Georg-August-Universität
Göttingen
Zentrum für Informatik

ISSN 1612-6793
Nummer ZFI-BM-2004-17

Bachelorarbeit

im Studiengang „Angewandte Informatik“

Design eines Lehrevaluationssystems mit Web Services

Anke Sawatzki

am Institut für Informatik

Bachelor- und Masterarbeiten
des Zentrums für Informatik
an der Georg-August-Universität Göttingen

8. Oktober 2004

Georg-August-Universität Göttingen
Zentrum für Informatik

Lotzestraße 16-18
37083 Göttingen
Germany

Tel. +49 (5 51) 39-1 44 02
Fax +49 (5 51) 39-1 44 03
Email office@informatik.uni-goettingen.de
WWW www.informatik.uni-goettingen.de

Ich erkläre hiermit, daß ich die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Göttingen, den 8. Oktober 2004

Danksagung

Meiner Betreuerin Edith Werner gilt ein besonderer Dank für die hilfreiche Unterstützung beim Erstellen meiner Bachelorarbeit. Sie hatte bei Fragen immer ein offenes Ohr für mich und ermöglichte mir einen eigenen Rechnerplatz am Institut für Informatik, so dass ich dort das Evaluationssystem implementieren konnte.

Auch bei den anderen Mitarbeitern des Instituts für Informatik, die mir gelegentlich bei Problemen zur Seite standen, bedanke ich mich.

INHALTSVERZEICHNIS

I ABBILDUNGSVERZEICHNIS

II Tabellenverzeichnis

III ABKÜRZUNGSVERZEICHNIS

API	Application Programming Interface
Axis	Apache Extensible Interaction System
DOM	Document Object Model
DTD	Document Type Definition
FTP	File Transfer Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JAX-RPC	Java API for XML-based RPC
JMS	Java Message Service
JSP	Java Server Pages
JVM	Java Virtual Machine
JWS	Java Web Service
ODBC	Open Database Connectivity
PDA	Personal Digital Assistant
PIN	Personal Identification Number
RPC	Remote Procedure Call
SAAJ	SOAP with Attachments API for Java
SAX	Simple API for XML Parsing
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
UBR	UDDI Business Registry
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Unique User Identification
W3C	World Wide Web Consortium
WSDD	Web Service Deployment Descriptor
WSDL	Web Service Description Language
WS Security	Web Services Security
XML	Extensible Markup Language
XSD	XML Schema Definition

1 EINLEITUNG

Seit einigen Jahren werden an den Universitäten Bewertungen der Lehrveranstaltungen durchgeführt. Bisher werden zu Ende des Semesters während der Vorlesung Fragebögen durch die Reihen gereicht, die jeder Student ausfüllen soll. Das Problem dabei ist, dass die Auswertung sehr viel Fleißarbeit bedeutet, da jeder Fragebogen per Hand ausgewertet werden muss. Außerdem kann es vorkommen, dass zum Evaluationszeitpunkt ein Student krank ist oder eine Klausur einer anderen Veranstaltung stattfindet.

Diese Gründe sprechen gegen eine manuelle Lehrevaluation. Deshalb ist am Institut für Informatik entschieden worden, Fragebögen zu den Vorlesungen über ein Webformular anzubieten. Diese Online-Evaluation hat folgende Vorteile:

Es gibt keinen bestimmten Zeitpunkt, zu dem die Evaluation stattfindet, sondern jeder Student kann zu jeder Tageszeit seine Vorlesungen bewerten. Damit kann man jeden Studenten erreichen und schafft die Möglichkeit ein aussagekräftigeres Ergebnis zu erzielen, da die Zahl der Teilnehmer an der Umfrage größer als bisher sein dürfte. Außerdem fällt die zeitaufwändige manuelle Auswertung weg. Sie findet von nun an automatisch per Mausklick statt.

Ein Nachteil dieser Methode ist, dass ein Student vergessen könnte an der Evaluation teilzunehmen. Allerdings kann er auch bei dieser Methode eine Teilnahme mangels Interesse ablehnen. Wenn aber in einer Vorlesung Evaluationsfragebögen durchgereicht werden, werden diese auch nicht von jedem ernsthaft ausgefüllt. Findet die Evaluation aber über ein Web-Formular statt, nehmen meist nur Studenten teil, die ein wirkliches Interesse an der Bewertung ihrer Vorlesungen haben. Dadurch erhält man mehr Antworten auf Fragen, zu denen sich Studenten Gedanken gemacht haben, welche zu einem genaueren Ergebnis führen.

Ein weiterer Nachteil könnte entstehen, wenn nicht gesichert ist, dass jeder Student jede Vorlesung nur einmal evaluieren kann.

Da am Institut für Informatik bisher nur Evaluierungsfragebögen ausgeteilt werden, soll nun zur besseren Durchführung der Evaluation ein Webformular bereitgestellt werden. Dieses System soll in die fünf folgenden unterschiedlichen Bereiche eingeteilt werden: Ein Administrationsbereich wird dem Verwalten der Vorlesungen, ein Teilnehmerbereich der Evaluierung durch die Studenten und ein Auswertemodul der Bewertung der evaluierten Vorlesungen dienen. Zur Speicherung der Daten werden zwei getrennte Datenbankmodule erstellt. Eine Datenbank speichert die Daten für die Autorisierung, die andere wird mit Daten der Fragebögen und Vorlesungen gefüllt. Im Gegensatz zu herkömmlichen Applikationen im Internet wird das Evaluationssystem aus mehreren über Web Services kommunizierenden Bereichen bestehen, aber auf der Benutzerseite die Gestalt eines einzigen Moduls haben.

In Kapitel 2 dieser Bachelorarbeit werden die Grundlagen, die für die Erstellung eines Web Services nötig sind, erläutert. Es werden sowohl theoretische Grundlagen, wie XML und Web Services mit ihren Komponenten, als auch die praktischen Grundlagen, wie der Webserver Tomcat und Axis, erklärt. Das Kapitel 3 stellt kurz die Evaluation mit ihren Anforderungen dar. Danach folgt in Kapitel 4 der Hauptteil, in dem das Evaluationssystem von der Idee über das Fachkonzept bis hin zu Kodebeispielen beschrieben wird. Zum Abschluss wird in Kapitel 5 noch eine Zusammenfassung gegeben.

2 GRUNDLAGEN

In diesem Kapitel werden sowohl die theoretischen als auch die praktischen Grundlagen erklärt. Im theoretischen Teil werden XML, Web Services mit ihren Bestandteilen und SQL anhand von Beispielen erläutert. Die für die Erstellung des Evaluationssystems benötigten Werkzeuge werden im letzten Teil genannt. Außerdem werden die zwei unterschiedlichen Arten von Web Services gezeigt.

2.1 Theorie

Da Protokolle von Web Services auf XML basieren, wird diese Metasprache mit ihren Erweiterungen im ersten Abschnitt beschrieben. Danach folgt eine Klärung des Begriffs Web Service. In den letzten Abschnitten werden das Protokoll SOAP, die Beschreibungssprache WSDL und der Verzeichnisdienst UDDI der Web Services dargestellt.

2.1.1 Extensible Markup Language (XML)

XML ist eine Metasprache und ähnelt HTML. Anders jedoch als bei HTML (Hypertext Markup Language) dient sie nicht der Darstellung von Daten im Browser, sondern ist für die Speicherung von semistrukturierten Daten zuständig. Sowohl bei HTML als auch XML werden Tags verwendet, die eine hierarchische Struktur bilden. Während ein HTML-Dokument aus vordefinierten Tags besteht, werden in XML-Dokumenten selbst definierte Tags verwendet, die Daten beschreiben. Diese Tags und ihre Schachtelungen können in einer Document Type Definition (DTD) festgelegt werden.
Anhand der Tags in einem XML-Dokument können die eingeschlossenen Daten nicht nur von Menschen verstanden, sondern auch von vielen Sprachen eingelesen und bearbeitet werden. Deshalb ist XML gut geeignet für die automatische Weiterverarbeitung von Daten. Wegen dieser Vorteile erfolgt der Datenaustausch mittels XML schon in vielen Bereichen wie dem Internet, dem Mobiltelefon und in etlichen anderen Anwendungen. [6]

2.1.1.1 Aufbau

Ein XML-Dokument beginnt mit dem Prolog, der unter anderem die XML-Version, die Kodierung (**`encoding`**), den Ort der DTD und den des Stylesheets angeben kann. Notwendig ist nur die Angabe der Zeile

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

ganz am Anfang des Dokuments, damit die Anwendung erkennt, dass es sich um ein XML-Dokument handelt. Der Wert der Kodierung in diesem Beispiel ermöglicht die Erkennung von deutschen Umlauten im Text.

Wenn zu dem Dokument eine DTD gehört, folgt folgende Zeile:

<!DOCTYPE beschreibung SYSTEM "beschreibung.dtd">

Über

<?xml-stylesheet href="layout.css" type="text/css"?>

werden die Layoutvorgaben referenziert.

Nach dem Prolog folgt der eigentliche Teil des XML-Dokuments. Hier stehen die Tags mit ihren Inhalten. Ein Tag mit seinem Inhalt nennt man Element. Es gibt zwei Möglichkeiten diese Elemente anzugeben. Hat das Element einen Inhalt, wird es in der Form

<tag>Inhalt</tag>

dargestellt. **<tag>** bedeutet den Anfang und **</tag>** das Ende des Elements.

Hat ein Element keinen Inhalt, wird ausschließlich ein **<tag/>** verwendet.

Ein Element kann nicht nur Text als Inhalt haben, sondern auch andere Elemente, die hierarchisch angeordnet werden. Elemente können außerdem noch Attribute enthalten, die diese spezifizieren. Ein Attribut kann nur einmal pro Element verwendet werden. Im unteren Beispiel sieht man die Verschachtelung von Elementen mit ihren Attributen, die im Tag hinter dem Elementnamen stehen.

```
<element>
    <subelement1 attribut="Attributinhalt">
        <subelement2>
            Inhalt des Subelements2
        </subelement2>
        <subelement3 attribut2="Attributinhalt2"/>
    </subelement1>
    <subelement4>
        Inhalt des Subelements4
    </subelement4>
</element>
```

Ein XML-Dokument ist nur brauchbar, wenn es wohlgeformt (well-formed) ist. Um diese Bedingung zu erfüllen, muss es einen zulässigen Prolog haben, mindestens ein Element besitzen und mit seinen Tags hierarchisch gegliedert sein. Letzteres bedeutet, dass die Elemente richtig geschachtelt sind und zu jedem Start-Tag ein passendes End-Tag gehört.

Damit ein XML-Dokument gültig (valid) ist, muss es sowohl wohlgeformt sein als auch zu der im Prolog angegebenen DTD konform sein. [6]

2.1.1.2Document Type Definition (DTD)

Einige XML-Dokumente haben eine zugehörige DTD, die ihre Struktur beschreibt. Eine DTD kann im XML-Dokument selbst stehen oder als eigene Datei mit der Endung **.dtd** existieren, die im Prolog des XML-Dokuments angegeben wird. Eine solche Datei spezifiziert die Menge der zulässigen Elemente, Attribute und deren hierarchische Anordnung.

Die Beschreibung eines Elementtyps hat die Form

<!ELEMENT elementname (elementtyp | subelementname)>.

Ein einfaches Element wird in der Form

<!ELEMENT elementname (#PCDATA)>

dargestellt. Der Elementtyp **#PCDATA** ist die Abkürzung für Parsed Character Data und bedeutet, dass der Inhalt des Elements aus einem einfachen Text besteht.

Ist dieser Inhalt leer, lautet die Beschreibung durch die DTD

<!ELEMENT elementname EMPTY>.

Ein Element kann aber nicht nur Text enthalten, sondern kann auch andere Elemente besitzen, die in einer Klammer in der Reihenfolge, in der sie im Dokument stehen sollen, angegeben werden.

<!ELEMENT elementname(subelement1, subelement2)>

Steht hinter einem Subelement ein Pluszeichen, bedeutet dies, dass das Element mindestens einmal vorkommt. Ein Stern besagt, dass es keinmal, einmal oder mehrmals auftreten kann. Befindet sich dort ein Fragezeichen, wird dieses Element einmal oder gar nicht existieren. Das Komma ist ein Verknüpfungsoperator, der bei einer Sequenz verwendet wird. Das ODER wird durch einen Strich **|** beschrieben.

Es gibt nicht nur einfache Elemente mit entweder Text oder Subelementen, sondern auch wie im folgenden Beispiel komplexe Verknüpfungen, die beides enthalten.

<!ELEMENT elementname(subelement1,(subelement2 | elementtyp)*)>

Das Element **elementname** enthält das Element **subelement1**. Zusätzlich kann es mindestens ein weiteres Element oder einen Elementtypen besitzen.

Kann ein Element beliebige Subelemente enthalten, wird dies mit dem Elementtyp **ANY** angegeben.

Wie vorher schon erwähnt, gibt es nicht nur Elemente sondern auch Attribute, die auch in der DTD beschrieben werden. Eine Attributbeschreibung ähnelt der eines Elements.

```
<!ELEMENT elementname (elementtyp)>
<!ATTLIST elementname   attributname attributtyp #OPTION
                        attributname2 attributtyp #OPTION>
```

An erster Stelle steht das zugehörige Element, dahinter folgt das Attribut mit seiner Spezifikation. Es können beim Attribut bestimmte Werte vorgegeben werden, wie z.B.

Währungen oder beliebige Werte. Attribute können vom Attributtyp **CDATA**, **ID**, **IDREF**, **IDREFS**, **NMTOKEN** oder **NMTOKENS** sein. **CDATA** besteht aus Text, **ID** gibt eine eindeutige Bezeichnung (ID) an, **IDREF** verweist auf eine an anderer Stelle vorkommende ID und **IDREFs** referenziert auf mehrere.

```
<!ATTLIST zielelement id ID>
<!ATTLIST linkelement ziel IDREF>
```

Das XML-Dokument würde folgende Gestalt haben:

```
[...]
<linkelement ziel="id">Verweis auf Zielelement</linkelement>
[...]
<zielelement id="id">Ziel</zielelement>
```

Wenn man nun auf das Element **linkelement** klicken würde, würde man zu dem Element **zielelement** gelangen. Wichtig ist, dass eine **ID** eindeutig ist und eine **IDREF** ein existierendes Ziel hat.
Ein Attribut vom Typ **NMTOKEN** besteht aus einem String ohne Leerzeichen, während **NMTOKENS** mehrere **NMTOKEN** enthält, die durch ein Leerzeichen getrennt sind.
Die Optionen geben an, wie ein Attribut im XML-Dokument behandelt wird. Steht an dieser Stelle **#IMPLIED**, ist die Angabe des Attributs im Dokument optional. **#REQUIRED** fordert die Existenz des Attributs und **#FIXED** besagt, dass der Wert des Attributs vorgegeben ist. [6]

2.1.1.3Namespaces und XML Schema

Bei größeren Anwendungen, wie bei einem Web-Browser, kann es vorkommen, dass es Elemente mit gleichen Namen gibt. Zum Beispiel benutzt HTML das Element **<b>**, aber es kann auch XML-bearbeitende Software vorkommen, die auch das Element **<b>** verwendet. Um diese gleichen Elementnamen auseinander halten zu können, gibt es im XML die Möglichkeit Namensräume (Namespaces) einzusetzen. Ein Namensraum wird immer in einem Element, üblicherweise dem Wurzelelement, definiert und ist für alle untergeordneten Elemente gültig. Er besteht aus einer Uniform Resource Identifier (URI), die im folgenden Beispiel [1] auf eine Seite des W3C verweist.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="ConfigInfo">
        <xsd:complexType>
        ...
<xsd:schema>
```

Damit erkennbar ist, dass die Elemente zu dem angegebenen Namensraum gehören, wird jedem Element das vorher definierte Kürzel zugewiesen. In diesem Beispiel lautet es **xsd**, weil es zu XML Schema gehört. Es kann aber in einer anderen Anwendung auch einen anderen Namen haben. In einem Dokument können mehrere Namensräume verwendet werden, deren Elemente im Dokument gemischt werden können.
Wie im oben angegebenen Beispiel werden bei XML Schema Namensräume verwendet. XML Schema ist eine Schemabeschreibungssprache und hat wie die DTDs die Funktion Strukturen für XML-Dokumenten festzulegen. Allerdings verwendet sie im Gegensatz zur DTD die XML-Syntax und enthält wesentlich detailliertere Informationen. XML Schema besitzt primitive Datentypen wie beispielsweise **string**, **boolean**, **float** oder **time**. Mit XSD (XML Schema Definition) ist es aber auch möglich Datentypen selbst zu definieren. Diese Datentypen können sowohl einfach (**simpleType**) als auch komplex (**complexType**) sein. Einfache Typen können durch Einschränkung und Listenbildung abgeleitet werden. Hier folgt ein Beispiel für eine Einschränkung:

```
<xsd:simpleType name="koerpergroesse">
    <xsd:restriction base="xsd:float">
        <xsd: minExclusive value="0"/>
    <xsd:restriction>
</xsd:simpleType>
```

Ein Element vom Typ **koerpergroesse** darf nicht kleiner als 0 sein. Der Typ **float** wird also auf den nicht-negativen Bereich verkürzt. Eine andere Form der Einschränkung ist die Aufzählung (Enumeration).

```
<xsd:simpleType name="antwortmoeglichkeiten">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="ja"/>
        <xsd:enumeration value="nein"/>
    <xsd:restriction>
</xsd:simpleType>
```

Hier werden die erlaubten Antwortmöglichkeiten **ja** und **nein** vordefiniert.
Komplexe Typen sind zusammengesetzte oder mehrwertige Datentypen. Sie können durch die Erweiterung eines einfachen Datentyps, Einschränkung oder Neudefinition eines Datentyps entstehen und einen einfachen (**simpleContext**) oder komplexen Inhalt (**complexContent**) besitzen. Es folgt ein Beispiel für eine Erweiterung eines einfachen Typs durch ein Attribut:

```
<xsd:complexType name="durchschnittsgroesse">
    <xsd:extension base="koerpergroesse">
        <xsd:simpleContent>
            <xsd:attribute name="tierart" type="string"/>
        </xsd:simpleContent>
    </xsd:extension>
</xsd:complexType>
```

Der komplexe Typ **durchschnittsgroesse** ist eine Erweiterung des Typs **koerpergroesse** um das Attribut **tierart**. In einem XML-Dokument würde ein Element vom Typ **durchschnittsgroesse** folgendes Aussehen haben.

<durchschnittsgroesse tierart="katze">0,40</durchschnittsgroesse>

Ein komplexer Typ mit komplexem Inhalt hat folgende Form:

```
<xsd:complexType name="frage">
    <xsd:attribute name="gebiet">
    <xsd:sequence>
            <xsd:element name="id" type="integer"/>
            <xsd:element name="fragetext" type="string"/>
            <xsd:element ref="antwortmoeglichkeiten"/>
    </xsd:sequence>
</xsd:complexType>
```

Beim komplexen Typ mit komplexem Inhalt kann die Angabe **<xsd:complexContent>** weggelassen werden. Ein Typ vom Element **frage** hat ein Attribut **gebiet** und die Subelemente **id**, **fragetext** und **antwortmoeglichkeiten**. Letzteres ist eine Referenzierung auf den vorher definierten Typ **antwortmoeglichkeiten**.[6]

2.1.1.4Parsen von XML

Um XML-Dokumente beim Datenaustausch zu verwenden, benötigt man ein Programm, das einen Zugriff auf diese Daten ermöglicht. Ein solches Programm wird XML-Parser genannt und kann Daten aus einem XML-Dokument lesen. Es gibt zwei verschiedene Arten von Parsern, der SAX- (Simple API for XML) und der DOM-Parser (Document Object Model). *„SAX-Parser teilen dem Benutzer bestimmt Ereignisse bei Parsen mit. Dazu gehören zum Beispiel das Erreichen des Anfangs und des Endes jedes Elements. Durch diese Art der Abarbeitung braucht der Parser nur wenig Speicher und ist auch sehr schnell.“*[1] In der folgenden Abbildung ist die Architektur eines SAX-Parsers dargestellt. Die API und der SAX-Parser werden vom Softwarehersteller mitgeliefert. Bei Java sind dies der Xerces-Parser oder die JAXP-Referenzimplementation Crimson.

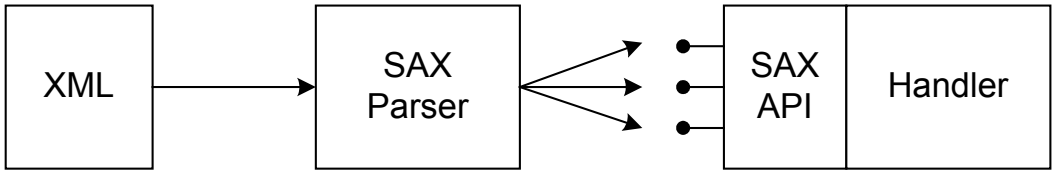

Abbildung 2-1 Architektur eines SAX-Parser. Aus [1]

Mit dem DOM-Parser kann gezielt auf einzelne Teile eines Dokuments zugegriffen werden. Im Gegensatz zu SAX, wo eine Anwendung Informationen schon während des Parsens erhält, wird hier erst, nachdem das gesamte Dokument verarbeitet worden ist, ein XML-Baum der Anwendung übergeben. Danach kann die Anwendung über die DOM-Schnittstelle auf den in Objekten gespeicherten XML-Baum zugreifen. Die Architektur eines DOM-Parsers hat folgende Gestalt.

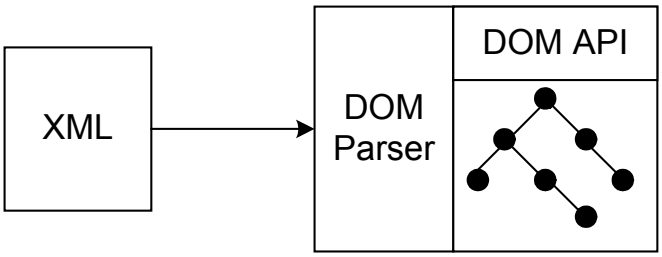

Abbildung 2-2 Architektur eines DOM-Parser. Aus [1]

Die Speicherung des kompletten XML-Baums ist ein Vorteil, da ständig auf alle Dokumentteile zugegriffen werden kann, während bei SAX wichtige Informationen in Variablen für den späteren Gebrauch zwischengespeichert werden müssen. Aber durch die Speicherung des gesamten Baums wird der Speicherverbrauch von DOM-Parsern sehr hoch. Aus diesem Grund sollte er bei großen Dokumenten nicht eingesetzt werden.[1]

2.1.2 Web Services

Hier werden Web Services anhand eines Beispiels erklärt. Außerdem werden das Nachrichtenformat SOAP, die Beschreibungssprache WSDL und der Verzeichnisdienst UDDI erläutert.

2.1.2.1 Definition von Web Services

Wenn man sich nach Web Services erkundigt, findet man viele Definitionen. Manche sind sehr unpräzise und behaupten, dass ein Web Service jede Anwendung sein kann, die man im Internet aufrufen kann. Andere wiederum definieren einen Web Service, indem sie Standards setzen. Zu letzteren gehört das W3C (World Wide Web Consortium) [9],

welches sowohl für Web Services als auch für XML und andere Internettechnologien offizielle Standarddefinitionen herausbringt.

Die Definition des W3Cs für einen Web Service lautet:

„Ein Web Service ist eine durch einen URI eindeutig identifizierte Softwareanwendung, deren Schnittstellen als XML-Artefakte definiert, beschrieben und gefunden werden können. Ein Web Service unterstützt die direkte Interaktion mit anderen Softwareagenten durch XML-basierte Nachrichten, die über Internetprotokolle ausgetauscht werden.“[1]

Das bedeutet, dass jeder Web Service eine Schnittstelle hat, die auf einem standardisierten XML-basierenden Protokoll basiert. Dies hat den großen Vorteil, dass es praktisch jedem problemlos ermöglicht wird, auf den Web Service zuzugreifen. So verhindert der Standard des Protokolls Inkompatibilitäten, die bei anderen Web-Applikationen aufgrund verschiedener Programmiersprachen auftreten. Außerdem ist es nicht sehr aufwendig XML-Kode zu verarbeiten, denn der Benutzer bekommt klar strukturierte Daten, die er nicht mehr auf Verwendbarkeit prüfen muss, sondern sie sofort verwerten kann. Am Beispiel aus [1] soll der Vorteil der Kommunikation mit Web Services erläutert werden.

Ein Geschäftsmann möchte ein Online-Reisebüro eröffnen, dabei aber nicht eine eigene Datenbank mit Flügen, Hotel- und Mietwagenangeboten erstellen, auf die die Kunden später zugreifen sollen. Diese sind nämlich schon in vielfacher Form vorhanden. Er möchte stattdessen direkt die Daten der Fluggesellschaften nutzen und alle auf einmal anbieten, damit jeder Reisesuchende sein Online-Reisebüro besucht. Nun besteht für ihn die Möglichkeit, jeden Tag auf den Webseiten der Reiseunternehmen vorbeizuschauen und deren Daten in sein System einzugeben. Dies würde aber zuviel Zeit verbrauchen und es könnte passieren, dass die Daten nach ein paar Stunden nicht mehr aktuell sind. Eine weitere Möglichkeit wäre der Zugriff auf die Datenbanken der Fluggesellschaften, aber diese werden kaum jemandem den Zugriff auf ihre Datenbanken gestatten. Wenn es möglich wäre, gäbe es noch das Problem, dass die Sprache, mit der die eigene Anwendung geschrieben worden ist, nicht zu der der Fluggesellschaften passen könnte. Es müssten deshalb sehr viele Schnittstellen implementiert werden, um mit allen Datenbanken zu kommunizieren.

Wesentlich einfacher wäre es, wenn er von den Fluggesellschaften Daten über das Internet abrufen könnte, von denen er genau wüsste, was sie bedeuten. Er müsste also nicht auf den internen Datenbestand eines Unternehmens zugreifen, sondern könnte sie von einem öffentlichen Web-Server bekommen. Solch eine Dienstleistung eines Unternehmens, die Daten über das einfache XML-basierte standardisierte Protokoll Simple Object Access Protocol (SOAP) anzubieten, nennt man Web Service.

Wenn nun jede Fluggesellschaft ihre Daten als Web Service anbieten würde, könnte der Geschäftsmann ganz einfach mit einem Klientenprogramm auf diese zugreifen und sie auf

seiner Webseite für seine Kunden anbieten. Um die Web Services zu finden, kann er bei einem Verzeichnisdienst (Universal Description, Discovery and Integration (UDDI)) nachschauen, bei dem sich die Web Services anmelden können. Dort sind sie in der Beschreibungssprache Web Service Description Language (WSDL) zu finden, die dem Benutzer ihre Funktionsweise erläutert.

2.1.2.2Simple Object Access Protocol (SOAP)

Um miteinander zu kommunizieren, benutzen Web Services das vom World-Wide-Web-Consortium (W3C) standardisierte Protokoll SOAP.
„SOAP ist ein leichtgewichtiges Protokoll, dessen Zweck der Austausch strukturierter Information in einer dezentralisierten verteilten Umgebung ist. Dazu wird auf der Basis von XML ein Rahmenwerk für den Austausch von Nachrichten beschrieben, wobei diese Nachrichten über eine Auswahl verschiedener Transportprotokolle übertragen werden können. Das Nachrichtenformat ist komplett anwendungsunabhängig, aber natürlich ist es möglich, anwendungsspezifische Daten in einer SOAP-Nachricht zu transportieren.“[1]
Bei der Entstehung von SOAP war das Ziel, ein sehr einfaches, aber erweiterbares Protokoll zu schaffen. Deshalb werden in der Grundausstattung dieses Protokolls Fragen wie Sicherheit, Verlässlichkeit und Routing von Nachrichten vernachlässigt. Trotzdem ist es möglich SOAP zu erweitern. Die aktuelle Version des SOAP ist im Moment die Version 1.2, die am 24. Juni 2003 vom W3C standardisiert wurde.

Eine SOAP-Nachricht hat eine geschachtelte Struktur. Die gesamte Nachricht wird von einem SOAP-Umschlag (Envelope) umrahmt. Dieser Umschlag enthält den SOAP-Kopf (Header) und den SOAP-Körper (Body). Im Kopf stehen optionale Angaben, die nicht direkt verwendet werden, sondern eher als Metadaten dienen. Dies können zum Beispiel Transaktionsnummern von Abrechnungs- oder von Authentifikationsinformationen sein, wenn die SOAP-Nachricht zu einer bestimmen gerade laufenden Transaktion gehört. Im Körper dagegen befinden sich die eigentlichen Informationen, die zwischen zwei Komponenten einer verteilten Anwendung ausgetauscht werden sollen. In der folgenden Abbildung ist die Struktur einer SOAP-Nachricht dargestellt.

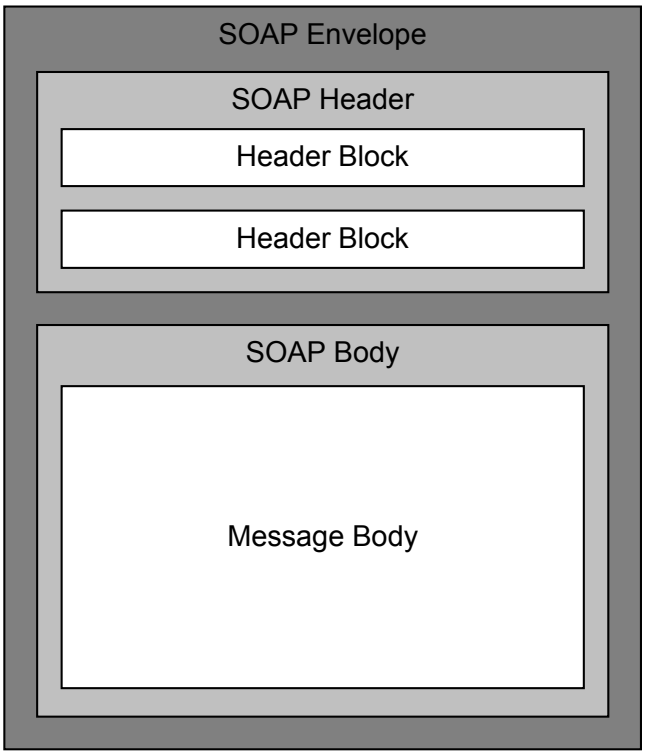

Abbildung 2-3 Struktur einer SOAP-Nachricht [1]

Eine SOAP-Nachricht basiert auf XML und verwendet einen eigenen Namensraum, der Eindeutigkeit garantiert und die Identifikation von SOAP erleichtert. Der unten angegebene Kode [1] beschreibt den SOAP-spezifischen Aufbau einer solchen Nachricht.

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
        …
    </env :Header>
    <env:Body>
        …
    </env:Body>
</env:Envelope>
```

Werden Nachrichten mit SOAP ausgetauscht, werden die Kommunikationspartner als SOAP-Knoten (SOAP nodes) bezeichnet. Der Sender der Nachricht wird mit „SOAP sender“ und der Empfänger mit „SOAP receiver“ benannt.
Wenn man das vorherige Beispiel mit den Fluggesellschaften mit SOAP weiterführt und den Online-Reisebürogründer einer Fluggesellschaft mitteilen lässt, einen bestimmten Flug zu buchen, würde der Online-Reisebürogründer der Sender sein und die Fluggesellschaft der Empfänger. Die SOAP-Nachricht würde folgende Gestalt [1] haben.

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
        …
    </env:Header>
    <env:Body>
        <r:reservierung xmlns:r="http://www.web-air.de/reservierung">
            <r:hinflug>
                <r:abflugort>Frankfurt</r:abflugort>
                <r:ankunftsort>Los Angeles</r:ankunftsort>
                <r:abflugdatum>2001-12-14</r:abflugdatum>
                <r:abflugzeit>10:25</r:abflugzeit>
                <r:sitzPraeferenz>Fenster</r:sitzPraeferenz>
            </r:hinflug>
            <r:rueckflug>
                ...
            </r:rueckflug>
        </r:reservierung>
    </env:Body>
</env:Envelope>
```

Der anwendungsspezifische Bereich wird am vom Reisebüro selbst definierten eigenen Namensraum erkannt. Hier wird nur ein Datensatz übertragen, d.h. es wird keine entfernte Operation aufgerufen. SOAP kann aber auch in Remote Procedure Call (RPC) Form verwendet werden. [1]

2.1.2.3Web Service Description Language (WSDL)

WSDL ist eine Sprache, die Web Services beschreibt. Sie basiert genauso wie SOAP auf XML und wird von einem Teil des W3C der Web Service Description Working Group standardisiert. Allerdings besteht noch kein offizieller WSDL-Standard, sondern nur ein so genanntes „Working Draft[10]“ in der Version 1.2. Diese Version unterstützt XML Schema und ermöglicht eine klare Definition von Bindungen, beispielsweise wie in diesem Fall an HTTP1.1. Einige Anforderungen, die zur Entstehung dieser Version beitrugen waren, dass die Sprache kein Programmiermodell oder Transportprotokoll vorwegnehmen darf, die Dienste in XML beschrieben werden, XML Schema unterstützt wird und WSDL selbst als XML Schema definiert ist.

WSDL legt die Schnittstellen eines Web Service nach außen hin offen und bekommt damit den Stellenwert des Interfaces in objektorientierten Sprachen. WSDL zeigt die Syntax des Aufrufs an und lässt die Semantik für den Klienten unsichtbar, d.h. ein Klient kann mit diesen Informationen den Web Service aufrufen, ohne wissen zu müssen, was im Dienst passiert. [2]
Eine solche Dienstbeschreibung für einen Web Service ist hierarchisch aufgebaut und besteht aus sechs Abschnitten **`definitions`**, **`types`**, **`message`**, **`interface`**, **`binding`** und **`service`**. Sie enthält einen abstrakten und einen konkreten Teil. Im abstrakten Teil werden der Name des Dienstes und die Nachrichten, mit denen man den Dienst

ansprechen kann, und seine Antwortnachrichten beschrieben. Seine Elemente sind **types**, **message** und **portType**. Im konkreten Teil befindet sich eine Beschreibung der Protokollbindung (**binding**) und die Adresse des Dienstes (**service**).
Eine WSDL-Beschreibung beschreibt die Nachrichten, die zwischen dem Diensterbringer und dem Dienstnutzer ausgetauscht werden. Diese Nachrichten, welche Sammlungen von Datenelementen sind, werden abstrakt beschrieben, müssen aber an ein bestimmtes Übertragungsprotokoll und an eine bestimmte Kodierung gebunden werden. Eine solche Datenübertragung wird in WSDL mit Operation bezeichnet. Ein Interface enthält Operationen und "*wird durch eine oder mehrere Bindungen wiederum mit einem Protokoll bzw. einem Nachrichtenformat assoziiert. Eine solche Bindung und damit auch das Interface sind über einen Endpunkt zugänglich, wobei jeder Endpunkt seine eigene URI besitzt. Der Service selbst ist schließlich eine Sammlung von Endpunkten, die an dasselbe Interface gebunden sind. Um all diese Dinge beschreiben zu können, wurden entsprechende XML-Elemente definiert. "*[1]

Die Beschreibung der Grundstruktur eines WSDL-Dokuments wird in folgender Form angegeben: (Die Beispiele in diesem Kapitel sind, soweit nicht anders angegeben, aus [1] entnommen.)

```
<definitions targetNamespace="xs:anyURI">
    <documentation />?
    [ <import /> | <include /> ]*
    <types />?
    [ <message /> | <interface /> | <binding /> | <service /> ]*
</definitions>
```

Das **<definitions>**-Element ist das Wurzelelement und dient als Container für die Servicebeschreibung. Es beinhaltet das Attribut **targetNamespace**, das den Namensraum des definierten Web Services identifiziert. Die Angabe des Namensraums des WSDL-Schemas selbst ist notwendig und wird mit dem Präfix **wsdl** bezeichnet. Das Element **<definitions>** kann die Subelemente **<documentation>** und **<types>** einmal oder gar nicht beinhalten. Die Subelemente dieser beiden können beliebig oft oder gar nicht vorkommen.

Hier folgt ein Beispiel für ein **<definitions>**-Element des Online-Reisebüros:

```
<wsdl:definitions
    targetNamespace="http://www.web-air.de/Buchung/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schema.xmlsoap.org/wsdl/soap/"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:s0="http://www.web-air.de/Buchung/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:wsdl="http://www.w3.org/2003/06/wsdl">
```

Hier werden viele Namensräume definiert. Für das unten folgende Beispiel ist der Namensraum **s** wichtig, der zur Sprache XML Schema gehört. WSDL bedient sich XML Schema, um einfache und komplexe Datentypen zu beschreiben. Das für die Datentypen zugehörige Element wird mit **<types>** bezeichnet und enthält die Datentypen, die später im Abschnitt **message** beim Datenaustausch zwischen Klient und Service angewendet werden. Dieses Element ist optional, da es nur komplexe Typen beschreibt. Die einfachen Datentypen wie **integer**, **float**, **double**, **date** und **string** sind nämlich schon in XML Schema definiert. [3]
Hier nun die Weiterführung des obigen Beispiels:

```
<wsdl:types>
        <s:schema elementFormDefault="qualified"
            targetNamespace="http://www.web-air.de/Buchung/">
        <s:element name="reserviere">
            <s:complexType>
                <s:sequence>
                    <s:element minOccurs="0" maxOccurs="1"
                        name="flugnummer" type="s:string" />
                    <s:element minOccurs="1" maxOccurs="1" name="sitze"
                        type="s:int" />
                    <s:element minOccurs="1" maxOccures="1" name="date"
                        type="s:dateTime" />
                </s:sequence>
            </s:complexType>
        </s:element>
        <s:element name="reserviereResponse">
            <s:complexType>
                <s:sequence>
                    <s:element minOccurs="0" maxOccurs="1"
                        name="reserviereResult" type="s:string" />
                </s:sequence>
            </s:complexType>
        </s:element>
        <s:element name="string" nillable="true" type="s:string" />
    </s:schema>
</wsdl:types>
```

Wie zu erkennen, gehören alle Elemente zur Sprache XML Schema. In der Schema-Wurzel **<s:schema>** werden zwei komplexe Typen **reserviere** und

reserviereResponse definiert. *„Eine Besonderheit hat es mit diesem dritten Element „vom einfachen Typ **string**“ auf sich: Es enthält das Attribut **nillable="true"**. Damit ist es möglich, auch einen Null-String zurückzugeben und bei der Interpretation einen Unterschied zu machen zu einem leeren String. Somit bedeutet **<string/>** die leere Zeichenkette, während **<string xsi:nil ="true" />** mit einem Null-Pointer gleichzusetzen ist.“* [1]
Ein **<message>**-Element beschreibt eine Nachricht, die zwischen Klient und Web Service beim Aufruf einer Operation ausgetauscht wird. Diese Elemente enthalten Subelemente vom Typ **<part>**, welche auf einen in **<types>** definierten komplexen Typ verweisen oder einen einfachen Datentyp darstellen. Solche Elemente sind Methodenparameter oder Rückgabewerte. [3]

Im Flugbuchungsbeispiel werden komplexe und einfache Datentypen versendet. Als erstes werden die beiden komplexen Datentypen benutzt, die bei einer Bindung an SOAP verwendet werden.

```
<message name="reserviereSoapIn">
    <part name="parameters" element="s0:reserviere" />
</message>
<message name="reserviereSoapOut">
    <part name="parameters" element="s0:reserviereResponse" />
</message>
```

Der Elementname wird mit **parameters** bezeichnet und der Typ der beiden Elemente ergibt sich aus den in **<types>** definierten Typen **reserviere** und **reserviereResponse**.
Beim einfachen Datentypen vom Typ **string** beziehen sich diese auf eine Bindung an HTTP GET oder HTTP POST. Sie werden rein in HTTP beschrieben und enthalten auf dem Request-Weg keine SOAP-Datenstrukturen. Hier werden keine komplexen Datenstrukturen benutzt, sondern jeder Parameter einzeln definiert.

```
<message name="reserviereHttpGetIn">
    <part name="flugnummer" type="s:string" />
    <part name="sitze" type="s:string" />
    <part name="date" type="s:string" />
</message>
<message name="reserviereHttpPostIn">
    <part name="flugnummer" type="s:string" />
    <part name="sitze" type="s:string" />
    <part name="date" type="s:string" />
</message>
```

Als Antwort wird immer nur ein wie oben definierter String zurückgeliefert, der als SOAP-Nachricht kodiert wird und deshalb hier mit dem Namen **Body** versehen wird.

```
<message name="reserviereHttpGetOut">
    <part name="Body" element="s0:string" />
</message>
<message name="reserviereHttpPostOut">
    <part name="Body" element="s0:string" />
</message>
```

Den letzten Teil des abstrakten Teils bilden die Interfaces. Zu einem Web Service können mehrere Interfaces gehören, von welchen jedes optionale **<documentation>**-Elemente besitzen kann. Ein Interface definiert von außen sichtbare Operationen (**<operation>**) eines Web Services. Eine Operation kann Elemente für die Eingabewerte (**input**) und Rückgabewerte (**output**) enthalten. Ein Musterattribut (**pattern**) legt die Abfolge einer Operation fest.
Ein Interface hat folgende Gestalt:

```
<interface name="BuchungSoap">
    <operation name="reserviere"
        pattern="http://www.w3.org/2003/06/wsdl/request-response">
        <input message="s0:reserviereSoapIn" />
        <output message="s0:reserviereSoapOut" />
    </operation>
</interface>
<interface name="BuchungHttpGet">
    <operation name="reserviere"
        pattern="http://www.w3.org/2003/06/wsdl/request-resonse">
        <input message="s0:reserviereHttpGetIn" />
        <output message="s0:reserviereHttpGetOut" />
    </operation>
</interface>
<interface name="BuchungHttpPost">
    <operation name="reserviere"
        pattern="http://www.w3.org/2003/06/wsdl/request-response">
        <input message="s0:reserviereHttpPostIn" />
        <output message="s0:reserviereHttpPostOut" />
    </operation>
</interface>
```

Das Attribut der Ein- und Rückgabeelemente **message** verweist auf die vorher definierten Nachrichten.
Die URI **http://www.w3.org/2003/06/wsdl/request-response** des Musterattributs **pattern** steht für Request-Response und bedeutet, dass einer **<input>**-Nachricht eine **<output>**-Nachricht folgt, die entweder eine normale Nachricht oder eine Fehlermeldung ist. Die **<output>**-Nachricht wird auf demselben Kanal zurückgeschickt, auf dem die **<input>**-Nachricht gekommen ist. Ein Request-Response-Verfahren wird immer dann verwendet, wenn der Klient eine Anfrage an den Web Service schickt und dieser eine Nachricht als Rückantwort gibt.
Es gibt noch drei andere Muster. Das Solicit-Response-Verfahren ist genau andersherum. Hier schickt der Web Service dem Klienten eine Nachricht. In der Operation steht also das **<output>**-Element vor dem **<input>**-Element. Beim One-Way-Verfahren gibt es nur

einen Eingabeparameter vom Klienten ohne eine Antwort des Web Services. Beim Notification-Verfahren erhält der Klient vom Web Service eine Nachricht, die aber keine Antwort auf eine Anfrage ist. Die Operation enthält dabei nur ein **<output>**-Element.
Die Definition der Operationen ist der letzte Teil des abstrakten Teils einer WSDL-Beschreibung.

Den Beginn des konkreten Teils, in dem die abstrakte Beschreibung des Web Services an tatsächliche Nachrichtenformate, Kodierungen (Encodings) und Protokolle gebunden wird, bildet das Element **<binding>**. Eine Bindung hat folgende Gestalt:

```
<binding name="BuchungSoap" type="s0:BuchungSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
            style="document" />
    <operation name="reserviere">
        <soap:operation                          soapAction="http://www.web-
air.de/Buchung/Buchung.asmx?reserviere" style="document" />
            <input>
                <soap:body use="literal" />
            </input>

            <output>
                <soap:body use="literal" />
            </output>
    </operation>
</binding>
```

Das Element **<binding>** umschließt eine Operation. Der Name der Bindung wird mit dem Attribut **name** und ihr Typ mit dem Attribut **type** festgelegt. Alle weiteren Angaben von Nachrichten und Operation werden sich nun auf das angegebene Interface beziehen. Das folgende Element besagt, dass es sich um eine SOAP-Bindung handelt und die Verwendung von SOAP-Nachrichten vorgibt. Das Attribut **transport** legt die Art und Weise des Transports der Nachrichten fest. In diesem Fall werden sie über HTTP übermittelt. Das SOAP Message Exchange Pattern wird durch das Attribut **style** angegeben. Nach diesem Element folgt die Definition des Formats der Interface-Operationen. Die Ein- und Rückgabe wird im Körper (Body) einer Nachricht übermittelt. Werden Nachrichten nicht über SOAP, sondern direkt über HTTP GET und HTTP POST übertragen, wird anstatt **soap http** im Binding eingesetzt.
Nun folgt das letzte Element des konkreten Teils. Es heißt **service** und bestimmt, an welche tatsächlichen Netzwerkadressen sich ein Klient wenden muss, um mit dem Web Service zu kommunizieren. Außerdem legt es fest, wie diese Kommunikation stattfindet. Ein Service kann mehrere Endpunkte besitzen, von welchen jeder mit einer Bindung assoziiert ist.
Der Rest des WSDL-Beispiels wird hier aufgeführt:

```
<service name="Buchung">
    <endpoint name="BuchungSoap" binding="s0:BuchungSoap">
        <soap:address                          location="http://www.web-
air.de/Buchung/Buchung.asmx" />
    </endpoint>
    <endpoint name="BuchungHttpGet" binding="s=:BuchungHttpGet">
        <http:address                          location="http://www.web-
air.de/Buchung/Buchung.asmx" />
    </endpoint>
    ...
</service>
```

Beim ersten Endpunkt handelt es sich um einen SOAP-Zugangspunkt, beim anderen um einen von HTTP. Diese Beschreibung des Dienstes war das letzte Element. Nun ist das WSDL-Dokument vollständig.

2.1.2.4Universal Description, Discovery and Integration (UDDI)

Da viele Web Services über das Internet angeboten werden, gestaltet es sich schwierig, sie zu finden. Aus diesem Grund wurde der Vermittlungsmechanismus UDDI, der Verbindungen zwischen Klienten und Web Services herstellen kann, von IBM, Microsoft und Ariba erstellt und ist heute De-facto-Standard für XML-Verzeichnisse von Web Services. Die aktuelle Version von UDDI ist 3.0 und wurde im Juli 2002 verabschiedet.
UDDI selbst ist eine Web Service-Anwendung, verwendet SOAP als Kommunikationsprotokoll und hat die Rolle eines Verzeichnisdienstes. Solch ein Namens- bzw. Verzeichnisdienst hat eine vorgegebene Schnittstelle, die aber auch bei jedem Dienst in ähnlicher Form vorzufinden ist.
Eine solche Schnittstelle enthält die Funktionen **`bind`**, **`rebind`**, **`delete`**, **`lookup`** und **`list`**. Die Funktion **`bind`** verbindet die Objekte mit ihren Attributen. Bei einem Namensdienst würde der Namen mit der Adresse verknüpft werden. Die Funktion **`rebind`** überschreibt die alten Attribute eines Objekts mit den neuen. Wenn eine solche Verbindung nicht geändert, sondern gelöscht werden soll, wird die Funktion **`delete`** (oder **`unbind`**) verwendet. Um einen Dienst zu finden, können die Funktionen **`lookup`**, die anhand angegebener Attribute einen Dienst sucht, und **`list`**, die alle in der Datenbank gespeicherten Dienste anzeigt, benutzt werden. In der folgenden Abbildung ist die Funktionsweise eines Verzeichnisdienstes bzw. Namensdienstes dargestellt.

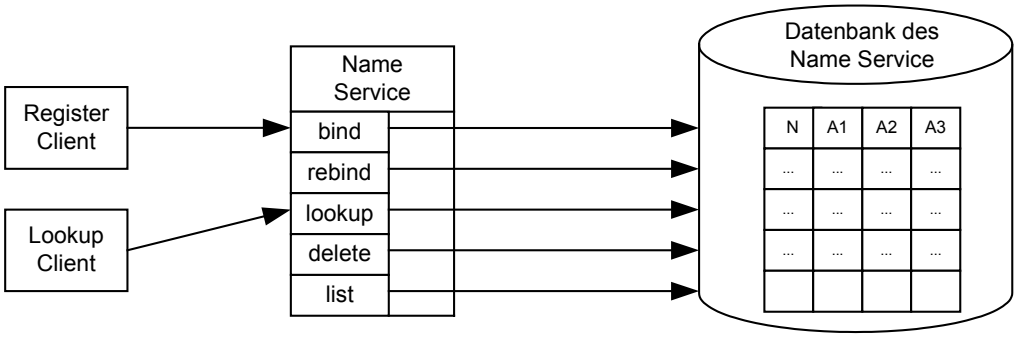

Abbildung 2-4 Funktionsweise eines Namensdienstes [1]

Der Register Klient ist für die Bindung der Dienste zuständig, der Lookup Klient ruft die Funktion **`lookup`** des Namensdienstes auf, um Dienste zu finden. Die Funktionen des Namensdienstes sind mit der Datenbank verbunden und verwalten deren Daten.
Bei UDDI gibt es drei Arten von Verzeichnisdiensten. Es verwaltet die Weißen Seiten (White Pages), die Gelben Seiten (Yellow Pages) und die Grünen Seiten (Green Pages). Bei den Weißen Seiten handelt es sich um die Sammlung von Namen, Beschreibungen und sämtlichen Kontaktinformationen der Unternehmen, die Web Services publiziert haben. Bei den Gelben Seiten werden Unternehmen mit ihren veröffentlichten Services nach ihren Kategorien, wie in einem Branchenbuch, geordnet. Die Grünen Seiten erweitern die Gelben Seiten noch um technische Zusatzinformationen wie WSDL-Dokumente, die die Web Services beschreiben.
Mit Hilfe dieser UDDI-Registries können die Benutzer sich genügend Informationen über die verschiedenen angebotenen Web Services holen und diese nutzen. Umgekehrt kann ein Anbieter von Web Services seine Dienste dort anmelden, damit andere auf sie zugreifen können.
Die Registries werden bei UDDI in zwei Arten aufgeteilt. Es gibt die öffentlichen Registries, die per URI erreichbar sind. Diese können von jedem meistens kostenlos genutzt werden. Die privaten Registries sind nicht öffentlich verfügbar und werden nur für einen bestimmten Zweck, beispielsweise im Intranet einer Firma, eingesetzt. Einzelne UDDI-Registries werden zu einem UDDI Business Registry (UBR) zusammengeschlossen, die durch ihre Synchronisierung untereinander immer auf aktuellem Stand gehalten werden. Der Zugriff auf UDDI-Verzeichnisse kann sowohl über eine reine Maschinenkommunikation, über die vorher beschriebene Schnittstelle, als auch über einen Web-Browser, bei dem sich ein Interessent Informationen holen kann, erfolgen.

Das Datenmodell von UDDI basiert auf XML und besteht aus den sechs Datenstrukturen **`<businessEntity>`**, **`<businessService>`**, **`<bindingTemplate>`**. **`<tModel>`**, **`<publisherAssertion>`** und **`<subscription>`**.

Die **<businessEntity>** repräsentiert eine Organisation, deren Attribute Namen, Aufgaben, Ziele, Telefonnummern sein können, und hat eine eindeutige ID in der Form einer UUID (Universal Unique Identifier). Sie enthält aber auch eine Liste mit ihren angebotenen Web Services (**<businessService>**-Elemente), die im folgenden Subelement beschrieben werden. Jeder **<businessService>** stellt einen Web Service dar, der eine eigene ID und die des Superelements besitzt. Dieses Element kann als Kinder mehrere Elemente von **<bindingTemplate>** und **<tModel>** haben. Das Element **<bindingTemplate>** beschreibt wie ein Web Service aufgerufen werden kann. Hier wird die Zugriffadresse meist als URI genannt. **<tModel>** enthält die technische Beschreibung der Schnittstellen des Dienstes und besitzt eine eindeutige Unique User Identification (UUID). Jedes **<tModel>** stellt ein eigenes Konzept dar, das keiner anderen Datenstruktur untergeordnet ist und deshalb nicht als Kindelement in anderen Datenstrukturen vorhanden ist, sondern referenziert wird. [3] Dieses Element ermöglicht die Implementierung dynamischer Anwendungen, die sich zur Laufzeit die zu einem vorgegebenen Modell passende Service-Instanz suchen. Über das Element **<publisherAssertion>** lassen sich Beziehungen zwischen **<businessEntity>**-Elementen ausdrücken. Das Element **<subscription>** stellt eine Hilfe für den Klienten dar, indem es über Änderungen von UDDI-Strukturen informiert. [1]
Die API der UDDI beinhaltet Funktionen für das Suchen und Registrieren von Web Services. Die Suche enthält zwei Arten von Funktionen, um die Einträge des Dienstes aus einem Verzeichnis (Registry) zu holen. Wenn noch nicht viel über den Dienst bekannt ist und zunächst ein Suchraum eingeschränkt werden muss, werden **find**-Funktionen eingesetzt, die eine Liste von Einträgen zurückliefern. Diese Suche wird mit „Browse-Muster“ bezeichnet. In der durch **find**–Methoden erzeugten Liste können **get**-Funktionen entsprechende Details zu den Diensten herausfinden. Dieser Vorgang erhält den Namen „Drill-Down-Muster“. Die **find**- und **get**-Methoden sind auf die folgenden vier wichtigsten UDDI-Datenstrukturen bezogen.

*„**find_business** und **get_businessDetail** werden verwendet, um Informationen über ein oder mehrere **<businessEntity>**-Elemente zu bekommen.*
***find_service** und **get_serviceDetail** werden verwendet, um Informationen über ein oder mehrere **<businessService>**-Elemente zu bekommen*
***find_binding** und **get_bindingDetail** werden verwendet, um Informationen über ein oder mehrere **<bindingTemplates>**-Elemente zu bekommen*
***find_tModel** und **get_tModelDetail** werden verwendet, um Informationen über ein oder mehrere **<tModel>**-Elemente zu bekommen.“* [1]

Zusätzlich können bei der Suche noch **<findQualifier>**-Elemente benutzt werden, die dem UDDI-Knoten Zusatzinformationen über die Suche liefern können.
Ein Beispiel [1] für eine Anfrage ist diese Frage, die nach einem oder mehreren **tModels** von Web Air sucht:

```
<uddi:find_tModel generic="3.0">
    <uddi:name>Web Air</uddi:name>
</uddi:find_tModel>
```

Folgende Antwort könnte der Klient von UDDI erhalten

```
<uddi:tModelList generic="3.0" operator="IBM"
        xmlns:uddi="urn.uddi-org:api_v3">
    <uddi:tModelInfos>
        <uddi:tModelInfo
            tModelKey="uddi:4CD7E4BC-648B-426D-9936-443EAAC8AE23">
            <uddi:name>Web_Air_Flugbuchung</uddi:name>
        </uddi:tModelInfo>
    </uddi:tModelInfos>
</uddi:tModelList>
```

Der zweite Teil der API der UDDI beschäftigt sich mit der Registrierung. Um einen Web Service im UDDI-Verzeichnis anzumelden, muss der Benutzer dort registriert sein. Deshalb muss sich der Dienstanbieter bei jedem Aufruf der Registrierungs-API ausweisen können.
Ist er ein neuer Benutzer, muss er sich interaktiv bei einem UDDI-Knoten mit Benutzername und Passwort anmelden. Mit der Nachricht **get_authToken** erhält er ein Authentifizierungstoken, das ihn jedes Mal, wenn er das Verzeichnis benutzt, erkennt und ihn berechtigt, seine Dienste zu verwalten. Mit den Methoden **save_business**, **save_service**, **save_binding** und **save_tModel** kann er Datenstrukturen anlegen oder verändern. Mit **delete_business**, **delete_service**, **delete_binding** und **delete_tModel** kann er sie wieder entfernen. Um eine Sitzung zu beenden, muss sich der Benutzer mit der Funktion **discard_authToken** abmelden.
UDDI muss wie SOAP und WSDL nicht per Hand programmiert werden, da es genügend Werkzeuge gibt, die diese Aufgabe übernehmen.

2.1.3 Structured Query Language (SQL)

SQL ist eine deklarative Datenbanksprache, basiert auf dem relationalen Datenmodell und bedient sich der relationalen Algebra. Sie wurde von IBM entwickelt und dient dem Erzeugen, Editieren und Anzeigen von Datenbanken und Tabellen. Ob in Großbuchstaben oder in Kleinbuchstaben geschrieben wird, ist bei SQL irrelevant. Wichtig ist, dass nach jedem SQL-Ausdruck (Statement) ein Semikolon steht. [7]

Um eine Datenbank zu erzeugen wird folgende Abfrage verwendet

```
CREATE DATABASE Datenbank;
```

Alle Datenbanken können durch den Befehl

```
SHOW DATABASES;
```

angezeigt werden. Die oben erstellte Datenbank kann durch den Befehl

```
USE Datenbank;
```

benutzt werden. Eine weitere Möglichkeit wäre, vor jede Spalte den Namen der zugehörigen Datenbank zu setzen (**`Datenbank.Spalte`**).

Eine Tabelle wird durch

```
CREATE TABLE Tabelle(Spalte Spaltentyp, Spalte2 Spaltentyp);
```

erstellt. Nach der Angabe des Namens der neuen Tabelle folgt eine Klammer, in der die Spalten durch Kommas getrennt werden. Notwendig bei einer Spalte sind die Nennung der Bezeichnung der Spalte und ihr Spaltentyp. In SQL gibt es numerische Typen wie z.B. **`smallint`**, **`int`**, **`bigint`**, **`float`**, **`double`**, Datums- und Zeittypen wie **`date`**, **`time`**, **`year`**, **`timestamp`** und Zeichenkettentypen wie **`char(m)`**, **`varchar(m)`**, **`text`**, **`longtext`**, **`enum('wert1', 'wert2', ...)`**, **`set('wert1', 'wert2', ...)`**. Nach der Angabe des Typs können noch weitere Angaben stehen. Beispielsweise könnte hier **`auto_increment`** stehen, was den Wert automatisch um eins erhöht, was bei Spalten, die eine ID repräsentieren, sinnvoll ist. Um zu verhindern, dass eine Spalte leer ist, kann **`not null`** eingesetzt werden.
Um den Primärschlüssel der Tabelle festzulegen, wird nach der letzten Spalte **`primary key(Spalte)`** eingefügt. Möchte man sich nun diese Tabelle mit all ihren Spalten anzeigen lassen, kann der folgende Befehl verwendet werden:

```
SHOW COLUMNS FROM Tabelle;
```

Hier werden die Spalten mit ihren Typen und Zusatzwerten angezeigt.
Wenn der Name einer Tabelle geändert werden soll, wird folgender Befehl verwendet:

```
ALTER TABLE alterNamederTabelle RENAME neuerNamederTabelle;
```

Soll nun die vorhin erzeugte Tabelle mit Daten gefüllt werden, wird das SQL-Statement

```
INSERT INTO Tabelle(Spalte, Spalte2) VALUES ("Wert1", "Wert2");
```

ausgeführt. Die Namen der zu füllenden Spalten werden in der ersten Klammer und ihre Werte in der zweiten angegeben.

Um den Inhalt aus einer Tabelle darzustellen, wird die **`SELECT-FROM-WHERE`**-Klausel eingesetzt. Mit **`SELECT`** werden die anzuzeigenden Spalten und mit **`FROM`** die Tabelle, in der sich die Spalten befinden, ausgewählt. Nach dem optionalen **`WHERE`** können Bedingungen gestellt werden. Bedingungen können mit Hilfe von Standardoperatoren **<**, **=**

und **>**, logischen Operatoren **AND** , **NOT** oder **OR**, Zeichenkettenvergleichen wie **LIKE** oder **NOT LIKE**, arithmetischen Operationen wie Addition (**+**), Subtraktion (**-**), Multiplikation (*****) und Division (**/**) und anderen mathematischen Funktionen, wie beispielsweise Modulo (**Mod(N, M)** oder **%**), **PI()**, Sinus (**SIN()**), dargestellt werden, die natürlich auch in anderen SQL-Ausdrücken stehen können.
Hier sind einige Beispiele zum Anzeigen des Tabelleninhalts:

```
SELECT * FROM Tabelle;
```

Alle in der Tabelle **Tabelle** enthaltenen Informationen werden angezeigt.

```
SELECT Spalte1, Spalte2 FROM Tabelle2
            WHERE Spalte3=12 AND Spalte2=Spalte3+3;
```

Aus der Tabelle **Tabelle2** werden die Stellen der Spalten **Spalte1** und **Spalte2** angezeigt, an denen die Spalte **Spalte3** den Wert **12** und die Spalte **Spalte2** den Wert **15** hat.

Um den Inhalt einer Spalte zu ändern, kann

```
UPDATE Tabelle SET Spalte=Wert;
```

eingesetzt werden. Natürlich können hier auch Bedingungen durch eine WHERE-Klausel angegeben werden.

Möchte man den Inhalt einer Tabelle löschen, erfolgt dies über

```
DELETE Spalte FROM Tabelle;
```

Dies kann auch wieder mit einer WHERE-Klausel verknüpft werden, um nur bestimmte Zeilen zu löschen.

Wenn nun die Tabelle mit ihrem gesamten Inhalt entfernt werden soll, wird folgender Befehl ausgeführt:

```
DROP TABLE Tabelle;
```

2.2 Praxis

Hier werden die Werkzeuge, mit denen das Evaluationssystem entwickelt wird, kurz vorgestellt und erklärt, wie sie installiert werden. Als Java Plattform wird J2SDK 1.4.2_05 [11] verwendet. Auf dieser setzt der Web-Server Apache Tomcat 5.0.25 [12] auf, die das SOAP-Framework Apache Axis 1.1 [8] einbindet. Der Datenbank-Server des Evaluationssystems ist MySQL 4.0.20a [4].

2.2.1 Werkzeuge

In diesem Kapitel werden die Werkzeuge, mit denen Web Services erstellt werden können, erläutert und deren Installation beschrieben. Da die in dieser Bachelorarbeit verwendeten Web Services Java-Klassen sind, wird eine Java Virtual Machine benötigt. Als Web-Server dient der Apache Tomcat, welcher durch Axis erweitert wird. Axis besitzt Bibliotheken, die den Einsatz von Web Services ermöglichen.

2.2.1.1 Java 2SDK

Eine Java Virtual Machine (JVM) ermöglicht die Entwicklung und Ausführung von Java-Programmen. JVMs werden von SUN [11] erstellt und zum kostenlosen Download bereitgestellt. Da die Web Services des Evaluationssystems mit Java entwickelt werden sollen, wird natürlich eine JVM benötigt.
Diese kann von der Homepage von SUN [11] heruntergeladen werden. Für die Arbeit mit Web Services sollte es sich um ein Java SDK ab der Version 1.3 handeln. In dieser Bachelorarbeit wird j2sdk1.4.2_05 verwendet. Falls die Systemvariable **`PATH`** nicht schon automatisch um den Pfad des Verzeichnisses **`C:\..\j2sdk1.4.2_05\bin`** ergänzt worden ist, sollte dies nachgeholt werden. Unter Windows XP findet man unter Start → Systemsteuerung → System → Erweitert → Umgebungsvariablen die Systemvariablen. Die Variable **`PATH`** wird zum Bearbeiten aufgerufen und an die dort vorhandenen Pfade wird der Pfad des Verzeichnisses angehängt.

2.2.1.2 Apache Tomcat

Der Web-Server Apache Tomcat [12] wurde vom Apache Jakarta Projekt [13], das zur Apache Foundation gehört, entwickelt und ist ein Open-Source Projekt.
Dieser auf einer JVM laufender Web-Server ist ein Java-Servlet-Container und zuständig für die Referenzimplementierung der von Sun entwickelten Standard Servlets und Java Server Pages (JSP).
Um mit auf Java basierenden Web Services zu arbeiten, muss der Apache Tomcat installiert sein.
Notwendig ist eine JVM auf die der Apache Tomcat aufsetzt. In dieser Arbeit wird die

Version Apache Tomcat 5.0.25 verwendet. Der Tomcat kann einfach installiert werden und danach in Windows durch einen Aufruf von Start → Programme → Apache Tomcat 5.0 → Start Tomcat gestartet werden. Alternativ kann man ihn auch im Verzeichnis **`\..\Apache Software Foundation\Tomcat 5.0\bin`** mit dem Befehl **`startup`** zum Start veranlassen. Wenn man nun im Browser den lokalen Computer über den Port 8080 mit **`http://localhost:8080`** anspricht und dort angezeigt wird, dass der Server erfolgreich aufgesetzt worden ist, kann mit dem Web-Server gearbeitet werden.

2.2.1.3Apache Extensible Interaction System (Axis)

„Axis ist eine Open-Source-Implementierung des Web-Service-Standards SOAP unter der Lizenz der Apache Software Foundation.“ [3] Axis ermöglicht die Entwicklung, den Betrieb und das Anbieten von Web Services.

Der Vorgänger von Axis ist die Apache-SOAP-Implementierung, die auf DOM basiert und relativ langsam ist. Dagegen verwendet Axis SAX, das im Gegensatz zu Apache SOAP kaum Ressourcen verschwendet. Eine weitere Neuerung ist die Verwendung von der JAX-RPC 1.0 Spezifikation, was *„eine Standardspezifikation, die API und Programmiermodell für die Web-Service-Entwicklung in Java definiert“* [3] ist. Weitere Eigenschaften dieses Produkts sind natürlich die Unterstützung von SOAP 1.1 (teilweise SOAP 1.2), WSDL 1.1 und SAAJ 1.1 (teilweise SAAJ 1.2). SAAJ bedeutet SOAP with Attachments API for Java und ist eine *„Standard-API, die im Wesentlichen dem Aufbau von SOAP-Nachrichten dient und die einzelnen Nachrichtenbestandteile in Form von Java-Klassen bzw. Interfaces repräsentiert.“* [3]

Axis bietet eine hohe Performanz, ist erweiterbar, lässt sich sehr flexibel konfigurieren und ist unabhängig vom Transport, d.h. es können sowohl HTTP, FTP, JMS als auch SMTP eingesetzt werden. Außerdem kann Axis durch Sicherheitspakete erweitert werden und es können Enterprise JavaBeans als Web Service verwendet werden. Weitere Vorteile sind, dass Axis den Aufbau einer Web-Applikation hat und deshalb in jedem Java-fähigen Web-Server eingesetzt werden kann sowie dass es mit vielen anderen Open-Source-Produkten wie beispielsweise WebObjects 5.2 von Apple, JBoss, Macromedia JRun, IBM Emerging Technologies Toolkit und Coldfusion MX und anderen kommerziellen Projekten verwendbar ist.

Die hier verwendete Axis-Distribution [8] beinhaltet einen Server in Form einer Web-Anwendung, einen Standalone-Server, der für PDAs sinnvoll ist, Bibliotheken zum Erstellen von Klienten, Kode-Beispiele und Dokumentationen. Sie besitzt auch noch ein SOAP Monitor Applet und eine Transmission Control Protocol (TCP) Monitor-Anwendung, mit denen SOAP-Nachrichten mitverfolgt werden können. Auch WSDL-Werkzeuge zur Code- und WSDL-Generierung und eine Testsuite auf der Basis des Testframeworks JUnit gehören zu Axis.

Wird Axis im Tomcat verwendet, wird es von diesem Web-Server als Servlet behandelt. Die Datei **web.xml** von Axis, die Konfigurationsinformationen für den Tomcat-Server enthält, hat unter anderen folgende Einträge [1]:

```
<servlet-mapping>
    <servlet-name>AxisServlet</servlet-name>
    <url-pattern>*.jws</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>AxisServlet</servlet-name>
    <url-pattern>/services/*</url-pattern>
</servlet-mapping>
```

Alle mit **.jws** endenden und alle mit **/services/** beginnenden HTTP-Anfragen werden hierdurch an das Axis-Servlet weitergereicht. Dieses parst die SOAP-Anfrage und lokalisiert den Service der zu implementierenden Klasse. Handelt es sich um eine JWS-Datei, muss diese erst kompiliert werden, bevor das Servlet die so genannten Handler zur Vorbearbeitung der Anfrage aufruft. Danach wird erst die eigentliche Implementation aufgerufen, wonach der Handler erneut zur Nachbearbeitung verwendet wird. Anschließend werden die Ergebnisse kodiert. Im Beispiel unten sind die Architektur und das Zusammenspiel von Tomcat als Web-Server und Axis als Web-Service-Engine dargestellt.
In Abbildung 2-5 wird eine JWS-Datei verwendet, die einen simplen Web Service darstellt.

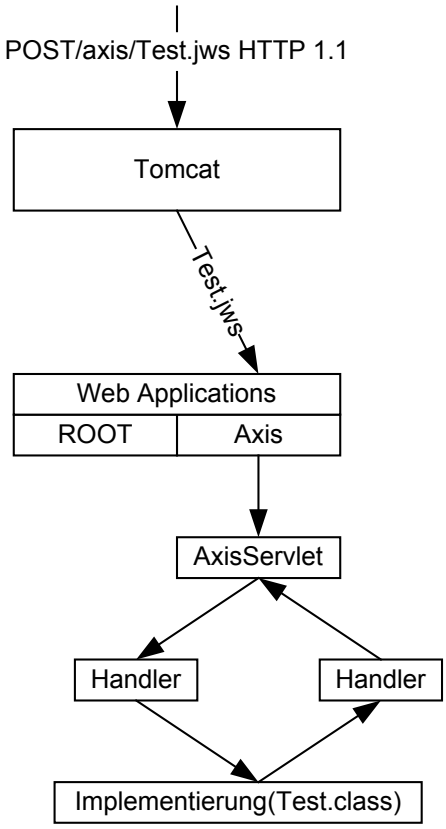

Abbildung 2-5 Architektur und Zusammenspiel von Tomcat als Java-Web-Server und Axis als Web-Service-Engine [1]

2.2.1.3.1 Installation von Axis

Voraussetzung für die Lauffähigkeit von Axis ist ein Java SDK ab der Version 1.3 und ein Web-Server. Um dieses Programm zu verwenden, muss als erstes von der Homepage von Axis [8] die neueste Release-Version heruntergeladen und in einem beliebigen Verzeichnis entpackt werden. Die entpackte Version enthält einen Ordner **`axis`**, dessen Unterverzeichnis **`webapps`** in das Verzeichnis **`\..\Tomcat 5.0\webapps`** des Tomcats kopiert werden muss. Dies ist das zukünftige Arbeitsverzeichnis, in denen die Web Services angelegt werden. Nun kann der Tomcat neu gestartet und die Seite **`http://localhost:8080/axis/happyaxis.jsp`** aufgerufen werden.

Falls eine Fehlermeldung erscheint, sollte eine Kopie des Java-Pakets **`tools.jar`** aus der JVM in dem Verzeichnis des Tomcats **`\..\Tomcat 5.0\common\lib`** abgelegt werden. Nach einem weiteren Neustart des Web-Servers kann die oben angegebene Seite erneut aufgerufen werden.

Wenn dieser Schritt erfolgreich war, erscheint eine Seite, die Informationen über die Konfiguration mit noch fehlenden Paketen angibt. Von der Homepage von Sun müssen nun die Java-Bibliotheken Java Beans Activation Framework (**`actionvation.jar`**) ab Version 1.0.2, Java Mail API (**`mail.jar`**) ab Version 1.3 und Java Servlet API (**`xmlsec.jar`**) ab Version 2.3 heruntergeladen und in das Tomcat-Verzeichnis

\..\Tomcat 5.0\common\lib gesetzt werden. Zusätzlich sollte noch die Bibliothek des XML-Parsers (**xerces.jar**) in diesem Verzeichnis abgelegt werden. Dann wird wiederum der Web-Server gestartet und die HappyAxis-Seite angezeigt.
Zum Schluss müssen noch Pfade gesetzt werden. Im Verzeichnis
\..\Tomcat 5.0\webapps\axis\WEB-INF\lib
müssen sich neben den Bibliotheken **axis.jar** und **axis-ant.jar** folgende Bibliotheken befinden: die Schnittstellen der JAX-RPC-Spezifikation (**jaxrpc.jar** und **saaj.jar**), die Unterstützung für Logging (**log4j-1.2.8.jar**) und WSDL **(wsdl4.jar**) und die Bibliotheken aus dem Jakarta-Commons-Projekt (**commons-discovery.jar** und **commons-logging.jar**). Als erstes ist eine Systemvariable (beispielsweise **AXIS_HOME**) mit dem Pfad, in dem die Bibliotheken sind, zu setzen. Danach können die Bibliotheken in den Klassenpfad (**CLASSPATH**) aufgenommen werden. Dort muss dann zusätzlich zu den anderen Bibliotheken Folgendes stehen:

```
.;
%AXIS_HOME%\axis.jar;
%AXIS_HOME%\axis-ant.jar;
%AXIS_HOME%\commons-discovery.jar;
%AXIS_HOME%\commons-logging.jar;
%AXIS_HOME%\jaxrpc.jar;
%AXIS_HOME%\log4j-1.2.4.jar;
%AXIS_HOME%\saaj.jar;
%AXIS_HOME%\wsdl4j.jar
```

Nun ist die Axis-Installation komplett.

2.2.1.4MySQL

MySQL [4] ist ein sehr weit verbreiteter Open Source Multi-Thread und Multi-User SQL-Datenbank-Server des Unternehmens MySQL AB. Er läuft auf sehr vielen Betriebssystemen und hat eine umfangreiche Bibliothek. In dieser Bachelorarbeit wird die Version MySQL 4.0.20a verwendet.
Um diese zu verwenden, wird die Binärdistribution einer stabilen Version heruntergeladen, da diese einfacher als eine Quelldistribution zu installieren ist. Nach der Installation kann mit MySQL gearbeitet werden. Um diesen Datenbank-Server aus Java-Programmen heraus aufrufen zu können, müssen die Bibliotheken im Java-Verzeichnis
\..\j2sdk1.4.2_05\jre\lib\ext
um **mysql-connector-java-3.0.14-production-bin.jar** ergänzt werden. Außerdem muss eine Systemvariable (beispielsweise **MYSQLPATH**) mit dem Pfad
C:\Programme\Java\j2sdk1.4.2_05\jre\lib\ext
gesetzt und dem Klassenpfad (**CLASSPATH**) die Zeile
%MYSQLPATH%\mysql-connector-java-3.0.14-production-bin.jar

hinzugefügt werden.
Damit auf eine Datenbank überhaupt zugegriffen werden kann, muss der Datenbank-Server erst gestartet werden. Dies erfolgt in der Kommandozeile im Verzeichnis **`\..\mysql\bin`** über den Befehl **`mysqld`**. Nun kann in einem weiteren Kommandofenster im selben Verzeichnis mit **`mysql`** der MySQL Monitor geöffnet werden. Es erscheint die Eingabeaufforderung in Form von **`mysql>`** . Bei einigen MySQL-Versionen reicht das Kommando **`mysql`** nicht aus, um sie vollständig zu nutzen. Es kann beispielsweise vorkommen, dass es nicht möglich ist, eine Datenbank zu erstellen, da einem dazu die Rechte fehlen. Deshalb muss man sich mit **`mysql -u root –p`** als Administrator einloggen, der über alle Rechte verfügt. **`u`** steht hier für Benutzer (User) und **`p`** für Passwort. Nun können Datenbanken und Tabellen mit SQL-Ausdrücken verwaltet werden. Um MySQL wieder zu verlassen, kann einfach **`exit;`** oder **`quit;`** eingegeben werden.

2.2.2 Beispiele für Web Services

Es gibt in Java zwei Möglichkeiten einen Web Service in einer Axis-Umgebung zu erstellen. Diese verschiedenen Web Services werden anhand ihres Deployments unterschieden. Das Deployment ist die Installation und Inbetriebnahme von Softwarekomponenten und bedeutet bei Axis, dass die Implementierung eines Web Services auf den Seiten eines Servers abgelegt und dieser konfiguriert wird. Dieses Deployment kann sowohl automatisch als auch benutzerdefiniert stattfinden. In den folgenden zwei Kapiteln werden diese Mechanismen an einem Einführungsbeispiel (HelloWorld) erklärt.

2.2.2.1 Automatisches Deployment mit Java Web Service (JWS)

Das automatische Deployment ist die einfachste Form einen Web Service einzusetzen. Dieses geschieht mit JWS-Dateien, die den Quellkode von Java-Web-Service-Klassen besitzen, aber als Endung anstatt **`.java .jws`** haben. Bei einem solchen Web Service übernimmt Axis die Serialisierung automatisch. Hier können aber nur Standard-Datentypen wie **`String`** und **`int`** und keine benutzerdefinierten Datentypen verarbeitet werden. Deshalb ist diese Methode ungeeignet für größere Anwendungen. Trotzdem soll zur Vollständigkeit gezeigt werden, wie dieser Web Service erstellt wird.
Eine JWS-Datei wird in dem Wurzelverzeichnis der Axis-Anwendung abgelegt.

Hier ist ein Beispiel für einen Web Service namens **HelloWorld.jws**:

```
public class HelloWorld {
    public String hello() {
        String s = "Hello World!";
        System.out.println(s);
        return s;
    }
}
```

Im Browser wird die Seite **http://localhost:8080/axis/HelloWorld.jws** aufgerufen und es erscheint eine Meldung mit einem Link, dass ein Web Service gefunden worden ist. Folgt man diesem Link, erscheint im Browser die zugehörige WSDL-Datei und in der Adressleiste die URL
http://localhost:8080/axis/HelloWorld.jws/WSDL.

Aber nicht nur auf dem Bildschirm hat sich einiges geändert, sondern auch im Ordner **\..\axis\Web-Inf**. Hier ist ein neuer Ordner namens **jwsClasses** entstanden, in dem die kompilierte Datei **HelloWorld.class** zu finden ist.

Nun können die Java-Stubs mit dem wsdl2java-Werkzeug für den Web Service über die Kommandozeile erzeugt werden. Im Verzeichnis **\..\axis** muss der Befehl

java org.apache.axis.wsdl.WSDL2Java
http://localhost:8080/axis/HelloWorld.jws?WSDL

ausgeführt werden. Lässt man sich den Inhalt des Verzeichnisses anzeigen, stößt man auf den Ordner **localhost**, der einen Ordner **axis** enthält, welcher wiederum einen Ordner namens **HelloWorld_jws** besitzt. In diesem befinden sich die Java-Klassen **HelloWorld**, **HelloWorldService**, **HelloWorldServiceLocator** und **HelloWorldSoapBindingStub**.

Nun ist der Service aufrufbar. Um ihn zu nutzen kann ein Klient im Axis-Verzeichnis erstellt werden.

```
import java.net.*;
public class HelloWorldClient {
    public static void main(String [] args) {
        HelloWorldServiceLocator loc =new HelloWorldServiceLocator();
        HelloWorld ser = loc.getHelloWorld(
            new URL("http://localhost:8080/axis/HelloWorld.jws"));
        System.out.println(ser.hello());
    }
}
```

Wird die Datei **HelloWorldClient.java** kompiliert und ausgeführt, wird **Hello World!** ausgegeben, was bedeutet, dass der Klient soeben den Dienst der Klasse HelloWorld erfolgreich in Anspruch genommen hat.

2.2.2.2Benutzerdefiniertes Deployment mit WSDD

Das benutzerdefinierte Deployment ist wesentlich flexibler und einsatzfähiger als das automatische Deployment. Axis stellt hierfür das Tool AdminClient und einen Web Service Deployment-Deskriptor (WSDD) bereit, mit denen die manuelle Konfiguration und Integration von Web Services möglich ist.

„Im Gegensatz zum automatischen Deployment über JWS-Dateien bilden hier bereits vorkompilierte Klassen die Grundlage. Die zu installierende Klasse und weitere Installations- und Konfigurationsparameter werden darüber hinaus in einer auf XML basierenden Konfigurationsdatei, dem ... Deployment-Deskriptor, zusammengefasst. Diese Deskriptor-Datei, die im Normalfall auf **`.wsdd`** *endet, wird dem von Apache Axis mitgelieferten AdminClient zugeführt und der Web Service damit in die Laufzeitumgebung integriert.*

Umfangreiche Steuerparameter und die damit verbundene Flexibilität machen den AdminClient zu einem fast schon unverzichtbaren Helfer, der neben dem klassischen Deployment auf dem lokalen Server auch für das sog. Remote Deployment, d.h. der Installation von Web Services auf entfernten Servern, und administrative Zwecke eingesetzt werden kann.“ [3]

Um einen Web Service mit dieser Methode zu erstellen, wird die oben gezeigte HelloWorld-Datei in **`HelloWorld.java`** umbenannt und kompiliert. Diese kompilierte Klasse wird in dem Verzeichnis **`\..\Tomcat 5.0\webapps\axis\WEB-INF\classes`** abgelegt. Natürlich kann dorthin auch die Java-Datei verlegt werden. Wenn nötig, muss die Verzeichnisstruktur eines Paketes (**`package service.hello`**) nachgebildet werden. Nun muss diese Klasse mit WSDD und AdminClient zur Nutzung angeboten werden.

Die WSDD-Datei wird im **`\..\classes`**-Verzeichnis abgelegt und kann beispielsweise mit **`HelloWorld_Deploy.wsdd`** bezeichnet werden und folgenden Inhalt haben:

```
<deployment
    xmlns=http://xml.apache.org/axis/wsdd/
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
    <service name="HelloWorld" provider="java:RPC">
        <parameter name="className" value"HelloWorld" />
        <parameter name="allowedMethods" value="hello" />
    </service>
</deployment>
```

Das Element **`<deployment>`** teilt dem Axis-Server mit, dass ein neuer Service namens **`HelloWorld`** eingebunden werden soll. Der **`provider`** ist an dieser Stelle **`java:RPC`**, was bedeutet, dass eintreffende SOAP-Anfragen die Java-Klasse instanziert und ihre Methode aufgerufen wird. Das Element **`<parameter>`** übermittelt die Daten zur

Konfiguration des Dienstes. Die Klasse **HelloWorld** und seine Methode **hello** werden angegeben. Hat eine Web Service mehrere Methoden, die mit **public** bezeichnet sind, können diese durch Kommas getrennt aufgeführt werden. Wird ein ***** verwendet, werden alle Methoden freigegeben.
Nun kann mit dem AdminClient gearbeitet werden. Er ruft den in der Axis-Engine vorhanden Web Service AdminService, der nicht entfernt werden sollte, auf. Der AdminClient wird nun mit dem Befehl

```
java org.apache.axis.client.AdminClient -l
    http://localhost:8080/axis/servlet/AxisServlet
```

in der Kommandozeile im Verzeichnis des Deployment-Deskriptors aufgerufen. Das AxisServlet ist eine Art Wrapper, der unter anderem den AdminService aufruft und eine Fernwartung von Axis ermöglicht. War das Deployment erfolgreich, erscheint in der Kommandozeile

<Admin>Done processing</Admin>

und man kann den AdminClient im Browser über

http://localhost:8080/axis/servlet/AxisServlet

aufrufen. Dort erscheint der neue Web Service HelloWorld mit seiner WSDL-Datei.
Ein solcher Web Service kann durch einen Deployment-Deskriptor wieder entfernt werden, welcher folgende Form hat:

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

Um den Web Service gegebenenfalls zu deinstallieren muss jetzt nur noch der Deployment-Deskriptor wie im obigen Bespiel aufgerufen werden.

Möchte man diesen Service nutzen, kann ein Klient namens **HelloWorldClient.java** geschrieben werden:

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.Qname;

public class HelloWorldClient {
    public static void main(String[] args) {
        try{
            String endpoint=
                "http.//localhost:8080/axis/services/HelloWorld";
            Service service = new Service();
            Call call = (Call) sevice.createCall();

            call.setTargetEndpointAddress( new java.net.URL(endpoint) );
            call.setOperationName("hello");

            String ret= (String) call.invoke(new Object[] {} );

            System.out.println("Der Web Service sagt: " + ret);
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}
```

Die importierten Klassen **Call** und **Service** implementieren die gleichnamigen JAX-RPC-Schnittstellen. Ihre Instanzen speichern Metadaten zum Aufruf des Dienstes und stellen Methoden zur Verfügung, mit denen der Dienst aufrufbar ist. Die Variable **endpoint** enthält die Adresse des Dienstes und der Instanz von **Call** wird seine aufzurufende Methode mitgeliefert. Die Adresse wird auch als Endpunkt (Endpoint) bezeichnet. Falls dem Web Service Parameter übergeben werden sollen, können Variablen mitgegeben werden. Zum Beispiel könnte die Variable

Integer zahl = new Integer(3);

beim Aufruf des Services

String ret = (String) call.invoke (new Object[] { zahl });

weitergegeben werden.

„Nach dem Starten der Klient-Anwendung wird zunächst der Service Endpoint definiert und in einer Variablen gespeichert. Danach folgt die Instanzierung des erforderlichen Service-Objekts, um daraus das Call-Objekt zu erzeugen. Dem Call-Objekt wird der Endpoint übergeben und über die Methode ***setOperationName*** *mitgeteilt, welche Web-Service-Methode aufzurufen ist. Der Aufruf des Web Services erfolgt schließlich über die Methode* ***invoke****, wobei hier ein Array aus Parametern übergeben werden kann.“* [3] Im oben dargestellten Beispiel wird allerdings kein Parameter übergeben.

3 EVALUATION

3.1 Definition der Evaluation

Laut Duden ist eine Evaluation eine *„Berechnung, Bewertung a) Bewertung, Bestimmung des Wertes, b) sach- und fachgerechte Einschätzung und Beurteilung [von Lehrplänen und Unterrichtsprogrammen bzw. von Forschungsvorhaben.“* [5]

Es gibt verschiedene Modelltypen der Evaluation. Die zwei wichtigsten sind die Qualitätsentwickelnde und die Entscheidungsgesteuerte Evaluation. Erstere beschäftigt sich mit der Qualitätsentwicklung dem Qualitätsbewusstsein in Organisationen. *„Der Evaluator versucht vor allem über kommunikative Methoden ein System der Selbstevaluation und Weiterentwicklung zu unterstützen. „Qualitätsmanagement“ geht vor Qualitätsbestimmung. Ziel der entscheidungsgesteuerten Evaluation ist, in einem rationalen und wertorientierten Verfahren systematisch aufbereitete und möglichst entscheidungsrelevante Informationen über die Qualität eines Evaluationsprojekts für den Auftraggeber zusammenzustellen.*

Evaluation sollte als ein kontinuierlicher Prozess der Reflexion und Verbesserung implementiert werden, bei dem DozentInnen und Studierende gemeinsam auf sehr spezifische Weise die positiven und die kritischen Aspekte einer Lehrveranstaltung beleuchten und bei Bedarf konkrete Verbesserungsvorschläge erarbeiten.“ [5]

Eine Evaluation an einer Universität hat das Ziel die Qualität der Lehre zu erhalten und zu verbessern. Für jemanden, der lehrt und nicht erst lernen muss, ist es sicherlich schwierig, seine Lehrqualitäten einzuschätzen, wenn er keine Rückmeldung von seinen Studenten bekommt. An Klausuren lässt sich noch erkennen, ob die Studenten den vermittelten Lernstoff verstanden haben und ob gut gelehrt worden ist. Was ist aber, wenn die Klausur schlecht ausfällt? War die Vorlesung gut, aber die gestellten Aufgaben einfach zu anspruchsvoll, wurde der Lernstoff schlecht vermittelt oder waren die Studenten nicht fleißig genug? Ein Dozent kann sich zur Verbesserung seiner Lehre auch an seine eigene Studentenzeit erinnern und die Lehrmethoden guter Dozenten annehmen und die Fehler der schlechteren durch eigene Ideen verbessern. Allerdings sind dies nur geringe Hilfen sich selbst einzuschätzen und gegebenenfalls zu verbessern. Die Lehrevaluation kann ihn dabei unterstützen gezielt an seiner Methodik zu feilen.

Eine Lehrevaluation kann sowohl mit Fragebögen, die den Studenten am Ende des Semesters zum Ausfüllen gereicht werden, oder über ein Online-Formular im Internet stattfinden.

3.2 Anforderungen

Um möglichst genaue Ergebnisse zu erzielen und datenschutzrechtliche Aspekte zu erfüllen, muss eine Evaluation folgenden Anforderungen entsprechen.

- Jede Vorlesung soll evaluierbar sein, um einen besseren Vergleich zu haben und das Lehrangebot des Instituts einschätzen zu können.
- Eine Vorlesung soll jederzeit evaluierbar sein, damit jeder Student die Möglichkeit hat seine Meinung zu äußern.
- Ein Student soll eine Vorlesung nur einmal evaluieren, damit das Ergebnis nicht verfälscht wird.
- Ein Student soll nur die Vorlesungen bewerten können, die er auch besucht hat, damit kein falsches Ergebnis entsteht.
- Studenten und ihre Daten sollen für andere anonym bleiben, da sonst der Datenschutz verletzt wird. Wenn außerdem bekannt wäre, wer welche Antworten gegeben hat, würde nicht jeder Student seine ehrliche Meinung angeben.
- Es soll nicht erkennbar sein, welche Vorlesungen ein Student noch belegt hat, damit man nicht anhand der evaluierten Vorlesungen auf ihn schließen kann.Es sollen unterschiedliche, den Lehrveranstaltungen angepasste Fragebogentypen existieren, da sich eine Vorlesung von einem Praktikum unterscheidet und unterschiedliche Bereiche evaluiert werden müssen.

4 ONLINE-EVALUATIONSSYSTEM

In diesem Abschnitt wird das Online-Evaluationssystem vom Konzept bis zur Implementierung beschrieben.

4.1 Idee

Die im vorherigen Kapitel gestellten Anforderungen an ein Evaluationssystem müssen beim Online-Evaluationssystem natürlich eingehalten werden.
Die Anforderung, dass jede Lehrveranstaltung evaluierbar sein soll, kann dadurch erfüllt werden, dass jede Vorlesung in das Evaluationssystem aufgenommen wird. Es muss also eine Schnittstelle geben, die es ohne direkten Zugriff auf eine Datenbank ermöglicht, auf einfache, für jeden verständliche Weise alle Lehrveranstaltungen in das System einzutragen. Dies kann in einem Administrationsbereich über eine Eingabemaske im Internet geschehen. Zusätzlich sollten hier auch die geforderten unterschiedlichen Fragebogentypen erstellbar und den Vorlesungen zuweisbar sein. Der Administrationsbereich sollte vor Missbrauch durch ein Passwort geschützt sein.
Für einen Studenten sollte es zielgerecht jederzeit möglich sein, seine Vorlesungen zu evaluieren. Dies kann dadurch erreicht werden, dass die Evaluation im Internet über eine Eingabemaske stattfindet. So hat jeder Student die Möglichkeit, wann immer er möchte, seine Vorlesungen zu bewerten.
Allerdings darf es ihm lediglich einmal möglich sein, eine Vorlesung zu evaluieren, damit sich das Ergebnis der Auswertung nicht verfälscht. Dies kann über die Vergabe von PINs erreicht werden, welche der Student zu jedem Fragebogen eingeben muss. Die Vergabe der PINs wirft aber einige Probleme auf.
Zunächst besteht das Problem der Anonymität. Da nur die jeweiligen Teilnehmer einer Vorlesung wirklich in der Lage sind, diese auch einwandfrei zu bewerten, kann einem Studenten lediglich für seine spezifischen Vorlesungen eine PIN gegeben werden. Dadurch könnte jedoch die Anonymität gefährdet werden, da man über die Teilnehmerlisten der Vorlesungen herausfinden könnte, wer bei den jeweiligen Vorlesungen evaluiert hat. Je kleiner der Teilnehmerkreis einer Vorlesung wäre, desto leichter können Rückschlüsse über die Identität gezogen werden.
Als weiteres Problem zeigt sich, dass es schwierig und aufwändig ist, an die entsprechenden Vorlesungen des jeweiligen Studenten zu gelangen. Wenn Daten aus dem an dem Institut für Informatik verwendeten Online-Prüfungssystem MUNOPAG verwendet werden, wird nicht nur der Datenschutz verletzt, sondern es werden auch nur die Vorlesungen vorzufinden sein, in denen sich der Student im aktuellen Semester prüfen lässt. Wenn er aber eine andere, in seinen Augen schlechte Vorlesung abgebrochen hat, wird man dies nicht herausfinden und den Sinn der Evaluation

zerstören. Eine Idee wäre es, jedem Studenten die Möglichkeit zu geben, jede Vorlesung zu evaluieren. Da ein Fragebogen, der eine gewisse Länge hat, nicht nur mit wenig Arbeit (ein paar Mausklicks) erledigt ist, bedeutet die Evaluation für den Anwender Aufwand und daher ist die Wahrscheinlichkeit für einen Missbrauch gering. Aus diesem Grund sollte jeder Student eine PIN für alle Vorlesungen erhalten, die pro Vorlesung aber nur einmal anwendbar ist. Somit bildet man die Basis für eine repräsentative Abbildung der studentischen Meinungen und erhält gleichzeitig die Anonymität der Benutzer.
Aber es ist nicht nur gefordert, die belegten Vorlesungen sondern auch die persönlichen Daten der Studenten von den Daten der Evaluation anonym zu halten. Dazu geben die Studenten bei der Anforderung einer PIN ihre Matrikelnummer und Emailadresse an. Diese Daten werden zusammen mit der vergebenen PIN gespeichert. Daran ist aber nicht zu erkennen, wer welchen Fragebogen ausgefüllt hat, da die evaluierten Fragebögen so gespeichert werden, dass man keine Rückschlüsse auf die Studentendaten ziehen kann.
Wenn eine PIN einem Studenten verloren gegangen ist, ist sie über die Studentendaten auffindbar. Der Student muss seine Matrikelnummer angeben und erhält dann eine Email mit seiner PIN.

4.2 Fachkonzept

Das Evaluationssystem stellt sich nach außen als ein einheitliches System dar, besteht aber intern aus mehreren Bausteinen, die über voneinander unabhängige Web Services miteinander kommunizieren.
Es gibt zwei eigenständige Datenbanken. Eine ist für die Autorisierung im System zuständig, d.h. sie enthält unter anderen die Zugangsdaten der Benutzer, die Daten der Studenten und Vorlesungen mit PINs und wird als ADB (Autorisierungsdatenbank) bezeichnet. Die andere mit der Bezeichnung FDB (Fragebogendatenbank) dient der Speicherung der Vorlesungen mit Fragebögen und Antworten. Jede dieser Datenbanken bietet als Schnittstelle einen Web Service an. Über diese Web Services können der Administrations-, der Teilnehmer- und der Auswertungsbereich zugreifen. Alle drei Bereiche besitzen eine Maske, über die die Benutzer des Evaluationssystems im Internet mit dem System kommunizieren können.
Die Administratoren des Systems können im Administrationsbereich Vorlesungen verwalten und neue Administratoren und Auswerter erzeugen. Dieser Bereich kommuniziert über die Web Services der ADB und FDB mit diesen Datenbankmodulen. Unbemerkt für den Administrator findet zwischen diesen beiden Datenbankmodulen ein weiterer Datenaustausch über einen Web Service statt. Hierbei wird der Inhalt der Vorlesungstabelle der FDB in die Vorlesungstabelle der ADB kopiert.
Der Teilnehmerbereich zeigt die Vorlesungen mit entsprechenden Fragebögen an. Hier können die Studenten eine PIN anfordern und danach evaluieren. Dieser Bereich

kommuniziert auch über Web Services, sowohl mit der ADB um PINs abzugleichen als auch der FDB um die Fragebögen anzuzeigen und die Antworten zu speichern. In dem Auswertungsbereich können berechtigte Personen die Ergebnisse der Umfragen anschauen. Hierzu wird zum Login eine Verbindung mit der ADB und zur Darstellung der Ergebnisse mit der FDB auch über Web Services hergestellt. Diese drei Webschnittstellen und zwei Datenbanken sind in den folgenden Abschnitten noch genauer beschrieben.

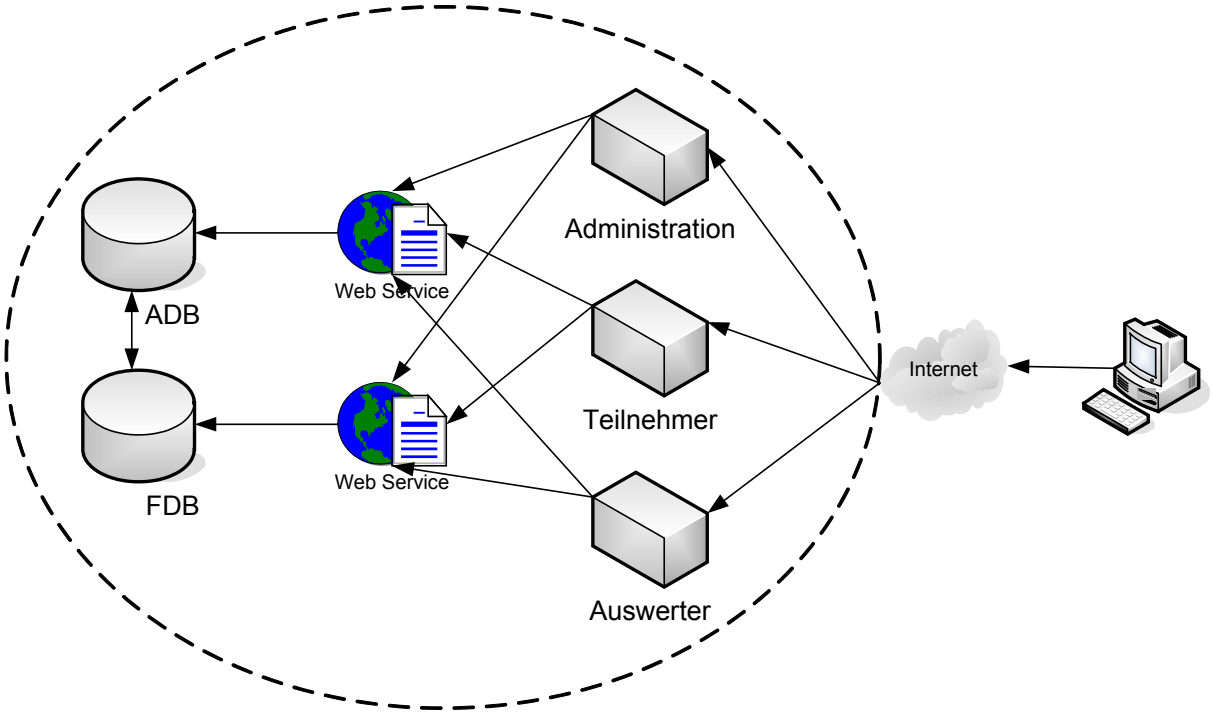

Abbildung 4-1 Kommunikationswege der Module des Evaluationssystems

4.2.1 Administrationsbereich

Diese Schnittstelle ist für die Administratoren des Systems über das Internet erreichbar. Hier kann sich ein Administrator mit seinen Zugangsdaten anmelden. Sowohl sein Loginname als auch sein Passwort werden an den Web Service der ADB übergeben. Sind diese korrekt, erhält der Administrator ein Auswahlmenü, in dem er Vorlesungen und Fragebogentypen editieren kann. Zusätzlich kann er weitere Administratoren und Auswerter anlegen.

Wird ein neuer Administrator erzeugt, werden sein Name und Passwort an den Web Service der ADB gesendet, der diese Daten speichert. Das gleiche geschieht bei der Erstellung eines Auswerters. Hierbei werden aber zusätzlich die Vorlesungen, deren Auswertungen er anschauen darf, hinzugefügt.

Bevor Vorlesungen in das System eingefügt werden können, müssen erst Fragebogentypen erstellt werden. Diesen müssen jeweils eine Bezeichnung und beliebig

viele Fragen mit Antwortmöglichkeiten zugewiesen werden. Die Reihenfolge der Fragen eines Fragebogens lässt sich auch bestimmen. Ist ein Fehler bei der Eingabe aufgetreten, können einzelne Fragen und Antwortmöglichkeiten hinzugefügt, editiert oder gelöscht werden. Auch ein gesamter Fragebogentyp kann wieder entfernt werden. Beim Erstellen, Anzeigen, Editieren und Entfernen der Fragebogentypen kommuniziert der Administrationsbereich ausschließlich über einen Web Service mit der Fragebogendatenbank (FDB).
Es gibt im Evaluationssystem drei unterschiedliche Arten von Fragen zur Auswahl. Die einfachste Form ist die Textfrage. Sie besteht aus einer Frage und einem Antwortfeld. Dieser Fragetyp kann beispielsweise für das Aufnehmen von Anmerkungen dienen. Eine Multiple-Choice-Frage besteht aus einer Frage, hat aber bis zu sieben Antwortmöglichkeiten. Diese werden in Form von Radiobuttons dargestellt, von welchen jeder eine Beschriftung hat. Es könnten entweder diese Werte oder stattdessen zur Berechnung Zahlen als Antwort gespeichert werden. Eine Reihenfrage besitzt eine ähnliche Gestalt. Statt der durchgängigen Beschriftung der Radiobuttons bei der Multiple-Choice-Frage, sind nur der erste und letzte Radiobutton mit gegensätzlichen Werten beschriftet. Die dazwischen liegenden Antwortmöglichkeiten sind Abstufungen zwischen diesen beiden Extrema. Bei diesem Fragetyp kann die Anzahl der Radiobuttons bestimmt werden, so dass z.B. auch eine Frage mit den Antwortmöglichkeiten „ja“ und „nein“ möglich ist. Die dabei gegebenen Antworten werden als Zahl gespeichert.

Wenn der gewünschte Fragebogentyp vorhanden ist, kann eine Vorlesung eingetragen und ihm zugeordnet werden. Die vorhandenen Fragebogentypen werden aus der FDB über einen Web Service herausgeholt und zum Zuordnen angezeigt. Nun kann der Name der Lehrveranstaltung eingegeben und ein Fragebogentyp ausgewählt werden. Diese Daten werden über einen Web Service der FDB übergeben. Der Web Service der ADB erhält nur den Namen der Vorlesung. Alle in der FDB enthaltenen Vorlesungen werden im Administrationsbereich zum Editieren und Löschen angezeigt. Die dafür erforderlichen Daten werden über einen Web Service aus der FDB geholt.
Möchte der Administrator den Administrationsbereich verlassen, kann er sich über einen Logout-Button abmelden.

4.2.2 ADB

Die ADB-Schnittstelle enthält eine Datenbank, die für die Autorisierung aller Daten des Systems zuständig ist.
Hier werden die Zugangsberechtigungsdaten der Administratoren und der Personen, die die Auswertung ansehen dürfen, gespeichert. Außerdem werden hier die Matrikelnummern und Emailadressen der Studenten abgelegt. Die Daten der Administratoren und Auswerter werden in der folgenden Tabelle **`user`** verwaltet.

user

name	password	is_admin
Adminname	Passwort1	1
Auswertername	Passwort2	0

Tabelle 4-1 Benutzer

Sie besteht aus der Primärschlüsselspalte **`name`**, die als Inhalt die Loginnamen der Administratoren enthält, und der Spalte **`password`**, in der die Passwörter abgelegt werden. Die Spalte **`is_admin`** gibt an, ob es sich um einen Administrator oder Auswerter handelt. Handelt es sich um einen Administrator ist der Wert auf 1 gesetzt.

Vorlesungen (**`lecture`**) werden in der Tabelle **`lectures`** gespeichert. Als Primärschlüssel dient eine ID (**`lecture_id`**). Diese Tabelle ist eine Kopie der Tabelle **`lectures`** aus der Fragebogendatenbank. Diese Redundanz wird bewusst in Kauf genommen, um eine feste Trennung der beiden Datenbankmodule zu erhalten.

lectures

lecture_id	lecture
1	Vorlesung1
2	Vorlesung2

Tabelle 4-2 Vorlesungen

Um zu erkennen zu welchen Vorlesungen ein Auswerter die Ergebnisse der Evaluation sehen darf, existiert die Tabelle **`user_lecture`**. Ein Tabelleneintrag enthält einen Benutzernamen und die ID einer Vorlesung, die der Benutzer auswerten darf. Da eine Vorlesung von mehr als einer Person ausgewertet werden kann, und eine Person mehrere Vorlesungen halten kann, wird für jedes Paar aus Benutzername und

Vorlesungs-ID ein eigener Tabelleneintrag angelegt. In dieser Tabelle dienen **`name`** und **`lecture_id`** als zusammengesetzter Primärschlüssel.

user_lecture

user	lecture_id
Auswertername1	1
Auswertername1	2
Auswertername2	2

Tabelle 4-3 Verknüpfungstabelle von Auswertern und Vorlesungen

Ein Administrator taucht in dieser Tabelle nicht auf, da er Zugriff auf die Ergebnisse aller Vorlesungen hat.

Hat ein Student in der Teilnehmerschnittstelle eine PIN zum Evaluieren der Vorlesungen angefordert, erhält die ADB seine Emailadresse (**`email`**) und seine Immatrikulationsnummer (**`matriculation_number`**). Der Student bekommt daraufhin eine PIN an diese Emailadresse gesendet. Diese PIN wird mit seinen anderen Daten auch in dieser Tabelle abgelegt. Die Immatrikulationsnummer dient hier als Primärschlüssel.

students

matriculation_number	email	pin
10345678	student1@uni-goettingen.de	123456789
20345679	student2@uni-goettingen.de	234567890

Tabelle 4-4 Studenten

Die PIN wird vom System automatisch erstellt, wobei darauf geachtet wird, dass sie nicht mit einer schon existierenden identisch ist.
Hat ein Student mit seiner PIN eine Vorlesung evaluiert, werden sowohl die PIN als auch die ID der Vorlesung in folgender Tabelle abgelegt:

pin_lecture

Pin	lecture_id
123456789	1
234567890	2
123456789	2
456789012	4
567890123	12

Tabelle 4-5 Verknüpfungstabelle von verbrauchten PINs und evaluierten Vorlesungen

Die PIN hat hier gemeinsam mit der ID der Vorlesung die Funktion des Primärschlüssels. Wenn eine Vorlesung evaluiert wird, wird in dieser Tabelle nachgeschaut, ob eine Vorlesung mit dieser PIN bereits bewertet worden ist. Wenn das der Fall ist, werden die Antworten nicht gespeichert.
Die ADB-Schnittstelle speichert also Daten für Autorisierungszwecke und kommuniziert über Web Services mit allen drei Schnittstellen, auf die die Anwender des Systems im Internet zugreifen können.

4.2.3 Teilnehmer

Die Teilnehmerschnittstelle ist als Maske über das Internet ohne Login verfügbar. Hier gibt es ein Auswahlmenü, in welchem man zwischen dem Anfordern einer PIN und dem Evaluieren der Vorlesungen auswählen kann. Um überhaupt an der Evaluation teilnehmen zu können, muss der Student erst eine PIN anfordern. Dazu trägt er seine Matrikelnummer und seine Emailadresse in ein Formular ein und sendet es ab. Seine Daten werden über einen Web Service der ADB überbracht und dort gespeichert. Von der ADB bekommt der Student seine PIN über eine Email zugesendet, die er in dieser Schnittstelle anwenden kann.
Das Teilnehmermodul fordert über den Web Service der FDB die Namen aller Vorlesungen an und stellt diese in einem Auswahlformular dar. Möchte der Student eine bestimmte Vorlesung evaluieren, wählt er diese dort aus. Daraufhin sendet die Teilnehmerschnittstelle den Namen der Vorlesung über den Web Service an die FDB und erhält von ihr den dazugehörigen Fragebogen, der als Formular mit Fragen und Antwortmöglichkeiten dargestellt wird. Nun kann der Student seine gewünschte Vorlesung bewerten. Um das Formular abzusenden, muss er seine PIN angeben. Seine Bewertung wird nun mit der PIN über einen Web Service an die FDB geschickt, die sich bei der ADB erkundigt, ob die PIN gültig ist und ob unter dieser PIN noch keine Bewertung für diese Vorlesung vorgenommen worden ist. Falls dies zutrifft, werden die Fragebogendaten abgespeichert.

4.2.4 FDB

Diese Datenbank beinhaltet alle Daten, die mit den Fragebögen zusammenhängen. Hier werden die Vorlesungen mit dazugehörigen Fragebögen, Fragen und Antworten gespeichert. Die FDB bietet einen Web Service zur Kommunikation mit den anderen Modulen an.

Da diese Datenbank eine größere Anzahl an Daten speichern muss, werden ihre Tabellen anhand eines Entity-Relationship-Models (ERM) erstellt.

Wenn eine Vorlesung erzeugt wird, erhält sie zu dem Namen der Vorlesung eine ID. Außerdem wird ihr ein Fragebogentyp zugeordnet. Ein Fragebogentyp enthält Fragen. Zusätzlich wird die Reihenfolge der Fragen im Fragebogentyp festgelegt. Eine Frage kann entweder vom Fragetyp Text, Reihe oder Multiple Choice sein. Ist sie vom Typ Text werden im Gegensatz zu den anderen beiden Fragetypen keine weiteren Daten benötigt. Bei einer Reihenfrage werden der Start- und Endtext sowie die Anzahl der Antwortmöglichkeiten gebraucht. Um eine Multiple-Choice-Frage zu erstellen, können bis zu sieben Antwortmöglichkeiten eingegeben werden. Außerdem muss entschieden werden, ob eine Antwort als Zahl gespeichert wird.

Wird ein Fragebogen zu einer Vorlesung ausgefüllt, erhält dieser eine ID. Er beinhaltet Antworten zu den Fragen.

Aus diesem Kontext wird folgendes ERM gestaltet, aus welchem die unterschiedlichen Tabellen der FDB erstellt werden.

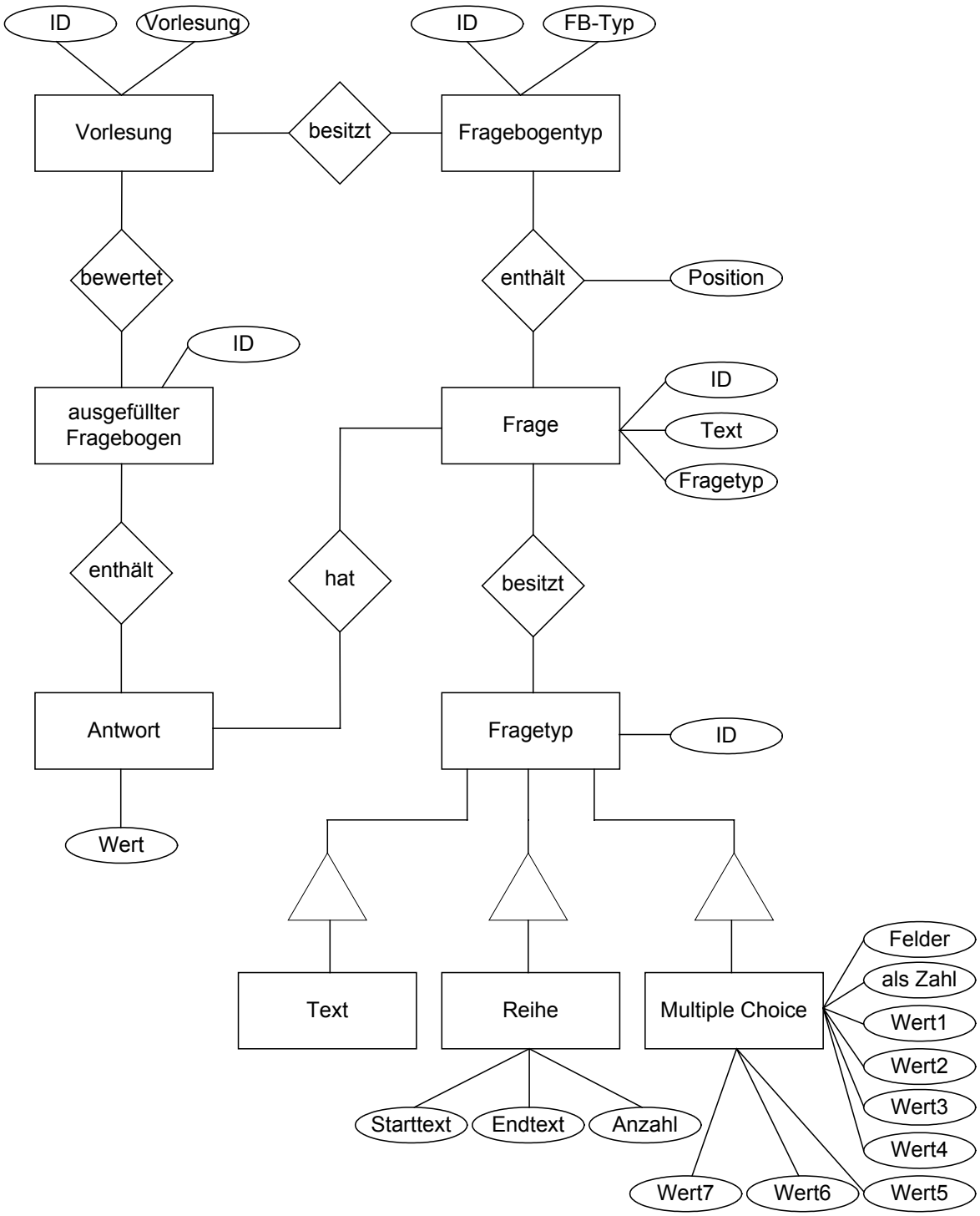

Abbildung 4-2 ERM zur FDB

Das ERM enthält die Entitäten **`Vorlesung`**, **`Fragebogentyp`**, **`ausgefüllter Fragebogen`**, **`Frage`**, **`Fragetyp`**, **`Text`**, **`Reihe`**, **`Multiple Choice`** und **`Antwort`**. Eine Vorlesung enthält als Attribute eine ID und den Namen der Vorlesung. Da eine Vorlesung einen Fragebogentyp besitzt, besteht zwischen diesen beiden eine Beziehung. Ein Fragebogentyp trägt als Attribut auch eine ID und den Namen des Typs. Ein Fragebogentyp besteht aus Fragen, die von den unterschiedlichen Fragetypen Text, Reihe und Multiple Choice sein können. Die Relation zwischen Fragebogentyp und Frage

enthält das Attribut **Position**, das die Position der Frage im Fragebogen angibt. Die Entität **Frage** beinhaltet die Attribute **ID** sowie den Text und den Typ der Frage. Zwischen **Frage** und **Fragetyp** besteht eine Beziehung, da eine Frage einen Fragetyp besitzt. Dieser Fragetyp kann eine Textfrage (**Text**), Reihenfrage (**Reihe**) oder Multiple-Choice-Frage (**Multiple Choice**) sein. Diese drei Entitäten erben von der Entität **Fragetyp**. Zusätzlich zu der ID der Entität **Fragetyp** können sie andere Attribute besitzen. Die Entität **Multiple Choice** enthält zusätzlich zu den sieben Antwortmöglichkeiten die Attribute **Felder** und **als Zahl**. Das Attribut **Felder** gibt die Anzahl der anzuzeigenden Antwortmöglichkeiten an, während **als Zahl** anzeigt, ob als Antwort eine Zahl oder der Name, den die ausgewählte Antwortmöglichkeit trägt, gespeichert werden soll. Die Entität **Reihe** trägt die Attribute **Starttext**, **Endtext** und **Anzahl**. Die Entität **Text** enthält zu der ID keine weiteren Attribute.
Die Entität **ausgefüllter Fragebogen** enthält als Attribut eine ID. Ein ausgefüllter Fragebogen bewertet Vorlesungen und enthält Antworten. Deshalb besteht zu beiden Entitäten eine Beziehung. Die Entität **Antwort** beinhaltet den Wert einer Frage als Attribut. Da zu einer Frage Antworten gehören, existiert zwischen den Entitäten **Antwort** und **Frage** auch eine Relation.

Wird in der Administrationsschnittstelle ein Fragebogentyp mit dem Namen „Vorlesungsfragebogen“ erstellt, wird er in der Tabelle **fb_types** abgelegt, welche folgende Form hat.

fb_types

fb_type_id	fb_type
1	Vorlesungsfragebogen

Tabelle 4-6 Fragebogentypen

Den Primärschlüssel der Tabelle bildet die Spalte **fb_type_id**, der Name des Fragebogentyps wird in der Spalte **fb_type** genannt. Diese Tabelle stellt die Entität **Fragebogentyp** dar.

Wenn dem Fragebogentyp Vorlesung Fragen hinzugefügt werden sollen, werden diese in der Tabelle **questions** gespeichert. Der Primärschlüssel ist die ID der Frage (**question_id**) und der Text der Frage wird in **text** abgelegt.

questions

question_id	text	question_type
1	Wie interessant war die Vorlesung für Sie?	range
2	Wie hoch war der Zeitaufwand?	multiple_choice
3	Anmerkungen	text

Tabelle 4-7 Fragen

Damit erkennbar ist, von welchem Typ die Frage ist, wird ihr der Fragetyp (**question_type**) zugeordnet. Diese Tabelle entspricht der Entität **Frage**.
Die Beziehung zwischen den Entitäten **Fragebogentyp** und **Frage** wird in der Tabelle **fb_types_questions** realisiert. Sie verknüpft die beiden IDs **fb_type_id** und **question_id** und bildet daraus den Primärschlüssel.

fb_types_questions

fb_type_id	question_id	position
1	1	1
1	2	2
2	2	1

Tabelle 4-8 Verknüpfungstabelle von Fragebögen und Fragen

Zusätzlich wird die Position, an der die Frage stehen soll, beigefügt.

Da die Entität **Fragetyp** nur eine ID enthält und diese in den von ihr erbenden Entitäten **Text**, **Reihe** und **Multiple Choice** auch vorkommt, wird aus dieser Entität keine Tabelle gebildet. Der Fragetyp Text wird auch nicht realisiert, weil er keine zusätzlichen Attribute enthält und bereits schon in der Tabelle **questions** enthalten ist. Die anderen beiden Entitäten **Reihe** und **Multiple Choice** werden aber in jeweils einer Tabelle dargestellt.

Wird eine Multiple-Choice-Frage erstellt, werden ihre Daten in der Tabelle **multiple_choice** dargestellt.

multiple_choice

mc_id	max_ fields	value_ as_nr	value1	value2	value3	value4	value5	value6	value7
1	4	0	<4h	4h-8h	8h-16h	>16h			

Tabelle 4-9 Multiple Choice

Die Werte der Frage „Wie hoch war der Zeitaufwand?“ mit der Frage-ID 2 lauten „<4h“, „4h – 8h“, „8h – 16h“ und „>16h“. Diese werden in den Feldern mit der Bezeichnung **valueN** gespeichert. Da im Beispiel nur vier Felder gefüllt werden, bleiben die restlichen leer und die Spalte **max_fields** wird auf 4 gesetzt. Die ID des Eintrags (**mc_id**) wird automatisch gesetzt und dient als Primärschlüssel. Normalerweise besteht die Antwort zu einer Frage wie in diesem Fall aus dem Wert der ausgewählten Antwortmöglichkeit. Dies könnte beispielsweise der Wert „4h – 8h“ sein und das Feld (**value_as_nr**) würde den Eintrag 0 erhalten. Soll die Antwort aber zur Berechnung als Zahl dargestellt werden, erhält das Feld den Wert 1.

Die Entität **Reihe** enthält zusätzlich zu dem Primärschlüssel **range_id**, zwei Textfelder, die die Anfangs- (**start_text**) und Endbezeichnung (**end_text**) beschreibt. Die Frage „Wie interessant war die Vorlesung für Sie?“ mit der ID 1 enthält als Starttext den Wert „sehr interessant“ und als Endtext den Wert „völlig uninteressant“. Dazwischen können beliebig viele Antwortmöglichkeiten stehen. Die Anzahl dieser Antworten wird in der Spalte **range** bestimmt. In diesem Beispiel erhält sie den Eintrag 5. Die Antwort einer solchen Frage wird als Zahl gespeichert.

range

range_id	start_text	end_text	range
1	sehr interessant	völlig uninteressant	5

Tabelle 4-10 Reihe

Die Beziehungen zwischen den Entitäten Reihe und Multiple-Choice zu der Entität Frage werden durch Verknüpfungstabellen realisiert.

Die Verknüpfungstabelle **question_range** von Reihe und Frage hat folgende Gestalt.

question_range

question_id	range_id
1	1

Tabelle 4-11 Verknüpfungstabelle von Fragen mit Reihen-Antwortmöglichkeiten

Der ID der Frage, die als Primärschlüssel dient, wird die ID der Reihentabelle (**range_id**) zugeordnet.

Die Tabelle, die die Multiple-Choice-Antwortmöglichkeiten mit den Fragen verknüpft, hat die gleiche Gestalt. Der einzige Unterschied zu der vorherigen Tabelle ist, dass sie anstatt der range_id die ID der Multiple-Choice-Fragen (**mc_id**) hat.

question_mc

question_id	mc_id
1	2
5	7

Tabelle 4-12 Verknüpfungstabelle von Fragen mit Multiple-Choice-Antwortmöglichkeiten

Wird nun in der Administrationsschnittstelle eine Vorlesung namens „Informatik 4“ erstellt, wird sie in der Tabelle **lectures** gespeichert, die aus der Entität Vorlesung entstanden ist.

lectures

lecture_id	lecture
1	Informatik 4

Tabelle 4-13 Vorlesungen

Sie wird aber nicht nur in der Tabelle **lectures** gespeichert, sondern auch in einer Tabelle, in der den Vorlesungen ein Fragebogentyp zugeordnet wird.

lecture_fb_type

lecture_id	fb_type_id
1	1

Tabelle 4-14 Verknüpfungstabelle von Vorlesungen und Fragebogentypen

Als Primärschlüssel dient die ID der Vorlesung (**lecture_id**). Die Spalte **fb_type_id** verweist auf die Tabelle der Fragebogentypen. Da der Vorlesung „Informatik 4“ der Fragebogentyp „Vorlesungsfragebogen“ zugeordnet worden ist, werden hier ihre ID und die ID des Fragebogentyps „Vorlesungsfragebogen“ abgelegt. Die Verknüpfungstabelle stellt die Beziehung zwischen den Entitäten Vorlesung und Fragebogentyp dar.

Wurde ein Fragenbogen von einem Studenten ausgefüllt, erhält dieser Fragebogen eine ID, die in der Tabelle **filled_out** gespeichert wird. Diese Tabelle ist eine Besonderheit. Sie hat nur eine Zeile, in der die ID des zuletzt benutzen Fragebogens steht, welche in diesem Beispiel 24 lautet.

filled_out

filled_out_id
24

Tabelle 4-15 ID des zuletzt ausgefüllten Fragebogens

Diese Tabelle entspricht der Entität ausgefüllter Fragebogen.

Die Beziehung der Entitäten ausgefüllter Fragebogen und Vorlesung werden durch die Verknüpfungstabelle **filled_out_lecture** realisiert. Der ID des ausgefüllten Fragebogens wird die ID der evaluierten Vorlesung zugeordnet und dient als Primärschlüssel. Diese Tabelle hat folgende Gestalt:

filled_out_lecture

filled_out_id	lecture_id
...	...
24	1

Tabelle 4-16 Verknüpfungstabelle von ausgefüllten Fragebögen und Vorlesungen

Die Antworten eines Fragebogens werden in der Tabelle **answer** gespeichert, die gleichzeitig die Entität Antwort und ihre Beziehungen zu den Entitäten ausgefüllter Fragebogen und Frage darstellt.
Diese besitzt den Primärschlüssel **answer_id**, die ID des ausgefüllten Fragebogens, die ID der Frage und ein Feld für die Antwort (**answer**). Anhand der IDs kann man den gesamten ausgefüllten Fragebogen rekonstruieren.

answer

answer_id	filled_out_id	question_id	answer
...	...	...	...
69	24	1	2
70	24	2	>16h
71	24	3	Mein Vorschlag wäre ...

Tabelle 4-17 Antworten

In diesem Beispiel stehen die Ergebnisse des ausgefüllten Fragebogens mit der ID 24. Die erste Antwort erhält die ID 69 gehört zu der Reihenfrage „War die Vorlesung interessant?“ und erhält als Wert die Zahl 2. Die nachfolgende Antwort ist eine Multiple-Choice-Frage und das Feld answer erhält den Wert „>16h“. Der letzte Eintrag der Tabelle besteht aus der Antwort auf eine Textfrage.

Soll bei den Antworten zu den Fragetypen Reihe und Multiple Choice eine Frage gespeichert werden, richtet sich diese nach den Einträgen **`range`** in der Tabelle **`range`** oder nach **`max_fields`** in der Tabelle **`multiple_choice`**. Wurde die Antwortmöglichkeit mit der Anfangsbezeichnung (**`start_text`**) oder des ersten Wertes (**`value1`**) gewählt, wird hier der Wert 1 gespeichert. Wenn die Endbezeichnung angekreuzt worden ist, enthält das Feld **`value`** den Wert von **`range`** oder **`max_fields`**. Ist eine Antwortmöglichkeit dazwischen ausgewählt worden, wird diese Zahl angegeben.

4.2.5 Auswerter

Der Web Service des Auswerterbereichs ist wieder eine Schnittstelle, die über das Internet ansprechbar ist. Hier können sich beispielsweise Dozenten über ein Formular einloggen und sich die Ergebnisse der Evaluationen zu ihren Vorlesungen anzeigen lassen. Die Login-Daten werden über einen Web Service an die ADB gesendet und dort verglichen. Daraufhin erhält das Auswertermodul die Namen der Lehrveranstaltungen, die der authentifizierte Benutzer anschauen darf. Der Auswerter kann dann eine dieser Vorlesungen auswählen. Daraufhin wird der Name der gewählten Vorlesung über einen Web Service an die FDB gesendet, welcher die Ergebnisse der Evaluation dieser Vorlesung zurückliefert. Diese Ergebnisse werden für den Dozenten graphisch dargestellt. Wenn er diesen Bereich verlassen möchte, muss er sich über einen Button abmelden.

4.3 DV-Konzept

In diesem Abschnitt werden die Klassen mit ihren Methoden und die Architektur der Online-Evaluationssystems beschrieben.

4.3.1 Architektur

Die Architektur des Online-Evaluationssystems orientiert sich an der Model-View-Control-Architektur. Die Anwendungslogik (Model) beinhaltet Datenbanken und Datenbankzugriffsfunktionen. Die Präsentation (View) ist das, was der Anwender sieht, und besteht aus JSP- und HTML-Seiten. Zur Ablaufsteuerung (Control) werden Servlets, die die Anwendungslogik und die Präsentation steuern, gezählt.

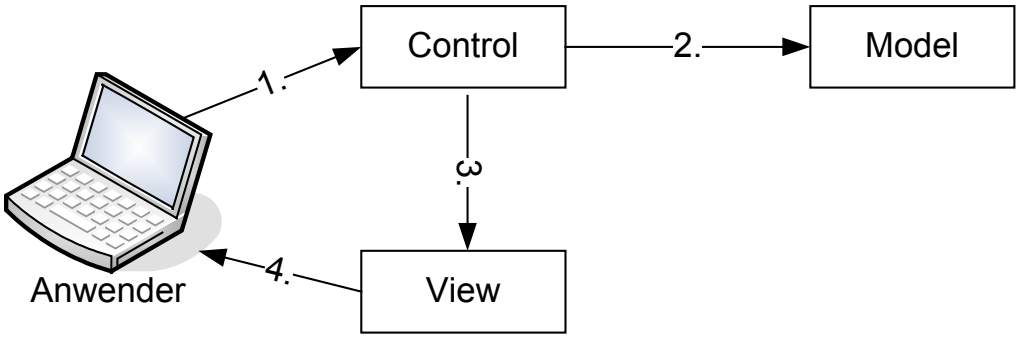

Abbildung 4-3 Model-View-Control

Der Anwender wählt die URI (Uniform Resource Identifier) der Ablaufsteuerung an und stellt eine Verbindung zu ihr her. Die Ablaufsteuerung fordert Daten von der Anwendungslogik und sendet diese an die Präsentation. Die Präsentation stellt schließlich diese Daten für den Anwender im Browser dar.

Beim Online-Evaluationssystem gibt es die zwei Anwendungslogiken ADB und FDB, die drei Ablaufsteuerungen und die drei Sichten für den Administrations-, den Auswerter- und den Teilnehmerbereich. Die Steuerung und Präsentation jedes Bereichs sind miteinander verbunden. Aber zwischen einer Anwendungslogik und einer Ablaufsteuerung besteht keine direkte Verbindung. Die Kommunikation findet nur über die von den Anwendungslogiken angebotenen Web Services statt. Jedem Steuerungsmodul sind Klientmethoden zugeordnet, die auf die beiden Web Services zugreifen können. In folgender Darstellung ist die erweiterte Model-View-Control des Administrationsbereichs zu sehen. Die des Teilnehmer- und Auswerterbereichs haben dieselbe Gestalt.

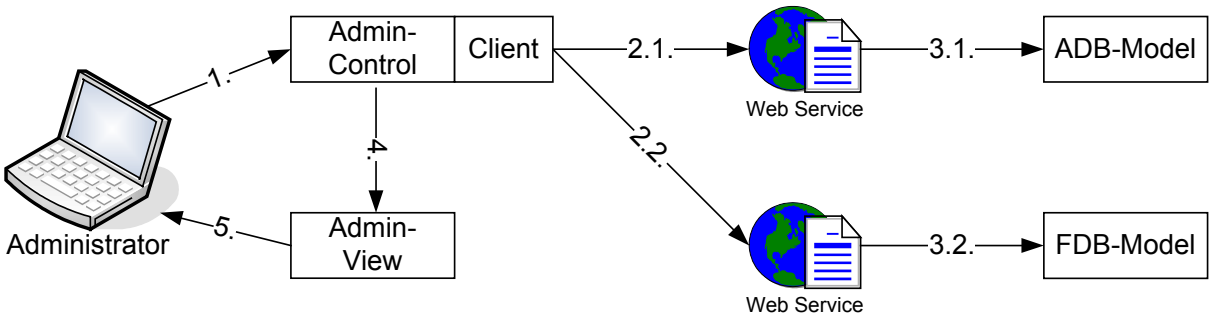

Abbildung 4-4 erweiterte Model-View-Control für den Administrationsbereich

Zwischen den beiden Datenbankmodulen findet auch Datenaustausch über Web Services statt - diese sind aber aus Gründen der Übersichtlichkeit in dieser Abbildung nicht zu finden.

Für die Anwender des Administrations-, Teilnehmer- und Auswerterbereich sind nur die unterschiedlichen Masken der Bereiche im Internet sichtbar, deren Aussehen durch JSP- und HTML-Dateien bestimmt wird. Diese Seiten enthalten keine Funktionen, sondern werden durch die Ablaufsteuerung gesteuert, die diese je nach übergebener Aktion im Formular aufruft und mit Informationen füllt. Auf der anderen Seite steuert sie die Kommunikation mit den Web Services über Klienten. Wenn der Benutzer einen bestimmten Datensatz sehen möchte und auf den entsprechenden Button in seiner Maske klickt, erhält sie eine Aktion und ruft den dazugehörigen Klienten auf, der diese Daten wiederum an den Web Service übergibt. Der Web Service stellt eine Verbindung zur Datenbank her und gibt den gewünschten Datensatz an den Klienten zurück. Die Ablaufsteuerung erhält diese Daten vom Klienten und übergibt sie anschließend an die JSP-, bzw. HTML-Dateien, die diese z.B. in einer Tabelle darstellen.

4.3.2 Methodenbeschreibung

Hier werden die Klassen und Methoden des Online-Evaluationssystems beschrieben.

4.3.2.1 Ablaufsteuerung des Administrationsbereichs

Die Module Administration, Teilnehmer und Auswerter stellen die Ablaufsteuerung und Präsentation dar. Jedes Modul besitzt eine eigene Ablaufsteuerungsklasse, die die Anzeige der JSP- und HTML-Seiten steuert. Die Klasse **`Control.java`** der Steuerung der Administration ist nicht nur für die Seiten der Administration zuständig, sondern regelt auch die allgemeine Startseite.

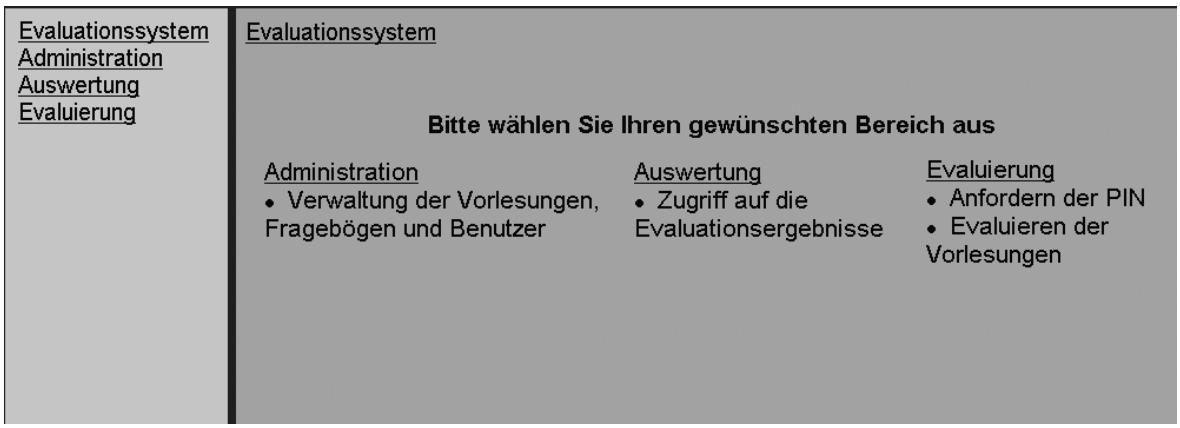

Abbildung 4-5 Startseite des Evaluationssystems

Diese Startseite verweist auf die Administration, die Auswertung und die Evaluierung. Beim Aufruf der beiden letztgenannten, wird die Steuerung der Auswertung (**`AnalyzerControl.java`**) und der Teilnehmer (**`StudentControl.java`**) aufgerufen. Diese werden einen ähnlichen Aufbau wie **`Control.java`** haben. Die Steuerung der Administration unterscheidet zwischen Seiten des Administrationsbereichs und der Startseite. Eine Variable **`siteindex`** weist die jeweilige Seite zu.

```
if(siteindex==1) response.sendRedirect("code/normal_view.jsp");
if(siteindex==2) response.sendRedirect("code/admin/admin_view.jsp");
```

Wird diese Variable auf **`1`** gesetzt, wird die Seite des Startbereichs angezeigt, wird sie auf **`2`** gesetzt, erscheint die Seite des Administrationsbereichs. Diese beiden Seiten bestimmen die Darstellung im Browser und integrieren bei jeder Aktion eine bestimmte Seite. Lautet die Aktion beispielsweise **`login`**, wird die Seite **`login.jsp`** eingebaut. Zusätzlich integrieren sie noch die Navigationsleiste.

Eine Aktion wird der URI der Ablaufsteuerung folgendermaßen mitgegeben:
`http://../Control?action=aktion`
Aus einer Sitzung wird dann der Parameter der Aktion gelesen.

```
HttpSession session = request.getSession();
String action=request.getParameter("action");
```

Neben dieser String-Variablen können noch andere Parameter wie beispielsweise Logindaten, Vorlesungen und Fragebogentypen der Steuerung mitgegeben und ausgelesen werden.
Ist der Sitzung keine Aktion mitgegeben, wird automatisch die Aktion **`userindex`** aufgerufen, welcher die Variable **`siteindex=1`** zugeordnet ist. Die Startseite des Online-

Evaluationssystems wird aufgerufen. Die Aktion **adminindex** ist das Gegenstück zu **userindex** und ihr wird der **siteindex 2** zugewiesen.

Möchte der Administrator sich einloggen, wird die Aktion **login** der Sitzung mitgegeben, dann wird eine Maske angezeigt, in der er aufgefordert wird, seinen Benutzernamen (**adminname**) und sein Passwort (**password**) einzugeben. Betätigt er den Button zum Absenden, werden der Sitzung die Aktion **checkit** und die Logindaten mitgegeben.

Die Ablaufsteuerung übergibt der Funktion **LoginCallADB** des Klienten **ADBServiceClient** die Parameter **adminname** und **password**. Als Rückgabewert erhält sie einen String. Hat dieser den Wert **ok**, stimmen die Logindaten mit den Daten in der ADB überein. Dem Administrator wird erlaubt, den Administrationsbereich zu betreten. Der **siteindex** wird auf **2** gesetzt und die Aktion **adminindex** übergeben, woraufhin die Startseite des Administrationsbereichs angezeigt wird.

```
if(action.equals("checkit")) {
    session.putValue("adminname",adminname);
    session.putValue("password", password);
    String testen="";
    String test1="";
    testen=call.LoginCallADB(adminname, password);

    if(testen.equals("ok")) {
        siteindex=2;
        action="adminindex";
    }
    else {
        siteindex=1;
        action="userindex";
    }
}
```

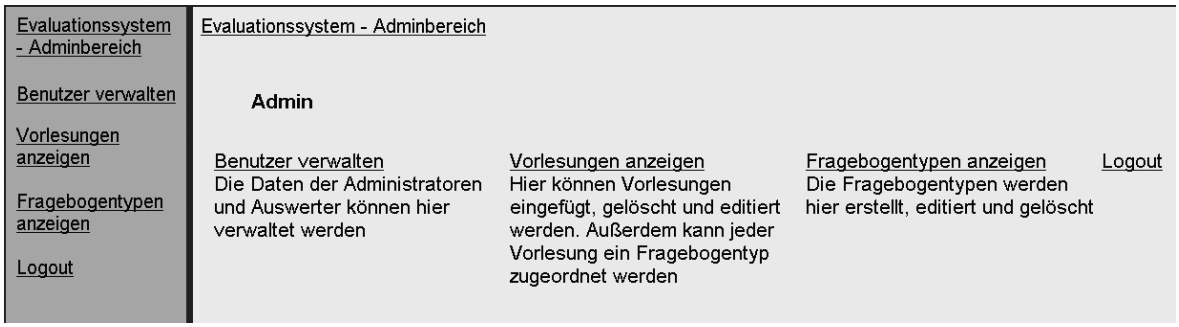

Abbildung 4-6 Startseite des Administrationsbereichs

Hier kann der Administrator zwischen dem Verwalten der Benutzer, Vorlesungen und Fragebogentypen wählen.

Möchte der Administrator neue Administratoren oder Auswerter hinzufügen, wählt er den Bereich „Benutzer verwalten“ aus. Dadurch wird die Aktion **`Users`** aufgerufen, die den Klienten des Web Services der ADB aufruft und von ihm die Daten der Benutzer in Form eines Vektors erhält.

```
if(action.equals("Users")){
    listL=call.showUsers();
    session.putValue("list", listL);
    siteindex=2;
}
```

Diese Daten werden in einer JSP-Seite für den Benutzer dargestellt. Für das Hinzufügen eines neuen Benutzers ist die Aktion **`newUser`** zuständig. Wird sie betätigt, erhält der Administrator eine Maske, in der er den Namen und das Passwort des neuen Benutzers eingeben kann. Zusätzlich befindet sich dort ein Auswahlfeld, in welchem er angeben kann, ob es sich bei der neuen Person um einen Administrator oder Auswerter handelt. Sind die Daten eingegeben kann der Button betätigt und damit die Aktion **`storeUser`** aufgerufen werden. Diese übergibt die Ablaufsteuerung dem Klienten des Web Services der ADB die neuen Benutzerdaten, welche in der ADB abgelegt werden.

```
if(action.equals("storeUser")){
    session.putValue("newUser", newUser);
    session.putValue("newPassword", newPassword);
    session.putValue("is_admin", is_admin);
    call.insertUser(newUser, newPassword, is_admin);
    siteindex=2;
}
```

Handelt es sich bei dem neu erzeugten Benutzer um einen Auswerter, können ihm Lehrveranstaltungen zugewiesen werden, deren Auswertungen er ansehen darf. Zusätzlich zu der Erstellung eines neuen Benutzers können diese editiert oder entfernt werden.

Klickt der Administrator auf „Vorlesungen anzeigen“, wird die Aktion **`showLectures`** aufgerufen. Nun werden die in der FDB enthaltenen Vorlesungen mit ihren Fragebogentypen angezeigt. Diese Aktion ruft den Klienten des FDB-Services auf, der die Vorlesungen mit entsprechendem Typ aus der Fragebogendatenbank holen soll. Diese Daten werden in Form eines Vektors von dem Klienten der **`Control`** übergeben, welche diesen Vektor **`lecturesList`** der Sitzung übergibt. Die JSP-Seite liest ihn aus der Sitzung aus und stellt die Vorlesungen mit ihren Fragebogentypen dar.

```
if(action.equals("showLectures")){
    lecturesList=fdb_call.showLecturesWithTypes();
    session.putValue("lectureL",lecturesList);
    siteindex=2;
}
```

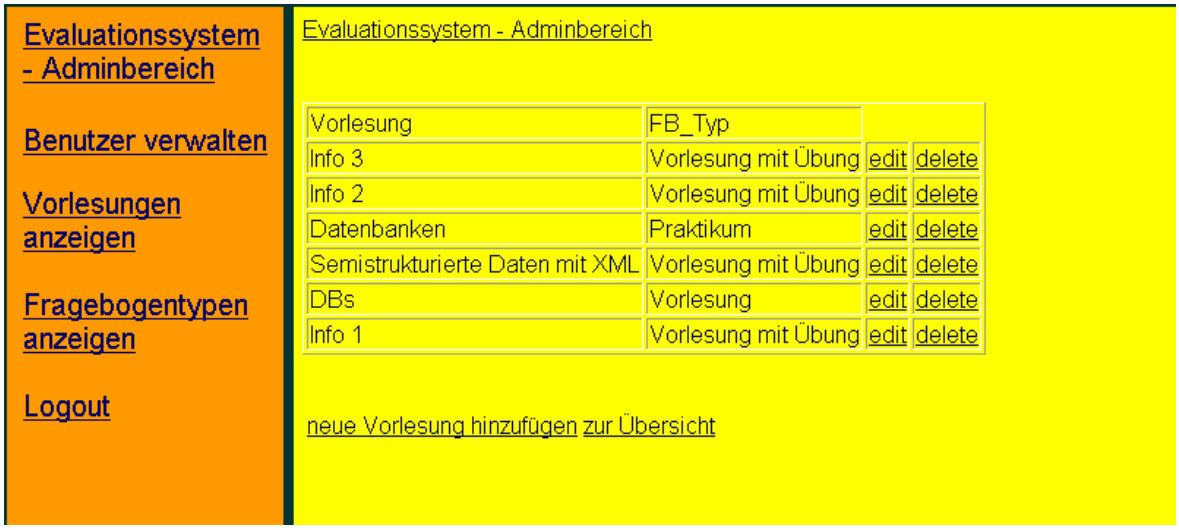

Abbildung 4-7 Vorlesungsübersicht

Hier kann der Administrator wählen, ob er eine neue Vorlesung hinzufügen (**`createLecture`**), editieren (**`editLecture`**) oder löschen (**`deleteLecture`**) möchte. Wählt er das Erstellen einer Vorlesung werden die Vorlesungstypen von **`FDBtoAdminService`** über den Klienten angefordert. Diese werden auch in einem Vektor ausgelesen und in einem Auswahlfeld angezeigt. Der Administrator kann den Namen einer Vorlesung eingeben und ihr einen Typ zuweisen. Dieser Name und Typ werden mit der Aktion **`newLecture`** der Sitzung mitgegeben. Diese Aktion übergibt dem Klienten des **`FDBtoAdminService`** beiden Daten.

```
if(action.equals("newLecture")){
    session.putValue("lecture", lecture);
    session.putValue("fb_type", fb_type);
    fdb_call.createLecture(lecture, fb_type);
    siteindex=2;
}
```

Soll eine Vorlesung editiert werden, wird die Aktion **`editLecture`** aufgerufen, die eine Maske anzeigt, in der der Name der Vorlesung und ihr Fragebogentyp geändert werden können. Betätigt der Administrator den Absende-Button, werden der Ablaufsteuerung die Aktion **`saveLecture`**, der alte und der neue Namen der Lehrveranstaltung (**`lectures`**) und ihr geänderter Fragebogentyp (**`fb_type`**) übergeben. Diese Aktion ruft, wie im obigen Beispiel, den Klienten des Web Services der FDB auf. Allerdings wird hier noch zusätzlich der alte Name der Vorlesung (**`lecture_old`**) mitgegeben.

```
if(action.equals("saveLecture")){
    session.putValue("lecture",lecture);
    session.putValue("fb_type", fb_type);
    session.putValue("lecture_old", lecture_old);
    fdb_call.saveLecture(lecture, fb_type, lecture_old);
    siteindex=2;
}
```

Wird eine Vorlesung gelöscht, wird die Aktion **removeLecture** aufgerufen. Diese gibt intern, d. h. es wird keine Klienten-Methode aufgerufen, die Daten der zu löschenden Vorlesung weiter und stellt eine Sicherheitsabfrage, ob diese Daten wirklich gelöscht werden sollen, im Browser dar. Wird dies bestätigt, werden in der Aktion **deleteLecture** dem Klienten des FDB-Web Services die Daten der zu löschenden Vorlesung übergeben.

Bei den Fragebogentypen sind die Aktionen ähnlich wie die der Vorlesungen aufgebaut. Es können Fragebögen angezeigt (**showFB_Types**), erstellt (**createFB_Type**, **newFB_Type**), geändert (**editFB_Type**) und entfernt (**removeFB_Type**, **deleteFB_Type)** werden. Die Fragen der jeweiligen Fragebogentypen können auch erstellt (**createQuestion**, **newQuestion**), hinzugefügt (**addQuestion**), geändert (**editQuestion**, **saveQuestion**) und gelöscht (**removeQuestion**, **deleteQuestion**) werden. Beim Anlegen der Fragen gibt es aber für jeden Fragetyp eine eigene Maske. Die Maske für eine Text-Frage wird durch die Aktion **textQuestion** aufgerufen. Soll eine Reihenfrage erstellt werden, wird die Aktion **rangeQuestion** gewählt. Eine Maske der Multiple-Choice-Frage erhält der Administrator durch das Betätigen der Aktion **mcQuestion**.

4.3.2.2Web Services

Jedes Datenbankmodul hat als Schnittstelle einen Web Service, auf den die Klienten aus dem Administrationsbereich zugreifen. Der Klient **ADBServiceClient** ruft den Web Service **ADBService** der ADB und der Klient **FDBtoAdminServiceClient** den Web Service **FDBtoAdminService** der FDB auf. Da die Lehrveranstaltungen aus der FDB in eine Tabelle in der ADB kopiert werden, bietet die ADB zusätzlich den Web Service **FDBtoADBService** an. Der Klient **FDBtoADBClient** der FDB übergibt diesem Web Service die Daten der Lehrveranstaltungen.

Damit die Klienten aber auf die Web Services zugreifen können, müssen diese mittels Deployment zur Verfügung gestellt werden. Dies geschieht durch die drei Deployment-Deskriptoren **ADBServiceDeploy.wsdd**, **FDBtoAdminServiceDeploy.wsdd** und **FDBtoADBServiceDeploy.wsdd**. Sind diese Web Services in der Axis Engine installiert, können sie genutzt werden. Wird das AxisServlet im Browser angesprochen, werden sie mit ihren Methoden neben den anderen Web Services AdminService und

Version aufgeführt. Folgt man den Links, gelangt man zu den WSDL-Dokumenten der Web Services.

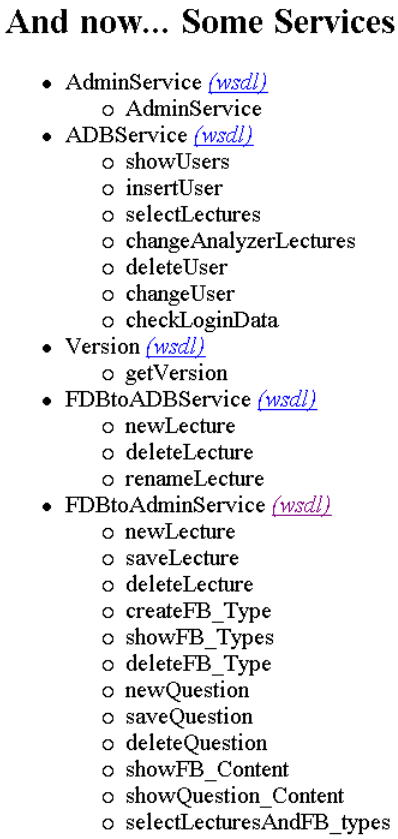

Abbildung 4-8 AxisServlet zeigt eine Auflistung aller Web Services

Wie man sieht, hat der Web Service der Autorisierungsdatenbank die Methoden **`showUsers`**, **`insertUser`**, **`selectLectures`**, **`changeAnalyzerLectures`**, **`deleteUser`**, **`changeUser`** und **`checkLoginData`**. Einige Methoden bekommen von den Klienten des Administrationsbereichs Werte übergeben, die sie an die Datenbanken weiterleiten sollen. Andere senden aber auch Daten aus der Datenbank an die Klienten, welche auf für die Benutzer des Systems dargestellt werden.

Genauso verhält es sich bei den Methoden des Web Services **`FDBtoAdminService`** der von dem FDB-Modul angeboten wird. Seine Methoden sind **`newLecture`**, **`saveLecture`**, **`deleteLecture`**, **`createFB_Type`**, **`showFB_Types`**, **`deleteFB_Type`**, **`newQuestion`**, **`saveQuestion`**, **`deleteQuestion`**, **`showFB_Content`**, **`showQuestionContent`** und **`selectLecturesAndFB_types`**.

Beim Web Service **`FDBtoADBService`** findet die Kommunikation nur zwischen den beiden Datenbanken statt. Bei einer Änderung der Daten der Tabelle **`lectures`** aus der FDB werden die neuen Daten an die Methoden **`newLecture`**, **`renameLecture`** oder **`deleteLecture`** gesendet.

Die folgende Abbildung zeigt einen möglichen Verlauf der Kommunikation mit den Web Services der ADB und FDB und dem Administrationsbereich.

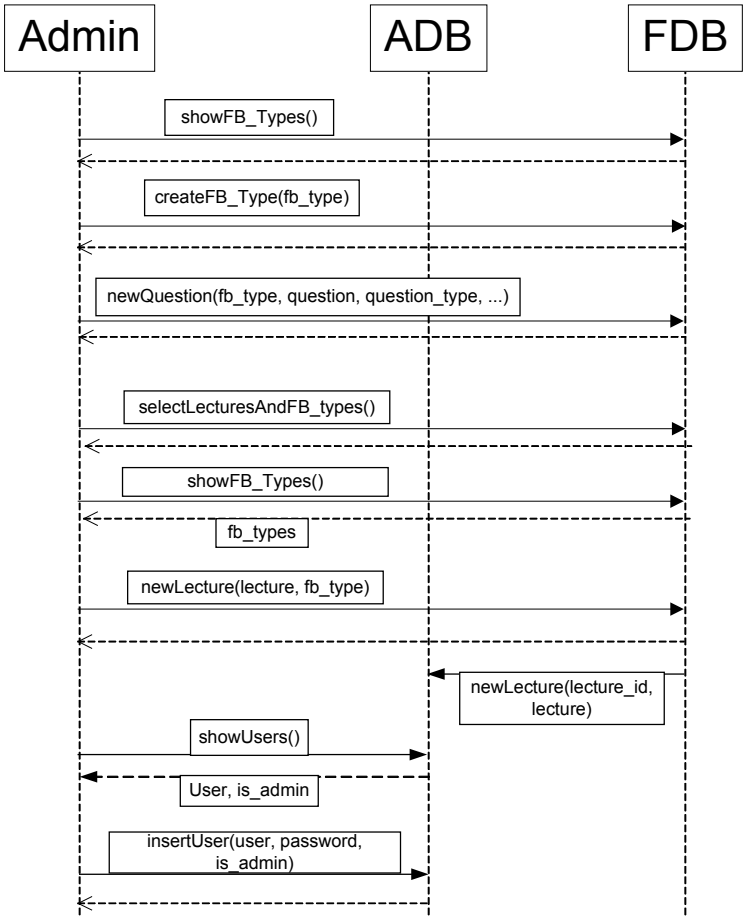

Abbildung 4-9 Kommunikation zwischen Administrationsbereich, ADB und FDB

Dieses Beispiel stellt den allerersten Besuch des Administrationsbereichs nach der Installation dar. Es sind noch keine Daten in den Datenbanken enthalten. Als erstes muss sich der Administrator in den Administrationsbereich einloggen. Hierbei werden der Methode **`checkLoginData`** des Web Services der ADB die Logindaten übergeben.

```
public String checkLoginData(String admin, String password) {
    String s;
    ADBoperations op = new ADBoperations();
    s = op.checkAdminLogin(admin, password);
    return s;
}
```

Diese Methode ruft die Datenbankmethode **`checkAdminLogin`** auf, um die Logindaten mit den Daten aus der ADB abzugleichen. Sind diese korrekt, erhält sie in Form eines

Strings die Erlaubnis den Administrationsbereich zu betreten. Dieser String wird an den Klienten weitergegeben, der diesen auswertet.

Jetzt müssen die Fragebogentypen erstellt werden. Dazu ruft der Klient die Methode **showFB_Types** des Web Services der FDB auf und lässt sich alle in der FDB befindlichen Fragebogentypen als Vektor zurückgeben.

```
public Vector showFB_Types() {
    Vector fb_types;
    fb_types =  new Vector();
    FDBoperations op = new FDBoperations();
    fb_types=op.selectAllFB_Types();
    return(fb_types);
}
```

Da aber keine Daten in der Datenbank vorhanden sind, bekommt der Klient einen leeren Vektor zurück. Wird nun ein Fragebogentyp erstellt, ruft der Klient die Methode **createFB_Type** des FDB-Web-Services auf und übergibt ihr den Parameter **fb_type**, den er von der Steuerung bekommen hat.

```
public void newFB_Type(String fb_type){
    try{
        Service service = new Service();
        Call call =(Call) service.createCall();

        call.setTargetEndpointAddress(new java.net.URL(endpoint));
        call.setOperationName("createFB_Type");
        String fbt = new String(fb_type);
        String test= (String) call.invoke(new Object[] {fbt});
    }catch(Exception e) {
        System.err.println(e.toString());}
}
```

Die aufgerufene Methode des **FDBtoAdminService** sendet nun den Namen des Fragebogentyps an die Methode **createFB_Type**, die diesen in der FDB speichert.

```
public void createFB_Type(String fb_type){
    FDBoperations op = new FDBoperations();
    op.createFB_Type(fb_type);
}
```

Wenn eine Frage hinzugefügt wird, erhält die Methode **newQuestion** den Namen des Fragebogentypen (**fb_type**), den Text der Frage (**question**), den Typ der Frage (**question_type**) sowie den Inhalt der Antwortmöglichkeiten.

```
public void newQuestion(String fb_type, String f_type, String question,
    String range, String start_text, String end_text, String v_as_nr,
    String v1, String v2, String v3, String v4, String v5, String v6,
    String v7, String position){

    FDBoperations op = new FDBoperations();
    op.insertQuestion(fb_type, f_type, question, range, start_text,
        end_text, v_as_nr, v1, v2, v3, v4, v5, v6, v7, position);
}
```

Der **FDBtoAdminService** übergibt diese Werte zur Speicherung in der FDB der Methode **insertQuestion**.
Wenn der Administrator nun Lehrveranstaltungen eintragen möchte, wird der Service der FDB durch die Methode **selectLecturesAndFB_types** aufgefordert, alle Vorlesungen mit ihren Fragebogentypen anzuzeigen. Hier wird wieder kein Wert zurückgeliefert, da die Datenbank noch keine Vorlesungen enthält. Bei der Erstellung einer neuen Vorlesung werden alle Fragebogentypen (**fb_types**) über den Web Service der FDB mit der Methode **showFB_Types** geholt und zur Auswahl gestellt. Der Name der Vorlesung (**lecture**) und des ausgewählten Fragebogentyps (**fb_type**) werden dem Klienten des **FDBtoAdminService** übergeben, der die Methode **newLecture** aufruft. Sobald die Daten beim FDB-Modul angekommen und in der FDB gespeichert worden sind, wird der **FDBtoADBServiceClient** aufgerufen. Dieser übergibt der Methode **newLecture** des **FDBtoADBService** der ADB die ID und den Namen der gerade in die FDB eingefügten Vorlesung zur Speicherung in der ADB.
Bei der Benutzerverwaltung wird die Methode **showUsers** des ADB-Services aufgerufen, welche die vorhandenen Benutzer des Systems anzeigt. Wenn nun ein neuer Benutzer eingefügt werden soll, werden der Methode **insertUser** die neuen Daten mitgegeben, welche veranlasst, dass diese in der ADB gespeichert werden.

In der nächsten Abbildung wird die Reihenfolge der Web-Service-Aufrufe für den Fall dargestellt, dass ein Fragebogentyp geändert werden soll. Hier wird nur mit dem Web Service der FDB kommuniziert, d.h. es wird nur mit Daten aus der FDB gearbeitet.

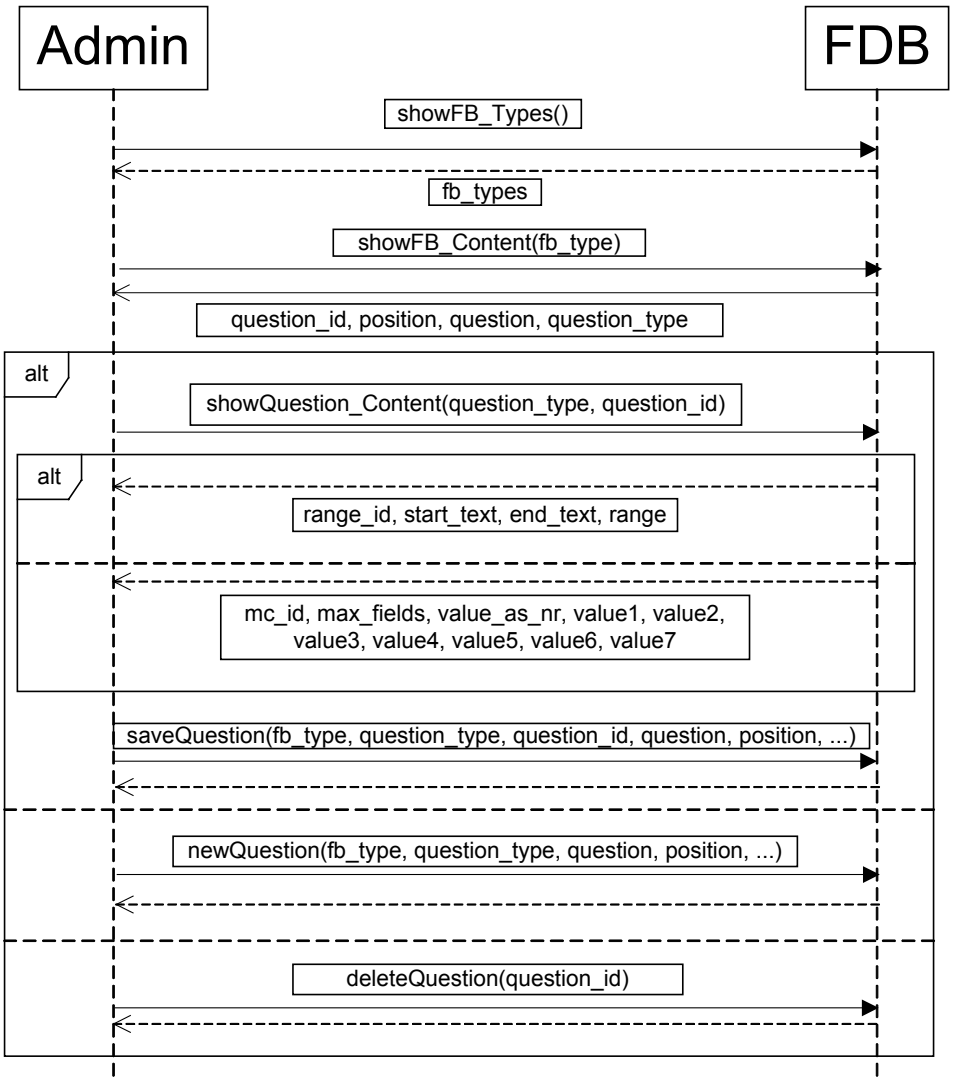

Abbildung 4-10 Kommunikation zwischen Administrationsbereich und FDB beim Erstellen eines Fragebogentyps

Als erstes ruft der **`FDBtoAdminClient`** die Methode **`showFB_Types`** auf und erhält als Rückgabewert einen Vektor (**`fb_types`**) mit allen in der FDB vorhandenen Fragebogentypen. In der Präsentation werden nun die Fragebogentypen aufgeführt. Wenn jetzt ein Fragebogen ausgewählt wird, muss sein Inhalt angezeigt werden. Dies geschieht über die Methode **`showFB_Content`**, die die ID (**`question_id`**), den Text (**`question`**), den Typ (**`question_type`**) und die Position (**`position`**) im Fragebogen aller Fragen des ausgewählten Fragebogens anzeigt.

Ist nun der Fragebogentyp mit seinen Fragen und Antwortmöglichkeiten im Browser dargestellt, hat der Benutzer mehrere Möglichkeiten. Er kann eine Frage ändern, hinzufügen oder löschen. Um eine Frage zu ändern, muss ihr gesamter Inhalt erst angezeigt werden. Dies geschieht durch den Aufruf der Methode **`showQuestionContent`**.

```
public Vector showQuestion_Content(String question_type, String
    question_id){
    Vector question_content;
    question_content = new Vector();
    FDBoperations op = new FDBoperations();
    question_content = op.selectQuestion_Content(f_type, question_id);
    return (question_content);
}
```

Diese Methode erhält vom Klienten den Typ und die ID der Frage, die sie an die Methode **selectQuestion_Content** der FDB weitergibt. Als Rückgabewert erhält sie einen Vektor, der die zur Frage dazugehörigen Antwortmöglichkeiten zurückgibt. Im Fall einer Reihenfrage sind dies die Werte **starttext**, **endtext**, **range** sowie die zugehörige ID (**range_id**). Bei einer Multiple-Choice-Frage erhält der Vektor die Werte **mc_id**, **max_fields**, **value_as_nr**, **value1**, **value2**, **value3**, **value4**, **value5**, **value6** und **value7**. Diese Rückgabewerte werden in einer dem Fragetyp angepassten Maske für den Administrator zum Editieren angezeigt. Nimmt dieser Änderungen vor, werden die Werte der Methode **saveQuestion** des **FDBtoAdminWebService** übergeben, die diese zur Speicherung an die FDB sendet.
Wird eine neue Frage erstellt, werden die Werte der neuen Frage an die Methode **newQuestion** gesendet, die veranlasst, dass diese in der FDB abgelegt werden.
Zum Löschen einer Frage erhält die Methode (**deleteQuestion**) ihre ID, worauf alle zugehörigen Daten der Frage aus der Datenbank gelöscht werden.

Die Vorgehensweise des Editierens einer Vorlesung wird in der folgenden Abbildung dargestellt.

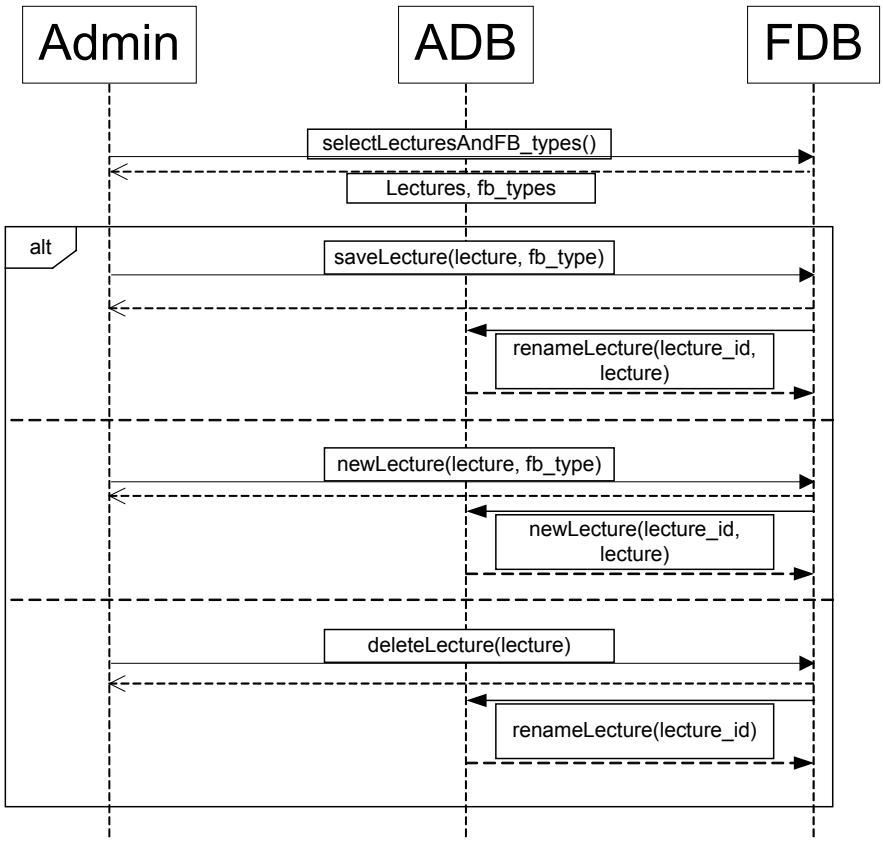

Abbildung 4-11 Kommunikation zwischen Administrationsbereich, ADB und FDB beim Editieren einer Vorlesung

Als erstes wird die bereits beschriebene Methode **`selectLecturesAndFB_types`** des **`FDBtoAdminService`** aufgerufen. Die Vorlesungen werden mit ihren dazugehörigen Fragebogentypen angezeigt. Nun können die Vorlesungen entweder geändert oder gelöscht werden. Es können aber auch neue Lehrveranstaltungen hinzugefügt werden, was im vorherigen Absatz schon beschrieben worden ist. Bei diesen Aktionen findet eine Kommunikation sowohl mit dem Web Service **`FDBtoAdminService`** als auch mit dem Web Service **`FDBtoADBService`** statt. Sobald eine Änderung bezüglich der Lehrveranstaltungen in der Tabelle **`lectures`** der FDB vorliegt, müssen diese Daten auch in der gleichnamigen Tabelle der ADB angepasst werden. Deshalb werden die jeweiligen Methoden des Web Services **`FDBtoADBService`** aufgerufen, welche die veränderten Daten empfängt und an die ADB weitergibt.

4.3.2.3Klassen der Datenbankmodule

Die Module der Datenbanken ADB und FDB enthalten die Klassen **ADB**, **ADBoperations**, **FDB** und **FDBoperations**. Die Klassen **ADB** und **FDB** stellen den direkten Kontakt zu den jeweiligen Datenbanken her. Hier wird eine Verbindung zur Datenbank aufgebaut und ein SQL-Statement ausgeführt. Diese Klassen bieten Methoden an, die die Datenbank öffnen, ihr Informationen entnehmen und den Datenbestand aktualisieren. Die Methode **openDatabase** stellt eine Verbindung zur Datenbank her. Der Ort des Treibers wird durch die globale Variable **TREIBER** geliefert.

```
final String TREIBER = "org.gjt.mm.mysql.Driver";
final String ODBC_QUELLE = "jdbc:mysql://localhost/ADB?user=root";

public void openDatabase() {
    try {
        Class.forName(TREIBER);
        con = DriverManager.getConnection(ODBC_QUELLE);
    }catch (ClassNotFoundException cnfe) {
        System.err.println(cnfe.getMessage());
    }catch (Exception e) {
        System.err.println(e.getMessage());
    }
}
```

Die Quelle der Datenbank wird durch die globale Variable **ODBC_QUELLE** genannt. Hier wird eine Verbindung zur ADB hergestellt.

Um Daten aus der Datenbank auszuwählen, wird folgende Methode verwendet:

```
public ResultSet selectData(String sql) {
    sqlStatement = sql;
    if (con != null) {
        try {
            Statement statement = con.createStatement();
            ResultSet rs = statement.executeQuery(sql);
            return rs;
        }catch (SQLException sqle){
            System.err.println(sqle.getMessage()); return null;
        }catch (Exception e) {
            System.err.println(e.getMessage());return null;
        }
    }
    else { return null;}
}
```

Aus einer Methode der Klasse **ADBoperations** wird ein SQL-Statement übergeben. Das Ergebnis wird in einem **ResultSet** gespeichert und der Methode der **ADB** zurückgeliefert. Ähnlich funktioniert die Methode **updateData**. Hier werden aber keine Informationen aus der Datenbank geholt, sondern Daten geändert.

```
public void updateData(String sql) {
    sqlUpdateStatement = sql;
    if (con != null) {
        try {
            Statement statement = con.createStatement();
            statement.executeUpdate(sql);
        }catch (SQLException sqle) {
            System.err.println(sqle.getMessage());
        }catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }
}
```

Hier wird nur das übergebene SQL-Statement ausgeführt, aber kein Wert zurückgeliefert. Die SQL-Statements werden in den Klassen **ADBoperations** und **FDBoperations** erstellt. Ein Beispiel für eine Methode dieser Klassen ist **renameLecture** aus der Klasse **ADBoperations**.

```
public void renameLecture(String lecture_id, String lecture){
    String sql="UPDATE lectures SET lecture='"+lecture+"' WHERE
                lecture_id='"+lecture_id+"'";
    db.updateData(sql);
}
```

Hier wird das SQL-Statement zum Ändern eines Eintrags in der Tabelle, die alle Lehrveranstaltungen enthält, erstellt. Anschließend wird es der Methode **updateData** der Klasse **ADB** übergeben und dort ausgeführt.
Auf den beiden Klassen **ADBoperations** und **FDBoperations** bauen die Web Services auf, die den SQL-Statements Daten liefern.

4.3.3 Zugriff auf Web Services aus dem Teilnehmer- und Auswerterbereich

Die Methoden, die dem Teilnehmer- und Auswerterbereich dienen sollen, sind noch nicht implementiert, werden aber in den folgenden Aktivitätsdiagrammen beschrieben.
Das erste Beispiel stellt die Situation dar, in der ein Student eine PIN anfordert.

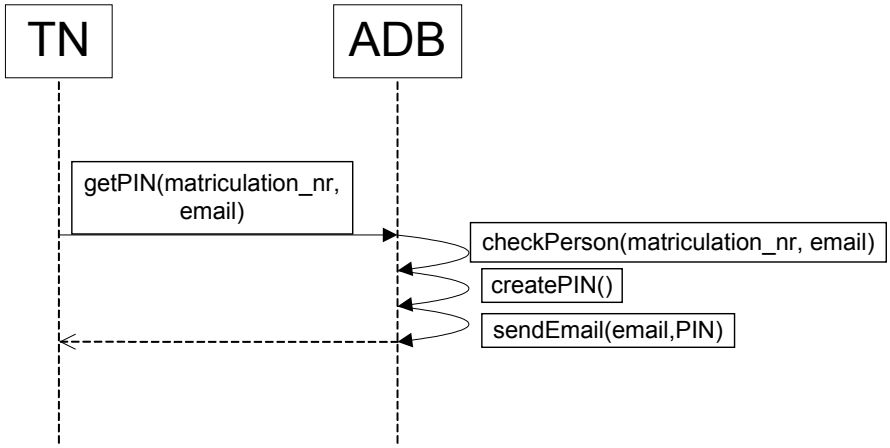

Abbildung 4-12 Kommunikation zwischen Teilnehmerbereich und ADB beim Anfordern einer PIN

In der Präsentationsschicht im Teilnehmerbereich wird eine Aktion gestartet, die eine Seite anzeigt, in der der Student seine Emailadresse und Immatrikulationsnummer einträgt und das Formular absendet. Der Service der ADB wird aufgerufen und der Methode **`getPIN`** diese Daten übergeben. In der ADB-Schnittstelle wird geprüft (**`checkPerson`**), ob diese Daten schon existieren. Ist dies der Fall, werden sie nicht gespeichert, ansonsten in die Datenbank aufgenommen. Es wird eine PIN per Zufall erstellt (**`createPIN`**) und der Methode **`sendEmail`** mit der Emailadresse des Studenten übergeben, welche eine Email mit dieser PIN verschickt.
Nun kann der Student mit der PIN aus der Email evaluieren.

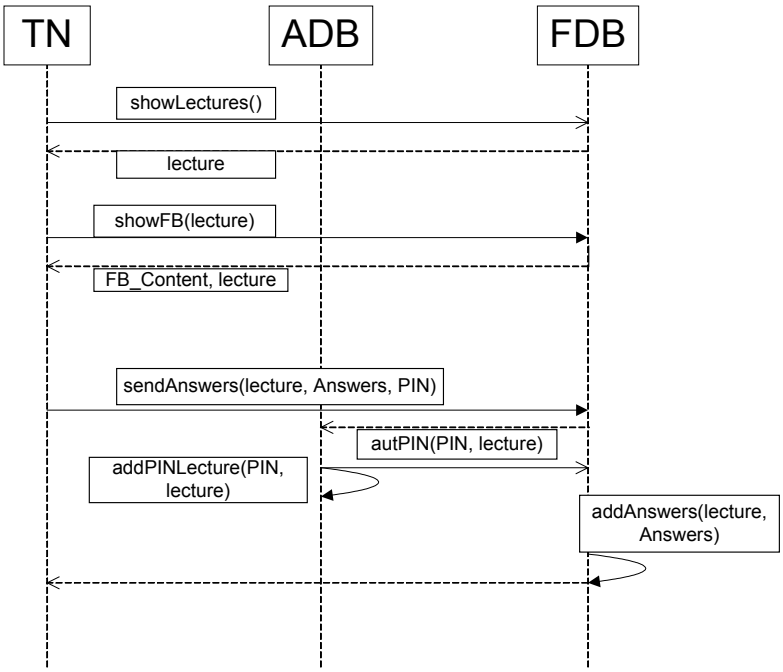

Abbildung 4-13 Kommunikation zwischen Teilnehmerbereich, ADB und FDB beim Evaluieren einer Vorlesung

Sollen alle Lehrveranstaltungen angezeigt werden, wird die Methode **`showLectures`** des Web Services der FDB aufgerufen. Diese liefert alle Vorlesungen zurück, die anschließend für den Teilnehmer der Evaluation in einer Auswahlmaske bereitgestellt werden. Der Student wählt eine zu evaluierende Vorlesung aus und die Teilnehmerschnittstelle spricht die Methode **`showFB`** mit dem Parameter der Vorlesung an. Als Rückgabe wird der Fragebogentyp mit dem Namen der Vorlesung zurückgeliefert. Diese Daten werden erneut als Maske dargestellt, in der der Student evaluieren kann. Am Ende des Evaluationsdokuments muss er noch seine PIN zur Authentifikation eingeben. Diese Ergebnisse und die PINs werden der FDB-Service-Methode **`sendAnswers`** übergeben. Das FDB-Modul fragt bei der ADB an, ob diese Vorlesung bereits mit dieser PIN evaluiert worden ist (**`autPIN`**). Wenn das nicht der Fall ist, wird die PIN mit der Vorlesung in der Tabelle **`pin_lecture`** eingetragen und die Methode **`addAnswers`** aufgerufen, die die Ergebnisse der Evaluation in der FDB speichert. Nachdem sie ausgeführt worden ist, bekommt der Student eine Bestätigung, dass seine Evaluierungsdaten erfolgreich gespeichert worden sind.

Nun fehlt noch der Bereich des Auswerters. Hier wird ebenfalls mit beiden Web Services beider Datenbanken kommuniziert.

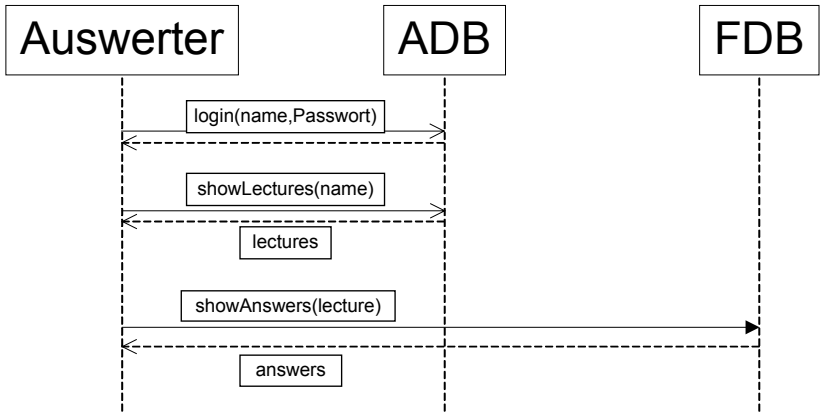

Abbildung 4-14 Kommunikation zwischen Auswerterbereich, ADB und FDB

Der Login erfolgt genauso wie beim Administrationsbereich. Zur Anzeige der Vorlesung, die der Auswerter ansehen darf, wird sein Name dem Web Service der ADB übergeben (**`showLectures`**). Als Rückgabe erscheinen die erlaubten Vorlesungen, die in einer Maske dargestellt werden. Der Auswerter wählt nun die Vorlesung aus, von der er die Ergebnisse der Evaluation sehen möchte. Der Methode **`showAnswers`** des Web Services der FDB wird diese Vorlesung mitgegeben. Die Antworten (**`answers`**) werden aus der Fragebogendatenbank herausgelesen und dem Auswerterbereich übergeben, der diese für den Auswerter darstellt.

5 AUSBLICK

In dieser Bachelorarbeit wurde ein Konzept eines Evaluationssystems auf der Grundlage von Web Services entworfen. Dieses Online-Evaluationssystem besteht aus den fünf Modulen ADB (Autorisierungsdatenbank), FDB (Fragebogendatenbank), Administrations-, Auswertungs- und Teilnehmerbereich. Die beiden Datenbankmodule ADB und FDB bieten Web Services an, auf die die anderen Module mit ihren Klienten zugreifen. In dieser Arbeit wurden die Datenbankmodule und der Administrationsbereich bereits implementiert.
Das Evaluationssystem hätte auch als einfache Web Applikation erstellt werden können, aber der Einsatz von Web Services hat den Vorteil, dass die unterschiedlichen Module voneinander unabhängig sind und auf verschiedenen Servern laufen können. Beispielsweise könnten die beiden Datenbankenmodule auf einem anderen Server als die restlichen Bestandteile laufen. So wären diese Daten noch unzugänglicher, besser vor unerwünschten Zugriffen geschützt und das System ausfallsicherer.

Web Services sind eine gute Alternative zu den bisher existierenden Web Applikationen. Sie werden den Umgang mit dem Internet revolutionieren. Es werden nicht mehr nur Daten zur Präsentation angeboten, sondern auch Metadaten bereitgestellt werden, die den Umgang mit Informationen wesentlich erleichtern. Es müssen nun nicht mehr per Hand Daten gesammelt und dann in Anwendungssysteme eingegeben werden, sondern sie können gleich durch den Aufruf eines Web Service genutzt werden. Dies unterstützt das Personal, reduziert den Zeitaufwand der Datenbeschaffung um einiges und spart dadurch Kosten.
Der Einsatz von Web Services hat nicht nur Vorteile, sondern kann auch Risiken mit sich bringen. Wenn diese beispielsweise im Internet angeboten werden, hat jeder darauf Zugriff und kann an für ihn nicht bestimmte Daten gelangen kann. Um dies zu unterbinden, wurden bereits Sicherheitsprotokolle für Web Services entworfen. Es gibt die Spezifikation Web Services Security (WS Security), die beschreibt, wie ein SOAP-Protokoll so erweitert werden kann, dass verschlüsselte oder digital signierte Nachrichten verwendet werden können. [3]

Die Umstellung der Evaluation vom Papier zum Online-Evaluationssystem ermöglicht erst eine effektive Durchführung der Beurteilung von Lehrveranstaltungen. Sie macht es nicht nur für Studenten bequemer die Vorlesungen zu bewerten, sondern bringt auch den Auswertern einen erheblichen Vorteil, da eine Auswertung per Hand sehr zeitaufwändig und kostspielig ist. Mit dieser Methode werden die Ergebnisse automatisch berechnet und angezeigt. Auch der Vergleich mit früheren Evaluationsergebnissen ist wesentlich leichter, da diese in digitaler Form vorliegen und sich somit Abweichungen sofort durch automatische Berechnungen anzeigen lassen.

Denkbar ist übrigens auch, ein solches System, entsprechend angepasst, auch in anderen Bereichen anzuwenden, wie etwa bei Kundenbefragungen durch Unternehmen oder Mitarbeiterbefragungen in Behörden und Unternehmen, die ein solches Führungsinstrument einsetzen.

6 LITERATUR

[1] **Eberhart, Andreas / Fischer, Stefan (2003):** Web Services – Grundlagen und praktische Umsetzung mit J2EE und .NET, Carl Hanser Verlag München Wien, 2003

[2] **Fröschle (Hrsg.), Hans-Peter (2003):** HDM 234 – Praxis der Wirschaftsinformatik Web Services, 40.Jg., Heidelberg, dpunkt.verlag GmbH, Dezember 2003

[3] **Wang (Hrsg.), Dapeng / Bayer, Thomas / Frotscher, Thilo / Teufel, Marc (2004):** Java Web Services mit Apache Axis, Javamagazin, Software & Support Verlag GmbH, 2004

[4] **MySQL AB (2004):** MySQL, **`http://www.mysql.com`**, Einsichtnahme am 24.08.2004

[5] **Studiendekanat Medizinische Fakultät Universität zu Lübeck (2004):** Studiendekanat – Evaluation, **`http://www.medizin.uni-luebeck.de/evaluation/index.php?content=faq.htm`**, Einsichtnahme am 24.08.2004

[6] **Prof. Dr. May, Wolfgang (2003):** Semistructured Data & XML, **`http://www.dbis.informatik.uni-goettingen.de/Teaching/SSD-SS03/`**, Einsichtnahme am 24.08.2004

[7] **Robitzsch, H. (Oktober 2001):** Leitfaden zum Informatik-Praktikum, 14. Auflage, Universität Göttingen

[8] **The Apache Software Foundation (2004):** Web Services – Axis, **`http://ws.apache.org/axis/`**, Einsichtnahme am 24.08.2004

[9] **W3C (2004):** World Wide Web Consortium, **`http://www.w3.org/`**, Einsichtnahme am 24.08.2004

[10] **W3C (2004):** Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language W3C Working Draft 3 August 2004, **`http://www.w3.org/TR/2004/WD-wsdl20-20040803/`**, Einsichtnahme am 24.08.2004

[11] **Sun Microsystems, Inc. (2004):** Java Technologie, **`http://java.sun.com`**, Einsichtnahme am 24.08.2004

[12] **The Apache Software Foundation (2004):** Apache Jakarta Tomcat, **`http://jakarta.apache.org/tomcat/`**, Einsichtnahme am 24.08.2004

[13] **The Apache Software Foundation (2004):** The Apache Jakarta Project, **`http://jakarta.apache.org`**, Einsichtnahme am 24.08.2004