Auszug aus
Axis2 Übungen

Version 1.1

Dieses Dokument ist ein Auszug aus unserem Übungsskript zur Axis2-Schulung. Es dient lediglich als Beispiel für unsere Kursunterlagen. Für die Durchführung der Übungen wird zusätzlich eine Kurs-CD und ein Foliensatz mit Hintergrundinformationen benötigt, auf der Projektvorlagen für die jeweiligen Übungen zu finden sind.

Falls Sie Interesse an dieser Schulung haben können Sie sich gerne an mich wenden. Mehr zum Kurs finden Sie unter:

http://www.thomas-bayer.com/axis2-schulung.htm

Thomas Bayer
Nikolausstraße 107
50937 Köln

Tel. +49 (221) 4249250
info@thomas-bayer.com

von Thomas Bayer und Marco Hippler

Übung: XML erzeugen mit Event-API

1. Betrachten Sie die Klasse:

```
com.thomas_bayer.stax.WriterMitEventAPI
```

2.Führen Sie die Klasse aus.
3. Erweitern Sie das erzeugte Dokument, so dass folgende Struktur erzeugt wird:

```
<?xml version="1.0" ?>
<f:foo xmlns:f="http://baz">
   <bar>Hello StAX</bar>
   <fooagain>
      <bar>Hello again</bar>
   </fooagain>
</f:foo>
```

1.3 AXIOM

Übung: XML mit AXIOM einlesen

1. Wechseln Sie zu Projekt zoohandlungclient.
2. Betrachten Sie die Klasse:

```
com.thomas_bayer.axiom.AXIOMLesenUndSchreiben
```

3. Betrachten Sie die einzulesende XML Datei:

```
resources/foo.xml
```

4. Führen Sie die Klasse aus.

Übung: Traversieren von XML mit AXIOM

1. Geben Sie die Kind-Elemente des Document-Elementes aus. Fügen Sie dazu nach der Ausgabe des Document-Elementes den folgenden Code in AXIOMLesenUndSchreiben ein:

```
Iterator<OMElement> elements=doc.getChildElements();
while(elements.hasNext())
{
   OMElement elem=elements.next();
   System.out.println(elem.getLocalName());
}
```

2. Führen Sie die Klasse aus.

2. Erste Schritte mit Axis2

Übung: Installation der Axis2 Webanwendung

Voraussetzung: axis2.war wurde aus Binär-Distribution erstellt oder heruntergeladen.

1. Kopieren Sie die Axis2 Webanwendung in das webapps Verzeichnis von Tomcat.
2. Starten Sie Tomcat.
3. Rufen Sie die Tomcat-Manager Webanwendung mit der folgenden URL auf:

http://localhost:8080/manager/html

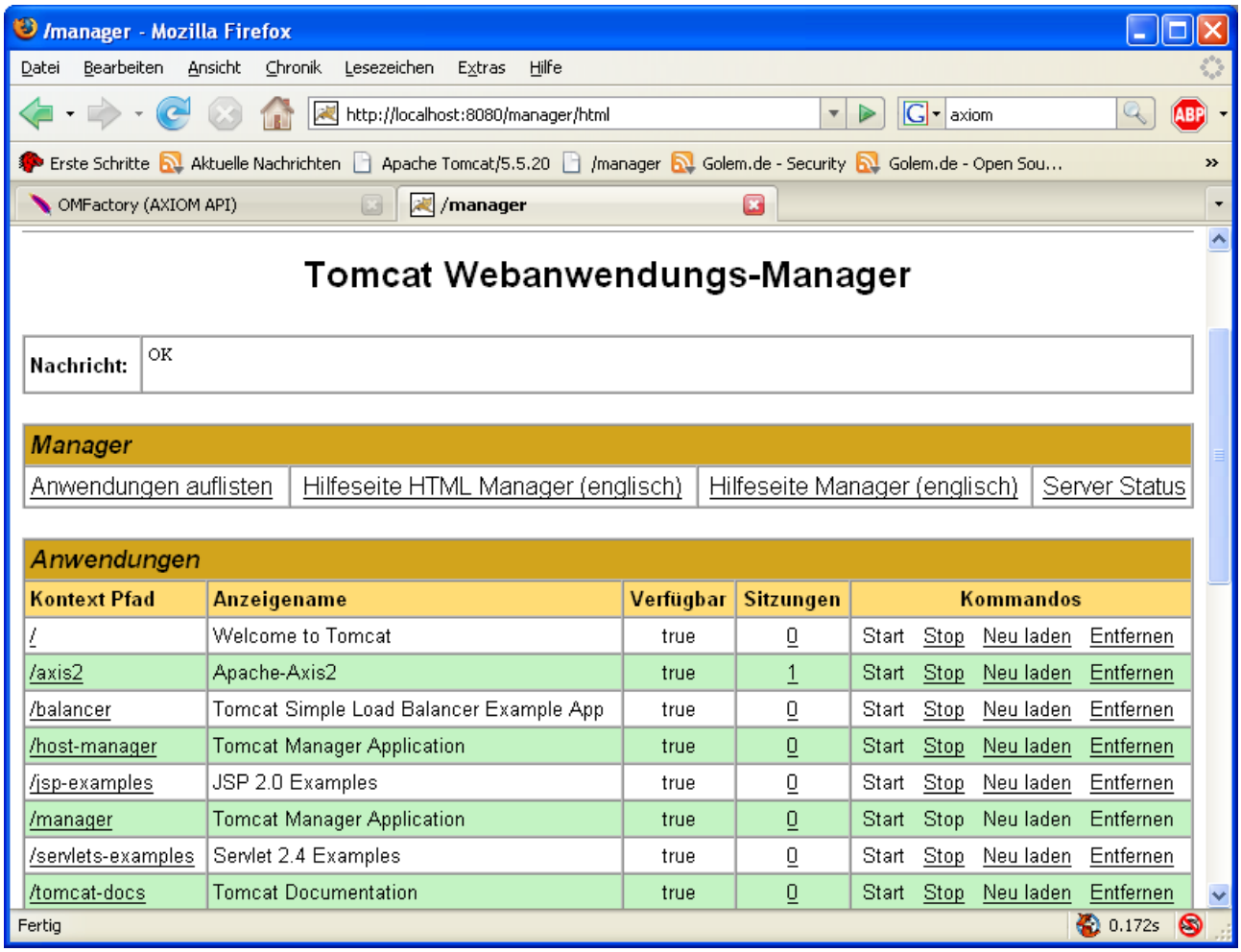

Bild1: Tomcat Manager

4. Rufen Sie die Axis2-Webanwendung auf

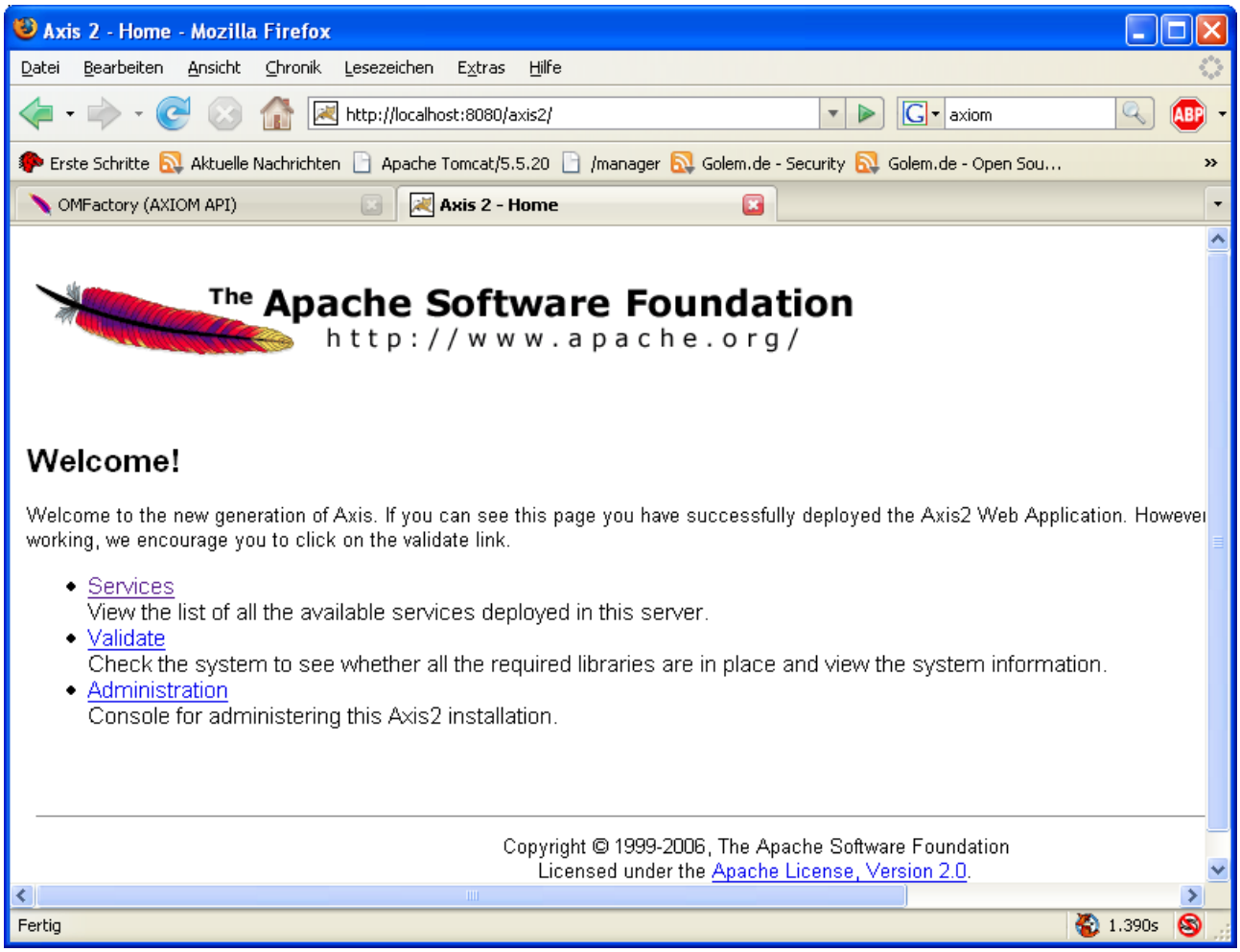

Bild2: Axis2 Startseite

5. Klicken Sie auf den Link Validate um die Installation zu überprüfen.

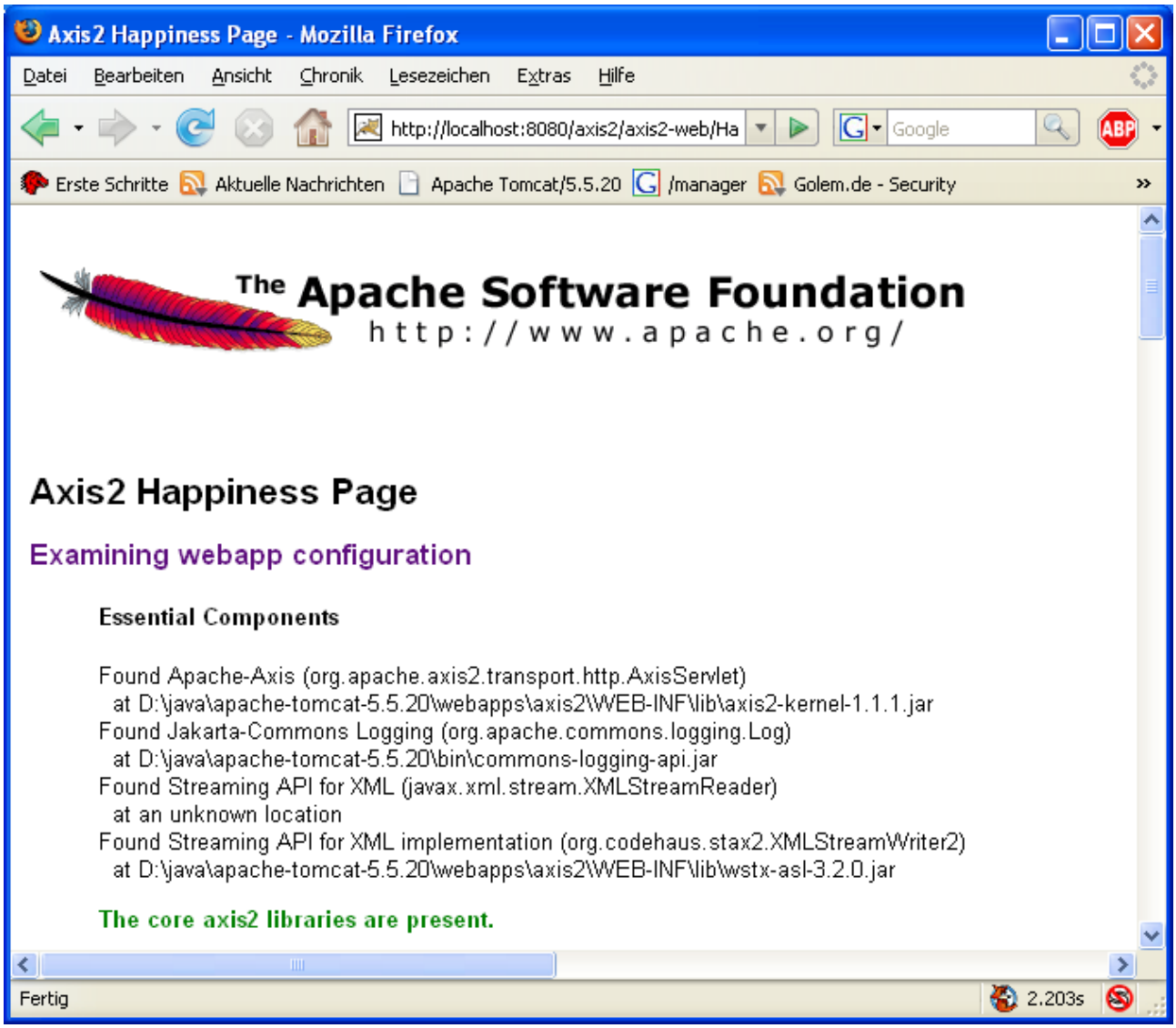

Bild3: Axis2 Validate-Seite

6. Gehen Sie zurück auf die Welcome-Seite und klicken Sie auf den Link Services.

Übung: Hotdeployment von Services

1. Erweitern Sie die Implementierung vom Buch-Service um die Ausgabe der ID:

```
public class BuchServiceImpl {

   public int addBuch(int id, String name)
   {
       System.out.println("Buch hinzugefuegt: "
                          +name+" id: "+id);
       return 0;
   }
}
```

2. Führen Sie ein Deployment des Buch-Services durch, indem Sie das Ant-Target deploy.service ausführen.
3. Starten sie den Client
4. Betrachten Sie die Ausgabe der Tomcat-Konsole. An der Ausgabe sollte sich nichts verändert haben.
5. Beenden Sie Tomcat und starten Sie Tomcat anschließend erneut. Danach führen Sie den Client erneut aus.
6. Öffnen Sie die folgende Datei im Editor ihrer Wahl:

```
$TOMCAT_HOME/webapps/axis2/WEB-INF/conf/axis2.xml
```

7. Setzen Sie den Parameter für Hotdeployment und Hotupdate auf true:

```
<axisconfig name="AxisJava2.0">
<!-- ================================================ -->
<!-- Parameters -->
<!-- ================================================ -->
<parameter name="hotdeployment" locked="false">true</parameter>
<parameter name="hotupdate" locked="false">true</parameter>
```

8. Starten Sie Tomcat neu.
9. Führen Sie den Client erneut aus.
10. Erweitern Sie die Methode addBuch um die Ausgabe der Methodensignatur:

```
public int addBuch(int id, String name)
{
    System.out.println("int addBuch(int id, String name)");
    System.out.println("Buch hinzugefuegt: "+name+" id: "+id);
    return 0;
}
```

11. Führen Sie ein Deployment des Buch-Services durch, indem Sie das Ant-Target deploy.service ausführen.
12. Führen Sie den Client abermals aus.
13. Betrachten sei die Tomcat-Konsole.

3. Data Binding mit XMLBeans

Übung: Einfacher Service mit XMLBeans Data Binding

1. Öffnen Sie das Projekt zoohandlungserver.
2. Betrachten Sie das folgende WSDL Dokument:

```
resources/META-INF/zoo.wsdl
```

3. Betrachten Sie die Service-Implementierung und das zugehörige Interface:

```
axis.zoo.xmlbeans.ZooHandlungServiceImpl
axis.zoo.xmlbeans.ZooHandlungSkeletonInterface
```

4. Betrachten Sie den Deployment-Descriptor des Services:

```
resources/META-INF/services.xml
```

5. Betrachten Sie wie das Ant-Target generate.service in build.xml das Bauen des Service vorbereitet und dann ein generiertes Ant-Script aufruft:

```
<target name="generate.service" depends="clean, init">

  <wsdl2java wsdlFilename="${basedir}/resources/META-INF/zoo.wsdl"
        output="${build.dir}/service"
        packageName="axis.zoo.xmlbeans"
        language="java"
        databindingName="xmlbeans"
        synconly="true" serverside="true"
        serverSideInterface="true"
        namespaceToPackages="http://thomas-
bayer.com/zoo=com.thomas_bayer.server.xmlbeans.xsd"
        generateservicexml="true" />
  <copy
file="${basedir}/src/axis/zoo/xmlbeans/ZooHandlungServiceImpl.java
" toDir="${build.dir}/service/src/axis/zoo/xmlbeans/"
overwrite="yes">
  </copy>
  <copy file="${basedir}/resources/META-INF/services.xml"
        toDir="${build.dir}/service/resources/"
        overwrite="yes">
  </copy>
  <ant dir="${build.dir}/service" />
</target>
```

6. Führen Sie das Target generate.service aus.
7. Betrachten Sie die generierten Dateien im build Verzeichnis.
8. Nehmen Sie das generierte Verzeichnis build/service/src in den Build-Path auf.
9. Fehler auf Grund von doppelten Klassen sollten jetzt angezeigt werden.

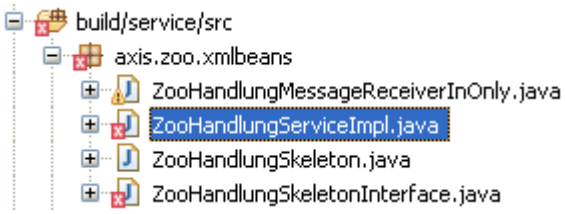

Bild6: Generierter Code für Service

10. Excludieren Sie die beiden generierten Klassen vom Build-Path.
11. Passen Sie die Ant-Property axis.services entsprechend an, um ein Deployment im Tomcat durchführen zu können.
12. Führen Sie das Ant-Target deploy.service aus.
13. Kontrollieren Sie in der Axis2-Webanwendung ob das Deployment erfolgreich war.

Tipp: Entwickeln mit Eclipse
Führen Sie nach jedem Aufruf eines Ant-Tasks der Code generiert ein Refresh auf Projektebene durch. Oder lassen Sie Eclipse automatisch einen Refresh nach dem Ausführen eines Ant-Targets durchführen.

4. Asynchrone Clients

Übung: Asynchroner Bestellen-Client

1. Wechseln Sie in das Projekt zoohandlungclient.
2. Sorgen Sie dafür, dass ein Stub erzeugt wird, der asynchrone Aufrufe ermöglicht.

```
<wsdl2java
wsdlFilename="http://localhost:8080/axis2/services/ZooHandlung?wsd
l" output="${build.dir}/client"
packageName="axis.zoo.xmlbeans.stub" databindingName="xmlbeans"
namespaceToPackages="http://thomas-
bayer.com/zoo=com.thomas_bayer.server.xmlbeans.xsd"
language="java" synconly="false" />
```

3. Betrachten Sie die generierte Callback-Klasse:

```
axis.zoo.xmlbeans.stub.ZooHandlungCallbackHandler
```

4. Erstellen Sie die folgende Klasse, die von der abstrakten Klasse ZooHandlungCallbackHandler erbt.

```
package axis.zoo.xmlbeans;

import com.thomas_bayer.zoo.BestellenResponseDocument;

import axis.zoo.xmlbeans.stub.ZooHandlungCallbackHandler;

public class ZooHandlungCallback extends
ZooHandlungCallbackHandler {

   boolean fertig;

   @Override
   public void receiveResultbestellen(BestellenResponseDocument
param2) {

      System.out.println("Ergebnis ist da");
      fertig=true;
   }

   public boolean isFertig() {
      return fertig;
   }


}
```

5. Rufen Sie anstelle von bestellen die Methode startbestellen auf und übergeben Sie eine Instanz der Klasse ZooHandlungCallback.

```
ZooHandlungCallback callback=new ZooHandlungCallback();
stub.startbestellen(bestellung, callback);
System.out.println("Bestellung verschickt");

while(!callback.isFertig()){
    Thread.sleep(100);
}
```

6. Führen Sie den Client aus.

Übung: Transport asynchroner Aufruf mit WS-Addressing

1. Wechseln Sie in das Projekt zoohandlungclient-asynclient.
2. Erweitern Sie die Klasse ZooHandlungClient um folgende Zeilen:

```
stub._getServiceClient().engageModule(new QName("addressing"));
stub._getServiceClient().getOptions().setUseSeparateListener(true)
;
```

3. Führen Sie den Client aus und betrachten Sie ggf. die Fehlermeldung.
4. Falls das addressing-Modul nicht gefunden werden konnte, können wir einen ConfigurationContext erzeugen, der ein bestehendes Repository verwendet. Wir verwenden das Repository der Binary-Distribution. Dazu erweitern wir den Client wie folgt:

```
ConfigurationContext
configContext=ConfigurationContextFactory.createConfigurationConte
xtFromFileSystem("D:\\java\\axis2-1.1.1-binary\\repository",
"D:\\java\\axis2-1.1.1-binary\\conf\\axis2.xml");

ZooHandlungStub stub=new ZooHandlungStub(configContext,
"http://localhost:2000/axis2/services/ZooHandlung");
```

Vergessen Sie nicht, die Pfade an Ihre Installation anzupassen.

5. Führen Sie den Client aus und betrachten Sie ggf. genau die Fehlermeldung.
6. Da wir Server und Client auf einem Rechner betreiben und beide den gleichen Port benutzen, kommt es zu einem Konflikt. Um das zu verhindern, ändern wir den Port für den transportReceiver in der axis2.xml auf 8081.

```
<transportReceiver name="http"
class="org.apache.axis2.transport.http.SimpleHTTPServer">
        <parameter name="port" locked="false">8081</parameter>
</transportReceiver>
```

7. Führen Sie den Client aus.
8. Betrachten Sie die Anfrage im TCP-Monitor.

Frage: Welche WS-Addressing Elemente wurden verwendet?

Frage: Warum sehen wir die Antwort des Servers nicht im TCP-Monitor?

Hinweis: Durch das Verwenden eines separaten Listeners fährt der Client einen Standalone-Server hoch, der auf Ergebnisse des Servers lauscht. Daher wird der Client nicht beim Erreichen des Endes der Main-Methode beendet.

Übung: Enumeration Kategorie

1. Erweitern Sie das Schema zoo.xsd des zoohandlungserver um das Element Kategorie

```
<complexType name="TierType">
     <sequence>
          <element name="id" type="integer"/>
          <element name="name" type="string"/>
          <element name="kategorie">
               <simpleType>
                    <restriction base="string">
                         <enumeration value="Vogel"/>
                         <enumeration value="Reptil"/>
                         <enumeration value="Insekt"/>
                    </restriction>
               </simpleType>
          </element>
     </sequence>
</complexType>
```

2. Führen Sie das Ant-Target deploy.service aus.
3. Betrachten Sie das WSDL Dokument des Services.
4. Geben Sie die Kategorie in der Service-Implementierung aus.
5. Wechseln Sie zum Projekt zoohandlungclient.
6. Führen Sie das Ant-Target generate.client aus.
7. Erweitern Sie den Client um die Angabe einer Kategorie:

```
tier.setKategorie(TierType.Kategorie.REPTIL);
```

8. Führen Sie den Client aus und betrachten Sie die Ausgabe in der Tomcat-Konsole.

6. Handler

Übung: LogHandler

1. Erstellen Sie eine Klasse LogHandler, die von AbstractHandler erbt.

```
package com.thomas_bayer.axis2;

import org.apache.axis2.AxisFault;
import org.apache.axis2.context.MessageContext;
import org.apache.axis2.handlers.AbstractHandler;

public class LogHandler extends AbstractHandler {

   public InvocationResponse invoke(MessageContext ctx) throws
  AxisFault {

      System.out.println(ctx.getEnvelope().toString());

      return InvocationResponse.CONTINUE;
   }

}
```

2. Editieren Sie die Konfigurationsdatei axis2.xml. Erweitern Sie die OperationInPhase im InFlow um ein Handler-Element:

```
    <phaseOrder type="InFlow">
...
        <phase name="OperationInPhase">
            <handler name="LogHandler"
                     class="com.thomas_bayer.axis2.LogHandler">
                <order phase="OperationInPhase" />
            </handler>
        </phase>

    </phaseOrder>
```

3. Führen Sie einen Client aus und betrachten Sie die Konsole.