IBM WebSphere Web Services Tutorial

Diplomarbeit

vorgelegt von

Joffrey Fitz
Matrikelnummer: 2163534

Betreuer:
Prof. Dr.-Ing. Wilhelm G. Spruth

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

22. April 2010

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

_____

Ort, Datum Unterschrift

Zusammenfassung

Die vorliegende Arbeit ist ein Tutorial für die Erstellung von Web Services mittels IBM Rational Developer for System z v7.5.0 (RDz) mit integriertem Rational Developer for WebSphere Software und dem IBM WebSphere Application Server Version 7.

Das Tutorial bietet eine Schritt-für-Schritt Anleitung zur Erstellung von Web Services nach der *bottom-up* Methode. Ausgehend von einer laufenden RDz Installation wird zunächst eine Java Enterprise Bean (EJB), *ConverterBean*, mit zwei Methoden zur Temperaturumwandlung (`celsiusToFahrenheit()` sowie `fahrenheitToCelsius()`) entwickelt. Zum Testen der EJB wird eine Webapplikation basierend auf einem Servlet programmiert.

Die bereitgestellten Methoden der EJB werden dann als Web Services inklusive einer WSDL (Web Service Description Language) bereitgestellt und auf dem Applikationsserver installiert. Die Web Services und die WSDL werden im darauffolgenden Kapitel mit den RDz Tool *Web Services Explorer* getestet und die Kommunikation zwischen Client und Web Service auf Protokollebene mit Hilfe des *TCP/IP Monitor* untersucht.

Ausgehend von der erstellten WSDL wird ein Standalone Java Thin Client programmiert, welcher aus einer eigenständigen Java Virtual Machine (JVM) heraus die vom Applikationsserver bereitgestellten Web Services konsumiert.

Wert wurde bei dieser Arbeit vor allem auf eine einheitliche und konsistente Vorgehensweise gelegt, die es ermöglicht, die zusammenhängende Entwicklung von der Enterprise Applikation über das Testen bis hin zum eigenständigen Web Service Konsumenten im Stil eines Ende-zu-Ende Tutorials darzustellen.

Inhaltsverzeichnis

1 Einführung

In dieser Einführung werden Web Services als technisches Fundament von SOAs vorgestellt und eine kurze Übersicht über die wichtigsten Web Services Standards und Protokolle gegeben. Weiter wird kurz die Java EE Unterstützung für Web Services besprochen und eine Übersicht der verwendeten IBM Produkte die Entwicklung besprochen.

1.1 Web Services

1.1.1 SOA und Web Services

In einer Serviceorientierten Architektur (SOA) werden Applikationen von eigenständigen lose gekoppelten Softwareservices aufgebaut. Die Hauptpfeiler einer SOA sind ein Service Provider, ein Service Broker und ein Service Requester. Der Service Provider stellt einen Service bereit und veröffentlicht die Servicedefinition und kann den Service beim Service Broker registrieren. Der Service Broker stellt ein Verzeichnisdienst für veröffentlichte Services dar (Service Registry) und vermittelt zwischen Service Provider und Service Requester. Letzterer ist der Service-Klient oder Konsument der den bereitgestellten Service als Teil seiner Implementierung nutzt. [RAD07, S.812f]

Web Services stellen die technische Grundlage für die Implementierung einer SOA dar. Dabei handelt es sich bei Web Services um unabhängige Software Dienste, die über einfache, bereits bestehende standardkonforme Netzwerkprotokolle plattformunabhängig aufrufbar sind. [RAD07, S.812f]

Die W3C Web Services Architecture Working Group definiert Web Services wie folgt:

Definition: *“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." [W3C04].

1.1.2 Web Services Standards

Ein Web Service stellt keine singuläre Technologie dar, er bestehen vielmehr aus einer Kombination unterschiedlicher Technologien, Protokolle, Spezifikationen und Schnittstellen. Dabei stellen die wichtigsten Komponenten das Transportprotokoll, Das Servicekommunikations-Protokoll, sowie die Servicebeschreibung dar.

Als Transportprotokoll werden derzeit das Hypertext Transfer Protokoll, normal und verschlüsselt über SSL, (HTTP/S), das Simple Mail Transfer Protocol (SMTP) und der Java Messaging Service (JMS) unterstützt. Neue Entwicklungen sind das Blocks Extensible Exchange Protocol [RFC3080] (BEEP) und WS-ReliableMessaging. [OASIS07]

Als Servicekommunikations-Protokoll dient SOAP. SOAP ist ein XML-basiertes Protokoll das es erlaubt, strukturierte und typisierte Daten und Information in einem Verteilten System auszutauschen [W3C07].

Die Web Service Beschreibung erfolgt über die Web Service Description Language (WSDL). Der W3C hat WSDL als Standard zur Basisbeschreibung von Web Services übernommen. Die WSDL beschreibt die Charakteristiken und das Schnittstellen-interface eines Web Services unter Benutzung eines oder mehrerer XML Dokumente. Die Beschreibung enthält grundlegende Informationen über den Zweck des Services, wie der Service erreichbar ist (den Endpoint), und wie der Service aufgerufen werden kann (Parametrisierung) [WSBI04, S.18ff].

Weitere Standards die zum Web Services Stack gehören sind ein Verzeichnisdienst (Universal Description Discovery and Integration, UDDI), Sicherheitsaspekte (WS-Security) und Web Services Interoperabilität zwischen unterschiedlichen Hersteller und Softwareplattformen (WS-I) [WHDev05, S.240].

1.1.3 Web Services Entwicklungsszenarien

Es gibt verschieden Ansätze, je nach Vorraussetzungen, für die Entwicklung von Web Services.

Ist der Ausgangspunkt ein bestehender Web Service mit einer verfügbaren WSDL, welche zum Beispiel durch einen Service Provider oder eines Geschäftspartners zur B2B Integration bereitgestellt wird, so spricht man vom *top*-down-Ansatz. Auf Basis der WSDL Definition werden Interfaces der zu erzeugenden Web Services erstellt, deren Geschäftslogik anschliessend implementiert werden muss. [WHDev05, S.265]

Der *bottom-up* Ansatz wird verwendet, um aus einer bestehende Applikation eine WSDL und, abhängig vom Programmier Modell, Proxies und Deployment Deskriptoren (welche z.B. Bei Java EE zum Einsatz kommen) zu generieren. Diesen Ansatz werden wir in dem in dieser Arbeit vorgestellten Tutorial verwenden. [WHDev05, S.264]

Neben diesen beiden generellen Ansätzen gibt es noch den hybriden *meet-in-the-middle* Ansatz, bei dem nur Teile der WSDL für die Implementierung des Web

Services genutzt werden können und für z.B. nicht unterstützte Datentypen der Web Service Plattform ein Wrapper eingesetzt werden muss, der eine Datenkonversion der von RDz erstellten Typen in die von der existierenden Applikation verwendeten umsetzten. Dies ist z.B. bei der Integration von CICS und Java EE Web Service der Fall.[CICSWS10, S.66]

Eine weiter Web Service Entwicklungsstrategie ist der *composite*-Ansatz, bei dem mehrere Services zu einem neuen integriert werden. [WHDev05, S.266]

1.2 Java EE und Web Services

Die Java Enterprise Edition (Java EE) stellt ein Programmiermodell und Implementierungsplattform für Web Services dar. Die aktuelle Version der Enterprise Edition ist Version 6. IBM WebSphere Application Server stellt eine zertifizierte Runtime für Java EE Version 5 (Stand April 2010) bereit [WJEE10].

Die Java EE Unterstützung von Web Services ist in einer Reihe von Java Specification Requests (JSRs) spezifiziert [WSBI04, S.21]. Die wichtigsten Spezifikationen sind Java APIs for XML-Based RPC, besser bekannt als JAX-RPC (definiert in [JSR101]), Java APIs for WSDL (JSR 110), Standard APIs für die Repräsentation und Veränderung von Services die durch WSDLs beschrieben sind [JSR110], sowie JSR 109, Implementing Enterprise Web Services, welches das Programmiermodell und die Runtime-Architektur für die Web Services Implementierung definiert [JSR109].

1.3 Übersicht der WebSphere Entwicklungstools

Für die Entwicklung von Java Applikationen, Web Applikationen und Web Services in diesem Tutorial verwenden wir IBM Rational Developer for System z v7.5 (RDz) mit integriertem Rational IBM Application Developer for WebSphere Software. Als Server kommt IBM WebSphere Application Server Version 7 zum Einsatz.

Rational Developer ist eine Integrierte Entwicklungsumgebung (Integrated Devlopment Environment, IDE) basierend auf Eclipse 3 und bietet eine einheitliche Oberfläche und zahlreiche integrierte Tools die das Rapid Application Devlopement, das Testen und Debuggen ermöglichen. Somit wird die Softwareentwicklung sowie Serverintegration und Softwaredepolyment vereinfacht.

Neben Java EE und Web Service Entwicklung bietet RDz Unterstützung für Cobol, Assembler, PL/I, Java, JEE, C/C++, SQL und Stored Procedures. [CICSWS10, S.122]

2 Java EE Beispielapplikation

In diesem Kapitel werden wir eine einfache Java Enterprise Web Applikation erstellen die wir in den folgenden Kapiteln als Basis für unsere Web Services verwenden werden. Um das Augenmerk auf die Technologien zu konzentrieren wurde die Logik der Beispiel Applikation bewusst simpel gehalten. Die Applikation besteht aus einer Enterprise Java Bean (EJB) welche die Programmlogik enthält und einem Servlet, der die Methoden der EJB aufruft und das Ergebnis mittels Java Server Pages (JSP) an einen Webbrowser ausliefert.

Zunächst erstellen wir die Interface für zwei Methoden, `celsiusToFahrenheit()` und `fahrenheitToCelsius()`. Dann implementieren wir die Geschäftslogik als EJB. Um die Funktionalität zu testen entwickeln wir eine Webapplikation basierend auf einem Servlet und einer JavaServer Page (JSP). Nach Abschluss der Projektimplementierung exportieren wir die Enterprise Applikation als Enterprise Archive (EAR), welches der Ausgangspunkt für Kapitel 3, die Erstellung der Web Services, sein wird.

2.1 Vorraussetzungen

Vorraussetzung für dieses Tutorial ist die Verfügbarkeit einer IBM WebSphere Application Server v 7.0 Runtime. Wir Überprüfen dies indem wir in die Voreinstellungen von RDz öffnen:

- Menü *Windows → Preferences* wählen
- Auswahl des Eintrages *Server*, *Installed Runtimes* anklicken
- Es sollte eine WebSphere Runtime wie in Abb. 2 verfügbar sein.

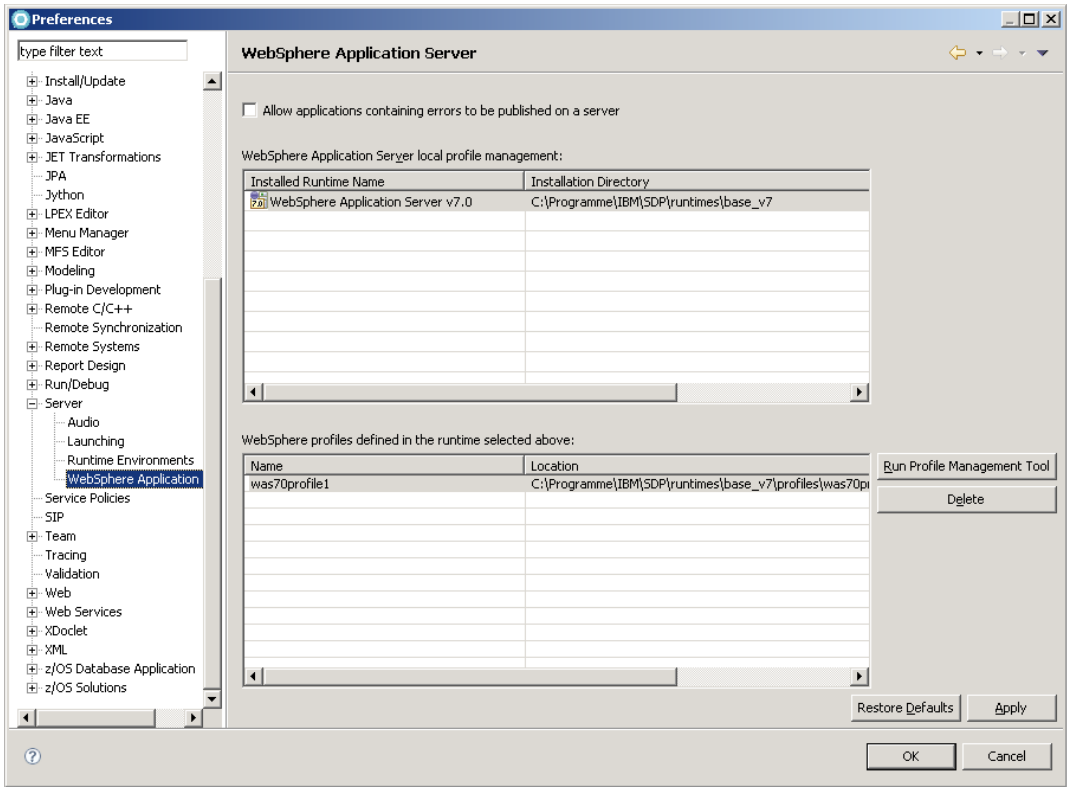

Abbildung 1: Server Einstellungen

Zunächst müssen wir einen neuen Applikations-Server definieren, mit dessen Runtime wir die Enterprise Applikation entwickeln werden. Hierzu sind folgende Schritte notwendig:

- Menü *Window -> Show View -> Servers* aktiviert die Server View
- Mittles Rechtsklick in die Server View wählen wir aus dem Kontextmenü *New*
- Im *Define a New Server* Wizard nehmen wir folgende Einstellungen vor (siehe Abb. 2):
  Server's host name: localhost
  Select the server type: Websphere Application Server v7.0

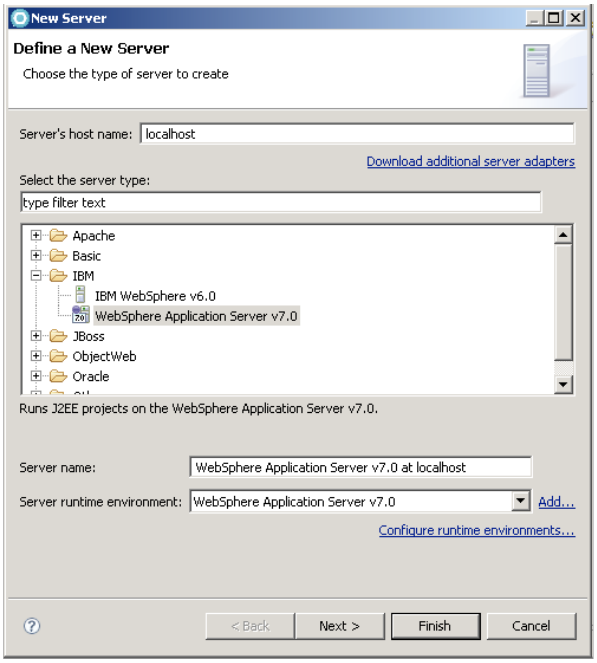

Abbildung 2: Einen neuen Server definieren

Mit *Finish* wird der Dialog geschlossen und eine neue Server Runtime zur Verfügung gestellt. Durch Rechtsklick auf den neuen Server in der *Server View* und Auswahl des Menüpunktes *Start* können wir den Server starten. Nach einigen Minuten, sollte der *State* in der *Server View* als *Started* sowie der Status als *Synchronised* angegeben sein. Mit einem Doppelklick auf den neu eingerichteten Server gelangen wir zu einer Übersichtseite und den Einstellungen des Server. Hier klappen wir den Bereich *Publish* auf und wählen *Never publish* automatically (siehe Abb. 3). Hiermit ist die Servereinrichtung abgeschlossen.

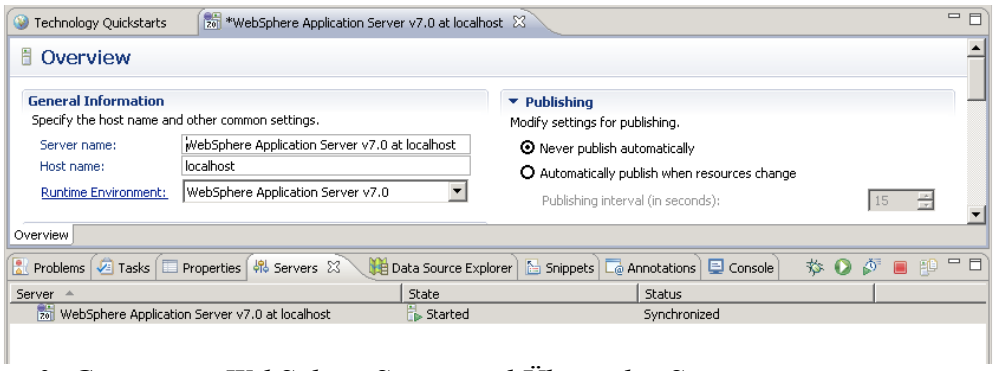

Abbildung 3: Gestarteter WebSphere Server und Übersichts Seite

2.2 Programmierung der EJB

2.2.1 Anlegen eines neuen EJB Projektes

Zur Entwicklung der Java EE Applikation benutzen wir die Java EE Perspektive von RDz. Falls diese noch nicht aktiv ist müssen wir nun wechseln:

- Menü *Window → Open Perspective → Other → Java EE*

Als nächstes legen wir ein neues EJB Projekt an, das die Enterprise Bean sowie deren Interfacedefinition enthält. Wir starten den *EJB Project Wizard* mit:

- Menü *File -> New -> Other -> EJB -> EJB project* öffnet den *New EJB Project* Wizard (Abb. 4). Hier wählen wir *EJB Project* aus und bestätigen mit *Next*.
- Auf der nächsten Dialogseite nehmen wir folgende Einstellungen vor:
  Project name: EJBConverterSample
  Target Runtime: WebSphere Application Server v7.0
  EJB Module version: 3.0
  Configuration: Default Configuration
  EAR Mebership: "Add project to an EAR" aktivieren
  Der EAR Project Name wurde automatisch auf EJBConverterSampleEAR ergänzt.
- Bestätigung der Angaben mit *Next*.

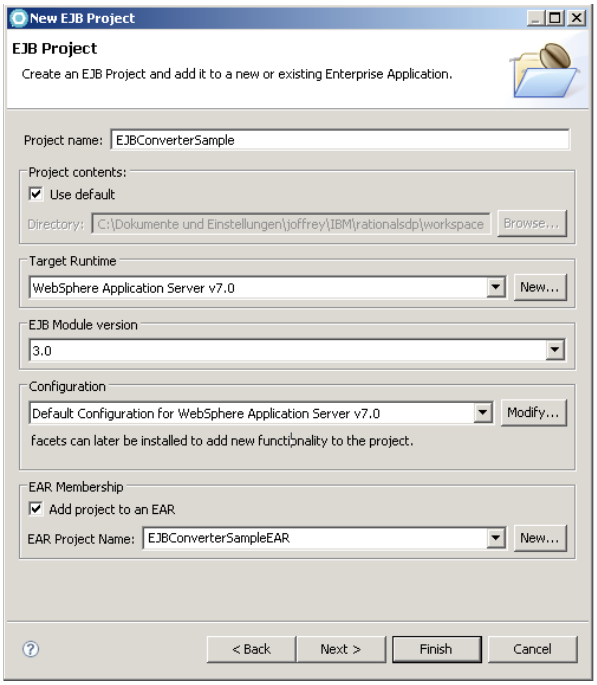

Abbildung 4: Ein neues EJB Projekt anlegen

- Auf der nächsten Dialogseite (Abb. 5) deaktivieren wir *Create an EJB Client*

JAR module to hold the client interface and classes und *Generate deployment descriptor*. Als Source Folder übernehmen wir den vorgeschlagenen Wert *ejbModule*

- Mit *Finish* schliessen wir den Wizard ab.

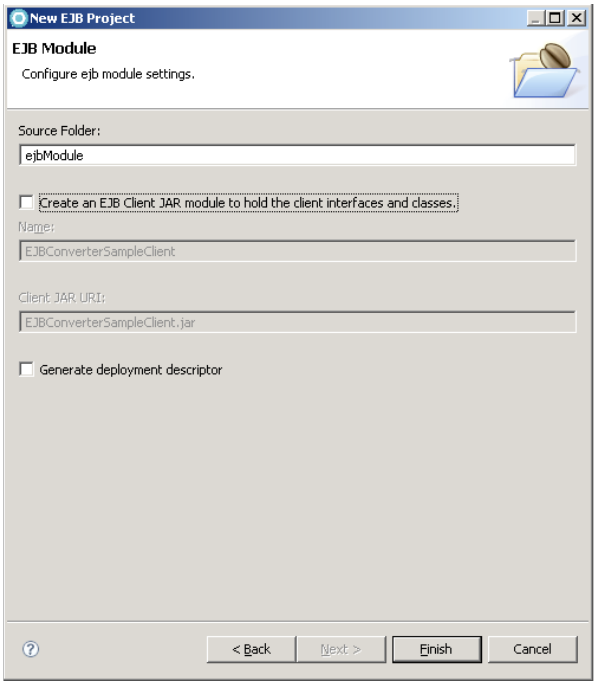

Abbildung 5: EJB Konfiguration

Im Projekt Explorer erscheinen nun zwei neue Projekte, *EJBConverterSample* und *EJBConverterSampleEAR*. Im Folgenden werden wir eine neue EJB zum *EJBConverterSample* EJB Projekt hinzufügen. Durch die Verknüpfung zu einer EAR (durch das Kontrollfeld *EAR Membership*, s.o.) können wir später eine einzige monolithische Enterprise Applikation erstellen, die als Container für EJBs und Servlets der Webapplikation fungiert.

2.2.2 Definition der Interfaces

Das neue Interface legen wir an, indem wir im Project Explorer das Kontextmenue durch

- Rechtskick auf *EJBConverterSample* und aus Kontextmenue *New -> Interface* wählen.
- Den erscheinenden Dialog (Abb. 6) füllen wir wie folgt aus:
  Package: org.wsi.websphere.tutorial.ejbsample.converter,
  Name: Converter
  Die weiteren Optionen übernehmen wir wie vorgeschlagen.
- Die Angaben bestätigen wir mit *Finish*.

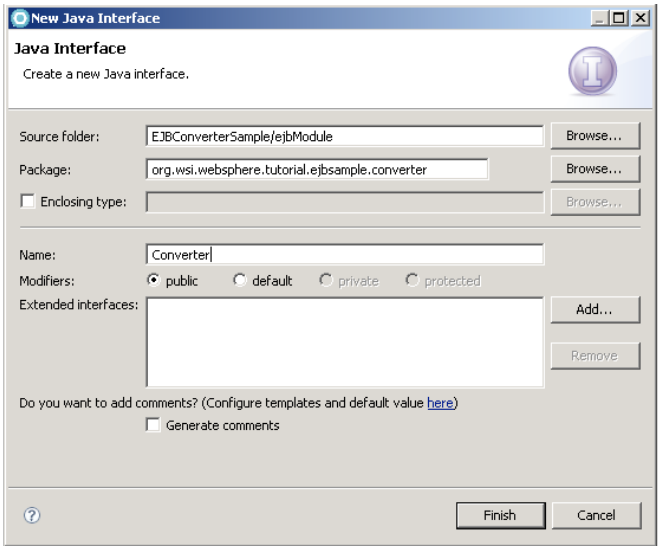

Abbildung 6: New Java Interface Dialog

Die erstellte Interface-Datei, `Converter.java`, wird automatisch im Source Editor Fenster geöffnet. `Converter.java` befindet sich im Project Explorer unter *EJBConverterSample → ejbModule → org.wsi.websphere.tutorial.ejbsample.converter → Converter.java* und kann auch durch Doppelklick geöffnet werden.

Im Source Editor für `Converter.java` ergänzen wir folgenden Code:

```
package org.wsi.websphere.tutorial.ejbsample.converter;

import javax.ejb.Remote;

@Remote
public interface Converter {
    public float fahrenheitToCelsius(float degreeFahrenheit);
    public float celsiusToFahrenheit(float degreeCelsius);
}
```

Converter.java

2.2.3 Implementierung der Enterprise JavaBean

Als ersten Schritt ergänzen wir *EJBConverterSample* um eine neue Klasse, `ConverterBean.java` welche die Business Logik unserer Anwendung enthält und die beiden Methoden aus der Interfacedefinition implementiert.

- Im Project Explorer: Rechtsklick auf *EJBConverterSample*
- Aus dem Kontextmenü *New -> Class* auswählen
  Wir uebernehmen die Vorgeschlagenen Werte und geben als Name *ConverterBean* an (Abb. 7)
- Mit *Finish* lassen wir uns eine neue *ConverterBean* Klasse erstellen.

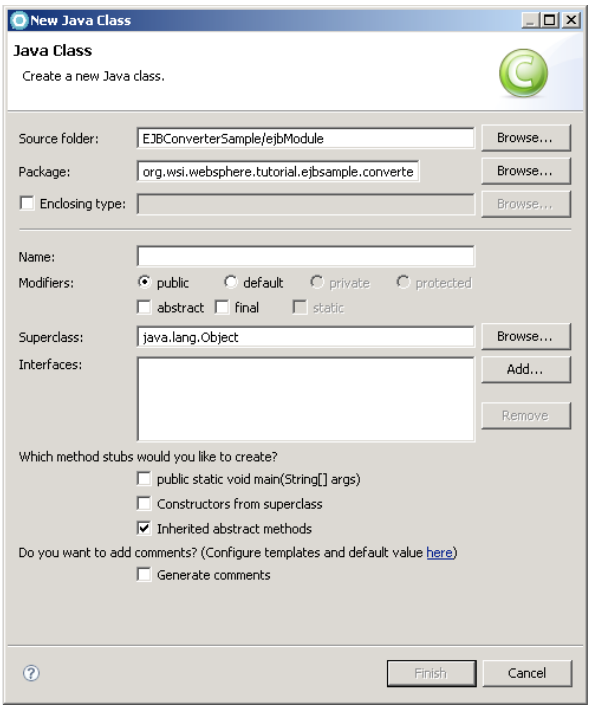

Abbildung 7: Neue Klasse ConverterBean erstellen

Die Klasse `ConverterBean.java` befindet sich, genauso wie das Interface `Converter.java` unter

- *EJBConverterSample → ejbModule → org.wsi.websphere.tutorial.ejbsample.converter → ConverterBean.java*

Der Source Editor für `ConverterBean.java` öffnet sich automatisch und wir ergänzen den Source Code wie im Listing *ConverterBean.java* angegeben.

```
package org.wsi.websphere.tutorial.ejbsample.converter;
import javax.ejb.*;

@Stateless
public class ConverterBean implements Converter {
    public float celsiusToFahrenheit(float degreeCelsius) {
        float result = degreeCelsius * 9/5 + 32;
        return (float)Math.round(result);
    }

    public float fahrenheitToCelsius(float degreeFahrenheit) {
        float result = (degreeFahrenheit - 32) * 5/9;
        return (float)Math.round(result);
    }
}
```

ConverterBean.java

`ConverterBean.java` und `Converter.java` können nun gespeichert und geschlossen werden.

2.2.4 Erstellen des Servlets

Zunächst erstellen wir ein neues Projekt, das die Artifakte für die Präsentationsschicht enthält, nämlich das Servlet und die zugehörige JSP.

- Menü *File -> New -> Other -> Web → Dynamic Web Project*, auswählen und *Next* klicken
  Als *Project name:* geben wir ConverterWeb an (Abb. 8).
- Wir aktivieren *Add project to an EAR* und wählen *EJBConverterSampleEAR* damit auch das Web Projekt Teil unserer Enterprise Applikation wird.
- Mit *Finish* beenden wir den Wizard.

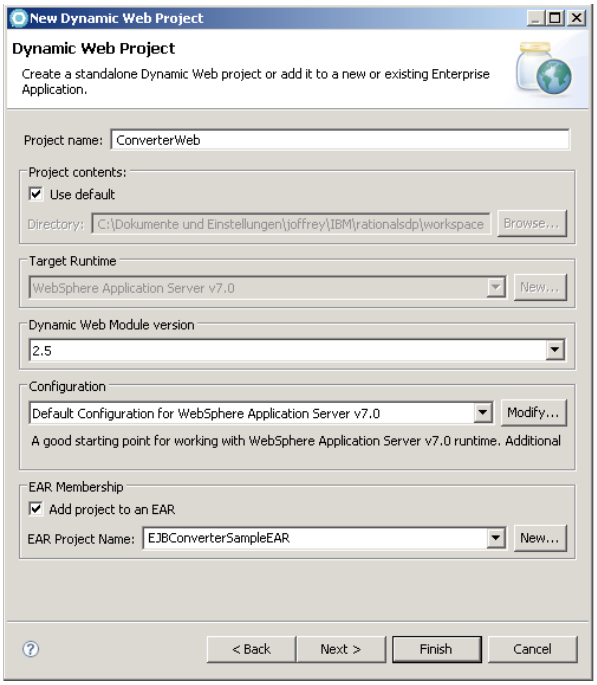

Abbildung 8: Erstellen eines neuen Web Projektes

Das neue Projekt *ConverterWeb* ist nun im *Project Explorer* verfügbar. In diesem Projekt erstellen wir nun ein neues Servlet (Abb. 9):

- Rechtsklick auf *ConverterWeb*, Kontextmenue *New -> Other ...* wählen.
- Im Dialog selektieren wir *Web -> Servlet* und bestätigen mit *Next*
  Als *Java package:* geben wir org.wsi.websphere.tutorial.ejbsample.converter an und als *Class name:* wählen wir ConverterServlet
- Mit *Finish* wird das Skeleton für unser neues Servlet generiert.

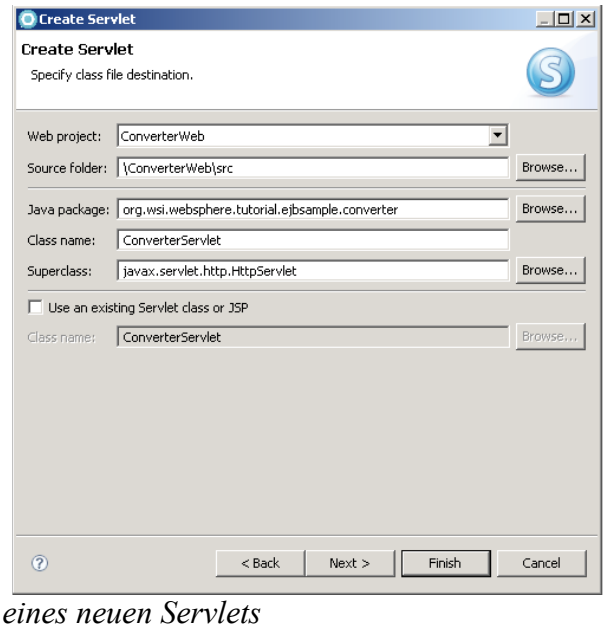

Abbildung 9: Erstellen eines neuen Servlets

Es wird die Datei `ConverterServlet.java` erzeugt, die im Project Explorer unter *ConverterWeb → Java Resources → src → org.wsi.websphere.tutorial.ejbsample.converter → ConverterServlet.java* zu finden ist.

Wir ersetzten den Source Code des Servlets durch das folgende Listing.

```
package org.wsi.websphere.tutorial.ejbsample.converter;

import java.io.IOException;
import javax.ejb.EJB;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
* Servlet implementation class ConverterServlet
*/
public class ConverterServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    // Use injection to get the ejb
    @EJB private static Converter converter;

    public ConverterServlet() {
        super();
    }

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        String msg = null;
        request.setAttribute("msg", msg);
        RequestDispatcher rd =
          getServletContext().getRequestDispatcher("/Converter.jsp");

        rd.forward(request, response);
    }

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response)
```

```
            throws ServletException, IOException {
        String msg = null;
        float param01;
        try {
            param01 = Float.valueOf(
                request.getParameter("param01").trim()).floatValue();
            try {
                float degreeCelsius =
                     converter.fahrenheitToCelsius(param01);
                msg = param01 + "F eq " + degreeCelsius + " C";

                msg += "<br>";
                float degreeFahrenheit =
                    converter.celsiusToFahrenheit(param01);
                msg += param01 + "C eq " + degreeFahrenheit + " F";
         } catch (Exception e) {
             msg = "Exception caught: " + e.toString();
         }
    } catch (NumberFormatException e) {
        msg = "Exception caught: " + e.toString();         }
        // Set attributes and dispatch the JSP.
        request.setAttribute("msg", msg);
        RequestDispatcher rd =
          getServletContext().getRequestDispatcher("/Converter.jsp");

        rd.forward(request, response);
    }
}
```

ConverterServlet.java

Im Source Code Editor werden nun Fehler angezeigt, dass das Objekt *Converter* nicht aufgelöst werden kann. Dies liegt daran, dass wir das Webprojekt noch nicht mit der EJB verknüpft haben (*Membership EAR* im Dialog weiter oben sagt nur aus, dass das EAR abhängig vom *ConverterWeb* Projekt ist, allerdings wird die Abhängigkeit nicht in die andere Richtung aufgelöst). Dies werden wir nun vornehmen:

- Rechtsklick auf *ConverterWeb*, aus dem Kontextmenü *Properties* wählen
- Im Properties Dialog (Abb. 10) *Java EE Module Dependencies* selektieren und im Reiter *J2EE Modules* EJBConverterSample.jar aktivieren und Dialog mit *OK* beenden.

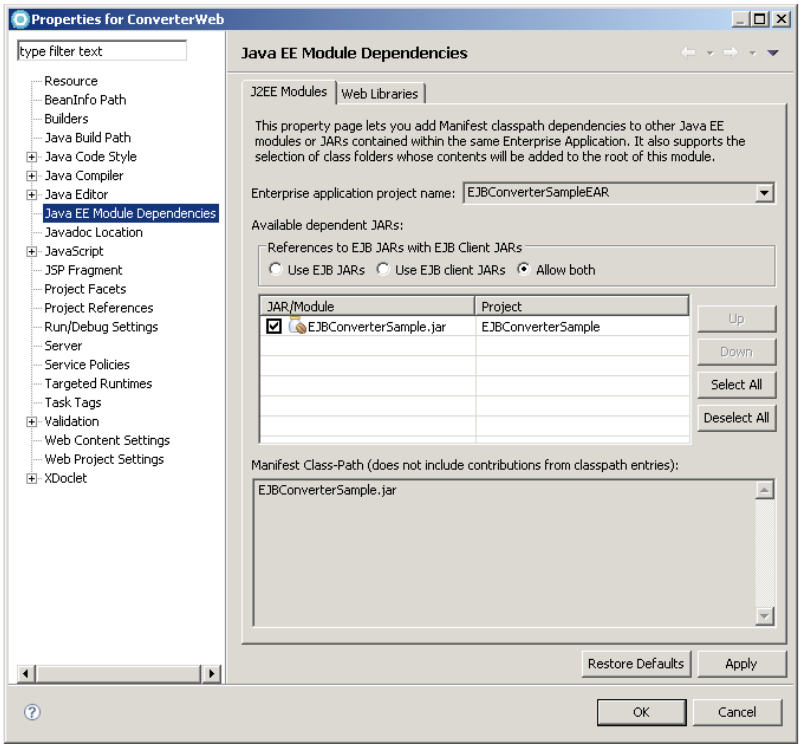

Abbildung 10: Abhängigkeiten des ConverterWeb Projektes definieren

2.2.5 JSP erstellen

Wir fügen dem *ConverterWeb* Projekt nun ein JSP hinzu:

- Rechtsklick auf *ConverterWeb*, Kontextmenü *New -> Other* wählen
- *Web -> JSP Wizard* wählen
- Als *File name:* wählen wir `Converter.jsp` und benden den Wizard mit *Finish*.

Die Datei Converter.jsp ist im Project Explorer unter *ConverterWeb -> WebContenet -> Converter.jsp* zu finden. Das JSP passen wir nun wie folgt an:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%> <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<FORM METHOD=POST ACTION="ConverterServlet">
<BR/>
<%
    response.setHeader("Pragma", "No-cache");
    response.setHeader("Cache-Control", "no-cache");
    response.setDateHeader("Expires",0);
    String msg = (String) request.getAttribute("msg");
    if (msg == null) msg = "";
%>
<B>Click on the button to receive server message</B>
<BR/><BR/>
<input type=text size=20 name="param01" />
<INPUT TYPE=SUBMIT VALUE="Convert">
</FORM>
<H3><%=msg%></H3>
</BODY>
</HTML>
```

Tabelle 1: Converter.jsp

2.2.6 Zusammenfassung der erstellten Artifakte

Bis jetzt haben wir die EJB, das dazugehörige Interface, das Servlet sowie die JSP erstellt. Durch die Verknüpfungen mit *EJBConverterSampleEAR* stehen uns nun auch das *EJBConverterSample.jar* und *ConverterWeb.war* unter *EJBConverterSampleEAR* → *Modules* im Project Explorer zur Verfügung. Der Projektbaum des Project Explorers sollte nun wie in Abbildung 11 aussehen.

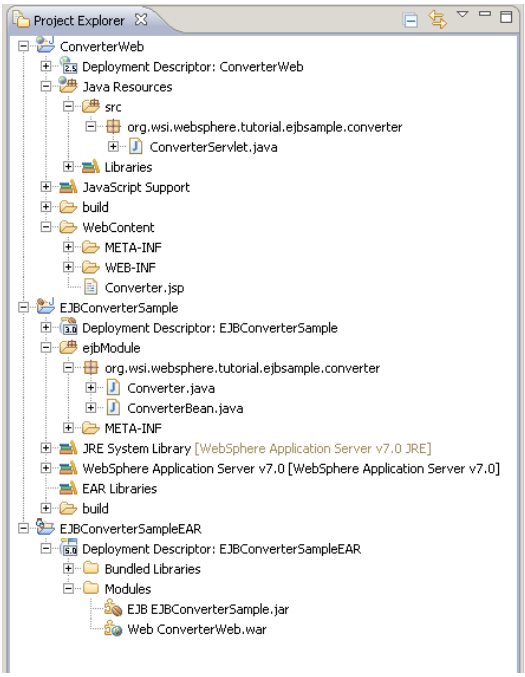

Abbildung 11: Project Explorer mit den neu erstellten Artifakten

2.2.7 Deployen der Applikation

- Im Project Explorer zu *ConverterServlet.java* navigieren (befindet sich unter *ConverterWeb -> Java Resources -> src -> ConverterServlet.java*)
- Rechtsklick auf *ConverterServlet.java*
  Run As -> Run on Server aus dem Kontexmenü wählen,
  WebSphere Application Server v7.0 at localhost aus Liste auswählen
- Mit *Finish* wird die Webapplikation auf dem WebSphere Server installiert.

Die Webapplikation wird in einem Browserfenster gestartet und kann getestet werden (siehe Abb. 12).

Abbildung 12: Die laufende Web Applikation

Falls notwendig, kann die Applikation kann wie folgt vom Server wieder entfernen

werden:

- Server View aktivieren: Menü *Window -> Show view -> Server -> Servers*
- *WebSphere Application Server v7.0 at localhost* expandieren
- Rechtsklick auf *EJBConverterSampleEAR*, Kontextmenü *Remove* wählen
- Nachfrage mit *OK* bestätigen

2.2.8 Exportieren der Applikation

Die Enterprise Applikation kann nun als Enterprise Archive (EAR) exportiert werden. Dieses EAR enthält alle notwendigen Komponenten für die Webapplikation inklusive der EJB (*ConverterWeb*, *EJBConverterSample* und *EJBConverterSampleEAR*).

- Rechtsklick auf *EJBConverterSampleEAR* im Project Explorer
  Kontextmenü *Export -> EAR file* wählen
- Im Dialog folgende Angaben machen:
  EAR project: EJBConverterSampleEAR
  Destination: C:\EJBConverterSampleEAR.ear
  Export source files und *Overwrite existing file* aktivieren.

Dieses EAR File wird im nächsten Kapitel für die Entwicklung der Web Services benötigt falls dieses Kapitel übersprungen worden ist. Das EAR ist auch auf der beiliegenden CD enthalten.

3 Erstellen der Web Services

In diesem Teil des Tutorials werden wir Methoden der in Kapitel 2 erstellten Enterprise Java Bean in Web Services transformieren. Dazu verwenden wir die von RDz bereitgestellten Wizards, die uns auf der Basis des bottom-up Ansatzes helfen, EJB Interfaces als Web Services bereitzustellen sowie zugehörige WSDLs automatisch zu generieren.

3.1 Vorraussetzungen

Für dieses Kapitel benötigen wir einen laufenden WebSphere Applikation Server sowie eine auf diesem Server installierte und veröffentlichte Instanz der Beispielapplikation aus Kapitel 2.

3.1.1 Importieren der EJB Beispielapplikation in den Eclipse Workspace

Falls im Project Explorer die beiden Projekte *ConverterBean* und *EJBConverterSample* und *ConverterWeb* nicht verfügbar sind müssen diese importiert werden. Beide Projekte sind auf der beiliegenden CD enthalten.

3.1.2 Starten des WebSphere Applikation Servers

Falls der WebSphere Application Server noch nicht gestartet wurde, so muss dies nun an dieser Stelle nachgeholt werden. Dazu öffnen wir die Server view:

- *Window → Show Views → Server*

Hat der Servereintrag *Websphere Applikation Server v7* den State *Stopped* starten wir den Server mittels *Rechtsklick → Start.* Der State sollte nun auf *Started* stehen.

3.1.3 Installieren der Beispielapplikation

Die Beispielapplikation *EJBConverterSample.ear* ist auf der CD enthalten. Um diese zu importieren wählen wir das Menü *File → Import ...* , selektieren im Import Wizard *Java EE → EAR file*. Auf der nächsten Dialogseite wählen wir das zu importierende EAR und beenden den Wizard mit *Finish.* Es stehen nun im Project Explorer drei Projekte zur Verfügung, die Webapplikation *ConverterWeb*, das EJB Projekt

EJBConverterSample und das Enterprise Application Archive *EJBConverterSampleEar*.

Um die Enterprise Applikation auf dem Applikationsserver zu deployen wählen wir aus dem Project Explorer *EJBConverterSample* und klicken auf *Run as... → Run in Server*.

3.2 Erstellen des Web Services

3.2.1 Web Service Wizard

Der Web Service Wizard erlaubt uns den Grad der automatisierten Erstellung der Web Services zu bestimmen. Die verschiedenen Entwicklungslevel, geordnet nach Ihrer Granularität, sind [RAD07, S.822]:

- Develop: Entwickelt die WSDL Definition und die Implementation des Web Services sowie die notwendigen Module und Java Dateien.
- Assemble: Verknüpfungen mit dem verwendeten EAR Projekt werden erstellt.
- Deploy: Generiert den deployment Code (Proxies) und Deskriptoren
- Install: Installiert das Web Module sowie das EAR auf dem Applikations-Server
- Start: Nach der Installation wird der Web Service gestartet.
- Test: Es wird zusätlich der Web Service Explorer gestartet.

Ab dem Entwicklungslevel *Install service* besteht zusätzlich die Möglichkeit, einen Client zu generieren. Auch hier stehen wieder verschiedene Entwicklungslevel zur Auswahl, welche wir in Kaptil 4 besprechen werden.

In diesem Tutorial werden wir den Wizard verwenden um nur die Basis des Web Services zu generieren (Level 2, *Assemble Service*). Auch werden wir keinen Client automatisch erzeugen, da wir in einem späteren Kapital einen eigenen Thin Client entwickeln werden.

Ausgehend von der *ConverterBean* Session Bean starten wir den Web Service Wizard:

- Wir navigieren Im Projekt Explorer zu der *ConverterBean*:
  EJBConverterSample → Deployment Descriptor → Session Beans → ConverterBean
- Durch Rechtsklick auf *ConverterBean* wählen wir *Web Services → Create Web service* aus dem Kontextmenue. Es wird der Wizard gestartet.
- Im Web Services Dialog (Abb. 13) nehmen wir folgende Einstellungen vor:
  Web service type: Bottom up
  Service level: Assemble service
  Configuration: Wie vorgeben.

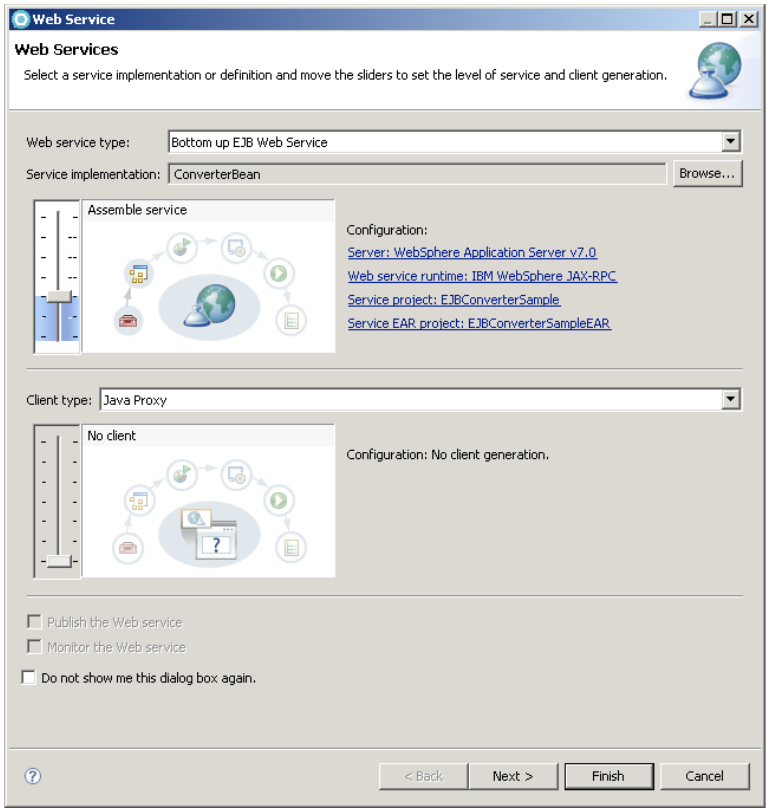

Abbildung 13: Der Create Web Services Wizard

- Wir bestätigen durch Klick auf *Next* und gelangen auf die 2. Dialog Seite, *Web Service EJB Configuration* (Abb. 14). Wir machen hier folgende Einstellungen:

  Use an existing service endpoint interface deaktivieren.
  HTTP router: ConverterWebRouter. Dies ist ein dynamisches Web Projekt, das die Web Service Configuration speichert und die Kommunikation zwischen EJB auf dem Server und dem Service uebernimmt.
  Bindings router: HTTP aktivieren

  Für alle weiteren Einstellung übernehmen wir die Standardeinstellungen und bestätigen den Dialog mit *Next*.

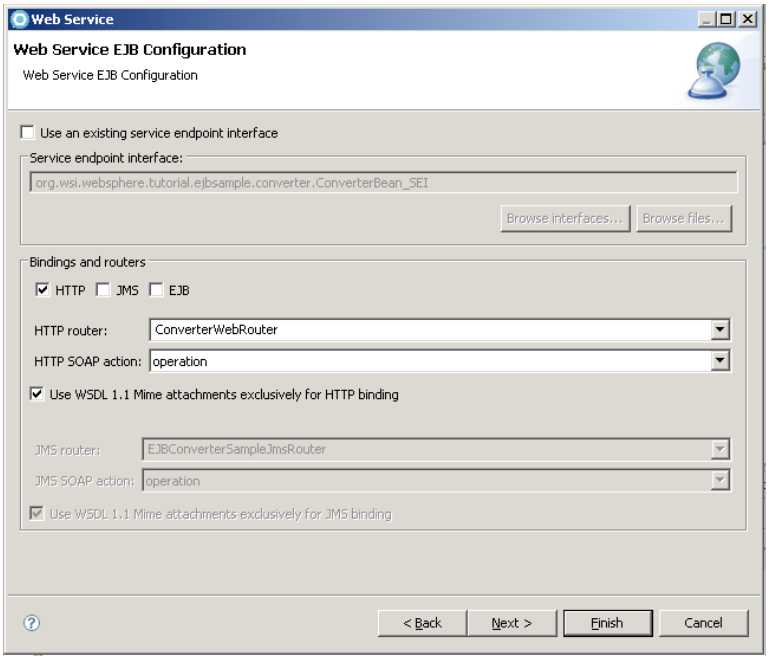

Abbildung 14: Web Service EJB Konfiguration

- Auf der Seite *Configure the Java bean as a Web service* (Abb. 15) definieren wir den Namen der WSDL Datei sowie die Methoden, die als Web Service bereitgestellt werden sollen:
  WSDL port name: ConverterBean
  WSDL file: ConverterBean.wsdl

  Wir selektieren beide methoden, *celsiusToFahrenheit(float)* und *fahrenheitToCelsius(float)*, da wir beide in unserem Client verwenden möchten.

  Die weiteren Einstellungen, *Void to return: two way*, *Style and use: document / literal,* und *Security configuration: no security* belassen wir auf den Standardeinstellungen. Mit *Next* übernehmen wir die Einstellungen.

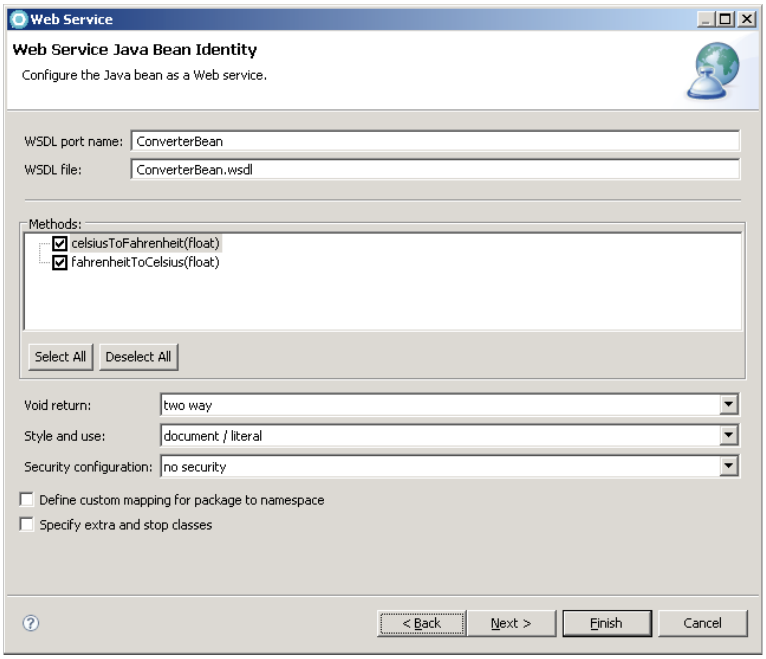

Abbildung 15: Web Service Java Bean Identity

- Auf der letzten Dialogseite, *Web Service Publication* deselektieren wir beide Optionen, *Launch the Web Services Explorer to publish to Unit test UDDI* und *Launch the Web Services Explorer to publish to UDDI.*
- Der Web Service Wizard kann nun mit *Finish* abgeschlossen werden.

3.2.2 Generierte Dateien

Das Projekt JSR-109 Web Services wurde neu angelegt und enthält die neu generierte WSDL (*JSR-109 Web Services -> Services -> ConverterBeanService*). Diese ist auch unter *EJBConverterSample -> ejbModule -> META-INF -> wsdl -> ConverterBean.wsdl* zu finden.

Die WSDL wird auch dem *EJBConverterSample* hinzugefügt und befindet sich unter *EJBConverterSample → ejbModule → META-INF → wsdl → ConverterBean.wsdl*

Weiter wird das Service Endpoint Interface, EJBConverterSample → ejbModule → org.wsi.websphere.tutorial.ejbsample.converter → *Converter_SEI.java* angelegt. Dies ist das Java interface, welches durch den Web Service implementiert wird [RAD07, S.828].

3.2.3 WSDL Editor

Als nächstes inspizieren wir die WSDL im WSDL Editor:

- Rechtsklick auf *EJBConverterSample -> ejbModule -> META-INF -> wsdl -> ConverterBean.wsdl*
- *Open with → XML Editor* wählen

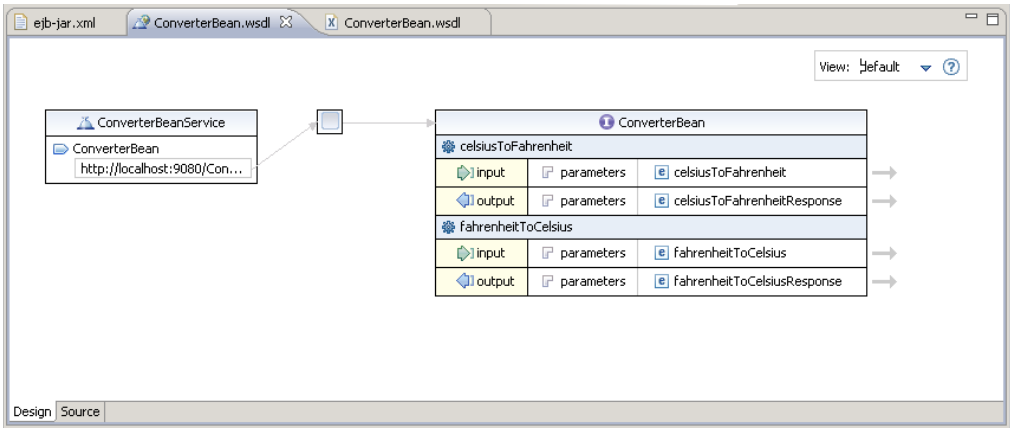

Abbildung 16: Die Web Services im WSDL Editor

In Abbildung 16 sehen wir, dass für unserer beiden Methoden Web Service Definition mit Ein- und Ausgabeparameter erstellt wurden. Nähere Informationen zum Web Service bekommen wir, indem wir mittels Rechtsklick auf *ConverterBean* im Kasten *ConverterBeanService* aus dem Kontextmenue *Properties* auswählen. Es üffnet sich nun eine neue View mit näheren Informationen, unter anderem dem verwendeten Endpoint, hier *http://localhost:9080/ConverterWebRouter/services/ConverterBean.* Diese URL benötigen wir im nächsten Abschnitt, Testen der Web Services.

4 Testen der Web Services

Nach der Erstellung der Services werden sollte die automatisch erzeugte WSDL sowie die Services validiert und getestet werden. Hilfe eines Webbrowsers und des – in RDz integrierten – Web Service Explorers testen. Weiter werden wir die SOAP Nachrichten auf HTTP Protokollebene mit Hilfe des TCP-IP Monitors beobachten.

4.1 Deployment der Web Services

Bevor wir mit dem Testen beginnen können, muss das komplette Projekt zuerst auf dem Server deployt werden. Dazu öffnen wir die Server View (*Menu Window -> Show View -> Server*), aktivieren mit Rechtsklick auf *WebSphere Application Server v7.0 at localhost* das Kontextmenü und wählen *Add and Remove Project...* . Im Dialog selektieren wir *EJBConverterSampleEAR* und fügen das EAR mit *Add >* zum Server hinzu. Den Dialog beenden wir mit *Finish*.

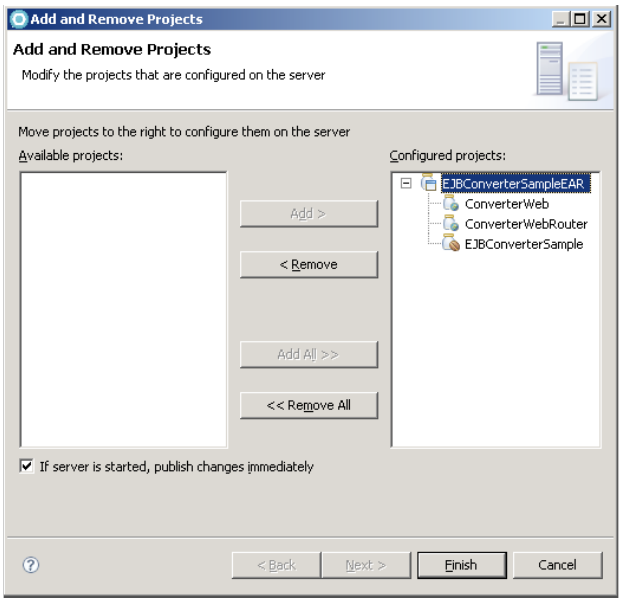

Abbildung 17: WebSphere Projektverwaltung

Wir überprüfen nun, ob die Komponenten erfolgreich auf dem Server installiert wurden. Dazu navigieren wir zur Server View (*Window -> View -> Server -> Servers*), und klappen den Einträge *WebSphere Application Server v7.0 at localhost* sowie *EJBConverterSampleEAR* auf. Es sollten nun drei Projekte sichtbar sein, *ConverterWeb*, *ConverterWebRouter*, und *EJBConverterSample* (Abb. 18).

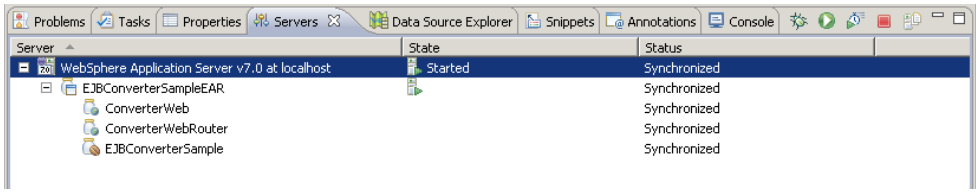

Abbildung 18: Applikation Server mit installierten Projekten

4.2 Validieren der WSDL

Wir expandieren im Project Explorer *EJBConverterSample → ejbModule → META-INF → wsdl*. Druch Rechtsklick auf *ConverterBean.wsdl* wählen wir aus dem Kontextmenü *Validate*. Ist die WSDL konsistent und enthält keine Fehler, sollte ein Fenster mit der Meldung *The validation completed with no errors or warnings“* erscheinen.

4.3 Verfügbarkeit der Web Services und WSDL überprüfen

Zunächst überprüfen wir, ob die Web Services und die WSDl erfolgreich auf dem Server deployt werden konnten. Hierzu verlassen wir RDz und starten einen externen Browser mit dem wir die URL des Web Services und der WSDL ansteuern werden. Die URL des Web Services ist, wie wir in Abschnitt 3.2.3 (WSDL Editor) mit Hilfe des Web Services Explorer herausgefunden haben:

- http://localhost:9080/ConverterWebRouter/services/ConverterBean

Abbildung 19: Testen der Verfügbarkeit des Web Services mittels eines externen Browsers

Die WSDL können wir aufrufen, indem wir *?WSDL* an die URL des Web Services anfügen (Abb. 20). Dies wird vom Application Server zur kompletten URL der WSDL interpoliert.

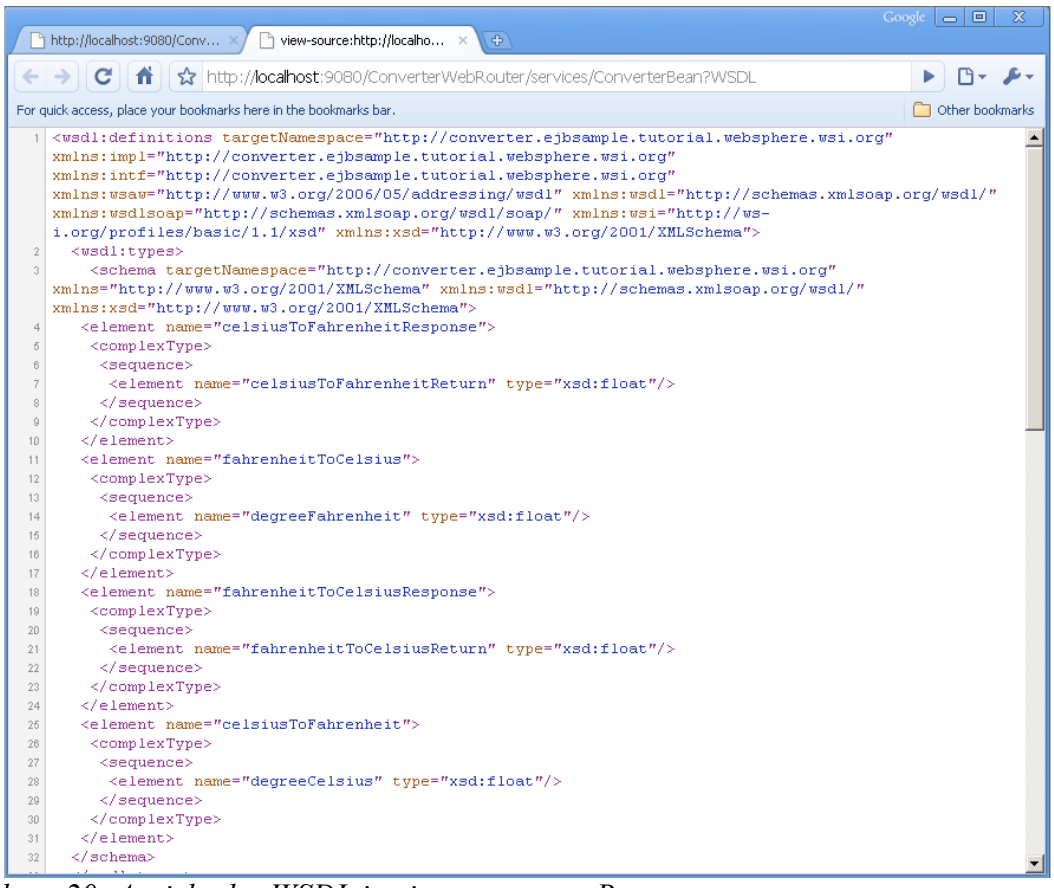

Abbildung 20: Ansicht der WSDL in einem externen Browser

4.4 Testen der Services mit dem Web Service Explorer

Nachdem wir die Verfügbarkeit der Web Services festgestellt haben, kehren wir wieder zu RDz zurück und testen die Services mit dem Web Services Explorer.

Diesen starten wir durch Rechtsklick auf *JSR-109 Web Services -> Services -> ConverterBeanService* und wählen *Test with Web Services* Explorer aus dem Kontextmenü. Alternativ können wir auch *EJBConverterSample – ejbModule → META-INF → wsdl →* ConverterBean.wsdl expandieren und aus dem Kontextmenü *Web Services → Test with Web Services Explorer* wählen. Es öffnet sich der Web Services Explorer mit den zwei geladenen Services, *fahrenheitToCelsius* und *celsiusToFahrenheit*. Im Status Panel sollte folgende Meldung erscheinen (Abb. 21).

> *IWAB0381I platform:/resource/EJBConverterSample/ejbModule/META-INF/wsdl/ConverterBean.wsdl was successfully opened.*

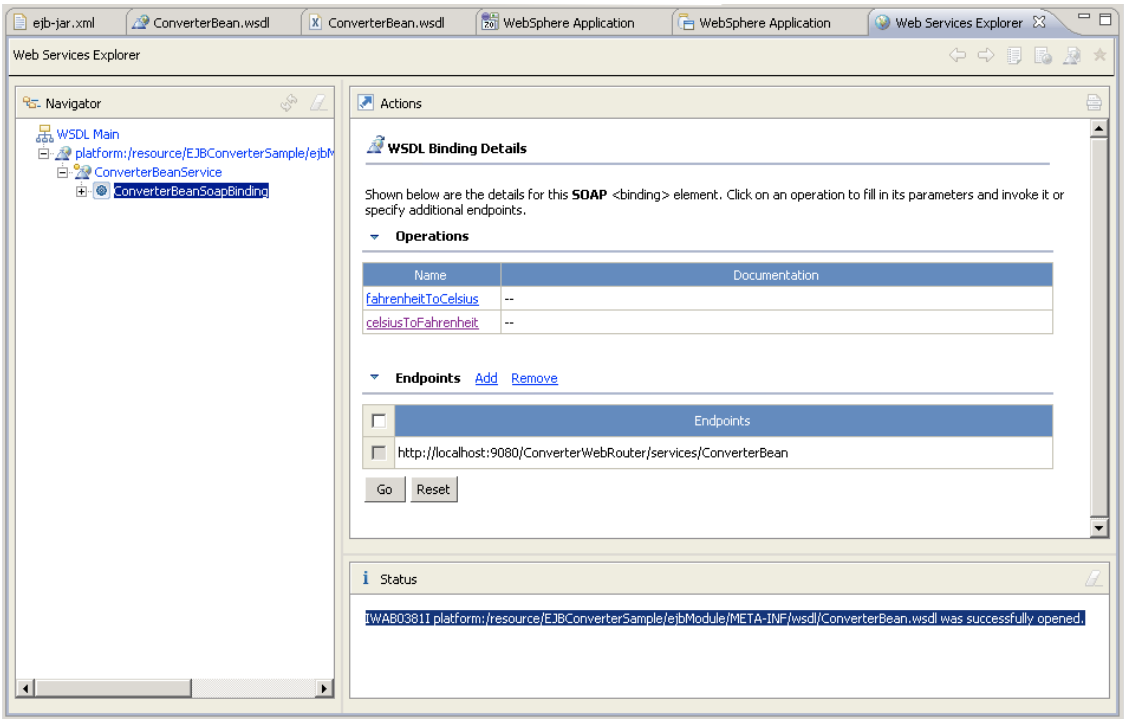

Abbildung 21: Web Services Explorer mit erfolgreich geladenen Services

Hinweis: Bei der Meldung *IWAB0135E An unexpected error has occurred. java.net.ConnectException Connection refused: connect* ist wahrscheinlich der Server gestoppt oder ConverterRouter nicht deployt.

Wir können nun die Services testen. Hierzu klappen wir im Navigator des Web Services Explorer *ConverterBeanService → ConverterBeanSoapBinding* auf und selektieren den Service *celsiusToFahrenheit*. Im Actions Panel belassen wir den Endpoint, geben eine Temperatur in das Eingabefeld ein und schicken den Request mit *Go* ab. Im Status Panel sehen wir nun die Response des Services, also die in

Fahrenheit umgerechnete Temperatur (Abb. 22).

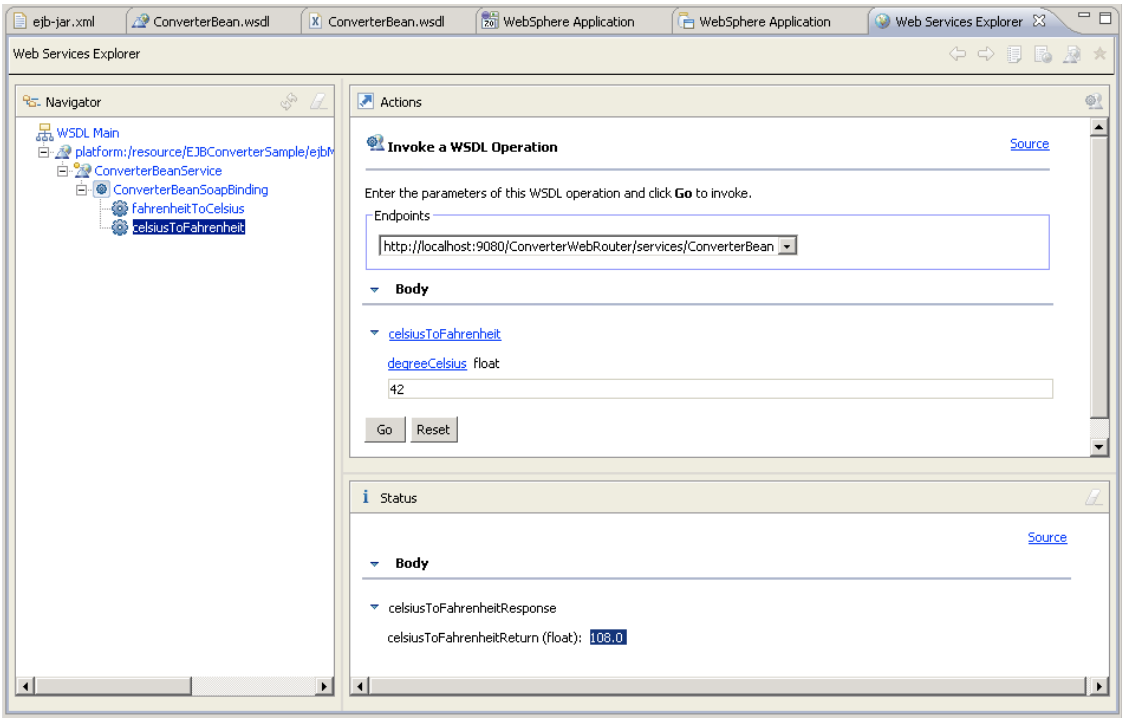

Abbildung 22: Erfolgreiche Response des Web Services

4.5 Testen der Web Services mit dem TCP/IP Monitor

Mittels des TCP/IP Monitor können wir die HTTP Requests / Response Elemente auf Protokollschicht beobachten. Dies kann sehr hilfreich sein, falls es zu Problemen bei dem Aufruf der Web Services oder bei der Rückgabeantwort kommen sollte. Auch wenn nachträglich das Transportprotokoll z.B. auf HTTPS geändert werden sollte, ist der TCP/IP Monitor ein nützliches Tool um die Funktionalität der Änderungen zu Testen [Cui08].

Als erstes müssen wir einen neuen Monitor einrichten. Hierzu öffnen wir die RDz Voreinstellungen (Menü *Window* → *Preferences*) und wählen *Run/Debug* → *TCP/IP Monitor*. Mit *Add...* öffnen wir den *Edit Monitor* Dialog. Wir machen folgende Angabe und bestätigen den Dialog dann mit *OK* (Abb. 23).

- *Local monitoring port:* 9081
- *Host name:* localhost
- *Port:* 9080
- *Type:* TCP/IP
- *Start monitor automatically* aktivieren

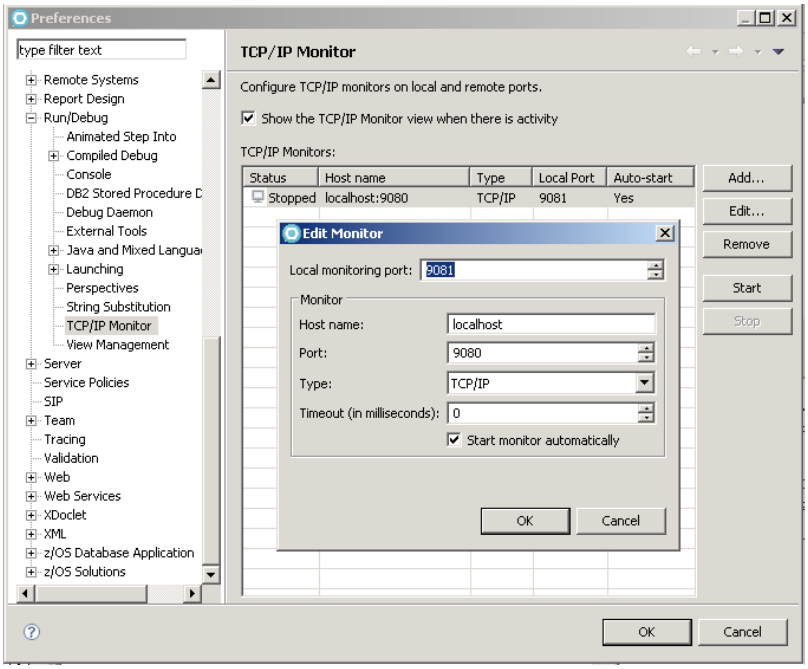

Abbildung 23: Einrichten eines neuen TCP/IP Monitors

Wir öffnen den TCP/IP Monitor indem wir die View über das Menu *Window → Show View → Other, Debug → TCP/IP Monitor* aktivieren. Als nächstes starten wir den Web Services Explorer und schicken einen Testrequest ab. Im TCP/IP Monitor sehen wir nun auf der rechten Seite den HTTP Request sowie die Response auf der linken Seite (Abb. 24).

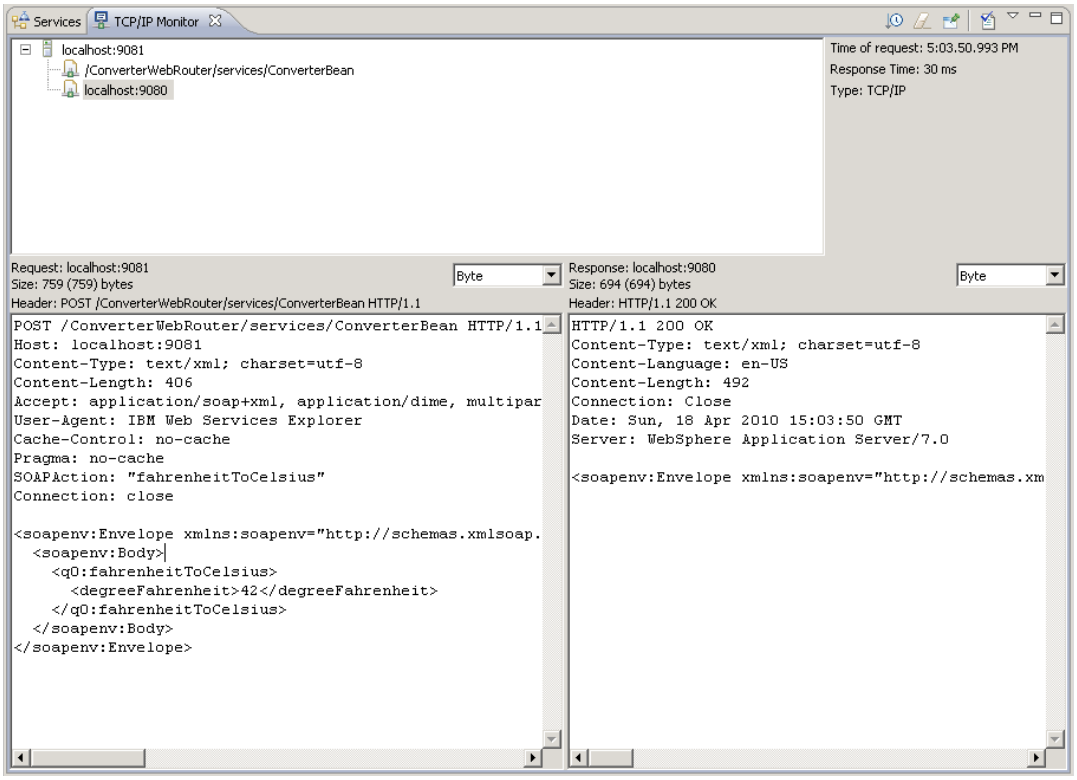

Abbildung 24: HTTP Request/Response Monitoring mit dem TCP/IP Monitor

Nach erfolgreichem Testen können wir den TCP/IP Monitor wieder deaktiveren. Hierzu öffenen Wir den Eintrag *Run/Debug → TCP/IP Monitor* aus den RDz Voreinstellungen (*Window → Preferences*) und löschen den von uns angelegte TCP/IP Monitor (Selektieren und *Remove* klicken).

5 Entwicklung eines Java Konsumenten (Thin Client)

Für die in Kapitel 3 erstellten Web Services werden wir nun einen Java Client entwickeln, um zu demonstrieren wie Web Services aus einer anderen JVM heraus, ausserhalb des Application Containers, konsumiert werden können. Dazu erstellen wir ein eigenständiges Java Projekt, das einen Java Command Line Client enthält und Stapelanfragen an die Web Services bearbeitet.

5.1 Client-Projekt Erstellen

Zuerst werden mit Hilfe der von RDz bereitgestellten Wizards und der Web Services Definition der WSDL die notwendigen Proxies generiert.

Wie auch bei der Erstellung der Web Services stehen uns bei der Clienterstellung mächtige Wizards zur Verfügung, die je nach Wunschlevel ein Client Skelett erstellen oder bis hin zum Testlevel alle Vorraussetzungen erfüllen, Code Snippets erstellen und die notwendigen administrativen Aktionen wie Deplozment und Starten des Clients übernehmen.

Mögliche Erstellungslevel stehen zur Verfügung [RAD07, S.831]:

- Develop Client
- Assemble Client
- Deploy Client
- Install Client
- Start Client
- Test Client

Wir wählen für unseren Client das Level *Deploy client*. Dieses Level generiert die nötigen Proxy Klassen welche im *Level Develop* und *Assemble* nicht verfügbar sind.

Wir starten den Web Services Wizard für den Web Service Client mit Rechtsklick auf *EJBConverterSample -> ejbModule -> META-INF -> wsdl -> ConverterBean.wsdl* und der Auswahl *Web Services -> Generate client*. Im Wizard (Abb. 25) setzen wir das Level auf *Deploy client*. In der Konfigurationssektion auf der linken Seite klicken wir auf *Client Project:* und nehmen folgende Änderungen vor (Abb. 26):

- *Client project:* JavaStandaloneClient
- *Client Project type:* Application Client Project
- *Client EAR project:* JavaStandaloneClientEAR

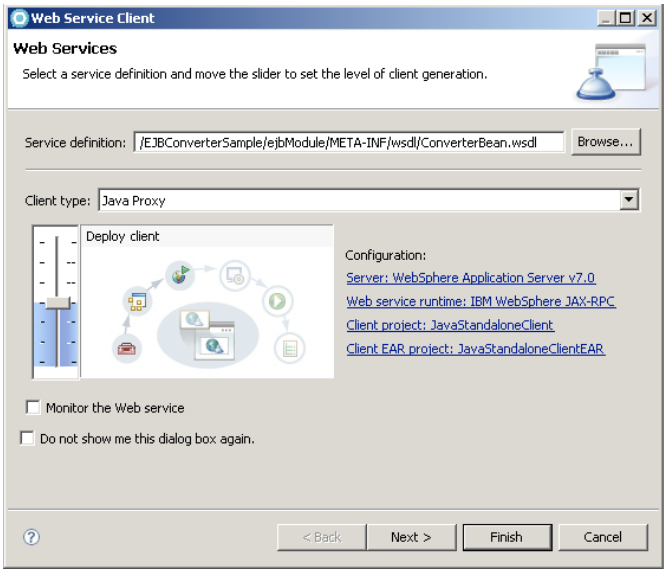

Abbildung 25: Der Web Service Client Wizard

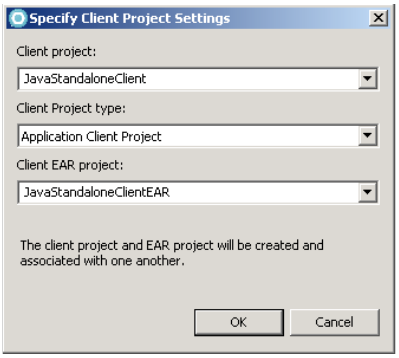

Abbildung 26: Web Service Client Projekt Einstellungen

Mit *Finish* verlassen wir den Wizard, und es wird das neue Projekt *JavaStandaloneClient* angelegt. Der Wizard startet im Hintergrund den *IBM Web services WSDL2Java emitter* und generiert aufgrund der WSDL eine Reihe von Java Klassen an welche für das Auffinden (Service Locator) und das SOAP Binding zuständig sind. In Abbildung 27 sind die erstellten Klassen zusammengefasst.

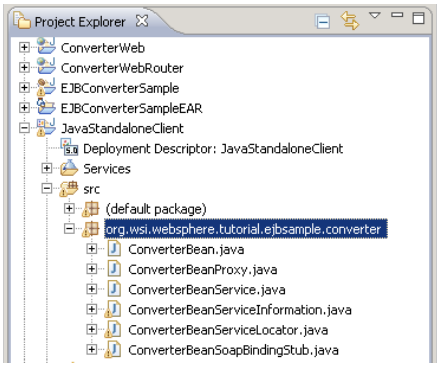

Abbildung 27: Die automatisch aus der WSDL erstellten Klassen für den JavaStandaloneClient

5.2 Implementierung des Clients

Wir erstellen nun eine neue Klasse und ein neues Package, das die Logik für unseren Client enthält sowie die Web Service Methoden aufruft. Dazu sind folgende Schritte notwendig:

- Im Projekt Explorer Rechtsklick auf *JavaStandaloneClient*
  Kontextmenue: *New -> Package*
- Im Dialog geben wir den Package-Name an und bestätigen mit *Finish*.
  Name: org.wsi.websphere.tutorial.client
- Um die neue Klasse anzulegen expandieren wir *JavaStandaloneClient → src*
  Rechtsklick auf *org.wsi.websphere.tutorial.client* und wählen aus dem Kontextmenü: *New -> Class*
- Als Klassenname wählen wir *ConverterClient* und schliessen den Wizard mit *Finish* ab.

Die Datei *JavaStandaloneClient → src → org.wsi.websphere.turorial.client → ConverterClient.java* wird nun im Source Editor geöffnet und wir ersetzten den Source Code mit folgendem Listing:

```
package org.wsi.websphere.tutorial.client;
import org.wsi.websphere.tutorial.ejbsample.converter.*;

public class ConverterClient {
    public static void main(String[] args) {
    try {
      ConverterBeanProxy proxy = new ConverterBeanProxy();
      System.out.println("Using endpoint: "+proxy.getEndpoint());
      System.out.println("Converting...");
      for(float i=13; i<24; i++) {
            float res = proxy.celsiusToFahrenheit(i);
            System.out.println(i + " Celsius = "+res+" Fahrenheit");
      }
    } catch (Exception e) {
        e.printStackTrace();
    }
    System.out.println("Finished.");
    } // end main()
}
```

ConverterClient.java

5.3 Starten des Clients in einer neuen JVM

Um die Appliaktion zu starten müssen wir zunächst eine neue Run Configuration erstellen und einige zusätzliche Bibliotheken integrieren [RADOH]. Die externen Bibliotheken (JAR Dateien) fügen wir mit folgenden Schritten zum Build Path hinzu:

- Rechtsklick auf *JavaStandaloneClient* im Project Explorer und *Properties* aus dem Kontextmenü wählen
- Im Dailog Properties for JavaStandaloneClient wählen wir Java Build Path aus und aktivieren den Reiter Libraries
- Durch Klick auf Add external Jars... öffnet sich ein Dateimanager und wir wählen aus dem Verzeichnis C:\Programme\IBM\SDP\runtimes\base_v7\runtimes\base_v7\runtimes folgende JARs aus:
    - *com.ibm.ws.ejb.thinclient_7.0.0.jar*
    - *com.ibm.jaxws.thinclient_7.0.0.jar*

- *com.ibm.ws.admin.client_7.0.0.jar*

- Die Änderungen bestätigen wir mit *OK*

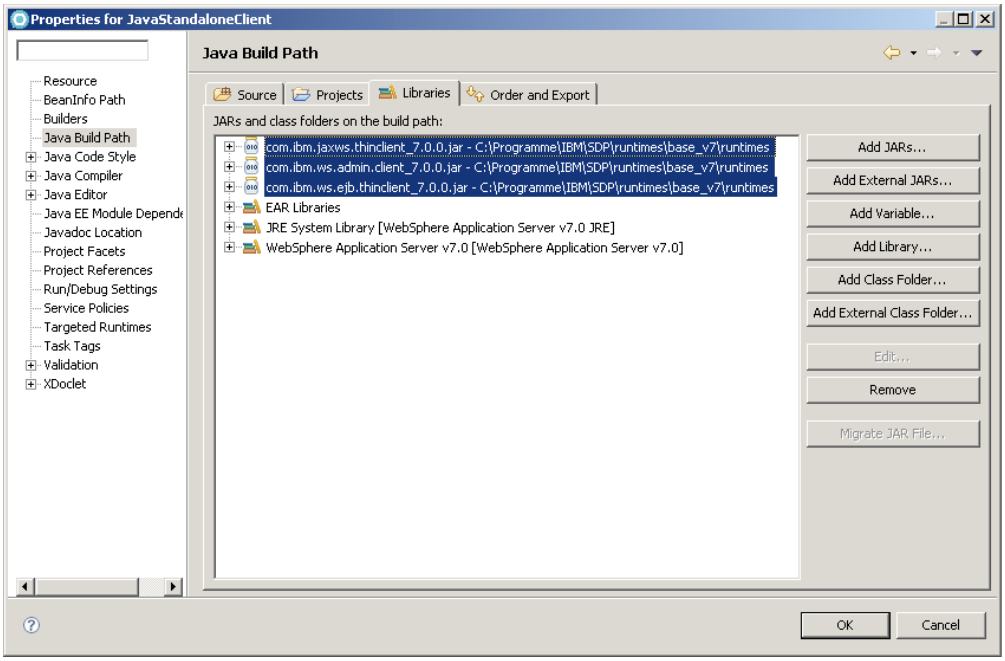

Abbildung 28: Build Path Einstellungen für JavaStandaloneClient

Als nächstes Erstellen wir eine neue Run Configuration für unseren Client. Für diese kopieren wir die Kommandozeilen, die auch für die Server Runtime verwendet werden.

- Rechtsklick im Project Explorer auf *JavaStandaloneProject*, *Run As → Run Configurations...* auswählen
- Doppelklick auf *WebSphere Application Server v7.0 Application Client*. Es wird nun eine neue Konfiguration mit dem Namen *New_configuration* angelegt.
- Im Reiter *Arguments* selektieren wir alle *VM arguments* und kopieren mit *CTRL+c* in die Zwischenablage (Abb. 29).

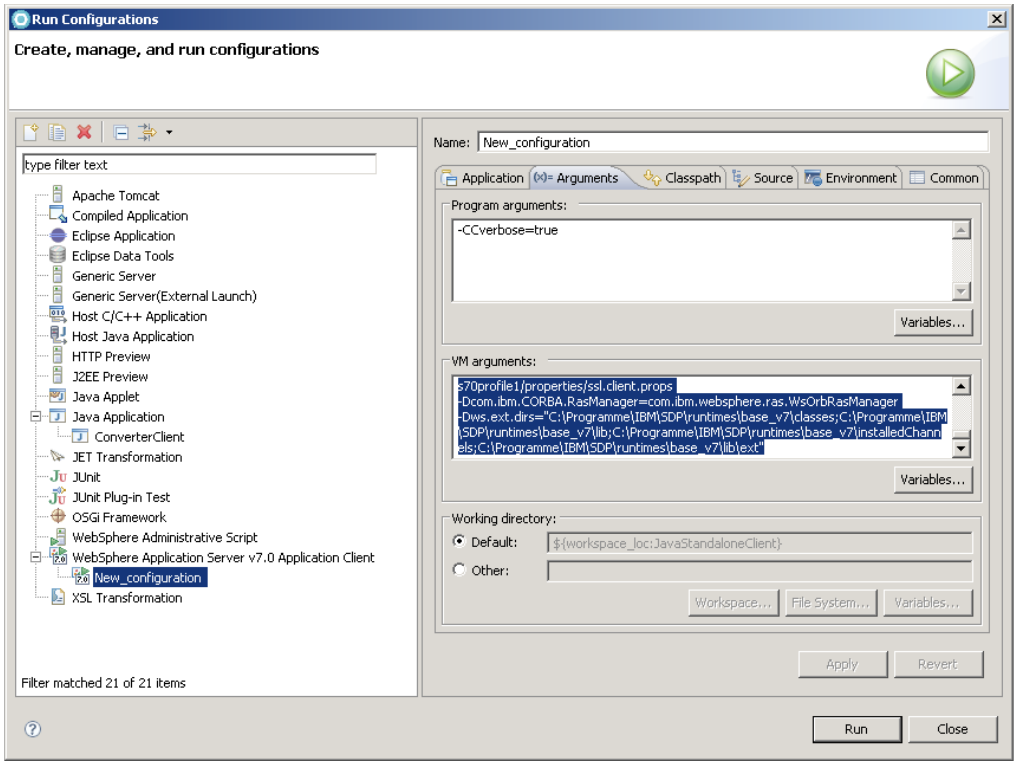

Abbildung 29: Run Configurations Dialog mit den selektierten VM Argumenten

- Wir wechseln zur Run Configuration von unserem Client, *Java Application → ConverterClient*
- Im Reiter *Arguments* fügen wir im Feld *VM arguments* mit CTRL+v die eben kopierten Argumente der Applikation Server VM ein.
- Im Reiter *JRE* wechseln wir die Projekt Runtime auf eine Standard Java Runtime, indem wir *Alternate JRE:* auf *jdk* setzen (Abb. 30).

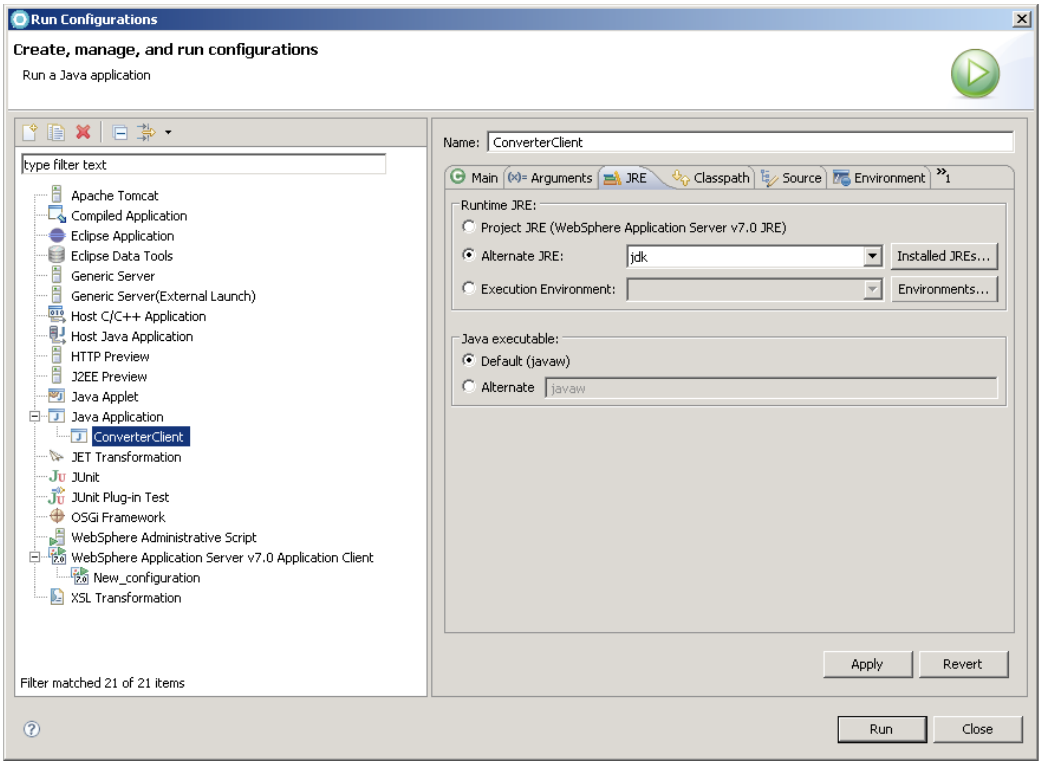

Abbildung 30: Wechsel des Java Runtime Environments (JRE)

- Mit Klick auf *Close* schliessen wir den Dialog und bestätigen die Nachfrage des Speicherns mit *Yes*
- JavaStandaloneClient kann nun gestartet Rechstklick und Auswahl von *Run As → Java Application* gestartet werden. Im Dialog *Select Java Application* muss nun noch *ConverterClient ausgewählt* werden.
- In der *Console View* sollte nun die Ausgabe des Clients wie in Abb. 31 zu sehen sein.

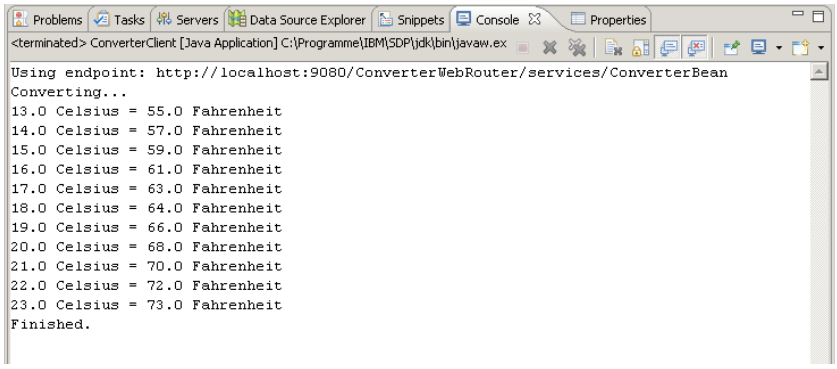

Abbildung 31: Ausgabe von ConverterClient

6 Zusammenfassung und Ausblick

In diesem Tutorial wurde eine eine Schritt-für-Schritt Anleitung von der Erstellung einer Enterprise Applikation basierend auf einer Enterprise Java Bean und eines Servlets, die Bereitstellung der EJB Funktionalität als Web Services über das Testen der Web Services bis hin zu der Entwicklung eines eigenständigen Java Client, der die Web Services konsumiert, vorgestellt.

Ein wesentliches Merkmal ist der Ende-zu-Ende Charakter des Tutorials, der den gesamten Entwicklungszyklus für die bottom-up-Entwicklung von Web Services demonstriert.

Zu den Aspekten die in dieser Arbeit nicht betrachtet worden sind gehören unter anderem die Anbindung von CICS Web Services und Web Services Erweiterungen wie Sicherheit (WS-Security) und Interoperabilität (WS-I). Dies könnten mögliche Themen für eine Erweiterung dieses Basisturorials sein.

Anhang A: Literaturverzeichnis

[RAD07] Ueli Wahli, Henry Cui, Craig Fleming, Maan Mehta, Marco Rohr, Pinar Ugurlu, Patrick Gan, Celso Gonzalez, Daniel M. Farrell, Andreas Heerdegen. *Rational Application Developer V7 Programming Guide*. IBM Redbook 2007. http://www.redbooks.ibm.com/redbooks/pdfs/sg247501.pdf (Abgerufen 20.4.2010)

[W3C04] Hugo Haas. *Web Services Glossary*. W3C Working Group Note 11 February 2004. http://www.w3.org/TR/ws-gloss/ (Abgerufen 20.4.2010)

[RFC3080] M. Rose. *The Blocks Extensible Exchange Protocol Core* (Request For Comments: 3080). IETF Network Working Group, March 2001. http://tools.ietf.org/html/rfc3080 (Abgerufen 20.4.2010)

[OASIS07] *Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1*. OASIS Standard, 14 June 2007. http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.1-spec-os-01.pdf (Abgerufen 20.4.2010)

[W3C07] Nilo Mitra, Yves Lafon. *SOAP Version 1.2 Part 0: Primer (Second Edition)*. W3C Recommendation 27 April 2007. http://www.w3.org/TR/2007/REC-soap12-part0-20070427/ (Abgerufen 20.4.2010)

[WSBI04] Geert Van de Putte, Joydeep Jana, Martin Keen, Sandhya Kondepudi, Roberto Mascarenhas, Satish Ogirala, Daniela Rudrof, Ken Sullivan, Peter Swithinbank. *Using WebServices for Business Integration*. IBM Redbook 2004. http://www.redbooks.ibm.com/redbooks/pdfs/sg246583.pdf (Abgerufen 20.4.2010)

[WHDev05] Ueli Wahli, Thomas Kjaer, Brett Robertson, Fumiko Satoh, Franz-Josef Schneider, Witold Szczeponik, Chris Whyley. *WebSphere Version 6 Web Services Handbook Development and Deployment*. IBM Redbook 2005. http://www.redbooks.ibm.com/redbooks/pdfs/sg246461.pdf

[CICSWS10] Chris Rayns, George Burgess, Paul Cooper, Tony Fitzgerald, Ankur Goyal, Peter Klein, Guo Qiang Li, SanYong Liu, Yan Sun. *Application Development for CICS Web Services*. IBM Redbook 2010. http://www.redbooks.ibm.com/redbooks/pdfs/sg247126.pdf (Abgerufen 20.4.2010)

[WJEE10] Wikipedia. Stichwort *"Java_Platform,_Enterprise_Edition"*. 18. April 2010. http://en.wikipedia.org/w/index.php?title=Java_Platform,_Enterprise_Edition&oldid=356762962 (Abgerufen 20.4.2010)

[JSR101] Roberto Chinnici. *Java API for XML based RPC 1.1*. JSR-101, Java Community Process (JCP), 2003. http://jcp.org/aboutJava/communityprocess/final/jsr101/index2.html (Abgerufen 20.4.2010)

[JSR109] Jim Knutson, Heather Kreger. *Implementing Enterprise Web Services*. JSR 109, Java Community Process (JCP) 2002. http://jcp.org/aboutJava/communityprocess/final/jsr109/index.html (Abgerufen 20.4.2010)

[Cui08] Henry Cui: Build Web services with transport-level security using Rational Application Developer V7, Part 3: Configure HTTPS. IBM developerWorks, 21 Feb 2008. http://www.ibm.com/developerworks/edu/ws-dw-ws-radsecurity3.html (Abgerufen 20.4.2010)

[RADOH] *Testing Java thin client against a Websphere Application Server*. IBM Rational Application Developer Version 7.5.5 Online Help. http://publib.boulder.ibm.com/infocenter/radhelp/v7r5/index.jsp?topic=/com.ibm.ws.ast.st.common.ui.doc/topics/tjavathin.html (Abgerufen 20.4.2010)

Anhang B: Abbildungsverzeichnis

Anhang C: Inhalt der CD

Auf der beiliegenden CD sind folgende Dateien und Verzeichnisse enthalten:

/archives/EJBConverterSampleEAR.ear	EJB und die Webapplikation
/archives/EJBConverterSampleEAR_final.ear	Enthält zusätzlich Router und Web Services
/archives/JavaStandaloneClient.ear	Der Java Thin Client
/src/	Alle Java und JSP Sourcen
/screenshots/	Alle verwendeten Abbildungen
/references/	IBM Redbooks und JSRs
/websphere_web_services_tutorial.pdf	Diese Arbeit im PDF-Format
/websphere_web_services_tutorial.odt	Diese Arbeit im Open Document Format