücksichtigen müssen wie z.B. Big Endian/Little Endian Darstellungen, ASCII/EBCDIC-Konvertierungen oder Ausrichtungen auf Wortgrenzen. Damit können Datentypen wie int, long, float usw. als solche übertragen werden.
Eine eventuell notwendige Kodierung/Dekodierung durch die Anwendungsprogramme entfällt, da diese von XATMI übernommen wird (gemäß den Regeln der XATMI U-ASE Definition).

Ein Datenpuffer-Objekt besteht aus vier Komponenten:

– Typ: Definiert den Typ des Puffers. Es gibt drei Typen (siehe unten).
– Subtyp: Definiert das Objekt der Klasse, d.h. die eigentliche Datenstruktur.
– Längenangabe
– Dateninhalt

Ein solcher Datenpuffer wird während der Laufzeit erzeugt und kann dann über seinen Variablen-Namen (=Subtyp-Namen) angesprochen werden. In C-Programmen werden solche Puffer dynamisch mit *tpallcoc()* erzeugt, man spricht dann von "typisierten Puffern", in COBOL-Programmen sind diese Puffer statisch festgelegt, man spricht von "typisierten Records".

Typen

Mit dem Typ eines Datenpuffers wird festgelegt, welche elementaren Datentypen der verwendeten Programmiersprache erlaubt sind. Dadurch wird ein gemeinsames Datenverständnis in einem heterogenen Client-Server-Verbund ermöglicht.

Bei XATMI sind drei Typen definiert:

X_OCTET	Untypisierter Datenstrom von Bytes ("Userbuffer"). Dieser Typ besitzt keine Subtypen. Es wird keine Konvertierung vorgenommen
X_COMMON	Alle von C und COBOL gemeinsam verwendbaren Datentypen. Die Konvertierung wird von XATMI vorgenommen.
X_C_TYPE	Alle elementaren C-Datentypen mit Ausnahme von Zeigern. Die Konvertierung wird von XATMI vorgenommen.

Subtypen

Subtypen haben einen bis zu 16 Zeichen langen Namen, unter dem sie im Anwendungsprogramm angesprochen werden. Jedem Subtyp ist eine Datenstruktur (C-Structure oder COBOL-Record) zugeordnet, die die Syntax des Subtyps bestimmt, siehe Seite 118.
Die Datenstrukturen dürfen nicht geschachtelt werden.

In der Local Configuration wird die Struktur eines Subtyps durch einen Syntaxstring repräsentiert, in dem jeder elementare Datentyp (Basistyp) durch einen Code gekennzeichnet ist, der im Bedarfsfall die Angabe von Feldlängen (<m> und <n>) enthält.
Die folgende Tabelle gibt einen Überblick über die Basistypen, deren Codes und den Zeichenvorrat der String-Typen.

Code[1]	**Bedeutung**	**ASN.1-Typ**	**X_C_TYPE**	**X_COMMON**
s	short integer	INTEGER	short	S9(4) COMP-5
S<n>	short integer array	SEQUENCE OF INTEGER	short[n]	S9(4) COMP-5 ...
i	integer	INTEGER	integer	--[2]
I<n>	integer array	SEQUENCE OF INTEGER	integer[n]	--
l	long integer	INTEGER	long	S9(9) COMP-5
L<n>	long integer array	SEQUENCE OF INTEGER	long[n]	S9(9) COMP-5 ...
f	float	REAL	float	--
F<n>	float array	SEQUENCE OF REAL	float[n]	--
d	double	REAL	double	--
D<n>	double array	SEQUENCE OF REAL	double[n]	--
c	character	OCTET STRING	char	PIC X
t	character	T.61-String	char	PIC X
C<n>	character array: Alle Werte von 0 bis 255 (dezimal)	OCTET STRING	char[n]	PIC X(n)
C!<n>	character array, durch Null ('\0') terminiert	OCTET STRING	char[n]	--
C<m>:<n>	character matrix[3]	SEQUENCE OF OCTET STRING	char [m][n]	--
C!<m>:<n>	character matrix, durch Null ('\0') terminiert	SEQUENCE OF OCTET STRING	char [m][n]	--
T<n>	Die abdruckbaren Zeichen A-Z, a-z und 0-9 plus[4] eine Reihe von Sonderzeichen und Steuerzeichen, siehe Seite 147.	T.61-String	t61str[n]	PIC X(n)

Code[1]	Bedeutung	ASN.1-Typ	X_C_TYPE	X_COMMON
T!<n>	character array, durch Null ('\0') terminiert	T.61-String	t61str[n]	--
T<m>:<n>	character matrix	SEQUENCE OF T.61-String	t61str[m][n]	--
T!<m>:<n>	character matrix, durch Null ('\0') terminiert	SEQUENCE OF T.61-String	t61str[m][n]	--

[1] Dient in der Local Configuration zur Beschreibung der Datenstrukturen

[2] -- : in X_COMMON nicht vorhanden

[3] eine character matrix ist ein zweidimensionales character array

[4] gemäß CCITT Recommodation T.61 bzw. ISO 6937

Die Zuordnung zwischen Datenstrukturen, Subtypen und gewünschten Services wird in der Local Configuration festgelegt, siehe Seite 121.

Zeichensatz-Konvertierung bei X_C_TYPE und X_COMMON

Die Datenpuffer werden im ASCII-Zeichensatz über das Netz geschickt.

Ein Partner kann jedoch eine andere Zeichensatzcodierung als ASCII verwenden, wie z.B. eine BS2000-Anwendung, die EBCDIC verwendet. In diesem Fall konvertiert die BS2000-XATMI-Bibliothek bei allen eingehenden und abgehenden Daten den ASN.1-Typ *T.61-String* von ASCII nach EBCDIC und umgekehrt. Daher darf für das jeweilige Trägersystem (openUTM/UPIC/OpenCPIC) keine automatische Konvertierung generiert werden.

Insbesondere darf bei den KDCDEF-Anweisungen OSI-CON und SESCHA nicht MAP=SYSTEM gesetzt werden.

Der Datentyp OCTET STRING wird nicht konvertiert.

4.4 Programmschnittstelle

Die folgenden Abschnitte geben einen Überblick über die XATMI-Programmschnittstelle für Server und Requester. Eine detaillierte Beschreibung der Programmschnittstelle und der Error- und Returncodes finden Sie in der X/Open-Spezifikation „Distributed Transaction Processing: The XATMI Specification“. Die Kenntnis dieser Spezifikation ist für die Erstellung von XATMI-Programmen unbedingt erforderlich.

Die Programm-Schnittstelle steht sowohl in C als auch in COBOL zur Verfügung.

4.4.1 XATMI-Funktionen

Die folgenden Tabellen listen alle unter openUTM erlaubten XATMI-Aufrufe auf und beschreiben, in welcher Rolle (C = Client oder S = Server) sie aufgerufen und bei welchem Kommunikationsmodell sie verwendet werden dürfen.

Aufrufe für das Request/Response-Modell

C-Aufruf	COBOL-Aufruf	Aufruf im Client/ Server	Beschreibung
tpcall	TPCALL	C	Service-Anforderung im synchronen Request/Response-Modell
tpacall	TPACALL	C	Service-Anforderung im asynchronen Request/Response-Modell bzw. Single Request-Modell (Flag TPNOREPLY gesetzt)
tpgetreply	TPGETRPLY	C	Response im asynchronen Request/Response-Modell anfordern
tpcancel	TPCANCEL	C	löscht eine asynchrone Service-Anforderung, bevor die angeforderte Response eingetroffen ist

Aufrufe für das Conversational-Model

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

Aufrufe für typisierte Puffer

C-Aufruf	COBOL-Aufruf	Aufruf im Client/ Server	Beschreibung
tpalloc	--	C, S	reserviert Speicherplatz für einen typisierten Puffer
tprealloc	--	C, S	verändert die Größe eines typisierten Puffers
tpfree	--	C, S	gibt einen typisierten Puffer frei
tptypes	--	C, S	ermittelt den Typ eines typisierten Puffers.

Allgemeine Aufrufe für Service-Routinen

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

4.4.2 Besonderheiten bei XATMI-Aufrufen

Im folgenden Abschnitt sind Besonderheiten von XATMI-Aufrufen unter openUTM aufgeführt.

tpcancel

Bei Kommunikation über das Transportprotokoll LU6.1
führt der Aufruf *tpcancel*() auf einen vorhandenen `call descriptor` im Gegensatz zur X/Open CAE Specification für XATMI nicht nur zum Löschen dieses Descriptors, sondern zu einer Vorgangsbeendigung, d.h. zu einem Rücksetzen aller Verbindungen beim Aufruf von *tpreturn*(). Die XATMI-Aufrufe werden unter openUTM abgebildet auf den generierungsabhängigen RSET und PEND FR .

Bei Kommunikation über dasTransportprotokoll OSI-TP
löscht *tpcancel*() den gewünschten Descriptor. Der XATMI-Aufruf wird unter openUTM abgebildet auf CTRLAB.

tpdiscon

Bei Kommunikation über das Transportprotokoll LU6.1 (mit oder ohne Commit)
führt der Aufruf *tpdiscon*() auf einen vorhandenen `call descriptor` im Gegensatz zur X/Open CAE Specification für XATMI nicht nur zum Löschen dieses Descriptors, sondern zu einer Vorgangsbeendigung, d.h. zu einem Rücksetzen aller Verbindungen beim Aufruf von *tpreturn*(). Die XATMI-Aufrufe werden unter openUTM abgebildet auf den generierungsabhängigen RSET und PEND FR .

Bei Kommunikation über dasTransportprotokoll OSI-TP ohne Commit
wirkt *tpdiscon*() wie bei Kommunikation über LU6.1.

Bei Kommunikation über dasTransportprotokoll OSI-TP mit Commit
werden auf der UTM-Serverseite bei *tpreturn*() die Dialog-TACs KDCTXCOM bzw. KDCTXRLB mit PEND KP aufgerufen. Im Falle eines Single Request Services (*tpacall* TPNOREPLY) wird PEND FR bzw. PEND FI aufgerufen.

4.4.3 Datenübergabe an die Service-Funktion

Eine Service-Funktion ist ein Teilprogramm, das wie bei openUTM üblich durch einen Transaktionscode gestartet wird.

Dieser Transaktionscode muss mit TAC... API=(XOPEN,XATMI) generiert sein und darf nur von einem XATMI-Client oder -Requester, nicht aber von einem Terminal aus aufgerufen werden.

Die Informationen des Client/Requester werden an den Service über die Service-Info-Struktur TPSVCINFO übergeben. Diese Struktur ist Bestandteil der XATMI-Schnittstelle. Die Übergabe an den Service ist in C und COBOL unterschiedlich:

- Bei C enthält die Service-Funktion als einzigen Parameter einen Zeiger auf die Service-Info-Struktur. Diese Struktur ist sofort nach dem Start der Service-Funktion versorgt. In der XATMI-Specification von X/Open ist das Template *tpservice()* für die Service-Funktion beschrieben.

  TPSVCINFO besitzt folgende Struktur:

```
typedef struct {
    char name[XATMI_SERVICE_NAME_LENGTH]; /* Service name */
    char *data;                  /* Pointer auf Datenbereich */
    long len;                    /* Länge des übertragenen Datenpuffers */
    long flags;                  /* Flags zur Aufrufsteuerung */
    int cd;                      /* Conversation Descriptor bei
                                    Conversational Modus, sonst
                                    undefiniert */
} TPSVCINFO
```

Bei COBOL wird die Struktur durch den XATMI-Aufruf TPSVCSTART versorgt. Dieser Aufruf muss der erste XATMI-Aufruf einer Service-Routine sein. Näheres finden Sie in der X/Open-Spezifikation „Distributed Transaction Processing: The XATMI Specification“ bei der Beschreibung von TPSVCSTART.

4.4.4 Ereignisse und Fehlerbehandlung

Wenn ein Ereignis eingetroffen oder ein Fehler aufgetreten ist, geben XATMI-Funktionen den Returnwert -1 zurück. Zur genaueren Bestimmung von Ereignis oder Fehler muss das Programm die Variable *tperrno* auswerten.

Bei der Conversational-Funktion *tprecv* zeigt *tperrno=TPEEVENT* an, dass ein Ereignis eingetroffen ist. Dieses Ereignis kann durch Auswerten des *tprevc*-Parameters *revent* bestimmt werden. Z.B. wird ein erfolgreiches Beenden eines Conversational Service wie folgt angezeigt:

```
Returncode von tprecv =-1
tperrno=TPEEVENT
revent=TPEV_SVCSUCC
```

Bei der Funktion *tpsend* hat der Parameter *revent* keine Bedeutung.

Außerdem kann das Service-Programm beim Ende der Service-Funktion mit *tpreturn* im Parameter *rcode* einen frei definierten Fehlercode zurückgeben, der clientseitig über die externe Variable *tpurcode* ausgewertet werden kann, siehe „Distributed Transaction Processing: The XATMI Specification“.

4.4.5 Typisierte Puffer erstellen

Typisierte Puffer werden definiert durch Datenstrukturen in Include-Dateien (bei C) bzw. COPY-Elementen (bei COBOL), die in den beteiligten Programmen verwendet werden müssen.

Der Datenaustausch zwischen den Programmen erfolgt auf Basis dieser Datenstrukturen, die daher sowohl dem Client als auch dem Server bekannt sein müssen. Dabei sind alle Datentypen erlaubt, die in der Tabelle auf Seite 112 bzw. in der X/Open-Spezifikation „Distributed Transaction Processing: The XATMI Specification“ beschrieben sind.

Die Include- bzw. COBOL-COPY-Dateien, in denen die typisierten Puffer beschrieben sind, dienen als Eingabe für das Generierungsprogramm `xatmigen`, näheres siehe Seite 127.

Für diese Dateien gelten folgende Regeln:

– C- und COBOL-Datenstrukturen müssen in getrennten Dateien stehen. Eine Datei, die sowohl C-Includes als auch COBOL-COPY-Elemente enthält, ist als Eingabe nicht erlaubt.

– Die Dateien dürfen nur die Definitionen der Datenstrukturen, Leerzeilen und Kommentaranweisungen enthalten.
Include-Dateien (für C) dürfen auch Makroanweisungen enthalten, d.h. Anweisungen, die mit '#' beginnen.

– Die Definitionen der Datenstrukturen müssen vollständig angegeben werden. Insbesondere müssen COBOL-Datensätze mit der Stufennummer "01" beginnen.

– Die Datenstrukturen dürfen nicht verschachtelt sein.

– Als Feldlängen sind nur absolute Werte und keine Makro-Konstanten erlaubt.

– Es sind nur die Datentypen erlaubt, die in der Tabelle auf Seite 112 beschrieben sind. Insbesondere sind bei C keine Zeiger-Typen zugelassen.

Mit Hilfe des Generierungstools `xatmigen` muss der Anwender ggf. die Character-Arrays auf die ASN.1-Stringtypen abbilden, da weder C noch COBOL diese Datentypen kennen, siehe Abschnitt „Das Dienstprogramm xatmigen“ auf Seite 127.

Für C stehen XATMI-Aufrufe für die Speicherbelegung zur Verfügung (*tpalloc* ...).

Im folgenden je ein einfaches Beispiel für C und COBOL.

Beispiel

1. C-Include für typisierten Puffer (Typed Buffer)

```
typedef struct {
        char   name[20];    /* Personenname */
        int    age;         /* Alter        */
        char   sex;
        long   shoesize;
} t_person;

struct t_city {
        char   name[32];             /* Staedtename */
        char   country;
        long   inhabitants;
        short  churches[20];
        long   founded;
}
```

2. COBOL-COPY für Typed Record

```
***** Personal-Record
 01 PERSON-REC.
    05 NAME     PICTURE X(20).
    05 AGE      PIC     S9(9) COMP-5.
    05 SEX      PIC     X.
    05 SHOESIZE PIC     S9(9) COMP-5.

***** City-Record
 01 CITY-REC.
    05 NAME        PIC   X(32).
    05 COUNTRY     PIC   X.
    05 INHABITANTS PIC   S9(9) COMP-5.
    05 CHURCHES    PIC   S9(4) COMP-5 OCCURS 20 TIMES.
    05 FOUNDED     PIC   S9(9) COMP-5.
```

Weitere Beispiele finden Sie in der X/Open-Spezifikation zu XATMI.

4.4.6 Charakteristika von XATMI in openUTM

Dieser Abschnitt beschreibt Besonderheiten, die durch die Implementierung der XATMI-Schnittstelle in openUTM auftreten.

- Es werden alle XATMI-Aufrufe unterstützt. Die beiden Aufrufe *tpadvertise()* und *tpunadvertise()* werden jedoch nur syntaktisch unterstützt, da sie task- bzw. Prozessbezogen sind und diese Aufrufe in der BS2000-Mehrtaskumgebung bzw. der Workprozessumgebung auf Unix- und Windows-Systemen von openUTM wirkungslos bleiben.
- Ein Intermediate-Service kann parallel bis zu 64 Services über das Asynchrone Request-Response Modell oder das Conversational Modell rufen.
- Es dürfen pro Service maximal 100 Pufferinstanzen gleichzeitig verwendet werden. Bei einer Service-Funktion in C heißt das z.B. maximal 100 *tpalloc()*-Aufrufe ohne *tpfree()*-Aufruf.
- Die maximale Nachrichtenlänge ist 32000 bytes. Sie wird jedoch auch durch folgende Parameter der UTM-Generierung begrenzt:

  MAX NB: maximale Länge für logische Nachrichten

  MAX TRMSGLTH: maximale Länge der physischen Nachrichten

  Die maximale Größe eines typisierten Puffers ist immer kleiner als die maximal mögliche Nachrichtenlänge, da die Nachrichten neben den Nettodaten noch einen "Overhead" enthalten. Je komplexer ein Puffer ist, desto größer ist der Overhead.

  Als Faustregel gilt: maximale Puffergröße = 2/3 der maximalen Nachrichtenlänge

  Bei größeren Datenmengen sollte daher immer das Conversational Modell (tpsend/tprecv) verwendet werden.
- Für die Namenslängen gelten folgende Grenzwerte:

  Servicename: 16 bytes

  Puffername: 16 bytes

  Nach dem Standard dürfen Servicenamen 32 Bytes lang sein (Konstante XATMI_SERVICE_NAME_LENGTH), von denen allerdings nur die ersten 16 Bytes relevant sind. Es empfiehlt sich daher, für Servicenamen nicht mehr als 16 Bytes zu verwenden.
- Prozesswechsel (Unix- und Windows-Systeme) bzw. Taskwechsel (BS2000-Systeme): Ein Service oder Request ist an einen Prozess bzw. eine Task gebunden, d.h. innerhalb eines Services oder Requests findet kein Prozess- bzw. Taskwechsel statt. Deshalb müssen Requests und Conversational Services innerhalb einer PGWT-TAC-Klasse ablaufen (siehe Abschnitt „TACs mit PGWT=YES“ auf Seite 130).

4.5 Konfigurieren

Für jede XATMI-Anwendung muss der Anwender eine Local Configuration erzeugen. Diese beschreibt die bereitgestellten und genutzten Services mit ihren Zieladressen sowie die verwendeten typisierten Puffer mit ihrer Syntax. Die Information ist in einer Datei hinterlegt, dem Local Configuration File (LCF), das von der Anwendung beim Starten einmal gelesen wird. Ein LCF ist sowohl für die Client- wie auch für die Serviceseite notwendig.

4.5.1 Local Configuration File erzeugen

Als Anwender müssen Sie eine Eingabe-Datei erstellen, genannt Local Configuration Definition File. Diese Eingabe-Datei muss aus einzelnen Zeilen aufgebaut werden, für die folgende Syntax gilt:

– Eine Zeile beginnt mit einer SVCU-, SVCP-, REQP- oder BUFFER-Anweisung. Mit diesen Anweisungen wird folgendes spezifiziert:

  SVCU: Genutzter Service
  SVCP: Angebotener Service
  REQP: Angebotener Request
  BUFFER: Subtyp (=typisierter Puffer)

– Zwei Operanden werden durch ein Komma getrennt.

– Eine Anweisungs-Zeile wird durch ein Semicolon (';') abgeschlossen.

– Nehmen die Operanden mehr als eine Zeile ein, dann muss jeweils am Zeilenende das Fortsetzungszeichen '\' (Gegenschrägstrich) stehen.

– Eine Kommentarzeile beginnt mit dem '#'-Zeichen in Spalte 1.

– Leerzeilen können eingefügt werden, z.B. zur besseren Lesbarkeit.

Aus der Datei, die die Local Configuration Definition enthält, erstellen Sie mit Hilfe des Tools `xatmigen` das eigentliche Local Configuration File (Seite 127).

Im folgenden werden die vier Anweisungen beschrieben.

SVCU-Anweisung: Aufrufbaren Service definieren

Eine SVCU-Anweisung beschreibt für den Client/Requester die Eigenschaften, die notwendig sind, um einen Service in der Partner-Anwendung aufrufen zu können. Sofern kein Default-Server verwendet wird, muss für jeden genutzten Service eine SVCU-Anweisung gegeben werden.

Die SVCU-Anweisung kann entfallen, wenn der XATMI-Service unter openUTM abläuft und in der UTM-Generierung die einstufige Adressierung verwendet wird mit *transaction-code* = *remote-service-name* = *internal-service-name*.
und für die restlichen Parameterwerte die Default-Einstellungen genügen.

Default-Server:

Zur Vereinfachung der Client-Server-Konfiguration bietet Ihnen openUTM die Möglichkeit, mit der Angabe DEST=.DEFAULT in der SVCU-Anweisung der Local Configuration File einen Default-Server zu vereinbaren.

Falls bei den Aufrufen `tpcall`, `tpacall` oder `tpconnect` ein Service *svcname2* verwendet wird, der keinen SVCU-Eintrag in der Local Configuration File besitzt, wird automatisch folgender Eintrag verwendet:

```
SVCU svcname2, RSN=svcname2, TAC=scvname2, DEST=.DEFAULT, MODE=RR
```

openUTM erwartet dann in der KDCDEF-Input-File einen passenden Default-Server-Eintrag, z.B.

```
LTAC svcname2, ... , LPAP=BRANCH9
```

Der Partner, in diesem Fall BRANCH9, muss openUTM natürlich weiterhin bekannt sein.

Operator	Operanden	Erläuterung
SVCU	internal-service-name	maximal 16 Byte
	[,RSN=remote-service-name]	Standard: internal-service-name
	[,TAC=transaktion-code]	Standard: internal-service-name
	,DEST={ destination-name | .DEFAULT }	Name der Partneranwendung
	[,MODE=RR / RN / CV]	RR=Request/Response (Standard) RN=Request no Response CV=Conversation
	[,BUFFERS=(subtype-1,...,subtype-n)]	Standard: kein Subtyp

internal-service-name

maximal 16 Byte langer Name, unter dem ein (ferner) Service in der lokalen Anwendung angesprochen wird. Dieser Name muss innerhalb der Anwendung eindeutig sein.

Mehrfachdefinitionen werden nicht überprüft. Der erste *internal-service-name* ist gültig, weitere gleichen Namens werden ignoriert.

Pflichtoperand!

RSN=remote-service-name
: maximal 16 Byte langer Name eines Services in der fernen Anwendung. Dieser Name wird an die ferne Anwendung übertragen (TPSVCINFO-Struktur); er darf in der LCF mehrfach vorkommen.

  Wird dieser Operand weggelassen, dann setzt das Tool `xatmigen` für RSN den Wert *internal-service-name* ein.

TAC=transaktion-code
: maximal 8 Byte langer Transaktionscode, mit dem der Service in der lokalen Anwendung angesprochen wird. Diesen Transaktionscode können Sie sich vom Dienstprogramm *xatmigen* in eine LTAC-Anweisung für die KDCDEF-Generierung umsetzen lassen. Es gelten die entsprechenden Namenskonventionen von openUTM.

  Wird dieser Operand weggelassen, dann setzt das Tool `xatmigen` für TAC den Wert *internal-service-name* ein und kürzt diesen ggf. auf die ersten 8 Byte.

DEST=destination-name
: Maximal 8 Byte lange Identifikation der Partneranwendung.

  Diesen Namen müssen Sie in einer OSI-LPAP oder LPAP-Anweisung bzw. MASTER-OSI-LPAP- oder MASTER-LU61-LPAP-Anweisungen als *lpapname* generieren.

  Pflichtoperand!

  .DEFAULT Es wird ein Default-Server verwendet.

MODE=RR / RN / CV
: Bestimmt, welches Kommunikationsmodell für den Service verwendet wird:

  RR Request-Response Modell (Standardwert)
  RN Request with no Response Modell (Single Request), bewirkt auf der openUTM-Seite den Start eines Asynchron-TACs.
  CV Conversational Modell.

BUFFERS=(subtype-1,...,subtype-n)
: Liste von Subtyp-Namen, die an den Service geschickt werden dürfen (der Typ X_OCTET ist immer erlaubt). Jeder Name darf maximal 16 Byte lang sein.

  Für jeden hier aufgeführten Subtyp muss eine eigene BUFFER-Anweisung angegeben werden, mit der die Eigenschaften des Subtyps definiert werden (siehe BUFFER-Anweisung Seite 126).

  Der Operand BUFFERS= ist stellungssensitiv und muss (falls angegeben) immer der *letzte* Operand der Anweisung sein.

  Wird BUFFERS= weggelassen, dann sollten an den Service nur Puffer vom Typ X_OCTET gesendet werden (eine Typüberprüfung findet nicht statt).

SVCP und REQP-Anweisung: Angebotenen Service/Request definieren

- Eine SVCP-Anweisung beschreibt für den Server die Eigenschaften eines angebotenen End-Services. Dieser Service ruft keine weiteren Services mehr auf. Ein End-Service mit dem Conversational Modell muss mit PGWT-TAC-Klassen generiert werden (siehe Seite 130)
- Eine REQP-Anweisung beschreibt für den Server die Eigenschaften eines angebotenen Intermediate-Services. Dieser Service ruft selber wieder einen Service auf. Solche Services müssen mit PGWT-TAC-Klassen generiert werden (siehe Seite 130).

Die Anweisungen SVCP und REQP sind optional und dienen dazu, dem Anwender die UTM-Generierung zu erleichtern. Wenn sie angegeben werden, dann können Sie sich vom Tool `xatmigen` für die Services die passenden KDCDEF-Anweisungen erzeugen, siehe Seite 127 und Seite 130. Andernfalls muss dies der Anwender tun.

Operator	Operanden	Erläuterung
SVCP REQP	internal-service-name	maximal 16 Byte
	[,TAC=transaktion-code]	Standard: internal-service-name
	[,PROG=programm-name]	Standard: internal-service-name
	[,COMP=compiler-language]	Standard: C (unter Unix- und Windows-Systemen) ILCS (unter BS2000-Systemen)
	[,MODE=RR / RN / CV]	RR=Request/Response, Standard RN=Request no Response CV=Conversation

SVCP Mit der Anweisung 'Service Provided' werden End-Services definiert.

REQP Mit der Anweisung 'Request Provided' werden Intermediate-Services definiert.

internal-service-name
: maximal 16 Byte langer Name, unter dem ein (ferner) Service in der lokalen Anwendung angesprochen wird. Dieser Name muss innerhalb der Anwendung eindeutig sein.

  Mehrfachdefinitionen werden nicht überprüft. Der erste *internal-service-name* ist gültig, weitere gleichen Namens werden ignoriert.

  Pflichtoperand!

TAC=transaktion-code
: maximal 8 Byte langer Transaktionscode, mit dem das Service-Teilprogramm gestartet wird. Es gelten die Namenskonventionen von openUTM.

Für diesen Transaktionscode können Sie sich vom Dienstprogramm `xatmigen` eine vom Kommunikationsmodell abhängige TAC-Anweisung für die KDCDEF-Generierung erzeugen lassen, ggf. mit zugehöriger TACCLASS-Anweisung .

Wird dieser Operand weggelassen, dann setzt das Tool `xatmigen` TAC=*internal-service-name* und kürzt diesen ggf. auf die ersten 8 Byte.

PROG=programm-name

maximal 8 Byte langer Name des Teilprogramms, das die Service-Funktion anbietet. Diesen Namen können Sie sich vom Dienstprogramm xatmigen in eine PROGRAM-Anweisung für die KDCDEF-Generierung umsetzen lassen.

Es gelten die Namenskonventionen von openUTM.

COMP=compiler-language

beschreibt die Programmiersprache des Service-Teilprogramms und wird in den Operanden COMP= bei der PROGRAM-Anweisung umgesetzt:

X/W	C	C-Programm, Standardwert unter Unix- und Windows-Systemen
X/W	COB2/MFCOBOL/NETCOBOL	
X/W		COBOL-Programm
X/W	CPP	C++-Programm
B	ILCS	Linkage über ILCS, Standardwert unter BS2000-Systemen

B Unter BS2000-Systemen darf nur COMP=ILCS verwendet werden.

MODE=RR / RN / CV

Bestimmt, welches Kommunikationsmodell für den Service verwendet wird:

RR	Request-Response Modell (Standardwert).
RN	Request with no Response Modell (Single Request). Dadurch wird ein Asychron-TAC generiert.
CV	Conversational Modell.

BUFFER-Anweisung

Eine BUFFER-Anweisung definiert einen typisierten Puffer. Gleichnamige Puffer müssen client- und serverseitig gleich definiert sein.
Mehrfachdefinitionen werden nicht überprüft. Der erste Puffer-Eintrag ist gültig, alle anderen werden ignoriert.

Puffer des Typs "X_OCTET" haben keine besonderen Eigenschaften und benötigen deshalb keine Definition. Typisierte Puffer werden mit folgenden Parametern definiert:

Operator	Operanden	Erläuterung
`BUFFER`	`subtype-name`	maximal 16 Byte
	`[,REC=referenced-record-name]`	Standard: subtype-name
	`[,TYPE=X_COMMON / X_C_TYPE]`	Standard: Tool `xatmigen` setzt TYPE automatisch

subtype-name
: Maximal 16 Byte langer Name des Puffers, der auch bei der SVCU-Anweisung im Operanden BUFFERS= angegeben werden muss. Der Name muss in der Anwendung eindeutig sein.

REC=referenced-record-name
: Name der Datenstruktur für den Puffer, z.B. ist dies bei C-Strukturen der Name des "typedef" bzw. der "struct-Name".

  Wird der Operand weggelassen, dann setzt `xatmigen` REC=*subtype-name* ein.

TYPE=
: Typ des Puffers, näheres zu den Typen siehe Seite 111.

  Wird der Operand weggelassen, dann setzt `xatmigen` den Typ auf X_C_TYPE oder X_COMMON, je nachdem, welche elementaren Datentypen verwendet wurden.

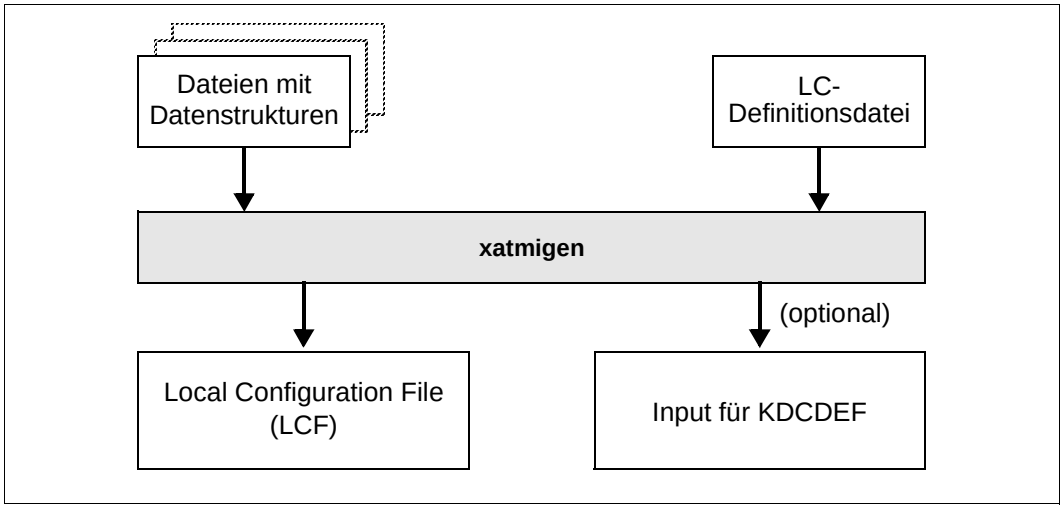
  TYPE wird ignoriert, wenn für TYPE nicht X_COMMON oder X_C_TYPE gesetzt ist oder wenn die Datenstruktur nicht vom angegebenen Puffertyp ist.

Das Tool `xatmigen` erzeugt beim Generierungslauf zusätzlich zwei Operanden mit folgender Bedeutung:

LEN=länge	Länge des Datenpuffers.
SYNTAX=code	Syntaxbeschreibung der Datenstruktur in der Code-Darstellung, wie sie in der Tabelle auf Seite 112 aufgeführt ist.

4.5.2 Das Dienstprogramm xatmigen

Das Dienstprogramm `xatmigen` bereitet aus einer Datei mit der Local Configuration-Definition (LC-Definitionsdatei) und einem oder mehreren Dateien mit C- oder COBOL-Datenstrukturen (LC-Description Files) eine Local Configuration File (LCF) auf, siehe folgendes Bild:



Die Local Configuration File ist gleich aufgebaut wie die LC-Definitionsdatei und unterscheidet sich von dieser nur in der zusätzlichen Beschreibung von Puffertyp, Pufferlänge und Syntaxstring des Puffers. D.h. die BUFFER-Anweisungen werden gegenüber der Definitionsdatei erweitert um die Operanden LEN=, SYNTAX= und ggf. TYPE=.

Falls in der LC-Definitionsdatei der Puffertyp nicht angegeben ist, generiert `xatmigen` den jeweils "kleinsten" Wertebereich für den Puffertyp, d.h. zuerst den Typ X_COMMON.

Alle Dateinamen müssen explizit angegeben werden. Als Option kann eine Datei erstellt werden, die Generierungsanweisungen für KDCDEF enthält.

X/W X/W Unter Unix- und Windows-Systemen werden Erfolgs- und Fehlermeldungen nach `stdout` und `stderr` geschrieben.

B B Unter BS2000-Systemen werden Erfolgs- und Fehlermeldungen nach SYSOUT und SYSLST geschrieben.

Obwohl das Editieren der LCFs prinzipiell möglich ist, wird dringend abgeraten.

Aufruf von xatmigen

X/W Unter Unix- und Windows-Systemen wird `xatmigen` aufgerufen mit

X/W `xatmigen` *parameter*

X/W `xatmigen` finden Sie in folgendem Dateiverzeichnis:

X *utmpfad*`/xatmi/ex` (Unix-Systeme) bzw.

W *utmpfad*`\xatmi\ex` (Windows-Systeme)

B B Unter BS2000 starten Sie `xatmigen` mit dem SDF-Kommando START-XATMIGEN. Alternativ können Sie `xatmigen` auch mit folgendem dem Kommando starten:

B B
```
/START-EXECUTABLE-PROGRAM FROM-FILE=*LIB
       ELEM(LIBRARY=$userid.SYSLNK.UTM.063.UTIL,ELEMENT-OR-SYMBOL=XATMI-GEN)
```

Es können folgende Parameter angegeben werden; dabei müssen die Schalter (-d, -l, -i, -c) klein geschreiben werden:

[utm]
␣**-d**␣lcdf-name
[␣**-l**␣lcf-name]
[␣**-i**]
[␣**-c**␣stringcode]
[␣descript-file-1]`...` [␣descript-file-n]

utm
: Wird "utm" angegeben, erzeugt `xatmigen` aus den SVCU-, SVCP- und REQP-Anweisungen eine Datei mit Generierungsanweisungen für KDCDEF. `xatmigen` erzeugt nur TAC- und TACCLASS-Anweisungen für Anwendungen mit Auftragssteuerung durch Prozessbeschränkung. Die von `xatmigen` erzeugten Generierungsanweisungen müssen vor der UTM-Generierung noch ergänzt werden, siehe Seite 130.

  "utm" muss, sofern angegeben, immer der erste Parameter von `xatmigen` sein. Fehlt die Angabe, dann werden keine Generierungs-Anweisungen erzeugt.

  Die erzeugte Datei hat den Namen `xtutm.def`. Die Datei wird in das aktuelle Dateiverzeichnis (Unix- und Windows-System) bzw. in die aktuelle Kennung (BS2000-Systeme) geschrieben.

-d␣lcdf-name
: Name der LC-Definitionsdatei, Pflichtangabe.

-l␣lcf-name
: Name der zu erzeugenden Local Configuration File. Der Name muss den Konventionen des jeweiligen Betriebssystems entsprechen.

Falls das LCF gleichzeitig für ein Unix- und Windows-System oder ein BS2000- und Windows-System verwendet wird, wird empfohlen, den Namen maximal 8 Zeichen lang zu wählen und mit der Erweiterung ".lcf" zu versehen.

Eine eventuell vorhandene, gleichnamige LCF wird kommentarlos überschrieben.

Wird der Schalter weggelassen, dann erzeugt `xatmigen` die Datei `xatmilcf` im aktuellen Dateiverzeichnis (Unix-/Windows-System) bzw. in der aktuellen Kennung (BS2000-System).

-i Interaktiver Modus: Bei jedem typisierten Puffer, der ein Character-Array enthält, wird dessen Stringcode erfragt. Die möglichen Angaben für den Stringcode sind bei Schalter "-c". beschrieben.

Der Schalter -i hat Vorrang vor einem eventuell ebenfalls vorhandenen Schalter -c.

Wenn `xatmigen` im Hintergrund bzw. Batchbetrieb abläuft, dann darf der Schalter -i nicht angegeben werden.

-c␣stringcode

Der angegebene Stringtyp gilt für den gesamten `xatmigen`-Lauf, d.h. für alle Character Arrays.
Im interaktiven Modus ("-i") wird Schalter "-c" ignoriert.

Für *stringcode* sind folgende Angaben möglich (siehe Tabelle auf Seite 112):

C Octet-String
C! Octet-String, durch '\0' terminiert
T T.61-String
T! T.61-String, durch '\0' terminiert

Bei fehlender Angabe wird T! eingesetzt.

Auch Einzel-Character werden als T.61-String (*stringcode*= t) interpretiert.

descript-file-1 . . . ␣descript-file-n

Liste der Dateien, die die Include-Dateien bzw. COPY-Elemente mit den Datenstrukturen der typisierten Puffer enthalten.

Fehlt die Angabe, ist nur der Puffertyp X_OCTET zugelassen.

Hinweis

Dem Schalter **-d** und, sofern angegeben, den Schaltern **-l** und **-c** muss jeweils der zugehörige Parameter folgen. Die Angabe des Schalters ohne nachfolgenden Parameter ist nicht zulässig.

4.5.3 KDCDEF-Generierung

Um eine XATMI-Anwendung funktionsfähig zu machen, müssen Sie

– in Anwendungen mit Prozessbeschränkung TAC-Klassen generieren, für die PGWT erlaubt ist

– in Anwendungen mit Priortätensteuerung TACs generieren, für die PGWT erlaubt ist.

– die Teilprogramme der lokal angebotenen Services definieren (TAC- und PROGRAM-Anweisungen)

– die Services definieren, die im fernen Rechner genutzt werden (LTAC-Anweisungen)

– die OSI-TP- bzw. LU6.1-Verbindungen generieren

4.5.3.1 TACs mit PGWT=YES

Ein Service ist immer an einen Workprozess (Unix- und Windows-Systeme) bzw. eine Task (BS2000-Systeme) gebunden. Sobald in einer Server-Anwendung sowohl Requests als auch Conversational Services enthalten sind, müssen mindestens zwei Workprozesse bzw. Tasks gestartet werden und für den TAC müssen PGWT-Aufrufe erlaubt sein.

Bei einer Anwendung, die ausschließlich Request/Response-Services enthält, ist dies nicht notwendig.

4.5.3.2 Anzahl der Session (LU6.1) bzw. Associations (OSI-TP)

Bei jedem XATMI-Aufruf *tpcall*(), *tpacall*() und *tpconnect*() wird in openUTM ein Vorgang adressiert (KDCS-Aufruf APRO). Daher muss für jeden dieser XATMI-Aufrufe eine eigene Session bzw. Association generiert werden.

4.5.3.3 Angebotene Services definieren

Für angebotene Services müssen folgende KDCDEF-Anweisungen gegeben werden:

- Eine PROGRAM- und TAC-Anweisung für jedes Teilprogramm, das eine Servicefunktion bereitstellt. Die TAC-Anweisung muss den Operanden API=(XOPEN,XATMI) enthalten.

  Wird beim Dienstprogramm `xatmigen` für den Parameter *trägersystem* "utm" angegeben, dann wird für jede SVCP- oder REQP-Anweisung je eine PROGRAM- und eine TAC-Anweisung erzeugt. Diese Anweisungen können unverändert als KDCDEF-Input übernommen werden.

Beispiel:

```
PROGRAM svcproc1, COMP = ILCS
TAC     service1, PROGRAM = svcproc1, API = (XOPEN,XATMI)
PROGRAM svcproc2, COMP = ILCS
TAC     service2, PROGRAM = svcproc2, API = (XOPEN,XATMI)
```

- Für jede Verbindung zu einem fernen Partner müssen die entprechenden Anweisungen gegeben werden:
  - Bei OSI-TP-Verbindungen zu einem openUTM- oder OpenCPIC-Partner die Anweisungen ACCESS-POINT, OSI-LPAP und OSI-CON, z.B. wie folgt:

```
* Name des lokalen Zugriffspunktes
ACCESS-POINT  server,
              P-SEL = *NONE, S-SEL = *NONE, T-SEL = C'tsel',

* lokaler Name für entfernten Partner
OSI-LPAP    lcltname,
            ASSOCIATION-NAMES = assoname,
            ASSOCIATIONS = 2, CONTWIN = 1, CONNECT = 1,
            APPLICATION-CONTEXT = {XATMIAC| XATMICCR}

* lokaler Name für die Connection
OSI-CON     lconname,
            P-SEL=*NONE, S-SEL=*NONE,
            T-SEL=C'tsel', N-SEL=C'machname',
            LOCAL-ACCESS-POINT = server,
            OSI-LPAP = lcltname
```

  - Bei LU6-Verbindungen zu einem openUTM- oder OpenCPIC-Partner müssen die Anweisungen CON, LPAP und LSES angegeben werden, siehe openUTM-Handbuch „Anwendungen generieren“.
  - Es darf keine automatische Konvertierung generiert werden, d.h. es darf nicht MAP = SYSTEM in der OSI-LPAP- bzw. SESCHA-Anweisung angegeben werden.
  - Für jeden UPIC-Partner muss eine PTERM-Anweisung gegeben werden, z.B. :

```
PTERM tnsclient, PTYPE=UPIC-R, PRONAM=DxxxSyyy (bei UPIC-Remote)
PTERM client1, PTYPE=UPIC-L                    (bei UPIC-Local)
```

    Falls der Client Security-Daten (Benutzer, Password) übergibt, ist noch eine USER-Anweisung nötig.

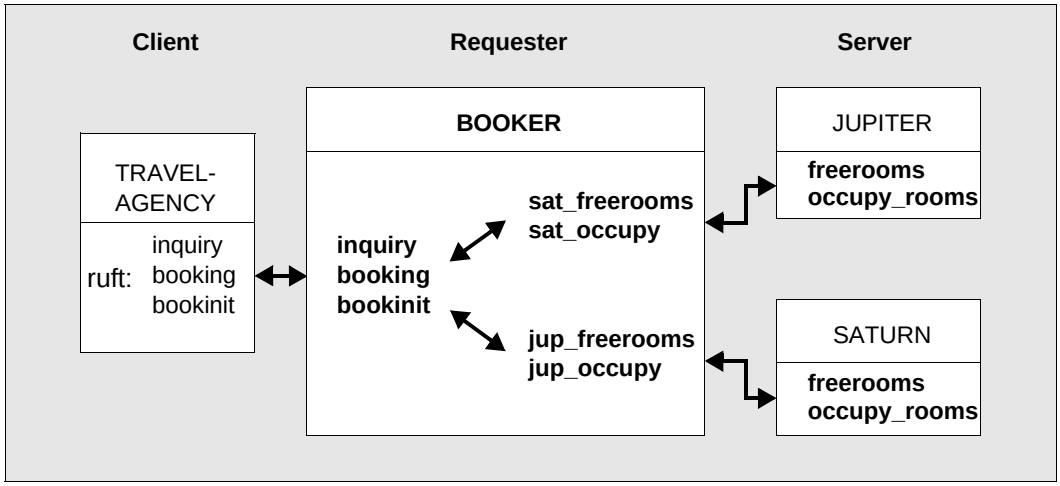
4.5.3.4 Genutzte Services definieren

Für jeden aufgerufenen Service muss eine LTAC-Anweisung gegeben werden. Wird `xatmigen` mit *trägersystem* "utm" aufgerufen, dann erzeugt `xatmigen` automatisch pro SVCU-Anweisung eine LTAC-Anweisung von der Form LTAC *ltac*.

Diese LTAC-Anweisungen müssen Sie noch ergänzen um den Operanden LPAP= und ggf. um weitere Operanden wie z.B. RTAC= oder TYPE=. Alle weiteren KDCDEF-Anweisungen für den fernen Partner wie z.B. LPAP oder OSI-LPAP müssen Sie ebenfalls einfügen.

4.5.3.5 Beispiel für eine Requester-Generierung

Das folgende Beispiel zeigt die Schritte einer XATMI-Requester Generierung. Es behandelt die Generierung für eine zentrale "Hotelbuchungs-Anwendung" "BOOKER" (Requester), die zum einen mit Client-Anwendungen "TRAVEL-AGENCY" in den Reisebüros, und zum anderen mit den beiden Hotelmanagement-Anwendungen "JUPITER" und "SATURN" (Server) in den Hotels verbunden ist.



BOOKER bietet die Services "inquiry", "booking" und "bookinit" an, die von den Clients aufgerufen werden können, die beiden Server bieten jeweils die End-Services "freerooms" und "occupy_rooms" an. Im einzelen gilt:

– "bookinit" ist ein End-Service und dient dazu, BOOKER zu initialieren.

– "inquiry" und "booking" sind Intermediate Services, die in verschiedenen Hotels nach Zimmern fragen und dann das Zimmer buchen. Sie sind als Funktionen "inqproc" und "bookproc" implementiert.

– "inquiry" benutzt das Conversational Modell und ruft die End-Services "freerooms" in den beiden Hotels auf. "freerooms" wird in BOOKER durch "sat_freerooms" bzw. durch "jup_freerooms" angesprochen. Der "freerooms"-Service benötigt keine typisierten Puffer.

– "booking" benutzt das Synchrone Request/Response- Modell und ruft die End-Services "occupy_rooms" in den beiden Hotels auf. "occupy_rooms" wird in BOOKER durch "sat_occupy" bzw. durch "jup_occupy" angesprochen. Der "occupy-rooms"-Service verlangt einen Personendatensatz "t_person" unter dem Puffernamen "personbuffer".

Die Generierung wird in folgenden Schritten durchgeführt:

1. Datenstrukturen bereitstellen (LC-Description Files)

   Die beteiligten Partner müssen sich auf einheitliche Datenstrukturen einigen und gegenseitig bekannt machen. Diese Datenstrukturen (in C oder Cobol) beschreiben die typisierten Puffer:

   Datei: userbuf.h

```
typedef struct {
          char  name[20];
          char  sex;
          short persons;
          float huge;
          int   duration_of_last_visits[16];
      } t_person;
```

2. Eingabedatei für das Local Configuration File erstellen (LC-Definition File)

   In dieser Datei werden die verwendeten Services und Puffer definiert:

   Datei: booker.def

```
# bereitgestellte Services:
SVCP bookinit;
REQP inquiry, PROG=inqproc,  MODE=CV;
REQP booking, TAC=booktac, PROG=bookproc, MODE=RR;

# gerufene Services:
SVCU sat_freerooms, RSN=freerooms,   TAC=frmtac, DEST=SATURN;
SVCU sat_occupy,    RSN=occupy_room, TAC=occtac, DEST=SATURN \
     BUFFERS=(personbuffer);
SVCU jup_freerooms, RSN=freerooms,   TAC=frmtac, DEST=JUPITER;
SVCU jup_occupy,    RSN=occupy_room, TAC=occtac, DEST=JUPITER \
     BUFFERS=(personbuffer);

# verwendete Subtypes:
BUFFER personbuffer, REC=t_person;
```

3. Local Configuration generieren

   Die Local Configuration wird mit dem Dienstprogramm `xatmigen` erstellt. Beim Aufruf müssen die Eingabedateien für den typisierten Puffer und für das LCF angegeben werden. Bei fehlerfreiem Generierungslauf erzeugt `xatmigen` die Local Configuration File, in diesem Beispiel zusätzlich die Datei mit dem Rahmen für KDCDEF

X/W X/W Aufruf unter Unix- und Windows-Systemen:

```
xatmigen utm -c C -l booker.lcf -d booker.def userbuf.h
```

B B B B

```
Aufruf unter BS2000-Systemen:
/START-XATMIGEN
%  CCM0001 PARAMETER EINGEBEN:
* params utm -c C -l BOOKER.LCF -d BOOKER.DEF USERBUF.H
```

`xatmigen` erstellt folgende Dateien:

Datei booker.lcf:

```
# bereitgestellte Services:
SVCP bookinit TAC=bookinit PROG=bookinit COMP=C    MODE=RR;
REQP inquiry  TAC=inquiry  PROG=inqproc  COMP=C    MODE=CV;
REQP booking  TAC=booktac  PROG=bookproc COMP=C    MODE=RR;
**COMP = ILCS unter BS2000-Systemen**

# gerufene Services:
SVCU sat_freerooms RSN=freerooms   TAC=frmtac DEST=SATURN  MODE=RR;
SVCU sat_occupy    RSN=occupy_room TAC=occtac DEST=SATURN  MODE=RR \
     BUFFERS=(personbuffer);
SVCU jup_freerooms RSN=freerooms   TAC=frmtac DEST=JUPITER MODE=RR;
SVCU jup_occupy    RSN=occupy_room TAC=occtac DEST=JUPITER MODE=RR \
     BUFFERS=(personbuffer);

# verwendete Subtypen:
BUFFER personbuffer REC=t_person TYPE=X_C_TYPE LEN=91 \
       SYNTAX=C2OcsfI16;
```

Datei xtutm.def:

```
*#bereitgestellte Services:
TAC     bookinit, PROGRAM = bookinit, API = (XOPEN,XATMI)
PROGRAM bookinit, COMP = C          // COMP = ILCS unter BS2000-Systemen
MAX TASKS-IN-PGWT=1
TACCLASS 1, TASKS-FREE=1, PGWT=YES
TAC     inquiry,  PROGRAM = inqproc, API = (XOPEN,XATMI)
PROGRAM inqproc,  COMP = C          // COMP = ILCS unter BS2000-Systemen
TAC     booktac,  PROGRAM = bookproc, API = (XOPEN,XATMI)
PROGRAM bookproc, COMP = C          // COMP = ILCS unter BS2000-Systemen
* #gerufene Services:
LTAC    frmtac , WAITTIME=(10,30)
```

```
LTAC    occtac , WAITTIME=(10,30)
LTAC    frmtac , WAITTIME=(10,30) // nachträglich löschen
LTAC    occtac , WAITTIME=(10,30) // nachträglich löschen
*# verwendete Puffer
```

4. Generieren der UTM-Anwendung

   Zuerst muss (z.B. per Editor) die Eingabedatei für den KDCDEF-Lauf erzeugt werden. Als Grundlage verwenden Sie die Datei "xtutmdef", generieren die Verbindung zur Anwendung HOTEL und fügen alle sonst noch notwendigen KDCDEF-Anweisungen hinzu, siehe openUTM-Handbuch „Anwendungen generieren“. Eine Mischung von XATMI-Teilprogrammen, CPI-C-Teilprogrammen und KDCS-Teilprogrammen in einer UTM-Anwendung ist erlaubt.

   Anschließend verfahren Sie mit dieser UTM-Anwendung wie mit jeder anderen openUTM- Anwendung auch, d.h. Sie generieren mit KDCDEF, binden die Anwendung und starten sie.

4.6 Erstellen von XATMI-Anwendungen

Eine XATMI-Anwendung ist eine UTM-Anwendung, die XATMI-Teilprogramme enthält. Sie wird wie eine "normale" UTM-Anwendung gestartet.

Für die Erstellung von XATMI-Anwendungen unter openUTM stehen Ihnen Include-Dateien für C-Programme und COPY-Elemente für COBOL-Programme zur Verfügung.

Welche Bibliotheken Sie für das Binden benötigen, ist in den Abschnitten "Binden der Anwendung unter Unix- und Windows-Systemen" und "Binden der Anwendung unter BS2000-Systemen" beschrieben.

4.6.1 Include-Dateien und COPY-Elemente

C-Module mit XATMI-Aufrufen benötigen folgende Include-Dateien:

X/W X/W
- Unter Unix- und Windows-Systemen die Include-Datei `xatmi.h`, die sich in folgendem Verzeichnis befindet:

  X
  – *utmpfad*`/xatmi/include` (Unix-Systeme)

  W
  – *utmpfad*`\xatmi\include` (Windows-Systeme)

  X/W X/W
  xatmi.h includiert die mit ausgelieferte Datei `xatmidef.h`, die sich im selben Dateiverzeichnis befindet.

B B B B
- Unter BS2000-Systemen die Include-Datei XATMI.H.
  Diese Datei ist in der Bibliothek *$userid*.SYSLIB.UTM.063.XOPEN enthalten (XATMI.H includiert die mitausgelieferte Datei XATMIDEF.H, die sich in derselben Bibliothek befindet).

- Die Datei(en) mit den Datenstrukturen für alle typisierten Puffer, die im Modul verwendet werden, siehe auch Seite 111.

COBOL-Module mit XATMI-Aufrufen benötigen folgende COPY-Elemente:

- Die COPY-Elemente TPSTATUS, TPTYPE, TPSVCDEF, TPSVCRET und TPRETURN.

  X/W X/W
  Unter Unix- und Windows-Systemen finden Sie die COPY-Elemente in folgendem Dateiverzeichnis:

  X
  – *utmpfad*`/xatmi/copy-cobol85` bzw. *utmpfad*`/xatmi/netcobol` (Unix-Systeme)

  W
  – *utmpfad*`\xatmi\copy-cobol85` bzw. *utmpfad*`\xatmi\netcobol` (Windows-Systeme)

  B B
  Unter BS2000-Systemen finden Sie die COPY-Elemente in der Bibliothek $userid.SYSLIB.UTM.063.XOPEN.

- Die Datei(en) mit den Datenstrukturen für alle "typed records", die im Modul verwendet werden.

B B Unter BS2000-Systemen müssen alle Teilprogramme mit der LINKAGE-Option für ILCS übersetzt werden.

4.6.2 Binden der Anwendung unter Unix- und Windows-Systemen

X/W X/W Beim Binden einer XATMI-Server-Anwendung müssen folgende Bibliotheken mit dazugebunden werden.

X/W
- Alle Service-Teilprogramme und -module

X
- Auf Unix-Systemen die XATMI-Server-Bibliothek *utmpfad*`/sys/libxopen`

X/W
- Die OSS-Bibliothek (nur bei OSI-TP-Protokoll notwendig)

X/W
- Die openUTM-Bibliothek

X
- Auf Unix-Systemen die mathematische Bibliothek (Angabe `-lm`)

4.6.3 Binden der Anwendung unter BS2000-Systemen

B B Beim Binden einer XATMI-Server-Anwendung müssen folgende Bibliotheken mit dazugebunden werden.

B 1. Alle Service-Teilprogramme und -module

B 2. Die XATMI-Server-Bibliothek $userid.SYSLIB.UTM.063.XOPEN

B 3. Die Bibliothek für CRTE, z.B. SYSLNK.CRTE

B 4. Die openUTM-Bibliothek

4.7 Umgebungs- bzw. Jobvariablen für XATMI

Für XATMI-Anwendungen werden von openUTM eine Reihe von Umgebungsvariablen (unter Unix- und Windows-Systemen) bzw. Jobvariablen (unter BS2000-Systemen) ausgewertet. Die Umgebungs- bzw. Jobvariablen müssen vor dem Start der Anwendung gesetzt werden.

Zur Diagnose bei laufender Anwendung können Traces eingeschaltet werden, siehe Abschnitt „Steuerung des Trace“.

4.7.1 Umgebungsvariablen unter Unix- und Windows-Systemen

X/W Für eine XATMI-Anwendung werden folgende Umgebungsvariablen ausgewertet:

X/W XTLCF — Dateiname der verwendeten Local Configuration File (LCF).
X/W Der Dateiname der Local Configuration File muss den Konventionen des
X/W Betriebssystems entsprechen.

X/W Falls XTLCF nicht gesetzt ist, wird im aktuellen Dateiverzeichnis unter dem
X/W Namen `xatmilcf` gesucht.

X/W XTPALCF — Definiert den Suchpfad für zusätzliche Beschreibungen von typisierten
X/W Puffern.
X/W Die Pufferbeschreibungen werden aus Local Configuration Files mit dem
X/W Namen xatmilcf bzw. dem in XTLCF festgelegten Namen gelesen.

X/W Alle wichtigen XATMI-Generierungen (z.B. SVCU ...) werden auch
X/W weiterhin in dem über XTLCF festgelegten Local Configuration File
X/W gesucht.

X/W Alle in XTPALCF angegebenen Dateiverzeichnisse werden nach Local
X/W Configuration Files durchsucht und die Beschreibungen der typisierten
X/W Puffer werden intern gesammelt (bei Namensgleichheit wirkt nur die erste
X/W Pufferbeschreibung).

X/W Der Suchpfad wird genauso aufgebaut wie in der Umgebungsvariablen
X/W PATH:

X *verzeichnis1*:*verzeichnis2*: ... (Unix-Systeme)

W *verzeichnis1*;*verzeichnis2*; ... (Windows-Systeme)

4.7.2 Jobvariablen unter BS2000-Systemen

B B Für eine XATMI-Anwendung können Jobvariablen gesetzt werden, die über folgende Linknamen (Kettungsnamen) mit der Anwendung verbunden werden:

B B B B B XTLCF Link auf Jobvariable mit dem Dateinamen für die Local Configuration File (LCF).
Der Dateiname der Local Configuration File muss den Konventionen des Betriebssystems entsprechen.
Die Datei wird unter der aktuellen Benutzerkennung gesucht.

B B Ist XTLCF keiner Jobvariablen zugeordnet, dann wird in der aktuellen Benutzerkennung unter dem Namen XATMILCF gesucht.

B B B B XTPALCF Link auf Jobvariable mit dem Suchpfad für zusätzliche Beschreibungen von typisierten Puffern.
Die Pufferbeschreibungen werden aus Local Configuration Files mit dem Namen XATMILCF bzw. dem über XTLCF festgelegten Namen gelesen.

B B B Alle wichtigen XATMI-Generierungen (z.B. SVCU ...) werden auch weiterhin in dem über XTLCF festgelegten Local Configuration File gesucht.

B B B B Unter allen im Suchpfad angegebenen Kennungen wird nach Local Configuration Files gesucht und die Beschreibungen der typisierten Puffer aus diesen Dateien werden intern gesammelt (bei Namensgleichheit wirkt nur die erste Puffer-Beschreibung).

B Der Suchpfad wird in der Form *kennung1***:***kennung2***:** ... angegeben.

4.8 Fehlerdiagnose in XATMI-Programmen

XATMI-Aufrufe werden auf KDCS-Aufrufe abgebildet. Der UTM-Dump enthält deshalb auch bei XATMI-Teilprogrammen nur die KDCS-Aufrufe. Sie können für die Diagnose von XATMI-Programmen jedoch zusätzlich zum UTM-Dump einen Trace der XATMI-Aufrufe erzeugen.

4.8.1 Steuerung des Trace

Der XATMI-Trace kann wie folgt gesteuert werden:

- Durch den UTM-Startparameter XATMI-TRACE kann der Trace beim Start der Anwendung eingeschaltet werden, siehe jeweiliges openUTM-Handbuch „Einsatz von openUTM-Anwendungen“.
- Über WinAdmin oder WebAdmin kann der Trace im laufenden Betrieb ein- und ausgeschaltet werden. Dazu verwenden Sie den Dialog *Eigenschaften der UTM-Anwendung*, Registerkarte *Diagnose und Abrechnung*, Feld *XATMI Trace*.
- Über die Programmschnittstelle zur Administration KDCADMI kann der Trace im laufenden Betrieb ein- und ausgeschaltet werden. Dazu dient das Feld *xatmi_trace* in der Datenstruktur *kc_diag_and_account_par_str*, siehe openUTM-Handbuch „Anwendungen administrieren“.

Sie können folgende Trace-Level einstellen:

Level	**Bedeutung**
ERROR	Es werden nur Fehler protokolliert.
INTERFACE	Umfasst den Level ERROR, zusätzlich werden alle XATMI-Aufrufe protokolliert.
FULL	Umfasst den Level INTERFACE, zusätzlich werden alle KDCS-Aufrufe, auf die die XATMI-Aufrufe abgebildet werden, protokolliert.
DEBUG	Umfasst den Level FULL, zusätzlich werden Diagnose-Informationen protokolliert.
OFF	Trace ist ausgeschaltet.

4.8.2 Name der Trace-Datei

X/W **Trace-Datei unter Unix- und Windows-Systemen**

X/W Die Trace-Sätze werden in die Datei KDC.TRC.XATMI.*appliname.hostname.pid* im
X/W Verzeichnis *filebase* geschrieben. Dabei bedeuten:

X/W appliname
X/W Name der Anwendung

X/W hostname
X/W Name des Rechners, auf dem die Anwendung läuft

X/W pid PID des Prozesses

B **Trace-Datei unter BS2000-Systemen**

B Die Trace-Sätze werden standardmäßig in die Datei
B KDC.TRC.XATMI.*appliname.hostname.tsn* geschrieben. Dabei bedeuten:

B appliname
B Name der Anwendung

B hostname
B Name des Rechners, auf dem die Anwendung läuft

B tsn TSN der UTM-Task

B Sie können auch in der UTM-Startprozedur für jede Task eine andere Trace-Datei
B einrichten und mit dem Kommando SET-FILE-LINK den Linknamen KDCXATMI zuweisen.

4.9 Zusammenarbeit mit der TX-Schnittstelle

Eine Beschreibung der TX-Schnittstelle unter openUTM finden Sie im Kapitel 5.

Unter openUTM sind XATMI-Programme immer Server. Da TX-Aufrufe zur Transaktionssteuerung nur bei XATMI-Client-Programmen benötigt werden, enthalten XATMI-Programme unter openUTM keine TX-Aufrufe.

Beim Aufruf eines XATMI-Services wird vom Client mit dem Aufrufparameter *flag* (in C) bzw. dem Feld TPTRAN-FLAG (in COBOL) gesteuert, ob ein aufgerufener UTM-Service in die globale Transaktion eingeschlossen wird.
Für die XATMI-C-Schnittstelle ist die Aufnahme des Service in die globale Transaktion der Standardwert. Soll der Service nicht in die globale Transaktion aufgenommen werden, muss explizit das Flag TPNOTRAN gesetzt werden.
Für die XATMI-COBOL-Schnittstelle gibt es keinen Standardwert, entweder TPTRAN oder TPNOTRAN muss gesetzt werden.

Wird der Service mit dem Flag TPTRAN gestartet, so ist er in die globale Transaktion eingeschlossen, TX-Aufrufe werden nicht benötigt.
Beim Aufruf tpreturn() wird durch den im Parameter *rval* zurückgelieferten Wert TPSUCCESS bzw. TPFAIL gesteuert, ob die Transaktion erfolgreich beendet oder zurückgesetzt wird.

Weitere Informationen zur Zusammenarbeit der TX- und der XATMI-Schnittstelle finden Sie in der X/Open-Spezifikation„Distributed Transaction Processing: The XATMI Specification“ im Abschnitt "Transaction Functions Affecting the XATMI Interface".

4.10 Meldungen

Im folgenden sind die Meldungen von XATMI und von XATMIGEN aufgeführt.

Meldungen von XATMI

Die Meldungen von XATMI haben die Form `XTnn meldungstext...` und werden unter Unix- und Windows-Systemen nach *stderr* bzw. unter BS2000-Systemen nach SYSOUT ausgegeben.

```
XT01    KATMI(&PID) SERVER initiated
        LC-file  : &LCF
        Tracefile: &TRACEFILE
```

Bedeutung
Startmeldung:
&PID: Prozess-ID (Unix- und Windows-System) bzw. Tasknummer (BS2000-System)
&LCF: Name der Local Configuration File
&TRACEFILE: Name der Trace-Datei ohne Generationsnummer

```
XT02    KATMI(&PID) Local configuration: &ERRTXT
```

Bedeutung
Fehler im Local Configuration File. &ERRTXT liefert einen ergänzenden Text.

```
XT03    XATMI(&PID) Error: &ERRTXT
```

Bedeutung
Fehler beim Aufruf einer XATMI-Funktion. &ERRTXT liefert einen ergänzenden Text.

```
XT04    KATMI(&PID) System error: &ERRTXT
```

Bedeutung
Systemfehler

Meldungen von XATMIGEN

Die Meldungen von XATMIGEN haben die Form `XGnn meldungstext...` und werden unter Unix- und Windows-Systemen nach *stderr* bzw. unter BS2000-Systemen nach SYSOUT ausgegeben.

X X Unter Unix-Systemen können Sie mit der Umgebungsvariablen LANG steuern, ob Sie deutsche oder englische Meldungen erhalten.

B B B Unter BS2000-Sytemen können Sie taskspezifisch unterschiedlichen Jobvariablen den Linknamen LANG zuweisen und den Wert 'D' oder 'E' als Sprachkennzeichen setzen. Damit können Sie steuern, ob Sie deutsche oder englische Meldungen erhalten.

```
XG01    Generierung des Local Configuration Files: &LCF / &DEF / &CODE
```

Bedeutung
Startmeldung des Tools.
&LCF Name des erzeugten Local Configuration Files
&DEF Name des erzeugten Generierungsfragments
&CODE Stringcode für Character Arrays

```
XG02    Generierung erfolgreich beendet
```

Bedeutung
Die LCF wurde erzeugt, die Generierung wurde erfolgreich beendet.

```
XG03    Generierung erfolgreich mit Warnungen beendet
```

Bedeutung
Die LCF wurde erzeugt. Es wird jedoch eine Warnung ausgegeben, da z.B. nicht benötigte Dateien angegeben wurden. Diese Warnung hat allerdings auf die Generierung keinen Einfluss.

```
XG04    Generierung wegen Fehlers beendet.
        Keine Datei erzeugt.
```

Bedeutung
Die LCF wurde nicht erzeugt, die Generierung konnte nicht durchgeführt werden. Die Ursache ist vorhergehenden Meldungen zu entnehmen.

```
XG05    &FTYPE Datei'&FNAME'
```

Bedeutung
Diese Meldung gibt die gerade bearbeitete Datei an in folgender Form:
&FTYPE: „Description“-File enthält Datenstrukturen
„Definition“-File enthält den LCF-Input
„LC“-File enthält die Local Configuration

&FNAME: Filename

```
XG10    Aufruf: &PARAM
```

Bedeutung
Syntaxfehler beim Aufruf von XATMIGEN:
&PARAM: Mögliche Aufrufparameter und Schalter

```
XG11    [Error] &FTYPE File 'FNAME' kann nicht erzeugt werden.
        &REASON
```

Bedeutung
Die Datei &FNAME des Typs &FTYPE kann nicht erzeugt werden.
&REASON enthält eine nähere Begründung.
&FTYPE: GEN = Generation Fragment File (=Generierungs-Anweisungen)
LC = Local Configuration File

XG12 `[Warning] Datei nicht gefunden.`

Bedeutung
Die Definition File oder eine Description File wurde nicht gefunden; möglicherweise existiert die Datei nicht.

XG13 `[Warning] Zu viele &OBJECTS, Maximum: &MAXNUM`

Bedeutung
Meldung über zu viele gefundene Objekte
&OBJECTS: Subtypen
&MAXNUM: Maximale Anzahl

XG14 `[Error] Zeile &LINE: Syntaxfehler, &HELPTEXT`

Bedeutung
Syntaxfehler in Zeile &LINE in der LC-Definition-Datei
&HELPTEXT: Hilfetext

XG15 `[Error] Zeile &LINE: Keine Record-Definition gefunden für Puffer &BUFF`

Bedeutung
Für den Puffer &BUFF in Zeile &LINE konnte keine zugehörige Record-Definition gefunden werden.

XG16 `[Error] Zeile &LINE: Basistyp-Fehler in Puffer &BUFF`

Bedeutung
Die Syntaxbeschreibung des Puffers &BUFF in Zeile &LINE der LCF enthält einen falschen Basistype (int, short usw.)

XG17 `[Error] &FTYPE File '&FNAME'kann nicht geöffnet werden.`
`&REASON`

Bedeutung
Die Datei &FNAME des Typs &FTYPE kann nicht geöffnet werden.
&REASON enthält eine nähere Begründung.
&FTYPE: DEF (= LC-Definition File)

XG18 `[Error] &REASON`

Bedeutung
Allgemeiner Fehler.
&REASON enthält eine nähere Begründung.

XG19 `[Message] Neuen Puffer erzeugt: '&BUFF'`

Bedeutung
&BUFF: Erzeugter Puffer

XG20 [Message] Servicename '&SVC' auf 16 Zeichen gekuerzt!

Bedeutung
&SVC : Servicename.

XG21 [Message] Zeile &LINE: unbekannte Anweisungszeile '&HELPTEXT'

Bedeutung
Meldung für die Zeile &LINE in der LC-Definition-Datei
&HELPTEXT: Hilfetext (ein Teil der LC-Zeile)

XG22 [Message] Zeile &LINE: Standardwert gesetzt MODE='&TEXT'

Bedeutung
Meldung für die Zeile &LINE in der LC-Definition-Datei
&TEXT: gesetzter Default Servicemode

4.11 T.61-Zeichensatz

	0	1	2	3	4	5	6	7	8	9	...	F
0			SP	0	@	P		p				
1			!	1	A	Q	a	q				
2			"	2	B	R	b	r				
3			#	3	C	S	c	s				
4			¤	4	D	T	d	t				
5			%	5	E	U	e	u				
6			&	6	F	V	f	v				
7			´	7	G	W	g	w				
8	BS		(	8	H	X	h	x				
9		SS2	)	9	I	Y	i	y				
A	LF	SUB	*	:	J	Z	j	z				
B		ESC	+	;	K	[	k		PLD	CSI		
C	FF		,	<	L		l	|	PLU			
D	CR	SS3	-	=	M	]	m					
E	LS1		.	>	N		n					
F	LS0		/	?	O	-	o					

Codetabelle T.61 gemäß CCITT Recommodation

Bedeutung der Abkürzungen:

BS=	BACKSPACE	SUB=	SUBSTITUTE CHARACTER
LF=	LINE FEED	ESC=	ESCAPE
FF=	FORM FEED	SS3=	SINGLE-SHIFT THREE
CR=	CARRIAGE RETURN	SP=	SPACE
LS1=	LOCKING SHIFT ONE	PLD=	PARTIAL LINE DOWN
LS0=	LOCKING SHIFT ZERO	PLU=	PARTIAL LINE UP
SS2=	SINGLE-SHIFT TWO	CSI=	CONTROL SEQUENCE INTRODUCER

5 X/Open-Schnittstelle TX

TX (Transaction Demarcation Interface) ist eine von X/Open standardisierte Programmschnittstelle zur Festlegung von Transaktionen über Rechnergrenzen hinweg. Sie steht in COBOL und C zur Verfügung.

Dieses Kapitel beschreibt die Besonderheiten der TX-Schnittstelle unter openUTM. Eine genaue Beschreibung der TX-Schnittstelle und der Aufruf-Formate finden Sie in der X/Open CAE Spezifikation „Distributed Transaction Processing: The TX (Transaction Demarcation) Specification“, vom April 1995.
Die Kenntnis dieser Spezifikation wird im Folgenden vorausgesetzt.

5.1 Charakteristik transaction_control

Transaktionen können mit TX entweder verkettet oder unverkettet ausgeführt werden.

Hat die Charakteristik *transaction_control* den Wert TX_CHAINED, muss nur die erste Transaktion explizit begonnen werden: das Transaktionsende markiert implizit den Beginn der nächsten Transaktion.

Hat die Charakteristik *transaction_control* den Wert TX_UNCHAINED, muss der Beginn jeder Transaktion explizit markiert werden.

openUTM arbeitet immer mit *transaction_control*=TX_CHAINED.
Beim Start eines Service unter openUTM wird automatisch eine Transaktion begonnen, daher entfällt auch die Markierung der ersten Transaktion.

5.2 Aufrufe der TX-Schnittstelle unter openUTM

Die folgende Tabelle listet alle TX-Aufrufe auf, die Ihnen unter openUTM zur Verfügung stehen.

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	Set von Resource Managern öffnen

5.2.1 Besonderheiten bei TX-Aufrufen unter openUTM

tx_commit, tx_rollback

Wird der Aufruf von tx_commit bzw. tx_rollback veranlasst durch den Empfang des Wertes CM_TAKE_COMMIT_DEALLOCATE in *status_received*, erhält das CPI-C-Teilprogramm die Kontrolle nicht zurück. tx_commit() bzw. tx_rollback() wird intern auf den KDCS-Aufruf PEND abgebildet und das Server-Programm wird beendet.

Die Aufrufe tx_commit und tx_rollback sind nicht zulässig, wenn mit mindestens einem an der globalen Transaktion beteiligten Partner über das LU6.1-Protokoll kommuniziert wird.

tx_set_commit_return

Unter openUTM ist nur der Wert TX_COMMIT_COMPLETED zulässig.

Der Wert TX_COMMIT_DECISION_LOGGED wird mit TX_NOT_SUPPORTED zurückgewiesen.

tx_set_transaction_control

Unter openUTM ist nur der Wert TX_CHAINED zulässig.

Der Wert TX_UNCHAINED wird mit TX_PROTOCOL_ERROR zurückgewiesen.

tx_open

Liefert immer den Wert TX_OK zurück.

Nicht unterstützte Aufrufe

tx_begin

Der Aufruf wird mit TX_PROTOCOL_ERROR zurückgewiesen, da openUTM beim Start eines Service automatisch eine Transaktion eröffnet.

tx_close

Der Aufruf wird mit TX_PROTOCOL_ERROR zurückgewiesen, da der Aufruf nur im transaktionslosen Zustand erlaubt ist und unter openUTM immer eine Transaktion geöffnet ist.

5.3 Zusammenarbeit mit der CPI-C-Schnittstelle

In CPI-C-Teilprogrammen wird die Aufnahme eines Service in eine globale Transaktion über die Conversation-Charakteristik *Sync_Level* gesteuert. Hat *Sync_Level* den Wert CM_SYNC_POINT oder CM_SYNC_POINT_NO_CONFIRM, wird der Service in die globale Transaktion eingeschlossen.

Das Transaktionsende wird bei Verwendung der CPI-C-Schnittstelle immer vom Client angefordert.

Einen Vorgangswiederanlauf nach Störungen oder Systemausfällen, wie er an der KDCS-Schnittstelle durchgeführt wird, gibt es an der CPI-C-Schnittstelle nicht, da CPI-C keinen Vorgang kennt.

Unter openUTM sind zwei Fälle für die Funktionalität der TX-Aufrufe in CPI-C-Services zu unterscheiden:

Fall 1: Der Initiator schließt den Akzeptor in eine globale Transaktion ein

Schließt ein Client (Initiator) einen unter openUTM laufenden CPI-C-Service in eine globale Transaktion ein, haben die TX-Aufrufe im unter openUTM laufenden CPI-C-Server-Programm bestätigende Funktionalität.

Der Akzeptor (Server) erhält in diesem Fall beim Aufruf Extract_Sync_Level den Wert CM_SYNC_POINT oder CM_SYNC_POINT_NO_CONFIRM. Beim Receive-Aufruf wird über den im Feld *status_received* vom Client gelieferten Wert gesteuert, ob ein tx_commit() oder tx_rollback() im aufgerufenen Service verlangt wird.

Fall 1 tritt ein, wenn der Client

– eine UTM-Anwendung ist, die die Verbindung über das OSI-TP-Protokoll aufbaut und die *Functional Unit* COMMIT ausgewählt hat

– eine OpenCPIC-Anwendung ist und die Conversation mit *Sync_Level* CM_SYNC_POINT oder CM_SYNC_POINT_NO_CONFIRM aufbaut

– eine Fremdanwendung ist, die die CPI-C-Schnittstelle verwendet und die Conversation mit *Sync_Level* CM_SYNC_POINT oder CM_SYNC_POINT_NO_CONFIRM aufbaut

Fall 2: Der Initiator schließt den Akzeptor nicht in eine globale Transaktion ein

Schließ ein Client (Initiator) einen unter openUTM laufenden CPI-C-Service nicht in eine globale Transaktion ein, wird das aufgerufene Server-Programm implizit zur Wurzel einer globalen Transaktion. Die TX-Aufrufe haben dann anfordernde Funktionalität und steuern die globale Transaktion.

Der Akzeptor (Server) erhält in diesem Fall beim Aufruf Extract_Sync_Level den Wert CM_NONE oder CM_CONFIRM. Mit den Aufrufen tx_commit() und tx_rollback() im Server-Programm wird gesteuert, ob die globale Transaktion erfolgreich abgeschlossen oder zurückgesetzt wird.

Fall 2 tritt ein, wenn der Client

– eine UTM-Anwendung ist, die die Verbindung über das OSI-TP-Protokoll aufbaut und die *Functional Unit* COMMIT nicht ausgewählt hat

– eine OpenCPIC-Anwendung ist und die Conversation mit *Sync_Level* CM_NONE oder CM_CONFIRM aufbaut

– eine UPIC-Anwendung ist

– eine Fremdanwendung ist, die die CPI-C-Schnittstelle verwendet und die Conversation mit *Sync_Level* CM_NONE oder CM_CONFIRM aufbaut

Weitere Informationen zur Zusammenarbeit der TX- und der CPI-C-Schnittstelle finden Sie in der X/Open-Spezifikation „Distributed Transaction Processing: The CPI-C Specification, Version 2“ im Abschnitt "Effects of Calls on Half-Duplex Conversations to X/Open TX Interface".

Im Kapitel "Program-to-Program Communication Tutorial" der X/Open-Spezifikation finden Sie außerdem zwei Beispiel-Szenarien für die Verwendung der TX-Aufrufe in CPI-C-Programmen, die dem Ablauf unter openUTM entsprechen: "Sending Program Issues a Commit" und "Two Chained Transactions".
In diesen Beispiel-Szenarien ist openUTM dabei immer "System Y", also der Server.

Die Funktionalität der TX-Schnittstelle in openUTM-Client-Programmen ist im Handbuch openUTM Client V4.0, Trägersystem OpenCPIC beschrieben.

5.4 Zusammenarbeit mit der XATMI-Schnittstelle

TX-Aufrufe werden zur Transaktionssteuerung in XATMI-Services unter openUTM nicht benötigt.

Die Aufnahme eines XATMI-Services in eine globale Transaktion wird über den Aufrufparameter *flag* (in C) bzw. das Feld TPTRAN-FLAG (in COBOL) festgelegt.

Mit dem XATMI-Aufruf tpreturn() wird gesteuert, ob eine Transaktion erfolgreich beendet oder zurückgesetzt wird, TX-Aufrufe werden nicht benötigt.
Abhängig davon, ob im Parameter *rval* der Wert TPSUCCESS bzw. TPFAIL zurückgeliefert wird, wird die Transaktion erfolgreich beendet oder zurückgesetzt.

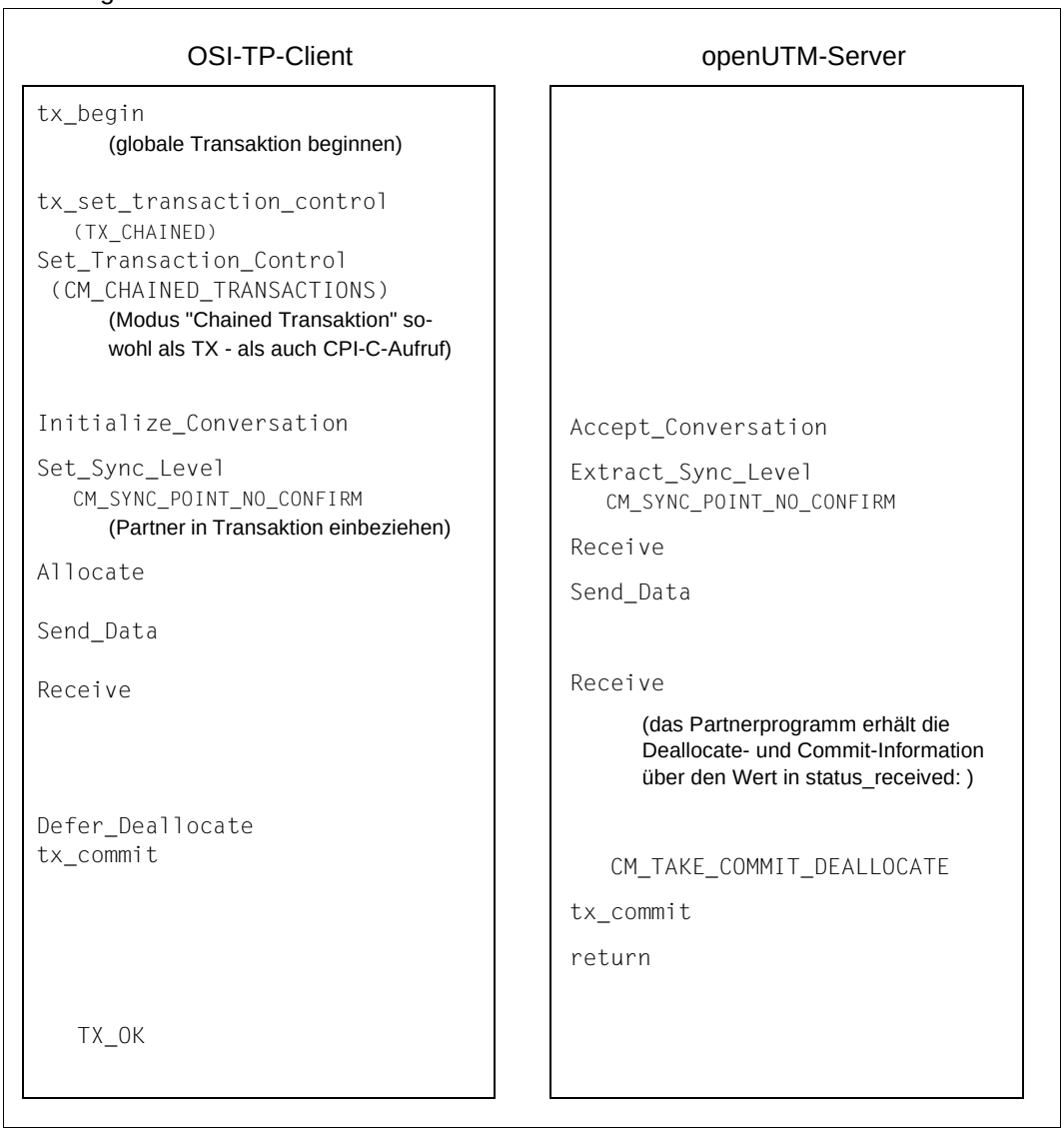
Für die XATMI-C-Schnittstelle ist die Aufnahme des Service in die globale Transaktion der Standardwert. Soll der Service nicht in die globale Transaktion aufgenommen werden, muss explizit das Flag TPNOTRAN gesetzt werden.
Für die XATMI-COBOL-Schnittstelle gibt es keinen Standardwert, entweder TPTRAN oder TPNOTRAN muss gesetzt werden.

Weitere Informationen zur Zusammenarbeit der TX- und der XATMI-Schnittstelle finden Sie in der X/Open-Spezifikation „Distributed Transaction Processing: The XATMI Specification“ im Abschnitt „Transaction Functions Affecting the XATMI Interface“.

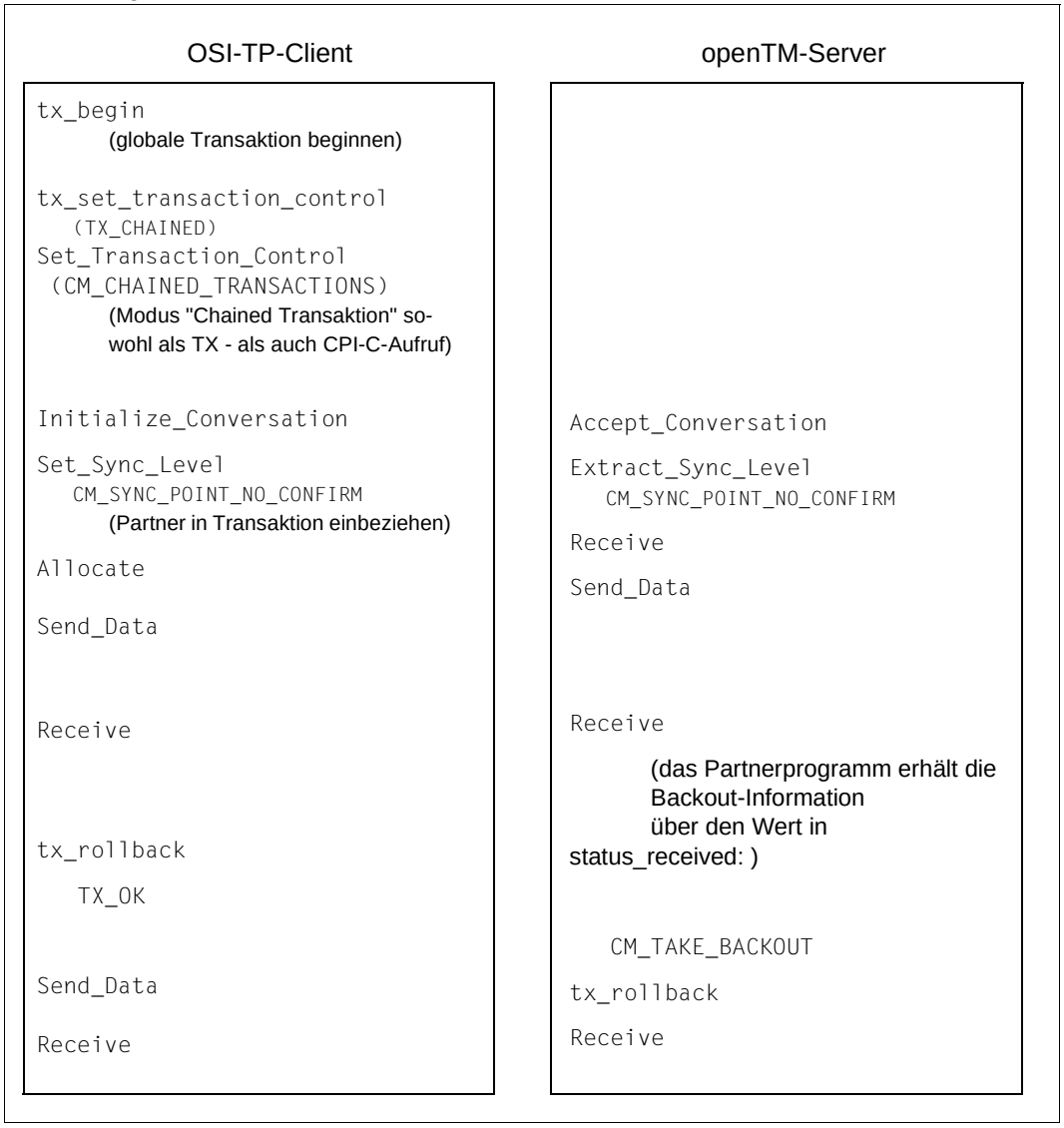
5.5 Beispiele für die Verwendung der TX-Schnittstelle

Die beiden folgenden Ablaufdiagramme zeigen schematisch, wie die CPI-C- und TX-Aufrufe aufeinander abgestimmt sein müssen, wenn der Client den aufgerufenen Service in eine globale Transaktion einbindet.

1. Server-Programm wird in die globale Transaktion eingeschlossen, die Transaktion wird erfolgreich beendet



2. Server-Programm wird in die globale Transaktion eingeschlossen, die Transaktion wird zurückgesetzt



Hinweis:

Bei Mehrschritt-Transaktionen sollten Sie nach jedem TX-Aufruf zur Transaktionssteuerung den Status der Conversation abfragen, mit der Sie weiter kommunizieren wollen (Extract_Conversation_State). Anhand des zurückgelieferten Status können Sie erkennen, ob Sie im Zustand *Send* oder *Receive* sind. Entsprechend setzen Sie den Programmablauf mit einem Receive- oder Send-Aufruf fort.
Durch diese Abfrage vermeiden Sie CM_STATE_CHECK-Fehler, die durch eine falsche Send-Receive-Aufrufreihenfolge hervorgerufen werden.

5.6 Erstellen einer Anwendung mit TX-Aufrufen

Für das Erstellen von Programmen mit TX-Aufrufen in C wird mit openUTM eine Include-Datei ausgeliefert, für COBOL werden COPY-Elemente ausgeliefert.

X/W X/W **Dateien und Bibliotheken für die TX-Schnittstelle unter Unix- und Windows-Systemen**

X/W
- Für C wird die Include-Datei `tx.h` ausgeliefert. Diese Datei liegt unter:
  - X – *utmpfadl*`tx/include/tx.h` (Unix-Systeme)
  - W – *utmpfad*`\tx\include\tx.h` (Windows-Systeme)

X/W X/W
- Für COBOL werden die Copy-Elemente `TXSTATUS` und `TXINFDEF` ausgeliefert. Diese Dateien liegen unter
  - X – *utmpfadl*`tx/copy-cobol85` bzw. *utmpfad*`/tx/netcobol` (Unix-Systeme)
  - W – *utmpfad*`\tx\copy-cobol85` bzw. *utmpfad*`\tx\netcobol` (Windows-Systeme)

X X
- Zum Binden der Programme auf Unix-Systemen müssen Sie folgende Bibliothek verwenden:
  - X – *utmpfadl*`sys/libxopen`

B **Dateien und Bibliotheken für die TX-Schnittstelle unter BS2000-Systemen**

B B Unter BS2000-Systemen wird die Bibliothek $*userid*.SYSLIB.UTM.063.XOPEN ausgeliefert.

B Diese Bibliothek müssen Sie zum Binden der Programme verwenden.

B B Als Elemente vom Typ S finden Sie in dieser Bibliothek auch die Include-Datei für C (TX.H) und die COPY-Elemente für COBOL (TXSTATUS und TXINFDEF).

Generieren einer Anwendung mit TX-Aufrufen

Für die Verwendung der TX-Schnittstelle ist keine spezielle Generierung erforderlich.

TX-Aufrufe sind unter openUTM nur in CPI-C-Programmen sinnvoll. Die Generierung von CPI-C-Anwendungen ist im Abschnitt „Generieren einer CPI-C-Anwendung“ auf Seite 96 beschrieben.

5.7 Fehlerdiagnose bei TX-Aufrufen

TX-Aufrufe werden auf KDCS-Aufrufe abgebildet. Der UTM-Dump enthält deshalb nur die KDCS-Aufrufe. Sie können für die Diagnose von Fehlern bei der Transaktionssteuerung jedoch zusätzlich zum UTM-Dump einen Trace der TX-Aufrufe erzeugen.

Die TX-Bibliothek gibt keine Meldungen aus.

5.7.1 Steuerung des Trace

Der TX-Trace kann wie folgt gesteuert werden:

- Durch den UTM-Startparameter TX-TRACE kann der Trace beim Start der Anwendung eingeschaltet werden, siehe jeweiliges openUTM-Handbuch „Einsatz von openUTM-Anwendungen“.
- Über WinAdmin oder WebAdmin kann der Trace im laufenden Betrieb ein- und ausgeschaltet werden. Dazu verwenden Sie den Dialog *Eigenschaften der UTM-Anwendung*, Registerkarte *Diagnose und Abrechnung*, Feld *TX Trace*.
- Über die Programmschnittstelle zur Administration KDCADMI kann der Trace im laufenden Betrieb ein- und ausgeschaltet werden. Dazu dient das Feld *tx_trace* in der Datenstruktur *kc_diag_and_account_par_str*, siehe openUTM-Handbuch „Anwendungen administrieren“.

Sie können folgende Trace-Level einstellen:

Level	**Bedeutung**
ERROR	Es werden nur Fehler protokolliert.
INTERFACE	Umfasst den Level ERROR, zusätzlich werden alle TX-Aufrufe protokolliert.
FULL	Umfasst den Level INTERFACE, zusätzlich werden alle KDCS-Aufrufe, auf die die TX-Aufrufe abgebildet werden, protokolliert.
DEBUG	Umfasst den Level FULL, zusätzlich werden Diagnose-Informationen protokolliert.
OFF	Trace ist ausgeschaltet.

5.7.2 Name der Trace-Datei

X/W **Trace-Datei unter Unix- und Windows-Systemen**

X/W Die Trace-Sätze werden in die Datei KDC.TRC.TX.*appliname.hostname.pid* im Verzeichnis
X/W *filebase* geschrieben. Dabei bedeuten:

X/W appliname
X/W Name der Anwendung

X/W hostname
X/W Name des Rechners, auf dem die Anwendung läuft

X/W pid PID des Prozesses

B **Trace-Datei unter BS2000-Systemen**

B Die Trace-Sätze werden standardmäßig in die Datei KDC.TRC.TX.*appliname.hostname.tsn*
B geschrieben. Dabei bedeuten:

B appliname
B Name der Anwendung

B hostname
B Name des Rechners, auf dem die Anwendung läuft

B tsn TSN der UTM-Task

B Sie können auch in der UTM-Startprozedur für jede Task eine andere Trace-Datei
B einrichten und mit dem Kommando SET-FILE-LINK den Linknamen KDCTX zuweisen.

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

Z