In diesem Kapitel
Um was geht's?
- Kreieren von Web Applikationen
- Installieren von Web Services
- Starten von Web Services
- Installation und Test
- Deployen von Web Services
- Testen
- Web Archive - war

Apache AXIS

1.1. Um was geht's?

Sie wollen Apache SOAP kennen lernen, ohne über alle möglichen Probleme zu stolpern. Einleitung.

Sie wollen sich mit Apache AXIS (*AXIS = Apache Extensible Interaction System*), einem SOAP Framework, vertraut machen. Axis ist eine Neuimplementierung von SOAP (*SOAP=Simple Object Access Protocol*).

Sie kennen Java und können auch verteilte Java Applikationen zum Laufen bringen, beispielsweise kennen Sie CLASSPATH, JAVA_HOME und ähnliche Java spezifischen Settings. Daneben verfügen Sie auch über Grundkenntnisse über Servlets und beispielsweise Jakarta Tomcat. Sie finden eine Starthilfe zu Tomcat (mit Servlet Beispielen) auf unserer Web Site.

Bemerkungen
1. Damit Axis problemlos funktioniert, benötigen Sie einen XML Parser. Falls Sie sich nicht wirklich gut auskennen, sollten Sie deswegen nicht die LE Version von Tomcat installieren, da diese keinen XML Parser enthält.
2. Axis benötigt Java 1.3 oder neuer.

1.2. Was Sie wissen sollten

Diese Einführung setzt voraus, dass Sie Java Kenntnisse haben und eventuell die Starthilfe zu SOAP durchgearbeitet haben (von SOAP sieht man eigentlich sehr wenig in Axis, aber wenn man will, kann man direct auf SOAP zugreifen und die Datenfelder / Methoden nutzen).

Hier eine kurze Zusammenstellung einiger Themen, die Sie kennen sollten:
1. Java Basisdatentypen, Klassen und OO Programmierkonzepte
2. Threads, Threads-Sicherheit, Synchronisation und Race Bedingungen
3. Klassenlader, hierarchische Klassenlader und typische Ursachen für die „ClassNotFoundException".
4. Stack Trace Analyse, NullPointerException und andere gängige Exceptions, deren Ursache und Behebung.
5. Web Applikationen: Servlets, Verzeichnisse für Servlets und Bibliotheken.
6. Web Applikation: starten, deployen einer Web Applikation
7. Netzwerke: speziell IP Adressen, Sockets, TCP/IP
8. http: Fehlercodes, Headers evtl. Authentisierung

9. XML: wohlgeformte und gültige Dokumente.

Ihr Verständnis dieser Themen kann unterschiedlich tief sein. Früher oder später werden Sie aber vertiefte Kenntnisse benötigen.

1.3. Axis Konzepte

Apache Axis ist ein Open Source SOAP Server und Client. SOAP ist ein Mechanismus für die Inter-Applikations-Kommunikation zwischen Systemen, welche in unterschiedlichen Programmiersprachen realisiert wurden. Als Basisprotokoll werden Internet Protokolle (SMTP, http, ...) verwendet. SOAP verwendet in der Regel http: der Client verwendet den POST Befehl, um einen SOAP Request an den Server zu übermitteln. Als Antwort erhält der Client entweder einen http Fehlercode oder Erfolgscode plus eine SOAP Antwort.

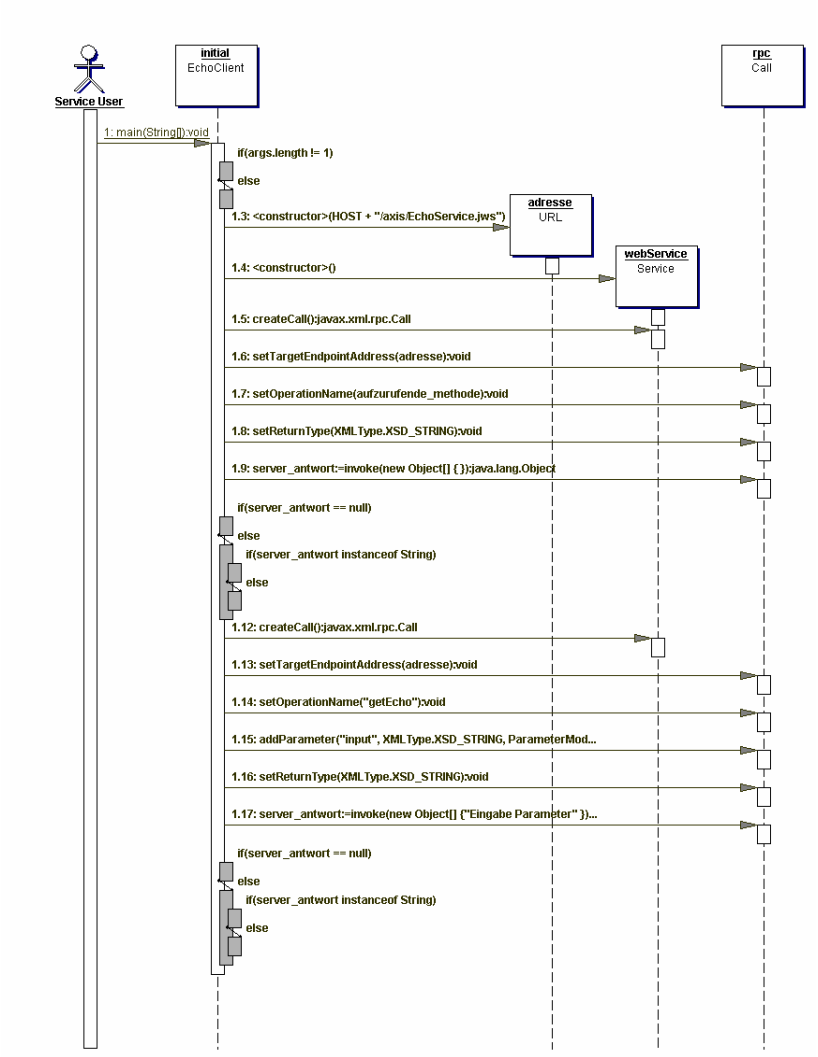
Da Axis Open Source ist, erhalten Sie zwar den Source Code (für nicht kommerzielle Nutzung); aber es steht Ihnen bei Problemen keine Support Organisation zur Verfügung. SOAP Messages / Request / Response sind XML Dokumente. Mithilfe strukturierter SOAP Messages kann Information zwischen SOAP Systemen ausgetauscht werden. Die SOAP Nachricht besteht aus einem oder mehreren SOAP Elementen, in einem SOAP Umschlag, Headers und dem SOAP Body / Rumpf. SOAP kennt zwei grundsätzliche Schemata: RPC ähnlich und Message basiert. Die meisten Beispiele benutzen das RPC Schema; für die Praxis sind aber Messages wichtiger.

Wir benötigen also einen Client und einen Server, wobei beide in unterschiedlichen Programmiersprachen erstellt und auf unterschiedlichen Betriebsystemen ausgeführt werden können.

1. der Client und Server besitzen ihre eigenen Attribute und Methoden
   Bevor ein Client einen Service nutzen kann, muss der Server gestartet werden.
   Für alle, welche das Activation Framework von Java kennen: vergleichbares gibt es bei Web Services noch nicht.
2. der Client muss den Server kennen, beispielsweise seine URL;
   der Client muss auch die Signatur / das „Protokoll“ der Server-Methode kennen: Parameter, Parametertypen, Rückgabetypen.
3. der Aufruf der Methode auf dem Server durch den Client geschieht je nach Framework leicht unterschiedlich.

Im Folgenden Sequenzdiagramm sehen Sie zwei Methodenaufrufe

- beide Aufrufe entsprechen grob einem Hello World
- der eine Aufruf übergibt einen String Parameter
- der andere Aufruf geschieht ohne Parameter und liefert einfach einen Antwort-String.



1. das Starten des Servers vergessen wir hier.
2. der Client zeigt ein einfaches „Pre-Processing“ (vorbereitende Abfragen und Initalisierungen), u.a. bestimmen der relevanten URL.
3. der Aufruf des URL Konstruktors stellt die Verbindung zum anbietenden Server her.
4. der eigentliche Service wird mithilfe des Axis Frameworks, speziell der `Call` Klasse realisiert.
   Diese Klasse benötigt verschiedene Angaben:
   den Target-Endpoint (Service URL);
   den Namen der remote Procedure (`setOperationName`), den Datentyp des Rückgabewertes (`setReturnType`) und falls Parameter angegeben werden, deren Datentyp und Wert (`addParameter`).
5. Die Antwort des Servers wird durch den Rückgabewert der Framework-Methode (`Call invoke`), wobei auch dieser Datentyp, wie in 4) erwähnt, im `Call` Objekt festgelegt werden muss (`setReturnType`).
6. Danach geschieht ein Post-Processing auf dem Client.

Axis wandelt Java Objekte in SOAP um. SOAP Fehler werden in Java Exceptions umgewandelt.

SOAP ist darauf ausgerichtet, unterschiedliche Systeme miteinander zu kombinieren. SOAP will nicht RMI oder CORBA, also eng gekoppelte Systeme, ersetzen.

Axis implementiert das JAX-RPC API, ein Standard-API für die Programmierung von Java Services. Wenn Sie sich auf diese API’s beschränken, dann kann Ihre Applikation auch mit Systemen von BEA oder Sun oder IBM oder .... kommunizieren.

Axis bietet auch Erweiterungen, die aber nur in einer Axis Umgebung funktionieren.

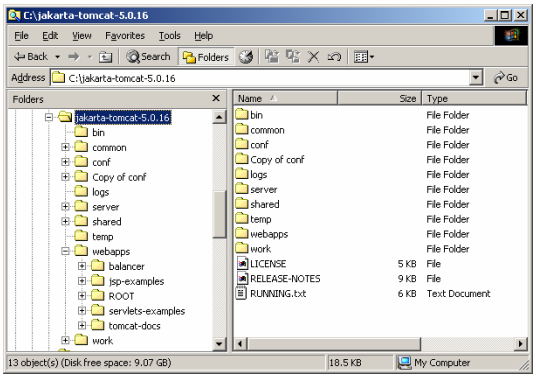
Axis ist als Java Archiv implementiert:

1. Axis Kernsystem : `axis.jar`
2. JAX-RPC: `jaxrpc.jar`, `saaj.jar`
3. Hilfsbibliotheken:
   Logging, WSDL Verarbeitung, Introspection

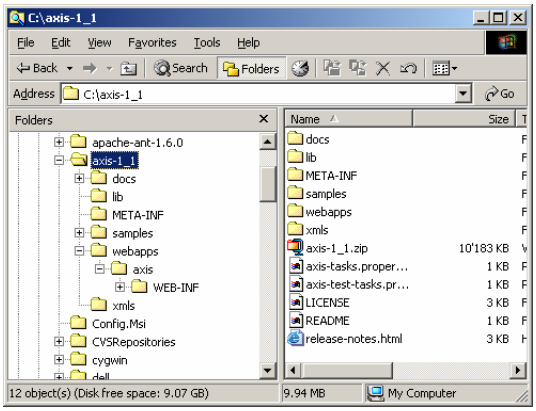
Alle benötigten Dateien können Sie in ein Web Archiv `axis.war` zusammengefassen. Dieses können Sie einfach in den Servlet Container kopieren; es entpackt sich beim nächsten Container Start (beispielsweise Tomcat). Axis kommt bereits mit mehreren SOAP Services. Sie können mit diesen starten und daraus eigene Services entwickeln.

1.4. Vorbereitung

Falls Sie Tomcat ohne Änderungen installiert haben, dann ist der Standardport 8080 auf `localhost`. Im Folgenden gehen wir davon aus, dass der Port 8080 ist. Ich habe Jakarta Tomcat 5 installiert (zip Version, im Verzeichnis `TOMCAT_HOME=C:\jakarta-tomcat-5.x.y`). Die Version mit Installer liefert eine Programmgruppe und einfachere Start und Stop-Routinen (aus dem Menü).

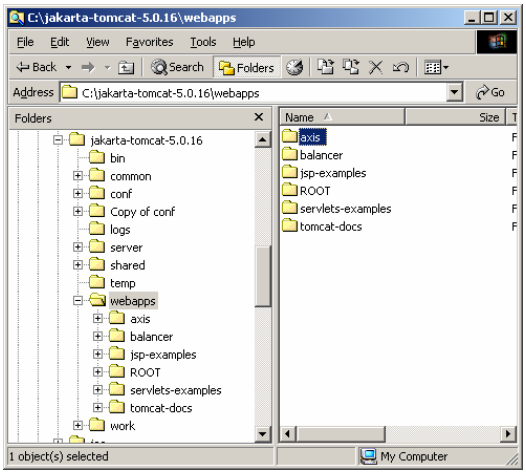


Sie sollten in Ihrem Servelt Container (Tomcat, BEA, ...) ein Verzeichnis für Web Applikationen finden. Laden Sie nun eine stabile Version von Axis herunter und entzippen Sie das Archiv:



`-Apache Site http://ws.apache.org/axis/`
`-Download http://ws.apache.org/axis/releases.html`
wählen Sie eine *final* Version (Axis ist noch etwas unreif)

Entzippen Sie das Archiv, beispielsweise in c:.Sie können dann das Web Applikations-Verzeichnis aus dem Axis Verzeichnis ins Web Applikations-Verzeichnis des Applikations-Servers kopieren.



Weiter unten werden wir ein `war` Archiverstellen. Damit wird die Installation automatisiert.

Das Verzeichnis muss nicht Axis heissen: Sie können einen beliebigen Namen für das Verzeichnis wählen, müssen aber im Folgenden einfach den passenden Namen wählen. Wir werden möglichst mit Symbolen arbeiten, also beispielsweise `AXIS_HOME`, statt einem konkreten Verzeichnis.

1.5. Aufsetzen der Bibliotheken

Im Axis Verzeichnis finden Sie ein `WEB-INF` Unterverzeichnis. Darin finden Sie einige Grundeinstellungen. Sie können darin aber auch Konfigurations-Einstellungen für Ihre Web Services unterbringen.

Axis benötigt einen XML Parser. Falls Sie Java 1.4 installiert haben, und JAVA_HOME definiert bzw das Java Laufzeitsystem (JRE) installiert haben, dann entfällt dieser Schritt. Java 1.4 enthält den Crimson Parser (nicht den Xerces Parser). Sonst müssen Sie einen JAXP 1.1 kompatiblen Parser herunterladen, beispielsweise Apache Xerces:

1. Apache Site : `http://xml.apache.org/xerces2-j/`
2. Download : `http://xml.apache.org/xerces2-j/download.cgi`

Selbst wenn Sie J2SDK 1.4 installiert haben, können Sie anstelle des Crimson Parsers einen andern Parser einsetzen, beispielsweise Xerces 2.6 (Nov 2003). Dazu müssen Sie den gewünschten Parser ins Verzeichnis `%AXIS_HOME%/WEB-INF/lib` kopieren.

1.5.1. Tomcat 4.x und Java 1.4

Sun hat die Package Struktur mit dem Release J2SDK 1.4 substantiell geändert. Falls Sie beim Testen der happyaxis.jsp Server Page Probleme haben, müssen Sie die Archive aus `axis/WEB-INF/lib` ins Verzeichnis `%CATALINA_HOME%/common/lib` kopieren und Tomcat neu starten.

1.6. Starten des Servlet Containers / Web Servers

Im Falle von Tomcat können Sie je nachdem welche Version Sie installiert haben, auf ein Icon klicken oder das Skript `%CATALINA_HOME%/bin/startup.bat` ausführen.

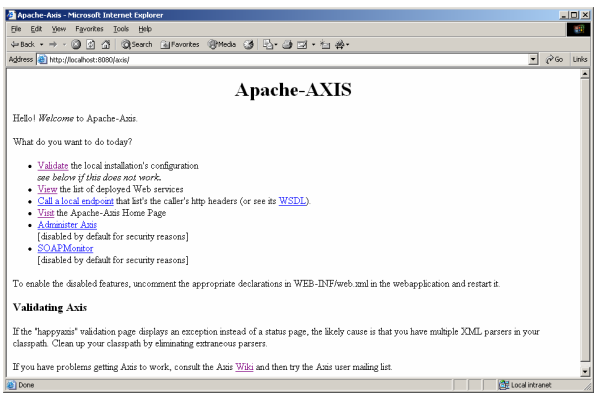
1.7. Validieren der Installation

Bevor Sie irgendetwas Eigenes versuchen, sollten Sie Ihre Installation validieren. Dieser Schritt ist wesentlich und sollte auf keinen Fall ausgelassen werden!

1.7.1. Startseite

Testen Sie als Erstes die Axis Startseite:

- Starten Sie den Server (`%CATALINA_HOME%/bin/startup.bat`)
- geben Sie im Browser die ULR für die Startseite ein:
  `http://localhost:8080/axis/`



Falls alles korrekt installiert wurde, erscheint die folgende Seite:

Falls diese Seite nicht erscheint, testen Sie zuerst Ihren Server, indem Sie `http://localhost:8080` eingeben. Im Falle von Apache Tomcat sollten Sie dann zumindest die Tomcat Startseite sehen.

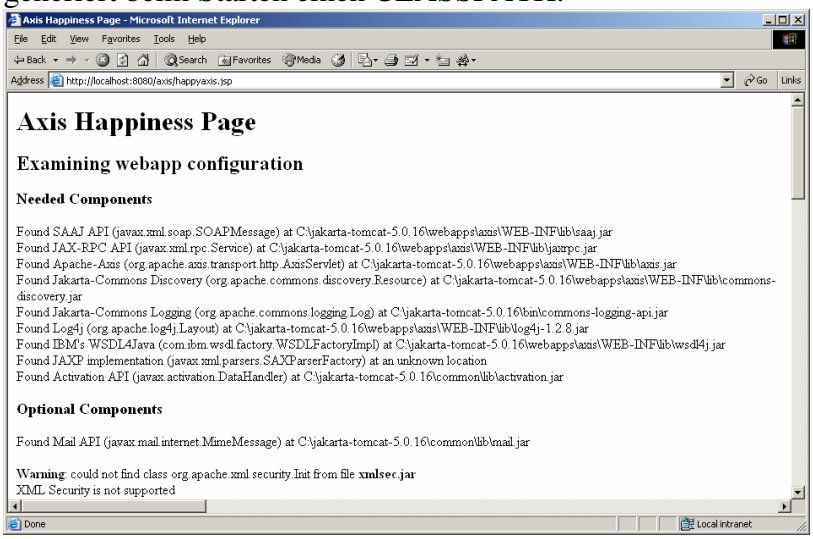
1.7.2. Validieren von Axis mithilfe von happyaxis

Wenn Sie auf der Axis Startseite die Validierung starten, wird im Hintergrund folgende Java Server Page gestartet:

`-http://localhost:8080/axis/happyaxis.jsp`

Diese Seite testet, ob alle Bibliotheken vorhanden sind, ob die Installation korrekt ist. *Diese Validierung ist ser wichtig! Versuchen Sie nichts Eigenes, bevor diese Validierung korrekt ist, alle Bibliotheken korrekt installiert und alles korrekt konfiguriert wurde.*

Alle optionalen Teile und entsprechende Fehlermeldungen können Sie zu Anfang ignorieren! Falls ein Stack Trace generiert wird, könnten mehrere Parser im CLASSPATH sein. Überprüfen Sie Ihren CLASSPATH; in der Regel lasse ich diesen ganz weg. Tomcat generiert beim Starten einen CLASSPATH.



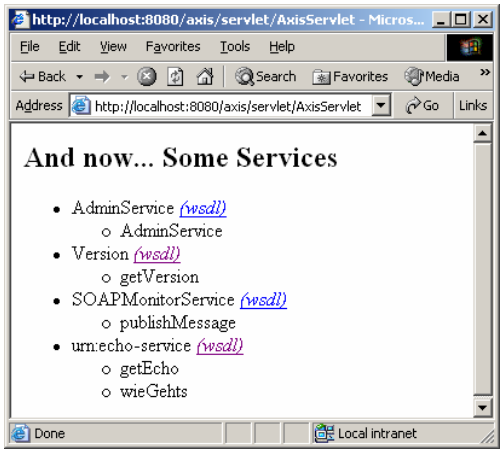
Axis liefert Ihnen wertvolle Hinweise für die Fehlerbeseitigung:

1. welche Bibliotheken fehlen?
2. Wo finden Sie diese?

Sie sehen aber auch, welche optionalen Komponenten noch installiert werden könnten.

1.7.3. Liste der bereits installierten Services

Auf der Startseite von Axis (siehe oben) können Sie die Seite „View the list of deployed Web services“ anwählen. Diese zeigt Ihnen die standardmässig installierten Web Services.Wenn Sie eine dieser „Dienste“ anwählen, erscheint eine XML / WSDL Beschreibung der entsprechenden Dienste. Sehr hilfreich ist diese Beschreibung für Ihren Start mit Axis aber nicht!



In der Abbildung sehen Sie neben den Standard Services den bereits installierten `echo-service` mit den zwei Methoden (`getEcho` und `wieGehts`) , die wir oben bereits im Sequenz-Diagramm angetroffen haben.

1.7.4. Testen eines SOAP Endpoints

Gemäss Definition von SOAP 1.1 wird nur http POST unterstützt. Axis hat aber, unter anderem für Testzwecke, eine einfache Art des GET Befehls implementiert. Damit können

wir die Version der Axis Installation bestimmen:
`http://localhost:8080/axis/services/Version?method=getVersion`

Sie sollten als Antwort etwas wie folgendes XML erhalten:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
<getVersionResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<getVersionReturn xsi:type="xsd:string">Apache Axis version: 1.1 Built on
Jun 13, 2003 (09:19:43 EDT)
</getVersionReturn>
</getVersionResponse>
</soapenv:Body>
</soapenv:Envelope>
```

Eventuell unterscheidet sich das von Ihnen installierte Axis. Der Aufbau des Dienstes erkennen Sie aus der WSDL aus dem vorherigen Abschnitt. Dort haben wir die Liste der vorhandenen Dienste angeschaut. Einer Der Dienste ist der Version Dienst :

- Version (`wsdl`)
  `http://localhost:8080/axis/services/Version?wsdl`
  `getVersion`

Wenn Sie auf WSDL Link fürdiesen Dienst anklicken, dann erhalten Sie die Web Service Description Language (WSDL) :

```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://localhost:8080/axis/services/Version"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://localhost:8080/axis/services/Version"
    xmlns:intf="http://localhost:8080/axis/services/Version"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
- <wsdl:message name="getVersionResponse">
  <wsdl:part name="getVersionReturn" type="xsd:string" />
    </wsdl:message>
  <wsdl:message name="getVersionRequest" />
- <wsdl:portType name="Version">
- <wsdl:operation name="getVersion">
  <wsdl:input message="impl:getVersionRequest" name="getVersionRequest" />
  <wsdl:output message="impl:getVersionResponse" name="getVersionResponse" />
    </wsdl:operation>
    </wsdl:portType>
- <wsdl:binding name="VersionSoapBinding" type="impl:Version">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="getVersion">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="getVersionRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://axis.apache.org" use="encoded" />
    </wsdl:input>
- <wsdl:output name="getVersionResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://localhost:8080/axis/services/Version" use="encoded" />
```

```
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
- <wsdl:service name="VersionService">
- <wsdl:port binding="impl:VersionSoapBinding" name="Version">
  <wsdlsoap:address location="http://localhost:8080/axis/services/Version" />
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Wir werden noch im Detail lernen, wie eine solche WSDL Beschreibung zu lessen ist.Wann immer Sie die WSDL eines Dienstes in Axis benötigen, nach obigem Schema gelangen Sie zur Beschreibung: **<host>/axis/services/<Dienst>?wsdl**

Hier eine erste kurze Version einer Erklärung:

1. die URL des Dienstes: siehe <wsdlsoap:address location="**http://localhost:8080/axis/services/Version**" />
2. der Name der Methode: siehe <wsdl:operation name="**getVersion**">

Der Aufruf einer Methode evtl. mit Parametern, geschieht nach folgendem Muster (von Hand, ohne Einsatz eines selbst programmierten Clients):

```
<location>?method=<Name der Methode>&[parameter=<Parameter>]*
```

1.7.5. Testen eines JWS Endpoints

JWS Dateien sind reine Java Dateien. Axis ist in der Lage, Java Dateien als Web Services zu verwenden:

- beim erstmaligen Aufruf werden die Java Dateien übersetzt (ins Verzeichnis `C:\jakarta-tomcat-5.0.16\webapps\axis\WEB-INF\jwsClasses`)
- nachfolgende Aufrufe verwenden direkt die Class Dateien.

Sie können Ihre Web Service Beschreibung in Java, die JWS Datei, ins `webapp` Verzeichnis kopieren.

- Die Datei muss die Extension JWS besitzten und
- das Verzeichnis darf nicht unterhalb des WEB-INF Verzeichnis sein!

Standardmässig kommt Axis mit einigen JWS Dateien, zum Testen der Installation. Sie finden diese im Axis Verzeichnis. Das SOAP-Monitor Applet habe ich bereits mit folgendem Skript nach der Axis Installation übersetzt (AXIS_HOME ist dabei auf das im obigen Bild erkennbare axis Verzeichnis gesetzt):

```
@echo off
if "%JAVA_HOME%"=="" goto homeLess
if "%AXIS_HOME%"=="" goto noAxis
%JAVA_HOME%\bin\javac -classpath %AXIS_HOME%/WEB-INF/lib/axis.jar
%AXIS_HOME%/webapps/axis/SOAPMonitorApplet.java
goto ende
:homeLess
@echo Sie müssen JAVA_HOME definieren
@echo Beispielsweise set JAVA_HOME = c:\J2SDK1.5.0
goto ende
:noAxis
@echo Sie müssen AXIS_HOME definieren
@echo Beispielsweise set AXIS_HOME = c:\axis-1_1
:ende
```

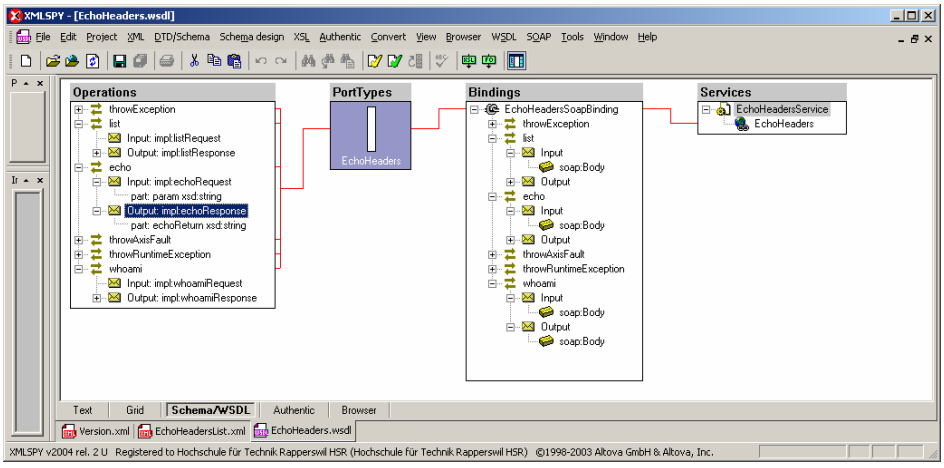
Wir können die in diesem Service vorhandenen Dienste / Methoden entweder mithilfe der WSDL bestimmen, oder einfach abfragen, nach folgendem Schema:
`<host>/axis/<Dienst>?method=list`

Beispiel
`http://localhost:8080/axis/EchoHeaders.jws?method=list`

Mit folgender Antwort:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
- <soapenv:Body>
- <listResponse
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
- <listReturn xsi:type="soapenc:Array" soapenc:arrayType="xsd:string[6]"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
  <item>accept:image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
    application/vnd.ms-excel, application/vnd.ms-powerpoint,
    application/msword, application/x-shockwave-flash, */*</item>
  <item>accept-language:de-ch</item>
  <item>accept-encoding:gzip, deflate</item>
  <item>user-agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR
    1.1.4322)</item>
  <item>host:localhost:8080</item>
  <item>connection:Keep-Alive</item>
  </listReturn>
  </listResponse>
  </soapenv:Body>
  </soapenv:Envelope>
```

Die Antwort wird Ihrer Umgebung entsprechend anders sein (zum Beispiel der Host).
Geben Sie `http://localhost:8080/axis/EchoHeaders.jws?wsdl`



ein, um die WSDL Beschreibung des Dienstes zu erhalten. In XML Spy sieht diese wie nebenstehend aus.

Aus der Beschreibung erkennen wir, dass ein Dienst `whoami` definiert ist. Diesen wollen wir kurz testen.

1. Eingabe: URL im Browser:
   http://localhost:8080/axis/EchoHeaders.jws?method=whoami
2. Ausgabe:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
- <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
- <soapenv:Body>
- <whoamiResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <whoamiReturn xsi:type="xsd:string">Hello caller from
127.0.0.1</whoamiReturn>
  </whoamiResponse>
  </soapenv:Body>
  </soapenv:Envelope>
```

In Ihrer Installation kann natürlich ein anderer Host ausgegeben werden.

1.8. Beispiel eines Echo Web Service

Dieses Beispiel zeigt eine, wenn auch nicht immer sinnvolle Art und Weise, wie auf einfachste Art und Weise ein Web Service an Axis übergeben werden kann. Der Web Service entspricht dem klassischen „Hello World“ Programm.

1.8.1. Der (Web) Service

Unser Service sendet auf Anfrage die Zeichenkette „Hallo, wie geht’s“ an den Client. Damit der Service nicht ganz so banal ist, habe ich noch zwei weitere Methoden hinzugefügt:

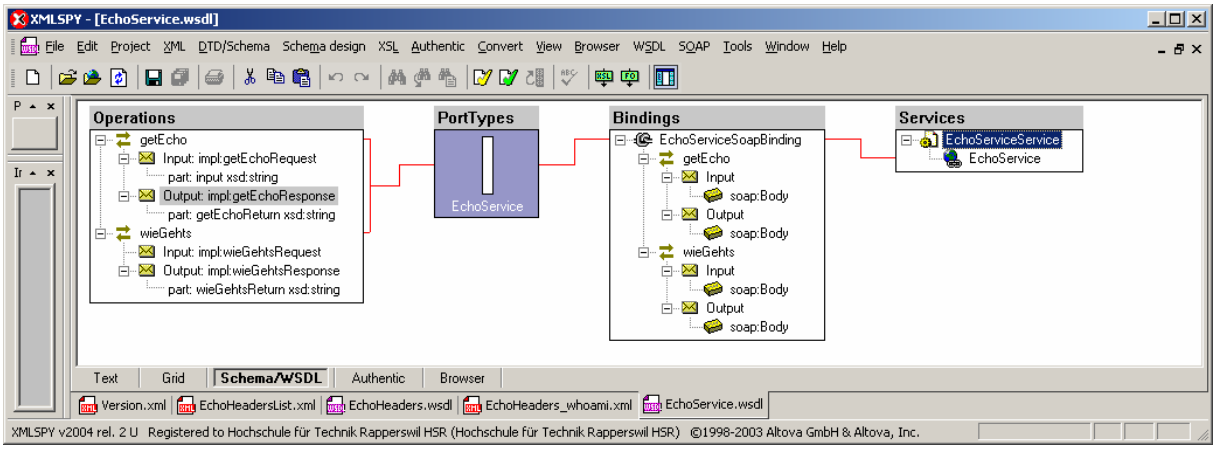
```
/**
* Kopieren Sie diese Datei ins AXIS_HOME Verzeichnis
* aendern Sie die Extension von java in jws
* starten Sie Axis / Tomcat neu
* Dann sollte dieser Service Axis bekannt sein
*
*/
public class EchoService {
	// der Rechnername des Servers
	public final String hostname = "localhost";
	// gibt den String eines Clients als Echo zurück
	public String getEcho(String input) {
		return("Echo von Host " + hostname + " : " + input);
	}
	// Hello World
	public String wieGehts() {
		return("Hello World");
	}
}
```

Gehen Sie jetzt so vor, wie oben im Kommentar beschrieben:
1) kopieren dieser Datei ins `AXIS_HOME` Verzeichnis
2) umbenennen der Datei in `EchoService.jws` (Java Web Service)
3) Shutdown und Restart von Tomcat
Nun wollen wir überprüfen, ob Axis wirklich den Service erkannt hat:

1. als erstes betrachten wir das Verzeichnis `%AXIS_HOME%\WEB-INF\jwsClasses`. Ohne unser Zutun wird die Class Datei unseres Dienstes dort *nicht* erscheinen, da der Dienst erst bei seinem ersten Aufruf übersetzt wird.
2. Auch beim Aufruf des Service Applets `http://localhost:8080/axis/servlet/AxisServlet` wird der neue Dienst nicht angezeigt.
3. Die Abfrage der WSDL generiert aber die Class Datei – diese ist nun im Verzeichnis `%AXIS_HOME%\WEB-INF\jwsClasses` vorhanden.

4. Im Tomcat Fenster erscheint die Meldung:
   `Unable to find config file. Creating new servlet engine config file: /WEB-INF /server-config.wsdd`
   In diese Datei (server-config.wsdd) werden die Dienste eingetragen.

Als Ergebnis erhalten wir die von Axis generierte WSDL unseres EchoServices, hier in XML Spy importiert (gespeicherte Ausgabe des Explorer, wie oben):



Axis hat aus unseren Java Methoden jeweils einen Input / Request und einen Output / Response Teil generiert. Wir können den Service direkt d.h. ohne Client testen:

1) `http://localhost:8080/axis/EchoService.jws?method=getEcho&input="Hallo"` liefert

   (der Name des Parameters ist dabei frei wählbar!)

```
<?xml version="1.0" encoding="UTF-8" ?>
- <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
- <soapenv:Body>
- <getEchoResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
>
  <getEchoReturn xsi:type="xsd:string">Echo von Host NINFPDW11
(80.218.25.236) : "Hallo"</getEchoReturn>
  </getEchoResponse>
  </soapenv:Body>
  </soapenv:Envelope>
```

2) http://localhost:8080/axis/EchoService.jws?method=wieGehts
   liefert

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
- <soapenv:Body>
- <wieGehtsResponse
```

```
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <wieGehtsReturn xsi:type="xsd:string">Hello World</wieGehtsReturn>
  </wieGehtsResponse>
  </soapenv:Body>
  </soapenv:Envelope>
```

Diese Deployment Methode ist nur in einfachen Fällen sinnvoll. In den meisten Fällen muss ein Web Service Deployment Descriptor erstellt oder angepasst werden!

1.8.2. Der Echo Client

Der Client ist komplexer als der oben beschriebene Service. Der Aufbau des Service Clients ist aber nicht so shwer zu verstehen:

```
/**
* Der EchoService Client
 */
import java.net.URL;

import javax.xml.rpc.ParameterMode;

import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;

public class EchoClient {

      public static void main(String args[]) throws Exception {

            final String HOST;
            // 1.Argument http://<rechnername>:<port>
            // Beispiel: http://pinfpgd11:8080
            if (args.length != 1) {
                  System.out.println(
                  "[usage] : EchoClient http://<rechnername>:<port> sein");
                  //return;
                  HOST = "http://localhost:8080";
            } else {
                  HOST = args[0];
            }
            System.out.println("[EchoClient]Host : "+HOST);

            // URL des Webservices
            URL adresse = new URL(HOST + "/axis/EchoService.jws");
      System.out.println("[EchoClient]Zieladresse: " + adresse.toString());

            // Aufzurufende Methode festlegen
            String aufzurufende_methode = "wieGehts";
            //getEcho(String), wieGehts()

            // Erzeugung eines Service-Objektes, mit dessen Hilfe der
            // RPC durchgeführt werden kann :
            Service webService = new Service();

            // Erzeugung eines Calls zur Übertragung der Daten
            // von Client zu Server:
            Call rpc = (Call) webService.createCall();

            // Spezifikation der Zieldaten des Calls :
            // Welcher Web Service
            // Welche Methode
            // Welche Parameter
```

```
// Welcher Rückgabetyp
rpc.setTargetEndpointAddress(adresse);
rpc.setOperationName(aufzurufende_methode);
/*rpc.addParameter("input", // Parameterbezeichnung
      XMLType.XSD_STRING, // XSL-Typ des Parameters
      ParameterMode.IN); // Nur übergeben
*/
rpc.setReturnType(XMLType.XSD_STRING);
// XSL-Typ des Rückgabewertes

// Aufruf der Methode invoke ruft den Webservice auf
// als Rückgabewert erhalten wir die Antwort des Servers
System.out.println("[EchoClient]Durchführung des RPC: ");
// für zweite Methode
//Object server_antwort = rpc.invoke(new Object[] {"Eingabe Parameter" });
Object server_antwort = rpc.invoke(new Object[] { });

// Antwort des Servers auswerten
if (server_antwort == null) {
      System.err.println("[EchoClient]Keine Übereinstimmung");
} else if (server_antwort instanceof String) {
      // wir haben einen String als Antwort erhalten
      System.out.println("[EchoClient]"+server_antwort);
} else {
      // kein String -> Fehler
      System.err.println("[EchoClient]"+
"Unbekannter Typ: " + server_antwort.getClass().getName());
}

// Erzeugung eines Calls zur Übertragung der Daten
// von Client zu Server:
rpc = (Call) webService.createCall();

// Spezifikation der Zieldaten des Calls :
// Welcher Web Service
// Welche Methode
// Welche Parameter
// Welcher Rückgabetyp
rpc.setTargetEndpointAddress(adresse);
rpc.setOperationName("getEcho");
rpc.addParameter("input", // Parameterbezeichnung
      XMLType.XSD_STRING, // XSL-Typ des Parameters
      ParameterMode.IN); // Nur übergeben

rpc.setReturnType(XMLType.XSD_STRING);
// XSL-Typ des Rückgabewertes
// Aufruf der Methode invoke ruft den Webservice auf
// als Rückgabewert erhalten wir die Antwort des Servers
System.out.println("[EchoClient]Durchführung des RPC: ");
server_antwort = rpc.invoke(new Object[] {"Eingabe Parameter"
       });
//server_antwort = rpc.invoke(new Object[] { });

// Antwort des Servers auswerten
if (server_antwort == null) {
      System.err.println("[EchoClient]Keine Übereinstimmung");
} else if (server_antwort instanceof String) {
      // wir haben einen String als Antwort erhalten
      System.out.println("[EchoClient]"+server_antwort);
} else {
      // kein String -> Fehler
      System.err.println("[EchoClient]"+
```

```
                "Unbekannter Typ: " + server_antwort.getClass().getName());
                }

        }
}
```

Falls Sie damit nicht zufrieden sind, haben Sie zwei Möglichkeiten:

1. Sie können den obigen Web Service Client so umbauen, dass er auch die parametrisierte Methode aufrufen kann.Dazu müssen Sie die Parameter angeben, die im obigen Client auskommentiert sind.Der Methodenaufruf bei `invoke()` kann ebenfalls einen Aufrufparameter besitzen.
2. Sie bauen einen eigenen Web Service.

Axis bietet ein Tool, mit dessen Hilfe aus WSDL eine Java Klasse generiert werden kann. Das schauen wir uns später an. Falls Sie beispielsweise Packages verwenden, ist das Deployment mithilfe eines Web Service Deployment Descriptors WSDD sehr einfach. Die obige Methode mit dem JWS dagegen eher mühsam bzw. offiziell nicht möglich.

1.9. Installation neuer Web Services

Jetzt sollten Sie Axis etwas getestet haben. Fassen wir kurz zusammen, wie das Entwickeln und ein Deployment geschieht. Grob geschieht dies in folgenden zwei Schritten:

1. erstellen Sie Ihre Klassen und Bibliotheken
2. infomieren Sie Axis mithilfe eines Deployment Descriptors.

Sie können Axis über den Admin Web Service bekannt geben, oder mit dem AdminClient Programm oder mit der `<axis-admin>` Ant Task. Der Admin Web Service ist ein spezieller Service, der aus Sicherheitsgründen nur lokal gestartet werden kann und Passwort geschützt ist. Die Ant Tasks werden wir noch kennen lernen.Also:

1. zuerst müssen Sie Ihre Klassen (Service) kreieren, übersetzen und ins „classes“ Verzeichnis (AXIS_HOME/WEB-INF/classes) kopieren. Beachten Sie dabei allfällige Package bedingte Unterverzeichnisse.
2. falls Sie wollen, können Sie auch ein JAR erstellen und dieses ins WEB-INF/lib Verzeichnis kopieren. Sie können auch weitere benötigte Bibliotheken dort hin kopieren.
3. nach dem Kopieren müssen Sie Ihren Web Server neu starten!
4. falls Sie die mit Axis gelieferte Authorisierung verwenden (reicht nicht im professionellen Umfeld), dann müssen Sie eine Datei `users.lst` ins Verzeichnis WEB-INF kopieren.

Damit sind zwar die Dateien beim Server, aber der Dienst ist dem Server noch nicht bekannt (ausser wir verwenden den JWS Mechanismus wie im obigen Beispiel).

1.10. Deployen eines Web Service

Nun geht es darum, den Web Service anzumelden, Axis bekannt zu geben. Axis benötigt einen Web Service Deployment Descriptor (eine WSDD Datei). In diesem wird beschrieben, in XML, was der Service ist, welche Methoden exportiert werden (als Services zur Verfügung stehen) sowie andere Aspekte des SOAP Endpoints.

Im nächsten Tutorial lernen Sie den Aufbau einer WSDD Datei kennen. Hier setzen wir einen bereits bestehende an. Als Beispiel verwenden wir einen Börsen-Ticker Service.

Da ich mit Eclipse entwickle, beschränke ich mich darauf, die CLASSPATH Einträge aufzulisten, die ich in meinem Axis Projekt als Java Classpath Properties angegeben habe:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <classpath>
  <classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER" />
  <classpathentry kind="var" path="TOMCAT_HOME/common/lib/servlet-api.jar" />
  <classpathentry kind="var" path="TOMCAT_HOME/common/lib/jasper-runtime.jar"
    />
  <classpathentry kind="var" path="TOMCAT_HOME/common/lib/jsp-api.jar" />
  <classpathentry kind="src" path="WEB-INF/src" />
  <classpathentry kind="src" path="work" />
  <classpathentry kind="lib" path="C:/axis-1_1/lib/wsdl4j.jar" />
  <classpathentry kind="lib" path="C:/axis-1_1/lib/axis-ant.jar" />
  <classpathentry kind="lib" path="C:/axis-1_1/lib/commons-discovery.jar" />
  <classpathentry kind="lib" path="C:/axis-1_1/lib/commons-logging.jar" />
  <classpathentry kind="lib" path="C:/axis-1_1/lib/jaxrpc.jar" />
  <classpathentry kind="lib" path="C:/axis-1_1/lib/log4j-1.2.8.jar" />
  <classpathentry kind="lib" path="C:/axis-1_1/lib/saaj.jar" />
  <classpathentry kind="lib" path="C:/axis-1_1/lib/axis.jar" />
  <classpathentry kind="output" path="WEB-INF/classes" />
  </classpath>
```

Sie benötigen in Ihrem CLASSPATH:
`axis.jar, commons-discovery.jar, commons-logging.jar, jaxrpc.jar, saaj.jar, log4j-1.2.8.jar`
XML Parser : jar File oder Files (z.B., `xerces.jar` oder J2SDK 1.4 oder neuer).

1.10.1. Deployment Descriptor

Für den Börsen-Ticker benötigen wir folgenden Axis WSDD:

```
<deployment name="Stock Service"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
    <service name="urn:xmltoday-delayed-quotes" provider="java:RPC">
        <parameter name="className" value="samples.stock.StockQuoteService"/>
        <parameter name="allowedMethods" value="getQuote test"/>
        <parameter name="allowedRoles" value="user1,user2"/>
        <parameter name="wsdlServicePort" value="GetQuote"/>
        <requestFlow name="checks">
            <handler type="java:org.apache.axis.handlers.SimpleAuthenticationHandler"/>
            <handler type="java:org.apache.axis.handlers.SimpleAuthorizationHandler"/>
        </requestFlow>
    </service>
    <service name="urn:cominfo" provider="java:RPC">
        <parameter name="className" value="samples.stock.ComInfoService"/>
        <parameter name="allowedMethods" value="getInfo"/>
        <parameter name="allowedRoles" value="user3"/>
        <requestFlow type="checks"/>
    </service>
</deployment>
```

Diese Beschreibung können wir auf folgende Weise dem Server bekannt geben:

1. `java org.apache.axis.client.AdminClient deploy.wsdd`
   oder

2. `java org.apache.axis.utils.Admin client|server deploy.wsdd`
   generiert bei „`client`“ : `client-config.wsdd` bzw. bei „`server`“ : `server-config.wsdd`

Sie finden Skripts für beide Varianten auf dem Server (siehe unten: wir testen beide Möglichkeiten). In einem weiteren Tutorial gehen wir im Detail auf diese Programme ein. Axis speichert die Informationen über die Services in der Datei `axis/WEB-INF/server-config.wsdd`.

Falls Sie eine Warnung am Bildschirm sehen, dass diese Datei nicht gefunden werde, dann ist es wahrscheinlich Ihr erstes Deployment, die Datei muss also durch Axis erst angelegt werden. Selbstverständlich benötigt Axis Zugriffsrechte auf das Verzeichnis und die Datei.

Der Börsen-Ticker verwendet seinerseits einen Axis Web Service, leitet also die Anfrage ans Web weiter.

1.10.2. Admin Client

Das Deployment mit dem AdminClient ist recht einfach:

1. Sie benötigen den Deployment Deskriptor (`deploy.wsdd`)
2. Sie müssen wissen, wo / auf welchem Server und welchem Port der Dienst angeboten werden soll
3. Und hier die Kommandozeile:

```
java -cp %AXIS_CLASSPATH% org.apache.axis.client.AdminClient
      -lhttp://localhost:8080/axis/services/AdminService deploy.wsdd
```

Dieser Befehl muss im Verzeichnis ausgeführt werden, in dem der Deployment Deskriptor (`deploy.wsdd`) steht.

1.11. Testen der Applikation

Nun wollen Sie sicher wissen, ob Ihre Applikation funktioniert. Dazu müssen Sie unter Umständen berechtigte Benutzer definieren und in eine Datei users.lst eintragen.

```
java -cp "%AXIS_CLASSPATH%" GetQuote -
lhttp://localhost:8080/axis/servlet/AxisServlet -uuser1 -wpass1 XXX
```

Als Ergebnis erhalten Sie den Standardwert "55.25".

1.12. Axis Web Modul

Bisher haben wir die Applikationen einfach manuell ins passende Verzeichnis kopiert. Nun wollen wir noch eine Alternative kennen lernen – Web Archive.

Vorgehen:

1. wechseln Sie ins Verzeichnis `AXIS_HOME\webapps\axis`
2. bilden Sie das Web Archiv (`axis.war`):
   `%JAVA_HOME%\bin\jar -cvf axis.war *`

Der Web Modul kann nun bei kommerziellen Applikations-Servern in der Regel mithilfe eines GUI basierten Tools oder einfach ins Verzeichnis webapps des Applikations-Servers kopiert werden.

Mit Axis wird ein Monitor Applet geliefert. Dieses finden Sie im Verzeichnis AXIS_HOME\webapps\axis. Dieses Applet muss nicht nur übersetzt werden, sondern auch deployed.

1. übersetzen:
   ```
   javac -classpath WEB-INF/lib/axis.jar SOAPMonitorApplet.java
   ```
2. deployen:
   ```
   java -classpath %AXIS_CLASSPATH% org.apache.axis.client.AdminClient -
   h <host>-p <port> deploy.wsdd
   ```

Der Deployment Descriptor ist dabei gegeben als:

```
- <deployment xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
- <handler name="soapmonitor"
    type="java:org.apache.axis.handlers.SOAPMonitorHandler">
  <parameter name="wsdlURL" value="/axis/SOAPMonitorService-impl.wsdl" />
  <parameter name="namespace"
    value="http://tempuri.org/wsdl/2001/12/SOAPMonitorService-impl.wsdl" />
  <parameter name="serviceName" value="SOAPMonitorService" />
  <parameter name="portName" value="Demo" />
   </handler>
- <service name="SOAPMonitorService" provider="java:RPC">
  <parameter name="allowedMethods" value="publishMessage" />
  <parameter name="className"
    value="org.apache.axis.monitor.SOAPMonitorService" />
  <parameter name="scope" value="Application" />
   </service>
   </deployment>
```

1.13. Schlussbemerkungen

Axis ist eine ziemlich einfache Implementierung von SOAP. Damit können Sie in der Regel einfache und funktionsfähige Web Applikationen erstellen, als Web Services.
Die Installation kann etwas trickreich sein, aber keine unüberwindbaren Probleme produzieren.

APACHE AXIS