Universität Leipzig

Fakultät für Mathematik und Informatik

Institut für Informatik

Web Service-Anwendungen unter RDz (WDz)

Bachelorarbeit

Leipzig, 02, 2010

vorgelegt von

Hoff, Oliver
Studiengang Informatik

Betreuender Hochschullehrer: Prof. Dr. W. Spruth

Executive Summary

Heterogene Serverlandschaften und die Ausführung heutiger Geschäftsprozesse erfordern große Mengen an intermaschineller Kommunikation. Um Sicherheit, Zuverlässigkeit und Performance zu gewährleisten, sind leistungsfähige Verfahren für die Systemintegration erforderlich. Web Services sind ein solches Verfahren – mit ihrer Hilfe lassen sich komplexe verteilte Softwaresysteme miteinander verbinden und integrieren.

Im ersten Teil dieser Arbeit werden die Konzepte der Web Services anhand der aktuellen *W3C Recommendation* von WSDL 2.0 vorgestellt und Verweise auf weiterführende Literatur gegeben. Es wird außerdem ein beispielhafter Überblick über die syntaktischen Eigenschaften der einzelnen Sprachbestandteile gegeben und auf die Unterschiede zwischen WSDL 1.1 und Version 2.0 eingegangen.

Im zweiten Teil wird die Systemintegration von verschiedener Software mit Hilfe der Entwicklungsumgebung *Rational Developer for System z* (RDz) vorgestellt. RDz ist speziell für die Erstellung von Großrechnersoftware ausgelegt. Dabei sind drei Tutorials entstanden, welche die *JavaEE* Anwendung von Web Services unter Verwendung des *Axis* Frameworks erläutern.

Die ersten Zwei (Kapitel *3.4. Web Services in Anwendungsservern mit Apache Axis*) beschäftigen sich mit dem Umgang der Code-Generation-Werkzeuge für die Bottom-Up und Top-Down Entwicklung und dem Publizieren von Web Services im *Web Sphere Application Server Community Edition* (WASCE).

Im Dritten (Kapitel *3.5. Skat-Ergebnistabelle als Web Service*) wird eine z/OS DB2 Installation mit einem *JavaEE* Middle-Tier auf einem WASCE integriert, dessen Geschäftsfunktionen über einen Web Service angeboten werden und damit von verschiedenartigen Frontends genutzt werden können.

Danksagung

Ich möchte mich bei Herrn Prof. Dr. Wilhelm G. Spruth und Herrn Nils Michaelsen für die Unterstützung während der Erstellung dieser Arbeit bedanken, sowie bei Frau Ines Dietrich für das Korrekturlesen.

Inhalt

1. Einleitung

Diese Arbeit soll einen Einstieg in das Thema *Web Services* geben und deren Anwendung mit Hilfe von Entwicklungswerkzeugen für Großrechner (Mainframes) mit Schwerpunkt auf der Verwendung in Enterprise Application Server Software. Dazu wird in Kapitel 2 eine Einführung in die Konzepte von Web Services anhand der *Web Services Description Language* (WSDL) Version 2.0 gegeben. In Kapitel 3 werden dann grundlegende Anwendungsbeispiele auf Basis der Entwicklungsumgebung *IBM Rational Developer for System z*[1] (RDz) gezeigt.

1.1. Was ist ein Web Service

Ein Web Service ist eine Anwendung, deren Funktionalitäten im Internet angeboten werden um von anderen Applikationen benutzt zu werden. Ein Problem stellt dabei die Heterogenität der im Internet zusammenarbeitenden Computersysteme dar, was zu Inkompatibilitäten im nativen Nachrichtenaustausch führt. Weiterhin muss beiden Kommunikationspartnern klar gemacht werden, welche Funktionen genutzt werden sollen bzw. können. Mit Hilfe einer generischen Beschreibungssprache – wie WSDL – sollen deshalb die syntaktischen Anforderungen an die Nachrichten, an die technischen Aspekte der Übertragung und ausführbare Operationen spezifiziert werden. Eine Laufzeitumgebung die diese Sprache versteht, kann dann anhand der festgelegten Beschreibungen Funktionen ihrer laufenden Applikationen anbieten und eine benutzende Anwendung nutzt dieselbe Beschreibung um auf diese zuzugreifen.

1.2. Voraussetzungen

Für die Einführung in die Web Services und WSDL sind fundierte Kenntnisse von XML und dessen Konzept der Namensräume erforderlich. Weiterhin sollten grundlegende Merkmale der Schemasprache XML-Schema und der Prinzipien von SOAP[2] bekannt sein. Die Grundlagen des Anwendungsprotokoll HTTP sollten verstanden worden sein.

Eine Einführung in XML-bezogene Themen befindet sich unter folgender Adresse:

Vorlesung: Extensible Markup Language (XML) von Mario Jeckle
http://www.jeckle.de/vorlesung/xml/index.html

Um die Durchführung der Beispiele nachzuvollziehen sind grundlegenden Kenntnissen der *Eclipse* IDE[3] vonnöten.

1.3. Software und Mainframe-Zugriff

Es wird in dieser Arbeit von der Entwicklungsumgebung *IBM Rational Developer for System z* ausgegangen. Dies ist ein auf der *Eclipse* IDE basierendes Werkzeug um Anwendungen für die IBM Mainframes zu entwickeln. Es kann sowohl für die Entwicklung der Mainframe-typischen Anwendungen wie COBOL und CICS verwendet werden, als auch für die Integration mit Java-Technologie oder eben Web Services. RDz stellt dafür eine Reihe von Assistenten bereit, um zum Beispiel WSDL-Dokumente und dazugehörige

1 Früher: *IBM WebSphere Developer for System z* (WDz)
2 SOAP; ehemals Akronym für Simple Object Access Protocol, heute Eigenname, bezeichnend für Prinzipien zur Erstellung von verteilten Anwendungen
3 Integrated Development Environment, Abk. IDE; dt. Integrierte Entwicklungsumgebung

Schemadefinitionen aus vorhandener Mainframe-Software zu generieren. Es wird gezeigt wie grundlegende Funktionen der IDE benutzt werden, um damit Web Services zu erstellen.

Eine 60-Tage Testversion kann von der IBM Website bezogen werden. Näheres dazu findet man in Kapitel 3.

Um alle erstellten Web Services in vollem Umfang zu testen ist ein Zugang auf einen IBM Mainframe nötig. Dieser kann bei Herrn Prof. Dr. W. Spruth für die Lehrmaschinen der Universität Leipzig bzw. Universität Tübingen bezogen werden. Weitere Informationen dazu findet man auf:

Der z900 Mainframe der Universität Leipzig
http://jedi.informatik.uni-leipzig.de/

Zu beachten ist, dass zum Zeitpunkt der Erstellung dieser Arbeit die DB2 Datenbank auf der Leipziger Maschine noch nicht ausreichend konfiguriert ist, um entfernte Zugriffe zuzulassen. Es wird an diesem Problem gearbeitet und Sie erhalten aktuelle Informationen darüber bei Herrn Prof. Spruth.

2. WSDL

Die *Web Services Description Language* (WSDL) ist eine XML-Sprache die meist in Verbindung mit SOAP und XML-Schema den syntaktischen und technischen Rahmen eines Web Service beschreibt. Aktuell ist die Version 2.0 welche durch folgende *W3C Recommendations* definiert wird:

Web Services Description Language (WSDL) Version 2.0 Part 0: Primer
http://www.w3.org/TR/wsdl20-primer

Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language
http://www.w3.org/TR/wsdl20

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
http://www.w3.org/TR/wsdl20-adjuncts

und weiterhin durch die W3C Working Group Note:

Web Services Description Language (WSDL) Version 2.0: Additional MEPs
http://www.w3.org/TR/wsdl20-additional-meps

Part 0 enthält eine Einführung der wichtigsten Sprachbestandteile und Anwendungsmöglichkeiten anhand von Beispielen, Part 1 die normative Beschreibung aller Sprachelemente, Part 2 bietet vordefinierte Spracherweiterungen für WSDL und die Note *Additional MEPs* erweitert die Menge der Nachrichten-Austauschmuster[4].

WSDL 2.0 unterteilt die Spezifikation eines Web Service in vier Teile. Die Nachrichtentypen (`types`-Element) und Schnittstellen (`interface`-Element) beschreiben was für Nachrichten ausgetauscht werden und welche Operationen bzw. Interaktionen mit einem Web Service möglich sind. Im `binding`-Element werden technische Aspekte des Datenaustauschs festgelegt und im `service`-Element wo diese konkreten Anwendungen erreichbar sein sollen. Alle Komponenten können mit Hilfe der Import- und Inklusionsmechanismen wiederverwendet werden, was vor allem für Typen, Schnittstellen und generische Bindungsdefinitionen sinnvoll ist.

4 Message Exchange Pattern(s), Abk. MEP(s); dt. Nachrichten-Austauschmuster

2.1. Allgemein

Das Wurzelelement eines WSDL-Dokument ist das `description`-Element. Es liegt wie alle WSDL-Kernkomponenten im Namensraum `http://www.w3.org/ns/wsdl`. Weiterhin muss ein Ziel-Namensraum für den definierten Web Service angegeben werden.

Wird XML-Schema zur Spezifikation von Nachrichtentypen genutzt, so muss zusätzlich der Namensraum in dem die Elemente definiert sind angegeben werden.

Folgende weitere Namensräume werden häufig gebraucht:

WSDL-Standarderweiterungen
`http://www.w3.org/ns/wsdl-extensions`

WSDL-SOAP-Erweiterungen
`http://www.w3.org/ns/wsdl/soap`

WSDL-HTTP-Erweiterungen
`http://www.w3.org/ns/wsdl/http`

WSDL-RPC-Style-Erweiterungen
`http://www.w3.org/ns/wsdl/rpc`

SOAP Namensraum
`http://www.w3.org/2003/05/soap-envelope`

Es existiert ein `documentation`-Element, das innerhalb aller anderen Elemente als erstes Kindelement vorkommen darf und grundlegende Dokumentationen, die im direkten Bezug zum jeweiligen WSDL-Dokument stehen, enthalten sollte. Es ist außerdem sinnvoll, Referenzen auf ergänzende externe Dokumentationen hinzuzufügen, welche den Einsatz des Web Service auf Anwendungsebene erläutern.

Ein Beispiel für ein `description`-Element mit Dokumentation:

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace="http://company.example.com/webservices/myService"
    xmlns:tns="http://company.example.com/webservices/myService"
    xmlns:sns="http://company.example.com/schemas/myService"

    xmlns:wsdlx="http://www.w3.org/ns/wsdl-extensions"
    xmlns:wrpc="http://www.w3.org/ns/wsdl/rpc"

    xmlns:wsoap="http://www.w3.org/ns/wsdl/soap"
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:whttp="http://www.w3.org/ns/wsdl/http">

  <documentation>
    Hier steht eine Dokumentation mit einer externen Referenz zu
    weiterführenden Dokumenten:
    http://company.example.com/myService-documentation.html
  </documentation>

  ...

</description>
```

2.2. Das `types`-Element – Inhalt von Nachrichten typisieren

Die Grundlage für den Datenaustausch bilden Nachrichten. Im Falle von WSDL bestehen diese aus XML-Dokumenten, deren syntaktischer Aufbau mit Hilfe von XML-Schema definiert werden. Es sind auch andere Schemasprachen möglich wie DTD oder RELAX NG, deren Verwendung unter folgender URL diskutiert wird:

Discussion of Alternative Schema Languages and Type System Support in WSDL
http://dev.w3.org/cvsweb/~checkout~/2002/ws/desc/wsdl20/altschemalangs.html?content-type=text/html;%20charset=utf-8&rev=1.3

Die einzelnen Nachrichtenformate werden dabei im `types`-Element festgelegt, welches direkt als erstes nachfolgendes Element eines etwaigen `documentation`-Element vorkommen muss. Wenn XML-Schema als Schemasprache genutzt wird, baut sich innerhalb des `types`-Element ein normaler XSD[5]-Dokumentbaum auf. Da alle Elemente, welche direkt als Kindelemente des `xs:schema`-Elements definiert werden, Wurzelelemente eines Dokuments sein dürfen, können alle Nachrichtenformate mit einem einzigen XSD-Dokument beschrieben werden. Es ist jedoch auch möglich, mehrere XSD-Dokumente innerhalb des `types`-Elements zu definieren oder externe Schemadefinitionen einzubinden. Entweder über die Import-/Inklusionsmechanismen von XSD, über die entsprechenden Methoden von WSDL oder den Importmechanismus vom `types`-Element. Die Wiederverwendung mit Hilfe dieser Funktionalitäten wird in Kapitel *2.6. Import- und Inklusionsmechanismen* erläutert. Eine Übersicht über die verschiedenen Methoden findet sich unter:

Web Services Description Language (WSDL) Version 2.0 Part 0: Primer
2.3.3 Summary of Import and Include Mechanisms
http://www.w3.org/TR/wsdl20-primer/#more-types-import-include-summary

Da die Typen-Definitionen aus anderen WSDL-Dokumenten importiert oder inkludiert werden können ist das `types`-Element optional. Es darf jedoch höchstens einmal vorkommen.

Ein Beispiel für ein `types`-Element:

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace="http://company.example.com/webservices/myService"
    xmlns:tns="http://company.example.com/webservices/myService"
    xmlns:sns="http://company.example.com/schemas/myService"
    ... >

  ...

  <types>
    <xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://company.example.com/schemas/myService"
        xmlns="http://company.example.com/schemas/myService">

      <xs:element name="sampleRequest" type="sampleRequestType" />
      <xs:complexType name="sampleRequestType">
        <xs:sequence>
          <xs:element name="requestDate" type="xs:date" />
          <xs:element name="requestContent" type="xs:string" />
```

5 XML-Schema-Definition(s), Abk. XSD(s); Definitionsdokumente von XML-Schema

```
        </xs:sequence>
      </xs:complexType>

      <xs:element name="sampleResponse" type="xs:int" />

      <xs:element name="sampleError" type="xs:string" />

    </xs:schema>
  </types>

  ...

</description>
```

2.3. Das `interface`-Element – Schnittstellen definieren

Ein Interface definiert Operationen, welche ein Web Service bereitstellt, sowie mögliche Fehlernachrichten – sogenannte Faults – die verwendet werden können. Dabei ist es möglich, innerhalb des Ziel-Namensraumes des Web Service mehrere Schnittstellen zu definieren. Unterschieden werden diese Schnittstellen über einen im `name`-Attribut des `interface`-Elements zugewiesen Namen. Der Name eines Interface muss in der Menge aller Interface-Namen eindeutig sein und besteht aus dem `xs:NCName`[6]-Attribut `name` und des Ziel-Namensraumes des Web Service.

Die `fault`-Elemente legen Fehlernachrichten fest. Sie müssen im Geltungsbereich des jeweiligen `interface`-Elements eindeutig über ihr `name`-Attribut benannt werden. Außerdem wird im `element`-Attribut der Nachrichtentyp spezifiziert. Es lässt die Werte `#none` für einen leeren Nachrichtentyp, `#any` für ein beliebiges Wurzelelement, `#other` für einen nicht mittels XSD definierten Typ oder ein `xs:QName`[7], der ein in XSD definiertes Wurzelelement referenziert, zu.

Die Operationen, welche ebenfalls eindeutig innerhalb eines Interface benannt werden müssen, beschreiben die genaue Abfolge von Nachrichten und an welchen Stellen des Nachrichtenaustauschs Fehlermeldungen auftreten dürfen. Diese Abfolge wird durch vordefinierte Message Exchange Patterns (MEPs) festgelegt und kann mittels des `pattern`-Attributs referenziert werden. Weiterhin müssen ein oder mehrere mögliche Operationsstile über das `style`-Attribut festgelegt werden und das in der WSDL-Erweiterung beschriebene `safe`-Attribut belegt werden.

Innerhalb des `operation`-Elements werden die einzelnen Nachrichten des MEP, welche eine Richtung und einen Bezeichner besitzen, mit Hilfe der `input`-, `output`-, `infault`- und `outfault`-Elemente spezifiziert. Dabei haben `input` und `infault` per Definition die Richtung *in* und `output` und `outfault` die Richtung *out*. Der Bezeichner wird über das `messageLabel`-Attribut gesetzt und der Nachrichtentyp von `input`- und `output`-Elementen über das `element`-Attribut, das die gleichen Werte wie das gleichnamige Attribut des `fault`-Elements akzeptiert. Bei den Fehlernachrichten wird kein `element`-Attribut angegeben sondern der entsprechende Fault über das `ref`-Attribut referenziert. Es sind mehrere Definitionen für Fehlernachrichten möglich um unterschiedliche Fehlertypen zu definieren.

Beispiel:

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace="http://company.example.com/webservices/myService"
    xmlns:tns="http://company.example.com/webservices/myService"
    xmlns:sns="http://company.example.com/schemas/myService"
    ... >
  ...

  <interface name="sampleInterface" >

    <fault name="sampleFault"
        element="sns:sampleError"/>
```

6 `xs:NCName`, XSD-Datentyp, non-colonized name; dt. Name ohne Doppelpunkt

7 `xs:QName`, XSD-Datentyp, qualified name; dt. Qualifizierter Name, das heißt ein Name mit Namensraumkürzel und lokalem Namensteil

```
    <operation name="sampleOperation"
        pattern="http://www.w3.org/ns/wsdl/in-out"
        style="http://www.w3.org/ns/wsdl/style/iri"
        wsdlx:safe="true">

      <input messageLabel="In"
          element="sns:sampleRequest"/>
      <output messageLabel="Out"
          element="sns:sampleResponse"/>
      <outfault ref="tns:sampleFault" messageLabel="Out"/>

    </operation>

  </interface>

  ...

</description>
```

2.3.1. Message Exchange Patterns – Nachrichten-Austauschmuster

Um den Nachrichtenfluss innerhalb einer Operation zu charakterisieren werden sogenannte Message Exchange Patterns verwendet. Es gibt in den Standarderweiterungen acht verschiedene Muster, welche die häufigsten Anwendungsfälle abdecken. Es ist jedoch auch möglich eigene Muster zu definieren.

2.3.1.1. in-only MEP

Beim in-only MEP wird nur eine Nachricht vom Client zum Service gesendet und es entsteht kein weiterer Nachrichtenverkehr.

```
... pattern="http://www.w3.org/ns/wsdl/in-only" ...
<input messageLabel="In" ... />
...
```

2.3.1.2. robust-in-only MEP

Ähnlich wie das in-only MEP, aber es kann im Falle eines Fehlers eine Fault-Nachricht an den Sender zurückgeschickt werden, die eine Beschreibung des aufgetretenen Fehlers enthält.

```
... pattern="http://www.w3.org/ns/wsdl/robust-in-only" ...
<input messageLabel="In" ... />
<outfault messageLabel="In" ... />*
...
```

2.3.1.3. in-out MEP

Das in-out MEP entspricht dem wohl bekannten Request-Response-Muster, wobei Fehler im einem speziellen Response-Typ mit Fehlercode zurückgesendet werden. Im Fehlerfall ersetzt die Fehlernachricht den Response.

```
... pattern="http://www.w3.org/ns/wsdl/in-out" ...
<input messageLabel="In" ... />
<output messageLabel="Out" ... />
<outfault messageLabel="Out" ... />*
...
```

2.3.1.4. Weitere MEPs

Die hier vorgestellten MEPs werden in

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
2.3 Message Exchange Patterns
http://www.w3.org/TR/wsdl20-adjuncts/#patterns

definiert. Folgende weitere MEPs werden in

Web Services Description Language (WSDL) Version 2.0: Additional MEPs
http://www.w3.org/TR/wsdl20-additional-meps

eingeführt:

in-optional-out

```
... pattern="http://www.w3.org/ns/wsdl/in-opt-out" ...
<input messageLabel="In" ... />
<output messageLabel="Out" ... />
<outfault messageLabel="In" ... />*
...
```

out-only

```
... pattern="http://www.w3.org/ns/wsdl/out-only" ...
<output messageLabel="Out" ... />
...
```

robust-out-only

```
... pattern="http://www.w3.org/ns/wsdl/robust-out-only" ...
<output messageLabel="Out" ... />
<infault messageLabel="Out" ... />*
...
```

out-in

```
... pattern="http://www.w3.org/ns/wsdl/out-in" ...
<output messageLabel="Out" ... />
<input messageLabel="In" ... />
<infault messageLabel="In" ... />*
...
```

out-optional-in

```
... pattern="http://www.w3.org/ns/wsdl/out-opt-in" ...
<output messageLabel="Out" ... />
<input messageLabel="In" ... />
<infault messageLabel="Out" ... />*
...
```

2.3.2. Operation Styles – Operationsstile

Der Operationsstil legt fest, in welcher Form ein Nachrichtenaustausch stattfindet, das heißt ob zum Beispiel eine ausführbare Funktion am Endpunkt angesprochen wird oder einfache Textnachrichten zwischen den Kommunikationspartnern ausgetauscht werden. Dabei werden abhängig vom gewählten Stil zusätzliche Einschränkungen an mögliche MEPs und die Nachrichtentypen geknüpft. Im Folgenden werden die drei vordefinierten Stile vorgestellt.

2.3.2.1. RPC Style

Der vordefinierte RPC Style beschreibt die Operation als einen Funktionsaufruf. Dabei muss die Signatur der Funktion separat im `wrpc:signature`-Attribut des `operation`-Elements angegeben werden. Außerdem sind lediglich das in-only und in-out MEP möglich. Alle Nachrichtentypen müssen mittels XSD definiert worden sein und zusätzlich dürfen die referenzierten Wurzelelemente lediglich vom Typ `xs:complexType`[8] sein, welcher eine `xs:sequence`[9] enthält in der nur `xs:simpleType`[10]-Elemente erlaubt sind.

```
... style="http://www.w3.org/ns/wsdl/style/rpc"
    wrpc:signature="param1 #in param2 #out param3 #inout result #return" ...
```

Weitere Bedingungen können unter folgender Adresse nachgelesen werden:

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
4.1 RPC Style
http://www.w3.org/TR/wsdl20-adjuncts/#RPCStyle

2.3.2.2. IRI Style

Beim IRI Style werden zusätzliche Anforderungen an den Typ der ersten Nachricht des verwendeten MEP gestellt, um zu gewährleisten das diese als IRI[11] serialisiert werden kann. Dies ist zum Beispiel notwendig damit die Operation an eine HTTP-GET Anfrage gebunden werden kann. Ähnlich wie beim RPC Style muss der Nachrichtentyp aus einem Wurzelelement, das mit Hilfe von XML Schema definiert wurde, bestehen dessen Inhaltsmodell ein `xs:complexType` mit einem `xs:sequence`-Element ist. Die Kindelemente dürfen nur vom Typ `xs:simpleType` (außer `xs:QName`, `xs:NOTATION`, `xs:hexBinary` oder `xs:base64Binary`) sein.

```
... style="http://www.w3.org/ns/wsdl/style/iri" ...
```

Weitere Bedingungen können unter folgender Adresse nachgelesen werden:

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
4.2 IRI Style
http://www.w3.org/TR/wsdl20-adjuncts/#_operation_iri_style

2.3.2.3. Multipart Style

Multipart Style stellt ähnlich Anforderungen wie IRI Style an die erste Nachricht des verwendeten MEP. Dabei wird diese nicht in eine IRI überführt, sondern in eine Anfrage mit dem Content-Type `Multipart/form-data`. Dieser Stil wird von der HTTP Erweiterung von WSDL aus Kompatibilitätsgründen zu XForms-fähigen Servern unterstützt.

```
... style="http://www.w3.org/ns/wsdl/style/multipart" ...
```

Weitere Bedingungen können unter folgender Adresse nachgelesen werden:

---

8 `xs:complexType`, XSD-Datentyp; alle Datentypen die ein XSD-Inhaltsmodell besitzen

9 `xs:sequence`, XSD-Inhaltsmodell; eine Sequenz von Elementen

10 `xs:simpleType`, XSD-Datentyp; alle vordefinierten einfachen Datentypen und deren Ableitungen

11 Internationalized Resource Identifier, Abk. IRI; Erweiterung der Uniform Resource Identifier um eine internationale Zeichenpalette

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
4.3 Multipart style
http://www.w3.org/TR/wsdl20-adjuncts/#_operation_multipart_style

2.3.3. Operation safety – Operationssicherheit

Die Operationssicherheit beschreibt, ob die Verwendung der jeweiligen Operation des Interface irgendwelche Konsequenzen, wie beispielsweise Statusänderungen, zur Folge hat. So ist der Abruf eines Aktienkurses (sofern er nicht für statistische Zwecke geloggt wird) als sicher einzuschätzen, während zum Beispiel eine SQL `INSERT`-Query Änderungen in der angesprochenen Datenbank zur Folge haben könnte und demzufolge unsicher wäre.

Weiterführende Informationen zur Sicherheit von Interaktionen im Web sind unter folgender Adresse zu finden:

Architecture of the World Wide Web, Volume One
3.4. Safe Interactions
http://www.w3.org/TR/webarch/#safe-interaction

2.4. Das `binding`-Element – Binden von Schnittstellen

Ein Binding legt fest, wie Nachrichten übertragen werden, das heißt welches Protokoll zum Einsatz kommt, in welchem Format die Nachrichten übertragen werden und mit welcher Kodierung. Es gibt zwei vordefinierte Erweiterungen um Operationen via SOAP oder HTTP zu binden.

Alle `binding`-Elemente müssen innerhalb des Ziel-Namensraum eindeutig über das `name`-Attribut benannt werden und residieren im Wurzelelement des WSDL-Dokuments. Im `type`-Attribut wird die Erweiterung spezifiziert die für die Bindung zum Einsatz kommen soll. Die Erweiterungen bringen dabei eigene Sprachbestandteile zu deren Konfiguration mit sich.

Bindungen können – müssen jedoch nicht – eine Schnittstelle innerhalb des `interface`-Attributs referenzieren für das die Nachrichtenübertragung konkretisiert wird. Falls kein Interface angegeben wird spricht man von einer wiederverwendbaren bzw. generischen Bindung. Dabei ist zu beachten, dass die für die Bindung genutzte Erweiterung, entsprechende Standardkonfigurationen bietet bzw. alle nötigen Konfigurationsinformationen im benutzenden Service-Endpoint bereitgestellt werden.

Wenn ein Interface referenziert wird ist es – abhängig von der genutzten Bindungserweiterung – nicht zwingend erforderlich alle einzelnen Operationen eines Interface explizit zu binden, sondern es können Standardwerte in den Erweiterungsattributen des `binding`-Elements angegeben werden oder die voreingestellten Wertebelegungen der jeweiligen Erweiterung werden benutzt.

Ein Beispiel für den Rahmen einer bereits an ein Interface gebundene Bindung:

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace="http://company.example.com/webservices/myService"
    xmlns:tns="http://company.example.com/webservices/myService"
    ... >

  ...

  <interface name="sampleInterface"> ... </interface>

  <binding name="sampleBinding"
      interface="tns:sampleInterface"
      type="..."
      ... >

    <operation ref="tns:sampleOperation" ... />
    <fault ref="tns:sampleFault" ... />

  </binding>

  ...

</description>
```

2.4.1. HTTP-Binding

Ein HTTP-Binding bindet die Operationen einer Schnittstelle an das HTTP-Protokoll. Die vordefinierte HTTP-Erweiterung von WSDL unterstützt HTTP 1.1 und HTTP 1.0. Die Verwendung der Erweiterung wird mit dem Wert `http://www.w3.org/ns/wsdl/http` im `type`-Attribut des `binding`-Elements angezeigt. Am bedeutendsten für die Bindung einer Operation ist die HTTP-Methode die genutzt wird, um den Nachrichtenaustausch zu vollziehen und das Verfahren mit dem die Nachricht in ein HTTP-Request/Response serialisiert wird.

2.4.1.1. HTTP-Method – Anfragemethode

Die Methode kann für alle Operationen des gebundenen Interface im `whttp:methodDefault`-Attribut des `binding`-Elements gesetzt und/oder separat im `whttp:method`-Attribut des `operation`-Kindelement gegebenenfalls überschrieben werden. Wird keine der beiden Möglichkeiten genutzt so wird anhand des `wsdlx:safe`-Attribut des zu bindenden Interface entschieden, welche Methode benutzt werden soll. Dabei ist es egal, ob die Schnittstelle bereits in der Bindung referenziert wird oder erst im Service-Endpoint zugewiesen wird. Ist die Operation sicher so wird `GET` benutzt, ansonsten `POST`.

2.4.1.2. Content-Type – Serialisierung

Das Serialisierungsverfahren einer Operation wird standardmäßig anhand der verwendeten HTTP-Methode der jeweiligen Operation festgemacht. Für `GET` wird die Eingabeserialisierung `application/x-www-form-urlencoded` und die Ausgabeserialisierung `application/xml` verwendet. Das jeweilige Serialisierungsformat wird im Header-Feld `Content-Type` der HTTP-Nachricht eingesetzt. Weitere Standardzuordnungen sind unter folgender Adresse zu finden:

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
6.4.4 Default input and output serialization format
http://www.w3.org/TR/wsdl20-adjuncts/#_http_binding_default_rule_dsf

Das Serialisierungsverfahren kann jedoch auch separat für jede Operation des gebundenen Interface festgelegt werden und zwar mittels des `inputSerialization`-, `outputSerialization`- und `faultSerialization`-Attributs im jeweiligen `operation`-Kindelement.

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    xmlns:whttp="http://www.w3.org/ns/wsdl/http"
    ... >
  ...
  <binding ... >
    <operation
        whttp:inputSerialization="..."
        whttp:outputSerialization="..."
        whttp:faultSerialization="..."
        ... >
      ...
    </operation>
  </binding>
  ...
</description>
```

Wenn die Serialisierung `application/x-www-form-urlencoded` verwendet wird, muss mittels des `whttp:queryParameterSeparatorDefault`-Attribut im `binding`-Element ein Separator für den Query-Teil der IRI angegeben werden, der genutzt wird um die einzelnen Query-Parameter voneinander zu trennen. Der Separator kann individuell für jede Operation über das `whttp:queryParameterSeparator`-Attribut des jeweiligen `operation`-Element gesetzt werden.

Wie genau einzelnen Serialisierungsmethoden umgesetzt werden kann man unter folgenden Adressen nachlesen:

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
6.8.2 Serialization as application/x-www-form-urlencoded
http://www.w3.org/TR/wsdl20-adjuncts/#_http_x-www-form-urlencoded

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
6.8.4 Serialization as multipart/form-data
http://www.w3.org/TR/wsdl20-adjuncts/#_http_operation_multipart_encoding

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
6.8.3 Serialization as application/xml
http://www.w3.org/TR/wsdl20-adjuncts/#_http_operation_xml_encoding

2.4.1.3. Content-Encoding – Kodierung

Die Kodierung der HTTP-Nachricht kann über das `whttp:contentEncodingDefault`- bzw. `whttp:contentEncoding`-Attribut gesetzt werden. Ähnlich wie die HTTP-Method kann die Kodierung in verschiedenen Geltungsbereichen definiert werden. Wird das Attribut gesetzt, so wird der Wert im Header-Feld `Content-Encoding` der HTTP-Nachricht eingefügt, sonst wird das Header-Feld nicht gesetzt. Empfohlen für das Web wird UTF-8 als Standardkodierung.

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    xmlns:whttp="http://www.w3.org/ns/wsdl/http"
    ... >

  ...

  <binding whttp:contentEncodingDefault="..." ... >

    <fault whttp:contentEncoding="..." ... />

    <operation whttp:contentEncodingDefault="..." ... >
      <input whttp:contentEncoding="..." ... />
      <output whttp:contentEncoding="..." ... />
    </operation>

  </binding>

  ...

</description>
```

2.4.1.4. Location

Die Location beschreibt den Pfad zu einer bestimmten Operation einer HTTP-Bindung. Sie wird relativ im `whttp:location`-Attribut des jeweiligen `operation`-Elements angegeben. Da aber für ein HTTP-Request eine absolute IRI notwendig ist wird die Location gegen die im Service-Endpoint festgelegte IRI-Adresse aufgelöst. Mit Hilfe der Location können Probleme mit nicht eindeutigen Operationsaufrufen aufgelöst werden. Zum Beispiel wenn es innerhalb eines Interface zwei Operationen mit dem in-out MEP gibt die den gleichen Eingabe-Nachrichtentyp verwenden. Sobald eine Anfrage unter der im Endpunkt festgelegten Adresse eintrifft, ist es für die dazugehörige Bindung nicht zweifelsfrei entscheidbar, welche Operation der gebundenen Schnittstelle aufgerufen werden soll. Ein weiterer Vorteil des `whttp:location`-Attributs liegt darin, dass in Abhängigkeit von der genutzten Eingabeserialisierung, der Inhalt von einem Präprozessor vorverarbeitet wird und sich dadurch Teile der Nachricht bereits in einem selbst festgelegten Muster in der Anfrage-IRI serialisieren lassen. Somit kann man ähnliche Effekte wie mit dem Apache Modul mod-rewrite erreichen.

Das folgende Beispiel zeigt wie das Datumsfeld der Anfrage als Teil des IRI-Pfad festgelegt wird:

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    xmlns:whttp="http://www.w3.org/ns/wsdl/http"
    ... >

  ...

  <types>
    <xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://company.example.com/schemas/myService"
        xmlns="http://company.example.com/schemas/myService">

      <xs:element name="sampleRequest" type="sampleRequestType"/>
      <xs:complexType name="sampleRequestType">
        <xs:sequence>
          <xs:element name="requestDate" type="xs:date"/>
          <xs:element name="requestContent" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>

    </xs:schema>
  </types>

  ...

  <binding ... >
    <operation whttp:location="{requestDate}" ... >
      ...
    </operation>
  </binding>

  ...

</description>
```

Zu beachten ist, dass der Typ `xs:NCName` benutzt wird um Elemente des Nachrichtentyps zu referenzieren. Der Namensraum wird über das gebundene Interface ermittelt.

Weiter Informationen und genaue Konstruktionsregeln für die Werte des `whttp:location`-Attributs befinden sich unter folgender Adresse:

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
6.8.1 Serialization of the instance data in parts of the HTTP request IRI
http://www.w3.org/TR/wsdl20-adjuncts/#_http_location_template

2.4.1.5. HTTP-Error-Code – Fehlercodes

Im `whttp:code`-Attribut der `fault`-Elemente, welche je einen Fault des gebundenen Interface referenzieren, kann diesem Fault ein HTTP-Error-Code zugewiesen werden. Zusätzlich ist auch `#any` als Wert möglich der zugleich Standardwert ist, falls das Attribut nicht gesetzt wird.

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    xmlns:whttp="http://www.w3.org/ns/wsdl/http"
    ... >

  ...

  <binding ... >
    <fault whttp:code="..." ... />
    ...
  </binding>

  ...

</description>
```

2.4.1.6. Weitere Elemente und Attribute

Ein oder mehrere `whttp:header`-Elemente können innerhalb von `fault`-Elementen und in den Elementen, die die Bindung einzelner Nachrichten einer Operation spezifiziert, definiert werden um weitere Header-Felder in den HTTP-Nachrichten zu setzen. Weitere Informationen:

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
6.6 Declaring HTTP Headers
http://www.w3.org/TR/wsdl20-adjuncts/#http-headers-decl

Mit Hilfe des `whttp:cookies`-Attribut des `binding`-Elements kann festgelegt werden das die Bindung die Unterstützung von Cookies erfordert. Weitere Informationen:

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
6.10 Specifying the Use of HTTP Cookies
http://www.w3.org/TR/wsdl20-adjuncts/#http-cookies-decl

Das `whttp:ignoreUncited`-Attribut des `operation`-Elements wird in Verbindung mit IRI-Serialisierung und dem `whttp:location`-Attribut verwendet. Es legt fest ob in der Location nicht angegebene Nachrichtenteile ignoriert werden sollen. Weitere Informationen:

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
6.8.2.2.2 Controlling the serialization of the query string in the request IRI
http://www.w3.org/TR/wsdl20-adjuncts/#_http_urlencoded_stop

2.4.1.7. Ein Beispiel für ein HTTP-Binding

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace="http://company.example.com/webservices/myService"
    xmlns:tns="http://company.example.com/webservices/myService"
    xmlns:whttp="http://www.w3.org/ns/wsdl/http"
    ... >

  ...

  <binding name="sampleHTTPBinding" interface="tns:sampleInterface"?
      type="http://www.w3.org/ns/wsdl/http"
      whttp:methodDefault="GET"
      whttp:queryParameterSeparatorDefault="&"
      whttp:cookies="false"
      whttp:contentEncodingDefault="UTF-8">

    <fault ref="tns:sampleFault" whttp:code="404" />

    <operation ref="tns:sampleOperation"
        whttp:location="{requestContent}"
        whttp:inputSerialization="application/x-www-form-urlencoded"
        whttp:outputSerialization="application/xml"
        whttp:faultSerialization="application/xml"
        whttp:ignoreUncited="false">

      <input messageLabel="In" whttp:contentEncoding="ISO-8859-1" />
      <output messageLabel="Out" whttp:contentEncoding="UTF-16" />

      <outfault ref="tns:sampleFault" messageLabel="Out" />

    </operation>

  </binding>

  ...

</description>
```

2.4.2. SOAP-Binding

Die WSDL-SOAP-Erweiterung bietet als Übermittlungsverfahren die SOAP-Versionen 1.2 und teilweise 1.1 an. Die Verwendung der Erweiterung wird mittels des Namensraum `http://www.w3.org/ns/wsdl/soap` als Wert des `type`-Attributs angezeigt und die Version im `wsoap:version`-Attribut festgelegt (Standardwert: `1.2`). Somit wird es ermöglicht, die Konzepte und Möglichkeiten die SOAP bereitstellt in Web Services zu verwenden.

2.4.2.1. SOAP-Protocol

SOAP Nachrichten werden meist via HTTP transferiert. Es sind jedoch auch andere zu Grunde liegende Protokolle möglich. Spezifiziert wird das Übertragungsprotokoll im `wsoap:protocol`-Attribut. Im Falle von HTTP mittels des Werts `http://www.w3.org/2003/05/soap/bindings/HTTP/`. Für die Verwendung von HTTP werden verschiedene HTTP-spezifische Elemente und Attribute des HTTP-Binding unterstützt um Konfigurationen für die Übertragung vorzunehmen. Die jeweilige Bedeutung wird im Kapitel *2.4.1. HTTP-Binding* erläutert. Mögliche Elemente sind:

```
whttp:header
```

weiterhin folgende Attribute innerhalb der jeweiligen Elemente:

```
whttp:queryParameterSeparatorDefault
whttp:queryParameterSeparator
whttp:contentEncodingDefault
whttp:contentEncoding
whttp:cookies
whttp:location
whttp:ignoreUncited
```

Weitere HTTP-Konfigurationen werden indirekt durch die Konfiguration der SOAP-Bindung vorgeschrieben.

2.4.2.2. SOAP-MEP

SOAP selbst nutzt verschiedene MEPs um den Ablauf eines Nachrichtenaustauschs zu charakterisieren. Dabei lässt sich über die Attribute `wsoap:mepDefault` (im `binding`-Element) und `wsoap:mep` (im `operation`-Element) das zu nutzende SOAP-MEP spezifizieren. Zu bemerken ist, dass sich mit Hilfe der vordefinierten SOAP-Binding Erweiterung nicht alle WSDL-MEPs auf ein entsprechendes SOAP-MEP abbilden lassen. Möglich sind folgende Zuordnungen:

WSDL-MEP	SOAP-MEP
`http://www.w3.org/ns/wsdl/in-out`	`http://www.w3.org/2003/05/soap/mep/request-response/`
`http://www.w3.org/ns/wsdl/in-only`	`http://www.w3.org/2003/05/soap/mep/request-response/`
`http://www.w3.org/ns/wsdl/robust-in-only`	`http://www.w3.org/2003/05/soap/mep/request-response/`
`http://www.w3.org/ns/wsdl/in-out`	`http://www.w3.org/2003/05/soap/mep/soap-response/`

Wenn das WSDL-in-out-MEP genutzt wird ist es nicht notwendig das SOAP-MEP explizit zu spezifizieren. In diesem Fall wird automatisch das Request-Response-MEP als Muster angenommen. Wird ein anderes WSDL-MEP genutzt so muss ein geeignetes SOAP-MEP angegeben werden.

Die HTTP-Methode wird am genutzten SOAP-MEP festgemacht:
`POST` bei `http://www.w3.org/2003/05/soap/mep/request-response/` und
`GET` bei `http://www.w3.org/2003/05/soap/mep/soap-response/`.

Detaillierte Informationen wie die Zuordnung geschieht kann unter folgender Adresse nachgelesen werden:

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
5.10.4 Binding WSDL 2.0 MEPs to SOAP 1.2 MEPs
http://www.w3.org/TR/wsdl20-adjuncts/#wsdl-mep-soap-mep

Weitere Informationen zu SOAP-MEPs sind unter folgender Adresse zu finden:

SOAP Version 1.2 Part 2: Adjuncts (Second Edition)
6. SOAP-Supplied Message Exchange Patterns and Features
http://www.w3.org/TR/soap12-part2/#soapsupmep

2.4.2.3. SOAP-Action

Die SOAP-Aktion kann im `wsoap:action`-Attribut einer jeden Operation angegeben werden, um den SOAP-Action Parameter des `application/soap+xml` Media Type zu spezifizieren. Damit ist es zum Beispiel möglich, Versionsunterschiede im gebundenen Interface anzuzeigen.

Wird das SOAP-MEP `http://www.w3.org/2003/05/soap/mep/request-response/` benutzt so wird dieses Attribut ignoriert.

Weitere Informationen zu SOAP-Action finden sich unter:

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
5.10.3 SOAP 1.2 Binding Rules
http://www.w3.org/TR/wsdl20-adjuncts/#soap12-defaults

RFC 3902 - The "application/soap+xml" media type
http://tools.ietf.org/html/rfc3902

2.4.2.4. SOAP-Error-Codes – Fehlercodes

Ein SOAP-spezifischer Fehlercode kann im `wsoap:code`- und `wsoap:subcode`-Attribut der `fault`-Elemente referenziert werden. Standardwert für beide Attribute ist `#any`. Die verschiedenen SOAP-Error-Codes werden im folgenden Dokument definiert:

SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)
5.4.6 SOAP Fault Codes
http://www.w3.org/TR/soap12-part1/#faultcodes

2.4.2.5. Weitere Elemente und Attribute

Das `wsoap:module`-Element erlaubt die Spezifizierung von der Bindung genutzter SOAP-Module. Weitere Informationen:

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
5.8 Declaring SOAP Modules
http://www.w3.org/TR/wsdl20-adjuncts/#soap-module-decl

Welche zusätzlichen Informationen in den Header-Feldern der SOAP-Nachrichten untergebracht werden, lässt sich mit Hilfe der `wsoap:header`-Elemente festlegen. Weitere Erläuterungen:

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
5.9 Declaring SOAP Header Blocks
http://www.w3.org/TR/wsdl20-adjuncts/#soap-headers-decl

2.4.2.6. Ein Beispiel für ein SOAP-Binding

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace="http://company.example.com/webservices/myService"
    xmlns:tns="http://company.example.com/webservices/myService"
    xmlns:sns="http://company.example.com/schemas/myService"
    xmlns:whttp="http://www.w3.org/ns/wsdl/http"
    xmlns:wsoap="http://www.w3.org/ns/wsdl/soap"
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    ... >

  ...

  <binding name="sampleSOAPBinding" interface="tns:sampleInterface"
      type="http://www.w3.org/ns/wsdl/soap"
      wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/"
      wsoap:version="1.2"
      whttp:contentEncodingDefault="UTF-8">

    <fault ref="tns:sampleFault" wsoap:code="soap:Sender" />

    <operation ref="tns:sampleOperation"
        wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>

      <input messageLabel="In">
        <wsoap:header element="sns:requestDate"/>
      </input>

    </operation>

  </binding>

  ...

</description>
```

2.5. Das `service`-Element – Anbieten von gebundenen Schnittstellen

Das `service`-Element legt fest, wo ein Service zu erreichen ist. Ein Service hat genau eine Schnittstelle die sein Interaktionsspektrum definiert. Diese wird im `interface`-Attribut referenziert. Zusätzlich muss jeder Service eindeutig innerhalb des Ziel-Namensraumes über das `name`-Attribut benannt werden.

In sogenannten Endpoints innerhalb des Service wird festgelegt unter welcher Adresse eine Bindung der angebotenen Schnittstelle erreichbar ist. Ein Service kann beliebig viele Bindungen anbieten um größtmögliche Kompatibilität zu möglichen Kommunikationspartnern zu bieten. Endpunkte müssen innerhalb des `service`-Elements eindeutig benannt werden. Bei wiederverwendbaren Bindungen wird das zu bindende Interface erst in dieser Endpunktdefinition zugewiesen.

Ein Beispiel:

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace="http://company.example.com/webservices/myService"
    xmlns:tns="http://company.example.com/webservices/myService"
    ... >

  ...

  <service name="sampleService"
      interface="tns:sampleInterface">

    <endpoint name="sampleEndpoint"
        binding="tns:sampleBinding"
        address="http://company.example.com/myService"/>

  </service>

</description>
```

2.5.1. Benutzerauthentifizierung bei HTTP- und SOAP-via-HTTP-Bindungen

Wird HTTP als zu Grunde liegendes Protokoll der im Endpoint referenzierten Bindung verwendet, so kann die Nutzung des Service durch eine Authentifizierung beschränkt werden. Die Beschränkung wird mit Hilfe der `whttp:authenticationScheme`- und `whttp:authenticationRealm`-Attribute im jeweiligen `endpoint`-Element konfiguriert. Zulässig Werte für das Schema sind `basic` und `digest` und Standardwert für den Realm ist die leere Zeichenkette.

Weitere Informationen wie die Authentifizierung umgesetzt wird ist in folgendem Dokument beschrieben:

RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication
http://tools.ietf.org/html/rfc2617

2.6. Import- und Inklusionsmechanismen

Um die Wiederverwendbarkeit zu fördern, ist es möglich, Teile von Web Service Beschreibungen in einzelne Dokumente auszulagern und diese in einem Anderen zu importieren bzw. inkludieren.

Eine Inklusion bezeichnet dabei eine Art Kopiervorgang und wird mit Hilfe des `include`-Elements und dessen `location`-Attribut angewendet. Inkludiertes und inkludierendes Dokument müssen den selben Ziel-Namensraum besitzen.

Um ein WSDL-Dokument mit einem anderen Ziel-Namensraum zu benutzen wird ein Import verwendet. Er wird im `import`-Element und dem dazugehörigen `namespace`-Attribut spezifiziert. Ein WSDL-Prozessor sollte automatisch die Komponenten dieses Namensraum, sprich die WSDL-Dokumente mit diesem Ziel-Namensraum, finden. Dem Prozessor kann aber über das `location`-Attribut Hinweise auf mögliche Orte geben werden, wo das (die) Dokument(e) zu finden sind. Zu beachten ist dabei, dass bei einem Import die XSD-Dokumente, welche von dem importieren Web Service benutzt werden, nicht innerhalb des Importierenden sichtbar sind.

Im `types`-Element kann außerdem mit Hilfe des `xs:import`-Elements eine XSD importiert werden. Über ein `namespace`-Attribut wird festgelegt welche Schemadefinition importiert werden soll und das `schemaLocation`-Attribut gibt wieder Hinweise auf den Ort. Mit diesem Mechanismus können auch die Schemata eines importierten Web Service verfügbar gemacht werden (eine Angabe der Location ist dabei nicht notwendig).

Ein Beispiel für verschiedene Importe und Inklusionen:

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace="http://company.example.com/webservices/myService"
    xmlns:tns="http://company.example.com/webservices/myService"
    xmlns:othertns="http://company.example.com/webservices/myOtherService"

    xmlns:sns="http://company.example.com/schemas/myService"
    xmlns:extrasns="http://company.example.com/schemas/myService-extra"
    xmlns:othersns="http://company.example.com/schemas/myOtherService"
    ... >

  ...

  <import namespace="http://company.example.com/webservices/myOtherService"
      location="http://company.example.com/webservices/myOtherService.wsdl" />

  <include location="myService-interfaces.wsdl" />

  <types xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:import
        namespace="http://company.example.com/schemas/myOtherService" />

    <xs:import namespace="http://company.example.com/schemas/myService-extra"
        schemaLocation="myService-extra.xsd" />

    <xs:schema targetNamespace="http://company.example.com/schemas/myService"
        xmlns="http://company.example.com/schemas/myService"
        xmlns:etwosns="http://company.example.com/schemas/myService-extra2">

      <xs:import
```

```
            namespace="http://company.example.com/schemas/myService-extra2"
            schemaLocation="myService-extra2.xsd" />

        <xs:include schemaLocation="myService-part2.xsd" />

      </xs:schema>
    </types>

    ...

  </description>
```

Weitere Beispiele und Informationen über die Import- und Inklusionsmechanismen findet man unter:

Web Services Description Language (WSDL) Version 2.0 Part 0: Primer
2.3.3 Summary of Import and Include Mechanisms
http://www.w3.org/TR/wsdl20-primer/#more-types-import-include-summary
3. Advanced Topics I: Importing Mechanisms
http://www.w3.org/TR/wsdl20-primer/#advanced-topic_ii

2.7. Unterschiede zu WSDL 1.1

WSDL 2.0 ist darauf ausgerichtet, unabhängig von Plattform, Programmiersprache oder Protokoll, Web Services zu beschreiben. Die ursprüngliche Entwicklung entstammt jedoch dem SOAP-Umfeld und WSDL 1.1 ist wesentlich stärker an jene Designkonzepte gebunden. Die WSDL 1.1 Spezifikation ist nicht als *W3C Recommendation* verfügbar, sondern lediglich als *W3C Note*:

Web Services Description Language (WSDL) 1.1
http://www.w3.org/TR/wsdl

Zu beachten ist, dass in Version 1.1 Schnittstellen und Endpunkte in `portType`- und `port`-Elementen definiert werden. Sie haben jedoch die gleiche Struktur wie die entsprechenden WSDL 2.0 Elemente. Außerdem trägt das Wurzelelement den Namen `definitions` und nicht `description`.

Weiterhin müssen Formatdefinitionen für Nachrichten in explizit definierten `message`-Elementen referenziert werden, die dann von den Operationen einer Schnittstelle benutzt werden können. Eine Nachricht kann in WSDL 1.1 dabei aus mehreren Teilen bestehen (`part`-Kindelement des `message`-Elements). So können Nachrichten aus einzelnen Teilen von Schemadefinitionen zusammengesetzt werden. Diese Funktion wird jedoch von vielen Schemasprachen wie zum Beispiel XSD selbst angeboten und das mit wesentlich flexibleren Möglichkeiten (Vererbung, Einschränkung, Vereinigung und Andere im Falle von XML-Schema), so dass die entsprechenden `message`-Elemente meist nur aus einem einzigen Teil bestehen. Diese Redundanz wurde in WSDL 2.0 entfernt und Formatdefinitionen werden direkt von den einzelnen Nachrichten einer Operation referenziert.

Die Bindungserweiterungen von WSDL 1.1 sind so gestaltet, dass die Konfiguration der Bindung in eigenen Elementen vorgenommen wird und nicht innerhalb von Attributen die den Elementen der entsprechenden WSDL 2.0-Komponenten zugeordnet sind. So sieht eine SOAP-Bindung in WSDL1.1 folgendermaßen aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
    targetNamespace="http://company.example.org/webservices/myService"
    xmlns:tns="http://company.example.org/webservices/myService"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

  ...

  <binding name="sampleBinding" type="tns:samplePort">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />

    <operation name="sampleOperation">
      <soap:operation
          soapAction="http://www.example.org/sampleService/sampleOperation"
          style="document" />

      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
```

```
        <fault name="sampleFault">
          <soap:fault use="literal" name="sampleFault" />
        </fault>

      </operation>

    </binding>

    ...

  </definitions>
```

Das Prinzip der erweiterbaren MEPs ist in WSDL 1.1 noch nicht vorhanden. So wird das Austauschmuster alleine über das (Nicht-) Vorhandensein von den `input`-, `output`- und `fault`-Elementen spezifiziert, sodass lediglich in-only, out-only, in-out und out-in – letztere beide mit optionaler Fault-Nachricht – als Muster möglich sind[12].

12 Die Austauschmuster werden in WSDL 1.1 noch folgend benannt: one-way, notification, request-response und solicit-response (gleiche Reihenfolge wie im Text)

3. Web Services und RDz

In diesem Kapitel wird die Entwicklungsumgebung *IBM Rational Developer for System z* (RDz) zuerst vorgestellt und grundlegende Anwendungen durchgeführt. Danach wird ein komplexeres Beispiel zur Integration von Mainframe-Software und Web Services vollzogen.

Anmerkung: Die derzeit aktuelle Version von RDz verwendet noch WSDL 1.1 und beinhaltet noch keine Assistenten oder spezielle Ansichten für die neuere Spezifikation. Weitere Informationen zu den Unterschieden zwischen WSDL 1.1 und 2.0 findet man in Kapitel *2.7. Unterschiede zu WSDL 1.1*.

3.1. Download und Installation

Zuerst muss man sich eine Kopie der IDE herunterladen. Von der IBM Website kann man eine 60-Tage Testlizenz der aktuellsten Version erhalten. Im Folgendem wird kurz vorgestellt wie Sie solch eine Kopie für Windows Betriebssysteme beziehen können.

Startpunkt ist die IBM Website:

IBM - Deutschland
http://www.ibm.de

In der Hauptnavigation (horizontal oberes Seitenviertel) wählen Sie *Support & Downloads → Downloaden → Test- und Demoversionen (US)* und gelangen dann zu dem englischen Download-Repository. Dort suchen Sie nach *Rational Developer* oder navigieren über *By product → R → Rational Developer* zur Liste verfügbarer Software. Da sollte als eines der ersten Resultate *Rational Developer for System z 7.6 Trial Multiplatform Multilingual* erscheinen (oder eine neuere Version). Wählen Sie dieses Element aus und auf der folgenden Seite fahren Sie mit *Continue* fort (die gewünschte Sprache ist in diesem Fall egal, da wir ein mehrsprachiges Paket herunterladen, sonst wählen Sie hier ihre gewünschte Sprache aus). Sie werden nun aufgefordert, sich mit Ihrer *IBM ID* einzuloggen. Falls Sie keine besitzen benutzen Sie den Link *Get an IBM ID* und füllen Sie alle nötigen Felder aus (mehrseitige Registrierung). Danach kommen Sie automatisch zum Login zurück und können sich nun anmelden. Nach dem Betätigen des *Sign-In* Buttons werden Ihnen einige Fragen zum Download gestellt. Beantworten Sie diese nach Ihren Bedürfnissen. (Achtung: Im Abschnitt *Privacy* die Checkbox *Other communications: Please do not use the information I have provided here* auswählen, wenn Sie auf die Weitergabe Ihrer Daten seitens IBM verzichten wollen.) Akzeptieren Sie nun noch die Lizenzvereinbarung und benutzen Sie den *I confirm* Button um zum Download zu gelangen.

Wählen Sie *Download using HTTP* aus. Das gesamte Paket ist ca. 2,2 GB groß. Benötigt für die klientenseitige Installation (die Entwicklungsumgebung) sind die ersten drei Teile:

IBM Rational Developer for System z Trial Setup Disk (Trial Part 1 of 6) Multiplatform Multilingual
RDz76_Trial_Setup.exe (200 MB)

IBM Rational Developer for System z Trial Installation Disk 1 (Trial Part 2 of 6) Multiplatform Multilingual
RDz76_Trial_Installation_Disk1.exe (593 MB)

IBM Rational Developer for System z Trial Installation Disk 2 (Trial Part 3 of 6) Multiplatform Multilingual
RDz76_Trial_Installation_Disk2.exe (548 MB)

Wenn Sie wollen können Sie ebenfalls noch die Dokumentation herunterladen. In den Abbildungen 1 bis 6 sind einige Vorgänge zum Download von RDz dargestellt.

Die heruntergeladenen Dateien sind selbst extrahierende Archive. Entpacken Sie alle in das gleiche Verzeichnis. Nach dem Entpacken befindet sich im Verzeichnis *RDz76_Trial_Setup* die *launchpad.exe* welche das Setup startet. Wählen Sie *Rational Developer for System z installieren* aus und klicken Sie *Rolle eines Serviceentwicklers* an. Der Installationsmanager startet nun und führt Sie durch die Installation. (Hinweis: Auch wenn später angezeigt wird, dass die Programmdatenträger heruntergeladen werden müssen, findet sie der Installationsmanager auf der Festplatte und sie werden nicht erneut heruntergeladen.)

3.1.1. *WebSphere Community Editon*

Um die erstellten Programme lokal zu testen ist ein Application Server notwendig. In diesen Anwendungsbeispielen wird der *WebSphere Application Server Community Editon* (WASCE) verwendet. Das ist ein Public Domain Server aus der Produktreihe der *WebSphere Application Server* von IBM.

Auf der IBM Website:

IBM - Deutschland
http://www.ibm.de

können Sie eine Kopie der aktuellen Version herunterladen. Gehen Sie dazu in der Navigation auf *Produkte* → *Software* → *WebSphere* und unter *Unser Angebot* (Box mittig des Fenster) auf *WebSphere Application Server*. Weiter unter *Ausgewählte Produkte zu Application Server* auf *WebSphere Application Server Community Editon* und von da aus zur englischen Downloadseite. Ab dort geht es ähnlich weiter wie beim Download von RDz. (Gegebenenfalls müssen Sie ihren *IBM ID* Account mit weiteren Informationen aktualisieren.) In der Downloadübersicht laden Sie lediglich die Installationsdatei für *Server-only* herunter. Nach dem herunterladen installieren Sie WASCE in ein neues Verzeichnis.

3.2. Die Arbeitsoberfläche

Öffnen Sie das Programm und wechseln Sie in die Perspektive *Ressource* (oben rechts). Standardmäßig sollte der Projektexplorer, der Editor, die Gliederung (Outline), die Fehleranzeige und die Eigenschaftsansicht vorhanden sein. Falls eines der genannten Dinge fehlt fügen Sie diese Ansicht hinzu (unten links). Nützlich ist auch noch die Navigatoransicht. Fügen Sie diese ebenfalls zur Perspektive *Ressource* hinzu. Dies ist der Ausgangspunkt für alle Anwendungsbeispiele.

3.2.1. Der WSDL-Editor und dazugehörige Komponenten

Legen Sie ein neues allgemeines Projekt mit dem Namen *wsdl-test* an. Wählen Sie im Kontextmenü des Projekts *Neu → Andere...* und da *Web-Services → WSDL*. Sie gelangen nun zum Assistenten um eine neue Web Service Definition zu erstellen.

Im ersten Schritt geben Sie die von *Eclipse* bekannten Ressourceninformationen an. Klicken Sie anschließend auf *Weiter*. Im Folgenden sind schon Voreinstellung getroffen, die Sie normalerweise anpassen sollten, aber für die Erkundung des WSDL-Editors belassen wir alle Eingaben wie Sie sind. Klicken Sie auf *Fertig stellen*.

Die neue WSDL-Datei öffnet sich im speziell angepassten Editor: dem WSDL-Editor (siehe Abbildung 7). Er bietet zwei verschiedene Ansichten zwischen denen in der unteren linken Ecke des Editors gewechselt werden kann. In der Entwurfsansicht sehen Sie eine symbolisierte Darstellung der WSDL-Datei und können hier über die Kontextmenüs der einzelnen Symbole die Definition editieren. In der Quellansicht sehen Sie den XML-Inhalt der WSDL-Datei.

Wechseln Sie zur Entwurfsansicht. Sie sollten nun ein PortType[13], eine Bindung und ein Service sehen. Die Symbole selbst können Untergliederungen enthalten, wie zum Beispiel ein PortType verschiedene Operationen enthält und diese wiederum Nachrichten. In jedem Unterbereich öffnet sich ein Kontextmenü mit zusätzlichen Funktionalitäten, die sich auf den ausgewählten Unterbereich beziehen. Im Kontextmenü des PortType zum Beispiel sehen Sie die Optionen *Operation hinzufügen*, *Löschen* und *Eigenschaften anzeigen*. Gehen Sie nun in das Kontextmenü der Operation NewOperation und Sie sehen eine zusätzliche Funktion: *Fehler hinzufügen*. Außerdem beziehen sich die Optionen *Löschen* und *Eigenschaften anzeigen* nun auf die ausgewählte Operation.

Wenn Sie in die freie Fläche des Editors rechtsklicken, können Sie eine neue Komponente zur Web Service Definition hinzufügen. Normalerweise manifestieren sich sofort alle Kontextaktionen in der Quellansicht, das heißt als XML-Dokument. Einzig die Inhalte eines Binding müssen Sie explizit nach dem Anlegen der Bindung und nach jeder Änderung im referenzierten PortType generieren lassen. Nutzen Sie hierzu die Funktion *Binding-Inhalt generieren...* im Kontextmenü der Bindung.

In der Eigenschaftsansicht sehen Sie die Attribute des aktuell ausgewählten Objekts und können sie ebenfalls in dieser Ansicht modifizieren. Außerdem gibt es noch die bekannte Gliederungsansicht (Outline) der das gesamte Dokument als Baum mit den WSDL-typischen Elementen anzeigt. Aus der Baumansicht sind die gleichen Kontextmenüs wie im Editor verfügbar.

13 WSDL 1.1 Terminologie; seit WSDL 2.0 in Interface umbenannt.

Machen Sie sich mit der Umgebung vertraut und experimentieren Sie etwas. Fügen Sie zum Beispiel neue Komponenten hinzu, löschen Sie andere und ändern Sie die Eigenschaften.

Um die Integrität der erstellten WSDL-Dateien zu Prüfen können Sie im Projektexplorer im Kontextmenü der zu prüfenden WSDL-Datei die Option *Prüfen* (*Validate*) aufrufen. Dies sollten Sie vor dem Einsatz jeder Definition durchführen und eventuelle Fehler oder Warnungen beheben oder zum Debugging vormerken.

3.2.2. *WebSphere* in RDz einrichten und steuern

Um WASCE aus RDz heraus zu steuern muss zuerst ein geeigneter Serveradapter für Eclipse installiert werden. Gehen Sie hierzu auf *Hilfe* → *Software-Updates...* und fügen Sie folgende Update-Site hinzu (siehe Abbildung 8):

IBM Eclipse Update-Site für WASCE
http://download.boulder.ibm.com/ibmdl/pub/software/websphere/wasce/updates/

Danach schließen Sie den Update-Manager wieder und gehen Sie unter *Fenster* (*Window*) auf *Benutzervorgaben* (*Preferences*) und wählen Sie in der Baumstruktur die Option *Server* → *Laufzeitumgebungen* aus. Klicken Sie auf *Hinzufügen...* und im folgenden Assistenten auf *Zusätzliche Serveradapter herunterladen*. Die Suche nach verfügbaren Adaptern kann etwas Zeit in Anspruch nehmen. Sobald die Liste fertig geladen ist suchen Sie nach den WASCE-Adaptern und installieren Sie Version 2.1 (siehe Abbildung 9). Danach muss die IDE neu gestartet werden. Nach dem Neustart müssen Sie eventuell noch die Adapterversion 2.0 installieren, da vielleicht nicht alle Abhängigkeiten ordnungsgemäß aufgelöst wurden. Haben Sie die nötigen Komponenten installiert, sollten Sie im *Hinzufügen...*-Assistenten nun unter IBM den WASCE v2.1 Serveradapter sehen. Wählen Sie diesen aus, haken Sie die Option *Neuen lokalen Server erstellen* an und klicken Sie auf *Weiter*. Nun müssen Sie das Verzeichnis angeben, in dem Sie WASCE installiert haben. Als Laufzeitumgebung verwenden wir *jdk*. Schließen Sie den Vorgang ab (alle vordefinierten Einstellungen beibehalten).

Sie können nun in der Serveransicht den Server starten und stoppen und haben die Möglichkeit, Anwendungen direkt zu aus RDz heraus zu deployen. Es kann sein, dass es beim Deployment eines Projekts aus RDz heraus zu Problemen kommt und trotz korrekter Vorgehensweise die Webanwendungen nicht ordnungsgemäß exportiert werden. In diesem Fall müssen Sie die Applikation als WAR bzw. EAR Archiv exportieren und dann manuell über die Administrationskonsole deployen. Diese ist unter der Adresse

WASCE Administrative Console
https://localhost:8443/console

zu erreichen sobald der Server gestartet ist. Benutzername und Passwort im Auslieferungszustand ist *system* und *manager*.

3.3. Einführungsbeispiel

Ein gutes Einführungsbeispiel das die Erstellung eines Web Service demonstriert ist unter folgender Adresse zu finden:

Web Services for CICS
http://jedi.informatik.uni-leipzig.de/de/tutor_wdz70/tutorial8_wsdl.pdf

Es zeigt wie Web Services mit CICS integriert und auf einem Großrechner installiert werden.

3.4. Web Services in Anwendungsservern mit *Apache Axis*

Um Web Services auf einem Application Server verfügbar zu machen kann man das *Apache Axis* Framework verwenden. Da es eine reine Java-Implementierung ist kann man es in beliebigen Webanwendungen verwenden, welche dann unter J2EE konformen Servern installiert und betrieben werden können (also auch unter *WepSphere Application Server for z/OS*).

Die Anfragen an einen Web Service werden dabei erst von einem `HttpServlet` das *Axis* mitbringt entgegengenommen, danach gemäß den Spezifikationen von WSDL 1.1 verarbeitet und die Antwort zurückgesendet. Die Implementierung eines WSDL PortType wird in Java umgesetzt. Dabei müssen zu dem PortType passende Java Interfaces definiert und anschließend in implementierenden Klassen konkretisiert werden. Zuletzt muss dann noch *Axis* konfiguriert werden, sodass die Web Service Definitionen und die dazugehörigen Klassen gefunden und einander zugeordnet werden können. Das geschieht zur Laufzeit mit Hilfe eines Web Service – genannt *AdminService* – der durch ein gesondertes Servlet angeboten wird. Mit diesem Service kann man entweder über das Kommandozeilenprogramm *AdminClient*[14] oder über einen *Ant Task* interagieren. Beide Varianten benutzen dazu Konfigurationsdateien – sogenannte Deskriptoren – im XML-Format, um dem *AdminService* mitzuteilen, welche Web Services angeboten oder wieder aus dem Angebot herausgenommen werden sollen. Die vorgenommenen Einstellungen werden persistent in der *server-config.wsdd*[15] gespeichert, sodass auch nach einem Neustart des Servers alle installierten Web Services im gleichen Angebotszustand bleiben.

RDz enthält Assistenten, die alle nötigen Konfigurationsdateien und Java-Komponenten bzw. Web Service Definitionen in wenigen Schritten erstellen können. Somit kann man sich auf die Implementierung der Businessfunktionen konzentrieren. Normalerweise sollte es auch möglich sein, die Web Services direkt aus RDz heraus auf dem Server zu installieren, allerdings war diese Funktion sehr fehleranfällig und führte mit der verwendeten RDz-Version nie zum Erfolg. Daher wird das Deployment in diesem Anwendungsbeispiel manuell vorgenommen (über die Administrationskonsole und mit Hilfe des *AdminClient*).

14 Main-Klasse: `org.apache.axis.client.AdminClient`

15 Diese befindet sich abhängig von der Projektstruktur und des genutzten Application Servers entweder im *META-INF* Verzeichnis der Webanwendung oder in serverweiten Konfigurationsverzeichnissen.

3.4.1. *Bottom-Up* Entwicklung eines Web Service

Beim *Bottom-Up* Entwicklungsprozess werden zuerst die Geschäftsfunktionen implementiert, also der Grund (*Bottom*) des Web Service konkretisiert und anschließend werden aus den erstellten Klassen Web Services abstrahiert und diese für Oben bzw. Außen (*Up*) zugänglich gemacht. RDz unterstützt diesen Prozess, indem es aus einer Klasse alle nötigen weiteren *Axis*-Komponenten ableitet und ein passendes WSDL-Dokument erzeugt.

Wir nehmen als einfaches Beispiel die Fibonacci-Zahlen. Legen Sie ein neues dynamisches Webprojekt mit der Java Version 1.4 und der Servlet API Version 2.4 an (siehe Abbildung 10) und erstellen Sie eine Java-Klasse die die Zahlenfolge berechnen kann, zum Beispiel:

```
package org.example.ws.fibonacci;

public class Fibonacci {

  public int calc(int f) {
    if(f < 0 || f > 20)
      return -1;
    if(f == 0)
      return 0;
    if(f == 1 || f == 2)
      return 1;
    int result = 0;
    int m = 1;
    int n = 1;
    for(int i = 2; i < f; i++) {
      result = m + n;
      n = m;
      m = result;
    }
    return result;
  }
}
```

Nun öffnen Sie im Projektexplorer das Kontextmenü der Klasse und wählen Sie *Web-Services → Web-Service erstellen*. Im folgenden Assistenten sollten alle Werte bereits richtig eingetragen sein. Überprüfen Sie sie mit Hilfe von Abbildung 11 und klicken Sie – nach etwaigen Korrekturen – auf *Weiter*. Wenn Sie die auf der DVD beiliegenden Ressourcen verwendet haben müssen Sie im folgenden Fenster den Haken vor der Methode *main* entfernen. Alle anderen Einstellungen können beibehalten werden und wir schließen den Assistenten mit *Fertig stellen* ab.

Es wurden nun alle nötigen Komponenten angelegt und Sie müssen lediglich noch die URL des Service-Endpunkts an den entsprechenden Pfad anpassen. Öffnen Sie dazu das soeben generierte WSDL-Dokument, welches sich im *WebContent/wsdl* Verzeichnis befindet, im WSDL-Editor und ändern Sie den Standardwert `http://tempuri.org/...` auf `http://localhost:8080/...` für die lokale Testumgebung bzw. später auf den DNS Namen des Geschäftsservers.

3.4.2. *Top-Down* Entwicklung eines Web Service

Die *Top-Down*-Vorgehensweise steht im Gegensatz zur *Bottom-Up*-Entwicklung. Dabei spezifiziert man zuerst die Web Service Definition, welche die Sicht von Oben bzw. Außen (*Top*) auf den Service beschreibt. Danach werden die Geschäftsfunktionen implementiert, also das, was Unten (*Down*) unter der Oberfläche passieren soll. Die Assistenten von RDz

erstellen dabei automatisch aus einem WSDL-Dokument alle nötigen Java-Interfaces und -Klassen. Die Rahmenklassen muss man dann nur noch um den gewünschten – mehr oder weniger komplexen – Programmcode ergänzen.

Wir nehmen wieder die Fibonacci-Zahlen als Beispiel. Legen Sie ein weiteres dynamisches Webprojekt an und erstellen Sie über dessen Kontextmenü mit der Option *Neu → Andere...* ein neues WSDL-Dokument direkt im Projektverzeichnis. Die Signatur der Berechnungsfunktion ist einfach, sodass wir direkt das vorgefertigte Gerüst verwenden können und lediglich die Typen des Operationsparameter und des Rückgabewerts in der Schemadefinitionen auf `xsd:int` umstellen müssen. Ändern Sie noch die Namen des automatisch generierten Inhalts in aussagekräftige Bezeichnungen um. Das Dokument sollte dann folgende Form haben:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="fibonacci"
    targetNamespace="http://www.example.org/fibonacci/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.example.org/fibonacci/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types>
    <xsd:schema targetNamespace="http://www.example.org/fibonacci/">
      <xsd:element name="calc">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="f" type="xsd:int" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="calcResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="result" type="xsd:int" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="calcRequest">
    <wsdl:part element="tns:calc" name="f" />
  </wsdl:message>
  <wsdl:message name="calcResponse">
    <wsdl:part element="tns:calcResponse" name="result" />
  </wsdl:message>
  <wsdl:portType name="Fibonacci">
    <wsdl:operation name="calc">
      <wsdl:input message="tns:calcRequest" />
      <wsdl:output message="tns:calcResponse" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="Fibonacci" type="tns:Fibonacci">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="calc">
      <soap:operation
          soapAction="http://www.example.org/fibonacci/calc" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
```

```
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="FibonacciService">
    <wsdl:port binding="tns:Fibonacci" name="Fibonacci">
      <soap:address location="http://www.example.org/" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Die Adresse des Endpunkts müssen Sie in diesem Schritt noch nicht festlegen – Sie können also den Standardwert stehen lassen. Das `soapAction`-Attribut der Operationsbindung sollten Sie jedoch auf einen besseren virtuellen Namen ändern oder auf leeren Inhalt reduzieren (nicht löschen). Prüfen Sie anschließend noch das Dokument über die entsprechende Funktion von RDz.

Benutzen Sie jetzt die Option *Web-Service → Gerüst-Java-Bean generieren* aus dem Kontextmenü der WSDL-Datei. Es erscheint der bereits bekannte Assistent mit entsprechenden Einstellungen. Gleichen Sie diese mit Abbildung 12 ab und klicken Sie anschließend auf *Fertig stellen*. RDz hat nun für jeden PortType ein entsprechendes Java Interface und eine Klasse, welches dieses Interface implementiert, angelegt und Sie müssen nur noch den Code zur Berechnung in die Klasse *FibonacciImpl* einfügen. Außerdem hat RDz eine überarbeitete Version des WSDL-Dokuments im *WebContent/wsdl* Verzeichnis erstellt, in diesem Sie noch die URL des Service-Endpunkt anpassen müssen. Dieses Dokument ist dann auch die Version, die Sie zum Testen im *Web Service Explorer* verwenden und die Sie an die Nutzer des Web Service verteilen müssen.

3.4.3. Web Services auf dem Application Server installieren

Prüfen Sie zunächst die *web.xml* der Webanwendung. Stellen Sie sicher, dass der Pfad zu unserer Applikation dem in der Web Service-Definition entspricht. Das heißt, wenn wir spezifiziert haben, dass die Funktionen des Service unter http://company.example.org/fibonacci/services/ erreichbar sein sollen und das Servlet-Mapping zum `AxisServlet` ist auf `/services/*` eingestellt, dann muss der sogenannte Kontextpfad der Anwendung auf `/fibonacci` gesetzt werden. Wie dieser dem Application-Server mitgeteilt wird ist von dem eingesetzten Server abhängig. Die *WebSphere*-Server erhalten den Kontextpfad der Anwendung entweder über einen externen Deployment-Descriptor, der bei der Installation der Webanwendung mitgeliefert wird oder über das `id`-Attribut der *web.xml*[16].

Exportieren Sie nun das Projekt, das den Web Service enthält als WAR-Datei und benutzen Sie die Administrationskonsole von WASCE um die Anwendung zu installieren. Gehen Sie hierzu in der Hauptnavigation unter *Applications* auf *Deploy New*. Kontrollieren Sie anschließend mit der Option *Applications → Web App WARs* ob die Applikation ordnungsgemäß läuft.

Als nächstes müssen Sie mit dem *AdminClient* von *Axis* den Web Service publizieren. Aufgerufen wird er wie ein normales Javaprogramm mit folgenden Parametern (siehe Abbildung 13 für ein konkreten Aufruf mit Ergebnis):

```
java -cp <AXIS-LIBS>
     org.apache.axis.client.AdminClient
     -lhttp://localhost:8080/<CONTEXT-PATH>/services/AdminService
     <LOCAL-PATH-TO-DEPLOYMENT-DESCRIPTORS>/deploy.wsdd
```

16 Im `id`-Attribut ohne die Pfadtrennelemente, zum Beispiel `id="fibonacci"`.

Das Programm benötigt verschiedene Anwendungsbibliotheken um ordnungsgemäß ausgeführt zu werden. Diese sind:

`axis.jar`
`commons-discovery-0.2.jar` (eventuell aktuellere Version)
`commons-logging.jar`
`jaxrpc.jar`
`saaj.jar`
`wsdl4j.jar`

Sie finden die Bibliotheken im *WEB-INF/lib* Verzeichnis Ihrer Webanwendung (entweder im Workspace von RDz oder im Anwendungsverzeichnis[17] des Application-Server). Über die `-cp`-Option können Sie die Archive – per Semikolon getrennt – manuell bei jedem Aufruf des *AdminClient* in den Klassenpfad der JVM einfügen oder Sie kopieren sie in das Standarderweiterungsverzeichnis Ihrer *Java Runtime Environment*[18], fügen sie dem globalen Klassenpfad hinzu oder exportieren sie als Variable in Ihrer Shell.

RDz generiert grundsätzlich zwei verschiedene Deskriptoren, um den Web Service zu publizieren (*deploy.wsdd*) und ihn wieder aus dem Serviceangebot zu entfernen (*undeploy.wsdd*). Sie finden diese ebenfalls in einem Unterordner des *WEB-INF* Verzeichnisses[19].

Der *AdminClient* gibt über die Kommandozeile einen Bericht über die an *Axis* vorgenommenen Konfigurationen aus. Zu beachten ist, dass in der Standardkonfiguration dieser Service ausschließlich lokal verwendet werden darf, das heißt vom gleichen Rechner aus auf dem der Anwendungsserver läuft. Sie müssen sich also gegebenenfalls per Remote auf dem entsprechenden System einloggen um die Installation des Web Service vorzunehmen.

3.4.4. Web Services mit dem *Web Service Explorer* testen

Der *Web Service Explorer* ist ein einfacher generischer Web Service Klient, um Web Services zu testen. Sie rufen ihn auf, indem Sie im Kontextmenü einer WSDL-Datei die Option *Web-Services → Mit Web-Services-Explorer testen* auswählen. Es öffnet sich eine Website mit einer Übersicht aller im Dokument definierten Services. Wenn Sie eine Operation ausgewählt haben können Sie die dazugehörigen Parameter eingeben und mit einem Klick auf *Los* starten Sie den Aufruf. Im Bereich *Status* finden Sie Meldungen über die zuletzt durchgeführte Aktion, also im Falle eines Aufrufs das erhaltene Resultat bzw. etwaige Fehlermeldungen. In Abbildung 14 sehen Sie ein Anwendungsbeispiel des *Web Service Explorer*. Sie können bei Bedarf weitere Endpunktadressen hinzufügen, um die Anfragen an andere Installationen des Web Service zu senden. Klicken Sie hierzu in der Baumansicht auf das entsprechende Binding, zu dem Sie Endpunkte hinzufügen möchten und wählen Sie die Option *Hinzufügen* im Abschnitt *Endpunkte*. Tragen Sie die gewünschte Adresse in die neu hinzugekommene Zeile ein und speichern Sie sie mit einem Klick auf *Los*. Sie können nun zwischen den Endpunkten bei der Ausführung einer Operation der Bindung wählen.

17 Bei WASCE ist dies das Verzeichnis *repository* im Installationsverzeichnis. Dort werden manuell installierte Anwendungen in das Verzeichnis *default/<YOUR-APPLICATION>/<UID>* entpackt.
18 Bei der *Sun* JRE ist dies das *lib/ext* Verzeichnis im Java Installationsordner.
19 RDz generiert dabei die Unterordner anhand der Service-Endpunkt URLs, zum Beispiel *org/example/company/<YOUR-SERVICE-NAME>*

3.4.5. Generierte Artefakte des Web Service

Wenn Sie in RDz mit *Apache Axis* und Web Services arbeiten, treffen Sie immer wieder auf ähnliche benannte Java Klassen- und Interface. Diese können dazu verwendet werden, um Klienten Zugriff auf den Web Service zu verschaffen und – wie Sie schon bei der *Top-Down* Entwicklung gesehen haben – die Geschäftsfunktionen zu implementieren. Im Folgenden wird eine kleine Referenz der generierten Artefakte gegeben:

Allgemein eingesetzte Artefakte:

`<YOUR-PORTTYPE-NAME>.java` – Java Interface
Dies ist ein einfaches Interface, welches die Operationen des entsprechenden PortType als Javamethoden deklariert.

Serverseitig eingesetzte Artefakte:

`<YOUR-PORTTYPE-NAME>Impl.java` – Java Klasse
Eine Klasse, die die Geschäftsfunktionen eines PortType und somit das entsprechende Java Interface implementiert.

`<YOUR-PORTTYPE-NAME>Skeleton.java` – Java Klasse
Wrapper um die Implementierung eines PortType. Reichert Sie mit Metainformationen an und verwaltet die Lebensdauer der Geschäftsobjekte.

Klientenseitig eingesetzte Artefakte:

`<YOUR-SERVICE-NAME>.java` – Java Interface
Definiert Methoden mit denen man sich die verschiedenen Objekte beschaffen, die den Service über einen bestimmten Endpunkt aufrufen.

`<YOUR-SERVICE-NAME>Locator.java` – Java Klasse
Implementierung des Service Interface.

`<YOUR-BINDING-NAME>Stub.java` – Java Klasse
Objekte dieser Klasse ermöglichen den Zugriff auf einen Service mit Hilfe des in der entsprechenden Bindung spezifizierten Mechanismus.

`<YOUR-PORTTYPE-NAME>Proxy.java` – Java Klasse
Verkapselt den kompletten Zugriff auf einen Web Service für eine einfache Benutzung gemäß OO-typischen Paradigmen.

3.4.6. Web Service Klientenentwicklung

Die Erstellung von Klienten für Web Services gestaltet sich unter RDz mit Hilfe der von *Axis* erstellten Klassen als simples Anlegen eines Klientenprojekts. Wählen Sie im Kontextmenü der WSDL für die Sie einen Klient entwickeln wollen *Web-Services → Client generieren* aus. Stellen Sie den Schieberegler auf *Client implementieren* und wählen Sie ein vorhandenes bzw. neues Zielprojekt aus. Sie können den Klienten als Java-Dienstprogramm (Kommandozeile) oder dynamisches Webprojekt konzipieren. Klicken Sie anschließend auf *Fertig stellen* und das Zielprojekt wird – wenn nicht bereits vorhanden – angelegt und entsprechend konfiguriert. Im Java-Quellordner finden Sie nun die Klasse `<YOUR-PORTTYPE-NAME>Proxy.java`, die Sie dazu verwenden um mit dem entsprechenden Service zu kommunizieren. Sie bietet zwei verschiedene Konstruktoren an: einen parameter-

losen der alle Kommunikationsparameter gemäß der Web Service Definition konfiguriert und einen der einen String entgegen nimmt, um die Endpunkt-Adresse der WSDL zu überschreiben. Bei Letzterem muss sichergestellt werden, dass der verwendete Endpunkt die entsprechende, in der Definition festgelegte Bindung verwendet. Das instanziierte Objekt implementiert das Interface des PortType. Die einzelnen Web Service-Operationen können somit wie ganz normale Java-Methoden aufgerufen werden.

3.5. Skat-Ergebnistabelle als Web Service

Um die Möglichkeiten von Web Services innerhalb von Enterprise Software Systemen aufzuzeigen, betrachten wir eine simple Problemstellung – die Verwaltung und Speicherung einer Skat-Ergebnistabelle – in einer komplexeren Multi-Tier-Architektur. Web Services fungieren hierbei als Verbundstück zwischen dem Klient und dem Middle-Tier, wo die Geschäftsfunktionen angestoßen und ausgeführt werden. Als Klient werden wir den *Web Service Explorer* von RDz verwenden wobei auch RichClients, Web Frontends oder andere Geschäftsprogramme denkbar sind. Das Backend stellt in unserem Falle die Datenhaltung mit Hilfe einer DB2-Datenbank eines z/OS Systems dar. Die Middle-Tier-Software wird auf einem JavaEE Server laufen und *Axis* als Web Service Framework verwenden. Der Application Server kann ebenfalls auf dem gleichen z/OS System laufen oder wie in unserem Falle innerhalb einer lokalen Testumgebung.

3.5.1. Vorbereitungen

Zunächst müssen die Funktionen festgelegt werden, die verwendbar sein sollen. Zuerst wäre da das Hinzufügen von Spielen zu den Ergebnissen. Ein Spiel besteht hierbei aus dem Namen eines Spielers und einem positivem oder negativem ganzzahligem Spielwert. Weiterhin soll eine Liste aller Spiele abgerufen und der derzeitige Punktstand eines gegebenen Spielers zurückgegeben werden können. Zuletzt sollte die Möglichkeit bestehen, die Liste zurück setzen zu können.

3.5.1.1. Auswahl des Implementierungsprozess

Die nächste Frage, die man sich stellen muss ist, welche Vorgehensweise bei der Implementierung verwendet werden soll. Da der Klient schon vorhanden ist würde sich eine Top-Down-Modellierung anbieten. Um jedoch die Flexibilität von Web Services aufzuzeigen werden, wir nach der Outside-In-Methode vorgehen, das heißt wir implementieren bzw. konfigurieren zuerst die äußeren Teile des Softwaresystems und zuletzt die Kernkomponenten. In unserem Falle müssen wir lediglich die Datenbank vorbereiten und ein entsprechendes – sehr überschaubares – Relationsmodell für unser Problem entwickeln. Zuletzt werden wir im Bottom-Up-Prozess unsere Middle-Tier-Software implementieren.

3.5.2. Aufsetzen der Datenbank

In dieser Arbeit werden wir eine z/OS-DB2-Installation verwenden, um unsere Datenbank zu verwalten und zu speichern. Den Zugang für ein solches System erhalten Sie von der Universität Leipzig oder der Universität Tübingen (Näheres hierzu in Kapitel *1.3. Software und Mainframe-Zugriff*). Wenn Sie noch keine Erfahrung mit DB2 unter z/OS und dessen Verwendung unter TSO/ISPF haben oder Sie Ihr Wissen auffrischen wollen, sollten Sie unbedingt folgendes Tutorial durchlesen und durchführen:

Tutorial 4 DB2
http://jedi.informatik.uni-leipzig.de/de/tutors/tutor4.pdf

Bitte beachten Sie, dass dieses Tutorial noch auf OS/390 basiert. Auf den Testmaschinen in Leipzig und Tübingen läuft z/OS, wo die Tools für den Datenbankzugriff etwas andere Namen haben und der Vorgang etwas anders abläuft, zum Beispiel haben Sie keine Rechte, um Datenbanken und StorageGroups anzulegen.

Das Relationsmodell für das gegebene Problem ist äußerst einfach. Benötigt wird nur eine Tabelle mit Primärschlüssel, einem `VARCHAR(100)` Feld `spieler` für den Spielernamen und ein `INT` Feld `spielwert` für den Spielwert. Falls auf Ihrem Account noch keine Datenbank vorhanden ist, wenden Sie sich an den Systemadministrator des verwendeten Großrechners. Erstellen Sie einen neuen sinnvoll benannten TableSpace und darin eine Tabelle `spiele` mit dem erwähnten Aufbau.

3.5.3. Implementierung des Middle-Tier

Zur Kommunikation mit der Datenbank benötigen wir einen JDBC-Treiber für DB2. Hierzu verwenden wir die Bibliothek *db2jcc*, welche bereits mit RDz mitgeliefert wird und einen Typ 4 Treiber für JDBC bietet, das heißt, dass alle Funktionen des Treibers in Java umgesetzt sind und kein externer Programmcode verwendet werden muss.

Legen Sie ein neues dynamisches Webprojekt in RDz an, mit den gleichen Kompatibilitätseinstellungen, die bisher verwendet wurden. Importieren Sie die Treiber-Bibliothek in das Projekt und konfigurieren den Buildpfad entsprechend. Öffnen Sie hierzu das Kontextmenü des Projekts und wählen Sie *Buildpfad → Buildpfad konfigurieren...* und fügen Sie im Reiter *Bibliotheken* die Bibliotheken *db2jcc.jar* und *db2jcc_license_cisuz.jar*[20] mit der Funktion *Externe JARs hinzufügen...* ein. Danach wählen Sie die Option *Java EE-Modulabhängigkeiten* aus den Projekteigenschaften – bestätigen Sie gegebenenfalls die Speicherung des Buildpfads – und haken unter dem Reiter *Webbibliotheken* die soeben hinzugefügten Dateien an.

Wir werden insgesamt vier Klassen benötigen: eine einfache Bean-Klasse, die einen Datensatz in der Datenbanktabelle repräsentiert bzw. die Ergebnisse eines einzelnen Spiels, weiterhin brauchen wir eine Klasse, die die Geschäftsfunktionen, das heißt unsere Datenbankzugriffe durchführt und deren Methoden später über den Web Service aufgerufen werden und zuletzt noch zwei Exceptions.

Der Rahmen der Geschäftsklasse sieht folgendermaßen aus:

```
package org.example.ws.skat;

...

public class Skat {

	//mainframe host name/ip address
	private static final String host = "";
	//host port (USS command: onetstat, ISPF command: netstat)
	private static final String port = "";
	//the server location identifier (zOS specific)
	private static final String location = "";
	private static final String user = "";
	private static final String pass = "";
	private static final String db = "";
```

20 Zu finden im RDz Installationsverzeichnis unter *shared/plugins/com.ibm.datatools.db2_2.0.103.v20090522_1813/driver*

```
        private static Connection getConnection() throws ServiceException {
              try {
                    Class.forName("com.ibm.db2.jcc.DB2Driver");
                    return DriverManager.getConnection("jdbc:db2://" + host +
":" + port + "/" + location, user, pass);
              } catch (SQLException e) {
                    throw new ServiceException(e);
              } catch (ClassNotFoundException e1) {
                    throw new ServiceException(e1);
              }
        }

        public Spiel[] getSpiele() throws ServiceException { ... }

        public void addSpiel(Spiel spiel) throws ServiceException { ... }

        public int getPunkte(String spieler) throws NoSuchSpielerException,
                    ServiceException { ... }

        public void clearErgebnisse() throws ServiceException { ... }

  }
```

Sie finden alle Klassen zum Import in Ihr Projekt auf der beiliegenden DVD. Tragen Sie die Verbindungsinformationen in die dafür vorgesehenen Konstanten der Geschäftsklasse ein.

3.5.4. Test der Anwendung und weiterführende Aufgabenstellungen

Zum Abschluss testen Sie das Projekt, indem Sie das Projekt als WAR-Datei exportieren und auf dem lokalen Application Server installieren. Vergessen Sie nicht vorher den Kontextpfad in der *web.xml* anzupassen. Führen Sie abschließend den *Axis AdminClient* mit dem entsprechenden Deployment-Deskriptor aus um den Web Service zu aktivieren. Nun können Sie mit dem *Web Service Explorer* die Anwendung testen.

Die Folgenden Aufgabenstellungen können Sie lösen, um sich weiter mit der Web Service-Entwicklung vertraut zu machen:

Treffen Sie im `types`-Element des WSDL-Dokuments entsprechende Definitionen, um nur Eingaben mit den gleichen Typbeschränkungen zuzulassen wie das Datenbankschema vorschreibt.

Fügen Sie eine Operation hinzu, die den Sieger ermittelt, das heißt den Spieler mit den meisten Punkten.

Erstellen Sie in einem separaten Projekt einen Endbenutzer-Klienten für den Web Service in Form einer Command Line Anwendung, einer Swing-Oberfläche oder eines Webfrontends.

4. Zusammenfassung

In dieser Arbeit wurden die Konzepte und Anwendungen von Web Services erläutert und beispielhaft umgesetzt. Dabei wurden anhand der WSDL 2.0 Spezifikation die einzelnen Komponenten vorgestellt und Konfigurationsmöglichkeiten erklärt.

Weiterhin entstanden drei Tutorials, welche aufbauend auf *Rational Developer for System z* (RDz) die Web Service-Entwicklung in verschiedenen Szenarien darstellen.

Das Bottom-Up-Tutorial zeigt wie man aus vorhandenen Softwareressourcen Web Service Definitionen erstellt und diese im *Web Sphere Application Server Community Edition* (WASCE) installiert und publiziert.

Parallel dazu erläutert das Top-Down Beispiel den umgekehrten Entwicklungsprozess um aus bereits definierten Zugriffsschnittstellen Rahmenklassen für die Geschäftsfunktionen generieren zu lassen.

Im dritten und letzten Tutorial wird die Integration einer z/OS DB2 Installation mit einem *JavaEE* Middle-Tier und einem generischen Frontend vorgestellt. Dabei offeriert das Middle-Tier seine Geschäftsfunktionen für die Frontends in Form eines Web Services und greift zur Umsetzung seiner Operationen mittels eines DB2 JDBC Typ 4 Treiber auf die Datenbank zu.

Mit Hilfe von Web Services wird der Datenaustausch in verteilten Softwaresystemen standardisiert und unabhängig von der verwendeten Laufzeit – sprich: Computersystem, Programmiersprache und Betriebssystem – gemacht. In Hinsicht auf WSDL 2.0 werden die Basiskonzepte gezielt erweitert und Web Services von SOAP emanzipiert, damit diese auch mit anderen Protokollen der Anwendungsschicht effektiv eingesetzt werden können.

4.1. Ausblick

Es gingen eine große Anzahl an Spezifikationen hervor, welche aufbauend auf SOAP und WSDL, viele unterschiedliche Anwendungsgebiete abdecken und interoperabel eingesetzt werden können. Zum Beispiel bietet die *WS-Business Process Execution Language* (WS-BPEL) Möglichkeiten, nicht nur die Kommunikation zwischen Geschäftsprozessen zu modellieren, sondern auch diese selbst. Zusammengefasst werden diese Arbeiten unter dem Pseudonym *WS-**[21].

Das neuere *Axis2* Framework, welches nun auch WSDL 2.0 unterstützt, stellt einen weiteren wichtigen Fortschritt der Web Service-Anwendungen dar. Allerdings wurde seit Version 1.5 die Unterstützung für das JDK 1.4 fallen gelassen, wodurch viele aktuelle Installationen der *WebSphere* Server für z/OS Schwierigkeiten haben könnten.

21 *WS-Star* ausgesprochen

A. Literaturverzeichnis

Web Service Tutorials:

Web Services for CICS
http://jedi.informatik.uni-leipzig.de/de/tutor_wdz70/tutorial8_wsdl.pdf

Develop a Web service without an IDE, Part 1: Focus on the Server: Create a Web service provider on the command line
http://www.ibm.com/developerworks/library/ws-noide1/index.html

Develop a Web service without an IDE, Part 2: Creating a "Hello World!" Web service client on the command line
http://www.ibm.com/developerworks/library/ws-noide2/index.html

WSDL 2.0 Dokumente:

Web Services Description Language (WSDL) Version 2.0 Part 0: Primer
http://www.w3.org/TR/wsdl20-primer

Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language
http://www.w3.org/TR/wsdl20

Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts
http://www.w3.org/TR/wsdl20-adjuncts

Web Services Description Language (WSDL) Version 2.0: Additional MEPs
http://www.w3.org/TR/wsdl20-additional-meps

Discussion of Alternative Schema Languages and Type System Support in WSDL
http://dev.w3.org/cvsweb/~checkout~/2002/ws/desc/wsdl20/altschemalangs.html?content-type=text/html;%20charset=utf-8&rev=1.3

WSDL 1.1 Dokumente:

Web Services Description Language (WSDL) 1.1
http://www.w3.org/TR/wsdl

SOAP 1.2 Dokumente:

SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)
http://www.w3.org/TR/soap12-part1

SOAP Version 1.2 Part 2: Adjuncts (Second Edition)
http://www.w3.org/TR/soap12-part2

RFC 3902 - The "application/soap+xml" media type
http://tools.ietf.org/html/rfc3902

Sonstige:

Vorlesung: Extensible Markup Language (XML) von Mario Jeckle
http://www.jeckle.de/vorlesung/xml/index.html

Architecture of the World Wide Web, Volume One
http://www.w3.org/TR/webarch

RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1
http://tools.ietf.org/html/rfc2616

RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication
http://tools.ietf.org/html/rfc2617

Tutorial 4 DB2
http://jedi.informatik.uni-leipzig.de/de/tutors/tutor4.pdf

B. Anlagenverzeichnis

C. Anlagen

Abbildung 1: ibm.de – Navigation zum Download-Repository

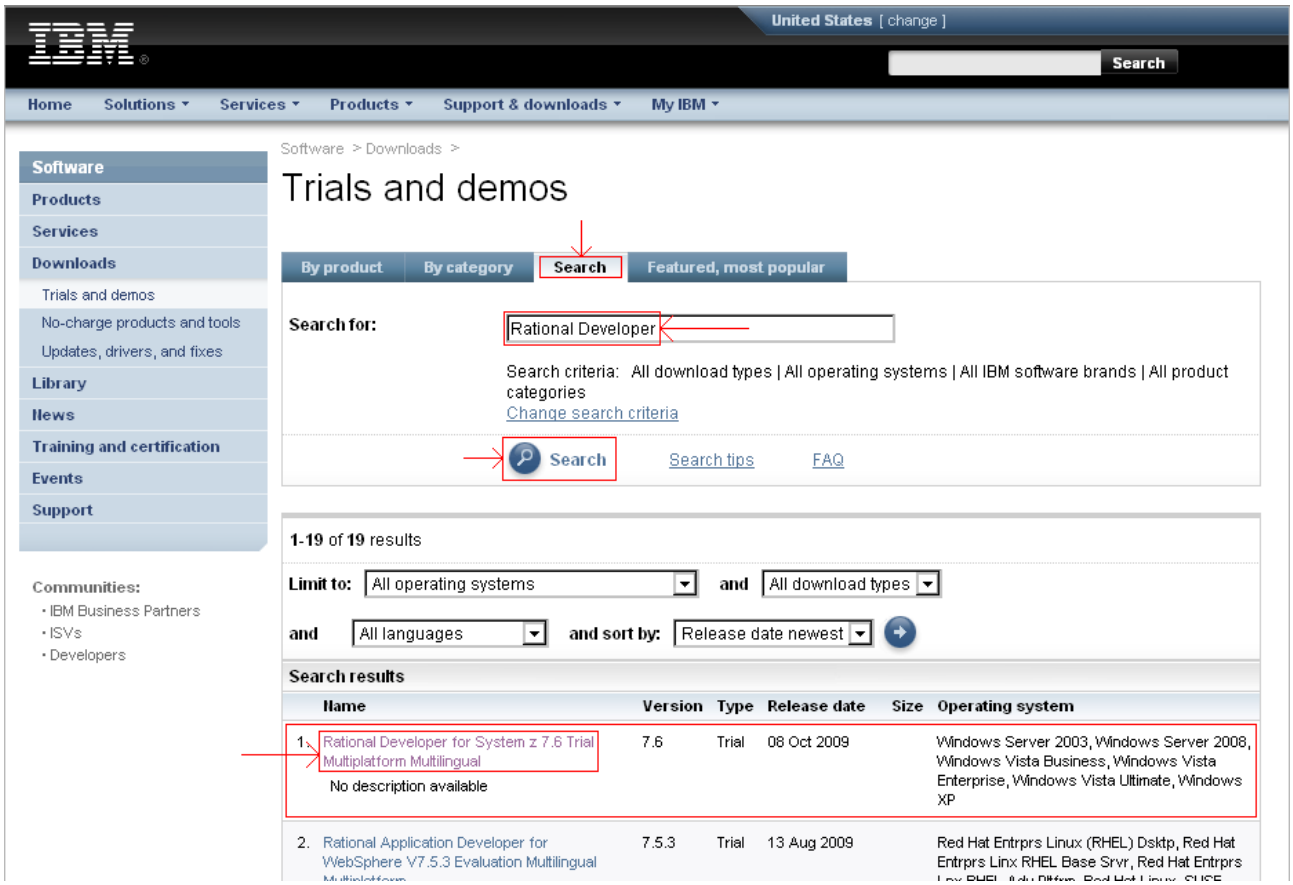

Abbildung 2: ibm.com – Suche nach RDz Trial Download

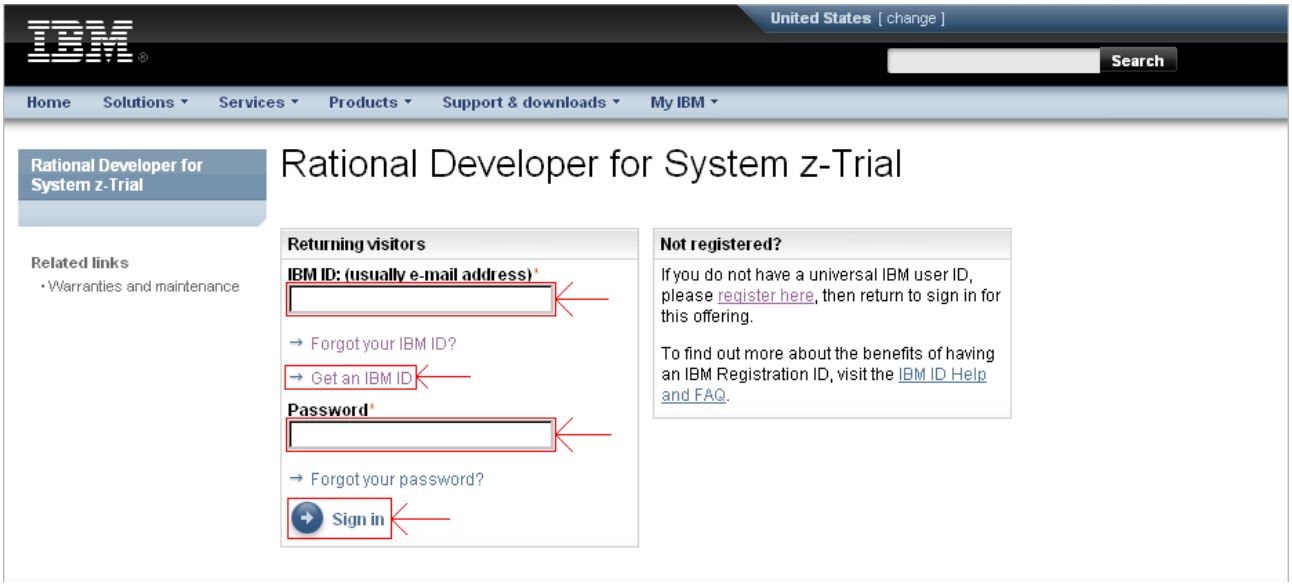

Abbildung 3: ibm.com – Login mit IBM ID

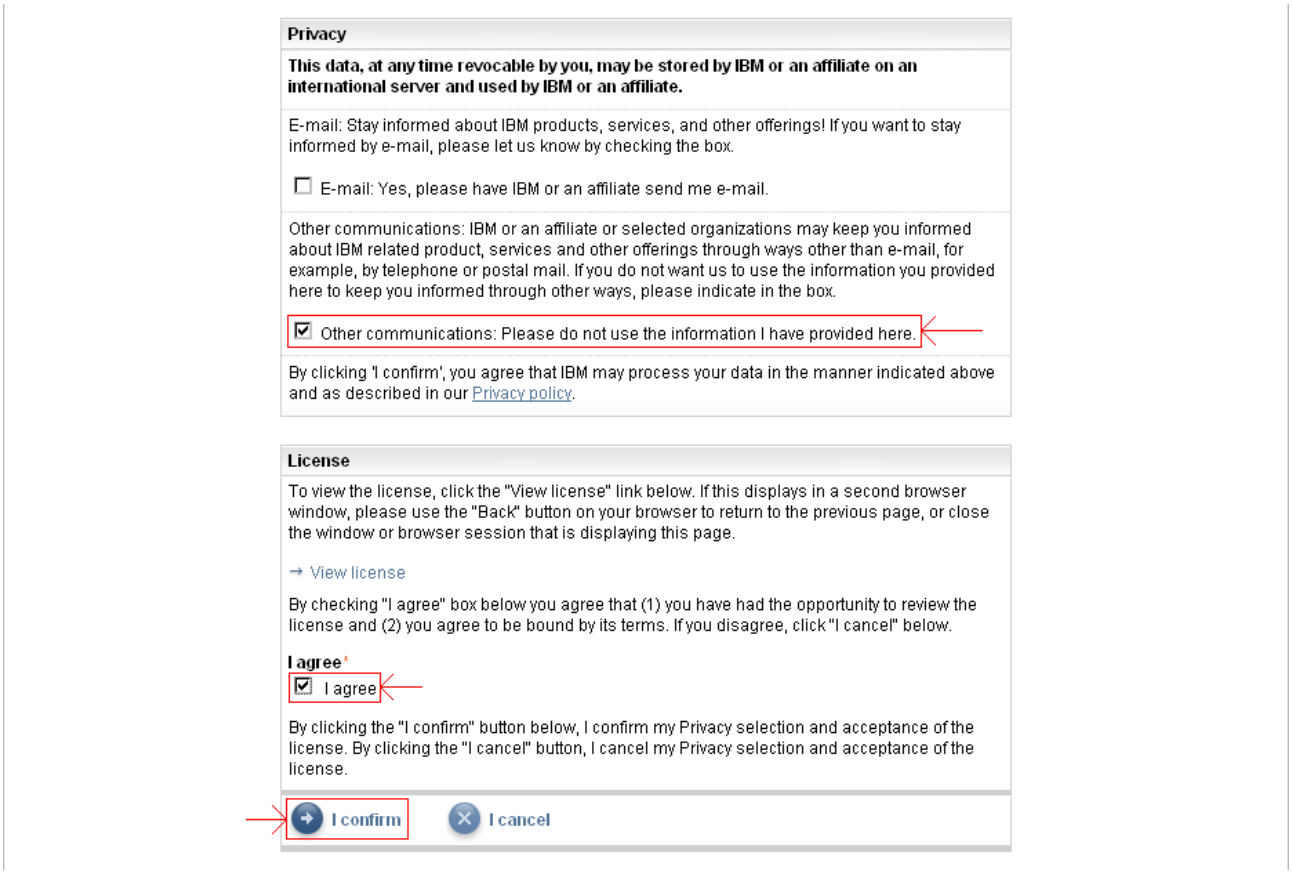

Abbildung 4: ibm.com – Fragebogen zum Download

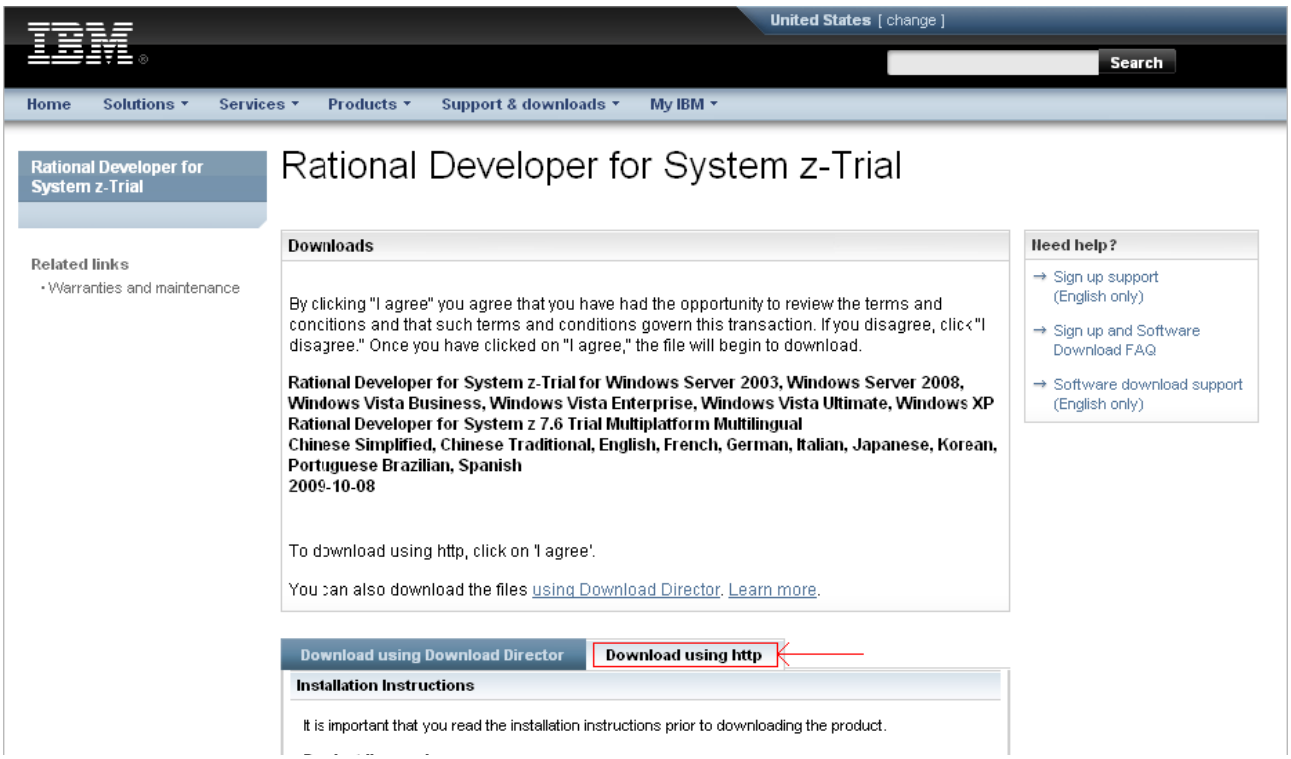

Abbildung 5: ibm.com – RDz Download Seite

Abbildung 6: ibm.com – RDz Download Seite Unterer Teil

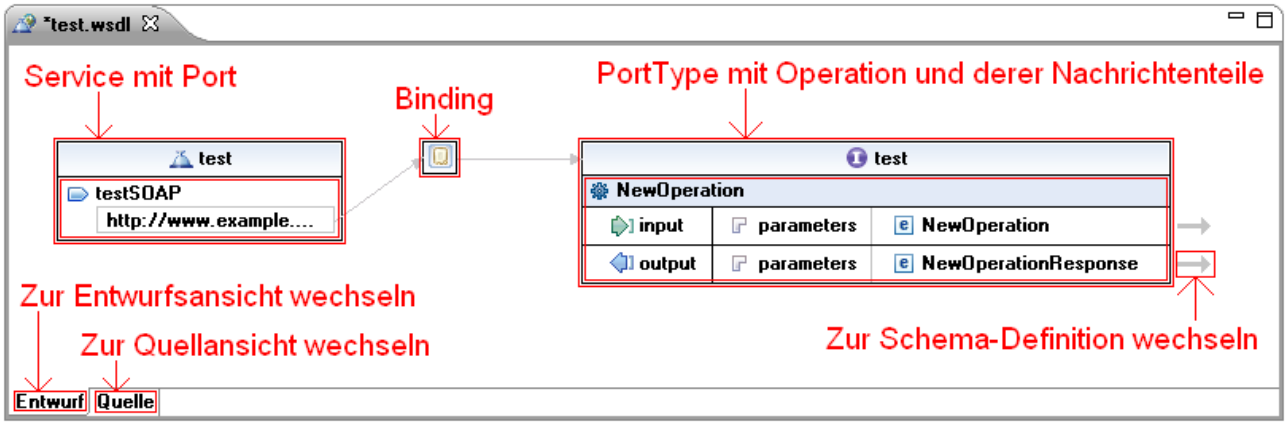

Abbildung 7: RDz – WSDL-Editor mit Standard-WSDL

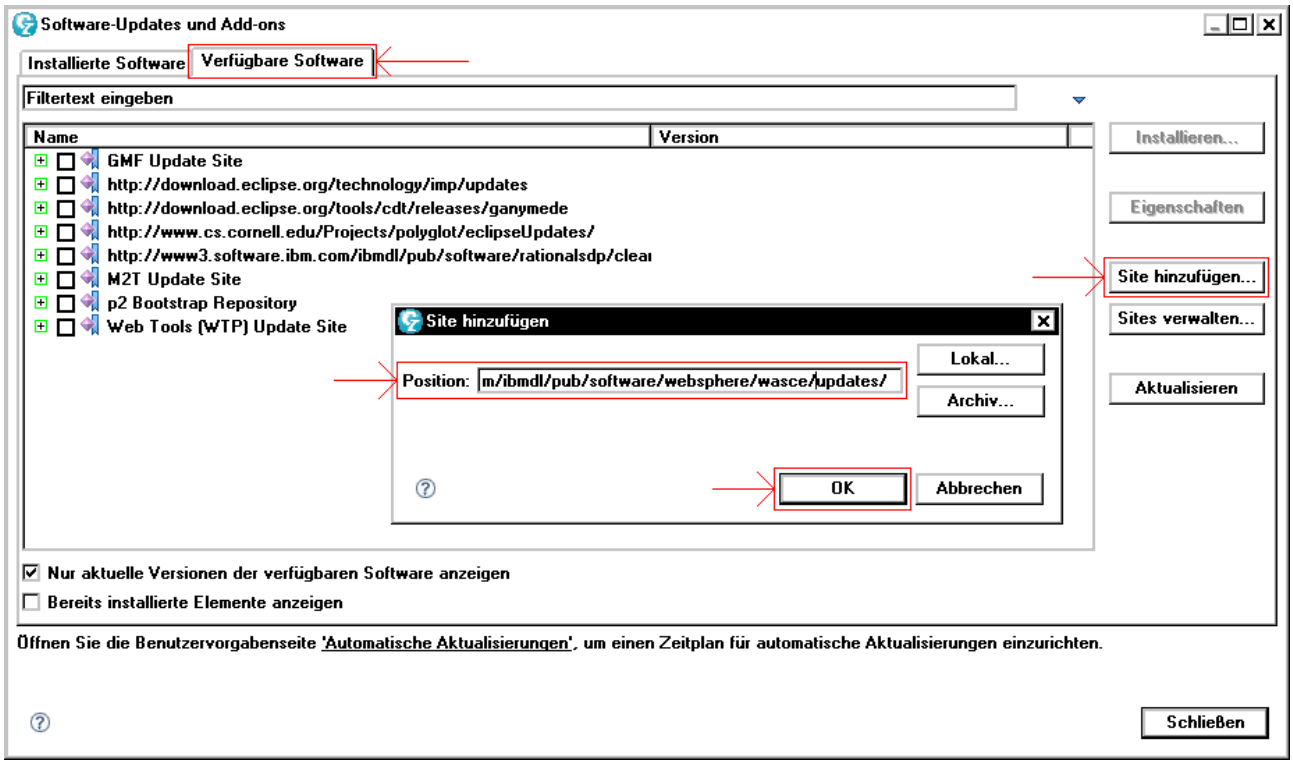

Abbildung 8: RDz – Update-Site hinzufügen

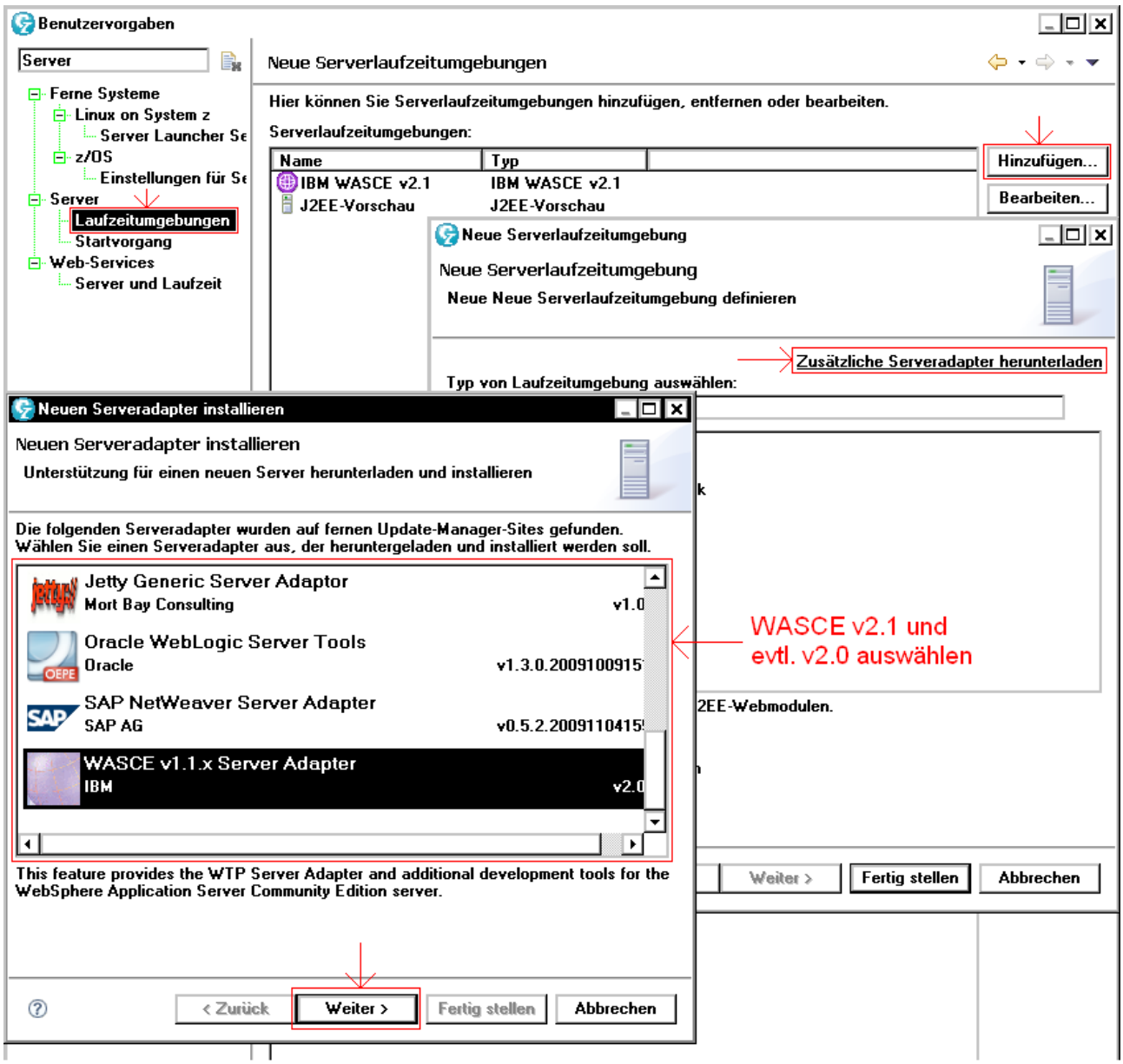

Abbildung 9: RDz – Serveradapter installieren

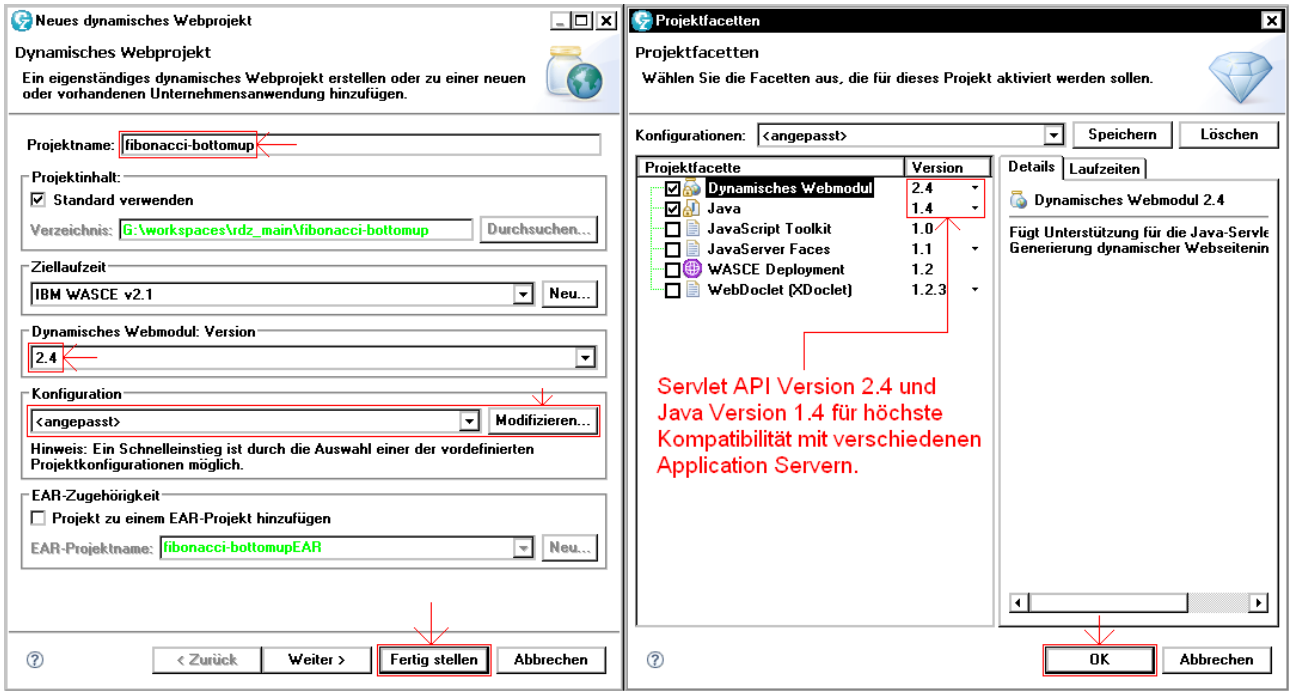

Abbildung 10: RDz – Dynamisches Webprojekt anlegen

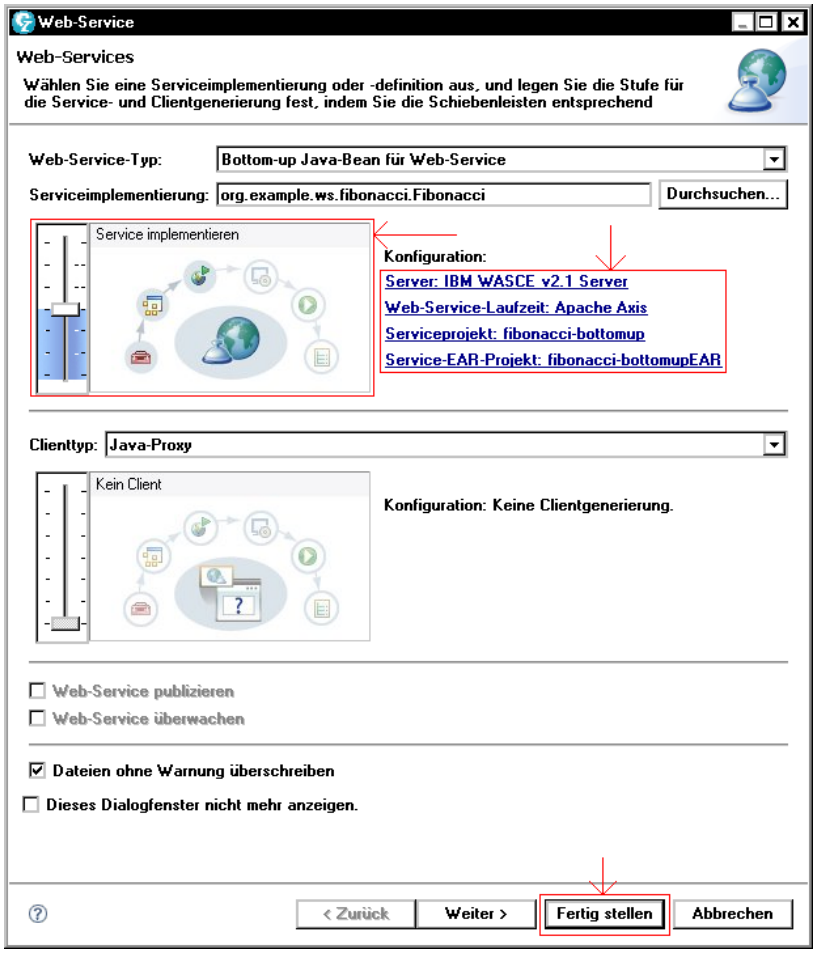

Abbildung 11: RDz – Web Service erstellen (Bottom-Up)

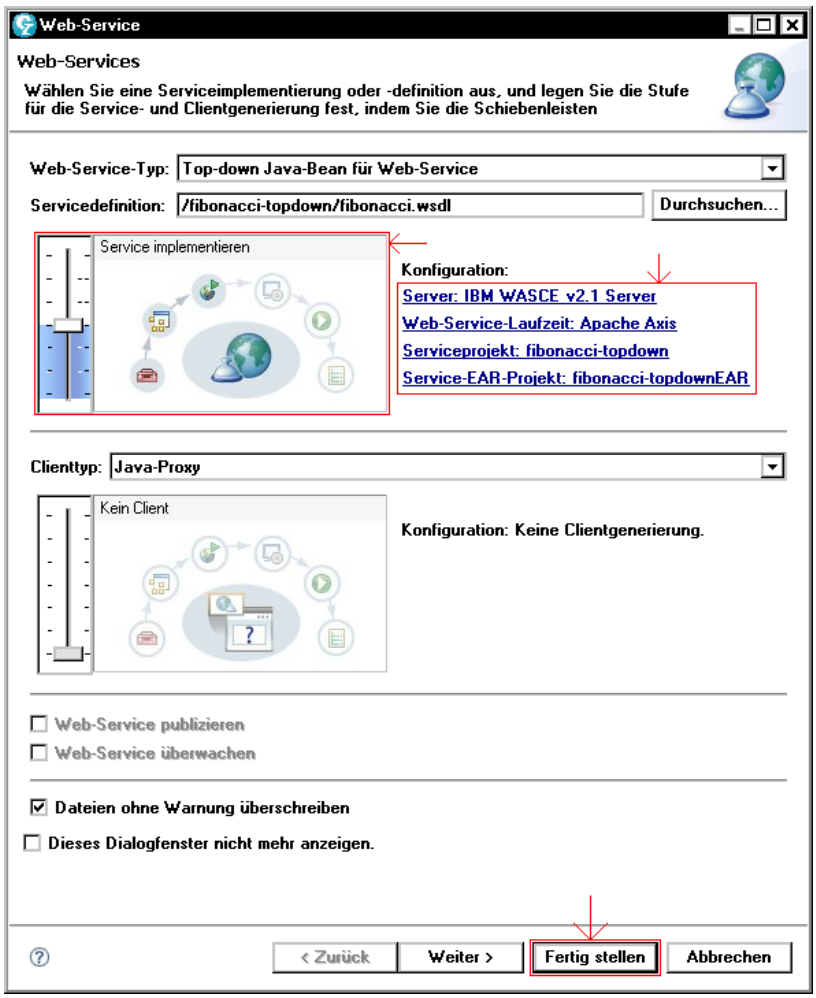

Abbildung 12: RDz – Web Service erstellen (Top-Down)

Abbildung 13: Command Line – Axis AdminClient Aufruf

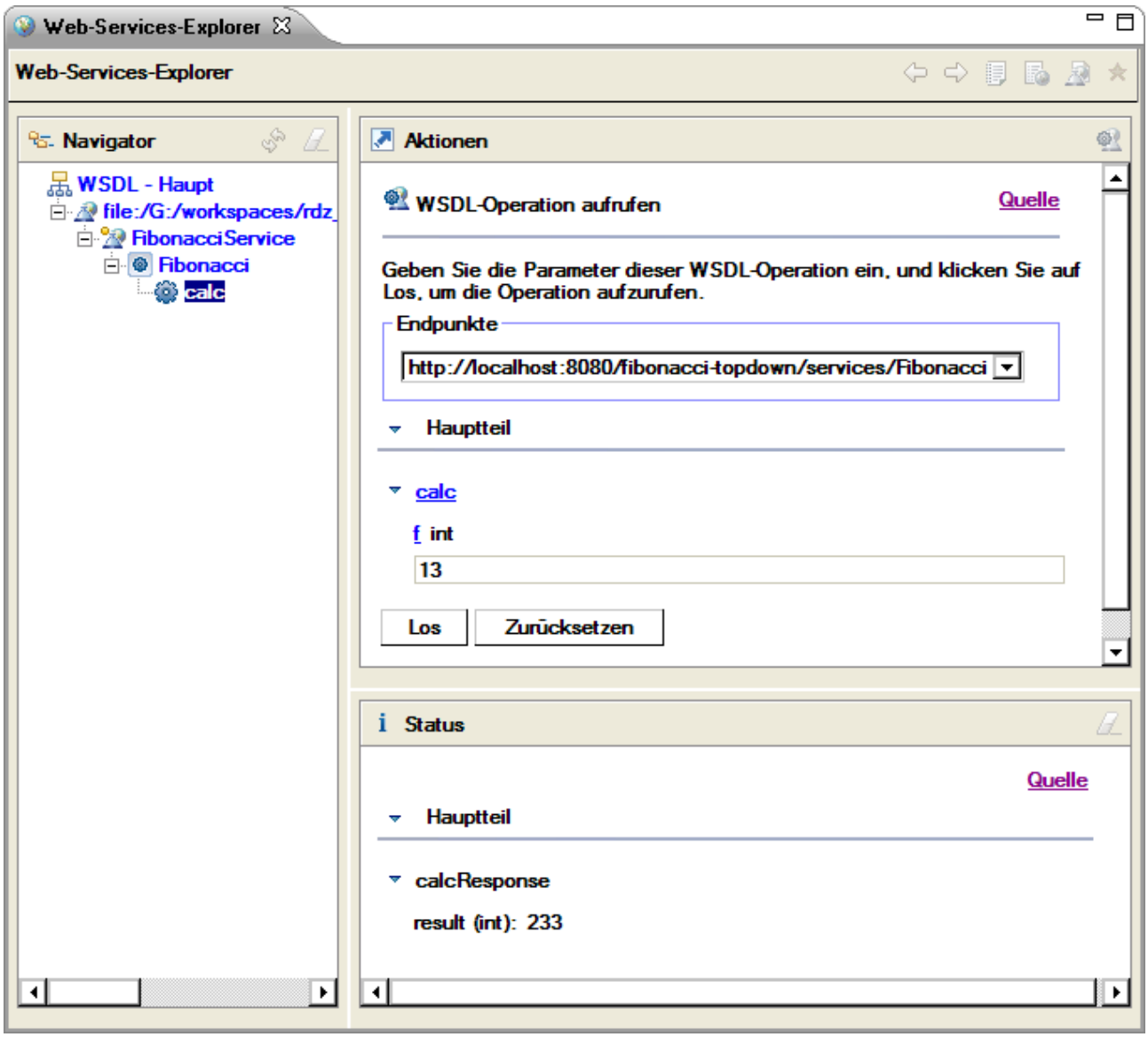

Abbildung 14: RDz – Web Service Explorer

D. Erklärung

"Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann."

Ort Datum Unterschrift