Diplomarbeit im Fach Informatik

Erstellung und Portierung mehrerer Tutorials mit dem Thema WebSphere Developer for zSeries für den studentischen Praktikumsbetrieb

Betreut von
Prof. Dr.-Ing. Wilhelm G. Spruth

Vorgelegt von
Matthias Beyerle
Matrikelnummer: 2087377

an der
Eberhard-Karls-Universität Tübingen
Wilhelm-Schickard-Institut für Informatik
Lehrstuhl Technische Informatik

Tübingen, den 30.08.2007

Zusammenfassung

Ziel der Diplomarbeit war es, Tutorials über die Funktionsweise von *WebSphere Developer for zSeries* (WDz) für den studentischen Praktikumsbetrieb aufzuarbeiten und auf dem zSeries Großrechner der Universität Leipzig verfügbar zu machen. Grundlage für die vorliegende Arbeit waren Tutorials, die bei einer IBM Veranstaltung in Hamburg von Frau Isabel Arnold im Sommer 2006 vorgestellt wurden. Themen der Übungen sind neben einer Einführung in die Funktionsweise von WDz eine Installationsanleitung für das benötigte lokale Arbeitsumfeld, lokale und remote Kompilierung von COBOL Programmen mit WDz, eine Einführung in die Grundlagen von Web Services, Tests von Web Services und Erstellung eines Web Service unter Verwendung einer mit dem CICS Transaktionsserver Version 3.1 verfügbaren Beispielapplikation.
Die benötigten Programme und Daten sind auf einer vorinstallierten Workstation vorhanden. Diese Workstation basiert auf Windows XP und wird als virtuelle Maschine auf einem Host installiert. Alle nötigen Dateien, Programme und Tutorials sind auf zwei DVDs vorhanden, die sich im Anhang dieser Arbeit befinden und die Studenten bei Praktikumsbeginn zur Verfügung gestellt werden.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

_____	_____
Ort, Datum	Unterschrift

Danksagung

An dieser Stelle möchte ich allen Personen danken, die mich bei der Erstellung dieser Diplomarbeit unterstützt haben.
Als erstes möchte ich meinen Eltern und meinen Großeltern danken, die mir während meines gesamten Studiums zur Seite standen und ohne die ich mein Studium nicht hätte abschließen können. Weiterer Dank gilt meiner Freundin Ana Pimentel für ihre Geduld und ihr Verständnis.

Besonderer Dank gilt meinem Betreuer Herrn Professor Dr. Wilhelm G. Spruth, der mir diese Arbeit ermöglicht hat und mich in vielfältiger Weise während der gesamten Umsetzung unterstützte.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Jedes Semester findet an der Universität Tübingen ein die Vorlesung Client/Server-Systeme begleitendes Praktikum statt, in dem Grundlagen verschiedener Bereiche im Client/Server Sektor in Form praktischer Arbeiten vermittelt werden.

Neben Übungen zu *CORBA* und *Java Remote Method Invocation* (RMI) ist ein Schwerpunkt dieses Praktikums die Einführung in die Grundlagen von Großrechnersystemen und deren Betriebssystem z/OS sowie dem auf diesen Komponenten aufbauenden Transaktionsserver *Customer Information Control System* (CICS) und dem *Database Management System DB2*.

Großrechner (oder Mainframes, im Folgenden werden diese Begriffe synonym verwandt) werden im Alltag immer mehr von Webanwendungen unterschiedlichster Ausprägung in Anspruch genommen.
Um diesen Trend im Praktikum zu veranschaulichen war die Aufgabe dieser Diplomarbeit, mehrere Tutorials zu erstellen, die sich gezielt mit diesen Themen auseinander setzen und in diesem Zusammenhang ein Basiswissen über die Verwendung von WebSphere Developer for zSeries geben. Grundlage für diese Tutorials war ein in Hamburg abgehaltenes, zweiwöchiges Seminar von Frau Isabel Arnold von der IBM über Anwendungsentwicklung und Host Modernisierung auf zSeries Rechnern.

Die aus der Arbeit hervorgegangenen Übungen, die sich zusammen mit der vorinstallierten Arbeitsumgebung als DVDs im Anhang befinden, werden erstmals im Wintersemester 2007/2008 Teil des Client/Server-Praktikums an der Universität Tübingen sein und werden voraussichtlich auch von ähnlichen Veranstaltungen anderer Hochschulen übernommen, die sich momentan an Tutorials wie sie in Tübingen Verwendung finden anlehnen.

1.2 Aufbau der Arbeit

Die schriftliche Ausarbeitung der Diplomarbeit ist bewusst kurz gehalten, da die eigentliche Arbeit, die sechs erstellten Tutorials, im Anhang in elektronischer Form auf einer DVD beiliegen und die Integration der Übungen in voller Länge den Rahmen der Diplomarbeit übersteigen würde.

Im weiteren Verlauf der Ausarbeitung wird in Kapitel zwei eine kurze Einführung in die verwendete Hardware des Servers gegeben, ein zSeries Rechner der Universität Leipzig, auf dem auch weitere schon bestehenden Übungen des Client/Server-Praktikums laufen. Weiter gibt es in Kapitel drei eine Übersicht über die verwendete Software *WebSphere Developer for zSeries* (WDz) und die Entwicklungsumgebung *Eclipse*, auf der WDz als Plugin aufsetzt. Im Anschluss daran werden in Kapitel vier die im Rahmen der Diplomarbeit entwickelten Tutorials vorgestellt, zusammen mit einer Beschreibung der jeweils darin behandelten Themen. Nachfolgend gibt es in Kapitel fünf eine Zusammenfassung und ein Ausblick auf mögliche weitere Studien- oder Diplomarbeiten, die auf der verwendeten Arbeitsumgebung aufbauen könnten. Die schriftliche Ausarbeitung wird von einem Anhang und dem Literaturverzeichnis abgeschlossen.

Kapitel 2

Hardware: zSeries

Die zSeries (zwischenzeitlich auch als System z bezeichnet) ist die aktuelle Generation an Großrechnern der *International Business Machines Corporation* (IBM).

Die Firma IBM blickt auf eine äußerst erfolgreiche sowie richtungweisende Geschichte ihrer Großrechner. Mit dem System/360 brachte IBM 1964 den ersten Rechner auf den Markt, der die bis dahin getrennten Ausrichtungen der Computer in wissenschaftliche und kommerziell genutzte Systeme auf einen gemeinsamen Nenner brachte.
Möglich wurde diese allgemeine Ausrichtung durch die Einführung von Microcode, der die bis dahin fest verdrahteten Programme ablöste und den Rechner dadurch für die unterschiedlichsten Anwendungen öffnete. Viele der ersten Programme, die in Assembler, Fortran, Cobol und PL/1 geschrieben wurden sind heute noch in Gebrauch.
Die Abwärtskompatibilität der aktuellen gegenüber der älteren Modellen ist ein weiterer Meilenstein der IBM Großrechner Geschichte. Um mit der rasanten Entwicklung der Computerindustrie Schritt zu halten, wurden regelmäßig neue Versionen von Großrechnern auf den Markt gebracht, die jeweils kompatibel zu ihren Vorgängern waren, gleichzeitig durch mehr und schnellere Prozessoren, mehr Arbeitsspeicher und neuen Automatismen zur Fehlererkennung und -behebung an das neue Arbeitsumfeld angepasst waren.

2.1 Eigenschaften der zSeries

Im Folgenden sollen einige der herausragenden Eigenschaften der zSeries hervorgehoben werden, ohne jedoch stark ins Detail zu gehen, da dies den Rahmen dieser Arbeit sprengen würde.

2.1.1 Abwärtskompatibilität

Die aktuellen Modelle der zSeries sind kompatibel zu allen älteren Großrechnergenerationen. Deshalb ist es möglich, Jahrzehnte alten Code ohne Änderungen auf neuen Großrechnern auszuführen. Selbst eine Neukompilierung der Programme ist nicht nötig.
Da es jedoch nicht produktiv ist, Abwärtskompatibilität auf Microcodeebene zu erhalten, werden bei der zSeries selten genutzte Maschinenbefehle, die nur dem Erhalt der Kompatibilität des Binärcodes dienen, durch den so genannten *Licenced Internal Code* (LIC) nachgebildet [Her04a]. Diese Softwarekomponente vermittelt zwischen Hardware und Software, nur sie kann die Hardware direkt ansprechen. LIC stellt Anwendungen alle nötigen Schnittstellen zur Verfügung. Dadurch können neue Programme von neuen, optimierten Befehlssätzen profitieren und ältere Programme

bleiben lauffähig.

2.1.2 Virtualisierung

Unter Virtualisierung versteht man Methoden, durch die Computerressourcen aufgeteilt werden können. Übertragen auf die zSeries bedeutet dies, dass mehrere Betriebssysteme gleichzeitig nebeneinander ausgeführt werden können, ohne sich gegenseitig zu beeinträchtigen. Grundsätzlich bietet die zSeries zwei unterschiedliche Ansätze hierfür: zum Ersten kann das System in so genannte *Logical Partitions* (LPARs) aufgeteilt werden, zum Anderen steht mit z/VM ein Betriebssystem bereit, welches weitere Virtualisierungsmechanismen anbietet [Her06].

2.1.3 Ausfallsicherheit

Die zSeries gilt aufgrund ihrer Architektur als äußerst ausfallsicher. IBM nennt für einzelne Rechner eine Verfügbarkeit von 99.999%. Dies ist besonders bei unternehmenskritischen Anwendungen von Bedeutung, da hier jede Minute Ausfallzeit bis zu 10 000$ kosten kann [CI99].

2.1.4 Skalierbarkeit, Parallel Sysplex und Coupling Facility

Im laufenden Betrieb können sich Anforderungen an die Hardware schnell ändern. Damit es zu keinen langen Wartezeiten bei Anfragen oder zu Ausfällen durch Überlastung kommt, kann die zSeries flexibel auf geänderte Anforderungen reagieren. Sind noch nicht alle Prozessoren einer zSeries aktiviert, können dynamisch weitere Prozessoren hinzugenommen werden um Lastspitzen auszugleichen. Es können auch ganze Großrechner zugeschaltet werden, die als Cluster *Parallel Sysplex* genannt werden. Diese Cluster wirken nach außen wie ein einzelner Rechner. Die Koordination zwischen den einzelnen Rechnern übernimmt die so genannte *Coupling Facility* (CF), eine Hardwarekomponente, die entweder Teil eines der im Verbund befindlichen Rechners ist, oder auf einem separaten Mainframe läuft. Da der Verwaltungsaufwand für das Cluster auf die CF beschränkt ist, steigt die Leistung eines Clusters fast linear mit den eingebundenen Knoten.

2.1.5 Workloadmanager

Der *Workloadmanager* (WLM) ist Teil des z/OS Betriebssystems und ermöglicht eine dynamische Verteilung der Ressourcen zwischen einzelnen so genannten Dienstklassen *(Service Classes)*. Hier können verschiedene Prioritäten und Zielvorgaben definiert werden. Der WLM kümmert sich automatisch um eine Optimierung der Lastverteilung und berechnet die Verteilung kontinuierlich neu, indem er Daten über vergangene Ereignisse sammelt und auswertet. Um dabei ein möglichst optimales Ergebnis zu erzielen, kommen verschiedene Lösungsansätze zum Einsatz [Her04b]. Unter anderem wird im IBM Labor in Böblingen an einem WLM geforscht, dem ein neuronales Netzwerk zugrunde liegt.

Kapitel 3

Software: Eclipse und WebSphere Developer for zSeries

WebSphere Developer for zSeries (WDz) ist eine integrierte Entwicklungsumgebung *(integrated developement enviroment, IDE)* für Großrechneranwendungen. WDz setzt auf die Open Source Platform *Eclipse* auf, deren gleichnamige Java-IDE laut [Cas04] von über 50% der Java-Programmierer genutzt wird.

3.1 Eclipse

Eclipse ist eine Open Source Plattform zur Entwicklung von Software [Ecl]. Besonders bekannt ist Eclipse für seine Entwicklungsumgebung für die Programmiersprache Java, es gibt jedoch auch eine Vielzahl an Plugins für weitere Programmiersprachen, unter anderem C, C++, Perl, PHP, Ruby und Python.

Eclipse basiert auf der IDE *Visual Age for Java 4.0*, die von IBM entwickelt wurde. Im November 2001 wurde der Quellcode für das Programm freigegeben und seitdem kontinuierlich als Open Source Projekt erweitert. Das von IBM geleitete *Eclipse Konsortium* mit über 80 Mitgliedern, neben IBM waren unter anderem Borland, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft und Webgain vertreten, gründete im Januar 2004 die rechtlich unabhängige *Eclipse Foundation*, die seitdem für die Entwicklung von Eclipse verantwortlich ist [Wik1].

Eclipse basiert auf einem Prinzip, bei dem sich alles als Plugin integrieren lässt:

- Erweiterbare Entwicklungsumgebung

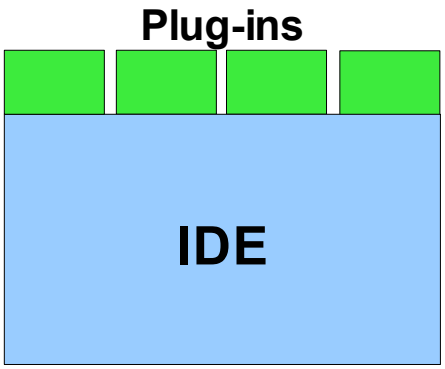

Abbildung 3.1.1: Eclipse als Entwicklungsumgebung

- Platform „everything is a plug-in“

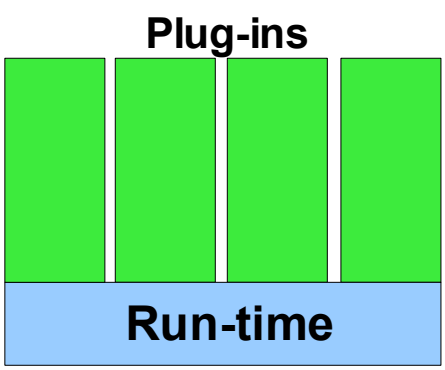

Abbildung 3.1.2: Eclipse als Run-time Enviroment

Die Eclipse Open Source Community versteht sich als Gemeinschaft, die eine kostenlose, erweiterbare Entwicklungsumgebung und ein Framework für die Laufzeit und Anwendungsentwicklung zur Verfügung stellt, welches die Entwicklung, Betreuung und Wartung eines Softwareprojekts über dessen gesamte Lebensdauer begleitet. Die Community stellt über 60 verschiedene Open Source Projekte bereit, die auf [Ecl2] in sieben Kategorien unterteilt werden :

- Enterprise Development
- Embedded und Device Development
- Rich Client Platform
- Rich Internet Applications
- Application Framework
- Application Lifecycle Management (ALM)
- Service Oriented Architecture (SOA)

3.2 WebSphere Developer for zSeries

WDz ist optimiert für die Zusammenarbeit mit IBM WebSphere Software und zSeries und enthält Funktionen, die die traditionelle Programmierung für Großrechneranwendungen, Webentwicklung und die Entwicklung für integrierte, gemischte Workloads beschleunigt. Neben Werkzeugen, die der schnelleren Entwicklung dynamischer Webanwendungen, Java und J2EE dienen, gibt es damit erstmals die Möglichkeit, dieselben Werkzeuge für traditionelle COBOL und PL/I-Programmierung auf Großrechnerebene zu verwenden. Auch die Anwendungsprogrammierung in der Programmiersprache *Enterprise Generation Language* (EGL) und die Erstellung von Web Services profitieren von diesen Neuerungen [IBM1, IBM2].

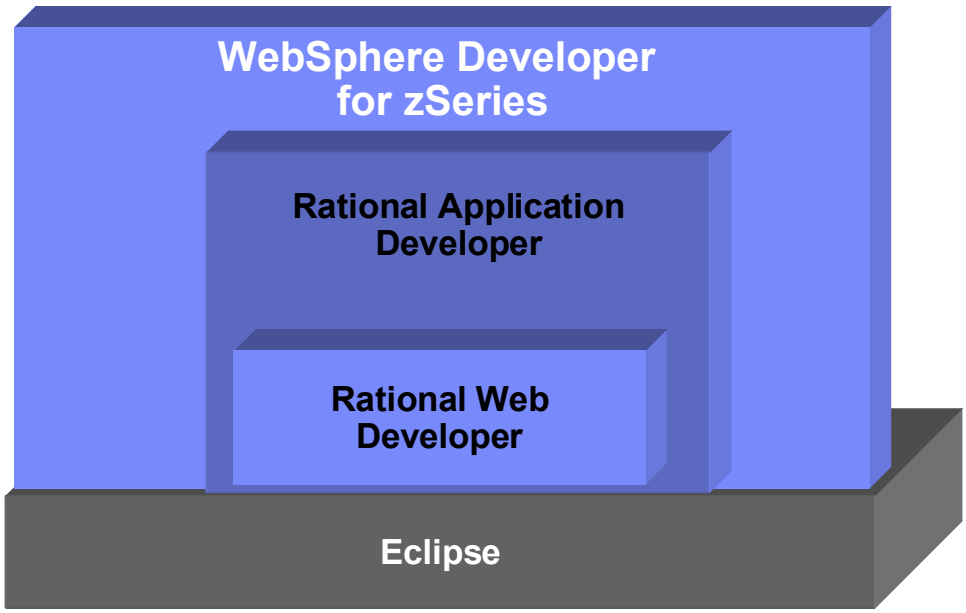

Abbildung 3.2.1: Plugin-Übersicht des WebSphere Developer for zSeries Pakets

WebSphere Developer for zSeries ist der Nachfolger von *WebSphere Studio Enterprise Developer* und baut als Plugin auf die Eclipse IDE auf. In dem Plugin enthalten sind die beiden IBM Rational-

Produkte *Rational Application Developer* und *Rational Web Developer*, die Funktionen für die schnelle und einfache Entwicklung von Java/J2EE-, Portal-, Web-, Web-Service- und SOA-Anwendungen bereitstellen. Weiterhin umfasst die Rational-Software Werkzeuge zur Integration, Realisierung und zum Testen entwickelter Anwendungen. Für weiterführende Informationen zu Rational Application Developer sei [RAD06] empfohlen.

Das WebSphere Developer for zSeries-Plugin als umgebender Rahmen des Gesamtmoduls ist für die z/OS Anwendungsentwicklung, XML Services, *Basic Mapping Support* (BMS) Map Editor, COBOL und PL/I DB2 Stored Procedures und die EGL COBOL Funktionalität verantwortlich.

Abbildung 3.2.2: Detailansicht des WDz Plugins

Im Folgenden werden die oben aufgeführten Kategorien genauer betrachtet.

3.2.1 z/OS Application Development

Mit WDz ist es möglich, sich auf ein z/OS System einzuloggen und mit den auf dem z/OS System liegenden Daten zu arbeiten, als ob es Dateien der Workstation wären. Dies wird durch das so genannte *z/OS File System Mapping* ermöglicht, bei dem auf der Workstation zusätzlich zu einzelnen Datasets auch spezielle Member dieser Datasets gemappt werden können. Das ermöglicht es, mit Datasets zu arbeiten, die Member unterschiedlichen Typs beinhalten. Die Daten werden von der Workstation gemäß ihrer Mappingkriterien behandelt. Von der Workstation aus können auch neue Datasets und Member angelegt und bearbeitet werden.

Der bei der Bearbeitung der Dateien verwendete Editor hat volle ISPF *(Interactive System Productivity Facility)* Funktionalität und ist durch eine Vielzahl weiterer Module auf dem Stand aktueller Entwicklungsumgebungen. Hervorzuheben sind hierbei ein *Content Assist* für COBOL

und PL/I, der bei Syntaxvervollständigung automatisch alle Ressourcen integriert, auf die ein Entwickler Zugriff hat. Weitere Funktionen sind lokale und remote Syntaxprüfung sowie die in Eclipse integrierten Werkzeuge *Compare With...* und *Replace with Local History*.

Ein weiterer Teil von WDz ist die Interaktion mit dem *Job Entry System* (JES), bei dem Jobs an den Großrechner übermittelt, überwacht und deren Ergebnisse betrachtet werden können. Die dabei verwendeten *Job Control Language*-Files (JCL-Files) für compile, link-edit und run können automatisiert aus dem vorhandenen Quellcode generiert und an das entsprechende z/OS System zur Ausführung gesandt werden.

Generell sind alle typischen Editier-, Kompilier- und Debugfunktionen lokal und auf dem remote z/OS System von der Workstation aus verfügbar. Die meisten Funktionen können auch im Offline-Modus verwendet werden. Hier ist als Voraussetzung nötig, entsprechende Projekte und Daten offline verfügbar zu machen, wofür eine Routine in WDz bereit steht. Um Änderungen lokal und offline testen zu können ist für die Testumgebung der Workstation ein CICS Transaktionsserver in WDz integriert, der CICS-Anweisungen lokal übersetzt und dadurch ein Testen ermöglicht. Hier gibt es jedoch seitens der IBM den Hinweis, ausschließlich lokal getestete z/OS Anwendungen nicht ohne weitere Prüfung auf einem Großrechner in die Produktion aufzunehmen.

3.2.2 XML Services for the Enterprise

In diesem Abschnitt werden der Zugang von *Service orientierter Architektur* (SOA) auf CICS V3.1 und IMS V9 COBOL Anwendungen, COBOL zu XML-mapping, COBOL XML Konverter und WSDL Erstellung behandelt.

Neben dem Zugriff auf CICS Anwendungen über WebSphere können CICS Anwendungen über das *Simple Object Access Protocol* (SOAP) auf den CICS Transaktionsserver V2 zugreifen. Der Nachrichtenaustausch der dabei verwendeten, in XML kodierten Nachrichten kann dabei über HTTP oder WebSphere MQ erfolgen. Durch die Verwendung von SOAP können so CICS-basierte Anwendungen als Web Services verfügbar gemacht werden, auf die Clients ohne einen dazwischen liegenden Anwendungsserver zugreifen können. Weitere Information bezüglich der *SOAP for CICS* Funktionalität ist auf [SA03] aufgeführt.

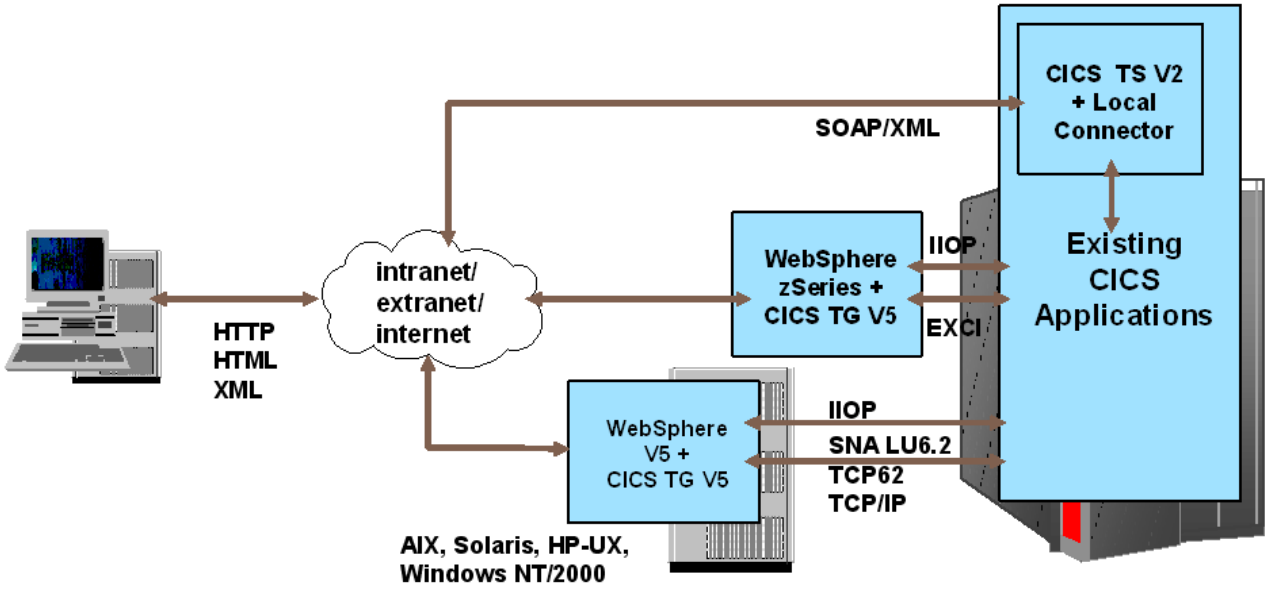

Abbildung 3.2.3: Übersicht externer Zugriffsmöglichkeiten auf CICS Anwendungen

Ähnlich kann das *Information Management System* (IMS) von IBM in eine SOA-Umgebung integriert werden. Für eine Ausführung über das *IMS SOAP Gateway* sei auf [IBM3] verwiesen.

XML Enablement for the Enterprise ermöglicht es COBOL-basierten Anwendungen, aus XML Nachrichten COBOL-Daten zu generieren, und andersherum aus COBOL-Daten XML Nachrichten zu erstellen. Die dabei verwendeten Konverter nutzen die hochperformanten XML Parserfähigkeiten des *IBM Enterprise COBOL Compilers* [IBM4].

WDz bietet mehrere Möglichkeiten, aus bestehendem Code Web Services zu generieren und diese mit dem integrierten *Web Services Explorer* zu testen. Weitere Ausführungen zu Web Services und deren Generierung mittels WDz finden sich in Kapitel 4 ab *4.4 Local Web Service* und den entsprechenden Tutorials.

3.2.3 BMS Map Support

Ein weiterer Teil von WebSphere Developer for zSeries ist die Erstellung und Bearbeitung des IBM *Basic Mapping Support*s (BMS). Der dabei verwendete Editor ermöglicht die Bearbeitung der BMS Maps über ein *Drag & Drop*-Verfahren. Dargestellt und bearbeitet werden können die BMS Maps in einem *Design View*, bei dem direkt in eine angezeigte BMS Map Teile eingefügt werden können, oder klassisch als *Source View* mit Sourcecode Bearbeitung. Es besteht die Möglichkeit, neue Map Sets zu erstellen oder Bestehende zu importieren. Alle Arbeiten können lokal oder remote ausgeführt, fertige Maps exportiert werden.

3.2.4 DB2 Stored Procedures – COBOL / PL/I

Mit WDz können COBOL und PL/I Stored Procedures unter z/OS erstellt, getestet und im Bedarfsfall auf der Workstation debugged werden. Für die automatisierte Generierung der SQL Definitionen und der COBOL und PL/I Stored Procedure Programme steht ein Wizard als Teil von WDz zur Verfügung.

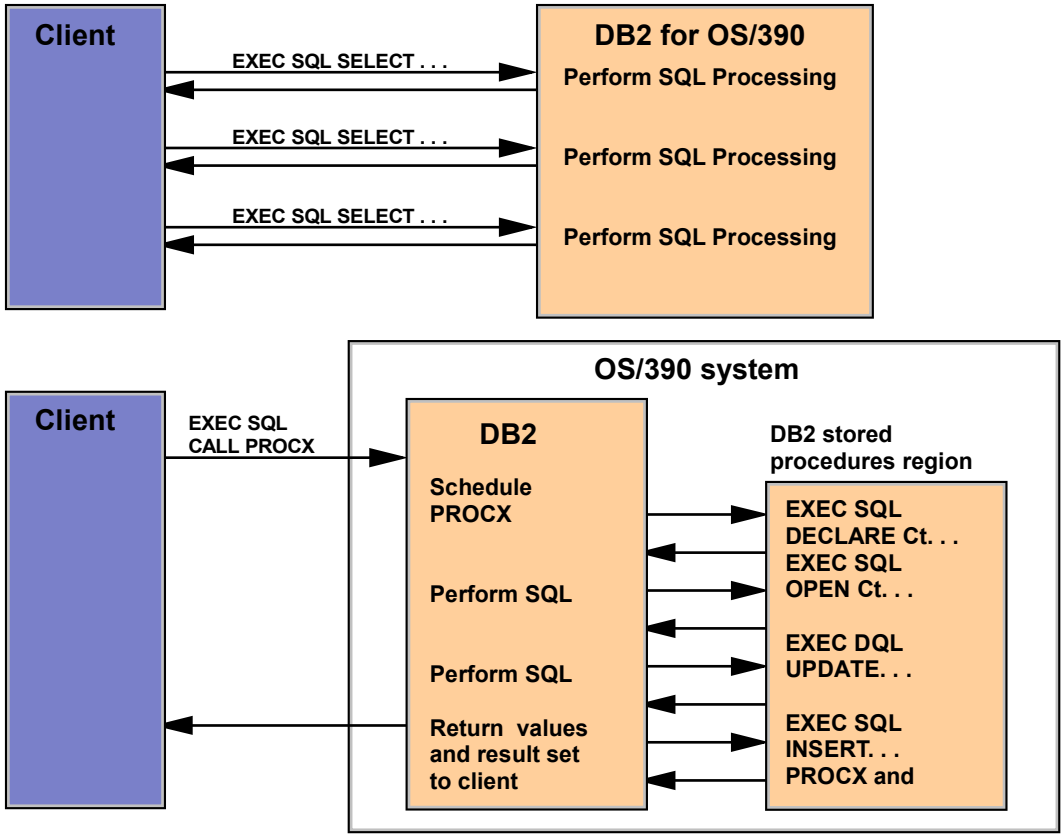

Abbildung 3.2.4: Geringer Netzwerkverkehr beim Einsatz von Stored Procedures

Einer der Nutzen der Verwendung von Stored Procedures ist in der obigen Abbildung dargestellt: an Stelle vieler einzelner Anfragen und Antworten wird eine einzelne Anfrage ausgeführt, die die auf dem Server liegende Stored Procedure aufruft. Bearbeitet wird die Anfrage bis zur Generierung eines Ergebnisses auf Serverseite, die Mitteilung dieses Ergebnisses an den Client geschieht wiederum in einer einzigen Nachricht. Für weiterführende Informationen bezüglich z/OS Stored Procedures sei [DB2SP06] empfohlen.

3.2.5 EGL COBOL Generation

Die *Enterprise Generation Language* (EGL) ist eine von IBM entwickelte Programmiersprache der vierten Generation. Verglichen mit Programmiersprachen der dritten Generation, zu denen unter Anderem Java und C++ gehören, treten die softwaretechnischen Aspekte in den Hintergrund. Der Entwickler konzentriert sich ausschließlich auf die Geschäftslogik und programmiert diese, EGL generiert daraus die benötigten Java- und/oder COBOL-Dateien. Entsprechend der Plattform, auf der das Programm ausgeführt werden soll, werden die erforderlichen Laufzeitartefakte automatisch angepasst. Für weiterführende Informationen bezüglich EGL sei [EGL] empfohlen.

Im Zuge des die Vorlesung Client/Server begleitenden Praktikums, für die die nachfolgenden Tutorials erstellt wurden, existieren zwei im Rahmen einer Masterarbeit erstellte Tutorials von Herrn Stefan Erras, die sich mit dem Thema WebSphere Developer for zSeries und EGL befassen, auf die an dieser Stelle verwiesen wird.

Kapitel 4

Tutorials

Im Folgenden wird eine Übersicht über die im Rahmen der Diplomarbeit erstellten Tutorials gegeben. Sie stellen einen Lehrgang in die Benutzung des WebSphere Developer for zSeries (WDz) dar und bieten Informationen zu den jeweils behandelten Themengebieten.

Da eine Darstellung der Tutorials in ihrer ganzen Länge zu umfangreich wäre, werden die jeweiligen Themen eingeführt und mit einigen Bildern aus den Übungen veranschaulicht. Die vollständigen Tutorials befinden sich im Anhang auf DVD.

4.1 WDz Tutorial 01 – Inbetriebnahme von WDz Version 6.0.1.2

In diesem Tutorial wird die Installation der auf zwei DVDs vorinstallierten Laufzeitumgebung beschrieben und eine erste Verbindung zum zSeries Mainframe in Leipzig aufgebaut. Das Tutorial ist wie die Übrigen mit einer Vielzahl an Screenshots versehen, um die einzelnen Arbeitsschritte zu veranschaulichen und die Nachvollziehbarkeit der Arbeit zu gewährleisten.

Die Laufzeitumgebung in der WebSphere Developer for zSeries gestartet wird ist eine auf Windows XP basierte Workstation, die auf einem Host, beispielsweise einem privaten PC, als virtuelles Gastsystem läuft. Als Virtualisierungssoftware dient *VMware* [VMware]. Diese Software ermöglicht es, eine einheitliche Arbeitsumgebung in Form eines vorinstallierten WindowsXP-Betriebssystems mit einem vorkonfigurierten Eclipse mit WDz-Plugin allen Praktikanten des Client/Server Praktikums zur Verfügung zu stellen. Um die virtuelle Maschine abspielen zu können wird die Software *Vmware Player* benötigt, welche als kostenloser Download für verschiedene Plattformen verfügbar ist und sich als Windows-Version auf einer DVD im Anhang befindet. Ein Vorteil dieses Verfahrens ist, neben der einheitlichen Arbeitsumgebung, die Möglichkeit, die virtuelle Maschine jederzeit neu aufsetzen zu können, ohne dabei Eingriffe in das Betriebssystem des Hosts machen zu müssen. Das Gastsystem ist dabei vollkommen vom Host isoliert und die Kommunikation untereinander geschieht ausschließlich über eine konfigurierte Netzwerkverbindung.

Die Übung ist in sieben Kapitel unterteilt: neben der Installationsanleitung für den Vmware Player finden sich Erläuterungen zur Anpassung der virtuellen Maschine, des WindowsXP Gastsystems, und Hinweise zur Netzwerkkonfiguration, auf die an dieser Stelle nicht näher eingegangen werden soll. Darauf folgt die Inbetriebnahme von WDz, die Verbindungserstellung zum zSeries Rechner *Padme* in Leipzig und weitere vorbereitende Maßnahmen für den erfolgreichen Ablauf der

restlichen Tutorials.

Da sich WDz vorinstalliert auf der virtuellen Maschine befindet, muss das Programm nur gestartet werden. Es wird empfohlen, anstelle des vorgegebenen Workspace einen Eigenen zu definieren, vorzugsweise wie im Tutorial beschrieben. Der Workspace definiert, an welchem Ort Projektdaten und dazu gehörige Metadaten gespeichert werden. Änderungen an dieser Stelle haben jedoch keine Auswirkung auf den restlichen Verlauf der Übungen, der Workspace definiert nur den Ort in unserer virtuellen Maschine, an der die unter WDz erstellten Arbeiten abgelegt werden. Wird WDz das erste Mal ausgeführt öffnet sich das Programm mit einem Welcome View:

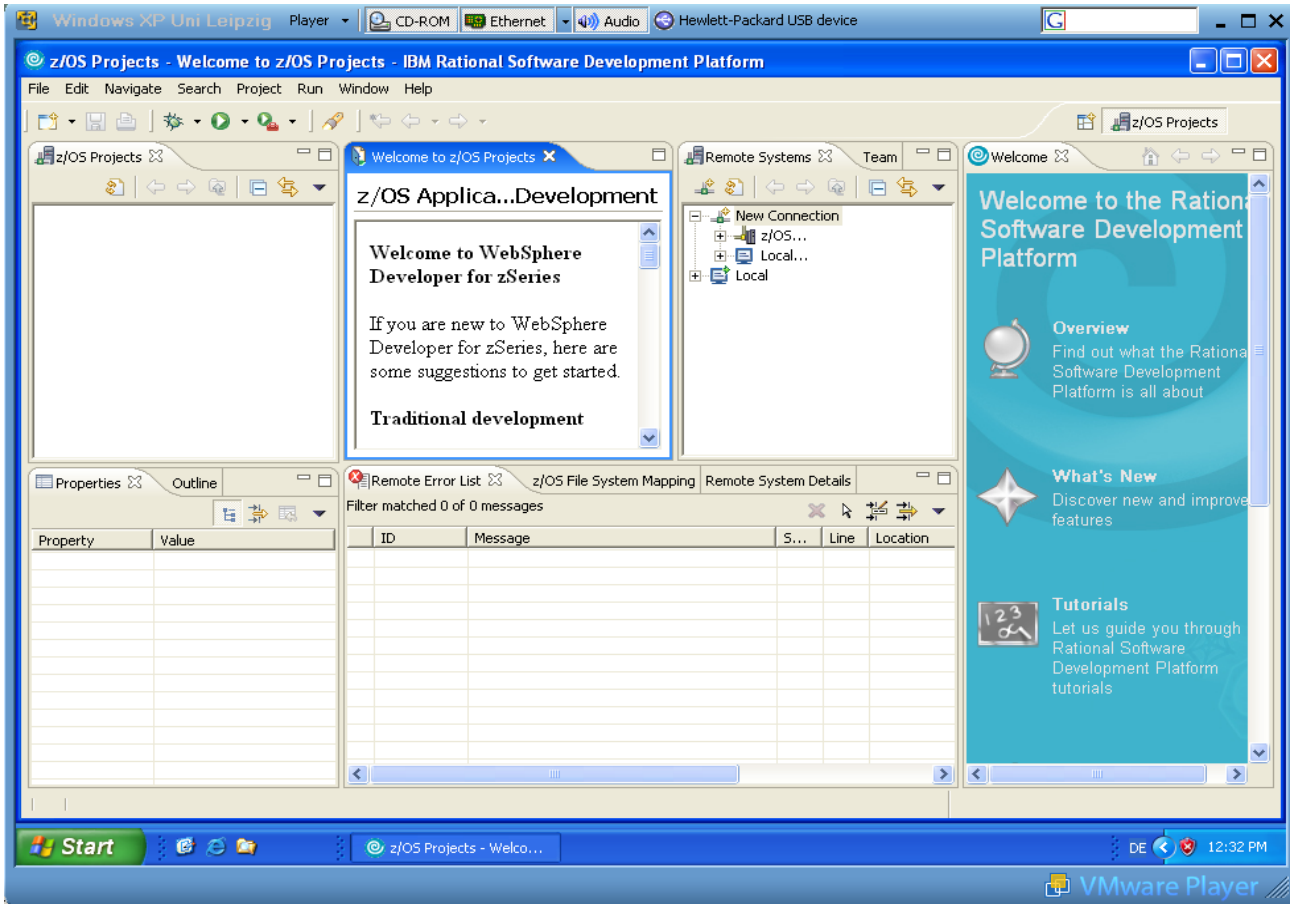

Abbildung 4.1.1: Erster Programmstart von WDz

WDz ist ab dieser Stelle voll Einsatzfähig und bedarf keiner weiteren Konfiguration.

Im nächsten Schritt wird die Verbindung zum z/OS-Rechner *Padme* in Leipzig mit der IP-Adresse 139.18.4.35 hergestellt. Um die Verbindung zu erstellen, wird der *New Connection* Assistent von WDz aufgerufen, wobei die Standardangaben übernommen werden können.

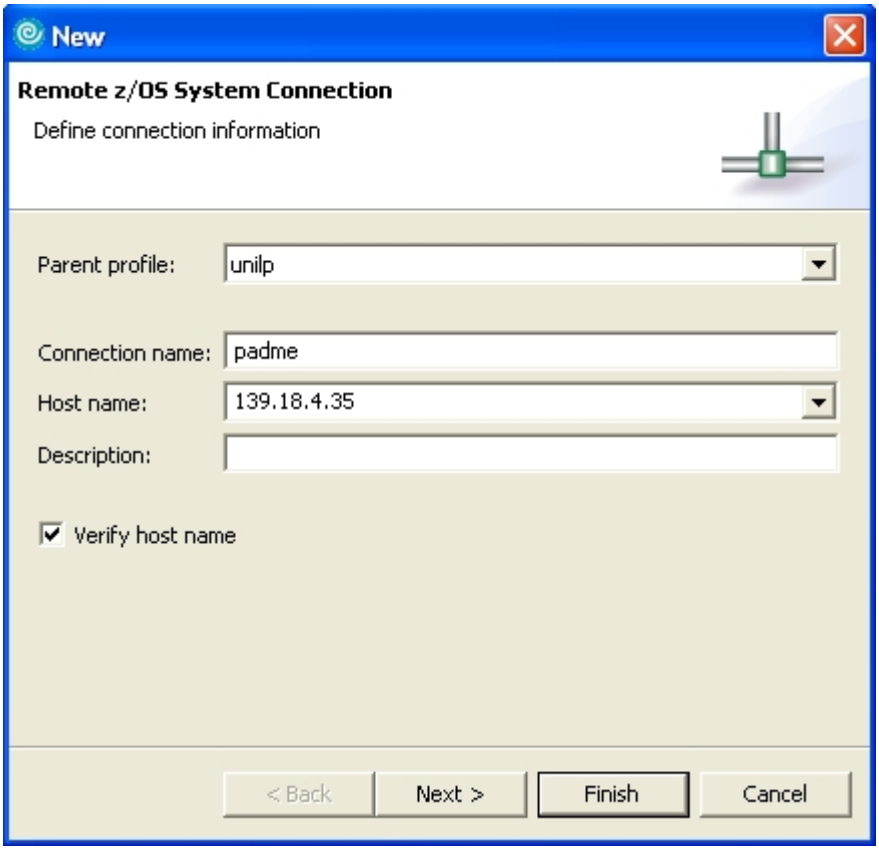

Abbildung 4.1.2: Erstellen einer neuen Verbindung

Für einen erfolgreichen Verbindungsaufbau ist eine im Voraus vom Praktikumsleiter vergebene Benutzerkennung mit Passwort erforderlich.

Abbildung 4.1.3: Verbindungsaufbau mit WDz

Es folgen einige Hinweise zum Aufbau der *Remote Systems*-Ansicht und eine Anleitung zum ordnungsgemäßen Trennen der Verbindung. Dies ist wichtig, da man sonst weiter als Benutzer auf dem zSeries Rechner eingeloggt ist und eine automatische Trennung erfolgen muss, was einige Zeit in Anspruch nimmt und währenddessen kein erneutes Einloggen mit derselben Benutzerkennung möglich ist.

Im nächsten Schritt gibt es eine Anleitung für das kopieren von Daten von DVD auf die virtuelle Maschine. Die kopierten Daten, die nun lokal auf dem Gastsystem verfügbar sind, werden in späteren Übungen benötigt.

Das Tutorial schließt mit der Vorgehensweise zur Beendigung des Gastsystems, welches wie ein reguläres Windows XP Betriebssystem heruntergefahren wird.

4.2 WDz Tutorial 02 – Local Cobol

Schwerpunkt ist die Benutzung von WDz am Beispiel eines COBOL Programms, das lokal auf dem Gastsystem bearbeitet und ausgeführt wird.

Das Erste der zehn Kapitel der Übung gibt eine Übersicht über einige generelle Aspekte von WDz. In Teil zwei wird lokal ein Beispielprogramm geladen, welches ein Bestandteil der vorkonfigurierten Arbeitsumgebung ist. Das Programm ist eine COBOL Applikation und wird in ein Projekt integriert, welches neu erstellt werden muss. Weiter werden in diesem Kapitel Kompilier- und Linkoptionen für das importierte COBOL-Programm konfiguriert.

Teil drei befasst sich mit dem Editor und dessen Interaktion mit weiteren in der Arbeitsoberfläche integrierten Ansichten wie dem *Properties view* oder dem *Outline view*, die die im Editor dargestellten Informationen entsprechend ihrer Aufgaben präsentieren. So wird im Outline view die Struktur des COBOL Programms dargestellt. Durch Anwählen einzelner Punkte des Outline views kann man im Sourcecode, der im Editor angezeigt wird, navigieren. Umgekehrt werden bei Navigation im Editor automatisch die entsprechenden Abschnitte im Outline view angezeigt.

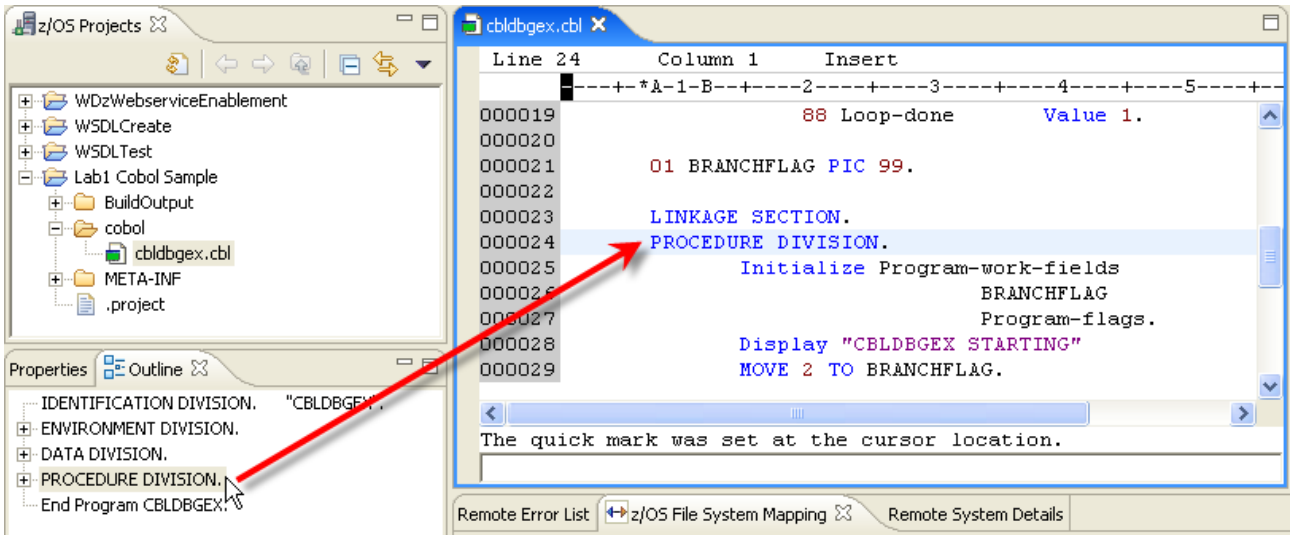

Abbildung 4.2.1: Navigation im Editor mittels der Outline Ansicht

Es werden weitere Editorfunktionen wie die Suchfunktion erklärt und Einstellungsmöglichkeiten vorgeführt, um die Eigenschaften des Editors auf das Verhalten des Editors der *Interactive System Productivity Facility* (ISPF) anzupassen, dem Standardeditor für Mainframes. Tabulatorabstände, Editor spezifische Präfixkommandos, Max- und Minimieren des Editorfensters, Editieren und Speichern der Änderungen sind zusätzlich Themen dieses Abschnitts.

Paragraph vier befasst sich mit der Benutzung einer Option von Eclipse, der *Local History,* die Teil der vorinstallierten Konfiguration ist. Bei jedem Speichervorgang wird eine Kopie des Speichervorgangs in der *Local History* abgelegt. Dies ermöglicht es zu jedem Zeitpunkt, aktuelle Zustände mit Früheren zu vergleichen und Änderungen falls nötig rückgängig zu machen, oder aktuelle Versionen durch ältere zu ersetzen.

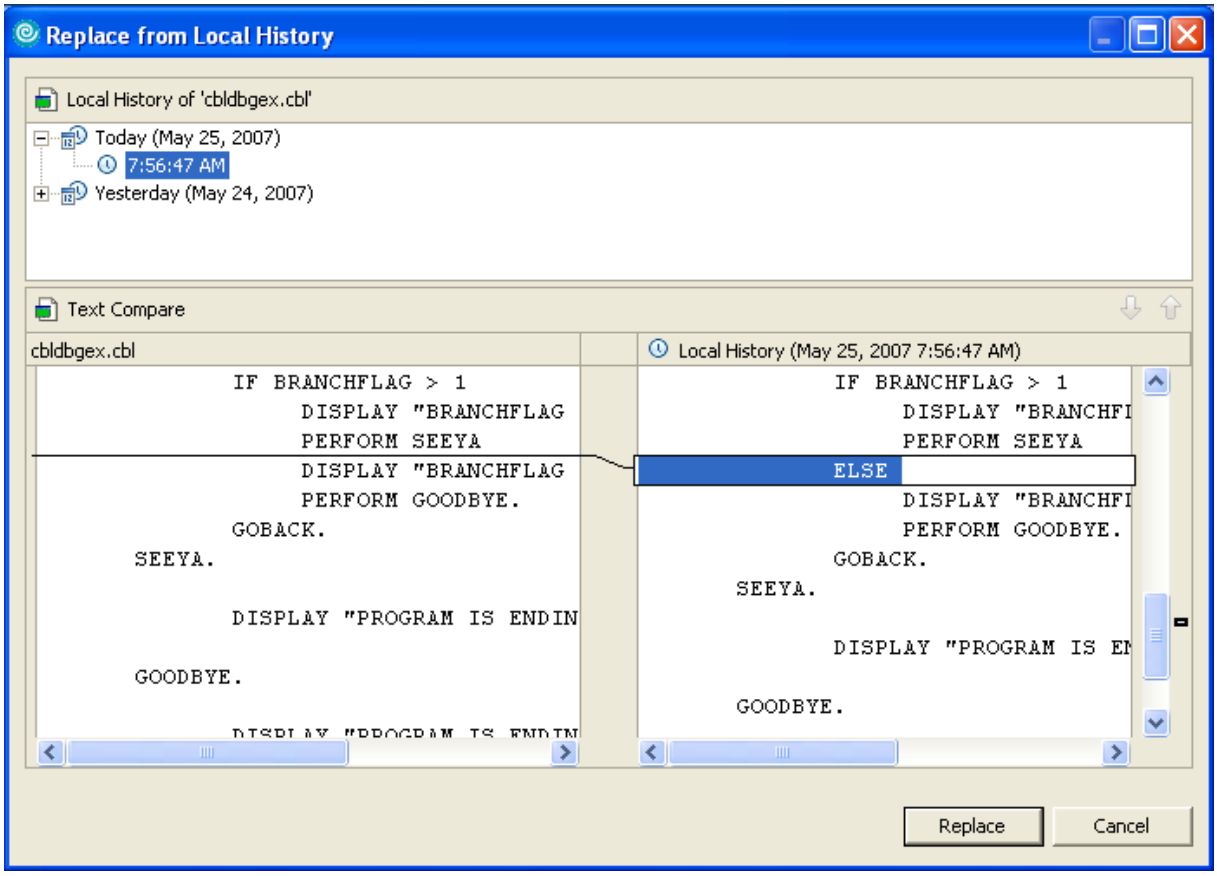

Abbildung 4.2.2: Benutzung der Local History Funktionalität

In Abschnitt fünf wird der Quellcode des COBOL Programms auf Fehler geprüft und das Zusammenspiel zwischen Editor und der *Remote Error List* Ansicht vorgeführt. Ein im vorhergehenden Kapitel absichtlich eingebauter Fehler wird hier erkannt und behoben, das korrigierte Programm gespeichert und erneut geprüft.
Im folgenden Bild wird die Navigation mittels der Remote Error List dargestellt: ein Doppelklick auf den angezeigten Fehler der Remote Error List setzt den Cursor automatisch auf die Zeile im Editor, in der der Fehler im Sourcecode erkannt wurde.

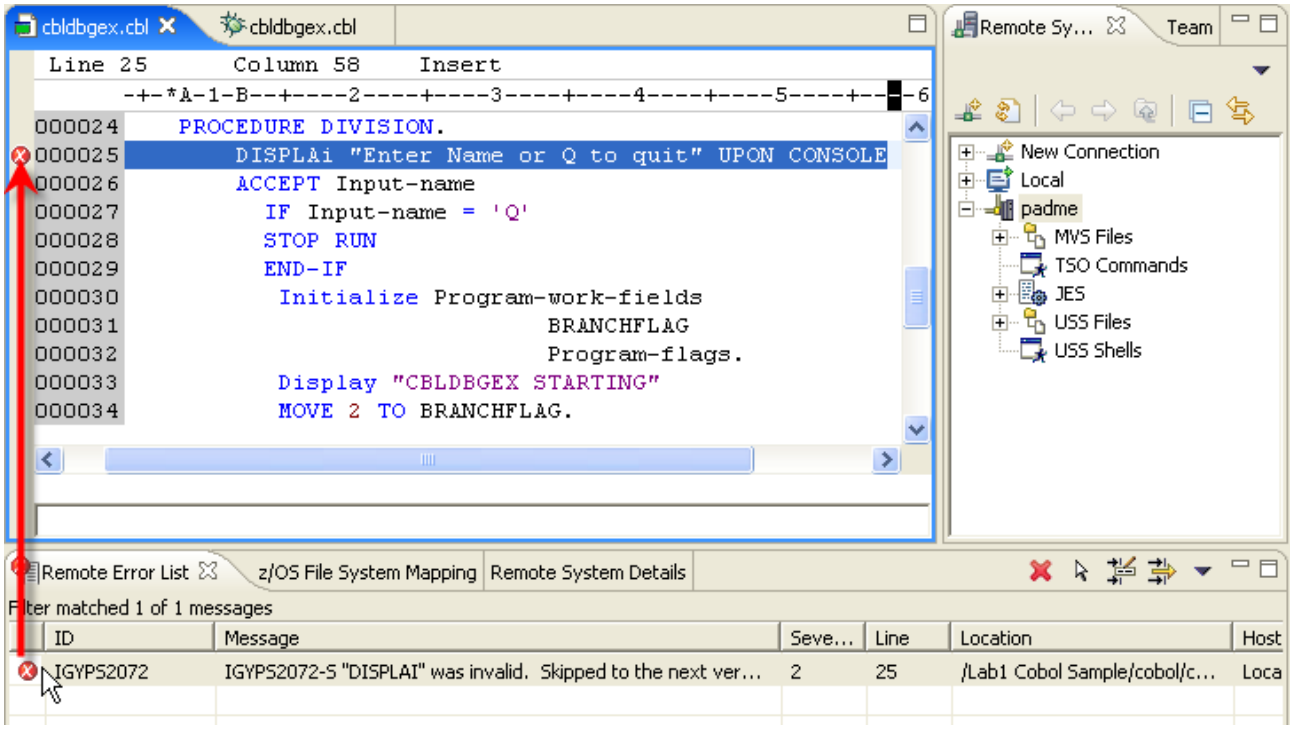

Abbildung 4.2.3: Benutzung der Remote Error List

Im nächsten Kapitel werden Änderungen im COBOL Quelltext vorgenommen, wobei Editorfunktionen vorhergehender Kapitel vertieft werden. Zusätzlich wird das Arbeiten mit dem so genannten *Code Assist* eingeführt. Code Assist zeigt alle COBOL Schlüsselwörter an, die an der Stelle, an der Code Assist aufgerufen wird, möglich sind. Dabei werden auch Variablen einbezogen, die im Quelltext oder verbundenen Quelltexten vorkommen und an der Eingabestelle möglich sind. Anschließend wird der Quelltext auf Fehler geprüft und die Syntax bezogene Hilfefunktion vorgestellt.

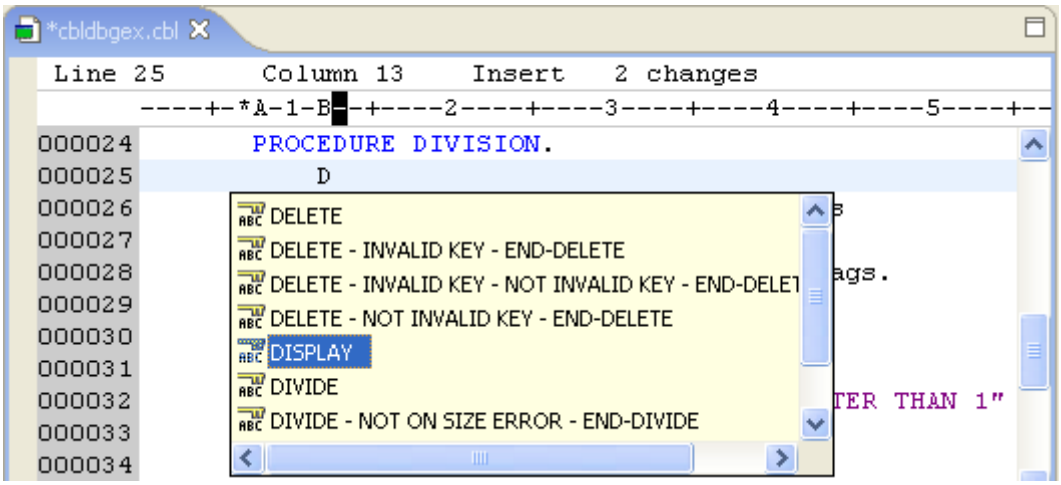

Abbildung 4.2.4: Verwendung von Code Assist

Bevor in Kapitel sieben das COBOL Programm kompiliert werden kann, muss die dem Beispielprojekt automatisch hinzugefügte, vorkompilierte Version gelöscht werden. Nachdem der COBOL Quelltext als Einstiegspunkt der Kompilation festgelegt wurde, kann das Projekt neu kompiliert werden. Der in diesem Zusammenhang gelöschte Ordner *BuildOutput* wird neu erstellt, zusammen mit dem ausführbaren Programm. Der anschließende Paragraph acht verifiziert die

fehlerfreie Kompilation anhand der beim Kompilationsvorgang erstellten Logdateien.

Kapitel neun behandelt die Konfiguration der Ablauf- und Debugumgebung. Dabei wird über ein in WDz integrierten Wizard definiert, was beim Start des kompilierten COBOL Programms in der WDz Laufzeitumgebung geschieht. Im Anschluss daran wird das kompilierte Programm zum ersten Mal ausgeführt.

Das abschließende zehnte Kapitel beschäftigt sich mit dem Debuggen des Beispielprogramms anhand der davor konfigurierten Debugumgebung. In diesem Zug wird die *Debug Perspective* aktiviert, die umfangreiche Werkzeuge für die Fehlererkennung und -behebung zur Verfügung stellt. Einige davon, wie das Setzen von *Breakpoints* und *Watchpoints* und das Abändern von Inhalten von Variablen werden im Laufe der Übung vorgestellt.

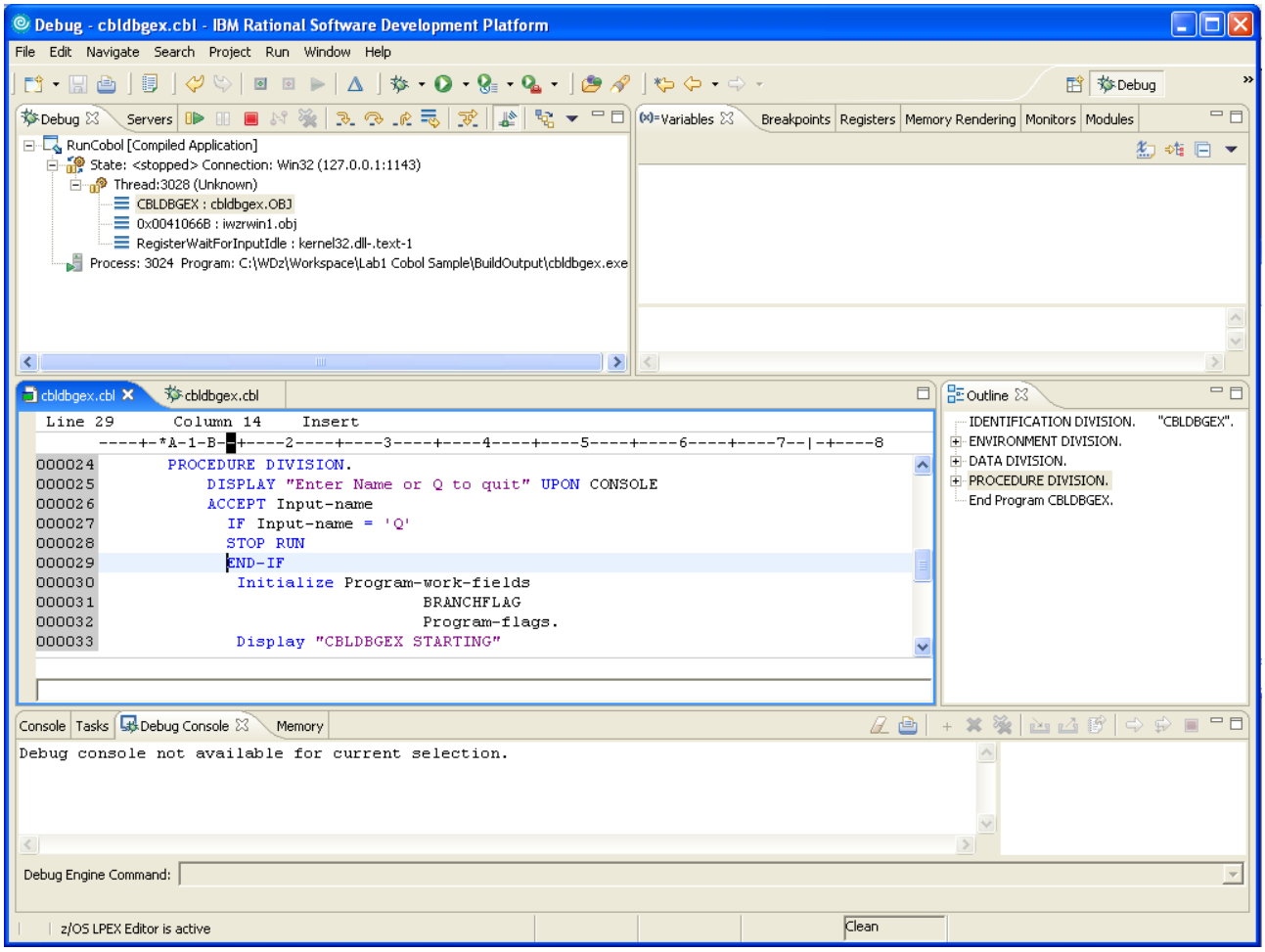

Abbildung 4.2.5: Startbildschirm der Debug Perspective

4.3 WDz Tutorial 03 – Remote Cobol

Dieses Tutorial verwendet die in WDz enthaltenen Werkzeuge zur z/OS Anwendungsprogrammierung, um ein COBOL Programm serverseitig zu kompilieren und auszuführen.

Im ersten Kapitel wird anhand ausführlicher Screenshots die Vorbereitung der Arbeitsoberfläche und die Erstellung einer Verbindung zum zSeries Großrechner in Leipzig dargestellt.

In Paragraph zwei wird eine *Job Control Language* (JCL)-Datei, welche in Tutorial 01 auf unser lokales Gastsystem kopiert wurde, in WDz geöffnet und bearbeitet. Dabei werden die im vorhergehenden Tutorial eingeführten Editorfunktionen verwendet. Das bearbeitete JCL-File wird gespeichert und mit dem *submit*-Befehl an den Server zur Ausführung gesendet.

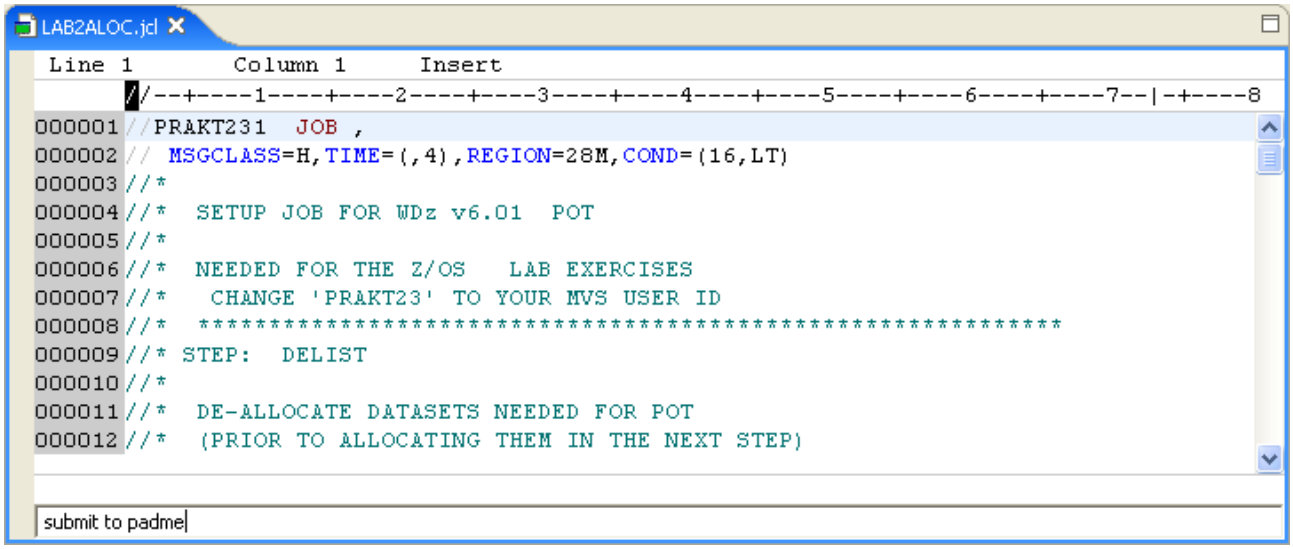

Abbildung 4.3.1: Ausführung eines JCL-Scripts auf dem Server via submit

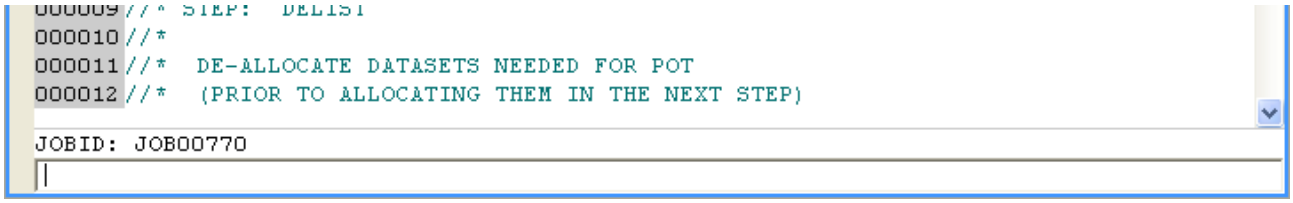

Abbildung 4.3.2: Wird eine JOBID vergeben, hat der Server das JCL-Script zur Verarbeitung angenommen

Weitere Schritte dieses Abschnitts sind die Auswertung der vom Server im *Job Entry System* (JES) generierten Logfiles und ein anschließendes Aufräumen des JES.

Im nächsten Teil werden vorgegebene z/OS Systemeigenschaften in den in Kapitel eins erstellten WDz Workspace importiert und im Einzelnen angepasst. Besprochen wird dabei die Angleichung der *JCL Job Card* an die Benutzerkennung und ein Blick auf die COBOL Einstellungen, wobei an dieser Stelle die Standardeinstellungen beibehalten werden.

In Kapitel vier wird ein *Partitioned Data Set* (PDS) mit den in WDz verfügbaren Werkzeugen erstellt und mit *Membern* gefüllt, die sich lokal auf dem Gastsystem befinden. Das Erstellen eines

PDS wurde im allerersten z/OS Tutorial des Client/Server Praktikums eingeführt und an dieser Stelle erneut aufgegriffen, um denselben Arbeitsschritt in einer modernen Arbeitsumgebung zu illustrieren. Einige der Eingabemasken wie Abbildung 4.3.3 sind den Praktikanten aus den TSO Eingabemasken bekannt.

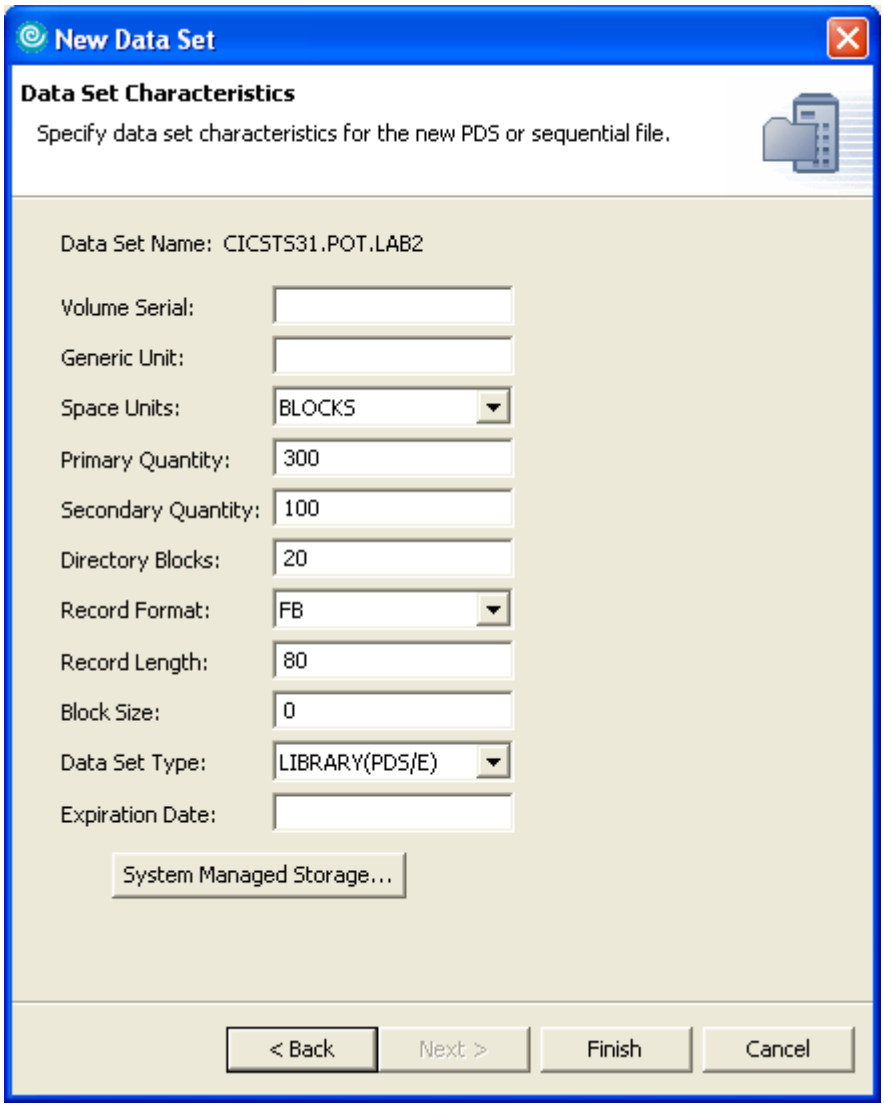

Abbildung 4.3.3: Definieren der Eigenschaften eines Partitioned Data Sets

Ein weiterer Schwerpunkt dieses Kapitels ist die Einführung in das *z/OS File System Mapping*, einem Werkzeug von WDz, welches es der Workstation ermöglicht, die zunächst unbekannten Großrechnerdateien zu verarbeiten. Hierbei werden Teile der Namen der PDS und ihrer Member als definierende Elemente benutzt. Wie als Beispiel in Abbildung 4.3.4 gezeigt, werden standardmäßig alle Dateien der Datasets, deren Name mit *LAB2* endet (einen Schritt davor wurde ein PDS mit dieser Endung angelegt), als *bms* (Basic Mapping Support) dargestellt. Ausnahme sind Member dieser Datasets, die mit *REGI* beginnen. Diese werden als *cbl*, also als COBOL-Datei, dargestellt.

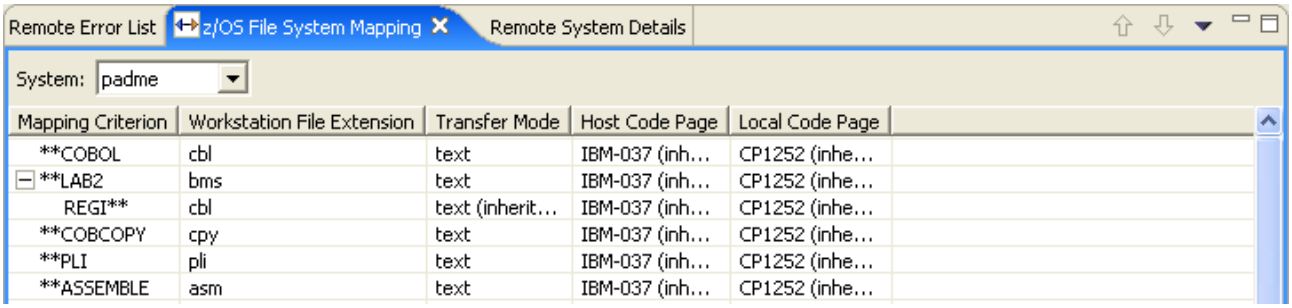

Abbildung 4.3.4: z/OS File System Mapping

In den erzeugten Dataset mit der Endung LAB2 werden drei COBOL-Programmstücke von der Workstation kopiert, mit denen später in der Übung weiter gearbeitet wird. Da diese Dateien jeweils mit den Buchstaben REGI* beginnen und in einem Dataset mit der Endung *LAB2 liegen, werden sie als COBOL-Dateien erkannt und können als solche von WDz verarbeitet werden.

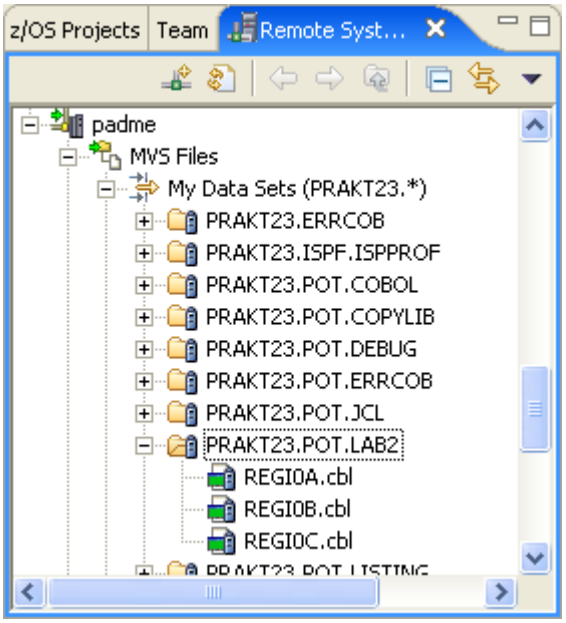

Abbildung 4.3.5: Mapping der REGI-Dateien als COBOL-Files*

MVS-Projekte sind das zentrale Thema des fünften Kapitels. Ein MVS-Projekt beinhaltet die Datasets und Member, auf denen im Laufe des Projekts der Fokus liegt, ohne eine unübersichtliche Menge an Dateien verwalten zu müssen. Ressourcen können im Verlauf der Arbeit dem Projekt hinzugefügt oder davon entfernt werden. Zur Ansicht weiterer Datasets oder Member, die nicht Teil des MVS-Projekts sind, steht der Remote Systems View zur Verfügung. Ausgehend von diesem werden im zweiten Teil des Kapitels Ausgewählte Datasets dem Projekt hinzugefügt.

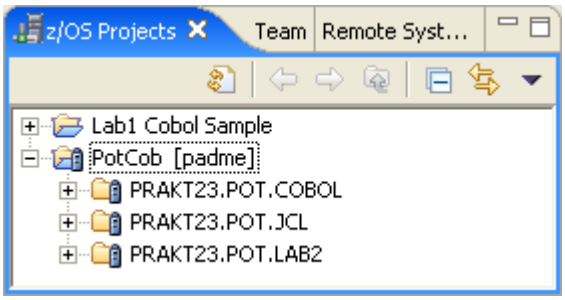

Abbildung 4.3.6: Erstelltes MVS-Projekt „PotCob“ mit Ressourcen

In Kapitel sechs wird ein COBOL-File, das ein Member eines auf dem Mainframe liegenden Datasets ist, lokal bearbeitet, wobei bereits eingeführte Editorfunktionen verwendet werden. Das veränderte Member wird remote gespeichert. Anschließend werden automatisiert JCL-Files generiert, mit denen die auf dem Großrechner gespeicherten COBOL-Dateien kompiliert, verlinkt und ausgeführt werden können. Als Veranschaulichung, dass die Dateien wirklich auf dem Server und nicht nur lokal erstellt wurden, wird eine weitere Funktion von WDz eingeführt, der so genannte *Host Connection Emulator Support*. Damit kann, während WDz weiter mit dem Großrechner verbunden ist, via 3270-Emulator auf den Mainframe eingeloggt und sich die erstellte JCL-Datei angeschaut werden. Nach erfolgreicher Ausführung der JCL-Skripte und der daraus resultierenden Ausführung des COBOL Programms wird eine Ausgabe generiert, auf der eine bei der Bearbeitung der COBOL-Datei veränderte Zeile dargestellt wird (siehe Abbildung 4.3.7).

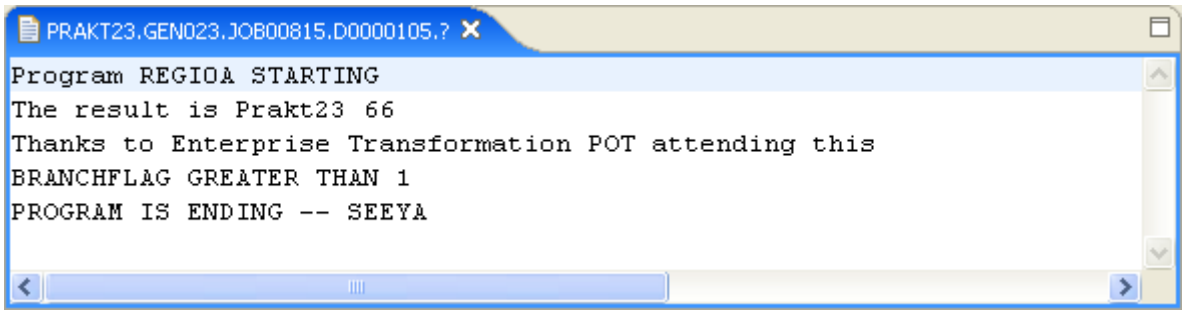

Abbildung 4.3.7: Ausgabe bei erfolgreicher Programmausführung

4.4 WDz Turorial 04 – Local Web Service

Schwerpunkt des Tutorials ist die Einführung in Web Services und wie Web Services mit WDz erstellt und bearbeitet werden können.

Die Übung beginnt mit einer Einführung in das Thema Web Service und die dazu gehörigen Protokolle und Techniken. Eine Definition zu Web Services findet sich auf der Seite des World Wide Web Consortiums[WSgloss04] und sei an dieser Stelle zitiert:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Der Standard für Web Services, der auch in der Übung Verwendung findet, ist ein in der *Web Service Description Language* (WSDL) verfasstes Dokument. Darin werden die vom Web Service benutzten Funktionen, Daten, Datentypen und Protokolle beschrieben. Ein Client, dem ein solches WSDL-Dokument zur Verfügung steht, kann mit Hilfe der darin enthaltenen Informationen den angebotenen Web Service aufrufen. WSDL Dokumente werden im Regelfall von einem Serviceanbieter einem Verzeichnisdienst zur Verfügung gestellt, der damit ein dynamisches Auffinden des Web Service auf Anfrage eines Clients ermöglicht. Der Verzeichnisdienst, der Standard hierfür ist *Universal Description, Discovery and Integration* (UDDI), kann Auskünfte öffentlich oder in definiertem Rahmen, zum Beispiel in dem Intranet einer Firma, anbieten [UDDI]. Möglich ist auch der direkte Versandt eines WSDL Dokuments über eMail, laden der WSDL Datei über FTP oder ähnliche Verbreitungsmöglichkeiten.

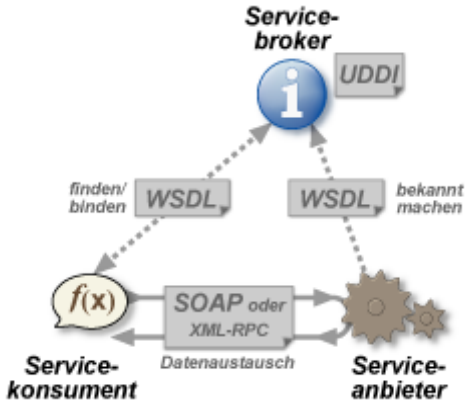

Abbildung 4.4.1: Kommunikationsschema von Web Services [Wik2]

WSDL bedient sich bei der Beschreibung seiner Inhalte der XML-Spezifikation (*Extensible Markup Language*, eine Definition hierfür findet sich bei [XML]). In einem WSDL-Dokument werden folgende sieben Hauptmerkmale definiert:

- **Datentypen:** Definition der beim Nachrichtenaustausch verwendeten Datentypen
- **Nachrichten:** Abstrakte, getypte Definition der Daten die versandt werden. Kann mehrere verschiedene getypte Teile enthalten.
- **Porttypen:** Abstrakte Beschreibung einer oder mehrerer Operationen eines oder mehrerer

Ports
- **Operationen:** Abstrakte Beschreibung der Vorgehensweise beim Nachrichtenaustausch (One-Way, Request-Response usw.)
- **Bindung:** Konkretes Protokoll und Datenformat für bestimmte Porttypen
- **Port:** Legt einen bestimmten Endpunkt fest
- **Services:** Zusammenfassung verwandter Ports

WSDL wird meistens zusammen mit *SOAP* und *XML Schema Definitionen* (XSD) verwendet, wobei SOAP das Netzwerkprotokoll für den Datenaustausch bereit stellt und XSD das Datenlayout der ausgetauschten Nachrichten definiert.

SOAP stand ursprünglich für *Simple Object Access Protocol*, wird jedoch seit Version 1.2 als eigenständiges Akronym verwendet. Unter dem Einsatz von SOAP als Netzwerkprotokoll werden bei Ausführung von Web Services Daten zwischen Client und Server ausgetauscht und Remote Procedure Calls ausgeführt. Dabei werden vorhandene Standards verwendet: XML zur Repräsentation der Daten und gängige Internetprotokolle zur Übertragung der Nachrichten, am häufigsten mit HTTP über TCP/IP [SOAP].

XML Schema Definitions definieren XML Dokumentstrukturen und sind selbst in XML verfasst. XSDs geben eine Reihe an Regeln vor, die ein XML Dokument einhalten muss, um als gültig eingestuft zu werden. Durch die Einhaltung dieser definierter Regeln ist es möglich, von Programmen generierte und in XML codierte Nachrichten zu versenden, die von einem Empfänger decodiert werden können, unabhängig der Plattform auf der die Nachricht generiert oder empfangen wurde. Für eine Einführung in XSD sei [XSD04] empfohlen.

Nach der Einführung wird in Kapitel eins die Anzahl der im XML Editor angezeigten Zeichen pro Zeile auf 100 erhöht. Dies wirkt Problemen mit der WSDL Spezifikation entgegen, welche keine Zeilenumbrüche erlaubt, die bei der Standardlänge von 72 Zeichen pro Zeile jedoch auftreten würden. Eine andere Möglichkeit um dies zu verhindern wäre die Benutzung kürzerer Namen von Variablen. Inhaltlich wird durch diesen Eingriff nichts an XML Dokumenten verändert.

In Kapitel zwei wird, nachdem die Ansicht auf den *Resource View* umgestellt wurde, ein neues Projekt namens *WSDLCreate* erstellt. Im Anschluss daran werden im dritten Teil XML Schema Definitionen generiert, welche den Datenstrukturen des verwendeten COBOL-Programms und des dazu gehörenden Copybooks entsprechen. Damit WDz dies über einen Wizard erledigen kann, werden das später verwendete COBOL Programm und das dazu gehörige Copybook in das erstellte Projekt importiert. (FUNDFILE.cbl agiert in diesem Fall als Copybook. Copybooks entsprechen dem *include*-Ausdruck in C/C++ Programmen.)

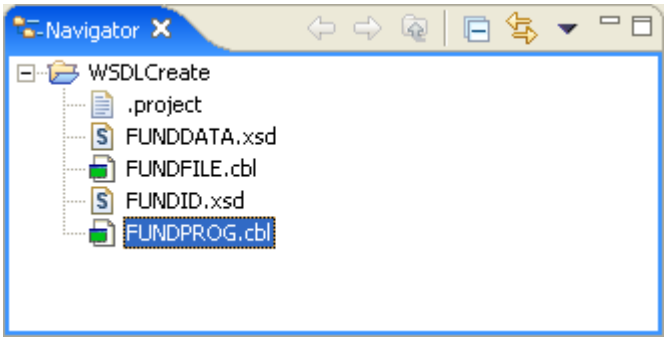

Abbildung 4.4.2: Projektansicht mit generierten XSDs

Am Ende dieses Kapitels wird verifiziert, ob in WDz die benötigten Rollen aktiviert sind. Benutzer können unter WDz verschiedene Rollen aktivieren oder deaktivieren, um die mit den Rollen verbundenen Einstellungsmöglichkeiten des Programms verfügbar zu machen oder unnötige Anzeigen auszublenden. Für den weitere Verlauf der Web Services Tutorials ist die Aktivierung der Rolle *z/OS Modernisation Developer (advanced)* nötig, da sonst entsprechende Funktionen nicht zur Verfügung stehen.

Abbildung 4.4.3: Aktivierung der z/OS Modernisation Developer Rolle

Im vierten Kapitel wird ein leeres WSDL-Dokument erstellt. Im weiteren Verlauf der Übung wird das leere Dokument mit Inhalten gefüllt.

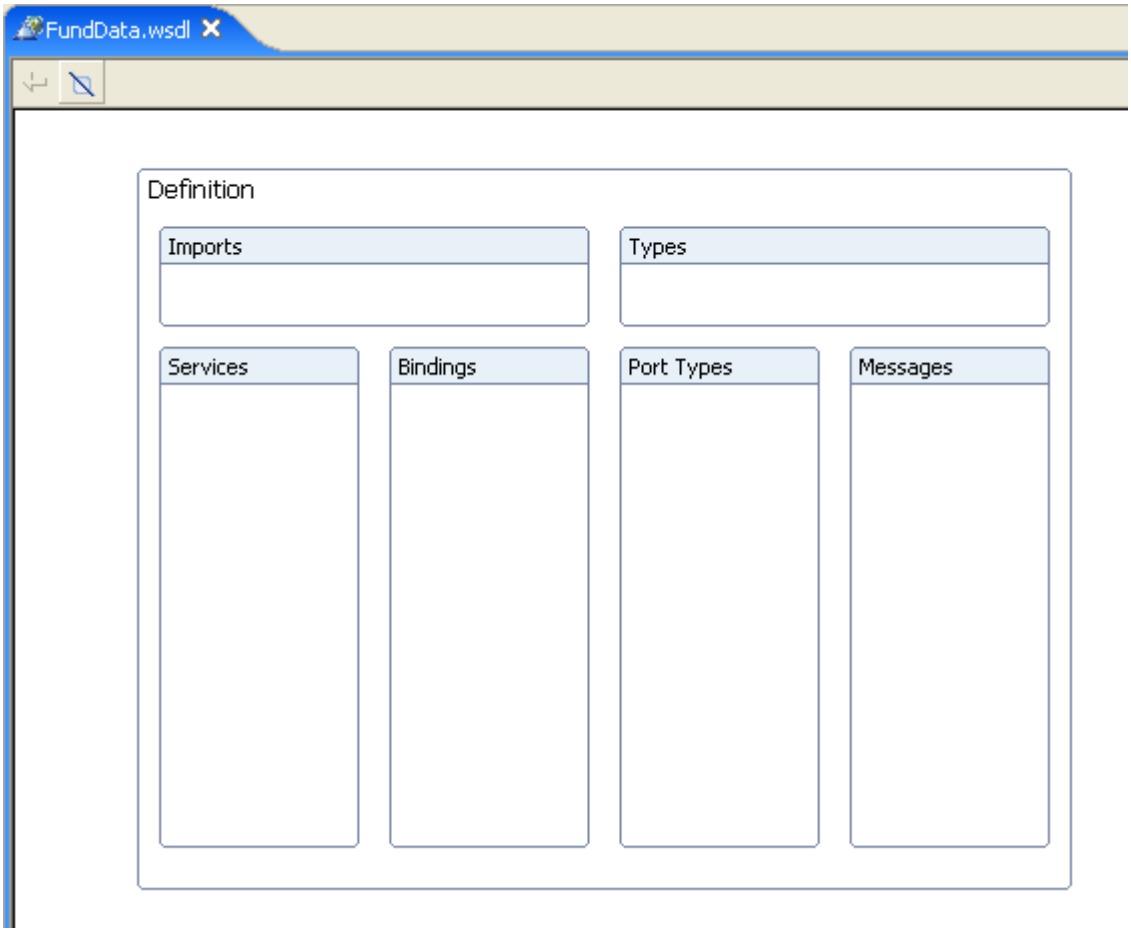

Abbildung 4.4.4: Erstelltes WSDL Dokument ohne Inhalte in Graphenansicht

Als nächstes wird die *addFund*-Operation im fünften Teil des Tutorials hinzugefügt. Sie enthält ein- und ausgehende Nachrichten, die auf die XSDs verweisen, die in einem vorangehenden Abschnitt erstellt wurden. Um die addFund-Operation hinzufügen zu können wird ein *Port Type* benötigt, welcher ebenfalls in diesem Kapitel erstellt wird.

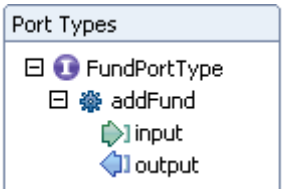

Abbildung 4.4.5: addFund-Operation mit Port

Im nächsten Schritt wird eine Nachricht dem WSDL-File hinzugefügt, die ein in Teil drei generiertes XSD enthält. Abbildung 4.4.6 zeigt die *FundData*-Nachricht mit der in dem XSD-File definierten Struktur.

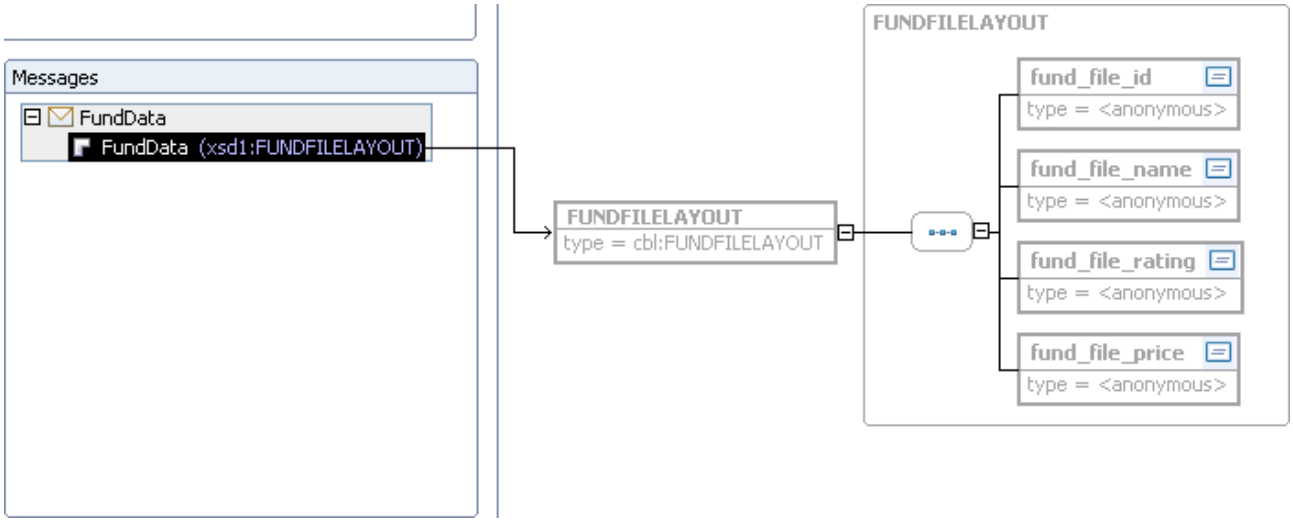

Abbildung 4.4.6: Erstellte Nachricht mit der Struktur des verwendeten XSD-Files

Die ein- und ausgehenden Nachrichten der addFund-Operation werden mit der erstellten FundData-Nachricht verknüpft, das Projekt gespeichert und auf Korrektheit geprüft.

Thema von Paragraph sieben ist die Erstellung einer Fehlernachricht, die Informationen zurück gibt falls die addFund-Operation fehl schlägt. Die dazu gehörende Nachricht wird von WDz automatisch generiert und verknüpft.

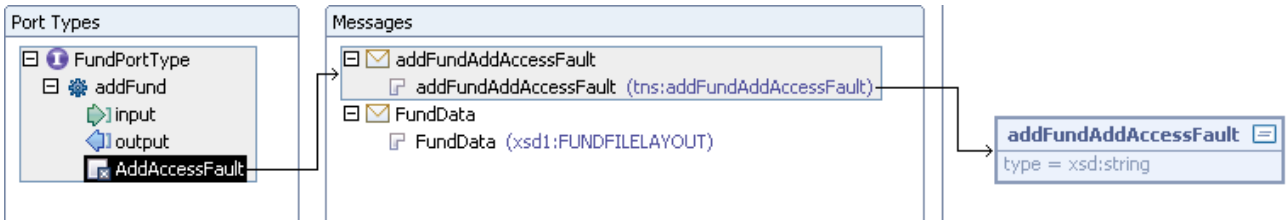

Abbildung 4.4.7: Generierte Fehlerroutine für die addFund-Operation

Die bisher beschriebenen Port Typen, Operationen und Nachrichten sind allgemein gehalten. Um diese als Web Services kenntlich zu machen wird im achten Kapitel die Bindung SOAP über HTTP als Protokoll definiert. Für das Einrichten der Bindung steht unter WDz ein Wizard zur Verfügung.

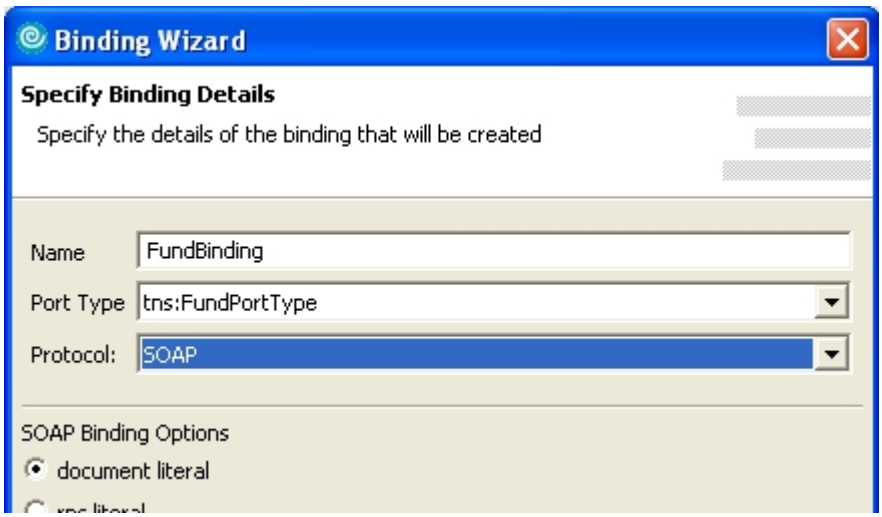

Abbildung 4.4.8: Binding Wizard

Um im letzten Schritt in Kapitel neun zu definieren, wo sich der Web Service befindet, wird ein Service namens *FundData* und für diesen Service ein Port erstellt. Der im dem Port definierte Endpunkt ist fiktiv, da die Erstellung des WSDL-Files zur Übung dient und nicht zum Einsatz kommt. Unter realen Bedingungen müsste hier die URL und der Port angegeben werden, an dem sich die Applikation, die den Web Service anbietet, befindet.

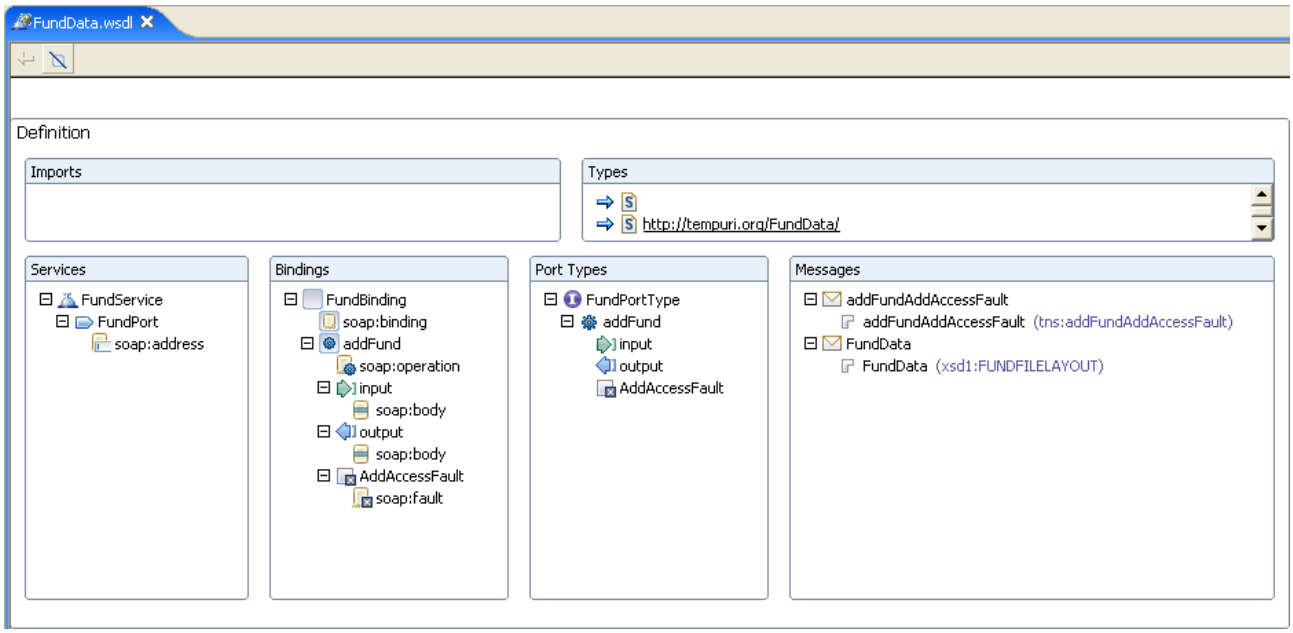

Abbildung 4.4.9: Fertiges WSDL Dokument in Graphenansicht

Die abschließenden Kapitel zehn und elf bieten Vorschläge zur weiteren Erstellung von WSDL-Dateien, um die erworbenen Kenntnisse zu vertiefen, und eine Zusammenfassung der in dem Tutorial vorgestellten Arbeit.

4.5 WDz Tutorial 05 – Web Service Explorer

In diesem Tutoral wird ein gegebener Web Service mit Hilfe des *Web Services Explorer* und des *TCP/IP Monitos*, beides in WDz enthaltene Werkzeuge, getestet.

Zu Beginn wird geprüft, ob benötigte Rollen in WDz aktiviert sind. Mit verschiedenen Rollen sind in WDz unterschiedliche Einstellungen verbunden, ohne deren Aktivierung die Übung nicht durchgeführt werden kann.

Kapitel eins gibt eine Übersicht über diverse Möglichkeiten der Verbreitung von WSDL-Files. Neben dem Versand als Anlage einer eMail können die Daten über verschiedenste Arten der Dateiübertragung, wie zum Beispiel unter Benutzung von FTP, verbreitet werden. Weiter stehen diverse Seiten im Internet zur Verfügung, auf denen Web Services kostenlos oder gegen Gebühr angeboten werden. Mit der von IBM und Microsoft gemeinsam entwickelten Spezifikation *Web Services Inspection* (WS-Inspection) steht eine Möglichkeit zur Verfügung, Web Server nach Web Services zu durchsuchen [WSIL]. Abschließend steht mit *Universal Description, Discovery and Integration* (UDDI) ein Verzeichnisdienst zur Verfügung, welcher Web Services auflistet und der nach bestimmten Kriterien durchsucht werden kann [UDDI]. Am Ende des Kapitels wird in WDz die *Resource Perspective* geöffnet und ein Projekt namens *WSDLTest* erstellt.

Teil zwei der Übung fügt dem angelegten Projekt eine WSDL Datei hinzu, die lokal auf dem Gastsystem gespeichert ist. Das zur Verfügung stehende WSDL-File greift auf einen Web Service, der Teil eines in CICS integrierten Beispielprogramms ist, zu.

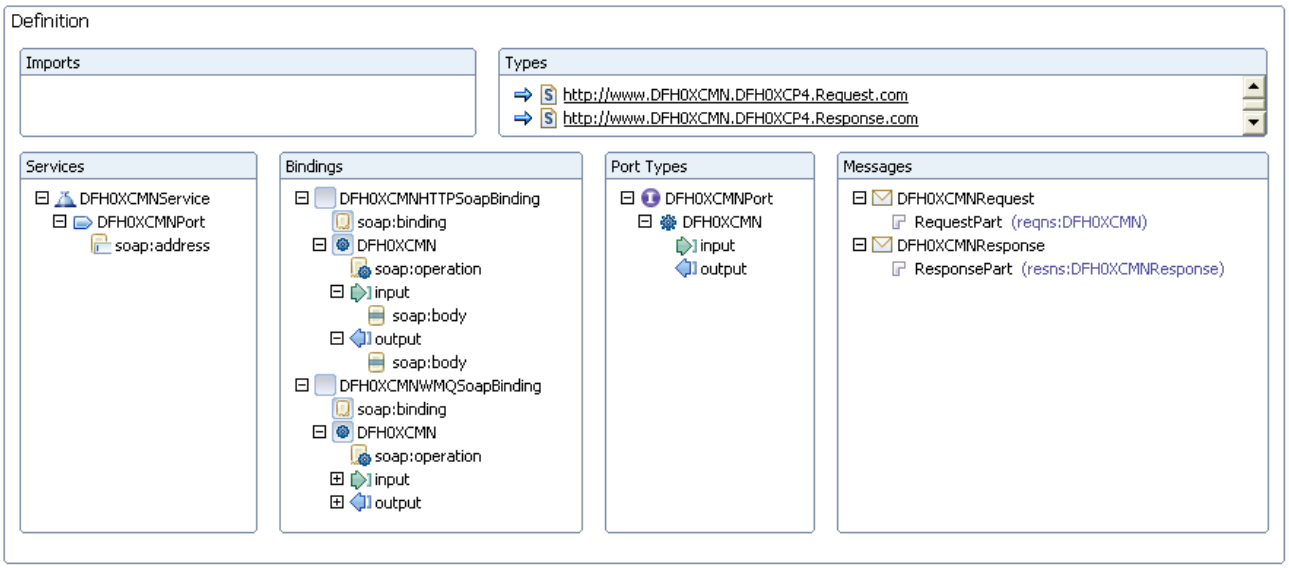

Abbildung 4.5.1: Graphenansicht der WSDL-Datei inquireSingle.wsdl

Im dritten Kapitel wird der Endpunkt des Web Service angepasst. Der Mainframe, auf dem sich die CICS Installation befindet auf die zugegriffen wird, ist der z/OS Großrechner der Universität Leipzig mit der IP Adresse 139.18.4.35. Das veränderte WSDL-Dokument wird lokal gespeichert.

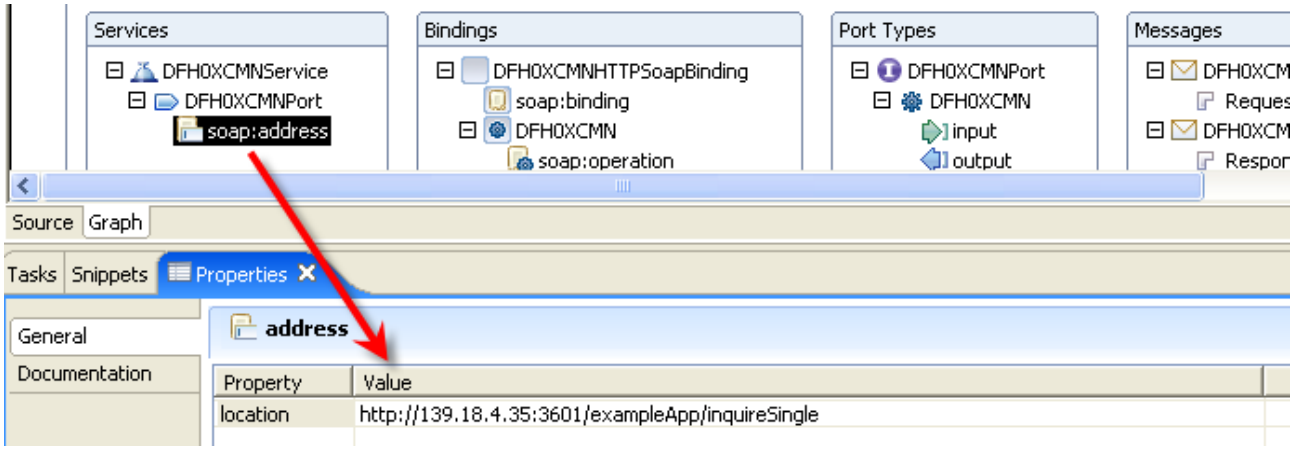

Abbildung 4.5.2: Anpassen des Web Service Endpunkts

Teil vier der Übung testet den Web Service unter Verwendung des in WDz enthaltenen *Web Services Explorer*. Durch die Verwendung dieses Werkzeugs entfällt die sonst notwendige Implementation eines Programms auf dem Client, welches die Anfrage ausführt. Abbildung 4.5.3 zeigt den Web Services Explorer im Einsatz. Im unteren Teil des Bildes, unter *Status,* ist ein Teil des Ergebnisse der erfolgreichen Anfrage zu sehen, während die Eingaben unter *Actions* angegeben werden. Hier muss auch die Adresse an der sich der Web Service befindet, also der Endpunkt, eingetragen werden. Zu den jeweiligen Eingabemasken kann bei Bedarf der Sourcecode aufgerufen und bearbeitet werden.

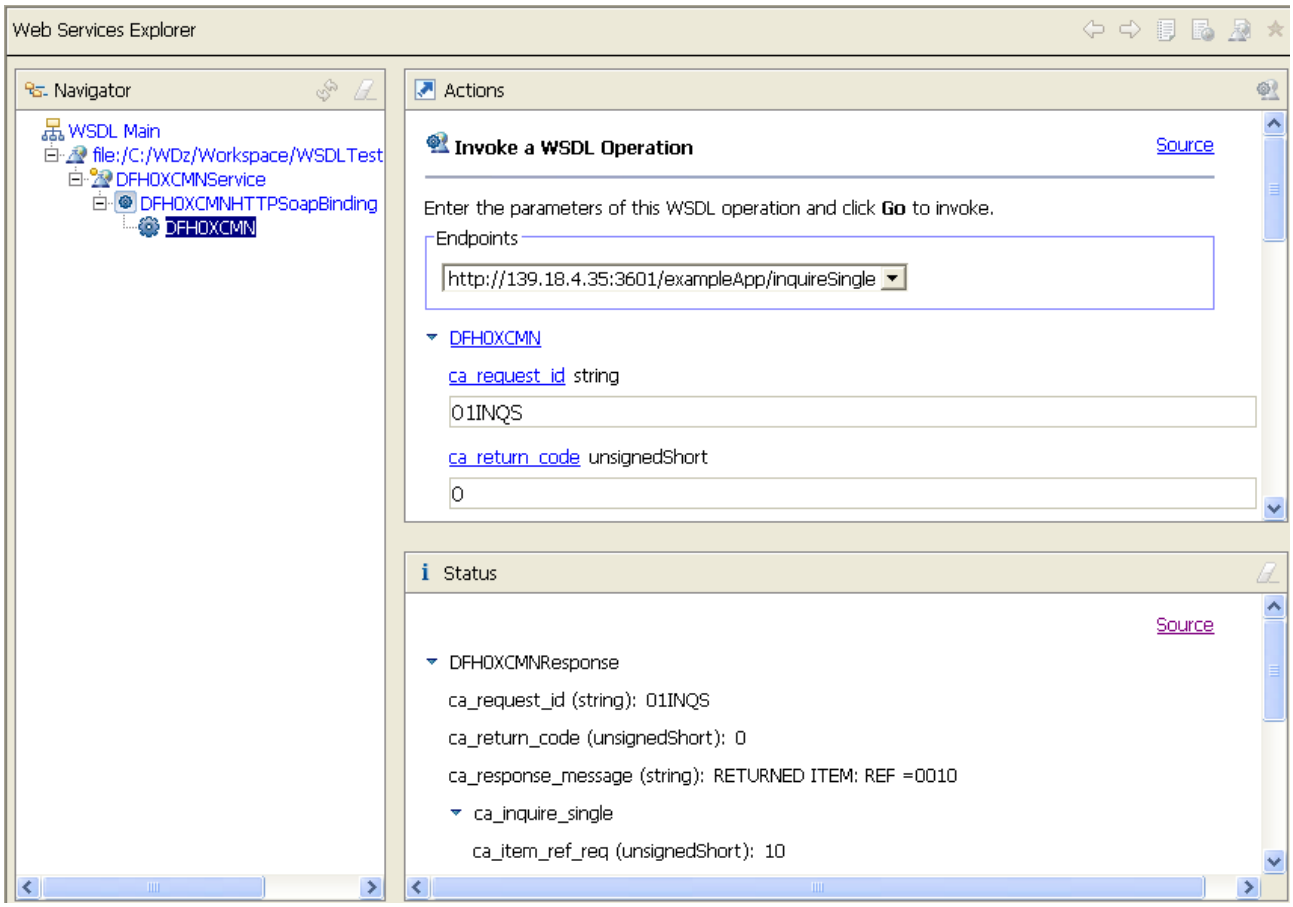

Abbildung 4.5.3: Web Services Explorer im Einsatz

In Kapitel fünf wird ein weiteres Werkzeug von WDz eingeführt, der *TCP/IP Monitor*. Er wird zwischen die Ausgabe des Web Services Explorer und den Server geschaltet und zeichnet alle ein- und ausgehenden Informationen auf. Der Endpunkt des Web Service muss auf den TCP/IP Monitor zeigen, welcher die Eingaben registriert und weiter auf den Mainframe in Leipzig verweist.

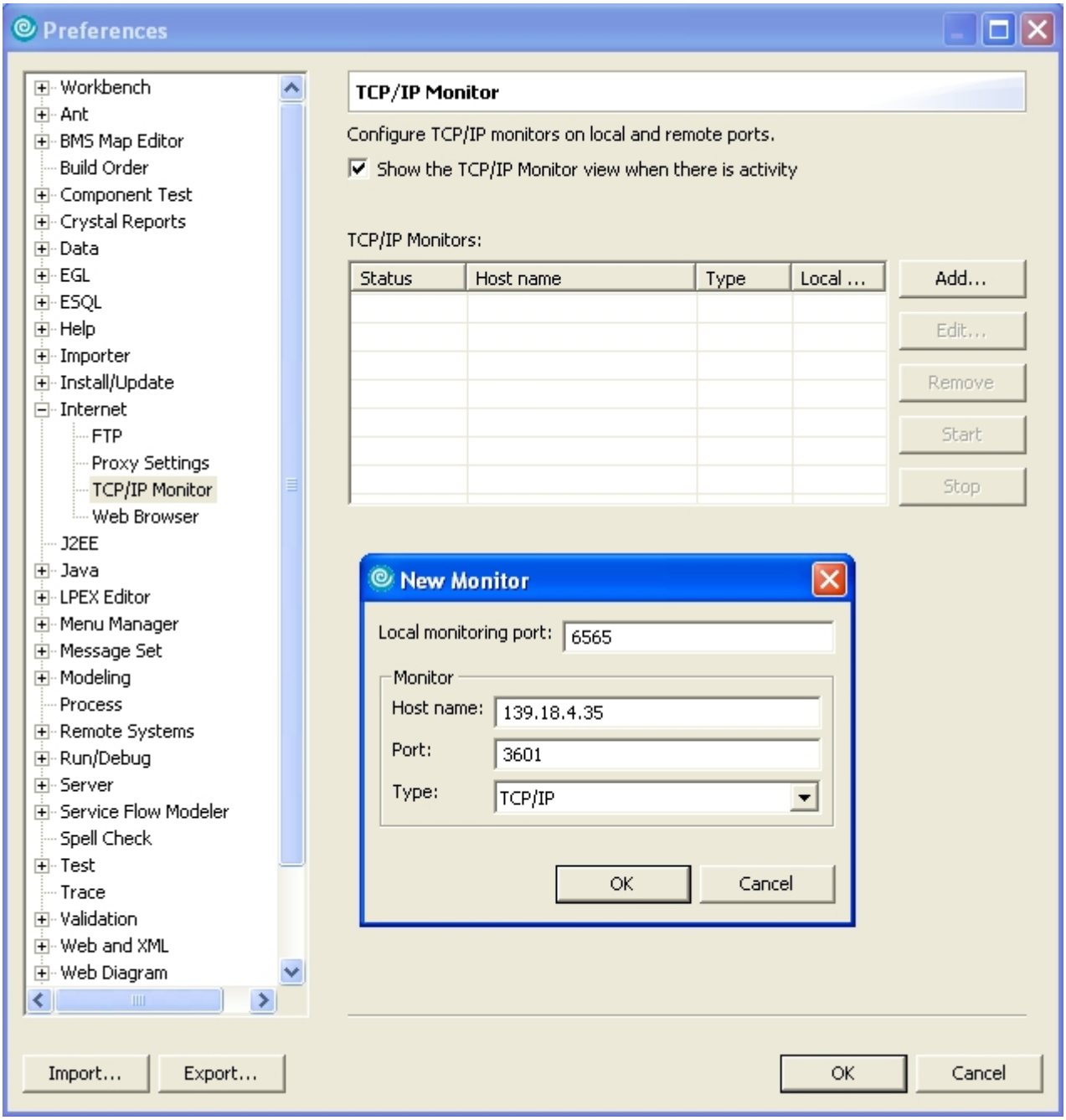

Abbildung 4.5.4: Konfiguration des TCP/IP Monitors

Abbildung 4.5.5 zeigt die Informationen an, die der TCP/TP-Monitors abfängt, wie zum Beispiel die *Response Time* der Anfrage oder die übermittelten HTML-Daten.

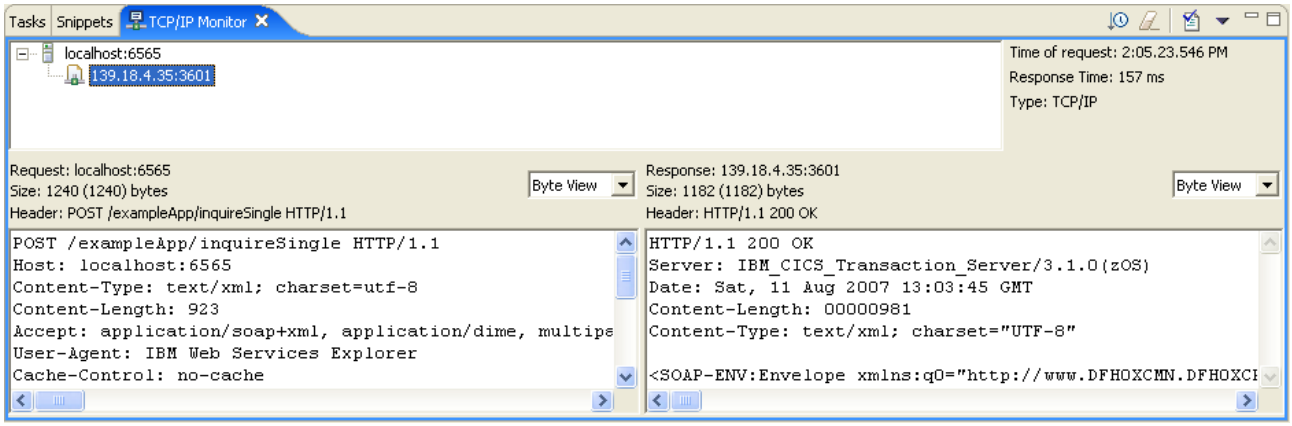

Abbildung 4.5.5: Darstellung der Web Service Anfrage im TCP/IP Monitor

Die Übung schließt ein Aufräumen der Arbeitsumgebung und eine Zusammenfassung der in dem Tutorial vorgestellten Arbeit ab.

4.6 WDz Tutorial 06 – Web Service Enablement for CICS

Aufgabe dieser Übung ist die Einrichtung einer Funktion des *CICS Catalog Managers* als Web Service. Da der Web Service auf ein bestehendes Programm aufbaut, also auf das fertige Programm aufgesetzt wird, nennt man diese Methode *bottom-up*.

Der CICS Catalog Manager ist eine in CICS eingebundene Beispielapplikation, die Teil des *CICS Transaction Servers v3.1* (CICS TS V31) ist. Sie dient der Veranschaulichung verschiedener Methoden des Verbindungsaufbaus zwischen CICS Anwendungen und externen Clients und Servern. Das Beispiel ist als einfache Katalogapplikation aufgebaut, in der einzelne Posten aufgelistet, Informationen zu bestimmten Posten abgefragt und Bestellungen aufgegeben werden können. Eine ausführliche Dokumentation zum CICS Catalog Manager und zu CICS Transaction Server v3.1 findet sich unter [IBM5].

Als Vorbereitung des Tutorials muss unter der *z/OS Projects*-Ansicht ein Filter zur Darstellung der Inhalte des CICS Catalog Managers erstellt werden. Nachdem eine Verbindung zum Mainframe in Leipzig aufgebaut wurde, wird unter dem Unterpunkt *MVS Files* ein neuer Filter mit dem Namen *Catman* erstellt, der den Inhalt des entsprechenden Datasets des Mainframes wiedergibt. Um mit den Inhalten des Filters arbeiten zu können, müssen mit den Daten übereinstimmende *Dataset* und *Member Mappings* erstellt werden, ohne die WDz die angebotenen Daten nicht verarbeiten kann.

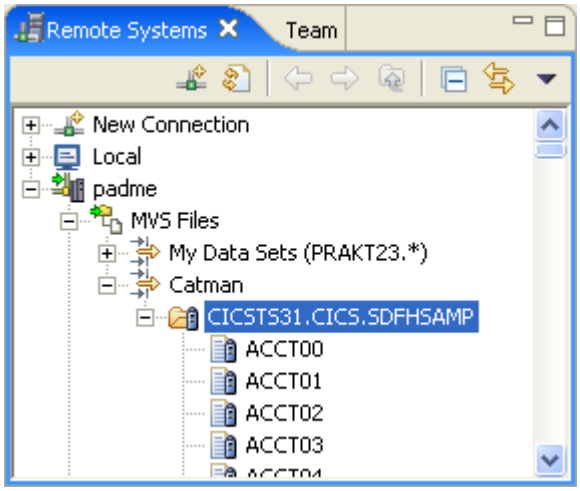

Abbildung 4.6.1: Anzeige des Catman-Filters

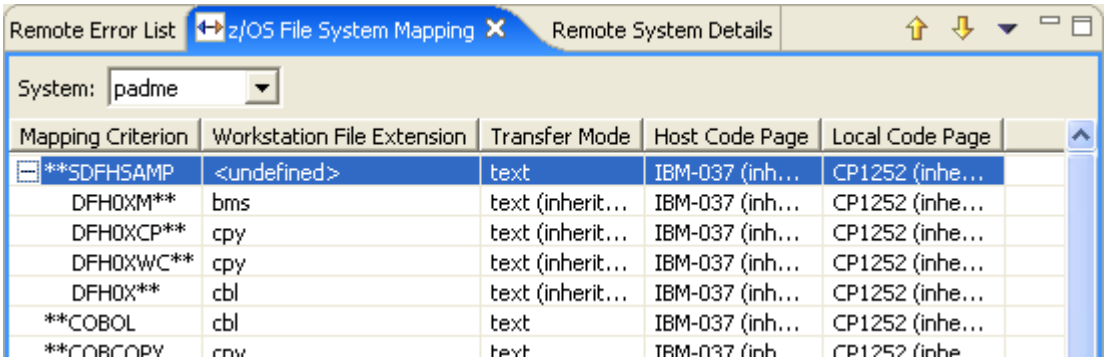

*Abbildung 4.6.2: Neu erstelltes z/OS File System Mapping **SDFHSAMP*

In Kapitel eins wird ein neues Projekt namens *WDzWebserviceEnablement* erstellt und mit Daten des Catalog Managers gefüllt. Auf diesen Daten aufbauend wird mit Hilfe des *CICS Web Service Assistants* ein WSDL-File und das dazu entsprechende WSBIND-File erstellt.
Der CICS Web Services Assistant ist Teil von CICS TS V31 und ermöglicht die automatische Generierung von Web Service Definitionen aus bestehenden Programmstrukturen. Andersherum können aus einem gegebenen WSDL-Dokument auch entsprechende Programmstrukturen generiert werden [IBM6]. Für die Erstellung der Dateien steht unter WDz ein Wizard bereit.

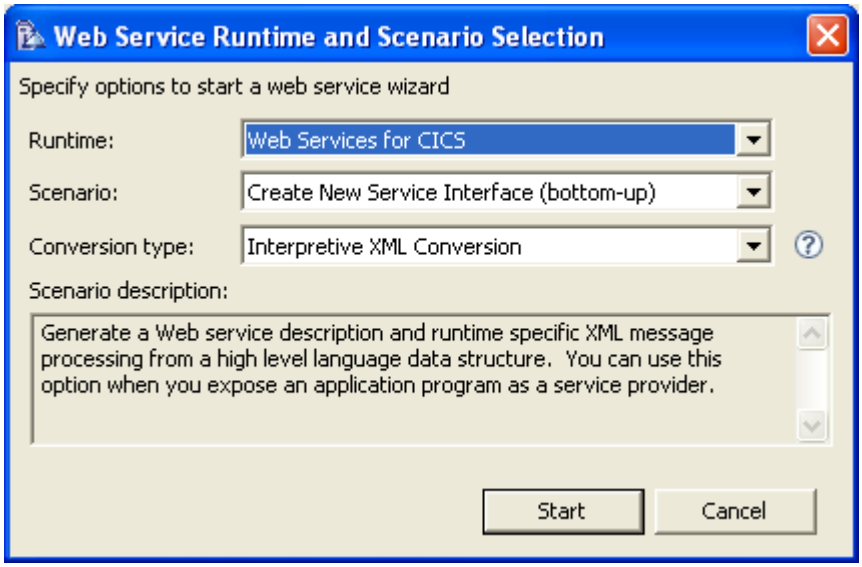

Abbildung 4.6.3: Start des WDz Wizards zur Generierung der Web Service Definitionen

Das generierte WSDL Dokument enthält das für die Kommunikation notwendige XML Schema, einzig die TCP/IP Adresse mit Port muss vor einem Einsatz auf Clientseite konfiguriert werden.
Das WSBIND-File enthält Metainformationen, die CICS während der Laufzeit benötigt, um die bei der Anfrage und Antwort verwendeten XML Daten in die von der Applikation vorausgesetzte Form zu wandeln. Die WSBIND Daten werden jedoch nicht bei jeder Anfrage benötigt, sie kommen nur bei der Installation des Web Service unter CICS zum Einsatz. WSDL und WSBIND-Dateien müssen immer gemeinsam verwendet werden. Werden Datenstrukturen geändert, müssen sowohl WSDL als auch WSBIND neu erstellt und an entsprechender Stelle eingesetzt werden.

Um ein CICS Programm als Web Service über HTTP zu kennzeichnen, werden Ressourcen zu einem so genannten *TCPIPService* und einer *PIPELINE* benötigt. In Kapitel zwei werden unter dem *UNIX System Service* (USS) die dafür erforderlichen *Hierarchical File System* (HFS) Verzeichnisse erstellt. In dieses angelegte Verzeichnis wird in einem späteren Schritt die erzeugte .wsbind-Datei kopiert, die bei der Installation der *PIPELINE* an dieser Stelle ausgelesen wird.

Das folgende Kapitel widmet sich der Erzeugung der *PIPELINE* Ressourcendefinition. Eine so definierte PIPELINE beinhaltet 0 oder mehr *message handler*, die die ein- und ausgehenden Nachrichten jeweils durchlaufen bis sie an ihren Endpunkt gelangen. Der im Beispiel verwendete *cics_soap_1.1_handler* entfernt bei eingehenden Nachrichten die SOAP Hülle der Nachricht und ruft das CICS Programm *DFHPITP* auf, welches unter anderem die Umsetzung der XML Daten in COMMAREA erledigt. Bei ausgehenden Nachrichten geschieht dasselbe in anderer Reihenfolge: zuerst wird die COMMAREA-Ausgabe in XML umgewandelt und danach der SOAP Umschlag hinzugefügt, bevor die Nachricht an den anfragenden Client weitergeleitet wird. Weitere Informationen zu dem in dem Beispiel verwendeten *cics_soap_1.1_handler* finden sich unter

[IBM7]. Für die Konfiguration der PIPELINE wird unter WDz der *Host Connection Emulator Support* verwendet.

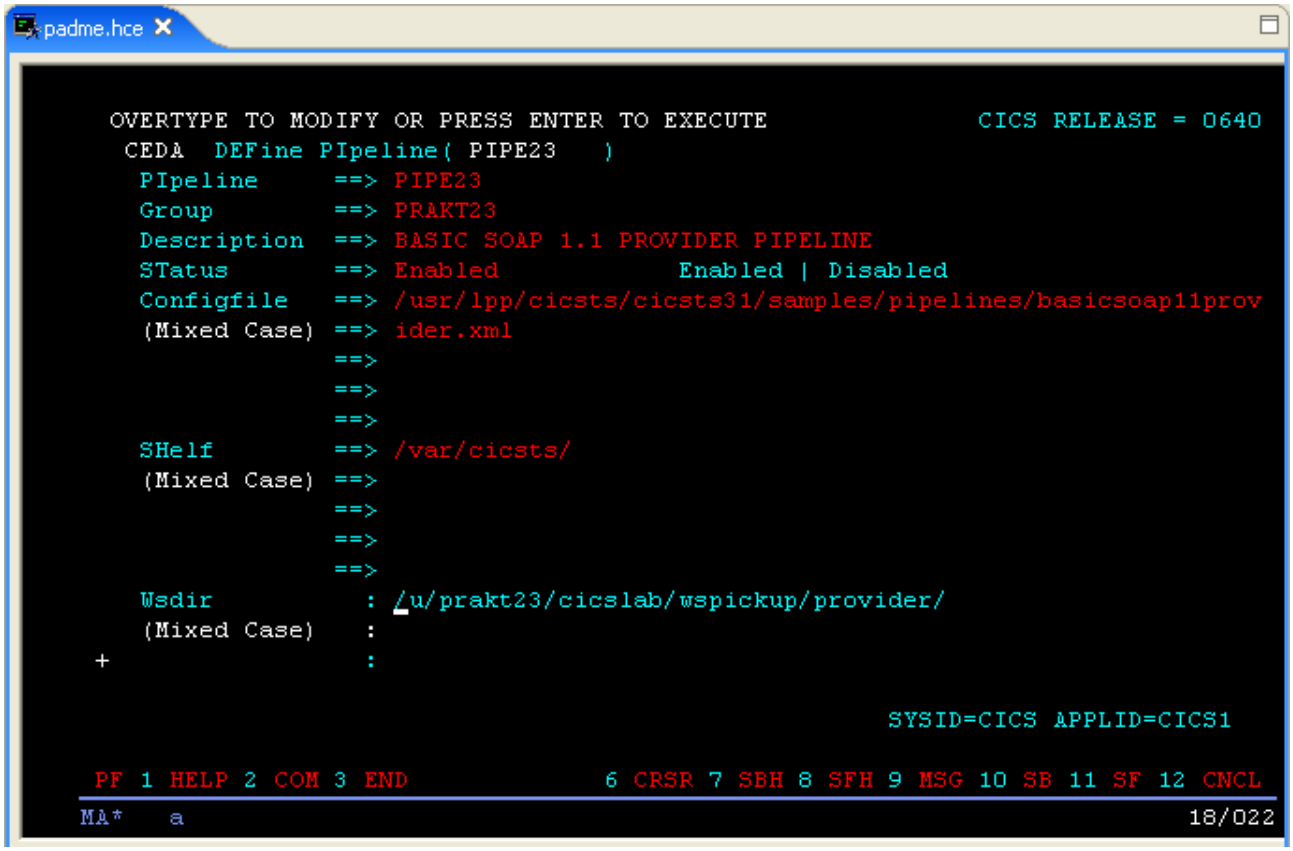

Abbildung 4.6.4: PIPELINE Konfiguration

Im vierten Kapitel wird die benötigte *TCP/IPService* Ressourcendefinition vorgenommen, damit CICS an definiertem Port Anfragen in HTML akzeptiert. Derselbe TCPIPService kann von mehreren Web Services und mehreren Pipelines benutzt werden.

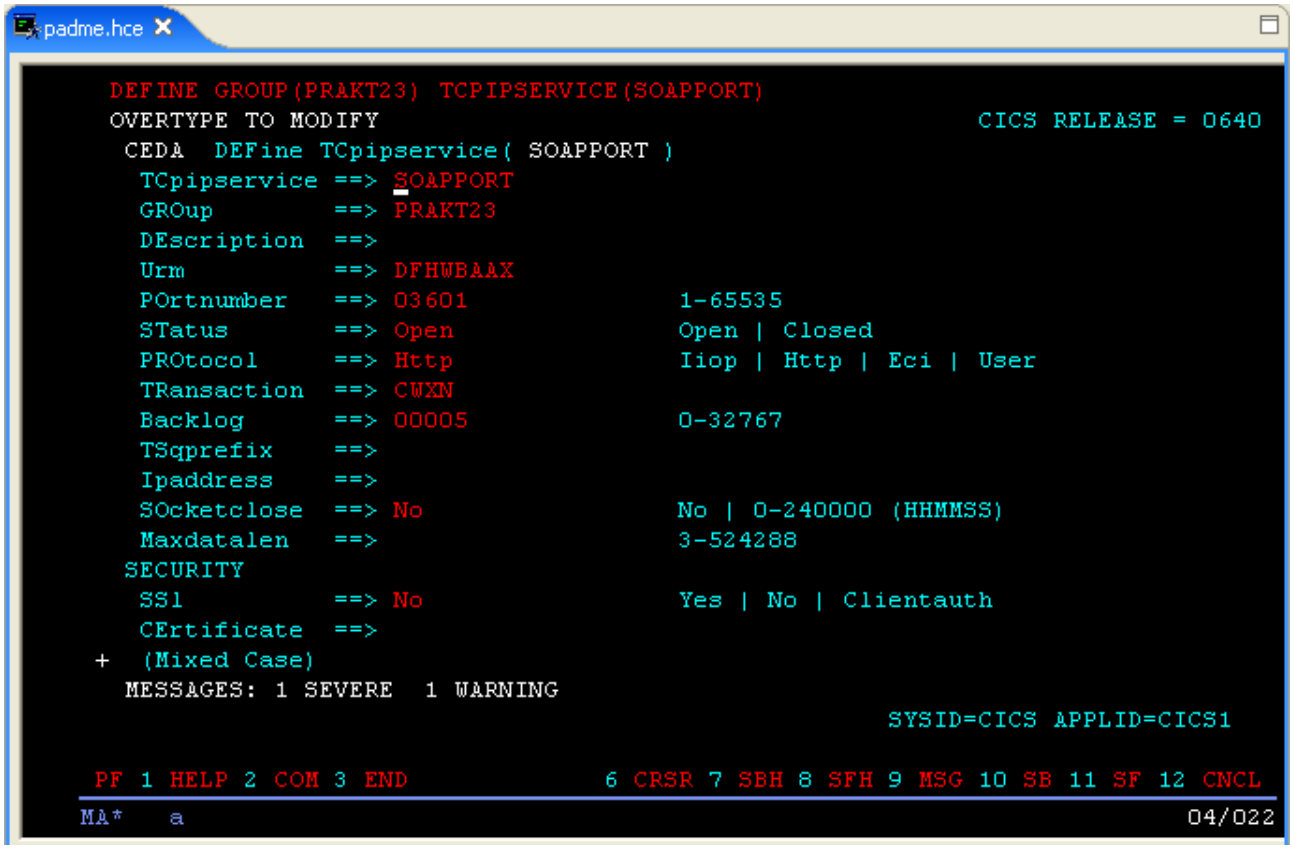

Abbildung 4.6.5: Definition des TCP/IP-Services namens SOAPPORT

In Teil fünf wird das zu Anfang generierte WSBIND-File in das erstellte HFS-Verzeichnis kopiert. Für den Datentransfer von der verwendeten Workstation auf den Mainframe wird WDz verwendet, wozu jedoch zuerst eine Einstellung für den binären Datenaustausch definiert werden muss. Ist die WSBIND-Datei im erstellten USS-Verzeichnis, kann unter CICS ein PIPELINE-Scan durchgeführt werden, was in Kapitel sechs geschieht. Durch den PIPELINE-Scan installiert CICS automatisch die verwendeten *URIMAP*, *WEBSERVICE* und *WSDL*-Ressourcendefinitionen. Die WEBSERVICE Definition wird benötigt, wenn wie in dem verwendeten Beispiel der *CICS Web Service Assistant* zur Generierung der WSDL und WSBIND Daten benutzt wurde. Darin werden Eigenschaften der Laufzeitumgebung gespeichert, die bei dem Einsatz eines CICS Anwendungsprogramms als Web Service nötig sind. Die *URIMAP* Definition (*URI* steht für *Uniform Resource Identifier*) wird von Serviceanbietern zur eindeutigen Identifikation des Web Service benötigt. Ein *URIMAP* verweist auf eine *PIPELINE* und eine *WEBSERVICE* Definition.

Ob die Ressourcen korrekt installiert wurden wird in Paragraph sieben verifiziert. In der Übung wird die Verifizierung wird unter Verwendung des Host Connection Emulator Support durchgeführt.

In Kapitel acht wird, wie in dem vorangegangenen Tutorial, der Web Service mittels des *Web Services Explorer* getestet. Dafür muss, nachdem der Web Services Explorer gestartet wurde, ein neuer Endpunkt eingegeben werden, der auf den installierten Web Service auf dem Mainframe zeigt.

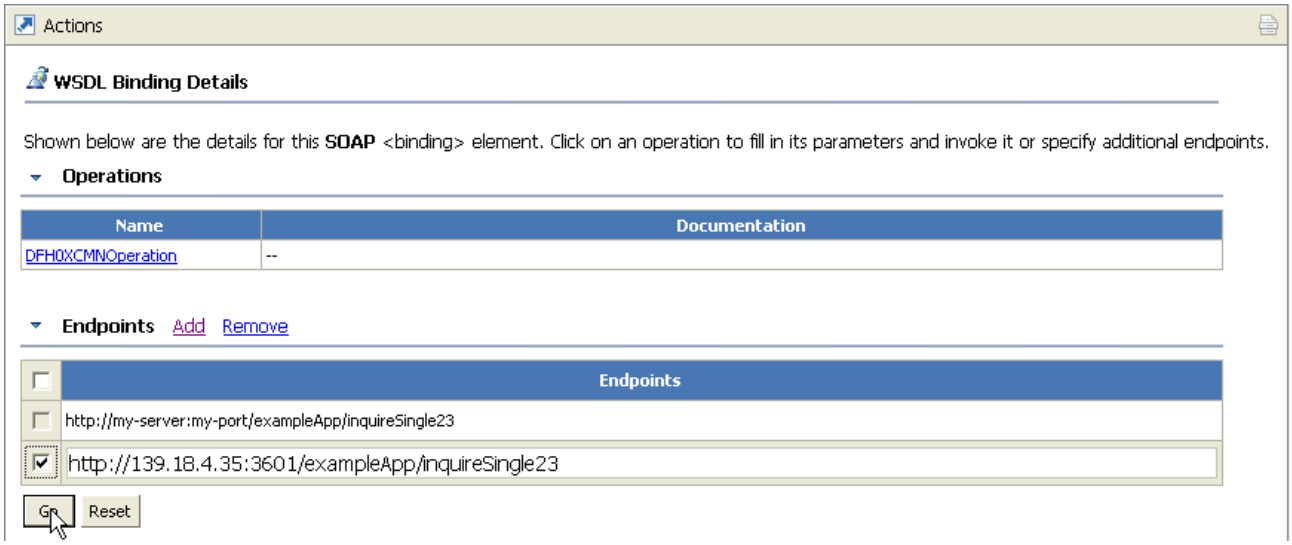

Abbildung 4.6.6: Festlegung des Endpunktes für den Web Service

Anschließend wird unter der Testumgebung die Eingabemaske mit Daten gefüllt und eine Anfrage abgesendet. Abbildung 4.6.7 zeigt einen Teil der Eingabemaske und die Antwort des Servers.

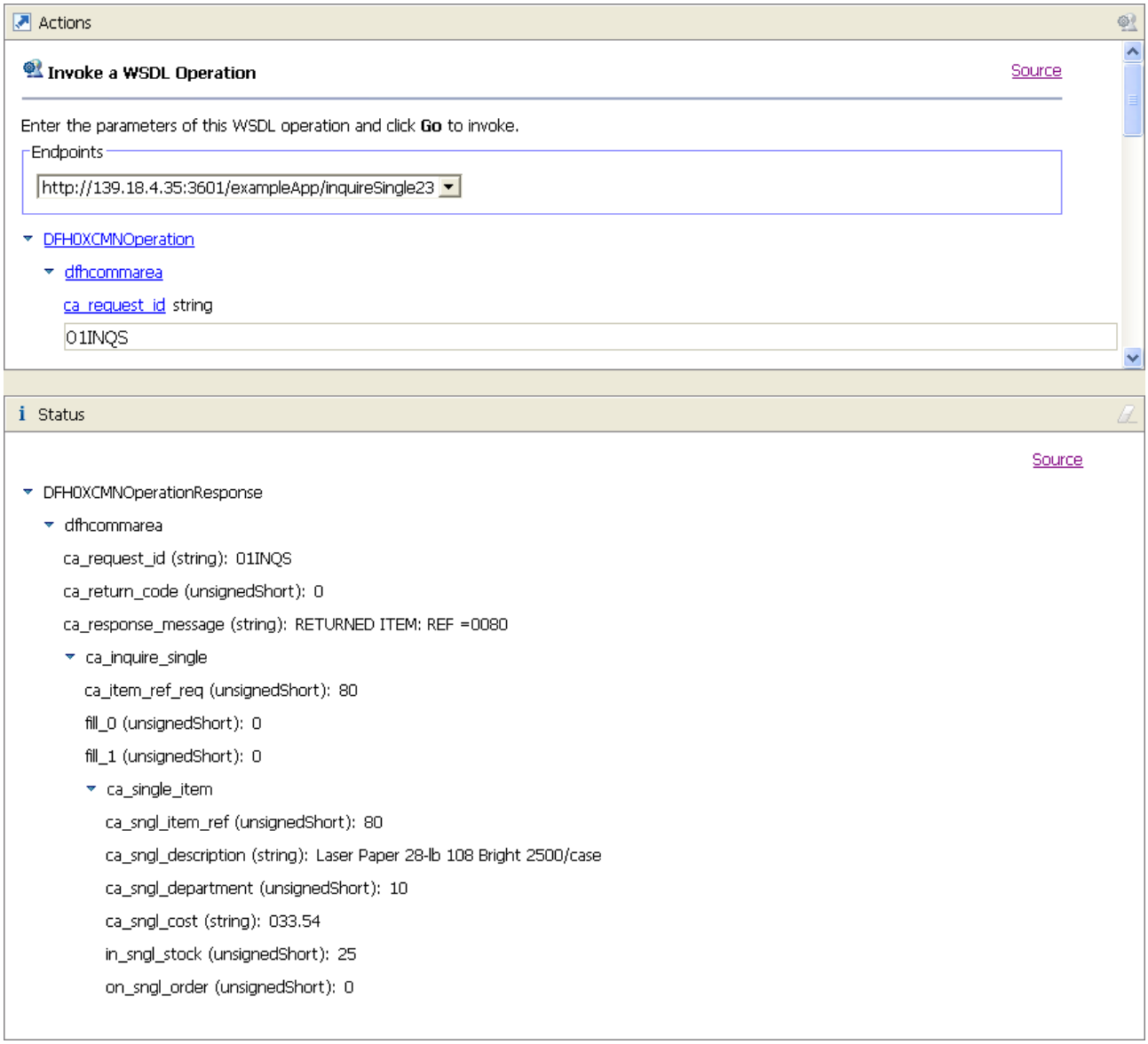

Abbildung 4.6.7: Anfrage mit Antwort des erstellten Web Services

In einem weiteren Schritt wird der entsprechende Sourcecode dargestellt, bevor die Übung in Kapitel neun mit einer Zusammenfassung abschließt.

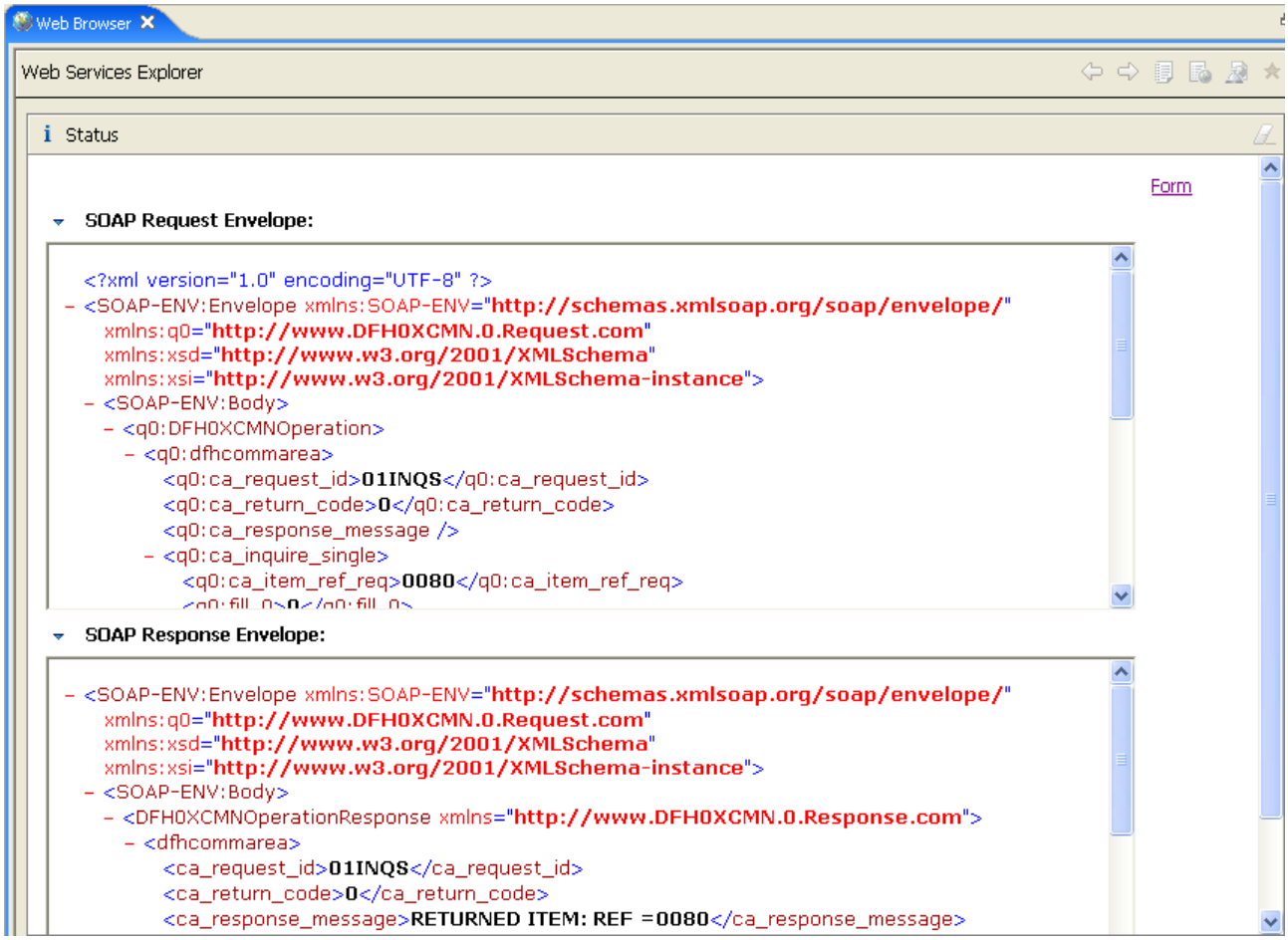

Abbildung 4.6.8: Ansicht des Sourcecodes der SOAP-Nachrichten

Kapitel 5

Zusammenfassung und Ausblick

Nach der Einleitung und einer Einführung in die Grundlagen in Kapitel zwei und drei wurden als Schwerpunkt der Arbeit die Tutorials vorgestellt, die in den nächsten Semestern im Rahmen des Client/Server Praktikums von Studenten der Universitäten Tübingen und Leipzig durchgeführt werden. Die in den Übungen bearbeiteten Themengebiete, COBOL Programmentwicklung und Generierung und Test von Web Services mit WebSphere Developer for zSeries, werden auch die nächsten Jahre tragende Rollen in der globalen Geschäftswelt spielen.

COBOL Programme sind seit der Einführung der Mainframes im Einsatz, teilweise laufen heute noch Programme, die vor über 30 Jahren erstellt wurden. Jährlich werden für Geschäftsanwendungen ungefähr 5 Millionen Zeilen COBOL neu erstellt, was 35% des Gesamtvolumens der neu geschriebenen Programme in diesem Bereich entspricht [Spruth06]. WDz stellt hier vor allem für Neueinsteiger ein wichtiges Hilfsmittel dar, da damit erstmals ein aktuelles IDE für diese Programmiersprache zur Verfügung steht. Auch der Einsatz der weit verbreiteten Eclipseplattform erleichtert Anfängern den Einstieg, da häufig Eclipse schon von der Programmierung mit Java oder C/C++ ein bekanntes Element ist.

Web Services finden vor allem im *Business-to-Business*-Bereich (B2B) Einsatz, da Unternehmen verstärkt auf die Wiederverwendung und Integration bestehender Programme gegenüber teurer Neuentwicklungen setzen. Der große Vorteil von Web Services bei der so genannten *Enterprise Application Integration* (EAI) ist neben Plattformunabhängigkeit der einfache Einsatz. Beispielsweise kann WDz für bestehende Programme die benötigten Komponenten weitestgehend automatisiert generieren, oder aus gegebenen WSDL-Files Programmgerüste erstellen. Die generierten Artefakte vereinfachen die Einbindung der benötigten Komponenten um ein Vielfaches. Da Web Services bisher kaum in Vorlesungen integriert sind, beinhalten die Tutoials eine Einführung in das Thema.

Auf dieser Arbeit aufbauend sind im Rahmen einer Studien- oder Diplomarbeit weitere Tutorials für das Client/Server Praktikum denkbar. Ein Beispiel hierfür wäre eine Abhandlung über den *WebSphere Host Access Transformation Service* (HATS), welcher 3270- und 5250-Bildschirme dynamisch in HTML umwandelt. Dadurch können traditionelle Mainframeanwendungen schnell und einfach mit dem Look-and-Feel von Webanwendungen versehen werden, ohne Änderungen am Quellcode vornehmen zu müssen.

Literaturverzeichnis

[Cas04] F. Cassia: *Eclipse.org eclipsing Borland's Jbuilder.* VNU Business Publications, 2004. http://www.theinquirer.net/default.aspx?article=15862 [Stand 18.08.2006]

[CI99] *IBM Weighs in With Uptime Guarantee*s. Computergram International, 1999. http://findarticles.com/p/articles/mi_m0CGN/is_1999_March_25/ai_54207476/pg [Stand 18.08.2007]

[DB2SP06] P. Bruni u.a.: *DB2 for z/OS Stored Procedures: Through the CALL and Beyond*. IBM Form No. SG24-7083-00, 2006. http://www.redbooks.ibm.com/abstracts/sg247083.html [Stand 20.08.2007]

[Ecl] *Eclipse – an open development platfo*rm. http://www.eclipse.org [Stand 20.08.2007]

[Ecl2] *Eclipse Newcomers FAQ*. http://www.eclipse.org/home/newcomers.php [Stand 20.08.2007]

[EGL] *IBM developerWorks: Enterprise Generation Language*. IBM. http://www-128.ibm.com/developerworks/rational/products/egl/ [Stand 20.08.2007]

[Her04a] P. Herrmann, U. Kebschull, W. G. Spruth: *Vorlesungsskript Internet Anwendungen unter z/OS und OS/390 – Teil 4.* Universität Tübingen, 2004.

[Her04b] P. Herrmann, U. Kebschull, W. G. Spruth: *Einführung in z/OS und OS/390*. 2. Auflage, Oldenbourg, 2004

[Her06] P. Herrmann, W. G. Spruth: *Vorlesungsskript Einführung in z//OS und OS/390 – Teil 7.* Universität Leipzig, 2006.

[IBM1] *WebSphere Developer for System z*. IBM. http://www-950.ibm.com/ecatalog/Detail.wss?locale=de_DE&synkey=H564288F91212R05 [Stand: 20.08.2007]

[IBM2] *WebSphere Developer for System z – Features and benefits*. IBM. http://www-306.ibm.com/software/awdtools/devzseries/ [Stand 20.08.2007]

[IBM3] *IMS SOAP Gateway.* IBM. http://www-306.ibm.com/software/data/ims/soap/ [Stand 20.08.2007]

[IBM4] T. Ross, N. Tindall, S. Tampa: *XML, COBOL and Application Modernization*. IBM Reference No. 7004198, 2007. http://www-1.ibm.com/support/docview.wss?rs=492&context=SS6SG3&q=&uid=swg27004198&loc=en_US&cs=utf-8?=en+en [Stand: 20.08.2007]

[IBM5] *The CICS catalog manager example application*. IBM. http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/index.jsp?topic=/com.ibm.cics.ts31.doc/dfhxa/topics/dfhxa_t100.htm [Stand 20.08.2007]

[IBM6] *The CICS Web services assistant*. IBM. http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/topic/com.ibm.cics.ts31.doc/dfhws/concepts/dfhws_utility.htm [Stand 20.08.2007]

[IBM7] *The `<cics_soap_1.1_handler>` element*. IBM. http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/index.jsp?topic=/com.ibm.cics.ts31.doc/dfhws/reference/pipeline/dfhws_cics_soap_11_handler.htm [Stand 20.08.2007]

[RAD06] J. Ganci u.a.: *Rational Application Developer V6 Programming Guide*. IBM Form No. SG24-6449-00, 2006. http://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246449.html [Stand 20.08.2007]

[SA03] *IBM SOAP for CICS feature delivers fully supported SOAP access to CICS*. IBM, Software Announcement 203-199, 2003. http://www-306.ibm.com/fcgi-bin/common/ssi/ssialias?infotype=an&subtype=ca&appname=Demonstration&htmlfid=897/ENUS203-199 [Stand 20.08.2007]

[SOAP] N. Mitra, Y. Lafon: *SOAP Version 1.2 Part 0: Primer (Second Edition)*. W3C, 2007. http://www.w3.org/TR/soap12-part0/ [Stand 20.08.2007]

[Spruth06] W. G. Spruth: *Vorlesungsskript: Client/Server Systeme – Teil 9*. Universität Tübingen, 2006

[UDDI] *OASIS UDDI*. http://www.uddi.org/ [Stand 20.08.2007]

[VMware] *VMware*. http://www.vmware.com/de/ [Stand 20.08.2007]

[Wik1] *Eclipse (IDE)*. Wikipedia. http://de.wikipedia.org/wiki/Eclipse_%28IDE%29 [Stand 27.07.2007]

[Wik2] *Web Service*. Wikipedia. http://de.wikipedia.org/wiki/Webservice [Stand 27.08.2007]

[WSgloss04] H. Haas, A. Brown: *Web Services Glossary*. W3C, 2004. http://www.w3.org/TR/ws-gloss/ [Stand 20.08.2007]

[WSIL] *An overview of the Web Services Inspection Language*. IBM. http://www.ibm.com/developerworks/library/ws-wsilover/index.html [Stand 20.08.2007]

[XML] *Extensible Markup Language (XML)*. W3C. http://www.w3.org/XML/ [Stand 27.08.2007]

[XSD04] D.C. Fallside, P. Walmsley: *XML Schema Part 0: Primer Second Edition*. W3C, 2004. http://www.edition-w3c.de/TR/2001/REC-xmlschema-0-20010502/ [Stand 27.08.2007]

Anhang

A - Inhalt der beigefügten DVDs

Die beigefügten DVDs enthalten folgenden Inhalt:

DVD Nr. 1

- \
  Imagedateien des vorinstallierten WindowsXP-Gastsystems
- \Diplomarbeit
  Diplomarbeit als PDF
- \Literaturreferenzen
  In elektronischer Form verfügbare Literaturreferenzen, einschließlich Internetseiten
- \VMware
  Installationsfile des Vmware-Players
- \WDz_Tutorials
  Verzeichnis, welches die im Rahmen der Diplomarbeit erstellten Tutorials enthält

DVD Nr. 2

- \
  Imagedateien des vorinstallierten WindowsXP-Gastsystems

B - Abkürzungsverzeichnis

ALM	Application Lifecycle Management
B2B	Business-to-Business-Bereich
BMS	Basic Mapping Support
CF	Coupling Facility
CICS	Customer Information Control System
CICS TS V31	CICS Transaction Servers v3.1
CORBA	Common Object Request Broker Architecture
EAI	Enterprise Application Integration
EGL	Enterprise Generation Language
HATS	Host Access Transformation Service
HFS	Hierarchical File System
IDE	integrated developement enviroment,
IMS	Information Management System
ISPF	Interactive System Productivity Facility
JCL	Job Control Language
JES	Job Entry System
LIC	Licenced Internal Code
LPAR	Logical Partitions
PDS	Partitioned Data Set
RMI	Remote Method Invocation
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol

UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
USS	UNIX System Service
WDz	WebSphere Developer for zSeries
WLM	Workloadmanager
WSDL	Web Service Description Language
WS-Inspection	Web Services Inspection
XML	Extensible Markup Language
XSD	XML Schema Definitionen