D I P L O M A R B E I T

INSTITUT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Entwurf und Implementierung eines Objektmodells für das Management von TP-Monitoren

Bernhard Fischer

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering

Betreuer: Alexander Keller
Dr. Bernhard Neumair
Dr. Peter Segner

D I P L O M A R B E I T

INSTITUT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Entwurf und Implementierung eines Objektmodells für das Management von TP-Monitoren

Bearbeiter:	Bernhard Fischer
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Alexander Keller Dr. Bernhard Neumair Dr. Peter Segner
Abgabedatum:	15. November 1996

Erklärung

Hiermit versichere ich, daß ich die Diplomarbeit selbständig verfaßt, und nur die hier angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. November 1996 ....

Zusammenfassung

Durch die Integration von Großrechensystemen in heterogene Client/Server-Architekturen werden Transaktionsmonitore nicht mehr nur auf Mainframes eingesetzt, sondern in zunehmendem Maße auch auf Workstation- und PC-Systemen. Die Vielfalt der Systeme erhöht die Komplexität des Managements von Transaktionsmonitoren, wodurch ein integriertes Management dieser Middleware-Anwendungen immer mehr an Bedeutung gewinnt. In dieser Diplomarbeit wurde ein Klassenmodell zum integrierten Management von Transaktionsmonitoren entworfen und implementiert. Da Transaktionsmonitore der Kategorie Middleware zugeordnet werden, ist diese Arbeit dem System- und Anwendungs-Management zuzuordnen.
Online-Transaction-Processing-Systeme (OLTP-Systeme) sind hochkomplexe Anwendungen, die nicht nur das Management verteilter Anwendungen übernehmen, sondern seit einigen Jahren auch selbst als verteilte Anwendung realisiert sind. Diese Verteilung bringt es mit sich, daß Komponenten eines TP-Monitors auf verschiedenen Plattformen Dienste zur Verfügung stellen. Das Management eines Transaktionsmonitors und der darunter arbeitenden Anwendungen erfordert daher eine Integration verschiedener Plattformen.
Die Komplexität bekommt durch den Einsatz verschiedener Produkte, die bei einem Betreiber oftmals parallel eingesetzt werden, eine weitere Dimension. Nur wenige Betreiber können es sich jedoch leisten, für jedes Produkt eine Mannschaft von Spezialisten zu beschäftigen. Oft hat ein Betreiber einen Benutzerservice oder eine Hotline, wo alle Probleme eingehen und eine Gruppe von Administratoren das Management des DV-Betriebes übernimmt. Für diese Administratoren ist ein integriertes Management über verschiedene Produkte hinweg eine große Erleichterung.
Auf Grundlage des bisher einzigen Management-Agenten für Transaktionsmonitore, der über einen CORBA-konformen Object-Request-Broker die Management-Information und -Funktionalität aus einem TP-Monitor einem Manager zur Verfügung stellen kann, wurde ein anwendungsbezogenens Objektmodell entwickelt. Dieses Objektmodell wurde anschließend auf einem IBM AIX-System prototypisch implementiert. Exemplarisch für andere OLTP-Systeme diente das Customer Information Control System (CICS) von IBM für die Arbeit als Grundlage.
Der Prototyp des Managers wurde auf einem CICS/6000-System, das auf einer IBM RS6000 mit dem Betriebssystem AIX installiert ist, implementiert.

Inhaltsverzeichnis

Kapitel 1

Einleitung

'EDV ohne Transaktionsverarbeitung ist wie ein Staat ohne Gesetze: Nur solange alles bestens funktioniert, kann man darauf verzichten.' [1] Wo viele Benutzer gleichzeitig mit verschiedenen Anwendungen auf eine gemeinsame Datenbasis zugreifen, gewährleisten On-Line-Transaction-Processing-Systeme (OLTP-Systeme) die Konsistenz und Integrität der Daten während und nach jeder einzelnen Transaktion. Der Haupteinsatzbereich von Transaktionsmonitoren ist in geschäftskritischen Anwendungen (Business Critical Applications), das heißt, bei einem Ausfall dieser Anwendung würde der gesamte Geschäftsbetrieb zum Erliegen kommen. Geschäftskritische Anwendungen stellen damit hohe Anforderungen an Verfügbarkeit und Sicherheit von Daten und deren Verarbeitung. In diesem Bereich ist es äußerst wichtig, daß die vernetzten Rechnersysteme und die auf ihnen gespeicherten Daten jederzeit verfügbar sind. Nichts darf verlorengehen, nichts darf verfälscht werden und nichts darf von Unbefugten gesehen oder gar verändert werden. Zunehmend werden Mainframes in neue Client/Server-Lösungen integriert oder von ihnen abgelöst. Dadurch entstehen neue Anforderungen an die Kommunikations-Schnittstellen zwischen den jeweiligen Architekturen. Trotzdem müssen Transaktionen sicher, performant und zuverlässig in verteilter Umgebung durch das gesamte Netz über mehrere beteiligte Rechner geroutet werden. Die Transaktionsverarbeitung hat sich deshalb zu einer heterogenen Anwendung entwickelt, wodurch neue Formen der Administration notwendig werden, um Systeme auf verschiedenen Plattformen rationell und sicher zu verwalten. Hier soll diese Diplomarbeit ansetzen und ein Modell für ein Management-Werkzeug entwickeln, mit dem Transaktionsmonitore in heterogener, verteilter Umgebung administriert werden können.

1.1 Was ist ein TP-Monitor und wozu wird er verwendet?

Ein TP-Monitor ist ein Steuerprogramm zur transaktionsorientierten Verwaltung von Anwendungsprogrammen für die **Online-Dialogverarbeitung** (**OLTP**) und für die **Stapelverarbeitung** (**Batchprocessing**). Die Funktionen eines TP-Monitors erleichtern dem Anwender das Erstellen und Betreiben komplexer Anwendungen in einem hohen Maß.
Sie unterstützen unter anderem:

- das effiziente Management der benötigten Ressourcen
- das effiziente Management von Aufträgen (z.B. Priorisierung, Überwachung)
- verteilte Anwendungen (Client-Server-Architekturen, verteilte Transaktionsverarbeitung)
- transaktionsgesicherte Asynchronverarbeitung/Hintergrundverarbeitung
- transaktionsgeschützte Dateisysteme
  (zum Beispiel der **Structured File Server** (**SFS**) von **ENCINA**))

[1] aus Gray, Jim, Reuter, Andreas;
Transaction Processing: Concepts and Techniques,
Morgan Kaufmann Publishers, San Mateo,
California 1993

- Zugangsschutz, Zugriffsschutz auf bestimmte Programme für bestimmte Benutzer
- transaktionsgeschützte Zugriffe auf Datenbanken

Er umfaßt Funktionen für dynamisches Laden von Programmen, Betriebsmittel-Kontrolle, Zwischenspeicherung, Datenverwaltung, Datenbank-Anschluß und Interprogramm-Kommunikation. Der Einsatz eines TP-Monitors erhöht den Durchsatz auf einem Server-Rechner bei gleichzeitiger Verringerung des Betriebsmittelverbrauchs. Der **TP-Monitor** ist ein Zusatz zum Betriebssystem und bildet eine Zwischenschicht unterhalb der Anwendung. Deshalb werden Transaktionsmonitore auch als **Middleware** bezeichnet. Im Allgemeinen bezeichnet man als Middleware die Software, die zwischen dem Betriebssystem oder auch Netz-Betriebssystem und den eigentlichen Anwendungen liegt. Middleware isoliert damit die Anwendungen von Plattform-Abhängigkeiten.

Andere Beispiele hierfür sind:

- Mail-Service
- Kommunikations-Dienste (zum Beispiel ein Object Request Broker (ORB))
- Fenster-Systeme (zum Beispiel X-Windows-Manager)
- Multimedia-Systeme
- Datenhaltung

Ein TP-Monitor führt außerdem alle Zugriffe auf Dateien und Terminals sowie Betriebssystem-Dienste für die Anwendung durch und sorgt dabei für Datenintegrität und Datenschutz im technischen Sinne.

Da OLTP-Anwendungen häufig für die Anwender extrem wichtige Daten bearbeiten, besteht in solchen Anwendungen die Forderung nach einer ständigen Konsistenz der Daten. Diese Eigenschaften werden als **ACID-Eigenschaften** bezeichnet (**ACID** = atomicity, consistency, isolation, durability). In einer TP-Monitor-Umgebung gelten diese Anforderungen nicht nur für die Daten der angeschlossenen Datenbanken, sondern auch für die Informationen (Ressourcen), die der TP-Monitor verwaltet, zum Beispiel Asynchronaufträge. Es müssen daher die Daten der Datenbanken mit den Informationen des TP-Monitors koordiniert werden.

Früher wurden Transaktionsmonitore in erster Linie für die Realisierung von Datenbankzugriffen verwendet, um die Konsistenz des Datenbestandes gewährleisten zu können.

Durch die fortschreitende Vernetzung von Computersystemen müssen Softwarekomponenten auf heterogenen Rechnerumgebungen unterschiedlicher Hersteller zusammenarbeiten. Deshalb werden heute TP-Monitore aufgrund ihrer Eigenschaften ebenso für verteilte Anwendungen, Parallelverarbeitung und im Client/Server-Bereich verwendet, wobei die Remote-Procedure-Call-ähnlichen Mechanismen ausgenutzt werden. Bei der verteilten Transaktionsverarbeitung greifen zwei Anwendungen über einen Transaktionsmonitor gleichzeitig auf den selben Datenbestand zu. In solchen Anwendungen müssen Transaktionen abgearbeitet werden, die sich über ein Netz hinweg erstrecken. Der Anwender verteilt den Datenbestand flexibel und erreicht dadurch eine geringere Datenredundanz im Netz. Sicherheitsrelevante Daten konzentriert er auf den Server und kann sich so bei der Datensicherung auf diesen Rechner beschränken.

Im Falle einer verteilten Anwendung oder verteilter Transaktionsverarbeitung sind die Anwendungsprogramme auf verschiedene Rechner verteilt, das heißt der Entwickler muß über geeignete Programmschnittstellen die Kommunikation zwischen den Anwendungsprogrammen realisieren und dafür sorgen, daß die verschiedenen Programme (auch Versionen der Programme) aufeinander abgestimmt sind. Diese Form der Verarbeitung bietet sich an, wenn ein Auftrag, zum Beispiel eine feste Buchung, an einen Server geschickt wird. Dort werden dann alle Zugriffe auf die Datenbank durchgeführt. Bei einer solchen Art der Verarbeitung wird die Menge des Datentransfers über das Netz minimiert. Beim entfernten Datenzugriff wird von einem Anwendungsprogramm auf eine Datenbank zugegriffen, die auf einem entfernten Rechner (Server) liegt. Dadurch bezieht der Anwender die Daten entfernter Datenbanken in die lokale Verarbeitung mit ein. Bei den verteilten Datenbanken sind die Daten über mehrere Rechner verteilt. Der Anwender verteilt den Datenbestand auf die Rechner, an denen die Daten vorrangig benötigt werden. Den Anwendungen bieten verteilte Datenbanken von jedem Punkt im Netz transparenten Zugriff auf den gesamten

Datenbestand. Darüberhinaus werden TP-Monitore in Bereichen verwendet, wo eine plattformübergreifende Intersystem-Kommunikation bzw. Interprozeßkommunikation benötigt wird. Ein weiterer Vorteil, der gerade heute nicht zu unterschätzen ist, liegt in der Portabilität der Anwendungen, wenn diese unter einem TP-Monitor realisiert wurden. Auf jeder Plattform, die vom TP-Monitor unterstützt wird und auf der ein entsprechender Compiler für den Programmcode zur Verfügung steht, kann eine Anwendung mit relativ geringem Portierungsaufwand übernommen werden. Dies setzt jedoch voraus, daß die Anwendung ohne direkte Abhängigkeiten von der Plattform oberhalb des Transaktionsmonitors programmiert wurde.

1.2 Umfeld der Diplomarbeit

Aus der Fülle von Transaktions-Monitoren ein geeignetes Produkt auszuwählen, an dem, exemplarisch für andere, die Problematik des Management von TP-Monitoren herausgearbeitet werden kann, hängt von verschiedenen Kriterien ab:

- Verfügbarkeit auf verschiedenen, gebräuchlichen Betriebssystemen
- Ausreichend große Installationsbasis, um genügend Erfahrungen aus der Praxis in die Arbeit einfließen zu lassen und die Arbeit nicht auf Spezialitäten eines Exoten aufzubauen
- Zugreifbarkeit auf Informationen aus der Praxis über Administratoren
- Unterstützung des Client/Server-Konzeptes

Es wäre wünschenswert, ein modernes System zu Verfügung zu haben, jedoch sind die Neuentwicklungen (zum Beispiel Tuxedo oder Encina) nur für offene Systeme verfügbar. Ein Großteil der Transaktionsverarbeitung findet aber nach wie vor auf Großrechensystemen, wie zum Beispiel IBM MVS/ESA oder Siemens BS2000, statt. Hier ist auch das größte Expertenwissen bezüglich Installation, Konfiguration und Kommunikation mit Fremdsystemen vorhanden. Nachdem sich bei den Vorarbeiten eine Zusammenarbeit mit den Entwicklungslabors von IBM in Hursley/UK abzeichnete und mehrere Spezialisten für Großrechen-Systeme und AIX-Systeme für die Dauer der Diplomarbeit zur Verfügung standen, fiel letztendlich die Wahl auf das Customer Information Control System von IBM, kurz CICS genannt. In einem späteren Kapitel wird noch genauer auf dieses und andere Systeme eingegangen, das Customer Information Control System (CICS) von IBM läßt sich jedoch mit einigen Eckwerten charakterisieren:

- CICS ist der weltweit führende Transaktionsmonitor, man kann ihn als de facto Standard bezeichnen.
- Es verfügt über eine weltweite Installationsbasis von ca. 70.000 Servern.
- Derzeit sind weltweit rund 500.000 erfahrene CICS Anwendungsentwickler beschäftigt.
- Investitionen von über 1 Milliarde US-Dollar stecken in CICS basierten Anwendungen.
- Verfügbarkeit auf vielen verbreiteten Systemen:
  Die CICS-Server sind auf allen wichtigen Betriebssystemen für Großrechner und Minicomputer verfügbar, wie zum Beispiel BS2000, MVS, OS/400, HP-UX und AIX, um nur einige zu nennen. Die Clients laufen unter den meisten Microcomputer- und PC-Betriebssystemen.

1.3 Infrastruktur für die Erstellung der Diplomarbeit

Die msg systeme gmbh, in der weiteren Arbeit kurz 'msg' genannt, stellt für die Dauer der Diplomarbeit einen Arbeitsplatz zu Verfügung, von dem aus die wichtigsten Plattformen erreichbar sind. Die msg besitzt unter anderem IBM AS/400, IBM RS6000, SUNserver 1000, SUN SPARCclassic und einem IBM-Server mit MVS, außerdem ist sie über Standleitungen mit verschiedenen IBM Großsystemen verbunden. Im Bereich der PC-Systeme sind MS-Windows-, OS/2-, Win95-, WinNT- und MS-DOS-Rechner verfügbar und mit CICS-Clients ausgerüstet oder können nachgerüstet werden. Als Entwicklungsrechner für die Diplomarbeit dient ein RS6000-System vom Typ J30 der Firma IBM, das mit der AIX Version 4.1 ausgerüstet ist. Für die computergestützte Objektmodellierung steht die Anwendung StP (Software through Pictures) von IDE zur Verfügung, die auch am Leibniz Rechenzentrum verwendet wird und mit der schon früher Diplomarbeiten am Lehrstuhl für Rechnernetze- und Kommunikation implementiert wurden. Außerdem stehen C++-Compiler und SOM/DSOM-Compiler auf verschiedenen Plattformen zur Verfügung.

1.4 Vorgehensweise zur Gewinnung des Modells

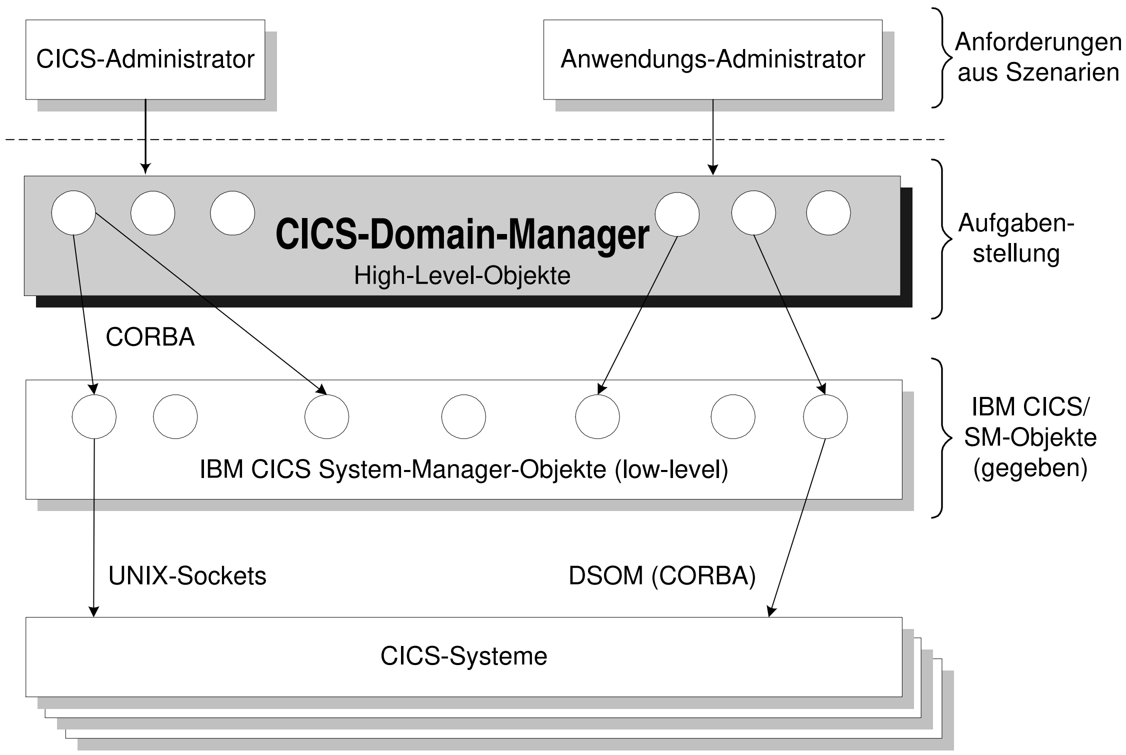

Abbildung 1.1: Vorgehensweise der Arbeit

Auf Grundlage von Szenarien, die aus den Bereichen CICS- und Anwendungs-Administration des Rechenzentrums der BMW AG und des Gerling Konzerns kommen, werden Betreiberanforderungen formuliert, die auf den häufigsten Tätigkeiten eines Systemverwalters basieren.

Aus den Anforderungen, die in der obersten Schicht der Abbildung 1.1 skizziert sind, werden die Objektklassen für das Management entwickelt (grau unterlegter Bereich). Diese Klassen sollen die managementrelevanten Informationen und Methoden enthalten, die notwendig sind, um den Anforderungen aus den Szenarien gerecht zu werden. Mit den Anforderungen wird ein großer Bereich der Managementaufgaben abgedeckt sein. Es wird jedoch in Kauf genommen, daß nicht alle Aufgaben des Anwendungsmanagement gelöst werden können. Dadurch kann weniger wichtige Information und Funktionalität vorerst ausgefiltert werden. Das durch diese Vereinfachung,wesentlich schlankere Modell erhöht die Übersichtlichkeit und verringert später den Ballast an selten benötigten Klassen.

In einem weiteren Schritt werden bereits vorhandene Produkte für die Administration auf wiederverwendbare Komponenten und Konzepte hin untersucht. Im Vordergrund wird dabei das Produkt *System Ma-*

nager der Firma IBM stehen. Mit dieser Management-Anwendung wird eine erste Implementierung eines Management-Agenten für Transaktionsmonitore mitgeliefert, die über eine CORBA-konforme Schnittstelle, einem Object Request Broker (ORB), Management-Informationen zur Verfügung stellt. Die Objektklassen des System Manager (dritte Ebene der Abbildung 1.1) könnten zur Informationsermittlung für die anwendungsbezogenen Objekte dienen und eine Schicht zwischen dem CICS-Domain-Manager und den CICS-Systemen (unterste Ebene).

Kapitel 2

Grundlagen zu TP-Monitoren

Um einen einheitlichen Wortschatz für dieses Thema zugrunde legen zu können, werden zu Anfang die wichtigsten Begriffe eingeführt und erläutert, die zum Verständnis der Arbeitsweise von Transaktionsmonitoren hilfreich sind.

Ebenso werden die wichtigsten Standards angesprochen, die in der Transaktionsverarbeitung Anwendung finden. Um nicht zu weit ins Detail zu gehen, werden nur die Standards genannt, die in den vorgestellten Transaktionsmonitoren implementiert sind oder für das Management von TP-Monitoren eine Rolle spielen.

Nach einer kurzen Vorstellung heute bekannter Transaktionsmonitore wird auf ein Produkt der Firma IBM, das Customer Information Control System, genauer eingegangen, da dieses System dieser Arbeit zu einem großen Teil als Grundlage dient.

2.1 Allgemeine Terminologie

Eine **Transaktion** besteht aus einer Folge von logisch zusammengehörigen Operationen, ist also eine abgeschlossene Verabeitungseinheit, in der zusammengehörende Datenmanipulationen abgewickelt werden. Sie wird daher auch als **technischer Konsistenzbereich** bezeichnet. Alle in einer Transaktion vorzunehmenden Änderungen am Datenbestand der Anwendung werden entweder vollständig oder gar nicht ausgeführt, diese Regel heißt auch die 'Alles-oder-nichts-Regel'. Kann eine Transaktion aus irgendeinem Grund nicht vollständig durchgeführt werden, so wird die Transaktion zurückgesetzt, d.h. alle Daten, die von der Transaktion verändert wurden, werden auf den Zustand vor dem ersten Zugriff der Transaktion auf diese Daten zurückgesetzt. Eine Transaktion muß also bis zu ihrem Ende rücksetzbar sein. Die Arbeitsergebnisse einer Transaktion führen in der Regel von einem bestimmten Konsistenzzustand zu einem neuen, ebenfalls konsistenten Zustand. Eine Transaktion wird durch die vier sogenannten **ACID**-Prämissen charakterisiert:

- Eine Transaktion ist **Atomic**, das heißt, sie ist nicht weiter aufteilbar und muß verarbeitungstechnisch als eine Einheit gesehen werden. Das bedeutet, daß alle Verarbeitungs-Anweisungen einer Transaktion komplett abgearbeitet werden oder, zum Beispiel im Falle einer Unterbrechung oder eines Fehlers, wieder auf den Zustand vor Beginn der Verarbeitung zurückgesetzt werden müssen.

- Daten müssen sich vor und nach einer Transaktion in einem konsistenten Zustand befinden. Es müssen daher standardisierte und zentrale Maßnahmen zur Erhaltung der **Konsistenz** (**Consistence**) bereitgestellt werden. Sind keine ausreichenden Maßnahmen zur Gewährleistung der Konsistenz an zentraler Stelle vorhanden, so müssen die Daten von den Anwendungen selbst auf einen konsistenten Zustand geprüft werden.

- Während der Verarbeitung einer Transaktion können sich Ressourcen temporär in einem inkonsistenten Zustand befinden, daher ist eine **Isolation** der Transaktions-Ressourcen zwingend erforderlich. Solange eine Transaktion auf eine Ressource zugreift, darf kein paralleler Prozeß in konsistenzgefährdender Weise auf die gleiche Ressource zugreifen. Ist eine Transaktion noch nicht vollständig abgeschlossen, bzw. zurückgesetzt worden, dürfen andere Prozesse nicht einmal lesend auf die von ihr veränderten Ressourcen zugreifen, da sonst Zustände angezeigt werden könnten, die nie wirklich existiert haben. Zum Beispiel könnte ein Vertrag gerade in Erstellung sein, wobei Name und Adresse bereits erfaßt sind. In einem zweiten Schritt wird eine KFZ-Versicherung hinzugefügt. Greift nun ein anderer Prozeß lesend auf die Vertragsdaten zu, so existiert für ihn möglicherweise ein Vertrag ohne Versicherungsdaten. Das entspricht aber nicht der Realität, da es einen Vertrag ohne Versicherung nicht geben darf und die Vertragserstellung erst mit Eingabe der Versicherungsdaten abgeschlossen ist.

- Ist eine Transaktion abgeschlossen, so bleiben ihre Ergebnisse (z.B. Änderungen am Datenbestand, Erzeugen von Nachrichten) auch nach einem Abbruch der Anwendung erhalten, denn sie werden am Transaktionsende gesichert. Änderungen, die eine Transaktion an einer Ressource ausgeführt hat, müssen **dauerhaft** (**durable**) sein und solange bestehen bleiben, bis sie explizit rückgängig gemacht wurden.

Diese ACID-Prämissen erreicht man bei Transaktions-Monitoren durch ein immer wiederkehrendes Synchronisieren nach jedem Verarbeitungsschritt. Diese Synchronisation wird mit sogenannten Sicherungspunkten erreicht. Mit Sicherungspunkt bezeichnet man das Ende einer Transaktion, an dem der TP-Monitor alle in der Transaktion vorgenommenen Änderungen der Anwendungsdaten festschreibt. Diese Änderungen sind dann nicht mehr rücksetzbar, sie sind gegen Systemausfall gesichert und werden für andere Transaktionen sichtbar. Ein **Transaktions-Monitor** organisiert also Zugriffe auf Daten in einer Weise, daß für jede Transaktion der Eindruck entsteht, sie würde alleine auf diese Daten zugreifen. Parallel ablaufende Transaktionen, wie sie zum Beispiel beim **Online-Transaction-Processing** (**OLTP**) auftreten, müssen über einen **Scheduler** so organisiert (serialisiert) werden, daß ihre Zugriffe für jeweils andere Transaktionen keine inkonsistenten Daten verursachen. Gleichzeitig erhöht ein Transaktionsmonitor durch seine eigene Betriebsmittelverwaltung die Systemleistung obwohl der Betriebsmittelverbrauch geringer wird, weil er Dienste bietet, für die die meisten Betriebssysteme nicht ausgelegt sind.

2.2 Programm-Konzepte in der Transaktions-Verarbeitung

Dialogorientiertes Programmkonzept: (conversational processing)

Es gibt zwei Formen von Dialogbetrieb, den Teilnehmer- und den Teilhaberbetrieb. Im **Teilnehmerbetrieb** arbeitet der Datenstationsanwender im Dialog mit dem Verarbeitungsrechner so, als stünde dieser ihm allein zur Verfügung. Er kann sämtliche Dienste und die Kommandosprache des Betriebssystems nutzen, und mit allen Anwendungsprogrammen arbeiten, für die er eine Berechtigung hat. Diese Form des Dialogbetriebs ist nur für bestimmte Aufgaben sinnvoll, z.B. für Programmentwicklung, Programmtest sowie das Aufbereiten von Daten auf die nicht gleichzeitig von verschiedenen Anwendern zugegriffen werden muß. Arbeiten viele Datenstationsanwender im Teilnehmerbetrieb, steigen Verwaltungsaufwand und Betriebsmittelbedarf des Systems stark an.

Eine bessere Möglichkeit der Betriebsmittelnutzung ist der Teilhaberbetrieb. Im **Teilhaberbetrieb** schließen sich die Datenstationsanwender an eine bereits gestartete Anwendung an und führen mit ihr einen Dialog. Bei dieser Betriebsart können nahezu beliebig viele Datenstationsanwender gleichzeitig ihre Aufgaben abwickeln. Über die Datensichtstation können nur ganz bestimmte vordefinierte Aufgaben an eine Anwendung gestellt werden. Diese Aufgaben werden durch Programme ausgeführt, die der Betreiber der Anwendung bereitgestellt hat. Dabei sind nur Eingaben möglich, die das Anwendungsprogramm anfordert bzw. zuläßt. Die Kommandosprache des Betriebssystems kann im Teilhaberbetrieb nicht verwendet werden. Die Steuerung des Dialogs liegt bei der Transaktionsmonitor-Anwendung.

Quasi-dialogorientiertes Programmkonzept: (pseudo-conversational processing)

Sämtliche Schritte eines Vorgangs können durch Vorgabe eines einzigen Transaktions-Identifikators durch den Anwender aktiviert werden. So stehen sämtliche Transaktionen eines Vorgangs untereinander durch Vorgabe des jeweils nächsten Transaktions-Identifikators in Verbindung. Dadurch können alle bereits eingegebenen oder verarbeiteten Daten von einem Dialogschritt zum nächsten übergeben

werden.
Der Vorteil gegenüber dem rein dialogorientierten Konzept besteht in der wesentlich kürzeren Verweildauer von Transaktionen im System, weil während der Bedenk- und Eingabezeiten des Anwenders keine Transaktion im System verbleibt. Nach der Anzeige einer Eingabemaske wird diese Transaktion beendet und läuft erst beim Absenden der Daten wieder an, verarbeitet die Eingabedaten und startet die Nachfolge-Transaktion. Die Transaktionen stehen dabei über den Transaktionskontext miteinander in Verbindung.

Transaktionsorientiertes Programmkonzept: (transactional processing)
Die Betriebsart, bei der Aufträge in einer oder mehreren Transaktionen bearbeitet werden, wird auch Transaktionsbetrieb genannt. Dabei werden die Aufträge von den Kommunikationspartnern der Anwendung erteilt, das können Datenstationsanwender sein, aber auch andere Anwendungen. Jeder Dialogschritt wird in einer eigenen Transaktion abgewickelt, das heißt eine Transaktion empfängt eine Eingabenachricht des Anwenders, verarbeitet die Anforderungen und gibt eine Ausgabenachricht aus. Damit ist diese Transaktion beendet und sie hat zu ihrer Vorgänger- und Nachfolgetransaktion keinerlei Beziehungen.

Verteilte Transaktionsverarbeitung: (distributed transaction processing)
Bei verteilter Transaktionsverarbeitung sind an der Bearbeitung eines Auftrags mindestens zwei Transaktionen in verschiedenen Anwendungen beteiligt. Das Ende dieser Transaktionen wird vom TP-Monitor synchronisiert, das heißt beide Anwendungen setzen gleichzeitig Sicherungspunkte. Im Fehlerfall sorgt der Monitor dafür, daß alle beteiligten Transaktionen zurückgesetzt werden. Die synchronisierten Transaktionen bilden also eine Einheit, die auch als verteilte Transaktion bezeichnet wird. Zur Bildung verteilter Transaktionen wird das sogenannte two-phase-commit-Verfahren [1] verwendet. Für das Anwendungsprogramm ist dieses Verfahren transparent, allerdings muß es hinsichtlich der Beendigung seiner Transaktion den Zustand seiner Kooperationspartner beachten.

Stapelverarbeitung: (batch-job processing)
Der Anwender kommuniziert mit dem System über feste Schnittstellen in Form von Eingabebelegen und Ausgabelisten. Es werden zunächst Daten mittels Eingabebelegen erfasst und zu einem späteren Zeitpunkt, bevorzugt in lastarmen Zeiten, verarbeitet. Das Ergebnis wird in Ausgabelisten abgelegt. Batch-Verarbeitung steht im Gegensatz zu den dialogorientierten Konzepten und wird meist für die Verarbeitung von Massendaten verwendet.

2.3 Die wichtigsten Standards aus dem Umfeld der Diplomarbeit

Diese Aufzählung erhebt keinen Anspruch auf Vollständigkeit, soll aber einen Überblick über die am häufigsten verwendeten Standards und Architekturmodelle geben, die für die Transaktionsverarbeitung und Kommunikation zwischen TP-Monitoren von Bedeutung sind.

Der CORBA-Standard der Object Management Group (OMG) wurde in diese Liste mit aufgenommen, obwohl sich dieser Standard in der Transaktionsverarbeitung noch nicht etablieren konnte. Es gibt zwar Ansätze, einen objektorientierten Transaktions-Dienst auf Basis des **Object Transaction Service** der OMG einzuführen, jedoch befindet sich dieser erst im Versuchsstadium. CORBA wurde in diese Arbeit mit aufgenommen, weil sich das Konzept für das Management von Transaktionsmonitoren, das Thema dieser Diplomarbeit, zu großen Teilen auf einem CORBA-konformen Kommunikationsmechanismus, dem Object Request Broker (ORB) abstützt.

[1] Bei einem two-phase-commit wird ein Transaktionsabschluß in zwei Phasen vollzogen:
In Phase 1 geben sich die Kommunikationspartner gegenseitig bekannt, daß sie zu einem Commit bereit sind und alle Vorbereitungen getroffen haben.
In Phase 2 wird dann der eigentliche Commit vollzogen

2.3.1 Der CORBA-Standard der Object Management Group (OMG)

CORBA ist die Common Object Request Broker Architecture der OMG und beschreibt den Aufbau und die Funktionsweise eines Object Request Brokers (ORB) (eine explizite Erläuterung wird noch im Verlauf dieser Arbeit gegeben). Die OMG ist ein herstellerübergreifendes Konsortium mit über 500 Mitgliedern, das 1989 gegründet wurde. Das Ergebnis der Bemühungen, eine Architektur für die Zusammenarbeit verteilter Anwendungen auf Basis objektorientierter Konzepte zu schaffen, ist die **Object Management Architecture** (**OMA**).

Die wichtigsten Ziele, die bei der Entwicklung der OMA zugrunde lagen, waren lokationstransparente Verwendung der Dienste von Objekten in einem Verbund und Sprachunabhängigkeit der Implementierungen der einzelnen Objekte. Die OMA setzt sich aus vier Komponenten zusammen:

- dem Object Request Broker (ORB)
- den CORBAfacilities
- den CORBAservices
- den Application Objects

Während die CORBAfacilities und CORBAservices einen eher allgemeinen Charakter haben und hauptsächlich für administrative Aufgaben bei der Objekterzeugung und -verwaltung genutzt werden und die Kommunikationmechanismen mit dem ORB für Objekte zur Verfügung stellen, sind die **Application Objects** die konkreten Implementierungen von Anwendungen, die ihre Dienste der Welt zur Verfügung stellen.

Das Kernstück und damit den interessantesten Teil dieser Architektur bildet der Object Request Broker, ohne ihn könnten die Objekte nicht miteinander kommunizieren, Dienste nutzen oder Dienste bereitstellen. Um dem CORBA-Standard zu erfüllen sollten Implementierungen eines ORB ihrerseits folgende Dienste anbieten:

Name Service

Über den *Name Service* ist die eindeutige Identifikation von Objekten in einer Domäne gewährleistet. Der Name eines Objektes ist damit innerhalb einer Domäne einmalig. Um ein bestimmtes Objekt zu finden, wird es über den Name Service angesprochen, der das gesuchte Objekt lokalisiert.

Security Service

Die Authentifizierung von zwei Kommunikationspartnern geschieht über den *Security Service* . Er bildet damit eine Erweiterung des Name Service.

Delivery Service

Der *Delivery Service* übernimmt die Vermittlung von Nachrichten von einem Objekt zu einem anderen über ein Netzprotokoll. Die Synchronisation von Nachrichten, die parallel bearbeitet werden sollen, wird ebenfalls durch diesen Dienst erbracht.

Request Dispatch Service

Welche Methoden an einem Objekt für externe Aufrufe zur Verfügung gestellt werden und welche nur dem Objekt selbst zum Aufruf vorbehalten sind, wird über den *Request Dispatch Service* ermittelt.

Parameter Encoding

Die Abbildung der maschinen- und sprachabhängigen Datentypen von Parametern, mit denen Methoden aufgerufen werden beziehungsweise die Methoden zurückliefern, in ein neutrales Datenformat wird durch das *Parameter Encoding* erreicht.

Exception Handling

Bei CORBA wird die Fehlerbehandlung eines Objektes über *Exceptions* geregelt und nicht, wie in konventioneller Programmierung, über die Auswertung von Rückkehrcodes. In einem Fehlerfall wird eine Exception gesendet, auf die von einem anderen Objekt asynchron reagiert werden kann, da Synchronität nicht immer garantiert oder erwünscht ist.

Activation und Synchronization

Der eigentliche Aufruf einer Methode wird über den Dienst *Activation* ausgelöst. Dieser übernimmt auch eine Art Transaktionsschutz für **Persistente Objekte**, indem vor der Veränderung von Zuständen diese gesichert werden, um im Fehlerfall wiederhergestellt werden zu können. In diesem Zusammenhang ist noch der *Synchronization*-Dienst von Bedeutung, der den zeitlichen Ablauf von Anfragen an Objekte und deren Rückantwort regelt.

Die Architektur eines CORBA-konformen ORB

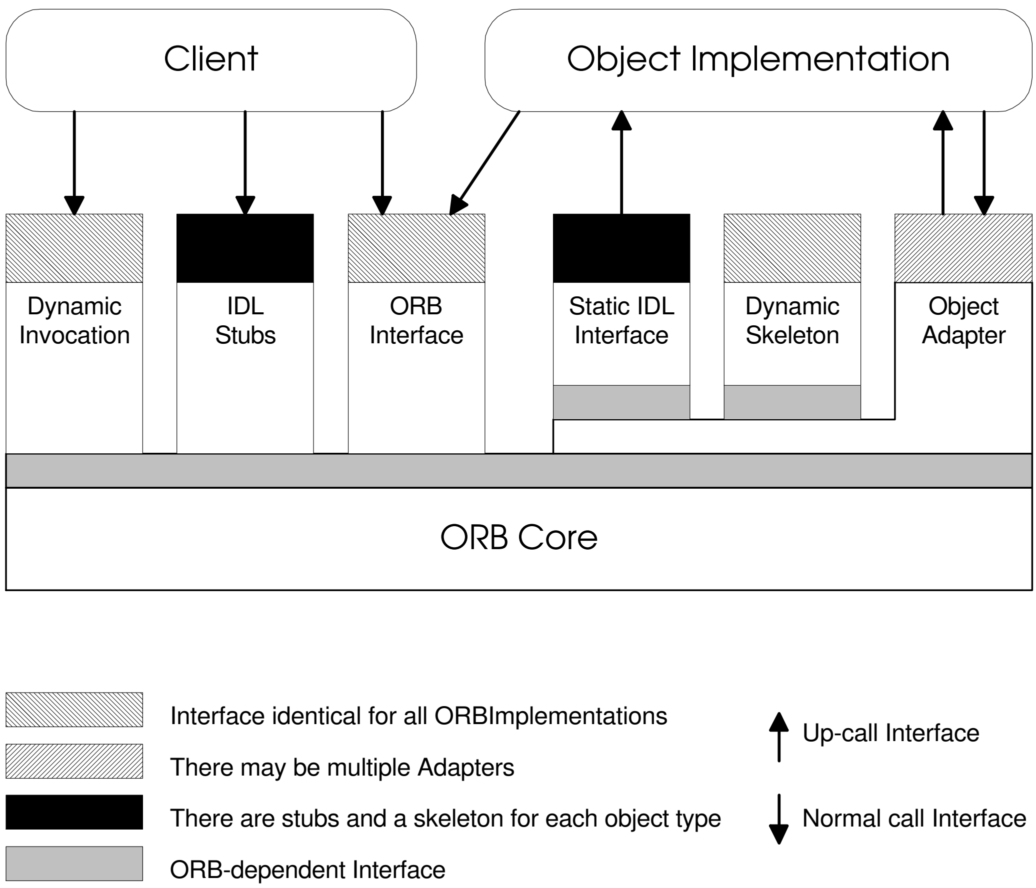

Abbildung 2.1: Die Architektur des Object Request Brokers (ORB)

In Abbildung 2.1 ist der prinzipielle Aufbau eines CORBA-konformen ORB skizziert. Der *ORB Core* bietet die Basisdienste, die in Abschnitt 2.3.1 beschrieben sind. Über den *Object Adapter* kommuniziert der ORB mit den implementierten Objekten. Er stützt sich dabei auf die statische Information, die über das *Static IDL Interface* festgelegt wurde. Mit dem *Dynamic Skeleton*, werden zusätzliche Methoden eines Objektes dem ORB mitgeteilt.

SOM/DSOM, eine Implementierung der CORBA von IBM

Inzwischen gibt es mehrere Implementierungen des CORBA-Standards, wie OpenDoc von Component Integration Lab., Distributed Objects Everywhere (DOE) von SunSoft und Distributed System Object Model (DSOM) von IBM.
DSOM, der ORB von IBM dient für die Implementierung der Agent-Manager-Kommunikation der Diplomarbeit als Grundlage, doch darauf wird im Kapitel 6 im Zusammenhang mit der Implementierung noch näher eingegangen.

2.3.2 X/Open-Standards

Die X/Open hat für die verteilte Transaktionsverarbeitung einen Rahmen geschaffen, der mehrere Standards für die verschiedenen Teilbereiche zusammenfaßt.

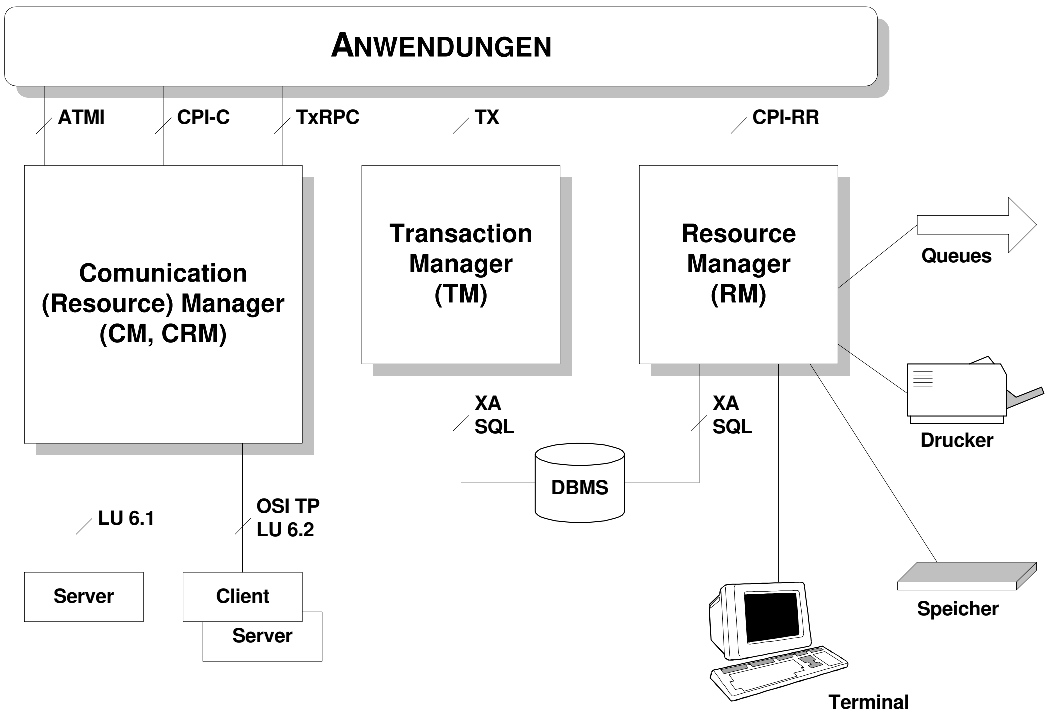

Abbildung 2.2: Das X/Open-DTP-Architekturmodell

X/Open-DTP-Architekturmodell: Das X/Open-Modell (siehe Abbildung 2.2) für verteilte Transaktionsverarbeitung (*Distributed Transaction Processing*) definiert eine Software-Architektur bestehend aus folgenden Komponenten:

- Resource Managern (RM), die Zugriff auf gemeinsam benutzbare Ressourcen wie Datenbanken oder Kommunikationsdienste bereitstellen. Ein Resource Manager für Kommunikationsdienste wird auch als Communication Manager (CM) oder Communication Resource Manager (CRM) bezeichnet.
- dem Anwendungsprogramm, das Transaktionen definiert und RM-Dienste im Rahmen dieser Transaktion verwendet.
- dem Transaction Manager (TM), der die Transaktionsendebehandlung durchführt, das heißt alle RMs koordiniert und für Recovery-Funktionen verantwortlich ist.

Mit dieser Architektur werden drei Ziele verfolgt:

- Portabilität von Anwendungsprogrammen, basierend auf den von X/Open genormten Schnittstellen, zum Beispiel SQL, CPI-C und andere.
- Zusammenarbeit in verteilten heterogenen Systemen, basierend auf genormten Kommunikationsprotokollen, zum Beispiel OSI-TP oder SNA LU6.2.
- Austauschbarkeit von Komponenten, basierend auf genormten Integrationsschnittstellen, zum Beispiel XA.

Innerhalb des X/Open-DTP-Architekturmodells sind folgende Spezifikationen definiert:

XA: Die **XA-Schnittstelle** ist eine bidirektionale Schnittstelle auf Systemebene, die 1994 von der *X/Open* festgelegt wurde. Diese Schnittstelle ermöglicht verschiedenen Resource-Managern innerhalb einer Umgebung für verteilte Transaktionsverarbeitung auf einem gemeinsamen, globalen Kontext zu arbeiten. Als Resource-Manager kann zum Beispiel eine XA-konforme Datenbank dienen. Diese Datenbank kann dann eine globale Transaktion nutzen, die von einem Transaktionsmanager mit XA-Schnittstelle koordiniert wird. Eine genaue Beschreibung der Anforderungen findet sich in [XA94].

CPI-C: (**Common Programming Interface for Communication**)
Die **CPI-C** ist eine Programmierschnittstelle für Anwendungen, die Programm-zu-Programm-Kommunikation benötigen. Sie dient als Abstraktionsschicht zwischen Anwendung und Kommunikationsschicht. Wird CPI-C verwendet, ist darunter eine Anbindung wahlweise zu SNA- oder OSI-Protokollen der Schichten 3-7 möglich.
Sie bietet:

- Senden und Empfangen von Daten
- Synchronisation der Verarbeitung von Programmen
- Benachrichtigung von Kommunikationspartnern bei Fehlern

Genauere Informationen findet man in [CPI-C]

CPI-RR: (**Common Program Interface for Resource Recovery**)
Wie die *CPI-C* ist auch die **CPI-RR** eine Programmierschnittstelle für Anwendungen. Sie bietet Unterstützung bei der Wiederherstellung von Betriebsmitteln nach einem Fehler oder einem Abbruch. CPI-RR wurde in Zusammenarbeit von X/Open und IBM definiert.

TX-Spezifikation: (**Transaction Demarcation Specification**)
TX ist eine API durch die ein Anwendungsprogramm einen Transaktion-Monitor anspricht, um globale Transaktionen gegeneinander abzugrenzen und ihr Abschließen zu steuern. [TX95]

TxRPC: Transactional Remote Procedure Call
TxRPC ist eine Schnittstelle zwischen einem Anwendungsprogramm und einem **Communications Resource Manager** (**CRM**), der transaktionsgeschützte RPC verwendet. In [TxRPC] wird neben dem TxRPC auch der **Remote Task Invocation Service** (**RTI**) definiert, der zusammen mit der Protokollspezifikation das **Application Service Element** (**ASE**) bildet.

ATMI: Das **Application-to-Transaction Manager Interface** ist, wie auch die TxRPC eine Schnittstelle zwischen einem Anwendungsprogramm und einem **Communications Resource Manager** (**CRM**), der jedoch in diesem Fall auf Client/Server-Basis arbeitet. [XATMI]

SQL: Die **Structured Query Language** ist die am häufigsten verwendete Datenbankabfragesprache. Dieser Standard hat sich bei Datenbanksystemen durchgesetzt und bietet eine einheitliche Schnittstelle zu Datenbanksystemen, um Abfragen zu formulieren.

2.3.3 SNA-Standards (Industriestandard von IBM)

LU1-Protokoll: (**Logical Unit 1**) LU1 definiert eine Session zwischen einem Host-Anwendungsprogramm und einem RJE-Endgerät (zum Beispiel Tastatur, Drucker, Kartenleser, Plattenlaufwerk).

LU2-Protokoll: (**Logical Unit 2**) Eine LU2-Session wird zwischen einem Host-Anwendungsprogramm und einem IBM-327x-Terminal aufgebaut.

LU3-Protokoll: (**Logical Unit 3**) Mit LU3 baut ein Host-Anwendungsprogramm zu einem IBM-327x-Drucker eine Session auf.

LU6.1-Protokoll: (**Logical Unit 6.1**) Die Bezeichnung LU6.1 wird zur Abgrenzung zu LU6.2 verwendet. Früher wurde sie als LU6 bezeichnet, woraus die LU6.2 hervorging. LU6.1 ist ein Protokoll zur Kommunikation zwischen eigenständigen Systemen.

LU6.2-Protokoll: (**Logical Unit 6.2**) Das SNA-Protokoll *LU6.2* beschreibt eine Session zwischen zwei Anwendungsprogrammen, die üblicherweise auf unterschiedlichen Systemen laufen. Dieses Protokoll hat unter dem Namen **Advanced Program to Program Communication** (**APPC**) im Bereich der Interprozeßkommunikation an Bedeutung gewonnen. Im OSI-Management hat die LU6.2 im OSI-TP seine Entsprechung.
Eine LU6.2 bietet unter anderem folgende Basisdienste an:

- Abbildung von Anwendungskommunikationsbeziehungen auf Sessions
- Aktivieren und Beenden entfernter Programme einschließlich Initialversorgung mit Daten
- Allokation und Deallokation von Betriebsmitteln
- Setzten von Wiederaufsetzpunkten (Sync-Points) bis Synchronisations-Level 2 (two-phase-commit)
- Zugangskontrolle (Authentifizierung, Verschlüsselung)
- Unterstützung von Datagramm-Dienst, sysnchronem und asynchronem Senden
- Dialogsteuerung

Für weiterführende Informationen sei auf [HEAB 93] und [CYPS92] verwiesen.

2.3.4 ISO/OSI-Standards

OSI-TP: (**Open System Interconnection-Transaction Processing**)
OSI-TP ist ein Protokoll der Schicht 7 des ISO-Referenzmodells, für Anwendungen, die Programm-zu-Programm-Kommunikation benötigen. OSI-TP entspricht in etwa dem SNA-Protokoll LU6.2.
OSI-TP bietet:

- Senden und Empfangen von Daten
- Synchronisation der Verarbeitung von Programmen
- Benachrichtigung von Kommunikationspartnern bei Fehlern

Genauere Informationen findet man in der ISO-Norm 10026.

FTAM: (**File-Transfer-Access-Management**)
FTAM ermöglicht die Übertragung ganzer Dateien zwischen unterschiedlichen Systemen, wobei Dateien über beliebige Netze und Transportsysteme bewegt werden können. (Pendant zu FTP in der Internet-Welt)

2.3.5 OSF-Standards

Die Standards der **Open Software Foundation** beziehen sich in erster Linie auf die verschiedenen UNIX-Derivate und UNIX-ähnliche Betriebssysteme, die noch keine OSF-Zertifizierung haben. Im folgenden werden diese Betriebssysteme unter dem Oberbegriff *offene Systeme* zusammengefaßt.

Distributed Computing Environment (DCE)

Der Betrieb von offenen Client/Server-Anwendungen erfordert hohe Sicherheit und Flexibilität auch über ein Netz hinweg, einfache Skalierbarkeit des Netzes, um mit steigenden Anforderungen wachsen zu können und Synchronisationsmöglichkeiten (zum Beispiel für Zeitstempel). Diese Anforderungen stellen auch Transaktions-Monitore, wodurch es durchaus Vorteile hat einem Transaktionsmonitor die Dienste dieses Frameworks für verteilte Anwendungen nutzbar zu machen.
DCE stellt für die Lösung dieser Aufgaben folgende Dienste bereit:

- Remote Procedure Call (RPC)
- Cell Directory Service (CDS)
- Time Service (DTS)
- Security Service
- Distributed File Service (DFS)

Transaktionsmonitore, die mit DCE-Unterstützung installiert wurden, nutzen in der Regel dabei nur den Cell Directory Service, um einen standardisierten Zugriff auf Datei-Systeme innerhalb einer Domäne, die dann DCE-Cell heißt, zu gewährleisten. Bisher verwenden nur TP-Monitore auf UNIX-Systemen die Dienste der DCE, wobei ebenso Installationen ganz ohne DCE üblich sind.

2.3.6 Zusammenfassung

Die wichtigsten Architekturen der verschiendenen Standardisierungs-Organisationen wurden oben vorgestellt. Der Grund, daß mehrere Transaktionsmonitore stark von diesen Standards abweichen, ist wohl der lange Zeitraum, in dem TP-Monitore bereits eingesetzt werden. Frühere Entwicklungen bezogen sich auf Industriestandards, die jeder Hersteller individuell festlegte. Bei den mehreren Entwicklungen, die teilweise auf Konzepten basieren, die mehr als 20 Jahre alt sind, gab es diese standardisierten Architekturmodelle noch nicht. Bei Portierungen auf moderne Betriebssysteme wurden oft die alten Konzepte noch übernommen, um kompatibel zu eigenen Anwendungen zu bleiben. Erst in neueren Entwicklungen findet man eine standardisierte Architektur mit einheitlichen Schnittstellen.
Nachdem die wichtigsten Begriffe und Standards, die im Zusammenhang mit Transaktionsmonitoren und dieser Arbeit stehen, eingeführt wurden, folgt im nächsten Abschnitt eine Vorstellung heute verwendeter Transaktionsmonitore.

2.4 Überblick bekannter TP-Monitore

Um einen kurzen Überblick über die heute wichtigsten Transaktionsmonitore zu schaffen, werden im folgenden einige Vertreter für offene Systeme und Großsysteme vorgestellt.

2.4.1 Encina von Transarc

Encina von **Transarc** ist ein Transaktions-Monitor für Kunden mit neuer DV-Infrastruktur, die auf UNIX-Systemen basiert. Er ist auf allen verbreiteten UNIX Plattformen verfügbar. Die Encina Produktfamilie basiert auf einer modularen, geschichteten Architektur. Sie baut auf den Basisdiensten des **Distributed Computing Environment** (**DCE**) der **Open Software Foundation** (**OSF**) auf und bietet eine Vielfalt von

Einrichtungen für verteilte Transaktionsverarbeitung. Encina besteht einerseits aus dem *Toolkit*, in dem die notwendigen Dienste für verteilte Transaktionsverarbeitung und das Management der wiederherstellbaren Daten enthalten sind, und andererseits aus den *Extended Services*, welche die Funktionalität des *Toolkit*, um eine leistungsfähige Umgebung für die Entwicklung verteilter Transaktionsverarbeitung erweitern.
ENCINA ist nur auf UNIX-Plattformen verfügbar.

Architektur

Die Architektur des Transarc Encina basiert auf dem Architekturmodell der X/Open für verteilte Transaktionsverarbeitung, das in Abbildung 2.2 bereits mit seinen Schnittstellen beschrieben ist. Encina basiert auf vielen Standards und de facto Standards, wie:

- dem **DCE** (**Distributed Computing Environment**) der **Open Software Foundation** (**OSF**).
- der X/Open **XA** und **ISAM**-Schnittstelle zur Anbindung an alle großen Datenbanksysteme, wie zum Beispiel DB2, INFORMIX und ORACLE. Siehe 2.3.2.
- dem **Transactional RPC** (**TxRPC**). Siehe 2.3.2
- dem X/Open Standard CPI-C. Siehe 2.3.2
- dem X/Open und IBM SAA Standard CPI-RR. Siehe 2.3.2

2.4.2 UTM/openUTM von Siemens

Die Anfänge des *UTM* liegen im Mainframe-Bereich, wo UTM unter dem Betriebssystem *BS2000* von Siemens entwickelt wurde. Vor sechs Jahren wurde UTM jedoch völlig neu gestaltet und zusätzlich auf den unten beschriebenen Plattformen angeboten. Das überarbeitete Produkt heißt seitdem *openUTM* und sein Schwerpunkt liegt bei den offenen Systemen. Dieser wird bereits in mehreren tausend Installationen eingesetzt. *openUTM* dient heute als Basis für Client/Server-Lösungen und ermöglicht offene Lösungen selbst dort, wo dies aus Gründen der Sicherheit bisher nicht möglich war. *openUTM* ist auf verschiedenen Plattformen verfügbar:

Mainframe: BS2000 (Siemens)

Offene Systeme: AIX (IBM), DC/OSx (DEC), HP-UX (Hewlett-Packard), ICL-SVR4.2 (ICL), SCO UNIX (Santa Cruz Operation), SINIX (Siemens), SOLARIS (SUN),Unixware (NOVELL)

PC-Systeme: WindowsNT (Microsoft)

Client-Komponenten: Für alle obengenannten Systeme verfügbar, zusätzlich auf Microsoft Windows.

Architektur

Durch seine offenen Schnittstellen entspricht *openUTM* (siehe Abbildung 2.3) dem X/Open-DTP-Architekturmodell. So sind zum Beispiel Programme, die für IBMs Customer Information Control System (CICS) entwickelt wurden, ohne Änderungen unter *openUTM* lauffähig. Der *openUTM* unterstützt die folgenden Konzepte der Transaktionsverarbeitung:

- Conversational Transaction Processing (siehe 2.2)
- Pseudo-Conversational Transaction Processing (siehe 2.2)
- Aufruf von Long-running-Transactions über Message Queues. (siehe 2.2)

Durch die völlige Umgestaltung des openUTM wurde es möglich, ein einheitliches Konzept für die Architektur auf den verschiedenen Systemen durchzusetzten. Dieser Vereinheitlichung verdankt es der openUTM, daß ein integrierten Management-Werkzeug für alle openUTM-Systeme angeboten werden kann. Das Produkt heißt **WinAdmin** und ist auf Windows NT implementiert. Es bietet Management-Funktionalität für alle openUTM-Systeme, unabhängig auf welcher Plattform. Leider ist WinAdmin ein SNMP-basiertes Management-Werkzeug, so daß für diese Diplomarbeit kein weiterer Management-Agent auf Basis von CORBA zur Verfügung steht.

2.4.3 TUXEDO von Novell/BEA

TUXEDO von Novell/BEA Systems ist auf vielen UNIX-Systemen verfügbar und wird vom Netware-Hersteller Novell vertrieben. Die eigentliche Entwicklung scheint bei der Firma *BEA Enterprise Middleware Solutions* zu liegen, wobei Novell nur den Vertrieb übernommen hat. Mitte des Jahres 1996 hat Novell die Zusammenarbeit mit BEA beendet und unterhält lediglich noch eine WWW-Seite mit Informationen zu TUXEDO.

TUXEDO besitzt eine Management-Komponente, mit einer ansprechenden, graphischen Bedienoberfläche. Leider stand für die Diplomarbeit keine Testversion zur Verfügung, um die Leistungsfähigkeit dieses Management-Werkzeuges zu testen. Informationen zu den Management-Protokollen wurden selbst nach mehrmaligem Anfragen nicht zur Verfügung gestellt.
Die **TUXEDO**-Server unterstützen folgende Betriebssysteme:

Mainframe: keine

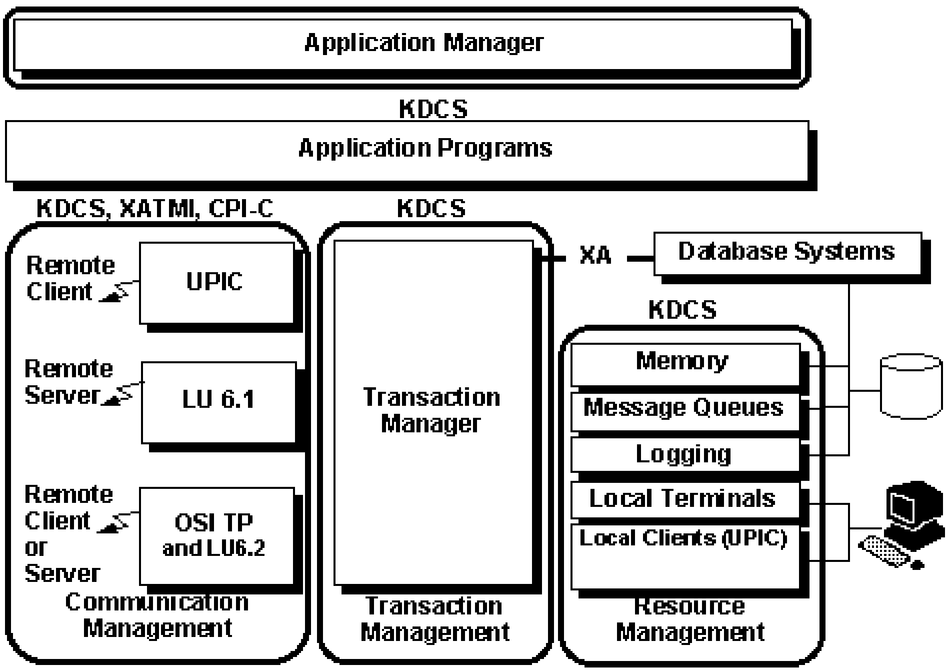

Abbildung 2.3: Die Architektur des openUTM von Siemens

Offene Systeme: AIX (IBM RS/6000), HP-UX (Hewlett-Packard 9000), Solaris und SunOS (SUN), UNIX SVR4 (NCR), UnixWare und NetWare (Novell), DG-UX (Data General), Alpha Windows NT und AXP UNIX (Digital), DC/OSx (Pyramid), SCO UNIX (Santa Cruz Organisation), DYNIX (Sequent) und IRIX (SGI)

PC-Systeme: Windows NT (Microsoft)

Client-Komponenten: MS-DOS, IBM OS/2 und Microsoft Windows3.1, Windows95, Windows NT und Macintosh

Unterstützt werden alle objektorientierten und relationalen Datenbanksysteme. Für verteilte Transaktionsverarbeitung ist jedoch eine XA-konforme Datenbank (siehe 2.3.2) notwendig.

Architektur

Die Aufbau des TUXEDO scheint startk angelehnt an das X/Open-DTP-Architekturmodell, da es eine verteilte Anwendung ist, die für offene Systeme konzipiert wurde. *TUXEDO* hat eine offene, flexible Architektur mit einem Kernsystem, dem *Transaction Manager*, basierend auf der X/Open-Schnittstelle *ATMI*, und verschiedene Schnittstellen-Komponenten, *Workstation*, *Queue services*, *Domains* und *DCE integration* für die Kommunikation mit Clients, Workstations, Datenbanken, DCE-Cells und Mainframes. Eine Übersicht zu den Komponenten bietet Abbildung 2.4. *TUXEDO* bietet die Möglichkeit des Anwendungsmanagements über eine graphische Anwenderschnittstelle, ein *Event Communication and Brokering System*, Sicherheitsmechanismen und Sprachunterstützung C, C++, COBOL. Durch *Two-Phase-Commit* wird auch verteilte Transaktionsverarbeitung unterstützt. Vernestete Transaktionen sind nicht möglich.

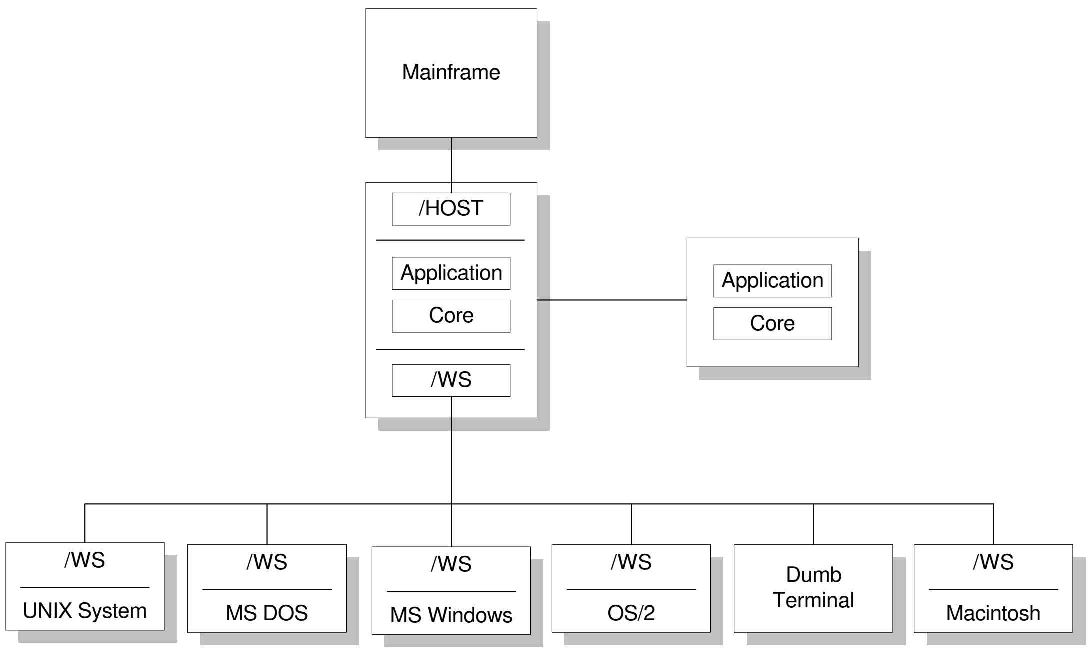

Abbildung 2.4: Die Architektur des TP-Monitors TUXEDO von Novell/BEA

2.4.4 Information Management System (IMS) von IBM

Vor 15 Jahren noch war **IMS** (**Information Management System**) der leistungfähigste TP-Monitor von IBM. Er enthält eine Datenbank - DL/1 - und hatte bereits damals bei seiner Einführung ein ausgefeiltes Security-System. Außerdem ist er mit einem sehr leistungsfähigen Scheduler ausgerüstet, was einen hohen Transaktions-Durchsatz unter großer Last ermöglicht. IMS ist kein Online-Transaktionsmonitor im eigentlichen Sinn, wie die anderen vorgestellten Systeme, sondern ein Batchsystem, das über eine Nachrichen-Warteschlange die Anforderungen von den Terminals bedient. Durch dieses **Queueing** wird ein dialogähnliches Verhalten bei IMS erreicht. Unterstützte Plattformen:

Mainframe: MVS auf IBM 3xx, 30xx, ES9000

Offene Systeme: keine

PC-Systeme: keine

Client-Komponenten: keine

Architektur

IMS besteht aus einem Kernsystem, der **Control Region**, und mehreren externen Komponenten für den Datenbankzugriff, Queueing und den Zugriff auf Terminals. Das Kernsystem und die externen Komponenten befinden sich während der Laufzeit zwar im selben Adreßraum, die Regions in denen die Anwendungen laufen, die **Dependent Regions**, liegen jedoch jeweils in eigenen Adreßräumen. Durch diese Trennung der Adreßräume in systemeigene und anwendungsnahe Bereiche wird eine höhere Stabilität des Systems erreicht. Anwendungsprogramme können dadurch ein IMS-System nicht so einfach zum Stillstand bringen, wie beispielsweise in einem System mit einem gemeinsamen Adreßraum. Datenbankzugriffe werden vom Datenbank-Controller an eine spezielle **Resource Owning Region** weitergeleitet, die den eigentlichen Zugriff bearbeitet. Das hat eine Lastverteilung zur Folge und schottet die Datenbank während der Laufzeit vom TP-Monitor ab (Abbildung 2.5). Das macht den IMS auch heute noch zu einem sehr stabilen System. Verbindungen zu anderen Transaktions-Monitoren werden direkt von der *Control Region* aufgebaut. Möglich sind hier LU6.1- und LU6.2-Verbindungen.

2.5 Customer Information Control System (CICS) von IBM

CICS enthält keine eigene Datenbank, ist aber auf den Betrieb mit der Datenbank **DB2** ausgerichtet. CICS ist vom Konzept her moderner als IMS und löst langsam aber sicher das IMS ab. Durch seine modernere Architektur und der Möglichkeit von echter Online-Verarbeitung, hat es Vorteile gegenüber dem IMS und ist zudem auf vielen verschiedenen Plattformen verfügbar:

CICS/ESA	IBM MVS/ESA (ES/9000)
CICS/MVS	IBM MVS (30xx)
CICS/VSE	IBM VSE
CICS for OS/2	IBM OS/2
CICS/NT	Microsoft WindowsNT
CICS/400	IBM AS/400
CICS/VM	IBM Virtual Machine
CICS on Open Systems:	
CICS/6000	IBM AIX (RS/6000)
CICS	HP-UX (HP)
CICS	OSF/1 (DEC)
CICS	Solaris (Sun)

CICS-Clients sind verfügbar für:
AIX, Sun, Apple Mac, IBM OS/2, MS-Windows 3.1, MS-Win 95, MS-Win NT, MS-DOS.

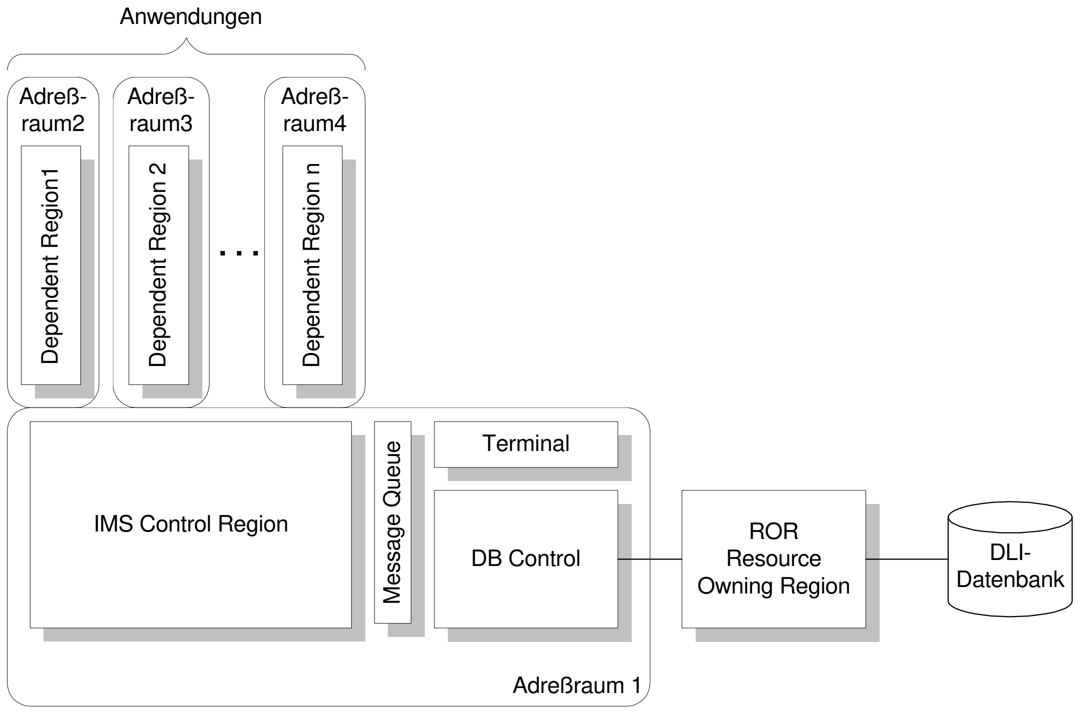

Abbildung 2.5: Der Zusammenhang der Komponenten des IBM Information Management System (IMS)

2.5.1 Architektur

Das Kernstück eines CICS ist die **Kernel Function**, kurz **KE**, die als Basis für die anderen Komponenten, den sogenannten **Domains**, des Systems dient. Das CICS ist modular aufgebaut und stellt für jeden Aufgabenbereich eine eigene Komponente zur Verfügung. Die einzelnen Module eines CICS sind in Abbildung 2.6 dokumentiert. An der **Application Domain** (**AP**) werden die verschiedenen Anwendungen bedient. Die Datenbank, hier DB2, wird unter CICS genauso angesprochen, wie eine Anwendung und kommuniziert daher auch mit der AP-Domain. Auch hier ist es natürlich möglich, die Datenbankzugriffe über spezielle Regions zu steuern. Die Verbindung zu anderen CICS-Systemen mittels **Intersystem Communication** (**ISC**) oder **Multi-Region Operation** (**MRO**) wird über die **Terminal Domain** (**TE**) hergestellt. Für Verbindungen über **APPC**, **LU6.1** und **LU6.2** ist die **External Communication Domain** (**ZC**) zuständig, die damit Client-Systeme und fremde Transaktions-Monitore bedienen kann. *Terminal Domain* und *External Communication Domain* bilden daher funktional eine Einheit, sind aber zwei eigenständige Komponenten. Eine Besonderheit bezüglich der Prozeßverwaltung des CICS ist noch erwähnenswert: Das CICS hat eine eigene Prozeßverwaltung, losgelöst vom Betriebssystem, die **kein** präemptives Multitasking unterstützt, sondern ein **kooperatives Multitasking**, wie es zum Beispiel von Microsoft Windows her bekannt ist. Der Vorteil ist offensichtlich: Weniger Verwaltungsaufwand während der Verarbeitung, da begonnene Vorgänge ohne immerwährende Unterbrechungen beendet werden können. Die Nachteile des kooperativen Multitaskings sind aber gerade seit Windows 3.0 unangenehm in Erinnerung: Fehlende Unterbrechungsmöglichkeiten für einen Prozeß, der Betriebsmittel nicht mehr freigibt und lange Wartezeiten bei Ein- und Ausgabeoperationen. Diese Nachteile hat man jedoch bei CICS vermeiden können, Performance ist eine der Grundvorraussetzungen für Transaktionsmonitore, indem die CICS-API wenig Möglichkeiten bietet, Betriebsmittel so dauerhaft zu beanspruchen, daß andere Prozesse keine Zugriffsmöglichkeiten über einen längeren Zeitraum haben.

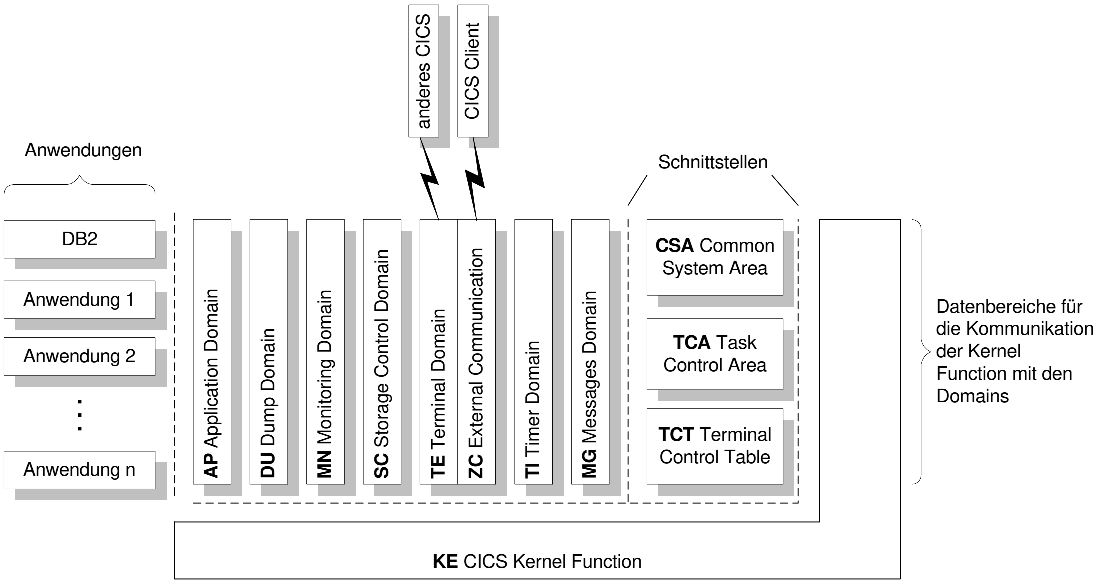

Abbildung 2.6: Die IBM CICS-Architektur mit ihren Domains

2.5.2 Übliche CICS-Konfiguration

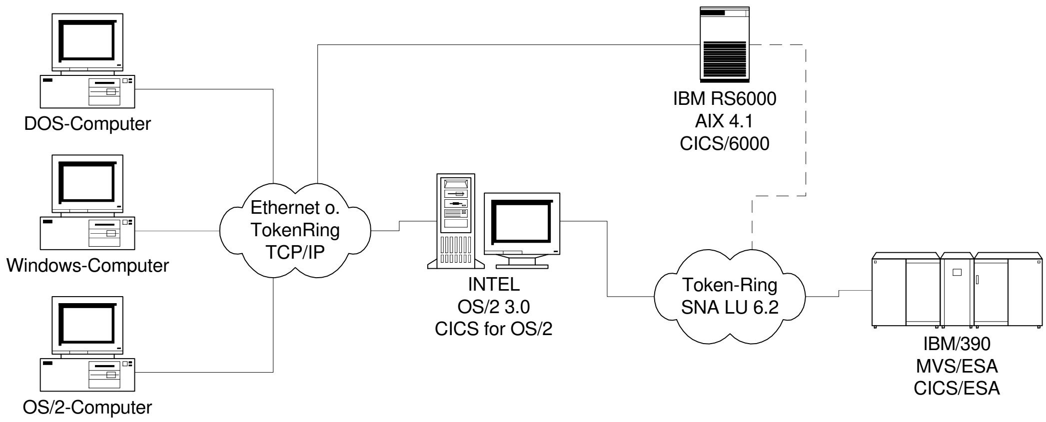

Abbildung 2.7: Übliche CICS-Konfiguration

Wie in Abbildung 2.7 dargestellt, sind die Arbeitsplatzrechner mit den PC-Betriebssystemen über ein TCP/IP-LAN auf Basis von Ethernet oder Token-Ring mit einem CICS-Server verbunden. Dieser dient zum einen als Gateway zwischen dem TCP/IP-Netz und einem SNA-Netz und zum anderen kann er als

Verteiler auf verschiedene CICS-Systeme fungieren. Dabei wird dann ein Teil der Transaktionsverarbeitung in den CICS-Server verlagert. Der CICS-Server befindet sich an einem Token-Ring auf dem das Protokoll SNA-LU6.2 aufsetzt und ist über dieses mit einem Großrechner verbunden. Wenn die Servermaschinen leistungsfähig genug sind kann man die Transaktionsverarbeitung auch vollständig auf diese verlagern und greift dann zum Beispiel auf ORACLE-Datenbanken zu. Seit der Implementierung von TCP/IP auf Großsystemen ist es auch möglich, die CICS-Clients direkt mit einem Großrechner zu verbinden, jedoch wird das in der Praxis noch nicht gemacht, weil TCP/IP zusammen mit SNA-Protokollen noch nicht stabil läuft und bei der Administration noch große Probleme bereitet.

2.5.3 Die CICS-Clients

Die **CICS-Clients** dienen als **Anwenderschnittstelle zum CICS**. Hier ist die **Präsentationslogik** implementiert. Für Hostanwendungen können auf diesen Client-Maschinen graphische Anwenderschnittstellen realisiert werden, die zum Beispiel mit GUI-Buildern erzeugt wurden. Es ist auch möglich, bereits hier Mechanismen zur **Lastbalancierung** zu implementieren, indem man verschiedene CICS-Server zur Auswahl stellt, und nach Zufallsalgorithmen oder abhängig vom jeweiligen Standort, einen Server zur Laufzeit auswählt.

2.5.4 Die CICS-Transaction-Server

In der neuesten Version der IBM CICS Familie gibt es die Unterscheidung zwischen Großrechner-CICS-Systemen und CICS-Servern nicht mehr. Sie sind nur noch über ihre Funktionalität zu unterscheiden. Zum Beispiel dienen die CICS-Server, CICS for OS/2 oder CICS/6000, meist als Gateways zwischen den **Protokolle**n TCP/IP und SNA LU6.2. Auf den CICS OS/2-Servern wird diese Gateway-Funktion von einer eigenen Anwendung, dem IBM **Communication Manager**, erbracht. Bei CICS/6000-Servern gibt es ein **ENCINA PPC Gateway**, das die Umsetzung realisiert. Die **CICS-Server** auf den oben genannten Systemen können darüber hinaus auch als Verteiler auf verschiedene CICS-Systeme auf Großrechnern verwendet werden. Auf die Verteilmechanismen von einem CICS-System auf verschiedene andere wird im Abschnitt 2.6.1 näher eingegangen. Bei leistungsstarken Server-Maschinen wird überdies die **Geschäftsprozeß**-Logik auf dem Server hinterlegt, so daß vom Großrechner nur noch die Funktionalität erbracht wird, jedoch die Reihenfolgeinformation, nach der die verschiedenen Arbeitsschritte erfolgen, zum Beispiel auf dem CICS/6000-Server bleibt. Die CICS-Server für MVS/ESA oder andere Großsysteme, übernehmen in den meisten Fällen das sogenannte Heavy-Load-Transaction-Processing mit mehreren hundert bis tausend Anwendern, die gleichzeitig im System arbeiten.

2.6 IBM CICS Terminologie

Die Terminologie innerhalb des IBM Customer Information Control System weicht an manchen Stellen von den allgemeinen Definitionen ab. Deshalb werden im folgenden die wichtigsten Begriffe nochmals erläutert, jedoch im Kontext von IBM CICS.

2.6.1 Grundlagen

In der CICS-Terminologie gibt es neben der Transaktion noch die **Task** und die **Logical Unit of Work** (**LUW**). Im CICS stellt die **CICS-Transaktion** eine logische Einheit im Sinne des DV-Entwicklers dar. Ihr wird ein eindeutiger, bis zu vierstelliger Transaktions-Identifikator zugeordnet. Die Verbindung zwischen der Transaktion und dem Einstiegsprogramm wird in einer System-Tabelle hergestellt. Im CICS/ESA und im CICS for OS/2 ist das die **Program Control Table** (**PCT**), im CICS/6000 heißt sie **Transaction Definition**. Aus dem Einstiegsprogramm heraus können weitere Programme aufgerufen werden. Ein **CICS-Programm** ist im Prinzip ein dynamisch ladbares Modul, das in einer Programmiersprache verfaßt und übersetzt wird. Es steht dann als Datei in einem Verzeichnis. In der PCT wird auch festgelegt, ob die Transaktion lokal oder als Remote-Transaktion definiert werden soll. Das Programm einer **Remote-Transaktion** ist nicht physisch in der Region installiert, in der die **Remote-Transaktion** aufgerufen wird, sondern wird nur durch einen Verweis auf eine Transaktion in einem anderen CICS-System festgelegt. Bei der Definition der Transaktion werden auch deren andere Attribute festgelegt, wie zum Beispiel die Rechte, die ein Anwender haben muß, um sie auszuführen oder Zeichensatztabellen die für die Anzeige zur Anwendung kommen. Die **CICS-Task** ist definiert als Ausführungseinheit einer Transaktion aus Sicht des TP-Monitors. Die Ausführung einer Task ist eine einmalige Aktion und erhält eine eindeutige Task-Nummer innerhalb des CICS-Systems. Eine Transaktion kann jedoch Grundlage für beliebig viele Tasks sein. Die Belegung von Betriebsmitteln beansprucht die CICS-Task durch ihre Verweildauer im System. Außerdem ist sie es, die Sicherheits-Einrichtungen zur Konsistenz-Sicherung beeinflußt. Der TP-Monitor protokolliert und sperrt demnach nur die aktuell von der CICS-Task angesprochenen Daten-Manipulationen. Die **Logical Unit of Work** (**LUW**) ist, wie der Name schon sagt, eine logische Arbeitseinheit und kommt der generellen Definition einer Transaktion am nächsten. Die LUW kontrolliert somit einen abgeschlossenen Konsistenzbereich. Sie ist auch vergleichbar mit der LUW im Sinne des OSI-Sprachgebrauches.

In der CICS-Terminologie finden sich viele technische Ausdrücke aus dem Bereich der **Systems Network Architecture** (**SNA**): Eine Maschine, die im physikalischen Sinne existiert, wird hier **Node**, **Physical Unit** (**PU**) oder auch **Knoten** genannt. Ihm ist zum Beispiel eine IP-Adresse oder eine System-ID zugeordnet, er hat Laufwerke, Speicher, CPU und mehr. Auf dem Knoten ist ein Betriebssystem installiert und ein Transaktionsmonitor nebst Subsystemen. Der Transaktionsmonitor kann mehrere Adressräume parallel auf dem Node verwalten. Solche Adressräume nennt man im CICS **Region**. Eine Region ist ein CICS-System, das in einem eigenen Adressraum läuft. Auf CICS/6000-Systemen werden in diesem Adressraum vom CICS mehrere Prozesse, die **Application Server**, gestartet. Diese Prozesse werden dann vom CICS dyna-

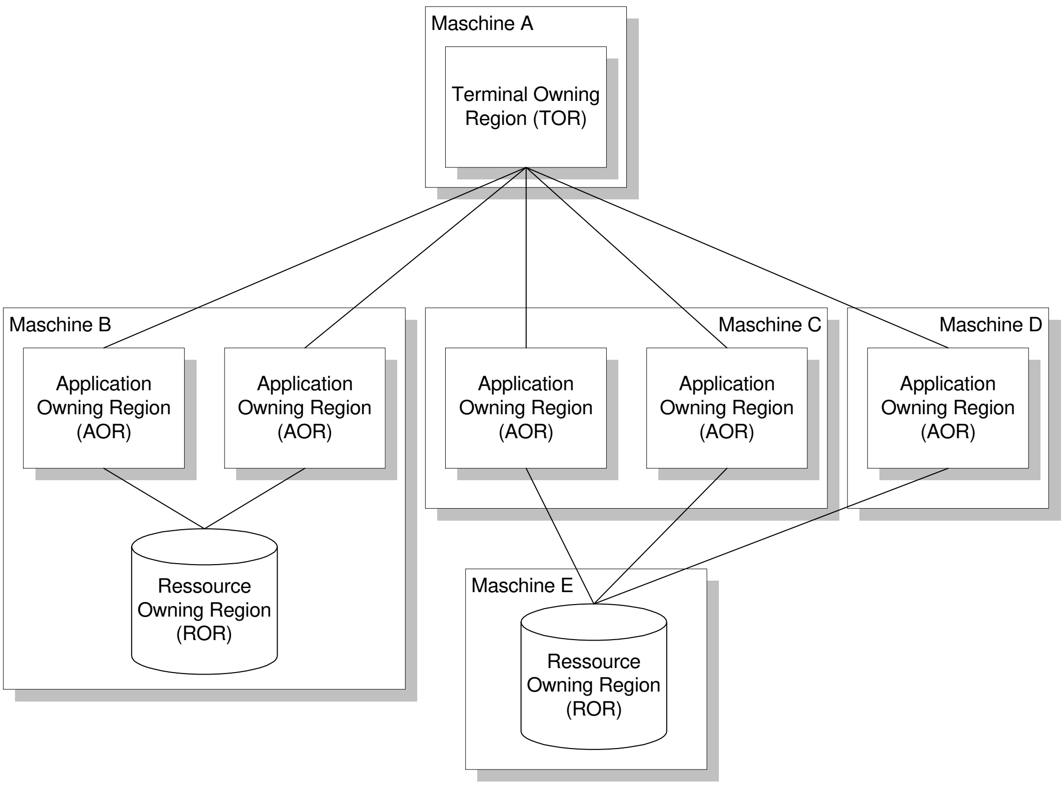

Abbildung 2.8: Verteilung von TOR auf AOR

misch den einzelnen Tasks der Transaktionen zugeordnet. Die Application Server stehen miteinander über Speicherbereiche in Verbindung, die innerhalb einer CICS-Region global schreib- und lesbar sind (shared memory).

Auf einem Node können mehrere Regions installiert sein, die jeweils ein eigenständiges CICS-System darstellen. Man kann jede Region über ihren Region-Namen identifizieren. Wenn eine Region, etwa zu Wartungszwecken, gestoppt wird, kann auf den anderen Regions eines Nodes unbeeinflußt weitergearbeitet werden. Fast immer werden bei einer CICS-Installation Regions mit verschiedenen Aufgabengebieten definiert. Abbildung 2.8 zeigt eine mögliche Konfiguration, in der es eine **Terminal Owning Region** (**TOR**) und mehrere **Application Owning Regions** (**AOR**) gibt. Eine TOR dient dabei als Zugang zu weiteren AORs. Die TOR enthält die Zuordnung aller Terminals, die sich von außen anmelden können, zu den entsprechenden AORs, in denen die Anwendungen liegen. Einer **Application Owning Region** sind unter funktionalen Aspekten die Anwendungs-Transaktionen und Programme zugeordnet.

Eine dritte Region-Kategorie ist die **Data Owning Region** oder auch **DOR**. In ihr werden die Dateien, Transient-Storage-Queues und Transient-Data-Queues verwaltet. Deshalb werden sie auch oft **Resource Owning Region** (**ROR**) genannt, wobei man sich der Sprachregelung des X/Open-DTP-Architekturmodells (siehe 2.2) angepaßt hat. Diese Klassifizierung der verschiedenen Regions und die Aufteilung der Transaktionen, Programme, Dateien usw. auf verschiedene AORs und DORs ist sinnvoll, wenn sich dadurch eine fachliche Strukturierung erreichen läßt. In der Praxis werden meist noch zusätzliche Regions für Testumgebungen angelegt, um Anwendungen, die in Produktion laufen nicht zu beeinflussen. Oft werden sogar darüberhinaus Regions angelegt, die zur Integration von neuen Anwendungen in die vorhandene Produktivumgebung dienen.

Ein Verbindungsaufbau von einem Terminal an eine Region oder auch die Verbindung von CICS-Systemen untereinander erfolgt über einem sogenannten **Listener**, der die Schnittstelle eines CICS-Systems nach aussen bildet. Der Listener ist ein Bestandteil des CICS-Systems, der im Falle von TCP/IP, über einen Port Pakete empfängt und im CICS-System entsprechende Aktionen auslöst. Listener gibt es auch für SNA-Protokolle.

Einen ähnlichen Mechanismus bieten die Gateways mit dem Unterschied, daß ein **Gateway** eine Schnittstelle zu einer Kommunikationseinrichtung in einem anderen CICS-System ist. Solche Kommunikationseinrichtungen sind zum Beispiel die ENCINA PPC Gateways.

2.6.2 Kommunikation

Für die Kommunikation zwischen zwei Systemen wird eine **Verbindung** (**Connection**) definiert. Eine Connection ist die Definition, wie zwei Regions untereinander eine logische Verbindung aufbauen sollen. Die Definition einer Connection enthält also nur die Information, wie eine logische Verbindung auf einer bereits bestehenden, physischen Verbindung zustandekommen kann.

Basisierend auf einer Connection können nun eine oder mehrere **Sitzungen** (**Sessions**) aufgebaut werden. Die Anzahl der Sitzungen wird dynamisch vergrößert, wenn eine entsprechende Nachfrage besteht. Eine Session gibt es ab dem Augenblick, in dem sich eine Region mit den Parametern, die in der Connection festgelegt sind, an eine zweite Region wendet und eine Kommunikation zustande kommt. Sessions sind im SNA-Schichtenmodell der Übertragungsschicht (Transmission Control Layer) zugeordnet. Im OSI-Modell wäre das zwischen der Transport- und der Sitzungschicht einzuordnen.

Auf jeder Session kann wiederum genau eine **Conversation** stattfinden. Conversations haben dynamischen Charakter. Jedesmal, wenn Daten zu einer Region, einem Client oder einem Terminal gesendet werden oder von dort zurückkommen, wird eine Conversation gestartet und nach Beendigung der Anfrage wieder beendet. Es können auch nur soviele Conversations zur gleichen Zeit offen sein, wie Sessions aufgebaut wurden.

Um Informationen auf Systemebene zwischen verschiedenen CICS-Systemen austauschen zu können, gibt es die **Intercommunication Facilities**. Der Überbegriff **Intersystem Communication** (**ISC**) faßt mehrere verschiedene Kommunikationsformen zusammen, welche die Kommunikation zwischen Systemen steuern:

- **Multiregion Operation** (**MRO**), bei der Regions desselben Knotens miteinander in Verbindung stehen, ist eine Kommunikationsform, die nur lokal auf demselben Knoten zur Verfügung steht. Sie wird auch **Interregion Communication** (**IRC**) genannt.

- **Advanced Program to Program Communication** (**APPC**) kann sowohl lokal, als auch zur Kommunikation mit entfernten Systemen verwendet werden.

Bei beiden Methoden dient **SNA LU6.2** (**Systems Network Architecture Logical Unit 6.2**) als Protokoll.

In dieser Umgebung können ausser den systemeigenen, noch gemeinsame Ressourcen definiert werden, auf die andere Systeme zugreifen dürfen. Diese müssen aber den verbundenen Systemen jeweils gegenseitig bekanntgemacht werden. Zur Nutzung fremder Ressourcen stehen im CICS mehrere Möglichkeiten zur Verfügung:

- Startet man am lokalen System eine Transaktion, die physisch nicht lokal definiert ist, sondern nur einen Verweis auf eine Transaktion in einem Remote-System enthält, so spricht man von **Transaction Routing**. Dieses Routing kann auch erreicht werden, indem in einer Region explizit eine **Routing Session** zu einem anderen System aufgemacht wird. Dieser Vorgang ist vergleichbar mit dem Öffnen eines Remote-Terminals auf einem fremden Rechner.

- Eine Systemverbindung auf Programmebene wird mit dem **Distributed Program Link** (**DPL**) erreicht. Hierbei wird die Ausführung von Programmen einer Transaktion in verschiedenen Systemen ermöglicht. Eine Transaktion oder ein Programm im lokalen System ruft dabei ein Programm auf dem entfernten System auf. Dieses Verfahren wird durch einem Remote Procedure Call ausgeführt.

- Beim **Function shipping** kann ein Programm Remote-Ressourcen (Dateien, DLI- und IMS-Datenbanken, und Queues für temporäre und transiente Daten) so verarbeiten, als ob sie lokal zur Verfügung stünden. Anforderungen werden automatisch an das Remote-System weitergeleitet (shipping). Diese Zugriffsmethode ähnelt dem DPL, ist aber eine Ebene tiefer einzuordnen, da hier der Zugriff auf eine konkrete Ressource erfolgt und kein Programmaufruf.

- Mit **Distributed Transaction Processing** (**DTP**) bezeichnet man zwei oder mehrere Transaktionen, die in unterschiedlichen Systemen laufen, aber auf einen gemeinsamen Datenbestand zugreifen. Er erweitert den DPL um die Synchronisationsmöglichkeit von Transaktionen in verschiedenen Systemen. Dafür wird, im Gegensatz zu den zuvor dargestellten Verfahren, kein automatischer Auf- und Abbau von Systemverbindungen unterstützt, sondern die Anwendungen müssen mit speziellen Kommandos selbst diese Aufgaben erfüllen.

2.7 Zusammenfassung

Nachdem in diesem Kapitel neben der Terminologie ein Überblick der wichtigen Standards gegeben wurde, sind in Abschnitt 2.4 die bekanntesten Transaktionsmonitore vorgestellt worden. Dabei konnte man feststellen, daß die meisten Standards nur teilweise oder überhaupt nicht verwendet wurden. Gerade TP-Monitore, die auf einer alten Architektur basieren, stützen sich auf eigene Standards ab und nicht auf die von einer der großen Standardisierungs-Organisationen definierten Schnittstellen. Die Heterogenität der Schnittstellen, über die Transaktionsmonitore mit Fremdsystemen kommunizieren, zusammen mit einer mangelnden Einhaltung der Standards, erschwert das Management solcher Systeme erheblich, da für jedes System, Produkt und manchmal sogar für Implementierungen des selben Produktes auf verschiedenen Plattformen, unterschiedliche Schnittstellen oder gar Konzepte verwendet werden. Am Beispiel des Customer Information Control System von IBM wird im Kapitel 3 noch gezeigt werden, wie sehr sich die Implementierungen desselben Produktes auf den verschiedenen Plattformen unterscheiden.

Kapitel 3

Anforderungen an das Management von TP-Monitoren

In diesem Kapitel werden Szenarien, die beim Betrieb eines CICS-Transaktionsmonitors auftreten, beschrieben und daraus die Anforderungen für das Management gewonnen. Die Szenarien stellen die *use cases* dar, die in der **Object Modeling Technique** (**OMT**) den ersten Schritt bei der Entwicklung eines Objektmodells bilden. In dieser ersten Analyse des Problemfeldes beschränkt man sich auf einfachere Szenarien, da ein Modell eine Vereinfachung der realen Welt darstellen soll, um komplexe Probleme in weniger komplexe Teilprobleme aufzuspalten.

Auf die Modellierung der Use-Cases wurde verzichtet, da eine informelle Darstellung der Szenarien und der daraus abgeleiteten Anforderungen für den Leser verständlicher ist.

Die ermittelten Anforderungen werden in Management-Information und Management-Funktionalität aufgeteilt, um im nächsten Kapitel aus den Anforderungen an die Management-Information die Klassen und Attribute zu entwickeln, und aus der Management-Funktionalität die Methoden der Klassen zu gewinnen.

Das Modell aus den Anforderungen wird nicht alle Bereiche des Management von TP-Monitoren vollständig abdecken, jedoch wird das in Kauf genommen, um das Modell zu vereinfachen. Für die erste Implementierung sollen nur die wichtigsten Attribute und Methoden zur Verfügung gestellt werden, die den größten Teil der Managementaufgaben abdecken. Dadurch kann man viele Parameter, die in TP-Monitoren gesetzt werden, ausklammern, da diese nur für einen kleinen Teil des Management gebraucht werden und nur zur Lösung von sehr speziellen Problemen oder der Optimierung eines Systems dienen.

3.1 Szenarien

Zu den wohl häufigsten Tätigkeiten eines Systemverwalters gehört das Anlegen einer neuen User-ID für das CICS. In den meisten Systemen wird jedoch nicht die eingebaute Sicherheitsfunktion des CICS verwendet, sondern ein externes Sicherheitssystem wie zum Beispiel **Resource Access Control Facility** (**RACF**) oder das **Top Secret System** (**TSS**) in Großrechnerumgebungen, beziehungsweise das **Distributed Computing Environment** (**DCE**) in offenen Systemen. Deshalb ist es schwierig, ein allgemein gültiges Szenario für CICS-Systeme aus dem Bereich des Sicherheits-Management zu beschreiben. Sollte man sich auf das CICS-eigene Sicherheitssystem beschränken, ergeben sich nur Anforderungen, die mit einem der unten beschriebenen Szenarien bereits abgedeckt werden können. Ein Spezialfall einer einfachen Benutzereintragung wäre das Eintragen auf einer bestimmten Route von Systemen wobei die Topologieinformation der Domäne in den Arbeitsvorgang miteinfließt.

Gerade in Entwicklungs- oder Testsystemen muß ein Administrator häufig eine CICS-Region stoppen und sie gleich danach wieder starten, um Verklemmungen oder Endlosschleifen von Programmen, die das System blockieren aufzulösen. Das kann in CICS-Systemen vorkommen, da, wie bereits in der Vorstellung des CICS erläutert, diese ein kooperatives Multitasking unterstützten, wodurch unkooperative Programme nur noch schwer zu unterbrechen sind. Dieses Stoppen und anschließende Starten von Regions ist jedoch ein triviales Problem, da hierfür eigene Kommandos zur Verfügung stehen. Die Variationen, die auftreten können, sind Sperren durch Lock-Dateien und Prozesse, die sich nicht beenden lassen. Die Lock-Dateien, die beim Stoppen eines Systems nicht ordnungsgemäß gelöscht wurden und ein Starten verhindern, sind über Betriebssystem-Kommandos vom Administrator zu löschen, dann kann das System neu gestartet werden. Prozesse, die sich nicht automatisch beenden lassen muß man von Hand beenden. Danach kann kann die Region heruntergefahren und erneut gestartet werden.

Ein sehr komplexes Szenario aus dem Konfigurations-Management, das Installieren einer neuen CICS-Region, wurde nicht als Beispiel herangezogen, da der CICS-Domain-Manager (CDM) vorerst nicht für diesen Einsatzbereich gedacht ist. Der CDM soll ein integriertes Management auf verschiedenen Plattformen ermöglichen und Systemverwaltern die täglichen Aufgaben erleichtern. Das Einrichten eines neuen CICS-Systems ist jedoch eine Aufgabe für Spezialisten für die jeweilige Plattform und wird aufgrund der Komplexität durch die Menge der Subsysteme und der hohen Performance-Anforderungen auf jeden Fall in naher Zukunft nicht durch einen Automatismus zu lösen sein. Man braucht sich dazu nur vorzustellen, daß ein UNIX-Spezialist die VTAM-Definitionen niederschreiben muß, die einem CICS/ESA-System die Kommunikation mit anderen Systemen ermöglichen.

In dieser Arbeit wurde bisher ausschließlich die Server-Seite betrachtet - die Client-Seite also völlig vernachlässigt, da sich die Installation eines CICS-Clients nur auf den Aufruf eines Installationsprogrammes beschränkt und die Konfiguration über eine Initialisierungsdatei vollzogen wird. Die Managementanforderungen an CICS-Clients sind demnach eher gering und damit von untergeordnetem Interesse für diese Arbeit. Die Initialisierungsdateien werden in der Regel über Software-Verteilungs-Programme auf die Clients kopiert und CICS-Clients darüber hinausgehend kaum vom Arbeitsplatz des Administrators gepflegt. Da es sich häufig um Windows- oder OS/2-Rechner handelt, kann man sich auf ihnen auch nicht von fremden Rechnern aus anmelden.

Die folgenden Szenarien wurden ausgewählt, weil sie zu den häufigsten Aufgaben eines Administrators zählen und zusätzlich noch in sich abgeschlossen sind. Viele der anderen Szenarien sind Teilmengen und Spezialfälle der unten beschriebenen Szenarien, wobei gerade beim Fehlermanagement die Variationen der Ursachen nahezu beliebig groß sind.

In den Szenarien wird der Einfachheit wegen nur von den CICS-Varianten für MVS/ESA, AIX und OS/2 ausgegangen, die am weitesten verbreitet sind. Dies ist ausreichend, um die Arbeitsschritte zu beschreiben und die Heterogenität der Werkzeuge zur Manipulation der Systemtabellen aufzuzeigen. Ebenso kann man schon an diesen Beispielen die Unterschiede in der Art und Weise der Definitionen aufzeigen.

Vorbemerkung: Definitionen werden im CICS/ESA und im CICS for OS/2 mit der Transaktion *CEDA* eingetragen. Sie dient als Editierwerkzeug für die Systemtabellen, in denen die Definitionen eines CICS-Systems gespeichert werden. Unter IBM AIX werden Betriebsmittel mit dem graphischen Werkzeug `smit`, bzw. `smitty` als textorientierte Variante, festgelegt. Seit Juni 1996 hat IBM ein Produkt, den *IBM CICS System Manager*, der eine graphisch Benutzerschnittstelle bietet und die meisten Definitionen editieren kann. Der Analyse dieser Werkzeuge ist jedoch ein eigenes Kapitel gewidmet.

3.1.1 Konfigurationsmanagement: Herstellen einer Verbindung zwischen CICS-Systemen.

Das Herstellen einer Verbindung zwischen zwei CICS-Systemen beinhaltet mehrere Einzelschritte. Zuerst braucht das lokale System einen Namen, unter dem es im Netz bekannt ist. Diesen Namen benötigt man auch für das entfernte System, zu dem man die Verbindung aufbauen will. Bei Kommunikation über TCP/IP benötigt man darüberhinaus für beide Systeme die IP-Adresse, beziehungsweise den Aliasnamen und den Port, über den kommuniziert werden soll. Danach muß man sicherstellen, daß auf beiden Systemen ein Listener zur Verfügung steht. In diesem Szenario wird davon ausgegangen, daß auf dem entfernten System bereits ein Listener definiert wurde. Nachdem die Connections auf beiden Sytemen definiert sind, kann man die Parameter für die Sessions in beiden Systemen festlegen. Als Abschluß der Definition wird getestet, ob sich Sessions aufbauen lassen und eine Kommunikation zustande kommt.

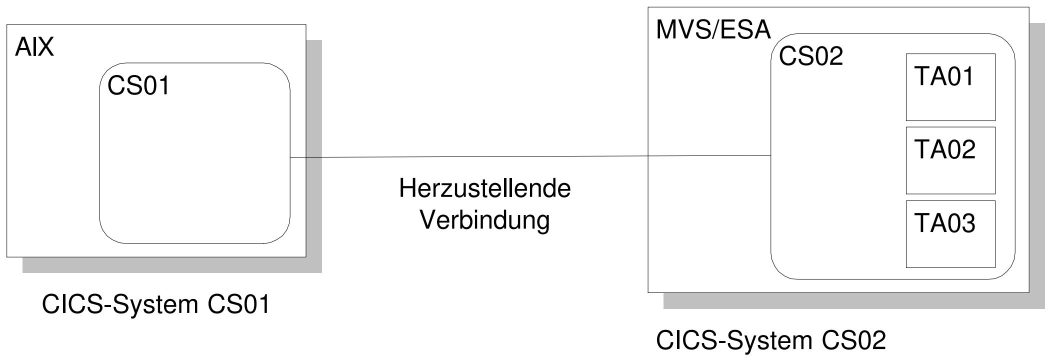

Abbildung 3.1: Szenario 'Herstellen einer Verbindung von CICS-System CS01 zu CICS-System CS02' (Konfigurationsmanagement)

Arbeitsschritte

1. Ermitteln des Namens des lokalen Systems
   Ein CICS-System hat mehrere Namen:
   - *sysid:* Lokaler Name des CICS-Systems. Wird im CICS/ESA und im CICS for OS/2 in der *System Initialisation Table (SIT)*, im CICS/6000 im Attribut *Region System Identifier* der *Region Definition*, eingetragen.
   - *applid:* Unter diesem Netz-Namen ist das CICS-System im *Intercommunication Network* bekannt und wird zum Beispiel von den CICS-Clients über diesen Namen identifiziert. Wird im CICS/ESA und im CICS for OS/2 in der *System Initialisation Table (SIT)*, im CICS/6000 im Attribut *Region Name* der *Region Definition*, eingetragen.
   - *VTAM generic resource name:* (Nur CICS/ESA) Für den Fall, daß das CICS-System in einer *VTAM generic resource group* definiert ist, wird dieser Name wird mit einem `VTAM APPLID`-Kommando im VTAM selbst festgelegt.
   - *netname:* Der *hostname* unter TCP/IP.
2. Ermitteln des Namens des entfernten Systems
   - *applid, sysid:* Aus der *SIT* beziehungweise aus den *Region Definitions* des entfernten Systems.
   - TCPIP: Zusätzlich TCP/IP-Adresse (bzw. Hostname) und Port des entfernten Systems. Dabei gibt es für OS/2 eine Besonderheit: CICS for OS/2 ermittelt aus *hostname* und *portnumber* einen generischen Namen. Zum Beispiel wird aus `aix5cicsland.com` und Port `1435` der Name `05G2OXPN`. Das läßt sich mit dem Kommando `cicstcpnetname` nachvollziehen.
3. Definition der Connection vom lokalen zum entfernten System
   - CICS/ESA: Eine *Connection* wird entweder durch das Kommando `CEDA DEFINE CONNECTION` oder einen Eintrag in der *Connection and Session Table* (*TCS*) definiert.
   - CICS/6000: Definition einer Connection in den *Communication Definitions*
   - CICS for OS/2: Eintrag in der *Connection and Session Table* (*TCS*).

4. Definition der Connection vom entfernten zum lokalen System
   Siehe *Definition der Connection vom lokalen zum entfernten System.*

5. Nur in CICS/ESA notwendig:
   Definition von *Sessions* auf einer *Connection* in der *TCS* mithilfe von `CEDA`

6. Aktivieren der Connection

   - CICS/ESA:
     und CICS for OS/2: Mit der Transaktion `CEMT` kann man den Befehl `SET CONNECTION ACQUIRED` absetzen. Dieser Befehl aktiviert eine definierte *Connection*, jetzt können auf dieser *Connection* mehrere *Sessions* aufgebaut werden.
   - CICS/6000: keine derartige Funktionalität vorhanden, da *Sessions* automatisch aufgebaut werden.

7. Testen der Verbindung (Session und Connection)

   - CICS/ESA: Aufbau einer *Routing Session* zum fernen System. Mit der Transaktion `CEMT` und den Parametern `INQUIRE CONNECTION` kann man den Status aller definierten *Connections* abfragen.
   - CICS/6000: Aufbau einer *Routing Session* zum fernen System.
   - CICS for OS/2: Aufbau einer *Routing Session* zum fernen System. Auch hier steht die Transaktion `CEMT` zur Verfügung.

Anforderungen

- Management-Information
  - Identifikatoren des Systems in der Domäne (1, 2)
  - Definitionen der *Connections* zwischen jeweils zwei Systemen (3, 4)
  - Definitionen der *Sessions* einer *Connection* (5)
- Management-Funktionalität
  - Ermitteln des Namens eines bestimmten Systems (1, 2)
  - Ändern der Definitionen von *Connections* (3, 4)
  - Aktivieren der *Connections* zwischen zwei Systemen (6)
  - Ändern der Definitionen von *Sessions* (5)
  - Aufbauen einer Verbindung zum entfernten System bzw. Start eines Testwerkzeuges für Verbindungen (7)

3.1.2 Fehlermanagement: CICS-Anwendung startet nicht.

Das Fehlerszenario 'Die CICS-Anwendung XYZ läßt sich vom Client nicht starten' (Abb. 3.2) ist eine typische Situation, wenn zum Beispiel Sachbearbeiter per CICS-Client versuchen, eine CICS-Anwendung zu starten und feststellen müssen, daß sie nicht arbeiten können. Daraufhin erhält meist die Hotline des Benutzerservice oder ein Operator einen Anruf. Der Operator prüft daraufhin von 'innen nach aussen' die beteiligten Systeme und erhält Informationen aus den Log-Dateien. Die Vorgehensweise, daß zuerst die letzte Station der Kette (die AOR, in der die CICS-Anwendung läuft) geprüft wird, hat den Hintergrund, daß meist Softwarefehler zu Funktionsstörungen führen. Diese Fehler treten dann in der AOR auf und sind meist auf den Abbruch einer Transaktion oder einer Zeitüberschreitung bei Datenbankzugriffen zurückzuführen. Für den Fall, daß der Fehler auf dem Weg vom Client zur AOR liegt, muß natürlich die ganze Route bis zum Client zurückverfolgt werden. Man muß herausfinden, ob sich die Systeme auf der Route überhaupt noch in einem aktiven Zustand befinden oder vielleicht so stark ausgelastet sind, daß Zeitüberschreitungen auftreten.

Selbverständlich muß ein inaktives System wieder in einen aktiven Status überführt und aktive Systeme deaktiviert werden. Falls Verbindungen definiert sind, ist es von Bedeutung, ob auf diesen Verbindungen auch tatsächlich Sitzungen aufgebaut werden. Dabei ist es für Systemtests von Vorteil, wenn Verbindungen zwischen CICS-Systemen temporär inaktiv gesetzt werden können, um zum Beispiel eine bestimmte Route von einem System zum anderen zu erzwingen. Im Server muß also auch der Kommunikations-Mechanismus, der die Protokollumsetzung macht, von außen zugreifbar sein. Die entscheidende Information ist jedoch hierbei, welche Transaktionen, Programme, Dateien und Systemresourcen zu einer bestimmten Anwendung gehören, ob **Affinitäten** bestehen. Affinitäten sind Abhängigkeiten aller Art, die verhindern können, daß Transaktionen, Programme und Anwendungen lokationstransparent ausgeführt werden können.

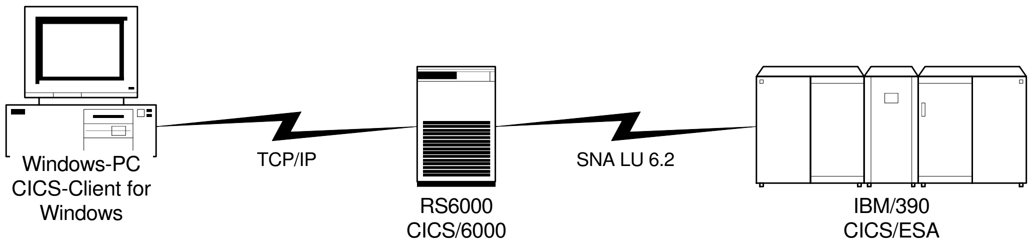

Abbildung 3.2: Szenario 'CICS-Anwendung startet nicht.'

Arbeitsschritte

1. Zuständige AOR identifizieren.
   Bis jetzt bietet sich in keinem CICS-System die Möglichkeit, die Zuordnung einer Region zu einer bestimmten Anwendung als Systeminformation zu speichern. Diese Zuordnung geschieht implizit, bei der Installation der Programme und Transaktionen. Es obliegt jedem Adminstrator, wo und wie dokumentiert wird in welcher Region eine Anwendung läuft. Aus diesem Grund ist es sehr schwierig herauszufinden, welche Region gerade ein Problem hat, wenn nur die Information zur Verfügung steht, daß eine bestimmte Anwendung nicht gestartet werden kann. Die *applid* des Gateway-Servers kann man zur Not aus der Initialisierungsdatei des *CICS-Client* ermitteln (wenn sichergestellt ist, daß diese Anwendung vor kurzer Zeit mit dieser Konfiguration noch lief) und dann in den Definitionen der Transaktionen und Programme etwaige Routing-Einträge suchen, die auf andere CICS-Regions verweisen. Das ist jedoch eine sehr unsichere Methode, da es verschiedene Routing-Mechanismen gibt (Default-Routing, Explizites Routing, Routing-Sessions) und daher die Spuren nicht immer nachvollzogen werden können.

2. Prüfen der Funktion der AOR.
   Um die Funktion einer Region zu testen, versucht man sich lokal an dieser Region mit einem CICS-Terminal anzumelden. Funktioniert das nicht, kann man noch in den Log-Dateien nachsehen. Die Vorgehensweise dazu wird im nächsten Punkt genauer beschrieben. In den Log-Dateien des CICS-Systems kann aufgrund der Meldungen auf den Zustand der *Region* schließen. Ist die letzte Meldung beispielsweise ein `shutdown of region xyz completed successfully`, wurde diese *Region* heruntergefahren und nicht wieder gestartet.

3. Log-Datei-Einträge der AOR auf Fehlermeldungen hin analysieren.
   Zum Beispiel deuten Meldungen, daß Transaktionen abgebrochen wurden (*ABEND*-Codes [1]) oder Dateisysteme voll sind, auf mögliche Ursachen hin. Dabei muß zuerst das *Terminal* ermittelt werden, über das sich ein Benutzer am *CICS-System* angemeldet hat. Bei diesem Schritt bekommt man gleichzeitig den *netname* des *CICS-Client*, unter dem er dem *Transaction-Server* bekannt ist.

   - CICS/ESA:
     (a) Die Transaktion *CEBR* ermöglich die Anzeige der Log-Dateien, die in eine *Data-Queue* geschrieben werden. Dieses Werkzeug ist direkt aus dem CICS heraus zu verwenden.
     (b) Mit der Anwendung *SDSF* ist es möglich, aus dem *TSO*, einer Art Kommandointerpreter unter *MVS*, den *Joboutput* einer Anwendung formatiert anzuzeigen. Da CICS selbst auch eine Anwendung ist, kann man mit SDSF die CICS-Log-Dateien ansehen.
   - CICS/6000: Log-Dateien werden im CICS/6000 als einfache ASCII-Dateien in das /var-Verzeichnis geschrieben und sind dort mit jedem Editor zu lesen. Es gibt mehrere Log-Dateien, die wichtigste ist `console.log`, in der zum Beispiel der Start des *CICS-Systems*, Datenbank-Anforderungen und Fehlermeldungen von Transaktionen geschrieben werden. Daneben gibt es noch Dateien für den automatischen Terminal-Installer, der das Anmelden von *CICS-Clients* ermöglicht, und für Terminals, die direkt per Emulation an das CICS gehen.
   - CICS for OS/2: Die Log-Datei `CICSMSG.LOG` im Installationsverzeichnis des *CICS-Servers* enthält alle Informationen über Terminals, Namen von Clients und Systemen. Sie ist im ASCII-Format und kann mit einem Editor gelesen werden.

4. Bei Fehlern die Ursachen prüfen.
   (z. B. Transaktionen, Programme oder Files, die zur Anwendung gehören, prüfen.)
   Es gibt bis jetzt keine Informationsquelle, die alle Betriebsmittel liefert, die einer Anwendung zugeordnet sind! Ein erstes Kriterium bei einem bestehenden System könnten die **CICS-Groups** sein, die eine Strukturierungsmöglichkeit der Definitionen bieten. CICS gestattet es, Operationen (installieren, verfügbarmachen, usw.) auf gesamte Gruppen auszuführen. Wenn Anwendungen so im System definiert wurden, daß jede Anwendung einer Gruppe angehört, läßt sich diese Information natürlich nutzen, verbindlich ist sie jedoch nicht. Findet man in einer Log-Datei einen Eintrag, daß eine bestimmte Transaktion abgebrochen wurde, also der TP-Monitor die Transaktion mit einem *ABEND*-Code beendet hat.

5. Prüfen der Funktion der TOR.
   (Siehe *Prüfen der Funktion der AOR*)

6. Prüfen der Verbindung zwischen TOR und AOR.

   - CICS/ESA: Mit der Transaktion `CEMT` kann man mit den Parametern `INQUIRE CONNECTION` den Status aller definierten *Connections* anzeigen.
   - CICS/6000: Im CICS/6000 kann man mit `smit` in den *Communication Definitions* den Status von *Connections* und *Sessions* abfragen. Ihr Vorhandensein ist Voraussetzung für einen erfolgreichen Verbindungsaufbau.
   - CICS for OS/2: Wie im CICS/ESA gibt es auch im CICS for OS/2 die Transaktion `CEMT`.

7. Protokoll-Verbindung zum Client prüfen.

   - TCP/IP: Wenn der *CICS-Client* über TCP/IP mit dem *CICS-Server* kommunizieren, wie in diesem Szenario, kann man die Werkzeuge `ping` oder auch `traceroute` verwenden, um die Protokollverbindung zu testen.

[1] ABEND heißt abandon ended. Bestimmte *ABEND-Codes* weisen zum Beispiel darauf hin, daß eine Transaktion angefordert wurde, die in dieser *CICS-Region* nicht definiert ist oder im Falle des *Transaction-Routing* eine falsche *sysid* angegeben war. Diese Eintragungen sind dann im betreffenden System zu machen.

  - NetBios: Es bietet sich nur die Möglichkeit, einen Requester starten und eventuelle Fehlermeldungen während des Startes zu analysieren. Testwerkzeuge, wie `ping` oder `traceroute` gibt es für dieses Protokoll nicht. prüfen.
  - SNA: Hier gilt bezüglich der Testwerkzeuge das gleiche, wie für NetBios. Eine Auswertung von Fehlermeldungen ist manchmal schwierig, weil nicht zwangsläufig eine Fehlermeldung angezeigt wird. In diesem Fall dauert der Aufbau einer Session länger als üblich. Der Status einer Session ist dann im System über eine längere Zeitspanne hinweg 'Im Aufbau', was mit der Transaktion *CEMT* nachgeprüft werden kann.

8. Konfiguration des Clients prüfen.
  - Einträge in der Konfigurationsdatei `CICSCLI.INI` prüfen, zum Beispiel, ob die Servernamen und TCP/IP-Adressen stimmen und die richtigen Portnummern eingetragen sind.
  - TCP/IP-Konfiguration überprüfen, ob die richtigen Router eingetragen sind.
  - Testen, ob die Namensauflösung funktioniert, zum Beispiel der richtige DNS-Server eingetragen ist.

Anforderungen

- Management-Information
  - Name der AOR, der eine Anwendung zugeordnet ist (1)
  - Log-Dateien einer *Region* (2, 3, 4)
  - Transaktionen, Programme, Dateien usw., die eine Anwendung bilden.(4)
  - Definitionen von Transaktionen, Programmen, Dateien usw. einer Anwendung (4)
  - User, der die Anwendung gestartet hat.
  - Terminal, mit dem sich der User an einem System angemeldet hat.
  - Verwendetes Protokoll zwischen *CICS-Client* und *TOR* (7)
  - Konfigurationinformation eines *CICS-Clients* (8)
- Management-Funktionalität
  - Ermitteln des Namens einer AOR, der eine bestimmte Anwendung zugeordnet ist. (1)
  - Funktiontest einer *Region* durch Anmelden oder Auswertung einer Log-Datei. (2, 5)
  - Anzeigen einer Log-Datei. (3)
  - Anzeigen des Status von Transaktionen, Programmen, Dateien usw. (4)
  - Prüfen einer Verbindung zwischen zwei *Regions* (6)
  - Testen der Protokollverbindung zwischen *CICS-Client* und *TOR* (7)
  - Anzeigen und Ändern der Konfigurationsdatei des *CICS-Client* (8)

3.1.3 Szenario: Installieren neuer CICS-Programme

Bei der Installation eines Anwendungsprogrammes sind detaillierte Informationen zu jedem Programm-Modul und zu jeder Transaktion notwendig. Die wichtigsten sind Lokation, Version und Remote-Eintragungen mit der Remote-System-ID. Ebenso müssen Anpassungen in den Subsystemen veranlaßt werden, wie zum Beispiel Änderungen an Steuertabellen im Datenbanksystem oder das Eintragen von Benutzern im Sicherheitssystem. Die Thematik Sicherheitssystem wird in dieser Arbeit nicht behandelt, da auf den meisten Plattformen nicht das CICS für die Sicherheit zuständig ist, sondern diese Aufgaben ein externes Sicherheitssytem übernimmt (auf hostbasierten Systemen wird **RACF** (**Resource Access Control Facility**) [2] , auf offenen Systemen das **DCE** verwendet).

[2] RACF ist in mainframe-basierten Umgebungen ein weitverbreitetes, externes Sicherheitssystem. RACF ist nicht Bestandteil von CICS oder eines anderen TP-Monitors

Für die Menge aller Transaktionen, Programme und User benötigt man detaillierte Informationen, auf welchen Systemen sie definiert sind. Dazu muß man ein geeignetes Modell finden, um diese Daten mit Lokationsinformation in einer Klasse abbilden zu können. Um mit der Vielfalt der Informationen überhaupt zurecht zu kommen, muß die Information nach bestimmten Kriterien strukturiert und gefiltert werden. Beispielsweise muß es möglich sein, für ein bestimmtes Programm herauszufinden, wo es physisch liegt. Programme können bekanntlich auf vielen CICS-Systemen definiert sein und dabei nur einmal auf einem Datenträger existieren, deshalb muß man auch herausfinden können, wo dieses Programm zusätzlich 'remote' aufgerufen werden kann. Diese Forderung gilt prinzipiell für alle Betriebsmittel, die mit gleichem Namen in der Domäne mehrfach vorhanden sind und die gleiche Funktionalität repräsentieren.

Arbeitsschritte

1. Sicherstellen, daß auf betroffene Ressourcen nicht zugegriffen werden kann, solange das neue Programm eingespielt wird.
   Transaktionen, Programme und Dateien können vor Zugriff und Ausführung geschützt werden:
   - CICS/ESA: `CEMT SET {TRANSACTION|PROGRAM|FILE} DISABLED`
   - CICS/6000: Setzen des Attributs `Transaction enable status` auf `disabled`, in der *Ressource Definition Transaction*.
   - CICS for OS/2: `CEMT SET {TRANSACTION|PROGRAM|FILE} DISABLED`
2. Falls bereits eine frühere Version installiert wurde.
   - Ermitteln der aktuellen Programmversion.
   - Erforderliche Version für den gewünschten Update der aktuellen Version prüfen.
3. Falls ein Programm erstmalig installiert wird.
   - Festlegen der AOR für diese Anwendung. Das ist die Aufabe eines Administrators und wird hoffentlich an geeigneter Stelle dokumentiert. Der *CICS Domain Manager* soll diese Information verwalten, um sie für andere Problemstellungen (siehe oben) nutzen zu können.
   - Zugriff auf Systeminformationen der verschiedenen AOR, um Vorraussetzungen der Anwendung in einer AOR zu prüfen.
4. Einspielen der Programme und Dateien der neuen Anwendung.
5. Anlegen der *Resource Definitions* für Programme, Transaktionen, Dateien, Verbindungen und Datenbankzugriffe.
6. Aktivieren der neuen Programme, Transaktionen und Dateien.
   Transaktionen, Programme und Dateien können mit folgenden Kommandos aktivert werden:
   - CICS/ESA und CICS for OS/2: `CEMT SET {TRANSACTION|PROGRAM|FILE} ENABLED`
     Anschließend wird mit dem Kommando `CEMT SET PROGRAM {NEWCOPY|PHASEIN}` eine neue Kopie des Programms in den Speicher geladen. Damit wird sichergestellt, daß nicht mit einer Vorgängerversion weitergearbeitet wird, die das CICS im Speicher gehalten hat.
   - CICS/6000: Setzen des Attributs `Transaction enable status` auf `enabled`, in der *Ressource Definition Transaction*.
7. Bei CICS/ESA veranlassen, daß für DB2 die Remote Control Table (RCT) gepflegt werden.
   Um Datanbankzugriffe zu beschleunigen, wird für eine DB2-Datenbank vor dem Einsatz ein sogenanntes *Bind-File* angelegt, in dem Indizes und Berechtigungen aller möglichen Benutzer einer Tabelle hinterlegt werden. Diese Bind-Dateien werden dann die *DB2 RCTs* eingetragen.
8. Zugriff auf die neue Version ermöglichen. Dieser Arbeitsschritt läßt sich in folgende Teilbereiche aufteilen:

- Eintragung der Definitionen für die AOR für die Anwendung in der TOR.
- Benutzer für die neue Version berechtigen.
  - Eintragung der Benutzer für die Anwendung im Sicherheitssystem veranlassen. (Zugriff auf Programme, Transaktionen, Systembibliotheken, Queues, usw.)
  - Eintragung der Benutzer bzw. der Anwendung im Datenbanksystem veranlassen.

Anforderungen

- Management-Information
  - Von der Veränderung betroffene Transaktionen, Programme, Dateien usw. (1, 6)
  - Installierte Version (2)
  - Aktuelle AOR für eine Anwendung (3)
  - Gewünschte AOR für eine Anwendung (3)
  - Verwendetes Sicherheitssystem und dessen Administrator (8)
  - Verwendetes Datenbanksystem und dessen Administrator (7,8)
- Management-Funktionalität
  - Transaktionen, Programme, Dateien usw. deaktivieren (1)
  - Ermitteln der aktuellen Programmversion (2)
  - Ermitteln der erforderlichen Version für den Update der aktuellen Version (2)
  - Unterstützung bei der Festlegung der Installations-AOR (3)
  - Übertragen von Anwendungen von Transferdatenträgern in das System (4)
  - Anlegen der *Ressource Definitions* in TOR und AOR (5, 8)
  - Transaktionen, Programme, Dateien usw. aktivieren. (6)
  - Kommunikationsmechanismus zum zuständigen Administrator des Datenbanksystems (7, 8)
  - Kommunikationsmechanismus zum zuständigen Administrator des Sicherheitssystems (8)

3.2 Zusammenfassung

Durch die Ausdehnung der Analyse der Szenarien auf verschiedene Betriebssysteme erkennt man schnell, daß die Management-Problematik hier nicht gelöst ist, sobald auf einer Plattform ein adäquates Modell gefunden wurde. Vielmehr braucht man für die verschiedenen Plattformen Spezialisierungen des Modells. Obwohl bei der Auswahl der Szenarien darauf geachtet wurde, daß diese in sich abgeschlossen sind und möglichst allgemeine Problemstellungen, die häufig auftreten beschreiben, werden sie komplizierter, sobald man die verschiedenen Betriebssysteme, hier nur drei, berücksichtigen will. Um die Übersichtlichkeit etwas zu erhalten, lag der Schwerpunkt deshalb auf der Betrachtung der AIX-Implementierung, wo auch die prototypische Implementierung erfolgen soll. Die gesamten Anforderungen, die aus den oben genannten Szenarien hervorgehen, hier nocheinmal in Zusammenfassung.

Management-Information

- Identifikatoren der Systeme in der Domäne
- Definitionen der *Connections* zwischen jeweils zwei Systemen
- Definitionen der *Sessions* einer *Connection*
- Name der AOR, der eine Anwendung zugeordnet ist
- Log-Datei einer *Region*
- Transaktionen, Programme, Dateien usw., die eine Anwendung bilden

- Definitionen von Transaktionen, Programmen, Dateien usw. einer Anwendung
- Verwendetes Protokoll zwischen *CICS-Client* und *TOR*
- Konfigurationinformation eines *CICS-Clients*
- Von der Veränderung betroffene Transaktionen, Programme, Dateien usw.
- Installierte Version
- Aktuelle AOR für eine Anwendung
- Gewünschte AOR für eine Anwendung
- Verwendetes Sicherheitssystem und dessen Administrator
- Verwendetes Datenbanksystem und dessen Administrator

Management-Funktionalität

- Ermitteln des Namens eines bestimmten Systems
- Ändern der Definitionen von *Connections*
- Aktivieren der *Connections* zwischen zwei Systemen
- Aufbauen einer Verbindung zum entfernten System bzw. Start eines Testwerkzeuges für Verbindungen
- Ermitteln des Namens einer AOR, der eine bestimmte Anwendung zugeordnet ist
- Funktiontest einer *Region* durch Anmelden oder Auswertung einer Log-Datei
- Anzeigen einer Log-Datei
- Anzeigen des Status von Transaktionen, Programmen, Dateien usw.
- Prüfen einer Verbindung zwischen zwei *Regions*
- Testen der Protokollverbindung zwischen *CICS-Client* und *TOR*
- Anzeigen und Ändern der Konfigurationsdatei des *CICS-Client*
- Transaktionen, Programme, Dateien usw. deaktivieren
- Ermitteln der aktuellen Programmversion
- Ermitteln der erforderlichen Version für den Update der aktuellen Version
- Unterstützung bei der Festlegung der Installations-AOR
- Übertragen von Anwendungen von Transferdatenträgern in das System
- Anlegen der *Ressource Definitions* in TOR und AOR
- Transaktionen, Programme, Dateien usw. aktivieren
- Kommunikationsmechanismus zum zuständigen Administrator des Datenbanksystems
- Kommunikationsmechanismus zum zuständigen Administrator des Sicherheitssystems

Kapitel 4

Ein anwendungsbezogenes Objektmodell

Das anwendungsbezogene Objektmodell kann nun aus den Anforderungen abgeleitet werden. Dieser zweite Modellierungsschritt im Rahmen der OMT, das Design des Objektmodells basierend auf den Anforderungen, soll eine Abbildung der problemorientierten Sichtweise auf ein implementierungsorientiertes Konzept ergeben. Die Anforderungen des Kapitels 3 dienen bei dieser Umsetzung als Grundlage.

Nach der Identifikation der Klassen werden in diesem Kapitel aus der Management-Information die Attribute und aus der Management-Funktionalität die Methoden abgeleitet. Eine allgemeinere Beschreibung der OMT nach Rumbaugh folgt im Abschnitt 4.1, um den Gesamtzusammenhang der einzelnen Modellierungsschritte aufzuzeigen und mit den Kapiteln dieser Arbeit in Relation zu setzten.

4.1 Generelle Vorgehensweise für die Gewinnung des Modells nach OMT

Wie bei Software-Entwurfsverfahren üblich, sind bei der Object Modeling Technique die Schritte der Software-Entwicklung in einem Lebenszyklus organisiert, der aus mehreren Entwicklangsphasen besteht. Die OMT-Methode unterstützt den gesamten Software-Lebenszyklus von der Formulierung des Problems über die Systemanalyse, das Design bis zur Implementierung. Sie besteht aus den Phasen der Analyse, des Systemdesigns und des Objektdesigns. Die generelle Durchgängigkeit objekt-orientierter Entwicklung ermöglicht die Wiederholung einzelner Entwurfsschritte auf jeder Detaillierungsebene. Man kann also von einem iterativen Entwicklungsprozeß sprechen. In den folgenden Abschnitten werden die verschiedenen Phasen mit ihren Aktivitäten genauer beschrieben.

4.1.1 Analyse

Bei der Analyse, dem ersten Schritt der OMT, entwirft man ein präzises, verständliches und korrektes Modell der realen Welt. Dafür untersucht man die Anforderungen, abstrahiert wichtige Eigenheiten und verschiebt die Betrachtung weniger wichtiger Details. Das Resultat der Analysephase sollte ein tiefgehendes Verständnis des Problembereiches als Vorbereitung auf die Designphase sein. Die Analyse beginnt mit der Problembeschreibung des Kunden oder möglicherweise auch der Software-Entwickler selbst.

Das Ergebnis der Analysephase ist ein formales Modell, das die drei wesentlichen Aspekte eines Systems abdeckt:

- Die Objekte und ihre Beziehungen
- Der dynamische Kontrollfluß
- Die funktionale Transformation der Datenstrukturen

Das dynamische Modell und das Funktionsmodell werden in dieser Arbeit nicht behandelt. Die Kontrollstrukturen eines Managers sind relativ einfach und Zustandsänderungen finden kaum statt. Deshalb wurde auf die Modellierung des dynamischen Modells und des Funktionsmodells in dieser Arbeit verzichtet. Die identifizierten Objekte sind die Anforderungen, die im Kapitel 3 anhand der Szenarien ermittelt wurden.

4.1.2 Systemdesign

Rumbaugh unterscheidet zwischen zwei Designphasen: dem Systemdesign und dem Objektdesign. Das Systemdesign beinhaltet Entscheidungen über die Organisation eines Systems in Subsysteme, der Zuweisung von Subsystemen auf Hardware- und Software-Komponenten, sowie Grundkonzepte für das anschließende detaillierte Design. Der Systementwickler muß dabei folgende Entscheidungen treffen:

- Aufteilen eines Systems in Subsysteme
- Identifizieren von Nebenläufigkeit
- Zuweisen der Subsysteme an Prozessoren und Prozesse
- Speicherung von Daten
- Bestimmen der Implementierung der Steuerung

Das Systemdesign findet in diesem Kapitel, durch Abbildung der Anforderungen auf ein adäquates Objektmodell, statt.

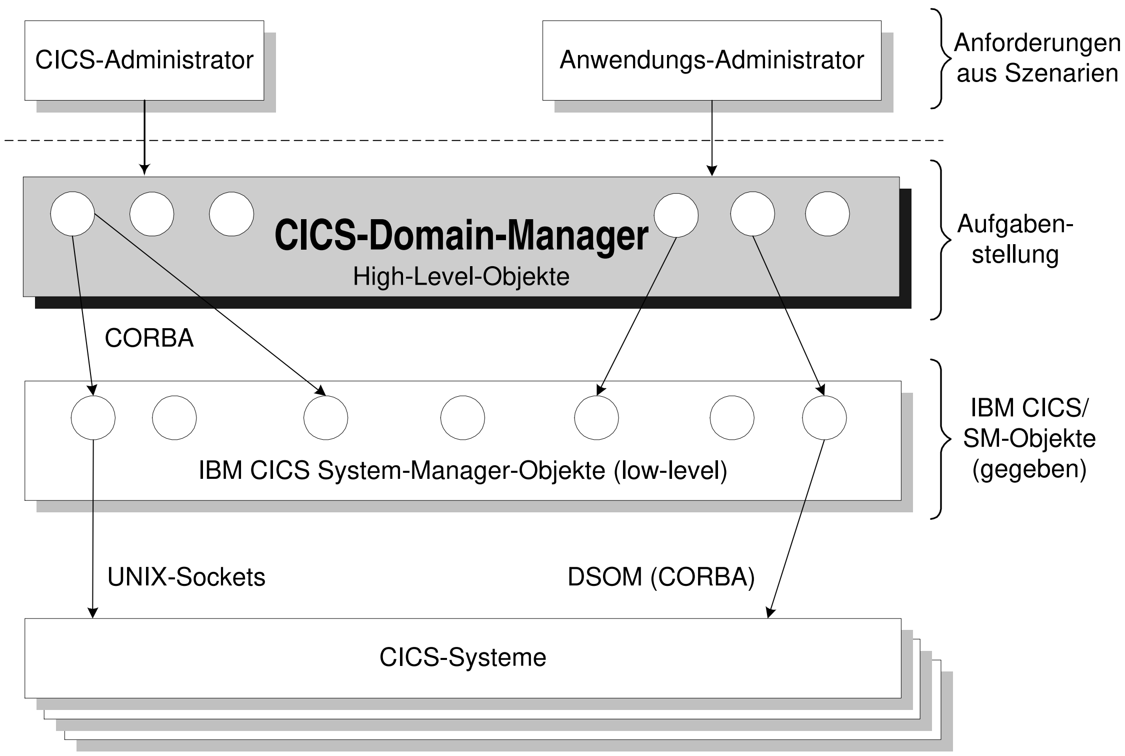

Abbildung 4.1: Einordnung der Arbeit

4.1.3 Objektdesign oder Implementierung

In der Phase des Objektdesigns werden die vollständigen Definitionen der Klassen und Assoziationen, die bei der Implementierung verwendet werden sowie die Schnittstellen und die Algorithmen der Methoden festgelegt. Interne Klassen zur Implementierung und Optimierung von Datenstrukturen und Algorithmen werden hinzugefügt. Die bei der Analyse identifizierten Klassen dienen als Gerüst des Objektdesigns, der Entwickler muß jedoch unter verschiedenen Möglichkeiten der Realisierung auswählen.
Während des Objektdesigns sollte der Designer folgende Schritte ausführen:

- Kombinieren der drei Modelle
- Entwurf von Algorithmen
- Optimierung des Designs
- Implementierung der Steuerung
- Anpassen der Vererbung
- Entwurf der Assoziationen
- Physikalische Gruppierung
- Dokumentieren des Designs

Im Kapitel 6 werden die Objekte, die während des Systemdesigns identifiziert wurden, wenn auch nur prototypisch, implementiert.

4.2 Namenskonventionen

Die Klassen werden folgendermaßen benannt: Für den Teilbereich CICS mit dem Präfix CDM_ für CICS Domain Manager, um weitere Klassen für IMS (IDM_), UTM (UDM_), Encina (EDM_) und Tuxedo (TDM_) namentlich zu unterscheiden. Die übergeordeneten Klassen, die es für alle TP-Monitore gemeinsam gibt, werden das Präfix DTM_ für Domain Transaction-processing Manager tragen. Die Spezialisierungen der Klassen für die jeweiligen Plattformen tragen den Namen des Betriebssystems als zweiten Präfix. Daraus ergibt sich beispielsweise für eine Klasse eines CICS-Systems auf OS/2 *CDM_OS2_System*. Zur Erhaltung der Übersichtlichkeit werden, gemäß den üblichen Konventionen der OMT nach Rumbaugh, alle Klassennamen groß geschrieben. Attribute und Methoden werden klein geschrieben, wenn ein Name aus mehreren Worten besteht, wird der Name zusammengeschrieben und jedes Wort innerhalb des Namens mit einem Großbuchstaben begonnen. Methoden werden durch nachgestellte Klammern von Attributen unterschieden.

4.3 Szenario 1: Herstellen einer Verbindung zwischen CICS-Systemen

Eine Verbindung zwischen zwei Systemen herzustellen, setzt voraus, diese Systeme zu identifizieren. Die erste Klasse repräsentiert also ein CICS-System (*System*), das in einer bestimmten Domäne liegt.

Ein komplett neues Aufsetzten eines Systems ist eine Aufgabe, die viel Erfahrung im Umgang mit dem jeweiligen TP-Monitor erfordert. Der **DTM** stellt jedoch ein Hilfsmittel dar, mit dem eine bestehende Domäne mit TP-Monitoren auf verschiedenen Plattformen administriert werden kann. Für den Einsatzbereich des **Domain Transaction Manager** ist es momentan nicht notwendig, ein System von Grund auf aufzubauen, das heißt, man kann sich im Moment darauf beschränken, die Systemdefinitionen als Informationsquelle zu nutzen. Man braucht außerdem nicht die gesame Anzahl von Attributen für ein System, sondern kann sich für dieses Szenario auf wenige beschränken.

Um eine Region zu identifizieren braucht man den Region-Namen (*sysName*) und den Netznamen (*netName*). Diese Namen benötigt man jeweils für beide Systeme zwischen denen die Verbindung definiert werden soll. Um nun die beiden Verbindungs-Hälften zu definieren, gibt es die Klasse (*HalfConnection*).

Zur Identifikation muß man ihnen jeweils einen Namen (*connectionName*) geben und festlegen, über welches Protokoll (*connectionProtocol*) die Verbindung laufen, ob und welches Gateway (*gatewayName*) und welche Zeichensatztabelle (*remoteCodepage*) auf der Gegenseite verwendet werden soll. Das ist notwendig, weil auf beiden Seiten unterschiedliche Zeichensätze dazu führen, daß zum Beispiel Paßwörter zur Verifikation der Berechtigungen nicht als richtig erkannt werden, obwohl die richtigen Paßwörter übermittelt wurden. In diesem Zusammenhang wird bei einer Halb-Verbindung angegeben, welche Sicherheitsmechanismen verwendet werden sollen, ob Userid, Paßwort, beides oder nichts übertragen werden soll (*security*). Diese Einstellungen müssen in den beiden zusammengehörenden Halb-Verbindungen übereinstimmen. Außerdem braucht man noch den Namen der DCE-Zelle (*dceCellName*) in der sich das andere System befindet. Um das andere System im Netz eindeutig zu identifizieren, benötigt man bei einer TCP/IP-Verbindung noch die TCP/IP-Adresse (*remTcpAddress*) und den Port (*remPortnr*). Diese Attribute werden aber sinnvollerweise bei dem jeweiligen System nochmals lokal gehalten (*localTcpAddress* und *localPortnr*), da sie auch lokal abgefragt werden sollen, um sie eventuell Anderen mitteilen zu können, die eine Verbindung zu diesem System aufbauen wollen. In SNA-Netzen ist die Identifizierung bereits über den Netznamen (*remNetName*) und den Region-Namen (*remSysName*) möglich, da diese eindeutig sind. Weil man eine Verbindung nur dann definieren kann, wenn beide beteiligten Systeme gestartet sind und laufen, braucht man eine Möglichkeit, festzustellen in welchem Zustand sich ein System gerade befindet (*systemStatus*).

Sollte in einem System während der Laufzeit ein Problem auftreten, kann es notwendig sein, ein System kurzzeitig herunterzufahren. Ein häufiger Grund ist zum Beispiel eine Anwendung, die aufgrund eines Programmierfehlers Betriebsmittel sperrt und nicht mehr freigibt. Das Stoppen sollte von einem Managementwerkzeug ebenso ermöglicht werden (*stop()*), wie das Starten.

Der Neustart (*start()*) eines Systems bereitet erfahrungsgemäß mehr Schwierigkeiten als das Stoppen, weil die Stop-Funktion noch im betreffenden System ausgeführt werden kann, ein nicht gestartetes System seinen eigenen Restart aber nicht ohne weiteres einleiten kann. Dazu muß auf die jeweils darunterliegende Ebene zurückgegriffen werden. Bei Betriebssystemen dient dazu beispielsweise das BIOS, im Falle der TP-Monitore müssen zum Starten Betriebssystem-Dienste verwendet werden.

Um einem Administrator mehr Übersichtlichkeit zu bieten, sollten jeweils die beiden Halb-Verbindungen zu einer logischen Verbindung zusammengefaßt werden und in einer Region oder besser noch global für einen Node oder eine Domäne angezeigt werden können, da ein Node mehrere Systeme enthalten kann. Dazu dient die Klasse *Connection*, die über den *connectionName* identifiziert wird und nur Metainformation enthält, über die jeweils die beiden dazugehörigen Halb-Verbindungen dereferenziert werden können (*halfConnectionA* und *halfConnectionB*).
Zur einfacheren Rechtevergabe und für selektives Sperren von Verbindungen und Halb-Verbindungen sollten diese in Gruppen (*group*) klassifiziert werden. Damit können zum Beispiel alle Verbindungen zu einem Testsystem geschlossen oder Produktiv-Systeme nur für spezielle Integrations-Systeme geöffnet werden. Um Verbindungen zu öffnen und zu sperren braucht man die Methoden *enable()* und *disable()*.

Auf einer Verbindung werden eine oder mehrere Sitzungen (*Session*) aufgebaut. Einer Verbindung muß nun mitgeteilt werden, wieviele Sitzungen aufgebaut werden sollen, wenn die Verbindung zum ersten Mal aktiviert wird [1]. Dazu dient das Attribut *sessionCount*. Der Zeitpunkt der Aktivierung einer Verbindung wird über das Attribut *activateAtStartup* festgelegt. Das CICS aquiriert während des Betriebes nach Bedarf weitere Sitzungen, falls die Konversationen auf den bestehenden Sitzungen nicht mehr bewältigt werden können. Nachdem eine Verbindung erfolgreich definiert wurde, sollte eine Methode zur Verfügung stehen, um sie einmal auf Funktion zu prüfen (*test()*).

[1] Verbindungen werden normalerweise beim Start eines Systems aktiviert. Es kann aber auch während des laufenden Betriebs manuell eine Verbindung geschlossen und danach wieder aktiviert werden.

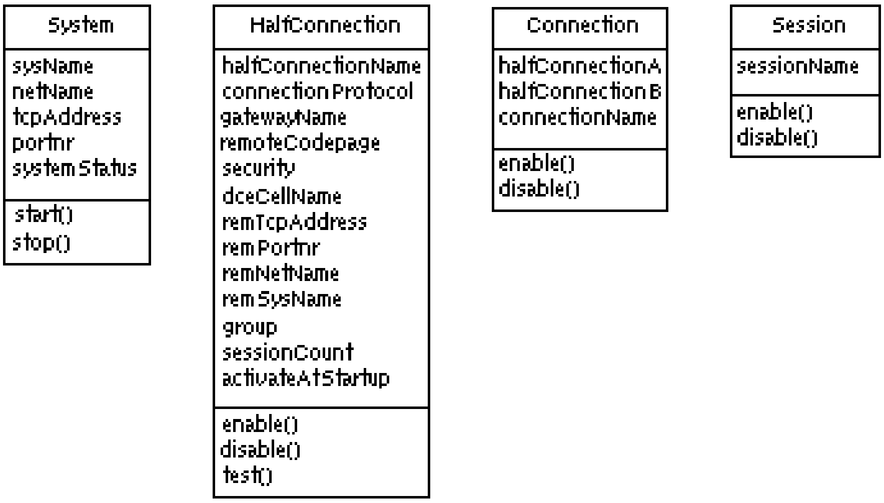

Abbildung 4.2: Die Klassen aus dem ersten Szenario

4.4 Szenario 2: CICS-Anwendung startet nicht.

Wenn ein Administrator einen Anruf erhält, daß sich eine bestimmte Anwendung nicht starten läßt, muß er zuerst herausfinden, in welchem System diese Anwendung aufgerufen wurde. Bisher ist es nicht üblich, diese Information in einer Anwendung zu speichern, der Administrator ist darauf angewiesen, daß bei der Installation einer Anwendung dokumentiert wurde, in welchem System die Transaktionen, Programme, Dateien und alle übrigen Betriebsmittel installiert wurden. In einer Anwendung muß also dokumentiert werden, in welcher Region sie physisch installiert wurde. Das geschieht im Attribut *regionName* der Klasse *Application* auf die im nächsten Abschnitt noch genauer eingegangen wird.

Hat man die Region identifiziert, muß man prüfen, ob diese überhaupt noch in einem aktiven Zustand ist. Dazu meldet man sich über ein Terminal am entsprechenden System an oder analysiert die Log-Dateien. Der Pfad zu den Log-Dateien wird sinnvollerweise im System gespeichert, dazu wird die Klasse *System* um *logPath*, *errorLogPath* und *loginLogPath* erweitert.

Da es in Großrechnerarchitekturen auch üblich ist, die Logs in eine Daten-Queue zu schreiben, wird es eine Spezialisierung für CICS/ESA geben, in der die Attribute *logQueue*, *errorLogQueue* und *loginLogQueue* vorhanden sind. Konnte man die Log-Datei lokalisieren, sollte am *System* eine Methode verfügbar sein, mit der man die Logs lesen kann (*readLog()*), damit ist es nicht mehr wichtig, ob das Dateisystem oder eine Daten-Queue als Log verwendet wird. Zuerst wir über die Userid des Anwenders das Terminal ermittelt, über das er sich angemeldet hat. Die Zuordung von angemeldeten Anwendern (*User*) zu den Terminal-Nummern (*Terminal*), die in Log-Dateien geschrieben werden, wird aus einem speziellen Login-Log ermittelt.

Dazu sollte es eine Methode am *User* geben, mit der dynamisch zur Laufzeit das Terminal ermittelt wird, unter dem der *User* dem System bekannt ist (*getTerminal()*). Ebenso ist es vorteilhaft, wenn über das Terminal ermittelt werden kann, welcher *User* ihm zu diesem Zeitpunkt zugeordnet war. Die Methode *getUser()* liefert eine *userId* als Rückgabewert.

Sollten nun Fehler im System aufgetreten sein, steht meist die Transaktion, die abgebrochen wurde, oder das Programm im Fehler-Log. An dieser Stelle kann man bereits sehen, ob das Starten der Anwendung überhaupt zu einem Transaktionsstart geführt hat. Ist das der Fall gewesen, sieht man bei dialogorientierten Anwendungen unmittelbar, welche Abbruchmeldungen bei der letzten ausgeführten Transaktion der Anwendung stehen. Man muß die Transaktionen daher erst einer Anwendung zuordnen. Dazu sind an der *Application* alle Transaktionen (*transactions*) referenziert, aus denen sie gebildet wird. Wenn jede Transaktion nur ein Programm enthalten würde, hätte man auch schon die Programme identifiziert, die zu einer Anwendung gehören. Weil aber ein Programm seinerseits wieder beliebig viele andere Programme zur Laufzeit aufrufen kann, muß man noch alle theoretisch aufrufbaren Programme (*programs*) einer Anwendung in der *Application* festhalten.

Hatte der Aufruf einer Anwendung jedoch keinen Transaktionsstart zu Folge, muß man die Funktion der Region prüfen, die als Terminal Owning Region, also meist auf einem Gateway-Server, fungiert. Das wird prinzipiell genauso gemacht, wie bei einer Application Owning Region, nur daß in der TOR selten Transaktionen und Programme installiert sind. Die Transaktionen und Programme werden entweder über dynamisches Transaktions-Routing oder mit einem Programm-Link auf eine fremde Region aufgerufen. Diese Eintragungen in der TOR müssen gegebenenfalls geprüft und berichtigt werden.

Es gibt folglich zu einer normalen Transaktion noch ein Gegenstück, die Remote-Transaktion, *RemTransaction*, und analog für Programme das Remote-Program*RemProgram*. Diese Betriebsmittel sollten ebenfalls in der dazugehörigen Anwendung gespeichert werden, in den Attributen *remTransaction* und *remProgram*. Sind in der TOR Remote-Transaktionen gestartet worden, die jedoch in der AOR nicht zu laufen begonnen haben, so liegt nahe, daß an der Kommunikationsverbindung ein Fehler aufgetreten ist. Verbindungen wurden bereits im obigen Abschnitt behandelt, daher wird an dieser Stelle nicht mehr darauf eingegangen. Ist der Aufruf der Transaktion auch in der TOR nicht angekommen, versucht man durch Aufbauen einer Protokollverbindung den Client-Rechner zu erreichen. Eine CICS-Verbindung kann man zu einem Client nicht herstellen, das ist nur in der Richtung vom Client zum Server, also der TOR, möglich. Dazu wird an der Klasse *Client* eine Methode eingeführt, mit der man die Protokoll-Verbindung aufbauen und testen kann (*pingClient()*). Hierzu benötigt man noch das Protokoll (*clientProtocol*), das vom Client verwendet wird, um das richtige Testwerkzeug auszuwählen. Ist mit der Verbindung zum Client alles in Ordnung, bleibt nur noch ein Fehler in der Client-Konfiguration. Für die CICS-Administration kann man sich hier auf die Konfigurationsdatei des CICS-Clients beschränken (*clientConfigFile*). Dazu muß eine Kopie dieser Datei lokal beim Administrator sein oder der Client einen Dateitransfer beziehungsweise ein Anmelden ermöglichen.

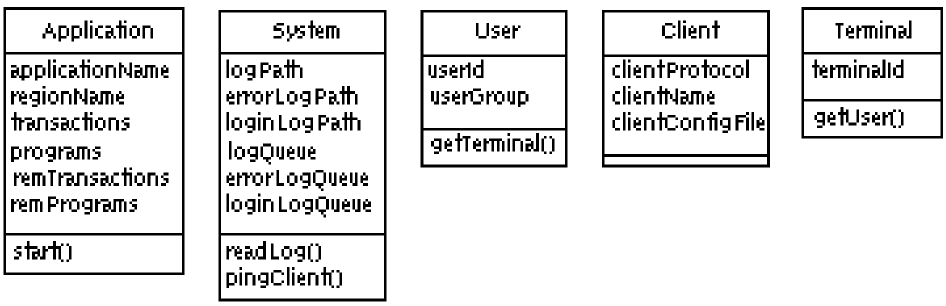

Abbildung 4.3: Die Klassen aus dem zweiten Szenario mit den Erweiterungen der Klasse System

4.5 Szenario 3: Installieren neuer CICS-Programme

Bevor neue Programme eingespielt werden, müssen alle Betriebsmittel der Anwendung, die aus diesen Programmen gebildet wird, vor Zugriff geschützt werden. Eine Anwendung (*Application*) im Sinne eines

TP-Monitors ist ein abstraktes Gebilde, da es nicht ein einzelnes Programm gibt, das aufgerufen wird und nach der Verarbeitung terminiert, sondern aus vielen Transaktionen (*Transaction*) besteht, die in einer bestimmten Reihenfolge gestartet werden. Hinter jeder Transaktion (*Transaction*) steht ein Programm, das beim Start einer Transaktion als erstes in den Speicher geladen und ausgeführt wird (*Program*). Welches Programm eine Transaktion als erstes aufruft, wird bei der Transaktion im Attribut *firstProgram* definiert. Dies Programm ist der Einstiegspunkt für die Programmkette, die im Rahmen einer CICS-Transaktion ausgeführt wird. Um eine Transaktion zu definieren, ist noch anzugeben, ob sie dynamisch geroutet werden darf (*dynRoute*) und welcher CICS-Gruppe (*CicsGroup*) sie zugeordnet wird (*group*). Die restlichen Attribute können mit Default-Werten oder optional angegeben werden. Zu einer Anwendung gehören aber typischerweise noch andere Betriebsmittel, wie Dateien (*File*), die entweder als Dateien eines Dateisystems oder auch als Datenbanktabellen realisiert sein können. Diese Unterscheidung wird mit dem Attribut *fileType* getroffen. Bildschirmmasken, die Anwender-Schnittstelle des CICS, werden *Map* genannt und sind ebenfalls Bestandteil der Anwendung. Über eine transiente Daten-Queue (*Tdq*) stellt eine Transaktion ihren Transaktionskontext der Nachfolge-Transaktion zu Verfügung. Es werden zum Beispiel Eingaben, die in einer Maske gemacht wurden, der nächsten Transaktion übergeben, damit der Anwender diese Eingaben nicht noch einmal machen muß. Ebenso kann man mit einer TDQ Ergebnisse aus einer Datenbankanfrage an Folgetransaktionen weitergeben, ohne die Anfrage nochmals zu starten. Für den Fall, daß eine Anwendung bereits installiert war und zum Beispiel nur neue Versionen der Ressourcen eingespielt werden, müssen die Betriebsmittel einer Anwendung vor Zugriff geschützt werden, solange sich die Anwendung in einem inkonsistenten Betriebsmittelzustand befindet. Jede Klasse muß daher eine Methode haben, die den Zugriff auf die entsprechenden Ressourcen sperrt (*lock()*). Außerdem braucht man die momentane Version (*version*) aller Betriebsmittel, um Kompatibilitätsprobleme frühzeitig zu erkennen.

Wird eine Anwendung erstmalig installiert, muß eine geeignete Region gefunden werden, die alle Voraussetzungen für die jeweilige Anwendung erfüllt. Der Region-Name wird in *regionName* eingetragen. Für das eigentliche Einspielen der Anwendung, das Kopieren von Band oder CD-ROM, werden die Betriebssystem-eigenen Werkzeuge verwendet. Hier kann der CICS Domain Manager keine Hilfestellung leisten, da sich die Übernahme-Verfahren in die Produktivumgebung abhängig vom Betreiber stark unterscheiden. Um die Definitionen für eine Anwendung anzulegen, wird bei einer Anwendung eine Datei oder ein Skript mitgeliefert, mit dem zum Beispiel über die Schnittstellen *DEFINE* in CICS/ESA und CICS for OS/2 oder das CICS/6000-Kommando *cicsadd* alle Definitionen auf einmal eingetragen werden können, ohne daß der Administrator jede einzelne Definition mit der Hand vornehmen muß. Die benötigten Verbindungen, zum Beispiel für Datenbankzugriffe, müssen wie in 3.1.1 eingetragen werden. Sind alle Definitionen getroffen, wird die Anwendung aktiviert. Das kann global für die gesamte Anwendung oder auch selektiv für jede einzelne Definition mit *activate()* geschehen. Jetzt könnte die Anwendung bereits lokal gestartet werden. Um Anwendern, die sich mit einem Client anmelden oder die an entfernten Terminals sitzen, den Zugriff auf die Anwendung zu erlauben, sind noch Eintragungen in den Terminal-owning Regions (TOR) notwendig, damit von dort die Anwendung 'remote' gestartet werden kann. Man benötigt also mindestens für Transaktionen und Programme jeweils eine Klassem, die eine Remote-Definition realisiert. *RemTransaction* und *RemProgram* sollen diese Klassen heißen und einen Verweis auf die Definition in der Application-owning Region (AOR) enthalten. (*transactionDef* und *programDef*). Als globale Informationen sollten noch die zuständigen Administratoren für das Sicherheitssystem (RACF, DCE, usw.) und für die Datenbanksysteme zur Verfügung stehen.

4.6 Identifikation der fehlenden Klassen

Damit sind die Klassen, die sich unmittelbar aus den Anforderungen ergeben, identifiziert. Jedoch sind die Klassen sozusagen aus dem Zusammenhang gerissen, es sind noch ergänzende Klassen notwendig und die Klassen müssen zueinander in Beziehung gesetzt werden.
Die Klassen, die nicht direkt aus den Anforderungen abgeleitet werden konnten, sind im folgenden beschrieben.
Für die Beschreibung wurde eine dem Data Dictionary ähnliche Form gewählt:

CDM_AIX_Domain: Repräsentiert die Domäne der CICS/6000-Systeme mit ihren Anwendungen und ihrer Topologie. Bei Systemen, die mit DCE-Unterstützung definiert sind, könnte hier auch die DCE-

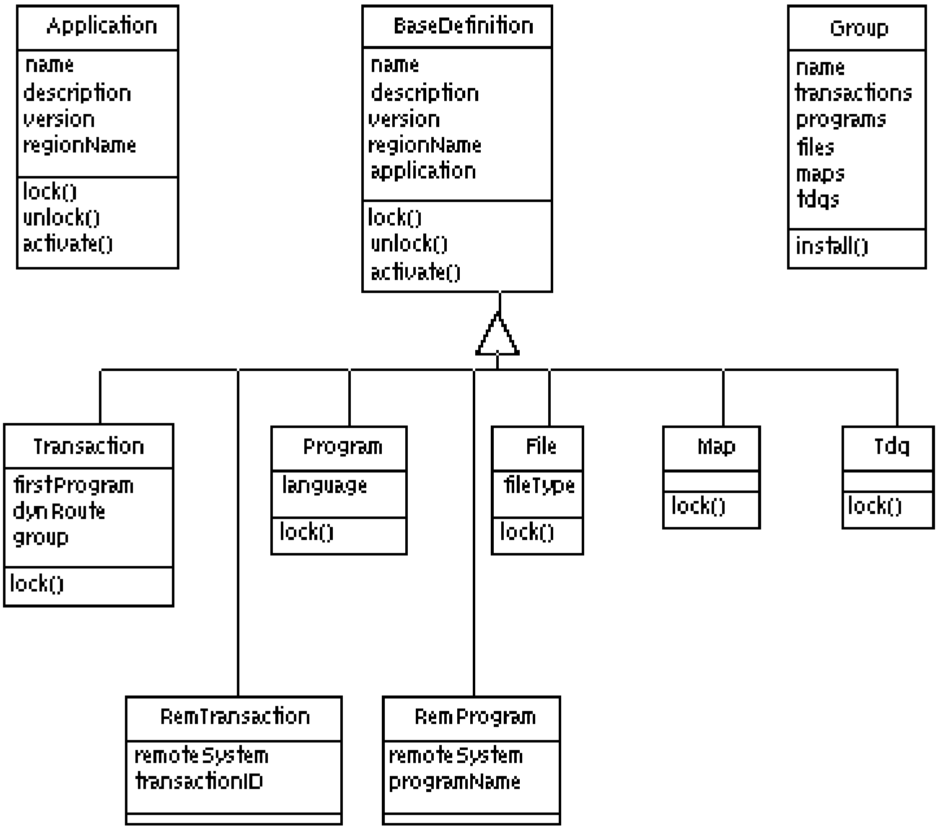

Abbildung 4.4: Die Klassen aus dem dritten Szenario mit den Erweiterungen

Zelle die Funktion der Domäne übernehmen.

CDM_AIX_Region: Spezialisierung eines CICS-Systems, da auch ein CICS-Client ein CICS-System mit eingeschränkten Möglichkeiten ist.

CDM_AIX_DomainTransaction: Enthält Informationen, wo eine bestimmte Transaktion innerhalb der Domäne existiert, sowohl lokal als auch remote eingetragen.

CDM_AIX_DomainProgram: Enthält Informationen, wo ein bestimmtes Programm innerhalb der Domäne existiert, sowohl lokal als auch remote eingetragen.

CDM_AIX_DomainFile: Enthält Informationen, wo diese Datei innerhalb der Domäne existiert, sowohl lokal als auch remote eingetragen.

CDM_AIX_DomainUser: Enthält Informationen, wo ein bestimmter Anwender innerhalb der Domäne existiert.

CDM_AIX_DomainMap: Enthält Informationen, wo eine bestimmte Bildschirmmaske innerhalb der Domäne existiert.

CDM_AIX_DomainTdq: Enthält Informationen, wo eine bestimmte Daten-Queue innerhalb der Domäne existiert.

CDM_AIX_DomainApplication: Enthält Informationen, auf welchen Systemen innerhalb der Domäne eine Anwendung definiert ist.

CDM_AIX_DomainTopology: In der Domain-Topology sind die Verbinungen der CICS-Systeme untereinander, damit verbundene Routeninformation und der Zustand von Verbindungen und Systemen gespeichert.

Für den Fall, daß eine Anwendung auf mehrere Regions verteilt wird, ist noch sinnvoll, an jeder Definition der Betriebsmittel den Region-Namen zu speichern (*regionName*). Um das Definieren von Ressourcen so einfach, wie möglich zu machen definiert man sich Modelle der *Resource Definitions*, die alle Standardwerte enthalten, damit hält man die Informationen, die bei jeder einzelnen Definition eingegeben werden müssen, so gering wie möglich.

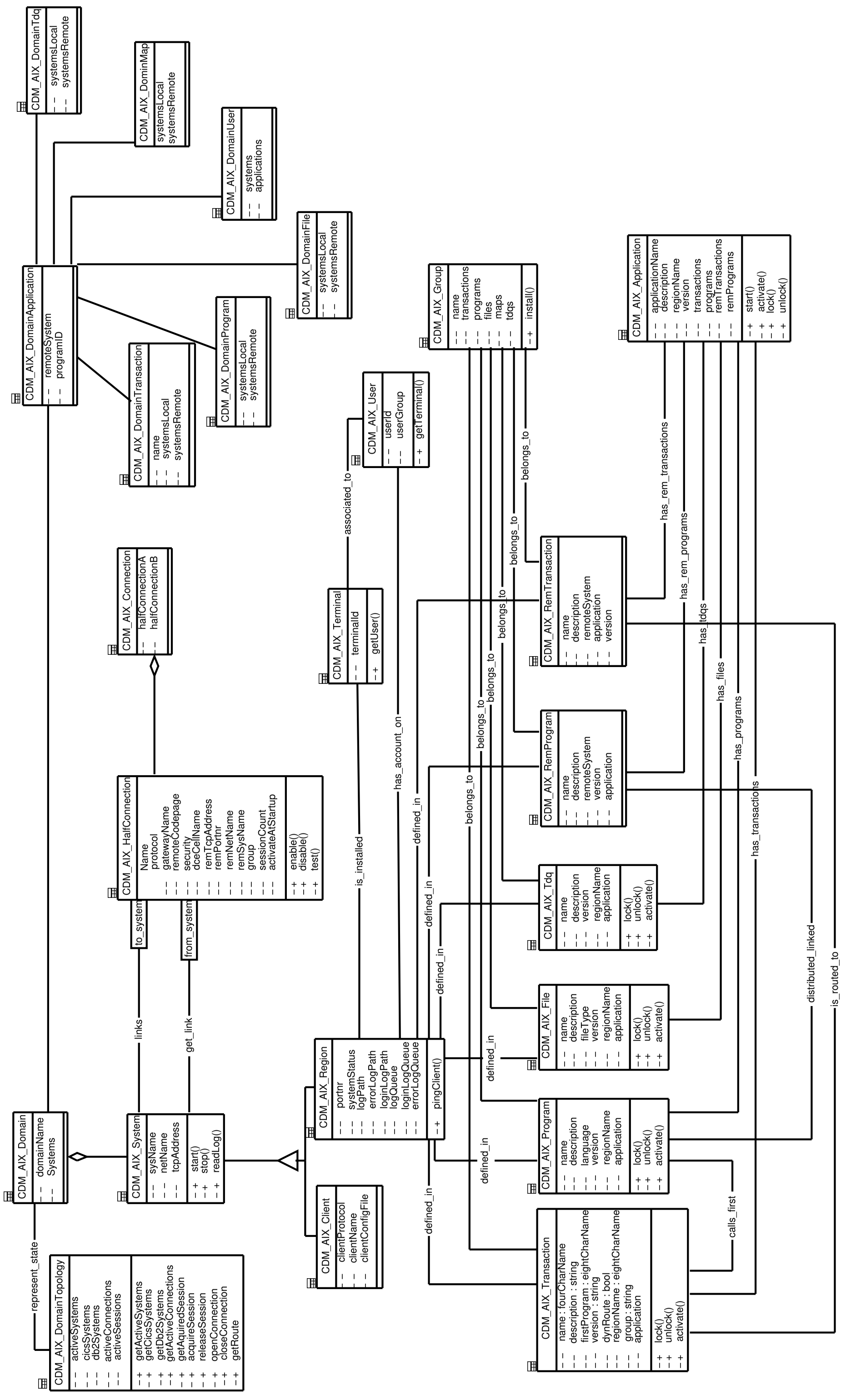

Abbildung 4.5: Das Klassenmodell des CDM mit ergänzenden Klassen und Beziehungen

Kapitel 5

Analyse von Werkzeugen für das Management von TP-Monitoren

Die Analyse der proprietären Werkzeuge, bei den jeweiligen Systemen mitgeliefert, beschränkt sich auf ein Minimum, da jedes Werkzeug nur für eine bestimmte Plattform einsetzbar und nicht in ein plattformübergreifendes Konzept integriert ist. Die Analyse dient auch dazu, eine Vorstellung zu bekommen, mit welchen Werkzeugen geschäftskritische Anwendungen heute adminstriert werden.

Darüberhinaus werden diese Werkzeuge unter der Prämisse betrachtet, daß eventuell Management-Agenten für MVS- und OS/2-Systeme nicht von IBM zur Verfügung gestellt werden und damit der CICS-Domain-Manager auf eigene Management-Agenten für CICS/ESA und CICS for OS/2 zurückgreifen muß, die aber dann erst noch zu entwickeln wären.

Der CICS System Manager ist wegen seines objektorientierten Konzeptes ausführlicher beschrieben und es wird im nächsten Kapitel eine tiefere Bewertung anhand der Anforderungen erfolgen. Der CICS System Manager wird noch nicht häufig eingesetzt, da dieses Werkzeug erst seit Mitte 1996 verfügbar ist.

5.1 Die Transaktion CEDA

Mit der Transaktion **CEDA** werden unter allen Mainframe-Betriebssystemen und unter OS/2 die CICS-Ressource-Definitionen in den Systemtabellen eingetragen. Neben der Transaktion *CEDA* gibt es noch die Variationen *CEDB* und *CEDC*, die sich aber nur dadurch von *CEDA* unterscheiden, daß man mit weniger Rechten auf die Systemtabellen zugreift. CEDA ist keine Abkürzung, sondern ist ein durch Namenskonventionen entstandenes Akronym. Alle Transaktionen, die zum TP-Monitor CICS gehören, beginnen mit 'C', mit der letzten Stelle wird die Unterscheidung nach den Rechten getroffen und die beiden mittleren Buchstaben bedeuten etwa 'edit definition'. Hinter der Transaktion CEDA steht das CICS-Programm **DFHCSDUP**, dessen Aufruf aus eigenen Programmen im nächsten Abschnitt erläutert wird. Die Anwenderschnittstelle *CEDA* ist rein textorientiert, wie Abbildung 5.1 zeigt, und kann nur innerhalb eines funktionierenden CICS-Systems gestartet werden. Aus diesem Einstiegsbild verzweigt man durch Markieren am Zeilenanfang in die einzelnen Systemtabelleneditoren. Als Beispiel ist hier die *Connection and Session Table* abgebildet (Abb. 5.2), da in Kapitel 3 im Szenario 3.1.1 in dieser Tabelle die Eintragungen gemacht werden. Um die verschiedenen Werkzeuge zum Editieren der Systemtabellen vergleichbar zu machen, wird bei *CEDA*, *smit* und *IBM CICS SM* jeweils die entsprechende Maske/Tabelle zur Definition einer Verbindung gezeigt.
Die *Connection and Session Table* gibt es in drei verschiedenen Ausprägungen, für APPC, NetBios und TCP/IP. Abbildung 5.2 zeigt die Maske für TCP/IP. Das obere Drittel der Maske ist bei allen drei Variationen identisch, der Rest wird nach Eingabe des Parameters *Connection Type* an die unterschiedlichen Parameter der verschiedenen Protokolle angepaßt. Mit der *CEDA*-Transaktion können alle Systemparameter verändert werden, die mit *CICS* zusammenhängen. Man kann das zur Transaktion *CEDA* gehörende Programm *DFHCSDUP* auch direkt von der Kommandozeile aufrufen und nur durch Parametrisierung die Definitionen verändern. Dieses Verfahren wird im nächsten Abschnitt beschrieben.

```
+--------------------------------------------------------------------------+
|                                                                          |
|  Process                                                                 |
|  Exit            Help                                                    |
|  FAAEDA1            Resource Definition Online                           |
|                                                                          |
|                                                                          |
| Type / against a Table Name. Then select one of the actions shown.       |
|                                                                          |
|                                                                          |
|  /   Table            Description                                        |
|  _   CVT              Data Conversion Table                              |
|  _   DCT              Destination Control Table                          |
|  _   FCT              File Control Table                                 |
|  _   PCT              Program Control Table                              |
|  _   PPT              Processing Program Table                           |
|  _   SIT              System Initialization Table                        |
|  _   SNT              Signon Table                                       |
|  /   TCS              Connection and Session Table                       |
|  _   TCT              Terminal Control Table                             |
|  _   TST              Temporary Storage Table                            |
|  _   WSU              Workstation Set Up                                 |
|  _   GRP              Group Processing                                   |
|                                                                          |
|  Enter   F1=Help    F3=Exit                         F10=Actions  F12=Cancel |
|                                                                          |
|                                                                          |
+--------------------------------------------------------------------------+
```

Abbildung 5.1: Die Resource Definition-Transaktion CEDA

```
+--------------------------------------------------------------------------+
|                                                                          |
|  Update          Add             View            Delete                  |
|  Exit            Help                                                    |
|  FAATCS5                 Connection and Session Table                    |
|                                                                          |
|  Connection Name. . . . . . . . . . . . . X1TA                           |
|  Group Name . . . . . . . . . . . . . . . ________                       |
|  Connection Type. . . . . . . . . . . . : TCP_     (APPC, NETB or TCP)   |
|  Connection Priority. . . . . . . . . . . 086      (0-255)               |
|  Description. . . . . . . . . . . . . . . CONNECTION TO MSGAX1.MSG.DE___ |
|                                                                          |
|  Session Details                                                         |
|      Session Count. . . . . . . . . . . . 01       (1-99)                |
|      Session Buffer Size. . . . . . . . . 16384    (512-40000)           |
|      Attach Security. . . . . . . . . . . L        (L=Local, V=Verify)   |
|      Partner Code Page. . . . . . . . . . 00037                          |
|                                                                          |
|  TCP/IP Details                                                          |
|      Local host name. . . . . . . . . . . *______________________________ |
|      Remote host name . . . . . . . . . . MSGAX1.MSG.DE__________________ |
|      Remote host port . . . . . . . . . . 1435_    (1-65535, or *)       |
|                                                                          |
|                                                                          |
|  Enter   F1=Help    F3=Exit                         F10=Actions  F12=Cancel |
|                                                                          |
|                                                                          |
+--------------------------------------------------------------------------+
```

Abbildung 5.2: Die Tabelle 'Connection and Session Table'

5.2 Das Programm DFHCSDUP

Mit dem Programm **DFHCSDUP** lassen sich alle Definitionen in einem CICS-System bearbeiten. Die Tabellen, in denen die gesamte Information eines Systems gespeichert ist, heißt im CICS **CSD** (**CICS System Definitions**). Der Name DFHCSDUP ist also zusammengesetzt aus dem Präfix, den alle CICS/ESA-Programme tragen (DFH), dem Einsatzgebiet des Programms (CSD) und das UP steht für *User Program*. Mit DFHCSDUP kann man, gesteuert über Aufrufparameter, Systemtabellen Initialisieren, Definitionen neu anlegen, ändern, kopieren, auflisten, löschen und die Daten einer Definition ausgeben. Außerdem können Gruppen in Listen aufgenommen und aus diesen auch wieder gelöscht werden.

Ein solches Kommando könnte für das Beispiel der Verbindung folgende Form haben:

```
EXEC CICS
     DEFINE CONNECTION(X1TA)            /* Name */
            CONNECTIONTYPE(CICS_TCP)
            PRIORITY(01)                /* Ausfuehrungsprioritaet */
            SESSIONCOUNT(5)             /* moegliche Sitzungen */
            BUFFERSIZE(4096)            /* Paketgroesse 4 kByte */
            SECURITY(LOCAL)             /* Sicherheitseinstellung */
            CODEPAGE(ISO8859-1)         /* verwendeter Zeichensatz */
            LOCALHOSTNAME(REGA)
            REMOTEHOSTNAME(AFIS)
            PORT(1435)
```

Das ermöglicht ein Eintragen von Definitionen mittels einer generierten Datei, die dann ohne Eingreifen des Administrators alle benötigten Definitionen trifft. Eintragungen dieser Art werden meist während eines Batch-Laufes erledigt. Kommandodateien für Anwendungen werden bei Neuinstallationen in der Regel von den Software-Herstellern mitgeliefert.
Nach der Definition aller Ressourcen müssen diese lediglich mit der **CEMT**-Transaktion in das System übernommen werden. Das kann einzeln für jede Definition geschehen oder aber gruppenweise für eine gesamte CICS-Gruppe. Das Programm DFHCSDUP kann zusätzlich zum Aufruf von der Kommandozeile und aus einem Stapelverarbeitungsprogramm auch aus eigenen PL/1, COBOL und inzwischen auch aus C-Programmen über eine API aufgerufen werden.

5.3 System Management Interface Tool (smit)

smit ist ein Konfigurationswerkzeug mit graphischer Oberfläche, das für IBM AIX verfügbar ist. Man kann es unter dem Namen **smitty** jedoch auch im Textmodus starten. Mit dem Werkzeug *smit* lassen sich alle Einstellungen für AIX und die darauf laufenden IBM-Anwendungen treffen. Für das CICS-Management ist *smit* das derzeit leistungsfähigste Werkzeug, mit dem alle Eintragungen in den *Resource Definitions* gemacht werden können. Ebenso ist es mit *smit* möglich, *CICS-Regions* zu stoppen, zu starten, miteinander zu verbinden, zu löschen, exportieren, importieren, kopieren und alle Aktionen auf Ressourcen auszuführen. Für die meisten Definitionen werden sogenannte *models* unterstützt, die eine Vordefinition von Definitions-Modellen erlauben. In diesen Modellen können die Verallgemeinerungen der Definitionen hinterlegt werden, die dann bei einer konkreten Definition nur noch durch die Spezialisierungen ergänzt werden müssen. Leider ist *smit* nur für IBM AIX verfügbar und unterstützt nur Anwendungen, die ihrerseits eine Zugansgmöglichkeit für den *smit* bereitstellen. Wenn eine Anwendung die Konfiguration durch *smit* nicht unterstützt, stehen diesem auch keine Informationen zur Konfiguration zur Verfügung. *smit* besitzt eine baumartige Struktur und setzt dadurch beim Administrator großes Datailwissen voraus, da es kein Navigationshilfmittel durch die Baumstruktur bereitstellt. Außerdem gibt es keine Strukturierung nach Themengebieten, die bei der Fehlersuche eine große Hilfe wären. So muß man zum Beispiel beim Testen einer Verbindung zwischen zwei *CICS-Systemen* Eintragungen im CICS und zusätzlich in den *SNA*-Definitionen prüfen, die jedoch in einem anderen Zweig der Hierarchie einzutragen sind. Automatische Mechanismen

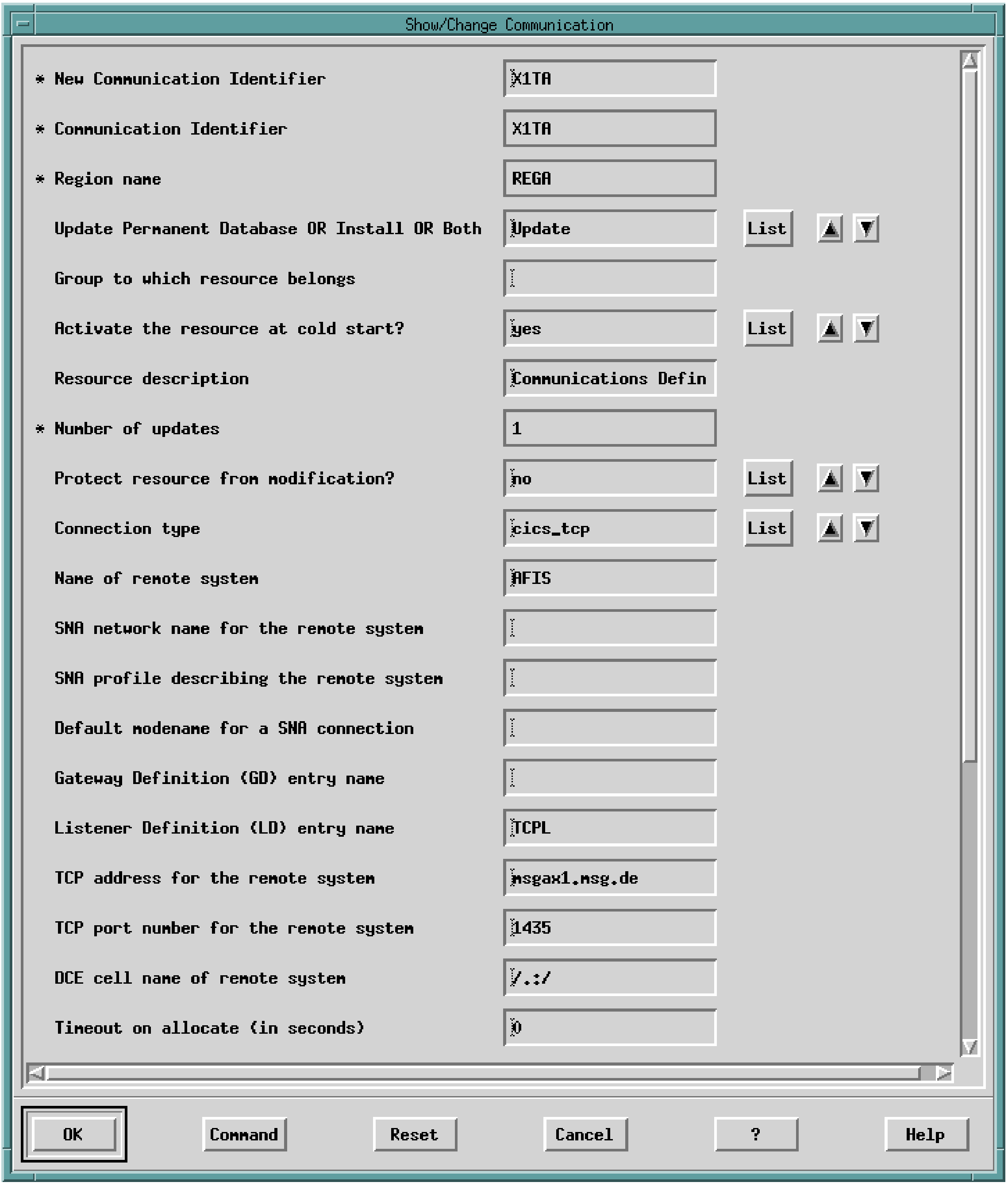

Abbildung 5.3: Definition einer Kommunikationsverbindung mit *smit*

zum Sammeln von managementrelevanter Information werden nicht unterstützt. Ähnlich dem CICS/ESA wird auch hier eine Befehlszeilenschnittstelle zur Definition bereitgestellt. Hierfür wird auf [ADM AIX] verwiesen, wo unter anderem die Kommandos `cicsadd`, `cicsupdate`, `cicsdelete` und `cicsget` beschrieben sind. Ein Beispiel einer Eingabemaske für Kommunikationsdefinitionen in *smit* findet man in Abbildung 5.3.

5.4 Der IBM CICS System Manager for AIX

5.4.1 Einführung

Der **IBM CICS System Manager** (**SM**) ist ebenso, wie *smit* ein graphisches Werkzeug, mit dem man Definitionen für CICS-Systeme treffen kann. Der SM ist modular aufgebaut und besteht aus einer **System Management Application** (**SM Application**), einem **Graphical User Interface** (**GUI**), einem **Resource Controller** und einer **CICS Agent Transaction**, die in einem bestimmten CICS-System läuft und Informationen sammelt. Der *Resource Controller* und die *Agent Transaction* bilden zusammen den **CICS SM Agent** und kommunizieren über *UNIX sockets* miteinander. Die *System Management Application* steht mit dem *Resource Controller* und dem *GUI* über **DSOM** (**Distributed System Object Model**) in Verbindung (Abbildung 5.4).

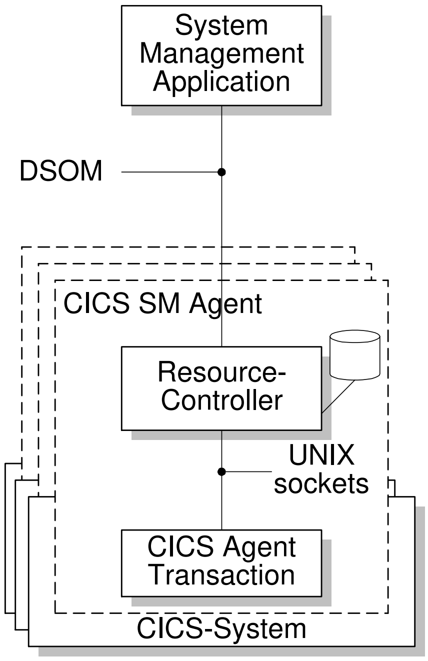

Abbildung 5.4: Die Architektur des IBM CICS System Manager

Was den SM von den anderen Werkzeugen grundsätzlich unterscheidet, ist das Konzept, mit dem die Managementinformation verwaltet wird. Der SM ist kein reiner Editor für Systemtabellen, sondern ein verteiltes Werkzeug, das über DSOM mit seinen Komponenten kommuniziert. Gerade dieses Konzept macht ihn für diese Arbeit interessant, weil so über eine **CORBA**-konforme Schnittstelle die Managementinformation und -Funktionalität nach außen hin zugreifbar wird. Damit erübrigt sich eine Implementierung eines Management-Agenten für CICS - zumindest im ersten, prototypischen Ansatz. Nachdem die Informationen der CICS-Systeme beim SM in CORBA-Objekten gehalten werden, kann man sich auch hier auf vorhandene Teile abstützen. Die Klassen, in denen die eigentliche Information gehalten wird, müssen nicht mehr implementiert werden, was eine große Erleichterung bringt. Diese Implementierung wäre in erster Linie eine Abbildung der Systemtabellen und würde für die Arbeit keinen Fortschritt bedeuten. Interessanter sind in diesem Zusammenhang die Container- und Metaklassen, mit denen die große Menge der Informationen strukturiert werden soll. Durch die Philosophie von **SOM/DSOM**, alle Objekte und deren Schnittstellen dynamisch zu registrieren, geht aus der Dokumentation nicht unmittelbar hervor, wie die

Klassen des SM genau heißen. Deshalb wurde versucht, mit Hilfe eines PERL-Scripts aus dem Interface-Repository des SM einen C++-Header mit den benötigten Klassen zu erhalten. Dazu dienten die Werkzeuge *ir2classes* und *classes2xh*, die im Anhang B.2 beigefügt sind. Damit kann auch die Vererbungshierarchie der Klassen ermittelt werden. Mit den CASE-Werkzeugen *sniff+* und *StP* wurde aus den Header-Dateien ein Klassendiagramm erzeugt. Dieses Diagramm war jedoch so umfangreich und unübersichtlich [1], daß dieses Diagramm zum Verständnis des Klassenmodells wenig geeignet ist. Diese Vorgehensweise ist zwar sehr exakt, aber wenig praktikabel. Außerdem ergab sich bei Stichproben, daß ein Klassendiagramm, das mit *sniff+* und *StP* erzeugt wurde, nicht die Vererbungshierarchie hat, die aus dem Interface-Repository und den C++-Header-Dateien hervorgeht. Diese Klassendiagramme sind also als Grundlage für eine fundierte Arbeit nicht geeignet! Eine zweite Möglichkeit, um Informationen über die Klassen und Attribute zu bekommen, ist im Manual [IBM SM] beschrieben. In diesem Manual (es wird getrennt von der Software auf einer speziellen CD-ROM vertrieben, die ausschließlich Dokumentation enthält) wird auf ein Werkzeug hingewiesen, mit dem die Klassen der API des SM angezeigt werden können. Dabei kann man auch alle Abhängigkeiten, Attribute und Methoden, mit Parametern, einer Klasse auslesen. Die Sichtweise, die man über dieses Werkzeug auf die SM-Klassen bekommt, ist aber eher eine funktionale Sicht, da hier nicht die eigentlichen Vererbungs- und Enthaltenseinsbeziehungen zwischen den Klassen dargestellt werden, sondern vielmehr die Beziehungen zwischen den CICS-Komponenten. In B.3 befindet sich ein PERL-Skript, mit dem automatisch alle Klassen mit Attributen, Methoden und Abhängigkeiten gelesen und in eine C++-Header-Datei konvertiert werden können. Dieser C++-Header kann dann wiederum mit *sniff+* und *StP* in ein Klassenmodell überführt werden. Bei genauerer Analyse der beiden Möglichkeiten, zu einem Klassenmodell zu kommen, fiel auf, daß die Identifikation der Attribute und Methoden von Klassen, die über die API angesprochen werden, nur auf Basis des Interface-Repository sehr schwierig ist. Man muß dabei die gesamte Vererbungshierarchie durchgehen und alle Attribute und Methoden zusammensammeln, wobei mit dieser Vorgehensweise sehr viele Datenelemente gefunden werden, die nicht von aussen zugänglich sind. Weil beim späteren Zugriff auf CICS-Definitionen jedoch entweder über die API zugegriffen werden wird oder gleich eigene Klassen verwendet werden, kann man sich auf die öffentlichen Attribute und Methoden beschränken und den einfacheren und weniger detaillierten Weg gehen. Außerdem werden für den Zugriff auf Attribute ohnehin GET- und SET-Methoden verwendet, die nur einen eingeschränkten Zugriff zulassen.

In einem weiteren Ansatz wurde versucht, auf bereits instanziierte Objekte in einer SM Application oder eines Resource Controllers, also dem Agenten der SM Application, zuzugreifen. Damit kann man sich die Informationen die konkret in einem CICS-System existieren über den Agenten des CICS SM ermitteln. Da der Resource Controller eine standardisierte Schnittstelle besitzt, nämlich SOM/DSOM, ist es naheliegend, direkt über SOM/DSOM auf diese Objekte zuzugreifen. Ein Beispielprogramm in C++ ist im Anhang B.1 zu finden. In dieser Form dient das Programm dazu, die Vorgehensweise zu dokumentieren, wie über SOM/DSOM auf Objekte und deren Attribute und Methoden zugegriffen werden kann. Für den konkreten Einsatz ist das Programm noch nicht zu nutzen, da ein Navigationshilfsmittel zu einem bestimmten Objekt bisher fehlt. In der SM Application gibt es ein solches Hilfsmittel nur zum Teil, da man immer ein Kontainerobjekt öffnen muß, um seinen Inhalt zu sehen. Ein Objekt kann nicht direkt mit seinem Namen eingegeben werden, um es zu bearbeiten.

Auf der Grundlage dieser Informationen besteht jetzt die Möglichkeit, die Anforderungen aus dem vorigen Kapitel mit der vorhandenen Funktionalität des SM zu vergleichen.

5.4.2 Leistungsumfang

- Das Klassenmodell des SM enthält eine Obermenge der Klassen, die für das Management von TP-Monitoren notwendig sind. Als Abbild der Systemtabellen sind diese Klassen gut verwendbar. Ein Klassenmodell wird von IBM nicht herausgegeben.

- Die Anwendung gibt es bereits als Klasse, jedoch ist eine Anwendung nur im Rahmen eines *Plexes* zu definieren. Die Definition eines *Plex* steht jedoch im Widerspruch zu einem integriertem Management, da eine *Plex*-Umgebung alle managementrelevanten Informationen verschattet. Es ist zum

[1] Das Diagramm enthält 445 Klassen, die miteinander über sogenannte *Diamond-Inheritance* verbunden sind. SOM/DSOM verwendet Diamond-Inheritance, wodurch eine Klasse die Eigenschaften von zwei oder mehreren Klassen erbt. Diese Klassen erben wiederum von einer gemeinsamen Klasse. Dadurch entsteht im Diagramm eine rauten- oder diamanten-förmige Vererbungshierarchie.

Beispiel nicht mehr möglich, innerhalb eines *Plex*, der aus mehreren Regions besteht, festzustellen, in welcher Region letztendlich ein Programm ausgeführt wird, also die dazugehörige Task läuft.

- Es gibt keine Befehlszeilen- oder Stapelverarbeitungs-Schnittstelle für Resource Definitions. Dadurch ist es notwendig, auf zusätzliche Werkzeuge zurückzugreifen, die diese Funktionalität bieten (zum Beispiel *smit* mit `cicsadd`), um große Mengen von Transaktionen oder Programmen zu definieren.
- Der IBM CICS SM ist nur für CICS-Systeme unter AIX verwendbar. Agenten-Transaktionen für andere Plattformen sind noch nicht verfügbar.
- Viewbildung und logische Strukturierung der Resourcen und Systeme wird nur insofern unterstützt,

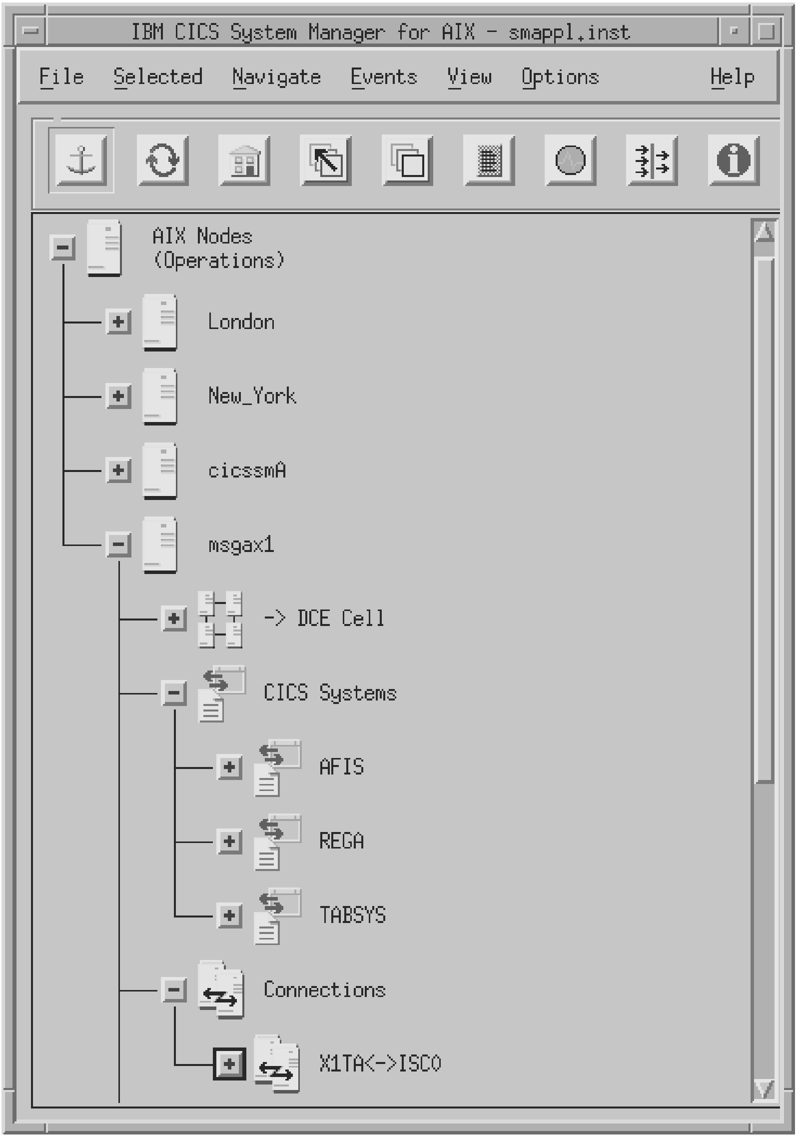

Abbildung 5.5: Die graphische Anwenderschnittstelle des IBM CICS System Manager

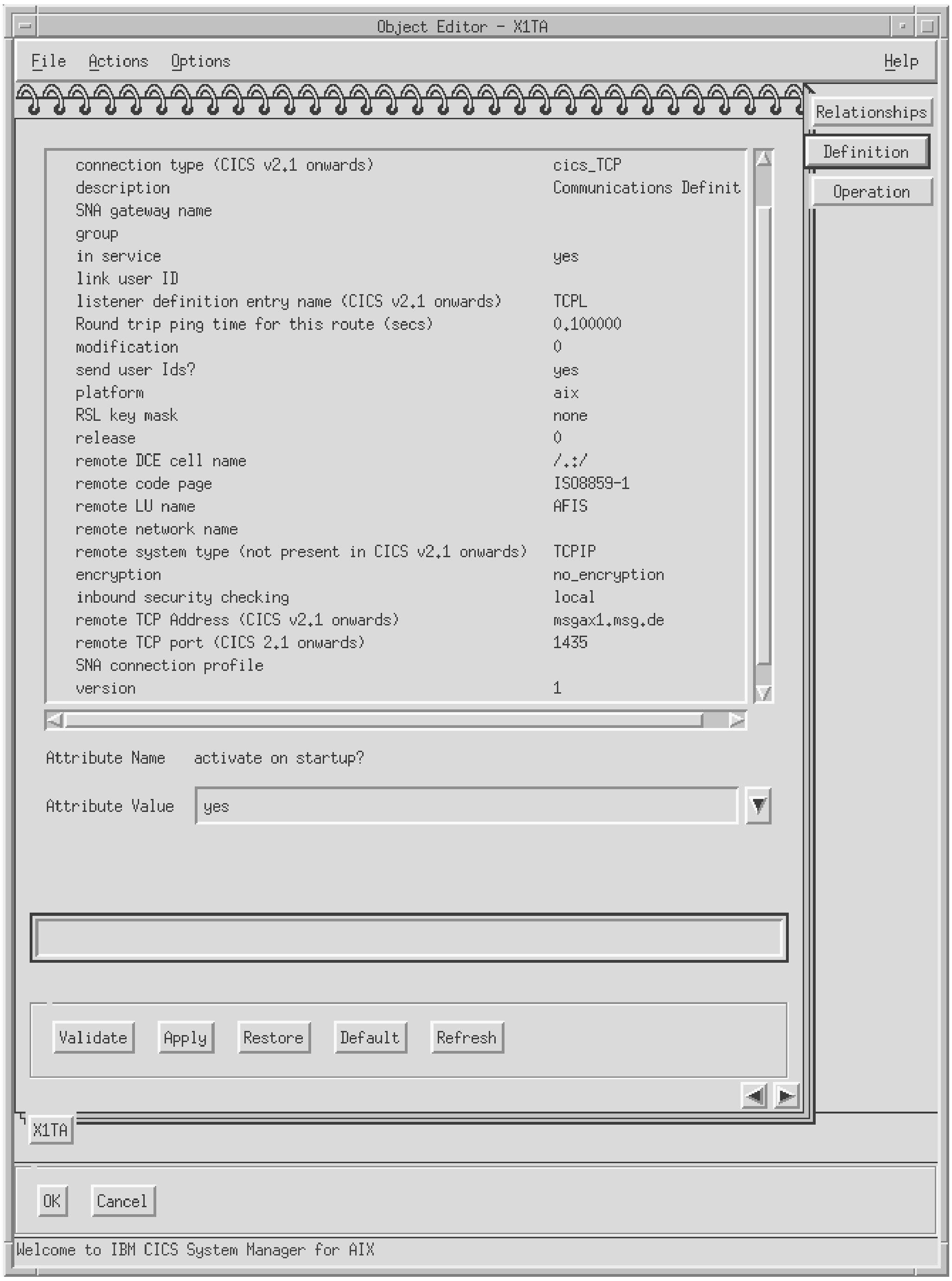

Abbildung 5.6: Definition einer Kommunikationsverbindung mit IBM CICS SM

daß es eine baumartige Struktur der instanziierten Objekte gibt, wobei *Nodes*, *CICS-Systems*, *DCE-Cells* jeweils Wurzeln sind. Außerdem gibt es die Möglichkeit, alle definierten *Connections* zwischen den Systemen eines jeden *Nodes* gesamthaft anzuzeigen und damit eine Erweiterung der *Half-Connections* zu schaffen, die jeweils nur in eine Richtung definiert werden.

- Der CICS SM besitzt eine graphische Bedienoberfläche, die Drag-and-Drop nur im Ansatz unterstützt. Man kann Objekte nicht durch Mausbedienung an eine andere Stelle auf der Oberfläche verschieben, sondern muß über ein Kontextmenü am Objekt den Menüpunkt *Drag* auswählen, ein anderes Objekt markieren und an dessen Kontextmenü auf den Eintrag *Drop* klicken. Dadurch bekommt das Verfahren mehr die Eigenschaften von Copy-and-Paste.
- Die Methode `Discover Systems` auf einen *Node* angewendet, sucht *CICS-Systems*, *DCE-Cells*, *Connections*, *Gateway Servers* und *Structured File Servers*.
- Die Methode `Discover` auf ein bestimmtes *CICS-System* ermittelt alle Attribut-Werte einer *Region*.
- Stoppen einer CICS-Region ist möglich.
  Dabei werden die gestarteten Application-Server-Prozesse terminiert, die Log-Dateien für einen Warmstart geschrieben, und das CICS-System heruntergefahren. Es besteht noch die Mögllichkeit, in den Systemdefinitionen Programme so einzutragen, daß sie in der Phase des Stoppens einer Region noch gestartet werden. Im Falle des CICS SM ist das ein Programm, das asynchron eine Nachricht an den CICS SM schickt, damit die Management-Application den Status einer Region von *up* auf *down* setzt. Leider wird diese Änderung nicht visualisiert, sondern steht nach dem Markieren einer Region lediglich in der Statuszeile.
- Starten einer CICS-Region ist nicht möglich
  Das erscheint logisch, weil alle Aktionen über die *Agent-Transaction* gesteuert werden. Wenn die CICS-Region nicht gestartet ist, kann in ihr auch keine Transaktion gestartet sein. Für den CDM hat das die Konsequenz, daß man sich zur Informationsgewinnung zwar auf derartige Transaktionen abstützen kann, für die Erbringung von Funktionalität jedoch auf extern laufende Komponenten zurückgreifen muß. Die Definition einer Region ohne **Distributed Computing Environment** (**DCE**) führte nicht zum Erfolg. Es wurden alle Schritte der Installationsanweisung genau befolgt, und obwohl in der Produktbeschreibung darauf hingewiesen wird, daß der IBM CICS Systems Manager auch ohne DCE läuft, werden in der Installationsanweisung Konfigurationen an der DCE verlangt. Daraufhin wurde eine Region mit DCE-Anbindung definiert und alle Installationsschritte des CICS SM wiederholt. Das Starten der Region war jedoch immer noch nicht mögich. Angemeldet als `root`, in der Gruppe `cicsadmin` und mit zusätzlicher Authentifizierung als DCE-Administrator ist es nicht möglich eine Region mit dem CICS SM zu starten, mit dem **AIX**-spezifischen Konfigurationswerkzeug `smitty` ist es mit diesen, und sogar weniger, Rechten möglich.
- Die **Agenten-Transaktion** `CAT` läßt sich in der Region lokalisieren, den Informationsfluß von ihr zum **Resource Manager** kann man jedoch nicht anzapfen, um Managementinformation zu bekommen. Dazu muß man über die zweite Komponente des Agenten, den *Resource Manager*, gehen, der eine SOM-Schnittstelle besitzt.

5.5 Vergleich zwischen CICS SM und den Anforderungen

Aufgrund der in der Analyse des IBM CICS SM gefundenen Leistungsmerkmale kann man nun bewerten, inwieweit sich die Anforderungen mit dem CICS SM abdecken lassen. Dazu werden die Szenarien unter dem Gesichtspunkt, daß die gefunenen Aufgaben mit dem CICS SM gelöst werden sollen.. Vorweg muß jedoch bemerkt werden, daß Management-Agenten für den IBM CICS SM, entgegen der ursprünglichen Ankündigung von IBM, nur auf AIX Systemen verfügbar sind. Daher können auch ausschließlich Informationen von CICS-Systemen, die unter AIX laufen vom CICS SM ermittelt und manipuliert werden. PC-Client-Systeme und CICS-Systeme auf Großrechenanlagen können derzeit nicht mit dem SM verwaltet werden.

5.5.1 Szenario 1: Herstellen einer Verbindung zwischen CICS-Systemen.

Als Verfeinerung des Szenarios 3.1.1 wird davon ausgegangen, daß der CICS-Administrator auf einer AIX-Maschine arbeitet. Da der CICS SM nur für CICS/6000 Systeme verfügbar ist, wäre ein Vergleich sonst wenig sinvoll.

1. Ermitteln des Namens des lokalen Systems
   Der *Application Identifier* (*APPLID*) und der *System Identifier* (*SYSID*) können über die graphische Oberfläche abgefragt werden. Über SOM sind diese Werte als Name einer Instanz der Klasse `bhg_cics6000_system` und im Parameter *localSysId* dieser Klasse zu erreichen.

2. Ermitteln des Namens des entfernten Systems
   Falls das entfernte System ein CICS/6000 ist und das System über eine TCP/IP-Verbindung zum lokalen System verfügt, kann diese Information ebenfalls problemlos abgefragt werden. Soll eine Verbindung zu einem Großsystem aufgebaut werden, so muß über systemspezifische Funktionen der Identifikator ermittelt werden.

3. Definition der Connection vom lokalen zum entfernten System
   Die sogennante *Half-connection* vom lokalen System zu einem Entfernten System läßt sich definieren. Die dazugehörige Klasse heißt `bhg_cics6000_half-connection`.

4. Definition der Connection vom entfernten zum lokalen System
   Ist das entfernte System ein CICS/6000 System, ist die Definition der *Half-connection* möglich, bei anderen Sytemen muß auf proprietäre Werkzeuge, wie zum Beispiel CEDA, zurückgegriffen werden.

5. Definition der Sessions für eine Connection
   Auf CICS/ESA-Systemen muß für jede Verbindung eine bestimmte Anzahl von Sessions angegeben werden, die aufgebaut werden sollen. Diese Sessions müssen zuvor definiert werden. Da im CICS/6000 keine Sessions explizit definiert werden müssen, stellt der SM auch keine Klasse dieser Art zur Verfügung. Die Anzahl der Sessions für eine Verbindung wird bereits bei der Connection Definition mit angegeben.

6. Aktivieren der Connection
   Das Aktivieren der Verbindung wir vom SM aus mit der Initialize-Methode der Klasse `bhg_cics6000_connection` durchgeführt. Auf anderen Systemen wird die Verbindung durch systemspezifische Werkzeuge aktiviert. Ebenso muß auf CICS/ESA für jede Verbindung eine Anzahl von Sessions angegeben werden, die aufgebaut werden dürfen. Diese Sessions müssen zuvor definiert werden. Da im CICS/6000 keine Sessions explizit definiert werden müssen, stellt der SM auch keine Klasse dieser Art zur Verfügung.

7. Testen der Verbindung (Session und Connection)
   Der Zustand einer Ressource wird im SM jeweils beim Anklicken des Symbols eines Betriebsmittels angezeigt. Es ist nur darauf zu achten, daß der Zustand nicht durch einen Polling-Mechanismus auf einem aktuellen Stand gehalten wird, sondern eine asynchrone Abfrage an das CICS-System abgesetzt werden muß, um den aktuellen Zustand zu diesem Zeitpunkt zu erhalten. Ob eine Verbindung jedoch wirklich verfügbar ist und nicht nur richtig definiert wurde, kann man erst beantworten, wenn eine Verbindung zu einem Remote-System aufgebaut wurde und eine Rückmeldung vom anderen System erfolgte. Das schließt ein Aktivieren von Sitzungen mit ein.

5.5.2 Szenario 2: CICS-Anwendung startet nicht.

1. Zuständige AOR identifizieren.
   Da diese Information in keinem CICS-System explizit gespeichert ist, kann auch der SM diese Information nicht ermitteln. Über den Umweg, auf der Terminal Owning Region die Remote-Eintragungen der Transaktionen beziehungsweise der Programme nachzusehen, ist eine AOR zu ermitteln, jedoch wird es schwierig, wenn mehrere Terminal Owning Regions zur Verfügung stehen und die Clients sich dynamisch an verschiedenen TORs anmelden.

2. Prüfen der Funktion der AOR.
   Mit den Discover-Methoden der System-Objekte kann man für CICS/6000 Systeme den Zustand und den Status einer Region abfragen. Für andere Systeme muß auf andere Mechanismen zurückgegriffen werden.

3. Log-Datei-Einträge der AOR auf Fehlermeldungen hin analysieren.
   Für jedes CICS/6000-System gibt es eine Methode an der Klasse, um ein *console.log* und ein *error.log* anzuzeigen. Da AIX ein UNIX Dateisystem verwendet und die Log-Dateien in Form von ASCII-Text in das /var-Verzeichnis geschrieben werden, können sie auch mit jedem Editor angezeigt werden.

4. Bei Fehlern die Ursachen prüfen.
   Ein Fehler, dessen Ursache bei den Betriebsmitteln einer Anwendung zu suchen ist, zum Beispiel ein Programm, das installiert wurde und dessen Status danach nicht auf *enabled* gesetzt wurde, wirft das Problem auf, alle Betriebsmittel einer Anwendung zu kennen. Dieses Problem löst der CICS SM nur zum Teil. Es ist möglich eine Anwendung innerrhalb eines Systems zu definieren und dieser Anwendung Betriebsmittel zuzuordnen. Das Anwendungs-Objekt gibt es aber nur im Kontext eines SYSplex. Da ein SYSplex aber jegliche Managementinformation der Systeme verschattet, wird man davon absehen, einen Plex zu definieren. [2] Man würde sich die Möglichkeit teuer erkaufen, die Information zu Ressourcen einer Anwendung speichern zu können, wenn mit einem Plex die Informationsgewinnung von Last-, Routing- und Prozeßstatusdaten wesentlich schwieriger wird. Daher muß man davon ausgehen, daß der SM die Anwendungen in Form eines Objektes nicht unterstützt. Die Klasse `bhg_application` kann jedoch in einem eigenen Werkzeug nach diesem Muster in einem allgemeinen Kontext implementiert werden.

5. Prüfen der Funktion der TOR.
   Hier gilt prinzipiell das gleiche, wie für die Funktionsprüfung der AOR, wenn man davon ausgeht, daß auch die TOR ein CICS/6000 System ist. Ist das nicht der Fall, muß abhängig vom System auf spezielle Werkzeuge zurückgegriffen werden.

6. Prüfen der Verbindung zwischen TOR und AOR.
   Eine Verbindung zwischen zwei Regions prüft man am besten durch den Aufbau einer *Routing Session*. Erst dann kann man sicher sein, daß die erbindung funktioniert. Der SM zeigt für jede *half-connection* den Status an. Dieser Status bezieht sich aber auf den Zeitpunkt des letzten Aufruf der Methode *discover* für die CICS-Region, in der die *half-connection* definiert ist.

7. Konfiguration des Clients prüfen.
   Um die Konfiguration des Clients zu prüfen sind detailierte Informationen aus den Konfigurationsdateien notwendig. Auf diese Informationen kann mit dem SM nicht zugegriffen werden. Ebenso ist ein Verwalten solcher Informationen nicht vorgesehen. Bei automatischen Software-Verteilsystemen für Clients könnte man die standardisierten Konfigurationsdateien, die dann abhängig vom Betriebssystem sind, lokal halten und bei Bedarf darauf zurückgreifen.

5.5.3 Szenario 3: Installieren neuer CICS-Programme

1. Sicherstellen, daß auf betroffene Ressourcen nicht zugegriffen werden kann, solange das neue Programm eingespielt wird.
   Für den Fall, daß ein sogenannter *Sysplex* definiert wurde, kann in der Klasse `bhg_application_definition` definiert werden, welche Ressourcen zu einer Anwendung gehören. Aus bereits erläuterten Gründen ist es aber für das integrierte Management nicht sinnvoll, einen SYSplex zu definieren. Demnach gibt es keine adäquate Möglichkeit, Transaktionen, Programme und andere Ressourcen zu einer Anwendung zu gruppieren.

2. Falls bereits eine frühere Version installiert wurde.

   - Ermitteln der aktuellen Programmversion.
     Die aktuelle Version eines Programmes kann in den Attributen `version` und `release` der Klasse `bhg_cics6000_program` gespeichert werden.

[2] Ein Plex aus mehreren Systemen wird nach aussen hin wie eine einzige Region dargestellt. Lastinformationen oder Routen für Transaktionen und Programme sind von aussen nicht mehr sichtbar.

- Erforderliche Version für den gewünschten Update der aktuellen Version prüfen.
  Eine Versionenkontrolle wie bei Projektmanagement-Anwendungen wird vom SM nicht unterstützt.

3. Falls ein Programm erstmalig installiert wird.

   - Festlegen der AOR für diese Anwendung.
     Die ausgewählte AOR kann keine Informationen speichern, die installierte Anwendungen betreffen. Ebensowenig kann eine Anwendung explizit speichern in welcher AOR sie installiert wurde.
   - Zugriff auf Systeminformationen der verschiedenen AOR, um Vorraussetzungen der Anwendung in einer AOR zu prüfen.
     Lastinformationen können durch das Attribut *system-health* des Objektes *bhg_cics6000_system* repräsentiert werden. Verfügbaren Plattenplatz erhält man über die entsprechenden Betriebssystemkommandos. Zugriffsmöglichkeiten auf Datenbanken und weitere Betriebsmittel müssen an anderer Stelle gehalten werden, das kann auch unter AIX der SM nicht verwalten.

4. Einspielen der Programme und Dateien der neuen Anwendung.
   Das kann vom jeweiligen Betriebssystem übernommen werden und bedarf keiner Funktionalität das SM.

5. Anlegen der *Resource Definitions* für Programme, Transaktionen, Dateien, Verbindungen und Datenbankzugriffe. Mit dem CICS SM können alle notwendigen Definition getroffen werden, die CICS-Systeme betreffen. Zugang zu Datenbanken kann man mit SM nicht definieren. Falls ein externes Sicherheitssystem und nicht DCE oder das CICS-eigene Sicherheitssystem verwendet wird, können mit dem CICS SM keine Benutzer für Zugriffe berechtigt werden.

6. Aktivieren der neuen Programme, Transaktionen und Dateien.
   Über das Attribut *enable status* können die neu eingespielten Betriebsmittel in einen aktiven Status versetzt werden. Daß auf die jeweils neue Version eines Betriebsmittels zugegriffen wird und nicht auf eine alte Version, die in einem Zwischenspeicher steht erreicht man bei Programmen mit der Methode `newcopy`, für Transaktionen benötigt man keine derartige Methode, da neue Transaktionen erst dann verwendet werden, wenn die gerade laufende Transaktion mit dem selben Namen beendet wurde. Bei Dateien wird erst nach dem Schließen einer Datei beim nächsten Zugriff eine neue Version verwendet.

7. Bei CICS/ESA veranlassen, daß DB2 RCT gepflegt werden.
   Entfällt bei CICS/6000. Zugriff auf CICS/ESA ist über SM nicht möglich.

8. Zugriff auf die neue Version ermöglichen. Das schließt folgende Teilbereiche ein:

   - Eintragung der Definitionen für die AOR für die Anwendung in der TOR.
     Remote-Eintragungen von Betriebsmitteln werden vom SM unterstützt, jedes Betriebsmittel, auf das von der CICS-Konzeption einen remote-Zugriff erlaubt, kann auch über den SM auf einem entfernten System definiert werden, falls das entfernte System ein CICS/6000-System ist.
   - Benutzer für die neue Version berechtigen.
     Die Vergabe von Berechtigungen für Benutzer ist mit dem CICS SM nur für das CICS-interne Sicherheitssystem und in der DCE-Cell möglich. Eintragungen in externen Sicherheitssystemen, wie dem weitverbreiteten RACF in MVS/ESA-Systemen, wird ebensowenig unterstützt, wie die notwendigen Eintragungen in Datenbanksystemen,
   - Eintragung der Benutzer für die Anwendung im Sicherheitssystem veranlassen. (Zugriff auf Programme, Transaktionen, Systembibliotheken, Queues, usw.)
   - Eintragung der Benutzer bzw. der Anwendung im Datenbanksystem veranlassen.

5.5.4 Anforderungen, die nicht mit dem CICS SM abgedeckt werden.

1. Eine Zuordnung von Ressourcen zu Anwendungen ist grundsätzlich nur in Verbindung mit der Definition eines CICSplex möglich. Definiert man jedoch einen solchen CICSplex, ist es wegen technischer Mängel am CICS SM trotzdem nicht möglich, Betriebsmittel einer Anwendung zuzuordnen, obwohl dies so dokumentiert ist.

2. In einer Region kann nicht gespeichert werden, welche Anwendungen lokal installiert sind, beziehungsweise welche Anwendungen über Remote-Mechanismen aus einer Region heraus aufgerufen werden können.

3. An einer Anwendung kann nicht gespeichert werden, in welchen Regions sie installiert ist. Das würde die Fehlersuche wesentlich erleichtern, wenn nur bekannt ist, daß sich eine Anwendung nicht starten läßt.

4. CICS-Clients sind im Konzept des CICS SM anscheinend nicht vorgesehen. Daher ist die Mindestanforderung nicht erfüllt, den Inhalt der Konfigurationsdateien eines CICS-Clients, die CICSCLI.INI und bei PC-Systemen die CONFIG.SYS und AUTOEXEC.BAT, als Text-Datei zu speichern.

5. Zuordnung einer Region zu Zugriffsmöglichkeiten auf besondere Betriebsmittel, wie zum Beispiel Datenbanken.

6. Definition von Zugriffen auf Datenbanken.(z. B. über eine XA-Schnittstelle)

7. Das Eintragen von Accounts für Anwender in einem externen Sicherheitssystem, wie dem Security Service der DCE, wird nicht unterstützt.

5.5.5 Das Objektmodell des IBM CICS Systems Manager

In der Dokumentation sind keine detaillierten Angaben zu finden, welche **SOM/DSOM**-Klassen vom CICS SM für das Management verwendet werden. Lediglich die Klassen werden in einem Produktüberblick genannt, diese jedoch scheinen direkt aus der **Resource Definitions** des CICS hervorgegangen zu sein. Über Attribute oder gar Methoden ist leider nichts zu finden. Diese Information ist laut Dokumentatin nur über ein C++-Programm (siehe B.1) zu erhalten, das mit SM-API Methoden die Methoden, Attribute und enthaltenen Objektes eines Objektes ausgeben kann. Ein sehr kleiner Auszug aus dem Objektmodell des CICS SM ist in Abbildung 5.7 dargestellt. Es wurden absichtlich die Attribute und Methoden der einzelnen Klassen weggelassen, weil sonst das Modell unlesbar wäre. Jede der Klassen hat etwa 40 Attribute und 10 Methoden. Das gesamte Klassenmodell besteht aus 445 Klassen, die über Mehrfachvererbung (diamond-inheritance) miteinander in Beziehung stehen. Darunter sind mehrere Klassen, die mehr als 100 Attribute besitzen (zum Beispiel hat die Klasse bhg_cics6000_system 143 Attribute). Diese enorme Zahl von Klassen kommt dadurch zustande, das für jede Klasse, die man als Ressource in einem System finden könnte, fünf weitere Klassen für die Definition, ein Topologieelement, ein Clone, ein Modell und eine Schablone angelegt wurden. Der praktische Nutzen bleibt dabei oft im Verborgenen. Beispielsweise kann man ein Topologieelement keineswegs dafür verwenden, eine Topologiekarte einer Domäne oder eines Systems anzulegen. Es zeigt nicht einmal den Status des Objekten an, dem es zugeordnet ist. Damit sind genau die Klassen noch interessant, die den Suffix *definition* tragen, da in ihnen die Parameter einer *Resource Definition* des CICS-Systems gespeichert werden.

5.5.6 Installation und Konfiguration des IBM CICS Systems Manager

Die Installation des CICS SM von einer CD-ROM benötigte ca. 2 Stunden und läuft automatisch ab. Damit hat man aber noch kein funktionsfähiges Werkzeug. Erst nach der Installation des ORB von IBM, dem *somdd* ist der CICS SM theoretisch lauffähig. Die Konfiguration des somdd bereitete etwas mehr Schwierigkeiten und es mußten mehrere Anläufe unternommen werden, für den somdd einen Port zu bestimmen, über den auch der CICS SM kommunizieren konnte, da sich der CICS SM nicht starten läßt, wenn kein somdd läuft, war es ein längerer Prozeß beide Komponenten in das System einzubinden. Nachdem sich der

CICS SM starten ließ, mußte noch eine Region mit DCE-Unterstützung auf dem CICS/6000 eingerichtet werden, da der SM für Regions ohne DCE, nur einen begrenzten Funktionsumfang bereitstellte. Nachdem der CICS SM seine Agenten-Transaktion in der CICS-Region installieren konnte, waren viele der Funktionen des SM verwendbar. Auf Regions ohne DCE-Unterstützung läßt sich die Agenten-Transaktion nicht installieren, obwohl das Management von solchen Regions im SM vorgesehen ist.

5.6 Weitere Anforderungen an ein integriertes Management

1. Unterstützung von Ressource Definitionen auf anderen Plattformen, wie zum Beispiel CICS/ESA und CICS for OS/2.
2. Die Klasse 'Session' muß für CICS-Systeme auf MVS/ESA und OS/2 entwickelt und implementiert werden, da sie dort benötigt wird, jedoch in CICS/6000-Systemen nicht vorgesehen ist.
3. Profiles, die beispielsweise in CICS/ESA für Transaktionen benötigt werden sind in CICS/6000 Systemen nicht vorgesehen.
4. Der Lastzustand von MVS/ESA und OS/2 Regions kann nicht ermittelt werden.

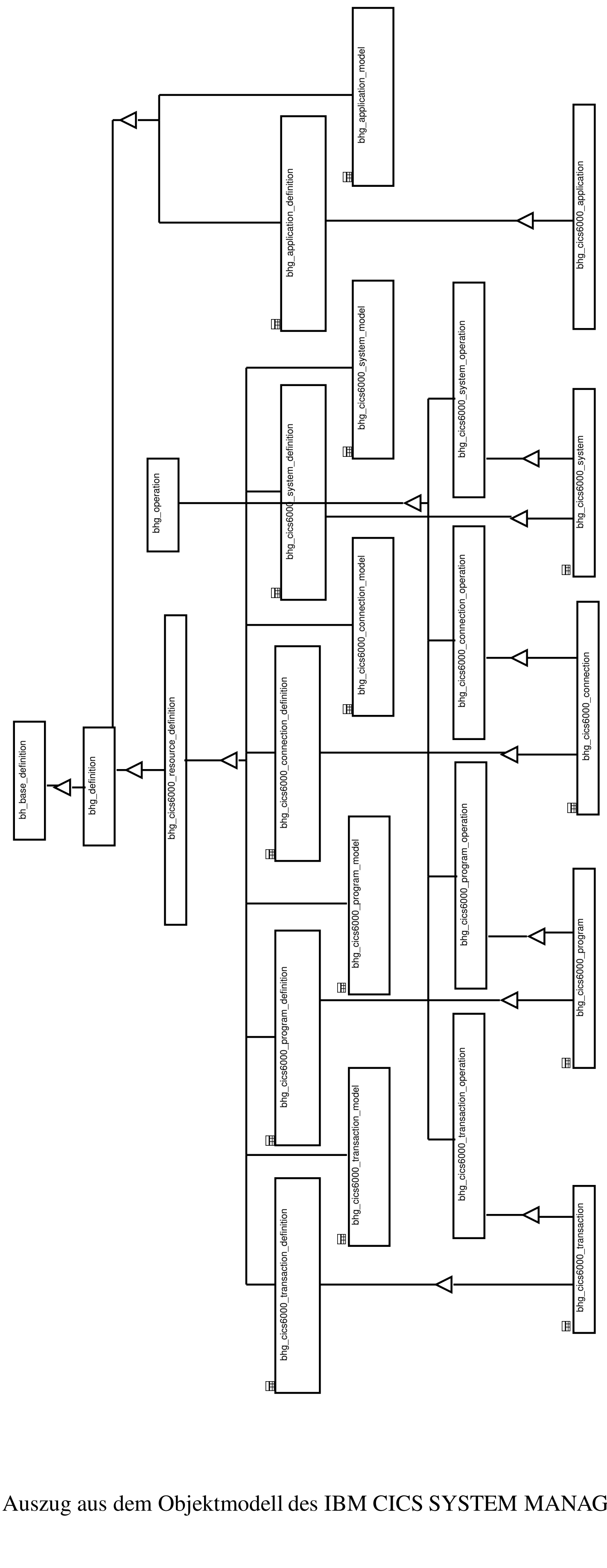

Abbildung 5.7: Auszug aus dem Objektmodell des IBM CICS SYSTEM MANAGER for AIX

Kapitel 6

Die Implementierung des CICS-Domain-Managers

Die Implementierung kann im Rahmen dieser Arbeit nur einen prototypischen Ansatz bieten, auf den während der Implementierungsphase zurückgegriffen werden kann. Es werden die Kommunikationsmechanismen von SOM/DSOM über den ORB auf Verwendbarkeit geprüft und Beispiele für die Navigation in einem Objektbaum gegeben. Der erste Abschnitt soll jedoch einen Überblick bieten, in welchem Kontext der CICS Domain Manager gesehen werden muß.

6.1 Das Konzept

Der CICS Domain Manager ist innerhalb des Domain Transaktion Manager (DTM) eingebettet in die Domain Manager für andere Transaktionsmonitore. Wie in Abbildung 6.1 zu erkennen, bildet der DTM einen Rahmen für die einzelnen Domain Manager. In ihm sind die allgemeinen Klassen enthalten, die für das Management aller Transaktionsmonitore von Bedeutung sind. System, Transaktion und Programm stellen die wichtigsten Elemente dar und sind auch ohne exakte Kenntnis der Betriebsmittel eines Produktes, für das Management zu identifizieren. Unter dem DTM werden die Spezialisierungen für die konkreten Transaktionsmonitore implementiert, wobei es notwendig sein kann (zum Beispiel bei CICS) zusätzlich nach Plattformen zu unterscheiden. Diese Unterscheidung wäre beispielsweise bei openUTM nicht zwingend notwendig, da dieser bereits eine einheitliche Architektur für alle unterstützten Plattformen hat. Die spezialiserten Klassen für die verschiedenen Plattformen müssten die eigentliche Implementierung, vor allem der Methoden, enthalten, da sonst sehr viel in den verallgemeinerten Klassen implementiert werden müsste. Das hätte den Nachteil, daß bei Änderungen am Konzept der Ressoure Definitionen für eine Plattform, zur Anpassung des Klassenmodells viele der verallgemeinerten Klassen von der Änderung betroffen sind.

Am Beispiel des CICS ist alles oberhalb der Spezialisierungen für die Plattformen nur in Form von abstrakten Klassen sinnvoll, nachdem bereits bei grundlegenden Betriebsmitteln, wie zum Beispiel Verbindungen, verschiedene Konzepte vorliegen, Beziehungen zwischen den Definitionen im System abzubilden.

Die Implementierung des CDM stützt sich zu großen Teilen auf die Klassen des CICS System Manager, da in den SM-Objekten die Management-Information gespeichert ist und Management-Funktionalität bereitgestellt wird. Der Management-Agent des CICS SM, der aus einer Agenten-Transaktion und einem Resource Controller besteht stellt die Management-Information und -Funktionalität über eine DSOM-Schnittstelle bereit. Über einen Object Request Broker (ORB), der bei DSOM auf AIX in Form eines Dämonen (somdd) im System läuft, greift der CDM auf den Resource Controller zu.

Für andere Plattformen als AIX, zum Beispiel OS/2 oder MVS/ESA muß auf einen anderen Kommunikationmechanismus zurückgegriffen werden. Auf OS/2 gäbe es zwar eine Implementierung von DSOM, aber es stehen noch keine Agenten für dieses System zur Verfügung. Für CICS aufMVS/ESA ist bereits ein Management-Agent angekündigt, jedoch ist auf diesem System kein ORB implementiert, so daß die Vermutung naheliegt, daß die Kommunikation zumindest auf MVS/ESA über einen anderen Mechanismus abgewickelt werden muß. Vorschläge für solche Mechanismen finden sich in den Abschnitten 6.3.2 und 6.3.3, in denen die Möglichkeiten einer eigenen Implementierung von Management-Agenten für diese Systeme erörtert wird.

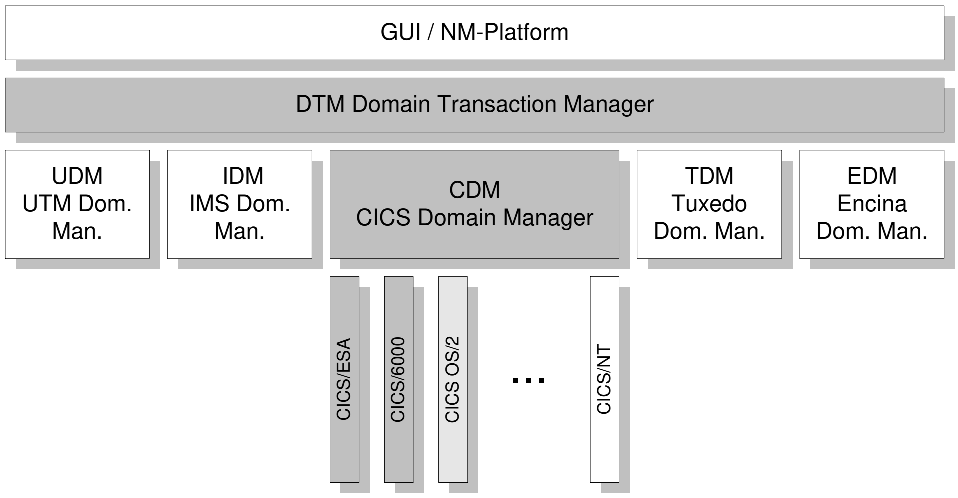

Abbildung 6.1: Die Architektur des Domain Transaction Manager

6.2 Der Manager

Im Rahmen dieser Diplomarbeit wurde nur der CICS-Domain-Manager mit der Spezialisierung auf ein CICS/6000-System implementiert, der unter AIX 4.1 läuft und auf einen Agenten in Form eines Resource Controllers, der ebenfalls auf einem CICS/6000-System installiert ist, zugreift. Das hat den Hintergrund, daß es bis jetzt genau diesen Management-Agenten gibt, den man über eine CORBA-konforme Schnittstelle, den Objekt-Request-Broker, ansprechen kann. Die Grundlagen für solche Agenten gibt es schon seit einiger Zeit von der OMG, jedoch hat bis jetzt außer der IBM niemand eine Implementierung gewagt. Sogar die Firma Siemens, die vor sechs Jahren damit begonnen hat ihren openUTM völlig neu zu gestalten und ein integriertes Management über Plattformgrenzen hinweg für ihr Produkt anbietet, verwendet für das Management wieder SNMP.

Der CICS-Domain-Manager (CDM) übernimmt Verwalteraufgaben, indem er auf die Informationen zugreift, die von einem Resource Controller (RC) in Form eines Agenten in einem CICS-System zur Verfügung gestellt werden. Über einen Object-Request-Broker (ORB), in diesem Fall der somdd-Dämon von IBM, kann auf die Instanzen zugegriffen werden. Dieser Zugriff ist mit einem für DSOM üblichen Verfahren realisiert: Über einen Cursor, ähnlich einem Datenbank-Cursor, wird durch den Objektbaum navigiert und ein bestimmtes Objekt ausgewählt. Auf dieses Objekt können verschiedene Methoden angewendet werden, wie zum Beispiel das Abfragen des Klassennamen, der Attribute und Methoden. Für die Navigation ist jedoch die Methode zum Ermitteln der enthaltenen Objekte des aktuellen Objektes am wichtigsten. Ohne sie wäre eine Navigation im Objektbaum nur schwierig zu realisieren, da man für eine absolute Adressierung eines Objektes seinen genauen Namen benötigt und den exakten Pfad über die verschiedenen, darüberliegenden

Objekte. Selbst wenn man diese Informationen besitzt, ergeben sich rein praktische Probleme, ein Objekt zu adressieren, weil abhängig von der Position eines Objektes im Pfad, verschiedene Trennzeichen in der Pfadangabe verwendet werden müssen. Für die absolute Adressierung von Objekten sollte man daher auf die Erweiterungen der IBM (Angekündigt Ende 1996) zurückgreifen, die den Object Naming Service um diese Möglichkeiten ergänzen. Abbildung 6.2 macht deutlich, daß der Management-Agent die Funktion eines Servers übernimmt und der CDM als Client Anfragen an den Agenten stellt.

Für die Navigation im Objektbaum hat der CDM eine Textschnittstelle, über die aus einer Liste von Objekten eines zur Bearbeitung ausgewählt werden kann. Durch die Baumstruktur reicht es aus, als Eingabeparameter für den Manager nur den Namen für den Resource Contoller anzugeben. Wurde eine Verbindung zu diesem Server hergestellt, werden alle Instanzen der ersten Ebene angezeigt: Nodes, DCE-Cells, SFS-Server und CICS-Systems. Aus diesen wählt man ein Objekt aus, zu dem wiederum alle enthaltenen Objekte zur Auswahl angeboten werden. Sollten zu einem späteren Zeitpunkt die Erweiterungen des Object-Naming-Service (ONS) der OMG implementiert worden sein, ist neben der relativen Adressierung eines Objektes ein absolutes Adressieren möglich. Ist man bei dem Objekt angekommen, das man ändern möchte, wählt man *Editieren von Attributen* aus und bekommt die aktuellen Werte aller Attribute des Objektes angezeigt und kann einen neuen Wert für das Attribut eingeben und speichern.

Damit sind alle wichtigen Funktionen des Managers verifiziert:

- Navigieren zwischen den Objekten eines Management-Agenten (Ressource Controller)
- Auswahl eines bestimmten Objektes
- Ermitteln der Management-Information und Nutzung der Funktionalität eines Objektes
- Manipulation der Attribute eines Objektes

6.3 Die Management-Agenten

Um Informationen aus einem System zu erhalten, hat es sich durchgesetzt, einen **Management-Agenten** für das betreffende System zu implementieren, der die Management-Anwendung oder besser eine Management-Plattform über eine definierte Schnittstelle mit den Daten aus dem System versorgt. Bisher gibt es nur einen Agenten für CICS/6000 auf AIX-Systemen, der über einen ORB kommunizieren kann und für das Management von Transaktionsmonitoren implementiert wurde. Die Agenten für CICS-Systeme unter MVS/ESA und OS/2 sind angekündigt, aber noch nicht verfügbar.

Nachdem IBM jedoch den TME 10 Management Server von Tivoli übernommen hat, bleibt abzuwarten, ob die Implementierung dieser objektbasierten Agenten noch erfolgt, da TME 10 ein SNMP-basiertes System darstellt.

6.3.1 Die Agenten-Transaktion CAT im CICS/6000

Im CICS/6000 gibt es bereits die **Agenten-Transaktion CAT**, die beim IBM CICS **System Manager** mitgeliefert wird und zusammen mit einem **Resource Controller** den Agenten bildet. Der Ressource Controller steht über UNIX-sockets mit der Agenten-Transaktion in Verbindung. Der Zugriff auf die Daten des Resource Controllers geschieht über einen **CORBA**-konformen **Object Request Broker** (**ORB**), der die Daten aus den CORBA-Objekten zur Management-Anwendung transportiert. Das bisher im Management häufig benutzte **Simple Network Management Protocol** (**SNMP**) wird hier nicht verwendet. Für die erste Implementierung wird auf diesen Agenten für CICS/6000 zurückgegriffen. Die Management-Information, die der Agent liefert, umfaßt Definitionen und Monitoring des CICS-Systems, sowie Eintragungen in der **DCE-Cell**. Die Agenten-Transaktionen für CICS/ESA- und OS/2-Systeme sind trotz der Ankündigung im Juni 1996 noch nicht verfügbar. Sollten auch in Zukunft keine weiteren Agenten für CICS-Systeme lieferbar sein, muß selbst ein Agent implementiert werden. Die folgenden Abschnitte geben einen groben Überblick über die Werkzeuge, die bei der Implementierung eines Management-Agenten für CICS-Systeme die benötigte Information und Funktionalität bereitstellen könnten.

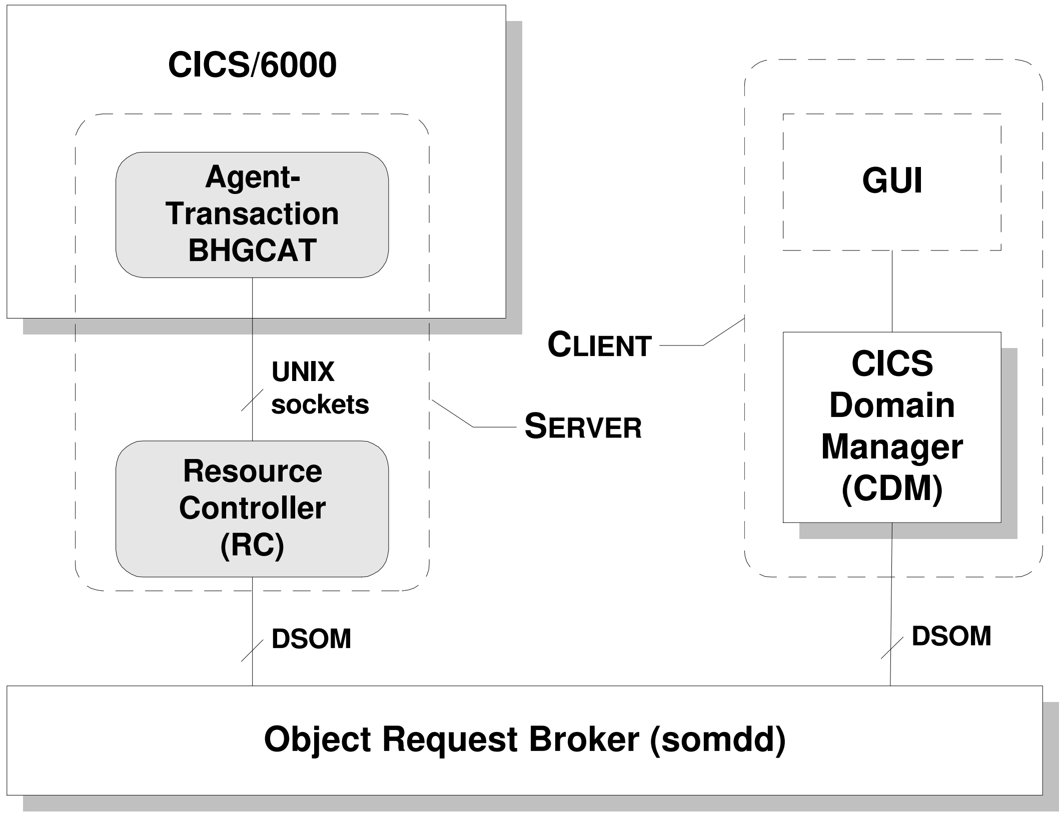

Abbildung 6.2: Die Architektur des CICS Domain Manager

6.3.2 CICS/ESA

Für die Implementierung eines Management-Agenten auf CICS/ESA-Systemen kommen folgende Programme aus dem Lieferumfang des CICS/ESA in Frage:

- Mit **DFHCSDUP** kann Management-Information aus einem CICS-System gewonnen werden. Das Programm sollte dazu aus einem eigenem CICS-Programm heraus aufgerufen werden. Das eigene Programm startet man im Rahmen einer Transaktion bei jedem Start des CICS-Systems. Das erreicht man durch einen Eintrag in der **System Initialisation Table** (**SIT**). Über Funktionsaufrufe der CICS-API können mit DFHCSDUP die Definitionen in einem bestehenden System ermittelt und verändert werden. Ebenso trägt man neue Definitionen in das System ein und strukturiert diese in Gruppen. Nachdem inzwischen auch die Programmiersprache C auf Großrechnern verfügbar ist, kann ein solches Programm portabel gehalten und ebenso für OS/2 eingesetzt werden.

- Für die Auswertung von Statistiken eines im Betrieb befindlichen Systems kann das Programm **DFHSTUP** auf ähnliche Weise, wie das Programm **DFHCSDUP** einen Teil der Funktionalität eines

Agenten übernehmen. Man müßte jedoch einen Zwischenschritt einfügen, in dem die Rohdaten ausgewertet und formatiert werden, bevor die Information zum Manager geht.

- Mit der **CEMT**-Transaktion wertet man dynamische Vorgänge in einem CICS-System aus. Folgende Betriebsmittel können mit CEMT manipuliert und abgefragt werden:
  - Connection
  - File
  - Program
  - Session
  - Task
  - Terminal
  - Trace
  - Transaction
  - User

  Die CEMT-Transaktion kann zum Beispiel den Status einer Transaktion abfragen, die aktivierten Sitzungen oder angemeldete Anwender anzeigen.

Die Informationsübermittlung zwischen dem Agenten und dem Management-Werkzeug wäre über die CICS-Client-API denkbar, wobei die Alternativen **External Call Interface** (**ECI**) oder **External Presentation Interface** (**EPI**) zur Verfügung stehen. Die API kann in den Programmiersprachen C, COBOL und PL/1 verwendet werden. Eine Kommunikation über den **ORB** ist noch nicht möglich, da es bisher weder SOM/DSOM, noch einen anderen ORB für Großrechner-Systeme gibt.

6.3.3 CICS for OS/2

Unter OS/2 werden die Systeminformationen in einer **BTRIEVE**-Datenbank gehalten. Für die Manipulation der Definitionen steht eine Programmierschnittstelle für C, COBOL und PL/1 zur Verfügung. In C verändert man Definitionen über einen Funktionsaufruf. Statistiken können über den **Performance Analyser** (**PA**), der bei CICS for OS/2 WARP mitgeliefert wird, erstellt und ausgewertet werden. Der PA erstellt eine Datei mit den Statistik-Daten, die als Standardeinstellung den Namen PA2STATS.DAT trägt. In ihr sind Hardware- und Software-Statistiken von Betriebssystem und Transaktionsmonitor enthalten.

Kapitel 7

Zusammenfassung

Um sich einen Überblick über die Anforderungen für das integrierte Management von Transaktionsmonitoren zu verschaffen, wurden Szenarien aus den Bereichen CICS-Systemadministration und Anwendungs-Administration innerhalb von Transaktionsmonitoren herangezogen. Diese Szenarien kommen einerseits aus der Administration des MVS-Rechenzentrums der BMW AG und zum anderen aus der Administration von CICS-Systemen auf AIX- und OS/2-Rechnern der Quelle Versicherungen und des Gerling Konzerns. Die Praxisnähe bedingt an manchen Stellen der Arbeit einen sehr technischen Umgang mit dem Thema, jedoch wurde durch anschließende Abstraktion versucht, eine globalere Sicht auf das Management von TP-Monitoren zu erreichen. Aus der Vielzahl von Szenarien, die im Betrieb eines Transaktionsmonitors vorkommen, wurden drei Szenarien ausgewählt und genau analysiert. Folgende Kriterien spielten bei ihrer Auswahl eine Rolle:
Die Szenarien sollten in sich abgeschlossen sein, um möglichst eine Einschränkung auf das Management von Transaktionsmonitoren zu erreichen und nicht die Administration von Sicherheitssystemen, Dateiverwaltung und Datenbankmanagement miteinbeziehen zu müssen.
Darüberhinaus sollten die Beispiele allgemein genug formuliert sein. Dadurch wird ein großer Bereich der Aufgaben abdgedeckt, ohne sich in Spezialfällen zu verlieren.
Nachdem drei geeignete Szenarien identifiziert waren, wurden ihre Rahmenbedingungen, wie die zugrundeliegenden Betriebssysteme und Kommunikations-Protokolle, festgelegt. Danach konnte mit der Erarbeitung der Anforderungen begonnen werden. Hierbei lag der Schwerpunkt auf dem CICS/6000-System, wobei auf Abweichungen der MVS- und OS/2-Systeme von der AIX-Version hingewiesen wurde, um sie für eine künftige Realisierung auf diesen Systemen zu identifizieren.
Die gefundenen Anforderungen sind keineswegs vollständig und decken, wie bereits weiter oben erwähnt, nur den allgemeineren Teil der Administration ab. Für das Modell ist es jedoch ausreichend, da bereits mit dieser Teilmenge die Zusammenhänge in einem Transaktionsmonitor dargestellt werden können.
Mit der Identifizierung der Anforderungen ist der erste Teil des in der *Object Modeling Technique (OMT)* von Rumbaugh vorgeschlagenen Vorgehen, der Analyse des Problemfeldes, abgeschlossen. Auf eine Modellierung der Szenarien in Form eines *Use-Case-Modells* wurde verzichtet, da keine zusätzliche Information damit gewonnen werden kann und eine informelle Darstellung mehr Spielraum für ergänzende Anmerkungen läßt.

Die zweite Phase der OMT ist das Systemdesign, in dem das eigentliche Klassenmodell entworfen wird. Der Entwurf stützt sich auf die Anforderungen, die in der Analyse-Phase gewonnen wurden. Nachdem die Klassen identifiziert wurden, können aus den Anforderungen an die Management-Information die Attribute der einzelnen Klassen und aus der Management-Funktionalität deren Methoden abgeleitet werden.
In Kapitel 4 wird dazu, entlang der Szenarien, aus den Anforderungen das Klassenmodell des CICS Domain Managers entwickelt. Anschließend wurden die Beziehungen zwischen den Klassen eingetragen und das Modell um ergänzende Klassen erweitert. Zum Beispiel gehen die Klassen mit dem Prefix *CDM_AIX_Domain* nicht unmittelbar aus den Anforderungen hervor, da keine grundsätzlich neue Information abgebildet wird. Diese Klassen enthalten jedoch Informationen, die nur über mehrere Objekte hinweg explizit ermittelbar ist und dienen damit der Strukturierung von Information. Nachdem die Modellierung der Klassen abgeschlos-

sen ist, sollte eine Analyse von vorhandenen Werkzeugen folgen, um den Bereich der Anforderungen zu identifizieren, der mit diesen Produkten bereits abgedeckt werden kann.

Das Kapitel 5 stellt Werkzeuge aus dem MVS/ESA- und AIX-Bereich vor, die dort zur Administration von CICS-Systemen eingesetzt werden.

Besonders wird auf das Management-Werkzeug *CICS System Manager for AIX (CICS SM)* eingegangen, das für CICS/6000-Systeme verwendet werden kann und einen Management-Agenten für diese Systeme anbietet. Der Agent wird dem CICS Domain Manager zur Informationsgewinnung dienen und für die prototypische Implementierung eine Rolle spielen. Deshalb wurde dem Vergleich des CICS SM gegen die Anforderungen ein eigener Abschnitt gewidmet, in dem als Ergebnis festgestellt werden mußte, daß alle Anforderungen, die über eine Abbildung der Systeminformation auf Objekte hinausgehen vom CICS SM nicht erfüllt werden können. Die einzige Ausnahme hierbei ist die Abbildung von CICS-Verbindungen auf eine Klasse, die eine globalere Sicht auf alle CICS-Verbindungen eines Knotens, also mehrerer CICS-Systeme, ermöglicht. Die anderen vorgestellten Werkzeuge sind in erster Linie Tabelleneditoren für Systemtabellen und bieten nur mit hohem Aufwand eine wenig befriedigende Integrationsmöglichkeit in ein übergreifendes Management-Konzept.
Nach dem Exkurs zu heute eingesetzten Werkzeugen wurde mit der Implementierung eines Prototypen begonnen.

Die Implementierung, der dritte Abschnitt der OMT, beschränkt sich auf die Klassen-Modelle für einen CICS/6000-Manager, dem die Plattform IBM AIX als Basis dient, und dem Prototypen eines Managers, der die Informationen aus einem CORBA-Agenten nutzt und die Kommunikation über einen Object-Request-Broker realisiert. Dieser Prototyp kann als Grundlage bei der weiteren Implementierung dienen, und zeigt die Vorgehensweise bei der Kommunikation mit SOM/DSOM von IBM in verteilter Umgebung. Bei den Plattformen fiel die Wahl auf IBM AIX, weil für diese Umgebung ein Object-Request-Broker und der einzige CORBA-konforme Agent zur Verfügung steht. Der Manager ist mit einer Textschnittstelle implementiert kann innerhalb der baumartig strukturierten Instanzen von Objekten navigieren. Ebenso können Objekte zur Bearbeitung ausgewählt und deren Attribute gelesen und verändert werden. Damit sind die wesentlichen Hürden bei der Implementierung des CDM genommen und es können die Klassenrümpfe, die aus dem Entwicklungswerkzeug *StP* generiert wurden, mit Funktionalität gefüllt werden.

Eine Implementierung von Management-Agenten wurde in dieser Diplomarbeit ausgeklammert, jedoch wurden im Zusammenhang mit der Analyse vorhandener Produkte und im Abschnitt 6.3 Implementierungsvorschläge gemacht, wie Management-Information aus den verschiedenen CICS-Systemen gewonnen werden kann und welcher Schnittstellen man sich bedienen kann. Diese Ausführungen sind aber bewußt kurz gehalten, da es Ankündigungen gibt, daß Management-Agenten bald auch für CICS-Systeme auf MVS/ESA und für OS/2 verfügbar sein werden.

Anhang A

Abkürzungsverzeichnis

A

ABEND	Abandon ended
AOR	Application Owning Region
AP	Application Domain
APPC	Advanced Program to Program Communication
ASE	Application Service Element

C

CDM	CICS Domain Manager
CORBA	Common Object Request Broker Architekture
CICS	Customer Information and Control System
CPI-C	Common Programming Interface for Communications
CPI-RR	Common Programming Interface for Resource Recovery
CRM	Communications Resource Manager

D

DCE	Distributed Computing Environment
DIN	Deutsches Institut für Normung
DOR	Data Owning Region
DPL	Distributed Program Link
DSOM	Distributed System Object Model
DTP	Distributed Transaction Processing

E

ECI	External Call Interface
EPI	External Presentation Interface

I

IMS	Information Management System
IP	Internet Protocol
IRC	Interregion Communication
ISAM	Index Sequential Access Method
ISC	Intersystem Communication
ISO	International Standardisation Organisation

K

KDCS	Kompatible Daten-Kommunikations-Schnittstelle

L

LU1	Logical Unit 1
LU2	Logical Unit 2
LU6.1	Logical Unit 6.1
LU6.2	Logical Unit 6.2
LUW	Logical Unit of Work

M

MRO	Multiregion Operation

O

OMT	Object Modelling Technique
ORB	Object Request Broker
OSF	Open Software Foundation
OSI	Open System Interconnection
OTS	Object TransactionService

P

PCT	Program Control Table
PU	Physical Unit

R

RACF	Resource Access Control Facility
RD	Region Definition
RJE	Remote Job Entry
ROR	Resource Owning Region
RPC	Remote Procedure Call
RTI	Remote Task Invocation

S

SFS	Structured File Server
SIT	System Initialisation Table
SNA	Systems Network Architecture
SNMP	Simple Network Management Protocol
SOM	System Object Model
StP	Software through Pictures

T

TCP	Transmission Control Protocol
TCS	Connection and Session Table
TD	Transaction Definition
TE	Terminal Domain
TOR	Terminal Owning Region
TP-Monitor	Transaction Processing Monitor
TSO	Time Sharing Option
TxRPC	Transactional Remote Procedure Call

V

VTAM	Virtual Telecommunication (Terminal) Access Method

Z

ZC	External Communication Domain

Anhang B

Code

B.1 ccsmread.C

```
/*
 * Sample Program to read an object-tree
 * Bernhard Fischer, 23.08.96
 */

#include <stdio.h>
#include <iostream.h>          // for cout/cerr
#include <somd.xh>             // DSOM header file
#include "bhcursor.xh"         // BHGCursor class
#include "bhfreem.xh"          // BHGFreeMem class
#include "bhfree.hh"           // BHGFree convenience functions
#include "bherror.xh"          // BHGErr module
#include "bhany.xh"            // BHGAny class
#include "ccsmread.xh"          // ccsmread header file

/*
 * FUNCTION:     nice
 * DESCRIPTION:  Format an Tree-Strukture of found objects.
 */
void
nice(int depth)
{
  for (int i=0; i<depth;i++)
    {
      cout << "  |";
    }
}

/*
 * CLASS:        CDMCursor
 * DESCRIPTION:  Privat cursor-definition for CICSDomainManager
 */
class CDMCursor
{
public:
  Environment &theEnv;
```

```
BHGCursor * theCursor;
int depth;

CDMCursor(Environment &env, int i=0)
  : theEnv(env),
    depth(i)
{
  theCursor = new BHGCursor;
}

CDMCursor(const CDMCursor &aCursor)
  : theCursor(new BHGCursor(aCursor.theCursor)),
    theEnv(aCursor.theEnv),
    depth (aCursor.depth+1)
{
}

~CDMCursor()
{
  delete theCursor;
}

operator string ()
{
  return theCursor->getName(&theEnv);
}

/*
 * METHOD:      Standard-Methods of BHGCursor for CDMCursor
 * DESCRIPTION:
 */
int
getContents(string filter,
            _IDL_SEQUENCE_BHGCursor_ObjectDescription *od)
{
  return theCursor->getContents(&theEnv, filter, od);
}

int navigate(string object)
{
  return theCursor->navigate(&theEnv, object);
}

BHGErr_ErrorInfo
getLastError()
{
  return theCursor->getLastError(&theEnv);
}


/*
 * Check that the attribute value (argv[4]) can be parsed
 * as a value of the correct type
*/
```

```
long setAttributeValue( Environment ev,
                        any *       default_value,
                        any         attribute_value,
                        char *      any_value)
{

  TCKind attribute_type;
  long   att_rc = 0;

  BHGAny     *any_obj =
    (BHGAny*) _BHGAny->sommGetSingleInstance(&ev);
   cerr << "***got a hold of single instance BHGAny" << endl;

  attribute_value._type  = NULL;
  attribute_value._value = NULL;
  // (the default value contains the attribute type information)

  attribute_type = TypeCode_kind(default_value->_type,&ev);

  switch (attribute_type)
    {
    case tk_short:
      {
        short att_val;
        sscanf(any_value,"%d",&att_val);
        att_rc = any_obj->any_from_short(&ev,att_val,
                                         &attribute_value);
        break;
      }

    case tk_long:
      {
        long att_val;
        sscanf(any_value,"%d",&att_val);
        att_rc = any_obj->any_from_long(&ev,att_val,
                                        &attribute_value);
        break;
      }

    case tk_ushort:
      {
        unsigned short att_val;
        sscanf(any_value,"%d",&att_val);
        att_rc = any_obj->any_from_unsigned_short(&ev,att_val,
                                                  &attribute_value);
        break;
      }

    case tk_ulong:
    case tk_enum:        // enums can be set as unsigned longs
      {
        unsigned long att_val;
        sscanf(any_value,"%d",&att_val);
        att_rc = any_obj->any_from_unsigned_long(&ev,att_val,
```

```
                                               &attribute_value);
        break;
      }

    case tk_double:
      {
        double att_val;
        sscanf(any_value,"%f",&att_val);
        att_rc = any_obj->any_from_double(&ev,att_val,
                                          &attribute_value);
        break;
      }

    case tk_boolean:
      {
        boolean att_val;
        att_val = ((any_value[0] == 'T')  | (any_value[0]=='t'));
        att_rc = any_obj->any_from_boolean(&ev,att_val,
                                           &attribute_value);
        break;
      }

    case tk_string:
      {
        att_rc = any_obj->any_from_string(&ev,any_value,
                                          &attribute_value);
        break;
      }

    case tk_float:
      {
        float att_val;
        sscanf(any_value,"%f",&att_val);
        att_rc = any_obj->any_from_float(&ev,att_val,
                                         &attribute_value);
        break;
      }

    default:
      // the other type code kinds are not supported...
      cerr << "Unsupported Attribute Type" << endl;
      delete any_obj;
      return (7);
    }  // end of switch
  return att_rc;
}

/*
 * METHOD:      dumpAttributes
 * DESCRIPTION: Prints the attribute-name of an object
 */
int dumpAttributes(boolean bVal)
{
  _IDL_SEQUENCE_BHGCursor_AttributeDescription ad;
```

```
  BHGAny     *any_obj =
    (BHGAny*) _BHGAny->sommGetSingleInstance(&theEnv);

  any    av;
  long   rc;
  string output;

  theCursor->describeAttributes(&theEnv, &ad);

  if( !ad._length == 0 )
    {
      nice(depth);
      cout << "  |-- ATTRIBUTES" << endl;
      for (int loop=0; loop<ad._length; loop++)
        {
          nice(depth);
          cout << "  |-- " << ad._buffer[loop].name;

          if (bVal)    // dump Values
            {

              theCursor->getAttribute(&theEnv,
                                      ad._buffer[loop].name,
                                      &av);
              any_obj->string_from_any(&theEnv, &av, &output );

              cout << ": " << output ;
              BHGFree(av);
            }
          cout << endl;
        }
      nice(depth);
      cout << endl;
    }
  BHGFree(ad);
  return 0;
}

/*
 * METHOD:      dumpActions
 * DESCRIPTION: Prints the actions-name of an object
 */
int dumpActions()
{
  _IDL_SEQUENCE_BHGCursor_ActionDescription actd;
  theCursor->describeActions(&theEnv, &actd);
  if( !actd._length == 0 )
    {
      nice(depth);
      cout << "  |-- ACTIONS" << endl;
      for (int loop=0; loop<actd._length; loop++)
        {
          nice(depth);
```

```
          cout << "  |-- " << actd._buffer[loop].name << endl;
        }
      nice(depth);
      cout << endl;
    }
  BHGFree(actd);
  return 0;
}

/*
 * METHOD:      dumpRelations
 * DESCRIPTION: Prints the relations-name of an object
 */
int dumpRelations()
{
  _IDL_SEQUENCE_BHGCursor_RelationshipDescription reld;
  theCursor->getRelations(&theEnv, "",&reld);
  if( !reld._length == 0 )
    {
      nice(depth);
      cout << "  |-- RELATIONS" << endl;
      for (int loop=0; loop<reld._length; loop++)
        {
          nice(depth);
          cout << "  |-- " << reld._buffer[loop].rel_name
               << " -- Class:"
               << reld._buffer[loop].class_name
               << endl;
        }
      nice(depth);
      cout << endl;
    }
  BHGFree(reld);
  return 0;
}

/*
 * METHOD:
 * DESCRIPTION: Prints the name and classname of an object
 */
void dump(boolean bAttr,
          boolean bVal,
          boolean bOper,
          boolean bRel)
{
//    nice(depth);
//    cout << endl;

  /* Nur bis zur Tiefe der Ressource Definition (7) */
  /* Auch einzelne Ressource Definition (8) */
  if (depth < DEPTH)
    {
      nice(depth);
      cout << "--- " << theCursor->getName(&theEnv);
```

```
        cout << " -- ClassName: "
             << theCursor->getClassName(&theEnv)
             << endl;
        if (bAttr) dumpAttributes(bVal);
        if (bOper) dumpActions();
        if (bRel) dumpRelations();
      }
  }

  /*
   * METHOD:      dumpContents
   * DESCRIPTION: Reads the contents of an object
   */
  _IDL_SEQUENCE_BHGCursor_ObjectDescription
  dumpContents(boolean bCont)
  {
    _IDL_SEQUENCE_BHGCursor_ObjectDescription od;
    int rc, loop;

  //  cerr << "--- Entering dumpContents() " << endl;
    rc =  theCursor->getContents(&theEnv,"",&od);
    if (rc != BHGErr_Success)
      {
        cerr << "!!! ERROR getContents !!!" << endl;
        BHGErr_ErrorInfo err_info;
        err_info = getLastError();
        cerr << err_info.error_text << endl;
        od._length = 0;
      }
    if (bCont)
      {
        if( !od._length == 0 )
          {
            for (loop = 0; loop < od._length; loop++)
              {
                nice(depth);
                cout << "| | |-- "
                     << loop
                     << ": "
                     << od._buffer[loop].name
                     << endl;
              }
            nice(depth);
            cout << "| |" << endl;
          }
      }
    return od;
  }
};

/*
 * FUNCTION:     allAboutAnObject
 * DESCRIPTION:  Prints many information about an object
 *               and navigates through the object-tree
```

```
 */
int allAboutAnObject(Environment *global_ev,
                     CDMCursor cursor,
                     string nextObject,
                     boolean bAttr,
                     boolean bVal,
                     boolean bOper,
                     boolean bRel,
                     boolean bCont)
{
  int rc, objectCounter;
  _IDL_SEQUENCE_BHGCursor_ObjectDescription od;

  if(nextObject)
    cursor.navigate(nextObject);
//  cerr << "*** Entering allAboutAnObject" << endl;

//  cerr << "=========================================" << endl;
  cursor.dump( bAttr, bVal, bOper, bRel);
//  cerr << "-----------------------------------------" << endl;

  od = cursor.dumpContents(bCont);

  for(int i=0; i<od._length; i++)
    {
      allAboutAnObject(global_ev,
                       cursor,
                       od._buffer[i].name,
                       bAttr,bVal,bOper,
                       bRel,bCont);
    }
  return 1;
}

int main(int argc, char * argv[])
  {
    boolean bAttr, bVal, bOper, bRel, bCont;

    /*
     * ------------- Application Initialization --------------
     */

    if (argc < 2)
    {
      cerr << "USAGE: ccsmread [-a|-v|-o|-r|-c] serveralias"
           << endl;
      cerr << "Required parameter is : "
           << endl;
      cerr << " 1. The alias of the server owning the object"
           << endl;
      cerr << "Optional parameters are : "
           << endl;
      cerr << "   -a    Print Attributes" << endl;
      cerr << "   -v    Print Values of Attributes"
```

```
          << endl;
    cerr << "   -o    Print Operations" << endl;
    cerr << "   -r    Print Relations" << endl;
    cerr << "   -c    Print Contents" << endl;
    return(1);
  }

  bAttr = FALSE;
  bVal  = FALSE;
  bOper = FALSE;
  bRel  = FALSE;
  bCont = FALSE;

  for (int i=1;i<argc;i++)
    {
      cerr << i << ": " << argv[i] << endl;
      (strcmp("-a",argv[i]) == 0) ? bAttr = TRUE : bAttr = bAttr;
      (strcmp("-v",argv[i]) == 0) ? bVal  = TRUE : bVal  = bVal  ;
      (strcmp("-o",argv[i]) == 0) ? bOper = TRUE : bOper = bOper;
      (strcmp("-r",argv[i]) == 0) ? bRel  = TRUE : bRel  = bRel;
      (strcmp("-c",argv[i]) == 0) ? bCont = TRUE : bCont = bCont;
    }


  /*
   * ------------- API Initialization ------------------------
   */

  // Declare an environment structure to be used by SOM
  Environment ev;

  int loop;
  cerr << " Declared an environment structure to be used by SOM"
       << endl;

  // Initialize the SOM and DSOM environment
  SOM_InitEnvironment(&ev);
  SOMD_Init(&ev);

  cerr << "*** Initialized the SOM and DSOM environment" << endl;

  // Initialization of DSOM may set an exception
  // in the environment
  if (ev._major != NO_EXCEPTION)
  {
    cerr << "*** EXCEPTION !!!" << endl;
    cerr << ev._major << endl;
    cerr << somExceptionId(&ev);
    cerr << endl;

    // Can't start the program -
    // perhaps the SOM daemon is not running?
    somdExceptionFree(&ev);
    cerr <<
```

```
    "DSOM initialization failed -
     perhaps the SOM daemon is not started?";
  cerr << endl;
  return (2);
}

cerr << "*** About to Initialize the API classes" << endl;

// Initialize the API classes
BHGCursorNewClass(0,0);
BHGFreeMemNewClass(0,0);
BHGAnyNewClass(0,0);

cerr << "*** Initialized the API classes "
     << endl;

// BHGFreeMem and BHGAny are "single instance" classes
// - lets get a hold of their single instances now
BHGFreeMem *mem_free_obj =
  (BHGFreeMem*) _BHGFreeMem->sommGetSingleInstance(&ev);
cerr << "***got a hold of single instance BHGAny" << endl;

// BHGAny     *any_obj =
//    (BHGAny*) _BHGAny->sommGetSingleInstance(&ev);

// cerr << "***got a hold of single instance BHGAny" << endl;

string temp;
temp = _BHGCursor->somGetName() ;
cerr << temp << endl;
SOMFree (temp);

/*
 * ------------------ Connection to Server -----------------
 */

// The next thing I have to do is establish a connection
//  to an SM Application or Resource Controller

long rc;
CDMCursor *my_cursor;

cerr << "*** Connecting to Server:  " ;
cerr << argv[argc-1] << endl;
//cerr << argv[1] << endl;

rc = _BHGCursor->connect(&ev,argv[argc-1]);
//rc = _BHGCursor->connect(&ev,argv[1]);

// did it work?
if (rc != BHGErr_Success)
{
  cerr << "*** ERROR occured!" << endl;
  BHGErr_ErrorInfo err_info;
```

```
      err_info = _BHGCursor->getLastError(&ev);
      cerr << err_info.error_text << endl;

      delete mem_free_obj;

      // Uninitialize SOM environment
      SOMD_Uninit(&ev);
      SOM_UninitEnvironment(&ev);
      return(3);
    }
    else
    {
      // it worked!
      // create a cursor located at the root of the network
      // within component

      cerr << "*** Connected !!!" << endl ;

      my_cursor = new CDMCursor(ev);

      if (my_cursor) cerr << "*** New Cursor!" << endl;
      rc = my_cursor->theCursor->setAtLocation(&ev,argv[argc-1]);
      //rc = my_cursor->theCursor->setAtLocation(&ev,argv[1]);

      allAboutAnObject(&ev, *my_cursor,
                       NULL,bAttr,bVal,bOper,bRel,bCont);
      cerr << "*** Exited allAboutAnObject!" << endl;

      BHGErr_ErrorInfo err_info;
      err_info = my_cursor->getLastError();
      cerr << err_info.error_text << endl;

      delete mem_free_obj;
     // delete any_obj;
      delete my_cursor;

      // Uninitialize SOM environment
      SOMD_Uninit(&ev);
      SOM_UninitEnvironment(&ev);

      SOMFree (&ev);

    } /* else */


    /*
     * ----------------------- Termination --------------
     */

    // Free up memory used by all objects
    delete my_cursor;
    delete mem_free_obj;
    //delete any_obj;
```

```
    // Uninitialize SOM environment
    SOMD_Uninit(&ev);
    SOM_UninitEnvironment(&ev);

    return 0;
  } // end of main program...


/*
 * END OF CODE ---------------------------------------------
 */
```

B.2 ir2classes.pl

```
#!/usr/local/bin/perl

$baseflag = 0;

while (<>)
{
    chop;
    /^([^ ]*) *([^ ]*) *([^ ]*) *([^ ]*)/;
    $w1 = $1;
    $w2 = $2;
    $w3 = $3;
    $w4 = $4;

#    print"$w1 : $w2 : $w3 : $w4\n";
#    print"$baseflag\n";
    if ( $w3 eq "InterfaceDef")
    {
        /["](\w+)["]/;
        $w4 = $1;
        print" $w4 ";
        $baseflag = 0;
    }

    if ($baseflag == 1)
    {
        /::(\w+)/;
        $baseclass = $1;
        print "$baseclass ";
    }

    if (($w3 eq "base"))
    {
        $baseflag = 1;
  #  print"=$baseflag";
    }

    if(($w2 eq "instanceData:") && ($baseflag == 1))
    {
        $baseflag = 0;
        print"\n";
    }

} # while (<>)
```

B.3 dict2xh.pl

```
#!/usr/local/bin/perl

while (<>) {
    /^([^ ]*) *(.*)$/;
    $classname = $1;
    $username = $2;
    STDERR << $classname ;
    if ((/CLASS/) || (/----/))
    {
        print"";
    }
    else
    {
        @attribs = `bhgqdict -c "$classname"
                             -a /usr/lpp/cicssm/data/bhgsm.dict`;

        print "\n\n/***************************************/\n";
        print "\n class $classname\n { \t\t /* $username */ \n";
        for (@attribs)
        {
            /^([^ ]*) *([^ ]*) *(.*)$/;
            $attrib = $1;
            $attr_type = $2;
            $username = $3;

            if ((/ATTRIBUTE/) || (/------/))
            {
                print"\t\t\t /* Attributes */\n";
            }
            else
            {
                print"     $attr_type \t\t $attrib;\t/* $username */\n";
            }
        }

    @allmethods = `bhgqdict -c "$classname"
                            -m /usr/lpp/cicssm/data/bhgsm.dict`;
    for (@allmethods)
    {
        if ((/METHOD/) || (/-----------/))
        {
            print"\t\t /* METHODS */\n";
        }
        else
        {
            if (/(.*)/)
            {
                /^([^ ]*) *(.*)$/;
                $methodname = $1;
                $username = $2;
```

```
            @methodparm = ‘bhgqdict
                           -c "$classname"
                           -m "$methodname"
                           -p /usr/lpp/cicssm/data/bhgsm.dict‘;
            print"\t\t\t /* METHOD: $username */\n";
            print "     void\t $methodname(";
            $kommaflag = 0;

            for (@methodparm)
            {
                if ((/PARAM/) || (/----/))
                {
                    print"";
                }
                else
                {
                    /^([^ ]*) *([^ ]*) *([^ ]*) *(.*)$/;
                    $parmname = $1;
                    $parmtype = $2;
                    $direction = $3;
                    $parmdescript = $4;
                    if ($kommaflag == 1){
                        print ", "; }
                    print"\n\t\t $parmtype\t $parmname\t
                                          /* $parmdescript */";
                    $kommaflag = 1;
                }
            }              # for (@methodparm)
        }              # if (/(.*)/)
        print");\n";
    }              #  else
}              #  for (@allmethods)

@relations = ‘bhgqdict -c "$classname"
                       -r /usr/lpp/cicssm/data/bhgsm.dict‘;
for (@relations)
{
    if ((/RELATIONSHIP/) || (/-----------/))
    {
        print"\t\t\t /* RELATIONSHIPS */\n";
    }
    else
    {
        if (/(.*)/)
        {
            /^([+|-])([^ ]*) *([^ ]*) *(.*)$/;
            $plusminus = $1;
            $relname = $2;
            $relclass = $3;
            $reldesc = $4;

            print"\t\t\t /* $reldesc*/\n";
            print" $relclass *\t $relname; \t /*
                   $reldesc */\n";
```

```
            }
        }                       # else
    }                     # for (@relations)


    print "\n }; /* end of class $classname */ \n";
#    print "\n/******************************************/\n";
    print "";
    }
}
```

Anhang C

Data Dictionary

C.1 Data Dictionary

CDM_AIX_Domain: Repräsentiert die Domäne der CICS/6000-Systeme mit ihren Anwendungen und ihrer Topologie. Bei Systemen, die mit DCE-Unterstützung definiert sind, könnte hier auch die DCE-Zelle die Funktion der Domäne übernehmen.

CDM_AIX_Region: Spezialisierung eines CICS-Systems, da auch ein CICS-Client ein CICS-System mit eingeschränkten Möglichkeiten ist.

CDM_AIX_DomainTransaction: Enthält Informationen, wo eine bestimmte Transaktion innerhalb der Domäne existiert, sowohl lokal als auch remote eingetragen.

CDM_AIX_DomainProgram: Enthält Informationen, wo ein bestimmtes Programm innerhalb der Domäne existiert, sowohl lokal als auch remote eingetragen.

CDM_AIX_DomainFile: Enthält Informationen, wo diese Datei innerhalb der Domäne existiert, sowohl lokal als auch remote eingetragen.

CDM_AIX_DomainUser: Enthält Informationen, wo ein bestimmter Anwender innerhalb der Domäne existiert. Zusätzliche Information, in welchen Regions und für welche Anwendungen der Anwender eingetragen ist.

CDM_AIX_DomainMap: Enthält Informationen, wo eine bestimmte Bildschirmmaske innerhalb der Domäne existiert.

CDM_AIX_DomainTdq: Enthält Informationen, wo eine bestimmte Daten-Queue innerhalb der Domäne existiert.

CDM_AIX_DomainApplication: Enthält Informationen, auf welchen Systemen innerhalb der Domäne eine Anwendung definiert ist.

CDM_AIX_DomainTopology: In der Domain-Topology sind die Verbinungen der CICS-Systeme untereinander, damit verbundene Routeninformation und der Zustand von Verbindungen und Systemen gespeichert.

CDM_AIX_System: CICS-Client oder CICS-Region

CDM_AIX_Client: Common Client Software

CDM_AIX_Listener: Kommunikationsschnittstelle, für SNA oder TCP/IP spezialisiert

CDM_AIX_Connection: Ressource Definition einer Connection in einer Region

CDM_AIX_Session: Ressource Definition einer Session in einer Region

CDM_AIX_Application: Anwendung mit dazugehörigen Transaktionen, Programmen, Dateien und Informationen über die AOR(s) in der sie akiv ist.

CDM_AIX_Group: CICS-Gruppe, nach der Ressourcen in CICS-Systemen eingeteilt sind.

CDM_AIX_Terminal: Angemeldetes Terminal, über das eine Anwendung oder ein Anwender mit er Region kommuniziert.

CDM_AIX_Task: Laufende Task eines Programmes oder einer Transaktion.

CDM_AIX_LUW: Logische Verarbeitungseinheit, die durch einen Syncpoint abgeschlossen wird.

CDM_AIX_DomainUserGroup: Benutzergruppe

CDM_AIX_SecuritySystem: Externe Sicherheitssysteme, die Zugriffe auf Resssourcen regeln. Hier nur als Schnittstelle für weitere Arbeiten!

CDM_AIX_CommunicationFacility: Kommunikationseinrichtung für ISC und MRO.

CDM_AIX_Gateway: Ressource Definition eines Gateway in einer Region.

CDM_AIX_PPCGateway: Encina PPC Gateway Server

CDM_AIX_SystemCommunicationRessource: Kommunikationskomponenten, wie Netzkarten, Protokollmaschinen etc.

CDM_AIX_TAprotectedFilesystem: Transaktionsgeschützte Dateisysteme, wie Encina SFS Server, DB2 oder Btrieve.

CDM_AIX_Interface: Kommunikations-Schnittstellen aller Art.

Abbildungsverzeichnis

Literaturverzeichnis

[ADM AIX] IBM Corp., *CICS for AIX R1 Administration Guide*, IBM Corp., 1993, 1995.

[CPI-C] X/Open Company, *The CPI-C Specification, Version 2*, X/Open Company Limited, Dezember 1995.

[CYPS92] R. J. Cypser, *Communications for Cooperating Systems: OSI, SNA, and TCP/IP*, Addison-Wesley, 1992.

[DENN 93] Norbert Denne, Heinz Viete, *CICS - Theorie und Praxis*, DGD-Dienstleistungsgesellschaft für Datenverarbeitung mbH, 1993.

[HEAB 93] Heinz-Gerd Hegering und Sebastian Abeck, *Integriertes Netz- und Systemmanagement*, Addison-Wesley, 1993.

[IBM SM] IBM Corp., *IBM CICS Systems Manager Programming Guide*, IBM Corp., 1996.

[ICG AIX] IBM Corp., *CICS on Open Systems R1 3HI10 Intercommunication Guide*, IBM Corp., 1993, 1995.

[ICG ESA] IBM Corp., *CICS/ESA V4R1 Intercommunication Guide*, IBM Corp., 1977, 1994.

[ICG OS/2] IBM Corp., *CICS for OS/2 Version 3.0 Intercommunication*, IBM Corp., 1989, 1995.

[KREG 93] Thomas Kregeloh, Stefan Schönleber, *CICS: eine praxisorientierte Einführung*, Vieweg, 1993.

[OTS94] Object Management Group, Inc., *Object Transaction Service*, Object Management Group, 1994.

[SEGN 95] Peter Segner, *Ein betreibergerechtes View-Konzept für das Netz-und Systemmanagement*, Dissertation, Technische Universität München, Juni 1995.

[TX95] X/Open Company, *The TX (Transaction Demarcation) Specification*, X/Open Company Limited, April 1995.

[TxRPC] X/Open Company, *The TxRPC Specification*, X/Open Company Limited, November 1995.

[VOSS] Gottfried Vossen, Margret Groß-Hardt, *Grundlagen der Transaktionsverarbeitung*, Addison-Wesley, 1993.

[XA94] Vilhelm Rosenquist, „XA + Integration Interface“, In *XROLTPF02 1994 TRANSACTION PROCESSING*, X/Open Company Limited, September 1994, Status: APPROVED.

[XATMI] X/Open Company, *The XATMI Specification*, X/Open Company Limited, November 1995.

Index