Universität Leipzig

Fakultät für Mathematik und Informatik

(Institut für Informatik)

# UNTERSUCHUNGEN ZUM CICS TRANSACTION GATEWAY IN DER J2EE-UMGEBUNG

**Diplomarbeit**

vorgelegt von Gerrit Schlüter

Leipzig, November 2004

# INHALTSVERZEICHNIS

# ABBILDUNGSVERZEICHNIS

## TABELLENVERZEICHNIS

# DANKSAGUNGEN

Mein besonderer Dank gilt meinem Betreuer Herrn Dr. Herrmann, der mir die Anregung zu dieser Diplomarbeit gab und mich bei Problemen immer unterstützte.

Weiterhin möchte ich mich bei Herrn Georg Müller bedanken, der stets dafür sorgte, dass der z/OS-Rechner verfügbar war, und bei Problemen an diesem immer behilflich war.

Nicht zuletzt möchte ich mich bei meinen Eltern bedanken, die mir das Studium ermöglicht haben und mich auch während dieser Diplomarbeit immer unterstützten.

*Kapitel 1*

# Einleitung

## 1.1 Motivation

Die *z/OS*-Umgebung eignet sich hervorragend für moderne *E-Business*-Anwendungen. Die Ursache dafür liegt darin, dass bei einer typischen E-Business-Anwendung viele verschiedene Arten von Aufgaben erfüllt werden müssen – das Ausführen von Webanwendungen, das Verarbeiten von Transaktionen, die Datenbankverwaltung und so weiter. Wenn diese Anwendungen auf anderen als *zSeries*-Systemen ausgeführt werden, ist es in der Regel so, dass für die verschiedenen Aufgaben voneinander getrennt agierende Systeme eingesetzt werden. Jedoch sind deren Betriebssysteme sowie deren Hardware nicht darauf ausgerichtet, mehrere Arten von Aufgaben koordiniert auszuführen. z/OS ist wiederum dafür entwickelt worden, verschiedene Arten von Aufgaben intelligent zu bearbeiten.

Ein einzelner zSeries-Server kann in mehrere logische Partitionen mit eigenen Betriebssystemen eingeteilt werden, von denen jedes verschiedene Aufgaben gleichzeitig ausführen kann. Weiterhin können die Funktionen des *IBM Parallel Sysplex* ausgenutzt werden, durch den bis zu 32 zSeries-Server zu einem einzigen logischen System verschmelzen. Damit kann eine E-Business-Anwendung die Ressourcen eines einzelnen zSeries-Rechners oder einer Parallel Sysplex-Lösung ausnutzen. Dabei werden die einzelnen Aufgaben der Anwendung intelligent und dynamisch verteilt werden, so dass deren Echtzeitanforderungen unter z/OS erfüllt werden können.

Auf der anderen Seite muss heutzutage für erfolgreiche Anwendungen die Möglichkeit bestehen, Teil solcher E-Business-Lösungen zu werden. Auch für das IBM-Transaktionssystem *CICS* gibt es mittlerweile einige Lösungen für dieses Problem. Eine davon ist das *CICS Transaction Gateway*. Dabei kommt auch die *J2EE*-Plattform von *Sun Microsystems* zum Einsatz, deren Stärken vor allem in Plattformunabhängigkeit, Modularisierung und standardisierten Schnittstellen liegen.

## 1.2 Ziel der Arbeit

Das Ziel dieser Arbeit bestand darin, eine Konfiguration zu erstellen, in der CICS-Programme im Web verfügbar gemacht werden können. Zum Zugriff auf CICS sollte das CICS Transaction Gateway eingesetzt werden, das eine J2EE-Schnittstelle bereitstellt. Diese

kann wiederum vom *WebSphere Application Server* genutzt werden, der den Zugang zum Web liefert. All diese Softwarekomponenten befanden sich in einem einzigen z/OS-System. Zum Test dieser Konfiguration sollte eine J2EE-Anwendung aus dem IBM-Redbook *CICS Transaction Gateway V5: The WebSphere Connector for CICS* [CG02b] eingesetzt werden, das eine sehr gute Grundlage für das Verständnis des CICS Transaction Gateway bildet. Schließlich sollte die Konfiguration erweitert werden, um Zugriff auf *DB2*-Datenbanken herzustellen, was von CICS aus zu geschehen hat. Diese Erweiterung sollte ebenfalls mit Hilfe der J2EE-Beispielanwendung getestet werden.

*K a p i t e l 2*

# Verwendete Software

## 2.1 Das z/OS-Betriebssystem

z/OS ist ein 64-Bit-Betriebssystem für die zSeries-Rechner von IBM, die zu den Großrechnern oder *Mainframes* zu zählen sind. Dabei steht *z* für *Zero Down-Time*, was die hohe Fehlertoleranz verdeutlichen soll. Es entstand im Jahre 2001 durch die Umbenennung des Betriebssystems *OS/390*, das wiederum im Jahre 1995 kreiert wurde, indem das Vorgängerbetriebssystem *MVS (Multiple Virtual Storage)* um *UNIX System Services* erweitert wurde. Die Grundbestandteile von z/OS sind jedoch in ihrer Struktur noch vorhanden, so dass zum Beispiel das MVS-Betriebssystem noch immer die Basis von z/OS bildet. Dieses entstand wiederum im Jahre 1974 über mehrere Entwicklungsstufen hinweg aus dem Betriebssystem *OS/360* und wurde seitdem ständig weiterentwickelt. Abbildung 2-1 zeigt die Grundstruktur des z/OS-Betriebssystems.

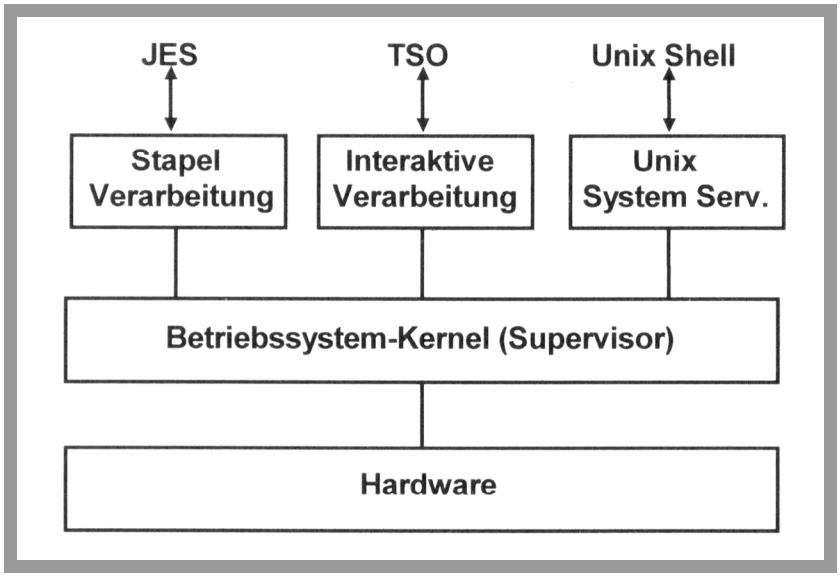


Abbildung 2-1: Grundstruktur des z/OS-Betriebssystems [HKS03]

### *2.1.1 MVS*

*Einführung*

MVS ist ein Betriebssystem, bei dem es nicht allzu sehr auf Geschwindigkeit ankommt. Stattdessen wird es eingesetzt, um große Datenmengen gleichmäßig und zuverlässig zu verarbeiten. Daher unterscheidet sich die Art der Leistungsfähigkeit eines Mainframes auch von der eines modernen Supercomputers. Diese können mit unglaublicher Geschwindigkeit

komplizierte Berechnungen durchführen, wobei dies oftmals bedeutet, dass dafür die *E/A*-Leistung (*Ein-/Ausgabe*) geopfert wird. Das Lesen und Schreiben von riesigen Datenmengen und das Ausführen relativ einfacher Berechnungen mit diesen Daten ist die Stärke von Mainframes mit dem Betriebssystem MVS. Zum Beispiel würden ein großes Unternehmen für die Gehaltsabrechnung ihrer Angestellten oder eine Warenhauskette zur Verwaltung des Inventars MVS einsetzen. Da es sich um ein Mehrbenutzerbetriebssystem handelt, können unter MVS viele Benutzer gleichzeitig mit denselben Programmen und Daten arbeiten.

Benutzer von *Personalcomputern (PCs)* machen sich oft über Mainframes mit dem MVS-Betriebssystem lustig, indem sie sie Dinosaurier nennen. Obwohl die Benutzeroberfläche primitiv erscheint, hatte MVS schon bei seiner Einführung 1974 viele Funktionen, die erst langsam in die Betriebssysteme von PCs integriert wurden oder werden. Zum Beispiel beinhaltet MVS Dienste zur Fehlerbehebung bei defekten Datenträgern oder sogar bei defekten Prozessoren in einem Mehrprozessorsystem. Die Sicherheit der Daten eines Benutzers vor anderen ist ein integraler Bestandteil des MVS-Betriebssystems.

*Stapelverarbeitung*

Der Name MVS – Multiple Virtual Storage – kommt von der eingesetzten Technik – der Speicherverwaltung. Ein Benutzer kann mit großen Speichermengen arbeiten, so dass sich MVS hervorragend für die Stapelverarbeitung eignet.

Ein Job in der Stapelverarbeitung ist das genaue Gegenteil von interaktiven Benutzerschnittstellen. Anstatt ein Programm zu starten, Eingaben zu tätigen und auf das Ergebnis zu warten, legt man die benötigten Bestandteile des Jobs schon zu Beginn fest. Man gibt dem System vor, welches Programm auszuführen ist, welche Daten zu verwenden sind und was mit den Ergebnissen zu geschehen hat. Dabei läuft ein Stapelverarbeitungsjob im Hintergrund ab, so dass man während dessen Ausführung andere interaktive Tätigkeiten durchführen kann.

Bei der Stapelverarbeitung sind die beiden Abkürzungen *JES2* und *JES3* von Bedeutung. Dabei handelt es sich um zwei Versionen des *Job Entry Subsystems*, dem Teil des MVS-Betriebssystems, der den zeitlichen Ablauf der Jobs steuert.

*JCL* ist die Sprache, die einem Mainframe-System mitteilt, wie mit einem Stapelverarbeitungsjob umzugehen ist. Mit Hilfe von JCL legt man die Größe, die Priorität, das Ausgabeziel, die Eingabedateien, die Drucker, die Prozessorzeit und den zu

verwendenden Plattenspeicher des Jobs fest. Auch in dieser Arbeit wird JCL für zahlreiche Aufgaben eingesetzt.

*Interaktive Schnittstellen*

MVS bietet einige Möglichkeiten zur Interaktion – darunter *TSO*, *ISPF* und *CICS*. Dabei wird CICS in *Abschnitt 2.2: CICS* näher beleuchtet.

*TSO* steht für *Time Sharing Option*. Es ist der Teil von MVS, mit dem das System interaktiv verwendet werden kann. Man kann Befehle eingeben und sieht das Ergebnis auf dem Bildschirm. TSO verfügt außerdem über Befehle zum Starten von Stapelverarbeitungsjobs sowie zum Überprüfen deren Fortschritts. TSO wurde 1969 zu einer Zeit eingeführt, als die interaktive Verarbeitung eine brandneue Funktion war, die einen Großteil des Speichers beanspruchte. Daher kommt auch das *O* im Namen: Am Anfang war TSO eine *optionale* Komponente, heute ist es Teil jedes MVS-Betriebssystems.

*ISPF* steht für *Interactive System Productive Facility*. Diese Alternative zu *TSO* ermöglicht das Ausführen der grundlegenden Funktionen, indem man in Menüs navigiert oder Formulare – so genannte *Panels* – ausfüllt. Da ISPF viele Dinge vereinfacht, hat es mit der Zeit mehr Popularität erlangt als TSO. Jedoch bietet TSO mehr Geschwindigkeit und Flexibilität, da das Eingeben eines Befehls mit allen Optionen schneller geht als das Abarbeiten einer Reihe von Menüs, in denen einige Optionen womöglich nicht verfügbar sind.

### *2.1.2 UNIX System Services*

UNIX System Services (USS) sind neben MVS eine weitere Komponente in den Betriebssystemen z/OS und OS/390. Es handelt sich um eine vollwertige *UNIX*-Implementierung (*XPG4 UNIX 95*), der ersten UNIX 95-Implementierung, die nicht dem *AT&T/SCO*-Quellcode entstammt. Durch die Integration in z/OS können neben UNIX-Diensten auch TSO-Befehle ausgeführt sowie UNIX-Dateien in ISPF bearbeitet werden. Durch JCL-Erweiterungen kann man diese Dateien auch in der Stapelverarbeitung verwenden.

Das wichtigste Anliegen von UNIX System Services besteht jedoch darin, dass UNIX-Anwendungen auf IBM-Mainframes ausgeführt werden können. Binäre Programme, die in ihrer Logik *ASCII*-Daten (*American Standard Code for Information Interchange*) verwenden, bilden dabei die einzige Inkompatibilität – die zSeries-Architektur setzt auf das Format *EBCDIC (Extended Binary Coded Decimal Interchange Code)* auf. USS verwenden als Dateisystem *HFS*

*(Hierarchical File System)*, was sich für den Rest von z/OS als eine spezielle Art von *Datasets* darstellt.

Der Vorläufer von USS heißt *OMVS (Open MVS)* und ist u.a. über ein ISPF-Menü zu erreichen. In dieser Arbeit wurde OMVS für einige Aufgaben eingesetzt.

## 2.2 CICS

CICS steht für *Customer Information Control System* und ist ein Transaktionssystem von IBM. Wenn ein Börsenmakler von seinem Computer aus Kauf- und Verkaufanweisungen ausführt, stehen die Chancen gut, dass am anderen Ende der Verbindung ein CICS-System läuft, da neun der zehn führenden Internetmaklerunternehmen CICS einsetzen [Ank01]. Die meisten großen Versicherungsunternehmen und Finanzinstitute verwenden ebenfalls CICS. Wenn man eine Chipkarte in einen Geldautomaten steckt, könnte am anderen Ende der Verbindung eine CICS-Transaktion gestartet werden.

Nach einer neueren IBM-Präsentation wurden pro Tag über 30 Milliarden CICS-Transaktionen durchgeführt [CI04]. Dies entspricht einem Geldwert von einer Billion Dollar. Insgesamt gibt es etwa 30.000 Unternehmen und 30 Millionen Benutzer, die CICS einsetzen. Es gibt derzeit etwa 950.000 Programmierer, die ihr Geld mit CICS verdienen [Ank01].

### *2.2.1 Historie*

CICS wurde 1968 unter dem Namen *Public Utility Customer Information Control* veröffentlicht. Es bestand zum damaligen Zeitpunkt nur aus etwa 19.000 Zeilen Code, im Moment schätzt man CICS auf zwölf bis 13 Millionen Zeilen. Kunden aus verschiedenen Industrien begannen, CICS zu benutzen, um Anwendungen zu entwickeln. Im Jahre 1969 entkoppelte IBM die Soft- von der Hardware, und CICS wurde eines der ersten kommerziellen Softwareprodukte.

Als CICS veröffentlicht wurde, waren Schreibmaschinenterminals die einzige Möglichkeit der Interaktion zwischen Menschen und Computern. Ein Mensch gab einen Befehl oder eine Anfrage ein, und CICS antwortete mit einer oder mehreren Zeilen auf der Schreibmaschine. Mit der Einführung der 3270-Bildschirme änderte sich dies.

Um den Kunden die Programmierung für die 3270-Bildschirme zu vereinfachen, entwickelte IBM eine Makrosprache, die die Definition von Feldern innerhalb des Bildschirms ermöglicht. Diese Makroanweisungen werden in ein Lademodul umgewandelt, das von

Anwendungsprogrammen verwendet wird. Dieses als *BMS (Basic Mapping Support)* bekannte Schema funktionierte zwar, erforderte jedoch meist eine ausgedehnte Fehlersuche, bis das gewünschte Ergebnis erzielt wurde. Nach 1980 wurde *SDF (Screen Definition Facility)* eingeführt. Dabei handelt es sich um ein eigenständiges Produkt, das Programmierern eine interaktive Erstellung der BMS-Bildschirme ermöglicht.

### *2.2.2 Web-Unterstützung*

Heutzutage könnte kein Programm überleben, ohne dass es eine Möglichkeit der Interaktion mit dem *World Wide Web* gibt. IBM veröffentlichte 1996 das *CICS Web Interface*. Seitdem gibt es mehrere Lösungen und Erweiterungen von CICS, um die Interaktion mit Webbrowsern zu ermöglichen, eine davon ist das *CICS Transaction Gateway*. Ältere CICS-Anwendungen können in Webbrowsern ausgeführt werden, als würde es sich um echte 3270-Terminals handeln. Neue Programme können mit modernen Webschnittstellen ausgestattet werden.

### *2.2.3 CICS und Datenbanken*

Zu Beginn seiner Geschichte besaß CICS einen eigenen, im Programm verankerten Datenbankmanager, der den heutigen relationalen Datenbanken ähnlich war. Diese Datenbankfunktionen wurden 1985 wieder aus dem Programm entfernt. Dabei handelte es sich eher um eine strategische Maßnahme. IBM vertrieb bereits zwei andere Datenbankprodukte: *Information Management System (IMS)* und *Database 2 (DB2)* und wollte daher CICS auf sein eigentliches Kerngebiet – das Transaktionsmanagement – beschränken.

IMS war teilweise ein Konkurrent zu CICS. Es besteht aus *IMS/DB* – einem hierarchischen Datenbankmanager – und *IMS/TS* – einem Transaktionsmanager. Die Legende besagt, dass das IMS-Team es ablehnte, mit dem CICS-Team eine Schnittstelle zwischen den beiden Systemen zu erarbeiten, da IMS schon einen Transaktionsmanager besaß. Daraufhin entwickelte das CICS-Team auf eigene Faust eine solche Schnittstelle, was zu vielen Problemen führte, da keine Seite die Verantwortung für mögliche Fehler übernehmen wollte. Über die Jahre hat sich dies jedoch gebessert, und das IMS- und das CICS-Team teilen sich die Verantwortung für die Schnittstelle.

1982 wurde die erste CICS-Version veröffentlicht, die IBMs relationalen Datenbankmanager DB2 unterstützte. Es schien, als hätte IBM aus den Problemen mit IMS gelernt. Im Gegensatz zu IMS ist DB2 ein reiner Datenbankmanager und daher abhängig von Transaktionsmanagern wie CICS oder IMS/TS.

Abbildung 2-2 zeigt die Komponenten von CICS unter z/OS und OS/390 sowie ihren Zusammenhang. Außerdem sind die Zugriffsmechanismen von CICS auf die verschiedenen Arten von Quelldaten dargestellt.

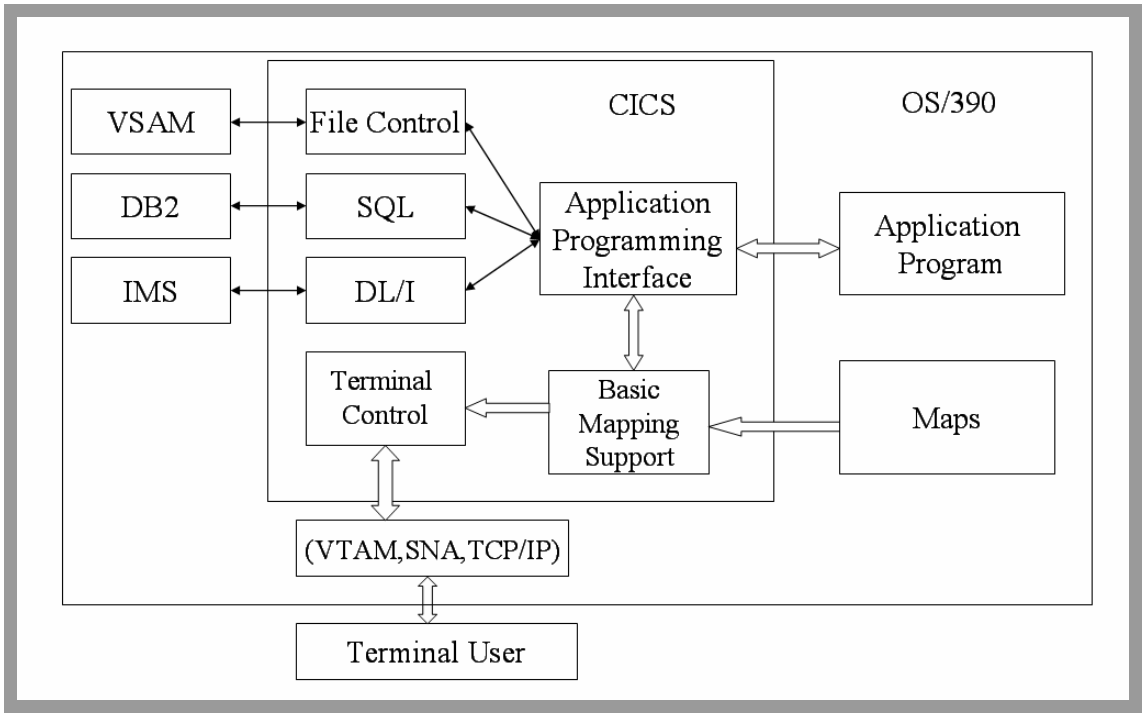


Abbildung 2-2: Der Aufbau von CICS unter z/OS und OS/390
[Hen01]

### *2.2.4 CICS auf verschiedenen Plattformen*

Als CICS 1968 vorgestellt wurde, war es nur für OS/360 verfügbar. In den späten 80er Jahren stand IBM an einem Wendepunkt. Die PCs gewannen immer mehr Marktanteile, und diese Entwicklung schien unaufhaltsam. Außerdem beschränkte sich die Verbreitung von UNIX nicht länger nur auf Universitäten. IBM musste sich entscheiden, ob die Entwicklung von CICS an einem logischen Ende angelangt war oder ob es mit großem Aufwand verjüngt werden sollte. IBM begegnete dieser Herausforderung, indem innerhalb relativ kurzer Zeit CICS-Versionen für mehrere verschiedene Plattformen entwickelt wurden. 1989 veröffentlichte IBM die erste PC-Version von CICS, *CICS OS/2*. Dieses Produkt wurde über mehrere Versionen fortgeführt, bis OS/2 von der Bildfläche verschwand. In der Zwischenzeit wurden CICS-Versionen für *PC-DOS*, *Microsoft Windows* und sogar *Apple Macintosh* entwickelt. IBM verkauft für diese Plattformen zwei Arten von CICS: den *CICS Client* und den *CICS Server*. Zusammen mit *Hewlett Packard* und *Sun* entwickelte IBM CICS-Versionen für die UNIX-Plattform, so dass es heutzutage nur sehr wenige Systeme gibt, für die CICS nicht verfügbar ist.

### *2.2.5 Unterstützte Programmiersprachen*

1968 mussten alle Anwendungsprogramme für CICS in *Assembler* geschrieben werden. 1970 wurde die Unterstützung von *COBOL* und *PL/I* hinzugefügt, da es sich um die damals gebräuchlichsten Sprachen handelte. Einige Programmiersprachen wie *RPG II* und *Ada* wurden vorübergehend unterstützt und dann wieder entfernt. CICS unterstützt seit 1994 ebenfalls die Sprache *Rexx*, die auf praktisch jeder IBM-Plattform vorhanden ist. 1997 lief diese Unterstützung aus, um 1999 wieder eingeführt zu werden.

Entscheidend für die Zukunftsfähigkeit und Popularität von CICS ist die Unterstützung von Standardprogrammiersprachen sowie objektorientierten Sprachen. 1990 erschien die erste CICS-Version mit einer Unterstützung für die Programmiersprache *C*. IBM erstellte eine gemeinsame Laufzeitbibliothek für COBOL, PL/I und C. Die erste von CICS unterstützte objektorientierte Programmiersprache war *C++* im Jahre 1995. Es handelte sich auch um die erste Sprache, die eine eigene Programmierschnittstelle zu CICS erhielt. Anstatt mit Aufrufen wie **EXEC CICS** zu arbeiten, rufen C++-Programme die CICS-Dienste mit Hilfe von objektorientierten Klassen auf. 1997 wurde die Unterstützung von *Java* hinzugefügt, wobei die CICS-Funktionen ebenfalls durch Klassenmethoden aufgerufen werden. Seit 2000 werden auch *Enterprise Java Beans (EJBs)* vollständig unterstützt. Somit wurde CICS zu einem vollwertigen Mitspieler in der objektorientierten und Internetwelt.

## 2.3 DB2

### *2.3.1 Historie*

1970 veröffentlichte E.F. Codd von der IBM-Forschungsabteilung ein Papier, das zu einer völlig neuen Art der Informationsverwaltung auf Computern führte. Es hieß *A Relational Model of Data for Large Shared Data Banks* und schlug ein relationales Datenmodell vor, das Anwendungsentwickler davon befreite, sich um die Struktur der zu verwendenden Daten kümmern zu müssen.

Vier Jahre später veröffentlichten die IBM-Mitarbeiter Don Chamberlin und Ray Boyce *SEQUEL: A Structured English Query Language*, die Basis für die heutige Standardsprache *SQL*. Das Stellen von Anfragen in dieser neuen Sprache hatte nun eine größere Bedeutung als die Art der Speicherung der Daten auf der Platte. Komplexere Anfragen konnten effizienter gestellt und beantwortet und Anwendungen schneller erstellt werden. Das relationale Datenbanksystem selbst kümmerte sich um die Verwaltung der Daten, so dass sich die Anwendungen auf die Businesslogik konzentrieren konnten.

Seit 1970 entwickelt IBM eine umfangreiche Familie von relationalen *Datenbankmanagementsystemen (RDBMS)*, die mittlerweile den Namen *DB2 Universal Database (DB2 UDB)* tragen. Eine Reihe von Forschungsprojekten trugen von Beginn an zur Entwicklung der DB2-Familie bei. Das Projekt *System R* führte zur ersten IBM-Implementierung des relationalen Datenmodells. Ein Projekt namens *ARIES* lieferte zeilenbasierte *Locking*-Mechanismen, die heutzutage in der Datenbankindustrie weit verbreitet sind. Das Projekt *R Star* erweiterte das relationale Modell auf verteilte Systemumgebungen. Weitere Projekte zu Optimierungsstrategien und zur Datenintegration zwischen verschiedenartigen Systemen folgten. Einen neuen Ansatz verfolgte eine aktuelle Testversion von DB2. Es wurde die Integration von Informationen aus *Web Services* und die Verwendung von *XQuery* als zusätzliche mächtige Abfragesprache zur Verwaltung von *XML*-Inhalten *(Extensible Markup Language)* demonstriert.

Die erste Implementierung der relationalen Technologien aus dem ursprünglichen Projekt *System R* war 1982 die in den *System/38*-Server integrierte Datenbank. 1982 wurde das ebenfalls auf *System R* basierende Produkt *SQL/DS* für die Mainframe-Betriebssysteme *VM* und *VSE* ausgeliefert. DB2, zuvor *DATABASE 2*, wurde erstmals 1983 für das Betriebssystem *MVS* veröffentlicht. Der Datenbankmanager von *OS/2 Extended Edition* war im Jahre 1987 die erste relationale Datenbank für verteilte Systeme. *SQL/400* für den neuen *AS/400*-Server kam 1988 auf den Markt. Neuere DB2-Versionen wurden für *AIX* (1993), *HP-UX* und *Solaris* (1994), Windows (1995) und *Linux* (1999) entwickelt. Heutzutage ist DB2 also auf einer Vielzahl an Systemen mit unterschiedlicher Hard- und Software zu finden, sogar auf den kleinsten: *DB2 Everyplace* unterstützt Handheld-Geräte und eingebettete Linux-Umgebungen und stellt Möglichkeiten der Datensynchronisation mit größeren Systemen zur Verfügung.

### *2.3.2 DB2 unter z/OS und OS/390*

Unter den Betriebssystemen OS/390 und z/OS wird DB2 in Verbindung mit anderen Erweiterungen des Betriebssystems und der Serverhardware entwickelt. Diese enge Kopplung führte zum so genannten *DB2 Data Sharing*, einer *Clustering*-Architektur, die die Parallel Sysplex-Eigenschaften der IBM System/390- und zSeries-Hardware ausnutzt. Einige der größten Datenbanken der Welt basieren auf DB2 in dieser Umgebung, wie eine regelmäßige Studie großer Datenbanken der *Winter Corporation* zeigt [Win03].

### *2.3.3 Anwendungsentwicklung für DB2*

Als Anwendungsentwickler hat man eine Reihe von Möglichkeiten zur Einbindung von DB2 als Datenbankserver. Das DB2-Team hat zusammen mit dem IBM WebSphere- und dem *IBM Rational XDE*-Team, mit der *Eclipse*-Gemeinde sowie mit der *Microsoft Visual Studio*-Gruppe Plugins zur Entwicklung von DB2-Anwendungen und zur visuellen Datenmodellierung erstellt. DB2 UDB umfasst mittlerweile eine eigene Entwicklungsumgebung *(DB2 Development Center)*, in der serverseitige Anwendungsteile wie gespeicherte Prozeduren *(Stored Procedures)* und benutzerdefinierte Funktionen mit Hilfe der Sprachen SQL und Java erstellt werden können.

Die Anfänge der Zusammenarbeit zwischen DB2 und Java liegen schon ein paar Jahre zurück – die Java-Unterstützung wurde erstmals 1996 in DB2 integriert. Ab diesem Zeitpunkt konnten nun gespeicherte Prozeduren und benutzerdefinierte Funktionen auch in Java erstellt werden. *JDBC (Java Database Connectivity)* ist ein Kommunikationsmechanismus zwischen Java-Anwendungen und Datenbanksystemen und wurde ebenfalls zu dieser Zeit eingeführt. Seitdem ist die Java-Unterstützung in DB2 um *SQLJ* – einer statischen Kommunikation mit DB2 – und *JOLAP (Java Online Analytical Processing)* – einem Java-basierten Standard zur Datenanalyse – angewachsen. Schließlich unterstützt DB2 auch die Anwendungsumgebung J2EE.

## 2.4 Die J2EE-Plattform

Die Plattform *Java 2 Enterprise Edition (J2EE)* definiert einen Standard zur Entwicklung mehrschichtiger Unternehmensanwendungen. Sie stellt standardisierte modularisierte Komponenten sowie zugehörige Dienste bereit, so dass viele Anwendungsdetails automatisch und ohne komplexe Programmierarbeit behandelt werden.

Die J2EE-Plattform baut auf *Java 2 Standard Edition (J2SE)* auf und nutzt zum Beispiel ebenfalls die JDBC-Bibliothek zum Datenbankzugriff, die Technologie *CORBA (Common Object Request Broker Architecture)* zur Interaktion mit vorhandenen Ressourcen sowie ein Sicherheitsmodell zum Schutz von Anwendungsdaten. Zusätzlich zu diesen grundlegenden Funktionen unterstützt die J2EE-Plattform *Enterprise JavaBeans (EJBs)*, *Java Servlets*, *Java Server Pages (JSPs)* und die XML-Technologie. Weiterhin stellt die J2EE-Spezifikation mit Hilfe des *Basic Profile* der Firma *WS-I* sicher, dass Web Services untereinander kompatibel sind.

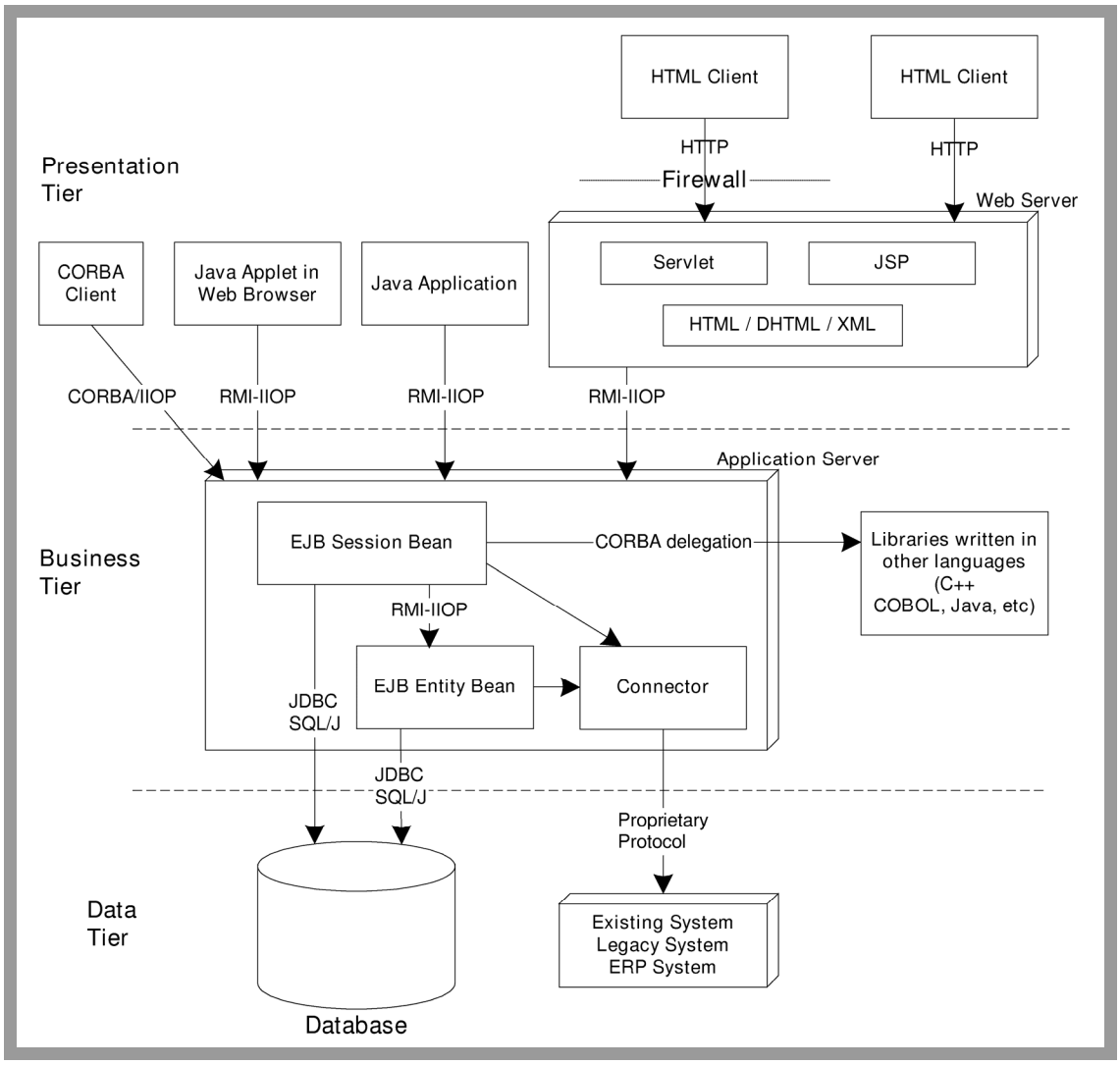


Abbildung 2-3: Das J2EE-Modell [RÖ99]

Ein Vorteil der J2EE-Architektur besteht darin, dass das Anwendungsmodell die verschiedenen Funktionalitäten in voneinander getrennten Komponenten zur Verfügung stellt (Abbildung 2-3). Die Businesslogik ist in der Regel in Enterprise JavaBeans enthalten, während Webseiten aus *Applets*, Java Servlets und Java Server Pages bestehen. Die Komponenten kommunizieren untereinander mit Hilfe verschiedener Standards: XML, *HTML*, *HTTP*, *SSL*, *RMI*, *IIOP* u.a. Der Entwickler kann nun aus einer Kombination von standardmäßigen, kommerziellen und selbst entwickelten Komponenten seine eigenen Anwendungen zusammenstellen. Weiterhin kann jede der Komponenten auf eine beliebige Datenbasis zugreifen.

### *2.4.1 Die Bausteine der J2EE-Plattform*

Das J2EE-Modell unterteilt Unternehmensanwendungen in drei grundlegende Bausteine:

- *Komponenten*,
- *Container* und
- *Konnektoren*.

Container vermitteln zwischen Clients und Komponenten und stellen beiden transparent Dienste zur Verfügung. Durch dieses Modell muss das Verhalten vieler Komponenten nicht im Programmcode bestimmt werden, sondern kann auch noch bei der Installation der Anwendung festgelegt werden. Konnektoren stehen etwas außerhalb der J2EE-Plattform, da sie eine portable Schnittstelle zu schon vorhandenen Anwendungsprogrammen herstellen. Auch das CICS Transaction Gateway beinhaltet einen Konnektor und wird in *Abschnitt 2.7: CICS Transaction Gateway* genauer betrachtet.

### *2.4.2 Container*

Die J2EE-Architektur besteht aus drei Containern:

- dem *Clientcontainer*,
- dem *Webcontainer* und
- dem *EJB-Container*.

Der Clientcontainer umfasst Java-Applets, Java-Anwendungen oder einen *HTML-Browser*. Der Webcontainer ist die Laufzeitumgebung für Servlets und JSPs, mit denen dynamische Webinhalte in Form von HTML-Seiten bereitgestellt werden können. Er interpretiert Anfragen vom Client und leitet diese an eine Webkomponente weiter. Diese erstellt das gewünschte Ergebnis und liefert dieses an den Webcontainer zurück, der es an den Client weiterleitet. Der EJB-Container stellt die Laufzeitumgebung für Enterprise JavaBeans dar. Diese liegen als *Java-Bytecode* vor und können nicht unmittelbar ausgeführt werden. Daher muss sich der EJB-Container um die Interpretation und Ausführung dieses Bytecodes kümmern.

### *2.4.3 Komponenten*

*Servlets*

Ein Servlet ist eine serverseitige Java-Komponente, die im so genannten *Servlet Container* abläuft, einem Teil des Webcontainers. Das Servlet erhält eine Anfrage vom Servlet

Container und leitet die Antwort an diesen weiter, der sie wiederum an den Client weiterreicht. Zwar können Servlets prinzipiell alle Anfrage-Antwort-Protokolle verarbeiten, in der Praxis werden sie jedoch in der Regel dazu verwendet, Webanwendungen um zusätzliche Funktionalitäten zu erweitern. Dabei verarbeiten die Servlets HTTP-Anfragen von Benutzern und liefern als Antwort ein Dokument im HTML-Format (Abbildung 2-4).

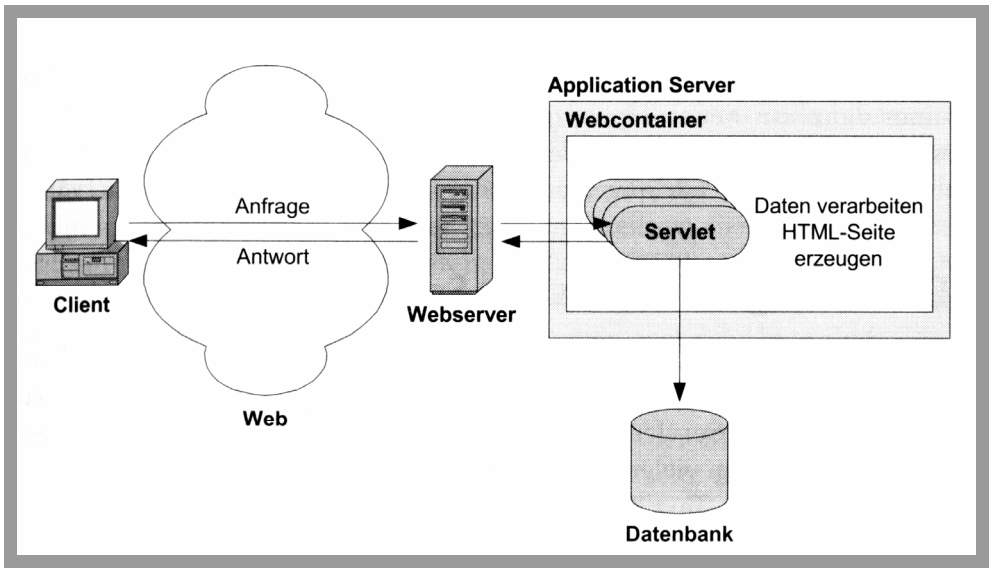


Abbildung 2-4: Servlets in Webanwendungen [SS02]

*Java Server Pages (JSPs)*

Eine Java Server Page (JSP) ist eine Textdatei auf HTML-Basis (oder XML-Basis), die Java-Quelltext und *benutzerdefinierte Tags* enthalten kann. Durch diese Erweiterungen erhält die HTML-Datei einen dynamischen Charakter. Aus dem Quelltext der JSP wird automatisch ein Servlet generiert, das bei jeder Anfrage dynamisch eine entsprechende Antwort erstellt. JSPs laufen im *JSP-Container* ab, einem weiteren Teil des Webcontainers. JSPs sind eine Erweiterung der Servletbibliothek und daher keine alternative Technologie. So wird in Wirklichkeit jede Anfrage an den JSP-Container mit einem bestimmten URL-Muster (z.B. `*.jsp`) an ein Servlet *(JSP-Servlet)* weitergeleitet. Dieses überprüft bei jeder neuen Anfrage, ob die angeforderte JSP-Datei in der aktuellen Version schon übersetzt wurde. Ist dies nicht der Fall, so wird ein neues Servlet erstellt, ansonsten wird die Anfrage an das vorhandene weitergeleitet (Abbildung 2-5).

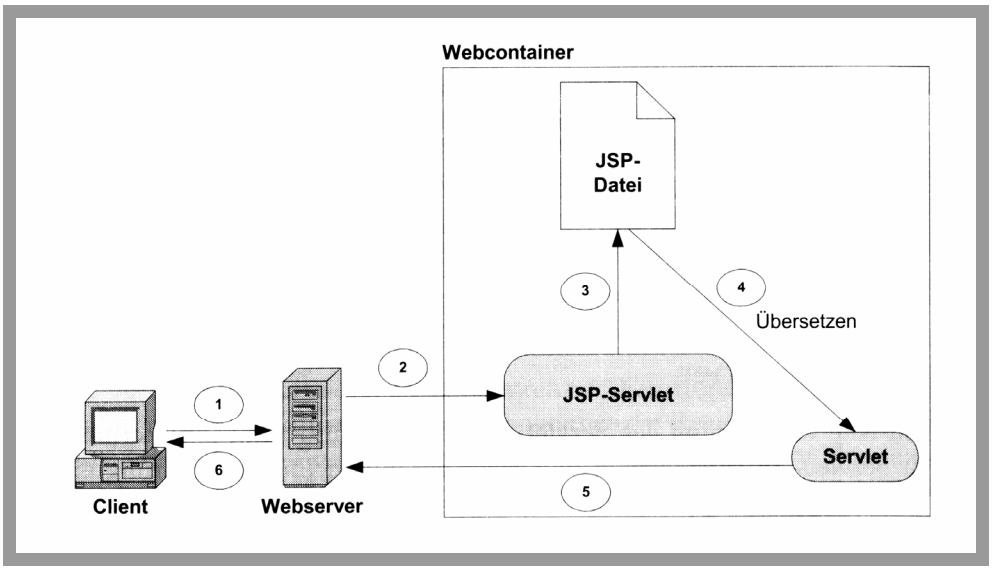


Abbildung 2-5: Ablauf einer Anfrage an eine JSP [SS02]

*Enterprise JavaBeans (EJBs)*

Auch bei Enterprise JavaBeans (EJBs) handelt es sich um serverseitige Komponenten. In ihnen ist die Logik der Anwendung implementiert, welche dann von Clientprogrammen verwendet werden kann. Die Laufzeitumgebung für EJBs ist der EJB-Container, der Dienste wie Persistenz, Transaktionen und Sicherheit bereitstellt. Weiterhin erledigt er Aufgaben wie Nebenläufigkeit von Programmen oder Kommunikation via *RMI (Remote Method Invocation)*, so dass sich der Entwickler nicht explizit um solche Details kümmern muss.

In der EJB-Architektur sind drei EJB-Typen festgelegt:

- *Session Beans*,
- *Entity Beans* und
- *Message Driven Beans*.

Session Beans stehen normalerweise für Geschäftsprozesse. Sie repräsentieren nicht direkt Geschäftsdaten, können jedoch auf solche zugreifen (z.B. über JDBC). Ihre Lebensdauer entspricht lediglich der aktuellen Sitzung. Dagegen repräsentieren Entity Beans Geschäftsobjekte, die mit bestimmten Daten in Verbindung stehen. Sie existieren so lange, wie die zugehörigen Daten vorhanden sind. Schließlich ist eine Message Driven Bean eine zustandslose Komponente, die durch Nachrichten angesprochen wird. Tabelle 2-1 stellt die Charakteristiken der verschiedenen EJB-Typen dar.

| *Charakteristik* | *Session Bean* | *Entity Bean* | *Message Driven Bean* |
|---|---|---|---|
| Identität | nein | ja | nein |
| Transaktional | ja | ja | ja |
| Clientzugriff | exklusiv | gemeinsam | indirekt |
| Kommunikation | synchron | synchron | asynchron |
| Persistent | nein | ja | nein |
| Lebensdauer | kurz | lang | kurz |

Tabelle 2-1: Vergleich der verschiedenen EJB-Typen [SS02]

### *2.4.4 Konnektoren – Die J2EE Connector Architecure (JCA)*

Die *J2EE Connector Architecture (JCA)* definiert standardisierte Java-Schnittstellen zur Vereinfachung der Integration von *Enterprise-Informationssystemen (Enterprise Information Systems – EIS)* mit J2EE-basierten Java-Anwendungen. Mit Hilfe dieser Schnittstellen – oder Konnektoren – kann ein J2EE-Entwickler auf vorhandene Systeme wie Datenbanken (z.B. DB2) und Transaktionssysteme (z.B. CICS) zugreifen. Ein Konnektor wird auch *Ressourcenadapter* genannt und bildet die Verbindung zwischen einer J2EE-Anwendung und einem EIS. Dabei sind die meisten Konnektoren plattformspezifisch, und der Entwickler muss erst den Umgang mit einer neuen Programmierschnittstelle erlernen.

Die JCA in Version 1.0 definiert eine Reihe von Komponenten, aus denen sich die Architektur in Abbildung 2-6 ergibt. Es gibt zwei grundlegende Bestandteile: Das *Common Client Interface (CCI)* und so genannte *Systemkontrakte*. Der Entwickler eines Konnektors (oder Ressourcenadapters) stellt die Schnittstelle zum CCI sowie seinen Teil des Systemkontrakts zur Verfügung. Der Hersteller des J2EE-Servers entwickelt dagegen den anderen Teil des Systemkontrakts als Bestandteil der J2EE-Plattform.

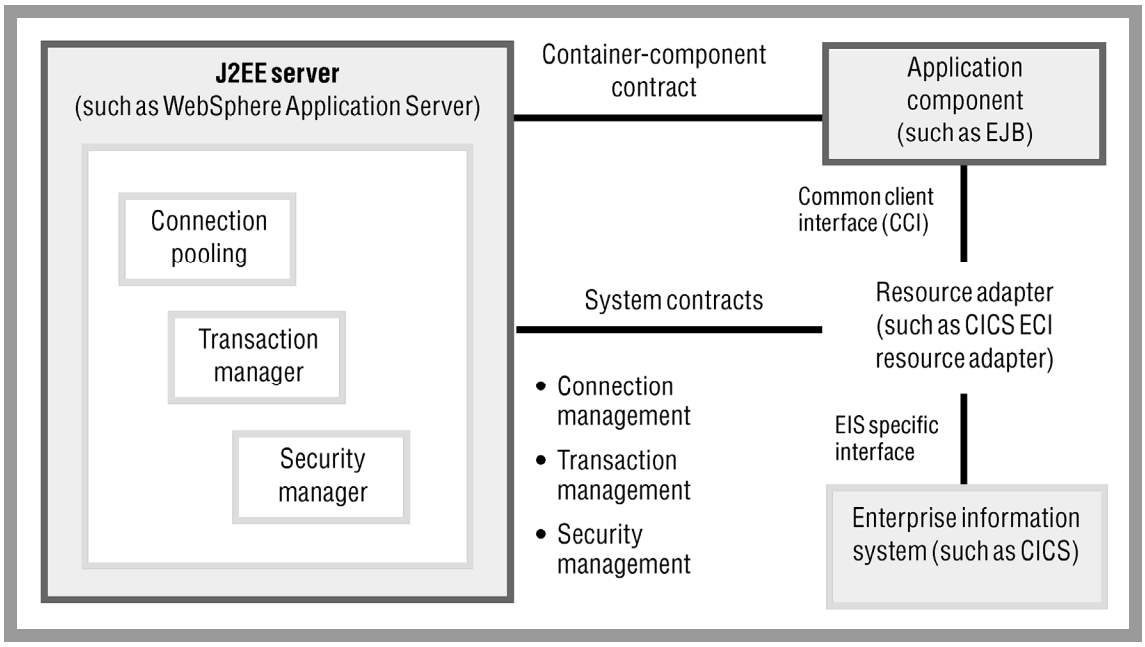


Abbildung 2-6: Die Komponenten der J2EE Connector Architecture [IWC]

*Das Common Client Interface (CCI)*

Das CCI definiert ein einheitliches Programmiermodell zur Interaktion mit Ressourcenadaptern und ist unabhängig vom einzelnen EIS. Natürlich unterscheidet sich der Programmcode zum Zugriff auf CICS von dem zum Zugriff auf IMS. Die Aufrufparameter sind unterschiedlich, aufgrund der einheitlichen CCI-Klassen ist der Ablauf der Befehle jedoch derselbe. Die CCI-Schnittstelle ähnelt anderen J2EE-Schnittstellen wie zum Beispiel JDBC.

*Systemkontrakte*

Die JCA definiert eine Reihe von systemspezifischen Kontrakten zwischen einem *J2EE-Anwendungsserver* und einem Ressourcenadapter. Diese Kontrakte umreißen den Bereich, der vom Anwendungsserver für die JCA-Komponenten zur Verfügung gestellt wird. Es gibt drei Standardkontrakte:

- *Ein Kontrakt zum Verbindungsmanagement* bietet ein konsistentes Programmiermodell zum Herstellen einer Verbindung und zum *Verbindungspooling* zwischen einer J2EE-Anwendung und einem EIS. Erst durch diesen Kontrakt kann ein Anwendungsserver eigene Dienste für Transaktionen und Sicherheit anbieten.

- *Ein Kontrakt zum Transaktionsmanagement* ermöglicht den transaktionalen Zugriff auf unterliegende Ressourcen. Dadurch werden auch Transaktionen über mehrere Ressourcenmanager hinweg möglich (globale Transaktionen).

- *Ein Kontrakt zum Sicherheitsmanagement* ermöglicht einen gesicherten Zugriff auf ein EIS. Der Sicherheitsmechanismus hängt dabei vom verwendeten Enterprise-Informationssystem ab.

Derartige Systemkontrakte sind für den Anwendungsentwickler unsichtbar, so dass er die Dienste nicht selbst programmieren muss. In Umgebungen wie dem WebSphere Application Server stellen Systemkontrakte vertragliche Vereinbarungen zwischen dem J2EE-Anwendungsserver und dem Ressourcenadapter dar, die festlegen, wie mit Verbindungen, Transaktionen und Sicherheitsangelegenheiten umgegangen werden soll.

## 2.5 Der WebSphere Application Server

In einer Zeitspanne von weniger als drei Jahren ist der Anwendungsserver zu einem der wichtigsten Begriffe innerhalb der Computerindustrie geworden. Im Allgemeinen beschreibt man einen Anwendungsserver als eine »Serverbox«, mit deren Hilfe E-Business-Anwendungen ausgeführt werden können. Weiterhin soll die schnelle und effiziente Inbetriebnahme von solchen Anwendungen ermöglicht werden. Es gibt jedoch erhebliche Unterschiede zwischen den verschiedenen Anwendungsservern. Diese liegen in der spezifischen Hardware, dem eingesetzten Betriebssystem und der ausgewählten *Middleware* und führen zu unterschiedlichen Ergebnissen in Gebieten wie Leistung, Skalierbarkeit und Sicherheit.

### *2.5.1 Eigenschaften von Anwendungsservern*

Da alle E-Business-Anwendungen von vorhandenen Systemdiensten wie Persistenz, Sicherheit und Transaktionen abhängen, wurde bei der Entwicklung von Anwendungsservern schnell deutlich, dass sich die Produktivität der Anwendungsentwickler stark verbessern ließe, wenn man diese Dienste standardisiert und für die Anwendungen transparent gestaltet. So kann der Anwendungsentwickler diese Systemdienste als kompakte Bausteine nutzen und in seine E-Business-Anwendung integrieren, ohne über ein tiefes Wissen der plattformspezifischen Infrastruktur verfügen zu müssen. Er kann sich also allein auf das Erstellen der Businesslogik konzentrieren, und der Anwendungsserver erledigt transparent die systemabhängigen Arbeiten.

Wie schon erwähnt, gibt es große Unterschiede zwischen den verschiedenen Anwendungsservern. Diese liegen vor allem in der Hardware (IBM-zSeries-, *S390*-, *Intel*- und *RISC*-Maschinen) und im verwendeten Betriebssystem (z/OS, Microsoft Windows, UNIX und Linux). Bei traditionellen Methoden der Anwendungsentwicklung musste der Zugriff auf die Systemdienste explizit programmiert werden. Waren Hardware und Betriebssystem ausgewählt, so konnte ein geschickter Programmierer extrem effizienten Code erstellen, der die Fähigkeiten der Plattform vollständig ausnutzte. Bei der J2EE-Plattform wechselt die Verantwortlichkeit der effizienten Nutzung der Systemressourcen vom Anwendungsentwickler zum Anbieter des Containers. Die Businesslogik ist auf allen Systemen dieselbe, die Unterschiede zwischen den Plattformen bestehen zum Beispiel im Datenzugriff, in der Transaktionsabwicklung oder in der Gewährleistung der Sicherheit. Es ist daher für einen Anwendungsserver entscheidend, J2EE-Container bereitzustellen, die die Fähigkeiten der verwendeten Plattform so gut wie möglich ausnutzen.

Es gibt also eine Reihe von Anforderungen, die an einen Anwendungsserver gestellt werden müssen. Er sollte auf möglichst vielen Plattformen zur Verfügung stehen und jeweils für diese optimiert sein. Bestehende Ressourcen des Systems sollten in neue J2EE-Anwendungen integriert werden können. In einer z/OS-Umgebung wären hier u.a. DB2 und CICS zu nennen. Ein weiterer Schlüssel zum Erfolg eines Anwendungsservers ist die Qualität der Werkzeuge zum Entwickeln und Installieren von Anwendungen.

### *2.5.2 Der WebSphere Application Server für z/OS und OS/390*

Die Firma IBM hat schon früh Java-Technologie eingesetzt und zur Weiterentwicklung dieser beigetragen. IBM hatte das Ziel, J2EE auf allen IBM- und Nicht-IBM-Plattformen zur Verfügung zu stellen. 1999 veröffentlichte IBM die WebSphere-Produktfamilie, die Unternehmen an allen Stellen der E-Business-Entwicklung unterstützen sollte. Der wichtigste Teil von WebSphere ist die so genannte *WebSphere Foundation*, die die wichtigsten E-Business-Funktionen zur Verfügung stellt. Die gesamte WebSphere-Software baut auf diese auf – auch der *WebSphere Application Server für z/OS und OS/390*.

Der WebSphere Application Server für z/OS und OS/390 macht die S/390- und zSeries-Architektur zu einer Plattform für moderne E-Business-Anwendungen und nutzt die ausgezeichneten Eigenschaften dieser Architektur in Punkten wie Skalierbarkeit, Leistungsfähigkeit, Sicherheit und Verfügbarkeit. Es werden zum Beispiel Funktionen wie Parallel Sysplex und Workload Management ausgenutzt.

Durch die Unterstützung von offenen Industriestandards wie HTML, HTTP, IIOP, J2EE und CORBA können neue E-Business-Anwendungen nun unabhängig von der Entwicklungsplattform auf dem WebSphere Application Server für z/OS und OS/390 installiert werden. Das wichtigste Element des Produktplans des WebSphere Application Server für z/OS und OS/390 besteht darin, dass die Fähigkeiten des J2EE-Containers auf z/OS und OS/390 mit dem Rest der WebSphere-Familie in Einklang gebracht werden. Daher wurden mit Version 4.0 die Spezifikationen für JSPs, Servlets und EJBs innerhalb der Familie vereinheitlicht.

WebSphere Application Server für z/OS und OS/390 unterstützt Enterprise JavaBeans (EJBs) in Version 1.1, Java Servlets in Version 2.2 und Java Server Pages (JSPs) in Version 1.1. Weiterhin ist mit Hilfe des *Inter-ORB-Protokolls (IIOP)* ein entfernter Prozeduraufruf (Remote Method Invocation – RMI) möglich. Außerdem existieren IMS- und CICS-Konnektoren (z.B. das CICS Transaction Gateway), die der J2EE-Konnektor-Architektur entsprechen. Durch die vollständige Unterstützung der J2EE-Komponenten ist der WebSphere Application Server für z/OS und OS/390 mit WebSphere-Servern auf anderen Plattformen sowie Nicht-IBM-J2EE-Servern kompatibel. Zusätzlich zur Unterstützung von J2EE-Anwendungen unterstützt der WebSphere Application Server für z/OS und OS/390 auch die verteilte Plattform CORBA in Version 2.1.

## 2.6 Windows-Tools für den WebSphere Application Server V4 für z/OS

### *2.6.1 Das System Management End-User Interface (SMEUI)*

Das System Management End-User Interface (SMEUI) besteht aus den beiden Windows-Anwendungen *Administration* und *Operations*, die mit dem WebSphere Application Server V4 für z/OS mitgeliefert werden. In Version 5 des WebSphere Application Server wird stattdessen eine webbasierte Schnittstelle eingesetzt.

*Administration*

Mit der Anwendung *Administration* kann man die Konfiguration des WebSphere Application Server V4 für z/OS einsehen und verändern. Man kann also J2EE-Anwendungen installieren und die zugehörige Ablaufumgebung konfigurieren.

Die Konfiguration wird in Form eines Baums von Objekten dargestellt, der die Beziehungen dieser darstellt. Bei Objekten handelt es sich um zum Beispiel um Sysplexe, Systeme, J2EE-

Server und -Instanzen sowie selbst erstellte J2EE-Anwendungen. Die höchste Objektebene ist jedoch die *Konversation*. Sie enthält die gesamte Konfiguration des WebSphere Application Server zu einem bestimmten Zeitpunkt. Abbildung 2-7 zeigt ein Beispielfenster der Anwendung Administration.

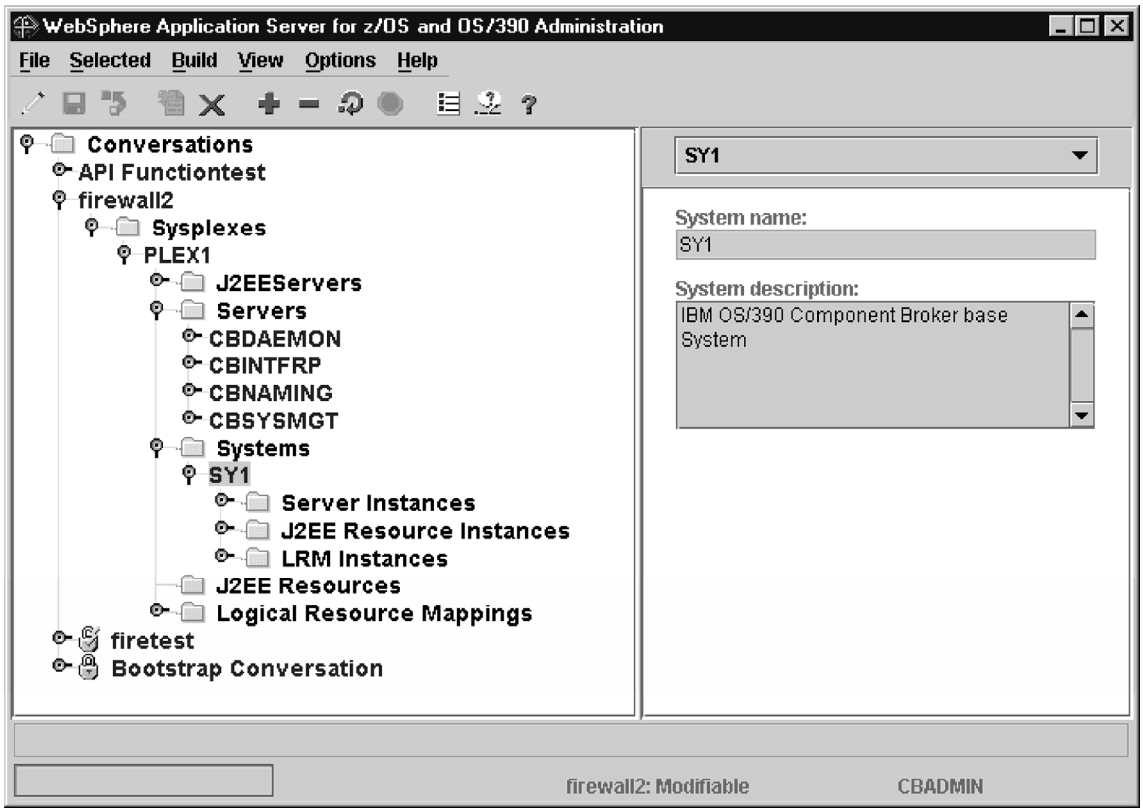


Abbildung 2-7: Die Anwendung *Administration* [WS02]

*Operations*

Mit der Anwendung *Operations* kann man die Server und Serverinstanzen des WebSphere Application Server V4 für z/OS verwalten.

Jeder Server und jede Serverinstanz der aktuellen Konfiguration werden als Symbol dargestellt. Man kann den Status der Serverinstanzen und die Eigenschaften der einzelnen Objekte abfragen. Weiterhin können Server und Serverinstanzen gestartet und angehalten werden. Abbildung 2-8 zeigt ein Beispielfenster der Anwendung *Operations*.

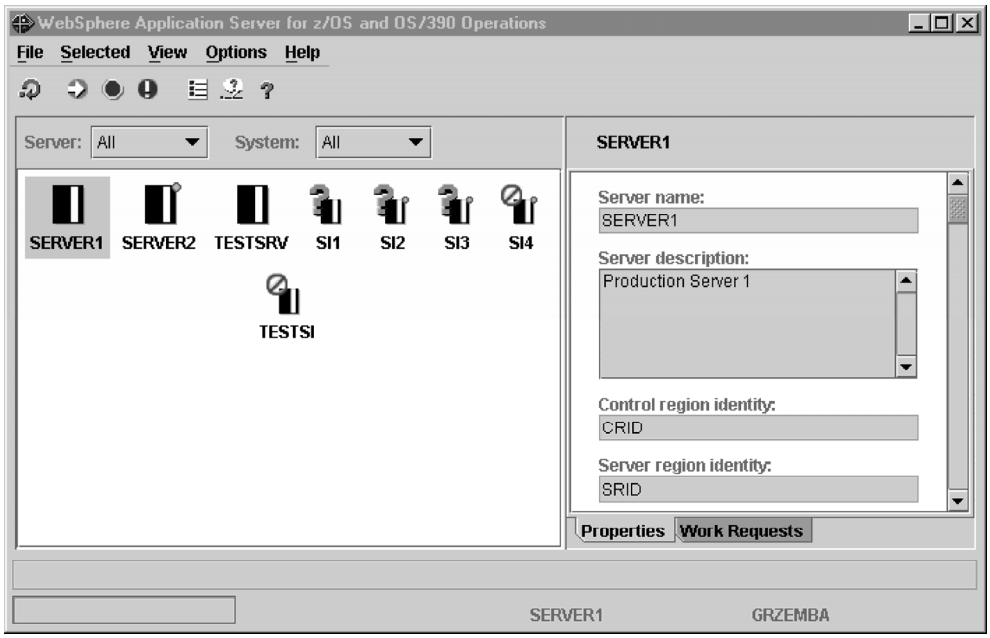


Abbildung 2-8: Die Anwendung *Operations* [WS02]

### *2.6.2 Das Application Assembly Tool*

Das *Application Assembly Tool* für den WebSphere Application Server V4 für z/OS ist ebenfalls eine Windows-Anwendung und wird benötigt, um vorhandene J2EE-Komponenten zu einer Anwendung zusammenzusetzen. Bei den J2EE-Komponenten kann es sich dabei um Enterprise JavaBeans (EJBs), Webanwendungen (bestehend aus Servlets und JSPs) oder aus einer Kombination von allem handeln.

Das Werkzeug beinhaltet eine grafische Benutzerschnittstelle, mit der die *Deployment*-Informationen während des Zusammenstellens der J2EE-Anwendung so abgeändert werden können, dass diese mit dem WebSphere Application Server V4 für z/OS kompatibel ist. Zum Beispiel werden EJB-Referenzen, Ressourcenreferenzen und Sicherheitsinformationen der J2EE-Anwendung modifiziert.

Neben anderen Werkzeugen für den WebSphere Application Server V4 für z/OS ist das Application Assembly Tool unter folgender Adresse herunterzuladen:

http://www-306.ibm.com/software/webservers/appserv/download_v4z.html.

## 2.7 CICS Transaction Gateway

Das *IBM CICS Transaction Gateway (CTG)* ermöglicht Webbrowsern und Netzwerkcomputern einen sicheren und einfachen Zugriff auf CICS-Anwendungen. Es sind viele verschiedene

Konfigurationen möglich, bei denen Standardinternetprotokolle zum Einsatz kommen. Das CICS Transaction Gateway kann zum Beispiel als Konnektor für den IBM WebSphere Application Server verwendet werden. Es wird eine Programmierschnittstelle zur Verfügung gestellt, durch die Java-Programmierer die Funktionen der J2EE-Plattform nutzen können.

Das CICS Transaction Gateway ist für viele verschieden Betriebssysteme verfügbar, zum Beispiel für IBM z/OS, Linux für zSeries, IBM AIX, HP-UX, Microsoft Windows, Linux und Sun Solaris. Von Windows- und UNIX-Rechnern aus kann das CICS Transaction Gateway auf mehrere Arten von CICS-Servern zugreifen, während das CICS Transaction Gateway für z/OS nur für den CICS Transaction Server für z/OS einsetzbar ist.

Abbildung 2-9 zeigt, wie ein Webclient auf CICS zugreift. Dabei ist das CICS Transaction Gateway auf demselben Rechner installiert wie der Webserver.

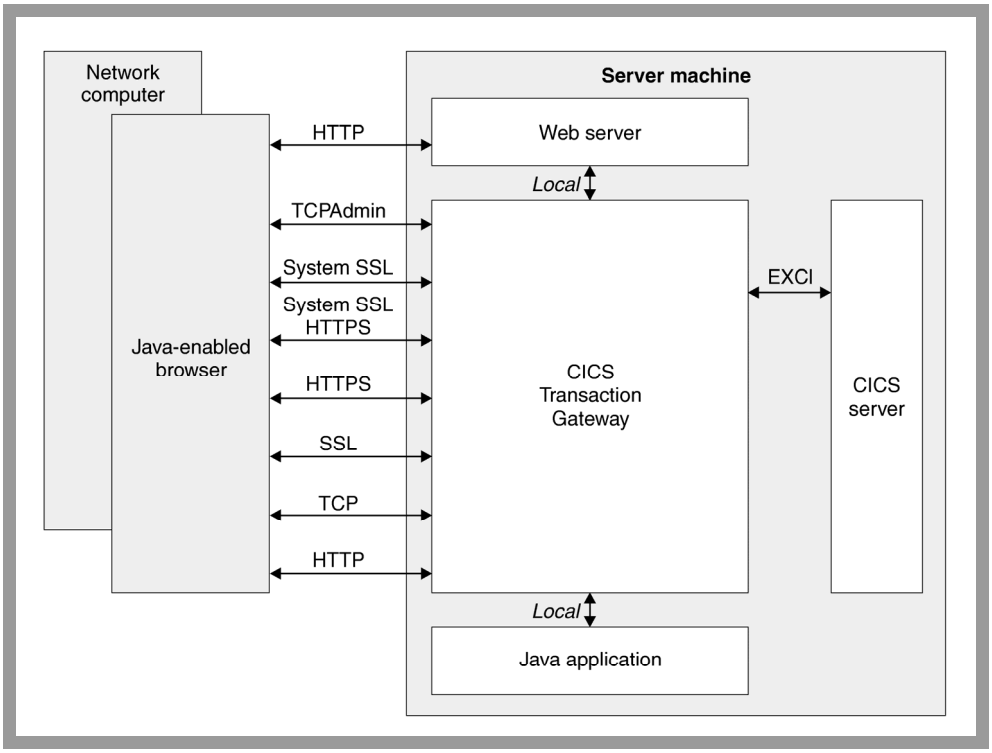


Abbildung 2-9: Überblick über das CICS Transaction Gateway
[CG02a]

Bei der Kommunikation zwischen dem CICS Transaction Gateway und den Clientanwendungen werden folgende Protokolle verwendet:

- Transmission Control Protocol/Internet Protocol (TCP/IP)

- Hypertext Transfer Protocol (HTTP)
- Secure Sockets Layer (SSL)
- HTTP über SSL (HTTPS)

### *2.7.1 Möglichkeiten zum Zugriff auf CICS*

Da die meisten heutigen Anwendungen in einem so genannten Enterprise-Informationssystem wie CICS Transaction Server ablaufen, ist es wichtig, dass vorhandene Komponenten wieder verwendet und wichtige Anwendungen in neue E-Business-Lösungen integriert werden können. Daher verwendet das CICS Transaction Gateway zum Zugriff auf CICS die J2EE Connector Architecture (JCA). Durch diese Architektur können Entwickler neue Anwendungen erstellen, die sie aus vorhandenen Komponenten wie zum Beispiel CICS-Programmen zusammensetzen.

Es gibt jedoch noch weitere Möglichkeiten zum Zugriff auf vorhandene CICS Anwendungen:

- Ein nachrichtenbasierter Ansatz mit *IBM WebSphere MQ*
- RMI-Aufrufe über das IIOP-Protokoll, falls die vorhandene CICS-Anwendung als Enterprise JavaBean zur Verfügung steht
- *SOAP*-Aufrufe *(Simple Object Access Protocol)* an CICS über HTTP oder WebSphere MQ

### *2.7.2 Schnittstellen des CICS Transaction Gateway*

Die Schnittstellen des CICS Transaction Gateway werden basierend auf den verwendeten CICS-Funktionen in drei Kategorien eingeteilt:

- *External Call Interface (ECI)*,
- *External Presentation Interface (EPI)* und
- *External Security Interface (ESI).*

*External Call Interface (ECI)*

Das ECI wird verwendet, um CICS-Programme aufzurufen, die eine *COMMAREA (Communications Area)* einsetzen. Bei der COMMAREA handelt es sich um einen Puffer, über den Client und CICS-Server Daten austauschen (Abbildung 2-10). Für CICS ist die Client-Anfrage nichts anderes als ein DPL-Befehl (Distributed Program Link). Das ECI ist die einzige für das CICS Transaction Gateway für z/OS verfügbare Schnittstelle, so dass im weiteren Verlauf der Arbeit nur auf diese eingegangen wird.

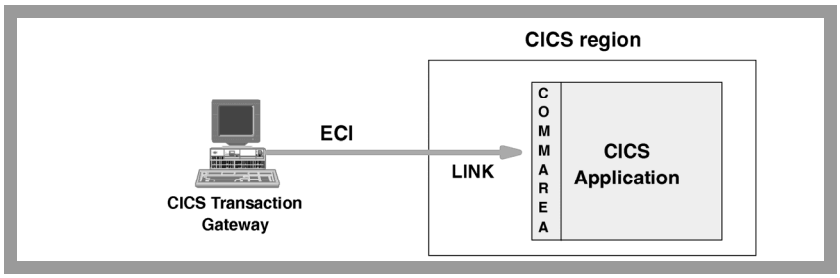


Abbildung 2-10: Das External Call Interface (ECI) [CG02b]

*External Presentation Interface (EPI)*

Das EPI wird zum Aufrufen von 3270-Transaktionen eingesetzt. In CICS wird ein Terminal installiert, und es wird eine Anfrage von einem entfernten Terminal aus gestellt, den das CICS Transaction Gateway steuert. Somit wird auch der Zugriff auf BMS-Programme ermöglicht. Jedoch ist das EPI nur auf verteilten Plattformen und nicht für das CICS Transaction Gateway für z/OS verfügbar.

*External Security Interface (ESI)*

Mit dem ESI können Benutzer-IDs und Kennwörter geändert werden, die im *ESM (External Security Manager)* von CICS – wie *RACF (Resource Access Control Facility)* – abgespeichert sind. Es basiert auf den *PEM*-Funktionen *(Password Expiration Management)* von CICS. Wie das EPI ist das ECI ebenfalls nicht für das CICS Transaction Gateaway für z/OS verfügbar.

### *2.7.3 Bestandteile des CICS Transaction Gateway*

Das CICS Transaction Gateway ist eine Zusammenstellung von Client- und Server-Komponenten, mit Hilfe derer Java-Anwendungen auf Dienste eines CICS-Servers zugreifen können. Bei der Java-Anwendung kann es sich dabei um ein Applet, ein Servlet oder eine Enterprise JavaBean handeln.

*Der Client-Daemon*

Der *Client-Daemon* ist auf allen verteilten Plattformen ein Bestandteil des CICS Transaction Gateway. Er stellt eine ähnliche Client/Server-Zugriffsmöglichkeit auf CICS dar wie der *CICS Universal Client*. Der Client-Daemon ist im CICS Transaction Gateway für z/OS und OS/390 nicht vorhanden.

*Der Gateway-Daemon*

Der *Gateway-Daemon* ist ein Prozess mit langer Lebensdauer, der als Server für clientseitige Java-Anwendungen fungiert, indem er einen festgelegten TCP/IP-Port abhört. Der Aufbau des Gateway-Daemon ist unter z/OS etwas anders als auf verteilten Plattformen. Auf diesen werden Funktionen bereitgestellt, die dem CICS Universal Client ähneln, während unter z/OS EXCI-Verbindungen eingesetzt werden, die auf vorhandene CICS-Programme zugreifen. Der Gateway-Daemon wird in der Regel nicht benötigt, wenn die Java-Anwendung auf demselben Rechner abläuft wie das CICS Transaction Gateway. Dann kann das `local`-Protokoll des CICS Transaction Gateway verwendet werden, das die unterliegenden Zugriffsmechanismen direkt anspricht. Abbildung 2-11 zeigt den Aufbau des Gateway-Daemon unter z/OS und OS/390.

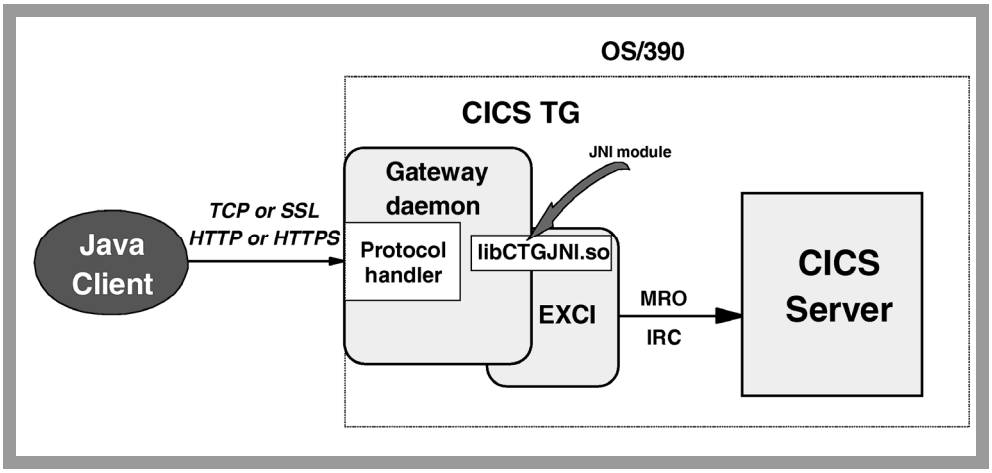


Abbildung 2-11: Der Gateway-Daemon unter z/OS und OS/390 [CG02b]

*Das Konfigurationstool*

Das Konfigurationstool (`ctgcfg`) ist eine auf Java basierende grafische Benutzerschnittstelle *(Graphical User Interface – GUI)*, die das CICS Transaction Gateway auf allen Plattformen bereitstellt. Es wird verwendet, um die Eigenschaften von Gateway- und Client-Daemon zu konfigurieren. Unter z/OS und OS/390 kann das Konfigurationstool nicht direkt angezeigt

werden. Stattdessen muss erst der Bildschirm (`DISPLAY`) auf einen anderen Rechner mit einem X-Windows-Server umgeleitet werden. Weiterhin können im Konfigurationstool Protokollinformationen sowie der Workload-Manager konfiguriert werden.

Abbildung 2-12 zeigt das äußere Erscheinungsbild des Konfigurationstools.

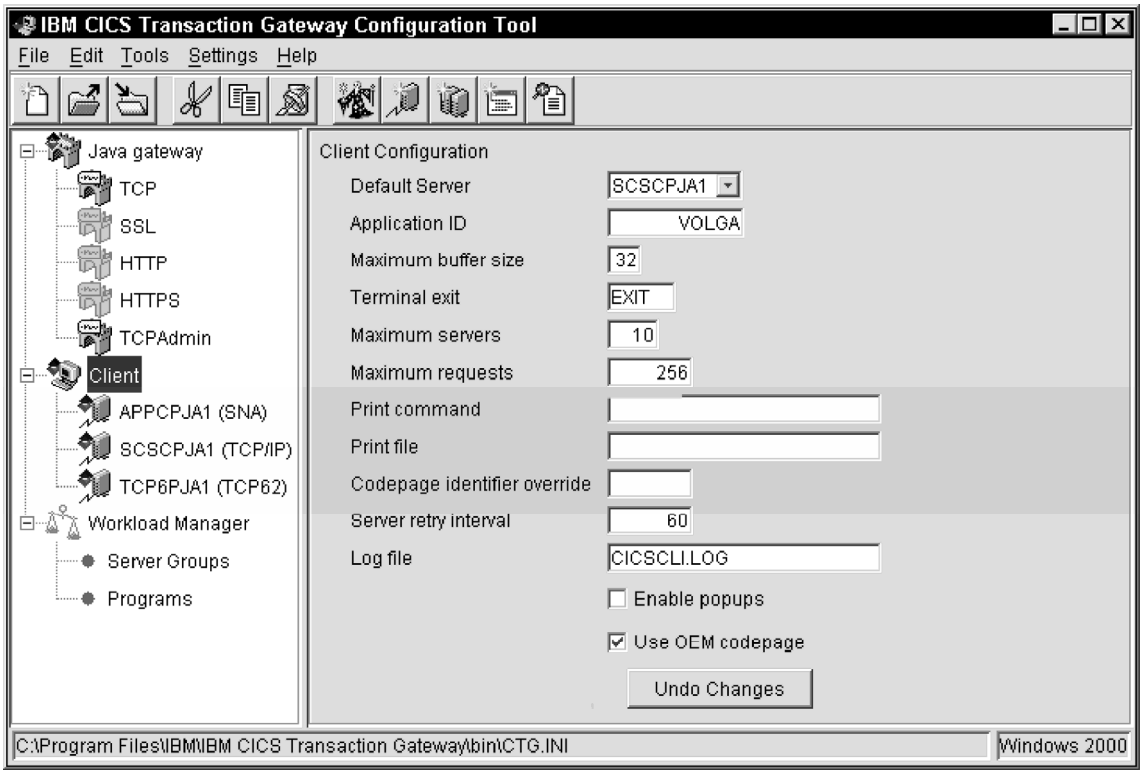


Abbildung 2-12: Das Konfigurationstool [CG02b]

*Der CICS-ECI-Ressourcenadapter*

Das CICS Transaction Gateway stellt zwei Ressourcenadapter bereit:

- den *CICS-ECI-Ressourcenadapter* und
- den *CICS-EPI-Ressourcendapter*.

Für das CICS Transaction Gateaway für z/OS ist jedoch lediglich der CICS-ECI-Ressourcenadapter verfügbar. Dieser ist zum Zugriff auf CICS-COMMAREA-Programme notwendig. Er entspricht der JCA und stellt eine *API (Application Programming Interface)* bereit, mit der der Anwender die Funktionen von J2EE-Laufzeitumgebungen (wie dem WebSphere Application Server) ausnutzen kann. Weitere Eigenschaften sowie die Struktur von Ressourcenadaptern sind in *Abschnitt 2.4.4* zu finden.

## 2.8 Übersicht

Tabelle 2-2 fasst die in den folgenden Kapiteln eingesetzten Hard- und Softwarekomponenten zusammen.

| ***Komponente*** | ***Beschreibung/Version*** | ***Betriebssystem*** |
|---|---|---|
| Rechner | Name: Yoda, IP-Adresse: 139.18.12.91 | |
| z/OS-Betriebssystem | Version 1.4 Sommer | |
| CICS Transaction Server | Version 2.2 | z/OS |
| DB2 | Version 7.10 | z/OS |
| WebSphere Application Server | Version 4.0.1 | z/OS |
| IBM HTTP Server | Version 5.3 | z/OS |
| CICS Transaction Gateway | Version 5.00 | z/OS |
| | | Windows |
| IBM Developer Kit, Java 2 Technology Edition | Version 1.3.1 | z/OS |
| System Management End-User Interface | Version 4.01.025 | Windows |
| Application Assembly Tool for z/OS and OS/390 | Version 4.00.032 | Windows |

Tabelle 2-2: Verwendete Hard- und Softwarekomponenten

*K a p i t e l 3*

# Zusammenspiel der Softwarekomponenten

In diesem Kapitel soll dargestellt werden, wie die in *Kapitel 2: Verwendete Software* vorgestellten Softwarekomponenten zusammenarbeiten und eine so genannte *Topologie* bilden, deren Ziel der Zugriff auf eine bestehende CICS-Anwendung ist. Es werden drei verschiedene Topologien vorgestellt. Eine davon wird genauer beleuchtet und ist Gegenstand aller nachfolgenden Kapitel.

## 3.1 Verschiedene Topologien um das CICS Transaction Gateway

Im Folgenden werden drei mögliche Topologien vorgestellt, die jeweils den WebSphere Application Server in Verbindung mit dem CICS-ECI-Ressourcenadapter einsetzen, um auf einen CICS-Server unter z/OS zuzugreifen:

*Topologie 1:* Der WebSphere Application Server und das CICS Transaction Gateway befinden sich beide in einer verteilten (Nicht-z/OS-) Umgebung.

*Topologie 2:* Der WebSphere Application Server befindet sich in einer verteilten Plattform, während das CICS Transaction Gateway im z/OS-System installiert wurde.

*Topologie 3:* Sowohl der WebSphere Application Server als auch das CICS Transaction Gateway befinden sich im z/OS-System.

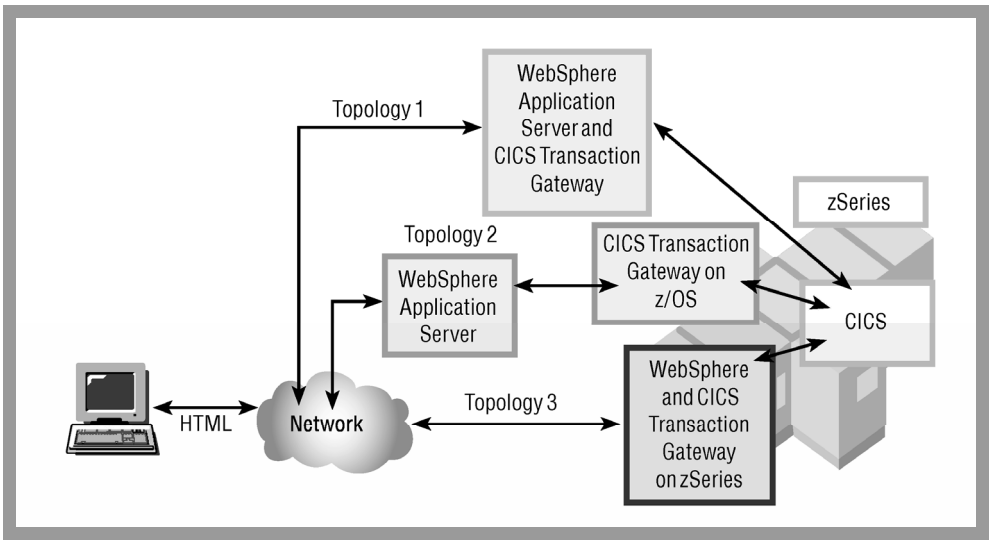


Abbildung 3-1: Verschiedene CTG-Topologien [IWC]

Da der WebSphere Application Server für z/OS verwendet werden sollte, kam nur Topologie 3 in Frage, bei der sich außer einem Browser alle Softwarekomponenten im z/OS-System befinden.

## 3.2 WebSphere Application Server und CICS Transaction Gateway im z/OS-System

In dieser Topologie wird lediglich der CICS-ECI-Ressourcenadapter unterstützt. Es wird ein lokales CICS Transaction Gateway eingesetzt, so dass der WebSphere Application Server einfach mit Hilfe von Cross-Memory-Aufrufen – in diesem Fall *EXCI*-Verbindungen *(EXternal CICS Interface)* – auf CICS zugreifen kann. Abbildung 3-2 stellt die sich aus diesen Voraussetzungen ergebende Topologie dar.

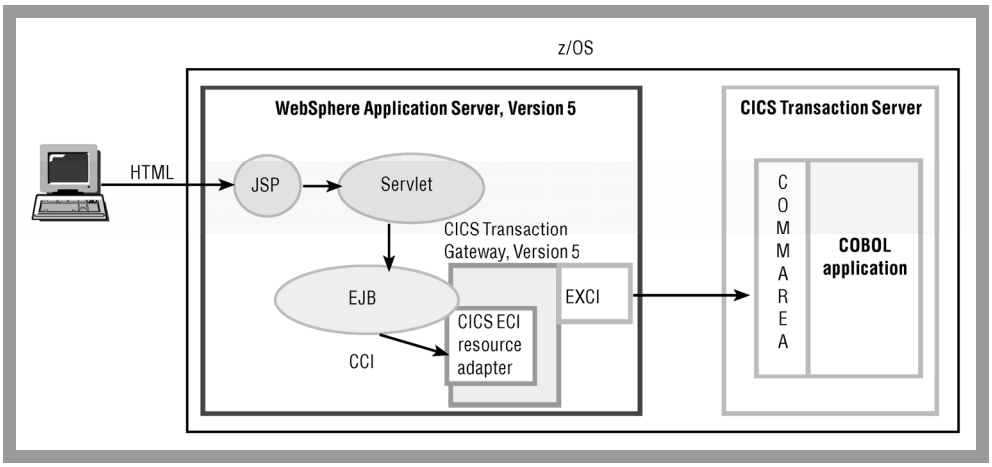


Abbildung 3-2: WebSphere Application Server und CTG unter z/OS [IWC]

### *3.2.1 Eigenschaften dieser Topologie*

Diese Art von Topologie und der dabei eingesetzte CICS-ECI-Ressourcenadapter verfügen über eine Reihe positiver Eigenschaften, die auf die in *Abschnitt 2.4.4* behandelten Systemkontrakte zurückgehen.

*Verbindungsmanagement*

Der Verbindungspool setzt sich aus einer Reihe von Verbindungsobjekten zusammen, die vom WebSphere Application Server verwaltet werden. Diese Verbindungen hängen jedoch nicht direkt mit den EXCI-Verbindungen zu CICS zusammen.

*Transaktionsmanagement*

Es wird mit Hilfe von *RRS (Resource Recovery Services)* – einem fundamentalen Bestandteil von z/OS – ein *Zwei-Phasen-Commit-Protokoll* eingesetzt. RRS agiert dabei als externer Transaktionsmanager, wobei sich der Transaktionsbereich vom WebSphere Application Server über CICS bis hin zu anderen z/OS-Subsystemen wie IMS/TM, IMS/DB, DB2 und WebSphere MQ erstrecken kann. Diese Topologie ist somit die einzige, in der sich der Transaktionsbereich des Zwei-Phasen-Commit-Protokolls über alle CICS-Resourcen erstreckt. Damit ist sichergestellt, dass beim Fehlschlagen einer Transaktion ein *Rollback* durchgeführt werden kann, das alle verwendeten Ressourcen vollständig wiederherstellt. Ist die Transaktion erfolgreich, wird mit *Commit* bestätigt.

*Sicherheitsmanagement*

Es wird sowohl eine container- als auch eine komponentengesteuerte Benutzeranmeldung unterstützt. Der WebSphere Application Server kann so konfiguriert werden, dass er eine

lokale Registrierung wie IBM RACF einsetzt. Dann wird die dem WebSphere Application Server zum Beispiel in einem Formular übergebene Benutzer-ID automatisch als RACF-Benutzer-ID aufgefasst. Diese Identität kann nun auch an CICS weitergegeben werden, so dass die Benutzer-ID tatsächlich von einem Ende der Topologie (dem Eingabeformular im Browser des PCs) zum anderen Ende (dem CICS Transaction Server) gelangt.

Im weiteren Verlauf dieser Arbeit spielen Sicherheitsfragen jedoch keine Rolle, so dass die vorgesehenen Sicherheitsmechanismen an allen Stellen der Topologie deaktiviert wurden.

### *3.2.2 Ablauf des Zugriffs auf CICS*

In der folgenden Aufzählung soll der Ablauf des Zugriffs vom WebSphere Application Server auf CICS sowie der Datenfluss vom Browser des Client-PCs bis hin zu CICS und wieder zurück veranschaulicht werden:

1. Der Webserver lädt und initialisiert das Servlet. Dies kann zum Beispiel geschehen, wenn der Webbrowser oder Netzwerkcomputer eine Anfrage an das Servlet stellt (zum Beispiel über ein Formular in einer HTML- oder JSP-Datei).

2. Das Servlet sucht per *JNDI (Java Naming and Directory Interface)* nach der EJB und ruft mit den entsprechenden Parametern eine geeignete Methode der Bean auf.

3. Abbildung 3-3 enthält den in der Enterprise JavaBean enthaltenen Code. Diese sucht ebenfalls per JNDI nach dem im WebSphere Application Server installierten CICS-ECI-Ressourcenadapter und erhält eine Verbindung aus dessen Verbindungspool (Abbildung 3-3, Zeilen 1 und 4).

```
1 ConnectionFactory cf = <Lookup from JNDI namespace>
  ECIConnectionSpec cs = new ECIConnectionSpec();
  cs.setXXX();    //Set any connection specific properties

4 Connection conn = cf.getConnection( cs );
  Interaction int = conn.createInteraction();
  ECIInteractionSpec is = new ECIInteractionSpec();
  is.setXXX();    //Set any interaction specific properties

8 RecordImpl in = new RecordImpl();
  RecordImpl out = new RecordImpl();

10 int.execute( is, in, out );
   int.close();
   conn.close();
```

Abbildung 3-3: Aufruf des Ressourcenadapters [CG02c]

4. Aus dieser Verbindung kann eine so genannte Interaktion erstellt werden (Abbildung 3-3, Zeile 5), die gestartet werden kann. Als Eingabewerte dieses Startbefehls werden die aus dem Servlet erhaltenen Parameter sowie eine Ein- und eine Ausgabevariable für die COMMAREA übergeben (Abbildung 3-3, Zeile 10).

5. Erst jetzt wird eine EXCI-Verbindung zu CICS hergestellt, die für CICS wie ein DPL-Befehl mit den entsprechenden Parametern erscheint.

6. CICS ruft das in den Parametern des DPL-Befehls angegebene Programm auf, das seine Ausgabe in die COMMAREA schreibt.

7. Nach erfolgreicher Ausführung stehen in der Ausgabevariable der COMMAREA die von CICS erhaltenen Daten.

8. Diese Daten werden nun von der EJB an das Servlet weitergegeben.

9. Dieses Servlet erstellt eine HTTP-Antwort, die zurück an den Webbrowser und somit auf den Bildschirm des Benutzers gesendet wird.

*K a p i t e l 4*

# Einrichtung der Software

In diesem Kapitel werden die Schritte beschrieben, die zur Einrichtung der Konfiguration aus *Kapitel 3: Zusammenspiel der Softwarekomponenten* notwendig sind. Daraufhin soll diese Konfiguration mit Hilfe einer J2EE-Beispielanwendung getestet werden.

## 4.1 EXCI-Verbindungen zu CICS

Das *EXCI (EXternal CICS Interface – externe CICS-Schnittstelle)* ermöglicht unter z/OS die Verbindung zwischen Nicht-CICS-Anwendungen (Client-Programmen) und Programmen in einer CICS-Region (Server-Programmen). Zum Beispiel können Stapelverarbeitungsprogramme und auch das CICS Transaction Gateway über EXCI-Verbindungen auf CICS zugreifen (Abbildung 4-1). Der Austausch von Daten zwischen Client- und Serverprogramm erfolgt mit Hilfe der COMMAREA.

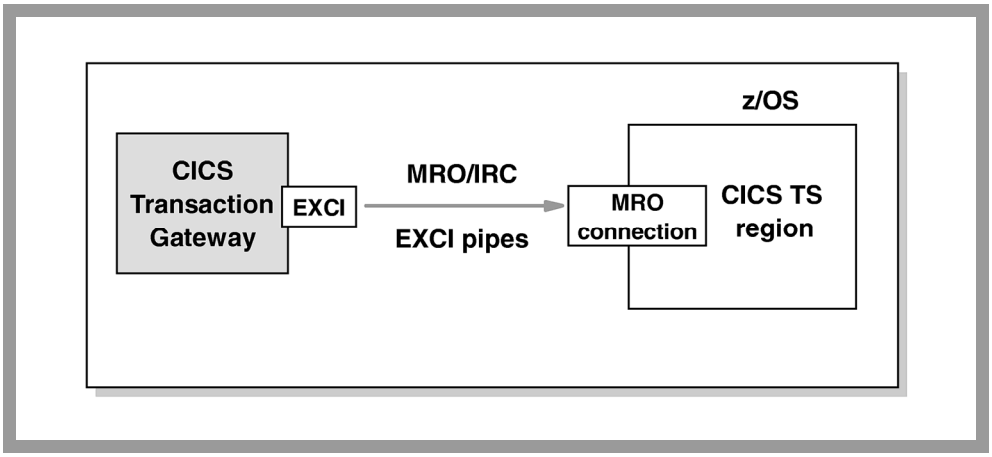


Abbildung 4-1: EXCI-Verbindungen zu CICS [CG02b]

Durch die Verwendung des EXCI wird das CICS Transaction Gateway unter z/OS auf das ECI beschränkt, so dass die Funktionen des EPI für 3270-Transaktion und des ESI für Passwortmanagement nicht einsetzbar sind.

Die EXCI-Programmierschnittstelle ermöglicht Benutzern das Reservieren und Öffnen von Verbindungen *(Connections)* und Sitzungen *(Sessions)* zu einer CICS-Region sowie das Senden von DPL-Anfragen über diese. Die *MRO*-Funktionen *(Multiregion Operation)* der CICS-*IRC (Interregion Communication)* unterstützen diese Anfragen, und jeder MRO-Sitzung ist einer Verbindung zugeordnet.

### *4.1.1 Die Programmierschnittstellen*

Das EXCI unterstützt zwei Arten von Programmierschnittstellen: **EXCI CALL** und **EXEC CICS**.

*Die Schnittstelle **EXCI CALL***

Diese Schnittstelle besteht aus sechs Befehlen, durch die man Sitzungen mit CICS reservieren und öffnen, DPL-Anfragen über diese Sitzungen senden und nach dem erfolgreichen Bearbeiten dieser Anfragen die Sitzungen wieder schließen und freigeben kann. Die sechs Befehle lauten:

- **Initialize_User** (Benutzer initialisieren)
- **Allocate_Pipe** (Sitzung reservieren)
- **Open_Pipe** (Sitzung öffnen)
- **DPL** (DPL-Anfrage senden)
- **Close_Pipe** (Sitzung schließen)
- **Deallocate_Pipe** (Sitzung freigeben)

*Die Schnittstelle **EXEC CICS***

Bei dieser Schnittstelle kann man durch einen einzigen Befehl – **EXEC CICS LINK PROGRAM** – erreichen, dass alle sechs Befehle der Schnittstelle **EXCI CALL** nacheinander ausgeführt werden (Abbildung 4-2).

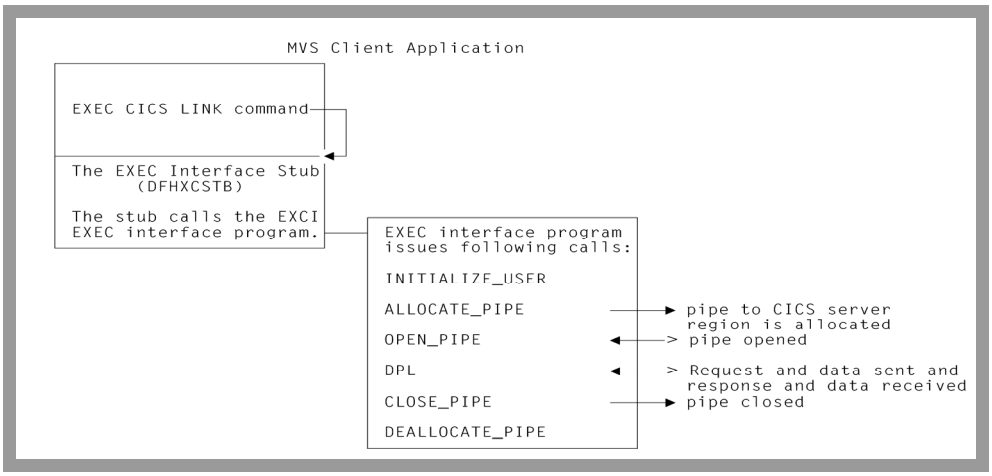


Abbildung 4-2: EXCI-Verbindungen über **EXEC CICS** [CS02a]

### *4.1.2 Definition von EXCI-Verbindungen (Connections) und -Sitzungen (Sessions)*

Damit Clientprogramme auf CICS-Serverprogramme zugreifen können, müssen EXCI-Verbindungen und -Sitzungen angelegt und installiert werden.

*Erstellen einer Verbindung (Connection)*

Zum Erstellen einer Verbindung ruft man innerhalb von CICS die *CEDA*-Transaktion auf und erreicht mit Hilfe des Befehls `DEFINE CONNECTION(JCOS) GROUP(MYEXCI)` den Bildschirm zur Festlegung der Verbindungsparameter für die neue Verbindung `JCOS` der Gruppe `MYEXCI` (Abbildung 4-3 und Abbildung 4-4).

```
 VIEW CONNECTION(JCOS) GROUP(MYEXCI)
 OBJECT CHARACTERISTICS                                  CICS RELEASE = 0620
  CEDA  View CONnection( JCOS )
   CONnection     : JCOS
   Group          : MYEXCI
   DEscription    :
  CONNECTION IDENTIFIERS
   Netname        : JGATE400
   INDsys         :
  REMOTE ATTRIBUTES
   REMOTESYSTem   :
   REMOTEName     :
   REMOTESYSNet   :
  CONNECTION PROPERTIES
   ACcessmethod   : IRc              Vtam | IRc | INdirect | Xm
   PRotocol       : Exci             Appc | Lu61 | Exci
   Conntype       : Specific         Generic | Specific
   SInglesess     : No               No | Yes
   DAtastream     : User             User | 3270 | SCs | STrfield | Lms
+  RECordformat   : U                U | Vb

                                                    SYSID=CICS APPLID=CICS

PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Abbildung 4-3: Verbindungsparameter – Bildschirm 1

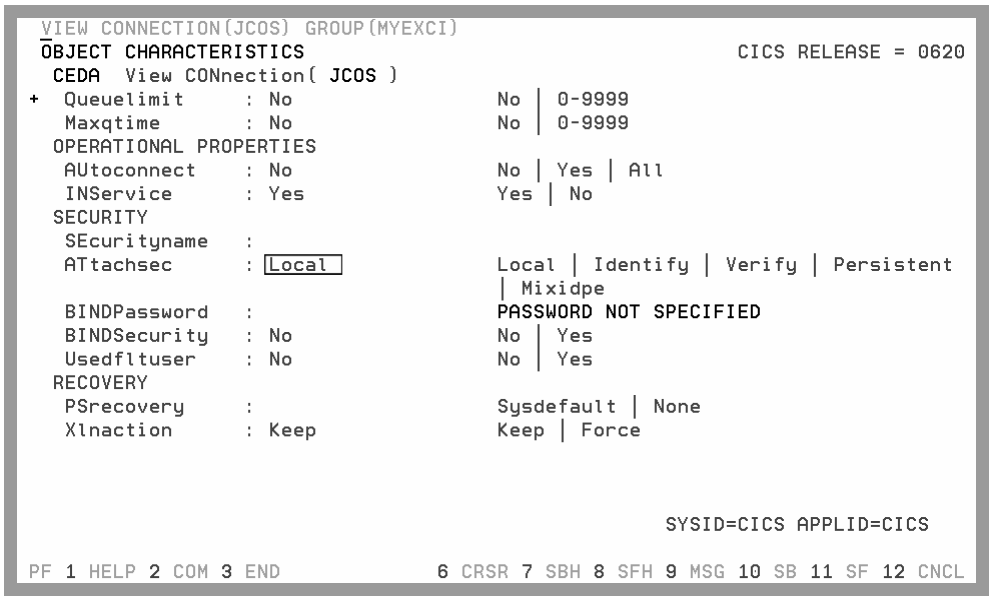


Abbildung 4-4: Verbindungsparameter – Bildschirm 2

Da ein `Netname` festgelegt ist, wird eine *spezifische* und keine *allgemeine* Verbindung angelegt. Das heißt, es handelt sich um eine MRO-Verbindung mit einer oder mehreren zugehörigen Sitzungen, die von einem einzelnen Benutzer eines Clientprogramms verwendet werden. Eine allgemeine Verbindung kann von mehreren Benutzern eingesetzt werden und darf nur einmal in einer CICS-Region vorhanden sein. Generell ist eine spezifische Verbindung zu bevorzugen, da die Verwaltung mehrerer Verbindungen ermöglicht und die Problembeseitigung vereinfacht wird.

Als `ACcessmethod` wird `IRC` und als Protokoll `EXCI` bestimmt. `ATtachsec` legt die Art der Auswertung der Benutzer-ID fest. Hier wird `Local` gewählt, so dass alle Anfragen unter dem Standard-CICS-Benutzer ablaufen. Im Gegensatz dazu wird im Fall von `Identify` bei jeder Anfrage die übergebene Benutzer-ID überprüft, wobei jedoch kein Kennwort erforderlich ist.

*Erstellen einer Sitzung (Session)*

Nachdem die Verbindung erfolgreich erstellt ist, muss man wenigstens eine zugehörige Sitzung definieren, ansonsten lässt sich die Gruppe gar nicht installieren. Dazu ruft man erneut die CEDA-Transaktion auf und erreicht mit Hilfe des Befehls `DEFINE SESSIONS(JCOS) GROUP(MYEXCI)` den Bildschirm zur Festlegung der Sitzungsparameter (Abbildung 4-5 und Abbildung 4-6).

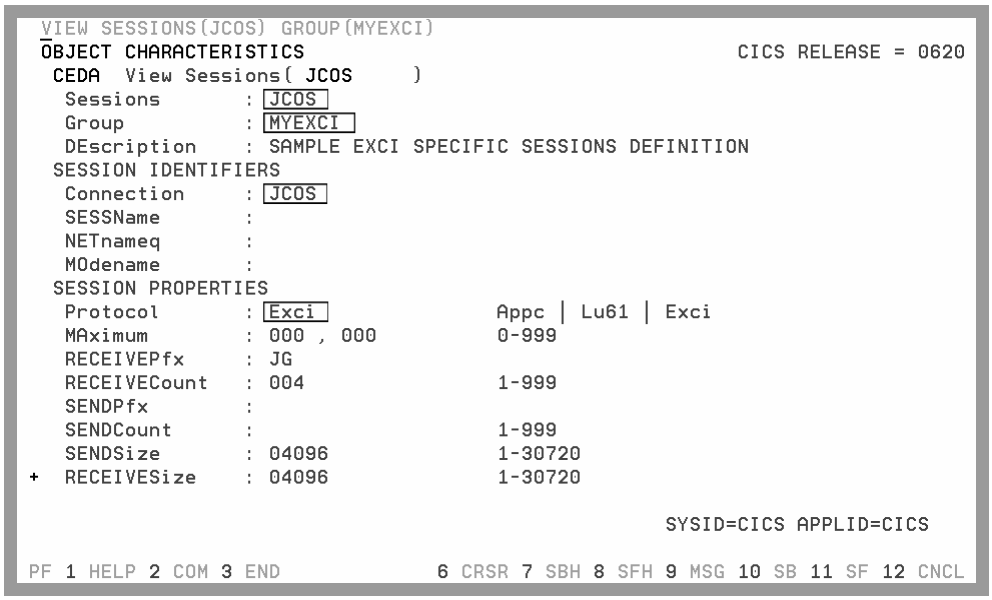


Abbildung 4-5: Sitzungsparameter – Bildschirm 1

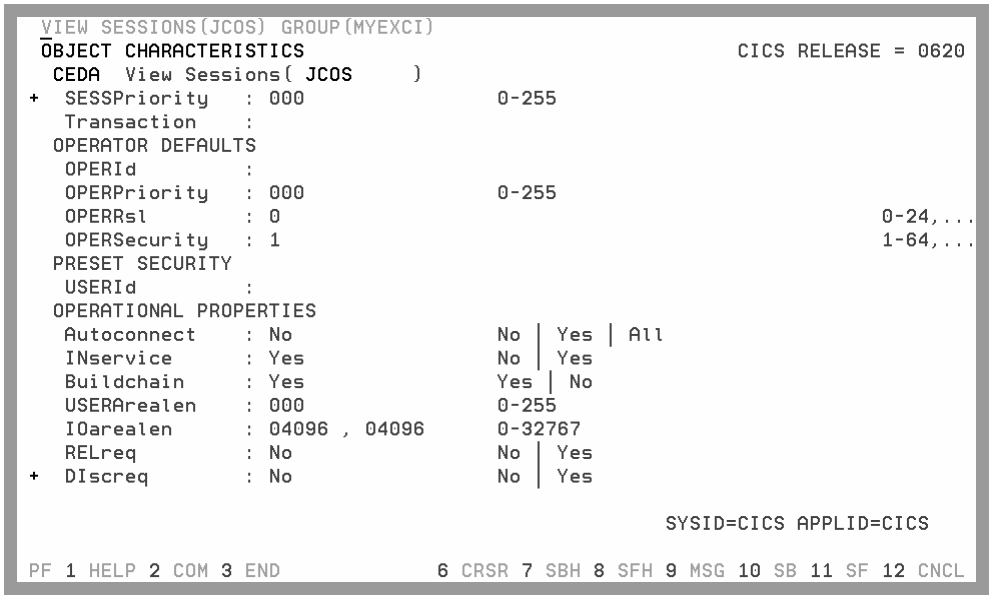


Abbildung 4-6: Sitzungsparameter – Bildschirm 2

Hierbei ist vor allem zu beachten, dass als `Connection` die zuvor erstellte Verbindung `JCOS` angegeben wird. Weiterhin ist als Protokoll `Exci` anzugeben.

### *4.1.3 Aktivieren der Interregion-Communication (IRC)*

Nachdem sowohl Verbindung als auch Sitzung definiert wurden, müssen beide installiert werden. Dazu wird der Befehl `CEDA INSTALL GROUP(MYEXCI)` ausgeführt (Abbildung 4-7).

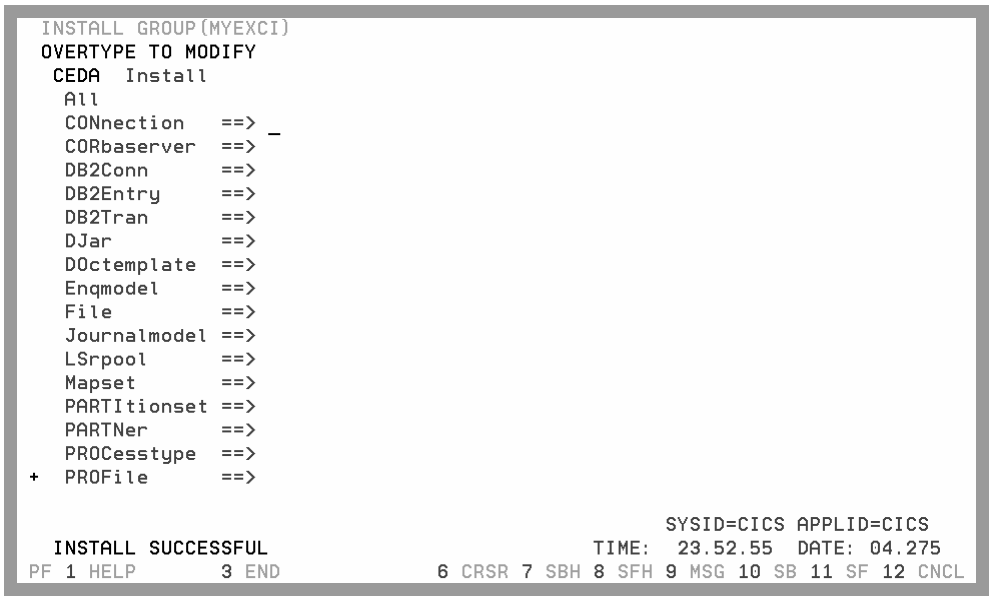


Abbildung 4-7: Installation der Verbindungs- und Sitzungsdefinitionen

Um die Interregion-Communiction (IRC) zu aktivieren, ist der Befehl `CEMT SET IRC OPEN` auszuführen. Dabei traten einige Probleme auf (Abbildung 4-8).

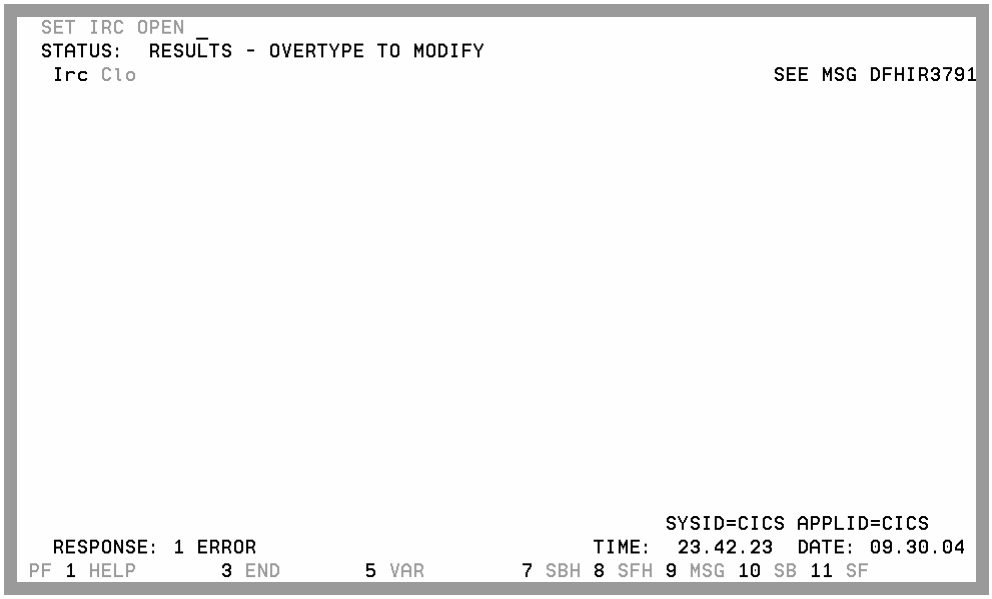


Abbildung 4-8: Die Fehlermeldung `DFHIR3791`

Die Interregion-Communication blieb geschlossen, und es erschien die Fehlermeldung `DFHIR3791`. Laut *CICS Messages and Codes* [CS02b] liegt das daran, dass in der *System Initialization Table (SIT)* von CICS der Parameter `ISC=NO` gesetzt wurde. Diese Tabelle legt die grundlegenden CICS-Parameter fest und wird beim Start von CICS geladen.

Die Prozedur zum Starten von CICS (`CICSA`) verwendet das Member `DFHSIT1` des Datasets `CICSTS22.SYSIN` zur Festlegung der SIT-Parameter. In dieses Member wurden die Einträge `ISC=YES` und `IRCSTRT=YES` eingefügt. Letzterer bewirkt, dass die Interregion-Communication gleich beim Start von CICS aktiviert wird.

Also wurde CICS mittels `CEMT PERFORM SHUTDOWN IMMEDIATE` beendet und in *SDSF (System Display and Search Facility)* mit Hilfe von `/S CICSA` neu gestartet. Die Ausgabe der Startprozedur deutete jedoch schon an, dass die Interregion-Communication erneut nicht gestartet wurde. Beim Versuch eines manuellen Starts mittels `CEMT SET IRC OPEN` erschien ebenfalls eine Fehlermeldung (Abbildung 4-9).

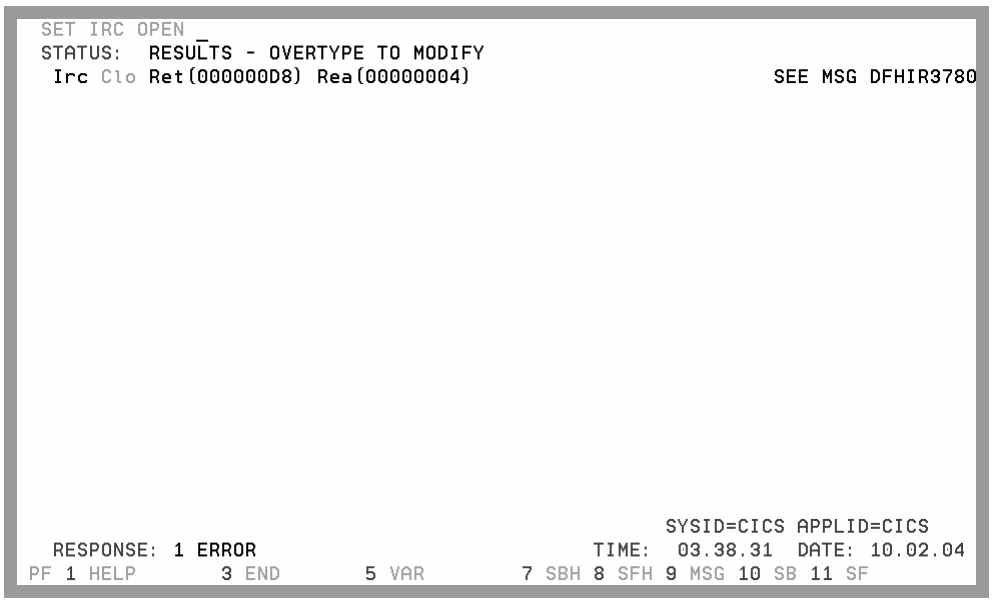


Abbildung 4-9: Die Fehlermeldung `DFHIR3780`

Das Handbuch [CS02b] war diesmal nicht sonderlich hilfreich und wies lediglich auf eine Reihe möglicher Fehler hin, die durch *Return-* und *Reasoncodes* näher beschrieben werden. Diese sind jedoch nur im Handbuch *CICS Data Areas* enthalten, das nicht frei verfügbar ist. Eine Suche auf der IBM-Webseite lieferte unter anderem eine Webseite [CEM], die der Fehlermeldung entsprach. Laut der Webseite lag die Ursache für die Fehlermeldung darin, dass CICS nicht als MVS-Subsystem definiert wurde, was im *Parmlib*-Member `IEFSSNxx` zu geschehen hat. Auf diesem System war dies das Member `USER.PARMLIB(IEFSSNLP)`, an dessen Ende drei neue Zeilen eingefügt wurden (Abbildung 4-10).

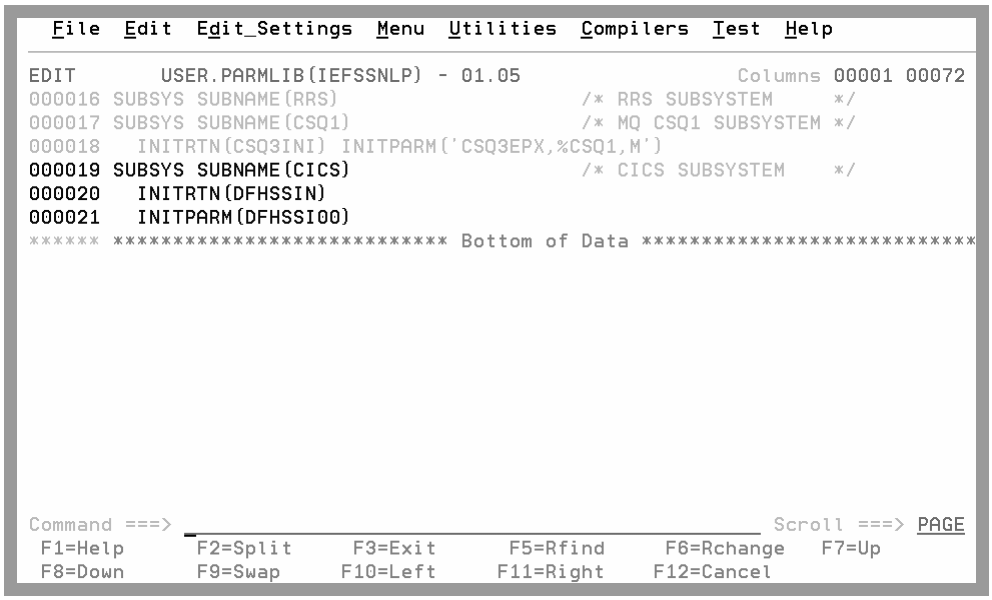


Abbildung 4-10: Das Parmlib-Member `IEFSSNLP`

Nun war ein Neustart des gesamten Systems, also ein *IPL (Initial Program Load)*, notwendig, um die Änderungen wirksam zu machen. Nachdem das System wieder lief und auch CICS neu geladen wurde, ließ die Ausgabe des Befehls `CEMT INQ IRC` darauf schließen, dass die Interregion-Communication erfolgreich gestartet wurde (Abbildung 4-11).

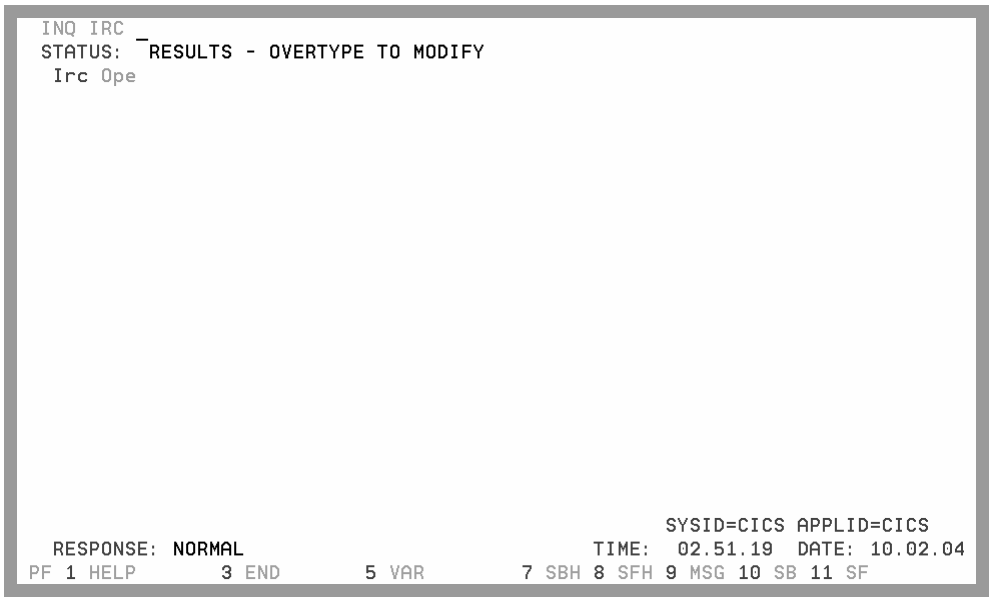


Abbildung 4-11: IRC erfolgreich gestartet

### *4.1.4 Testen der Verbindung zu CICS*

Nachdem nun die Interregion-Communication erfolgreich gestartet wurde, sollte die Verbindung zu CICS auch getestet werden. Dazu stellt CICS eine Reihe von Beispielprogrammen bereit: Mehrere CICS-Clientprogramme und ein CICS-Serverprogramm, deren Quellcode unter `CICSTS22.CICS.SDFHSAMP` zu finden ist. Die Clientprogramme liegen in Assembler, COBOL, PL/I und C vor, während das Serverprogramm in Assembler geschrieben wurde (Tabelle 4-1). Die Assembler-Clientprogramme liegen auch in übersetzter Form unter `CICSTS22.CICS.SDFHEXCI` vor.

| *Sprache* | *Name* | *Art des Programms* |
|---|---|---|
| Assembler | DFH$AXCC | Clientprogramm |
| Assembler | DFH$ATXC | Clientprogramm |
| Assembler | DFH$AXCS | Serverprogramm |
| COBOL | DFH$CXCC | Clientprogramm |
| PL/I | DFH$PXCC | Clientprogramm |
| C | DFH$DXCC | Clientprogramm |

Tabelle 4-1: EXCI-Beispielprogramme [CS02a]

Das Serverprogramm greift auf eine *VSAM*-Datei *(Virtual Storage Access Method)* namens `FILEA` aus der Gruppe `DFH$FILA` zu und gibt sie aus. Das Serverprogramm selbst befindet sich in der Gruppe `DFH$EXCI`. Also müssen beide Gruppen mit Hilfe des Befehls `CEDA INSTALL GROUP` installiert werden, damit die Beispielprogramme ausgeführt werden können. Als Test der Verbindung zu CICS wurde das Clientprogramm `DFH$AXCC` mit Hilfe eines JCL-Programms ausgeführt (Abbildung 4-12). Dabei ist besonders zu beachten, dass als Parameter des Programms die Anwendungs-ID (`CICS`) und der CICS-Benutzername (`CICSUSER`) angegeben werden müssen. Weiterhin ist es notwendig, `CICSTS22.CICS.SDFHEXCI` in die `STEPLIB`-Konkatenation einzufügen.

```
000100 //STARTAX  JOB (),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID,
000200 //         TIME=1440,REGION=100M
000300 //ASM      EXEC PGM=DFH$AXCC,PARM='CICS,CICSUSER'
000400 //STEPLIB  DD DSN=CICSTS22.CICS.SDFHEXCI,DISP=SHR
000500 //SYSPRINT DD SYSOUT=A
```

Abbildung 4-12: JCL zum Starten des EXCI-Beispielprogramms

Beim Ausführen des Jobs trat ein Fehler auf (Abbildung 4-13). Laut [CS02a] bedeutet `Reason-Code 423 SURROGATE_CHECK_FAILED`, dass die vom Clientprogramm übergebene Benutzer-ID die CICS-Surrogatüberprüfung nicht bestanden hat.

```
*===================== EXCI Sample Batch Client Program ======================*
*                                                                            *
*     Parameters: APPLID=CICS        USERID=CICSUSER                         *
*                                                                            *
*  EXEC Level Processor.                                                     *
*    Setting up the EXEC level call.                                         *
*    The Link Request has successfully completed.                            *
*    Server Response:                                                        *
*      The file is set to a browsable state.                                 *
*                                                                            *
*  CALL Level Processor.                                                     *
*    Initialise_User call complete.                                          *
*    Allocate_Pipe call complete.                                            *
*    Open_Pipe call complete.                                                *
*    The connection has been successful.                                     *
*      The target file follows:                                              *
*                                                                            *
*=========================== Top of File ====================================*
*      The Dpl request has failed.                                           *
*      Link return codes are:                                                *
*    Response = 00000012 Reason = 00000423 Subreason =  00000004             *
*      Dpl return codes are:                                                 *
*        Resp = 00000000  Resp2 = 00000000  Abend Code:                      *
*    >>>> Aborting further processing <<<<                                   *
*    Closing Dpl Request has been attempted.                                 *
*    Close_Pipe call complete.                                               *
*    Deallocate_Pipe call complete.                                          *
*                                                                            *
*=================== End of EXCI Sample Batch Client Program ================*
```

Abbildung 4-13: Fehler beim Ausführen des Beispielprogramms

Diese Überprüfung wird in der EXCI-Optionstabelle `DFHXCOPT` mit Hilfe des Parameters `SURROGCHK=YES` festgelegt. Diese Einstellung ist standardmäßig aktiv. Da derartige Sicherheitsfragen nicht Gegenstand dieser Arbeit sind, wurde die Surrogatüberprüfung der Benutzer-ID deaktiviert.

Die Quelle der `DFHXCOPT`-Standardtabelle liegt in `CICSTS22.CICS.SDFHSAMP` vor, das Lademodul in `CICSTS22.CICS.SDFHEXCI`. Die Quelldatei ist entsprechend zu editieren (Abbildung 4-14) und mit Hilfe der Prozedur `DFHAUPLE` aus `CICSTS22.XDFHINST` in ein Lademodul zu übersetzen (Abbildung 4-15).

```
DFHXCO TYPE=CSECT,
       TIMEOUT=0,              No timeout
       TRACE=OFF,              Only Exception trace entries
       TRACESZE=16,            16K trace table
       DURETRY=30,             Retry SDUMPS for 30 seconds
       TRAP=OFF,               DFHXCTRA - OFF
       GTF=OFF,                GTF - OFF
       MSGCASE=MIXED,          Mixed case messages
       CICSSVC=0,              EXCI will obtain CICS SVC number
       CONFDATA=SHOW,          Show user commarea data in trace
       SURROGCHK=NO            Perform surrogate-user check @P1C
END    DFHXCOPT
```

Abbildung 4-14: Ausschnitt der Quelldatei von DFHXCOPT

```
000100 //GSCHLUC  JOB (),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
000200 //MYLIBS1  JCLLIB ORDER=CICSTS22.XDFHINST
000300 //ASMTAB   EXEC DFHAUPLE,NAME=SDFHEXCI
000400 //ASSEM.SYSUT1 DD DSN=GSCHLU.CICS.EXCI(DFHXCOPT),DISP=SHR
```

Abbildung 4-15: JCL zum Übersetzen der Tabelle DFHXCOPT

Danach musste CICS neu gestartet werden, um die Änderungen wirksam werden zu lassen. Eine erneute Ausführung des Beispielprogramms führte zum Erfolg (Abbildung 4-16). Es wurde die VSAM-Datei FILEA ausgegeben.

Nach einem erfolgreichen Test mit Hilfe eines mit CICS mitgelieferten Beispielprogramms ist sichergestellt, dass man mit Hilfe der Interregion-Communication auf CICS zugreifen kann. Dies ist eine unabdingbare Notwendigkeit für den Betrieb des CICS Transaction Gateway, da dieses auf EXCI-Verbindungen und damit ebenfalls auf die Interregion-Communication aufsetzt. Daher kann man nun zur Installation des CICS Transaction Gateway übergehen.

```
*==================== EXCI Sample Batch Client Program ======================*
*                                                                            *
*     Parameters: APPLID=CICS        USERID=CICSUSER                         *
*                                                                            *
*  EXEC Level Processor.                                                     *
*    Setting up the EXEC level call.                                         *
*    The Link Request has successfully completed.                            *
*    Server Response:                                                        *
*      The file is set to a browsable state.                                 *
*                                                                            *
*  CALL Level Processor.                                                     *
*    Initialise_User call complete.                                          *
*    Allocate_Pipe call complete.                                            *
*    Open_Pipe call complete.                                                *
*    The connection has been successful.                                     *
*      The target file follows:                                              *
*                                                                            *
*========================== Top of File =====================================*
 000100S. D. BORMAN        SURREY, ENGLAND     3215677826 11 81$0100.11*********
 000102J. T. CZAYKOWSKI    WARWICK, ENGLAND    9835618326 11 81$1111.11*********
 000104M. B. DOMBEY        LONDON,ENGLAND      1284629326 11 81$0999.99*********
 000106A. I. HICKSON       CROYDON, ENGLAND    1948567326 11 81$0087.71*********
 000111ALAN TULIP          SARATOGA,CALIFORNIA 4612075301 02 74$0111.11*********
 000762SUSAN MALAIKA       SAN JOSE,CALIFORNIA 2231212101 06 74$0000.00*********
 000983J. S. TILLING       WASHINGTON, DC      3451212021 04 75$9999.99*********
 001222D.J.VOWLES          BOBLINGEN, GERMANY  7031555110 04 73$3349.99*********
 001781TINA J YOUNG        SINDELFINGEN,GERMANY7031999021 06 77$0009.99*********
 003210B.A. WALKER         NICE, FRANCE        1234567026 11 81$3349.99*********
 003214PHIL CONWAY         SUNNYVALE, CAL.     3411212000 06 73$0009.99*********
 003890BRIAN HARDER        NICE, FRANCE        0000000028 05 74$0009.99*********
 004004JANET FOUCHE        DUBLIN, IRELAND     7111212102 11 73$1259.99*********
 004445DR. P. JOHNSON      SOUTH BEND, S.DAK.  6121212026 11 81$0009.99*********
 004878ADRIAN JONES        SUNNYVALE, CALIF.   3221212010 06 73$5399.99*********
 005005A. E. DALTON        SAN FRANCISCO, CA.  0000000101 08 73$0009.99*********
 005444ROS READER          SARATOGA, CALIF.    6771212020 10 74$0809.99*********
 005581PETE ROBBINS        BOSTON, MASS.       4131212011 04 74$0259.99*********
 006016SIR MICHAEL ROBERTS NEW DELHI, INDIA    7033121121 05 74$0009.88*********
 006670IAN HALL            NEW YORK, N.Y.      2121212031 01 75$3509.88*********
 006968J.A.L. STAINFORTH   WARWICK, ENGLAND    5671382126 11 81$0009.88*********
 007007ANDREW WHARMBY      STUTTGART, GERMANY  7031100010 10 75$5009.88*********
 007248M. J. AYRES         REDWOOD CITY, CALF. 3331212111 10 75$0009.88*********
 007779MRS. A. STEWART     SAN JOSE, CALIF.    4151212003 01 75$0009.88*********
 009000P. E. HAVERCAN      WATERLOO, ONTARIO   0987654321 01 75$9000.00*********
 100000M. ADAMS            TORONTO, ONTARIO    0341512126 11 81$0010.00*********
 111111C. BAKER            OTTAWA, ONTARIO     5121200326 11 81$0011.00*********
 200000S. P. RUSSELL       GLASGOW,  SCOTLAND  6373829026 11 81$0020.00*********
 222222DR E. GRIFFITHS     FRANKFURT, GERMANY  2003415126 11 81$0022.00*********
 300000V. J. HARRIS        NEW YORK, U.S.      6473980126 11 81$0030.00*********
 333333J.D. HENRY          CARDIFF, WALES      7849302026 11 81$0033.00*********
 400000C. HUNT             MILAN, ITALY        2536373826 11 81$0040.00*********
 444444D. JACOBS           CALGARY, ALBERTA    7788982026 11 81$0044.00*********
 500000P. KINGSLEY         MADRID, SPAIN       4445464026 11 81$0000.00*********
 555555S.J. LAZENBY        KINGSTON, N.Y.      3994442026 11 81$0005.00*********
 600000M.F. MASON          DUBLIN, IRELAND     1239878026 11 81$0010.00*********
 666666R. F. WALLER        LA HULPE, BRUSSELS  4298384026 11 81$0016.00*********
 700000M. BRANDON          DALLAS, TEXAS       5798432026 11 81$0002.00*********
 777777L.A. FARMER         WILLIAMSBURG, VIRG. 9187613126 11 81$0027.00*********
 800000P. LUPTON           WESTEND, LONDON     2423338926 11 81$0030.00*********
 888888P. MUNDY            NORTHAMPTON, ENG.   2369163926 11 81$0038.00*********
 900000D.S. RENSHAW        TAMPA, FLA.         3566812026 11 81$0040.00*********
 999999ANJI STEVENS        RALEIGH, N.Y.       8459163926 11 81$0049.00*********
*========================== End of File =====================================*
*                                                                            *
*    Closing Dpl Request has been attempted.                                 *
*    Close_Pipe call complete.                                               *
*    Deallocate_Pipe call complete.                                          *
*                                                                            *
*================== End of EXCI Sample Batch Client Program ==================*
```

Abbildung 4-16: Ausgabe des Beispielprogramms `DFH$AXCC`

## 4.2 Einrichtung des CICS Transaction Gateway

Dieser Abschnitt beschäftigt sich mit der Installation und Konfiguration des CICS Transaction Gateway für z/OS. Dieses läuft in der Umgebung USS oder OMVS ab, so dass sowohl unter MVS als auch unter USS/OMVS verschiedene Tätigkeiten auszuführen sind. Weiterhin werden von mehreren Plattformen aus Java-Beispielprogramme (`EciB1` und `EciI1`) ausgeführt, die mit Hilfe des CICS Transaction Gateway auf CICS-Beispiele (`EC01` und `EC02`) zugreifen, um die Funktionstüchtigkeit der Konfiguration zu überprüfen (Abbildung 4-17).

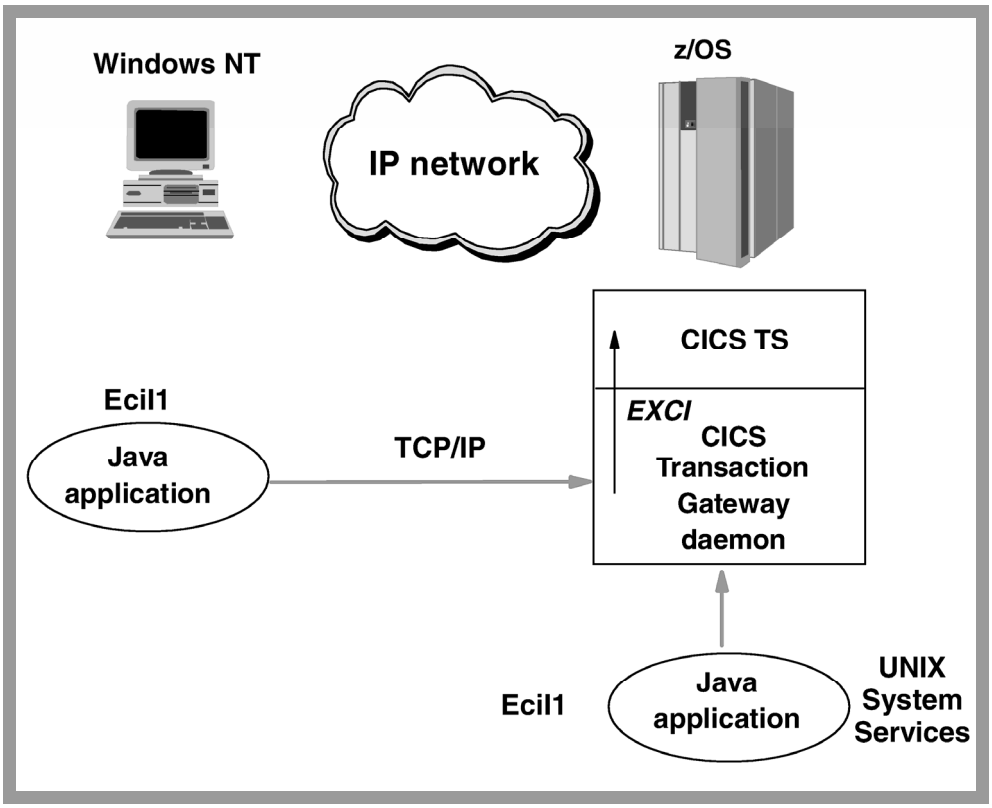


Abbildung 4-17: Umgebung zum Test der CTG-Konfiguration
[CG02b]

### *4.2.1 Voraussetzungen*

Das CICS Transaction Gateway für z/OS verwendet EXCI-Verbindungen, um auf CICS zuzugreifen. Einzelheiten zur Einrichtung solcher Verbindungen sind in *Abschnitt 4.1 EXCI-Verbindungen zu CICS* zu finden.

### *4.2.2 Installation des CICS Transaction Gateway*

Das CICS Transaction Gateway für z/OS ist in einer einzelnen Datei namens `ctg-500m.tar.Z` enthalten. Es wird angenommen, dass diese Datei z.B. per *FTP (File Transfer*

*Protocol)* auf den z/OS-Rechner übertragen wurde und im HFS-Verzeichnis `/tmp` vorliegt. Wenn dies der Fall ist, kann die Installation beginnen.

*Erstellen des HFS-Datasets*

Zur Aufnahme des CICS Transaction Gateway muss ein neues HFS-Dataset erstellt und nach `/usr/lpp/ctg500` *gemountet* werden, wobei der *Mountpoint* auch anders gewählt werden kann.

Zum Allokieren dieses Datasets wird die ISPF-Option 3.2 gewählt, der gewünschte Name ausgewählt und `A` als Option eingetragen. Daraufhin erscheint der Bildschirm `Allocate New Data Set` (Abbildung 4-18).

```
 Menu  RefList  Utilities  Help
                          Allocate New Data Set
                                                              More:     +
Data Set Name  . . . : CTGUSER.CTG500.HFS

Management class . . . ________       (Blank for default management class)
Storage class  . . . . ________       (Blank for default storage class)
 Volume serial . . . . HFS001         (Blank for system default volume) **
 Device type . . . . . ________       (Generic unit or device address) **
Data class . . . . . . ________       (Blank for default data class)
 Space units . . . . . CYLINDER       (BLKS, TRKS, CYLS, KB, MB, BYTES
                                       or RECORDS)
 Average record unit   _               (M, K, or U)
 Primary quantity  . . 250            (In above units)
 Secondary quantity    250            (In above units)
 Directory blocks  . . 0              (Zero for sequential data set) *
 Record format . . . . U
 Record length . . . . 0
 Block size  . . . . . 0
 Data set name type  : HFS            (LIBRARY, HFS, PDS, or blank)  *
Command ===> _____________________________________________________________
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Abbildung 4-18: Bildschirm zur Erstellung des HFS-Datasets

Die Werte `Primary Quantity` und `Secondary Quantity` sind sehr großzügig gewählt worden, laut Dokumentation [CG02b] genügen schon jeweils 50 Zylinder. Als Typ wurde `HFS` gewählt. Mit Hilfe der Eingabetaste wird die Erstellung des Datasets ausgelöst.

Nach der Erstellung des HFS-Datasets muss dieses gemountet werden. Dies kann zum Beispiel mit dem TSO-Befehl **MOUNT** oder mit **ISHELL** geschehen. Zuerst muss jedoch unter OMVS ein entsprechendes Verzeichnis erstellt werden. Dazu wechselt man ins Verzeichnis `/usr/lpp` und führt den Befehl **su** aus, um Superuser-Berechtigungen zu erhalten. Dann wird das Verzeichnis `ctg500` erstellt und mit dem Befehl **ls** angezeigt. Die genaue Befehlsabfolge ist in Abbildung 4-19 enthalten.

```
GSCHLU:/u/gschlu: >cd /usr/lpp
GSCHLU:/V1R4S/usr/lpp: >su
GSCHLU:/V1R4S/usr/lpp: >mkdir ctg500
GSCHLU:/V1R4S/usr/lpp: >ls -l ctg500
drwxr-xr-x  10 BPXOINIT ADMIN       8192 Sep 25 14:30 ctg500/
```

Abbildung 4-19: Anlegen des Mountpoints

In diesem Fall trat jedoch das Problem auf, dass das Verzeichnis `ctg500` nicht erstellt werden konnte, da das Dataset, in dem `/usr/lpp` enthalten war, nicht beschrieben werden konnte. Daher mussten die Attribute des Datasets von `/usr/lpp` entsprechend geändert werden, woraufhin die Erstellung von `ctg500` möglich war.

Nun kann der tatsächliche Mountvorgang durchgeführt werden. Dazu verwendet man die ISPF-Option 6 (`Command`) und gibt den Befehl **ISHELL** ein. Daraufhin wechselt man ins Menü `File_systems` und wählt Punkt 3 (`Mount`). Es erscheint der Bildschirm aus Abbildung 4-20, dessen Einstellungen mit der Eingabetaste übernommen werden können.

```
 File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
                          Mount a File System

 Mount point:
                                                              More:      +
    /usr/lpp/ctg500
    ____________________________________________________________________

 File system name   ctguser.ctg500.hfs_____________________________________
 File system type   hfs_____  New owner  . . . . . ________
 Owning system  . . _________  Character Set ID . . _____

 Select additional mount options:
 _  Read-only file system          _  Set automove attribute...
 _  Ignore SETUID and SETGID       _  Text conversion enabled
 _  Bypass security

 Mount parameter:
 _____________________________________________________________________

  F1=Help        F3=Exit        F4=Name        F6=Keyshelp   F12=Cancel
F10=Actions  F11=Command  F12=Cancel
```

Abbildung 4-20: Bildschirm zum Mounten eines Dateisystems

Wenn man möchte, dass das gemountete HFS-Dateisystem auch beim nächsten IPL zur Verfügung steht, sollte man das Parmlib-Member `BPXPRMxx` ändern. Dazu fügt man zum Beispiel den Eintrag aus Abbildung 4-21 ein.

```
000124 MOUNT    FILESYSTEM('CTGUSER.CTG500.HFS')
000125          TYPE(HFS)
000126          MODE(RDWR)
000127          MOUNTPOINT('/usr/lpp/ctg500')
```

Abbildung 4-21: Eintrag ins Parmlib-Member `BPXPRMxx`

*Entpacken der CTG-Datei*

Wie schon erwähnt liegt die Installationsdatei des CICS Transaction Gateway `ctg-500m.tar.Z` im Verzeichnis `/tmp` vor. Diese Datei muss zuerst dekomprimiert und dann ins Verzeichnis `/usr/lpp/ctg500` entpackt werden. Die genaue Befehlsabfolge ist in Abbildung 4-22 enthalten.

```
cd /tmp
uncompress -v /tmp/ctg-500m.tar.Z
cd /usr/lpp/ctg500
tar -xopfv /tmp/ctg-500m.tar
```

Abbildung 4-22: Entpacken der CTG -Datei

Damit sind die entsprechenden Unterverzeichnisse erstellt, und das CICS Transaction Gateway liegt nun im Verzeichnis `/usr/lpp/ctg`.

*Ausführung des Installationsskripts*

Als nächstes muss das Installationsskript des CICS Transaction Gateway ausgeführt werden. Dazu wird zunächst das Verzeichnis `/usr/lpp/ctg500` betrachtet (Abbildung 4-23).

```
GSCHLU:/V1R4S/usr/lpp/ctg500: >ls -l
total 72
drwxr-xr-x   9 BPXOINIT ADMIN       8192 Jul 15  2002 ctg
-rwxr-xr-x   1 BPXOINIT ADMIN         66 Jul 15  2002 ctg_install
-rwxr-xr-x   1 BPXOINIT ADMIN         20 Jul 15  2002 ctginstall
-rw-r--r--   1 BPXOINIT ADMIN       1675 Jul 15  2002 ctginstall.readme
-rwxr-xr-x   1 BPXOINIT ADMIN         28 Jul 15  2002 ctginstall_IBM-930
-rwxr-xr-x   1 BPXOINIT ADMIN         28 Jul 15  2002 ctginstall_IBM-935
-rwxr-xr-x   1 BPXOINIT ADMIN         28 Jul 15  2002 ctginstall_IBM-937
-rwxr-xr-x   1 BPXOINIT ADMIN         28 Jul 15  2002 ctginstall_IBM-939
```

Abbildung 4-23: Das CTG-Programmverzeichnis

Es muss das Skript `ctginstall` ausgeführt und `1` eingegeben werden. Daraufhin erscheint die Lizenzvereinbarung (Abbildung 4-24). Es muss `3` eingegeben werden, um diese zu akzeptieren und die Installation abzuschließen.

```
GSCHLU:/V1R4S/usr/lpp/ctg500: >ctginstall
To install CICS Transaction Gateway you must accept the following license
agreement.
If the license file does not display
correctly, try restarting the
installation using the command:
'/V1R4S/usr/lpp/ctg500/ctg/bin/ctgsetup
40'.
Enter 1 to view the license, or 2 to quit.
 ===> 1_
1
>>> Beginning of the License File >>>
International Program License Agreement


Part 1 - General Terms


PLEASE READ THIS AGREEMENT CAREFULLY BEFORE USING THE PROGRAM. IBM WILL LICENSE
THE PROGRAM TO YOU ONLY IF YOU FIRST ACCEPT THE TERMS OF THIS AGREEMENT. BY
USING THE PROGRAM YOU AGREE TO THESE TERMS. IF YOU DO NOT AGREE TO THE TERMS OF
THIS AGREEMENT, PROMPTLY RETURN THE UNUSED PROGRAM TO THE PARTY (EITHER IBM OR
ITS RESELLER) FROM WHOM YOU ACQUIRED IT TO RECEIVE A REFUND OF THE AMOUNT YOU
PAID.

The Program is owned by International Business Machines Corporation or one of
its subsidiaries (IBM) or an IBM supplier, and is copyrighted and licensed, not
sold.
--------------------------------------------------------------------------------
Enter 1 to page down, 2 to page up, 3 to accept or 4 to decline.
 ===> 3_
Wait while the installation continues.
The installation is complete.
License files can be viewed using the command:
'/V1R4S/usr/lpp/ctg500/ctg/bin/ctgbrowse'
CTG6112W To configure CICS Transaction Gateway for use with RACF userid and
password authentication, a user with READ access to the BPX.FILEATTR.PROGCTL
FACILITY class will need to run the command 'extattr +p
/V1R4S/usr/lpp/ctg500/ctg/bin/lib*.so'.
CTG6112W To configure CICS Transaction Gateway for use with RACF userid and
password authentication, a user with READ access to the BPX.FILEATTR.PROGCTL
FACILITY class will need to run the command 'extattr +p
/V1R4S/usr/lpp/ctg500/ctg/bin/SECURES'.
CTG6113W To configure CICS Transaction Gateway for use with z/OS Automatic
Restart Manager, a user with authority to update RACF group BPX.FILEATTR.APF
will need to run the command 'extattr +a /V1R4S/usr/lpp/ctg500/ctg/bin/ctgarm'.
To view the PDF documentation, you should obtain a copy of Adobe Acrobat Reader
from www.adobe.com.
Now create 'CTG.INI' and 'ctgenvvar' by copying 'CTGSAMP.INI' and
'ctgenvvarsamp' and editing them to configure the CICS Transaction Gateway.
GSCHLU:/V1R4S/usr/lpp/ctg500: >
 ===> _
```

Abbildung 4-24: Die Lizenzvereinbarung des CICS Transaction Gateway

Als Ergebnis des Installationsskripts ist die Verzeichnisstruktur des CICS Transaction Gateway erstellt worden (Abbildung 4-25). Die Installation ist damit erfolgreich abgeschlossen.

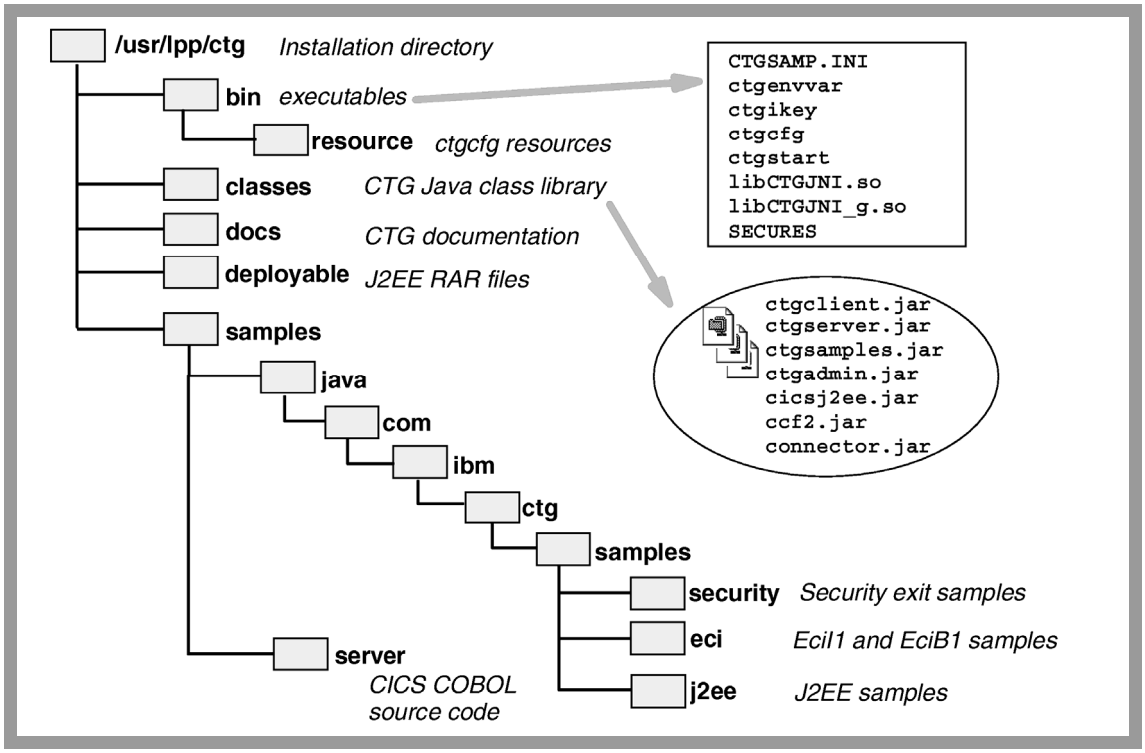


Abbildung 4-25: Der Verzeichnisbaum des CICS Transaction Gateway [CG02b]

### *4.2.3 Konfiguration und Start*

Die Konfigurationsparameter des CICS Transaction Gateway werden in zwei Dateien im CTG-Unterverzeichnis `bin` festgelegt: `ctgenvvar` und `CTG.INI`. Nach der Installation sind diese Dateien jedoch noch nicht vorhanden. Stattdessen enthält das Verzeichnis `bin` zwei Beispieldateien namens `ctgenvvarsamp` und `CTGSAMP.INI`. Diese beiden Dateien sollte man mit Hilfe von **cp** zuerst nach `ctgenvvar` und `CTG.INI` kopieren.

Nun gibt es mehrere Möglichkeiten, die Konfigurationsparameter des CICS Transaction Gateway festzulegen. Eine davon ist das Konfigurationstool, das mit Hilfe des Befehls **ctgcfg** gestartet wird und die Konfigurationsdateien verändert. Vorher muss man jedoch mit dem Befehl **export DISPLAY** die Ausgabe auf den eigenen PC umleiten, und auf diesem muss ein *X-Windows-Server* gestartet sein. Genauere Anweisungen dazu sind in [CG02a] zu finden.

Die dadurch hergestellte Ausgabe der X-Windows-Fenster auf dem eigenen PC funktionierte zwar, der Aufbau war jedoch so langsam, dass ein ernsthafter Einsatz des Konfigureationstools nicht in Frage kam. Daher wurden die Konfigurationsdateien des CICS Transaction Gateway von Hand bearbeitet.

*Die Datei CTG.INI*

Die Konfigurationsdatei `CTG.INI` kann im Prinzip unverändert bleiben. Die wichtigste Einstellung ist die des TCP-Protokolls (Abbildung 4-26).

```
EDIT       /V1R4S/usr/lpp/ctg500/ctg/bin/CTG.INI          Columns 00001 00072
000196 #    Enables Client Authentication for the specified secure protocol.
000197 #
000198 ###############################################################################
000199 # 3. PROTOCOL SETTINGS
000200 #
000201 protocol@tcp.handler=com.ibm.ctg.server.TCPHandler
000202 protocol@tcp.parameters=port=2006;\
000203                          sotimeout=1000;\
000204                          connecttimeout=2000;\
000205                          idletimeout=600000;\
000206                          pingfrequency=60000
000207
000208 #protocol@http.handler=com.ibm.ctg.server.HttpHandler
000209 #protocol@http.parameters=port=8080;\
000210 #                          sotimeout=1000;\
000211 #                          connecttimeout=2000;\
000212 #                          idletimeout=120000;\
000213 #                          requiresecurity
```

Abbildung 4-26: Die Datei `CTG.INI`

Der dargestellte Eintrag legt fest, dass der Gateway-Daemon den TCP-Port `2006` abhören soll, um eingehende Anforderungen zu bearbeiten. Diese Einstellung ist jedoch schon standardmäßig vorhanden. Alle anderen Einstellungen können ebenfalls so belassen werden.

*Die Datei ctgenvvar*

Beim Konfigurationsskript `ctgenvvar` sind deutlich mehr Einstellungen vorzunehmen als in `CTG.INI`. Hier werden dem CICS Transaction Gateway vor allem Informationen über den Zugriff auf CICS sowie über Sicherheit und über den verwendeten Transaktionsmanager mitgeteilt. Tabelle 4-2 enthält die Variablen mit den festgelegten Werten sowie zugehörigen Beschreibungen.

| *Variable* | *Wert* | *Beschreibung* |
|---|---|---|
| RRM_NAME | "CCL.CTG" | Dies ist der Name, den das CICS Transaction Gateway für transaktionale EXCI-Aufrufen in RRS registriert. Die Endung, `.IBM.UA` wird automatisch angehängt. Für transaktionale EXCI-Aufrufe ist diese Einstellung notwendig. |
| EXCI_LOADLIB | "CICSTS22.CICS. SDFHEXCI" | Dies ist der Name der Bibliothek mit den CICS-EXCI-Modulen. |
| DFHJVPIPE | "JGATE400" | Dieser Wert muss dem `Netname` der angelegten spezifischen EXCI-Verbindung entsprechen. Trägt man keinen Wert ein, wird die allgemeine Verbindung verwendet. |
| DFHJVSYSTEM_00 | "CICS-Default Example Server" | Die Syntax dieser Variable lautet: `DFHJSYSTEM_nn = aaaaaaaa-Literal` [CG02b]:<br>• `nn` = `00` bis `99`<br>• `aaaaaaaa` = Die ID der CICS-Region<br>• `Literal` = Eine Beschreibung der CICS-Region |
| AUTH_USERID_ PASSWORD | "" | Dieser Wert legt fest, ob bei ECI-Aufrufen der Benutzer-ID und das Kennwort von RACF überprüft werden sollen. Diese Überprüfung wurde deaktiviert, da Sicherheitsfragen nicht Gegenstand dieser Arbeit sind. |

Tabelle 4-2: Variablen von `ctgenvvar`

Damit auch transaktionale Programme funktionieren, muss noch ein seltsamer Fehler des CICS Transaction Gateway in Version 5.00 behoben werden. Wie in Tabelle 4-2 beschrieben, wurde als `RRM_NAME` der Wert `CCL.CTG` gewählt, so dass der gesamte Name des Ressourcenmanagers `CCL.CTG.IBM.UA` lautet. Dieser wird jedoch nicht ordnungsgemäß gesetzt, da im Skript `ctgenvvar` ein Fehler vorliegt, der in späteren CTG-Versionen behoben wurde [CGE]. Es müssen in einer `if`-Abfrage zwei Leerzeichen eingefügt werden, damit diese korrekt ausgewertet wird (Abbildung 4-27).

**Local fix**

```
In CTGENVVAR change the line:
if    $CTG_RRMNAME=".IBM.UA"
to
if    $CTG_RRMNAME = ".IBM.UA"
```

Abbildung 4-27: Fehlerbehebung in `ctgenvvar` [CGE]

Nach dieser Änderung wird der Name des Ressourcenmanagers korrekt übergeben, und auch transaktionale Programme sollten ausführbar sein, was in einem späteren Abschnitt getestet werden soll.

*JCL-Prozedur zum Starten des Gateway-Daemon*

Zum Bearbeiten von Anfragen von anderen Rechnern aus soll der Gateway-Daemon gestartet werden. Für rein lokale Anfragen ist dies jedoch nicht notwendig. Zum Start des Gateway-Daemon soll eine JCL-Prozedur eingesetzt werden, die das Skript `ctgstart` aus dem Verzeichnis `bin` aufruft (Abbildung 4-28). `ctgstart` verwendet wiederum die im vorherigen Abschnitt erstellten Konfigurationsdateien.

```
000100 //CTG500   PROC
001200 //*
001300 //* Main step: run CTG
001400 //*
001500 //CTG      EXEC  PGM=BPXBATCH,REGION=0M,TIME=NOLIMIT,
001700 //         PARM='SH /usr/lpp/ctg500/ctg/bin/ctgstart -noinput'
001800 //STDOUT   DD    PATH='/usr/lpp/ctg500/ctg/logs/ctgstart.out',
001900 //         PATHOPTS=(OWRONLY,OCREAT,OAPPEND),
001910 //         PATHMODE=(SIRWXU,SIRWXG,SIROTH)
002000 //STDERR   DD    PATH='/usr/lpp/ctg500/ctg/logs/ctgstart.err',
002100 //         PATHOPTS=(OWRONLY,OCREAT,OAPPEND),
002110 //         PATHMODE=(SIRWXU,SIRWXG,SIROTH)
```

Abbildung 4-28: JCL-Prozedur zum Starten des Gateway-Daemon

Das Member mit dieser JCL-Prozedur trägt den Namen `CTG500` und wird im Dataset `USER.PROCLIB` erstellt. Es wird das Programm `BPXBATCH` aufgerufen, das den Zugriff auf Programme und Dateien unter USS/OMVS ermöglicht. Man kann die MVS-Standarddateien `STDIN`, `STDOUT` und `STDERR` allokieren, wobei es sich hier auch um HFS-Dateien handeln muss. In diesem Fall werden die Ausgabedateien in das CTG-Unterverzeichnis `logs` umgeleitet, das jedoch zuerst mit **mkdir logs** erstellt werden muss.

Da das aufgerufene Startskript `ctgstart` Java-Aufrufe verwendet, muss für die OMVS-Umgebung der Java-Pfad gesetzt werden. Der Einfachheit halber wurde an dieser Stelle der Pfad zu Java in der Datei `/etc/profile` festgelegt, so dass allen Benutzer automatisch Java zur Verfügung steht. In diesem Fall musste folgende Zeile eingefügt werden:

```
export PATH=/usr/lpp/java/IBM/J1.3/bin:$PATH
```

Somit wird der zuvor vorhandene Pfad um den Java-Pfad erweitert.

Nun wechselt man in SDSF und startet mittels **/S CTG500** die soeben angelegte Prozedur. Mit **ST** kann man den Status der gestarteten Jobs einsehen (Abbildung 4-29).

```
SDSF STATUS DISPLAY ALL CLASSES                          LINE 35-51 (729)
NP   JOBNAME  JobID    Owner    Prty Queue      C  Pos  SAff  ASys Status
     IMS81CR1 STC04940 START2     15 EXECUTION           SYS1  SYS1
     IMS81DL1 STC04941 START2     15 EXECUTION           SYS1  SYS1
     IMS81RC1 STC04942 START2     15 EXECUTION           SYS1  SYS1
     BBOASR2  STC05161 CBACRU2    15 EXECUTION           SYS1  SYS1
     BBOASR2S STC05162 CBASRU2    15 EXECUTION           SYS1  SYS1
     HTTPD1   STC05164 WEBSRV     15 EXECUTION           SYS1  SYS1
     BPXAS    STC05413 START2     15 EXECUTION           SYS1  SYS1
     CTG500   STC05426 START2     15 EXECUTION           SYS1  SYS1
     $MASCOMM STC00001            15 PRINT       A     1
```

Abbildung 4-29: Status der CTG-Startprozedur

Die Prozedur wurde unter dem Benutzernamen `START2` gestartet und bleibt aktiv, da es sich um einen Prozess mit langer Lebensdauer handelt. Die Ausgabe der Prozedur steht in der Datei `/usr/lpp/ctg500/ctg/logs/ctgstart.out` (Abbildung 4-30).

```
CTG6111I File 'ctgenvvar' found. Using the configuration in script 'ctgenvvar'
to start up the application.
CICS Transaction Gateway, Version 5.0.0, 5724-D12. Build Level c500-20020715.
(C) Copyright IBM Corporation 1999, 2002. All rights reserved.
CCL8400I: Using ini file /V1R4S/usr/lpp/ctg500/ctg/bin/CTG.INI.
CCL6577I: Java version is 1.3.1.
CCL6502I: Initial ConnectionManagers = 10, Maximum ConnectionManagers = 90,
CCL6502I: Initial Workers = 10, Maximum Workers = 90, tcp: Port = 2006
CCL6574I: Connection logging has been disabled.
CCL6505I: Successfully created the initial ConnectionManager and Worker threads.
CCL6524I: Successfully started handler for the tcp: protocol.
```

Abbildung 4-30: Die Ausgabe des CTG-Startskripts

Man erkennt, dass die Konfigurationsdateien `ctgenvvar` und `CTG.INI` sowie die Java-Umgebung gefunden wurden. Es ist ebenfalls zu sehen, dass der Gateway-Daemon erfolgreich gestartet wurde und den TCP-Port 2006 abhört. Um dies zu testen, wird der Befehl **onetstat** verwendet, durch den der Netzwerkstatus des lokalen Hosts angezeigt wird.

```
MVS TCP/IP onetstat CS V1R4       TCPIP Name: TCPIP              04:32:49
User Id  Conn     Local Socket          Foreign Socket          State
-------  ----     ------------          --------------          -----
BBOASR2A 0000197E 0.0.0.0..1081         0.0.0.0..0              Listen
BBOASR2A 0000198D 139.18.12.91..1083    139.18.12.91..900       Establsh
BBOASR2S 00001993 139.18.12.91..1084    139.18.12.91..1389      Establsh
BBOLDAP  0000036D 139.18.12.91..1389    217.187.225.75..3019    Establsh
BPXOINIT 00000009 0.0.0.0..10007        0.0.0.0..0              Listen
CTG5002  00003AAA 0.0.0.0..2006         0.0.0.0..0              Listen
DAEMON01 000002FE 139.18.12.91..5550    217.187.225.75..4188    Establsh
```

Abbildung 4-31: Ausgabe des Befehls **onetstat**

Die Ausgabe des Befehls **onetstat** zeigt, dass der Gateway-Daemon aktiv ist und auch tatsächlich den Port 2006 abhört (Abbildung 4-31). Nun soll die bislang erstellte Konfiguration getestet werden.

### *4.2.4 Test der Konfiguration*

Zum Testen der Konfiguration sind im CTG-Unterverzeichnis `samples/java` mehrere Java-Beispielprogramme im Quellcode vorhanden. Diese greifen auf CICS-COBOL-Programme `EC01` und `EC02` zu, deren Quellcode im CTG-Unterverzeichnis `samples/server` in Form zweier Dateien namens `ec01.ccp` und `ec02.ccp` vorliegt. Diese Quelldateien müssen übersetzt und die dadurch entstandenen Programme in CICS installiert werden. Daraufhin können die beiden Java-Beispiele `EciB1` und `EciI1` mit Hilfe kurzer Shellskripte ausgeführt werden.

Bei der Ausführung dieser Java-Programme tritt jedoch das Problem auf, dass die von CICS zurückgelieferten Daten im EBCDIC-Format vorliegen und in Java nicht lesbar sind, da dieses mit *Unicode* arbeitet. Daher muss CICS dazu veranlasst werden, die EBCDIC-Daten der Beispielprogramme nach *ASCII* zu konvertieren, da Java mit dieser Kodierung umgehen kann. Diese Konvertierungseinstellungen sind nur für Java unter USS/OMVS notwendig, da die späteren J2EE-Beispiele die Daten selbst konvertieren können. Die notwendigen Schritte zur Einrichtung der Konvertierung sind in *Anhang A: Datenkonvertierung in CICS* zu finden.

*Übersetzen der CICS-Programme*

Der Einfachheit halber wurden die beiden Quelldateien `ec01.ccp` und `ec02.ccp` mit Hilfe des OMVS-Befehls **cp** in das *Partitioned Dataset* `GSCHLU.CICS.EXCI` kopiert, so dass die Quelldateien als Members `EC01` bzw. `EC02` vorlagen.

Die Übersetzung und Installation der beiden CICS-COBOL-Programme `EC01` und `EC02` soll am Beispiel von `EC01` durchgeführt werden. Die Übersetzung von `EC02` erfolgt analog.

Zur Übersetzung von `EC01` wird ein JCL-Programm namens `STARTEC` eingesetzt, das ebenfalls im Dataset `GSCHLU.CICS.EXCI` angelegt wurde (Abbildung 4-32).

```
000001 //GSCHLUB  JOB (),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
000002 //S1       EXEC DFHEITVL
000003 //TRN.SYSIN DD DSN=GSCHLU.CICS.EXCI(EC01),DISP=SHR
000004 //LKED.SYSIN DD *
000005       NAME EC01(R)
000006 /*
000007 //
```

Abbildung 4-32: JCL zum Übersetzen des CICS-Programms `EC01`

Es wird die Prozedur `DFHEITVL` verwendet, mit der zuerst die `EXEC CICS`-Aufrufe in native Befehle umgewandelt sowie Kompilieren und Linken durchgeführt werden. Analog gibt es auch ähnliche Prozeduren für C, PL/I und Assembler (z.B. `DFHEITDL` für C).

Nachdem der Job erfolgreich war, kann in CICS gewechselt werden. Hier wurde zunächst mittels `CEDA DEFINE PROGRAM(EC01) GROUP(GSCHLU)` das Programm `EC01` in der schon vorhandenen Gruppe `GSCHLU` definiert. Dann wird als Sprache noch `Le370` eingetragen und bestätigt (Abbildung 4-33). Nach der Definition des Programms musste die Gruppe mit Hilfe des Befehls `CEDA INSTALL GROUP(GSCHLU)` neu installiert werden.

```
 DEFINE PROGRAM(EC01) GROUP(GSCHLU)
 OVERTYPE TO MODIFY                                        CICS RELEASE = 0620
  CEDA  DEFine PROGram( EC01     )
   PROGram        : EC01
   Group          : GSCHLU
   DEscription  ==>
   Language     ==> Le370              COBol | Assembler | Le370 | C | Pli
   RELoad       ==> No                 No | Yes
   RESident     ==> No                 No | Yes
   USAge        ==> Normal             Normal | Transient
   USElpacopy   ==> No                 No | Yes
   Status       ==> Enabled            Enabled | Disabled
   RSl            : 00                 0-24 | Public
   CEdf         ==> Yes                Yes | No
   DAtalocation ==> Below              Below | Any
   EXECKey      ==> User               User | Cics
   COncurrency  ==> Quasirent          Quasirent | Threadsafe
  REMOTE ATTRIBUTES
   DYnamic      ==> No                 No | Yes
+  REMOTESystem ==>

                                                    SYSID=CICS APPLID=CICS
  DEFINE SUCCESSFUL                               TIME:  02.42.05  DATE: 04.295
PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Abbildung 4-33: Definition des Programms `EC01`

Die Funktionstüchtigkeit dieses CICS-Programm kann auch unabhängig vom CICS Transaction Gateway getestet werden. Dazu wird die *CECI*-Transaktion verwendet, die `EXEC CICS`-Anweisungen auswerten kann. Der Befehl `LINK PROGRAM(EC01) COMMAREA(' ') LENGTH(18)` innerhalb der CECI-Transaktion entspricht also einem `EXEC CICS LINK PROGRAM`-Aufruf mit den entsprechenden Parametern. Es wird das Programm `EC01` gestartet, eine leere COMMAREA übergeben und deren Länge auf 18 festgelegt (Abbildung 4-34).

```
 LINK PROGRAM(EC01) COMMAREA(' ') LENGTH(18) _
 STATUS:  COMMAND EXECUTION COMPLETE                        NAME=
  EXEC CICS  LInk Program( 'EC01    ' )
   < Commarea( '21/10/04 03:10:44.' ) < Length( +00018 ) > < Datalength() > >
   < SYSid() >
   < SYNconreturn >
   < Transid() >
   < INPUTMSG() < INPUTMSGLen() > >









  RESPONSE: NORMAL              EIBRESP=+0000000000 EIBRESP2=+0000000000
PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

Abbildung 4-34: Test von `EC01` mit Hilfe der CECI-Transaktion

Als Ausgabe werden Datum und Uhrzeit zurückgeliefert, da `EC01` diese abfragt und in die COMMAREA schreibt. Das CICS-Programm selbst funktioniert also, nun muss noch die Zusammenarbeit mit dem CICS Transaction Gateway sichergestellt werden.

*Test der Java-Beispielprogramme*

Zunächst wird das Java-Beispielprogramm `EciB1` betrachtet, das mit ECI-Aufrufen über das CICS Transaction Gateway auf das CICS-Programm `EC01` zugreift. Dieses Java-Programm soll mit Hilfe eines kurzen Shellskripts namens `EciB1Test` von USS/OMVS aus aufgerufen werden (Abbildung 4-35).

```
000001 export CLASSPATH=/usr/lpp/ctg500/ctg/classes/ctgsamples.jar
000002 export CLASSPATH=$CLASSPATH:/usr/lpp/ctg500/ctg/classes/ctgclient.jar
000003 echo $CLASSPATH
000004 java com.ibm.ctg.samples.eci.EciB1 127.0.0.1 2006
```

Abbildung 4-35: Shellskript `EciB1Test` zum Start des Java-Programms `EciB1`

Zum Klassenpfad werden zwei `jar`-Dateien hinzugefügt, wobei `ctgsamples.jar` die Beispielprogramme und `ctgclient.jar` die Java-Klassen für die ECI-Aufrufe enthält. Dem Beispielprogramm `EciB1` werden als Parameter die Adresse des Hostrechners sowie der verwendete Port übergeben. Hier wird mit `127.0.0.1` der lokale Rechner sowie mit `2006` der Standardport des Gateway-Daemon ausgewählt.

Nun muss das erstellte Shellskript mit Hilfe des Befehls **chmod +x EciB1Test** noch ausführbar gemacht werden. Durch Eingabe von **EciB1Test** wird das Skript gestartet. Bei der Ausführung muss der CICS-Server ausgewählt werden. Es wurde nur ein CICS-Server namens `CICS` definiert, so dass dieser durch Eingabe von `1` ausgewählt wird. Die Ausgabe ist in Abbildung 4-36 zu sehen.

```
GSCHLU:/u/gschlu: >EciB1Test
/usr/lpp/ctg500/ctg/classes/ctgsamples.jar:/usr/lpp/ctg500/ctg/classes/ctgclient
.jar

CICS Transaction Gateway Basic ECI Sample 1
-------------------------------------------
Usage: java com.ibm.ctg.samples.eci.EciB1 [Gateway Url]
                                          [Gateway Port Number]
                                          [SSL Classname]
                                          [SSL Password]

To enable client tracing, run the sample with the following Java option:
 -Dgateway.T.trace=on

The address of the Gateway has been set to 127.0.0.1 Port:2006

CICS Servers Defined:

         1. CICS -Default example server


Choose Server to connect to, or q to quit:
1

Program EC01 returned with data:-

        Hex: 32312f31302f30342030333a34383a34370
        ASCII text: 21/10/04 03:48:47
```

Abbildung 4-36: Ausgabe des Java-Testprogramms `EciB1`

Wiederum werden Datum und Uhrzeit ausgegeben. Der ASCII-Test ist lesbar, da für das CICS-Programm `EC01` die Datenkonvertierung aktiviert wurde (siehe *Anhang A: Datenkonvertierung in CICS*). Anderenfalls käme zwar auch eine Ausgabe derselben Länge zustande, es würde sich jedoch um ein Durcheinander an Buchstaben und Sonderzeichen handeln.

Da nun die grundlegende Funktionstüchtigkeit des CICS Trasnsaction Gateway sichergestellt ist, kann auch das zweite Beispielprogramm `EciI1` ausgeführt werden, mit dem die transaktionalen Funktionen getestet werden sollen. Dazu ist ebenfalls eine kurze Skriptdatei namens `EciI1Test` zu erstellen, die genauso aussieht wie `EciB1Test`, mit der Ausnahme, dass in der letzten Zeile `EciB1` durch `EciI1` ersetzt werden muss (Abbildung 4-35). Dann muss auch dieses Skript mit Hilfe des Befehls **chmod +x EciI1Test** als ausführbar gekennzeichnet werden. Bei der Ausführung wird wiederum durch Eingabe von `1` zuerst der CICS-Server namens `CICS` ausgewählt, dann kann man das Programm mehrmals ausführen und am Ende ein `Commit` oder `Rollback` ausgelöst werden. Das hat bei diesem einfachen Programm zwar keine echten Auswirkungen, soll aber die grundlegende Funktionsweise von Transaktionen über das CICS Transaction Gateway demonstrieren (Abbildung 4-37).

```
GSCHLU:/u/gschlu: >EciI1Test
/usr/lpp/ctg500/ctg/classes/ctgsamples.jar:/usr/lpp/ctg500/ctg/classes/ctgclient
.jar
--------------------------------------------------
CICS Transaction Gateway Intermediate ECI Sample
--------------------------------------------------
Usage: java com.ibm.ctg.samples.eci.EciI1 [GatewayUrl] [Port] [SSL Classname] [S
SL Password]
Gateway: tcp://127.0.0.1:2006/ SSL Classname:  SSL Password:

127.0.0.1Gateway opened...

Number of systems = 1

--------------------------------------------------
Systems:
--------------------------------------------------
1: CICS - Default example server
--------------------------------------------------

Select a server by number or 'Q' quit

1

Chosen server = CICS

Waiting for a reply to a request to flow request
*
Reply received
No errors reported

--------------------------------------------------
ASCII:
        1st run OF EC02
--------------------------------------------------
Hex:
20202020202020203173742072756e204f462045433032202020202020202020202020202020202000
00000000000000000000000000000000000000000000
--------------------------------------------------
Do you wish to run the program again in this Logical Unit of Work? - (Y/N)
y

Waiting for a reply to a request to flow request
*
Reply received
No errors reported

--------------------------------------------------
ASCII:
        2nd run OF EC02
--------------------------------------------------
Hex:
20202020202020326e642072756e204f462045433032202020202020202020202020202020202000
00000000000000000000000000000000000000000000
--------------------------------------------------
Do you wish to run the program again in this Logical Unit of Work? - (Y/N)
n

C - Commit or R - rollback?
c
Committed logical unit of work on CICS
Gateway closed
```

Abbildung 4-37: Ausgabe des Java-Testprogramms `EciI1`

Somit funktionieren auch die transaktionalen Funktionen des CICS Transaction Gateway, so dass bei komplexeren Programmen eine Konsistenz der verwendeten Ressourcen sichergestellt werden kann.

Um festzustellen, ob der Gateway-Daemon auch tatsächlich Anfragen von anderen Rechnern aus bearbeitet, soll das Java-Programm `EciB1` noch vom heimischen Windows-Rechner aus getestet werden, in diesem Fall mit dem Betriebssystem Windows XP. Dazu ist jedoch zuerst das CICS Transaction Gateway 5.00 für Windows zu installieren, damit die nötigen Java-Klassen vorhanden sind. Außerdem ist eine Java-Umgebung einzurichten, die mit vielen IBM-Produkten mitgeliefert wird. Zum Beispiel kann man die Java-Umgebung der Tools für den WebSphere Application Server für z/OS verwenden. Zum Starten des Beispiels soll eine ausführbare `CMD`-Datei namens `EciB1Test.CMD` erstellt werden, in der auch der Klassenpfad gesetzt wird (Abbildung 4-38). Es ist zu beachten, dass hier als Parameter die IP-Adresse des z/OS-Rechners angegeben werden muss.

```
@echo off
set CLASSPATH=.;C:\Programme\IBM\IBM CICS Transaction Gateway\classes\ctgsamples.jar
set CLASSPATH=%CLASSPATH%;C:\Programme\IBM\IBM CICS Transaction Gateway\classes\ctgclient.jar
java com.ibm.ctg.samples.eci.EciB1 139.18.12.91 2006
```

Abbildung 4-38: CMD-Datei zum Start von `EciB1` unter Windows

Die Ausgabe des Programms ist analog zum Test unter z/OS und zeigt, dass der Gateway-Daemon auch tatsächlich Anfragen aus dem Netzwerk bearbeitet (Abbildung 4-39).

```
C:\Programme\IBM\IBM CICS Transaction Gateway>EciB1Test

CICS Transaction Gateway Basic ECI Sample 1
-------------------------------------------
Usage: java com.ibm.ctg.samples.eci.EciB1 [Gateway Url]
                                          [Gateway Port Number]
                                          [SSL Classname]
                                          [SSL Password]

To enable client tracing, run the sample with the following Java option:
 -Dgateway.T.trace=on

The address of the Gateway has been set to 139.18.12.91 Port:2006

CICS Servers Defined:

         1. CICS -Default example server


Choose Server to connect to, or q to quit:
1

Program EC01 returned with data:-

        Hex: 32322f31302f30342030323a32393a32340
        ASCII text: 22/10/04 02:29:24
```

Abbildung 4-39: Ausgabe von `EciB1` unter Windows

Nach dem erfolgreichen Test der Java-Beispielprogramme ist die Einrichtung CICS Transaction Gateway für z/OS abgeschlossen. Nun kann mit der Einrichtung des WebSphere Application Server für z/OS begonnen werden.

## 4.3 Konfiguration des WebSphere Application Server

In diesem Abschnitt soll beschrieben werden, wie der WebSphere Application Server V4 für z/OS konfiguriert wird, um mit Hilfe der Dienste des CICS Transaction Gateway auf CICS zuzugreifen. Dabei wird angenommen, dass ein funktionsfähiger WebSphere Application Server vorhanden ist. Zur Konfiguration wird vor allem das *System Management End-User Interface (SMEUI)* eingesetzt, das aus den Anwendungen *Operations* und *Administration* besteht (siehe *Abschnitt 2.6: Windows-Tools für den WebSphere Application Server V4 für z/OS*). Das Hauptziel besteht darin, den CICS-ECI-Ressourcenadapter zu installieren, der Bestandteil des CICS Transaction Gateway ist.

### *4.3.1 Voraussetzungen*

Es müssen die EXCI-Verbindungen zu CICS konfiguriert sowie das CICS Transaction Gateway installiert und erfolgreich getestet worden sein (siehe *Abschnitt 4.1: EXCI-Verbindungen zu CICS* und *4.2 Einrichtung des CICS Transaction Gateway*). Es ist nicht notwendig, den Gateway-Daemon zu starten, da der WebSphere Application Server für z/OS direkt über die Gateway-Klassen auf CICS zugreift.

### *4.3.2 Vorbereitung der Konfiguration*

*Installation des System Management End-User Interface (SMEUI)*

Um Änderungen der Konfiguration durchführen und verschiedene Serverinstanzen starten und stoppen zu können, muss zuerst das System Management End-User Interface (SMEUI) des WebSphere Application Server für z/OS installiert werden.

Es muss die Datei `bboninst.exe` auf den heimischen Windows-Rechner übertragen werden, die sich in diesem Fall im Verzeichnis `/usr/lpp/WebSphere/bin` befand. Das kann unter anderem per FTP geschehen. In diesem Fall wurde auf dem heimischen Rechner ein FTP-Server eingerichtet, dann wurde sich von OMVS aus an diesen angemeldet und mit dem Befehl **PUT** die Datei im Modus **BINARY** übertragen. Danach muss die Datei unter Windows ausgeführt werden, woraufhin die Installation des *SMEUI* beginnt. Unter Beachtung der Installationshinweise wird die Installation zu Ende gebracht.

*Überblick über die aktuelle Konfiguration*

Um sich einen Überblick über die vorliegende Konfiguration zu verschaffen, kann man die Anwendung *Operations* ausführen. Nach dem Start erscheint das Fenster `Login`. Hier muss zunächst der Name des z/OS-Rechners oder dessen IP-Adresse eingegeben werden. Der Port wird standardmäßig auf `900` eingestellt. Dann müssen noch Benutzername und Kennwort eingegeben werden (Abbildung 4-40).

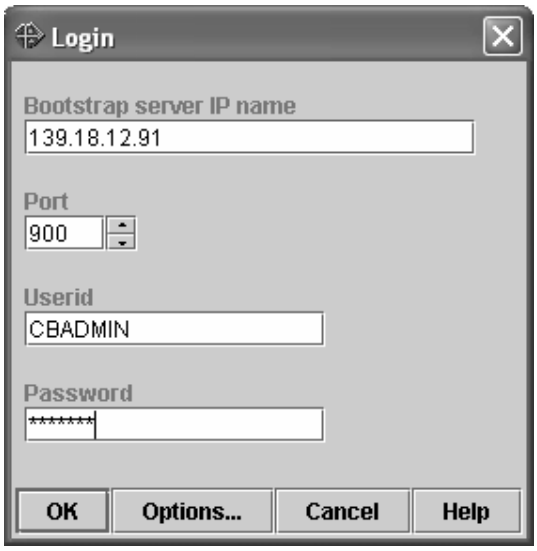


Abbildung 4-40: Das Login-Fenster des *SMEUI*

Nachdem die Anmeldung erfolgreich war, dauert es einige Sekunden, bis die eigentliche Anwendung erscheint (Abbildung 4-41).

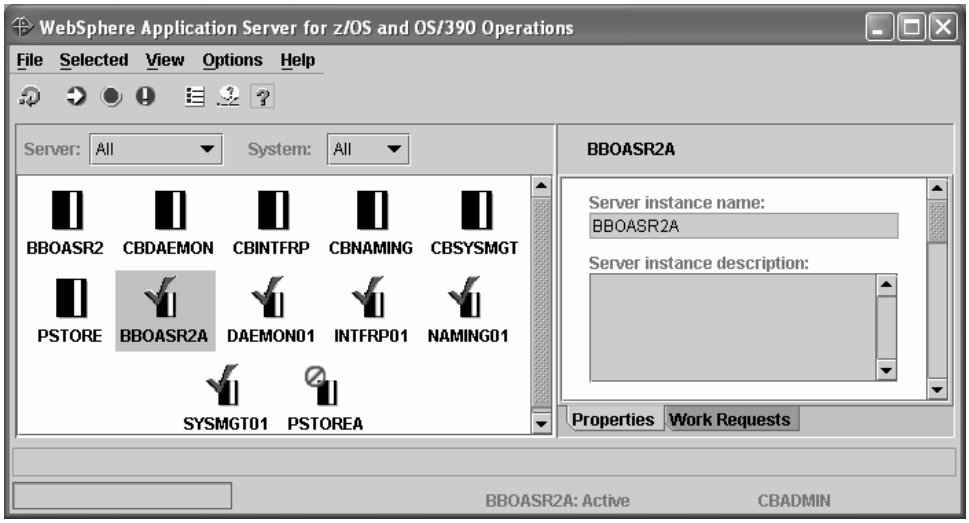


Abbildung 4-41: Die Anwendung *Operations*

Im linken Teil des Fensters sind die verschiedenen Server und deren Instanzen zu sehen. Dabei ist BBOASR2A die Instanz des J2EE-Servers BBOASR2, und genau auf diese Instanz konzentrieren sich die Konfigurationsaufgaben dieses Abschnitts.

*Definition der EXCI-Bibliothek*

Der J2EE-Server, der den CICS-ECI-Ressourcenadapter nutzen soll (BBOASR2A), muss auf das CICS-Modul DFHXCSTB aus der EXCI-Bibliothek CICSTS22.CICS.SDFHEXCI zugreifen können. Dazu soll die Startprozedur des J2EE-Servers modifiziert werden. Diese war in diesem Fall im Member BBOASR2S von ADCD.ZOSV14S.PROCLIB enthalten. Es muss die STEPLIB-Konkatenation der Prozedur um die EXCI-Bibliothek erweitert werden (Abbildung 4-42).

```
000001 //BBOASR2S PROC IWMSSNM='BBOASR2A',PARMS='-ORBsrvname '
000002 //  SET CBCONFIG='/WebSphere390/CB390'
000003 //  SET RELPATH='controlinfo/envfile'
000004 //BBOASR2S EXEC PGM=BBOSR,REGION=0M,TIME=NOLIMIT,
000005 // PARM='/ &PARMS &IWMSSNM'
000006 //STEPLIB  DD DISP=SHR,DSN=WAS401.SBBOULIB
000007 //         DD DISP=SHR,DSN=CICSTS22.CICS.SDFHEXCI
000008 //BBOENV DD PATH='&CBCONFIG/&RELPATH/&SYSPLEX/&IWMSSNM/current.env'
000009 //CEEDUMP  DD SYSOUT=*
000010 //SYSOUT   DD SYSOUT=*
000011 //SYSPRINT DD SYSOUT=*
```

Abbildung 4-42: Modifikation der Startprozedur des J2EE-Servers

Der J2EE-Server wird nach den notwendigen Veränderungen aus den nächsten Abschnitten in jedem Fall neu gestartet, so dass die Änderung an der Startprozedur wirksam wird.

### *4.3.3 Die Konfigurationsdateien des J2EE-Servers*

Eine solche J2EE-Serverinstanz verwendet mehrere Konfigurationsdateien, die sich in diesem Fall im Verzeichnis /WebSphere390/CB390/controlinfo/envfile/ADCDPL/BBOASR2A befinden. Hier soll jedoch nur auf zwei dieser Konfigurationsdateien eingegangen werden:

- current.env (enthält die aktuellen Umgebungsvariablen) und
- jvm.properties (enthält die Einstellungen der *Java Virtual Machine*).

Die Datei jvm.properties ist von Hand zu editieren. Es muss ein Eintrag hinzugefügt werden (classloadermode), der die Art und Weise, auf die Klassen geladen werden,

beeinflusst (Abbildung 4-43). Es muss der Kompatibilitätsmodus (1) eingestellt werden, damit die J2EE-Anwendung CTGTesterCCI aus [CG02b] funktioniert [FCC].

```
com.ibm.ws390.wc.config.filename=/WebSphere390/CB390/
controlinfo/envfile/ADCDPL/BBOASR2A/webcontainer.conf
com.ibm.ws390.trace.settings=/WebSphere390/CB390/
controlinfo/envfile/ADCDPL/BBOASR2A/trace.dat
com.ibm.ws390.server.classloadermode=1
```

Abbildung 4-43: Die Datei jvm.properties

Die Datei current.env sollte nicht von Hand editiert werden, da solche Veränderungen bei Aktivierung einer Konversation in der Anwendung *Administration* wieder verloren gehen. Bevor diese Anwendung gestartet werden soll, müssen jedoch noch einige Schritte durchgeführt werden.

### *4.3.4 Vorbereitung des CICS-ECI-Ressourcenadapters*

Ressourcenadapter liegen in RAR-Dateien *(Resource Adapter Archive)* vor, bei denen es sich analog zu EAR-Dateien um Archive handelt, die aus JAR-Dateien und einem *Deployment-Deskriptor* bestehen. Der CICS-ECI-Ressourcenadapter ist in der Datei cicseciRRS.rar enthalten, die sich im Verzeichnis deployable des CICS Transaction Gateway befindet.

Dieser Ressourcenadapter ist nur für den WebSphere Application Server V4 für z/OS vorgesehen. Parallel gibt es noch einen zweiten Ressourcenadapter namens cicseci.rar, der für alle anderen Anwendungsserver gedacht ist. In späteren Versionen des CICS Transaction Gateway (ab 5.1) wurde diese Unterscheidung beseitigt, und es gibt nur noch einen einzigen CICS-ECI-Ressourcenadapter namens cicseci.rar.

Im Gegensatz zu anderen Anwendungsservern wie zum Beispiel *WebSphere Application Server Advanced Edition* kann der WebSphere Application Server V4 für z/OS nicht unmittelbar mit RAR-Dateien arbeiten. Stattdessen müssen die JAR-Dateien aus der RAR-Datei entpackt und einzeln in den Klassenpfad des J2EE-Servers eingefügt werden.

Dazu müssen in USS/OMVS einige Schritte durchgeführt werden. Zuerst muss sichergestellt sein, dass Zugriff auf eine Java-Umgebung besteht, also der Pfad entsprechend gesetzt wurde. Ein Vorschlag dazu ist in *Abschnitt 4.2.3* zu finden. Weiterhin soll ein Verzeichnis angelegt werden, auf das der WebSphere Application Server zugreifen kann. In diesem Fall wurde im Verzeichnis /WebSphere390/CB390 ein Unterverzeichnis namens jbo erstellt.

Der Versuch, im Verzeichnis des CICS Transaction Gateway ein derartiges Unterverzeichnis einzurichten, führte nicht zum Erfolg, da der WebSphere Application Server nicht direkt auf dieses Verzeichnis zugreifen durfte. Nun wechselt man in das soeben erstellte Verzeichnis und kopiert den Ressourcenadapter dorthin. Die genaue Befehlsabfolge bis dahin lautete also:

```
cd /WebSphere390/CB390
mkdir jbo
cd jbo
cp /usr/lpp/ctg500/ctg/deployable/cicseciRRS.rar .
```

Dann muss der Ressourcenadapter mit Hilfe des Befehls **jar** ausgepackt werden. Danach werden die Berechtigungen zur Ausführung gesetzt:

```
jar -xvf cicseciRRS.jar
chmod ugo+x *
```

Daraufhin sollte das Verzeichnis wie folgt aussehen (Abbildung 4-44):

```
GSCHLU:/WebSphere390/CB390/jbo: >ls -l
total 5152
drwxr-xr-x   2 BPXOINIT CBCFG1      8192 Oct  5 15:16 META-INF
-rwxr-xr-x   1 BPXOINIT CBCFG1     11805 Oct  5 15:16 ccf2.jar
-rwxr-xr-x   1 BPXOINIT CBCFG1     30695 Oct  5 15:17 cicseciRRS.jar
-rwxr-xr-x   1 BPXOINIT CBCFG1   1005620 Oct  5 15:15 cicseciRRS.rar
-rwxr-xr-x   1 BPXOINIT CBCFG1     45637 Oct  5 15:17 cicsframe.jar
-rwxr-xr-x   1 BPXOINIT CBCFG1    525800 Oct  5 15:16 ctgclient.jar
-rwxr-xr-x   1 BPXOINIT CBCFG1    284446 Oct  5 15:17 ctgserver.jar
-rwxr-xr-x   1 BPXOINIT CBCFG1    356352 Oct  5 15:17 libCTGJNI.so
-rwxr-xr-x   1 BPXOINIT CBCFG1    356352 Oct  5 15:17 libCTGJNI_g.so
```

Abbildung 4-44: Das Verzeichnis des Ressourcenadapters

Damit ist der Ressourcenadapter vorbereitet, so dass die nächsten Schritte mit Hilfe der Anwendung *Administration* durchgeführt werden können.

### *4.3.5 Definition einer CICS-ECI-Verbindung*

Die Anmeldung an *Administration* erfolgt genauso wie an *Operations*. Die Anwendung selbst ist jedoch wesentlich komplexer gestaltet. Auf der linken Seite befinden sich die Konversationen, die Konfigurationsänderungen am WebSphere Application Server für z/OS entsprechen. Dabei enthält jede einzelne Konversation die gesamte Konfiguration. Zur Definition der CICS-ECI-Verbindung soll also eine solche Konversation erstellt werden.

Dazu klickt man mit der rechten Maustaste auf den Ordner `Conversations` und wählt `Add`. Daraufhin ist der Name der Konversation einzugeben. Hier wurde `CTG 5.10.2004` gewählt, also ein Name auf Basis des Datums. Nachdem dies geschehen ist, wählt man im Menü `Selected` den Punkt `Save`, woraufhin die neue Konversation in der Liste auf der linken Seite erscheint (Abbildung 4-45).

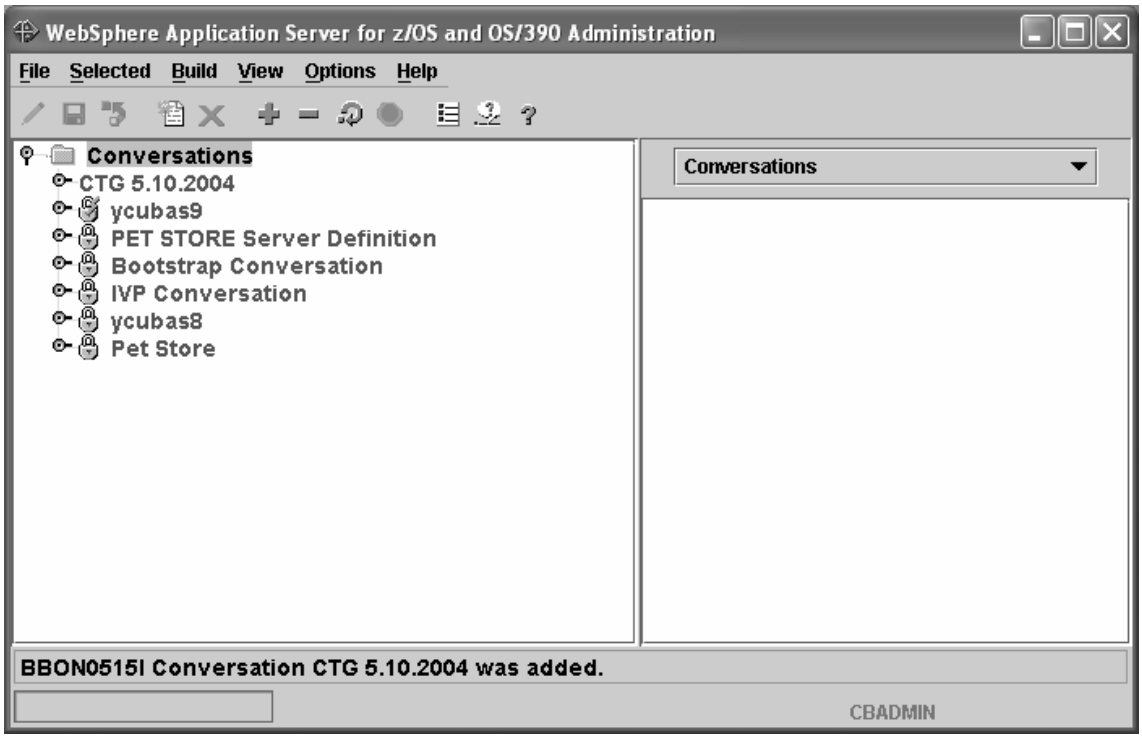


Abbildung 4-45: Die neu erstellte Konversation

Die neu erstellte Konversation enthält die gesamte Konfiguration der letzten aktivierten Konversation. Im Folgenden soll diese Konfiguration so abgeändert werden, dass die entsprechenden Umgebungsvariablen des Servers gesetzt werden und eine neue J2EE-Ressource installiert wird, der CICS-ECI-Ressourcenadapter.

*Aktivieren des Verbindungsmanagements*

Bevor die eigentlichen Variablen gesetzt werden, soll jedoch zuerst im aktuellen Sysplex das *Verbindungsmanagement* aktiviert werden. Dazu expandiert man erst die aktuelle Konversation und dann den Ordner `Sysplexes`. Dort gab es in diesem Fall nur einen Sysplex namens `ADCDPL`, den man mit der rechten Maustaste anklicken und `Modify` wählen muss. Dann können das Kontrollkästchen `Connection Management` aktiviert und die Änderungen mittels `Selected -> Save` übernommen werden.

*Setzen der Umgebungsvariablen des J2EE-Servers*

Als nächstes ist zuerst der Sysplex und dann die Liste der J2EE-Server zu expandieren. Man wählt den J2EE-Server aus, der zusammen mit dem CICS-ECI-Ressourcenadapter verwendet werden soll. In diesem Fall war das `BBOASR2`. Man klickt diesen Server mit der rechten Maustaste an und wählt erneut `Modify`. Im rechten Fenster ist nun die Liste der Umgebungsvariablen zu suchen (Abbildung 4-46).

Environment variable list:

| | Level | Name | Value |
|---|---|---|---|
| 1 | SRV | LIBPATH | /usr/lpp/db2/db2710/lib:/usr/lpp/ja... |
| 2 | SRV | CLASSPATH | /usr/lpp/db2/db2710/classes/db2j... |
| 3 | SRV | DFHJVPIPE | JGATE400 |
| 4 | SPX | BBOLANG | ENUS |
| 5 | SPX | CBCONFIG | /WebSphere390/CB390 |
| 6 | SPX | DAEMON_PORT | 5550 |
| 7 | SPX | DAEMON_SSL_PORT | 5551 |
| 8 | SPX | DATASHARING | 1 |

Abbildung 4-46: Liste der Umgebungsvariablen des J2EE-Servers

Nun doppelklickt man zuerst auf die Zeile mit `CLASSPATH`. In dem neu geöffneten Fenster fügt man die folgenden Dateien aus Verzeichnis `/WebSphere390/CB390/jbo` hinzu (Abbildung 4-47):

- `cicseciRRS.jar`,
- `cicsframe.jar`,
- `ctgserver.jar` und
- `ctgclient.jar`.

Danach ist mit `OK` zu bestätigen. Bei der Änderung der Variable `LIBPATH` ist ebenso vorzugehen. Hier fügt man jedoch das Verzeichnis des CICS-ECI-Ressourcenadapters hinzu. In diesem Fall war das `/WebSphere390/CB390/jbo`.

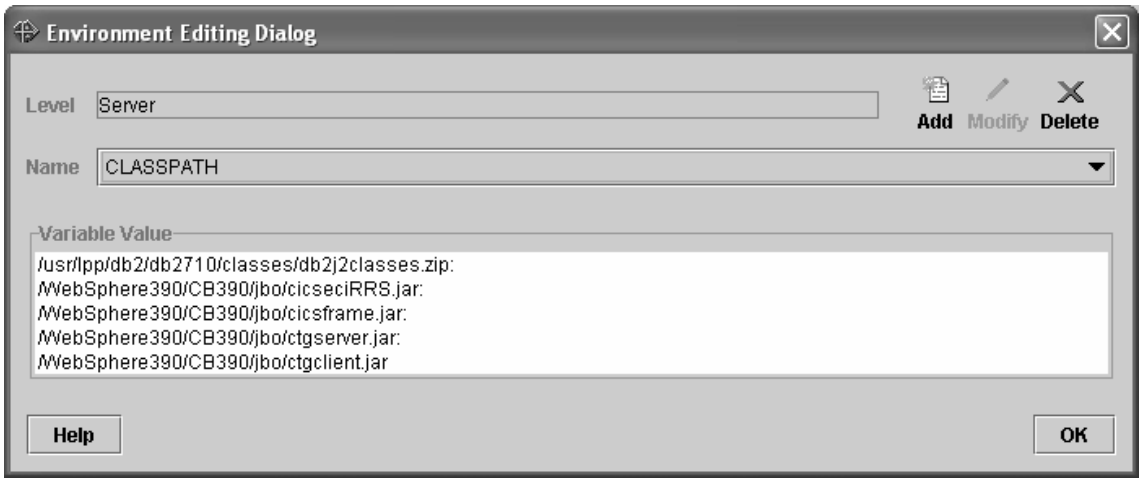


Abbildung 4-47: Änderungen am Klassenpfad des J2EE-Servers

Nun ist noch die Variable `DFHJVPIPE` hinzuzufügen. Dazu doppelklickt man auf die leere Zeile am Ende der Liste der Umgebungsvariablen. Im neu geöffneten Fenster trägt man den Namen `DFHJVPIPE` sowie den Wert `JGATE400` ein. Dieser entspricht dem `Netname` der spezifischen EXCI-Verbindung – siehe *Abschnitt 4.1.2*. Definiert man die Variable `DFHJVPIPE` nicht, so wird eine allgemeine EXCI-Verbindung verwendet. Nun ist erneut mit `OK` zu bestätigen. Die Variable `DFHJVPIPE` sollte nun in der Liste der Umgebungsvariablen auftauchen (Abbildung 4-46).

Da nun alle Umgebungsvariablen gesetzt sind, können die Änderungen mittels `Selected -> Save` übernommen werden.

*Definition einer J2EE-Ressource für den Ressourcenadapter*

Zur Definition einer neuen Ressource ist zuerst im Sysplex der Ordner `J2EE Resources` zu suchen. Dieser muss mit der rechten Maustaste angeklickt werden, danach ist `Add` zu wählen. Auf der rechten Seite können nun mehrere Eingaben getätigt werden. Der Name kann beliebig gewählt werden, in diesem Fall wurde `CICS` eingetragen. Wichtig ist noch der letzte Parameter `J2EE Resource type`. Hier muss aus der Liste unbedingt `CICS_ECIConnectionFactory` gewählt werden. An dieser Stelle zeigt sich die Ähnlichkeit zu JDBC: Zum Einrichten einer Verbindung zu einer Datenbank müsste lediglich `DB2datasource` als Ressourcentyp eingetragen werden.

Da nun die J2EE-Ressource vorhanden ist, kann eine Instanz dieser Ressource erstellt werden. Dazu ist die eben erstellte J2EE-Ressource zu expandieren. Der nun sichtbare Ordner `J2EE Resource Instances` ist mit der rechten Maustaste anzuklicken, und es ist

`Add` auszuzwählen. Die nun im rechten Fenster erschienene Parameterliste ist etwas komplexer als die zur Erstellung der Ressource selbst (Abbildung 4-48).

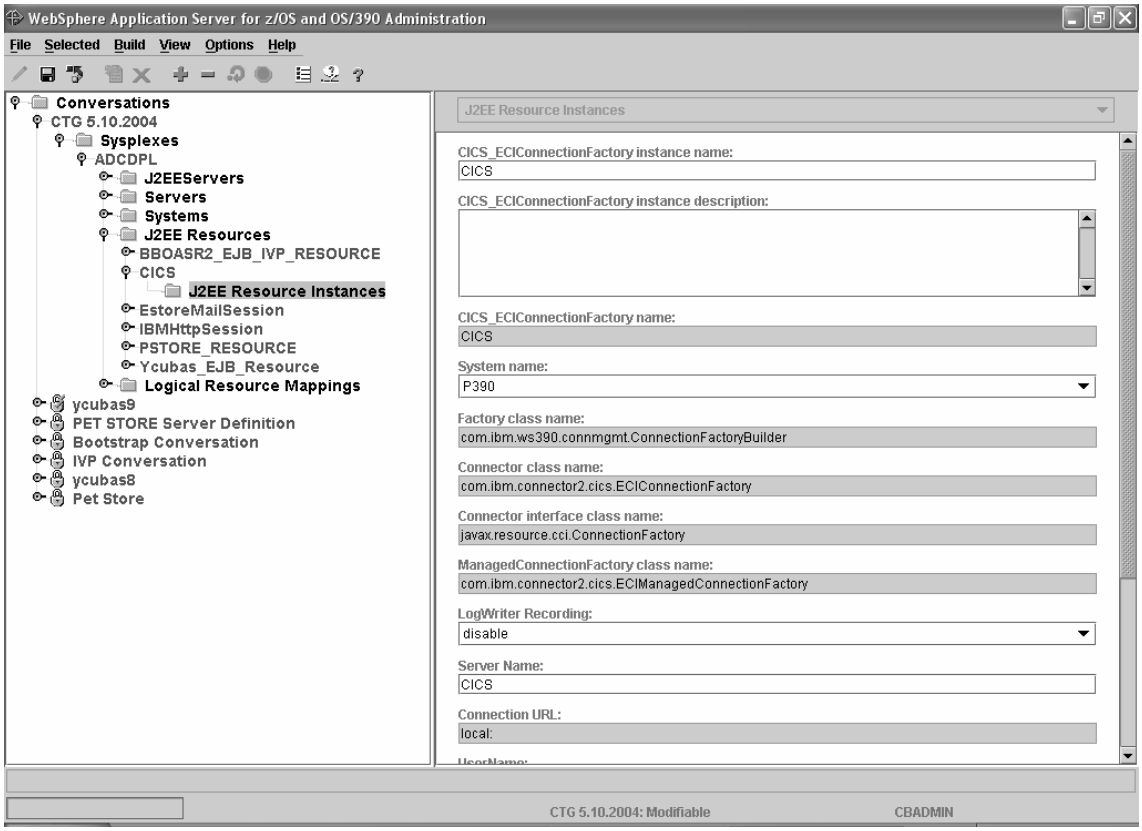


Abbildung 4-48: Erstellung der J2EE-Ressourceninstanz des Ressourcenadapters

Der Instanzname (`CICS_ECIConnectionFactory instance name`) kann erneut frei gewählt werden. Hier wurde erneut `CICS` eingetragen. Als Systemname (`System name`) konnte hier nur `P390` gewählt werden, da es das einzig verfügbare System war. Entscheidend ist der Wert des Servernamens (`Server name`). Hier muss unbedingt der korrekte Name der CICS-Serverregion eingegeben werden, in diesem Fall war das `CICS`. Als Verbindungs-URL (`Connection URL`) ist `local:` obligatorisch, da sich der WebSphere Application Server und das CICS Transaction Gateway auf ein und demselben Rechner befinden. Für die anderen Parameter können die Standardwerte unverändert bleiben. Mit Hilfe von `Selected -> Save` werden erneut die Änderungen übernommen.

Wenn keine weiteren Tätigkeiten auszuführen sind, ist die Einrichtung an dieser Stelle abgeschlossen, und die Konversation kann aktiviert werden. In diesem Fall soll jedoch noch eine J2EE-Anwendung installiert werden, so dass die Konversation offen bleiben soll.

Anweisungen zur Aktivierung einer Konversation sind am Ende des nächsten Abschnitts zu finden.

## 4.4 Installation und Test einer J2EE-Beispielanwendung

In diesem Abschnitt soll eine J2EE-Beispielanwendung auf dem WebSphere Application Server für z/OS installiert und auch getestet werden. Es handelt sich dabei um die Anwendung `CTGTesterCCI` aus dem IBM-Redbook [CG02b]. Diese Anwendung ist in der Datei `CTGTesterCCI.ear` enthalten, die auf der Redbook-FTP-Seite `ftp://www.redbooks.ibm.com/redbooks/SG246133/` oder auch auf der beiliegenden CD erhältlich ist.

### *4.4.1 Der Aufbau der Beispielanwendung*

Bei `CTGTesterCCI` handelt es sich um eine J2EE-Anwendung mit einer Webanwendung und einer zustandslosen Session Bean. Es wird über den im letzten Abschnitt installierten CICS-ECI-Ressourcenadapter auf ein CICS-Programm zugegriffen, und die Ergebnisse werden auf dem Bildschirm dargestellt. Abbildung 4-49 zeigt die vollständige Struktur der J2EE-Anwendung.

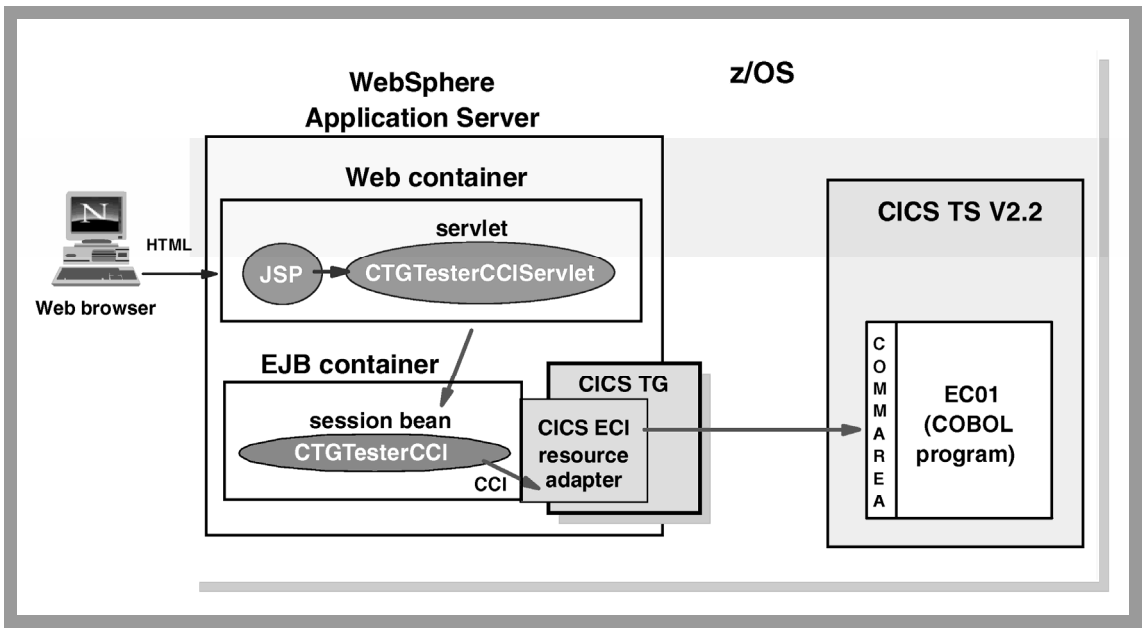


Abbildung 4-49: Die Struktur der J2EE-Anwendung `CTGTesterCCI` [CG02b]

Der Ablauf der Anwendung sieht dabei wie folgt aus:

1. Die JSP-Datei `index.jsp` beinhaltet das Eingabeformular und reicht die erhaltenen Daten an das Servlet `CTGTesterCCIServlet` weiter. Es werden mehrere Daten übergeben, darunter der Name des CICS-Programms und die Länge der COMMAREA.

2. Das Servlet leitet die Daten an die Session Bean weiter.

3. Die Session Bean tätigt mit Hilfe des CICS-ECI-Ressourcenadapters den ECI-Aufruf an CICS, und liefert das Ergebnis zurück an das Servlet.

4. Das Servlet reicht die Ergebnisse an eine entsprechende JSP-Datei weiter.

5. Die JSP-Datei stellt das Ergebnis auf dem Bildschirm dar.

### *4.4.2 Anpassung der Anwendung mit Hilfe des Application Assembly Tools für z/OS*

EAR-Dateien aus dem WebSphere Studio können nicht direkt im WebSphere Application Server für z/OS installiert werden. Zuerst muss das Application Assembly Tool (AAT) für z/OS verwendet werden, um eine EAR-Datei zu generieren, mit der WebSphere für z/OS umgehen kann. Weiterhin können auch Deployment-Informationen geändert werden. Das AAT für z/OS kann unter folgender Adresse heruntergeladen werden:

https://www6.software.ibm.com/dl/WebSphere20/zosos390-p.

Nach dem Start des AAT für z/OS sind im linken Teil die verschiedenen J2EE-Anwendungen zu sehen. Zuerst klickt man mit der rechten Maustaste auf den Ordner `Applications` und wählt `Import`. In dem neu geöffneten Fenster ist die Datei `CTGTesterCCI.ear` auszuwählen. Beim Importieren der `EAR`-Datei treten mehrere Warnungen auf, die sich auf fehlende Klassen beziehen. Bei der ersten Warnung wird gefragt, ob eine Datei geladen werden soll, die die fehlende Klasse `javax.resource.ResourceException` enthält. Diese Frage ist mit `Yes` zu beantworten, woraufhin die Klasse `connector.jar` aus dem CICS Transaction Gateway 5.00 für Windows gesucht und ausgewählt werden muss (Abbildung 4-50).

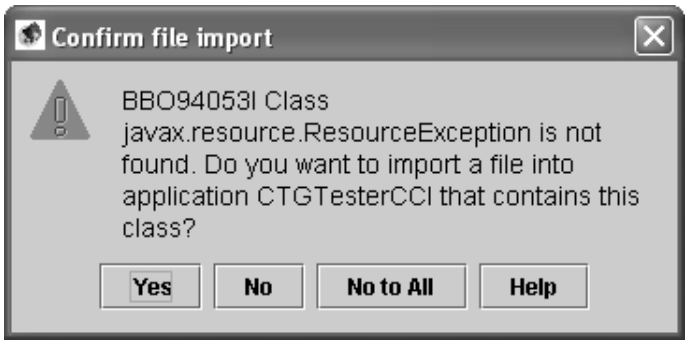


Abbildung 4-50: Warnung aufgrund fehlender Klassen

Nach der ersten Warnung treten auch wiederum welche auf, bei denen jedoch keine weiteren Dateien importiert werden, da die entsprechenden Klassen aus der WebSphere-Laufzeitumgebung verwendet werden sollen. Also wird bei der zweiten Warnung `No to All` gewählt, woraufhin keine weiteren Warnungen auftauchen und die Anwendung importiert wird.

Unter dem Ordner `Applications` sollte nun `CTGTesterCCI` vorhanden sein. Diese Anwendung kann nun expandiert werden, um alle Bestandteile anzuzeigen (Abbildung 4-51).

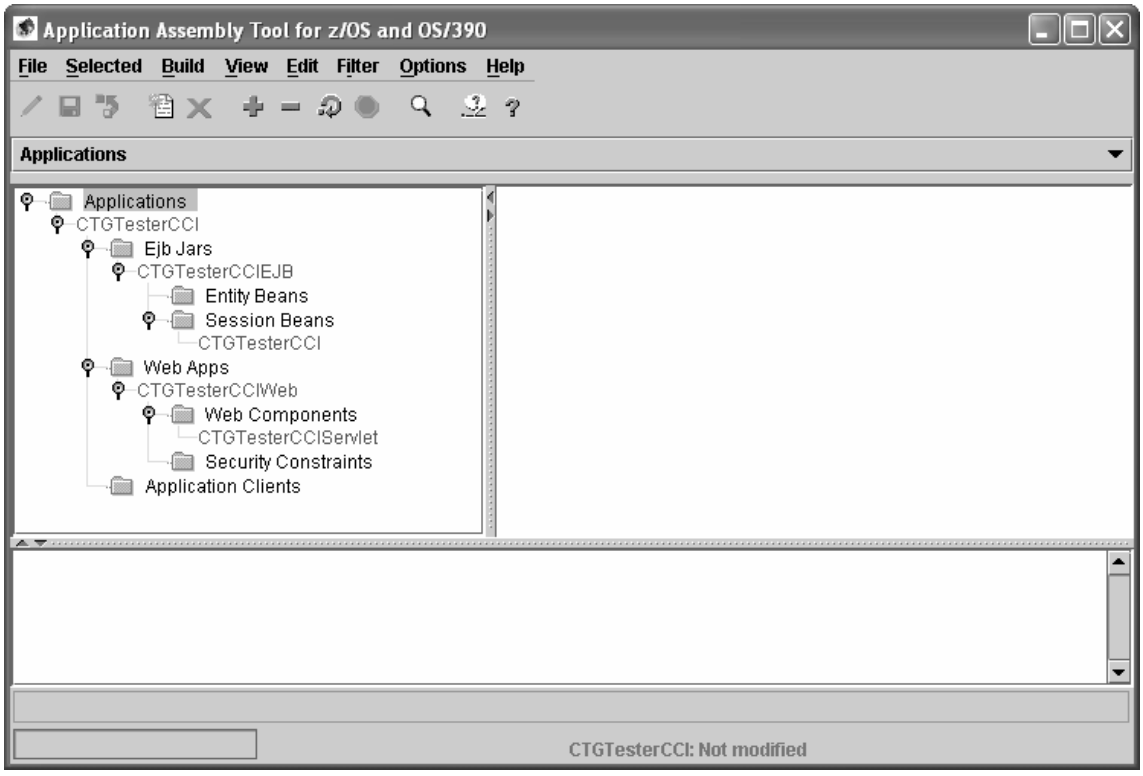


Abbildung 4-51: AAT mit der Beispielanwendung `CTGTesterCCI`

Nun ist sicherzustellen, dass die Webanwendung `CTGTesterCCIWeb` und die Session Bean korrekt miteinander verbunden sind. Dies ist notwendig, da das Servlet eine JNDI-Suche nach der Bean durchführt. Also ist zuerst die Webanwendung `CTGTesterCCIWeb` auszuwählen und im Menü `Selected` auf `Modify` zu klicken. In der Registerkarte `EJBs` ist in der Zeile mit der EJB-Referenz das leere Feld der Spalte `Link` anzuklicken. In der Liste sucht man die Session Bean `CTGTesterCCI` und wählt sie aus. Danach ist im Menü `Selected` der Punkt `Save` zu wählen, um die Änderungen zu übernehmen (Abbildung 4-52).

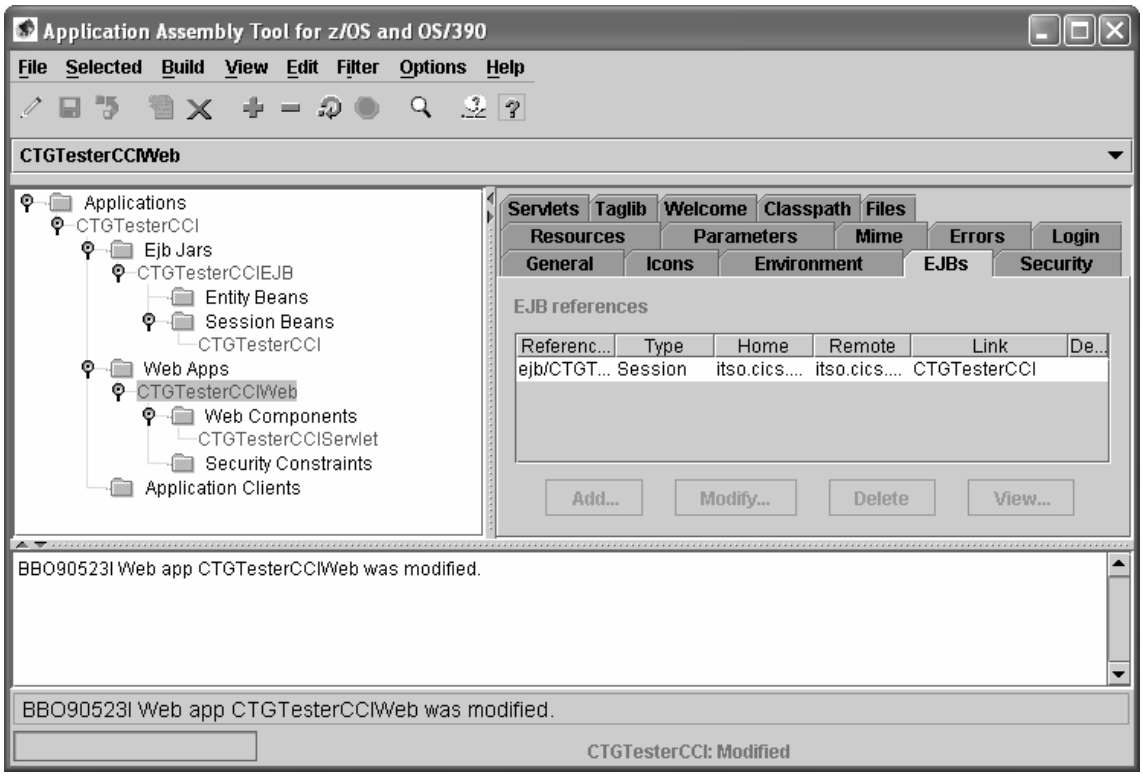


Abbildung 4-52: Veränderungen an der Webkomponente

Nun ist im Menü `Build` zuerst der Punkt `Validate` und nach erfolgreicher Ausführung der Punkt `Deploy` zu wählen. Die Warnungen aufgrund fehlender Klassen sind dabei mit `No` zu beantworten. Nach erfolgreicher Ausführung sollte im unteren Teil folgende Zeile erscheinen:

```
BBO94009I Application CTGTesterCCI was deployed.
```

Schließlich muss die veränderte J2EE-Anwendung noch exportiert werden. Dazu klickt man die Anwendung `CTGTesterCCI` mit der rechten Maustaste an und wählt den Menüpunkt `Export`. In dem neu geöffneten Fenster ist sicherzustellen, dass das Kontrollkästchen `WebSphere for z/OS Version 4.0 compatible` aktiviert ist (Abbildung 4-53).

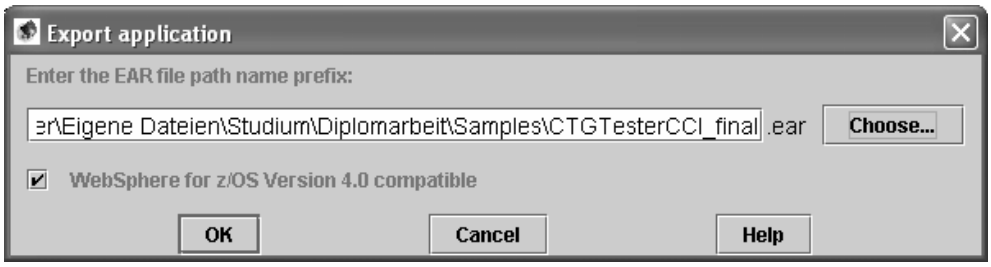


Abbildung 4-53: Exportieren der J2EE-Anwendung

Nachdem mit `OK` bestätigt wurde, wird die Anwendung in eine `EAR`-Datei exportiert.

### *4.4.3 Installation der J2EE-Anwendung*

Nachdem die J2EE-Anwendung an den WebSphere Application Server für z/OS angepasst wurde, kann sie auf diesem installiert werden. Dazu ist erneut die Anwendung *Administration* auszuführen. Hierbei wird angenommen, dass noch eine Konversation geöffnet ist (siehe *Abschnitt 4.3.5* zum Anlegen einer neuen Konversation).

In dieser Konversation ist der J2EE-Server mit der rechten Maustaste anzuklicken, und es ist `Install J2EE Application` zu wählen. Im nächsten Fenster wählt man die im letzten Abschnitt exportierte `EAR`-Datei aus und bestätigt mit OK. Daraufhin öffnet sich das Fenster `Reference and Resource Resolution`, wo Verweise und Ressourcen festgelegt werden sollen.

Dort expandiert man den Ordner `CTGTesterCCIEJB` und klickt die Session Bean `CTGTesterCCI` an. In der Registerkarte `EJB` ist die Schaltfläche `Set Default JNDI Path & Name` anzuklicken, so dass die Bean den Standard-JNDI-Namen erhält. Danach ist die Registerkarte `J2EE Resource` anzuklicken. Hier wird die von der Session Bean verwendete Ressource mit dem im letzten Abschnitt definierten CICS-ECI-Ressourcenadapter verknüpft. Es ist das leere Feld `J2EE Resource` anzuklicken und die Ressource auszuwählen. In diesem Fall lautete der Name `CICS` (Abbildung 4-54).

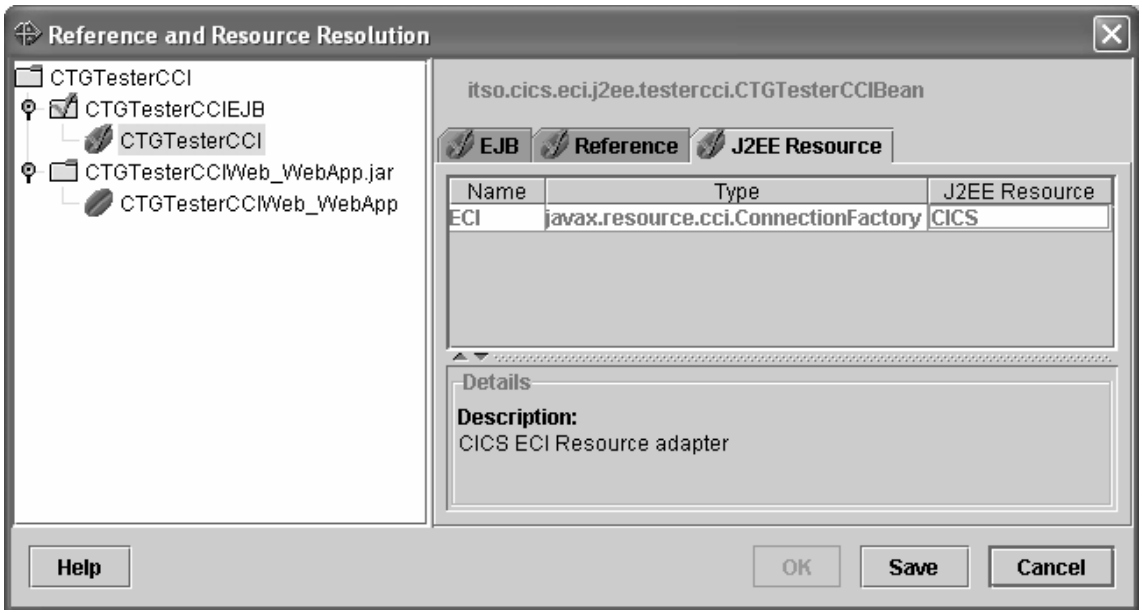


Abbildung 4-54: Auswahl der J2EE-Ressource für die Session Bean

Nun ist der Ordner `CTGTesterCCIWeb_WebApp.jar` zu expandieren und mit der linken Maustaste `CTGTesterCCIWebWebApp` auszuwählen. Auch hier ist in der Registerkarte EJB die Schaltfläche `Set Default JNDI Path & Name` anzuklicken. In der Registerkarte `Reference` sollte nun die aufgelöste Verknüpfung zur Session Bean angezeigt werden. Außerdem sollten im linken Teil des Fensters alle Komponenten der Anwendung mit Häkchen versehen sein, so dass alle Verweise und Ressourcen erfolgreich aufgelöst wurden (Abbildung 4-55).

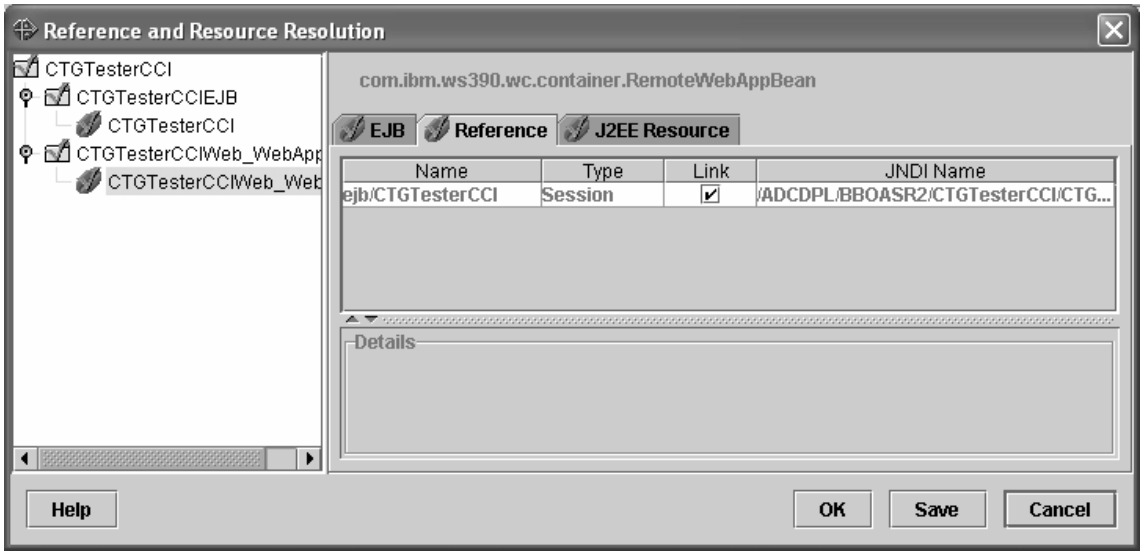


Abbildung 4-55: Aufgelöste Verknüpfung von der Webanwendung zur Session Bean

Nach der Bestätigung mit OK wird die Anwendung per FTP an den Server übertragen (Abbildung 4-56).


Abbildung 4-56: Übertragung der J2EE-Anwendung an der Server

Nachdem die Übertragung erfolgreich war, erscheint in der Statuszeile die Meldung

```
BBON0470I EAR file CTGTesterCCI_deployed_resolved.ear has been
successfully installed on server C1OASR2.
```

Nun kann die Konversation aktiviert werden. Dazu sind folgende Schritte auszuführen:

1. Auf `Validate` klicken.
2. Auf `Commit` klicken und mit `Yes` bestätigen.
3. `Complete -> All Tasks` auswählen und mit `Yes` bestätigen.
4. Auf `Activate` klicken und mit `Yes` bestätigen.

Daraufhin werden alle Definitionen zum J2EE-Server übertragen und aktiviert. Nun wird auch der J2EE-Server neu gestartet.

Das Redbook [CG02b] enthält noch eine andere J2EE-Anwendung namens `CTGTesterECI`. Diese ist weniger komplex aufgebaut, und der Zugriff auf die CTG-Klassen erfolgt direkt vom Servlet aus, so dass keine Session Bean zum Einsatz kommt. Der CICS-ECI-Ressourcenadapter wird ebenfalls nicht verwendet. Die Installation dieser einfacheren Anwendung erfolgt analog zu `CTGTesterCCI`.

### *4.4.4 Test der J2EE-Anwendung*

Nun soll die installierte J2EE-Anwendung auch getestet werden. In der Konfiguration des verwendeten Servers wurde der *IBM HTTP Server* verwendet. Dieser fungiert sozusagen als

Router für die HTTP-Anfrage, die an das *WebSphere-Plugin* und von dort an den Webcontainer des J2EE-Servers weitergeleitet wird (Abbildung 4-57). Deswegen kann der HTTP-Server auch als *Proxy* für den J2EE-Server oder für Sicherheitsabfragen eingesetzt werden.

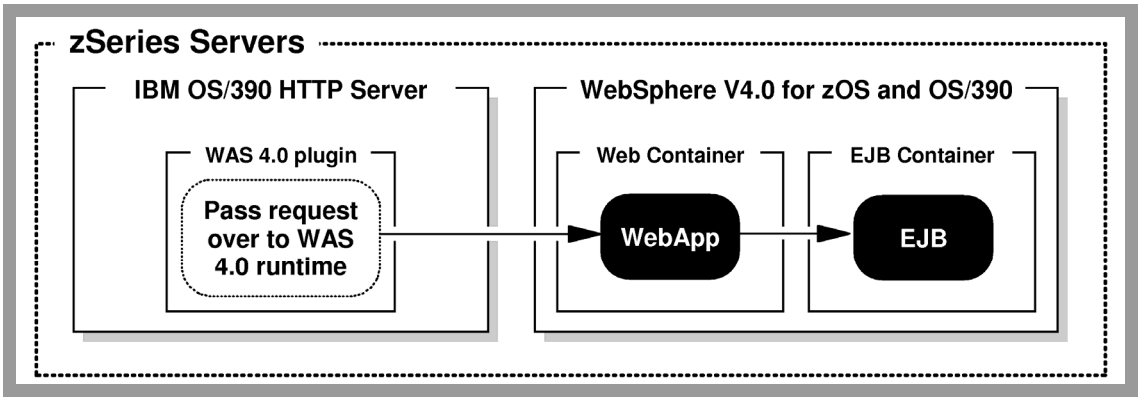


Abbildung 4-57: Der IBM-HTTP-Server für z/OS [CWA]

Die Konfiguration des HTTP-Servers wird in der Datei `/web/httpd1/httpd.conf` festgelegt. Dort soll folgende Zeile eingefügt werden:

```
Service /CTGTester*/*
      /usr/lpp/WebSphere/WebServerPlugIn/bin/was400plugin.so:service_exit
```

Durch die `Service`-Anweisung wird eine Anfrage an die WebSphere-Plugin-Umgebung weitergegeben. Falls die *URL (Uniform Resource Locator)* dem angegebenen Schema entspricht (`/CTGTester*/*`), dann wird die Anfrage an das ausführbare Modul der WebSphere-Plugin-Umgebung (`was400plugin.so`) weitergeleitet.

Nun muss der HTTP-Server neu gestartet werden. Dazu wechselt man in SDSF und beendet den aktuellen HTTP-Server mittels `/C HTTPD1`. Nachdem der HTTP-Server beendet wurde, wird er mittels `/S HTTPD1` neu gestartet. Nun dauert es eine Weile bis die Initialisierung abgeschlossen ist. Danach sollte der HTTP-Server wieder Anfragen entgegennehmen.

Als nächstes soll die folgende URL in einen Browser eingegeben werden:

http://techspru2.informatik.uni-leipzig.de/CTGTesterCCIWeb/

Daraufhin erscheint die Startseite der J2EE-Beispielanwendung (Abbildung 4-58).

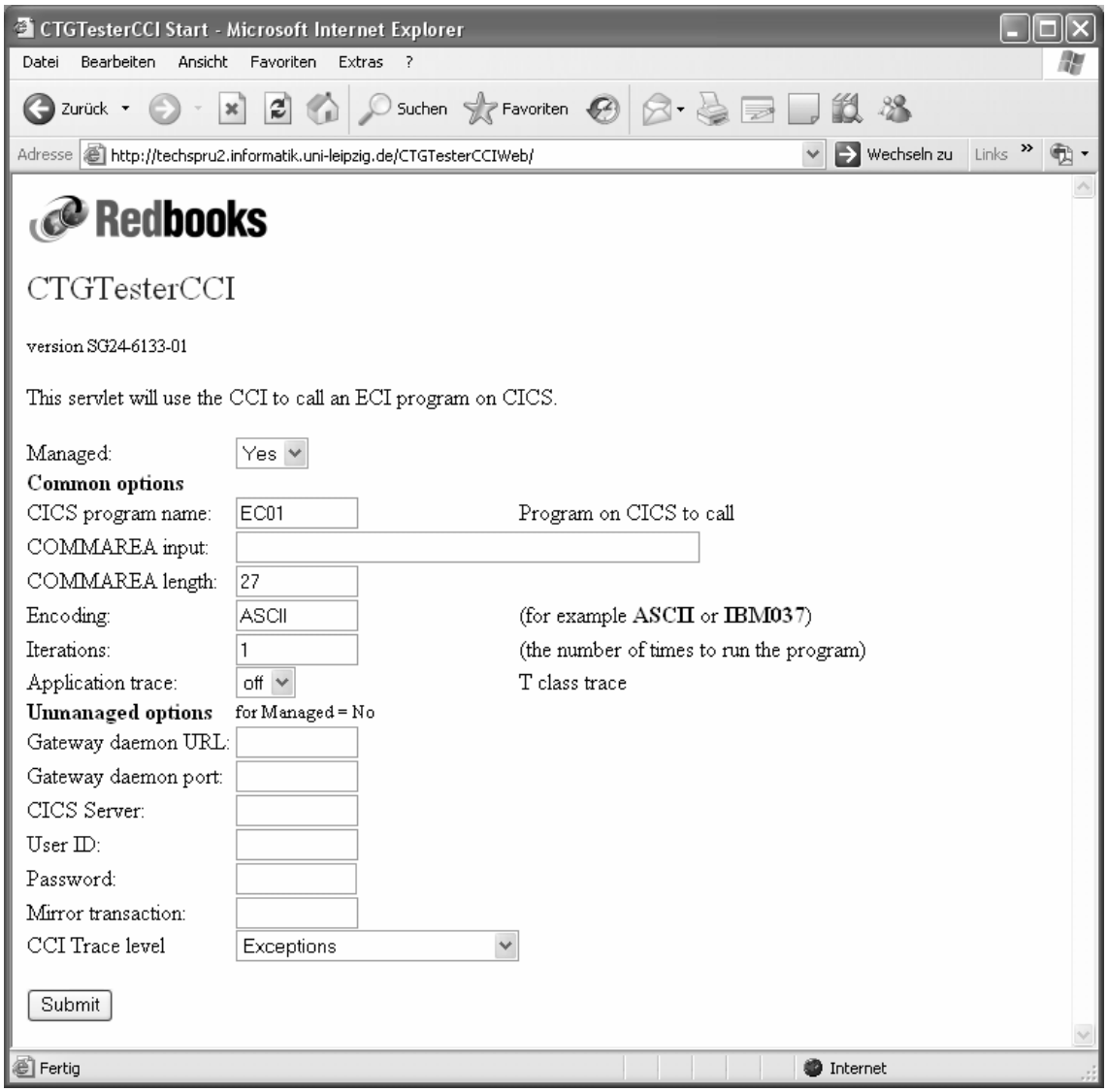


Abbildung 4-58: Die Startseite der J2EE-Beispielanwendung

Da die Session Bean auf den Ressourcenadapter zugreift und dort schon viele Einstellungen getätigt wurden (wie zum Beispiel der Name des CICS-Servers), müssen nur wenige Parameter festgelegt werden. Als erster Test wurden die Beispielangaben übernommen, so dass das CICS-Programm `EC01` mit einer COMMAREA-Länge von `27` und einer ASCII-Kodierung aufgerufen werden soll. Die Option `Managed` legt fest, ob der CICS-ECI-Ressourcenadapter verwendet werden soll oder nicht. Ist dies nicht der Fall, so müssen unter `Unmanaged Options` noch einige Werte eingegeben werden, die normalerweise durch den Ressourcendapter festgelegt sind.

Unterhalb der Formularfelder der Startseite liegt die Schaltfläche `Submit`, durch die die festgelegten Eingabewerte übernommen und die Anfrage gestellt wird. Nach der Betätigung dieser Schaltfläche erscheint die Ergebnisseite (Abbildung 4-59).

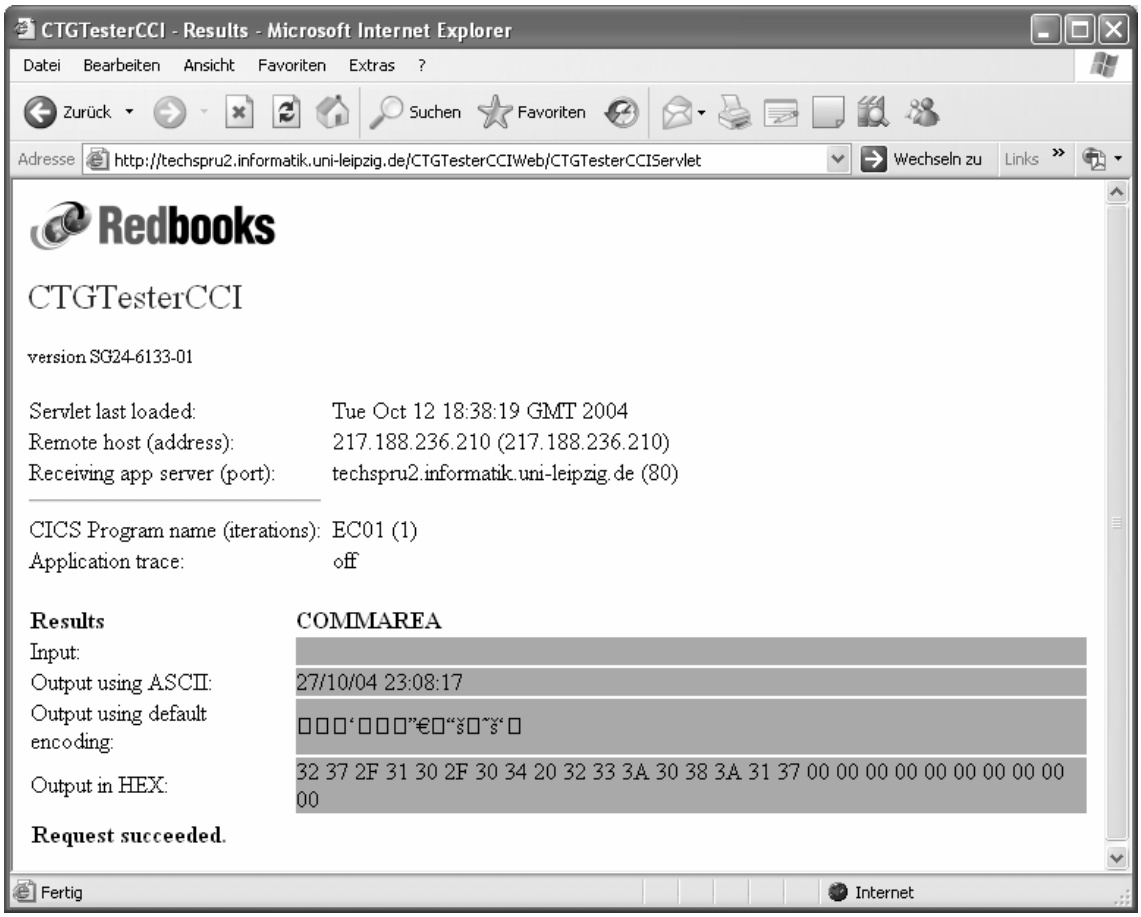


Abbildung 4-59: Die Ergebnisseite der J2EE-Beispielanwendung

Die letzte Zeile auf dieser Seite deutet an, dass die Anfrage erfolgreich war. Im oberen Teil werden zuerst der letzte Ladezeitpunkt des Servlets, die IP-Adresse des eigenen PCs sowie die Adresse des Anwendungsservers angezeigt. Im unteren Teil werden die Ausgabedaten aus der COMMAREA in verschiedenen Kodierungen dargestellt. Die Daten in der ASCII-Kodierung sind lesbar, da in CICS eine Konvertierung von EBCDIC zu ASCII vorgenommen wurde. Es werden erneut Datum und Uhrzeit ausgegeben, wie schon in den Tests aus den *Abschnitten 4.1.4* und *4.2.4*. Die Ausgabe in der Standardkodierung (in diesem Fall EBCDIC) ist dagegen nicht lesbar. Schließlich werden die Ergebnisse am Ende noch als Hexadezimaldaten dargestellt.

Damit wurde die grundlegende Funktionstüchtigkeit des WebSphere Application Server in Kombination mit dem CICS Transaction Gateway demonstriert. Nun können beliebige J2EE-Anwendungen zum Zugriff auf CICS-Programme erstellt werden, sofern bei diesen der Datenaustausch über die COMMAREA geschieht.

*K a p i t e l 5*

# Erstellen einer eigenen Anwendung zum Zugriff auf DB2

In diesem Kapitel soll die bestehende Konfiguration dazu eingesetzt werden, um auf DB2-Datenbanken zuzugreifen. Der Zugriff auf DB2 geschieht dabei im CICS-Programm. Daher ist es lediglich notwendig, ein CICS-Programm zu erstellen, das Daten aus einer DB2-Datenbank ausliest und diese dann in die COMMAREA schreibt. Wenn dies sichergestellt ist, kann das CICS Transaction Gateway mit diesem CICS-Programm kommunizieren. Diese Konfiguration soll dann erneut mit der J2EE-Beispielanwendung getestet werden.

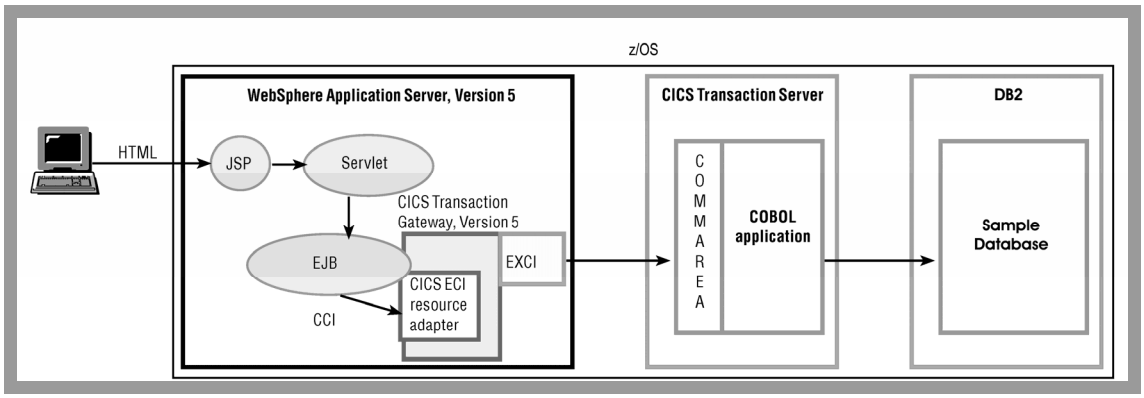


Abbildung 5-1: Die Konfiguration zum Zugriff auf DB2 (nach [IWC])

## 5.1 CICS und DB2

Neben dem Zugriff auf VSAM-Dateien stellt CICS auch eine Schnittstelle zu DB2 bereit, die so genannte *CICS DB2 Attachment Facility*. Durch diese können Programme, die in der CICS-Umgebung ablaufen, auf DB2 zugreifen. Dabei koordiniert CICS die Wiederherstellung sowohl der DB2- als auch der CICS-Daten, falls während einer Transaktion ein Fehler auftritt.

### *5.1.1 DB2CONN*

Dabei stellt die CICS-DB2-Attachment-Facility zum einen eine ständige Verbindung zu DB2 her – *DB2CONN*. Diese Verbindung kann nun von CICS-Anwendungen verwendet werden, um Befehle und Anfragen an DB2 zu senden. Sie kann zu jedem Zeitpunkt beendet und neu gestartet werden. Als Parameter zum Erstellen einer DB2CONN werden unter anderem das

DB2-Subsystem und eine Benutzer-ID verwendet. Im gesamten CICS-Subsystem darf jeweils nur eine `DB2CONN` aktiviert sein.

### *5.1.2 DB2ENTRY*

Innerhalb der ständigen Verbindung zu CICS können mehrere *Threads* eingesetzt werden. Ein Thread stellt hier eine individuelle Verbindung zu DB2 dar, die für jede Transaktion aufgebaut wird, welche auf DB2 zugreift. Damit kann eine CICS-Transaktion auf DB2-Ressourcen wie den *Kommandoprozessor* und *Pläne* zugreifen. Zur Reservierung solcher Threads wird ein *DB2ENTRY* eingesetzt. Dieser legt fest, welche Transaktionen auf bestimmte Pläne in DB2 zugreifen können. Um einen DB2ENTRY anlegen zu können, muss eine DB2CONN vorhanden sein.

### *5.1.3 DB2TRAN*

Mit Hilfe einer *DB2TRAN*-Definition können weitere Transaktionen mit einem bestimmten DB2ENTRY verbunden werden. Für eine Transaktion kann jeweils nur eine DB2TRAN-Definition erstellt werden.

## 5.2 Implementierung der CICS-Anwendung

### *5.2.1 Das CICS-DB2-Programm*

Wie schon erwähnt, soll das zu erstellende CICS-Programm auf DB2 zugreifen und die erhaltenen Daten in die COMMAREA schreiben. Der Einfachheit halber wurde die Beispieldatenbank aus [TUT4] verwendet, wo auch Anweisungen zum Erstellen dieser oder einer ähnlichen Datenbank enthalten sind. Diese Datenbank enthält nur eine Tabelle mit zwei Spalten, in denen Vor- und Nachnamen stehen. Abbildung 5-2 zeigt die Ausgabe einer `SELECT`-Abfrage nach allen Datensätzen dieser Datenbank, die unter *DB2 Admin* ausgeführt wurde.

```
DB2 Admin -------------- DB2 Result of the SQL SELECT -------- Row 1 to 4 of 4

L VNAME                NNAME
  *                    *
- -------------------- --------------------
  HANS                 BAUER
  FRED                 MAYER
  JORG                 WAGNER
  GERRIT               SCHLÜTER
```

Abbildung 5-2: Inhalt der Beispieldatenbank

Weiterhin wurde das CICS-COBOL-Programm aus [TUT5] modifiziert. Dort wurden Daten aus einer DB2-Datenbank gelesen und in einen BMS-Bildschirm exportiert. In diesem Fall wurden die Daten jedoch in die COMMAREA geschrieben. Abbildung 5-3 zeigt den Quelltext des CICS-Programms zum Zugriff auf die DB2-Datenbank.

```
000100        IDENTIFICATION DIVISION.
000200        PROGRAM-ID. ECIDB2.
000300        ENVIRONMENT DIVISION.
000400        DATA DIVISION.
000500        WORKING-STORAGE SECTION.
000600            EXEC SQL
000700               INCLUDE SQLCA
000800            END-EXEC.
000900        01  NAME-TAB.
001000            02 VORNAME     PICTURE  X(20).
001100            02 NACHNAME    PICTURE  X(20).
001300        LINKAGE SECTION.
001310        01 DFHCOMMAREA.
001320           03 VNAM1I       PICTURE  X(20).
001330           03 NNAM1I       PICTURE  X(20).
001340           03 VNAM2I       PICTURE  X(20).
001350           03 NNAM2I       PICTURE  X(20).
001360           03 VNAM3I       PICTURE  X(20).
001370           03 NNAM3I       PICTURE  X(20).
001380           03 VNAM4I       PICTURE  X(20).
001390           03 NNAM4I       PICTURE  X(20).
001400        PROCEDURE DIVISION.
001410            MOVE SPACES TO DFHCOMMAREA.
001500            EXEC SQL
001600              DECLARE C1 CURSOR FOR
001700              SELECT VNAME,NNAME FROM GSCHLU.TAB020
001800            END-EXEC.
001900            EXEC SQL OPEN C1 END-EXEC.
002000            EXEC SQL FETCH C1 INTO :VORNAME, :NACHNAME END-EXEC.
002100            MOVE VORNAME TO VNAM1I.
002200            MOVE NACHNAME TO NNAM1I.
002300            EXEC SQL FETCH C1 INTO :VORNAME, :NACHNAME END-EXEC.
002400            MOVE VORNAME TO VNAM2I.
002500            MOVE NACHNAME TO NNAM2I.
002600            EXEC SQL FETCH C1 INTO :VORNAME, :NACHNAME END-EXEC.
002700            MOVE VORNAME TO VNAM3I.
002800            MOVE NACHNAME TO NNAM3I.
002900            EXEC SQL FETCH C1 INTO :VORNAME, :NACHNAME END-EXEC.
003000            MOVE VORNAME TO VNAM4I.
003100            MOVE NACHNAME TO NNAM4I.
003200            EXEC SQL CLOSE C1 END-EXEC.
003700            EXEC CICS RETURN
003800                      END-EXEC.
003900            EXIT.
```

Abbildung 5-3: Der Quelltext des CICS-Programms zum Zugriff auf DB2

Die DB2-Anweisungen beginnen mit `EXEC SQL`. Zuerst wird für eine `SELECT`-Abfrage nach den Vor- und Nachnamen aus der Tabelle ein Cursor `C1` erstellt (Zeile `001500`). Mit Hilfe dieses Cursors werden die Ergebnisdaten zeilenweise in die Variablen `VORNAME` und `NACHNAME` eingelesen (z.B. Zeile `002000`). Danach werden die Daten in die entsprechenden Felder der `DFHCOMMAREA` kopiert (z.B. Zeilen `002100` und `002200`), die in der `LINKAGE SECTION` (ab Zeile `001300`) definiert wurden.

Nachdem alle vier Zeilen in die COMMAREA geschrieben wurden, kann der Cursor geschlossen werden (Zeile `003200`). Mit `EXEC CICS RETURN` wird die Kontrolle an die nächst höhere Ebene zurückgegeben, und das Programm ist beendet.

### *5.2.2 Die Übersetzungsschritte*

Ein CICS-DB2-Programm wird auf dieselbe Art und Weise übersetzt wie ein normales CICS-Programm. Es muss jedoch ein zusätzlicher Schritt vorgenommen werden – das *Binding*. Bei diesem Prozess wird in DB2 ein *Anwendungsplan* erstellt. Dieser enthält alle SQL-Anweisungen aus dem CICS-Programm und wird bei der Ausführung des Programms aufgerufen. Abbildung 5-4 stellt die notwendigen Schritte zur Übersetzung des CICS-DB2-Programms dar.

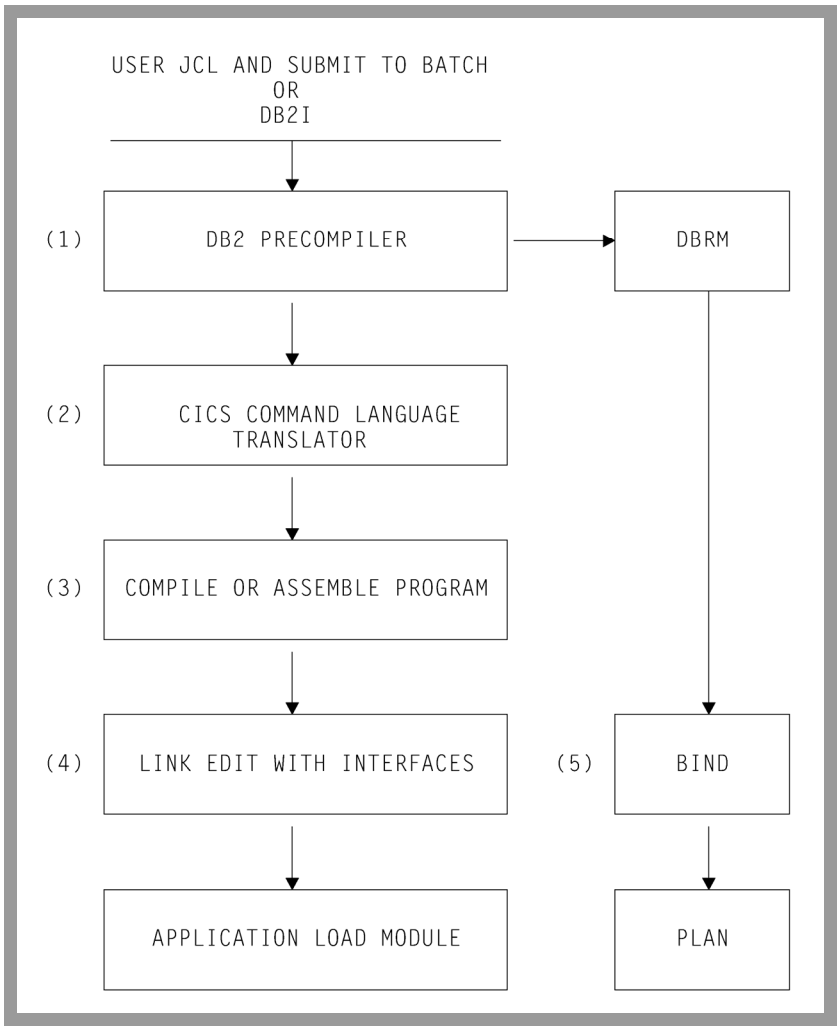


Abbildung 5-4: Die Schritte zur Übersetzung des CICS-DB2-Programms [CS02c]

*Der DB2-Precompiler*

In Schritt `(1)` wird der *DB2-Precompiler* aufgerufen, der ein *DBRM*-Modul (*Database Request Model*) erstellt, das Informationen über die SQL-Anweisungen des Programms beinhaltet. Abbildung 5-5 enthält ein JCL-Programm zum Ausführen des DB2-Precompilers (`DSNHPC`).

```
000100 //GSCHLUP  JOB (),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
000300 //PCOMP    EXEC PGM=DSNHPC,REGION=4096K,
000310 //          PARM='HOST(COB2)'
000320 //STEPLIB  DD DISP=SHR,DSN=DSN710.SDSNEXIT
000330 //          DD DISP=SHR,DSN=DSN710.SDSNLOAD
000400 //DBRMLIB  DD DSN=GSCHLU.DBRMLIB.DATA(DB2CM),DISP=OLD
000500 //SYSCIN   DD DSN=GSCHLU.CICSDB2.ECI(DB2OUT),DISP=SHR
000600 //SYSLIB   DD DSN=GSCHLU.CICSDB2.ECI,DISP=SHR
000800 //SYSPRINT DD SYSOUT=*
000900 //SYSTERM  DD SYSOUT=*
001000 //SYSUDUMP DD SYSOUT=*
001100 //SYSUT1   DD SPACE=(800,(500,500),,,ROUND),UNIT=SYSDA
001200 //SYSUT2   DD SPACE=(800,(500,500),,,ROUND),UNIT=SYSDA
001300 //SYSIN    DD DISP=SHR,DSN=GSCHLU.CICSDB2.ECI(ECIDB2)
001400
```

Abbildung 5-5: JCL-Programm zum Ausführen des DB2-Precompilers

Dabei war der Quellcode des CICS-DB2-Programms in `GSCHLU.CICSDB2.ECI(ECIDB2)` enthalten (Zeile `001300`). Die Ausgabe des neuen CICS-Programms mit auskommentierten `EXEC SQL`-Aufrufen ist in `GSCHLU.CICSDB2.ECI(DB2OUT)` enthalten (Zeile `000500`), während das DBRM-Modul nach `GSCHLU.DBRMLIB.DATA(DB2CM)` geschrieben wurde (Zeile `000400`).

*CICS-Translator, Compiler und Linker*

Die Schritte `(2)`, `(3)` und `(4)` müssen für alle CICS-Programme ausgeführt werden, ob sie nun auf DB2 zugreifen oder nicht. In Schritt `(2)` wird der *CICS-Translator* ausgeführt, der die `EXEC CICS`-Aufrufe in native Befehle umwandelt. In Schritt `(3)` wird das nun lediglich aus COBOL-Befehlen bestehende Programm kompiliert, woraufhin in Schritt `(4)` mit Hilfe des *Linkers* ein ausführbares Lademodul erstellt wird. Abbildung 5-6 enthält ein JCL-Programm mit den notwendigen Schritten für diese drei Aufgaben sowie den Binding-Prozess, der weiter hinten behandelt wird.

```
000001 //GSCHLUC  JOB (),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
000002 //COMP     EXEC DFHEITVL
000003 //TRN.SYSIN DD DSN=GSCHLU.CICSDB2.ECI(DB2OUT),DISP=SHR
000004 //COB.SYSLIB DD DSN=GSCHLU.LIB,DISP=SHR
000005 //LKED.CICSLOAD DD DSN=CICSTS22.CICS.SDFHLOAD,DISP=SHR
000006 //LKED.DB2LOAD DD DSN=DSN710.SDSNLOAD,DISP=SHR
000007 //LKED.SYSIN DD *
000008       INCLUDE CICSLOAD(DSNCLI)
000009       INCLUDE DB2LOAD(DSNHADDR)
000010       INCLUDE DB2LOAD(DSNHADD2)
000011       NAME ECIDB2(R)
000012 /*
000013 //BIND     EXEC PGM=IKJEFT01
000014 //STEPLIB  DD DISP=SHR,DSN=DSN710.SDSNEXIT
000015 //         DD DISP=SHR,DSN=DSN710.SDSNLOAD
000016 //DBRMLIB  DD DISP=OLD,DSN=GSCHLU.DBRMLIB.DATA(DB2CM)
000017 //SYSPRINT DD SYSOUT=*
000018 //SYSTSPRT DD SYSOUT=*
000019 //SYSUDUMP DD SYSOUT=*
000020 //SYSTSIN  DD *
000021   DSN S(DSN1)
000022   BIND PLAN(DB2C) MEMBER(DB2CM) ACTION(REP) RETAIN ISOLATION(CS)
000023   END
000024 //GRANT    EXEC PGM=IKJEFT01
000025 //STEPLIB  DD DISP=SHR,DSN=DSN710.SDSNLOAD
000026 //SYSPRINT DD SYSOUT=*
000027 //SYSTSPRT DD SYSOUT=*
000028 //SYSUDUMP DD SYSOUT=*
000029 //SYSTSIN  DD *
000030   DSN S(DSN1)
000031   RUN PROGRAM(DSNTIAD) PLAN(DSNTIA71) -
000032        LIBRARY('DSN710.RUNLIB.LOAD')
000033   END
000034 //SYSIN    DD *
000035   GRANT EXECUTE ON PLAN DB2C TO PUBLIC
000036 /*
```

Abbildung 5-6: JCL-Programm für die restlichen Schritte

Zum Übersetzen der `EXEC CICS`-Aufrufe, zum Kompilieren und zum Linken wird wie schon in *Abschnitt 4.2.4* die Prozedur `DFHEITVL` verwendet (Zeile `000002`). Diese Prozedur verwendet als Eingabe die vom DB2-Precompiler erstellte Quelldatei mit den auskommentierten `EXEC SQL`-Aufrufen. Für den Linker werden noch einige Lademodule aus CICS und DB2 benötigt (Zeilen `000005` bis `000011`). Besonders zu beachten ist dabei das Modul *DSNCLI (CICS DB2 Language Interface Module)*, das die DB2-Aufrufe ermöglicht.

*Der Binding-Prozess*

In Schritt `(5)` – dem Binding-Prozess (ab Zeile `000013`) – wird das in Schritt `(1)` erstellte DBRM-Modul benutzt (Zeile `000016`), um in DB2 den Anwendungsplan `DB2C` zu erstellen

(Zeile 000022). Dazu wird das Programm IKJEFT01 verwendet, dem die DB2-Anweisungen zum Erstellen des Plans übergeben werden.

Schließlich wird im Step GRANT (ab Zeile 000024) noch sichergestellt, dass auch alle Benutzer auf den soeben erstellten Plan zugreifen dürfen.

Nachdem die beiden JCL-Programm fehlerfrei durchgelaufen sind (*Condition Codes* 0 oder 4), ist das CICS-DB2-Programm erfolgreich erstellt worden. Nun kann es in CICS eingebunden und getestet werden.

### *5.2.3 Einbindung des Programms in CICS*

*Einrichtung der Datenbankverbindung*

Vor der Einbindung des Programms soll noch die CICS-Datenbankverbindung erstellt werden. Es war eine DB2CONN zu erstellen, was in CICS mit Hilfe des Befehls CEDA DEFINE DB2CONN(DB2C) GROUP(GSCHLU) geschah. Abbildung 5-7 und Abbildung 5-8 stellen die Attribute der neuen Datenbankverbindung dar.

```
 VIEW DB2CONN(DB2C) GROUP(GSCHLU)
 OBJECT CHARACTERISTICS                                  CICS RELEASE = 0620
  CEDA  View DB2Conn( DB2C     )
   DB2Conn         : DB2C
   Group           : GSCHLU
   DEscription     : DB2 CONNECTION
  CONNECTION ATTRIBUTES
   CONnecterror    : Sqlcode           Sqlcode | Abend
   DB2Groupid      :
   DB2Id           : DSN1
   MSGQUEUE1       : CDB2
   MSGQUEUE2       :
   MSGQUEUE3       :
   Nontermrel      : Yes               Yes | No
   PUrgecycle      : 00 , 30           0-59
   Resyncmember    : Yes               Yes | No
   SIgnid          :
   STANdbymode     : Reconnect         Reconnect | Connect | Noconnect
   STATsqueue      : CDB2
+  TCblimit        : 0012              4-2000

                                                    SYSID=CICS APPLID=CICS

PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Abbildung 5-7: Definition einer DB2CONN – Bildschirm 1

```
 VIEW DB2CONN(DB2C) GROUP(GSCHLU)
 OBJECT CHARACTERISTICS                                  CICS RELEASE = 0620
  CEDA  View DB2Conn( DB2C     )
+  THREADError    : N906D             N906D | N906 | Abend
  POOL THREAD ATTRIBUTES
   ACcountrec     : None              None | TXid | TAsk | Uow
   AUTHId         :
   AUTHType       : Userid            Userid | Opid | Group | Sign | TErm
                                      | TX
   DRollback      : Yes               Yes | No
   PLAN           :
   PLANExitname   : DSNCUEXT
   PRiority       : High              High | Equal | Low
   THREADLimit    : 0003              3-2000
   THREADWait     : Yes               Yes | No
  COMMAND THREAD ATTRIBUTES
   COMAUTHId      :
   COMAUTHType    : Userid            Userid | Opid | Group | Sign | TErm
                                      | TX
   COMThreadlim   : 0001              0-2000

                                                  SYSID=CICS APPLID=CICS

PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Abbildung 5-8: Definition einer DB2CONN – Bildschirm 2

Dabei ist besonders zu beachten, dass als `DB2Id` der korrekte Name des DB2-Subsystems (in diesem Fall `DSN1`) eingegeben wird. Als `AUTHType` und `COMAUTHType` soll `Userid` verwendet werden.

Nun musste die Gruppe mit Hilfe von `CEDA INSTALL GROUP(GSCHLU)` neu installiert werden. Danach muss mit Hilfe des Befehls `CEMT SET DB2CONN CONNECTED` die Verbindung zu DB2 hergestellt werden (Abbildung 5-9).

```
 SET DB2CONN CONNECTED
 STATUS:  RESULTS - OVERTYPE TO MODIFY                          NORMAL
   Accountrec( None )             Planexitname( DSNCUEXT )
   Authid(          )             Priority( High )
   Authtype( Userid )             Purgecyclem( 00 )
   Comauthid(          )          Purgecycles( 30 )
   Comauthtype( Cuserid )         Resyncmember(              )
   Comthreadlim( 0001 )           Signid( CICS     )
   Comthreads(0000)               Security(              )
   Connecterror( Sqlcode )        Standbymode( Reconnect )
   Connectst( Connected )         Statsqueue( CDB2 )
   Db2groupid(      )             Tcblimit( 0012 )
   Db2id( DSN1 )                  Tcbs(0000)
   Db2release(0710)               Threaderror(N906d)
   Drollback(Rollback)            Threadlimit( 0003 )
   Msgqueue1( CDB2 )              Threads(0000)
   Msgqueue2(      )              Threadwait( Twait )
   Msgqueue3(      )
   Nontermrel( Release )
   Plan(          )

                                                  SYSID=CICS APPLID=CICS
  RESPONSE: NORMAL                          TIME:  17.08.05  DATE: 10.29.04
PF 1 HELP       3 END       5 VAR       7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

Abbildung 5-9: Aktivieren der Datenbankverbindung

*Installation des CICS-Programms*

Nun wurde das CICS-Programm mit Hilfe von `CEDA DEFINE PROGRAM(ECIDB2) GROUP(GSCHLU)` definiert. Als `Language` wurde `Le370` angegeben (Abbildung 5-10).

```
 OVERTYPE TO MODIFY _                                        CICS RELEASE = 0620
  CEDA  DEFine PROGram( ECIDB2   )
   PROGram        : ECIDB2
   Group          : GSCHLU
   DEscription  ==> CICS DB2 PROGRAM
   Language     ==> Le370              CObol | Assembler | Le370 | C | Pli
   RELoad       ==> No                 No | Yes
   RESident     ==> No                 No | Yes
   USAge        ==> Normal             Normal | Transient
   USElpacopy   ==> No                 No | Yes
   Status       ==> Enabled            Enabled | Disabled
   RSl            : 00                 0-24 | Public
   CEdf         ==> Yes                Yes | No
   DAtalocation ==> Below              Below | Any
   EXECKey      ==> User               User | Cics
   COncurrency  ==> Quasirent          Quasirent | Threadsafe
  REMOTE ATTRIBUTES
   DYnamic      ==> No                 No | Yes
+  REMOTESystem ==>

                                                  SYSID=CICS APPLID=CICS
  DEFINE SUCCESSFUL                          TIME:  17.21.57  DATE: 04.303
PF 1 HELP 2 COM 3 END           6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Abbildung 5-10: Definition des CICS-DB2-Programms

*Definition des DB2ENTRY*

Es muss nun noch ein DB2ENTRY erstellt werden, um Threads für die Verbindung zur Datenbank bereitzustellen. Die Erstellung des `DB2ENTRY` namens `DB2COMM` geschah mittels `CEDA DEFINE DB2ENTRY(DB2COMM) GROUP(GSCHLU)` (Abbildung 5-11).

```
 VIEW DB2ENTRY(DB2COMM) GROUP(GSCHLU)
 OBJECT CHARACTERISTICS                                      CICS RELEASE = 0620
  CEDA  View DB2Entry( DB2COMM  )
   DB2Entry       : DB2COMM
   Group          : GSCHLU
   DEscription    :
  THREAD SELECTION ATTRIBUTES
   TRansid        : *
  THREAD OPERATION ATTRIBUTES
   ACcountrec     : None               None | TXid | TAsk | Uow
   AUTHId         : GSCHLU
   AUTHType       :                    Userid | Opid | Group | Sign | TErm
                                       | TX
   DRollback      : Yes                Yes | No
   PLAN           : DB2C
   PLANExitname   :
   PRIority       : High               High | Equal | Low
   PROtectnum     : 0000               0-2000
   THREADLimit    : 0000               0-2000
   THREADWait     : Pool               Pool | Yes | No

                                                  SYSID=CICS APPLID=CICS

PF 1 HELP 2 COM 3 END           6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Abbildung 5-11: Attribute des DB2ENTRY

Dabei ist als PLan der Name DB2C anzugeben. Als TRansid wurde * gewählt, so dass für jede Transaktion, innerhalb der das Programm ECIDB2 abläuft, Threads zum Zugriff auf DB2 reserviert werden. Dies ist notwendig, da das Programm sowohl von der CECI-Transaktion als auch vom CICS Transaction Gateway gestartet wird, so dass es sich schon um mindestens zwei verschiedene Transaktionen handelt. Als AUTHId wurde eine Benutzer-ID gewählt, die auf DB2 zugreifen darf.

Nun konnte die Gruppe mit Hilfe von CEDA INSTALL GROUP(GSCHLU) neu installiert werden. Mögliche Warnungen aufgrund einer schon installierten und aktiven DB2CONN können ignoriert werden.

*Test innerhalb von CICS*

Um für den späteren Test mit der J2EE-Anwendung mögliche Fehler in CICS auszuschließen, soll das erstellte CICS-DB2-Programm innerhalb von CICS getestet werden. Dies geschieht wie schon in *Abschnitt 4.2.4* mit Hilfe der CECI-Transaktion. Diesmal lautet der Befehl CECI LINK PROGRAM(ECIDB2) COMMAREA(' ') LENGTH(160). Die Länge von 160 ist notwendig, da vier Datensätze mit je 40 Zeichen abgerufen werden. Die Ausgabe ist in Abbildung 5-12 dargestellt.

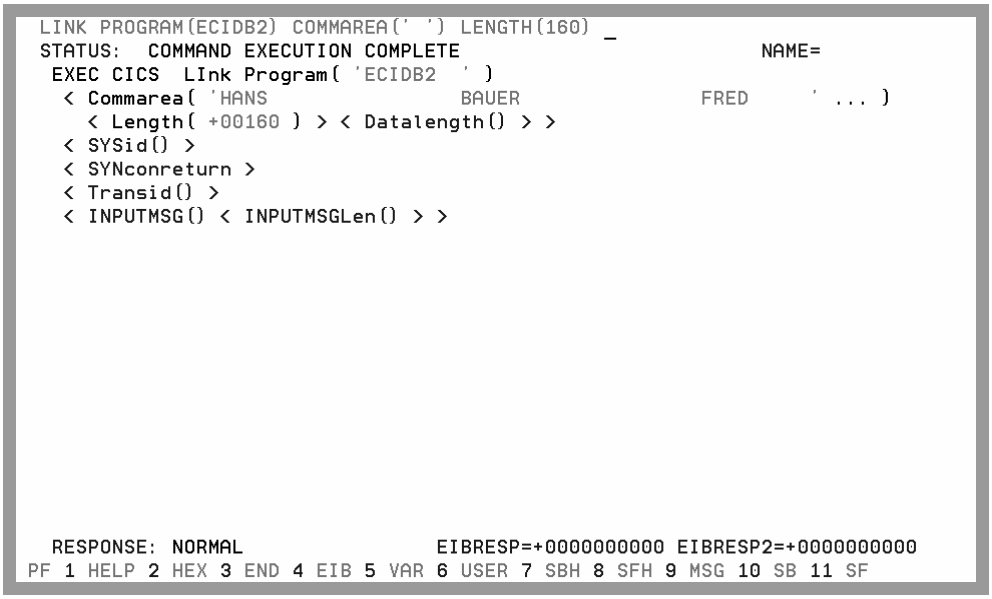


Abbildung 5-12: Test von ECIDB2 mit Hilfe der CECI-Transaktion

Wie erwartet werden die Namen aus der Datenbank dargestellt. Es sind jedoch nicht alle Namen zu sehen, da die Ausgabezeile nicht lang genug ist. Damit ist die Funktionstüchtigkeit

des CICS-Programms sichergestellt, nun soll dieses mit Hilfe der J2EE-Anwendung über das CICS Transaction Gateway getestet werden.

## 5.3 Test mit Hilfe der J2EE-Anwendung

Wiederum soll die in *Abschnitt 4.4.3* installierte J2EE-Anwendung `CTGTesterCCI` eingesetzt werden. Diesmal sollen die Standardwerte im Formular der Startseite verändert werden (Abbildung 5-13).

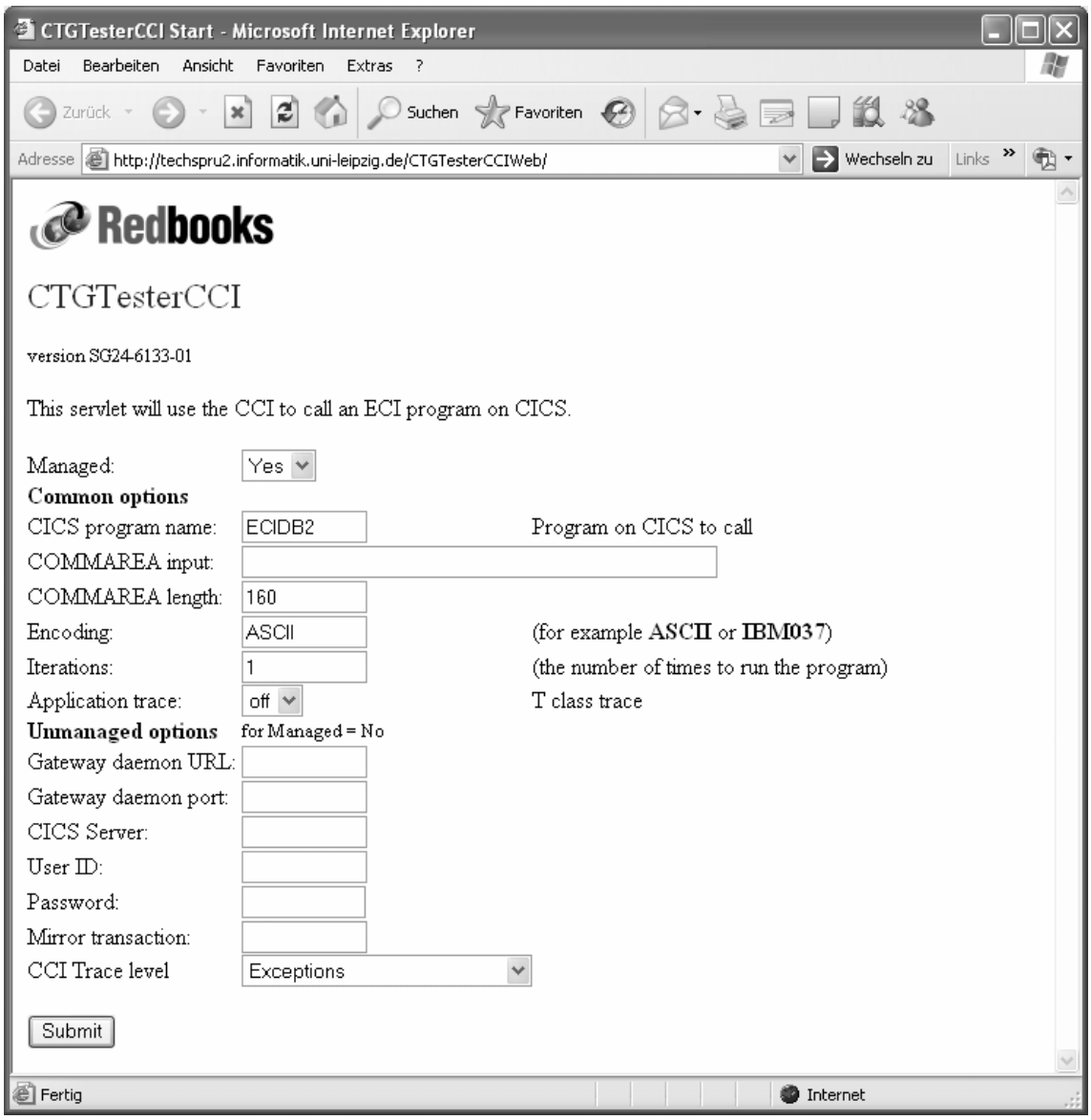


Abbildung 5-13: Die Startseite der J2EE-Beispielanwendung mit angepassten Parametern

Analog zur CECI-Transaktion muss als CICS-Programmname `ECIDB2` und als Länge `160` angegeben werden. Mit Hilfe der Schaltfläche `Submit` wird die Ausführung in Gang gesetzt. Die Ergebnisse sind in Abbildung 5-14 dargestellt.

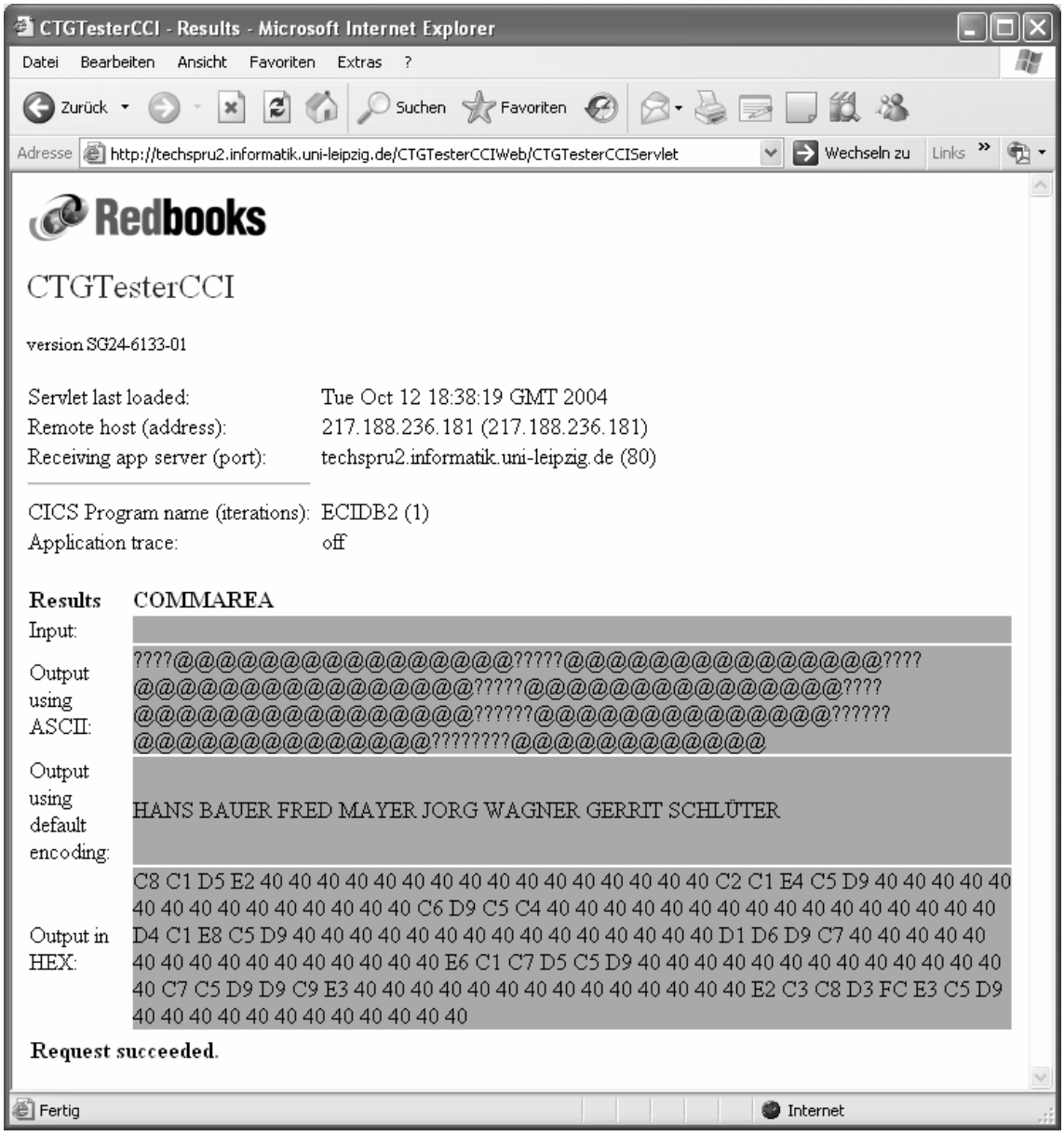


Abbildung 5-14: Die Ergebnisseite der J2EE-Anwendung für `ECIDB2`

Die Daten aus der Datenbank werden in der Zeile der Standardkodierung ausgegeben. Damit war der Test erfolgreich. Die Ausgabe in ASCII ist nicht lesbar, da für das Programm `ECIDB2` kein Eintrag in das Makro `DFHCNV` eingefügt wurde (siehe *Anhang A: Datenkonvertierung in CICS*). Die Daten in der Zeile der Standardkodierung wurden vom J2EE-Programm umgewandelt. Die hexadezimale Ausgabe zeigt, dass zwischen den

einzelnen Namen die korrekte Anzahl an Leerzeichen vorhanden ist. Diese werden jedoch bei der Ausgabe in den anderen Zeilen jeweils zu einem Leerzeichen zusammengefasst.

Somit wurde gezeigt, dass mit Hilfe einer J2EE-Anwendung über das CICS Transaction Gateway eine Verbindung zu einem CICS-Programm hergestellt werden kann, das wiederum Daten aus einer DB2-Datenbank ausliest. Dies ist neben JDBC eine weitere Möglichkeit zum Datenbankzugriff in J2EE-Programmen. Jedoch ist allein das CICS-Programm für den Datenbankzugriff zuständig, so dass der volle DB2-Funktionsumfang nicht direkt von der J2EE-Anwendung aus gesteuert werden kann.

*Kapitel 6*

# Zusammenfassung und Ausblick

In *Kapitel 2* wurden die verwendeten Softwarekomponenten vorgestellt. Es handelte sich um ein z/OS-Betriebssystem, auf dem vor allem CICS Transaction Server, CICS Transaction Gateway, WebSphere Application Server und DB2 zum Einsatz kamen.

Daraufhin wurde in *Kapitel 3* beschrieben, wie diese verschiedenen Softwarekomponenten miteinander interagieren, um mit Hilfe der J2EE-Umgebung über das CICS Transaction Gateway einen Webzugriff auf Anwendungsprogramme des CICS Transaction Server zu erlangen.

In *Kapitel 4* wurde diese Konfiguration erfolgreich erstellt, wobei die einzelnen Schritte zur Konfiguration genau beschrieben und die Zwischenergebnisse jeweils getestet wurden. Zuerst wurde der Zugriff auf CICS ermöglicht und das CICS Transaction Gateway installiert. Daraufhin wurde der WebSphere Application Server konfiguriert, und die Konfiguration wurde mit Hilfe einer J2EE-Anwendung getestet.

In *Kapitel 5* sollte der Datenbankzugriff über das CICS Transaction Gateway ermöglicht werden. Dazu wurde erfolgreich ein CICS-DB2-Programm erstellt und mit dem J2EE-Programm darauf zugegriffen.

Die vorgestellte z/OS-Installation erfüllte fast alle Anforderungen, insbesondere eine lauffähige WebSphere-Umgebung. Lediglich das CICS Transaction Gateway war nicht vorhanden und musste erst installiert werden.

Die meiste Zeit entfiel auf die Einarbeitung in das z/OS-System, wobei die OS/390-Tutorials der Universität Leipzig [TUT] eine große Hilfe waren. Die meisten Probleme bereitete die Einrichtung der Kommunikation mit CICS, bei der einige Fehler auftraten, die zum Teil nicht vollständig dokumentiert waren.

Die nun auf dem z/OS-System vorliegende Konfiguration bildet eine gute Grundlage für weitere Untersuchungen oder Studienarbeiten:

- Erstellung einer neuen J2EE-Anwendung zum Zugriff auf DB2 über das CICS Transaction Gateway, so dass neben dem Anzeigen des Inhalts der Datenbank auch

das Einfügen und Löschen von Datensätzen möglich ist. Dabei kann auch das vom CICS Transaction Gateway unterstützte Zwei-Phasen-Commit-Protokoll eingesetzt werden, um die Konsistenz der Änderungen sicherzustellen.

- Einrichtung der verschiedenen Sicherheitsmechanismen zur Benutzerautorisierung mit RACF. Außerdem können Technologien wie SSL (Secure Sockets Layer) und *PKI (Public Key Infrastructure)* eingesetzt werden.

- Anwendung der Konfigurationsschritte auf eine neuere z/OS-Version in Verbindung mit einem CICS Transaction Gateway ab Version 5.1. Insbesondere wäre auch die Einrichtung des WebSphere Application Server V5 interessant, da hier die Konfiguration über eine Webschnittstelle erfolgt und die `RAR`-Dateien der Ressourcenadapter direkt installiert werden können.

- Einrichtung einer verteilten Umgebung zum Zugriff auf CICS (Topologien 1 und 2 aus Kapitel 3). Dabei ist mit Hilfe des CICS-EPI-Ressourcenadapters auch der Zugriff auf vorhandene CICS-Programme möglich, die zur Ausgabe BMS-Bildschirme einsetzen.

- Erstellung eines Tutorials, in dem die Benutzung des CICS Transaction Gateway in Verbindung mit der J2EE-Plattform Schritt für Schritt erklärt wird.

*Anhang A*

# Datenkonvertierung in CICS

Falls die Datenkonvertierung von EBCDIC in ASCII und umgekehrt in CICS stattfinden soll, so geschieht dies durch das CICS-Datenkonvertierungsprogramm `DFHCCNV` in Kombination mit dem Makro `DFHCNV`. Eine Alternative dazu ist die Konvertierung im Java-Programm, wie es zum Beispiel in der J2EE-Anwendung `CTGTesterCCI` der Fall ist.

## A.1 ECI-Anwendungen

ECI-Anwendungen verwenden das CICS-Mirrorprogramm, um eine Verbindung zum angegebenen CICS-Programm herzustellen, wobei für Ein- und Ausgabe ein als COMMAREA bekannter Puffer eingesetzt wird. Das Mirrorprogramm (`DFHMIRS`) ruft die Dienste des Datenkonvertierungsprogramms `DFHCCNV` auf, um die Konvertierung der COMMAREA durchzuführen (Abbildung A-1).

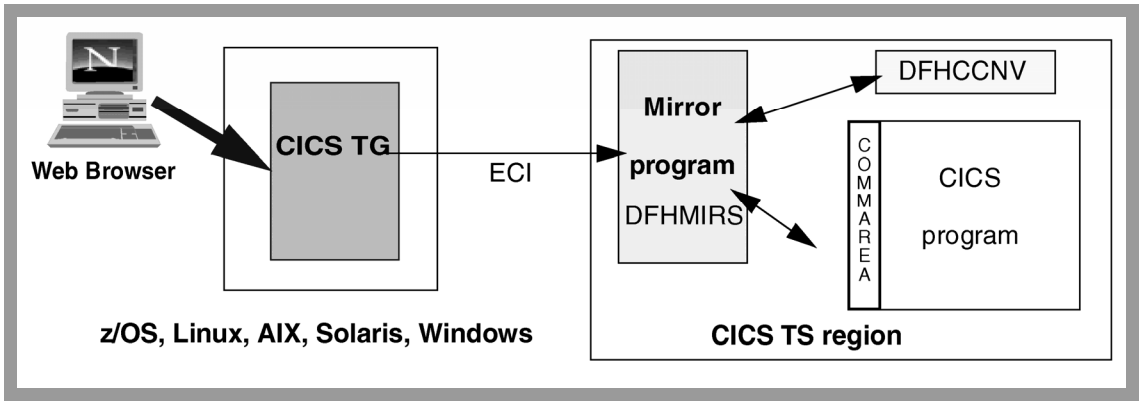


Abbildung A-1: ECI-Datenkonvertierung [CG02b]

Jedoch nur dann, wenn `DFHCCNV` in der Tabelle `DFHCNV` eine Konvertierungsvorlage findet, die dem aufgerufenen CICS-Programm entspricht, wird eine Konvertierung der COMMAREA durchgeführt.

## A.2 Java in z/OS

Java-Programme in z/OS, die auf die ECI-Methoden des CICS-Transaction Gateway zugreifen, stellen eine Besonderheit dar. Java-Strings sind in Unicode abgespeichert, während Java-Bytearrays in der nativen Codepage des Betriebssystems vorliegen. Die in der ECI-Anforderung an CICS übergebene COMMAREA ist ein solcher Bytearray, wird jedoch

oftmals aus einem String erstellt. Daher muss in einem Java-Programm unter z/OS bei der Konvertierung eines String in einen Bytearray die Kodierung angegeben werden, wie in dem folgenden Codebeispiel:

```
byte abCommarea[] = new byte[];
abCommarea="abcd".getBytes("8859_1");
```

Abbildung A-2 zeigt den Ablauf der Datenkonvertierung in einer solchen Umgebung.

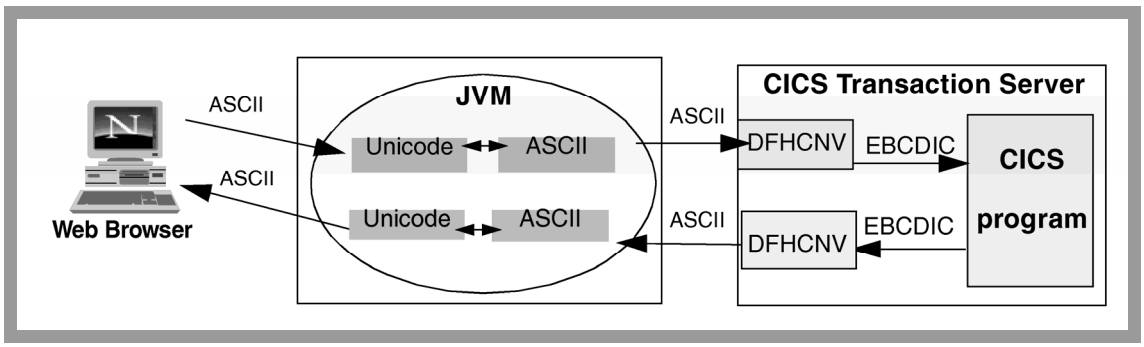


Abbildung A-2: Konvertierung in einer Umgebung mit Java unter z/OS [CG02b]

## A.3 Vorgehensweise

Ähnlich wie die Tabelle `DFHXCOPT` muss auch `DFHCNV` zunächst modifiziert werden, woraufhin die Prozedur `DFHAUPLE` eingesetzt wird, um aus der Tabelle ein Lademodul zu erstellen (siehe *Abschnitt 4.1.4*).

In diesem Fall wurde einfach das Beispielmodul für CICS-Webfunktionen `DFHCNVW$` aus `CICSTS22.CICS.SDFHSAMP` in ein eigenes Dataset kopiert und dort modifiziert. Für das CICS-Programm `EC01` muss zum Beispiel ein Eintrag wie in Abbildung A-3 in `DFHCNVW$` eingefügt werden. Der Einfachheit halber kann auch ein vorhandener Eintrag angepasst werden.

```
DFHCNV TYPE=ENTRY,RTYPE=PC,RNAME=EC01,USREXIT=NO,                *
      SRVERCP=037,CLINTCP=850
DFHCNV TYPE=SELECT,OPTION=DEFAULT
DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=18,         *
      LAST=YES
```

Abbildung A-3: `DFHCNV`-Eintrag für das Programm `EC01`

Dabei muss als RNAME das CICS-Programm – EC01 – eingegeben werden. Als Server-Codepage (SRVERCP) muss 037 (EBCDIC) und als Client-Codepage (CLINTCP) 850 (ASCII) eingeben werden. DATALEN entspricht der Länge der vom CICS-Programm zurückgelieferten COMMAREA – in diesem Fall 18.

Nun muss diese neu erstellte DFHCNV-Tabelle mit Hilfe der Prozedur DFHAUPLE in ein Lademodul übersetzt werden (Abbildung A-4).

```
000100 //GSCHLUC  JOB (),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
000200 //MYLIBS1  JCLLIB ORDER=CICSTS22.XDFHINST
000300 //ASMTAB   EXEC DFHAUPLE,NAME=SDFHLOAD
000400 //ASSEM.SYSUT1 DD DSN=GSCHLU.CICS.EXCI(DFHCNVW$),DISP=SHR
```

Abbildung A-4: JCL-Programm zum Übersetzen von DFHCNV

Dabei befand sich die Prozedur DFHAUPLE im Dataset CICSTS22.XDFHINST. Das Ziel des erstellten Lademoduls ist CICSTS22.CICS.SDFHLOAD, was durch den Eintrag NAME=SDFHLOAD festgelegt wird.

Nachdem der Job erfolgreich abgearbeitet wurde, musste innerhalb von CICS der Befehl CEMT PERFORM SHUTDOWN IMMEDIATE ausgeführt werden, um CICS herunterzufahren. In SDSF wurde CICS dann mit Hilfe von /S CICSA neu gestartet.

*Anhang B*

# Inhalt der beiliegenden CD

| ***Verzeichnis*** | ***Inhalt*** |
|---|---|
| /Literatur/Bücher | Alle in elektronischer Form verfügbaren Bücher aus dem Literaturverzeichnis |
| /Literatur/Internet | Alle Internetquellen aus dem Literaturverzeichnis |
| /Tools | Windows-Tools für den WebSphere Application Server V4 für z/OS (Application Assembly Tool und SMEUI) |
| /Abbildungen | Alle Abbildungen dieser Diplomarbeit |
| /Dokument | Dieses Dokument als DOC- und PDF-Datei |
| /Beispiele/CICS | CICS-Programme und Quellen der Lademodule sowie JCL-Dateien zur Übersetzung |
| /Beispiele/CTG | Konfigurationsdateien und JCL-Datei zum Starten des Gateway-Daemon sowie Skriptdateien zum Starten der Beispielprogramme |
| /Beispiele/HTTP | Konfigurationsdatei des IBM HTTP Server |
| /Beispiele/J2EE | J2EE-Beispielanwendung aus dem IBM-Redbook [CG02b] |
| /Beispiele/WebSphere | Konfigurationsdateien des WebSphere Application Server für z/OS sowie JCL-Datei zum Start des J2EE-Servers |

Tabelle B-1: Inhalt der beiliegenden CD

LITERATURVERZEICHNIS

## Referenzen aus Zeitschriften und Büchern


[CG02a] *CICS Transaction Gateway: z/OS Administration V5.0*. IBM Form No. SC34-6191-01, Juli 2002

[CG02b] *CICS Transaction Gateway V5: The WebSphere Connector for CICS*. IBM Form No. SG24-6133-01, August 2002

[CG02c] *CICS Transaction Gateway: Programming Guide V5.0*. IBM Form No. SC34-6141-00, Juli 2002

[CS02a] *CICS External Interface Guide*. IBM Form No. SC34-6006-00, Januar 2002

[CS02b] *CICS Messages and Codes*. IBM Form No. GC34-6003-00, Januar 2002

[CS02c] *CICS DB2 Guide*. IBM Form No. SC34-6014-00, Januar 2002

[DC01] B. DuCharme: *The Operating Systems Handbook*, 2001

[HKS03] P. Herrmann, U. Kebschull, W.G. Spruth: *Einführung in z/OS und OS/390*. Oldenbourg, 2003

[Jon04] J. Jones: *The Big Picture: IBM DB2 Information Management Software and DB2 Universal Database*. IBM Data Management Solutions, Februar 2004

[RÖ99] E. Roman, R. Öberg: *The Technical Benefits of EJB and J2EE Technologies over COM+ and Windows DNA*. Artikel der Middleware Company, Dezember 1999

[SS02] S. Schäffer, W. Schilder: *Enterprise Java mit IBM WebSphere*. Addison Wesley, 2002

[WS02] *WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management User Interface*. IBM Form No. SA22-7838-04, September 2002


## Internet-Referenzen


[Ank01] T. Scott Ankrum: *The Evolution of CICS 30 Years Old and Still Modern*. http://objectz.com/columnists/tscott/evolution.doc, Januar 2001

[CEM] IBM: *DFHIR3780 with return code x'D8' when opening IRC*. http://www-1.ibm.com/support/docview.wss?uid=swg21107283

[CGE] IBM: PQ65823: *SETTING CTG RRMNAME IN CTGENVVAR DOES NOT TAKE EFFECT* http://www-1.ibm.com/support/docview.wss?uid=swg1PQ65823

[CGL] IBM: *CICS Transaction Gateway Version 5 Library*. http://www-306.ibm.com/software/htp/cics/library/cicstgv5.html

[CI04] IBM: *CICS - An Introduction: Customer Information Control System – 35 years of reliable transactions*. ftp://service.boulder.ibm.com/software/htp/cics/PDF/cics_introduction.pdf

[CS02d] IBM: *CICS Transaction Server for z/OS: An Enterprise Solution for Enterprise Java.* http://www-306.ibm.com/software/htp/cics/library/whitepapers/websphereandcicswhitepaperfinal.pdf

[CSL] IBM: *CICS Transaction Server for z/OS Version 2.2 Library.* http://www-306.ibm.com/software/htp/cics/library/cicstsforzos22.html

[CWA] IBM: *WebSphere Application Server V4.0 and V4.0.1 for z/OS and OS/390: Configuring Web Applications.* ftp://ftp.software.ibm.com/software/mktsupport/techdocs/was4_configuring_web_apps.pdf

[DBL] IBM: *DB2 for z/OS and OS/390 - DB2 Version 7 Library.* http://www-306.ibm.com/software/data/db2/zos/v7books.html

[FCC] IBM: FAQ - *Why do we receive the error "ClassCastException" when trying the CTGTesterCCI sample under CICS Transaction Gateway for z/OS?* http://www.developer.ibm.com/tech/faq/individual?oid=2:79825

[Hen01] Hendela: *IBM's Customer Information Control System, CICS.* http://www.hendela.com/pptpackage/cics/CICSOverview.ppt, 2001

[IWC] IBM: *Integrating WebSphere Application Server and CICS using the J2EE Connector Architecture*, Januar 2004. http://publibfp.boulder.ibm.com/epubs/pdf/22472180.pdf

[Sun00] Sun Microsystems: *J2EE Connector Architecture Promises to Simplify Connection to Back-End Systems* http://java.sun.com/j2ee/connector/giga/RPA-112000-00018.htm, 2000

[Sun04] Sun Microsystems: *Java 2 Platform, Enterprise Edition (J2EE) Overview.* http://java.sun.com/j2ee/overview.html, 2004

[TUT] Uni Leipzig. *OS/390 Tutorials.* http://jedi.informatik.uni-leipzig.de/tutor.html

[TUT4] Uni Leipzig: *OS/390 Tutorial 4 – DB2.* http://jedi.informatik.uni-leipzig.de/tutors/tutor4.pdf

[TUT5] Uni Leipzig: *OS/390 Tutorial 5 – Datenbankzugriff mit CICS (COBOL).* http://jedi.informatik.uni-leipzig.de/tutors/tutor5b.pdf

[Win03] Winter Corporation: *2003 TopTen Award Winners*, 2003. http://www.wintercorp.com/vldb/2003_TopTen_Survey/All Winners.pdf

[WS01] IBM: *Introducing the IBM WebSphere Application Server Version 4.0 for z/OS an OS/390*, 2001. http://www-1.ibm.com/servers/eserver/zseries/library/whitepapers/pdf/gf225146.pdf,

[WSD] IBM: *WebSphere Application Server for z/OS and OS/390 – Download.* http://www-306.ibm.com/software/webservers/appserv/download_v4z.html

[WSL] IBM: *WebSphere Application Server for z/OS - Library.* http://www-306.ibm.com/software/webservers/appserv/zos_os390/library/

# Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, 16. November 2004