Universität Leipzig
Institut für Informatik
Abteilung Computersysteme
Prof. Dr.-Ing. Wilhelm G. Spruth

UNIVERSITÄT LEIPZIG

Diplomarbeit zum Thema

Erstellen der J2EE-Anwendung mit Zugriff auf DB2 über das CICS Transaction Gateway

vorgelegt von Pavel Selesnjov
Leipzig, Oktober 2006

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Programmverzeichnis

Abkürzungsverzeichnis

COMMAREA	Communication area
DBRM	Database Request Module
EAR	Enterprise Application Archive
ECI	External Call Interface
EPI	External Presentation Interface
ESI	External Security Interface
EXCI	External CICS Interface
IRC	Interregion Communication
ISPF	Interactivity System Product Facility
JCA	J2EE Connector Architecture
JCL	Job Control Language
JES	Job Entry Subsystem
JNDI	Java Naming and Directory Interface
MRO	Multiregion Operation
OMVS	Open Multiple Virtual Storage
RACF	Resource Access Control Facility
RAD	Rational Application Developer
RAR	Resource Adapter Archive
RSDP	Rational Software Development Platform
SDSF	Spool Search and Display Facility
TSO	Time Sharing Option
WAR	Web Application Archive
WAS	WebSphere Application Server
WDz	WebSphere Developer for zSeries
WSADIE	WebSphere Studio Application Developer Integration Edition
WSDL	Web Service Description Language
WSIF	Web Services Invocation Framework

Danksagung

An dieser Stelle möchte ich mich bei allen Leuten bedanken, die bei der Diplomarbeit mich mit ihrer Hilfe und Betreuung unterstützt haben. Ein besonderer Dank geht an meinen Betreuer Dr. rer. nat. Paul Herrmann. Nicht zuletzt möchte ich mich auch bei meinen Eltern und bei meiner Frau bedanken, die mich über das ganze Studium hinweg unterstützt haben.

Kapitel 1 Einleitung

1.1 Motivation

Obwohl Mainframe-Systeme als „*Relikt*“ aus vergangenen Zeiten bezeichnet werden, dienen sie immer noch vielen Unternehmen als Basis für geschäftskritische Anwendungen. Als Marktführer ist hier IBM S/390-Großrechner mit seinem OS/390-Betriebssystem. In seiner Basisstruktur unterscheidet sich OS/390 nicht von einem modernen Betriebssystem. Sie basiert auf dem herkömmlichen 3-Schichtenmodell und besteht aus Hardware, Betriebssystem und Benutzer-Prozessen [HWG03]. Zwischen dem eigentlichen OS/390-Betriebssystem und den Benutzer-Prozessen werden ähnlich wie z.Bsp. bei Windows-Betriebssystemen verschiedene Subsysteme eingeschoben. Für die Transaktionsverarbeitung ist CICS (*Customer Information Control System*)-Transaktions-Server eingesetzt, der am weitesten verbreitete Transaktions-Monitor ist. Immer mehr Großunternehmen entscheiden sich für die S/390 Architektur und setzen CICS als Transaktions-Monitor (TM) ein. Weltweit generieren sie täglich Milliarden von Transaktionen über CICS mit einem Volumen von mehreren Billionen US-Dollar. Die wichtigsten Aufgaben des CICS-TM sind die Verwaltung und die Steuerung von Transaktionen und Management von Ressourcen.

Wachsender Welt des E-Business erfordert die Anpassung der Anwendungen und Daten von Backend-Systemen, wie z.Bsp. CICS, IMS oder SAP, an moderne Web-Umgebungen. Für die Integration von CICS-Anwendungen in das Internet existieren verschiedene Möglichkeiten, fünf Wichtigsten davon sind:

- CICS Transaction Gateway (CICS TG)
- CICS Web Support (CWS)
- CICS Enterprise JavaBeans (EJB)
- CICS Link3270 bridge
- SOAP for CICS feature

Das CICS TG stellt ein wichtiges Bindeglied zwischen den J2EE-Komponenten und CICS dar. In Verbindung mit WebSphere Application Server (WAS) bietet er sichere, leistungsfähige und skalierbare Konnektivität zwischen den Web-Anwendungen und CICS an. Die Implementierung der CICS Transaction Gateway Software erfordert minimale Änderungen zum CICS System und normalerweise keine Änderungen zu den bestehenden CICS-Anwendungen. Dieser Ansatz bildet den Gegenstand dieser Arbeit und wird im weiteren Verlauf vorliegender Diplomarbeit näher beschrieben.

1.2 Aufbau der Arbeit

Diese Diplomarbeit beschäftigt sich mit der Erstellung der J2EE-Anwendung und dem Zugriff dieser Anwendung auf DB2-Datenbank.

Die Anwendung wird in der so genannten *drei-Tier*-Konfiguration implementiert. Hierbei beinhaltet das CICS-Programm die Anwendungs-Logik, dessen Zweck das Ändern und Auslesen von Datensätzen der Datenbank ist. Die Präsentations-Logik übernimmt die J2EE-Anwendung, die typischerweise mit einem Web-Anwendungsserver realisiert wird. Die Web-Anwendung wird an CICS mit Hilfe des Java-Konnektors, dem *CICS Transaction Gateway*, angebunden.

Außerdem wird das Zusammenspiel des WebSphere Anwendungsservers, CICS Transaction Gateway und CICS in zwei Konfigurationen untersucht. Es werden auch einige Sicherheitsmechanismen zur Benutzerautorisierung und -authentifizierung vorgestellt.

Kapitel 2 verwendete Software und Technologien

Dieser Kapitel gibt kurze Anleitung zur J2EE Connector Architecture, die aufzeigt, wie J2EE-Anwendungsserver über Konnektoren (in diesem Fall CICS Transaction Gateway [CICS TG]) mit externen Enterprise-Information-Systemen (EIS) kommunizieren können. Außerdem wird die Struktur des CICS TG und die wichtige Merkmale der Entwicklungsplattform WSADIE (*WebSphere Studio Application Developer Integration Edition*) erläutert.

2.1 J2EE Konnektor Architektur

JCA (*Java 2 Enterprise Edition Connector Architecture*) ist eine von Sun herausgegebene Spezifikation, welche eine Standardarchitektur für die Anbindung von Informationssystemen, in der J2EE Terminologie *Enterprise Information Systems* (EIS) genannt, an die webbasierten Anwendungen definiert. Die aktuelle Version der JCA-Spezifikation ist 1.5, die unter [Sun] zu finden ist.

JCA besteht aus drei Hauptelementen: dem Ressourcenadapter, der System Contracts und der Client-API. Der Ressourcenadapter[1] bildet dabei den Kern der Architektur. Er stellt einen Software-Treiber auf Systemebene dar, der von einem Anwendungsserver als auch von einer Anwendungskomponente verwendet werden kann, um eine Verbindung zum EIS aufzubauen. Für die Erstellung des Adapters sind die jeweiligen EIS-Hersteller zuständig.

Der Ressourcenadapter kann in zwei Umgebungen eingesetzt werden, welche in JCA Spezifikation beschrieben sind. Erste ist die nicht verwaltete Umgebung (**non managed environment**). Hier wird kein Anwendungsserver benötigt. Die Java-Anwendungen verwenden Ressourcenadapter direkt, um auf ein EIS zuzugreifen. In diesem Fall kann der Verbindungspool vom Ressourcenadapter bereitgestellt werden oder die Client-Anwendung selbst verwaltet die Verbindungen zum EIS.

Die zweite Umgebung, welche den Schwerpunkt der JCA-Spezifikation bildet, ist eine so genannte verwaltete Umgebung (**managed environment**), die in dieser Arbeit näher untersucht wird (*s. Abbildung 2.1 auf der nächsten Seite*). Im Gegensatz zur nicht verwalteten Umgebung sind die J2EE-Anwendungskomponente und der Ressourcenadapter über so genannte Kontrakte (**contracts**) mit einem Anwendungsserver verbunden. Die Entwickler des Anwendungsservers müssen nur einmalig die *Connector-Architecture* in ihr Produkt integrieren, um beliebige Ressourcenadapter einbinden zu können. Dadurch kann der Anwendungsserver jedes beliebigen EIS, vorausgesetzt es ist dafür ein Adapter erhältlich, in seiner Umgebung integrieren. Entsprechend müssen auch die

[1] Die Ressourcenadapter werden auch Konnektoren genannt.

EIS-Hersteller sich an die JCA-Standards halten, um ihre Ressourcenadapter in jeden Anwendungsserver einbetten zu können.

Der Anwendungsserver und der Ressourceadapter einigen sich auf die Form der Kommunikation über System-Contracts. Der Ressourcenadapter implementiert dabei diese Kontrakte stellvertretend für das zu integrierende EIS. Hierzu gehören drei wichtige System-Kontrakte (*s. Abbildung 2.1*).

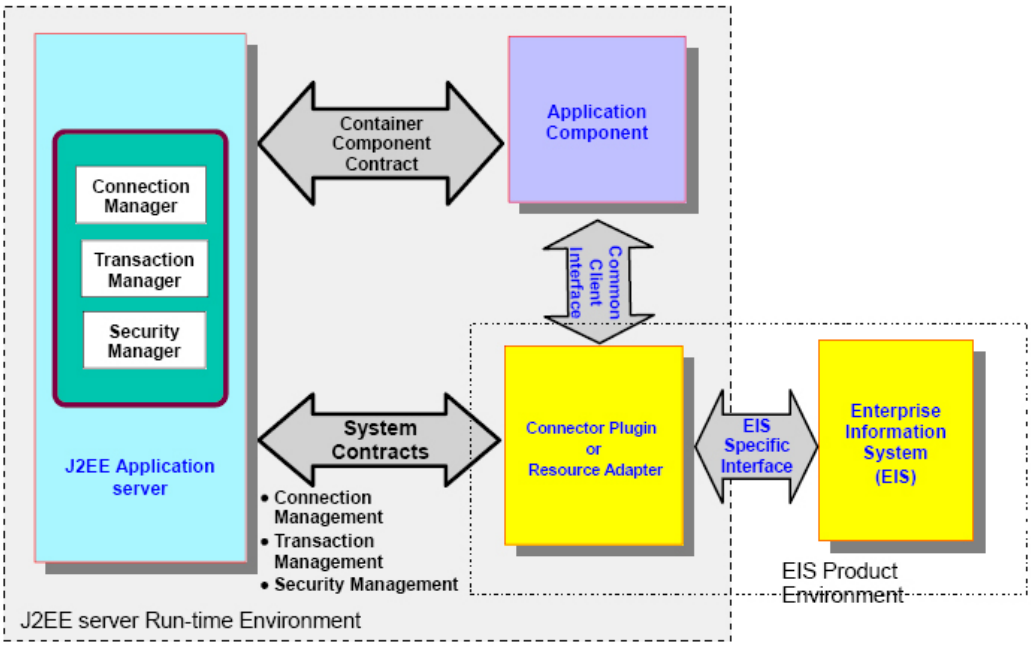


Abbildung 2.1: Komponenten der JCA-Architektur.

- Kontrakt für das Verbindungsmanagement
  ermöglicht dem Anwendungsserver, einen Verbindungspool zum EIS aufzubauen. Die Verwendung vom Verbindungspool sorgt für besseren Verbindungsaufbau, was zu einer großen Anzahl von Anwendungskomponenten beim Aufbau einer Verbindung zum EIS führen kann.

- Kontrakt für das Transaktionsmanagement
  Der Kontrakt erlaubt einen transaktionssicheren Zugriff auf den Ressourcenmanager eines EIS. Dazu verwendet ein Anwendungsserver einen Transaktionsmanager, um die Transaktionen über mehrere Ressourcenmanager eines EIS zu verwalten.

- Kontrakt für das Sicherheitsmanagement
  Dieser Kontrakt ist für einen sicheren Zugriff auf ein EIS zuständig. Die Informationen, die zwischen dem Anwendungsserver und einem EIS ausgetauscht werden, können mittels *container-managed* oder *component-managed* Verfahren geschützt werden. Bei dem *container-managed* Verfahren ist der Web Anwendungsserver selbst für die übergebene Sichetheitsinformationen zu EIS zuständig. Bei dem *component-managed* Mechanismus ist es die Anwendung, die für die zu schützende Informationen verantwortlich ist.

Die J2EE-Anwendungen können aus mehreren Komponenten bestehen, die beispielsweise En-

terprise Java Beans (EJBs), Java Server Pages (JSPs) oder Java Servlets sein können. Eine oder mehrere Anwendungskomponenten bilden dann ein Modul, der im entsprechenden Container eines Anwendungsservers installiert und ausgeführt werden kann. Die Container können folgende sein:

- **Web-Container**
  enthält Web-Module, die aus Servlets, JSP-Dateien und statischen HTML-Seiten zusammengesetzt sind.
- **Enterprise-Bean-Container**
  enthält EJB-Module, die aus einer oder mehreren Enterprise-Beans zusammengesetzt sind.
- **Anwendungsclient-Container**
  ist für die eigenständige Anwendungsclients zuständig.

Die Kommunikation zwischen einzelnen J2EE-Anwendungskomponenten erfolgt über einen Container-Komponenten-Kontrakt (**Container Component Contract**), der die Container mit dem Anwendungsserver verbindet (*s. Abbildung 2.1 auf der vorherigen Seite*).

Die Client-API stellt die Schnittstelle zur Kommunikation zwischen den Anwendungskomponenten und einem Ressourcenadapter bereit. Dazu kann eine Adapter-spezifische Schnittstelle, die in der Connector-Architecture nicht spezifiziert ist, oder **Client Connector Interface** (CCI) als Basis verwendet werden. Das CCI stellt die standardisierte und vom EIS unabhängige Java-Schnittstelle zur Verfügung und ermöglicht den einheitlichen Zugriff auf alle Ressourcenadapter für Anwendungskomponenten. Der Ressourcenadapter seinerseits kommuniziert mit einem EIS unter Benutzung einer EIS-spezifischen Schnittstelle. Diese Schnittstelle ist abhängig vom EIS und somit kein Teil des JCA-Standards.

2.1.1 Verbindungsaufbau in der verwalteten Umgebung

Im Folgenden wird der Ablauf des Zugriffs von einer Anwendung über einen Ressourcenadapter auf EIS in einer verwalteten Umgebung beschrieben. Dazu soll der Ressourcenadapter die Systemkontrakte (*s. vorherigen Abschnitt auf Seite 3*) implementieren, die als Schnittstellen von der JCA-Spezifikation bereitgestellt werden. Für die Kommunikation mit den Anwendungskomponenten soll der Ressourcenadapter außerdem die Schnittstellen für die Verbindungsfactory und die Verbindung implementieren. Im Fall der Verwendung des Common Client Interface (CCI)[2] werden die CCI-Schnittstellen **ConnectionFactory** und **Connection** eingesetzt.

Zuvor soll jedoch der Ressourcenadapter in der Umgebung des Anwendungsservers konfiguriert werden. Die Konfigurationsinformationen des Ressourcenadapters wie z. Bsp. der Servername oder die Portnummer werden über den Deployment-Descriptor Mechanismus gesetzt.

Ist der Adapter erfolgreich im Server installiert, wird eine Verbindungsfactory beim **Java Naming and Directory Interface**- (JNDI-)[3] Dienst registriert. Durch diese Verbindungsfactory wird später

[2] Der Adapter kann seine eigene Client API für den Zugriff von Applikationskomponenten auf EIS bereitstellen.

[3] JNDI - Java Naming and Directory Interface, ist eine API für den Zugriff auf Namens- und Directory-Dienste. Mit Hilfe von JNDI können Daten und Objektreferenzen anhand eines Namens abgelegt und später über den Namen wieder gefunden werden.

ein Ressourcenadapter mit einer Anwendung verknüpft. Ein so konfigurierter Ressourcenadapter wird vom Anwendungsserver für die Herstellung physikalischer Verbindungen zum darunterliegenden EIS verwendet.

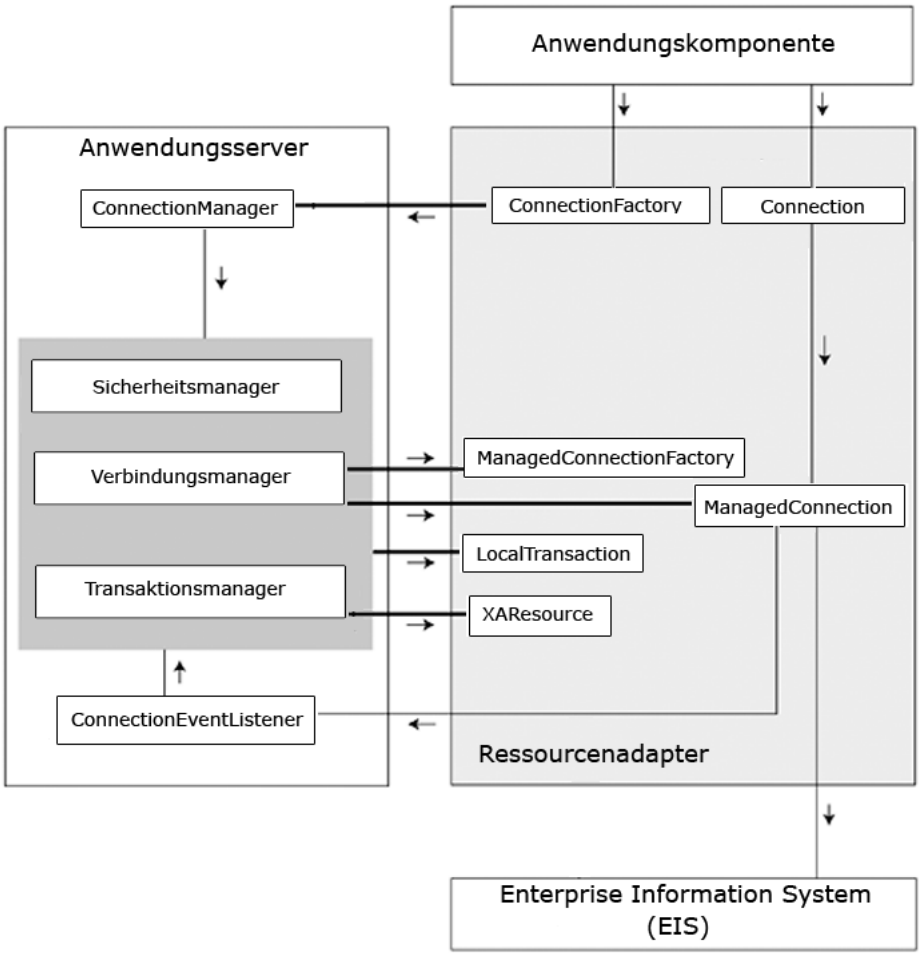


Abbildung 2.2: Verbindungsaufbau in der verwalteten Umgebung [SS01].

Die Abbildung 2.2 verdeutlicht, wie eine Verbindung zu einem EIS in einer J2EE-verwalteten Umgebung hergestellt werden kann:

1. Zuerst verwendet eine Anwendungskomponente den JNDI-Dienst, um Verbindungsfactory Objekte mit Hilfe eines JNDI-Namens zu suchen. Bei der *ConnectionFactory*-Schnittstelle handelt es sich um eine CCI-Schnittstelle, die für die Erzeugung von physikalischen Verbindungen zuständig ist.

2. Als nächstes ruft die Anwendungskomponente die *getConnection*-Methode des *ConnectionFactory* Objekts auf, um eine Verbindungsanfrage an die *ConnectionManager*-Schnittstelle weiterzuleiten. Die *ConnectionManager*-Schnittstelle in einer verwalteten Umgebung befindet sich dabei innerhalb eines Anwendungsservers.

3. Die *ConnectionManager*-Instanz überprüft den Verbindungspool innerhalb des Anwen-

dungsservers nach einer bereits vorhandenen Verbindung. Falls eine geeignete Verbindung gefunden wird, wählt der Anwendungsserver sie aus. Andernfalls erzeugt der Anwendungsserver mit Hilfe der *ManagedConnectionFactory*-Schnittstelle eine neue physikalische Verbindung zum EIS, die durch eine *ManagedConnection*-Instanz repräsentiert ist. Anschließend wird diese neu erzeugte Verbindung zum Verbindungspool des Servers hinzugefügt.

4. Anschließend wird mit Hilfe der *getConnection*-Methode der *ManagedConnection*-Schnittstelle ein Verbindungshandle auf Applikationsebene erzeugt, über das eine Applikationskomponente auf ein EIS zugreifen kann. Bei dem Verbindungshandle handelt es sich dabei um eine *Connection*-Instanz.

2.2 CICS Transaction Gateway

Das **CICS Transaction Gateway** (CICS TG) besteht aus Client- und Server-Softwarekomponenten, welche ermöglichen den Web-Anwendungen auf Dienste eines CICS-Servers zuzugreifen. CICS TG ist multiplatformfähig und kann auf verschiedenen Betriebssystemen eingesetzt werden: *z/OS*, *Linux* für *zSeries*, *AIX*, *HP-UX*, *Sun Solaris*, *Microsoft Windows* und *Linux* für *Intel*. Beispielsweise auf Windows oder Linux Betriebssystemen kann er mit einer Reihe von verschiedenen CICS-Servern (z.Bsp. CICS Transaction Server für z/OS oder TXSeries-Server) verbunden werden, während auf z/OS nur mit dem CICS Transaction Server für z/OS.

CICS TG stellt drei wichtige Programmierschnittstellen zur Verfügung:

- **External Call Interface** (ECI)
  Diese Schnittstelle ermöglicht der nicht-CICS-Anwendungen[4] den Aufruf von COMMAREA-basierenden CICS-Programmen. Die **CICS Communikation Area** (*Commarea*) übernimmt dabei als Pufferbereich innerhalb einer CICS-Region die Übergabe von Ein- und Ausgabedaten beim Aufruf von CICS-Programmen. Sie beschränkt sich allerdings auf eine Datenmenge von 32 KByte. Das ist die einzige Schnittstelle, die vom CICS TG für z/OS unterstützt wird.

- **External Presentation Interface** (EPI)
  Diese Programmierschnittstelle wird zum Aufruf von 3270-basierenden Transaktionen eingesetzt. Eine Client-Anwendung kann dabei einen virtuellen Terminal erzeugen, dessen Ein- und Ausgaben über diese API gesteuert werden können. Die Benutzung vom EPI ist nur dann möglich, wenn CICS TG auf verteilten Systemen eingesetzt wird.

- **External Security Interface** (ESI)
  Diese Schnittstelle ermöglicht den Zugriff auf die erweiterte Funktionen der Benutzer- und Kennwortverwaltung, welche durch PEM (*Password Expiration Management*) auf einem CICS-Server bereitgestellt sind. Das ESI ist ebenfalls wie das EPI nur auf verteilten Plattformen verfügbar.

Der CICS Transaction Gateway besteht hauptsächlich aus folgenden Komponenten [CP05]:

- **Gateway daemon**
  Als so genannter *long-running* Prozess, fungiert er als Verbindungsglied zwischen externen

[4] Ein Programm, das nicht unter einem CICS Transaktions-Monitor läuft.

Clients und CICS-Anwendungen. Dabei hört er über einen festgelegten TCP/IP-Port die eingehende (*Inbound*) Client-Anfragen ab, die durch eines der folgenden Protokollen (TCP, HTTP, SSL oder HTTPS) geleitet werden.

- Client daemon
  ist ein Bestandteil des CICS Transaction Gateway auf allen verteilten Systemen. Die Funktion dieser Software besteht darin, die ECI-, EPI- oder ESI-Aufrufe zu einem CICS-Server weiterzuleiten. Diese Aufrufe können dabei nebenläufig unter der Verwendung von SNA, TCP62 oder TCP/IP Protokollen ausgeführt werden. Auf z/OS Systemen ist kein Client-deamon vorhanden, stattdessen wird das EXCI (*External CICS Interface*) (*s. Abschnitt 3.2 auf Seite 15*) benutzt, um auf CICS-Programme zugreifen zu können.

- Configuration Tool
  Das Konfigurationstool ist eine auf Java basierende grafische Benutzeroberfläche (*GUI - graphical user interface*), welche auf allen unterstützten Plattformen vom CICS TG bereitgestellt wird. Das Tool wird für die Konfiguration der Eigenschaften des Gateway- und Client-deamon verwendet.

- Ressourcen Adapter
  Das sind CICS JCA Ressourcenadapter, die von der J2EE-Komponenten auf einem J2EE-Server benutzt werden. Der CICS Transaction Gateway stellt dazu zwei Ressourcenadapter zur Verfügung für ECI und EPI. Falls in einer Anwendung die Aufrufe zum CICS direkt über CICS TG Schnittstellen (*s. Programmierschnittstellen auf der vorherigen Seite*) erfolgen, können die CICS TG Klasses auch direkt eingesetzt werden.

Als J2EE-Anwendungsserver wird in dieser Arbeit der WebSphere Anwendungsserver (WAS) verwendet, der in Verbindung mit dem CICS TG in drei folgenden Konfigurationen (*Topologien*) eingesetzt werden kann:

- Topologie 1.
  Der Anwendungsserver und das CICS TG sind beide in einem verteilten System (z.Bsp. Windows) installiert.
- Topologie 2.
  Der Anwendungsserver befindet sich in einer verteilten Umgebung und das CICS TG ist im z/OS-System installiert.
- Topologie 3.
  Der Anwendungsserver und das CICS TG befinden sich beide in einem z/OS-System.

Die zweite und dritte Topologien bilden den Gegenstand dieser Arbeit und werden in nachfolgenden Kapiteln genauer beleuchtet.

2.2.1 Remote CICS-Verbindung

Die zweite Topologie stellt eine sogenannte *remote* Konfiguration dar, wo der WebSphere Anwendungsserver (WAS) auf einer verteilten Plattform[5] eingesetzt ist. Der Zugriff auf CICS erfolgt über den Gateway-deamon mit Hilfe von EXCI-Schnittstelle (*s. Abbildung 2.3 auf der nächsten*

[5] Der Anwendungsserver kann auch auf einer anderen LPAR (*Logical Partition*) installiert werden.

Seite). CICS Transaction Gateway muss immer noch auf demselben System wie der WebSphere Anwendungsserver installiert werden. Obwohl der aktive CICS TG *task* ist auf dem verteilten System nicht erforderlich.

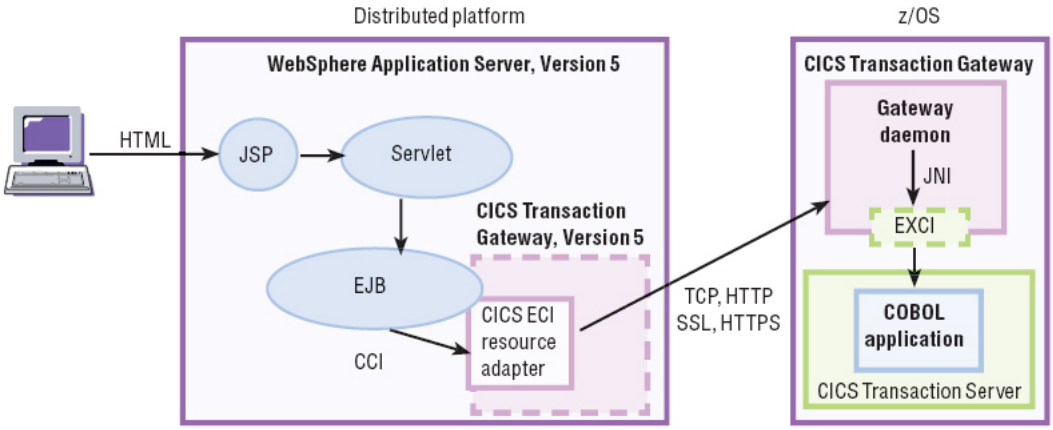


Abbildung 2.3: Der Zugriff auf CICS über CICS Gateway-deamon [CP05].

Der CICS-ECI-Ressourcenadapter und WAS, wie in jeder verwalteten Umgebung, kommunizieren über die Systemkontrakte (*siehe Abschnitt J2EE Konnektor Architektur auf Seite 3*). Diese Systemkontrakte, beziehend auf diese Topologie, verfügen über die folgende *Qualities of Service*:

- Verbindungsmanagement
  Der Verbindungspool stellt die physikalische Verbindungen zwischen WebSphere Application Server und Gateway-daemon auf z/OS-System dar. In solch einer Konfiguration ist es notwendig einen leistungsfähigen Verbindungspooling-Mechanismus zu haben. Sonst kann sich ein erheblicher Anteil der ganze Zeit, die für die Kommunikation mit CICS gebraucht wird, nur für die Herstellung und Terminierung von Verbindungen erstrecken. Der JCA Verbindungspooling-Mechanismus vermeidet dieses Kommunikations-*Overhead* indem die Verbindungen durch WebSphere Anwendungsserver Pool-Manager verwaltet werden.

- Transaktionsmanagement
  In dieser Konfiguration ist die Verbindung vom WebSphere Anwendungsserver zum CICS-deamon eine *single-phase-commit* Verbindung. Der CICS-ECI-Ressourcenadapter, der nur eine einphasige Koordination unterstützt (z. Bsp. über ein *LocalTransaction*-Interface), hat dabei eine beschränkte Unterstützung für die globale Transaktionen[6]. Die ECI-Anfragen zu einer CICS-Region können aber zu einer globalen Transaktion gehören mit einer beliebigen Anzahl von zweiphasigen Ressourcenmanagern. In diesem Fall wird *Last Participant Support* Mechanismus des WAS eingesetzt.

- Sicherheitsmanagement
  In dieser Konfiguration stellt der Gateway-daemon den Ausgangspunkt des z/OS-Systems dar und ist somit für die Authentifizierung der ankommenden ECI-Anfragen des Clients zuständig. Dazu soll CICS TG entsprechend konfiguriert werden, um die Überprüfung der Benutzer-ID und des Kennworts durchzuführen. Für die zusätzliche Sicherheitskontrolle

[6] In der J2EE-Architektur Terminologie ist eine globale Transaktion als eine verteilte Transaktion bezeichnet.

können die CICS-Autorisationsmechanismen, wie *MRO bind security*, *Link security* oder *Surrogate security*, verwendet werden.

2.2.2 Lokale CICS-Verbindung

In der dritten Topologie, die in der Abbildung 2.4 dargestellt ist, kann nur CICS-ECI-Ressourcenadapter verwendet werden. In dieser Konfiguration ist der CICS Transaction Gateway (CICS TG) im *lokalen* Modus eingesetzt. Das bedeutet, dass der WebSphere Anwendungsserver (WAS) und der CICS-Server sich in derselben z/OS logical partition (LPAR) in getrennten Regionen (*virtuelle Adressräume*) befinden und zur Kommunikation die *cross-memory* EXCI-Verbindung verwenden. In diesem Fall kann das *lokale* Protokoll des CICS TG benutzt werden, das die unterliegenden Transportmechanismen des JNI-Moduls *libCTGJNI.so* direkt aufruft. Der Gateway-deamon ist in dieser Konfiguration nicht erforderlich.

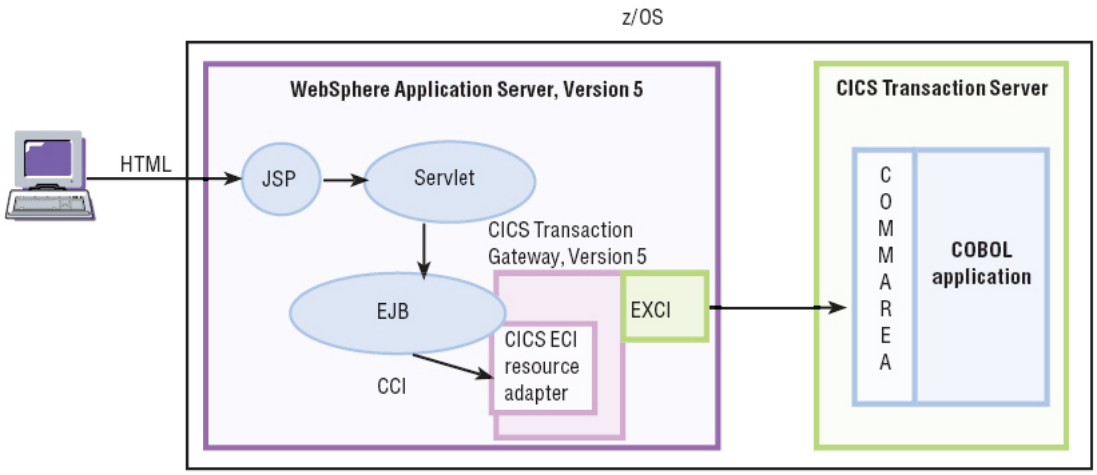


Abbildung 2.4: Der Zugriff auf CICS unter Verwendung vom lokalen CICS TG [G2204].

Die spezifischen JCA Qualities of Service (*QoS*), die sich auf diese Topologie beziehen, sind folgende:

- Verbindungsmanagement
  Der Verbindungspool stellt eine Reihe von Verbindungensobjekten dar, die durch WebSphere Anwendungsserver verwaltet werden. Diese Verbindungsobjekte hängen aber nicht direkt mit EXCI-Sitzungen (*pipes*), die von CICS TG reserviert werden, zusammen.

- Transaktionsmanagement
  In dieser Konfiguration sind CICS TG und WAS im selben virtuellen Adressraum. Dies ist von Vorteil, da beide die Funktionen des RRS (*MVS Resource Recovery Services*) benutzen können. Das RSS, welches ein grundlegendes Teil von z/OS ist, tritt als ein externer Transaktionsmanager auf. Mit der Transaktionsunterstützung von RRS ist es möglich, dass ein *one-phase-commit-fähigen* Ressourcenmanager wie CICS-ECI-Ressourcenadapter an einer globalen Transaktion mit einer beliebigen Anzahl von zweiphasigen Ressourcenmanagern, wie CICS, IMS/TM, IMS/DB, IBM DB2 oder WebSphere MQ, teilnehmen kann. Damit ist es die einzige Konfiguration, in der sich der Transaktionsbereich des two-phase-commit-

Protokolls über alle CICS-Ressourcen erstreckt.

- Sicherheitsmanagement
  Im Gegensatz zur Topologie 2 (*Abschnitt 2.2.1 auf Seite 8*), können WAS und CICS in dieser lokalen Topologie die Funktionen des RACF (*Resource Access Control Facility*)-Sicherheitssystems gemeinsam nutzen, um die Authentifizierungs- und Autorisierungsüberprüfungen durchzuführen. Die Authentifizierung von Benutzern findet in der Regel innerhalb des WAS (z.Bsp. durch Anmeldung in einem Formular) statt. In diesem Fall wird die authentifizierte Benutzer-ID automatisch als RACF-Benutzer-ID aufgefasst und zum CICS-ECI-Ressoursenadapter und dann zu CICS weitergeleitet. Da der Benutzer sich beim WAS bereits identifiziert hat, wird das Kennwort mit der ECI-Anfrage nicht übertragen. Für diese so genannte *Trust*-Beziehung zwischen WAS und CICS können auch wie bei der zweiten Topologie *MRO Bind*-, *Surrogate*- und *Link*-Sicherheitsmechanismen eingesetzt werden.

2.3 WebSphere Developer für zSeries

Die J2EE-Anwendung in dieser Arbeit wird mit Hilfe des WebSphere Developer für zSeries V6 (*WDz*) [IBMc] erstellt. WDz ist eine auf Eclipse basierende integrierte Entwicklungsumgebung (*IDE, Integrated Development Environment*) für das Erstellen, Testen und Bereitstellen von sowohl J2EE-Anwendungen als auch z/OS Betriebssystem basierten Anwendungen.

WDz basiert auf dem IBM Rational Application Developer (*IRAD*) [IBMb], dessen frühere Version als das Websphere Studio Application Developer bekannt ist. Die beide Produkte gehört zur Produktfamilie IBM Rational Softwareentwicklungsplattform und unterscheiden sich nur in Plugins, welche in den einzelnen Konfigurationen verfugbar sind.

Die eigentliche Erstellung der Anwendung kann unter der Verwendung nur des IRAD realisiert werden. Diese Anwendung, die in diesem Fall als J2C-JavaBean bereitgestellt wird, beschreibt das Zusammenspiel zwischen dem CICS-COBOL-Programm und dem CICS-ECI-Ressourcenadapter.

Für die Testzwecke sollen aber die Funktionen und Komponenten des WDz herangezogen werden. Der generierte Code kann nur in dem Fall getestet werden, wenn WDz mit IRAD verwendet wird. Hiermit können die Ressourcen beider Produkte ausgenutzt werden.

Kapitel 3 Erstellung und Test der CICS-Anwendung

Dieses Kapitel befasst sich mit der Einrichtung der im Kapitel 2 vorgestellten Topologien. Insbesondere wird hier auf die Erstellung und das Testen des CICS-Programms und die Konfiguration der Software-Komponenten eingegangen. Die Software- und Hardware-Komponenten, die in diesem und nachfolgendem Kapitel eingesetzt werden, sind in der folgenden Tabelle aufgelistet.

Komponente	Beschreibung
Rechner	IP-Adresse: 139.18.4.35 hostname: padme.informatik.uni-leipzig.de
z/OS Betriebssystem	Version 1.5
CICS Transaction Gateway (Windows, z/OS)	Version 5.1
CICS Transaction Server (z/OS)	Version 2.3
IBM WebSphere Application Server (z/OS)	Version 5.0.0, Build Level W502008
WebSphere Studio Application Developer Integration Edition (Windows)	Version 5.1.1
CICS Transaction Server (z/OS)	Version 2.3
DB2 Datenbank (z/OS)	Version 7.10

Tabelle 3.1: verwendete Soft- und Hardware-Komponente.

3.1 Konfiguration des CICS Transaction Gateway

Die Anwendung, die in dem Kapitel 4 auf Seite 33 erstellt wird, verwendet eine application-managed Connection (*anwendungsverwaltete Verbindung*), bei der die Anwendung selbst für die Bereitstellung von Authentisierungsinformationen (*Benutzer-ID und Kennwort*) zuständig ist. Da diese Anwendung in zwei Topologien getestet wird, soll hierzu der CICS-ECI-Ressourcenadapter und Gateway-deamon für die Überprüfung der Benutzer-ID und des Kennworts mit RACF konfiguriert werden.

Wenn CICS Transaction Gateway (CICS TG) in einem lokalem Modus (*siehe Abschnitt: Lokale CICS-Verbindung auf Seite 10*) innerhalb des WebSphere Anwendungsservers (WAS) auf z/OS läuft, soll die neue Umgebungsvariable innerhalb des Anwendungsservers erstellt werden. Dazu öffnet man die Administrationskonsole des Servers, indem man in einem Browser die folgende URL eingibt:

http://padme.informatik.uni-leipzig.de:9148/admin.

Danach klickt man im linken Menü der Administrationskonsole auf **Umgebung** und dann auf **WebSphere Variablen verwalten**. Auf der rechten Seite erstellt man die neue Variable **AUTH_USERID_PASSWORD**, indem man auf die Schaltfläche **Neu** klickt. Anschließend setzt man deren Wert auf **Yes**.

Zusätzlich soll sichergestellt werden, dass die CICS SDFHEXCI und SDFHLINK Bibliotheken als programmgesteuert (*program-controlled*) kennzeichnet sind [TB05]. Die CICS SDFHEXCI Bibliothek enthält die CICS-EXCI-Module für CICS Transaction Gateway und kann zur Laufzeit aus der STEPLIB geladen werden. Auf dem *padme*-Rechner waren diese Bibliotheken in Datasets CICSTS23.CICS.SDFHEXCI und CICSTS23.CICS.SDFHLINK installiert und sollen zum RACF-Profil ***** in die Klasse **PROGRAM** hinzugefühgt werden. Dazu wählt man von der Benutzeroberfläche ISPF (*Interactivity System Product Facility*) die Option **6** (*Command*), um zum *"Enter TSO or Workstation commands"* Panel zu gelangen. In der Kommando-Zeile des Panels gibt man dann nacheinander folgende RACF-Befehle ein:

```
RALT PROGRAM * ADDMEM('CICSTS23.CICS.SDFHEXCI'//NOPADCHK)
RALT PROGRAM * ADDMEM('CICSTS23.CICS.SDFHLINK'//NOPADCHK)
SETROPTS WHEN(PROGRAM)REFRESH
```

In dem *remote* Modus (*siehe Abschnitt 2.2.1 auf Seite 8*) soll die Konfigurationsdatei *ctgenvvars* des CICS TG, das bereits installiert war und befand sich im HFS-Verzeichnis (*Hierarchical File System*) */V1R5/usr/lpp/cicsts/ctg51/ctg*, abgeändert werden. Dazu wechselt man unter OMVS-Subsystem ins Verzeichnis **/V1R5/usr/lpp/cicsts/ctg51/ctg/bin** und führt den Befehl *oedit ctgenvvar* aus, um das Konfigurationsscript zu bearbeiten.

Es müsste nur der Wert der **AUTH_USERID_PASSWORD** Variable auf **Yes** umgestellt werden (*s. Abbildung 3.1*). Alle anderen Umgebungsvariablen sollen unverändert bleiben.

```
1 RRM_NAME="CCL.CTG"
2 EXCI_OPTIONS=""
3 EXCI_LOADLIB="CICSTS23.CICS.SDFHEXCI"
4 export DFHJVPIPE="JGATE500"
5 AUTH_USERID_PASSWORD="YES"
6 export DFHJVSYSTEM_00="CICS-Default example server"
```

Abbildung 3.1: Ausschnitt aus der Konfigurationsdatei **ctgenvvars**.

DFHJVPIPE gibt den *Netname* der spezifischen Verbindung zwischen CICS TG und CICS an (*s. Abschnitt: 3.2 auf der nächsten Seite*). Falls kein Wert für diese Variable eingesetzt wird,

verwendet CICS TG eine allgemeine (*generic*) Verbindung. Die **EXCI_LOADLIB** und **EXCI_OPTIONS** Variablen stellen gemeinsam die STEPLIB-Konkatenation dar. Dabei ist in der EXCI_LOADLIB Variable die SDFHEXCI-Bibliothek mit den EXCI-Lademodulen definiert. Die EXCI_OPTIONS Variable hinweist auf Dataset, welcher die EXCI-Optionstabelle **DFHXCOPT** enthält. Diese Tabelle enthält die Parameter für die EXCI-Schnittstelle, deren Lademodul sich standardmäßig in SDFHEXCI Dataset befindet.

Zusätzlich sollen auch die Lademodule des CICS TG als programmgesteuert kennzeichnet werden [PJ02]. Dazu führt man unter OMVS-Subsystem folgende Befehle aus:

```
extattr +p /V1R5/usr/lpp/cicsts/ctg51/ctg/bin/lib*.so
extattr +p /V1R5/usr/lpp/cicsts/ctg51/ctg/bin/SECURES
```

Schließlich soll der Gateway-deamon neu gestartet werden. Dies geschieht mit Hilfe des SDSF-Werkzeuges, das für die Steuerung, Verwaltung und die Ausgaben der Jobs und vielfältiger Systeminformationen zuständig ist. Es wird über die Benutzeroberfläche ISPF aufgerufen. Durch die Eingabe der Option **m** im ISPF gelangt man zum *"IBM Products Panel"*. Hier wählt man die Option **5** (*SDSF*), um zur *"Spool Search and Display Facility"* zu wechseln.

Für den Neustart des Gateway-deamon soll die JCL-Prozedur **CICSTG**, die sich im Dataset **ADCD.ZOSV1R5.PROCLIB** befand, zuerst gestoppt und dann wieder gestartet werden. Dazu gibt man in der Kommando-Zeile innerhalb des SDSF zunächst den Befehl *"***/P CICSTG***"* (*Purge*) und dann *"***/S CICSTG***"* (*Start*) ein. Mit der Option **ST** (*Status of Jobs*) kann danach überprüft werden, ob das CICS TG erfolgreich gestartet wurde.

3.2 EXCI-Verbindung zu CICS

Dieser Abschnitt beschreibt wie die EXCI-Verbindung konfiguriert werden soll, damit CICS Transaction Gateway (CICS TG) auf einem z/OS-System mit einer CICS-Region kommunizieren kann.

Das EXCI (*External CICS interface*) ist eine Programmierschnittstelle, welche von CICS bereitgestellt wird. Sie ist in allen verfügbaren CICS-Versionen unterstützt und stellt eine vereinfachte Version der ECI-Schnittstelle dar. Die EXCI-Schnittstelle verbindet die nicht-CICS Programme (*Client-Programme*), wie beispielsweise die Stapelverarbeitungsprogramme (*batch jobs*) oder Gateway-deamon selbst, mit Programmen in einer CICS-Region (*Server-Programme*). Dabei werden die Daten zwischen den Programmen durch die COMMAREA (*Communication area*) ausgetauscht.

Die EXCI-Programmierschnittstelle ermöglicht den Client-Programmen das Reservieren und Öffnen von Sitzungen (*Sessions*[1]) zu einer CICS-Region und das Senden von DPL-Anfragen (*distributed program link*) über diese. Zur Kommunikation zwischen Regionen werden die MRO (*Multiregion Operation*)-Funktionen der CICS IRC (*Interregion Communication*) benutzt.

Die Definition einer EXCI-Verbindung besteht aus **CONNECTION**- und **SESSIONS**-Definitionen, die in einer CSD (*CICS System Definitions*)-Datei abgelegt sind, in deren die gesamte Informationen eines CICS-Systems gespeichert sind [SC304]. In diesem Fall waren diese Definitionen vorhanden

[1] Sessions werden auch *pipes* benannt.

und befanden sich in der Gruppe **GRNEXCI**.

Um die Verbindung zwischen CICS TG und CICS nur für die autorisierte Benutzer zu ermöglichen, soll die Connection-Definition, die in diesem Fall **JCOS** heißt, abgeändert werden. Dazu schließt man zunächst innerhalb der CICS-Region die IRC mit dem Befehl **CEMT SET IRC CLOSED**. Danach wird mit Hilfe der CICS-Transaktion **CEDA** die Connection-Definition aufgerufen (*s. Abbildung 3.2*).

```
1  CEDA ALTER CONNECTION(JCOS) GROUP(GRNEXCI)
2   OBJECT CHARACTERISTICS                                  CICS RELEASE = 0630
3    CEDA ALter CONnection( JCOS )
4     CONnection    : JCOS
5     Group         : GRNEXCI
6     DEscription   :
7   CONNECTION IDENTIFIERS
8     Netname       : JGATE500
9     INDsys        :
10  CONNECTION PROPERTIES
11    ACcessmethod  : IRc          Vtam | IRc | INdirect | Xm
12    PRotocol      : Exci         Appc | Lu61 | Exci
13    Conntype      : Specific     Generic | Specific
14    SInglesess    : No           No | Yes
15  SECURITY
16    SEcurityname  :
17    ATtachsec     : Identify     Local | Identify | Verify | Persistent
```

Abbildung 3.2: Definition der EXCI-Verbindung.

Es handelt sich dabei um eine spezifische EXCI-Verbindung (*Conntype*-Attribut ist *Specific*). Für den spezifischen Verbindungstyp soll das **Netname**-Attribut festgelegt werden, das in diesem Fall der Umgebungsvariable **DFHJVPIPE** des CICS TG entspricht. Der **ACcessmethod**-Parameter für die EXCI-Verbindung soll auf **IRC** eingesetzt werden.

Nun ändert man der Eintrag unter dem **ATtachsec**-Attribut von **Local** auf **Identify**. Local und Identify sind die einzige Parameter, die für eine EXCI-Verbindung verfügbar sind. *Identify* gibt an, dass mit jeder EXCI-Anfrage vom CICS TG ein Benutzer-ID übergeben und durch RACF ausgewertet wird. Dabei wird kein Kennwort erwartet. Der CICS geht davon aus, dass das Kennwort vom CICS TG oder vom WebSphere Anwendungsserver überprüft wurde.

Basierend auf der Connection-Definition können nun eine oder mehrere Sitzungen aufgebaut werden, welche von einem einzelnen Benutzer eines Client-Programms reserviert werden können. In diesem Fall wird die gleichnamige Session **JCOS** verwendet, deren Eigenschaften in der Abbildung 3.3 dargestellt sind. Diese EXCI-Session ist mit der JCOS Connection-Definition verknüpft (*Zeile 8*) und kann ohne Änderungen übernohmen werden.

Abbildung 3.3: Definition der EXCI-Sitzung.

```
1  VIEW SESSIONS(JCOS) GROUP(GRNEXCI)
2   OBJECT CHARACTERISTICS                                  CICS RELEASE = 0630
```

Fortsetzung auf der nächsten Seite . . .

Fortsetzung der Abbildung 3.3

```
   CEDA View Sessions( JCOS )
    Sessions       : JCOS
    Group          : GRNEXCI
    DEscription    :
   SESSION IDENTIFIERS
    Connection     : JCOS
    SESSName       :
    NETnameq       :
    MOdename       :
   SESSION PROPERTIES
    Protocol       : Exci              Appc | Lu61 | Exci
    MAximum        : 000 , 000         0-999
    RECEIVEPfx     : <
    RECEIVECount   : 010               1-999
    SENDPfx        :
    SENDCount      :                   1-999
    SENDSize       : 04096             1-30720
    RECEIVESize    : 04096             1-30720
    SESSPriority   : 000               0-255
    Transaction    :
+  OPERATOR DEFAULTS
                                                        SYSID=CICS APPLID=CICS
```

Nun muss die modifizierte Connection-Definition aktualisiert werden. Dazu soll die gesamte Gruppe mit dem Befehl **CEDA INSTALL GROUP(GRNEXCI)** installiert werden. Damit die MVS-Programme die EXCI-Verbindung benutzen können, muss die *Interregion Communication* mit dem folgenden Befehl wieder geöffnet werden: **CEMT SET IRC OPEN**.

3.3 Sicherheitsvorbereitungen zwischen WebSphere und CICS

Die für die Kommunikation mit CICS verwendete EXCI-Aufrufe werden innerhalb der CICS-Region nicht authentifiziert. Es wird lediglich die übertragene Benutzer-ID für die Autorisierungszwecke vom CICS verwendet. Zur Erhöhung der Sicherheit zwischen dem WebSphere Anwendugsserver (WAS) und CICS können mehrere Sicherheitsmechanismen eingesetzt werden. Einige davon werden in diesem Abschnitt vorgestellt. Da in dieser Arbeit zwei Topologien eingesetzt sind, wird in den folgenden Schritten dieses Abschnittes zwischen *CICS Transaction Gateway deamon* (*remote Verbindung*) und *CICS-ECI-Ressourcenadapter* (*lokale Verbindung*) unterschieden.

Wie es im vorigen Abschnitt erwähnt wurde, erfolgen die Zugriffe auf CICS über die EXCI-Schnittstelle. Deshalb soll zuerst die EXCI-Verbindung zu CICS abgesichert werden. Die EXCI-Anfragen funktionieren sehr ähnlich wie DPL (*distributed program link*) und ebenfalls wie DPL-Anfragen laufen sie unter der *mirror transaction* **CSMI**. Deshalb soll sichergestellt werden, dass die Benutzer-IDs, die mit einer Verbindung vom WAS übertragen werden, den Zugriff auf diese geschützte Transaktion haben [SC305a].

Dazu erstellt man ein Profil für diese Transaktion in der RACF-Klasse **TCICSTRN**, indem man folgenden RACF-Befehl im *"TSO or Workstation commands"*-Panel eingibt:

```
RDEFINE TCICSTRN CSMI UACC(NONE)
```

Jedes Profil verfügt über eine Zugriffsliste, die auflistet, welche Benutzer-IDs den Zugriff auf die RACF-geschützte Ressourcen haben können.

```
PERMIT CSMI CLASS(TCICSTRN) ID(PSELES) ACCESS(READ)
PERMIT CSMI CLASS(TCICSTRN) ID(ASSR1) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

Für die CSMI-Transaktion wird ein Lesezugriff für PSELES-ID erteilt, wenn der WebSphere Anwendungsserver sich in einem verteilten System befindet und Zugriff auf CICS durch CICS Transaction Gateway deamon (*CICS TG*) auf dem z/OS-System stattfindet.

Falls CICS TG im lokalen Modus innerhalb des WAS auf z/OS eingesetzt ist, dann ist es die *WebSphere Servant region ID* (**ASSR1**), für die der Zugriff auf das CSMI-Profil freigegeben werden soll. Anschließend muss mit Hilfe des Befehls SETROPTS die TCICSTRN-Klasse aktualisiert werden, damit alle Änderungen wirksam werden.

MRO-Bind-security

Um den unautorisierten Zugriff auf die CICS-Region zu verhindern, wird die von CICS unterstützende *MRO Bind security* verwendet. Es muss in unserem Fall nur die *spezifische* Verbindung mit CICS zugelassen werden, die entweder vom WAS auf z/OS System aufgebaut wird oder über Gateway deamon erfolgt. Erreicht wird das durch die Erstellung folgenden Profile in der **FACILITY** Klasse in RACF.

```
RDEFINE FACILITY (DFHAPPL.JGATE500) UACC(NONE)
RDEFINE FACILITY (DFHAPPL.CICS) UACC(NONE)
```

Das erste Profil steht für die spezifische Verbindung, die durch pipe **JGATE500**[2] identifiziert ist. Das **DFHAPPL.CICS**-Profile wird für den gesicherten Zugriff auf die CICS-Region vom CICS TG verwendet. Als nächstes müssen die Zugriffsrechte zu diesen Profile über das PERMIT-Kommando erteilt werden.

```
PERMIT DFHAPPL.JGATE500 CLASS(FACILITY) ID(ASSR1) ACCESS(UPDATE)
PERMIT DFHAPPL.CICS CLASS(FACILITY) ID(ASSR1) ACCESS(READ)
```

Ebenso wie bei der CSMI-Transaktion wird für die lokale Verbindung **ASSR1** verwendet.

[2] Für die lokale Verbindung ist DFHJVPIPE als eine WebSphere Umgebungsvariable für lokale JCA-Verbindungsfactory definiert. Für die remote Verbindung wird DFHJVPIPE aus der Konfigurationsdatei *ctgenvvars* des CICS TG verwendet.

```
PERMIT DFHAPPL.JGATE500 CLASS(FACILITY) ID(CICSC001) ACCESS(UPDATE)
PERMIT DFHAPPL.CICS CLASS(FACILITY) ID(CICSC001) ACCESS(UPDATE)
SETROPTS RACLIST(FACILITY) REFRESH
```

Für remote Verbindung wird die *CICS TG Started Task User-ID* (CICSC001) eingesetzt.

Surrogate-security

Die andere Sicherheitsmaßnahme, die ebenfalls von CICS unterstützt wird, ist die *Surrogate security*. Diese Art der Sicherheit bietet die Möglichkeit innerhalb CICS in Namen eines anderen Benutzers zu agieren. Für die Benutzer-IDs, die mit einer EXCI-Verbindung vom CICS TG zu CICS übertragen werden, kann diese Sicherheitsmaßnahme wie folgt realisiert werden.

- Zuerst muss sichergestellt werden, dass in der CICS EXCI-Optionstabelle **DFHXCOPT**, deren Quellcode in CICSTS23.CICS.SDFHSAMP und Lademodus in CICSTS23.CICS.SDFHEXCI vorliegt, der Parameter **SURROGCHK** auf **YES** eingesetzt ist. Standardmäßig war die Surrogate-Überprüfung aktiviert, so dass keine weitere Änderungen notwendig waren.
- Danach erstellt man **PSELES.DFHEXCI**-Profil in der Klasse **SURROGAT**, welches den Zugriff vom CICS TG zu CICS für Benutzer-ID **PSELES** beschreibt.

```
RDEFINE SURROGAT PSELES.DFHEXCI UACC(NONE) OWNER(PSELES)
```

- Anschließend müssen Zugangsberechtigungen erteilt werden. Für die remote Verbindung ist es CICS TG deamon, der stellvertretend für PSELES-ID eingesetzt wird. Wenn CICS TG im lokalen Modus innerhalb des WebSphere Anwendungsservers läuft, dann ist es die ID der WebSphere-Region, die als *Surrogate* fungiert.

```
PERMIT PSELES.DFHEXCI CLASS(SURROGAT) ID(CICSC001) ACCESS(READ)
PERMIT PSELES.DFHEXCI CLASS(SURROGAT) ID(ASSR1) ACCESS(READ)
SETROPTS RACLIST(SURROGAT) REFRESH
```

3.4 Verbindung zwischen CICS und DB2

Zum Zugriff auf die DB2-Datenbank stellt CICS eine **CICS DB2 attachment facility**-Schnittstelle zur Verfügung. Diese Schnittstelle bereitet eine ständige Verbindung (*DB2CONN*) zwischen CICS und DB2, um die Befehle oder Anfragen an das DB2-Subsystem von CICS-Anwendungen zu senden. Eine DB2-Datenbank kann dabei gemeinsam von verschieden CICS-Systemen benutzt werden aber jedes CICS-System nur mit einem DB2-Subsystem in Verbindung gebracht werden kann. Die Wiederherstellung von Daten in beiden DB2- und CICS-Systemen, falls ein Transaktion- oder System-Fehler auftritt, wird von CICS koordiniert [SC303].

Die *DB2CONN*-Definition stellt die Hauptkomponente einer CICS-Datenbankverbindung dar und kann nur einmal innerhalb vom CICS-System installiert werden. Zur dieser ständigen Verbindung können von *"CICS DB2 attachment facility"* mehrere *Threads* zugeordnet werden. Sie stellen die so genannte individuelle Verbindungen für jede aktive Transaktion dar, die auf DB2-Ressourcen zugreift. Es gibt drei Arten von Threads, die *"CICS DB2 attachment facility"* unterstützt:

- Command threads
  Die von der DSNC-Transaktion verwendete Threads, die zum Senden von Befehlen zu einem DB2-Subsystem verwendet werden.

- Entry threads
  Das sind spezielle Threads, die nur für die Transaktionen mit spezifischen Anforderungen bestimmt sind. Die *Entry*-Threads werden über das Makro DB2ENTRY definiert, in dem Relationen zwischen Transaktion, DB2-Plan und Programm festgelegt werden. Alle DB2ENTRY-Definitionen sind mit einer DB2CONN-Definition verbunden und können deswegen nur dann installiert werden, wenn eine DB2CONN vorhanden ist [SC305b].

- Pool threads
  Pool-Threads sind für alle andere Transaktionen vorgesehen, die zwei oben genannten Typen von Threads nicht verwenden oder nicht erhalten können. Es konnten auch die so genannte *overflow* Transaktionen sein, für die kein *Command*- oder *Entry*-Thread übrig bleibt. Solche Threads werden innerhalb der POOL-THREAD Sektion einer DB2CONN-Definition festgelegt.

Bevor mit der Einrichtung der DB2-Verbindung angefangen wird, soll die so genannte *AUTHTYPE-security* eingerichtet werden. Dieser Sicherheitsmechanismus stellt fest, dass nur die autorisierte Benutzer die AUTHID-, COMAUTHID-, AUTHTYPE- oder COMAUTHTYPE-Attribute der DB2ENTRY- und DB2CONN-Definitionen verändern können. Dies wird erreicht, indem ein Profil der Form DFHDB2.AUTHTYPE.*authname*, wo *authname* der Name der DB2ENTRY oder DB2CONN ist, in der FACILITY-Klasse in RACF erstellt wird. Dazu sollen die folgenden TSO-Befehle innerhalb des *"ISPF Command Shell"*-Panels nacheinander ausgeführt werden.

```
RDEFINE FACILITY DFHDB2.AUTHTYPE.DB2CON UACC(NONE)
PERMIT DFHDB2.AUTHTYPE.DB2CON CLASS(FACILITY) ID(PSELES) ACCESS(READ)
```

Das Profil DB2CON steht für die gleichnamige DB2-Verbindung. Die Leseberechtigungen werden dem Benutzer-ID PSELES erteilt, um oben genannte Attribute der DB2CONN-Definition verändern zu können.

```
RDEFINE FACILITY DFHDB2.AUTHTYPE.DB2CONE UACC(NONE)
PERMIT DFHDB2.AUTHTYPE.DB2CONE CLASS(FACILITY) ID(PSELES) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Die gleiche Operationen sind für DB2ENTRY-Definition namens DB2CONE durchzuführen. Danach muss mit Hilfe des Befehls SETROPTS die FACILITY-Klasse aktualisiert werden.

3.4.1 Definition der Datenbankverbindung

Die DB2-Verbindung namens **DB2CON** wird mit Hilfe der CEDA-Transaktion erstellt. Die relevante Parameter dieser Verbindung sind in der folgenden Abbildung dargestellt.

```
CEDA DEFINE DB2CONN(DB2CON) GROUP(PSELES)
  OVERTYPE TO MODIFY                                              CICS RELEASE = 0630
   CEDA  DEFine DB2Conn( DB2CON )
   DB2Conn         :  DB2CON
   Group           :  PSELES
   DEscription     ==>
  CONNECTION ATTRIBUTES
   CONnecterror    ==> Sqlcode           Sqlcode | Abend
   DB2Groupid      ==>
   DB2Id           ==> DB7G
  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
                                                        SYSID=CICS APPLID=CICS
  DEFINE SUCCESSFUL
```

Abbildung 3.4: Definition der CICS-DB2-Verbindung.

Jede Transaktion, die auf DB2 zugreift, soll einen DB2-Identifikator enthalten, der so genannte Autorisierungsidentifikator (kurz *AutorisierungsID*), welcher zur Sicherheitsüberprüfungen innerhalb DB2 benutzt wird. Aus diesem Grund wurde für den *AUTHType*-Parameter **Userid** ausgewählt, so dass der Benutzer-ID, der mit einer CICS-Transaktion assoziiert ist, als AutorisierungsID verwendet wird. Als *DB2ID* wurde hier der Name des DB2-Subsystems eingetragen.

Nachdem die DB2CONN erfolgreich erstellt wurde, müsste sie mit dem Befehl **CEDA INSTALL DB2CONN(DB2CON) GROUP(PSELES)** installiert und mit Hilfe der **CEMT**-Transaktion (*s. Abbildung 3.5*) gestartet werden.

Abbildung 3.5: Das Starten der CICS-DB2-Verbindung.

```
CEMT 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(              )
```

Fortsetzung auf der nächsten Seite ...

Fortsetzung der Abbildung 3.5

```
7     Comthreadlim( 0001 )          Signid( CICS     )
8     Comthreads(0000)              Security(             )
9     Connecterror( Sqlcode )       Standbymode( Reconnect )
10    Connectst( Connected )        Statsqueue( CDB2 )
11    Db2groupid(      )            Tcblimit( 0012 )
12    Db2id( DB7G )                 Tcbs(0000)
13    Db2release(0710)              Threaderror(N906d)
14    Drollback(Rollback)           Threadlimit( 0003 )
15    Msqqueue1( CDB2 )             Threads(0000)
16    Msgqueue2(      )             Threadwait( Twait )
17    Msqqueue3(      )
18    Nontermrel( Release )
19    Plan(          )
20                                                    SYSID=CICS APPLID=CICS
21  RESPONSE: NORMAL
```

Der Verbindungsstatus **Connected** (*Zeile 10*) zeigt, dass der Startvorgang erfolgreich durchgeführt wurde.

3.4.2 Definition des DB2ENTRY

Die Entry-Thread Definition für die CICS-Datenbankverbindung wird in derselben Gruppe **PSELES** erstellt (*Abbildung 3.6*).

```
CEDA DEFINE DB2ENTRY (DB2CONE) GROUP(PSELES)
1   OBJECT CHARACTERISTICS                                CICS RELEASE = 0630
2    CEDA  DEFINE DB2Entry( DB2CONE )
3     DB2Entry       : DB2CONE
4     Group          : PSELES
5     DEscription    :
6    THREAD SELECTION ATTRIBUTES
7     TRansid        : *
8    THREAD OPERATION ATTRIBUTES
9     ACcountrec     : None            None | TXid | TAsk | Uow
10    AUTHId         :
11    AUTHType       : Userid          Userid | Opid | Group | Sign | TErm | TX
12    DRollback      : Yes             Yes | No
13    PLAN           : DB2C
14    PLANExitname   :
15    PRIority       : High            High | Equal | Low
16    PROtectnum     : 0000            0-2000
17    THREADLimit    : 0000            0-2000
18    THREADWait     : Pool            Pool | Yes | No
19                                                  SYSID=CICS APPLID=CICS
20  DEFINE SUCCESSFUL
```

Abbildung 3.6: Definition des Entry-Thread.

Mit dem *TRansid*-Attribut werden die Transaktionen (*in diesem Fall alle Transaktionen*) festgelegt, die diese Art des Threads verwenden können. Wie bei der DB2CONN-Definition (*s. Abbildung 3.4 auf Seite 21*) wird für das *AUTHType*-Attribut **Userid** eingegeben. Als *Plan* trägt man den Name des Anwendungsplanes an, der von jeder Transaktion dieser Definition verwendet wird. Als *THREAD-Wait* wurde **Pool** gewählt, so dass jede Transaktion, für die kein freier Entry-Thread gefunden wird, ein Pool-Thread (*POOL THREAD Sektion innerhalb der DB2CONN-Definition*) reserviert wird. Dabei verwenden solche umgeleiteten Transaktionen den festgelegten Anwendungsplan des DB2ENTRY.

Anschließend könnte die DB2ENTRY mittels **CEDA INSTALL DB2ENTRY(DB2CONE) GROUP(PSELES)** installiert werden.

3.5 CICS-Konfiguration

Für die EXCI-Verbindung und auch für die Sicherheitsmechanismen, die in vorigen Abschnitten beschrieben wurden, soll die CICS-Region mit den entsprechenden Konfigurationsparametern gestartet werden. Die Konfiguration der CICS-Region erfolgt in der SIT (*System Initialization Table*). Die Quelldatei dieser Tabelle befindet sich auf dem *padme*-Rechner in Member **DFH$SIPX** des Datasets **CICSTS23.SYSIN**.

Als Erstes soll RACF für die CICS-Security aktiviert werden. Dies erfolgt durch Setzen des Parameters **SEC=YES** innerhalb der CICS-Parametertabelle. Für die EXCI-Verbindung sind folgende drei SIT-Parameter erforderlich:

- **ISC=YES** dient zur Aktivierung der *Intersystem Communication*, welche für die *Interregion Communication* (IRC) erforderlich ist.
- Mit dem Parameter **IRCSTRT=YES** wird die *Interregion Communication* während der Initialisierung der CICS-Region automatisch gestartet.
- Die Unterstützung des *Resource Recovery Services* (RRS) wird mit dem Parameter **RRMS=YES** aktiviert.

In der SIT kann auch definiert werden, ob die DB2-AutorisationsIDs (*s. Abschnitt 3.4.1 auf Seite 21*) über RACF geschützt werden sollen. Obwohl dieses Form der Sicherheit nicht zur *Surrogate-security* gehört, wird sie in der CICS-Region durch SIT-Parameter **XUSER=YES** gesteuert.

Anschließend müsste CICS neu gestartet werden, damit die neue Konfiguration in Kraft treten kann. Dazu innerhalb der CICS führt man den Befehl **CEMT PERFORM SHUTDOWN IMMEDIATE** aus, um CICS zu stoppen. Danach vom SDSF (*System Display and Search Facility*)-Panel wird CICS mit dem Befehl **/S CICSA** wieder gestartet.

3.6 Erstellung des CICS-DB2-Programms

Nachdem alle Vorbereitungen in den vorherigen Abschnitten dieses Kapitels vorgenommen wurden, kann es an dieser Stelle mit der Erstellung des CICS-COBOL-Programms begonnen werden. Mit

Hilfe des zu erstellenden CICS-Programms sollen die Daten in einer Datenbank geändert, neue hinzugefügt oder bestehende gelöscht werden. Die Datenbank kann unter Anweisungen des Tutorials 4 [Tut03] oder mit Hilfe des folgenden JCL-Programms erstellt werden.

```
0  PSELES.CICS.DB2(CREATEDB)
1   //CRTDBTB JOB (),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
2   //RUNTIAD EXEC PGM=IKJEFT01
3   //STEPLIB DD DISP=SHR,DSN=DSN710.SDSNLOAD
4   //DBRMLIB DD DISP=OLD,DSN=PSELES.DBRMLIB.DATA(CRTDB)
5   //SYSPRINT DD SYSOUT=*
6   //SYSTSPRT DD SYSOUT=*
7   //SYSUDUMP DD SYSOUT=*
8   //SYSTSIN DD *
9     DSN SYSTEM(DB7G)
10     RUN PROGRAM(DSNTIAD) PLAN(DSNTIA71) PARMS('RC0') -
11        LIBRARY('DSN710.RUNLIB.LOAD')
12    END
13  //SYSIN DD *
14      CREATE STOGROUP STOGRPS1
15             VOLUMES (Z5DB21)
16             VCAT DSN710;
17      COMMIT;
18      CREATE DATABASE DBPS1
19             STOGROUP STOGRPS1
20             BUFFERPOOL BP0;
21      COMMIT;
22      CREATE TABLESPACE TABSPPS1 IN DBPS1
23           USING STOGROUP STOGRPS1
24                 PRIQTY 20
25                 SECQTY 20
26                 ERASE NO
27           BUFFERPOOL BP0
28           CLOSE NO;
29      COMMIT;
30      CREATE TABLE TABPS1
31           ( VNAME CHAR(20) NOT NULL,
32             NNAME  CHAR(20) NOT NULL PRIMARY KEY,
33           ) IN DBPS1.TABSPPS1;
34      COMMIT;
35      CREATE UNIQUE INDEX KINDEX ON PSELES.TABPS1 (NNAME);
36      COMMIT;
37  /*
```

Programm 3.1: JCL-Programm zum Erstellen einer Datenbank.

Diese Datenbank enthält eine leere Tabelle mit zwei Feldern VNAME (*Vorname*) und NNAME (*Nachname*). Dabei ist zu beachten, dass die Werte für VOLUMES und VCAT des Speicherplatzes (*STOGROUP*) installationsspezifisch sind (*Zeilen 15-16*). Für den *padme*-Rechner sind es Z5DB21 und DSN710.

Aus Gründen der Übersichtlichkeit wird hier nur eine gekürzte Version des CICS-Programms dargestellt (*s. Programm 3.2 auf der nächsten Seite*), welche die grundlegende Funktionalitäten des CICS-Programms aus der Kapitel 4 auf Seite 33 enthält. Für das zu erstellende Programm sollen drei Datasets PSELES.CICS.DB2, PSELES.DBRMLIB.DATA und PSELES.LIB vorhanden sein, die unter Anweisungen des Tutorial 14 [Tut05] erstellt werden können.

```
0  PSELES.CICS.DB2(DB2PCICS)
1        IDENTIFICATION DIVISION.
2        PROGRAM-ID. DB2PCICS.
3        ENVIRONMENT DIVISION.
4        DATA DIVISION.
5        WORKING-STORAGE SECTION.
6        01 NAME-TAB.
7           02 VORNAME        PIC X(20) VALUE 'Pawel'.
8           02 NACHNAME       PIC X(20) VALUE 'Selesnjov'.
9           02 UPDATE-NNAME   PIC X(20) VALUE 'Morlang'.
10          02 VN             PIC X(20).
11          02 NN             PIC X(20).
12       01 REQUEST.
13          02 GINS           PIC X(06) VALUE 'insert'.
14          02 GUPD           PIC X(06) VALUE 'update'.
15          02 GDEL           PIC X(06) VALUE 'delete'.
16          EXEC SQL INCLUDE SQLCA END-EXEC.
17          EXEC SQL DECLARE C1 CURSOR FOR
18              SELECT VNAME,NNAME FROM PSELES.TABPS1
19          END-EXEC.
20       LINKAGE SECTION.
21       01 DFHCOMMAREA.
22          03 REQ            PIC X(06).
23          03 VNAM           PIC X(20).
24          03 NNAM           PIC X(20).
25       PROCEDURE DIVISION.
26          EVALUATE REQ
27            WHEN GINS
28                EXEC SQL INSERT
29                    INTO PSELES.TABPS1(VNAME, NNAME)
30                        VALUES (:VORNAME, :NACHNAME)
31                END-EXEC
32                PERFORM DBAUSGABE
33            WHEN GUPD
34                EXEC SQL UPDATE PSELES.TABPS1
35                    SET VNAME=:VORNAME, NNAME=:UPDATE-NNAME
36                    WHERE VNAME=:VORNAME
37                END-EXEC
38                PERFORM DBAUSGABE
39            WHEN GDEL
40                EXEC SQL DELETE FROM PSELES.TABPS1
41                    WHERE NNAME=:UPDATE-NNAME
42                END-EXEC
43                PERFORM DBAUSGABE
44            WHEN OTHER
45                PERFORM DBAUSGABE
46          END-EVALUATE.
47          MOVE SPACES TO REQ.
48          EXEC CICS RETURN END-EXEC.
49          EXIT.
50       DBAUSGABE SECTION.
51          EXEC SQL OPEN C1 END-EXEC.
52          EXEC SQL FETCH C1 INTO :VN, :NN END-EXEC.
53          IF SQLCODE IS ZERO
54            MOVE VN TO VNAM
55            MOVE NN TO NNAM.
56          EXEC SQL CLOSE C1 END-EXEC.
57       DBAUSGABE-EXIT.
```

Programm 3.2: Der Quelltext des Programms DB2PCICS.

Die COMMAREA (*ab Zeile 21*) dieses Programm enthält drei Felder. Das REQ-Feld steht für die Abfragen, die an das CICS-Programm gesendet werden. Die zwei anderen Felder VNAM und

NNAM sind für das Auslesen der Daten aus der Datenbank reserviert.

In der EVALUATE-Anweisung (*ab der Zeile 26*) werden verschiedene Anweisungsblöcke in Abhängigkeit vom Wert des REQ-Feldes ausgeführt. So wird eine vordefinierte Zeile (VORNAME und NACHNAME Variablen) in die Datenbank hinzugefügt (*Zeilen 27-32*), wenn REQ-Wert gleich "insert" ist. Falls REQ-Variable den Wert "update" hat, wird die erste Zeile der Datenbank abgeändert (*Zeilen 33-38*). Und wenn REQ den Wert "delete" besitzt, wird diese Zeile gelöscht (*Zeilen 39-43*).

Nach jeder Operation wird die erste Zeile der Tabelle mit Hilfe des Cursors `C1` in Variablen VN und NN eingelesen (*die Funktion DBAUSGABE ab der Zeile 50*) und mittels MOVE-Anweisung zur COMMAREA übertragen.

3.6.1 Übersetzung des CICS-DB2-Programms

Jedes CICS-Programm, welches eingebettete SQL-Anweisungen enthält, kann wie ein gewöhnliches CICS-Programm kompiliert werden, mit dem einzigen Unterschied, dass noch ein Binden-Schritt hinzukommt. Das Binden ist ein Prozess, bei dem ein Anwendungsplan erstellt wird, der ein CICS-Programm während der Ausführung für den Zugriff auf die DB2-Ressourcen und zur Unterstützung von SQL-Anforderungen benötigt.

Grundsätzlich unterteilen sich die Übersetzungsschritte eines Programms in vier Gruppen: Vorübersetzen, Übersetzen, Binden und Erteilung von Berechtigungen (*Programm 3.3*).

Programm 3.3: JCL-Programm zur Übersetzung des CICS-DB2-Programms [SC106].

```
0  PSELES.CICS.DB2(DB2CCOMP)
1   //DB2CCOMP JOB (),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
2   //VARS SET HOME='PSELES.CICS.DB2',
3   //     DATIN=DB2PCICS,
4   //     DATOUT=PCOMP
5   //* Precompiling
6   //PC   EXEC PGM=DSNHPC,
7   //     REGION=4096K,
8   //     PARM='HOST(COBOL),XREF,FLAG(I)'
9   //STEPLIB DD DISP=SHR,DSN=DSN710.SDSNEXIT
10  //        DD DISP=SHR,DSN=DSN710.SDSNLOAD
11  //DBRMLIB DD DISP=OLD,DSN=PSELES.DBRMLIB.DATA(DB2DL)
12  //SYSCIN  DD DISP=(OLD,PASS),DSN=&HOME(&DATOUT),UNIT=SYSDA,
13  //        SPACE=(800,(500,500))
14  //SYSLIB  DD DISP=SHR,DSN=&HOME
15  //SYSPRINT DD SYSOUT=*
16  //SYSTERM DD SYSOUT=*
17  //SYSUDUMP DD SYSOUT=*
18  //SYSUT1  DD SPACE=(800,(500,500),,,ROUND),UNIT=SYSDA
19  //SYSUT2  DD SPACE=(800,(500,500),,,ROUND),UNIT=SYSDA
20  //SYSIN   DD DISP=SHR,DSN=&HOME(&DATIN)
21  //* Bind
22  //BIND    EXEC PGM=IKJEFT01,
23  //        COND=((4,LT,PC))
24  //STEPLIB DD DISP=SHR,DSN=DSN710.SDSNEXIT
25  //        DD DISP=SHR,DSN=DSN710.SDSNLOAD
26  //DBRMLIB DD DISP=OLD,DSN=PSELES.DBRMLIB.DATA(DB2DL)
```

Fortsetzung auf der nächsten Seite ...

Fortsetzung des Programms 3.3

```
27 //SYSPRINT DD SYSOUT=*
28 //SYSTSPRT DD SYSOUT=*
29 //SYSUDUMP DD SYSOUT=*
30 //SYSTSIN DD *
31   DSN SYSTEM(DB7G)
32    BIND PACKAGE (DB2C) MEMBER(DB2DL) ACTION(REPLACE) ISOLATION(CS)
33    BIND PLAN(DB2C) PKLIST(DB2C.*)
34   END
35 //* Compile
36 //CICS      EXEC DFHYITVL
37 //TRN.SYSIN DD DISP=SHR,DSN=&HOME(&DATOUT)
38 //COB.SYSLIB DD DSN=PSELES.LIB,DISP=SHR
39 //LKED.CICSLOAD DD DISP=SHR,DSN=CICSTS23.CICS.SDFHLOAD
40 //LKED.DB2LOAD DD DISP=SHR,DSN=DSN710.SDSNLOAD
41 //LKED.SYSIN DD *
42   INCLUDE CICSLOAD(DSNCLI)
43   INCLUDE DB2LOAD(DSNTIAR)
44   INCLUDE DB2LOAD(DSNHADDR,DSNHADD2)
45   NAME DB2PCICS(R)
46 /*
47 //* Grant
48 //GRANT EXEC PGM=IKJEFT01
49 //STEPLIB DD DISP=SHR,DSN=DSN710.SDSNLOAD
50 //SYSPRINT DD SYSOUT=*
51 //SYSTSPRT DD SYSOUT=*
52 //SYSUDUMP DD SYSOUT=*
53 //SYSTSIN DD *
54   DSN SYSTEM(DB7G)
55     RUN PROGRAM(DSNTIAD) PLAN(DSNTIA71) LIBRARY('DSN710.RUNLIB.LOAD')
56   END
57 //SYSIN DD *
58   GRANT EXECUTE ON PLAN DB2C TO PUBLIC
59 /*
```

- Schritt **Precompiling**
  Das Programm, das sich im Dataset PSELES.CICS.DB2(DB2PCICS) befindet (*Zeile 20*), wird mit Hilfe des DB2-Precompilers[3] (`DSNHPC`) (*Zeile 6*) analysiert. Daraufhin wird ein abgeänderter Quellcode des Programms in die Datei PSELES.CICS.DB2(PCOMP) (*Zeile 12*) geschrieben, welcher bei der späteren Kompilierung verwendet wird. In diesem modifizierten Quellcode sind die SQL-Anweisungen in Aufrufe der DB2-Laufzeit-API umgewandelt und auskommentiert.

  Ist das Cobol-Programm fehlerfrei, erzeugt der Precompiler auch ein Datenbankanforderungsmodul (*DBRM - Database Request Module*), das alle SQL-Anweisungen und *Host*-Variablen des Programms enthält und für das spätere Binden benötigt wird.

- Schritt **Binding**
  Ist die Vorkompilierung erfolgreich[4] abgeschlossen wurde, wird im Binden-Schritt (*ab der Zeile 22*) der zuvor erstellte DBRM-Modul `DB2DL` (*Zeile 26*) an einen Anwendungsplan gebunden. Es wird mit Hilfe des Programms `IKJEFT01` das Paket (*package*) generiert (*Zeile 32*), das zu einer *Collection*[5] `DB2C` hinzugefügt wird. Dabei ist zu beachten, dass

[3] Eine andere Alternative wäre es die Verwendung eines *SQL-Statement Coprocessors*, der von einem Compiler zur Verfügung gestellt werden kann.
[4] *Condition Code* des Precompilers ist kleiner als 4 (*Zeile 23*).
[5] Die Collection stellt lediglich einen Verweis zur einer Gruppe von vereinigten Paketen dar.

ein neues Paket mit der Bindeoption ACTION(REPLACE) erstellt werden soll.

Anschließend werden die Pakete der Collection `DB2C` in die Paketliste (*Package List*) des Planes `DB2C` aufgenommen (*Zeile 33*). Der Vorteil der Verwendung der Paketen liegt darin, dass beim erneuten Binden eines Programms nur die einzelnen Pakete zum Einsatz kommen, ohne es der gesamte Plan erneut erstellt werden muss.

- Schritt **Compiling**
  Im nächsten Schritt (*ab Zeile 36*) wird das CICS-Programm mit Hilfe der CICS-Prozedur `DFHYITVL` kompiliert. Die Prozedur enthält drei Schritten zur Translation, Kompilierung und zum Linken des Programms.

  Vor der eigentlichen Kompilierung, die durch den COBOL-Compiler (`IGYCRCTL`) erfolgt, wird zuerst der COBOL-Translator (`DFHECP1$`) ausgeführt. Der Translator verwendet als Eingabe die vom DB2-Precompiler erstellte Datei `PCOMP` (*Zeile 37*), in der alle CICS-Befehle (*EXEC CICS*- oder *EXEC DLI*-Anweisungen) in die *COBOL CALL*-Anweisungen umgewandelt werden, so dass der Compiler sie verstehen kann.

  Nach der Kompilierung wird mit Hilfe des *Linkers* ein ausführbares Lademodul erstellt, das standartmäßig in Dataset CICSTS23.CICS.SDFHLOAD abgelegt wird. Dafür benötigt man einige Schnittstellenmodule (*interface modules*) aus CICS und DB2. Entscheidend dabei ist das Modul `DSNCLI` (*CICS DB2 language interface module*), das die Konnektivität mit DB2 ermöglicht (*Zeile 42*).

- Schritt **Grant**
  Im letzten Schritt (*ab Zeile 47*) werden die Ausführungsrechte für den Plan `DB2C` an alle Benutzer vergeben.

Mit dem SUBMIT-Befehl `SUB` führt man die oben genannte Schritte aus, um das Programm **DB2PCICS** zu kompilieren.

```
14.56.34 JOB06567 $HASP165 DB2CCOMP ENDED AT N1 MAXCC=0 CN(INTERNAL)
***
```

Nachdem JES (*Job Entry Subsystem*) den Job `DB2CCOMP` ausgegeben hat (*Condition Code 0 oder 4*), kann das übersetzte Programm in dem CICS-Subsystem installiert werden. Dazu muss es zuerst definiert werden, was mit Hilfe der CEDA-Transaktion geschieht (*s. Abbildung 3.7*).

Abbildung 3.7: Definition des Programms **DB2PCICS**.

```
1  CEDA DEFINE PROGRAM(DB2PCICS) GROUP(PSELES)
2  OVERTYPE TO MODIFY                                    CICS RELEASE = 0630
3   CEDA  ALter PROGram( DB2PCICS )
4    PROGram        : DB2PCICS
5    Group          : PSELES
6    DEscription  ==>
7    Language     ==> Le370              CObol | Assembler | Le370 | C | Pli
8    RELoad       ==> No                 No | Yes
```

Fortsetzung auf nachster Seite . . .

Fortsetzung der Abbildung 3.7

```
     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
```

Als *Language*-Parameter wählt man `Le370` aus. Die andere Parameter können ohne Änderungen übernohmen werden.

Nun muss das Programm DB2PCICS mit dem CEDA-Befehl **CEDA INSTALL PROGRAM(DB2PCICS) GROUP(PSELES)** installiert werden. Nachdem das Programm erfolgreich eingerichtet wurde, ist es für den nachfolgenden Testlauf bereit.

3.7 Test des CICS-Programms

Zum Testen des zuvor erstellten und installierten CICS-Programms auf seine Funktionalität können zwei Vorgehensweisen verwendet werden. Das Eine ist das Testen innerhalb von CICS mit Hilfe der CECI-Transaktion. Die andere Möglichkeit ist die Verwendung der J2EE-Anwendung **CTGTesterCCI** aus dem Tutorial 14 [Tut05], die in diesem Test eingesetzt wird.

Die Anwendung besteht aus einem *Servlet* und einer *Session Bean* und speziell für die Aufrufe von Programmen innerhalb der CICS-Region entwickelt ist [PJ02]. Die *Session Bean* der Anwendung greift auf ein CICS-Programm über den CICS-ECI-Ressourcenadapter, der in diesem Fall auf dem WebSphere Anwendungsserver für z/OS installiert und konfiguriert war.

Insgesamt besteht der Test aus drei Durchläufen für das Einfügen, Ändern und Löschen von Datensätzen in der Datenbank, der im Abschnitt 3.6 auf Seite 23 erstellt wurde. Als Voraussetzung für die Durchführung des folgenden Tests muss die Datenbank leer sein.

Es wird mit dem Einfügeschritt angefangen. Dazu ruft man die J2EE-Anwendung unter der Adresse *http://padme.informatik.uni-leipzig.de:9148/CTGTesterCCIWeb* auf. Die Abbildung 3.8 auf der nächsten Seite zeigt die Startseite dieser J2EE-Anwendung.

Die Anwendung wird in einer verwalteten Umgebung ausgeführt (für die Option **managed** ist **Yes** ausgewählt). Das bedeutet, dass der Ressourcenadapter für den Zugriff auf das CICS-Programm die Parameter der *Connection Factory*, solche wie Gateway deamon URL oder der Name des CICS Servers, verwendet.

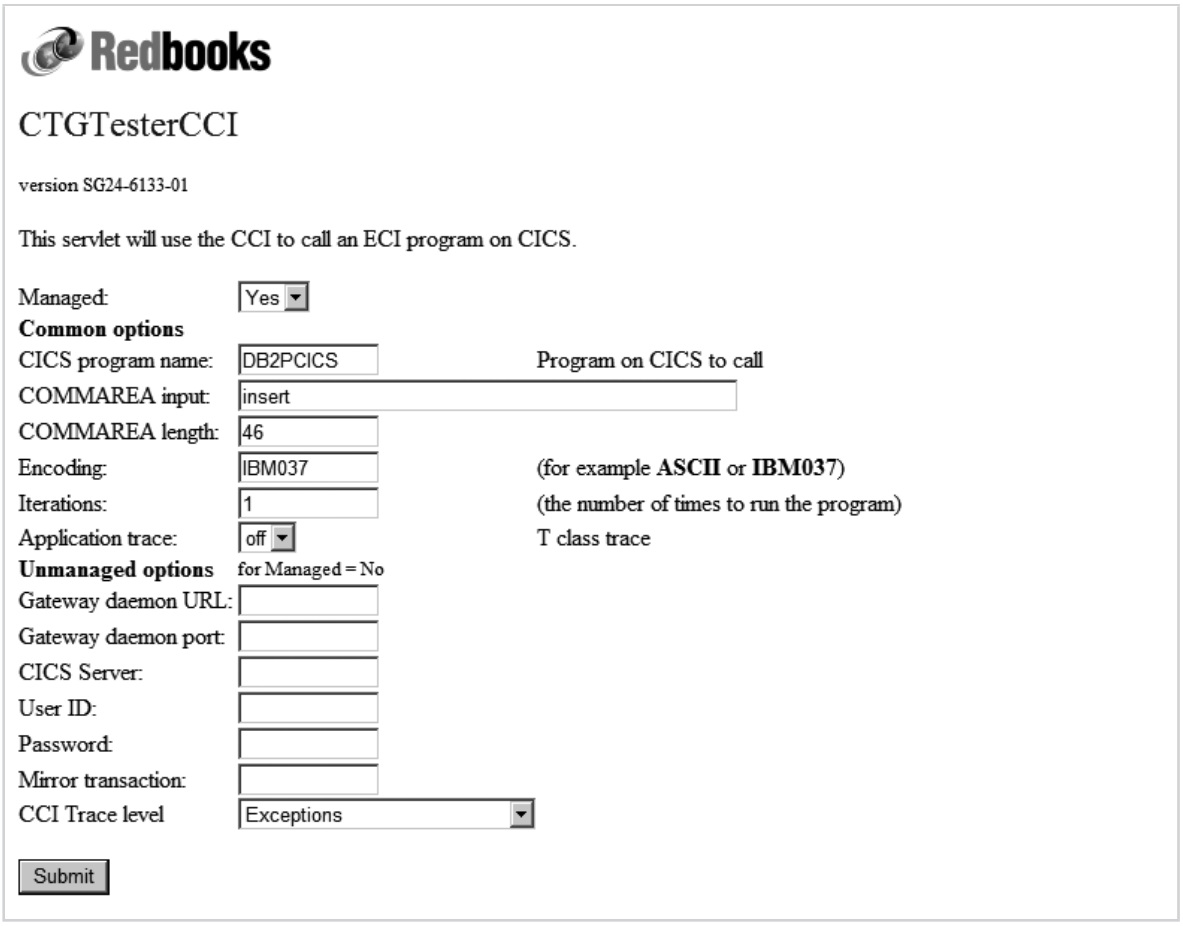


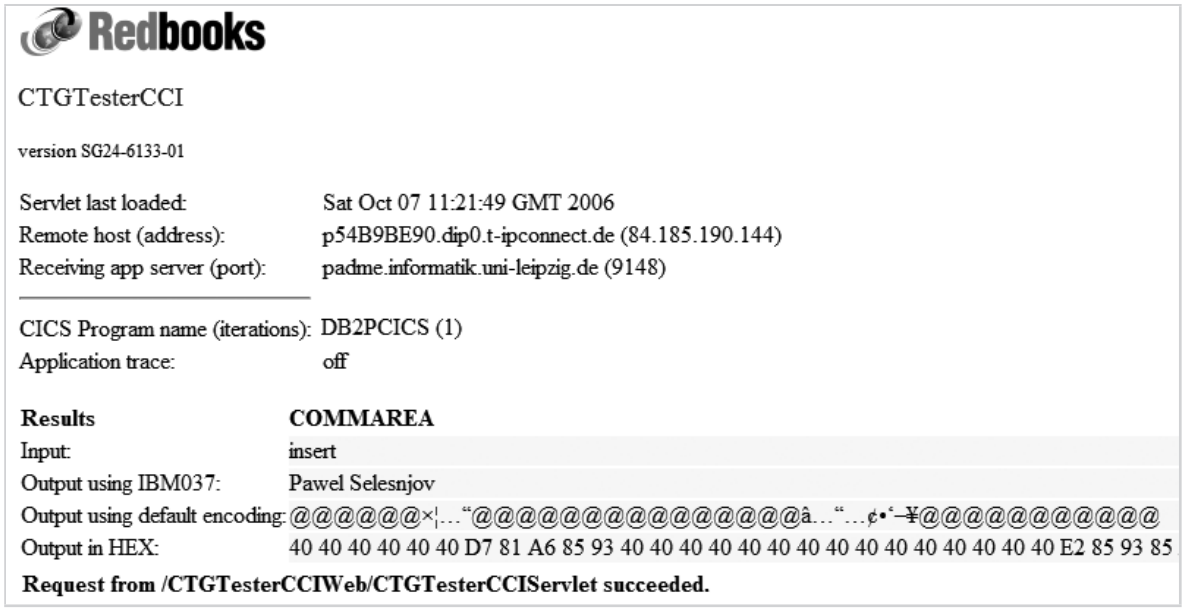
Abbildung 3.8: Startseite der J2EE-Anwendung CTGTesterCCI.

Weiterhin müssen einige spezifische Parameter des CICS-Programms DB2PCICS festgelegt werden. Für die Eingabedaten der COMMAREA, für die die ersten 6 Zeichen der COMMAREA reserviert sind, soll insert angegeben werden. Zusammen mit Ausgabedaten der COMMAREA beträgt die gesammte COMMAREA-Länge 46 Zeichen.

Die Kodierung muss in IBM037 verändert werden, da für das CICS-Programm kein Eintrag in das DFHCNV-Makro[6] eingetragen wurde, und somit keine Datenkonvertierung für das DB2PCICS-Programm von EBCDIC in ASCII durchgeführt wird. Anschließend betätigt man die Schaltfläche Submit.

Daraufhin erscheint die Ergebnisseite der J2EE-Anwendung (Abbildung 3.9 auf der nächsten Seite). Die gerade eingefügte Zeile wurde aus der Tabelle PSELES.TABPS1 in die COMMAREA geschrieben (*siehe Ablauf des CICS-Programms 3.2 auf Seite 25*) und in verschiedenen Kodierungen ausgegeben. Die einzige lesbare Ausgabe ist in der IBM037-Kodierung angezeigt. Die hexadezimale Ausgabe zeigt, dass die ersten sechs Zeichen leer sind. Sie sind in unserem CICS-Programm für Anfrage reserviert und wurden absichtlich als Leerzeichen ausgegeben.

[6] Die Tabelle, welche die entsprechende Konvertierungsregeln (*Schablonen*) für die CICS-Ressourcen enthält.



Redbooks

CTGTesterCCI

version SG24-6133-01

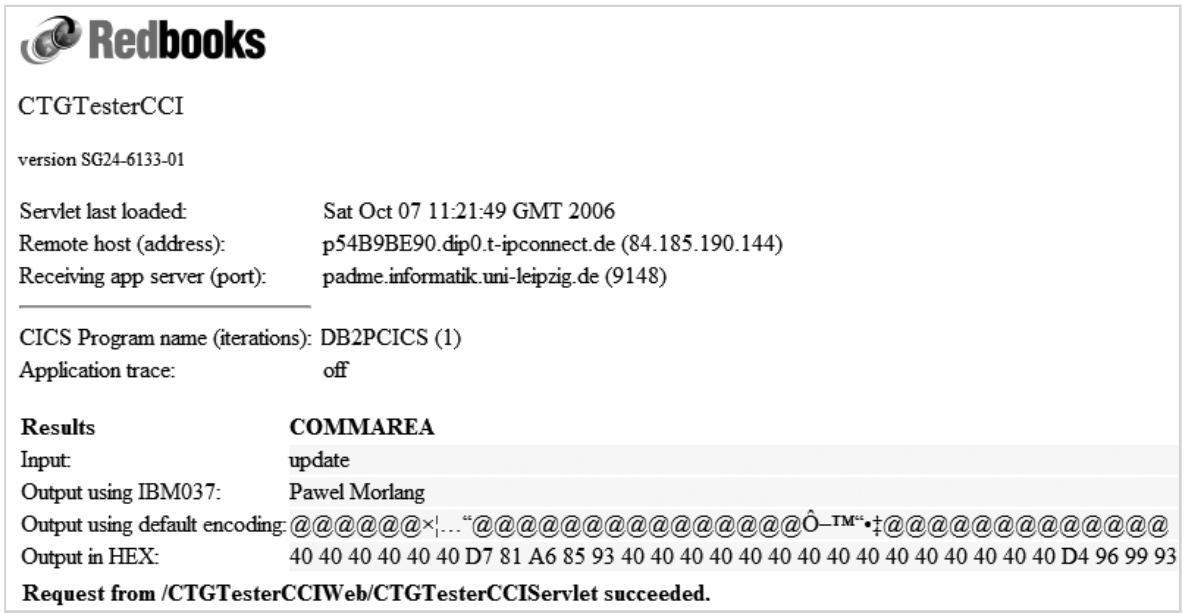
Servlet last loaded:	Sat Oct 07 11:21:49 GMT 2006
Remote host (address):	p54B9BE90.dip0.t-ipconnect.de (84.185.190.144)
Receiving app server (port):	padme.informatik.uni-leipzig.de (9148)
CICS Program name (iterations):	DB2PCICS (1)
Application trace:	off

Results	**COMMAREA**
Input:	insert
Output using IBM037:	Pawel Selesnjov
Output using default encoding:	@@@@@@×¦…“@@@@@@@@@@@@@@@â…“…¢•‘–¥@@@@@@@@@@
Output in HEX:	40 40 40 40 40 40 D7 81 A6 85 93 40 40 40 40 40 40 40 40 40 40 40 40 40 40 E2 85 93 85

Request from /CTGTesterCCIWeb/CTGTesterCCIServlet succeeded.

Abbildung 3.9: Einfügen eines Datensatzes in die Datenbank.

Nun wiederholt man den Schritt aus der Abbildung 3.8 mit dem Parameter update im COMMAREA input Feld. Als Ergebnis wird der geänderte Nachname angezeigt (*Abbildung 3.10*).



Redbooks

CTGTesterCCI

version SG24-6133-01

Servlet last loaded:	Sat Oct 07 11:21:49 GMT 2006
Remote host (address):	p54B9BE90.dip0.t-ipconnect.de (84.185.190.144)
Receiving app server (port):	padme.informatik.uni-leipzig.de (9148)
CICS Program name (iterations):	DB2PCICS (1)
Application trace:	off

Results	**COMMAREA**
Input:	update
Output using IBM037:	Pawel Morlang
Output using default encoding:	@@@@@@×¦…“@@@@@@@@@@@@@@@Ô–™“•‡@@@@@@@@@@@@@
Output in HEX:	40 40 40 40 40 40 D7 81 A6 85 93 40 40 40 40 40 40 40 40 40 40 40 40 40 40 D4 96 99 93

Request from /CTGTesterCCIWeb/CTGTesterCCIServlet succeeded.

Abbildung 3.10: Ändern eines Datensatzes in der Datenbank.

Anschließend wird die Zeile aus der Datenbank entfernt. Dazu kehrt man zurück zur Startseite der Anwendung (*s. Abbildung 3.8 auf der vorherigen Seite*) und nennt das Feld COMMAREA input in delete um. Es erscheint die Ergebnisseite (*Abbildung 3.11 auf der nächsten Seite*), wo das Feld Output using IBM037 leer ist.

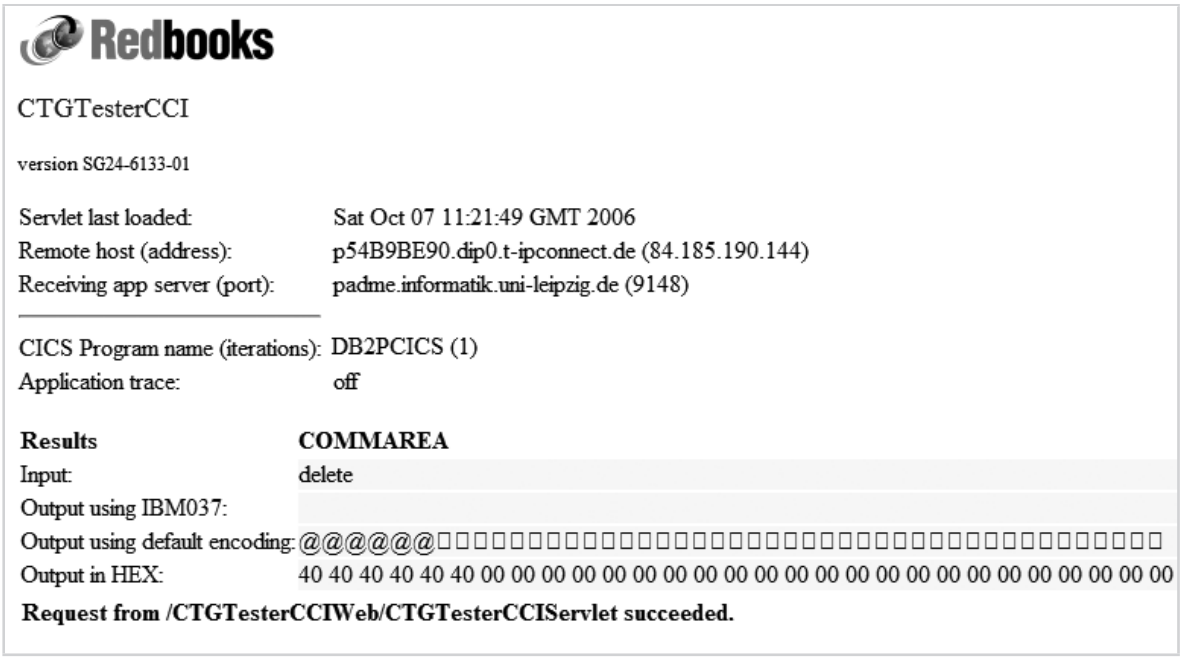


Abbildung 3.11: Löschen eines Datensatzes aus der Datenbank.

Somit wurde der Test erfolgreich absolviert. Es wurde das Zusammenspiel zwischen der J2EE-Anwendung und dem CICS-Programm demonstriert. Die Kommunikation zwischen beiden geschah dabei über den CICS-ECI-Ressourcenadapter. An dieser Stelle kann mit der Erstellung der eigenen Java-Anwendung begonnen werden, die speziell für das CICS-Programm angepasst ist.

Kapitel 4 Erstellung der J2EE-Anwendung

Die Java-Anwendungen für einen Web Anwendungsserver (besonders Servlets, JSPs, Beans und EJBs) können prinzipiell mit beliebigen Entwicklungswerkzeugen, z.Bsp. dem *Java Development Kit* (*JDK*) erstellt werden. In dieser Arbeit wird die Entwicklungsumgebung *IBM Rational Application Developer* (*IRAD*) eingesetzt. Eine vorhandene Installation der IRAD-Entwicklungsumgebung der Version 6.0 befindet sich auf dem Windows Server 2003 Rechner (IP-Adresse: *139.18.8.211*, hostname: *websphere.informatik.informatik.uni-leipzig.de*).

Das IBM Rational Application Developer stellt einen kompletten Satz von Tools für die Entwicklung und für das Deployment von J2EE-Anwendungen zur Verfügung. Es bietet zahlreiche Hilfethemen und Beispiele, um Arbeit zu erleichtern und die schnelle Einarbeitung in die Software zu ermöglichen. Die meisten folgenden Schritte sind aus dem Beispiel "Using the CICS Resource Adapter to create J2C applications" [IBMa] entnommen. Es beschreibt, wie man eine J2EE-Anwendung erstellt, die auf die CICS-Programme und -Transaktionen mit Hilfe des CICS Ressourcenadapters zugreift.

4.1 Anlegen der J2EE-Projekten

Zuerst soll die Verbindung zu dem Windows Server 2003 hergestellt werden. Unter Windows Betriebssystemen verwendet man dazu die *Remotedesktopverbindung*. Nachdem die Anmeldung erfolgreich durchgefürt wurde, startet man *IRAD*, indem man "Start > Programme > IBM Rational > IBM Rational Application Developer V6.0 > Rational Application Developer" auswählt. Wenn man *IRAD* zum ersten Mal startet, wird die Arbeit standardmäßig in dem Arbeitsbereichverzeichnis "Eigene Dateien\IBM\rationalsdp6.0\workspace" gespeichert.

Die zu erstellende J2EE-Anwendung soll verschiedene Web-Ressourcen wie JSPs, Servlet und Enterprise Java Bean enthalten. Für die Verwaltung dieser Anwendungskomponenten sollen zunächst die Web- und EJB-Projekten angelegt werden, die schließlich zu einem Enterprise Application-Projekt als Module eingeschlossen werden.

Jeder Projekttyp wird mittels eines passenden Assistenten angelegt und im Strukturbaum des *Projekt-Explorers* innerhalb der J2EE-Perspektive angezeigt. Es wird mit der Erstellung eines neuen EAR-Projektes angefangen. Dazu wählt man im Hauptmenü Datei > Neu > Unternehmensanwendungsprojekt aus. Daraufhin erscheint ein Assistent (*Abbildung 4.1 auf der nächsten Seite*), in dem man einen belibigen Namen für das Projekt vergibt (hier: DB2CTGTesterEAR).

Da die Anwendung später auf einem WebSphere Anwendungsserver der Version 5, entweder unter Windows oder z/OS Betriebssystem, laufen soll, muss unter den erweiterten Eigenschaften für den

J2EE-Stand die Stufe 1.3[1] ausgewählt werden.

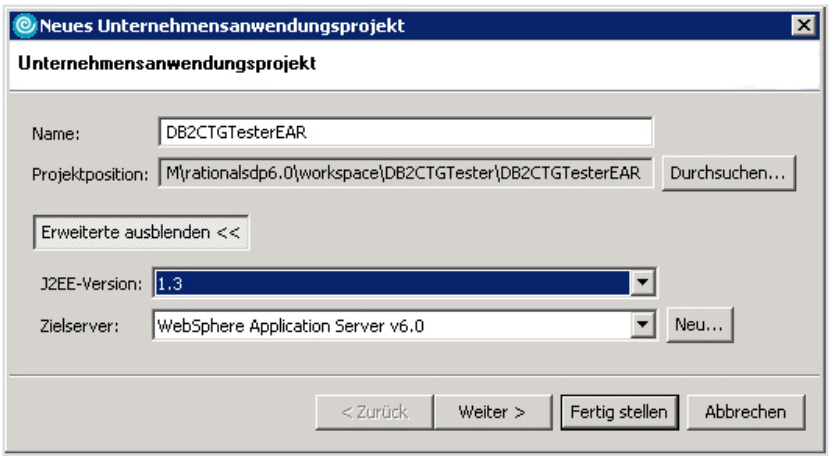


Abbildung 4.1: Der Assistent zur Einrichtung des Unternehmensanwendungsprojektes.

Als Zielserver wählt man **WebSphere Anwendungsserver V6.0**. Er ist die einzige auf dem *websphere*-Rechner installierte lokale Testumgebung. Abschließend kann das Einrichten des neuen EAR-Projektes mit **Fertig stellen** beendet werden.

Als nächstes erstellt man das Web-Projekt. Dazu wählt man im Hauptmenü **Datei > Neu > Dynamisches Webprojekt** aus. Daraufhin öffnet sich der **Neues dynamisches Webprojekt**-Wizard (*Abbildung 4.2*), in dem der Name für das Projekt frei gewählt werden kann (in diesem Fall **DB2CTGTesterWeb**).

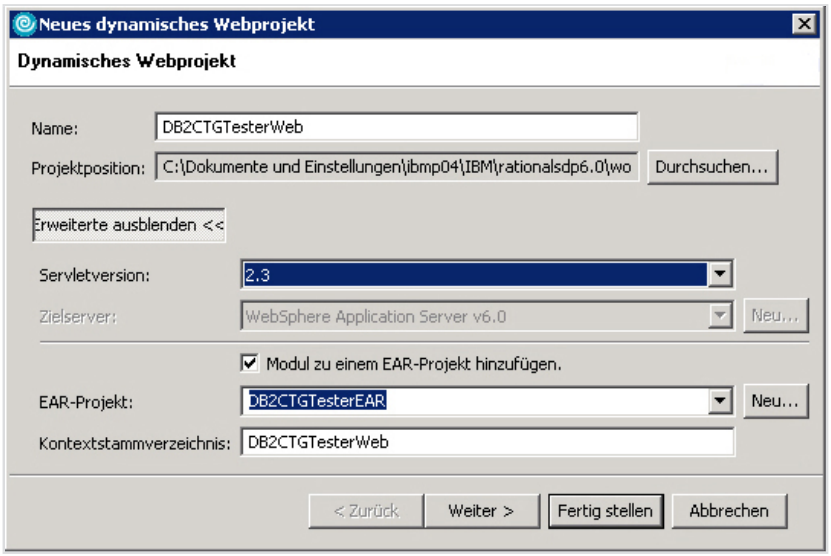


Abbildung 4.2: Der Assistent zur Einrichtung des Web-Projekts.

Als EAR-Projekt wählt man das zuvor erstellte **DB2CTGTesterEAR** Projekt aus. Im *Kontextstammverzeichnis*-Feld wird der Alias für die Anwendung eingegeben. Über diesen Alias wird später in der URL der

[1] J2EE 1.3 enthält Connector Architecture Spezifikation 1.0, Servlet Spezifikation 2.3, JSP Spezifikation 1.2 und EJB Spezifikation 2.0.

Anwendung auf die Projekt-Ressourcen zugegriffen.

Zuletzt legt man EJB-Projekt an (s. *Abbildung 4.3*). Ebenso wie beim Web-Projekt trägt man zuerst einen beliebigen Namen ein und wählt danach das EAR-Projekt aus. Demzufolge wird die EJB Version auf Stufe 2.0 eingesetzt, welche zur J2EE Spezifikation 1.3 gehört.

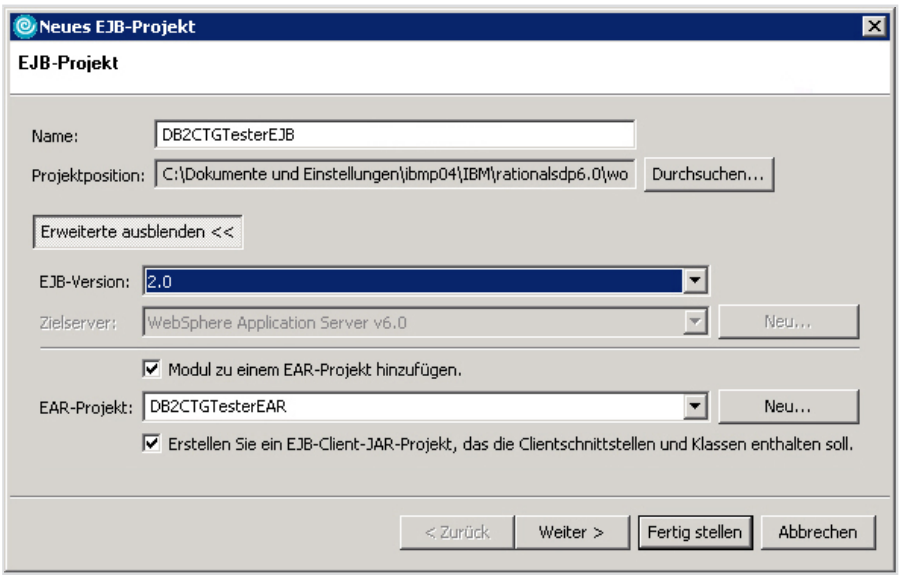


Abbildung 4.3: Der Assistent zur Einrichtung des EJB-Projekts.

4.2 Erstellen der J2C-JavaBean

Es soll eine Java-Bean entworfen werden, die auf der Interaktion zwischen dem CICS-ECI-Ressourcenadapter und dem CICS-COBOL-Programm aufbaut. Für die Erstellung der Java-Bean braucht *IRAD* nur die Linkage-Section des Cobol-Programms, welche die DFHCOMMAREA-Beschreibungen enthält. Daraus erstellt das Entwicklungswerkzeug die Java-Klassen, die als Ein- und Ausgabeparameter zum Aufrufen vom CICS-COBOL-Programm verwendet werden.

4.2.1 Importieren der COBOL-Datei

Im ersten Schritt soll das *DB2PCICS*-Programm, dessen gekürzte Version im Abschnitt 3.6 auf Seite 23 dargestellt wurde, vom *padme*-Rechner in den Arbeitsbereich des Web-Projektes importiert werden.

Dazu wechselt man zur *z/OS-Projekte*-Perspektive und in der *Ferne Systeme*-Sicht klickt man auf das Symbol *z/OS* mit der rechten Maustaste und wählt *Neue Verbindung* aus (*Abbildung 4.4*).

Abbildung 4.4: z/OS Verbindung.

Es wird Assistent für die Erstellung der Verbindung zum fernen z/OS-System aufgerufen. Hier wird lediglich die Adresse des padme-Rechners (139.18.4.35) im *Hostname*-Feld eingegeben. Alle anderen voreingestellten Werte können unverändert gelassen werden.

Danach expandiert man die erstellte Verbindung und loggt sich als TSO-Benutzer ein. Man markiert das DB2PCICS-Programm aus dem Verzeichnis *MVS-Dateien* und zieht es bei gedrückter Maustaste in das *DB2CTGTesterWeb* Projekt herüber (*Abbildung 4.5*).

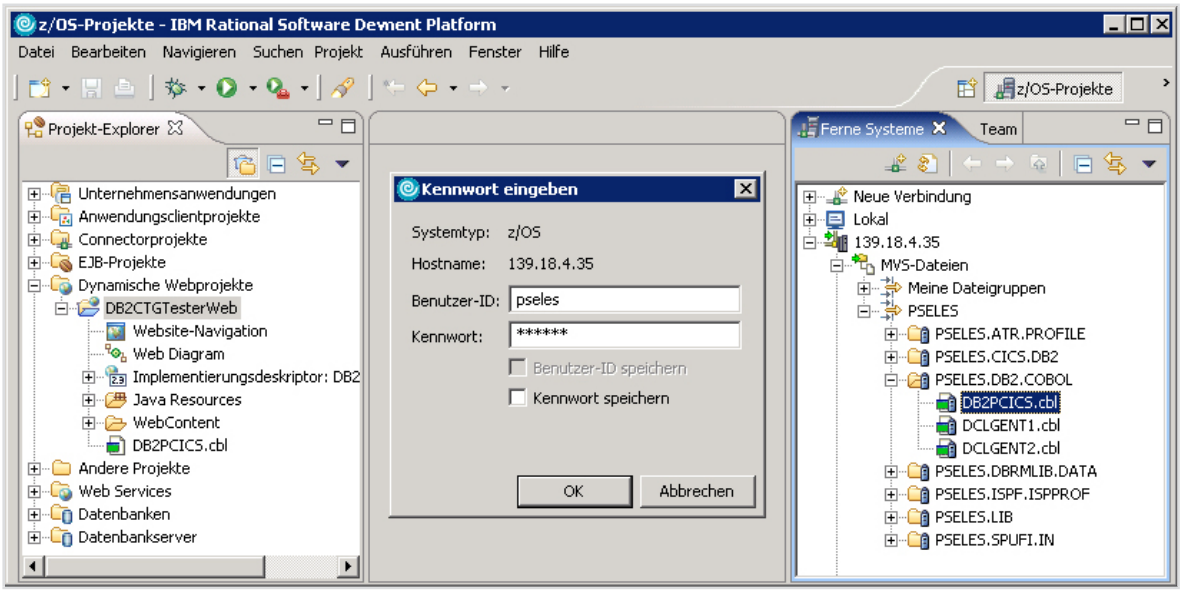


Abbildung 4.5: Import der Cobol-Datei.

Die Commarea des gerade abgelegten Programms ist in der Abbildung 4.1 dargestellt. Ebenfalls wie beim Programm aus dem Abschnitt 3.6 sind die ersten 6 Zeichen der Commarea für die Anfragen reserviert (*Zeile 2*).

Programm 4.1: Commarea des DB2PCICS-Programms.

```
1  01 DFHCOMMAREA.
2     03 GETREQUEST     PIC X(06).
3     03 EINGABE1       PIC X(20).
4     03 EINGABE2       PIC X(20).
5     03 WSPROMSG.
6        05 WsRead1     PIC X(01).
7        05 WsUpdate1   PIC X(01).
8        05 WsRead2     PIC X(01).
9        05 WsUpdate2   PIC X(01).
10    03 WSDB1.
11       05 WsVNAME     PIC X(20).
12       05 WsNNAME     PIC X(20).
13       05 WsBetragEin PIC X(10).
14    03 WSDB2.
15       05 WsSpnr      PIC ZZZZ9.
16       05 WsBesitzer  PIC X(20).
17       05 WsEinlageEin PIC X(10).
18    03 DBT1.
19       05 DBT1ITEM OCCURS 3 TIMES INDEXED BY DB1INDX.
20          10 TsVNAME  PIC X(20).
```

Fortsetzung auf der nächsten Seite ...

Fortsetzung des Programms 4.1

```
21          10 TsNNAME  PIC X(20).
22          10 TsBETRAG PIC -ZZZZZZZZ9.
23       03 DBT2.
24          05 DBT2ITEM OCCURS 3 TIMES INDEXED BY DB2INDX.
25             10 TsSPNR  PIC ZZZZ9.
26             10 TsBESITZER PIC X(20).
27             10 TsEINLAGE PIC -ZZZZZZZZ9.
28       03 ERROR-MSG.
29          05 ERROR-MSGIT OCCURS 4 TIMES INDEXED BY INDX.
30             10 ERR-TEXT PIC X(72).
```

Zur Ausgabe von Datensätzen der Datenbank werden die *Tabellen*[2] verwendet (*ab Zeile 18*). Es werden gleichzeitig drei Datensätze ausgegeben, die durch die OCCURS-Klausel deklariert sind. Dabei besteht jedes Datenfeld aus Feldern der auszugebenden Tabelle. Es werden auch einige Hilfsvariablen benötigt, so z.Bsp. die Tabelle *ERROR-MSG* ab Zeile 28, welche für die Ausgabe der Fehlermeldungen des Cobol-Programms steht.

4.2.2 Erstellen der Java Data Binding-Klassen

Die von CICS zurückgelieferten Daten, die in EBCDIC-Format vorliegen, sind in Java nicht lesbar, da die J2EE-Anwendungen mit Java-Objekten in Unicode arbeiten. Daher soll die Konvertierung der EBCDIC-Daten des COBOL-Programms nach ASCII und umgekehrt stattfinden.

Dazu stellt *IRAD* den **CICS/IMS Java Data Binding** Assistenten bereit, der die Java-Klassen aus Cobol-Datenstrukturen generiert. Diese Klassen, die sogenannten *Java Data Binding* Klassen, werden als Ein- und Ausgabetypen zum Aufrufen vom CICS-COBOL-Programm verwendet. Zum Starten des Assistenten wählt man zuerst **Datei > Neu > Andere** und anschließend in der Kategorie **J2C** die Option **CICS/IMS Java Daten-Binding** aus (*Abbildung 4.6*).

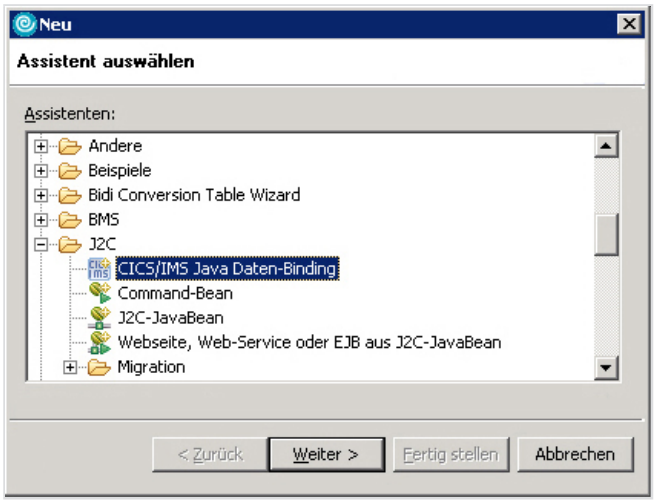


Abbildung 4.6: Auswahl des **CICS Java Daten-Binding** Assistenten.

[2] Die Tabellen gleichen mit der Datenstruktur des Arrays, welche in COBOL nicht vorhanden sind.

Im ersten Schritt des Assistenten wählt man im Zuordnung-Feld **COBOL in Java** und als COBOL-Datei **DB2PCICS.cbl**[3], die im vorigen Abschnitt zu dem Projekt hinzugefügt wurde (*Abbildung 4.7*).

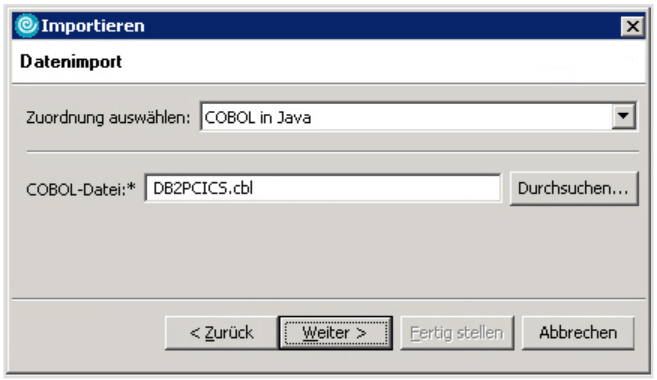


Abbildung 4.7: Auswahl des Cobol-Programms.

Im nächsten Schritt wird die Konvertierung der übertragenen Daten zwischen der J2EE-Anwendung und dem CICS-Programm festgelegt. Wenn die Datenkonvertierung bereits in CICS stattfindet, dann geschieht dies mit Hilfe des CICS Datenkonvertierungsprogramms *DFHCCNV*. In diesem Fall soll im *Plattform*-Feld **Win32** ausgewählt werden. Dabei muss sichergestellt werden, dass keine zweifache Konvertierung vorkommt, was zur Verfälschung der Daten führen kann [AC05].

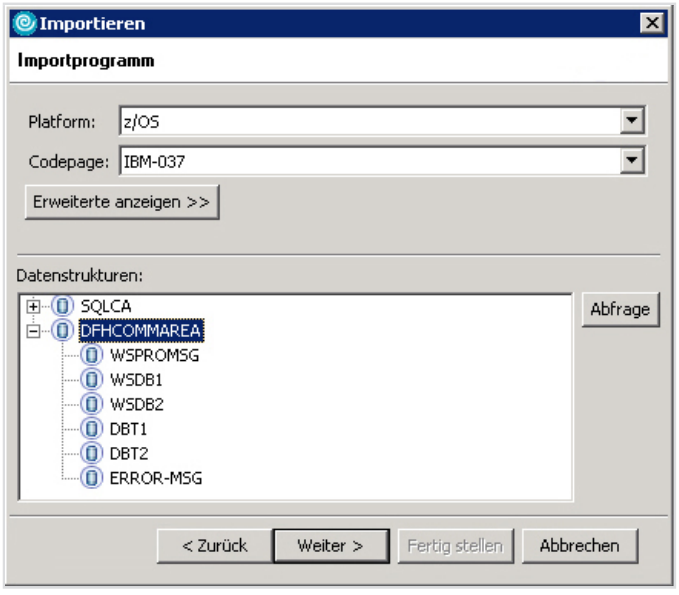


Abbildung 4.8: Auswahl der COBOL Communication area.

In unserem Fall soll die z/OS-Plattform ausgewählt werden (*s. Abbildung 4.8*). Dabei werden die Konvertierungseinstellungen verändert (unter *erweiterte Anzeige*), die ohne Änderungen übernommen werden. Nun betätigt man den Knopf **Abfrage**. Es werden die Datenstrukturen der DB2PCICS.cbl Datei angezeigt.

[3] Für die Cobol-Programmen werden von *IRAD* nur folgende Datei-Endungen unterstützt: *cbl*, *cob*, *ccp* oder *cpy*.

Danach wählt man nur die *DFHCOMMAREA*-Sektion des Cobol-Programms und bestätigt die ausgewählten Einstellungen mit dem Klick auf **Weiter**.

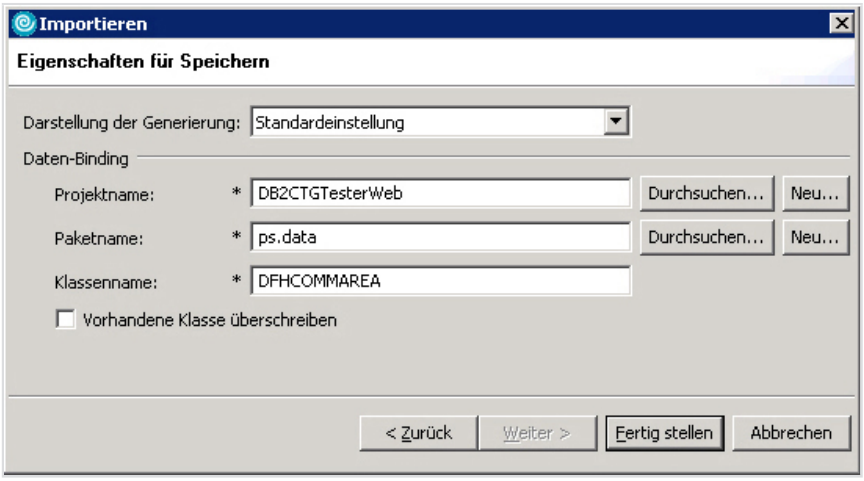


Abbildung 4.9: Bestimmung der Speichereigenschaften (Java Data Binding).

Anschließend wird die Java-Klasse unter den Namen **DFHCOMMAREA** im Paket **ps.data** des Web-Projektes gespeichert (*s. Abbildung 4.9*).

4.2.3 Erstellen und Konfigurieren einer Server-Instanz

Für nachfolgende Schritte wird ein Server benötigt, der zum Projekt hinzugefügt und konfiguriert werden soll. Auf dem *websphere*-Rechner ist nur der IBM WebSphere Application Servers V6.0 vorhanden, der als die integrierte Testumgebung in den *IRAD* installiert ist.

Nun wird mit der Konfiguration des Servers in Teilschritten angefangen.

1. Es muss zuerst ein eigenes Server-Profil erstellt werden. Dazu wählt man im Hauptmenü **Fenster > Benutzervorgaben**. Es öffnet sich das Fenster, in dem man die Kategorie **Server > WebSphere** auswählt (*s. Abbildung 4.10 auf der nächsten Seite*).

   Mittels **Profile erstellen**[4] gelangt man in einen Assistenten mit dem ein neues Profil erzeugt werden kann. Im ersten Schritt des Assistenten gibt man dem Profil einen eindeutigen Namen (in diesem Fall *AppSrv_PSELESS*). In nächsten Schritten müssen die voreingestellten Werte für den Knotenname (*websphereNode01*), den Hostname (hier: *websphere.informatik.informatik.uni-leipzig.de*) und den Portnummern (z.Bsp. Port für die Administrationskonsole) übernommen werden.

   Anschließend bestätigt man die Erstellung des Profiles mit **Fertig stellen**, der standartmäßig im Verzeichnis *C:\Programme\IBM\Rational\SDP\6.0\runtimes\base_v6\profiles* gespeichert wird.

2. Als nächstes wählt man in der *Serverkonfiguration*-Sicht mit dem Rechtsklick **Neu > Server**

[4] Den Assistenten kann auch direkt aufgerufen werden:
C:\Programme\IBM\Rational\SDP\6.0\runtimes\base_v6\bin\ProfileCreator\pctWindows.exe.

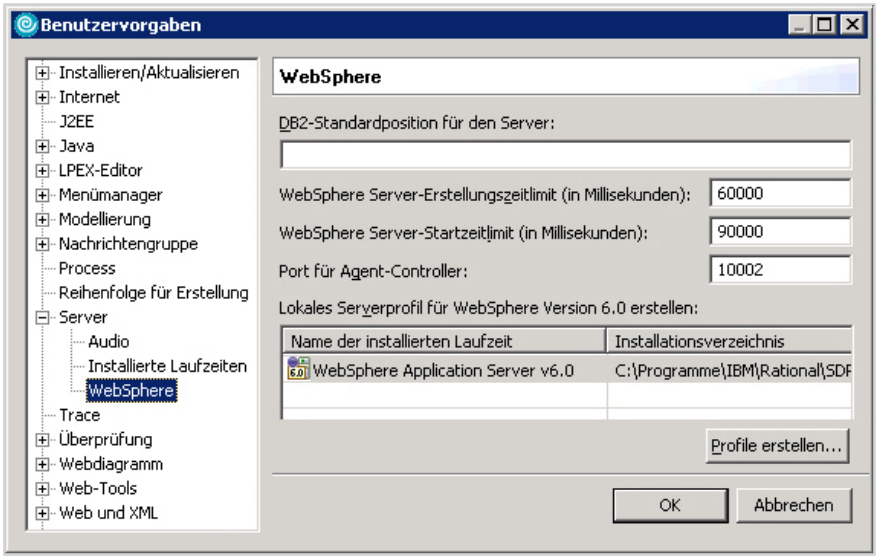


Abbildung 4.10: Erstellung des Server-Profiles.

aus. Daraufhin wird der Assistent für die Erstellung des neuen Servers aufgerufen. Als Servertyp wählt man hier **WebSphere v6.0 Server** und fortsetzt die Konfiguration mit **Weiter**.

3. Im nächsten Schritt wählt man das zuvor erstellte Profil aus. Mit dem Mausklick auf die Schaltfläche **Fertig stellen** bestätigt man die Erstellung eines Exemplars des Servers, der auf dem Port **9080** läuft und in der lokalen Testumgebung des *Rational Application Developer* ausgeführt wird.

Der neu definierter Server wird auf der Registerkarte **Servers** angezeigt und kann per Rechtsklick[5] gestartet werden.

4.2.4 J2C-JavaBean

An dieser Stelle wird die JavaBean erstellt, die mit einem unternehmensweiten Informationssystem über die J2EE Connector Architecture kommuniziert. In unserem Fall wird den Zugriff auf den CICS-Server unter der Verwendung vom ECI Ressourcenadapters realisiert.

Man ruft den Assistenten zur Erstellung der JavaBean aus. Dazu wählt man im Hauptmenü **Datei > Neu > J2C > J2C-JavaBean** aus.

Im neu geöffneten Fenster soll zunächst der ECI Ressourcenadapter ausgewählt werden (*s. Abbildung 4.11 auf der nächsten Seite*). Ausser diesem Adapter verfügt *IRAD* über die weiteren Ressourcenadapter, die der JCA-Spezifikation 1.0 oder 1.5 entsprechen und im Installationsverzeichnis vom *IRAD* im Verzeichnis namens *Resource adapters* vorhanden sind.

Aus den Kompatibilitätsgründen wählt man hier den ECI Ressourcenadapter der JCA-Spezifikation V1.0, welcher auf dem WebSphere Anwendungsserver ab der Version 5.0.2 ausgeführt werden kann.

[5] Server kann auch per Kommandozeile gestartet werden: *Profile_Verzeichnis\bin\startServer.bat server1 -profileName AppSrv_PSELESS.*

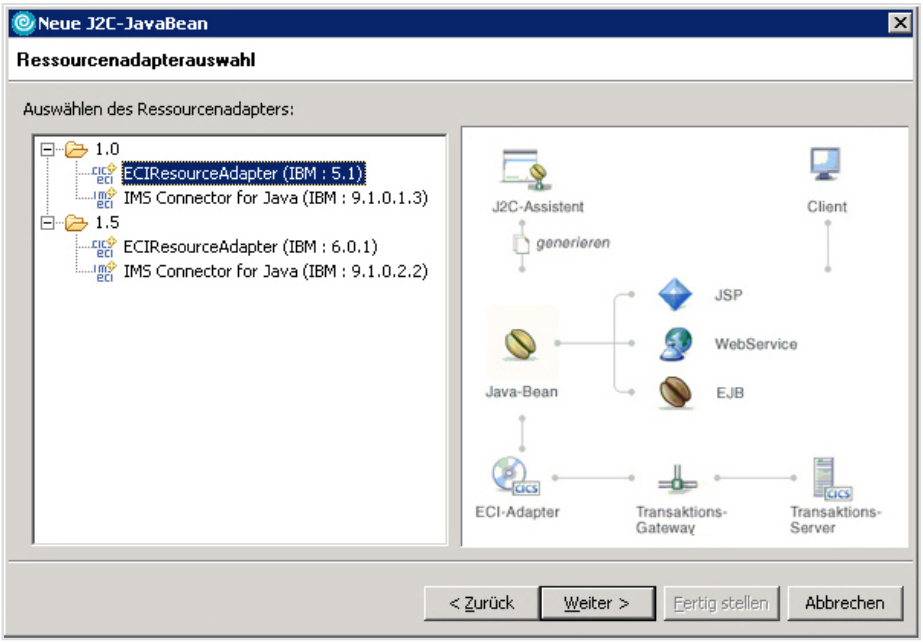


Abbildung 4.11: Auswahl des ECI Resourceadapters.

Für den Ressourcenadapter muss auf dem Server eine Connection-Factory (*Verbindungsfactory*) konfiguriert werden. Die Verbindungsfactory besteht aus einer Gruppe von Konfigurationswerten, die von dem Ressourcenadapter benötigt werden, um sich zu einem bestimmten EIS (in diesem Fall CICS) zu verbinden. Diese Werte werden vom Manager des J2C-Verbindungspools während der Laufzeit vom Anwendungsservers verwendet. Dabei greifen Anwendungskomponenten nicht direkt auf den Ressourcenadapter, sondern durch die Ressourcenreferenzen, die in Deployment-Deskriptoren definiert sind und auf die Verbindungsfactory verweisen.

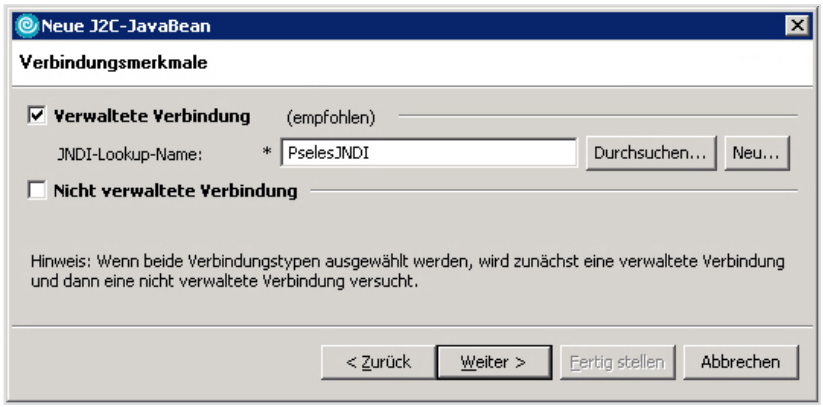


Abbildung 4.12: Auswahl des Verbindungstyps.

Dazu wählt man im nächsten Schritt für den Typ der Verbindung **Verwaltete Verbindung** aus (*s. Abbildung 4.12*). Für den JNDI-Name von Verbindungsfactory trägt man einen beliebigen Namen ein (in diesem Fall **PselesJNDI**) und klickt danach auf **Neu**.

Als nächstes wählt man den Server, wo der Ressourcenadapter implementiert werden soll. Hier

wählt man **WebSphere Anwendungsserver V6.0** und mit dem Klick auf **Weiter** geht man zum nächsten Schritt über, wo die verbindungsspezifische Merkmale für die Verbindungsfactory festzulegen sind (*s. Abbildung 4.13*).

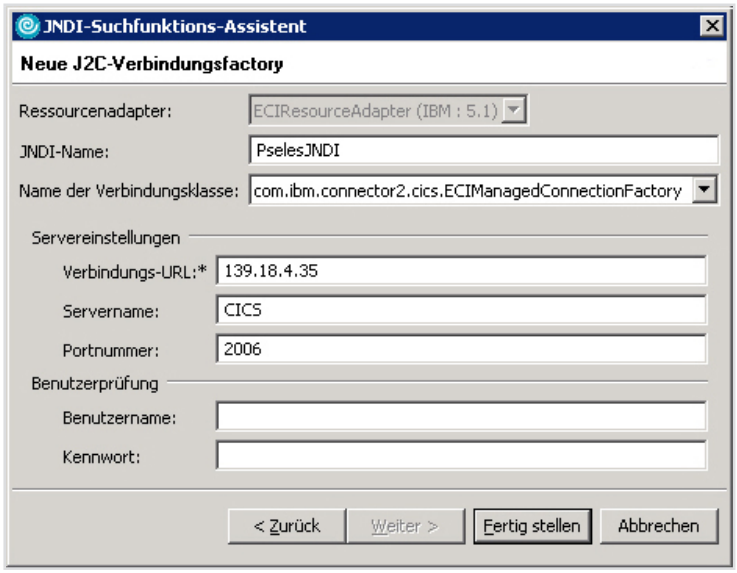


Abbildung 4.13: Festlegen von Verbindungseigenschaften.

Dabei entscheidend sind die Werte des Servernamens und der Verbindungs-URL. Hier muss der Name der CICS-Serverregion, in diesem Fall war das **CICS**, und die Adresse vom CICS Transaction Gateway (hier: **139.18.4.35**) eingegeben werden. Der voreingestellte Wert für die Verbindungsklasse muss beibehalten werden.

Nach dem Klick auf **Fertig stellen** kommt man wieder zum *J2C-JavaBean*-Assistenten zurück (*s. Abbildung 4.12 auf der vorherigen Seite*). Man klickt auf **Weiter** und geht zum nächten Schritt des Assistenten über (*s. Abbildung 4.14*).

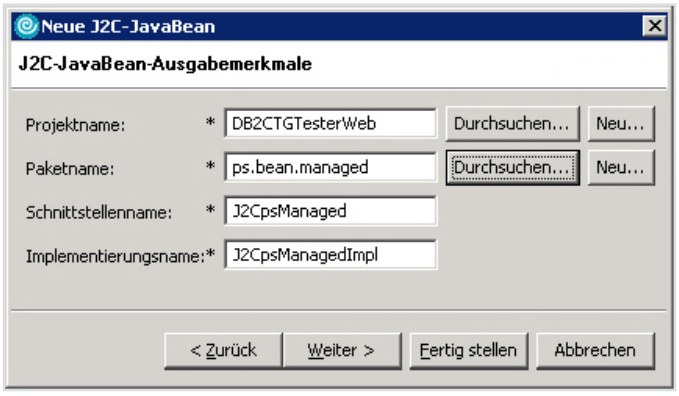


Abbildung 4.14: Bestimmung der Speichereigenschaften (J2C-JavaBean).

Nun speichert man die J2C-JavaBean, die aus einem Interface und einer Implementation-Klasse besteht, im Paket **ps.bean.managed** des Web-Projektes und schließt die Erstellung mit **Fertig stellen** ab.

Der im vorigen Abschnitt erstellte Server ist jetzt für die Kommunikation mit der CICS-Region über CICS ECI Ressourcenadapter konfiguriert. Es wurde der Ressourcenadapter zur Konfiguration des Servers hinzugefügt und die Verbindungsfactory erstellt (*s. Abbildung 4.15*).

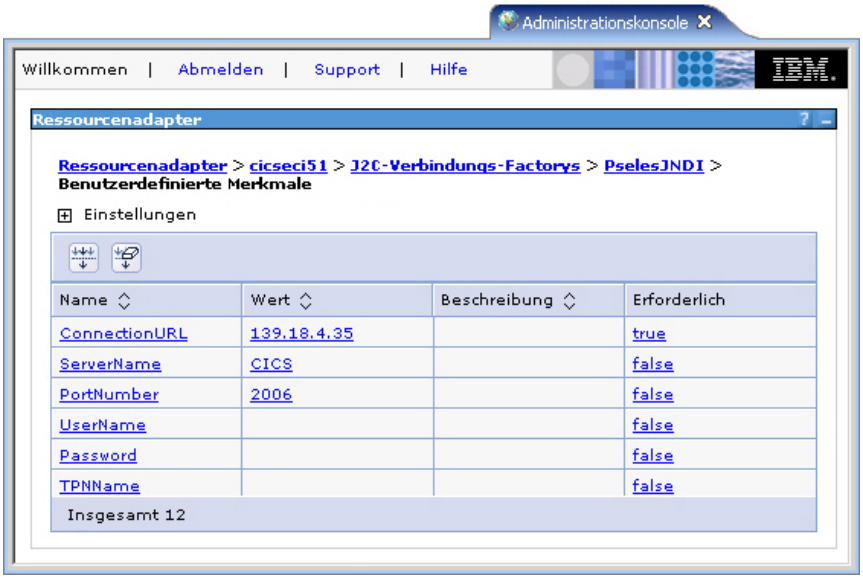


Abbildung 4.15: Administrationskonsole des lokalen Anwendungsservers.

Die erstellte J2C-JavaBean enthält momentan nur die Methoden, die unter Verwendung des javax.resource.cci-Interfaces über den Ressourcenadapter mit dem CICS kommunizieren. Die Interaktion zwischen der J2C-JavaBean und dem Ressourcenadapter geschieht dabei über den JNDI-Name der Verbindungsfactory.

```
1  /**
2   * @j2c.connectionFactory jndi-name="PselesJNDI"
3   * @generated
4   */
5  ...
6   protected void initializeBinding() throws ResourceException {
7     ConnectionFactory cf = null;
8    String jndiName = "PselesJNDI";
9     javax.naming.Context ctx = null;
10    try {
11      ctx = new javax.naming.InitialContext();
12      cf = (ConnectionFactory) ctx.lookup("java:comp/env/" + jndiName);
13 ...
```

Programm 4.2: Auszug aus der J2C-JavaBean Implementierungsklasse J2CpsManagedImpl.

In der J2C-JavaBean ist der JNDI-Name durch einen J2C doclet tag eingebunden (*s. Programm 4.2, Zeile 2*). Die Lokalisierung dieser Verbindungsfactory ist mit Hilfe einer JNDI-Lookup-Operation realisiert (*Zeilen 7-12*).

4.2.4.1 Hinzufügen der Methode zur J2C-JavaBean

Um die neue Methode zur JavaBean hinzuzufügen, klickt man die J2C Implementierungsklasse mit rechter Maustaste an und wählt aus dem Kontextmenü den Eintrag **Quelle > Methode zu J2C-JavaBean hinzufügen** (*Abbildung 4.16*).

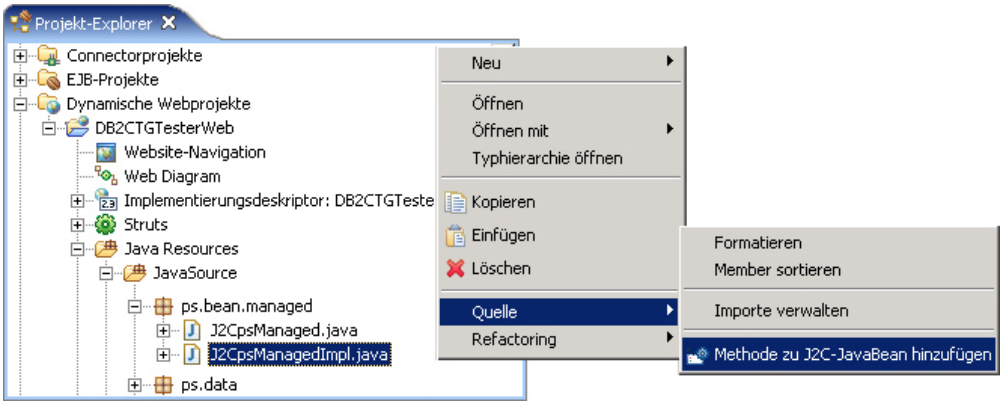


Abbildung 4.16: Hinzufügen einer neuen J2C-Methode vom Kontexmenü.

Im neu geöffneten Fenster trägt man einen beliebigen Namen für die Methode ein (In diesem Fall **callDB2PCICS**). Da die Commarea des CICS-Programms nicht nur für die Ausgabe sondern auch für die Eingabe von Daten dient, muss im nächsten Schritt sichergestellt werden, dass das Kontrollkästchen **Den Eingabetyp für die Ausgabe verwenden** aktiviert ist (*s. Abbildung 4.17*).

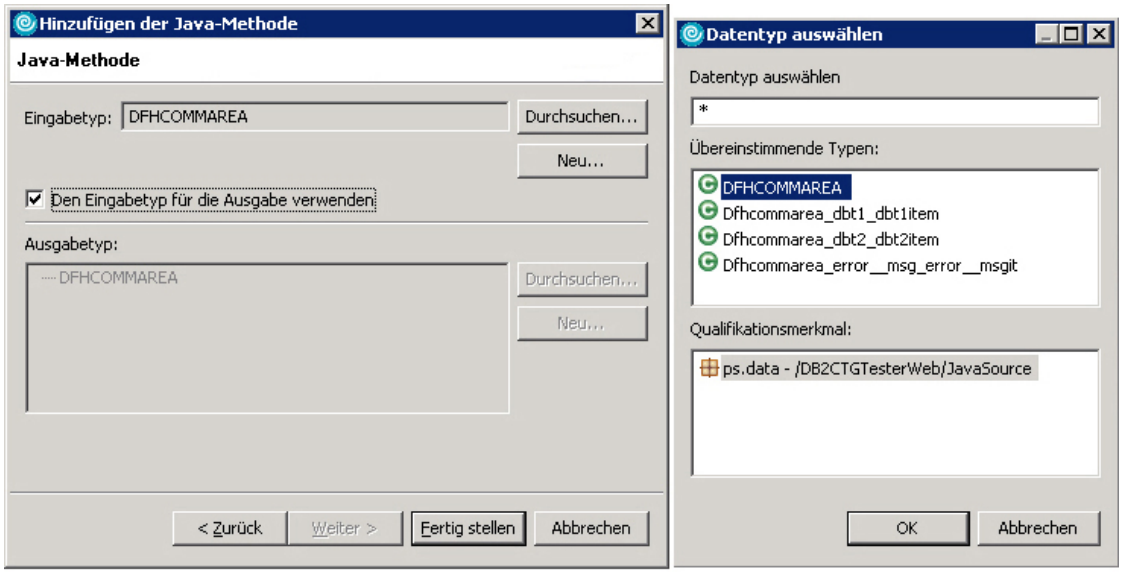


Abbildung 4.17: Hinzufügen einer J2C-Methode.

Danach klickt man auf **Durchsuchen** neben dem Eingabetyp-Feld, wie es die Abbildung 4.17 zeigt. In neu geöffneten Fenster wählt man die im Abschnitt 4.2.2 generierte *Java Data Binding* Klasse (DFHCOMMAREA).

Schließlich müssen im letzten Schritt einige *InteractionSpec*-Eigenschaften festgelegt werden. Das sind der Name des aufzurufenden Programms und die Länge der Commarea (*s. Abbildung 4.18*).

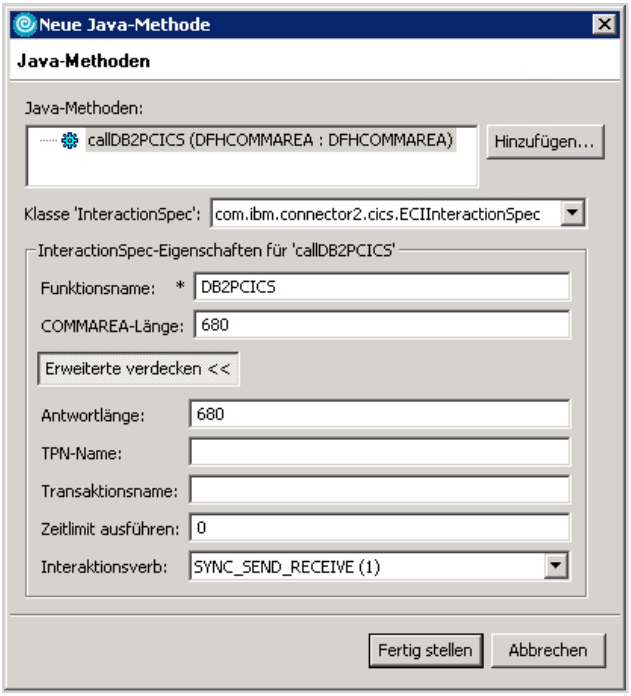


Abbildung 4.18: J2C-JavaBean Eigenschaften.

Anschließend klickt man auf die Schaltfläche **Fertig stellen**. Demzufolge wird die neue Methode zur Implementation-Klasse der J2C-JavaBean hinzugefügt. Von diesem Punkt können die erstellten Java-Klassen für den CICS-Zugriff verwendet werden.

Dabei geschieht der eigentlicher Aufruf des CICS-Programms mit Hilfe der **callDB2PCICS**-Methode, der als Argument in diesem Fall die gesamte Commarea übergeben wird. Für die Konvertierung der übertragenen Daten zwischen Java-Objekten und Cobol-Datenstrukturen wird die **DFHCOMMAREA**-Klasse verwendet.

4.3 Generieren des Implementierungscode für die J2C-JavaBean

IBM Rational Application Developer bietet die Möglichkeit aus der von einer J2C-JavaBean bereitgestellten Funktionen automatisch Java-Klassen zu erstellen, die als Wrapper-Klassen für die JavaBeans dienen. Dazu stellt *IRAD* drei Möglichkeiten, wie die J2C-JavaBeans bereitgestellt werden können: Webservice, Enterprise Java Bean (*EJB*) oder Java Server Page (*JSP*).

Man etnscheidet sich für die EJB. Wie gewohnt stellt IRAD für die Erstellung einer Session Bean einen Assisten zur Verfügung, der wie folgt aufgerufen werden kann.

Zunächst markiert man die J2C Implementierungsklasse im Web-Projekt mit rechter Maustaste und wählt aus dem Kontextmenü den Eintrag **Neu > Andere**. Im Fenster **Assistent auswählen** unter **J2C** wählt man **Webseite, Web-Service oder EJB aus J2C-JavaBean** und klickt auf **Weiter**.

Daraufhin wird der Assistent J2EE-Ressource von J2C-JavaBean gestartet, in dem man als *J2EE-Resourcentyp* EJB auswählt (*Abbildung 4.19*).

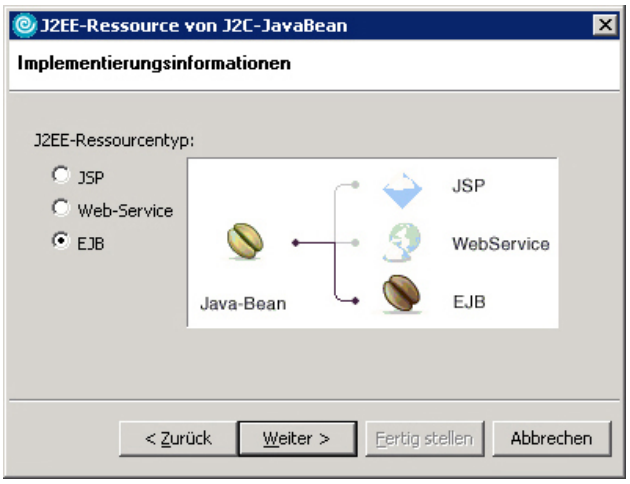


Abbildung 4.19: J2C-JavaBean Implementierungsinformationen.

Im nächsten Schritt sind einige Werte bereits vordefiniert und können unverändert übernommen werden. Es musste lediglich ein beliebiger Name für die Session-Bean eingetragen werden (hier: CTGDB2). Anschließend kann der Assistent mit Fertig stellen beendet werden.

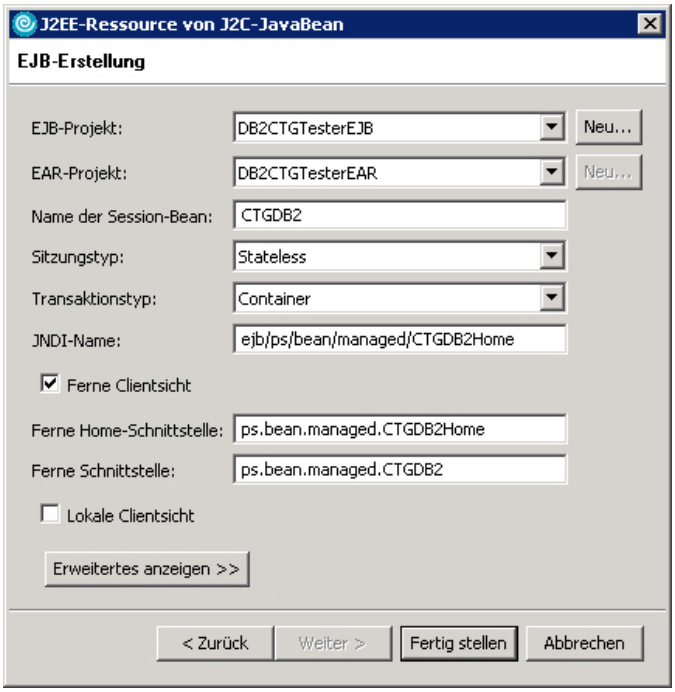


Abbildung 4.20: Assistent für die EJB-Erstellung.

Da die EJB auf Basis der J2C-JavaBean erstellt wurde, muss der J2C-JavaBean Code im EJB-Projekt vorhanden sein. Deswegen werden zwei Klassen der J2C-JavaBean aus dem Web-Projekt in das EJB-Projekt kopiert.

Das EJB-Projekt DB2CTGTesterEJB enthält die EJB CTGDB2 in Form einer Session Bean (*Abbildung 4.21*).

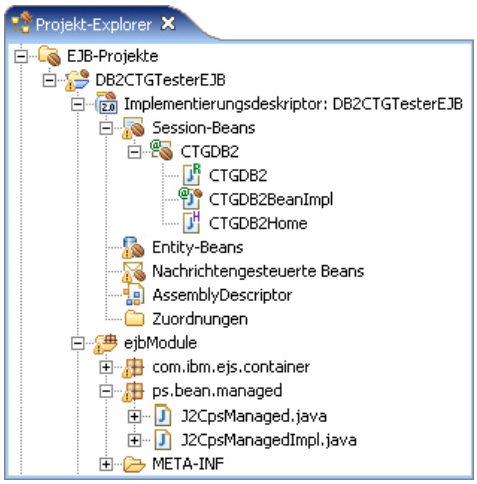


Abbildung 4.21: Enterprise Bean CTGDB2.

Diese EJB besteht aus einer Implementierungsklasse und zwei Schnittstellen, den so genannten *Home*- und *Remote*-Interfaces.

- Das Home-Interface (CTGDB2Home) enthält die Methoden zur Steuerung des Lebenszyklus einer EJB-Komponente. Über diese Schnittstelle kann das Erzeugen, Lokalisieren und Löschen des Beans realisiert werden.
- In dem Remote-Interface (CTGDB2) werden die Methoden definiert, die von einer Enterprise Bean nach außen angeboten werden. In diesem Fall ist es die callDB2PCICS Methode, welche identische Signatur enthält und die gleiche Funktionalität hat, wie die Methode der J2C-JavaBean.
- Session Bean (CTGDB2BeanImpl) implementiert die im Remote- und Home-Interfaces spezifizierte Methoden und erhält die J2C-Implementierungsklasse J2CpsManagedImpl als *Superklasse* zugeordnet.

In der nun erstellten J2C-JavaBean müssen einige Änderungen vorgenommen werden, damit der Name des aufzurufenden CICS-Programms, der Benutzername und das Kennwort dynamisch von der Klient-Anwendung geändert werden können. Um dies zu ermöglichen, muss die J2CpsManagedImpl-Klasse mit Hilfe *InteractionSpec*- und *ConnectionSpec*-Eigenschaften um functionName, userName und password erweitert werden.

Programm 4.3: Ausschnitt aus der J2CpsManagedImpl-Klasse.

```
1 package ps.bean.managed;
2 ...
3 public class J2CpsManagedImpl implements ps.bean.managed.J2CpsManaged {
4   /**
5    * @j2c.connectionSpec class="com.ibm.connector2.cics.ECIConnectionSpec"
6    * @j2c.connectionSpec-property name="password" argumentBinding="argpassword"
7    * @j2c.connectionSpec-property name="userName" argumentBinding="arguserName"
```

Fortsetzung auf der nächsten Seite ...

Fortsetzung des Programms 4.3

```
8    * @j2c.interactionSpec class="com.ibm.connector2.cics.ECIInteractionSpec"
9    * @j2c.interactionSpec-property name="functionName" argumentBinding="argfunctionName"
10   * @j2c.interactionSpec-property name="commareaLength" value="680"
11   * @j2c.interactionSpec-property name="replyLength" value="680"
12   * @ejb.interface-method view-type="remote"
13   * @generated
14   */
15  public ps.data.DFHCOMMAREA callDB2PCICS(ps.data.DFHCOMMAREA arg,
16     java.lang.String argfunctionName, java.lang.String arguserName,
17     java.lang.String argpassword)
18     throws java.rmi.RemoteException, javax.resource.ResourceException;
19
20   ConnectionSpec cs = getConnectionSpec();
21   if (cs == null) {
22     cs = new com.ibm.connector2.cics.ECIConnectionSpec();
23     ((com.ibm.connector2.cics.ECIConnectionSpec) cs).setPassword(argpassword);
24     ((com.ibm.connector2.cics.ECIConnectionSpec) cs).setUserName(arguserName);
25   }
26
27   InteractionSpec is = interactionSpec;
28   if (is == null) {
29     is = new com.ibm.connector2.cics.ECIInteractionSpec();
30     ((com.ibm.connector2.cics.ECIInteractionSpec) is).setFunctionName(argfunctionName);
31     ((com.ibm.connector2.cics.ECIInteractionSpec) is).setCommareaLength(680);
32     ((com.ibm.connector2.cics.ECIInteractionSpec) is).setReplyLength(680);
33   }
34
35   ps.data.DFHCOMMAREA output = new ps.data.DFHCOMMAREA();
36   invoke(cs, is, arg, output);
37   return output;
38  }
39 }
```

Der Benutzername und das Kennwort sind Bestandteile des *Connection*-Objektes und werden daher durch ein ConnectionSpec-Objekt definiert. Diese Eigenschaften werden durch die *J2C doclet tags* festgelegt (*Zeilen 5-7*). *Rational Application Developer* definiert daraus vollautomatisch ein ECIConnectionSpec-Objekt und die entsprechende *getter* und *setter* Methoden (*Zeilen 22-24*). Die gleiche Vorgehensweise benutz man für den Name des Programms.

Danach müssen die erstellten Parameter in der Methodesignatur aufgenommen (*Zeilen 16-17*) und die Anwendung erneunt publiziert werden, damit vorgenommene Änderungen in Kraft treten könnten. Demzufolge werden argfunctionName, arguserName und argpassword Parameter zum Remote-Interface der EJB (CTGDB2) und zur Intarface-Klasse (J2CpsManaged) hinzugefügt.

4.3.1 Bindung der EJB-Referenzen

Für die Verbindung mit dem CICS soll die generierte Stateless-Session-Bean mit der Verbindungsfactory verknüpft werden. Hierzu kommt eine Ressourcenreferenz zum Einsatz. Sie erlaubt der Anwendung, einen logischen Namen beim Suchen nach einer externen Ressource zu verwenden (in diesem Fall der JNDI-Name der Verbindungsfactory). Die eigentliche Referenzierung wird über den Deployment-Descriptor-Mechanismus der EJB-Komponente definiert:

- Als erstes ist in der *Projekt-Explorer*-Sicht DB2CTGTesterEJB > ejbModule > META-INF zu ex-

pandieren. Man doppelklickt auf die **ejb-jar.xml**-Datei, um sie im *EJB Deployment-Deskriptor* zu öffnen.

- Danach wählt man im Editor die *Verweise*-Registerkarte. Man markiert **CTGDB2**-Bean und klickt auf **Hinzufügen**. Im neu geöffneten Fenster wählt man für den Typ des Verweises **Ressourcenreferenz** und klickt auf **Weiter**.

- Im nächsten Schritt *"Ressourcenreferenz"* gibt man unter *Name* den Namen dieser Ressourcenreferenz ein. Als Ressourcentyp wird der Ressourcenreferenz **javax.resource.cci.ConnectionFactory** zugewiesen. Da in unserem Fall die Authentifizierungsdaten im Programmcode des EJB verwaltet sind, muss schließlich für Authentifizierung die Option **Application** ausgewält werden.

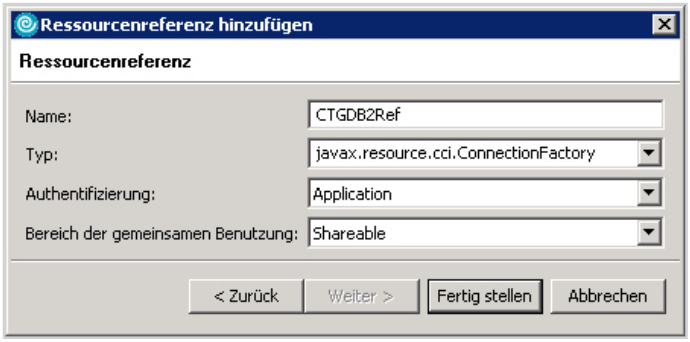


Abbildung 4.22: Erstellung der Ressourcenreferenz.

- Demzufolge wird die Ressourcenreferenz **CTGDB2Ref** zur *CTGDB2*-Bean hinzugefügt (*Abbildung 4.23*). Unter der *WebSphere-Bindings*-Sektion trägt man für den JNDI-Name **PselesJNDI** ein. Das ist der Name der Verbindungsfactory, die zur Serverkonfiguration im Abschnitt 4.2.4 hinzugefügt war. Anschließend schließt man den Editor und bestätigt mit *"OK"*, um Änderungen zu übernehmen.

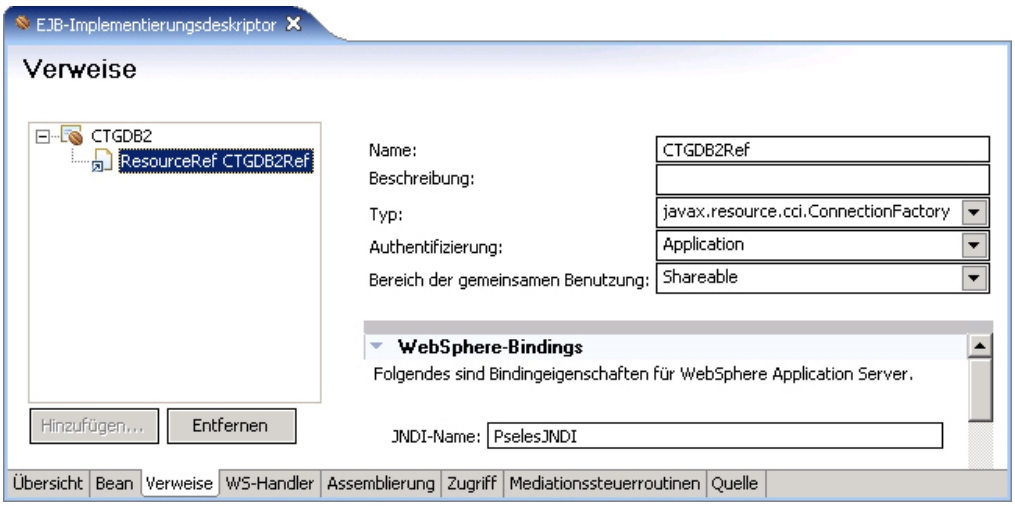


Abbildung 4.23: Ressourcenreferenz in dem EJB-Implementierungsdeskriptor.

Beim späteren Deployment[6] der Anwendung auf dem Anwendungsserver wird dann diese Ressourcenreferenz an den tatsächlichen Namen der Verbindungsfactory gebunden.

4.4 Test mit IBM Universeller Testclient

An dieser Stelle kann die aus dem vorigen Abschnitt erstellte Session-Bean mit Hilfe des **IBM Universeller Test Clients** auf dem bereits erstellten Anwendungsserver getestet werden. Dieses Tool läuft selbst auf dem Server und ermöglicht die Methoden und Objekte der Session Bean aufzurufen.

Die Funktionalität der erstellten EJB wird an einem Beispiel veranschaulicht, in dem man ein Datensatzt in die Datenbank hinzufügt. Hierbei geht man wie folgt vor, um den Test mittels dieses Werkzeuges durchzuführen:

1. In der *Projekt-Explorer* Sicht wählt man Session-Bean **CTGDB2** aus dem EJB-Projekt. Man klickt diese Bean mit der rechten Maustaste an und wählt aus dem Kontextmenü den Eintrag **Ausführen > Auf Server Ausführen**.

2. In dem neu geöffneten Fenster wählt man den WebSphere Anwendungsserver V6.0, der bereits vorhanden ist. Anschließend klickt man auf **Fertig stellen**. Dabei wird die Enterprise-Anwendung zur Server-Konfiguration hinzugefügt und der Server wird gestartet.

3. Nach dem Routineablauf des Serversaufbau wird der *IBM Universeller Testclient* in der Workbench aufgerufen (*s. Abbildung 4.24*). Nachfolgend ist in der *EJB-Beans*-Sektion die *CTGDB2* und dann die *CTGDB2Home* zu expandieren.

   Hier wählt man die **CTGDB2.create()**-Methode der Home-Schnittstelle aus. Danach klickt man in der *Parameter*-Sektion auf den Knopf **Aufrufen**, um eine Instanz der Session-Bean zu erstellen.

4. Nun klickt man auf den Knopf **Mit Objekt arbeiten**, woraufhin eine neue Bean **CTGDB2 1** in der *EJB-Beans*-Sektion erscheint. Jetzt expandiert man diese Bean und wählt die Methode

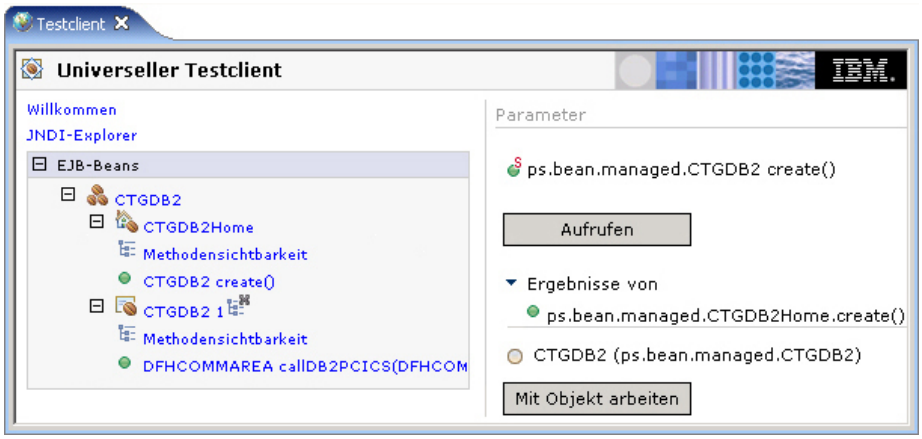


Abbildung 4.24: Erstellung einer Instanz der Session-Bean.

[6] Das Deployment einer Anwendung umfasst die Installation und Konfiguration der Anwendung.

callDB2PCICS aus (*Abbildung 4.24 auf der vorherigen Seite*).

5. Im *Parameter*-Fenster werden die Eingabeparameter der *callDB2PCICS*-Methode angezeigt (*s. Abbildung 4.25*). Dabei stellt das *DFHCOMMAREA*-Objekt die Commarea des CICS-Programms dar. Und die drei String-Parameter stehen für den Name des CICS-Programms, des Benutzernames und des Kennwortes.

   Es wird ein Datensatzt in die Datenbank hinzugefügt. Dazu gibt man für das Datenelement **getrequest** den Wert **getIns** ein. Für Vorname und Nachname können beliebige Daten eingetragen werden.

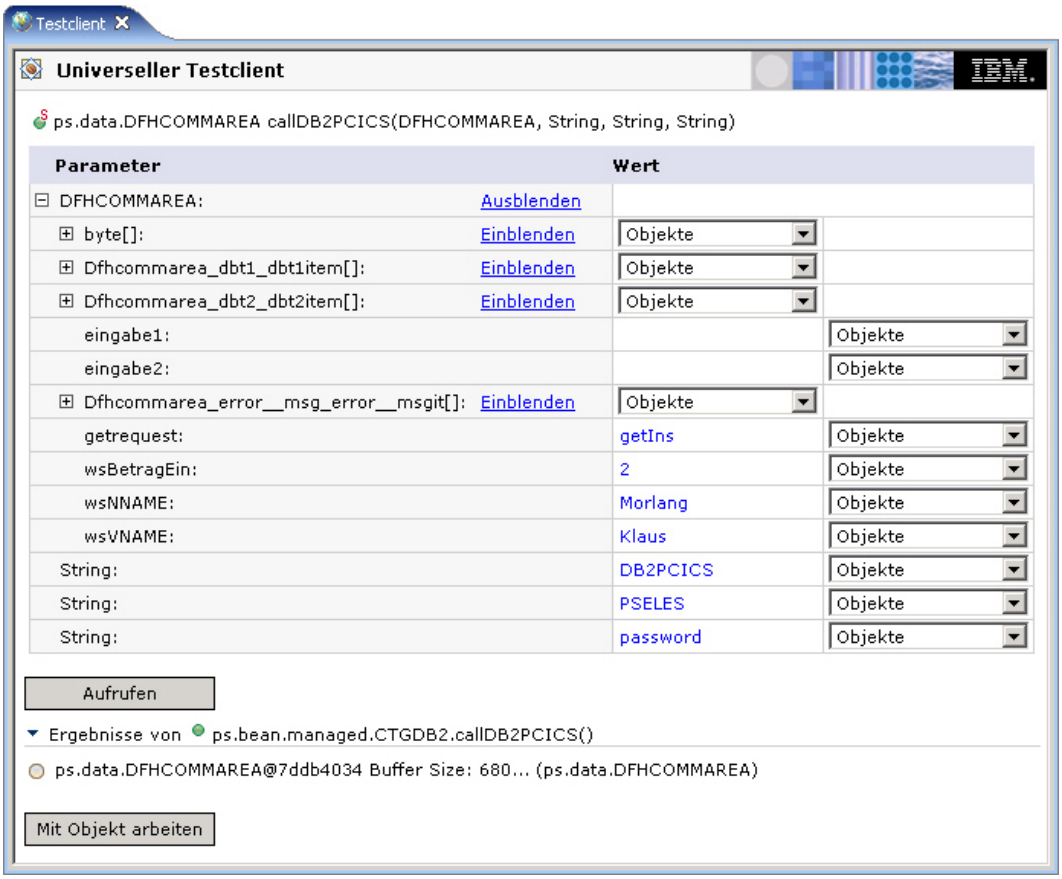


Abbildung 4.25: Aufruf der callDB2PCICS Methode.

6. Anschließlich klickt man auf den Knopf **Aufrufen**, um diese Methode auszuführen. Ist der Testlauf erfolgreich, wird das Ergebnis in einem *DFHCOMMAREA*-Objekt gespeichert.

Das Endergebnis der Ausführung kann unter **DB2 Admin** auf dem z/OS Rechner angesehen werden. Die Abbildung 4.26 auf der nächsten Seite zeigt die Ausgabe einer *SELECT*-Abfrage nach allen Datensätzen der Datenbank, die unter *DB2 Admin* ausgeführt wurde.

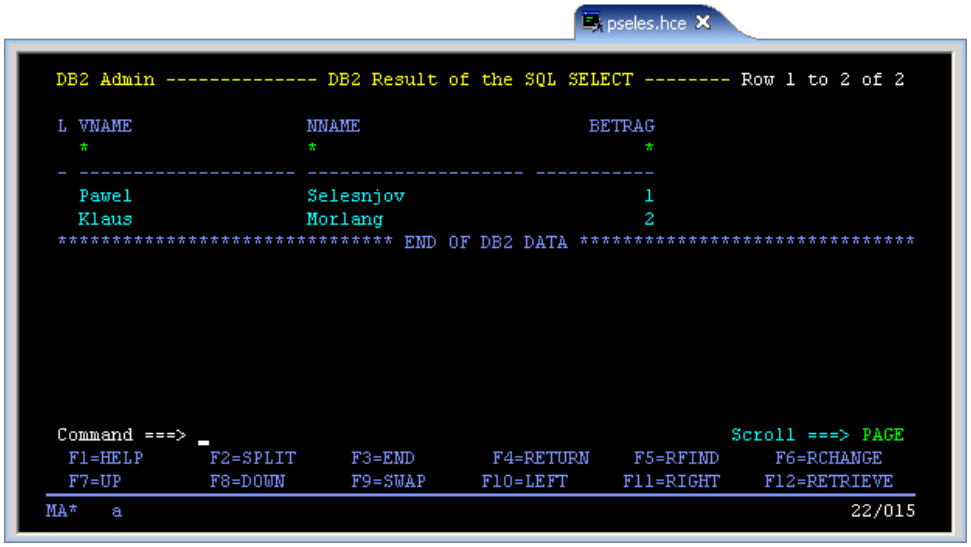


Abbildung 4.26: Ausgabe der Datenbank nach der SELECT-Abfrage.

An dieser Stelle, nachdem die J2C-JavaBean als Enterprise Bean erstellt und erfolgreich getestet wurde, kann zur Präsentationslogik der Anwendung übergegangen werden. In den folgenden Abschnitten werden die grundsätzlichen Schritte erklärt, um die Präsentationsschicht von Web-Anwendung mit Hilfe von Struts zu erstellen.

4.5 Implementierung einer Struts-basierten Webanwendung

Struts ist ein Open-Source-Framework, das im Jahre 2000 der Apache Software Foundation hinzugefügt wurde und dem Jakarta Projekt [Apaa] untersteht. Mittlerweile gibt es das Framework in der Version 1.2.9. Es bietet eine komfortable und einfache Schnittstelle zur Realisierung von Web-Applikationen gemäß der MVC Model 2-Architektur zur Verfügung.

Das MVS-Konzept fordert die Zuordnung der Anwendungskomponenten zu einer der drei Rollen Model (Datenhaltung), View (Präsentation) und Controller (Ablaufsteuerung).

- Eine Model-Komponente beinhaltet den Zustand einer Anwendung und Methoden zur Änderung dieses Zustandes. Das Model kennt keine der Komponenten (*View* und *Controller*), weshalb es unabhängig von beiden realisiert werden kann.

- Eine View-Komponente ist für die Darstellung der Daten eines Models zuständig. Dies geschieht über die *getter*-Methoden des Models. Die Eingaben, die der Benutzer auf der View ausführt, werden durch den Controller an die entsprechenden Methoden des Models weitergeleitet.

- Eine Controller-Komponente steuert die Interaktion zwischen dem Anwender und dem Model. Der Controller kennt dagegen die beiden Komponenten View und Model. Er überprüft die Benutzereingabe und ändert das Model über den Aufruf der jeweiligen *setter*-Methoden des Models.

Die Abbildung 4.27 zeigt die Kommunikation und den Datenaustausch zwischen den Komponenten einer MVC-Architektur.

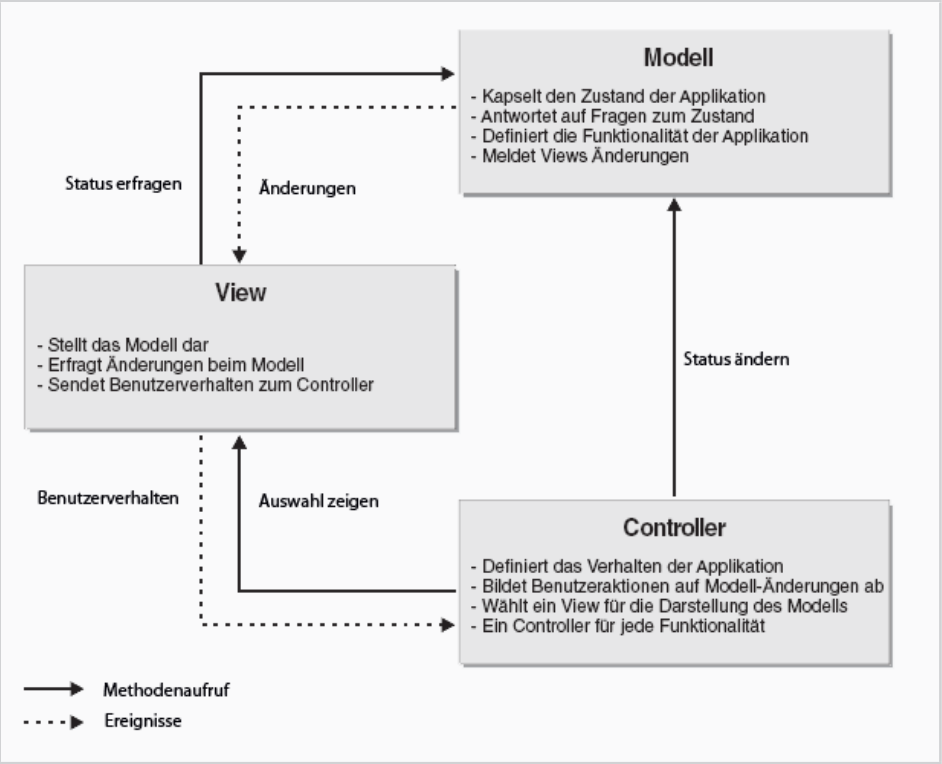


Abbildung 4.27: Das Model-View-Controller-Muster.

Die Rolle des Controller in *Struts* wird von einem Controller-Servlet, dem so genanntem *ActionServlet*, einem *RequestProcessor*, einer Konfigurationsdatei und einer Menge von *Action*-Klassen übernommen. Das ActionServlet ist hierbei der zentraler und wichtigster Bestandteil innerhalb des Struts-Frameworks.

Ein ActionServlet, welches innerhalb des Servlet Containers läuft, nimmt die Anfragen der Clients entgegen und leitet sie entweder direkt an eine JSP-Seite weiter, oder übergibt die Weiterverarbeitung der Anfrage an eine Action-Klasse. Die Parameter der Anfrage, die beispiellweise die Benutzereingaben aus einem HTML-Formular sein können, werden vom ActionServlet ausgewertet und in einer Java Bean, so genannte *Form Bean*, abgelegt. Diese Form-Bean wird als Methodenparameter an eine Action-Klasse übergeben. Innerhalb einer Action-Klasse werden die Funktionalitäten der Geschäftslogik bzw. des Modells aufgerufen.

Schließlich wird nach der Ausführung aller benötigten Operationen die Kontrolle von der Action-Klasse an das ActionServlet übergeben, das mit Hilfe des ActionMapping die entsprechende JSP oder weitere Action-Klasse auswählt. Dabei werden die View-Komponenten (*normalerweise JSP*) über die Form Beans mit geänderten Werten aus dem Modell gefüllt.

4.5.1 Modifikation des EJB-Codes

Die im Abschnitt 4.3 erstellte Session Bean benutzt die ganze COMMAREA als *input/output* Parameter zum Zugriff auf den CICS-Server. Die Daten der COMMAREA sind als *DFHCOMMAREA*-Objekte dargestellt und dienen zum Datenaustausch zwischen CICS Transaction Gateway und dem CICS-Programm.

Um die präzise Aufrufe an das CICS-Programm zu senden und die einzelnen Informationen aus dem zurückgelieferten Objekt zu extrahieren, soll die Implementierungsklasse **CTGDB2BeanImpl** der Session Bean modifiziert werden.

```
1   public class CTGDB2BeanImpl extends ps.bean.managed.J2CpsManagedImpl implements
2   javax.ejb.SessionBean {
3
4   private String getrequest;
5   private ps.data.Dfhcommarea_dbt2_dbt2item[] dbt2item;
6   private ps.data.Dfhcommarea_error__msg_error__msgit[] error__msgit;
7   private ps.data.DFHCOMMAREA commarea = new ps.data.DFHCOMMAREA();
8   private String functionName;
9   private String userid;
10  private String password;
11
12  public CTGDB2BeanImpl() throws javax.resource.ResourceException {
13    super();
14  }
15
16  public void ejbCreate() {
17  }
18
19  /**
20   * @param commarea The commarea to set.
21   */
22  public void setCommarea(ps.data.DFHCOMMAREA commarea) {
23    this.commarea = commarea;
24  }
25
26  /**
27   * @return Returns the commarea.
28   */
29  public ps.data.DFHCOMMAREA getCommarea() {
30    return commarea;
31  }
```

Programm 4.4: Auszug aus der Implementierungsklasse der EJB: **CTGDB2BeanImpl**

Es muss eine Abfrage an das CICS-Programm gesendet und die abgefragten Informationen (Daten der Datenbank, Fehlermeldungen) nach dem CICS-Aufruf dargestellt werden. Hierzu sollen im Programm 4.4 die Variablen der DFHCOMMAREA deklariert werden (*Zeilen 4-10*).

Anschließend generiert man für jede diese Variable die getter- und setter-Methoden, indem man den Eintrag **Quelle > Getter und Setter generieren** aus dem Kontextmenü auswählt.

4.5.2 Konfiguration des Web-Projektes

Im *Rational Application Developer* beginnt die Struts-Unterstützung mit der Erstellung des Web-Projektes. Beim Stuts-Projekt handelt es sich um einen Spezialfall des Web-Projektes, welches enthält:

- Die Konfigurationdatei **struts-config.xml**, die normalerweise sich im Verzeichnis *"Web Content/WEB-INF"* befindet.
- Die Struts-Laufzeitbibliothek (**struts.jar**), aus dem Verzeichnis *"WEB-INF/lib"*.
- Die *Struts Resource Bundles* steht für die lokalisierte Textausgabe und die Fehlerdarstellung. Sie bietet die Möglichkeit die Ressourcen verschiedensprachig zur Verfügung zu stellen. Dabei handelt es sich um einen Satz von Dateien (*ApplicationResources.properties*), in denen zu jeweils einem Schlüsselwert der Text in der jeweiligen Sprache hinterlegt ist.

Die Struts-Unterstützung kann über die Projekteigenschaften aktiviert werden (*s. Abbildung 4.28*). Dazu wählt man in der *Projekt-Explorer*-Sicht den Knoten **Dynamische Webprojekte / DB2CTGTesterWeb** aus und klickt mit der rechten Maustaste darauf. Danach im Kontextmenü wählt man den Punkt **Eigenschaften**.

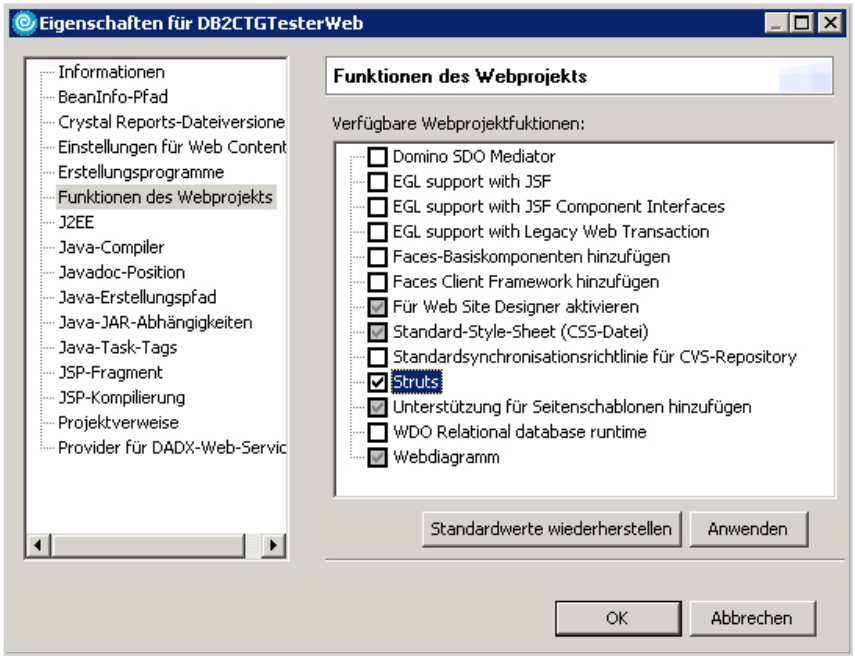


Abbildung 4.28: Funktionen des Webprojektes.

Nach dem Klick auf **Anwenden** können die Einstellungen für die Struts festgelegt werden, wie die Abbildung 4.29 auf der nächsten Seite zeigt. Es werden folgende Werte ausgewählt oder eingegeben:

- Das Kontrollkästchen: **Standardeinstellungen überschreiben**
- Struts-Version: **1.1**
- Standardpräfix für Java-Paket: **ps.web**

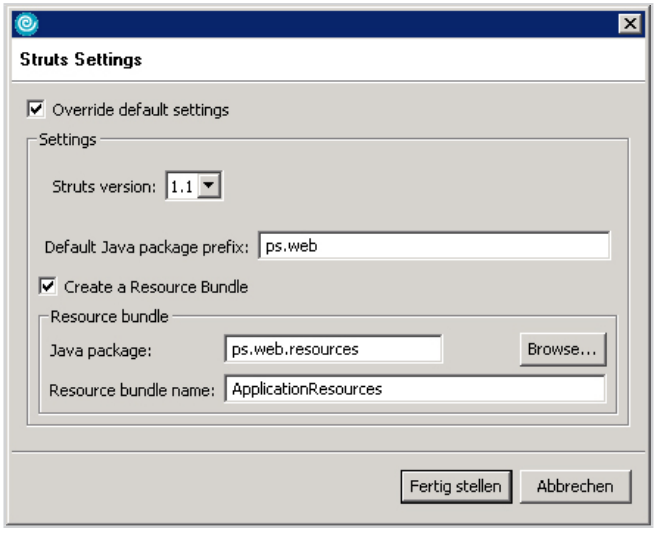


Abbildung 4.29: Web-projekt: die Struts-Einstellungen.

Danach ist mit **Fertig stellen** die Aktualisierung des Projektes zu bestätigen. Daraufhin werden Tag-Libraries, die Konfigurationsdatei *struts-config.xml* sowie Deployment-Descriptor der Web-Anwendung *web.xml* erstellt und registriert.

Mit Hilfe des Deployment-Descriptors werden verschiedene Eigenschaften des Action Servlets dem Servlet-Container eines Web Application Servers mitgeteilt. Damit das Action Servlet überhaupt verwendet werden kann, muss es mit Hilfe des *<servlet>*-Elementes im Deployment-Descriptor eingetragen werden (*s. Programm 4.5*). Durch diese Definition wird eine Instanz des Controller-Servlets erzeugt und mit den Namen **action** verknüpft.

Außerdem enthält das Action Servlet eine Vielzahl von Initialisierungsparameter (*<init-param>*). Einer der wichtigsten Parameter hierbei ist **config**. Mit diesem Parameter wird dem Action Servlet die zentrale Struts-Konfigurationsdatei (*/WEB-INF/struts-config.xml*) bekannt gegeben.

Programm 4.5: Auszug aus der Konfigurationsdatei **web.xml**.

```
<servlet>
   <servlet-name>action</servlet-name>
   <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
   <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
   </init-param>
   ...
</servlet>
...
<servlet-mapping>
   <servlet-name>action</servlet-name>
   <url-pattern>*.do</url-pattern>
</servlet-mapping>
...
<taglib>
   <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
   <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
```

Fortsetzung auf der nächsten Seite ...

Fortsetzung des Programms 4.5

```
19 </taglib>
20 <taglib>
21    <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
22    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
23 </taglib>
24 <taglib>
25    <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
26    <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
27 </taglib>
```

Für jeden <*servlet*>-Eintrag muss wenigstens ein <*servlet-mapping*>-Eintrag existieren. Das *Mapping* beschreibt, unter welchem URL-Muster welches Servlet erreichbar ist (*Zeilen 11-14*). So beispielsweise, wenn der Servlet Container eine Anfrage mit der Endung ***.do** [7] empfängt, wird diese an das Struts Action Servlet und somit auf die entsprechende Struts-Action weitergeleitet.

Zum Beispiel verweist die Anfrage *http://www.mycompany.com/myapplication/myaction.do* auf die Struts-Action **/myaction**. Die Verknüpfung zwischen der relativen URL **myaction** und einer konkreten Action-Klasse, die für die Bearbeitung der Anfrage zuständig ist, wird in der Struts-Konfigurationsdatei festgelegt.

4.5.3 Implementierung der Komponenten einer Struts-Anwendung

Nach der Konfiguration des Web-Projektes beginnt man mit der Erstellung der Struts-Komponenten. Dazu stellt IRAD einen **Web-Diagramm** Editor zur Verfügung.

Die Komponenten, aus denen die Web-Anwendung künftig bestehen wird, seien es JSPs, Action-Forms oder Actions, werden Schritt für Schritt dem Diagramm hinzugefügt. Die gesamte Struktur der Web-Anwendung ist in der Abbildung 4.30 auf der nächsten Seite dargestellt. Abhängigkeiten zwischen den einzelnen Komponenten werden durch die *Verbindungen* abgebildet. Die Verbindungen stellen dabei den Datenfluss zwischen unterschiedlichen Komponenten dar.

Die Implementierung von Elementen der Web-Anwendung wird auf einem Beispiel dargestellt, das in der Abbildung 4.30 in Schwarz hervorgehoben ist. Das Beispiel erfasst nur die Ausgabe der Datensätzen einer Datenbank, dessen Ablauf hier kurz erläutert wird.

Zuerst soll sich der Benutzer authentifizieren. Die Eingabe der Login-Informationen wie Programmname, Benutzername und Kennwort erfolgt dabei mit einem HTML-Formular über die JSP (**login.jsp**). Nach Abschicken des Formulars werden die Daten an das Action-Servlet gesendet. Das Action-Servlet enthält dann entweder eine ActionForm-Bean (**formBeanlogon**) oder es erzeugt eine solche und füllt sie über die *setter*-Methoden mit den entsprechenden Formulardaten.

Danach wird die aktualisierte ActionForm-Klasse vom Action-Servlet als Übergabeparameter an die Methode **execute()** der Action-Klasse (**/clogon**) weitergegeben. Die *execute*-Methode holt die Daten aus der ActionForm mittels *getter*-Methoden und führt die jeweiligen Aktionen in der Geschäftslogik aus. Dabei wird die Methode der EJB **CTGDB2BeanImpl** aufgerufen.

Das Ergebnis der Ausführung der **callDB2PCICS**-Methode wird in der ActionForm-Bean gespeichert.

[7] *.do* wird im Struts-Framework standardmäßig an URLs angehängt.

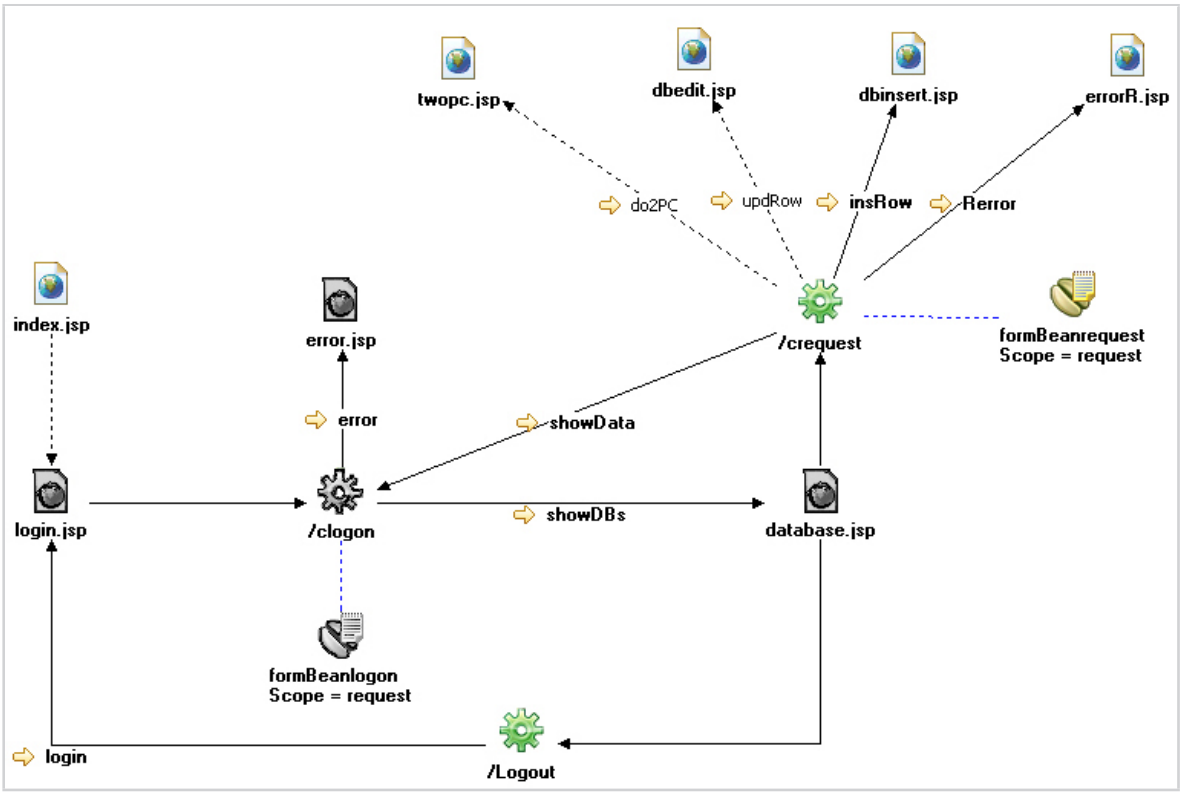


Abbildung 4.30: Komponenten der Struts-Anwendung im Web-Diagram Editor.

Die Kontrolle wird zurück an das Action-Servlet übergeben, das je nach Rückgabewert der Action-Klasse die nächste Action ausführt oder die JSP (database.jsp) ausgibt. In den folgenden Abschnitten werden die genannte Struts-Elemente und ihr Zusammenspiel schrittweise vorgestellt.

4.5.3.1 ActionForm Bean

Die ActionForm-Beans in Struts sind von der abstrakten Klasse org.apache.struts.action.ActionForm abgeleitete Java-Bean Klassen, die zur Verwaltung und Validierung der Elementen eines HTML-Formulars dienen. Das Struts Framework nimmt generell an, daß zu jedem Formular in einer Webanwendung auch die entsprechende ActionForm-Bean existieren muß. Dazu enthält eine Form-Bean für jedes Formularfeld ein gleichnamigen Attribut (*Property*) mit *setter*-und *getter*-Methoden.

Laut der Beschreibung der *Struts User-Guides* [Apab] werden die ActionForm-Beans als Teil der Controller-Komponente betrachtet, weil sie meistens die Daten einer Modell-Komponente verwalten.

Zur Implementierung einer ActionForm-Bean in einer Struts-Anwendung wird ein Assistent verwendet, der per Doppelklick auf das entsprechende Bean-Symbol im Web-Diagramm gestartet wird. Im neu geöffneten Fenster des Assistenten klickt man auf **Weiter** bis die Seite *"Erstellen neuer Felder für Ihre ActionForm-Klasse"* auftaucht (*Abbildung 4.31*).
Hier trägt man die Variablen ein, die für die Verwaltung von Daten des Anmeldeformulares

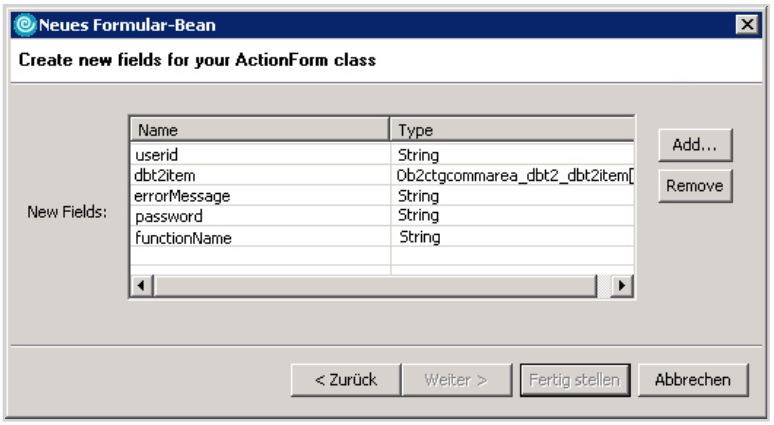


Abbildung 4.31: Assistent zur Erstellung der ActionForm-Bean.

(*login.jsp*) und der Antwortseite (*database.jsp*) benötigt werden. Anschließend wird die Erstellung der ActionForm-Klasse mit **Fertig stellen** bestätigt. Die ActionForm-Bean wird in *struts-config.xml* angemeldet und mit mindestens einem Element aus dem *Action Mapping* verknüpft. So verweist die Action im Programm 4.6 auf die erstellte *FormBeanlogon*-Klasse, die zuvor unter dem symbolischen Namen *formBeanlogon* bekannt gemacht wurde.

```
<form-beans>
    <form-bean name="formBeanlogon" type="ps.web.forms.FormBeanlogon"></form-bean>
</form-beans>

<action-mappings>
      <action path="/clogon"
                  type="ps.web.actions.ClogonAction"
                  name="formBeanlogon"
                  scope="session"
                  validate="true"
                  input="/jsp/login.jsp">
            <forward name="showDBs" path="/jsp/database.jsp"/>
            <forward name="error" path="/jsp/error.jsp"/>
      </action>
</action-mappings>
```

Programm 4.6: Ausschnitt aus der Struts-Konfigurationsdatei: Form-Bean Definition.

Außer den Attributen für die beiden JSPs sind in der neu erstellten *FormBeanlogon*-Klasse standardmäßig zwei Methoden zum Validieren und zum Zurücksetzen von Formulardaten implementiert, die bei jedem Request von dem ActionServlet automatisch aufgerufen werden.

Mit der Validierungs-Methode **validate()** werden die Benutzereingaben vor der Übergabe an die Action-Klasse auf ihre Gültigkeit geprüft. Kommt es bei der Validierung zu Fehlern, dann werden diese in einem *ActionErrors*-Objekt an das Eingabeformular zurückgeschickt und mit Hilfe des *<html:errors>*-Tag angezeigt. So im Programm 4.7 vorgestellte *validate*-Methode überprüft, ob ein Wert für das Attribut **functionName** gesetzt wurde.

```
1  public ActionErrors validate(ActionMapping mapping,HttpServletRequest request) {
2    ActionErrors errors = new ActionErrors();
3    if(mapping.getPath().equals("/clogon") && request.getMethod().equals("POST")){
4      if ((functionName == null) || (functionName.length() < 1))
5        errors.add("functionName", new org.apache.struts.action.ActionError("error.functionname
             "));
6    }
7    return errors;
8  }
```

Programm 4.7: Die *validate*-Methode der ActionForm-Klasse FormBeanlogin.

Für die automatische Validierung durch die Form-Klasse muss zusätzlich die Konfiguration in der *struts-config.xml*-Datei angepasst werden. Es muss der validate-Parameter für die entsprechende Action (*in diesem Fall ClogonAction-Klasse*) auf true gesetzt werden. Falls Validierung fehlschlägt, wird zur in input-Parameter definierte Eingabeseite zurückverwiesen (*siehe Programm 4.6 auf der vorherigen Seite, Zeilen 10-11*).

4.5.3.2 Erstellung der View-Komponenten

Als View-Komponenten werden in einer Web-Applikation, so wie auch beim Struts Framework, meist JSPs (*Java Server Pages*) verwendet.

Es wird mit der Erstellung des Login-Formulars angefangen. Wie gewohnt bietet *IRAD* dazu einen Assistent, der per Doppelklick auf das login.jsp Symbol im Web-Diagramm gestartet wird (*Abbildung 4.32*).

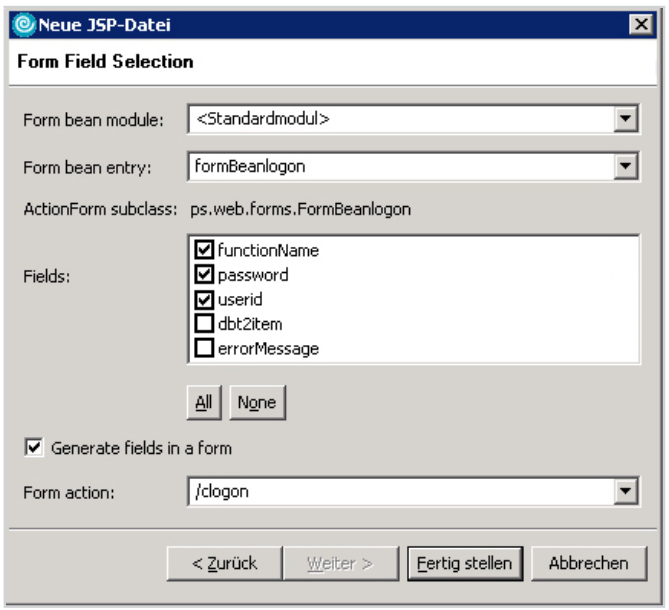


Abbildung 4.32: Assistent zur Erstellung der JSP-Datei.

Als ActionForm-Bean wählt man hier (FormBeanlogon), die bereits implementiert und in der Konfigurationsdatei registriert ist.

Danach markiert man die *Properties* der ausgewählten ActionForm (**functionName**, **password** und **userid**), die zur Definition der Eingabefelder des Formulars verwendet werden. Als nächstes wählt man die Action-Klasse (**/clogon**), welche die Benutzereingaben aus dem Formular empfangen wird. Anschließend kann der Erstellungsvorgang mit **Fertig stellen** abgeschlossen werden.

Das Programm 4.8 zeigt den Quelltext der gerade erstellten JSP Datei. Alle HTML-Tags sind mit Struts-eigenen Tags abgebildet, die ermöglichen in der JSP so wenig Java-Code (*scriptlets*) wie möglich zu verwenden. Dazu sind in ersten zwei Zeilen die Struts-Tag-Libraries (**html** und **bean**) deklariert.

```
1  <@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" >
2  <@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" >
3  <html:html>
4    <head><title><bean:message key="msg.prompt.login.title"/></title></head>
5    <body>
6    <bean:message key="msg.prompt.login.heading"/>
7    <html:form action="/clogon">
8      <table>
9        <tr>
10         <td><bean:message key="msg.prompt.login.functionName"/></td>
11         <td>
12           <html:text property="functionName"></html:text>
13         </td>
14       </tr>
15       <tr>
16         <td><bean:message key="msg.prompt.login.userid"/></td>
17         <td>
18           <html:text property="userid"></html:text>
19         </td>
20       </tr>
21       <tr>
22         <td><bean:message key="msg.prompt.login.password"/></td>
23         <td>
24           <html:password property="password" redisplay="false"></html:password>
25         </td>
26       </tr>
27     </table>
28   <html:submit property="submit"></html:submit>
29   <html:reset></html:reset>
30   </html:form>
31   <center><html:errors/></center>
32   </body>
33 </html:html>
```

Programm 4.8: Quelltext des Login-Formulars **login.jsp**.

Innerhalb des **<html:form>**-Tags sind die Formularfelder mit der entsprechenden Eigenschaften aus der ActionForm-Bean verknüpft. Das Textfeld **<html:text property="functionName"></html:text>** in der Zeile 12 hat zum Beispiel das property-Attribut **functionName**, das dem gleichnamigen Attribut aus der Klasse **FormBeanlogon** entspricht. Beim Absenden des Formulars werden die Daten aus diesem Feld mit Hilfe der Methode **setFunctionName()** in der ActionForm-Bean gespeichert.

Zur Internationalisierung (*Mehrsprachigkeit*) der Anwendungen wird in Struts ein so genanntes *message*-Tag aus der Bibliothek **struts-bean.tld** verwendet. In der Zeile 10 wird beispielsweise die Beschriftung des Eingabefeldes auf dem Formular ausgegeben. Das *bean*-Tag verweist hier auf den

Schlüssel **msg.prompt.login.functionName** in einer Properties-Datei[8], wo der gewünschte Text für die Beschriftung hintergelegt ist.

Auf der Ergebnisseite (*database.jsp*) soll der Inhalt der Datenbank und mögliche Fehlermeldungen angezeigt werden. Die Erstellung der *database.jsp* Seite wird wie zuvor durch einen Assistent durchgeführt, in dem man die **dbt2item** und **errorMessage** Attribute der FormBeanlogon-Klasse markiert (*siehe Abbildung 4.32 auf Seite 60*). Auf der JSP-Ausgabe werden dann die Werte diesen Attributen mittels des <bean:write>-Tags dargestellt.

4.5.3.3 Erstellung der Action-Klasse

Die Action-Klasse in Struts dient als *Wrapper*-Klasse für die *Business Logic* der Anwendung, d.h. sie ruft die Klasse der eigentlichen Geschaftslogik auf. Die Action-Klasse kontrolliert somit nur die Dialogsteuerung und die Fehlerbehandlung der Anwendung, soll aber keine Geschäftslogik beinhalten.

Um die Action-Klasse zu erstellen wird erneut der Assistent verwendet, der per Doppelklick auf das Symbol **/clogon** im Web-Diagramm gestartet wird (*s. Abbildung 4.33*).

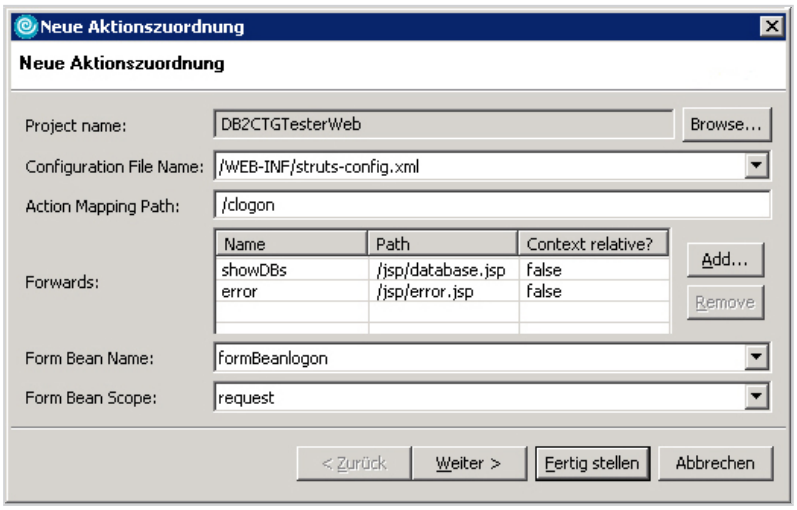


Abbildung 4.33: Action-Mapping-Assistent.

Der Assistent stellt eine vordefinierte Liste von Eigenschaften bereit. So steht *clogon* für den auf den Kontext der Anwendung bezogene relative Pfad für die Action, der mit der URL eines eingehenden *Requests* verglichen wird. Im *Forwards*-Fenster sind die so genannte *ActionForwards* platziert. Dabei handelt es sich um eine Weiterleitung des Benutzers auf eine JSP-Seite oder eine Action. Es müsste nur noch die Formular-Bean[9] **formBeanlogon** ausgewählt werden, damit die Action auf die Formularinhalte zugreifen kann.

Im nächsten Schritt des Assistenten legt man der Name für die Action-Klasse fest, wie in der Abbildung 4.34 auf der nächsten Seite zu sehen ist. Anschließend klickt man auf **Fertig stellen**, um

[8] kann beliebig benannt werden z.Bsp. *ApplicationResources.properties*
[9] Ein anderer Name für die ActionForm-Bean.

die *Action* vom Assistenten erstellen zu lassen.

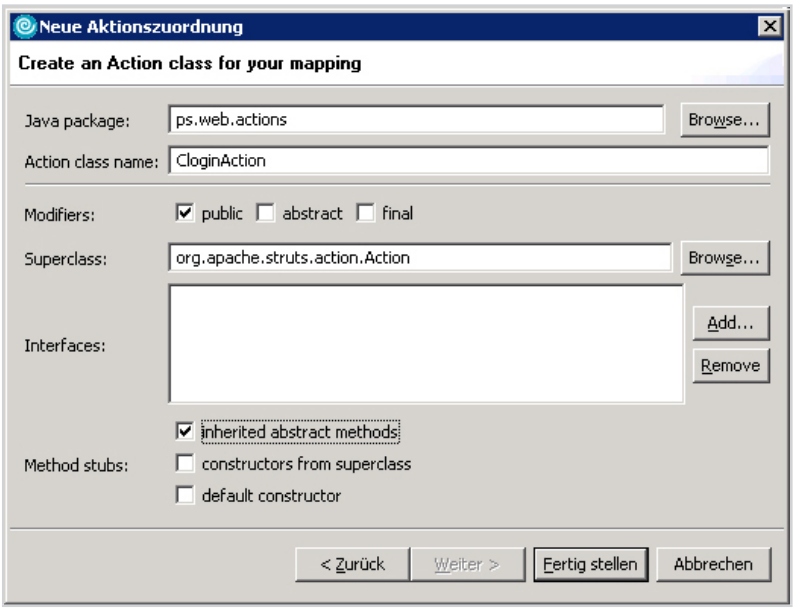


Abbildung 4.34: Erstellung der Action-Klasse.

Es werden die Parametern der Action-Klasse in die Struts-Konfiguration Datei geschrieben und die ClogonAction-Klasse erstellt, deren Quelltext in der folgenden Abbildung dargestellt ist.

Programm 4.9: Ausschnitt aus der Action-Klasse ClogonAction.

```
1  package ps.web.actions;
2  ...
3  public class ClogonAction extends Action {
4     public ActionForward execute(
5        ActionMapping mapping, ActionForm form,
6        HttpServletRequest request, HttpServletResponse response)
7        throws Exception {
8
9        ActionErrors errors = new ActionErrors();
10       ActionForward forward = new ActionForward(); // return value
11       FormBeanlogin formBeanlogin = (FormBeanlogin) form;
12
13       try {
14          // do something here
15          ps.data.DFHCOMMAREA commarea = new ps.data.DFHCOMMAREA();
16          ps.bean.managed.CTGDB2BeanImpl sbean = new ps.bean.managed.CTGDB2BeanImpl();
17
18          sbean.setCommarea(commarea);
19          sbean.setGetrequest("getDBs");
20
21          sbean.setCommarea(sbean.callDB2PCICS(commarea,
22                                    formBeanlogon.getFunctionName(),
23                                    formBeanlogon.getUserid(),
24                                    formBeanlogon.getPassword()));
25
26          formBeanlogon.setDbt2item(sbean.getCommarea().getDbt2item());
```

Fortsetzung auf der nächsten Seite ...

Fortsetzung des Programms 4.9

```
27          catch (Exception e) {
28             // Report the error using the appropriate name and ID.
29             errors.add("clogin_err", new ActionError("error.clogin"));
30             formBeanlogon.setErrorMessage(e.toString());
31          }
32
33          // If a message is required, save the specified key(s)
34          // into the request for use by the <struts:errors> tag.
35          if (!errors.isEmpty()) {
36             saveErrors(request, errors);
37             forward = mapping.findForward("error");
38          }
39          else
40             forward = mapping.findForward("showDBs");
41          // Finish with
42          return (forward);
43       }
44    }
```

Die wichtigste Methode, die in einer Action-Klasse bereitgestellt werden muss, ist **execute()**. Innerhalb dieser Methode soll die Klasse der eigentlichen Geschäftslogik aufgerufen werden. Dabei handelt es sich um die Implementierungsklasse **CTGDB2BeanImpl** der im Abschnitt 4.3 erstellten EJB Session Bean (*Zeilen 16-24*). Die Ergebnisse der Ausführung werden danach in der ActionForm-Bean abgelegt (*Zeile 26*).

Als Rückgabewert liefert die Methode ein ActionForward-Objekt, das an das Action-Servlet zurückgegeben wird. Nach der erfolgreichen Ausführung von execute() wird die Steuerung an **database.jsp** Seite weitergeleitet (*Zeile 40*). Falls eine *Exception* auftritt, wird der Text dieser Exception und eine Fehlermeldung (*Zeilen 29, 30*), die im *Property Resource Bundle* der Anwendung gespeichert ist, auf der Seite **error.jsp** angezeigt (*Zeile 37*).

Die restlichen Komponenten (*ActionForm-Beans*, *JSPs* und *Actions*) der Anwendung aus der Abbildung 4.30 auf Seite 58, die in diesem Beispiel nicht berücksichtigt wurden, sind nach dem gleichen Prinzip zu erstellen. An dieser Stelle ist die Anwendung für den nachfolgenden Testlauf bereit. Der Test wird dabei auf einem lokalen und fernen Anwendungsserver durchgeführt (*s. Abschnitt 4.7 auf Seite 68*).

Um die Anwendung im WebSphere Anwendungsserver auf dem padme-Rechner zu testen, soll sie zunächst im EAR-Format aus *IRAD* exportiert werden. Dazu wählt man im Hauptmenü **Datei > Exportieren**. Im neu geöffneten Fenster des Assistenten wählt man als Exportziel **EAR Datei** aus und klickt auf **Weiter**.

Im nächsten Fenster wählt man das Enterprise Application-Projekt **DB2CTGTesterEAR** und das Zielverzeichnis, wo die zu erstellende EAR-Datei gespeichert werden soll. Durch Klick auf **Fertig stellen** werden alle Module der Anwendung in einer Datei namens **DB2CTGTesterEAR.ear** gespeichert. Dabei werden die Informationen des Deployment Descriptors verwendet, der die Metadaten für den Export des Projekts in eine EAR-Datei und für das Ausführen des Projekts auf einem Server enthält (*Abbildung 4.10 auf der nächsten Seite*).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application 1.3//EN" "
    http://java.sun.com/dtd/application_1_3.dtd">
<application id="Application_ID">
      <display-name>DB2CTGTesterEAR</display-name>
      <module id="EjbModule_1143065062750">
            <ejb>DB2CTGTesterEJB.jar</ejb>
      </module>
      <module id="WebModule_1143065456937">
            <web>
                  <web-uri>DB2CTGTesterWeb.war</web-uri>
                  <context-root>DB2CTGTesterWeb</context-root>
            </web>
      </module>
</application>
```

Programm 4.10: Deployment Descriptor der Anwendung (application.xml).

In der integrierten WebSphere-Testumgebung des *IRAD* sind die oben genannte Schritte nicht notwendig. Die Anwendung wird automatisch beim jedem Start des Servers auf Basis der Zuordnungsinformationen des Deployment Descriptor zur Serverkonfiguration hinzugefügt.

4.6 Installieren der Anwendung mit Hilfe des wsadmin

Zur Installation der erstellten J2EE-Anwendung auf dem WebSphere Anwendungsserver (*WAS*) auf dem z/OS-Rechner wird die Scripting-Schnittstelle wsadmin verwendet. Das WAS-Tool wsadmin ist die nicht grafische Alternative für die Administrationskonsole, das für das Konfigurieren und Verwalten von WebSphere Anwendungsserver verwendet wird. Die Verwaltung der Konfigurationen von WAS und der aktiven Anwendungen erfolgt durch die folgende wsadmin-Objekte: *AdminConfig*, *AdminControl*, *AdminApp* und *Help* [CDC04]. Jedes Objekt hat eine Anzahl von Methoden (*Befehlen*), die interaktiv, in einem Script oder an der Eingabeaufforderung eines Betriebssystems ausgeführt werden können.

Wsadmin wird in dem so genannten *bath*-Modus verwendet, wo die Scripting-Befehle in einer Datei enthalten sind. Es werden zwei Skript-Dateien benötigt, die unter einem Windows Betriebssystem erstellt werden können. Die Scripts sind in der Script-Sprache *Jacl*[10] zu schreiben. Erstes Skript ist für die Installation (*Programm 4.11*) und zweites für das Starten der J2EE-Anwendung (*Programm 4.12*) bedacht.

```
set node "w1node1"
set cell "w1cela"
set server "w1sr01"
set appname "DB2CTGTesterEAR"
set ear "/u/PSELES/DB2CTGTesterEAR.ear"
set options [list -node $node -cell $cell -server $server -appname $appname]
$AdminApp install $ear $options
$AdminConfig save
```

Programm 4.11: Jacl-Script install.jacl zur Installation der Anwendung.

[10]Das Tool wsadmin unterstützt zwei Scripting-Sprachen: *Jacl* und *Jython*.

Zur Installation der Anwendung wird der Befehl **install** des *AdminApp*-Objektes verwendet (*s. Abbildung 4.11 auf der vorherigen Seite, Zeile 7*). Als Parameter werden hier der vollständig qualifizierte Name der EAR-Datei und die Installationsoptionen übergeben, die durch die Variablen **ear** und **options** definiert sind.

Die Optionen **node**, **cell** und **server** enthalten die Identifikation des Anwendungsservers und stehen für den Zellennamen, Knotennamen und Servernamen (*Zeilen 1-3*). Die Option **appname** gibt den Anzeigename der Anwendung an (*Zeile 4*). Anschließend müssen die WAS-Konfigurationsänderungen mit dem Befehl in der Zeile 8 abgespeichert werden.

```
1 set node "w1node1"
2 set cell "w1cela"
3 set server "w1sr01"
4 set appname "DB2CTGTesterEAR"
5 set aM [$AdminControl queryNames cell=$cell,node=$node,type=ApplicationManager,process=
      $server,*]
6 $AdminControl invoke $aM startApplication $appname
```

Programm 4.12: Jacl-Script **start.jacl** zum Starten der Anwendung.

Mit Hilfe des zweiten Scripts soll die Anwendung gestartet werden. Dazu wird *AdminControl*-Objekt verwendet, das für die Ausführung von Betriebsbefehlen steht.

Zunächst wird der Anwendungsmanager-MBean für den Server ermittelt (*Zeile 5*). Die MBeans (*Managed Beans*) sind Java-Objekte, die in einem WebSphere-Serverprozess ausgeführt werden, und für die Darstellung von JMX[11]-Ressourcen zuständig sind.

Zum eigentlichen Starten der Anwendung wird die Methode **invoke** des *AdminControl*-Objektes verwendet. Dabei wird die **startApplication**-Operation des Anwendungsmanagers MBean aufgerufen, die als Parameter den Namen der zu startenden Anwendung zugewiesen bekommt (*Zeile 6*).

Als nächstes müssen die beide Scripts und die J2EE-Anwendung **DB2CTGTesterEAR.ear** per FTP im binären Transfermodus auf den z/OS-Rechner übertragen und im HFS-(*Hierarchical File System*)-Verzeichnis "**/u/PSELES**" abgelegt werden.

Dafür startet man von einem Windows-Rechner die Eingabeaufforderung und wechselt ins Verzeichnis, das die Script-Dateien und *Jar*-Datei enthält (*s. Abbildung 4.35 auf der nächsten Seite*). Danach gibt man den Befehl **ftp 139.18.4.35** und loggt sich als TSO-Benutzer ein. Bevor die Dateiübertragung gestartet wird, stellt man die Verbindung auf binary-Übertragungsmodus um. Nun können Dateien mit dem **put**-Befehl hochgeladen werden.

[11] JMX (*Java Management Extensions*) ist eine Spezifikation zur Verwaltung und Überwachung von Java-Objekten.

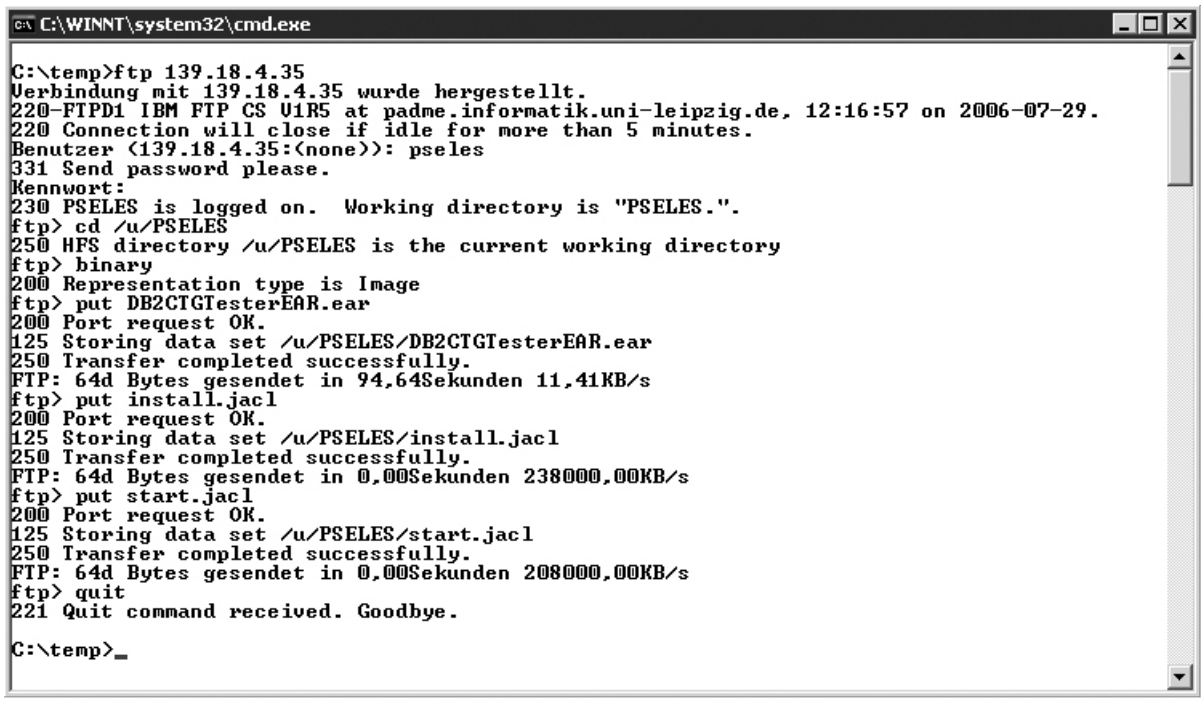


Abbildung 4.35: FTP-Vorgang.

Danach führt man die erstellte Scripts mit Hilfe des *wsadmin-Scripting-Clients* aus, das sich standardmäßig im bin Verzeichnis des Anwendungsservers befindet (in diesem Fall /WebSphere/V5R0M0/AppServer/bin).

```
1  PSELES:/WebSphere/V5R0M0/AppServer/bin: >./wsadmin.sh -conntype SOAP -host localhost
2  -port 9140 -f /u/PSELES/install.jacl
3  WASX7209I: Connected to process "w1sr01" on node w1node1 using SOAP connector;The type of
       process is: UnManagedProcess
4  ADMA5016I: Installation of DB2CTGTesterEAR started.
5  ADMA5005I: Application DB2CTGTesterEAR configured in WebSphere repository
6  ADMA5001I:Application binaries saved in /WebSphere/V5R0M0/AppServer/wstemp/
       Script10a3f36b7ef/workspace/cells/w1cela/applications/DB2CTGTesterEAR.ear/
       DB2CTGTesterEAR.ear
7  ADMA5011I: Cleanup of temp dir for app DB2CTGTesterEAR done.
8  ADMA5013I: Application DB2CTGTesterEAR installed successfully.
9
10 PSELES:/WebSphere/V5R0M0/AppServer/bin: >./wsadmin.sh -conntype SOAP -host localhost
11 -port 9140 -f /u/PSELES/start.jacl
```

Programm 4.13: Ausführung der Jacl-scripts mit dem wsadmin-Client.

Die Abbildung 4.13 zeigt die Ausführung der beiden Skripten. Dabei ist zu beachten, dass das Script-Client mit dem Server über SOAP-Connector verbunden sein soll, da der Application Server gegenwärtig aktiv ist. Anschließend soll das Zusammenspiel des CICS-Programms und der installierten J2EE-Anwendung getestet werden.

4.7 Test der J2EE-Anwendung

Wie bereits zuvor erwähnt wurde, kann die Anwendung innerhalb des Windows- oder z/OS-Betriebssystem getestet werden. Zur Durchführung des Tests auf dem Websphere Anwendungsserver des IRAD soll zuerst der Server, dessen Instanz im Abschnitt 4.2.3 erstellt wurde, gestartet werden. Dazu in der *Serverkonfiguration*-Sicht wählt man Server und startet dessen Instanz über das Kontextmenü. Wurde der Server ordnungsgemäß gestartet, was in einem Startprotokoll in der Konsole ermitteln werden kann, so kann die Anwendung mit einem normalen Webbrowser aufgerufen werden.

In der Adresszeile des Browsers gibt man in Abhängigkeit davon, ob die Anwendung in einem lokalen oder fernen Anwendungsserver getestet werden soll, entweder die Adresse *http://localhost:9080/DB2CTGTesterWeb* oder *http://padme.informatik.uni-leipzig.de:9148/DB2CTGTesterWeb* ein. Daraufhin erscheint die Anmeldeseite der Web-Anwendung (*Abbildung 4.36*).

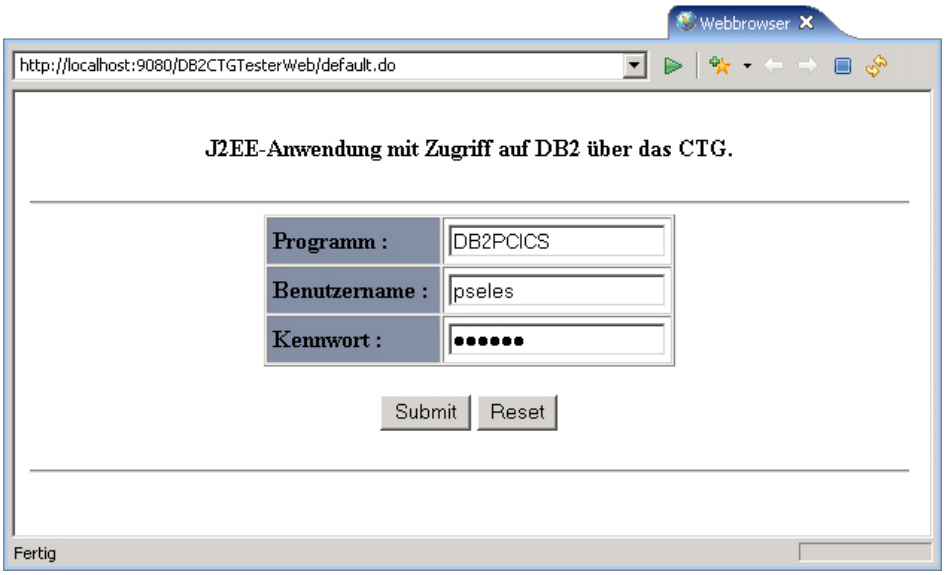


Abbildung 4.36: Anmeldeseite der Web-Anwendung.

In diesem Formular müssen das CICS-Programm **DB2PCICS** und die Authentisierungsinformationen, die auf dem z/OS-Rechner gültig sind, eingegeben werden. Nach dem Absenden des Formulares durch die Betätigung der **Submit**-Taste wird die Anfrage an CICS gesendet. Dabei können einige Fehler bei der Ausführung der Anwendung oder dem Zugriff auf das CICS-Programm auftreten. Die drei häufigsten Fehlermeldungen, die während des Tests auftraten, sind folgende:

```
com.ibm.connector2.cics.CICSTxnAbendException:
CTG9638E: Transaction Abend occurred in CICS. Abend Code=AEI0
```

Laut **CICS Messages and Codes** [GC305] besagt diese Fehlermeldung, dass ein aufrufbares Programm nicht gefunden werden kann. In diesem Fall liegt der Fehler darin, dass die Definition des **DB2PCICS** Programms nicht installiert ist. Um diesen Fehler zu beheben, muss der Befehl

CEDA INSTALL PROGRAM(DB2PCICS) GROUP(PSELES) erneut ausgeführt werden.

Falls CICS und DB2 nicht miteinander verbunden sind, wird dem Programm den Abend Code AEY9 zurückgegeben.

```
com.ibm.connector2.cics.CICSTxnAbendException:
CTG9638E: Transaction Abend occurred in CICS. Abend Code=AEY9
```

Das Problem in diesem Fall liegt darin, dass entweder die CICS-DB2-Verbindung nicht installiert oder gestartet ist. Um dies genauer herauszufinden, müssen folgende Befehle innerhalb CICS ausgeführt werden.

```
CEMT INQUIRE DB2CONN
CEMT INQUIRE DB2ENTRY
```

In Abhängigkeit davon, welche Ergebnisse die beiden Befehle liefern, müssen die folgende Befehle aus dem Abschnitt 3.4 auf Seite 19 wiederholt werden.

```
CEDA INSTALL DB2CONN(DB2CON) GROUP(PSELES)
CEMT SET DB2CONN CONNECTED
CEDA INSTALL DB2ENTRY(DB2CONE) GROUP(PSELES)
```

Die Hauptursache für die beiden oben genannten Fehler liegt in der CICS-Konfiguration. CICS hat eine so genannte GROUP LIST mit den Namen von Gruppen, welche bei jedem Neustart des CICS automatisch mitinstalliert werden. Auf dem *padme*-Rechner heißt diese Liste XYZLIST und in der CICS-Parametertabelle (SIT) durch den Parameter GRPLIST=(XYZLIST) definiert ist. Da die PSELES-Gruppe nicht zur XYZLIST-Liste gehört, müssen alle Ressourcendefinitionen der PSELES-Gruppe, bei jedem neuen Start der CICS-Region, erneut installiert werden.

Für die Beseitigung dieser Fehler müsste die PSELES-Gruppe zur GROUP LIST mit Hilfe des Befehls CEDA ADD GROUP(PSELES) LIST(XYZLIST) hinzugefügt werden. Selbst danach kann der zweite Fehler noch auftreten. Die Ursache dafür liegt darin, dass die Verbindung zwischen CICS und DB2 nicht automatisch startet. Dazu müsste der Parameter DB2CONN=YES in der SIT-Tabelle festgelegt werden. Anschließend muss CICS-Region neu gestartet werden, damit alle vorgenommene Änderungen in Kraft treten könnten.

Der dritte Fehler besagt, dass die Verbindung vom CICS Transaction Gateway zum CICS Server nicht hergestellt werden kann.

```
javax.resource.spi.CommException:
CTG9631E: Error occurred during interaction with CICS. Error Code=ECI_ERR_NO_CICS
```

Als Ursache dafür könnte es sein, dass der CICS-Server nicht aktiv oder die IRC (*Inrerregion Communication*)-Verbindung nicht geöffnet ist. Hierzu muss entweder CICS-Region neu gestartet oder IRC erneut geöffnet werden.

Falls keine oben genannte Fehler auftreten, erscheint nach dem Anmeldevorgang die Ergebnisseite, in der der Inhalt der Datenbank in tabellarischer Form mit Spalten und Überschriften angezeigt wird (*Abbildung 4.37 auf der nächsten Seite*). Die Bedienoberfläche der Anwendung ist selbsterklärend

und enthält Schaltflächen zum Verändern von Daten der Datenbank. Dabei werden die gleiche Vorgänge wie beim Test aus dem Abschnitt 3.7 durchgeführt.

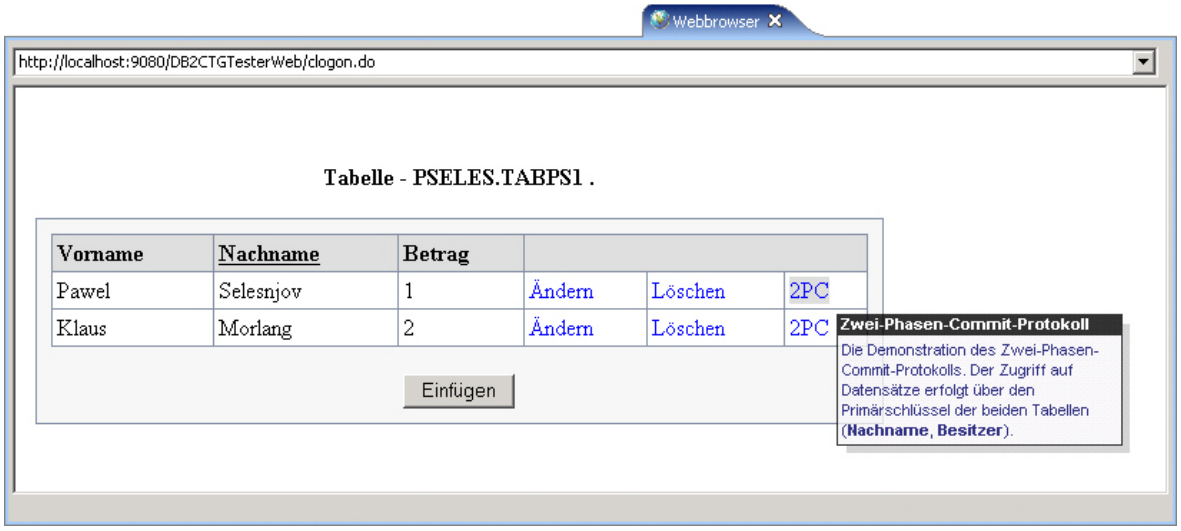


Abbildung 4.37: Die Ergebnisseite der Web-Anwendung.

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 und verändert. Dabei der Zugriff auf CICS ist durch CICS ECI Ressourcenadapter und Gateway deamon auf dem z/OS-System realisiert.

Kapitel 5 Zusammenfassung

Im Kapitel 2 wurden die verwendete Software kurz dargestellt und ihr Verwendungszweck erläutert. Es wurden der Hauptzweck vom CICS Transaction Gateway (*CICS TG*) und seine Einsatzmöglichkeiten in verschiedenen Topologien dargestellt.

Das dritte Kapitel befasst sich mit der Konfigurierung des CICS TG und CICS und der Erstellung des CICS-DB2-Programms. Dabei wurden das CICS TG und CICS durch einige Sicherheitsmaßnahmen gestärkt. Auch wurde der Aufbau einer Verbindung zwischen CICS- und DB2-Subsystemen schrittweise erklärt. Anschließend wurde das Programm mit Hilfe der J2EE-Anwendung im lokalen Modus auf dem z/OS-System getestet.

Im letzten Kapitel wurde basierend auf den Konfigurationen des dritten Kapitels eigene J2EE-Anwendung mit Hilfe des IRAD entwickelt. Deren Erstellung wurde schrittweise erklärt und im WebSphere Test Environment, das Teil des Rational Application Developer ist, getestet. Anschließend wurde die Anwendung im WebSphere Anwendungsserver auf dem z/OS-Betriebssystem installiert und ebenfalls getestet.

Das komplette Zusammenspiel der bereits genannten Komponenten kann die Grundlage für weitere Untersuchungen bilden.

Es kann zum Beispiel ein CICS-Programms erstellt werden, das neben dem Zugriff auf DB2-Tabellen auch auf VSAM-Dateien oder IMS-Segmente zugreifen kann. IMS und CICS können dabei mit Hilfe DBCTL (*IMS Database Control*)-Schnittstelle eingebunden werden. Außerdem kann vom CICS TG unterstützte *two-phase-commit* Protokoll verwendet werden, wenn der Zugriff auf CICS vom WebSphere Anwendungsserver für z/OS über CICS-ECI-Ressourcenadapter stattfindet.

Statt der Version 5.1 des CICS Transaction Gateway kann in dieser Konfiguration die aktuelle Version 6.1 vom CICS TG eingesetzt und getestet werden. Für ECI-Aufrufe bietet CICS TG V6 zusätzlich zu dem ECI- und EPI-Ressourcenadapter auch einen neuen ECI-XA-Ressourcenadapter.

ECI-XA ist der einzige Ressorcenadapter, der die *two-phase commit* Unterstützung bietet, wenn die Verbindung zu CICS durch z/OS *Gateway deamon* erfolgt. Somit können CICS Transaktionen mit *two-phase commit* über XA-Protokoll im Rahmen einer plattformübergreifenden verteilten Transaktion abgewickelt werden. Nähere Informationen über das CICS TG V6 und dessen vielseitigen Einsatzmöglichkeiten sind im Redbook [CS06] zu finden.

Anhang A Inhalt der beiliegenden CD

Verzeichnis	Beschreibung
/Diplom	Diplomarbeit im Windows Word- und PDF-Format
/Quellen	einige innerhalb dieser Diplomarbei verwendete Literaturquellen in digitaler Form
/Quelltexte/CICS	CICS-Programme und JCL-Scripte dieser Diplomarbeit
/Quelltexte/WDz	mit WebSphere Developer für zSeries hergestellte J2EE-Anwendung
/Abbildungen	Alle Abbildungen dieser Diplomarbeit

Tabelle A.1: Inhalt der beiliegenden CD.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfaßt und keine anderen als die angegebenen Literatur und Hilfsmittel verwendet habe.

Ort, Datum

Unterschrift

Literaturverzeichnis

[AC05] Alex, Kooijmans ; Chris, Backhouse: *WebSphere for z/OS V6 Connectivity Handbook, IBM Form Nr. SG24-7064-02*. dritte Ausgabe. http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg247064.pdf : IBM, International Technical Support Organization, Dezember 2005. – 595 S. – ISBN 073849253 I

[Apaa] Apache. *Apache Struts*. http://struts.apache.org, zuletzt abgerufen im Juni 2006.

[Apab] Apache. *Apache Struts userGuide*. http://struts.apache.org/1.x/userGuide/index.html, zuletzt abgerufen im Juni 2006.

[CDC04] Carl, Wohlers ; Donald C., Bagwell: *WSADMIN Scripting Primer. IBM Form No. WP100421*. zweite Ausgabe. http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100421 : IBM Washington Systems Center, Mai 2004. – 107 S

[CP05] Chris, Rayns ; Pingze, Gao: *Revealed! Architecting e-business Access to CICS, IBM Form Nr. SG24-5466-04*. fünfte Ausgabe. http://www.redbooks.ibm.com/redbooks/pdfs/sg245466.pdf : IBM, International Technical Support Organization, Februar 2005. – 354 S. – ISBN 0738490970

[CS06] Colin, Alcock ; Steven, Day: *CICS Transaction Gateway for z/OS Version 6.1, IBM Form Nr. SG24-7161-00*. erste Ausgabe. http://www.redbooks.ibm.com/redbooks/pdfs/sg247161.pdf : IBM, Mai 2006. – 406 S. – ISBN 0738496227

[G2204] *Integrating WebSphere Application Server and CICS using the J2EE Connector Architecture, IBM Form Nr. G224-7218-00*. e-business solutions White paper. IBM, Januar 2004. – 20 S

[GC305] *CICS Messages and Codes. IBM Form No. GC34-6241-03*. vierte Ausgabe. http://www.elink.ibmlink.ibm.com/public/applications/publications/ : IBM, September 2005. – 1368 S

[HWG03] Herrmann, Paul ; Wilhelm G., Spruth: *Einführung in z/OS und OS/390*. München, Wien : Oldenbourg, 2003. – 298 S. – ISBN 3-486-27214-4

[IBMa] IBM. *Introduction to Rational Application Developer*. http://publib.boulder.ibm.com/infocenter/radhelp/v6r0m1/, zuletzt abgerufen im Februar 2007.

[IBMb] IBM. *Rational Application Developer for WebSphere Software V7.0*. Download unter http://www-128.ibm.com/developerworks/downloads/

[IBMc] IBM. *WebSphere Developer for System z*. Download unter http://www-306.ibm.com/software/awdtools/devzseries/library/.

[PJ02] Phil, Wakelin ; John, Joro: *CICS Transaction Gateway V5, The WebSphere Connector for CICS, IBM Form Nr. SG24-6133-01*. zweite Ausgabe. http://www.redbooks.ibm.com/redbooks/pdfs/sg246133.pdf : IBM, Dezember 2002. – 418 S. – ISBN 0738426687

[SC106] *Application Programming and SQL Guide. IBM Form No. SC18-7415-03*. vierte Ausgabe. http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db2.doc.pdf/ : IBM, Februar 2006

[SC303] *CICS TS for z/OS, CICS DB2 Guide. IBM Form No. SC34-6252-03*. dritte Ausgabe. IBM, Dezember 2003. – 218 S

[SC304] *CICS Transaction Gateway, z/OS Administration. IBM Form No. SC34-6191-01*. zweite Ausgabe. IBM, März 2004. – 208 S

[SC305a] *CICS External Interfaces Guide, IBM Form No. SC34-6006-11*. dritte Ausgabe. IBM, September 2005. – 343 S

[SC305b] *CICS TS for z/OS, Resource Definition Guide. IBM Form No. SC34-6430-00*. erste Ausgabe. IBM, März 2005. – 716 S

[SS01] Sharma, Rahul ; Stearns, Beth: *J2EE Connector Architecture and Enterprise Application Integration*. erste Ausgabe. Addison Wesley, Dezember 2001. – 416 S. – ISBN 0–201–77580–8

[Sun] Sun, Microsystems. *J2EE Connector Architecture*. http://java.sun.com/j2ee/connector/index.jsp, zuletzt abgerufen im August 2006.

[TB05] Tamas, Vilaghy ; Bob, Cronin: *WebSphere for z/OS V5 Connectivity Handbook, IBM Form Nr. SG24-7064-01*. zweite Ausgabe. http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg247064.pdf : IBM, International Technical Support Organization, April 2005. – 394 S. – ISBN 073849206X

[Tut03] *Tutorial 4, DB2, Universität Leipzig*. Download unter http://jedi.informatik.uni-leipzig.de/tutors/tutor4.pdf. Januar 2003

[Tut05] *Tutorial 14, CICS Transaction Gateway, Universität Leipzig*. Download unter http://139.18.4.35/tutor_z/tutor14.pdf. Januar 2005

Stichwortverzeichnis