

Knowledge - Java

Table of contents

1 JMock und Unittesting.....	2
2 Pattern in Java.....	2
2.1 Singleton.....	3
2.2 Visitor.....	3
3 RMI-Applikation unter Java 5.....	4
3.1 1. RMI-Interfaces definieren.....	4
3.2 2. Servant-Klasse erstellen.....	5
3.3 3. RMI Server erstellen.....	5
3.4 4. RMI-Client erstellen.....	6
4 XML Webservice aufrufen und parsen.....	8
4.1 Verbindung zum Webservice aufbauen und den SAX-Parser starten.....	8
4.2 Der SAX-EventHandler.....	9
5 XML – XSL – Transformation in Java.....	10
6 AXIS Webservice unter Tomcat.....	10
6.1 Lokaler Test-Server.....	10
6.2 Deploying bzw. Einrichten eines WebServices.....	11
6.3 Erstellung Client-Stubs.....	14
6.4 WebServices und Sicherheit.....	14

1. JMock und Unittesting

Zu JUnit möchte an dieser Stelle nichts verlieren, denn zu diesem Thema gibt es ja bereits genug zu lesen und somit keiner weiteren Erklärung. Anders sieht bei dem Framework JMock aus, welches noch nicht so verbreitet ist. Dieses ermöglicht es Serverabhängigkeiten während der Unittests durch das einfache Erstellen von Mock-Objekten zu lösen. Siehe hier zu auch das folgende Beispiel und die JMock-Projekt-Website ([link](http://www.jmock.org/) (http://www.jmock.org/)).

```
package com.aperto.moduler.utils;

import com.day.cq.contentbus.Page;
import org.jmock.MockObjectTestCase;
import org.jmock.Mock;

/**
 * Test der HandleTools.
 * @author frank.sommer (28.02.2007)
 */
public class HandleToolsTest extends MockObjectTestCase {
    protected void setUp() throws Exception {
        super.setUp();
    }

    protected void tearDown() throws Exception {
        super.tearDown();
    }

    public void testGetNavigationLevel() {
        // set up
        Mock mockPage = mock(Page.class);
        Page page = (Page) mockPage.proxy();
        // expectations & stubs
        mockPage.stubs()
            .method("getHandle")
            .will(returnValue("/content/sitecontent/bu/portal/de"));
        // execute
        int navigationLevel = HandleTools.getNavigationLevel(page);
        assertEquals(0, navigationLevel);
    }
}
```

2. Pattern in Java

In meinem nebenstudentischen Berufsleben trifft man immer wieder auf Vertreter der Entwurfsmuster, die ich bislang nur theoretisch und unzureichend im Studium kennengelernt habe. Hier nun sollen einige Pattern unter Java vorgestellt werden.

2.1. Singleton

Singleton ist ein Klasse von welcher zur Laufzeit immer maximal eine Instanz existiert. Eine Anwendung hierfür sind zum Beispiel Services. Ein bekannte Singleton-Vertreter in Java ist die Random-Klasse.

```
//Singleton (lazy initializing and thread safe)
public class SingletonExample {
    private static SingletonExample c_instance;

    public static SingletonExample getInstance() {
        if (c_instance == null) {
            synchronized(SingletonExample.class) {
                if (c_instance == null) {
                    c_instance = new SingletonExample();
                }
            }
        }
        return c_instance;
    }

    private SingletonExample() {
    }

    //some methods...
}
```

2.2. Visitor

Das Visitor-Pattern ermöglicht eine Methode einer Klassenstruktur zentral in eine eigene Klasse abzulegen. Dies kann zum Beispiel nötig sein, wenn der Zugriff auf die Klassenstruktur nicht gegeben ist oder um den Änderungsaufwand zu verringern. Jedoch lohnt sich ein Visitor-Pattern nur, wenn sich die Klassenstruktur weniger häufig als die Methodenanzahl ändert.

visitor-pattern

```
//Visitor
public class Pet {
    public void accept(ExampleVisitor visitor) {
        visitor.visit(Pet pet);
    }
}

//inherited classes
...
```

```

public interface ExampleVisitor {
    void visit(Pet pet) ;
    void visit(Dog dog) ;
    void visit(Cat cat) ;
    void visit(Rabbit rabbit) ;
}

public class ExampleVisitorImpl implements ExampleVisitor {

    public void visit(Pet pet) {
        pet.eat("gras");
    }

    public void visit(Dog dog) {
        dog.eat("meat");
    }

    public void visit(Cat cat) {
        cat.eat("milk");
    }

    public void visit(Rabbit rabbit) {
        cat.eat("carrot");
    }
}

public class ExampleVisitorUser {

    ExampleVisitorImpl _visitor = new ExampleVisitorImpl();

    public void useVisitor(Pet anyPet) {
        anyPet.accept(_visitor);
    }
}

```

3. RMI-Applikation unter Java 5

Mein Beispiel ist ein kleines verteiltes Gästebuch ohne Persistenz.

3.1. 1. RMI-Interfaces definieren

Das Interface ist von Remote abgeleitet. Jede Methode muss eine RemoteException werfen.

```

import java.rmi.Remote;
import java.rmi.RemoteException;
/**
 * @author Frank Sommer
 * Interface für das Gästebuch
 */
public interface IF_GB extends Remote {

```

```
public String sendAEntry(int index) throws RemoteException;
public boolean saveAEntry(String entry) throws RemoteException;
public int sendCount() throws RemoteException;
}
```

3.2. 2. Servant-Klasse erstellen

Diese Klasse implementiert das Interface und für die Methoden mit Logik.

```
import java.util.ArrayList;

/**
 * @author Frank Sommer
 * Servant-Klasse implementiert das Interface
 */
public class GB implements IF_GB {

    private ArrayList<String> entryList;

    public String sendAEntry(int index) {
        String entry = "";
        entry = entryList.get(index);
        return entry;
    }

    public boolean saveAEntry(String entry) {
        entryList.add(entry);
        return true;
    }

    public int sendCount() {
        return entryList.size();
    }

    public GB() {
        super();
        String welcome = "Willkommen im RMI-Gästebuch!";
        entryList = new ArrayList<String>();
        entryList.add(welcome);
    }
}
```

3.3. 3. RMI Server erstellen

Da ich Java 5 benutzte, braucht man nicht explizit die Stubs bzw. Skeletons erstellen. Das starten der RMI Registry wird ebenfalls vom Server übernommen. Die Security-Einstellungen können auch mitunter weggelassen werden, das kommt ganz auf das System an.

```
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;
```

```

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.*;

/**
 * @author Frank Sommer
 */
public class GBServer {

    /**Konstante für den Server-Port*/
    public final static int PORT = 5555 ;

    public static void main(String[] args) {
        try {
            GB obj = new GB();
            IF_GB stub = (IF_GB) UnicastRemoteObject.exportObject(obj, 0);

            //Policy-File in Project-Root
            System.setProperty("java.security.policy", "policy");
            if (System.getSecurityManager() == null) {
                System.setSecurityManager(new RMISecurityManager());
            }

            try {
                LocateRegistry.createRegistry( PORT ); //starten der Registry
                Registry registry = LocateRegistry.getRegistry( PORT ); // Bind the
remote object's stub in the registry
                registry.rebind("guestbook", stub); //rebind überschreibt gleichnamige
Services, nicht so bind
                System.out.println("Server ready");
            }
            catch ( RemoteException e ) {
                System.err.println("Konnte Registry nicht starten!");
            }
            catch (Exception e) {
                System.err.println("Server exception: " + e.toString());
                e.printStackTrace();
            }
        }
    }
}

```

Nun kann der Server kompiliert und gestartet werden.

3.4. 4. RMI-Client erstellen

Der Client gibt dann die Beiträge aus bzw. schickt sie an den Server. Bei meinem Client handelt es um einen simplen Konsolen-Client.

```

import java.rmi.RemoteException;

import uebung2.gbserver.IF_Constants;

/**

```

Knowledge - Java

```
* @author Frank Sommer
* Gaestebuch-Client
*/
public class GBClient implements IF_Constants {

    public static void main(String[] args) {

        short command = 0; //Kommandos sind in der IF_Constants definiert
        String entry = "";

        GBOutput gbo; //Klasse für den Output des Gästebuchs
        try {
            gbo = new GBOutput();

            GBMenue gbm = new GBMenue(); //Klasse, welche das Menü darstellt und
            verarbeitet
            gbm.printHelp();

            do {
                command = gbm.readCommand();
                switch (command) {
                    case ERROR:
                        System.out.println("Falsche Eingabe");
                        break;
                    case DOWNLOAD:
                        try {
                            gbo.printEntries();
                        } catch (RemoteException e) {
                            System.out.println("Fehler beim Lesen vom Server!");
                        }
                        break;
                    case UPLOAD:
                        entry = gbm.readEntry();
                        if (entry.length() < 2) {
                            System.out.print("Fehler beim Einlesen\n");
                        } else {
                            try {
                                gbo.writeEntry(entry);
                            } catch (RemoteException e) {
                                System.out.println("Fehler beim Schreiben auf Server!");
                            }
                        }
                        break;
                    default:
                        command = ENDE;
                }
            } while (command != ENDE);
        } catch (Exception e) {
            System.out.println("Sorry, kann keine Connection zum Server
            herstellen.");
        }
    }
}
```

```
}

```

Der komplette Quellcode kann auch an dieser Stelle heruntergeladen werde. [Quellcode](http://stuff.fts-online.de/gb-rmi.zip) (<http://stuff.fts-online.de/gb-rmi.zip>)
 Weitere Informationen zu RMI gibt auch bei [SUNs RMI Seite](http://java.sun.com/products/jdk/rmi/) (<http://java.sun.com/products/jdk/rmi/>) .

4. XML Webservice aufrufen und parsen

In diesem Abschnitt zeige ich den Verbindungsaufbau mit dem [HTTPClient](http://jakarta.apache.org/commons/httpclient/) (<http://jakarta.apache.org/commons/httpclient/>) von Apache und das anschließende Parsen des XML, welches der Webservice lieferte.

4.1. Verbindung zum Webservice aufbauen und den SAX-Parser starten.

1. Übergabeparameter festlegen, die mittels POST übertragen werden.

```
//Übergabeparameter setzen
NameValuePair[] data = {
    new NameValuePair("eventid", id)
};

```

2. Client instanziiieren und Konfiguration setzen wie Proxy.

```
// Create an instance of HttpClient.
HttpClient client = new HttpClient();

//TODO: Proxysachen bei Übertragung entfernen
ProxyHost proxy = new ProxyHost("proxy.de.test",8080);

HostConfiguration hconfig = new HostConfiguration();
hconfig.setProxyHost(proxy);

client.setHostConfiguration(hconfig);

client.getState().setProxyCredentials(
    new AuthScope(AuthScope.ANY),
    new NTCredentials("user", "password", "proxy.de.test", "domain")
);

```

3. POST-Methode wählen und Daten setzen

```
// Create a method instance.
PostMethod method;
try {
    method = new PostMethod(configuration.getUrl());

    method.setRequestBody(data);

    // Provide custom retry handler is necessary
    method.getParams().setParameter(HttpMethodParams.RETRY_HANDLER,
        new DefaultHttpClientRetryHandler(3, false));
}

```

```
    ...
} catch (DataException ex2) {
    throw new ServiceException("URL in ConfigurationData isn't set.");
    //ex2.printStackTrace();
}
```

4. Die entfernte Webseite aufrufen und Status abfragen.

```
// Execute the method.
int statusCode = client.executeMethod(method);

if (statusCode != HttpStatus.SC_OK) {
    //System.err.println("Method failed: " + method.getStatusLine());
    logger.warn("http status isn't ok.");
    return false;
}
```

5. Nun wird der Stream von der Webseite an ein Objekt übergeben, welche mittels des SAX-Parsers das XML-Dokument filtert und die relevanten Daten somit herausfischt. Der SAX-Parser arbeitet ereignisgesteuert. Das hat den Vorteil gegenüber DOM-Parsern, dass das XML-Dokument nicht komplett in den Speicher geladen werden muss.

```
SAXEvent handler = new SAXEvent(pLong);
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser;
try {
    saxParser = factory.newSAXParser();
    try {
        saxParser.parse( method.getResponseBodyAsStream(), handler );
        status = true;
    }
    catch (IOException ex1) {
        //ex1.printStackTrace();
        logger.warn("Error by access the inputstream in travel-event");
    }
}
catch (ParserConfigurationException ex) {
    logger.warn("Error by configuration the SAX-parser in travel-event");
    //ex.printStackTrace();
}
catch (SAXException ex) {
    logger.warn("Error by parsing the xml in travel-event");
    //ex.printStackTrace();
}
```

6. Und zu guterletzt noch die Verbindung beenden.

```
// Release the connection.
method.releaseConnection();
```

4.2. Der SAX-EventHandler

Wie schon erwähnt arbeitet der SAX-Parser ereignisgesteuert. Um auf diese Ereignisse gewünscht zu reagieren, muss man eine Klasse schreiben die von der Klasse

DefaultHandler abgeleitet ist. Nun muss man nur noch die gewünschten Event-Methode neu definieren. Solche Events sind z.B. Start des Dokumentes oder ein öffnendes XML-Tag.

5. XML – XSL – Transformation in Java

Als erstes muss ein StreamResult-Objekt erzeugt werden, welches den Writer als Parameter hat, der die Ausgaben in den HTML-Code generiert. Über die schon bekannte Connection wird das XML-File eingelesen. Das XSL-File wird aus einer Datei eingelesen. Über eine TransformerFactory und einen Transformer wird der transformierte Stream, direkt an den Output geleitet.

```
htmlOutput = new StreamResult(Output);
xslFile = xsl;

// URL
URL myURL;
URLConnection connection;

myURL = new URL(strUrl);
connection = myURL.openConnection();

TransformerFactory xslFactory;
Source xmlSrc = null;

//XML Quelle einlesen
xmlSrc = new StreamSource(connection.getInputStream());

// Exemplar der Factory holen
xslFactory = TransformerFactory.newInstance();

//XSL-Quelle holen
Source xslSrc = new StreamSource(xslFile);

// Transformer erzeugen
Transformer trans = xslFactory.newTransformer(xslSrc);

trans.transform(xmlSrc, htmlOutput);
```

6. AXIS Webservice unter Tomcat

6.1. Lokaler Test-Server

6.1.1. Tomcat

Es bietet sich an für Tests und auch zur Einarbeitung sich den tomcat lokal zu installieren. Dies läßt sich leicht bewerkstelligen, da die Installation recht einfach verläuft. Man muss nur die Installationsdatei herunterladen und ausführen. [Download-Link](#)

(http://jakarta.apache.org/site/downloads/downloads_tomcat.html)

Note:

Mit dem Tomcat-Server 5.0 lief es einwandfrei. Bei Tomcat 4 muss man ein paar Anpassungen im AXIS bezüglich der Java-Bibliotheken vornehmen.

6.1.2. AXIS

Um AXIS unterhalb des Tomcat-Servers zu betreiben, muss man sich zuerst AXIS von [hier](http://ws.apache.org/axis/releases.html) (<http://ws.apache.org/axis/releases.html>) herunterladen. Das Archiv entpackt man dann an eine beliebige Stelle. Aus den entpackten Verzeichnissen kopiert man nun das axis-Verzeichnis unter das webapps-Verzeichnis des Tomcat-Servers.

Note:

Voraussetzung für AXIS ist mind. Java 1.3

6.1.3. Testszenario

Wenn der Tomcat-Server gestartet wurde, kann AXIS auf seine Funktionalität getestet werden.

Note:

Die nachfolgenden Links beziehen sich auf die lokale Standard-Installation. Bei Abweichungen muss die URL angepasst werden.

Als erstes sollte die Startseite verfügbar sein: [TEST](http://127.0.0.1:8080/axis/). (<http://127.0.0.1:8080/axis/>)

Nun sollte die Installation geprüft werden. Entweder du folgst dem Validate-Link auf der Startseite oder Du klickst [TEST](http://127.0.0.1:8080/axis/happyaxis.jsp). (<http://127.0.0.1:8080/axis/happyaxis.jsp>) Die Seite zeigt alle benötigten Komponenten an, so dass diese noch eingestellt werden können.

Auf der AXIS-Seite sind noch [weitere Tests](http://ws.apache.org/axis/java/install.html#LookForSomeServices) (<http://ws.apache.org/axis/java/install.html#LookForSomeServices>) vorgeschlagen, die hier jedoch nicht weiter erwähnt werden sollten.

6.2. Deploying bzw. Einrichten eines WebServices

6.2.1. Automatisches Deploying

Bei automatischen Deployen nimmt man die Quellcode-Datei (*.java) seines Webservices und stellt diese auf den Webserver (unterhalb von AXIS) mit der Dateiendung jws. Wenn

man diese jws-Datei aufruft wird der Webservice automatisch eingerichtet. Jedoch empfiehlt sich diese Art des Deploying nur für Testzwecke, da es schnell zu Problemen kommen kann. Ich hatte zum einen Probleme bei der Verwendung mehrerer Klassen, wie es normalerweise unter Java üblich ist. Dafür mussten die kompilierten Klassen unter classes abgelegt werden. Jedoch weigerte er sich die Paketstruktur anzuerkennen. Desweiteren gibt es Problemen mit komplexen Datentypen.

Ein weiterer Nachteil ist, dass immer der Quellcode vorliegen muss, damit man einen Webservice einrichten kann.

6.2.2. Manuelles Deploying

Hierfür benötigt man nicht die Quellcodes wie im automatischen Deployen, sondern die kompilierten Klassendateien. Diese sind in das Verzeichnis `axis/WEB-INF/classes` abzulegen. Die Paketstruktur muss in der Verzeichnisstruktur abgebildet sein. Liegen die Klassen als Jar-File vor muss dieses wie weitere Bibliotheken in das lib-Verzeichnis abgelegt werden.

Für das manuelle Deploying muss man nun eine WSDO -Datei erzeugen. Diese ist eine Datei in XML-Syntax, mit welcher der Webservice eingerichtet wird. Über diese Datei werden auch Einstellungen zur Serialisierung von eigenen Java-Klassen (komplexe Datentypen) AXIS vermittelt. Dies war in meinem Fall ebenfalls notwendig.

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="Addresses" provider="java:RPC">
    <requestFlow>
      <handler type="soapmonitor"/>
    </requestFlow>
    <responseFlow>
      <handler type="soapmonitor"/>
    </responseFlow>
    <parameter name="className"
value="de.test.office.webservices.Addresses"/>
    <parameter name="allowedMethods" value="*/>
    <beanMapping qname="myNS:Address" xmlns:myNS="urn:Addresses"
languageSpecificType="java:de.test.office.webservices.Address"/>
    <beanMapping qname="myNS:SomeAddresses" xmlns:myNS="urn:Addresses"
languageSpecificType="java:de.test.office.webservices.SomeAddresses"/>
  </service>
</deployment>
```

Diese Datei wird dann auf den Server unter AXIS kopiert und mittels starten eines Java-Programms wird der Webservice eingerichtet.

```
>> java org.apache.axis.client.AdminClient Addresses.wsdd
Processing file Addresses.wsdd
<Admin>Done processing</Admin>
>>
```

6.2.2.1. De-/Serializer

Bei WeBservices werden die Java-Datentypen auf einheitliche XML-Datentypen abgebildet. Dieser Prozess nennt sich auch Serialisierung bzw. Deserialisierung, je nachdem aus welcher Richtung man die Abbildung betrachtet. Dies ist unter anderem notwendig, um eine Plattformunabhängigkeit zu erreichen.

Java-Datentyp	XML-Datentyp
java.lang.String	xsd:string
int	xsd:int
float	xsd:float

Table 1: Auszug der Java-Datentyp-Zuordnung

Desweiteren ist es auch kein Problem für AXIS Felder von Basisdatentypen zu serialisieren. Ein Sonderfall für die Serialisierung sind sogenannte Beans. Hierfür bringt AXIS schon BeanSerializer mit. Jedoch muss für die Verwendung dieser das WSDD- Dokument erweitert werden. (siehe [hier](#))

Note:

Beans sind Klassen mit einer gewissen Struktur. Zu jedem Attribut gibt es eine get- und eine set-Methode. Zum Beispiel value, setValue(...) und ... getValue().

Wenn Beans nicht mehr ausreichen sollten, müssen zur Serialisierung eigene Serializer bzw. Deserializer programmiert und implementiert werden. Dazu siehe auch [hier](#) (<http://ws.apache.org/axis/java/user-guide.html#WhenBeansAreNotEnoughCustomSerialization>)

6.2.3. Undeployment

Beim automatischen Deployment löscht man einfach die erstellte Klassendatei und die jws-Datei.

Beim manuellen Undeployment ist es ähnlich wie beim Deployment. Man ruft wieder den AdminClient auf, nun jedoch mit einem Undeployment-Descriptor.

```
<undeployment xmlns="http://xml.apache.org/axis/wsdd/">  
  <service name="Addresses"/>  
</undeployment>
```

Das ständige kopieren und deployen lässt übrigens mittels des Tools [Ant](#) (<http://de.wikipedia.org/wiki/Ant>) automatisieren. (siehe [hier](#))

6.3. Erstellung Client-Stubs

Client-Stubs sind anhand einer WSDL-Datei generierte Java-Klassen, welche den Zugriff auf den Webservice erleichtern sollen. Die automatische Generierung hat jedoch nicht nur Vorteile. Durch die automatische Generierung werden komplexe Datentypen, so gemäß Java-Bean-Konventionen generiert. Das heißt, dass es zu jedem Attribut eine Setter- und Getter-Methode gibt und der Zugriff nur darüber stattfindet. Daher wird angeraten nach der fertigen Entwicklung des Webservice passende Wrapper-Klassen zu entwickeln, welche die Zugriffe zusammenfassen und somit vereinfachen.

6.3.1. Generierung aus dem WSDL

Die Generierung kann entweder über die Oberfläche in Eclipse, vorausgesetzt das PlugIn Lombok oder Spotter ist installiert, geschehen oder mittels Kommandozeile. Für die Kommandozeile ist es ratsam sich eine kleine Batchdatei zu schreiben. Diese Batchdatei kann zum Beispiel auch direkt auf die Web-URL verweisen. Damit wird immer das aktuelle WSDL-Dokument benutzt. Voraussetzung ist, dass kein Proxy einem den Zugriff erschwert, dann sind noch ein paar mehr Anpassung notwendig.

```
set CLASSPATH= D:\\Programme\\Java\\j2sdk1.4.2_06;.
set CLASSPATH= %CLASSPATH%;.\\lib\\activation.jar
set CLASSPATH= %CLASSPATH%;.\\lib\\mail.jar
set CLASSPATH= %CLASSPATH%;.\\lib\\xerces.jar
set CLASSPATH= %CLASSPATH%;.\\lib\\soap.jar
set CLASSPATH= %CLASSPATH%;.\\lib\\jaxrpc.jar
set CLASSPATH= %CLASSPATH%;.\\lib\\axis.jar
set CLASSPATH= %CLASSPATH%;.\\lib\\commons-discovery.jar
set CLASSPATH= %CLASSPATH%;.\\lib\\commons-logging.jar
set CLASSPATH= %CLASSPATH%;.\\lib\\log4j-1.2.8.jar
set CLASSPATH= %CLASSPATH%;.\\lib\\saa.jar
set CLASSPATH= %CLASSPATH%;.\\lib\\wsdl4j.jar

java -cp %CLASSPATH% org.apache.axis.wsdl.WSDL2Java -p clientstubs -o src
.\\addresses.wsdl
```

6.4. WebServices und Sicherheit

6.4.1. Schwachstellen bei XML und Internetkommunikation

Die Schwachstellen die es bei der Verarbeitung von XML gibt, betreffen natürlich auch SOAP, da es XML-konform aufgebaut ist.

1. Große XML Dokumente

Bei der Verarbeitung von großen XML-Dokumenten mit DOM-Parsern kann es leicht zu Speicherengpässen auf den Servern kommen (DoS Attack). Lösungsmöglichkeit wäre die

Verwendung von SAX als Parser.

2. Referenzen innerhalb von XML auf Entities
Referenzen innerhalb von XML, entweder auf sich selbst oder das Dateisystem können ebenfalls zu Problemen führen. Gegen solche Probleme ist AXIS jedoch immun, da es keine Entities unterstützt.
3. Übernahme der Session
Manipulation des Systems kann auch entstehen, wenn die Session-ID übernommen wurde und nun falsche Nachrichten an die Sitzung geschickt werden. Dies kann z. B. durch Speicherung von zusätzlichen Informationen (IP-Adresse etc.) zum Teil vermieden werden.
4. SQL Injection
Parameter welche aus dem XML benutzt werden, um sie in einer Datenbankabfrage zu benutzen, müssen vor der Benutzung geprüft werden. Zum Beispiel läßt sich durch ein zusätzliches Semikolon ein weiterer SQL-Befehl absetzen, welcher die gleichen Zugriffsrechte besitzt wie der Webservice. Lösungsansatz ist die Länge und den Inhalt (regular expressions, only characters) zu prüfen. Benutze Prepared Statements, denn dabei escaped die JDBC-Runtime irgendwelche Sonderzeichen. Baue die SQL-Befehle nicht per Hand zusammen.

6.4.2. AXIS-System Sicherheitsvorkehrungen

Es gibt je nach Anforderungen verschiedene Vorkehrungen, die getroffen werden können, um die Kommunikation und somit die Daten vor dem Zugriff anderer zu sichern.

Detailliertere Angaben finden sich [hier](http://ws.apache.org/axis/java/security.html) (<http://ws.apache.org/axis/java/security.html>)

1. HTTPS und HTTP authentication
Eine Lösung wäre HTTPS. Hierbei geschieht die Verschlüsselung oberhalb von AXIS. Hierfür ist lediglich eine Konfiguration des Servers und des Clients notwendig. Der Webservice bleibt unangetastet. Zusätzlich könnte eine Basic Athenication über HTTP statttfinden.
2. AXIS verstecken
Bei allen Antworten von AXIS sollte im Header ein Verweis auf AXIS entfernt werden. Dafür ist jedoch ein bearbeiten des Quellcodes notwändig. Damit soll vermieden werden, dass bekannte Bugs innerhalb AXIS nicht ausgenutzt werden. Desweiteren sollten bekannte Anlaufpunkte wie der AdminService umbenannt werden.
3. Filterung für Extra-Authentifizierung
Wenn nur bestimmte User auf den Webservice zugreifen sollen, können diese durch bestimmte Filterkriterien (IP-Adresse, caller credentials, etc.) zugelassen werden. Dies kann über Firewalls oder im Servlet direkt geschehen.

6.4.3. WS-Security

WS-Security ist einer von der OASIS entwickelter Standard zur Sicherung von Webservices. Für Java und AXIS gibt es eine Implementation dieses Standards (WSS4J). Weitere Informationen dazu [hier](http://ws.apache.org/wss4j/) (http://ws.apache.org/wss4j/).

Note:

WSS4J benutzt das Apache XML Security Project, um XML Security (XML Signature and XML Encryption) einzubinden. WSS4J Axis handlers benötigen Axis V1.2, da es einige Probleme mit früheren Axis-Versionen gab.

6.4.3.1. Installation von WSS4J

1. Download die Binaries oder Kompiliere die Sourcen.
2. Kopiere die Jar-Files (siehe README.txt in WSS4J.zip) in das WEB-INF/lib-Verzeichnis unter AXIS
3. XML-Security muss ebenfalls verfügbar sein, siehe happyaxis.jsp

6.4.3.2. UsernameToken

Die einfachste Implementierung innerhalb WSS4J ist die Einstellung eines Passwortes. Dieser wird in Klartext bzw. im "Digest Mode" verschickt und bietet somit keine wirkliche Sicherheit. Ein Einführungstutorial zu diesem Thema ist [hier](http://ws.apache.org/wss4j/) (http://ws.apache.org/wss4j/) zu finden.