Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Entwicklung eines Provisions Service mit einer Web Service-Schnittstelle

Judith Rüesch

Semesterarbeit

Departement Informatik
Wintersemester 2006/2007

Professor: David Basin
Betreuer: Paul Sevinç

Inhaltsverzeichnis

Kapitel 1

Einleitung

Ein Access Control System kontrolliert den Zugriff auf Objekte. Die Entscheidung, ob der Zugriff erlaubt ist oder nicht, wird vom Policy Decision Point (PDP) gefällt. Anhand der Policy und verschiedenen Services fällt er die Entscheidung über die Zugriffsberechtigung. Der Provisions Service (PS), der in dieser Semesterarbeit implementiert wird, ist ein Service, welcher den Policy Decision Point unterstützt, diese Entscheidung zu treffen. Dazu verwaltet der PS in einer Datenbank die Provisions, welche von einem User getroffen werden. Der PDP liest aus der Policy jene Provisions aus, welche ein User getroffen haben muss, um die Erlaubnis für den Zugriff auf ein Objekt zu erhalten. Der PDP erhält durch die Verwendung des Provisions Service die Information, ob der User die nötigen Provisions getroffen hat.

Der Provisions Service kann als Web Service genutzt werden und wird in Java implementiert. Die Vorteile von Web Services sind, dass sie plattform- und sprachenunabhängig sind. Da die gesendeten Daten sensitiv sind, müssen Sicherheitsanforderungen getroffen und implementiert werden.

1.1 Kapitelübersicht

Das zweite Kapitel beschreibt den Nutzen eines Provisions Service, die benötigten Funktionalitäten, sowie die Sicherheitsanforderungen die im Provisions Service implementiert werden müssen. Zudem betrachten wir die Merkmale von Web Services.

Im dritten Kapitel wird die Spezifikation des Provisions Service als Web Service erklärt und das vierte Kapitel enthält das Entwicklerhandbuch. Darin werden die verwendeten Tools beschrieben, sowie die Vorgehensweise bei der Implementation des Provisions Service. Am Schluss folgt eine Zusammenfassung mit Erläuterungen zur Entwicklung des Provisions Service.

Kapitel 2

Hintergrund

2.1 Provisions Service

Ein Provisions Service (PS) ist ein Teil eines Access Control Systems (ACS), welches den Zugriff auf Objekte kontrolliert. Die Erlaubnis für den Zugriff auf ein Objekt ist bei einem ACS, welcher ein PS beinhaltet, abhängig von Provisions.

2.1.1 Provisions

Eine Provision ist eine Vorkehrung, die ein User treffen muss, damit er auf ein Objekt zugreifen kann. Die Vorkehrung muss vom User vor dem ersten Zugriff auf das Objekt getroffen werden. Das System speichert die getroffenen Vorkehrungen ab. Als Beispiel wollen wir zwei Provisions betrachten.

- **Sign:** Der User muss vor dem Zugriff auf ein Objekt ein Abkommen digital unterschreiben. Damit wird erreicht, dass das ACS Dritten gegenüber beweisen kann, dass der User dem Abkommen zugestimmt hat.
- **Agree:** Bevor der User auf ein Objekt zugreifen kann, muss er einer Vereinbarung zustimmen. Dies kann zum Beispiel umgesetzt werden, in dem ein User durch das Drücken eines Agree-Buttons einer Lizenz zustimmt. Dem ACS ist es jedoch nicht möglich, einer dritten Partei gegenüber zu beweisen, dass der User der Vereinbarung zugestimmt hat.

Dies sind zwei mögliche Provisions, welche im Projekt auch implementiert werden. Beliebige Provisions sollen jedoch hinzugefügt werden können.

2.1.2 Access Control System

Abbildung 2.1 zeigt anhand eines Beispiels, wie ein PS im ACS eingebettet ist. Bei den Objekten handelt es sich in diesem ACS um Dokumente. Darum wird im Folgenden von Dokumenten die Rede sein.

Der Policy Enforcement Point (PEP) im ACS sorgt für die klare Trennung zwischen User und Dokument. Möchte ein User auf ein Dokument zugreifen (1), so wertet der Policy Decision Point (PDP) die Anfrage aus (2) und entscheidet, ob der Zugriff erlaubt ist oder nicht. Dazu liest der PDP die Provisions aus der Policy aus, die für den Zugriff auf das Dokument getroffen sein müssen. Mit

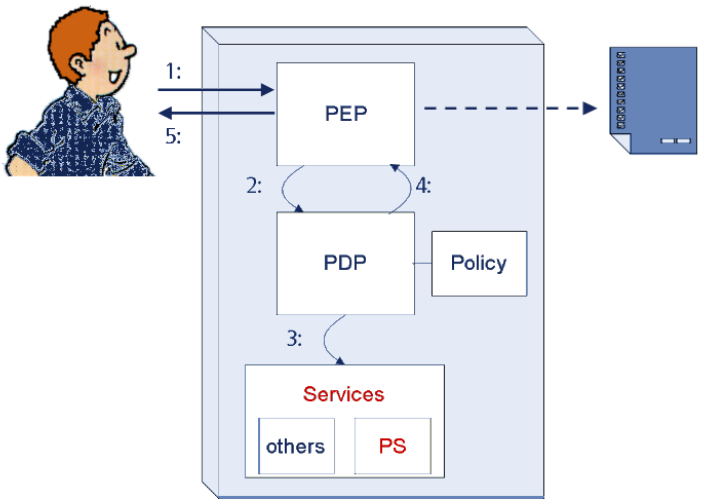

Abbildung 2.1: Access Control System

Hilfe des Provisions Service (3) entscheidet er, ob der Zugriff erlaubt ist oder nicht. Diese Entscheidung wird dem PEP mitgeteilt (4), welcher dem User den Zugriff entsprechend verweigert bez. gewährt (5).

2.1.3 Implementation des Provisions Service

Die Web Service-Schnittstellen des Provisions Service umfasst zwei Funktionen: `filterMadeProvisions` und `addMadeProvision`. Aus Sicht des PDP ist nur die Erste relevant.

filterMadeProvisions

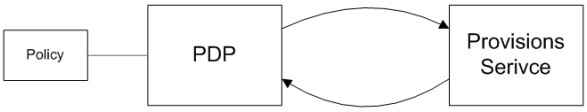

Abbildung 2.2: Implementation des Provisions Service

Wie die Abbildung 2.2 zeigt, liest der PDP aus der Policy die zu treffenden Provisions aus. Diese Provisions sendet er an den Provisions Service. Die Funktion filtert aus dieser Menge alle Provisions raus, die bereits getroffen sind und sendet die übrigen zurück. Erhält der PDP eine leere Menge zurück, so weiss er, dass der Zugriff erlaubt werden kann. Ansonsten muss der Zugriff verweigert werden.

Aus Sicht des Web Service entspricht der PDP dem Client und der Provisions Service dem Server.

addMadeProvision

Die zweite Funktion die zu implementieren ist, fügt Provisions in die Datenbank ein. Bevor eine Provision eingefügt wird, soll überprüft werden, ob die Vorkehrung auch wirklich getroffen worden ist.

2.1.4 Sicherheitsanforderungen

Die Informationen, die zwischen dem PDP und dem Service ausgetauscht werden sind sensitiv, da Dritte anhand dieser Daten herausfinden können, auf welche Objekte ein User zugreifen möchte. Um das Problem zu lösen, sollen sinnvolle Sicherheitsvorkehrungen getroffen und implementiert werden.

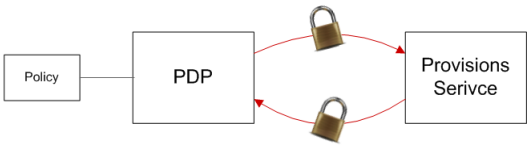

Abbildung 2.3: Sicherheitsvorkehrungen treffen

Client - Server

Niemand darf den Inhalt der Nachricht vom Client zum Server lesen können. Wäre das Lesen der Nachricht für einen Aussenstehenden möglich, so wüsste er, auf welche Objekte der User zugreifen möchte. Dies muss vermieden werden. Wir benötigen also Vertraulichkeit (Confidentiality).

Damit der Provisions Service nicht umgangen werden kann, wollen wir auch Integrität erreichen. Die Nachricht, welche der Client abschickt, muss unverändert beim Server ankommen. Wäre das Verändern der Nachricht möglich, so könnte ein Angreifer alle Provisions aus der Nachricht löschen. Dies hätte zur Folge, dass immer eine leere Menge an den PDP zurückgesendet wird und darum der Zugriff auf das gewünschte Objekt immer erlaubt ist, unabhängig davon, ob die Provisions getroffen sind oder nicht.

- Vertraulichkeit wird durch Verschlüsselung der Nachricht erreicht.
- Integrität wird durch Signieren der Nachricht erreicht.

Server - Client

Die Kommunikation vom Server zum Client muss auch vertraulich sein, denn auch aus diesen Informationen können Dritte ausfindig machen, auf welche Objekte ein User zugreifen möchte.

Der Client möchte sicher sein, dass er mit dem richtigen Service kommuniziert, da er nur diesem vertraut, dass es die Auswertung richtig vornimmt. Wir benötigen darum Authentizität.

- Vertraulichkeit wird durch Verschlüsselung der Nachricht erreicht.
- Authentizität wird durch Signieren der Nachricht erreicht.

2.2 Web Service

2.2.1 Definition

Das World Wide Web Konsortium (W3C) versteht unter Web Services (WS) folgendes: Ein Web Service ist ein Software-System, welches sich durch eine URI [RFC 2396] auszeichnet und deren öffentliches Interface durch die Verwendung von XML definiert und beschrieben ist. Deren Definition kann von anderen Software-Systemen entdeckt werden. Diese Systeme können daraufhin mit dem Web Service, durch das Versenden von XML-basierten Nachrichten über Internet Protokolle interagieren, wie dies durch die Definition vorgeschrieben ist. [16]

Diese Definition sagt aus, dass der Zugriff auf den Service klar definiert ist. Jedem, der den Service nutzen möchte, ist es dadurch möglich, einen Client zu schreiben, der auf den Service zugreifen kann. [15, S.124]

2.2.2 SOAP

SOAP steht für **S**imple **O**bject **A**ccess **P**rotokoll und wird im Applikations-Layer verwendet. Es stützt sich auf andere Standards wie IP und TCP. Für die Übertragung der Nachricht können verschiedene Transport-Protokolle verwendet werden, wie z.B. HTTP oder SMTP. Für die Repräsentation der Daten wird XML verwendet. SOAP definiert, wie Informationen über XML strukturiert und typisiert werden, so dass sie zwischen zwei Rechnern ausgetauscht werden können [15].

SOAP ermöglicht dem Web Service Sprachenunabhängigkeit. Die zu übermittelnden Nachrichten, werden vor dem Senden in SOAP-Nachrichten verpackt (marshalling) und beim Empfangen als Erstes entpackt. Diese Aufgabe wird auf der Client-Seite vom Stub übernommen und auf der Server-Seite vom Skeleton. Für den Entwickler bedeutet dies, dass er kaum mit den SOAP-Nachrichten in Berührung kommt.

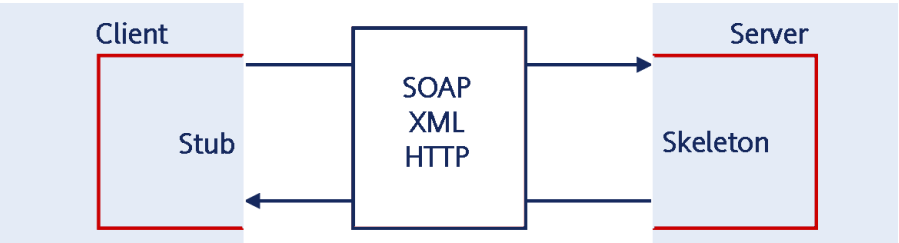

Abbildung 2.4: Kommunikation von Web Services

Die Nachricht besteht aus einem optionalen Header und einem Body. Diese beiden sind in einem Envelope verpackt. Der Header enthält Information über das Routing, Zugehörigkeit zu Transaktionen oder die Sicherheit. Der Body beinhaltet Daten, die der Empfänger interpretieren muss. Dies können z.B. Fehlermeldungen, Methodenaufrufe oder reine Daten sein.

Die Abbildung 2.5 zeigt eine fast vollständige SOAP Nachricht (die Namespaces wurden durch Punkte ersetzt). Der äusserste Rahmen bildet das Envelope. Wir erkennen, dass diese Nachricht keinen Header enthält, sondern nur

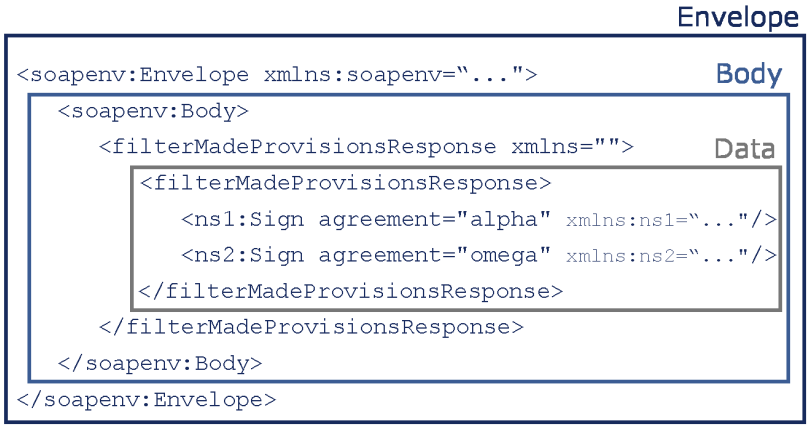

Abbildung 2.5: SOAP Nachricht

den Body. Im innersten Rahmen sehen wir die zu übermittelnden Daten im Klartext. Die Abbildung 2.7 zeigt eine SOAP-Nachricht mit Header.

2.2.3 Spezifikation eines Web Service

Der Web Service muss in einem standardisierten Format beschrieben werden, damit der Client auf den Service zugreifen kann. Dieses Format heisst WSDL und basiert auf XML. WSDL steht für **W**eb **S**ervices **D**efinition **L**anguage.

Die Spezifikation eines Web Service in WSDL besteht aus fünf Teilen.

- **Types:** Unter diesem Bauteil werden alle Typen spezifiziert, die von den Übergabeparametern benötigt werden.
- **Message:** Jede Nachricht, die der Service verschickt, wird hier definiert. Die Nachricht erhält einen Namen und Übergabeparameter. Den Übergabeparametern werden neben einem Namen, auch die Typen zugeordnet.
- **PortType:** Jede Nachricht, die zuvor spezifiziert worden ist, wird hier einer Methode als Input oder Output zugeordnet.
- **Binding:** Die Methoden werden zusammen mit einigen SOAP-Formatierungen dem Service zugeordnet.
- **Service:** Unter diesem Abschnitt wird dem Client mitgeteilt, über welche URL der Service aufgerufen werden kann.

Die Abbildung 2.6 zeigt anhand eines groben Beispiels eine Übersicht, wie das WSDL-File aufgebaut ist. Sie zeigt die Abhängigkeiten zwischen den verschiedenen Bausteinen.

Das WSDL-File ist die Basis von jedem Web Service. Es beinhaltet alle Angaben, die ein User über den Service kennen muss, um diesen verwenden zu können.

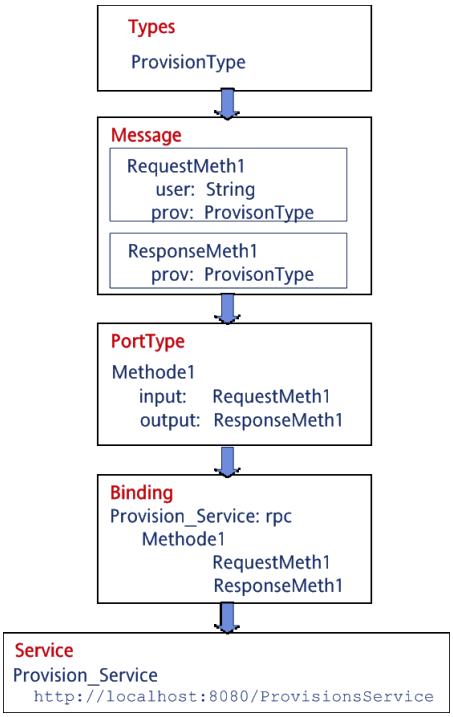

Abbildung 2.6: WSDL-File

2.3 Web Service Security

Web Service Security (WS-Security) ist ein Standard der internationalen Standardisierungsorganisation OASIS (**O**rganization for the **A**dvancement of **S**tructured **I**nformations **S**tandards) für die Implementation von Sicherheitsfunktionen bei Web Services.

WS-Security stellt Funktionalitäten bereit, um Integrität und Vertraulichkeit der Nachricht zu gewährleisten. Es ermöglicht das Signieren und Verschlüsseln von SOAP-Nachrichten sowie das Anhängen von weiteren Sicherheits-Tokens. Solche Sicherheits-Tokens werden für die Authentifizierung verwendet. Beispiele sind Username und Passwort, X.509 Zertifikate oder Binärtoken. WS-Security setzt die Standards XML Signature und XML Encryption des W3Cs ein [9].

2.3.1 SOAP

Der Aufbau einer signierten und verschlüsselten SOAP-Nachricht zeigt die Abbildung 2.7.

Abbildung 2.7: Verschlüsselte SOAP Nachricht

Kapitel 3

Spezifikation des Provisions Service als Web Service

Dieses Kapitel erläutert die Spezifikation des Provisions Service, welcher in diesem Projekt implementiert wird. Die ganze Spezifikation befindet sich im Anhang.

Die Ausgangslage jedes Web Service ist das WSDL-File, da darin alle benötigten Angaben des Services beinhaltet sind. Die Spezifikation besteht aus fünf Teilen. In diesem Kapitel wird die Spezifikation des Web Services aus Sicht des Clients betrachtet.

3.1 Types

```
<wsdl:types>
        <schema elementFormDefault="qualified"
                 targetNamespace="http://service.ps/docsec/accesspolicy">
            <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
            <complexType name="ProvisionsType">
                <sequence>
                   <element maxOccurs="unbounded" minOccurs="0" name="Agree„
                            nillable="true">
                        <complexType>
                            <sequence/>
                            <attribute name="agreement" type="xsd:string"/>
                        </complexType>
                    </element>

                    <element maxOccurs="unbounded" minOccurs="0" name="Sign"
                             nillable="true">
                        <complexType>
                            <sequence/>
                            <attribute name="agreement" type="xsd:string"/>
                        </complexType>
                    </element>
                </sequence>
            </complexType>
        </schema>
    </wsdl:types>
```

Abbildung 3.1: Types

In Abbildung 3.1 sehen wir die Spezifikation des Typs `ProvisionsType`. Dieser beinhaltet zwei weitere komplexe Typen `Agree` und `Sign`. Diese entsprechen

den beiden Provisions, welche im Projekt implementiert werden (siehe Abschnitt 2.1.1). Der Typ `ProvisionsType` kann beliebig viele Provisions beinhalten.

Beide Provisions `Agree` und `Sign` enthalten ein Attribut `agreement` vom Typ String.

3.2 Messages

```
<wsdl:message name="addMadeProvisionResponse">
</wsdl:message>

<wsdl:message name="filterMadeProvisionsRequest">
    <wsdl:part name="user" type="xsd:string"/>
    <wsdl:part name="provisions" type="tns1:ProvisionsType"/>
</wsdl:message>

<wsdl:message name="addMadeProvisionRequest">
    <wsdl:part name="user" type="xsd:string"/>
    <wsdl:part name="provision" type="tns1:ProvisionsType"/>
    <wsdl:part name="proof" type="xsd:anyType"/>
</wsdl:message>

<wsdl:message name="filterMadeProvisionsResponse">
    <wsdl:part name="filterMadeProvisionsResponse"
               type="tns1:ProvisionsType"/>
</wsdl:message>
```

Abbildung 3.2: Messages

Die Abbildung 3.2 zeigt die Spezifikation der vier Nachrichten. Eine Nachricht kennzeichnet sich durch die Übergabeparameter und deren Typen aus.

Nachricht	**Übergabeparameter**
addMadeProvisionRequest	provision: ProvisionsType user: string proof: anyType
addMadeProvisionResponse	
filterMadeProvisionsRequest	provisions: ProvisionsType user: string
filterMadeProvisionsResponse	filterMadeProvisionsResponse: ProvisionsType

3.3 PortType

In der Abbildung 3.3 ist die Implementation von PortType abgebildet. Dieser Teil des WSDL-Files ordnet die zuvor definierten Nachrichten den Operationen zu, welche vom Web Service zur Verfügung gestellt werden.

Operation: filterMadeProvisions

Diese Operation erwartet die Nachricht `filterMadeProvisionsRequest` als Input. Die Parameter der Nachricht sind `provisions` vom Typ `ProvisionsType`

```
<wsdl:portType name="ProvisionsService">
    <wsdl:operation name="filterMadeProvisions"
                    parameterOrder="user provisions">
        <wsdl:input message="impl:filterMadeProvisionsRequest"
                    name="filterMadeProvisionsRequest"/>
        <wsdl:output message="impl:filterMadeProvisionsResponse"
                     name="filterMadeProvisionsResponse"/>
    </wsdl:operation>

    <wsdl:operation name="addMadeProvision" parameterOrder="user
                    provision proof">
        <wsdl:input message="impl:addMadeProvisionRequest"
                    name="addMadeProvisionRequest"/>
        <wsdl:output message="impl:addMadeProvisionResponse"
                     name="addMadeProvisionResponse"/>
    </wsdl:operation>
</wsdl:portType>
```

Abbildung 3.3: PortType

und `user` vom Typ `string`. Der erste Parameter, der mitgegeben werden muss, ist `user`, der zweite `provisions`.
Dem Output wird die Nachricht `filterMadeProvisionsResponse` zugeordnet. Diese Nachricht enthält einen Parameter vom Typ `ProvisionsType`. Darum weiss ein User, dass die Methode einen Parameter vom Typ `ProvisionsType` zurückgibt.

Operation: addMadeProvision

Die Input-Nachricht dieser Operation ist `addMadeProvisionRequest`. Dies bedeutet, dass die Inputparameter `provisions` vom Typ `ProvisionsType`, `user` vom Typ `string` und `proof` vom Typ `anyType` sind. Die Reihenfolge der Inputparameter sind `user`, `provision` und `proof`.
`addMadeProvisionResponse` ist die Nachricht, welche dem Output der Operation zugeordnet wird. Diese Nachricht enthält keine Übergabeparameter. Dies bedeutet, dass die Methode keinen Wert zurückgibt.

3.4 Bindings

Die Abbildung 3.4 spezifiziert die Bindings des Provisions Service. Unter diesem Teil wir das Transport-Protokoll und das Arbeitsformat definiert. Das Transport-Protokoll wird im Attribut `transport` festgelegt. In unserem Fall ist das Protokoll `http`.
Das Format bestimmt, wie das WSDL-Binding auf eine SOAP-Nachricht abgebildet wird. [6]

3.5 Service

Wie in der Abbildung 3.5 gezeigt wird, wird unter Service dem Client mitgeteilt, unter welcher URL er den Service aufrufen kann.

```
<wsdl:binding name="ProvisionsServiceSOAPSoapBinding" type="impl:ProvisionsService">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>

    <wsdl:operation name="filterMadeProvisions">
        <wsdlsoap:operation soapAction="http://service.ps/filterMadeProvisions"/>
        <wsdl:input name="filterMadeProvisionsRequest">
            <wsdlsoap:body namespace="http://service.ps"  use="literal"/>
        </wsdl:input>

        <wsdl:output name="filterMadeProvisionsResponse">
            <wsdlsoap:body namespace="http://service.ps"  use="literal"/>
        </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="addMadeProvision">
        <wsdlsoap:operation   soapAction="http://service.ps/addMadeProvision"/>
        <wsdl:input name="addMadeProvisionRequest">
            <wsdlsoap:body namespace="http://service.ps"  use="literal"/>
        </wsdl:input>

        <wsdl:output name="addMadeProvisionResponse">
           <wsdlsoap:body namespace="http://service.ps"  use="literal"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

Abbildung 3.4: Bindings

```
<wsdl:service name="ProvisionsService">
    <wsdl:port binding="impl:ProvisionsServiceSOAPSoapBinding"
               name="ProvisionsServiceSOAP">
        <wsdlsoap:address
                location="http://localhost:8080/axis/services/ProvisionsServiceSOAP"/>
    </wsdl:port>
</wsdl:service>
```

Abbildung 3.5: Service

Kapitel 4

Entwicklerhandbuch

Dieses Kapitel erläutert den Ablauf der Entwicklung des Provisions Service. Im ersten Teil wird auf die Installation der verwendeten Tools eingegangen, wobei deren Aufgabenbereich zugleich erklärt wird. Der zweite Teil erklärt die Vorgehensweise, wie der Web Service ohne Sicherheitsanforderungen implementiert wird und der letzte Abschnitt fügt die Implementation der Sicherheitsanforderungen hinzu.

4.1 Installation

Die Implementation des Services erfolgt auf Windows XP.

Die verwendeten Versionen und wo die Applikationen runter geladen werden können, ist aus dem Literaturverzeichnis ersichtlich.

4.1.1 Java

Der Web Service wird in Java implementiert. Darum muss zuerst die Java Umgebung auf dem Computer installiert werden. Hier wird Java SE Development Kit 6 verwendet. Setze `JAVA_HOME` (hier: `C:\Programme\Java\jdk1.6.0`) [13] und füge `JAVA_HOME\bin` dem `PATH` hinzu.

4.1.2 Apache Tomcat

Apache Tomcat ist ein Web Server der eine Umgebung für die Ausführung von Java Code bereitstellt, was auch Servlet-Container genannt wird. Axis (siehe Abschnitt 4.1.3) wird häufig als Servlet verwendet, welches in einem Container betrieben wird. Dadurch ist es Axis möglich, Anfragen von Clients entgegen zu nehmen, zu bearbeiten und zu beantworten.

Installation

Lade eine Version von Apache Tomcat runter [1] und entzippe das File. Setze den Pfad `CATALINA_HOME` auf den neu erstellten Ordner (hier: `C:\Programme\apache-tomcat-5.5.23`) und `CATALINA_HOME\bin` auf den `PATH`.

Nun sollte der Server über die Konsole gestartet werden können, indem im Ordner `C:\tomcat\bin` das File `startup.bat` ausgeführt wird. Über den Browser kann über die URL `http://localhost:8080` auf den Server zugegriffen werden. Das File `shutdown.bat` beendet den Server.

4.1.3 Axis

Axis steht für **A**pache **X**tensible **I**nteraction **S**ystem. Es ist ein Framework, welches die Entwicklung von Web Services unterstützt. Axis stellt einige Tools zur Verfügung, um die Entwicklung eines Web Services in Java stark zu vereinfachen.

- Durch ein gegebenes Java Interface vom zukünftigen Web Service kann das WSDL-File des Service generiert werden. Diese Funktion ist vor allem als Ausgangspunkt hilfreich. Das generierte File kann erweitert und dem Service angepasst werden.
- Aus dem WSDL-File können Stub und Skeleton generiert werden.
- Der implementierte Service wird veröffentlicht, damit Clients darauf zugreifen können. Dies wird Deployment genannt.
- Axis stellt auch einen TCP Monitor zur Verfügung, welcher dem Axis Nutzer ermöglicht, die SOAP Nachrichten, welche vom und aus dem Host gesendet werden, zu lesen.

Axis unterstützt den Enwicklungsprozess eines Services von Anfang bis Ende, was in Abbildung 4.1 sichtbar ist. Die roten Pfeile kennzeichnen die Schritte, die von Axis übernommen werden. Der hellblaue Pfeil entspricht der Arbeit des Entwicklers.

Abbildung 4.1: Entwicklungsprozess mit Axis

Axis kann einfach in einen Web Container wie Tomcat integriert werden. Dies ist nötig, damit ein Service dem Client zugänglich gemacht werden kann.

Installation

Lade die Version 1.4 von Axis runter [2] und entzippe das File. Anschliessend setze den Pfad `AXIS_HOME` (hier: `C:\Programme\axis-1_4`), `AXIS_LIB` (hier: `AXIS_HOME\lib`) und `AXISCLASSPATH` (hier: `AXIS_LIB\commons-discovery-0.2.jar; AXIS_LIB\wsdl4j-1.5.1.jar; AXIS_LIB\jaxrpc.jar; AXIS_LIB\saaj.jar; AXIS_LIB\log4j-1.2.8.jar; AXIS_LIB\commons-logging-1.0.4.jar; AXIS_LIB\axis-ant.jar; AXIS_LIB\axis.jar;`).

Zudem verwendet Axis JavaMail [12] und das JavaBeans Activation Framework [11]. Diese müssen runtergeladen und entpackt werden. Kopiere `JAVAMAIL_HOME\mail.jar` und `JAF_HOME\activation.jar` in den Ordner `AXIS_HOME\webapps\axis\WEB-INF\lib` und füge die beiden jar-Files dem `AXISCLASSPATH` hinzu.

Falls WSS4J für die Implementation der Sicherheitsanforderungen verwendet wird, wird zudem Xerces benötigt. Kopiere die jar-Files `Xerces_HOME\xml_apis.jar` und `Xerces_HOME\xercesImpl.jar` in den Ordner `AXIS_HOME\webapps\axis\WEB-INF\lib` und füge sie dem `AXISCLASSPATH` hinzu [5].

Als nächstes muss der Axis Server in Tomcat registriert werden. Erstelle dazu ein File `axis.xml` unter `CATALINA_HOME\conf\Catalina\localhost`. Der Inhalt des Files ist aus Abbildung 4.2 ersichtlich.

```
<Context  docBase="C:\Programme\axis-1_4\webapps\axis"
          path="/axis"/>
```

Abbildung 4.2: Einbinden von Axis in Tomcat

Beim Zugriff über den Browser mit der URL `http://localhost:8080/axis/` sollte folgendes Bild erscheinen (Abbildung 4.3).

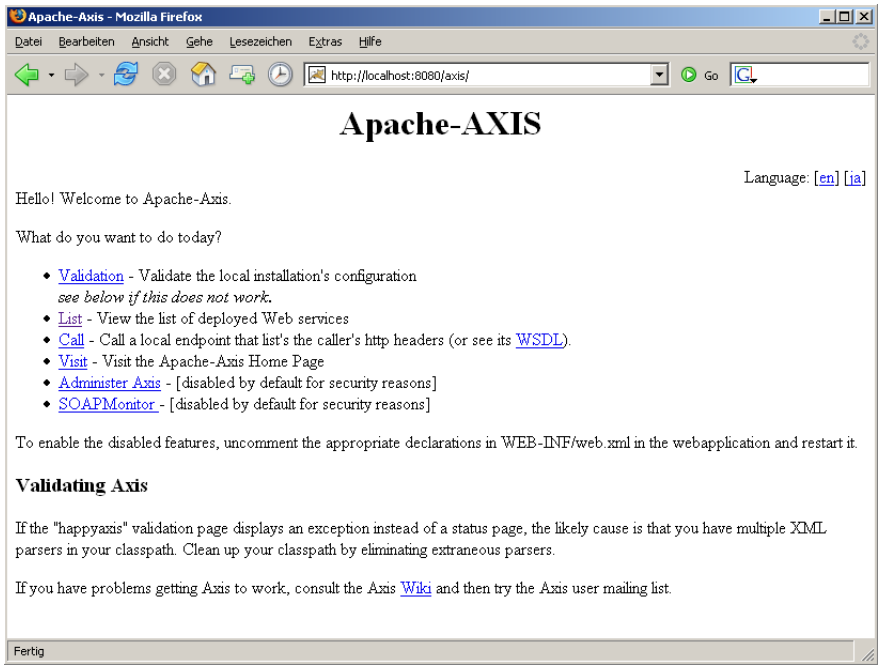

Abbildung 4.3: Axis Server

4.1.4 hsqldb

hsqldb ist eine relationale Datenbank, welche in Java geschrieben wurde und für kleine Anwendungen praktisch zu nutzen ist.

Installation

Lade die Datenbank runter [10] und entzippe das File. Kopiere anschliessend das File hsqldb.jar in den Ordner `CATALINA_HOME\common\lib`.

4.1.5 WSS4J

Es gibt verschiedene Implementationen von Web Service Security. Eine davon ist WSS4J (Web Service Security for Java). Es bietet Unterstützung für Axis.

Um WSS4J nutzen zu können, wird der Handler-Mechanismus von Axis benutzt. Ein Handler kann als Filter in den Eingangs- oder Ausgangsdatenstrom eingehängt werden. Bei WSS4J sind das die beiden Handler-Klassen WSDoAllSender und WSDoAllReceiver. Der Receiver wird immer in den Eingangsdatenstrom, der Sender in den Ausgangsdatenstrom eingehängt. Wie Abbildung 4.4 zeigt, ist das beim Service eingehend der RequestFlow und ausgehend der ResponseFlow, beim Client ist es genau umgekehrt, eingehend der ResponseFlow und ausgehend der RequestFlow [17].

Abbildung 4.4: Handlermechanismus

Installation

Um die Sicherheitskriterien umsetzen zu können, benötigen wir die WSS4J. Lade darum die beiden zip-Files WSS4J [3] und WSS4J-Otherjars [4] vom Internet und entzippe sie in den gleichen Ordner. Setzte `WSS4J_HOME` (hier: `C:\Programme\wss4j`) und `WSS4J_CLASSPATH` auf `WSS4J_HOME\wss4j-1.5.1.jar; WSS4J_HOME\lib\addressing-1.0.jar; WSS4J_HOME\lib\bcprov-jdk15-132.jar; WSS4J_HOME\lib\commons-codec-1.3.jar; WSS4J_HOME\lib\opensaml-1.0.1.jar; WSS4J_HOME\lib\xalan-2.7.0.jar; WSS4J_HOME\lib\xmlsec-1.3.0.jar`).

Kopiere das File `WSS4J_HOME\lib\bcprov-jdk15-132.jar` in den Ordner `CATALINA_HOME\common\lib` .

4.2 Implementation des Web Service

Die hier erwähnten Befehle befinden sich auf der CD im Ordner `batFiles` und stehen als bat-Files zur Verfügung.

4.2.1 Spezifikation des Web Service

Die Ausgangslage jedes Web Services ist das WSDL-File, da darin alle benötigten Angaben des Services beinhaltet sind. Unser WSDL-File ist `ProvisionsService.wsdl`. Die Typen werden aus den beiden wsd-Files im Ordner `Schemas` importiert.

4.2.2 Generieren von Skeleton

Das un-/marshalling übernimmt auf der Serverseite das Skeleton. Für die Generation des Skeleton führe als erstes den Befehl `java -cp ‘‘AXISCLASSPATH‘‘ org.apache.axis.wsdl.WSDL2Java ProvisionsService.wsdl` aus (bat-File: `wsdl2java.bat`). Dieser Befehl generiert folgende Files:

- `.\ps\service\ProvisionsService_PortType.java`

- .\ps\service\ProvisionsService_Service.java
- .\ps\service\ProvisionsService_ServiceLocator.java
- .\ps\service\ProvisionsServiceSOAPStub.java

Der zweite Befehl generiert weitere Files. `java -cp ‘‘AXISCLASSPATH‘‘ org.apache.axis.wsdl.WSDL2Java -server-side -skeletonDeploy true ProvisionsService.wsdl` (bat-File: `sceleton.bat`):

- .\ps\service\deploy.wsdd
- .\ps\service\ProvisionsServiceSOAPImpl.java
- .\ps\service\ProvisionsServiceSOAPSkeleton.java
- .\ps\service\ProvisionsServiceSOAPStub.java
- .\ps\service\undeploy.wsdd
- .\ps\service\docsec\AttributeType.java
- .\ps\service\docsec\UserIDType.java
- und weitere Files unter .\ps\service\docsec\accesspolicy

4.2.3 Implementation des Service

Im nächsten Schritt soll der Service in der bereits generierten Klasse `.\ps\service\ProvisionsServiceSOAPImpl.java` implementiert werden. Für die Implementation werden zwei weitere Klassen verwendet, die sich im Package `ps.service.helpFkt` befinden. Die Klasse `Proof.java` überprüft, ob die Provision, welche in die Datenbank eingefügt werden soll, getroffen ist. (Diese Methode gibt bis jetzt nur *true* zurück, da die Art der Überprüfung noch nicht klar ist.) In der Klasse `OrganiseDB.java` befindet sich die Verwaltung der Datenbank. Die Variable `path` muss dem System angepasst werden. Kompiliere anschliessend die Files: `javac -cp ‘‘AXISCLASSPATH‘‘;. ps\service\*.java` (bat-File: `compileSceleton.bat`).

4.2.4 Deployment

Um den Service öffentlich zugänglich zu machen, muss der Ordner `ps` in den Ordner `AXIS_HOME\webapps\axis\WEB-INF\classes` kopiert werden. Starte anschliessen Apache Tomcat. Nun kann der Service mit folgendem Befehl öffentlich zugänglich gemacht werden: `java -cp ‘‘AXISCLASSPATH‘‘;. org.apache.axis.client.AdminClient ps\service\deploy.wsdd` (bat-File: `deploy.bat`).

4.3 Implementation des Clients

Der erste Schritt um den Service verwenden zu können, besteht im Generieren des Stubs. Dazu kann der Befehl `java -cp ‘‘AXISCLASSPATH‘‘ org.apache.axis.wsdl.WSDL2Java -p ps.clientStub http://localhost:8080/axis/`

services/ProvisionsServiceSOAP?wsdl (batFile: stub.bat) verwendet werden. Auf der URL
http://localhost:8080/axis/services/ProvisionsServiceSOAP.wsdl befindet sich das WSDL-File, welches alle Informationen enthält, um mit dem Service kommunizieren zu können.

Implementiere anschliessend den Client. Um eine Methode remote aufzurufen, muss der Code aus Abbildung 4.5 verwendet werden.

```
//Without encryption
ProvisionsService_Service service = new ProvisionsService_ServiceLocator();
ProvisionsService_PortType portType= service.getProvisionsServiceSOAP();

try {
          portType.addMadeProvision(user, provisions, proof);
          result=portType.filterMadeProvisions(user, provisions);
} catch (RemoteException e) {
          System.out.println("Error");
}
```

Abbildung 4.5: Entfernter (Remote) Methodenaufruf

Kompiliere anschliessend die Klasse.

4.3.1 Teste den Service

Bevor der Service getestet werden kann, muss zuerst eine Tabelle in der Datenbank erstellt werden. Im Package testService befindet sich eine Klasse CreateTable.java, welche lokal eine Tabelle erstellt. Der Pfad zur erstellten Datenbank befindet sich im Package ps.service.helpFkt in der Klasse OrganiseDB.java. Dieser muss angepasst werden. Zwei weitere Klassen stehen zur Verfügung, welche nützlich sein können: InsertEntries.java, welche Einträge in die Datenbank einfügt, und ViewEntries.java, welche den Inhalt der Datenbank anzeigt.

Nun kann der Service getestet werden. Führe den Befehl java -cp ‘‘AXISCLASSPATH‘‘;. ps.client.ClientImplementation aus.

4.4 Sicherheit

4.4.1 Zertifikate erstellen

Mit dem keytool von JDK generieren wir das Schlüsselpaar. Das bat-File generateKey.bat enthält den Code aus Abbildung 4.6, um die Zertifikate zu erstellen.

Nun müssen die Zertifikate in die richtigen Ordner verschoben werden, so, dass die Parteien darauf zugreifen können. client.ks muss ins PackageRoot des Clients, und server.ks in den Ordner AXIS_HOME\ webapps\axis\WEB-INF\classes verschoben werden.

4.4.2 Weitere Vorkehrungen

Für die Umsetzung der Sicherheitskriterien werden weitere Files benötigt:

```
keytool -genkey -keyalg rsa -alias client -keystore client.ks -dname
          cn=client -keypass clientKeyPass -storepass clientStorePass
keytool -genkey -keyalg rsa -alias server -keystore server.ks -dname
          cn=SemesterArbeit -keypass serverKeyPass -storepass serverStorePass

keytool -export -keystore client.ks -alias client -storepass
          clientStorePass -file client.cer
keytool -export -keystore server.ks -alias server -storepass
          serverStorePass -file server.cer

keytool -export -keystore client.ks -alias client -storepass
          clientStorePass -file client.cer
keytool -export -keystore server.ks -alias server -storepass
          serverStorePass -file server.cer

keytool -import -noprompt -alias server -file server.cer -keystore
          client.ks -storepass clientStorePass
keytool -import -noprompt -alias client -file client.cer -keystore
          server.ks -storepass serverStorePass
```

Abbildung 4.6: Generieren von Zertifikaten

File-Name	**File in diesem Ordner platzieren**
`client_deploy.wsdd`	`ps\client`
`client-provider.properties`	PackageRoot des Clients
`server-provider.properties`	`AXIS_HOME\webapps\axis\WEB-INF\classes`
`PasswordProvider.java`	`ps\service`
`PasswordProvider.java`	`ps\client`

Zudem muss das File deploy.wsdd angepasst werden. Die Properties-Files enthalten die Passwörter um aufs Zertifikat zugreifen zu können. Und die Klassen `PasswordProvider.java` enthalten das Passwort des privaten Schlüssels, damit der Schlüssel in der Konfiguration nicht als Klartext hinterlegt werden muss.

Kopiere den Ordner `ps` erneut in `AXIS_HOME\webapps\axis\WEB-INF\classes` und führe das Deployment erneut aus.

4.4.3 Implementation des Clients

Der Methodenaufruf mit Verschlüsselung ist etwas anders als jener ohne Verschlüsselung. Das File `client_deploy.wsdd` muss für das Empfangen und Senden der Nachrichten eingebunden werden, da darin die Handler für die Signatur und Ver- bez.Entschlüsselung enthalten sind.

```
//With encryption
EngineConfiguration config = new FileProvider(client_deploy.wsdd);
ProvisionsService_Service locator = new ProvisionsService_ServiceLocator(config);
ProvisionsService_PortType portType= locator.getProvisionsServiceSOAP();

try {
          result=portType.filterMadeProvisions(user, provisions);
} catch (RemoteException e) {
          System.out.println("Error");
}
```

Abbildung 4.7: Entfernter (Remote) Methodenaufruf mit Verschlüsselung

Kapitel 5

Zusammenfassung

Ein Policy Decision Point (PDP) eines Access Control Systems möchte überprüfen können, ob alle Provisions, die für den Zugriff auf ein Objekt nötig sind, erfüllt sind. Dazu greift der PDP auf einen Provisions Service (PS) zu, der die Auswertung der Provisions übernimmt. Der PS wird als Web Service implementiert, da er Sprachen- und Plattformunabhängigkeit gewährleistet.

Neben der Implementation des Web Service sollen auch sinnvolle Sicherheitsanforderungen getroffen und umgesetzt werden.

5.1 Erläuterungen

Dieser Abschnitt geht auf einige Probleme und Fragestellungen ein, die während dem Entwicklungsprozess aufgetreten sind.

5.1.1 Erstellen des WSDL-Files

Das erstellen des WSDL-Files kann schwierig sein. Darum ist es hilfreich, zuerst ein Java Interface des zukünftigen Web Service zu schreiben. Anschliessen kann anhand der Funktion `Java2WSDL` von Axis ein WSDL-File generiert werden. Das File muss nun nur noch angepasst werden.

5.1.2 Objekt-Orientierte Web Services

Da der Service in Java, einer objekt-orientierter Programmiersprache, implementiert wurde, wurden die Provisions so konzipiert, dass der Polymorphismus ausgenutzt wurde. Alle Provisions, wie zum Beispiel Agree und Sign (siehe Abschnitt 2.1.1), erben von einer Klasse `Provision`, wie Abbildung 5.1 zeigt. Die Methode erwartet einen Parameter vom Typ `Provision`, welcher zur Laufzeit, durch den Polymorphismus, auch einen Subtyp von `Provision` sein kann.

Da der Methodenaufruf remote ist, ist dies nicht möglich. Der Fehler geschieht beim Skeleton. Er erwartet ein Parameter vom Typ `Provision`, erhält aber zum Beispiel ein Parameter vom Typ `Agree`. Das Skeleton erkennt nicht, dass eine Subtyp-Relation vorliegt und gibt einen Fehler aus.
Wie die Abbildung 5.2 zeigt, wurde das Problem so gelöst, dass die Klasse `ProvisionsType` einen Array von jeder existierenden Provision enthält.

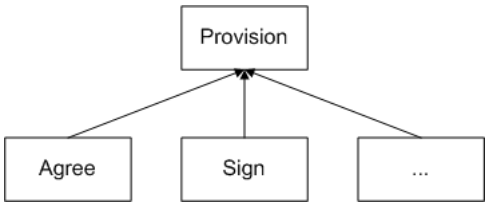

Abbildung 5.1: Polymorphismus

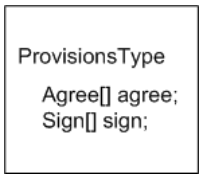

Abbildung 5.2: Lösung mittels Arrays

5.1.3 Warum nicht mit SSL verschlüsseln?

Bei HTTPS handelt es sich um eine Punkt-zu-Punkt-Verschlüsselung. Deshalb können die übermittelten Nachrichten nicht durch andere Beteiligte verarbeitet, ergänzt oder modifiziert werden. Darüber hinaus bleibt die Verschlüsselung auf ganz oder gar nicht beschränkt. Das Verschlüsseln der Daten für unterschiedliche Empfänger und die digitale Signatur als verbindlicher Nachweis von Ursprung und Authentizität sind kein Bestandteil des Protokolls. Abhilfe für diese Probleme bietet der alternative oder ergänzende Einsatz von Verfahren auf der Datenebene. Diese Mechanismen wirken Peer-to-Peer und ermöglichen neben der Verschlüsselung vor allem die Authentifizierung von Absender und Empfänger und die digitale Signatur. Als Standard für den Einsatz solcher kryptografischer Verfahren hat sich hierfür WS-Security etabliert [17].

5.1.4 Verschlüsselung

Da wir davon ausgehen, dass wir es mit einer geschlossenen Gruppe zu tun haben, können wir Public-Key-Verschlüsselung anwenden. Der Sender verschlüsselt einen symmetrischen Schlüssel mit dem Public-Key des Empfängers und verschlüsselt die Nachricht mit dem symmetrischen Schlüssel.

Die Implementation für eine offene Gruppe mittels symmetrischer Verschlüsselung ist eine Aufgabe für zukünftige Arbeiten.

5.1.5 Sicherheitsanforderungen nicht in der Spezifikation

Ein Problem von WSS besteht darin, dass die Sicherheitsanforderungen nicht in der Spezifikation des Web Service enthalten sind. Darum benötigt ein Client weitere Informationen ausserhalb der WSDL-Files, um den Service richtig aufrufen zu können.

Anhang A

WSDL-File des Provisions Service

```
 <wsdl:definitions targetNamespace=‘‘http://service.ps‘‘>
<wsdl:types>
<schema elementFormDefault=‘‘qualified‘‘
targetNamespace=‘‘http://service.ps/docsec/accesspolicy‘‘>
<import namespace=‘‘http://schemas.xmlsoap.org/soap/encoding/‘‘/>
<complexType name=‘‘ProvisionsType‘‘>
<sequence>
<element maxOccurs=‘‘unbounded‘‘ minOccurs=‘‘0‘‘ name=‘‘Agree‘‘
nillable=‘‘true‘‘>
<complexType>
<sequence/>
<attribute name=‘‘agreement‘‘ type=‘‘xsd:string‘‘/>
</complexType>
</element>
<element maxOccurs=‘‘unbounded‘‘ minOccurs=‘‘0‘‘ name=‘‘Sign‘‘
nillable=‘‘true‘‘>
<complexType>
<sequence/>
<attribute name=‘‘agreement‘‘ type=‘‘xsd:string‘‘/>
</complexType>
</element>
</sequence>
</complexType>
</schema>
</wsdl:types>

<wsdl:message name=‘‘addMadeProvisionResponse‘‘>
</wsdl:message>

<wsdl:message name=‘‘filterMadeProvisionsRequest‘‘>
<wsdl:part name=‘‘user‘‘ type=‘‘xsd:string‘‘/>
<wsdl:part name=‘‘provisions‘‘ type=‘‘tns1:ProvisionsType‘‘/>
```

```
</wsdl:message>

<wsdl:message name=‘‘addMadeProvisionRequest‘‘>
<wsdl:part name=‘‘user‘‘ type=‘‘xsd:string‘‘/>
<wsdl:part name=‘‘provision‘‘ type=‘‘tns1:ProvisionsType‘‘/>
<wsdl:part name=‘‘proof‘‘ type=‘‘xsd:anyType‘‘/>
</wsdl:message>

<wsdl:message name=‘‘filterMadeProvisionsResponse‘‘>
<wsdl:part name=‘‘filterMadeProvisionsResponse‘‘ type=‘‘tns1:ProvisionsType‘‘/>
</wsdl:message>

<wsdl:portType name=‘‘ProvisionsService‘‘>
<wsdl:operation name=‘‘filterMadeProvisions‘‘ parameterOrder=‘‘user provisions‘‘>
<wsdl:input message=‘‘impl:filterMadeProvisionsRequest‘‘
name=‘‘filterMadeProvisionsRequest‘‘/>
<wsdl:output message=‘‘impl:filterMadeProvisionsResponse‘‘
name=‘‘filterMadeProvisionsResponse‘‘/>
</wsdl:operation>

<wsdl:operation name=‘‘addMadeProvision‘‘ parameterOrder=‘‘user provision
proof‘‘>
<wsdl:input message=‘‘impl:addMadeProvisionRequest‘‘
name=‘‘addMadeProvisionRequest‘‘/>
<wsdl:output message=‘‘impl:addMadeProvisionResponse‘‘
name=‘‘addMadeProvisionResponse‘‘/>
</wsdl:operation>
</wsdl:portType>

<wsdl:binding name=‘‘ProvisionsServiceSOAPSoapBinding‘‘ type=‘‘impl:ProvisionsService‘‘>
<wsdlsoap:binding style=‘‘rpc‘‘ transport=‘‘http://schemas.xmlsoap.org/soap/http‘‘/>
<wsdl:operation name=‘‘filterMadeProvisions‘‘>
<wsdlsoap:operation soapAction=‘‘http://service.ps/filterMadeProvisions‘‘/>
<wsdl:input name=‘‘filterMadeProvisionsRequest‘‘>
<wsdlsoap:body namespace=‘‘http://service.ps‘‘ use=‘‘literal‘‘/>
</wsdl:input>
<wsdl:output name=‘‘filterMadeProvisionsResponse‘‘>
<wsdlsoap:body namespace=‘‘http://service.ps‘‘ use=‘‘literal‘‘/>
</wsdl:output>
</wsdl:operation>

<wsdl:operation name=‘‘addMadeProvision‘‘>
<wsdlsoap:operation soapAction=‘‘http://service.ps/addMadeProvision‘‘/>
<wsdl:input name=‘‘addMadeProvisionRequest‘‘>
<wsdlsoap:body namespace=‘‘http://service.ps‘‘ use=‘‘literal‘‘/>
</wsdl:input>
<wsdl:output name=‘‘addMadeProvisionResponse‘‘>
<wsdlsoap:body namespace=‘‘http://service.ps‘‘ use=‘‘literal‘‘/>
</wsdl:output>
</wsdl:operation>
```

```
</wsdl:binding>

<wsdl:service name=‘‘ProvisionsService‘‘>
<wsdl:port binding=‘‘impl:ProvisionsServiceSOAPSoapBinding‘‘
name=‘‘ProvisionsServiceSOAP‘‘>
<wsdlsoap:address
location=‘‘http://localhost:8080/axis/services/ProvisionsServiceSOAP‘‘/>
</wsdl:port>
</wsdl:service>

</wsdl:definitions>
```

Anhang B

CD

Literaturverzeichnis

[1] Apache Software Foundation. apache-tomcat-5.5.23.zip. http://tomcat.apache.org/download-55.cgi, März 2007.

[2] Apache Software Foundation. axis-bin-1_4.zip. http://ws.apache.org/axis/, März 2007.

[3] Apache Software Foundation. wss4j-bin-1.5.1.zip. http://ws.apache.org/wss4j, März 2007.

[4] Apache Software Foundation. wss4j-otherjars-1.5.1.zip. http://ws.apache.org/wss4j, März 2007.

[5] Apache Software Foundation. Xerces_j-tools.2.5.0.zip. http://archive.apache.org/dist/xml/xerces-j, März 2007.

[6] IBM. Which style of wsdl should i use? http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/, März 2007.

[7] Kent Tong Ka Iok. *Developing Web Services with Apache Axis*. TipTec Development, 2005.

[8] Torsten Langner. *Web Services mit Java*. Mark+Technik, 2003.

[9] OIO. Die ws-* spezfikationen. http://www.oio.de/public/xml/web-service-specifications.htm, März 2007.

[10] Sourceforge. hsqldb_1_8_0_7.zip. http://sourceforge.net/projects/hsqldb, März 2007.

[11] Sun. jaf-1_1-fr.zip. http://java.sun.com/products/javabeans/jaf/downloads/index.html, März 2007.

[12] Sun. javamail-1_4.zip. http://java.sun.com/products/javamail/downloads/, März 2007.

[13] Sun. jdk-6-windows-i586.exe. http://java.sun.com/javase/downloads/index.jsp, März 2007.

[14] Marc Thalman. Wss4j-web services security for java. Technical report, Fachochschule Nordwestschweiz, 2006.

[15] G.Alonso; F.Casati; H.Kuno; V.Machiraju. *Web Services*. Springer, 2004.

[16] W3C. Web services architecture requirements. http://www.w3.org/TR/wsa-reqs, März 2007. Freie Übersetzung der Autorin dieser Dokumentation.

[17] XML-Magazin. Ws-security mit apache wss4j. http://xml-magazin.de/itr/online_artikel/psecom,id,762,nodeid,69.html, März 2007.