4 Werkzeuge, Spezifikationen und Arbeitsumgebung

4.1 Überblick

Eigentlich sollte es ja ganz einfach sein, mit der Entwicklung von Web Services loszulegen. Aber dennoch hat man ab und zu mit der Konfiguration der jeweiligen Entwicklungsumgebung zu kämpfen. Das hat seine Gründe vor allem in der großen Versionsvielfalt der beteiligten Software. In diesem Kapitel werden wir deshalb erst einmal eine gemeinsame Basis für die nachfolgenden Beispiele schaffen. Bei der Übertragung auf die realen Anforderungen eines Projektes können Sie sich dann von diesem Ausgangspunkt leiten lassen.

4.2 Java API for XML Web Services – JAX-WS

Die *Java API for XML Web Services*, kurz JAX-WS, beschreibt das Mapping zwischen der Sprache Java und der *Web Service Description Language* (WSDL), also – vereinfacht ausgedrückt alles, was benötigt wird, um Web Services anzusprechen und zu implementieren. Der Standard selbst wird im Rahmen des *Java Community Process* (JCP) [17] entwickelt. Die Spezifikation ist auch unter ihrer derzeitigen Nummer *JSR 224* zu finden [15].

Erst diese Spezifikation hat die verschiedensten APIs der verschiedenen Hersteller von Web-Service-Engines in der Java-Welt vereinheitlicht. Die Referenzimplementierung, kurz auch als *JAX-WS RI* bezeichnet, ist selbst Bestandteil des *Metro Web Service Stack* oder auch der *Java Standard Edition*. Zu finden ist diese unter `http://jaxws.dev.java.net` [14].

An dieser Stelle noch einige Worte zur Bedeutung des JCP. Dieser ist seit 1998 fester Bestandteil der Java-Welt. Neue Spezifikationen, deren Referenzimplementierungen und Testfälle werden hier entwickelt, getestet und vor der finalen Freigabe der öffentlichen Entwicklerwelt

zur Diskussion präsentiert. Mithilfe des JCP entwickelt sich Java Stück für Stück, unter Berücksichtigung aller interessierten Personen, weiter.

4.3 Metro Web Service Stack

Metro ist die Bezeichnung für den *Java Web Service Stack* in der JRE sowie der oben genannten Referenzimplementierung. Eine der wesentlichen Eigenschaften dieses Stacks ist, dass er über die JAX-WS RI vollständig in die *Java Standard Edition* integriert ist. Er skaliert von sehr einfachen Diensten bis hin zu sicheren, transaktionalen Anforderungen in nahezu beliebigen Umgebungen (also von der *Standard Edition* bis hin zur *Enterprise-Welt*).

Die Bestandteile von Metro (`http://metro.dev.java.net` [20]) lassen sich wie folgt identifizieren:

- *JAX-WS RI*
  Die Referenzimplementierung der JAX-WS-Spezifikation.
- *Java Architecture for XML Binding (JAXB) RI*
  JAXB ist eine Spezifikation, die beschreibt, wie Java-Klassen an ein Schema gebunden werden. Das bedeutet genau genommen, wie Java-Objekte nach XML und zurück serialisiert werden. Die JAXB-Referenzimplementierung [16] wird parallel zum *Web Service Stack* entwickelt und ist seit der *Java Standard Edition 6* enthalten. JAX-WS nutzt JAXB als einzigen Weg, um Java-Objekte nach XML zu serialisieren (zum Beispiel für SOAP-Nachrichten oder für Parameter und auch Rückgabewerte einer Dienstoperation).
- *Web Service Interoperability Technologies (WSIT)*
  Alle Techniken zur *Web Service Interoperability Technologies* (kurz WSIT [65]) werden in einem speziellen Projekt gebündelt. Besonderes Augenmerk wird auf die Kompatibilität zwischen der Java-Plattform [12] und der *Windows Communication Foundation* (WCF) [39] gelegt. Die besondere Bedeutung ergibt sich hierbei daraus, dass die Java-Plattform den kompletten Unterbau für jede Java-basierte Anwendung darstellt und im eigentlichen Sinne das gesamte Java-Ökosystem beschreibt. WCF wiederum beschreibt in der Microsoft .NET-Welt eine vereinheitlichte Kommunikationsplattform (früher auch unter dem Codenamen *Indigo* bekannt). Diese Kommunikationsplattform fasst nun alle bisher vorhanden Standards (wie DCOM und Web Services) unter einem Dach zusammen. Besonderes Augenmerk wird hierbei auf die Unterstützung serviceorientierter Architekturen gelegt. Mit der Herstellung

der Kompatibilität zwischen Java-Plattform und WCF auf Dienstebene ist die Welt für den Entwickler etwas einfacher geworden.

- Web-Service-Standards
  Metro enthält eine Reihe von Implementierungen diverser Web-Service-Standards. Einige sind in den späteren Kapiteln des Buches zu finden. Die genaue Liste und die Versionsnummern finden Sie auf der Metro-Webseite [20].
- *Ant*-Tasks, Metro-Beispiele und Werkzeuge
  Zusätzliche Ant-Tasks und Metro-Beispiele machen das Leben des Entwicklers leichter. Auch finden wir in der aktuellen Distribution die Kommandozeilenwerkzeuge, die auch im Bin-Verzeichnis des jeweiligen JDK vorhanden sind.

Aufgrund der Zusammensetzung des *Metro Web Service Stack* ergeben sich zum aktuellen Zeitpunkt der Veröffentlichung die folgenden API-Abhängigkeiten. Wie wir noch sehen werden, spielen die Versionsnummern eine zentrale Rolle:

Tab. 4-1 *API-Bestandteile von Metro*

Metro	JAXWS-RI	JAXB RI	WSIT
1.1	2.1.3	2.1.6	1.1
1.2	2.1.3	2.1.6	1.2
1.3 und 1.4	2.1.4	2.1.7	(nicht mehr getrennt versioniert)
1.5	2.1.7	2.1.11	(nicht mehr getrennt versioniert)
Vorschau *2.0EA*	*2.2EA*	*2.2EA*	*(nicht mehr getrennt versioniert)*

Die Version 1.5 ist seit dem 17. April 2009 verfügbar und war zu dem Zeitpunkt, als dieses Buch geschrieben wurde, noch nicht Bestandteil des JDK, JRE oder des verwendeten OpenESB.

Welcher Teil ist wo enthalten?

Betrachtet man die einzelnen Bestandteile von Metro, fällt auf, dass je nach Werkzeug nur ein spezieller Teil des Gesamtpaketes enthalten ist. Um die Sache nicht noch komplizierter zu machen, finden Sie in Abbildung 4-1 einen ungefähren Überblick, was wo enthalten ist. »Ungefähr« deshalb, weil der implementierte Web-Service-Standard je nach JAX-WS-Implementierung auch im JDK liegen kann (zum Beispiel WS-Addressing).

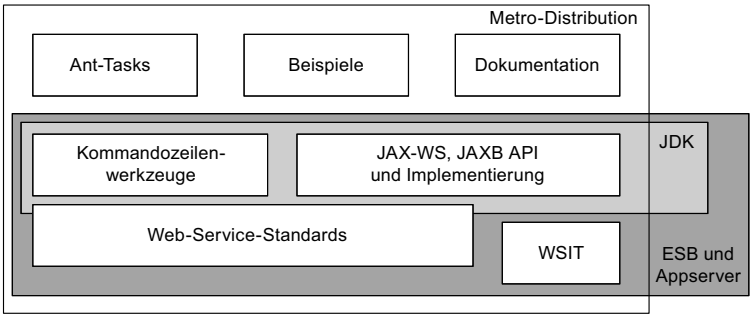

Abb. 4-1 *Welcher Bestandteil von Metro ist wo enthalten?*

4.4 Java Development Kit und Web Services

4.4.1 Überblick

Das *Java Development Kit* (JDK) oder auch das *Java Runtime Environment* (JRE) bilden die Grundlage der Entwicklung unter Java. Das ist an sich ja nichts Neues. Jedoch ist ab der Version 6 der *Java Standard Edition* (Java SE 6) eine JAX-WS-Implementierung standardmäßig im Paket enthalten. Genau genommen handelt es sich dabei sogar um die Referenzimplementierung selbst. Für die vorhergehende Java-Version, J2SE 5, lässt sich die JAX-WS-Implementierung einfach nachrüsten.

JAX-WS-Implementierungen

Etwas komplizierter wird es, wenn es um die Version der JAX-WS-Implementierung geht. Durch die ungleiche Reihenfolge der Releases zwischen der *Java Standard Edition* (oder genauer gesagt deren Update-Versionen) und dem *Web Service Stack* müssen wir schon genauer auf die Versionsnummer achten. Zum aktuellen Zeitpunkt (der Veröffentlichung) ergaben sich folgende Abhängigkeiten:

Tab. 4-2 *Web-Service-Unterstützung im JRE*

Java SE	JAXWS-RI	JAXB RI
6...6u03	2.0	2.0
6u04...6u13	2.1.1	2.1.3
6u14...6u16	2.1.6	2.1.10

JAX-WS-Versionsnummer

Die Versionsnummer der aktuell enthaltenen JAX-WS-Version lässt sich auch leicht mit dem Kommando `wsimport -version` herausfinden. An dieser Stelle sollten Sie sich die Zielumgebung nochmals anschauen. Wie war das gleich noch einmal – erst JAX-WS RI 2.1 und höher bringen die Basis für die *Web Service Interoperability Technoloy* mit ([64]). Das bedeutet aber auch: Selbst innerhalb einer Java-Hauptversion kann

sich die Web-Service-Implementierung stark verändern. Ein besonderes Augenmerk ist demnach auf das Deployment in ein vorhandenes JDK/JRE bzw. dessen Update zu legen.

4.4.2 Java Endorsed Standards Override Mechanism

Hinter dem Begriff *Java Endorsed Standards Override Mechanism* versteckt sich eine Möglichkeit, die Java-Plattform auf den neusten Stand zu bekommen. Wie man an den Versionsnummern der beteiligten Web-Service-Technologien (JAX-WS, WSIT, ...) leicht sieht, werden diese schneller an neue Gegebenheiten angepasst, als die Java-Plattform selbst. Um die Java-Plattform auf dem aktuellsten Stand zu halten, kann man die neueren Pakete in das *Endorsed-Standards-Override*-Verzeichnis kopieren. Das Verzeichnis selbst ist wie folgt zu ermitteln:

- Microsft Windows: `<java-home>\lib\endorsed`
- Solaris und Linux: `<java-home>/lib/endorsed`

Die entsprechenden Bibliotheken der JAX-WS-Implementierung oder des *Metro Web Service Stack* sind dort abzulegen und überschreiben zur Laufzeit (mithilfe der Hierarchie der *Classloader*) die in der *Java Standard Edition* enthaltenen Bibliotheken. Leider lässt dieser Mechanismus nicht zu, dass verschiedene Versionen gleichzeitig installiert sind. Wenn Sie also schon eine JAX-WS-Implementierung dort vorfinden, dann müssen wir eine andere Taktik zur Konfliktbeseitigung wählen (siehe hierzu Abschnitt 4.5). Ein im *Endorsed-Standards-Override*-Verzeichnis abgelegtes Java-Archiv hat automatisch Vorrang vor allen anderen Archiven, inklusive der im *Java Runtime Environment* installierten.

4.4.3 JAX-WS RI unter J2SE 5.0

Java 5

Manchmal ist es nicht möglich, die neuste Version von Java zu verwenden. Zumindest für die *Java 2 Platform Standard Edition 5.0* ist es relativ unkompliziert, den Web Service Stack oder auch nur die JAX-WS RI nachzurüsten. Im Prinzip stehen dazu zwei Wege offen:

1. Der *Java Endorsed Standards Override Mechanism* wird benutzt (siehe Abschnitt 4.4.2).
2. Die entsprechenden Bibliotheken werden zum Bestandteil der Anwendung gemacht.

4.5 Dem API-Versionsdilemma entkommen – praktische Tipps

Spätestens nach der Lektüre der verschiedenen Web-Service-APIs und deren Implementierungen in J2SE oder auch Standalone-Systemen bleiben Fragen nach dem Versionsdilemma übrig. Grundsätzlich bieten sich einige Auswege an, der »Goldene Weg« wird jedoch wohl erst mit dem nächsten Java, respektive der angedachten Versionierung von Modulen, möglich sein. Doch zurück zu unserem Problem:

Als Erstes muss man sich fragen, für wen oder was entwickle ich den Service? Ist die Laufzeitumgebung vorgegeben, kann man sich voll darauf konzentrieren und mögliche Neuerungen (leider) nicht nutzen. Das ist immer dann der Fall, wenn wir einen Enterprise Service Bus, einen Anwendungsserver oder auch eine Web-Service-Engine schon vorfinden. Hier hilft dann meistens nur noch das Upgrade auf eine neuere Version. Zuvor sollten wir uns aber vergewissern, ob wir das letzte neue Feature wirklich brauchen.

Schwieriger wird es, wenn die Entscheidung bei uns selbst liegt. So ist es natürlich möglich, über den *Endorsed Standards Override Mechanism* das API bzw. die Implementierung auszutauschen. Jedoch sollten wir dabei bewusst prüfen, ob fremde Dienste oder Dienstenutzer in so einem Fall noch richtig funktionieren. Notfalls hilft dann doch ein privates *Java Runtime Environment* für unsere Anwendung. Das ist nicht schön, aber es funktioniert sicher.

Gerade im Fall der Anwendungsserver oder auch der Web-Container (mit Service-Engine) sollte man den Weg der Java-Klassen durch die Classloader-Hierarchie von Java genau kennen. Probleme können sehr schnell entstehen, wenn ältere oder neuere Implementierungen eher sichtbar sind. Hier hilft dann ein Blick in die jeweilige Dokumentation: In den meisten Fällen kann man die Reihenfolge über spezielle Umgebungseinstellungen verändern.

4.6 OpenESB – GlassFish Enterprise Service Bus

OpenESB

Ab einer gewissen Funktionalität eines Dienstes ist eine erweiterte Infrastruktur notwendig. Das beinhaltet zum Beispiel Dinge wie Sicherheit, Transaktionen oder auch die garantierte Übertragung von Nachrichten. Aus diesem Grund nutzen wir ab Kapitel 9 den *GlassFish Enterprise*

Service Bus als Grundlage unserer Beispiele. Der Enterprise Service Bus selbst ist frei verfügbar und basiert auf dem *GlassFish Enterprise Server* – über https://open-esb.dev.java.net kann dieser bezogen werden.

4.7 Apache-Ant-Build-Werkzeug und Ant Tasks für Web Services

Für einige Beispiele wird Apache Ant als Build-Werkzeug benötigt. Das Werkzeug selbst ist frei verfügbar (`http://ant.apache.org`).

Ant Tasks

In einigen Beispielen des Buches sind spezielle Ant Tasks erforderlich. Bitte seien Sie nicht verwundert, wenn jetzt noch nicht alle Details bis ins Kleinste erklärt werden. Für uns reicht es an dieser Stelle, die Grundlagen für die späteren Kapitel zu schaffen. Voraussetzung ist in jedem Fall die vorherige Installation des Metro Web Service Stack. In den folgenden Kapiteln werden wir dann nur noch die Nutzung des Tasks vorstellen (nicht mehr die Definition selbst). Die Ant Tasks sind nicht im Java Development Kit enthalten, dort finden Sie lediglich die entsprechenden Werkzeuge für die Konsole.

Ant Tasks und JDK

Um die Java-Artefakte für Dienst oder Dienstenutzer aus einer *Web Service Description Language* (Contract-First-Ansatz, siehe Kapitel 7) zu erzeugen, benötigen wir die folgende `wsimport`-Task-Definition:

Listing 4.1
Ant Task »wsimport«

```
<project name="ATM Service" default="generate"
  basedir=".">

  <!-- Pfad muss angepasst werden -->
  <path id="metro.classpath">
    <fileset dir="<Ihr-Verzeichnis> .../metro/lib">
      <include name="**/*.jar" />
    </fileset>
  </path>

  <taskdef name="wsimport"
          classname="com.sun.tools.ws.ant.WsImport">
    <classpath refid="metro.classpath" />
  </taskdef>
</project>
```

Um die Java-Artefakte für Dienst oder Dienstenutzer aus einer Java-Klasse (Code-First-Ansatz; Java-Binary-Klassen als Ausgangspunkt, siehe Kapitel 7) zu erzeugen, benötigen wir die folgende `wsgen`-Task-Definition:

Listing 4.2 *Ant Task »wsgen«*

```
<project name="ATM Service" default="generate"
  basedir=".">

  <!-- Pfad muss angepasst werden -->
  <path id="metro.classpath">
    <fileset dir="<Ihr-Verzeichnis> .../metro/lib">
      <include name="**/*.jar" />
    </fileset>
  </path>

  <taskdef name="wsgen"
          classname="com.sun.tools.ws.ant.WsGen">
    <classpath refid="metro.classpath" />
  </taskdef>
</project>
```

Um die Java-Artefakte für Dienst oder Dienstenutzer aus einer Java-Klasse (Code-First-Ansatz; Java-Klassen als Ausgangspunkt, siehe Kapitel 7) zu erzeugen, benötigen wir die folgende apt-Task-Definition:

Listing 4.3 *Ant Task »apt«*

```
<project name="ATM Service" default="generate"
  basedir=".">

  <!-- Pfad muss angepasst werden -->
  <property name="METRO_HOME"
    value="<Ihr-Verzeichnis> .../metro" />
  <property name="JAVA_HOME"
    value="C:/development/jdk1.6.0_11" />

  <path id="metro.classpath">
    <pathelement location="${JAVA_HOME}/lib/tools.jar" />
    <fileset dir="${METRO_HOME}/lib">
      <include name="*.jar" />
    </fileset>
  </path>

  <taskdef name="jaxws-apt"
          classname="com.sun.tools.ws.ant.Apt">
    <classpath refid="metro.classpath" />
  </taskdef>
</project>
```

4.8 Die Eclipse-Entwicklungsumgebung

Eclipse IDE

In diesem Buch nutzen wir die Java-Entwicklungsumgebung *Eclipse IDE for Java EE Developers 3.4* (Ausgabe *Ganymed*) für die Beispiele. Natürlich ist es auch möglich, jede beliebige andere Entwicklungsumgebung einzusetzen (inklusive eines Texteditors und des JDK). Die Projekte sind jedoch so gehalten, dass diese direkt in Eclipse importiert und genutzt werden können. Die Java-Entwicklungsumgebung Eclipse selbst ist frei verfügbar (`http://www.eclipse.org`).

Für einige Beispiele im Buch ist es notwendig, ein Ant-Skript auszuführen. Sie können dieses Skript direkt aus der Eclipse-Entwicklungsumgebung mithilfe der rechten Maustaste starten (siehe Abbildung 4-2). Bitte beachten Sie jedoch, dass eventuelle Anpassungen im Pfad der Werkzeuge (im Kopfteil des Ant-Skripts) zu erfolgen haben.

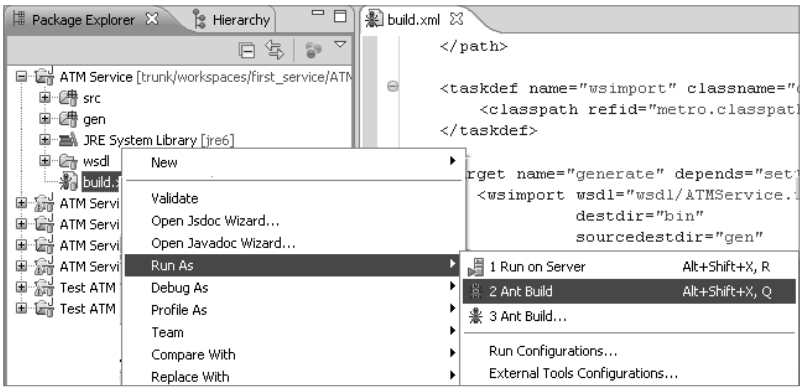

Abb. 4-2
Ausführung eines Ant-Skripts in Eclipse

4.9 Microsoft Visual Studio C# Express

Visual Studio C#

Für die weiterführenden Beispiele im Kapitel zur Interoperabilität (siehe Kapitel 10) ist die Entwicklungsumgebung *Microsoft Visual C# 2008 Express Edition* notwendig. Diese Version lässt sich kostenfrei von `http://www.microsoft.com/express` beziehen. Nach der Installation ist alles Notwendige getan. Weiterführende Informationen finden Sie später in Kapitel 10.

4.10 Zusammenfassung

In diesem Kapitel haben wir erfahren:

- welche Werkzeuge für das Buch und die Beispiele benötigt werden
- Hintergründe zu den Werkzeugen selbst
- welche Versionsprobleme beim Einsatz der Werkzeuge zu lösen sind und wie diese gelöst werden können
- wie bestimmte Artefakte für Dienst und Dienstenutzer mittels dem Ant-Werkzeug zu generieren sind

Sie sollten jetzt in der Lage sein, alle Beispiele des Buches nachzuvollziehen. Wenn der in den Kapiteln enthaltene Quelltext einem Beispiel direkt entnommen wurde, finden Sie einen Hinweis wie »Beispiel first_service/ATM Service« am Rande des Textblockes. So können Sie mit der Entwicklungsumgebung Eclipse das Projekt direkt anwählen und bearbeiten – natürlich nur, wenn Sie dies auch möchten.

Beispiele und Quelltexte unter: http://www.soa-academy.com

4.11 Checkliste

Für die Arbeit mit den ersten Beispielen des Buches (bis Kapitel 9) müssen die folgenden Werkzeuge installiert und funktionsfähig sein:

- *Java Development Kit* – Version 6 (mindestens Update 7)
- die Eclipse-Entwicklungsumgebung – Version 3.4 (Ausgabe Ganymed), Java Enterprise Edition
- Metro Web Service Stack – Version 1.4 bzw. 1.5

Ab Kapitel 9 muss zusätzlich installiert und konfiguriert worden sein:

- OpenESB – Version v2

Sollten Sie den Build-Prozess der zusätzlichen Quellcode-Artefakte nicht mit der integrierten Ant-Version aus Eclipse durchführen wollen, wird folgende Software zusätzlich benötigt:

- Apache Ant – Version 1.7

Wenn Sie die weiterführenden Beispiele zur Interoperabilität ausprobieren möchten, ist zu installieren:

- Microsoft Visual C# Express Edition – Version 2008