Diplomarbeit

Anschluss eines Rich-Clients an eine Web-Applikation mit Hilfe einer Service-Architektur

Universität Hamburg
Departement Informatik

Autor:
Claus Andersen
In de Krümm 36b
21147 Hamburg
Matrikelnummer: 5212123
E-Mail: clausan@freenet.de

Erstbetreuer: Dr. Wolf-Gideon Bleek
Zweitbetreuer: Prof. Horst Oberquelle

Erklärung

Ich, Claus Andersen, bestätige hiermit, dass ich die vorliegende Diplomarbeit allein und selbstständig angefertigt habe.

Hamburg, 11.09.2006, ...
(Claus Andersen)

Danksagung

Ich danke allen, die mich bei der Erstellung dieser Arbeit und während meines gesamten Studiums unterstützt habe. Dazu gehören meine Eltern, meine Freunde, das LAssi-Projekt für die Unterstützung durch ein Laptop, sowie mein Erstbetreuer für diese Diplomarbeit, Dr. Wolf-Gideon Bleek. Dank auch an die anderen Mitglieder des CommSy-Diplomarbeiterprojektes für manchen hilfreichen Hinweis.

Inhaltsverzeichnis

Kapitel 1 Einleitung

Dieses erste Kapitel gibt einen Überblick über diese Diplomarbeit und stellt die Problemstellung und den Kontext dar, in dem sich diese Arbeit bewegt. Dazu wird kurz das geplante Vorgehen und der Aufbau der Arbeit beschrieben.

1.1 Überblick

Diese Diplomarbeit beschäftigt sich mit der Frage, ob und wie externe Client-Systeme an das CommSy (bzw. JCommSy), angebunden werden können. Es gibt grundlegend zwei verschiedene Typen von Client-Systemen, Thin-Clients und Rich-Clients. Thin-Clients beziehen ihre Funktionalität vom Server, mit dem sie zusammenarbeiten. Im Falle von CommSy sind dies die Webseiten, über die mit einem Internet-Browser auf das System zugegriffen werden kann. Ein Rich-Client ist ein System, in dem neben der Ein- und Ausgabe auch die Verarbeitung von Daten realisiert ist. LAssi ist ein Rich-Client-System.

Es ist zu klären, auf welcher Ebene eine Anbindung sinnvoll erscheint und wie dies technisch zu realisieren ist. Ist diese Frage erörtert, soll eine Rich-Client-Schnittstelle für die Lernumgebung LAssi exemplarisch implementiert werden.

1.2 Kontext

„CommSy“ ist ein am Departement Informatik der Universität Hamburg entwickeltes Open-Source-Projekt. Es ist ein web-basiertes Community-System, das es den Nutzern ermöglicht, miteinander zu kommunizieren und Projekte zu koordinieren. Materialien (Dateien) und Nachrichten können so gespeichert, dass andere Nutzer diese lesen und ggf. verändern können. CommSy bietet hierfür Räume an, auf die jeweils ausgewählte Nutzer Zugriff haben. In diesen Räumen stellt das System die Rubriken „Ankündigungen“, „Termine“, „Materialien“, „Diskussionen“, „Personen“ und „Gruppen“ zur Verfügung. Die Rubrik „Materialien“ hebt sich von den anderen ab, da in dieser beliebige Dateien abgespeichert werden können, die dann für alle Nutzer des CommSy-Raumes zugreifbar sind. Alle Daten, außer Dateien und Bildern, die im CommSy enthalten sind, werden in einer Datenbank abgelegt.

Das System ist in PHP geschrieben und läuft auf einem Apache-Server. Mit Hilfe einer Web-Oberfläche können Nutzer mit dem System interagieren. Zur Zeit wird das CommSy schrittweise nach Java migriert (JCommSy). Dies geschieht vertikal, indem die einzelnen Rubriken nacheinander in Java überführt werden. Als Server fungiert für die Java-Version ein Apache Tomcat. Bisher sind die Rubriken „Termine“ sowie „Ankündigungen“ in Java umgesetzt.

„LAssi“ ist ein Rich-Client-System zur Unterstützung von Lernen. Dieses System wird im Projekt „Reinventing Education - Werkzeuge für das Lernen“, einer Kooperation von IBM und der Universität Hamburg entwickelt und basiert auf dem Eclipse-RCP-Framework. LAssi baut grundlegend auf der Verwaltung und Manipulation von Karteikarten auf. Diese Karteikarten enthalten Texte, Grafiken und Links und können vom Nutzer frei erstellt, mit Inhalt gefüllt und auf sogenannten Desktops gespeicherte Karteikarten angeordnet und strukturiert werden. Hierzu können Karten zusammengefasst oder in mehrere einzelne Karten geteilt werden. Außerdem gibt es Sortierkästen, in die die Karten einsortiert werden können. Diese Sortierkästen lassen sich ebenfalls auf dem Desktops anordnen. Zudem besteht die Möglichkeit, neue Karten anzulegen.

Der erste Entwurf von LAssi sah Gruppenfähigkeiten vor. Diese Ideen umfassten Funktionen wie ein gemeinsamer Desktop, den mehrere Personen an verschiedenen Rechner gleichzeitig nutzen können, eine Chat-Funktion und eine Gruppenunterstützung, also die Möglichkeit, neue Gruppen anzulegen und dann auch den Online-Status der anderen Gruppenmitglieder anzeigen zu können.

Da CommSy gut geeignet ist, Gruppenarbeit zu unterstützen, bietet sich eine Verbindung zwischen diesen beiden System an. Wie die technische Schnittstelle auf der CommSy-Seite aussehen könnte, soll in dieser Diplomarbeit geprüft und teilweise umgesetzt werden.

1.3 Problemstellung

Für LAssi wäre es wünschenswert, wenn auch Gruppen an verschiedenen Rechnern die Software für gemeinsames Lernen nutzen könnten, zum Beispiel indem Karten auf einem gemeinsamen Desktop angezeigt, bearbeitet und ausgetauscht werden können. Um dies zu realisieren, bietet sich eine Anbindung an das CommSy an. Da das CommSy aber ein geschlossenes System ist, das nur über ein Web-Interface in einem Browser verwendet werden kann, sind folgende technische Probleme zu lösen:

1. Problem

Die von CommSy zur Verfügung gestellten Dienste (Services) sind im derzeitigen System nach Außen nicht sichtbar. Es muss die Frage beantwortet werden, wie diese Funktionalitäten für andere Clients sichtbar und benutzbar gemacht werden.

2. Problem

Es muss eine Möglichkeit gefunden werden, die Benutzer aus anderen Clients am JCommSy anzumelden. Hier ist auch das Problem zu lösen, wie der Anmeldestatus mit den Clients synchronisiert werden kann.

3. Problem

Es muss geklärt werden, wie die Daten, die ein Client über das CommSy anderen Clients zur Verfügung stellt, abgespeichert synchronisiert und wieder abgerufen werden können. Dies ist besonders im Hinblick auf LAssi wichtig, da die Benutzung von verschiedenen Karten auf einem gemeinsamen Desktop unter Umständen ständige Synchronisation erfordert.

1.4 Vorgehen

Die Fragen, die in dieser Arbeit geklärt werden sollen, werden schrittweise geklärt. Zunächst wird erörtert werden, welche softwaretechnischen Konzepte für eine Web Service-Anbindung mit den gegebenen technischen Voraussetzungen und auch später für die Architektur des Web Service-Systems sinnvoll eingesetzt werden können. Dazu werden die bereits im CommSy und JCommSy verwendeten Konzepte und Architekturstile betrachtet, um dann an diesen anschließen zu können. Zudem wird diskutiert, ob, und wenn ja wie, die JCommSy-Architektur angepasst werden muss, um Web Services anzubinden.

1.5 Aufbau der Arbeit

Zu Beginn soll die Architektur des CommSy-Systems und die neue Architektur des JCommSy dargestellt und diskutiert werden, mit besonderem Blick auf die schon vorhandene Service-Architektur des JCommSy. Zudem wird das LAssi-System vorgestellt, das später als Demonstration an das JCommSy angeschlossen werden soll.

Anschließend werden die Technologien vorgestellt, die bei der Realisierung der Rich-Client-Anbindung mit Hilfe von Web Services benötigt werden (also SOAP, XML-RPC, WSDL).

Nach diesem einleitenden Teil werden die besprochenen Technologien auf Vor- und Nachteile im Hinblick auf eine Nutzung bei der Lösung der beschriebenen Probleme überprüft. Das Ergebnis dieser Überprüfung soll bei der Entscheidung helfen, welche Technologien bei der Anbindung von beliebigen Client-Systemen an das JCommSy benutzt werden sollen.

Ist die Entscheidung gefallen, wird ein Entwurf erarbeitet, wie das JCommSy-System durch die neuen Funktionalitäten erweitert werden kann. Dieser Entwurf soll schon berücksichtigen, dass als erste Beispielanwendung das LAssi-System an das JCommSy angeschlossen werden soll. Dies schließt auch die Entscheidung mit ein, welche Rubriken des JCommSy für LAssi benötigt werden.

Ist ein erster Entwurf vorhanden, wird dieser mit den vorher festgelegten Technologien prototypisch implementiert. Dies schließt sowohl die Schnittstelle nach Außen als auch eventuell noch nicht vorhandene, aber benötigte Rubriken im JCommSy ein.

Kapitel 2 Kontext

In diesem Kapitel werden mit den Software-Systemen CommSy und LAssi der Kontext vorgestellt, in dem sich diese Arbeit bewegt. Zunächst wird das CommSy, das im Laufe dieser Arbeit um eine Web Service-Schnittstelle erweitert werden soll, vorgestellt. Im zweiten Teil des Kapitels wird die Lernsoftware LAssi, die exemplarisch an diese neue Web Service-Schnittstelle des CommSy angeschlossen werden soll, vorgestellt.

2.1 CommSy

2.1.1 Beschreibung des Systems

Die Entwicklung des CommSy wurde 1999 am Fachbereich Informatik der Universität Hamburg initiiert und wird seitdem stetig weitergeführt. Seit April 2003 geschieht dies als Open-Source-Projekt. Das System ist ein Community-System, das es Benutzern ermöglicht, über das Internet miteinander zu kommunizieren und Daten auszutauschen. Die Benutzeroberfläche des CommSy wird mit Hilfe eines Web Browser dargestellt. Die grundlegende Unterteilung sind die CommSy-Räume. Diese können eingerichtet und mit einem Namen versehen werden. Diese Räume sind über die Einstiegsseite des CommSy sichtbar (siehe Abbildung 2.1). Jeder Raum ist entweder offen, also für jeden CommSy-Nutzer zugänglich, oder geschlossen und damit nur einer bestimmten Benutzergruppe zugänglich. Die Entscheidung darüber, wer welchen Raum betreten darf, wird von den Moderatoren der jeweiligen Räume selbst getroffen.

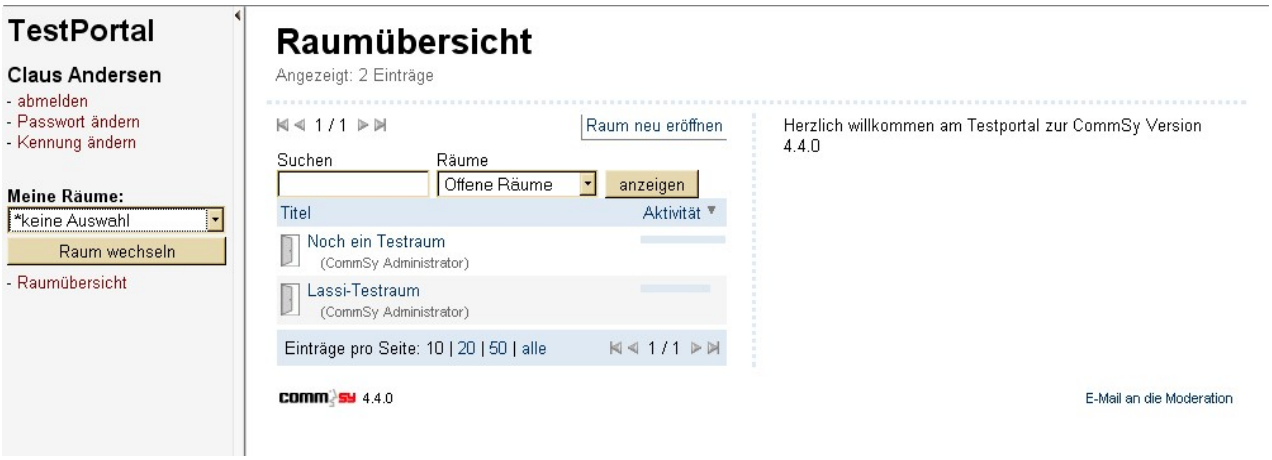

Abbildung 2.1 Die CommSy-Einstiegseite

Die Kommunikation zwischen Nutzern erfolgt innerhalb der Räume in verschiedenen Rubriken. Welche Rubriken ein Raum anbietet, kann der Moderator des Raumes frei einstellen. Nach dem Anmelden und dem Aufrufen eines Raumes erscheint die Home-Seite eines Raumes (siehe Abbildung 2.2).

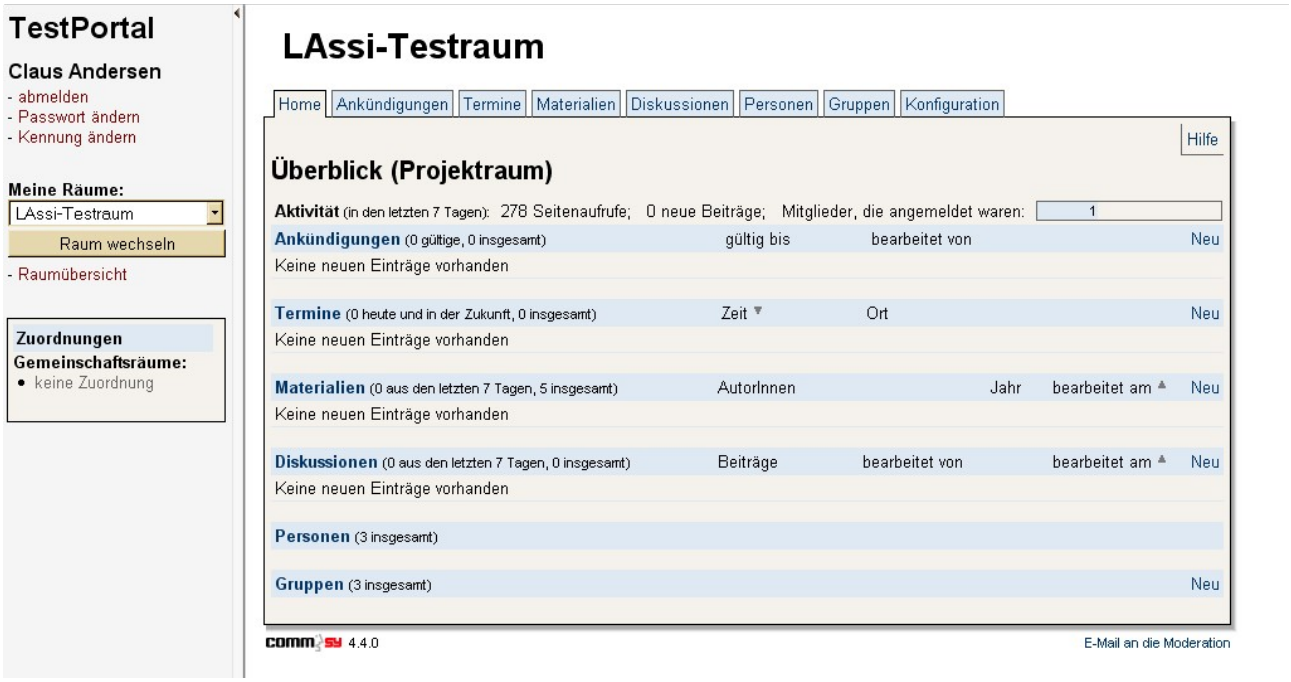

Abbildung 2.2 Home-Seite eines CommSy-Raums

Auf dieser Home-Seite sind die Rubriken sichtbar, die der aktuelle Raum anbietet. Durch Klick auf die Menuleiste kann zwischen den einzelnen Rubriken gewechselt werden. Das System stellt Rubriken mit folgender Funktionalität zur Verfügung. Jeder Nutzer kann in jeder Rubrik neue Elemente anlegen:

Ankündigungen:

In der Rubrik „Ankündigungen“ können Neuigkeiten gespeichert werden, die alle Raummitglieder betreffen.

Termine:

Die Rubrik „Termine“ bietet die Möglichkeit, Termine festzulegen. Hierbei werden zu jedem Termin Ort sowie Beginn- und Endzeit abgespeichert.

Materialien:

In der Rubrik „Materialien“ können Daten vom Nutzer abgespeichert werden und mit einem Kommentar versehen werden. Alle Mitglieder des zugehörigen Projekt-Raums haben die Möglichkeit, auf die abgelegten Materialien zuzugreifen.
Diskussionen:

Die Rubrik „Diskussionen“ bietet die Möglichkeit, wie in einem Internet-Forum Diskussionen mit Hilfe von Textnachrichten zu führen.

Personen:

In der Rubrik „Personen“ sind alle Mitglieder des aktuellen Projektraums aufgeführt. Das System bietet auch die Möglichkeit, E-Mails an einzelne oder mehrere Personen zu versenden.

Gruppen:

Die Mitglieder eines Projektraums können zu Gruppen zusammengefasst werden. Diese werden in der Rubrik „Gruppen“ angezeigt und können vom Moderator des Raums editiert werden. Es gibt die Möglichkeit, eine E-Mail an einzelne Gruppen zu schicken, außerdem kann in den Rubriken „Materialien“ für jedes Material festgelegt werden, für welche Gruppe dieses Material relevant ist.

Chat:

Die Rubrik „Chat“ bietet eine Chatfunktion für die Mitglieder des Projektraums an.

Alle diese Rubriken sind optional und können vom Moderator des Projektraums einzeln aktiviert bzw. deaktiviert werden. Außerdem lässt sich einstellen, ob neue Beiträge in bestimmten Rubriken direkt auf des „Home“ Seite des Projektraumes angezeigt werden.

Eine Kernfunktionalität des CommSy ist, dass Objekte miteinander verknüpft werden können. Zu Gruppen lassen sich Materialien, Ankündigungen, Diskussionen und Termine zuordnen. Die zugeordneten Objekte werden dann in der Ansicht der jeweiligen Gruppe angezeigt und es besteht die Möglichkeit, diese durch einen Link direkt zu erreichen.

2.1.2 Funktionalität und Architektur

CommSy wurde ursprünglich in PHP geschrieben. Die im Einsatz befindlichen CommSy-Versionen (aktuell zu diesem Zeitpunkt ist Version 4.4.0) sind eine Weiterentwicklung der Ursprungsversion und liegen ebenfalls in PHP vor.

CommSy läuft auf einem Apache-Server, daher kann auf das System über HTTP und Port 80 zugegriffen werden. Der Zugriff erfolgt mit Hilfe eines Web Browser (beispielsweise MS Internet Explorer oder Mozilla Firefox). Beim ersten Aufruf eines CommSy-Systems erhält man eine Sicht auf das Portal des Servers (siehe Abbildung 2.1). Auf dieser Seite kann man sich mit seiner CommSy-Kennung anmelden und erhält Zugriff auf die Gemeinschaftsräume, die für jeden CommSy-Nutzer freigegeben sind und auf die Räume, in denen man angemeldet ist. Es wird beim Login ein Cookie gesetzt, so dass der Benutzer angemeldet bleibt, bis er sich wieder abmeldet. Akzeptiert der Nutzer keine Cookies, werden die URLs um eine SessionID ergänzt.

Die Informationen über die registrierten Nutzer, die Räume, die auf einem CommSy-Server liegen und die Inhalte der einzelnen Rubriken jedes Raumes liegen in einer MySQL-Datenbank, mit der der CommSy-Server kommuniziert. Zu den in der Rubrik Materialien abgespeicherten Dateien sind Verweise in der Datenbank abgelegt, die auf den tatsächlichen Speicherort hinweisen. Jeder Eintrag (hier am Beispiel einer Ankündigung) wird mit folgenden Attributen in der Datenbank abgelegt:
Einer „Item_ID“, die die eindeutige Identifizierung jedes Eintrages in jeder Rubrik und jedes Raumes ermöglicht, eine „Context_ID“, die die ID des Raumes enthält, zu dem diese Ankündigung gehört, eine „Creator_ID“, eine „Modifier_ID“ und eine „Deleter_ID“, die die Nummer des

Benutzers enthält, der die Ankündigung erzeugt, modifiziert bzw. gelöscht hat. Des weiteren gibt es ein „Creation_Date“, ein „Modification_Date“ sowie ein „Deletion_Date“, die enthalten, wann eine Ankündigung erzeugt, modifiziert oder gelöscht wurde. Außerdem hat eine Ankündigung einen Titel („Title“) und eine Beschreibung („Description“).

Das PHP-CommSy ist als Model-View-Control-Architektur aufgebaut [siehe BD04, Seite 272ff]. Nach diesem Architekturstil besteht ein Softwaresystem aus drei Komponenten. Dem Model, das alle Daten enthält, die von der Anwendung dargestellt werden können. Dabei wissen die Daten nichts darüber, wie oder wie oft sie dargestellt werden. Die zweite Komponente ist die View-Komponente. Diese enthält nur die Informationen, wie die Daten der Model-Komponente dargestellt werden und muss bei Änderung der Materialien aktualisiert werden. Die dritte Komponente ist die Control-Komponente. Diese verwaltet die Views und nimmt Benutzereingaben entgegen und wertet diese aus. Im PHP-CommSy existieren form-Klassen, die die Rolle der View-Komponente übernehmen und Daten , die in der Datenbank abgelegt sind, anzeigen. Diese Daten repräsentieren die Model-Komponente. Die Control-Komponente wird von Klassen erfüllt, die nach einer Benutzereingabe die Model-Daten verändern.

2.1.3 JCommSy

Im Jahr 2005 wurde im Rahmen eines Projektes damit begonnen, das PHP-CommSy in die Programmiersprache Java zu migrieren. Diese Migration geschieht vertikal, die einzelnen Rubriken werden nacheinander in Java geschrieben. Dies ist möglich, da die einzelnen Rubriken unabhängig voneinander sind. Für jede Rubrik existiert ein eigener Service, der die Schnittstelle bildet zwischen dem zugehörigen Servlet und der Datenbank. Die Service-Klassen enthalten Methoden, um Objekte der Rubrik (zum Beispiel Termine oder Materialien) aus der Datenbank zu holen, abhängig von Parametern wie User und Raum, in dem sich der User gerade befindet. Die Datenbankanbindung wird mit Hibernate realisiert.

Die Software-Architektur folgt dem Werkzeug- und Material-Ansatz (siehe [Z^{+}98]). Dabei fungieren die JCommSy-Items, alle Objekte, die von der Klasse Item erben, als Materialien. Dies sind Gruppen, Materialien, User, Räume, Termine, Ankündigungen und Annotationen. Diese Materialien werden durch die Werkzeuge, in Fall des JCommSy die Services, bearbeitet.

Als Server dient in der Java-Version ein Apache Tomcat in der Version 5.0. Um den Aufbau der Software zu veranschaulichen wird in Abbildung 2.3 die Struktur der Packages des JCommSy dargestellt, die für diese Arbeit am relevantesten sind.

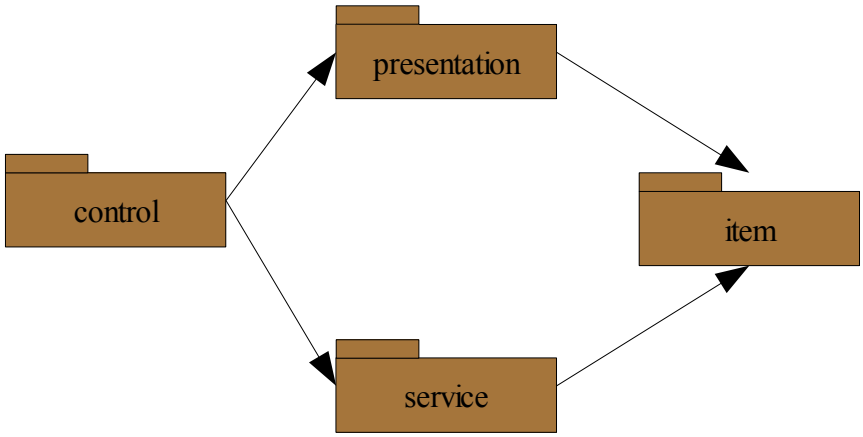

Abb. 2.3 Package-Struktur JCommSy

Im Folgenden werden die Packages, aus denen das JCommSy besteht, kurz vorgestellt. Besondere Aufmerksamkeit gilt dem für diese Arbeit wichtige `service`-Package.

1. Control

Das `Control`-Package enthält die Servlets und andere Klassen, die die Requests verarbeiten, die von den JSPs (Java Server Pages) an das System weitergegeben werden. Diese Servlets verwenden zur Anzeige der JCommSy-Materialien die im `Presentation`-Package vorhandenen Java-Bean-Klassen. Diese werden mit Hilfe der Services mit den Inhalten der JCommSy-Materialien gefüllt, deren Klassen im `item`-Package liegen.

2. Presentation

Das `Presentation`-Package enthält alle für das System benötigten Java-Beans.

3. Domainvalue

Das `Domainvalue`-Package enthält alle benötigten Fachwerte, zum Beispiel den Fachwert `ItemType`, der alle möglichen Objekt-Typen enthält, wie zum Beispiel Appointment (Termin), Announcement (Ankündigung) usw.

4. Item

Das `Item`-Package enthält die Klassen für die Objekte, die im System verarbeitet werden, also Appointment, Announcement usw. Diese Klassen erben alle von einem Interface `Item`.

5. Migration

Das `Migration`-Package enthält Klassen, die dafür sorgen, dass die PHP- und die Java-Version von CommSy gemeinsam betrieben werden können (also einige Rubriken in PHP und einige in Java).

6. Util

Das `Util`-Package Hilfsklassen, die der Handhabung der `Item`-Objekte im System dienen. Zudem enthält das Package Klassen zur Fehlerbehandlung.

7. Service

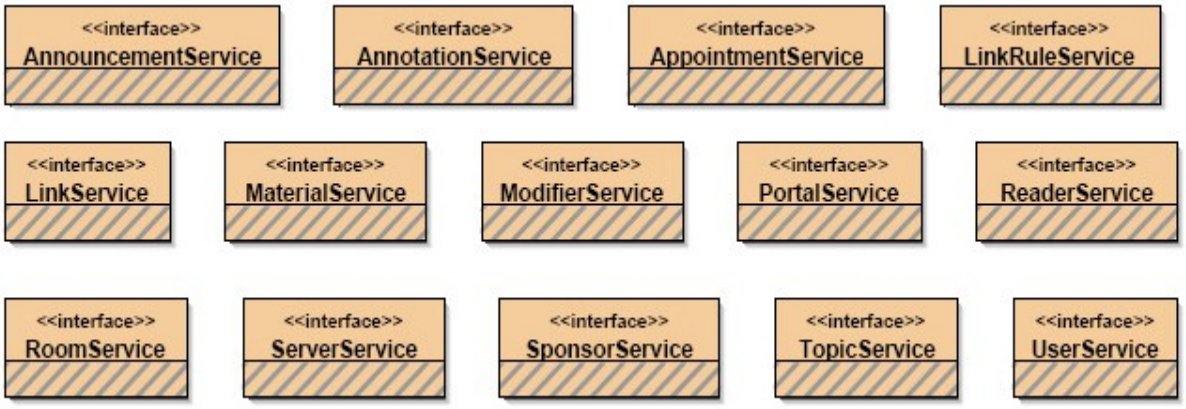

Abbildung 2.4 Klassendiagramm des `Service`-Packages

Das `Service`-Package enthält, neben weiteren Klassen, für jede CommSy-Rubrik ein Interface für einen Service, der die Funktionalität der Rubrik zur Verfügung stellt (siehe Abbildung 2.4). Die eigentliche Implementierung der Services liegen im Subpackage `Impl`. Die Services bilden die Schnittstelle zwischen den Servlets, die die Anfragen vom Web-Interface entgegennehmen, und der Datenbank. Jeder Service stellt mehrere Methoden zur Verfügung, um mit Objekten vom jeweils benötigten Typ umzugehen. Diese Methoden sind zum einen Get-Methoden, die ein Objekt mit Hilfe von Hibernate aus der Datenbank holen und zum anderen Add- bzw. Update-Methoden, mit denen Objekte neu in der Datenbank gespeichert werden bzw. vorhandene Objekte verändert werden können. Am Beispiel der Implementierung des Appointment-Service wird die Funktionalität einiger Methoden näher erklärt:

Die Klasse `AppointmentServiceImpl.java` bietet folgende Methoden an:

- `Appointment getItem(ItemID id)`

  Diese Methode wird verwendet, um ein Objekt mit einer bestimmten ID aus der Datenbank zu holen und als Appointment zurückgegeben. Jedes Objekt im System hat eine eindeutige ID vom Typ `ItemID`. Dies ist ein selbst definierter Fachwert, die Implementierung findet sich im Package `domainvalue`.

- `public void updateItem(Appointment item, CallContext cc)`

  Ein bereits in der Datenbank vorhandenes Appointment-Objekt wurde verändert und wird mit dieser Methode neu abgespeichert. Als weiterer Parameter wird ein CallContext übergeben. Dies ist ebenfalls ein selbstdefinierter Fachwert und findet sich im Package `domainvalue`. Der CallContext modelliert die Umgebung, in der der Aufruf der Methode erfolgt ist und besteht aus zwei Objekten vom Typ ItemID. Eine ItemID stellt die

Raumnummer dar, um zu ermitteln, in welchem Raum sich der Benutzer gerade befindet. Die zweite ItemID enthält die ID des Users. Es wird geprüft, ob der CallContext gültig ist (also ob der Benutzer das Recht hat, aus seinem aktuellen Raum dieses Appointment-Objekt zu verändern). Ist dies der Fall, wird das Appointment mit den neuen Daten in der Datenbank abgelegt. Als Modifier_ID wird die UserID aus dem CallContext gesetzt.

- `public void deleteItem(ItemID id, CallContext cc)`

  Soll ein in der Datenbank vorhandenes Appointment-Objekt mit einer bestimmten ID gelöscht werden, wird diese Methode verwendet. Hier wird zunächst geprüft, ob der CallContext gültig ist, und anschließend das gewünschte Objekt gelöscht. Das Objekt wird in der Datenbank als gelöscht markiert und bekommt die UserID aus dem CallContext als Deleter_ID gesetzt.

- `public void addItem(Appointment item, CallContext cc)`

  Es ist ein neues Appointment-Objekt erzeugt worden und dies soll in der Datenbank gespeichert werden. Dies geschieht mit dieser Methode. Das Objekt wird in der Datenbank abgelegt und bekommt die UserID aus dem CallContext als Creator_ID gesetzt.

Zu den beschriebenen Methoden kommen noch weitere hinzu, um mehrere Objekte, die bestimmten Kriterien entsprechen, in einer Liste zurückzugeben,.

Damit die Services durch die Servlets verwendet werden können, müssen diese dem Servlet bekannt gemacht werden. Dies wird im Unterpackage `registry` in der Klasse `ServiceRegistry.java` gemacht. In dieser Klasse sind alle Services eingetragen, die zur Verfügung stehen. Das Servlet kann auf die ServiceRegistry zugreifen und sich die Instanz des gewünschten Services geben lassen, so dass dessen Methoden verwendet werden können.

Die Oberfläche und die Funktionalität des Web-Interfaces werden in der Java-Version durch Java Server Pages (JSPs) realisiert. Diese greifen auf Servlets zu, die ihrerseits die beschriebenen Services benutzen. Abbildung 2.5 zeigt diese Struktur.

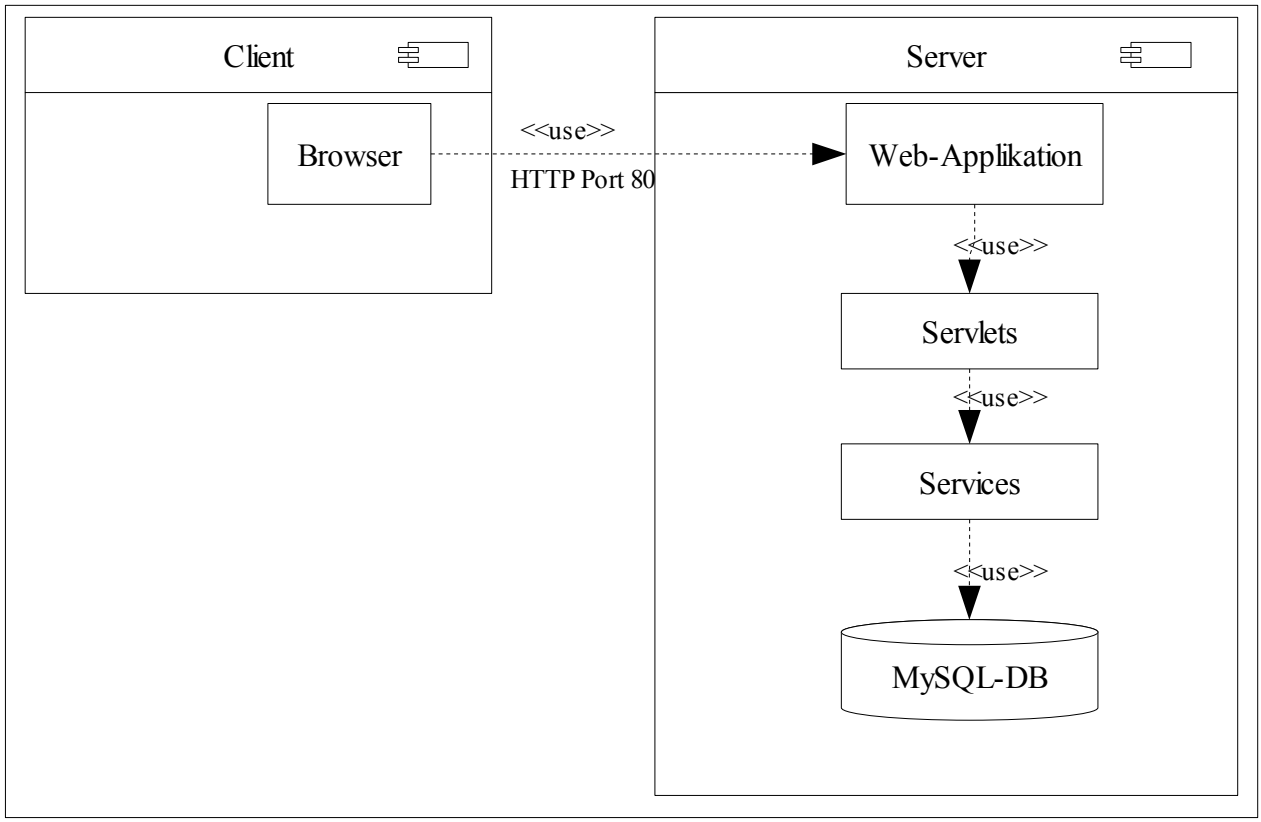

Abbildung 2.5 Gesamtstruktur des JCommSy-Systems

Die JCommSy-Architektur ist grundlegend eine Klient-/Anbieter-Architektur [siehe BD04, Seite 174f]. Das JCommSy-System bietet über die Web-Oberfläche Klienten Dienste an. Der Klient greift mit einem Browser auf das System zu und benutzt die angebotenen Dienste.

2.2 LAssi (Learners Assistant)

2.2.1 Beschreibung des Systems

Die LAssi-Software wird im Projekt „Reinventing Education-Werkzeuge für das Lernen“, einer Kooperation von IBM und der Universität Hamburg entwickelt. Sie ist eine Anwendung zur Unterstützung von individuellem Lernen. Das Hauptaugenmerk liegt bei LAssi darauf, dass es anpassbar für alle Altersklassen ist und so jeden bei seinen individuellen Lernbedürfnissen unterstützen kann.

2.2.2 Funktionalität und Architektur

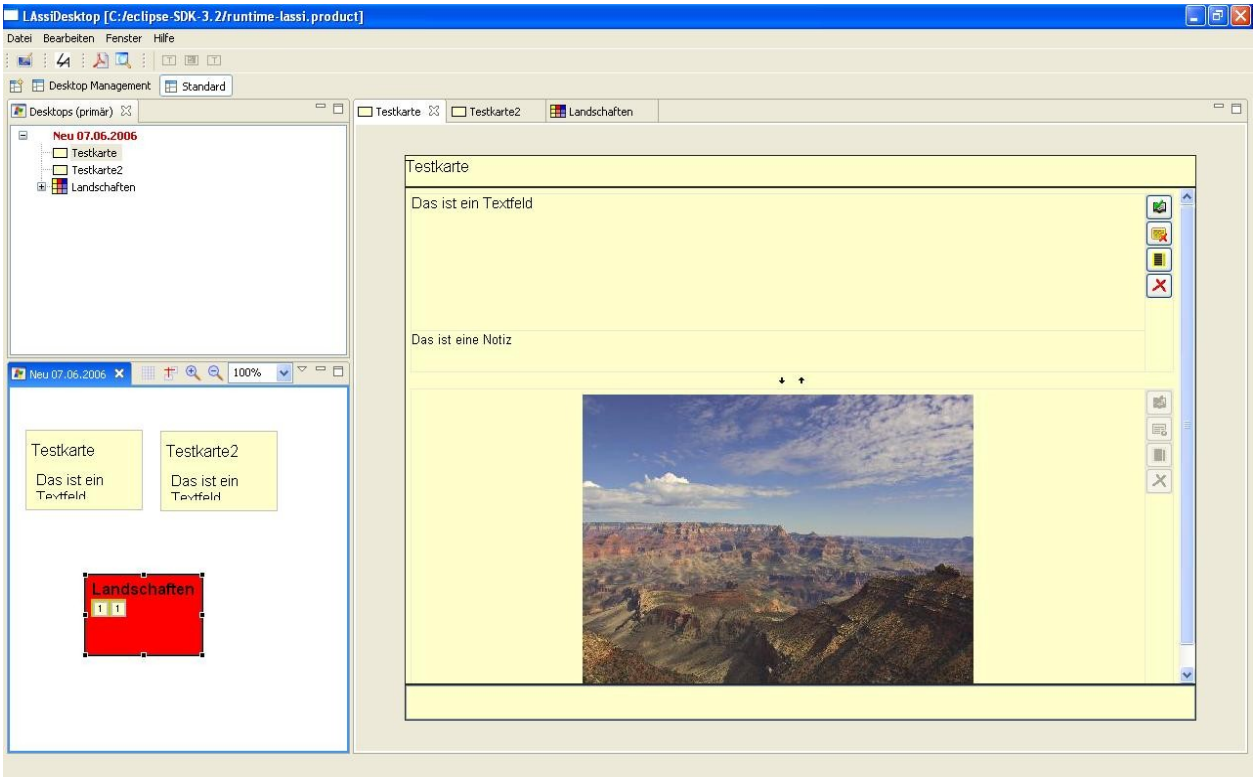

Abbildung 2.6 Screenshot LAssi-Desktop

LAssi baut auf einer Karteikarten-Metapher auf, die auf Arbeitsflächen, sogenannten Desktops angeordnet werden können. Abbildung 2.6 zeigt einen Bildschirm von einem LAssi-Desktop. Dort ist auf der rechten Seite groß eine Karteikarte zu erkennen. Diese Karteikarten können verschiedene Inhalte haben: Texte, Bilder oder auch Links zu externen Quellen und Dokumenten. Die Karteikarte findet sich auf dem Desktop in der linken Bildschirmhälfte wieder. Auf jedem Desktop (jeder Benutzer kann eine beliebige Anzahl von Desktops erstellen) können beliebig viele Karteikarten angelegt werden. Diese können zur Strukturierung frei verschoben werden. Des weiteren bietet LAssi die Möglichkeit, mehrere Karten zu Stapel zu gruppieren oder diese in sogenannte Pro-und Kontra-Kästen einzusortieren.
Das LAssi-System basiert auf der Eclipse Rich-Client-Platform (siehe [Dau05]). Daher ist das System als Plug-In- Architektur angelegt. Jedes dieser Plug-Ins enthält zwei Komponenten. Eine

enthält die Logik, die Andere enthält die grafische Darstellung. Es ist möglich, über sogenannte Extension Points an den bestehenden Plug-Ins weitere Softwarebausteine anzuschließen. Extension Points sind ein Konstrukt der Eclipse Rich-Client-Platform und bieten die Möglichkeit, zu jedem definierten Plug-In anzugeben, an welchen Stellen die Funktionalität des Plug-Ins durch weitere Plug-Ins erweitert werden kann.

Kapitel 3 Technologiegrundlagen

Da die in dieser Arbeit betrachtete Software, das JCommSy, auf einer Service-Architektur aufbaut, erscheint es naheliegend, diese durch Web Services zu erweitern und so für viele Client-Systeme zugreifbar zu machen. In diesem Kapitel wird zunächst der Begriff „Web Services“ kurz definiert, um dann näher auf die für diese Arbeit wichtigen Technologien SOAP und XML-RPC einzugehen.

3.1 Web Services

Der Begriff „Web Service“ ist nicht eindeutig definiert, es lassen sich verschiedene Interpretationen finden. Einige Beispiele:

„Jeder Service, der über das Internet verfügbar ist, ein standardisiertes XML-Nachrichtensystem verwendet und nicht an ein einzelnes Betriebssystem oder eine einzelne Programmiersprache gebunden ist, ist ein Web Service“ [Cer02, Seite 3]

„A Web Service is a software application identified by a URI, whose interface and bindings are capable of being identified, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols“ [World Wide Web Consortium, MMST03, Seite 4]

„Webservices sind aktive Programme/Prozesse oder Softwarekomponenten in einer speziellen Umgebung, die den Zugriff und das Management gewisser Ressourcen erlauben, um bestimmte Funktionen für diese Umgebung erfüllen zu können“ [Dum03, Seite 41]

Das World Wide Web Consortium beschäftigt seit 2002 mehrere Arbeitsgruppen mit der Entwicklung verschiedener Bereiche um den Begriff Web Services, darunter fallen die Gruppen, die die Web Service Description Language (WSDL) standardisiert haben.

Die Web Services, die in dieser Arbeit entwickelt werden sollen, folgen der ersten oben aufgeführten Definition.

Der Begriff „Web Service“ lässt sich am besten durch die Eigenschaften beschreiben, die Web Services haben. Dies sind folgende:

- Web Services sind Softwarestücke, die auf einem Server laufen
- sie sind für einen geeigneten Client über das Internet zugänglich
- sie bieten eine klar spezifizierte Funktionalität an

Es lässt ergänzend sagen, dass Web Services eine Schnittstelle bilden, über die Clients eine sonst nur auf einem lokalen Rechner verfügbare Software über ein Netz, wie zum Beispiel das Internet, nutzen können.

Um Web Services zu realisieren, gibt es viele verschiedene Konzepte und Technologien. Diese Arbeit beschäftigt sich primär mit den Technologien SOAP und XML-RPC, die für die Realisierung der Anbindung von Clients an das JCommSy in Frage kommen.

Nach der Vorstellung dieser Technologien wird betrachtet, wie Web Services bekannt und zugreifbar gemacht werden können, nämlich mit Hilfe der Web Service Description Language (WSDL).

3.2 SOAP

SOAP ist laut der Spezifikation ein „leichtgewichtiges Protokoll, das darauf abzielt, in einer dezentralen, verteilten Umgebung strukturierte Informationen auszutauschen“ (siehe [SOAP]).

Das Protokoll SOAP (Simple Object Access Protocol) entstand 1998 aus einer Idee für ein XML-basierten Remote Procedure Call-Mechanismus von Dave Winer von der Firma Userland Technologies. In Zusammenarbeit mit der Firma DevelopMentor und Microsoft entstand etwa ein Jahr später SOAP 0.9. Diese Version wurde als Entwurf an die IETF (Internet Engineering Task Force) gegeben, und durch die Mitarbeit mehrerer weiterer Firmen wie zum Beispiel DevelopmentMentor und Rogue Wave überarbeit. Im November 1999 gab Microsoft die Veröffentlichung von SOAP 1.0 bekannt, zusammen mit der Bekanntgabe, dass es bereits eine große Gruppe von Firmen in der Industrie gibt, die SOAP aktiv unterstützen wollen. Als Folge dieser Veröffentlichung schlossen sich auch IBM und SAP der Entwicklung an. Bereits im Jahr 2000 entstand aus dieser Kollaboration die Version 1.1 von SOAP, die an das World Wide Web Consortium (W3C) gegeben wurde, um sie zu einem offiziellen Standard zu machen. Dort wurde eine Arbeitsgruppe gegründet, die das Protokoll prüfen und gegebenenfalls weiterentwickeln sollte. Aus dieser Arbeit ist der Standard SOAP 1.2 entstanden. Der Begriff SOAP wurde ab diesem Zeitpunkt als Eigenname verwendet.

3.2.1 Die Spezifikation

Die SOAP-Spezifikation definiert, wie mit Hilfe von XML strukturierte Informationen zu einer Nachricht zusammengesetzt werden. Eine SOAP-Nachricht ist ein Plain-Text XML-Dokument (genannt „Envelope“), das mehrere Teile hat. Einen optionalen Header mit Informationen über die Nachricht sowie genau einen Body, der die eigentliche Nachricht enthält. Eine SOAP-Nachricht sieht beispielsweise wie folgt aus (die Nummerierung der Zeilen dient nur der Orientierung, diese sind in der eigentlich Nachricht nicht vorhanden):

```
1. <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"        Envelope
2.   env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
3.   <env:Header>                                                          Header
4.    <n:Aufrufe xmlns:n="http://firma.org/aufrufe"
5.       SOAP:mustUnderstand>
6.         5
7.    </n:Aufrufe>
8.   </env:Header>
9.    <env:Body>                                                           Body
10.    <m:zeitgeber xmlns:m="http://firma.org/zeitgeber">
11.     <m:ort>Hamburg</m:ort>
12.    </m:zeitgeber>
13.   </env:Body>
14. </env:Envelope>
```

Über das Attribut xmlns in der ersten Zeile wird der XML-Namespace angegeben, zu dem diese SOAP-Nachricht gehört, in diesem Beispiel „`http://www.w3.org/2003/05/soap-envelope`". XML-Namespace ist ein W3C-Standard, der dazu dient, Namenskollisionen in XML-Dokumenten zu verhindern. Namensräume werden als URI (Uniform Resource Identifier) beschrieben. In der zweiten Zeile wird die Kodierung der Nachricht festgelegt. Diese lässt sich entweder global für die gesamte Nachricht festgelegt werden, kann aber auch für jeden Header und Body-Block separat festgelegt werden. Die von der SOAP-Spezifikation angebotenen Standardkodierung, die unter „`http://www.w3.org/2003/05/soap-encoding`" zu finden ist, basiert auf einem grundlegenden Typsystem. Dabei handelt es sich um folgende Typen:

- int: Integer-Zahlen
- float: Gleitkomma-Zahlen
- String: Zeichenketten
- enumerations: Aufzählungen von zulässigen Werten
- array of bytes: Daten, die in Binärdarstellung vorliegen

Hinzu kommen zusammengesetzte Typen:

- array: Entspricht dem gleichnamigen Konstrukt in diversen Programmiersprachen. In einem Array werden Elemente gleichen Datentyps zusammengefasst und sind über den Array-Namen und einen Index zugreifbar

- struct: Entspricht dem struct-oder record-Konstrukt in vielen Programmiersprachen

- weitere Datentypen, die sich aus Verschachtelungen der oben genannten zusammen-gesetzten Datentypen ergeben

Datenelementen in einer SOAP-Nachricht lässt sich explizit der gewünschte Typ zuordnen. Um zum Beispiel einem Datenelement den Typ „String" zuzuordnen, benötigt man folgende Syntax:

```
<text xsi:type="SOAP-ENC=string"> Inhalt </name>
```

Um dies tun zu können, müssen mehrere Namensräume eingebunden werden. Das Attribut „xsi:type" stammt aus dem Namensraum „`http://www.w3.org/2001/XMLSchema-instance`" und muss vorher in der SOAP-Nachricht eingeführt worden sein. Zusätzlich muss auch der Namensraum „`http://www.w3.org/2001/XMLSchema`" eingeführt worden sein. Dieser Namensraum enthält alle Typen des XML-Schema-Standards. Da SOAP zusätzliche Eigenschaften von Typen bietet, muss auch der mit SOAP-ENC bezeichnete Namensraum „`http://schemas.xmlsoap.org/soap/encoding/`" in der SOAP-Nachricht eingeführt worden sein.

Eine dieser Besonderheiten ist, dass bereits vorhandene Datenelemente mit einem Identifikator versehen werden können, auf den später mit einem Verweis zugegriffen werden kann, damit sich oft wiederholende Datenelemente die Nachricht nicht unnötig vergrößern. Dieser Verweis wird mit den Schlüsselworte `id` und `href` realisiert. Ein Beispiel:

```
<text id="text1" xsi:type="SOAP-ENC=string"> Inhalt </name>
```

Auf den so gesetzten Bezeichner `"text1"` kann dann mittels `href` zugegriffen werden:

```
<neuertext href="#text1"/>
```

Diese Verweise sind für zusammengesetzte Datentypen wie `struct` und `array` zulässig.

Da Nachrichten in Netzen üblicherweise über mehrere Knoten laufen, ist der Header (im Beispiel findet sich der Header in den Zeilen 2-8) einer SOAP-Nachricht dafür vorgesehen, Informationen für die Vermittelnden Knoten zu enthalten. Dabei kann der Header zwei Attribute enthalten, die spezifizieren, für welche Knoten die Informationen relevant sind. Das Attribut `role` legt genau fest, welcher Knoten gemeint ist. Die SOAP-Spezifikation definiert in diesem Zusammenhang drei Rollen: Den nächsten Knoten (next), keinen Knoten (none) und den Zielknoten (ultimateReceiver). Dies wird mit einer URI festgelegt, zum Beispiel gibt das Tag `<role:"http://www.w3.org/2003/05/soap-envelope/role/next">` an, dass der nachfolgende Headerteil für den nächsten Knoten auf dem Weg relevant ist. Zudem kann das Attribut `mustUnderstand` gesetzt werden. Wird dieses Attribut auf den Wert "1" gesetzt, muss der betreffende Knoten den Inhalt des Header-Teils ausführen. Wird `mustUnderstand` auf den Wert "0" oder gar nicht gesetzt, darf der Teil ignoriert werden. Im Rahmen einer eigenen Implementierung können frei weitere Rollen definiert werden, die Spezifikation macht keine Vorschriften, wie diese Rollen auszusehen haben.

Wird eine SOAP-Nachricht an einen Empfänger versendet, so ist von der Spezifikation nicht vorgesehen, über welche Knoten diese Nachricht verschickt wird. Erreicht eine SOAP-Nachricht einen Knoten, muss dieser wie folgt vorgehen, um die Nachricht zu verarbeiten:

1. Feststellen, in welcher Rolle sich der Knoten befindet
2. Identifizieren aller Header-Blöcke, die an diesen Knoten gerichtet sind und in denen das Attribut `mustUnderstand` den Wert „1" hat
3. Sollte einer oder mehrere der in Schritt 2 identifizierten Blöcke nicht verständlich sein, muss ein Fehler vom Typ `SOAP-Fault` (siehe unten) generiert und zurückgegeben werden. Wird so ein Fehler generiert, darf keine weitere Verarbeitung der Nachricht erfolgen.
4. Verarbeiten der für den aktuellen Knoten relevanten und verpflichtenden SOAP-Header-Blöcke und, falls der aktuelle Knoten der Zielknoten ist, auch des SOAP-Body-Blocks.
5. Falls der aktuelle Knoten ein Zwischenknoten ist und kein Fehler bei der Verarbeitung aufgetreten ist, wird die Nachricht weitergeleitet.

Die Reihenfolge, in der die Header-Blöcke und der Body-Block verarbeitet werden, ist von der Spezifikation nicht festgelegt. Es besteht aber die Möglichkeiten, einen Header-Block zu schreiben, der vorgibt, in welcher Reihenfolge die Verarbeitung erfolgen soll. Es kann für Logging-Zwecke zum Beispiel sinnvoll sein, einen Header-Block vorzugeben, der dafür sorgt, dass die restlichen Header-Blöcke überlappend mit dem Body-Block zu verarbeiten sind

Die Weiterleitung einer Nachricht unterliegt bestimmten Kriterien, insbesondere was die Weiterleitung der Header-Blöcke angeht. Erhält ein Knoten eine SOAP-Nachricht, prüft er, ob es Header-Blöcke gibt, die ihn betreffen. Je nach Resultat dieser Prüfung gibt es verschiedene Möglichkeiten, wie weiterverfahren wird:

1. Gibt es keinen Header-Block, der den aktuellen Knoten betrifft und ist dieser Knoten nicht der Empfänger der Nachricht, leitet der Knoten die gesamte Nachricht an den nächsten Knoten weiter.

2. Gibt es Header-Blöcke, so findet eine weitere Unterscheidung statt. Wird ein Header-Block verarbeitet, wird er aus der Nachricht entfernt, es sei denn, es ist ein `reinsert`-Befehl eingefügt. Dann wird der Block in die weitergeleitete Nachricht wieder eingefügt. Wird ein Header-Block ignoriert (wenn `mustUnderstand` den Wert "0" hat oder nicht gesetzt ist), so wird er ebenfalls aus der Nachricht entfernt, es sei denn, ein `relay`-Befehl ist vorhanden.

Tritt während der Verarbeitung einer SOAP-Nachricht ein Fehler auf, soll ein SOAP-Fault generiert werden. Dieser Fehler ist eine SOAP-Nachricht, die in ihrem Body ein einzelnes `Fault` Element hat. In diesem Element muss ein `Code`-Element und ein `Reason`-Element vorhanden sind. Das `Code`-Element enthält einen Fehlercode. Die SOAP-Spezifikation bietet hierfür eine kleine Anzahl von Fehlercodes [SOAP]; weitere können selbst hinzugefügt werden. Zudem kann ein `Subcode`-Element vorhanden sein, das weitere Fehlercodes enthält. Dieses Element besteht ebenfalls aus einem `Code`- und einem optionalen `Subcode`-Element. Die Tiefe der Verschachtelungen ist durch die Spezifikation nicht begrenzt.
Das `Reason`-Element der Fehlermeldung enthält eine für Menschen lesbare Fehlermeldung.

3.2.2 SOAP Nachrichten versenden

Die Versendung von SOAP-Nachrichten durch Netze erfolgt mit Hilfe eines nicht näher definierten Protokolls. In der Spezifikation wird eine Bindung an HTTP näher beschrieben [SOAP], SOAP-Nachrichten können ebenfalls über SMTP (Simple Mail Transfer Protocol) oder direkt über TCP versendet werden. Der Vorteil von HTTP liegt darin, dass HTTP auf dem Request-Response-Prinzip arbeitet, das für viele Anwendungen von Nutzen ist. Dabei werden Request und Response synchron in einer Verbindung übertragen. Ein HTTP-Request mit einer SOAP-Nachricht könnte wie folgt aussehen:

HTTP Request

```
1. POST /soapservlet HTTP/1.1
2. Content-Type: text/xml; charset="utf-8"
3. Content-Length: nnnn
4. SOAPAction: ""
5.
```

SOAP Request

```
6. <env:Envelope …
7. …
8. </env:Envelope>
```

Die Zeilen 1-8 enthalten den gesamten HTTP-Request, in den Zeilen 6-8 ist im HTTP-Body ein SOAP-Request eingebettet.

3.2.3 SOAP und RPC

Die für diese Arbeit wichtigste Funktionalität von SOAP ist der Remote Procedure Call mit Hilfe von SOAP. Basierend auf der Bindung an HTTP erfolgt der RPC in SOAP in einem Request-Response-Mechanismus. Um einen RPC mit SOAP durchzuführen, wird eine Plain-Text-SOAP-Nachricht generiert. Allerdings befinden sich nicht nur allgemeine Dokumentinformationen in der Nachricht, sondern der Name der aufzurufenden Methode so wie deren Parameter werden serialisiert als Struktur im Body-Teil der Nachricht eingetragen. Dabei muss jeder Parameter der Methode im Body-Teil vorhanden sein und in der Reihenfolge aufgeführt sein, wie er in der Methodensignatur vorkommt. Ein Beispiel für einen Request:

HTTP-Request

```
1. POST /soapservlet/Konto HTTP/1.1
2. Content-Type: text/xml; charset="utf-8"
3. Content-Length: nnnn
4. SOAPAction: "http://beispiel.org/Konto"
```

SOAP-Request

```
5. <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
6.  <env:Body>
7.   <op:getKontostand xmlns:op="http://beispiel.org/Konto">
8.    <kontonummer>12345678</kontonummer>
9.    <blz>20050050</blz>
10.  </op:getKontostand>
11. </env:Body>
12.</env:Envelope>
```

In diesem Request wird an der Adresse „`http://beispiel.org/Konto"` die Methode `getKontostand(int Nummer, String Bank)` ausgeführt.

Die Antwort sieht so aus:

HTTP-Response

```
1. HTTP/1.1 200 OK
2. Content-Type: text/xml; charset="utf-8"
3. Content-Length: nnnn
```

SOAP-Response

```
4. <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
5.  <env:Body>
6.   <op:getKontostandResponse xmlns:op="http://beispiel.org/Konto">
7.    <result>100</result>
8.   </op:getKontostand>
9.  </env:Body>
10.</env:Envelope>
```

Das Ergebnis der Methode `getKontostand` wird im Body-Teil der Nachricht als Struktur repräsentiert, wobei alle Rückgabewerte der Methode vorhanden sein und in der richtigen Reihenfolge vorliegen müssen.

Tritt ein Fehler auf, ist ein geeignetes SOAP-Fault-Element statt der Methodenergebnisses zu generieren und zurückzugeben.

Für weiterführende Informationen zu SOAP siehe auch [KPRR04, Seite 156f], [Eng02] und [Cer02, Seite 49ff].

3.3 XML-RPC

Das Protokoll XML-RPC (XML-Remote Procedure Call) wurde 1999 von Dave Winer bei der Firma Userland Software entwickelt. Am 15. Juni 1999 wurde die Spezifikation veröffentlicht [XML-RPC]. Das Ziel von XML-RPC ist es, auf möglichst einfache Weise eine Schnittstelle zu schaffen, über die unterschiedlichste Systeme miteinander kommunizieren können. Dabei ist es im Gegensatz zu SOAP exklusiv darauf ausgelegt, Prozeduren aufzurufen und Ergebnisse zurückzugeben.

XML-RPC ist an HTTP gebunden und nutzt das so gegebene Request-Response-Prinzip. Ein XML-RPC-Request sieht wie folgt aus:

HTTP-Request

```
1. POST /RPC HTTP/1.1
2. User-Agent: CommSy/4.1.1 (WinXP)
3. Host: commsy3.informatik.uni-hamburg.de
4. Content-Type: text/xml
5. Content-length: 181
```

XML-RPC-Request

```
6. <?xml version="1.0"?>
7. <methodCall>
8.  <methodName>appointmentService.existsAppointment</methodName>
9.   <params>
10.   <param>
11.    <value><i4>8086</i4></value>
12.   </param>
13.  </params>
14. </methodCall>
```

Die Zeilen 1-5 enthalten den HTTP-Request, die Zeilen 7-15 den XML-RPC-Request. Im HTTP-Teil muss als Content-Type text/xml enthalten sein und das Feld Content-length muss vorhanden sein und die korrekte Länge des Requests enthalten. Der XML-RPC-Teil muss genau ein Element mit Namen `<methodCall>` enthalten. Dieser `<methodCall>` enthält seinerseits die genau ein Element `<methodName>` und beliebig viele Elemente vom Typ `<params>`. Das Element `<methodName>` enthält den Namen der aufzurufenden Methode als String, das Element `<params>` enthält die Parameter, die die aufzurufende Methode hat, in der Reihenfolge, in der sie in der Methodensignatur stehen. Diese Parameter haben ein `<value>`-Element, dass den Typ und den Wert des Parameters kennzeichnet. XML-RPC kennt folgende Typen:

`Int`, `double`, `boolean`, `string`, die den in Programmiersprachen üblichen Konstrukten entsprechen. Hinzu kommen der Typ `dateTime.iso8601`, der das Datum und die Zeit im Format `20060419T14:08:55` abspeichert, und der Typ `base64`, der Binärdaten codiert abspeichert.

Zu diesen einfachen Datentypen kommen die zusammengesetzten Typen `struct` und `array`. Der Typ `struct` enthält ein oder mehrere `<member>`-Elemente, die ihrerseits ein `<name>`-Element und ein `<value>`-Element haben. Das `<value>`-Element kann wieder `<struct>`-Elemente enthalten. Ein Beispiel für eine Struktur mit zwei Elementen:

```
<struct>                                          XML-RPC-Konstrukt

 <member>                                         Elementarer Datentyp
  <name>Kontonummer</name>
  <value><int>12345678</int></value>
 </member>

 <member>                                         Elementarer Datentyp
  <name>Bank</name>
  <value><string>Sparkasse</string></value>
 </member>

</struct>
```

Der Typ `array` enthält ein Datenelement, dass mehrere Werte enthält. Arrays können neben den simplen Datentypen auch die zusammengesetzten Datentypen enthalten. Ein Beispiel:

```
<array>                                           Array-Definition

 <data>                                           enthaltene Datentypen
  <value><int>12345678</int></value>
  <value><string>TestString</string></value>
  <value><boolean>0</boolean></value>
 </data>

</array>
```

Eine HTTP-Response zu dem gezeigten Request würde so aussehen:

```
1. HTTP/1.1 200 OK                                HTTP-Response
2. Connection: close
3. Content-Length: 158
4. Content-Type: text/xml
5. Date: Thu, 19 Apr 2006 19:55:08 GMT
6. Server: CommSy Server (WinXP)
                                                  XML-RPC-Response
7. <?xml version="1.0">
8.  <methodResponse>
9.    <params>
10.      <param>
11.         <value><boolean>true</boolean></value>
12.      </param>
13.    </params>
14.  </methodResponse>
```

Die erste Zeile enthält immer den Code `200 OK`, wenn kein HTTP-Fehler aufgetreten ist. Der XML-RPC-Teil muss genau ein `<methodResponse>`-Element enthalten, das, analog zum `<methodCall>`-Element des Requests, ein `<params>`-Element hat, das einen oder mehrere `<param>`-Elemente besitzt. Diese Elemente enthalten die Rückgabewerte der im Request aufgerufenen Methode als `<value>`-Elemente. Tritt bei der Verarbeitung des Requests ein Fehler auf, enthält der XML-RPC-Teil innerhalb des `<methodResponse>`-Elements ein `<fault>`-Element. Dieses enthält einen Integer-Fehlercode und ein Faultstring-Element, dass eine lesbare Fehlermeldung enthält.

3.4 WSDL

Die Web Service Description Language (WSDL) ist eine auf XML basierende Sprache, die es ermöglicht, Web Services präzise zu beschreiben und so für Clients zugreifbar zu machen. Entwickelt wurde die WSDL durch Microsoft und IBM. Die Version 1.1 wurde im Mai 2001 an das World Wide Web Consortium weitergegeben und dort durch eine Arbeitsgruppe weiterentwickelt. Aktuell ist die Version 2.0, die am 27.März 2006 den Status eines W3C Candidate Recommendation erhalten hat, also zur Diskussion steht, eine Empfehlung des W3C zu werden [WSDL].

Dieses Kapitel behandelt WSDL in der Version 1.1, da alle verfügbaren Implementierungen von SOAP bis heute nur die Version 1.1 unterstützen

Ein Web Service wird mit WSDL durch sechs Elemente eindeutig beschrieben:

- Types: Enthält Elemente, die die Typen, die im Web Service verwendet werden, beschreiben
- Message: Enthält Elemente, die alle vom Web Service möglichen Nachrichten beschreiben
- Port Type: Enthält Elemente, die alle vom Web Service angebotenen Methoden beschreiben, ohne zu spezifizieren, welches Protokoll verwendet wird oder wo die Methoden liegen.
- Binding: Enthält die Beschreibung des Protokolls, das der Web Service verwendet, zum Beispiel SOAP
- Service: Enthält den Namen des Services sowie einen oder mehrere Ports
- Port: Enthält die genaue Netzwerkadresse, an denen die vom Service referenzierten Methoden liegen

Die ersten drei dieser Elemente sind vollständig abstrakt gehalten, sie enthalten weder Informationen darüber, welches Protokoll für den Web Service verwendet werden, noch an welchem Ort die konkreten Methoden liegen. Eine Abstraktionsebene tiefer liegt das Binding-Element, das das Protokoll festlegt. Dabei verweist dieses Element auf die Ports, die beim Aufruf der Methoden verwendet werden sollen. Diese Ports stellen die konkrete Ebene des WSDL-Dokuments dar, sie enthalten jeweils genau eine URI, die angibt, wo die Methoden zu finden sind, die unter PortType beschrieben wurden. Zur Veranschaulichung soll das folgende Beispiel einer WSDL-Datei dienen:

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <wsdl:definitions
   targetNamespace="http://localhost:8080/commsy/services/Version"
   xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:impl="http://localhost:8080/commsy/services/Version"
   xmlns:intf="http://localhost:8080/commsy/services/Version"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3.    <wsdl:message name="getVersionResponse">
4.       <wsdl:part name="getVersionReturn" type="soapenc:string"/>
5.    </wsdl:message>
6.    <wsdl:message name="getVersionRequest">
7.    </wsdl:message>
```

```
8.    <wsdl:portType name="Version">
9.       <wsdl:operation name="getVersion">
10.         <wsdl:input message="impl:getVersionRequest"
               name="getVersionRequest"/>
11.         <wsdl:output message="impl:getVersionResponse"
                     name="getVersionResponse"/>
12.      </wsdl:operation>
13.   </wsdl:portType>
14.   <wsdl:binding name="VersionSoapBinding" type="impl:Version">
15.      <wsdlsoap:binding style="rpc"
                     transport="http://schemas.xmlsoap.org/soap/http"/>
16.      <wsdl:operation name="getVersion">
17.         <wsdlsoap:operation soapAction=""/>
18.         <wsdl:input name="getVersionRequest">
19.            <wsdlsoap:bodyencodingStyle="http://schemas.xmlsoap.org/soap/
                encoding/" namespace="http://axis.apache.org" use="encoded"/>
20.         </wsdl:input>
21.         <wsdl:output name="getVersionResponse">
22.            <wsdlsoap:bodyencodingStyle="http://schemas.xmlsoap.org/soap/
                   encoding/" namespace="http://localhost:8080/commsy/
                   services/Version" use="encoded"/>
23.         </wsdl:output>
24.      </wsdl:operation>
25.   </wsdl:binding>
26.   <wsdl:service name="VersionService">
27.      <wsdl:port binding="impl:VersionSoapBinding" name="Version">
28.         <wsdlsoap:addresslocation="http://localhost:8080/commsy/services/
               Version"/>
29.      </wsdl:port>
30.   </wsdl:service>
31.</wsdl:definitions>
```

In dieser Datei ist durch die Markierung der Pfad sichtbar, der von der aufrufbaren Methode (Zeile 8 und 9) zu dem zugehörigen SoapBinding für diese Methode (Zeilen 14-16) bis zu dem Port-Element führt, das zu dem SoapBinding die konkrete Klasse und die Adresse des zugehörigen Services angibt (Zeilen 26-28).

3.5 Einordnung der Technologien

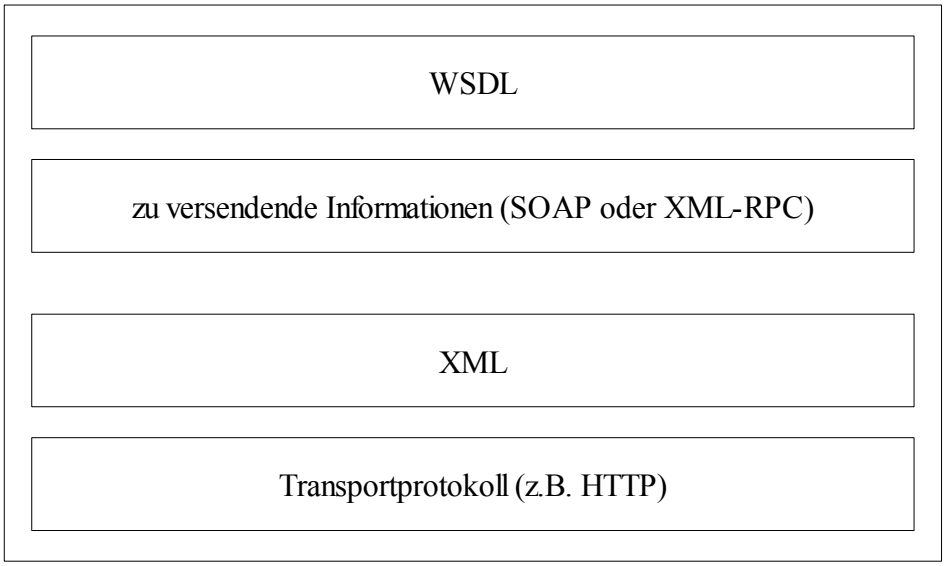

Abbildung 3.1 Web Service-Schichtenmodell

Die in diesem Kapitel beschriebenen Technologien sind notwendig, um eine Web Service-Schnittstelle für das JCommSy erstellen zu können. Abbildung 3.1 zeigt ein Schichtenmodell (siehe KPRR04, Seite 154f], das die Web Service-Architektur darstellt. Als unterste Schicht fungiert ein Transportprotokoll, im Fall von SOAP und XML-RPC ist dies üblicherweise HTTP. Darüber liegt die XML-Schicht, auf der die beiden in 3.2 und 3.3 besprochenen Transportprotokolle aufbauen. Die Web Service-Beschreibungssprache WSDL liegt darüber und beschreibt, welche Funktionalitäten durch den Web Service zur Verfügung gestellt werden und welches Transport und Übertragungsprotokoll verwendet wird.

Historisch gesehen haben sich Web Services seit der Begründung des World Wide Web entwickelt. Schon in der Anfangszeit des WWW wurde es nötig, Anwendungen über das Netz zu steuern oder zu verwenden. Dies wurde besonders durch die Entwicklung der Java RMI-Schnittstelle (Java Remote Method Invocation) populär. Da diese aber nur Interaktionen zwischen Java-basierten System erlaubt, wurde es bald nötig, weitere Standards zu entwickeln, die Interoperabilität zwischen (fast) beliebigen Programmiersprachen ermöglicht. Eine Möglichkeit, dies zu tun, sind Middleware-Lösungen wie CORBA, die aber zu komplex sind, um sie für einfache Anwendungen einsetzen zu können. Daher wurde Ende der 90er Jahre des 20 Jahrhunderts von mehreren Firmen, darunter Microsoft, eine neue Technologie vorangetrieben, die mit Hilfe von XML universell Daten austauschen konnte. Aus diesen Bemühungen entstand zunächst XML-RPC und davon abgespalten später das SOAP-Protokoll. Ab Anfang 2000 wurde der Begriff „Web Service“ durch das World Wide Web Consortium eingeführt.

Kapitel 4 Anforderungen an die Client-Anbindung

Dieses Kapitel beschäftigt sich mit der Frage, auf welcher Ebene eine Rich-Client-Anbindung an das JCommSy möglich und sinnvoll ist. Im zweiten Teil des Kapitels wird festgelegt, welche der im vorangegangenen Abschnitt besprochenen Technologien, SOAP oder XML-RPC, für die Anbindung eingesetzt werden soll.

4.1 Ebene der Anbindung

Abbildung 2.5 zeigt die Gesamtstruktur des JCommSy-Systems, insbesondere die verschiedenen Schichten, aus denen das System besteht. Es wird im Folgenden geklärt, auf welcher Schicht ein Client-System angebunden werden kann. Dazu werden die einzelnen Schichten besprochen und die Möglichkeit einer Anbindung bewertet.

Die oberste Schicht ist die Web-Oberfläche, also die Webseiten des JCommSy. Um an dieser Schicht ein Client-System mit Web Services anzubinden, muss der Web Service in der Lage sein, die Informationen der Websites auszuwerten und an den Client zu übergeben. Diese Lösung erscheint jedoch aufwendig und unpraktikabel, insbesondere falls sich die Struktur der Webseiten ändern sollte.

Die Service-Schicht des JCommSy enthält die Services, die den Datenaustausch zwischen dem CommSy-Servlet und der Datenbank mit Hilfe von Hibernate realisieren. Um eine Client-Anbindung auf dieser Schicht zu realisieren muss die Funktionalität der jeweiligen Services über eine oder mehrere Web-Service-Klassen gewrappt und so über die Schnittstelle für Clients verfügbar macht.

4.2 Anforderungen an das JCommSy

Um bewerten zu können, welche der in Kapitel 3 vorgestellten Technologien für die Anbindung von Clients an das JCommSy sinnvoll und umsetzbar ist, werden zunächst die grundlegenden Anforderungen vorgestellt, die eine solche Anbindung stellt.

Um Rich-Clients (in dieser Arbeit am Beispiel der Lernsoftware LAssi) an das JCommSy anschließen zu können, muss es ermöglicht werden, die bestehenden Services des JCommSy zu nutzen. Hierbei gibt es mehrere Bereiche, die abgedeckt werden müssen:

1. Userverwaltung

Die derzeit bestehende Version von JCommSy funktioniert nur in Verbindung mit der PHP-Version von CommSy. Insbesondere geschieht die Nutzer-Anmeldung über das PHP-System und wird vom Java-System übernommen. Der Benutzer meldet sich über das Web-Portal mit seiner Kennung und seinem Passwort an und bekommt einen Cookie zurück, der seinen Status als angemeldeter User sichert. Diese Anmeldung muss direkt im JCommSy ermöglicht werden. Zum einen muss die Anmeldung über die Webseiten durch Cookies und eine SessionID ermöglicht werden, zum Anderen muss die Anmeldung durch Rich Clients verfügbar sein. Eine Möglichkeit, die Anmeldung mit Rich-Clients zu realisieren, ist, bei einem Login eines Nutzers von einem Rich-Client aus eine

SessionID zu vergeben, die der Nutzer bei jeder weiteren Anfrage an den Server wieder angeben muss. Alternativ kann das verwendete Protokoll eine Session öffnen, über die die gesamte folgende Kommunikation bis zum Ausloggen läuft. Zudem sollte es eine Möglichkeit geben, wichtige Daten wie das Passwort sicher zu übertragen.

2. Gruppenverwaltung

Da das mögliche Client-System LAssi mit Hilfe von JCommSy um Gruppenfunktionalität erweitert werden soll, muss das JCommSy eine entsprechende Gruppenverwaltung anbieten. Die Zuordnung von Nutzern zu einzelnen Gruppen ist im JCommSy bereits vorhanden, zusätzlich muss noch eine Schnittstelle angeboten werden, über die ein Client sehen kann, welche Nutzer aus seiner Gruppe derzeit am System angemeldet sind.

3. Materialverwaltung

Der Hauptzweck von JCommSy aus der Sicht von LAssi wird das Speichern und Verwalten von Materialien sein. Es muss möglich sein, Dateien – im Fall von LAssi Karteikarten-Dateien – an den JCommSy-Server zu übertragen. Dieser legt die Karten in seiner Datenbank ab, zusammen mit Informationen darüber, wer das Material wann angelegt hat.

4.3 Vor- und Nachteile von SOAP und XML-RPC

1. Userverwaltung

Die Benutzeranmeldung ließe sich als einfacher Web-Service realisieren, den der Client mit Benutzername und Passwort aufruft und als Antwort eine SessionID oder ein Session-Objekt bekommt, in dem die SessionID und die zugehörige UserID abgelegt sind. Wird die SessionID als Integer oder Long gesendet, ließe sich dies mit XML-RPC realisieren. Ein selbst definiertes Objekt vom Typ Session ist mit XML-RPC nicht zu realisieren. Hier bietet SOAP die besseren Möglichkeiten, insbesondere die volle Unterstützung für vom Programmierer selbst entwickelte Datentypen. Hinzu kommt, dass beim Architekturentwurf entschieden wird, ob es einen Web-Service für alle erwünschten Funktionalitäten geben soll oder ob es mehrere Web-Services geben soll, von denen jeder analog zur Architektur des JCommSy eine Funktionalität realisiert. Fällt die Entscheidung zugunsten von mehreren Web-Services, so muss an jedem einzelnen Service eine Anmeldung möglich sein, die dann auch den anderen Web Services bekannt sein muss.

Ein weiterer Vorteil von SOAP besteht darin, dass es Toolkits gibt, mit denen eine sichere Übertragung möglich wird. Die einzige Möglichkeit, mit XML-RPC Sicherheit zu garantieren, wäre durch die Benutzung des HTTPS-Protokolls. Dies ist aber nicht ausreichend, da die Knoten, die die Nachricht bei der Übertragung weiterleiten, den Klartext der Nachricht lesen können.

2. Gruppenverwaltung

Die Gruppenverwaltung, also die Betrachtung der vorhandenen Gruppen sowie die Erstellung neuer Gruppen kann über die Benutzung des entsprechenden Services des JCommSy realisiert werden. Auch die Zuordung eines Benutzers zu einer Gruppe lässt sich über den Service realisieren. Mit Hilfe von XML-RPC ist dies nur insoweit möglich, als dass einzelne Attribute einer Gruppe vom

System zum Client und zurück gesendet werden können. Mit Hilfe von SOAP ist es hingegen möglich, einen Typ „Gruppe“ zu definieren und Objekte von diesem Typ zu versenden. Dies würde es einem Client-Programmierer erleichtern, das Objekt „Gruppe“ aus dem JCommSy in seine eigene Applikation anzubinden. Auf der Client-Seite besteht dann die Möglichkeit, die Attribute einer Gruppe zu editieren und das Objekt zum JCommSy-Server zurückzusenden, wo es im JCommSy aktualisiert wird.

Um einem Nutzer von LAssi sichtbar zu machen, welche anderen Mitglieder seiner Gruppe(n) auch am System angemeldet sind, kann die in Punkt 1 beschriebene User-Verwaltung in Zusammenarbeit mit dem Gruppenservice verwendet werden, der oben beschrieben wurde. Die Wahl des Protokolls hängt in diesem Punkt von der Userverwaltung und der Gruppenverwaltung ab.

3. Materialverwaltung

Es sind zwei Fälle zu betrachten, nämlich einmal eine allgemeine Materialverwaltung von Dateien beliebigen Typs und die Verwaltung der von LAssi verwendeten Datentypen. XML-RPC ist grundsätzlich für beide Fälle geeignet, aber es gelten die identischen Einschränkungen wie in den beiden vorangegangenen Punkten. Da es im JCommSy den Typ „Material“ gibt, wäre dieser mit XML-RPC nicht direkt modellierbar, es könnten nur Attribute ausgetauscht werden, keine Objekte. In diesem Typ Material wird ein Link abgelegt, unter welchem Namen die Datei auf dem JCommSy-Server abgelegt wurde. Dieser Typ lässt sich mit SOAP modellieren. Die Übertragung der Datei an sich lässt sich sowohl in XML-RPC als auch in SOAP als Base64 codierten Datenstrom realisieren. In SOAP existiert zusätzlich die Möglichkeit, Attachments zu verwenden, die an eine SOAP-Nachricht angehängt werden können. Im Spezialfall der von LAssi verwendeten Datentypen besteht auch die Möglichkeit, diese direkt zu versenden, da diese in XML vorliegen, also nicht mehr gesondert serialisiert werden müssen. Es wäre auch möglich die gesamte XML-Datei als String in einem Material-Objekt zu versenden, dies würde von SOAP, aber nicht von XML-RPC unterstützt werden.

4.4 Technologiefestlegung

Aufgrund der in 4.2 beschriebenen Einschränkungen, die für XML-RPC gelten, fällt für den in dieser Arbeit betrachteten Kontext die Wahl der Technologie auf SOAP. Das Hauptargument ist hierbei, dass SOAP die Definition von eigenen Datentypen unterstützt und so die Programmierung sowohl auf der Client als auch auf der Server-Seite vereinfacht. Da im JCommSy mit vielen Materialien umgegangen wird, die auch über den Web Service verfügbar sein müssen, muss es eine praktikable Möglichkeit geben, diese Materialien mit Hilfe des Protokolls zu serialisieren und zu transportieren.

Kapitel 5 Entwurf einer Lösung

In diesem Kapitel wird beschrieben, wie eine Architektur aussehen kann, die es ermöglicht, das JCommSy um Web Service-Funktionalitäten zu erweitern. Dazu wird zunächst diskutiert, welche SOAP-Implementierung verwendet werden soll, diese wird ausführlicher beschrieben. Anschließend wird dargestellt, wie diese SOAP-Implementierung in das JCommSy eingebunden werden kann. Im Anschluss wird die Architektur anhand der Anforderungen, die LAssi an das JCommSy stellt, entworfen.

5.1 SOAP-Implementierung

Es besteht im Prinzip die Möglichkeit, die SOAP-Implementierung selbst zu schreiben, da es aber einige bereits bestehende SOAP-Implementierungen gibt, ist es von Vorteil, eine von diesen einzusetzen. Im Folgenden werden zwei dieser Implementierungen kurz vorgestellt, da die weiteren existierenden SOAP-Implementierungen zum größten Teil kommerzielle Produkte sind. Am Schluss dieses Abschnitts wird eine Implementierung ausgewählt, die am besten für die Integration in das JCommSy geeignet ist.

- Apache SOAP

Apache SOAP baut auf IBMs SOAP4J-Toolkit auf, das im Jahr 2000 der Apache Software Foundation zur Verfügung gestellt und dort als Open-Source-Projekt weiterentwickelt wurde. Die Entwicklung endete im Mai 2002 mit der Veröffentlichung der Version 2.3.1. Apache SOAP baut auf DOM als XML-Parser.

- Apache Axis

Apache Axis ist das Nachfolgeprojekt von Apache SOAP und wurde Ende 2000 ins Leben gerufen. Mitte 2001 gab es die erste Alphaversion. Axis ist eine komplette Neuentwicklung und baut nicht auf Apache-SOAP auf, es ist als Open-Source-Projekt angelegt. Axis verwendet den gegenüber DOM schnelleren SAX-XML-Parser und ist konform zur Sun JAX-RPC Spezifikation. Aktuell ist die Version 1.4. , die am 22. April 2006 veröffentlicht wurde [AXIS1].

Um eine Implementierung auszuwählen werden folgende Kriterien zu Grunde gelegt:

- Empfohlener Servlet-Container
- Aufwand, Java-Klassen in einen Web Service zu integrieren
- Unterstützung durch Werkzeuge zur Erzeugung von WSDL-Beschreibungen
- Unterstützung für Client-Entwickler durch Werkzeuge zur Erzeugung von Client-Stubs
- Erfüllung des JAX-RPC-Standards
- Performanz, insbesondere des XML-Parsers
- Zukunftssicherheit

In der folgenden Tabelle werden die beschriebenen Implementierungen verglichen.

Kriterium	Apache SOAP	Apache Axis 1.4
empfohlener Servlet-Container	Apache Tomcat	Apache Tomcat
erfüllt JAX-RPC	Nein	Ja
Generierung WSDL	Ja	Ja
Generierung Client Stubs	Ja	Ja
XML-Parser	DOM	SAX
wird weiterentwickelt	Nein	Ja (Axis2)
kostenfreie Lizenz	Ja	Ja

Tabelle 5.1 Vergleich SOAP-Implementierungen

Nach Evaluierung (siehe Tabelle 5.1) der Implementierungen hat sich ergeben, dass Apache Axis am besten geeignet erscheint, um Web-Services in das JCommSy zu integrieren. Dabei sind folgende Vorteile ins Gewicht gefallen:

- Axis arbeitet als Servlet in einem beliebigen Servlet-Container, von Apache empfohlen wird ein Apache Tomcat-Server. Da für das JCommSy bereits ein Tomcat-Server als Servlet-Container zur Verfügung steht, ist die Einbindung von Axis und damit von Web-Services einfach.

- Aus diesen Java-Klassen kann Axis eine WSDL-Beschreibung des Web-Services erstellen, aus der dann beliebige Client-Systeme entwickelt werden können.

- Axis bietet Werkzeuge an, um aus einer WSDL-Beschreibung Client-Java-Code zu erzeugen und erleichtert so die Programmierung von Client-Systemen.

- Da Axis den JAX-RPC-Standard von Sun erfüllt, können für Axis geschriebene Clients auch mit anderen Web-Services, die nicht Axis auf der Server-Seite verwenden, benutzt werden.

- Axis verwendet den SAX-XML-Parser. Dieser bietet Geschwindigkeitsvorteile gegenüber dem von Apache SOAP verwendetet DOM-XML-Parser.

- Axis ist modular aufgebaut, kann also an die eigenen Bedürfnisse angepasst werden.

- Die Lizenz von Axis lässt die kostenfreie Verwendung in eigenen Anwendungen zu.

Mögliche Nachteile von Axis:

- Axis unterstützt nur synchrone Aufrufe, das heißt, dass die Client-Applikation warten muss, wenn sie eine SOAP-Anfrage gestellt hat, bis eine Antwort erfolgt ist.

5.2 Apache Axis im Detail

Apache Axis ist grundsätzlich ein sogenannter SOAP-Prozessor, es kann also die Verarbeitung von SOAP-Nachrichten übernehmen. Axis 1.4 unterstützt die SOAP-Spezifikation in der Version 1.1. Serverseitig läuft Axis dabei als Servlet in einem Serlvet-Container und nimmt SOAP-Anfragen entgegen, ermittelt den Web-Service, den diese aufrufen wollen und leitet einen Java-Aufruf an den Web-Service weiter. Das Ergebnis, das vom Service zurückkommt, wird in eine SOAP-Nachricht übersetzt und an den Aufrufer zurückgegeben. Abbildung 5.1 zeigt einen entsprechenden Aufruf.

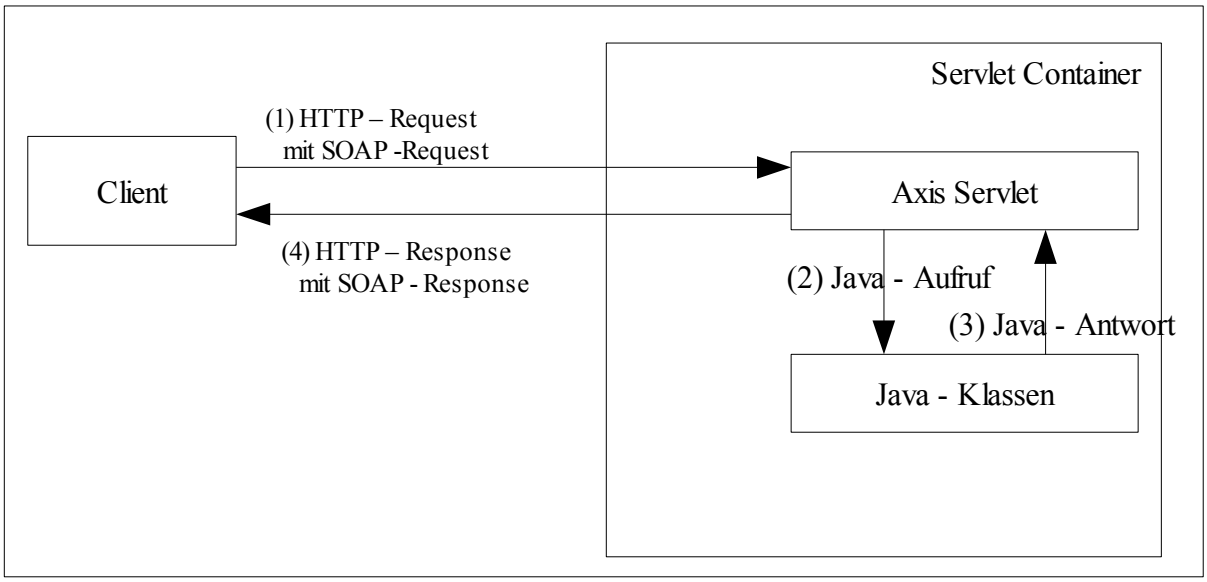

Abbildung 5.1 Ein SOAP-Request-Response-Zyklus mit Axis

Um in Apache Axis einen Web Service zu definieren, sind zwei Dinge nötig. Die Java-Klasse(n), die die Funktionalität des Web Services enthalten und ein Deployment-Deskriptor. Dieser wird mit dem Admin-Servlet von Axis aufgerufen und macht den Web-Service dem Axis-Servlet bekannt. Ein Deployment-Descriptor ist grundsätzlich wie folgt aufgebaut:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="MyWebService" provider="java:RPC">
    <parameter name="className" value="myPackage.myWebServiceClass" />
    <parameter name="allowedMethods" value="*" />
  </service>
  <typeMapping
   languageSpecificType="java:myPackage.myType"
   qname="ns1:myType"
   serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
   deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  />
</deployment>
```

Im <deployment>-Abschnitt werden die Namensräume festgelegt, die dieser Web-Service benötigt, zum einen, um zu kennzeichnen, dass es ein Deskriptor für einen Axis-Web-Service ist und zum anderen wird der Java-Namensraum festgelegt.
Im <service> - Abschnitt wird genau beschrieben, wie der Web Service erreichbar ist. Dies legt die

URI fest, über die dieser Web-Service erreichbar sein wird. Außerdem wird festgelegt, welcher Provider verwendet wird. Da mit Java gearbeitet wird und ein RemoteProcedureCall ausgeführt werden soll, ist dies der von Axis für Java-RemoteProcedureCall vorgesehene Provider Java:RPC.

Im <service> - Abschnitt werden noch weitere Parameter definiert: Der className gibt genau die Java-Klasse an, die die Methoden für diesen Web-Service zur Verfügung stellt. Der Parameter allowedMethods bestimmt, welche Methoden aus der im className-Tag angegebenen Klasse über den Web Service sichtbar gemacht werden sollen.

Ein optionaler Abschnitt ist der Abschnitt <typeMapping>. Dieser wird benötigt, wenn vom Programmierer selbst definierte Datentypen mit Hilfe von Axis verschickt werden sollen. Es muss dem Axis-Servlet bekannt sein, wie diese zu serialisieren sind. Axis liefert diverse Serialisierer mit, in diesem Beispiel wird der neue Datentyp „myType“ mit dem BeanSerializer von Axis (de-) serialisiert. Dies funktioniert nur, wenn „myType“ konform mit der JavaBean-Spezifikation ist.

Ist der Deployment-Descriptor fertiggestellt, wird der Service durch aufrufen des Deployment-Descriptors mit dem AdminClient des Axis-Frameworks am AxisServlet, das im Servlet-Container (zum Beispiel ein Tomcat-Server) läuft, registriert werden. Ist dies geschehen, steht die Service-Funktionalität über den Server zur Verfügung.

Weitere nützliche Tools sind die automatische Generierung der zum Web Service gehörenden WSDL-Datei. Diese ist standardmäßig über die URL des Web Services verfügbar, im oben genannten Fall zum Beispiel: „`http://192.168.0.1:8080/axis/services/MyWebService?wsdl`“
Mit Hilfe dieser WSDL-Datei lassen sich die für einen Client benötigten Proxyklassen erzeugen. Axis bietet hierzu das Werkzeug WSDL2Java an. Bei Aufruf des Tools mit einer WSDL-Datei werden alle benötigten Objekt-Klassen erzeugt, die bei Aufruf des Web Services übergeben oder zurückgegeben werden. Zusätzlich entstehen Stubs, die die Verbindungsdaten zum Axis-Server und damit zum Web Service beinhalten.

5.3 Einbindung von SOAP in das JCommSy

Es ist nun die Aufgabe, das Axis-Framework in das JCommSy einzubinden und als Resultat davon zu sehen, wie die neuen Web Service-Klassen in die bestehende Architektur des JCommSy eingepasst werden können.

Wie in Kapitel 2 beschrieben, bildet das JCommSy grundlegend eine 2-Schichten-Architektur oder auch Client/Server-Architektur (siehe [KPRR04, Seite 87f]). Im Besonderen realisiert das JCommSy auch die JSP-Model-2-Architektur (siehe [KPRR04, Seite 89f]), die an das JCommSy angepasst in Abbildung 5.2 dargestellt ist. Die Pfeile in dieser Abbildung zeigen den Kontrollfluss bei einer Anfrage an den Webserver durch einen Client.

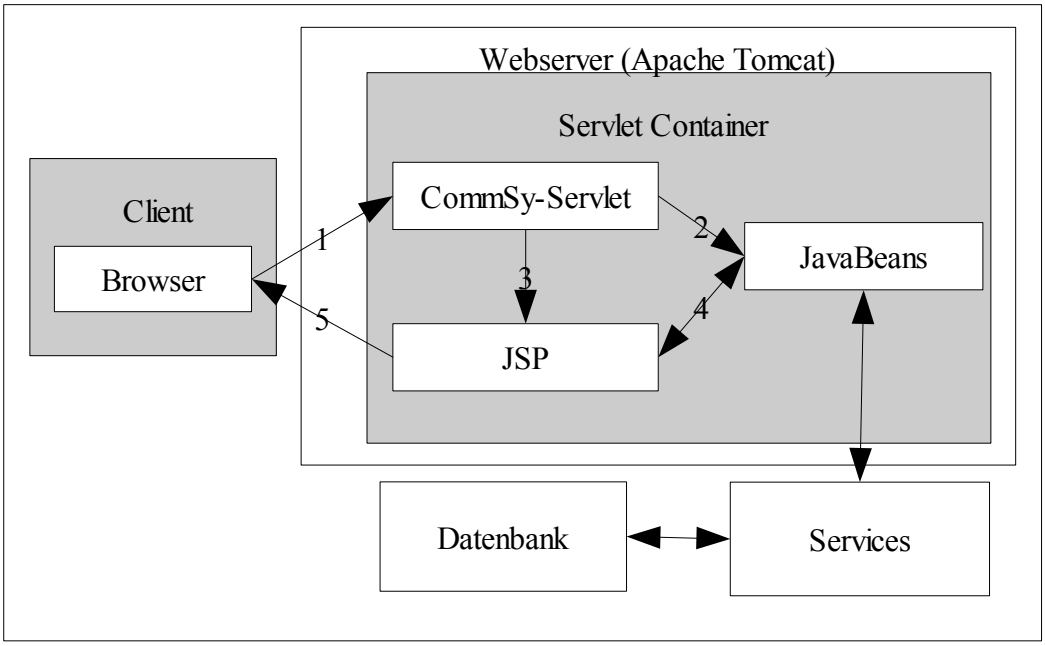

Abbildung 5.2 JSP-Model2-Architektur des JCommSy (nach [KPRR04, Seite 90])

Wie in Abbildung 5.2 dargestellt, besteht das JCommSy aus einem Servlet, dem CommSy-Servlet, das in einem Servlet-Container läuft, sowie einem auf Services aufbauenden Software-System. Der Servlet-Container ist hier ein Apache Tomcat. Die HTTP-Anfragen, die vom Client, also vom Browser kommen, werden an das CommSy-Servlet weitergeleitet. Dieses Servlet benutzt dann die Services, die das JCommSy-System anbietet, um die Anfrage zu beantworten. Als persistenter Datenspeicher liegt eine MySQL-Datenbank unter den Services. Der Datenaustausch zwischen Services und Datenbank wird mit Hilfe von Hibernate realisiert.

Da das Axis-Framework ebenfalls ein Servlet zur Verfügung stellt, das Axis-Servlet, muss dieses in den Servlet-Container, in dem auch das CommSy-Servlet läuft, eingebunden werden. Alle HTTP-Anfragen, die SOAP-Anfragen enthalten, werden an das Axis-Servlet delegiert und dort weiterverarbeitet. Das Axis-Servlet ruft dann die Web Service-Klassen auf, die im JCommSy-System bereitgestellt sein müssen. Nach Bearbeitung generiert das Axis-Servlet eine HTTP-Antwort, die eine SOAP-Antwort enthält.

Aus dem dargestellten Entwurf, wie die SOAP-Engine Axis in das JCommSy einzubinden ist, ergeben sich mehrere Punkte, die bei der Realisierung der Umsetzung zu beachten sind:

- Die neuen Web Service-Klassen bauen auf den bestehenden Services auf und benutzen deren Methoden, sie implementieren das Wrapper-Entwurfsmuster (siehe [GHJV95]). Sollten die Web Services Methoden und Dienste von den Services benötigen, die diese noch nicht bieten, weil die Migration vom PHP-CommSy zum JCommSy noch nicht weit genug fortgeschritten ist, müssen die Services um diese Funktionalitäten erweitert werden.

- Zu den Funktionalitäten der Services müssen die Web Services weitere Funktionalitäten implementieren, die im web-basierten JCommSy nicht durch die Services abgedeckt werden. Dies sind die Funktionalitäten, die mit Hilfe der Web-Oberfläche und den Servlets realisiert sind, zum Beispiel das Betreten eines Raumes durch einen Benutzer, das auf der Webseite durch einen Mausklick geschieht. Diese Funktionalitäten müssen identifiziert und in den Web Services abgebildet werden.

- Die Sicherheit der Kommunikation sowohl auf der Transportebene als auch auf der Anwendungsebene muss gewährleistet werden. Das SOAP-Protokoll bietet keine Verschlüsselungs- oder Signaturmechanismen, daher müssen diese auf anderem Wege implementiert werden.

5.4 Von LAssi benötigte Schnittstellen

Um eine konkrete Architektur entwerfen zu können, müssen zunächst die Anforderungen, denen die neue Architektur genügen muss, definiert werden. In diesem Fall werden die Anforderungen von LAssi definiert, da dieses System in dieser Arbeit als Beispiel für einen an das JCommSy anzuschließenden Rich-Client verwendet wird.

LAssi benötigt folgende Funktionalitäten:

- Authentifikation von Benutzern am JCommSy

- Auflisten aller CommSy-Räume, in denen der Benutzer Mitglied ist

- Betreten von Räumen, wenn Anmeldung am JCommSy erfolgt ist

- Auflistung aller Materialien und Gruppen in den Räumen, in denen der Benutzer Mitglied ist
  - Auflistung aller Materialien in einem Raum
  - Auflistung aller Mitglieder einer Gruppe

- Erzeugen, modifizieren und löschen von Materialien und Gruppen

- Anhängen von Dateien an Materialien

- Beitreten zu Gruppen bzw. austreten aus Gruppen

- Andere Benutzer zu Gruppen zuordnen

- Materialien zu Gruppen zuordnen bzw. Zuordnung entfernen

- Auflisten der zu einer Gruppe verlinkten Materialien

Die Anforderungen lassen sich in vier große Bereich zusammenfassen:

1. Userverwaltung

Da die Benutzeranmeldung bisher im JCommSy noch nicht vorhanden ist, muss diese implementiert werden. Da die Anmeldung am PHP-System über Sessions funktioniert ist es naheliegend, auch im JCommSy mit Sessions zu arbeiten. Um diese zu realisieren ist eine Instanz notwendig, die Anmeldungen entgegen nimmt, prüft, ob Benutzername und Passwort korrekt sind und bei Erfolg eine neue Session für den Benutzer erzeugt und diesem eine SessionID zurückgibt. Von dieser Instanz darf es im System nur eine geben, da es sonst möglich ist, dass die Prüfung, ob ein Benutzer am System angemeldet ist, fehlschlägt. Sie sollte also das Singleton-Pattern implementieren [GHJV95, Seite 127ff]. Bei jedem weiteren Methodenaufruf muss der Benutzer seine SessionID mit übergeben, damit der Session-Verwalter prüfen kann, ob er angemeldet ist.

2. Raum-und Gruppenverwaltung

Die Raum- und Gruppenverwaltung ist im JCommSy bereits vorhanden, es müssen also die Funktionen des JCommSy-Services über die Web Service-Schnittstelle sichtbar gemacht werden. Dies lässt sich über eine normale Klasse, die als Wrapper für die Funktionalitäten der Services fungiert und die Anfragen eines Clients an den Service durchgibt, realisieren.

3. Materialverwaltung

Auch die Materialverwaltung ist im JCommSy bereits vorhanden, zusätzlich muss hier eine Lösung gefunden werden, wie Dateien an das JCommSy geschickt und wieder zurückgeholt werden können. Da die Dateien für den Transport serialisiert werden müssen, muss ein Serialisierer vorhanden sein, der mit Dateien umgehen kann. Das Axis-Framework bietet eine Lösung dafür, da es mit DataHandler-Objekten aus dem Java-Activation-Framework umgehen kann. Diese DataHandler-Objekte können eine Datei aufnehmen und transportieren.

4. Erzeugen von Objekten

Es wäre direkt möglich, die JCommSy-Materialien wie zum Beispiel Material oder Gruppe über die Web Service-Schnittstelle zu übertragen, indem dem Axis-Servlet zunächst ein Serialisierer für jedes Objekt mitgeteilt wird, das nicht den bereits integrierten primitiven Objekttypen (zum Beispiel String, Integer, Array) entspricht. Es ist möglich für die einzelnen Objekte Serialisierer zu schreiben, dann müssten alle Objekte, von denen die zu serialisierenden Objekte erben oder die sie enthalten, mit serialisiert werden. Am Beispiel des Material-Objekts wird deutlich, dass dies ungünstig wäre.

Die Klasse Material erbt von der Klasse `ExtrasImpl`, die ihrerseits von der Klasse `ItemImpl` erbt. Diese beiden Klassen müssten neben der Materialklasse zusätzlich serialisiert werden, damit das Material übertragen werden kann. Dazu kommen die Fachwerte `Publicity`, `ItemID` und `ItemType` sowie die Klasse `IDBean`. Dies wären acht Objekte, für die Serialisierer vorhanden sein müssten, um das Material zu übertragen. Bei der Erzeugung der Client-Stubs durch das Axis-Framework werden zu allen Objekten, die auf die oben beschriebene Weise serialisiert werden, Klassen auf der Client-Seite erzeugt.

Um dies zu vermeiden, werden Web Service-Repräsentationen der JCommSy-Objekte benötigt, die eine JavaBean-Struktur haben, also neben einem Konstruktor nur get-und set-Methoden enthalten. Dies hat den Vorteil, dass das Axis-Framework einen Serialisierer für Beans mitliefert. Wird dann zum Beispiel ein Material vom Client angefordert, wird im Web Service das JCommSy-Objekt geholt und der Inhalt des Objektes wird an die Web Service-Repräsentation des Objektes weitergegeben und diese Web Service-Repräsentation wird an den Client übertragen.

Kapitel 6 Einbindung der Web Services in das JCommSy

In diesem Abschnitt wird beschrieben, wie die im vorangegangenen Kapitel entworfene Architektur in das bestehende JCommSy-System eingefügt wird. Zur Entwicklung der Software wurde als Entwicklungsumgebung Eclipse WebToolsPlatform in der Version 1.0.2 bzw. 1.5 (zu Eclipse siehe [Dau105]) eingesetzt. Als SOAP-Engine kam das in Kapitel 5 beschriebene Axis-Toolkit von Apache in der Version 1.4 zum Einsatz [AXIS1].

6.1 Das Ausgangssystem

Abbildung 2.3 zeigt eine Übersicht über die Packages des JCommSy-Systems. Interessant für die Implementierung der Web Services sind die Packages `service`, das alle JCommSy-Services enthält, deren Funktionalitäten durch die Web Services verfügbar gemacht werden sollen, und das Package `item`, das die JCommSy-Objekte enthält, die über die SOAP-Schnittstelle ausgetauscht werden sollen.

Der Architektur-Entwurf sieht vor, dass die Funktionalitäten des JCommSy gewrappt und über eine SOAP-Schnittstelle nach außen sichtbar gemacht werden. Die zugehörigen Klassen werden im Package `service` zu den bestehenden Subpackages `filter`, `hibernate`, `impl`, `mem`, `persistence` und `registry` in das neue Subpackage `webservice` eingefügt.

Neben den genannten Subpackages enthält das Package `service` die Interfaces für alle JCommSy-Services. Wichtig für die Web Services ist das Subpackage `registry`. Es enthält die Klasse `ServiceRegistry`. An dieser Klasse werden die Implementierungen der einzelnen JCommSy-Services registriert, sie muss also von den Web Services verwendet werden, um die Funktionen des JCommSy benutzen zu können.

6.2 Implementierung der Web Services

Um analog zur JCommSy-Architektur die einzelnen Rubriken getrennt zu halten wird zu jedem JCommSy-Service ein eigener Web Service geschrieben. Dazu wird für jeden gewünschten Web Service zunächst ein Interface geschrieben, das die zu implementierenden Funktionen enthält. Durch die von LAssi vorgegebenen Anforderungen werden folgende JCommSy-Services benötigt:

- MaterialService: Der Material-Service wird benötigt, um Materialien im JCommSy abzulegen und abzufragen.

- GroupService: Der Group-Service wird benötigt, um neue Gruppen anzulegen, bestehende Gruppen zu editieren und um Gruppen beizutreten bzw. sie zu verlassen. Außerdem ist Auskunft darüber möglich, in welchen Gruppen ein Benutzer in dem aktuellen Raum Mitglied ist.

- RoomService: Der Room-Service wird benötigt, um einen CommSy-Raum zu betreten und um anzuzeigen, in welchen Räumen ein Benutzer Mitglied ist.

- UserService: Der User-Service wird benötigt, um nähere Informationen zu Benutzern des JCommSy zu holen

Zu diesen vier Services werden Interface-Klassen im Package `webservice` erzeugt. Zusätzlich kommt ein Interface mit dem Namen `CommsyWebService` hinzu, das die Login- und Logout-Methoden zur Verfügung stellt. Von diesem Interface erben alle anderen Service-Interfaces, damit sich Benutzer über jeden Web Service am JCommSy anmelden können.

Um die Benutzeranmeldung realisieren zu können ist es notwendig, eine Instanz zu schaffen, die die angemeldeten Benutzer verwaltet. Hier heißt diese Klasse `SessionManager`. Da es im gesamten System immer nur eine Instanz dieser Klasse geben darf, implementiert diese das Singleton-Pattern [GHJV95, Seite 127ff]. Mit Hilfe dieser Klasse wird bei der Anmeldung eines Benutzers über einen Web Service eine Session angelegt, die eine eindeutige SessionID, den Benutzernamen, den aktuellen CommSy-Raum, in dem sich der Benutzer befindet und die BenutzerID enthält. Die SessionID wird über eine SessionIDFactory erzeugt, es muss dafür gesorgt werden, dass diese SessionID eindeutig ist. Die angesprochenen Klassen befinden sich alle im Package `webservice`, Abbildung 6.1 zeigt das zugehörige Klassendiagramm.

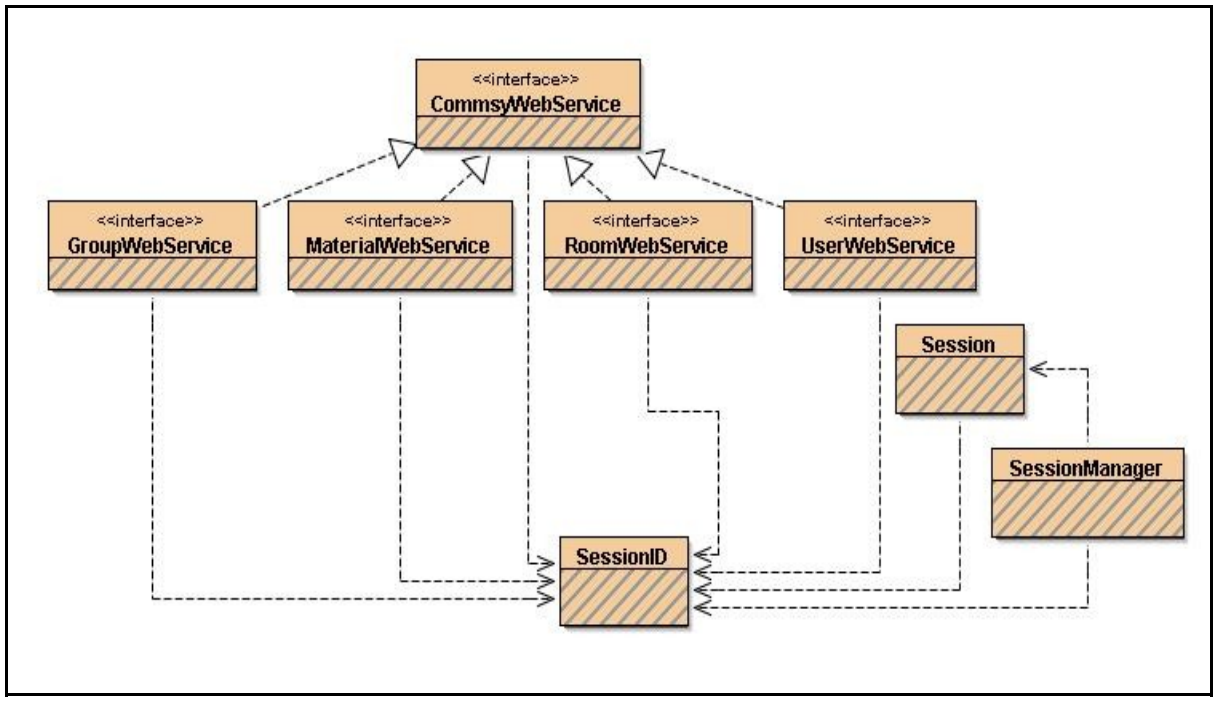

Abbildung 6.1 Klassendiagramm des `webservice`-Packages

Die eigentlichen Implementierungen der Web Services werden in einem weiteren Subpackage realisiert, im Package `impl`. Zu jedem Interface wird eine konkrete Klasse erzeugt, die jeweils den zugehörigen JCommSy-Service benutzt und dessen Funktionalitäten nach außen gibt. Abbildung 6.2 zeigt die neu hinzugekommen Klassen in Beziehung zu den oben beschriebenen Interfaces.

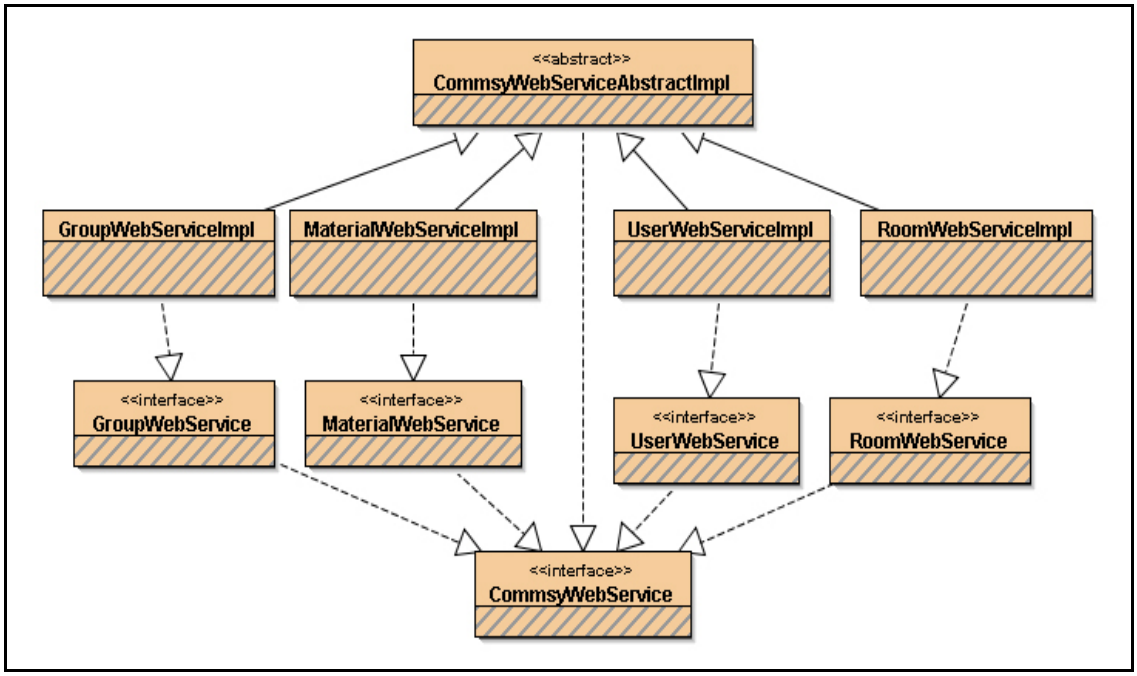

Abbildung 6.2 Klassendiagramm der Web Service-Implementierungen und der Interfaces

Das Klassendiagramm zeigt, dass die Implementierung des CommsyWebServices eine abstrakte Klasse ist. Dies ist der Fall, da es nicht erwünscht ist, dass diese Klasse instantiiert wird. Diese Klasse stellt den anderen Web Services lediglich die Implementierung der login- und der logout-Methode durch Vererbung zur Verfügung.

Die einzelnen Web Services stellen dabei folgende Funktionalitäten zur Verfügung, um die Anforderungen, die durch LAssi gestellt werden, zu erfüllen (siehe Abschnitt 5.3):

MaterialWebService:

- Anzeigen aller Materialien in einem CommSy-Raum
- Holen eines Materials mit einer bestimmten ID
- Anlegen eines neuen Materials
- Verändern eines Materials
- Löschen eines Materials
- Materialien zu Gruppen zuorden

GroupWebService:

- Anzeigen aller Gruppen in einem CommSy-Raum
- Benutzer kann Gruppen beitreten
- Benutzer kann Gruppen verlassen
- Anlegen von neuen Gruppen
- Verändern von Gruppen
- Löschen von Gruppen
- Andere Benutzer zu einer Gruppe hinzufügen

RoomWebService:

- Anzeigen aller CommSy-Räume, in denen ein Benutzer Mitglied ist
- Betreten von CommSy-Räumen

UserWebService:

- Anzeigen von Informationen über Benutzer

Die Web Service-Klassen implementieren dabei Methoden, die zur Erfüllung der oben angegebenen Funktionalitäten notwendig sind. Dies sind zum einen Methoden, die die JCommSy-Services direkt anbieten und die der Web Service an die Clients durchreicht. Zum anderen sind dies Funktionalitäten, die von den JCommSy-Services nicht abgedeckt werden, da sie auf der Web-Oberfläche und durch die Servlets abgedeckt werden. Die Funktion, über den RoomWebService einen CommSy-Raum zu betreten, ist beispielsweise eine solche Funktion. Um diese Funktionalitäten umzusetzen, werden die Web Service durch zusätzliche Methoden erweitert, die in den JCommSy-Services nicht vorliegen. Diese Methoden benutzen die JCommSy-Services, um ihre Funktion zu erfüllen. Das ausgewählte Beispiel des Betreten eines CommSy-Raumes wird gelöst, indem der in der Session abgelegte aktuelle Raum geändert wird, nachdem mit Hilfe des JCommSy-Services geprüft wurde, ob der Benutzer Mitglied des Raumes ist, den er betreten möchte.

Es gibt zwei Typen von Operationen, die durch die Web Services angeboten werden. Der erste Typ sind die Operationen, die einen Rückgabewert haben, je nach Web Service sind dies die get-Operationen für die Materialien, die der Web Service unterstützt. Hinzu kommt die login-Operation, die eine SessionID zurückgibt und die Operationen, die Mengen von IDs zurückgeben. Wird keine Rückgabewert benötigt (beispielsweise, wenn ein Benutzer einen Raum betritt oder einer Gruppe beitritt), wird nur ein „true“ zurückgegeben, um dem Client zu signalisieren, dass die Operation erfolgreich war. Dies ist der zweite Typ Operation.

Am Beispiel des MaterialWebService soll deutlich gemacht werden, wie die Web Services in das bestehende JCommSy-System integriert werden. In der implementierenden Klasse werden zwei private Variablen definiert, ein SessionManager und ein MaterialService. Die Singleton-Instanz des SessionManagers wird mit dem Befehl `SessionManager.getInstance()` geholt, der MaterialService des JCommSy mit dem Aufruf `ServiceRegistry.getMaterialService()`. Soll nun beispielsweise ein bestimmtes Material aus dem CommSy über den Web Service geholt werden, so verwendet der Web Service die entsprechende Methode der an das private Feld gebundenen Instanz des JCommSy-MaterialService.

Damit die genannten Funktionen zur Verfügung gestellt werden können, müssen die Repräsentationen für die CommSy-Materialien implementiert werden. Im JCommSy liegen diesen Materialien im Package `item`, alle Materialien erben direkt oder indirekt vom Interface `Item`, dass die Felder enthält, die alle Materialien wichtig sind. Dies sind zum Beispiel die ItemID, die ContextID, das CreationDate etc. In diesem Kontext werden die Materialien Group, Material, Room und User benötigt. Um diese für die Web Services verfügbar zu machen gibt es mehrere Möglichkeiten. Eine Möglichkeit ist, die bestehenden JCommSy-Materialien-Klassen zu verwenden und diese serialisierbar zu machen. Dazu ist es notwendig, Methoden in die Materialien-Klassen einzufügen, die die komplexen Fachwerte, die nicht serialisiert werden sollten, als primitive Werte zurückgeben. Gegen diese Möglichkeit spricht, dass zum einen alle Klassen, von eine Material-

Klasse erbt, auch serialisiert werden müssen (siehe auch Kapitel 4). Für diese Klassen wird analog zur JCommSy-Architektur ein Package `item` im Package `webservice` angelegt. Jedes benötigte Material wird als Klasse realisiert und durch ein Präfix „WS“ zu den JCommSy-Materialien abgegrenzt. Abbildung 6.3 zeigt das Package `item` mit den neuen Klassen.

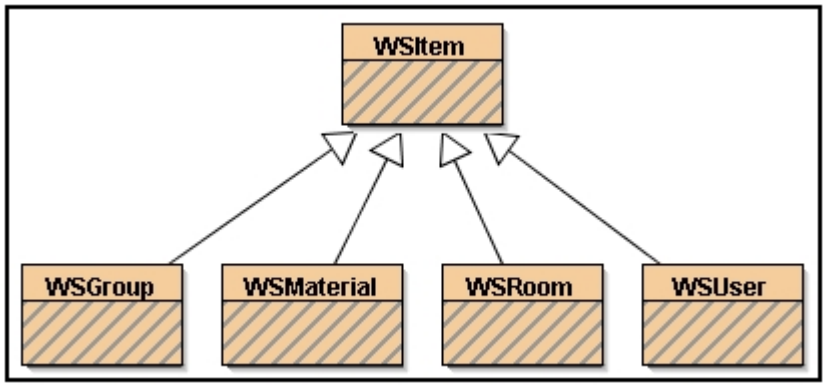

Abbildung 6.3 Klassendiagramm des `webservice.item`-Package

Die WS-Klassen entsprechen JavaBeans-Standard, sie enthalten also nur primitive Datentypen und die zugehörigen Get- und Set-Methoden. Da diese Klassen die Materialien des JCommSy repräsentieren sollen, werden die komplexeren Datentypen der JCommSy-Materialien in den Web Services in primitive Datentypen konvertiert und an die Web Service-Repräsentationen übergeben.

Die Oberklasse `WSItem` enthält die Attribute, die allen Materialien gemeinsam sind, analog zur Klasse `Item` im JCommSy-System.

6.3 Integrieren des Axis-Frameworks in das JCommSy

Bevor die im Abschnitt 6.2 beschriebenen Web Services verwendet werden können, wird das Axis-Framework in das JCommSy-System integriert. Dazu werden die von Axis benötigten Bibliotheken in das JCommSy-System eingefügt und die Datei `web.xml` angepasst. Diese XML-Datei beinhaltet alle Informationen über die Web-Applikation, insbesondere, welche Servlets vorhanden sind. Da der Kernteil des Axis-Frameworks ein Servlet ist, wird dieses in die Datei `web.xml` eingetragen und so dem Servlet-Container (einem Tomcat-Server) bekannt gemacht. Ist dies geschehen, kann auf das Axis-Servlet zugegriffen werden.

Um die Web-Services bekannt zu machen, bekommt jeder Web Service einen Deployment-Descriptor. Dieser enthält die konkrete Klasse, die aufgerufen werden muss, wenn der Web Service aufgerufen wird und den Namen des Web Services, wie ihn ein Client zu sehen bekommt. Der GroupWebService wird am AxisServlet unter dem Pfad „`CommSyGroupService`“ registriert und ist somit unter der URL „`http://<commsyserver>:8080/commsy/CommSyGroupService`“ erreich- und benutzbar.

Außerdem enthält der Descriptor Informationen darüber, welche Objekte bzw. Klassen über diesen Web Service transportiert werden können. Es wird festgelegt, wie Objekte für den Transport serialisiert und beim Empfang deserialisiert werden können. In diesem Fall wird als Serialisierer für alle zu übertragenden Materialien der von Axis mitgelieferte BeanSerializer verwendet, da alle Materialien einen Aufbau wie eine JavaBean haben (siehe Abschnitt 6.2).

6.4 Exception-Handling

Im Falle von Fehlern im Server selbst oder einer fehlerhaften Anfrage an der Web Service ist es nötig, dem Client einen entsprechenden Mechanismus zur Fehlerbehandlung zur Verfügung zu stellen. Um dies zu realisieren, wurde die Klasse `SOAPException` implementiert und ins `webservice`-Package eingefügt. Diese Klasse erbt von der Klasse `AxisFault`, die vom Axis-Framework geliefert wird. `AxisFault` sorgt dafür, dass Fehler in eine SOAP-Message umgewandelt und so an der Client geschickt werden können. Die Klasse `SOAPException` übernimmt im derzeitigen Stadium einen String und sendet diesen an den Client. Ein Fehlerfall ist zum Beispiel die Anmeldung einer Benutzers mit einem falschen Passwort. Dies gibt derzeit eine `SOAPException` mit der Message „Falsches Passwort“ an den Client zurück. Angepasste Meldungen gibt es für Server-Fehler und weitere mögliche Fehler.

6.5 Kommunikation über die Web Service-Schnittstelle

Um zu demonstrieren, wie die Kommunikation zwischen einem Client und dem JCommSy über die Web Service-Schnittstelle abläuft, werden zwei Szenarien an Hand von Interaktionsdiagrammen dargestellt und diskutiert.

Im ersten Szenario wird angenommen, dass ein Client nach Anmeldung am JCommSy einen CommSy-Raum betritt und dort ein neues Material anlegt. Abbildung 6.4 zeigt das zugehörige Interaktionsdiagramm. Es wird für dieses Diagramm angenommen, dass kein Fehlerfall auftritt.

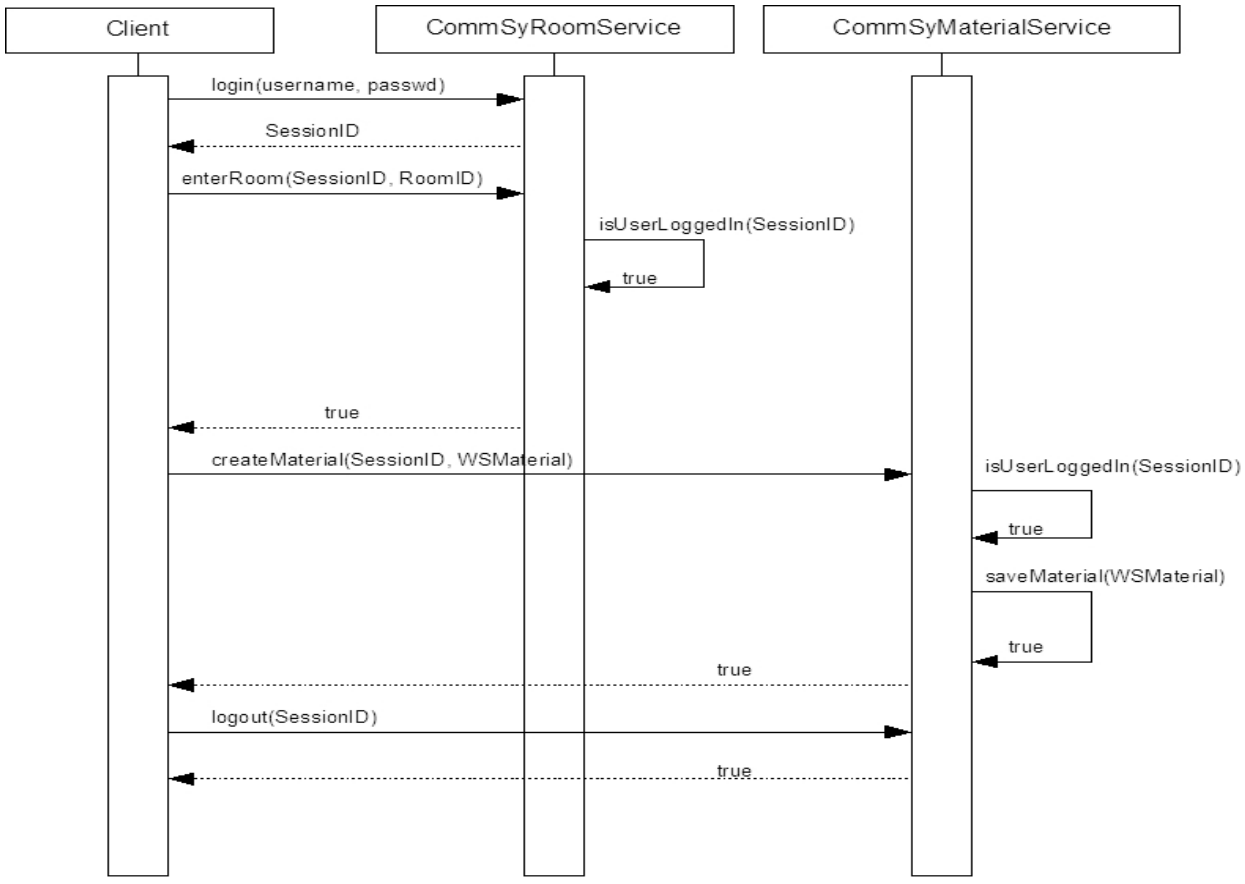

Abbildung 6.4 Interaktionsdiagramm zur Erstellung eines neuen Materials

Ein zweites Beispiel zeigt die Anmeldung des Benutzers am CommSy und den Beitritt zu einer bestehenden Gruppe in einem CommSy-Raum. Abbildung 6.5 zeigt das Interaktionsdiagramm.

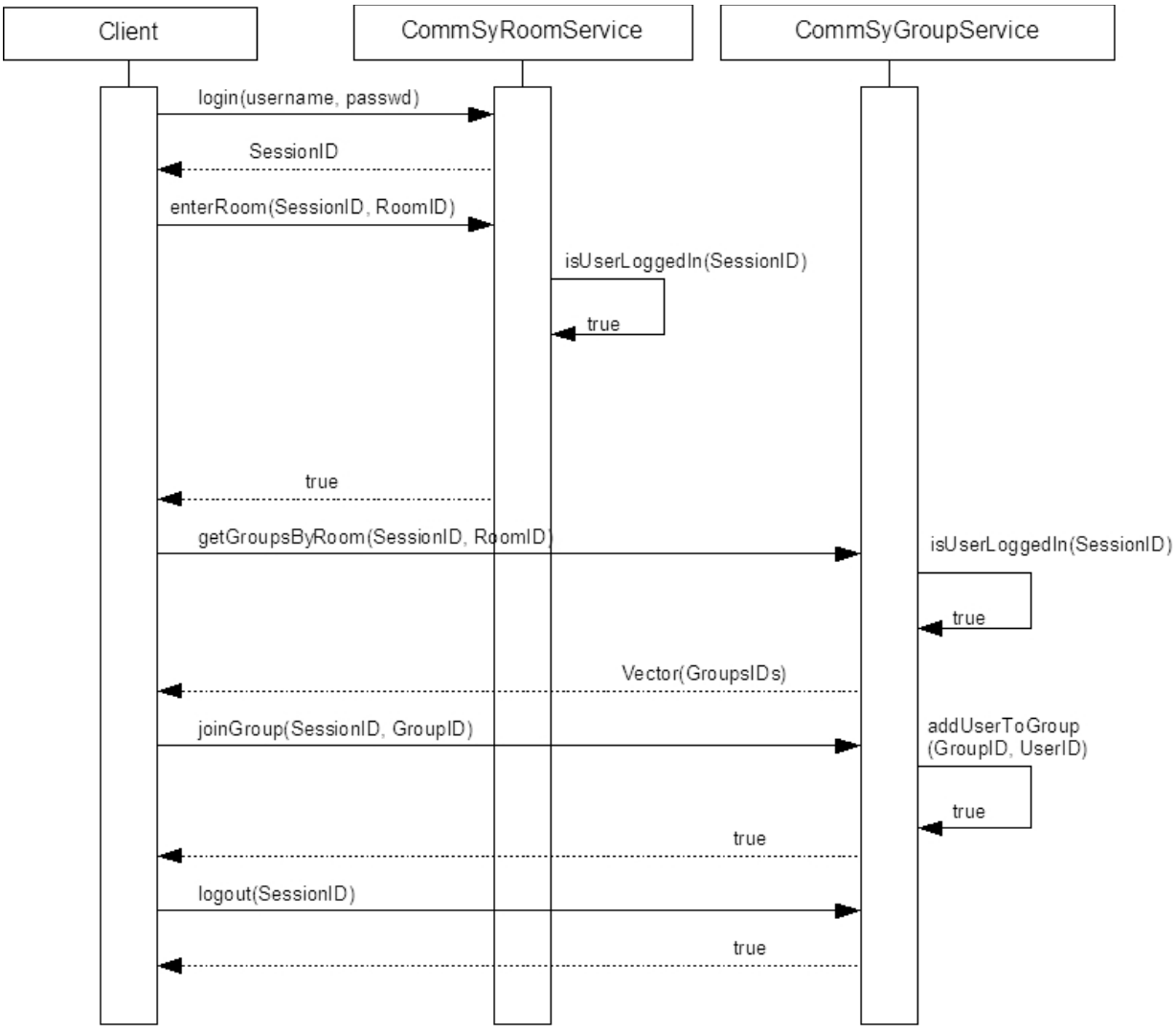

Abbildung 6.5 Interaktionsdiagramm zum Beitritt zu einer Gruppe

Die Interaktionsdiagramme zeigen, dass die Kommunikation mit jedem Web Service, den das CommSy anbietet, gleichförmig abläuft. Die An- und Abmeldung ist an jedem Web Service möglich. Ist der Nutzer einmal authentifiziert, muss er dies bei jedem Aufruf durch mitsenden seiner SessionID nachweisen. Diese SessionID wird beim Abmelden vom System gelöscht.

6.6 Hinzufügen weiterer Web Services

Um weitere Funktionalitäten des JCommSy über die Web Service-Schnittstelle verfügbar zu machen, sind mehrere Schritte erforderlich:

Es gibt grundsätzlich zwei Szenarien, die auftreten können. Entweder kommt zu einem bereits als Web Service verfügbar gemachten Service weitere Funktionalität hinzu. Dann muss diese in die betreffende Web Service-Klasse eingefügt werden. Oder es soll ein Service, der noch nicht als Web Service realisiert ist, verfügbar gemacht werden. Dann wird wie folgt vorgegangen:

Es wird geprüft, ob mit dem neuen Service CommSy-Materialien über die Web Service-Schnittstelle übergeben werden sollen, die bisher noch nicht als Web Service-Implementierung vorhanden sind. Ist dies der Fall, muss eine Web Service-Repräsentation des betreffenden Materials erstellt werden, das von der Klasse `WSItem` erbt. Ist dies geschehen, kann die Web Service-Klasse analog zu den bereits vorhanden implementiert werden. Diese Klasse muss am Axis-Servlet registriert werden. Dies geschieht durch einen Eintrag in der Datei `server-config.wsdd`, die sich im `WebContent\WEB-INF`-Unterverzeichnis des JCommSy-Projektes befindet. Der Eintrag muss diesem Schema folgen (im Beispiel der Eintrag für den CommSyUserWebService):

```
<service name="CommsyUserService" provider="java:RPC">
  <parameter name="allowedMethods" value="*"/>
  <parameter name="className"
      value="de.unihamburg.informatik.jcommsy.service.webservice.impl.
             UserWebServiceImpl"
  />
  <beanMapping qname="ns:SessionID"
      xmlns:ns="http://localhost:8080/commsy/CommsyUserService"
      languageSpecificType="java:de.unihamburg.informatik.jcommsy.service.webser
                            vice.SessionID"/>
  <beanMapping qname="ns:SOAPException"
      xmlns:ns="http://localhost:8080/commsy/CommsyUserService"
      languageSpecificType="java:de.unihamburg.informatik.jcommsy.service.webser
                             vice.SOAPException"
  />
  <beanMapping qname="ns:WSItem"
      xmlns:ns="http://localhost:8080/commsy/CommsyUserService"
      languageSpecificType="java:de.unihamburg.informatik.jcommsy.service.webser
                             vice.item.WSItem"
  />
  <beanMapping qname="ns:WSUser"
      xmlns:ns="http://localhost:8080/commsy/CommsyUserService"
      languageSpecificType="java:de.unihamburg.informatik.jcommsy.service.webser
                             vice.item.WSUser"
  />
 </service>
```

Ist diese Registrierung erfolgt, muss der Server neu gestartet werden, um den Service zu veröffentlichen. Über die URL „`http://<CommSy-Server>:8080/commsy/<ServiceName>?wsdl`“ ist dann die Web Service-Beschreibung zugreifbar.

Um mit einem Client-System auf die Web Services zuzugreifen wird auf dem Client zunächst ein SOAP-Implementierung für die verwendete Programmiersprache benötigt. Ist das Client-System ein Java-basiertes System sollte Apache Axis verwendet werden. Mit Axis können aus der Beschreibung eines Web Services, die über die oben genannte URL zugänglich ist, die für die Kommunikation nötigen Klassen und Stubs erzeugt werden. Der Client-Entwickler kann diese Stubs direkt benutzen. Diese Vorgehensweise funktioniert mit jeder Programmier-Sprache, für die es SOAP-Implementierungen gibt.

Kapitel 7 Fazit

Dieses Kapitel fasst die Ergebnisse dieser Diplomarbeit zusammen und bewertet sie in Hinblick auf die in Kapitel 1 diskutierten Problemstellungen. Zudem wird diskutiert, welche Defizite die jetzige Implementierung hat und wie die gelöst werden können. Abschließend wird ein Ausblick gegeben, welche Möglichkeiten die Web Service-Erweiterung des JCommSy in Zukunft bietet.

Die allgemeine Aufgabe dieser Arbeit war es, zu prüfen, ob es möglich ist, eine Web Service-Schnittstelle für das CommSy bzw. das JCommSy zu schreiben. Die Motivation war es, ein Rich-Client-System, in diesem Fall die Lernsoftware LAssi mit dem CommSy zu verbinden.

Um das Ergebnis dieser Arbeit beurteilen zu können, werden an dieser Stelle die bereits in Kapitel 1 beschriebene Problemstellungen wiederholt. Es galt, folgende Probleme in Hinblick auf die Anbindung von Rich-Client-Systemen an das JCommSy zu lösen:

1. Problem

Die von CommSy zur Verfügung gestellten Dienste (Services) sind nach Außen im derzeitigen System nicht sichtbar. Es muss die Frage beantwortet werden, wie diese Funktionalitäten für andere Clients sichtbar und benutzbar gemacht werden.

2. Problem

Es muss eine Möglichkeit gefunden werden, die Benutzer aus anderen Clients am JCommSy anzumelden. Hier ist auch das Problem zu lösen, wie der Anmeldestatus mit den Clients synchronisiert werden kann.

3. Problem

Es muss geklärt werden, wie die Daten, die ein Client über das CommSy anderen Clients zur Verfügung stellt, abgespeichert synchronisiert und wieder abgerufen werden können. Dies ist besonders im Hinblick auf LAssi wichtig, da die Benutzung von verschiedenen Karteikarten auf einem gemeinsamen Desktop unter Umständen ständige Synchronisation erfordert.

Um diese Probleme zu lösen, wurden zunächst die betreffenden Systeme (J)CommSy und LAssi vorgestellt. Anschließend wurden die Technologien vorgestellt, die für die Umsetzung der Schnittstelle nötig waren. Der Fokus lag auf den Technologien XML-RPC und SOAP, die beide als Protokoll für den späteren Web Service in Frage kamen. Auf Basis der Anforderungen, die LAssi an eine Schnittstelle stellt, wurde SOAP ausgewählt. Verwendet wurde nach Diskussion mehrerer SOAP-Implementierungen das Apache Axis-Framework.

Im nächsten Schritt wurde die Architektur entworfen und dargestellt, wie die Web Service-Schnittstelle aussehen und in das JCommSy integriert werden kann. Dabei fiel auch die Entscheidung, für jeden JCommSy-Service einen einzelnen Web Service zu schreiben, der die Funktionalitäten des JCommSy-Service wrappt und über die Web Service-Schnittstelle zur Verfügung stellt. Des weiteren wurde beschrieben, dass die Materialien, die im JCommSy verarbeitet werden, aus Aufwandsgründen eine Web Service-Repräsentation durch eine jeweils eigene Klasse erhalten würden. Dies war die Lösung für Problem eins.

Problem 2 wurde durch die Implementierung eines SessionManagers gelöst, der für jeden angemeldeten Benutzer eine Session speichert, die neben der CommSy-Kennung eine eindeutige SessionID enthält. Der Benutzer muss bei jeder Anfrage an das System über die Web Service-Schnittstelle seine SessionID übergeben und nachzuweisen, dass er aktuell angemeldet ist.

Das dritte Problem wurde gelöst, indem die Daten durch den Material Web Service an einem Ort auf der Festplatte des Servers abgelegt werden. So kann der Client ständig auf die abgespeicherten Daten, die an JCommSy-Materials angehängt sind, zugreifen. Dies ist eine vorübergehende Lösung, bis im JCommSy ein Service angelegt ist, der die Dateien an einem geeigneten Ort ablegt.

Damit wurde das konkrete Problem, LAssi über eine Web Service mit dem JCommSy zu verbinden, erfolgreich gelöst. Es wurde somit auch gezeigt, dass es möglich ist, allgemeine Client-Systeme, die Web Services unterstützen, mit dem JCommSy zu verbinden. Diese Möglichkeit ist gegeben, da die Beschreibung der vom JCommSy zur Verfügung gestellten Web Services öffentlich zugänglich ist und in der standardisierten Web Service-Beschreibungssprache WSDL vorliegt. Jeder Client kann auf Basis dieser Beschreibung die nötigen Softwarebausteine erzeugen, um mit dem JCommSy über den Web Service kommunizieren zu können. Es ist dabei unabhängig, welche Programmiersprache das Client-System verwendet. Für einen großen Teil von Sprachen liegen SOAP-Implementierungen vor, die ähnlich wie Apache Axis für Java automatisch aus der WSDL-Beschreibung die nötigen Stubs auf der Client-Seite erzeugen. Es gibt Implementierungen für C, C++, Delphi, Perl, PHP, Python, Ruby, tcl, Smalltalk und Microsofts .NET-Plattform.

Die Architektur des JCommSy wurde durch die Einbindung der Web Services nicht verändert, die Web Services sind zusätzlich an das System angeschlossen worden. Dies zeigt, dass die Architektur des JCommSy flexibel genug ist, um mit Web Services umzugehen. Die Modularität des JCommSy bleibt erhalten. Es ist mit den Web Services möglich, zusätzliche Rubriken zum JCommSy hinzuzufügen und diese über die Web Service-Schnittstelle für die meisten Client-Systeme zugreifbar zu machen.

Das JCommSy ist somit durch die Erweiterung um die Web Services-Schnittstelle für die meisten Client-Systeme offen. Die derzeitige Implementierung hat aber Grenzen, es sind weitere Dinge zu tun, um die Web Service-Schnittstelle für den aktiven Betrieb freigeben zu können.

Im jetzigen Entwicklungsstadium werden die Nachrichten, die das JCommSy über die Web Service-Schnittstellen mit seinen Clients austauscht im Klartext über das HTTP-Protokoll transportiert. Dies ist eine zwar funktionierende, aber keine sichere Lösung. Um das System abzusichern gibt es mehrere Möglichkeiten. Man kann den Transport selbst sicher machen, also beispielsweise statt HTTP das HTTPS-Protokoll verwenden. Dies wäre aber nur eine Teillösung des Problems, da die Knoten, die zwischen Sender und Empfänger liegen, den Inhalt der transportierten Nachrichten nach wie vor lesen können. Es ist also notwendig, die SOAP-Nachrichten, die ausgetauscht werden, zu verschlüsseln. Eine Möglichkeit bietet Apaches WSS4J (WebServicesSecurity for Java)-Toolkit, das neben Verschlüsselung auch Möglichkeiten zur Signierung der Nachrichten bietet [WSS4J]. In diesem Zusammenhang kann es auch sinnvoll sein, eine Migration von Apache Axis 1.4 auf Apache Axis2 vorzunehmen. Diese neue Version des Axis-Frameworks bietet einige Verbesserungen auf der Performance-Seite und bindet zudem das WSS4J-Toolkit direkt in das Framework mit ein. Dies sollte es vereinfachen, die Kommunikation zu sichern.

Eine weitere Möglichkeit bietet die von Komathy, Vivekanandan, Ramachandran beschriebene Idee zur Absicherung von SOAP-basierten Web Services [siehe [KVR03]).

Bisher sind nur vier der JCommSy-Services als Web Service implementiert worden, nämlich nur jene, die für die LAssi-Anbindung erforderlich waren. Ein Grund dafür ist, dass einige CommSy-Rubriken noch nicht nach Java migriert wurden und daher auch die entsprechenden Services noch nicht vorhanden sind. Um eine volle Schnittstelle zu gewährleisten kann es notwendig werden, zu jedem Service, der im JCommSy neu entsteht, einen Web Service zu implementieren.

Literatur

[AXIS1] *Apache Axis 1.4 Homepage,* besucht am 09.09.2006

[BD04] Brügge, B.; Dutoit, A.H., *Objektorientierte Softwaretechnik*, Pearson Education Deutschland, 2004

[Cer02] Cerami E., *Web Services Essentials*, O'Reilly, 2002

[Coy02] Coyle, F., *Web Services and the Data Revolution*, Pearson Education, 2002

[Dau05] Daum, B., *Java-Entwicklung mit Eclipse*, dPunkt Verlag, 2005

[Dau105] Daum, B., *Rich-Client-Entwicklung mit Eclipse*, dPunkt Verlag, 2005

[DLWZ03] Dumke, R.; Lother, M.; Wille, C.; Zbrug, *Web Engineering*, Pearson Studium, 2003

[Eng02] Englander, R., *Java and SOAP*, O'Reilly, 2002

[GHJV95] Gamma E.; Helm, R.; Johnson R.; Vlissides, J., *Design Patterns*, Addison-Wesley, 1995

[Jab04] Jablonski, S., *Guide to Web application and platform architectures*, Springer, 2004

[KPRR04] Kappel, G.; Pröll, B.; Reich, S.; Retschitzegger, W., *Web Engineering*, dPunkt 2004

[KVR03] Komathy, K.; Vivekanandan, P.;Ramachandran, V.; *Secure SOAP-based web services,* in: International journal of computer systems science & engineering 01/03 Seiten 27-34, London 2003

[KW02] Kuschke, M.; Wölfel, L., *Web Services Kompakt*, Spektrum Akad. Verlag, 2002

[Lan04] Langner, T., *Web-basierte Anwendungsentwicklung*, Spektrum Akad. Verlag, 2004

[MMST03] McGovern, J.; Mathew, S.; Stevens, M.; Tyagi, S., *Java Web Services Architecture*, Elsevier Science, 2003

[Oel01] Oellerman, W., *Architecting Web Services*, Springer Verlag, 2001

[See02] Seely, S., SOAP, *Cross Platform Web Service Development using XML*, Prentice Hall, 2002

[SOAP] *SOAP 1.2 Spezifikation,* http://www.w3c.org/TR/soap, besucht am 09.09.2006

[WSDL] *WSDL Spezifikation*, http://www.w3c.org/TR/wsdl, besucht am 09.09.2006

[WSS4J] WSS4J Homepage, http://ws.apache.org/wss4j, besucht am 09.09.2006

[XML] *XML Spezifikation,* http://www.w3c.org/TR/xml, besucht am 09.09.2006

[XML-RPC] *XMP-RPC Spezifikation,* http://www.xmlrpc.com/spec, besucht am 09.09.2006

[Z+98] Zuellighoven, H. et al.; *Das objektorientierte Konstruktionshandbuch*, dPunkt Verlag, 1998