Integration von PolyPhone in myETH

Semesterarbeit

Gianmatteo Costanza
<costanzg@student.ethz.ch>

Prof. Dr. Bernhard Plattner
Dr. Michele De Lorenzi, Marcel Baur

ETH World
Technisches Institut für Informatik
Department Informationstechnologie und Elektrotechnik

8. November 2005

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Abstract

An der ETH laufen unterschiedliche Projekte, die von ETH World und den Informatikdiensten gefördert werden, mit dem Ziel, den Angehörigen der Hochschule neue Informations- und Kommunikationsdienstleistungen anzubieten. PolyPhone ist eines dieser Projekte und bietet Audio/Video-Diensten wie auch Kurzmitteilungen (Instant Messages) und Präsenz an, die auf dem Signalisierungsprotokoll SIP basieren. Das Hauptziel dieser Semesterarbeit ist die Integration einer Auswahl von PolyPhone-Diensten in myETH, einem Channel-basierten Portal der ETH Zürich. Der gewählte Ansatz ist die Entwicklung eines nativen Channels, der in Java geschrieben ist, verbunden mit dem Einsatz von AJAX, was die Auffrischung der Benutzeroberfläche mittels asynchronen Übermittlungen von XML-Datenströmen erlaubt.

Different projects at the Swiss Federal Institute of Technology, sponsored by ETH World and the Computer Services, aim to improve communication and interaction among users. PolyPhone is one of these projects with this idea in mind. It provides audio/video-services, instant messaging and presence notification based on the Session Initiation Protocol SIP. The main objective of this semester thesis is the integration of a selection of PolyPhone services into myETH, a channel-based community portal of ETH Zürich. The approach chosen is to write a native channel in Java and the use of AJAX, that allows for updating the user interface by asynchronously transmitting XML datastreams between the server and the client.

Danksagung

Diese Semesterarbeit wäre nicht zustande gekommen ohne die Hilfe von Yves Serrano und dem restlichen myETH-Team sowie Francesco Piovano und den Informatikdiensten, die während zahlreichen Stunden Logfiles durchforstet und mit mir zusammen Bugs gejagt haben.
Besonderen Dank möchte ich meinen Betreuern Michele De Lorenzi und Marcel Baur aussprechen für die zahlreichen guten Tipps und die stete Unterstützung bei meinen Problemen.

Inhaltsverzeichnis

1 Einführung

1.1 Ausgangslage

An der ETH Zürich laufen unterschiedliche Projekte[1] mit dem Ziel, den Angehörigen der ETH Zürich neue Informations- und Kommunikationsdienstleistungen anzubieten. Durch eine Integration dieser Dienstleistungen kann ein Mehrwert für die Benutzerinnen und Benutzer erzielt werden. Diese Semesterarbeit hat als Hauptziel eine Integration von PolyPhone in myETH.

Das Projekt PolyPhone entwickelt Voice, Video, Präsenz und Instant-Messaging-Dienstleistungen auf Basis des SIP-Protokolls [14]. SIP ist ein Signalisierungsprotokoll, das in vielen Belangen an HTTP angelehnt ist und zum Aufbau von Multimedia-Sitzungen in IP-Netzwerken verwendet wird. Alternative Protokolle sind H.323 oder Skype, jedoch bietet SIP den Vorteil, dass es ein offener Standard ist, der breiten Anklang findet und auch in Hardware-VoIP-Telefonen benutzt wird.
Die Dienstleistungen von Polyphone richten sich an Studierende und Mitarbeitende der ETH und sollen nach einer angemessenen Pilotphase in betriebliche Strukturen überführt werden.

myETH ist das Portal der ETH Zürich und eröffnet den Benutzenden den Zugang zu webbasierten Informationen und Diensten der Hochschule. Benutzerinnen und Benutzer können sich die Inhalte nach ihren individuellen Bedürfnissen zusammenstellen und ihre Darstellung konfigurieren. myETH richtet sich an alle Mitarbeitenden, Forschenden und Studierenden der ETH Zürich.

[1] Für eine ausführliche Übersicht sei auf den folgenden Link der Homepage von ETH World verwiesen: `http://www.ethworld.ethz.ch/technologies/index`

1.2 Projektziele

Ausgewählte Funktionen von PolyPhone sollen für das Portal myETH in Form von *Channels* nutzbar gemacht werden. Damit soll erreicht werden, dass Benutzerinnen und Benutzer von PolyPhone möglichst viele Dienste via Web nutzen können, auch ohne einen eigenen Rechner mit SIP-Client zur Verfügung zu haben. Mögliche Funktionen stehen dabei zur Auswahl:

- **Instant Messaging (IM):** Versenden und Empfangen von IM innerhalb von PolyPhone
- **Präsenz:** Anzeige und Änderung von Präsenzinformationen von PolyPhone
- **Audio:** Telefonanrufe entgegen nehmen und ausführen (nur Audio)

Auf Video wird verzichtet, da nicht davon ausgegangen werden kann, dass öffentlich zugängliche Rechner mit einem Web Browser über eine Kamera verfügen.
Nach Abschluss der Semesterarbeit soll mindestens Instant Messaging in myETH verfügbar sein. Ferner soll ein Konzept vorgestellt werden, wie die anderen Funktionen allenfalls späteren hinzugefügt werden können.

1.3 Funktionalität

Authentisierung und Autorisierung

- Automatische Anmeldung mit n.ethz-Benutzername
- Speicherung von Einstellungen und Präferenzen

Instant Messaging

- Verschicken von IM (der SMS-Channel kann als Vorlage benutzt werden)
- Empfang von IM: Mechanismen für die laufende Anzeige von neuen Meldungen sollen, soweit in myETH möglich, implementiert werden. Die Benutzerschnittstelle soll ähnliche Funktionalitäten anbieten wie herkömmliche IM-Tools
- Hinzufügen, Ändern und Löschen von Kommunikationspartnern
- Suchfunktion nach Name und E-Mail-Adresse der n.ethz-Benutzenden

Präsenz

- Integration von Präsenzstatus (anwesend, abwesend, besetzt) für die angemeldete Person als Channel (bzw. Integration in IM- oder Audio-Channel)
- Abfrage von (freigegebenen) Präsenzinformationen anderer PolyPhone-Benutzer

Audio

- Integration eines Web-Client für SIP-Telefonie als Channel

2 Anforderungsanalyse

2.1 Vorgehensweise

Die Arbeit ist in folgende Schritte (Meilensteine) aufgeteilt mit anschliessender interner Präsentation:

Anforderungsanalyse

- Aufnahme der Anforderung der verschiedenen Stakeholder (myETH, PolyPhone, ETH World)
- Übersicht der technischen Infrastruktur der ETH Zürich
- Marktübersicht möglicher Plattformen

Konzept

- Technisches Konzept
- Betriebliches Konzept

Implementierung

- Umsetzung des Konzeptes
- Dokumentation der Resultate und Anleitung für den Betrieb bzw. für die Erweiterung der Infrastruktur
- Demonstration der funktionierenden Installation

2.2 Stakeholder

In diesem Projekt sind viele Parteien involviert, die einen spezifischen Aufgabenbereich abdecken. Die Koordination mit diesen Einheiten an der ETH ist eine grundlegende Voraussetzung für eine tragfähige Entwicklung eines PolyPhone-Channels.

2.2.1 ETH World

ETH World [2] ist ein Programm zur Entwicklung und Einführung von Technologien für die Kommunikation und Kooperation unabhängig von Zeit und Ort. Das Programm unterstützt alle ETH-Angehörigen in ihren Kernaufgaben – Lehren, Lernen, Forschen und den dazugehörenden Managementleistungen.
Sie trägt die Projektverantwortung für diese Semesterarbeit und dient als Initiator, ganz im Sinne seines Auftrages.

2.2.2 myETH: Channel-basiertes Portal

Das Portal myETH [1] versteht sich als Aggregator von Informationen von innerhalb und ausserhalb der ETH, die dem Benutzer in Form von Channels zur Verfügung gestellt werden. Die Applikation basiert auf uPortal [4] und wurde innerhalb der ETH erweitert. Die Entwicklung konzentriert sich auf das Anpassen und Erweitern von Channel-Angeboten. Die beliebtesten Angebote sind gemäss Aussagen des Projektleiters die RSS-Feeds, Bibliothekssuche und der SMS-Channel.

2.2.3 Informatikdienste: Betreiber der PolyPhone-Infrastruktur

Das Projekt PolyPhone[3] entwickelt Voice, Video, Präsenz und Instant-Messaging-Dienstleistungen auf Basis von SIP [14]. Zur Zeit befindet es sich in der Entwicklungsphase[1], und die Informatikdienste der ETH sind zum jetzigen Zeitpunkt dabei, diese Dienstleistungen in betriebliche Strukturen überzuführen.

2.3 Anforderungen

Damit PolyPhone angemessen in myETH eingebunden werden kann, wurden bei den Stakeholder in Form von Interviews die Anforderungen aufgenommen. Diese Anforderungen sind hier kurz zusammengefasst.

2.3.1 Allgemeine Anforderungen

Im Allgemeinen muss der Channel so gestaltet werden, dass der Benutzer möglichst einfach die PolyPhone-Dienste nutzen kann. Dazu gehören:

[1] ETH World hat im Rahmen der technologischen Exploration einen Prototypen *sipETH* entwickelt (siehe `http://www.ethworld.ethz.ch/technologies/sipeth`)

- Automatische Authentifizierung und Autorisierung über den n.ethz-Account
- Plattformunabhängigkeit (Java, HTML)
- Automatische Konfiguration (eigene SIP-Identität, Wahl des SIP Proxy Servers)
- Registrierung beim SIP Registrar

2.3.2 Instant Messenger

Der Instant Messenger muss über die Basisfunktionalität verfügen, die heutzutage von einer solchen Applikation vorausgesetzt werden:

- Versenden und Empfangen von Nachrichten (Instant Messages)
- Editierbare Liste von Gesprächspartnern (Buddyliste)
- Signalisation (graphisch, akustisch) von ankommenden Nachrichten in der Benutzeroberfläche

2.3.3 Presence

Der Gebrauch von Präsenzinformationen bedingt, dass folgende Funktionalität gegeben sein muss:

- Setzen des eigenen Präsenzstatus {*online, offline, busy, away*}
- Anzeigen des Präsenzstatus von ausgewählten Kommunikationsteilnehmern (Buddies)

2.3.4 Voice Kommunikation

Der Auf- und Abbau einer Audio-Kommunikation findet nicht gänzlich auf der Protokollebene von SIP statt[2]. SIP-Pakete können SDP[3]-Beschreibungen enthalten, mit welchen RTP[4]-Streams beschrieben werden, die für die bidirektionale Audiokommunikation gebraucht werden sollen. Daraus leitet sich ab:

- Ein Client muss im Channel integriert werden, der RTP Streams verarbeiten und anzeigen kann[5]

[2]Tatsächlich ist es die Aufgabe von SIP, eine Voice-Session zu initiieren und zu beenden. Allerdings liegt das Aushandeln der konkreten Kommunikationsparameter (Codec, Bandbreite, Transport ..) nicht in seinem Aufgabenbereich und überlässt diese Aufgabe anderen Protokollen wie beispielsweise SDP.

[3]Session Description Protocol [17]

[4]Realtime Transport Protocol [18]

[5]Mehrere Möglichkeiten sind dabei denkbar: Java Media Foundation (z. Z. allerdings nur für Windows und Linux verfügbar), Flash..

2.4 Vorhandene Infrastruktur

Die Infrastruktur, die von PolyPhone und myETH vorgegeben ist, umfasst mehrere Server (siehe Abb. 2.1): auf `sip.ethz.ch` laufen die SIP Proxy- und Registrar-Dienste, auf `myeth.ethz.ch` ein Webserver[6] mit dem Portal.

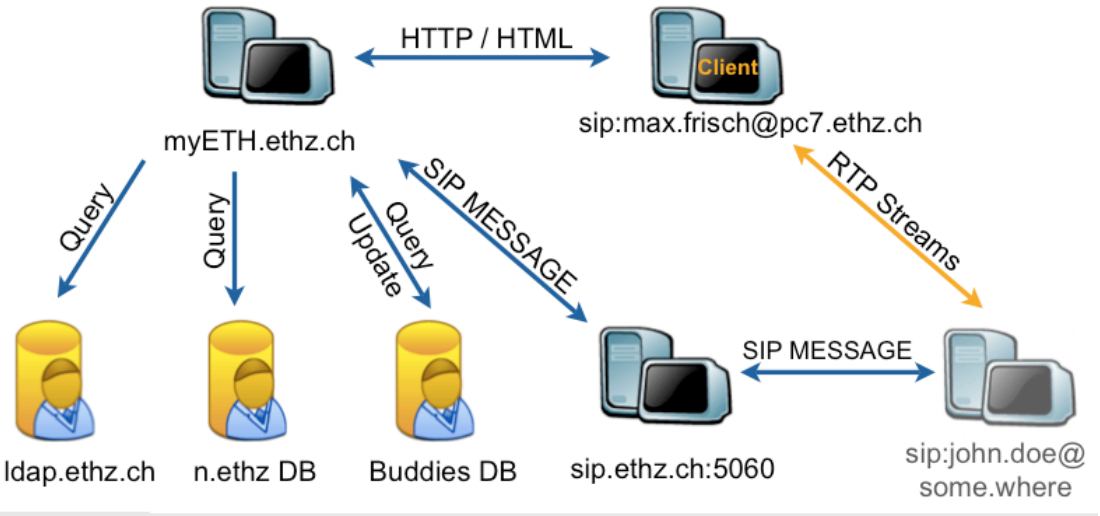

Abbildung 2.1: Übersicht über die involvierten Server und ihre funktionale Verbindungen

Ein Client nimmt mit dem Webserver Verbindung auf und authentifiziert sich (mit Zuhilfenahme der LDAP- und n.ethz-Datenbanken) über die Login-Seite. Danach stehen diesem Benutzer die Channels von myETH zur Ansicht zur Verfügung.
Der SIP-Server registriert neu eingetroffene Benutzer in seiner Location-Datenbank und versucht diese anzusprechen, wenn für sie bestimmte SIP-Nachrichten eintreffen. Eine SIP-Nachricht, die beispielsweise von *John Doe* gesendet wird, gelangt zuerst an `sip.ethz.ch`, und dieser leitet es an jene IP-Adresse weiter, die in der Location-Datenbank gespeichert ist.
Wie wir in den weiteren Kapiteln sehen werden, wurde eine Implementierungsform gewählt, die `myethz.ethz.ch` die Rolle des SIP-Endpunkts zuweist für den angemeldeten Benutzer. Im oben dargestellten Beispiel befindet sich der SIP-Benutzer *max.frisch@ethz.ch* aus Sicht der SIP-Infrastruktur auf dem myETH-Server.
Kommt schliesslich eine Voice-Session zustande, wird diese von der SIP-Infrastruktur losgelöst betrieben: die beiden Teilnehmer verfügen über eine direkte Verbindung. Kurznachrichten oder Präsenz-Informationen hingegen werden über die SIP-Infrastruktur transportiert[7] ohne eine eigene Sitzung aufzubauen.

[6] Tatsächlich handelt es sich hier um einen Loadbalancer, der mehrere einzelne Server zu einem logischen Server zusammenfasst. Die einzelnen Maschinen nennen sich $uportal_i.ethz.ch$.

[7] Hier werden Instant Messages im sogenannten *page mode* [15] verschickt, im Gegensatz zum *session mode* [16], wie es zur Zeit in einem IETF Draft vorgeschlagen wird.

3 Technisches Konzept

3.1 Grundarchitektur

Das myETH-Portal läuft auf einem Tomcat-Server [5] und beherbergt eine angepasste uPortal-Umgebung (siehe 3.2.1), die als Servlet betrieben wird. Die Einbettung eines Channels in das myETH-Gerüst führt unweigerlich zu einer Multi-Tier-Architektur.
Die Präsentationsschicht des PolyPhone-Channels teilt sich in eine *statische* und eine *dynamische* Komponente auf. Das statische Rendern einer Seite ist die Aufgabe des *uPortal*-Frameworks, wohingegen das dynamische Auffrischen der Seite (vornehmlich beim Eintreffen von Instant Messages) über das Publish/Subscribe-Framework `Pushlet` (siehe 3.2.2) vollzogen wird. Die Business- und Kontroll-Logik ist im Channel innerhalb des uPortal-Servlets implementiert und greift auf andere Tiers (SIP-Proxy, Datenbanken) zu.

3.2 Kernkomponenten

3.2.1 uPortal: ein Servlet-basiertes Portal

uPortal organisiert die Grundbausteine des Portals als Channels, deren Anordnung und Darstellung den Einstellungen und Wünschen des Benutzers überlassen ist. uPortal unterstützt, abhängig von den Rohdaten, mehrere Arten von Channels. So gibt es beispielsweise Channels, die Bilder, RSS-Feeds oder ganze Webseiten in einen Channel kapseln. Für unsere Bedürfnisse sind diese vorgefertigte prototypische Channels unbrauchbar, weil sie nur die Darstellung der Rohdaten ändern und keine Anwendungslogik beherbergen. Folglich müssen wir eine *eigene Implementierung* entwickeln.
Dies bedeutet, dass wir eine Java-Anwendung schreiben, die sich mittels vorgegebene *Interfaces*

ins Framework einbetten lässt. Die Aufgabe dieses Channels wird es sein, einen XML-Strom an gewünschten Daten zu generieren und die angemessenen, d.h. die vom Benutzer verursachten *Events* entsprechenden, Methoden auszuführen (siehe Abb. 3.1). Für die Transformation von XML zu HTML werden XSLT [1]-Stylesheets verwendet.

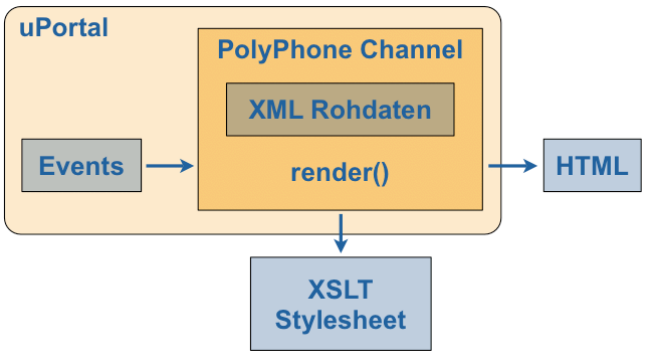

Abbildung 3.1: Die Hauptaufgabe von uPortal: das Rendern der sichtbaren Channels einer Webseite

3.2.2 Pushlet: ein Publish/Subscribe Framework

Die Grundidee hinter Pushlets [9] ist das Verteilen von XML-Nachrichten in einem Publish/Subscribe-Verfahren, wobei diese Nachrichten Event-gesteuert auf effiziente Weise *gepushed* werden – im Gegensatz zu einem Verfahren, das regelmässiges Polling verwendet. Die zugrunde liegende Idee lässt sich mit einem Akronym zusammenfassen: AJAX [8]. Es steht für *Asynchronous JavaScript and XML* und stellt keine eigene Technologie für sich dar, sondern eine Kombination von schon vorhandenen wie DOM, CSS, JavaScript und XMLHttpRequest.
Im PolyPhone-Channel dient der integrierte Pushlet dazu, die im Web Browser fertig dargestellte Seite mit neuen Daten aufzufrischen. Sobald die Seite im Browserfenster fertig aufgebaut ist, registriert sich diese über JavaScript beim Server und abonniert einen für den Benutzer dedizierten Kanal. Danach ist die Webseite imstande, einen XML-Datenstrom zu empfangen (und auch zu versenden).

3.2.3 jSIP: Java SIP Library

Zentrale Bedeutung kommt der SIP Library zu, denn es legt die Basis für unser Vorhaben, eine Java-Applikation zu schreiben, die auf die PolyPhone-Infrastruktur der ETH zugreift. Die Wahl fällt auf jSIP [10], weil es einerseits Messaging- und Presence-fähig ist und andererseits, weil es quelloffen vorliegt und Anpassungen (siehe 4.3.2) erlaubt. Auch liegt eine prototypische Chat-

[1] XSLT [6] ist eine vom W3C standardisierte Sprache, die ein XML Dokument in ein anderes XML Dokument übersetzt, in unserem Fall eine XHTML-Seite.

Software vor, die die Benutzbarkeit der Library unter Beweis stellt. Eine alternative freie Java Library ist Jain-Sip [11].

3.2.4 PostgreSQL: Datenpersistenz

Zur Speicherung von Benutzerdaten (PolyPhone-Nummer und -Passwort, Buddyliste) legen wir eine eigene Datenbank an, wobei wir auf den schon vorhandenen PostgreSQL zurück greifen. Der Zugriff erfolgt über ein eigens definiertes Interface (siehe A.1.2), damit in einer zukünftigen Version eine Anbindung an einen Buddy-Server wie XCAP [7] oder ähnliche Lösungen vereinfacht wird. Die Datenbank-Schemen und das Java Interface sind im Anhang abgedruckt.

3.3 Integration in uPortal

Der Betrieb der PolyPhone Services in myETH wird mit zwei *kooperierenden Channels* realisiert. Der eine Channel wird wie üblich in einer Box dargestellt und stellt das GUI bereit für den eigentlichen Betrieb. Der andere Channel läuft versteckt im Hintergrund und benachrichtigt den Benutzer über neu eingetroffene Instant Messages, falls er sich in einem Modus befindet, in welchem die PolyPhone-Box nicht sichtbar und somit nicht operativ ist.

4 Implementierung

Wie schon in Kapitel 3 dargelegt kommen in dieser Applikation viele verschiedene Technologien zum Einsatz: Java, XML, JavaScript, SQL, XSLT, DHTML, CSS. Ausgehend vom technischen Konzept sollen hier die technischen Ausgestaltungen erklärt und weitergehende Informationen angegeben werden.

4.1 Software-Architektur

Die PolyPhone-Lösung basiert auf zwei interagierenden Servlets, welche unter uPortal laufen (siehe Abb. 4.1 und Konfiguration unter 6.2.1). Der Kern bildet dabei der PolyPhone-Channel (siehe 4.2), der im myETH-Servlet eingebettet ist. Er besorgt das statische Rendern einer Seite. Der Hilfs-Servlet `PolyPhonePushlet` ist für das dynamische Auffrischen der Seite zuständig, indem eine persistente HTTP-Verbindung vom Web Browser (Client) zum Servlet hergestellt wird.

4.1.1 Klassenübersicht

Präsentationsschicht

Wie schon angedeutet gibt es zwei Arten von Präsentationsmodi: das statische Rendern durch uPortal und das dynamische Ergänzen durch `PolyPhonePushlet`.

Neben dem Umstand, dass `CPolyPhone` die Wurzelklasse der Applikation darstellt, ist das statische Rendern der Seite die herausragende Aufgabe dieser Java-Klasse. Das implementierte Interface `IChannel` erlaubt eine Einbindung ins uPortal-Framework. Über verschiedene

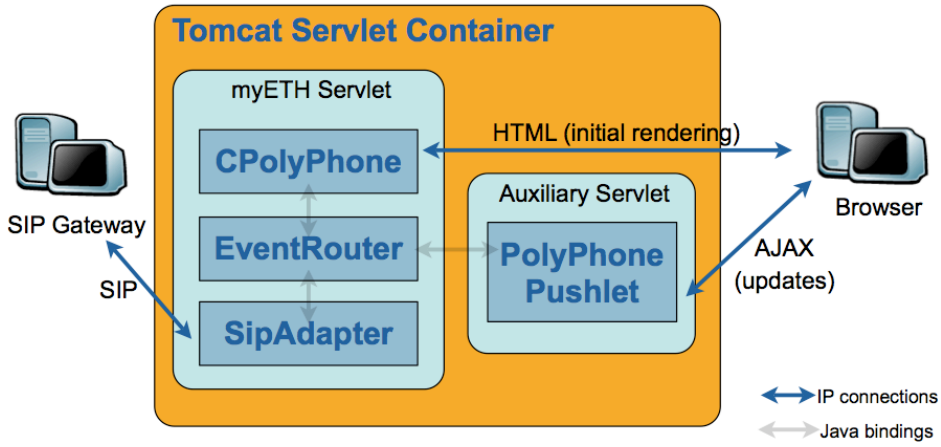

Abbildung 4.1: Interaktion der Java-Entitäten innerhalb ihrer Servletgrenzen

`render()`-Methoden – dem aktuellen *Ansichtsmodus*[1] entsprechend – wird eine Transformation über XSLT-Stylsheets von XML zu HTML vorgenommen. Die verfügbaren Ansichtsmodi sind `normal`, `edit`, `about` und `help`. Die Erstellung der Inhalte, d.h. die Erzeugung des XML-Strings, wird auch in diesen Methoden vorgenommen – in einem Refactoring-Schritt könnte dieser Teil, der eigentlich eine Aufgabe der Business-Schicht ist, aus diesen Methoden extrahiert und sogar in eine andere Klasse ausgelagert werden.

`PolyPhonePushlet` ist eine vom originalen Pushlet (siehe 3.2.2) abgeleitete Klasse. Durch die persistent gehaltene HTTP-Verbindung auf den Kontext `uPortal/pushlet/srv` werden in XML enkodierte Nachrichten ausgetauscht. Bei Empfang einer Update-Anweisung werden im JavaScript des Web Browsers die passenden `receive()`-Methoden ausgeführt, welche über CSS und DOM neue Instant Messages, Präsenz- oder Metainformationen in die bestehende Seite einfügen.

Ein weiterer Channel `CNotifier` dient als Gerüst, um eine Signalisation (z.B. durch das Anzeigen eines passenden graphischen Symbol, oder durch ein akustisches Signal) von ankommenden Meldungen vorzunehmen, falls der PolyPhone-Channel nicht aktiv (angezeigt, gerendert) ist. Dieser Channel basiert auf den gleichen Pushing-Mechanismen und muss *versteckt* in die Navigationsleiste von myETH eingebaut werden. Der jetztige Entwicklungsstand ist als prototypisch zu betrachten.

Business-Logik

Die Aufbereitung von Inhalten ist über mehrere Klassen verteilt. Bei der Instanzierung von `CPolyPhone` wird ein Thread der Klasse `EventRouter` gestartet, der wiederum über

[1] Jeder Channel wird in einer Box dargestellt, die oben eine Menuleiste enthält. Durch Klicken der entsprechenden Symbole in dieser Leiste kann der Benutzer in einen anderen Ansichtsmodus wechseln.

`SipAdapter` die Registrierung des Benutzers vornimmt. `EventRouter` ist dabei der Angelpunkt, wo Nachrichten von und nach dem SIP Gateway oder Web Browser *geroutet* werden. Dabei werden diese Nachrichten in einem *Cache* gepuffert, um bei einem Rendering-Vorgang auf die vergangenen Instant Messages zurück zu greifen und somit wieder dem Benutzer anzuzeigen. Dass diese Vorgänge in einem entkoppelten eigenen Thread ablaufen müssen, liegt am Umstand, dass `CPolyPhone` nur dann ausgeführt wird, wenn der Benutzer (durch Herumklicken im Web Browser) einen Event auslöst, der von einem Channel abgefangen wird. Nachrichten können aber jederzeit eintreffen; das Empfangen und Verschicken von SIP-Nachrichten und Pushlet-Anweisungen verlaufen asynchron.

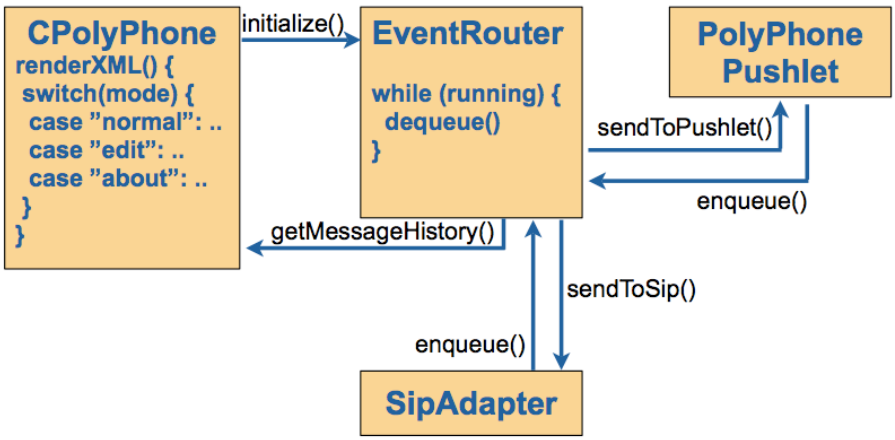

Abbildung 4.2: Methodenaufrufe zwischen den Javaklassen

Die Kommunikation zwischen `EventRouter` und `PolyPhonePushlet` wird durch statische `getInstance()`-Methoden realisiert, die als Parameter die uPortal-Session-ID enthalten. Dadurch, dass myETH- und Pushlet-Servlet im gleichen Tomcat-Kontext gestartet werden, erben beide Prozesse den gleichen Klassenlader, was die Interprozess-Kommunikation untereinander problemlos ermöglicht. Siehe dazu auch Abb. 4.2 für eine graphische Übersicht.
`PolyPhonePushlet` und `SipAdapter` agieren auf der einen Seite als *Producer*, `EventRouter` auf der anderen Seite als *Consumer*.

Session Management

Das uPortal-Framework verwaltet für jeden angemeldeten Benutzer eine persistente Session. Die uPortal-Session-ID kann mit der Methode `getUPortalID()` am `PolyPhoneUser`-Objekt abgerufen werden. Diese Session-ID ist für die ganze Session-Dauer persistent und dient uns als Identifier des Benutzers durch die ganze Applikation hindurch.
Das Pushlet-Framework kennt auch eine Session, die aber temporären Charakter hat. Jedesmal, wenn CPolyPhone eine Seite neu rendert und der Web Browser die Seite darstellt, wird durch ein JavaScript ein Drei-Wege-Handshake mit dem PolyPhonePushlet ausgeführt, womit

die Kreierung einer Pushlet-Session angestossen wird. Dabei muss ein Mapping hergestellt werden zwischen sitzungspersistenter uPortal-Session-ID und der temporären Pushlet-Session-ID. Dies geschieht durch gezieltes Nachführen dieser Identitäten bei `join()` oder `leave()` einer Pushlet-Sitzung. Mit der Methode `getPushletID()` in der Klasse `PolyPhonePushlet` kann jeweils dieses Mapping abgerufen werden.

4.1.2 Nachrichten- und Informationsfluss

Der Aufbau und die Kommunikation der Client-Server-Verbindung erfolgt über mehrere Komponenten. Eine schematische Darstellung soll deren Anatomie verdeutlichen.

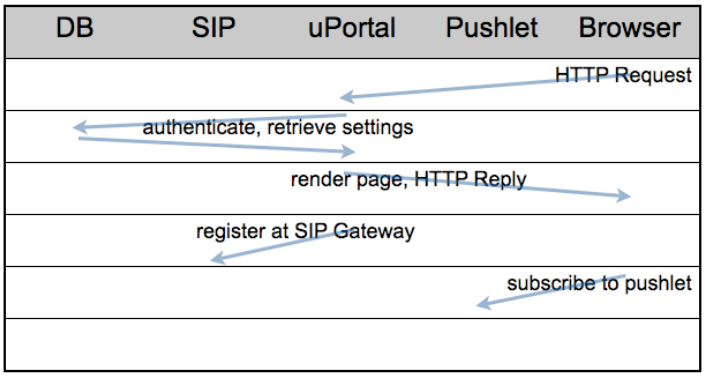

Abbildung 4.3: Nachrichtenfluss in der Initialisierungsphase

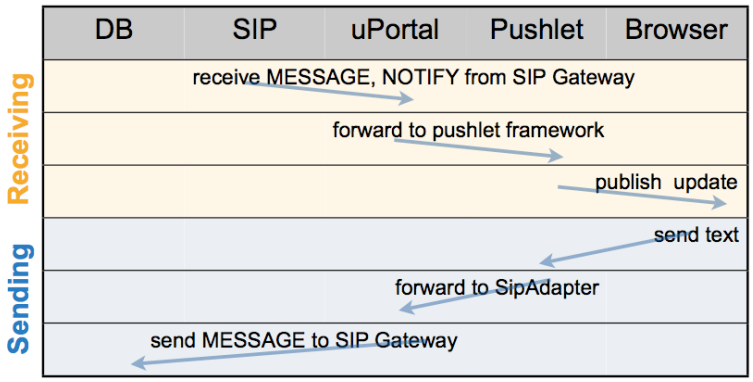

Abbildung 4.4: Nachrichtenfluss im laufenden Betrieb

4.2 Essentielle Java-Klassen

CPolyPhone

Diese Klasse ist die Wurzelklasse der PolyPhone-Applikation. Sie erstellt bei der Instanzierung das Objekt `user` vom Typ `PolyPhoneUser` und startet den Thread `EventRouter`. Bei jedem *Refresh* greift er auf seinen Nachrichtenpuffer zurück, um die `messageHistory` abzufüllen.
Der XML-String `OUTPUT` wird durch String-Konkatenationen innerhalb der `render()`-Methoden formatiert und danach der privaten Methode `doRender()` übergeben, um die effektive XSLT-Transformation vorzunehmen.

CNotifier

Ist der Channel, der für die Signalisation von einkommenden Nachrichten gebraucht werden kann. Die Methode `renderXML()` gibt die Anzahl der sich im Puffer befindenden, aber noch nicht angezeigten Instant Messages aus.

EventRouter

Diese Klasse läuft als eigener Thread und versucht, in einer Endlos-Schleife die Objekte in seiner Warteschlange abzuarbeiten (konsumieren). Diese kann Objekte vom Typ `Message`, `Presence` oder `FeedBack` aufnehmen und ihrer Bestimmung entsprechend dem `SipAdapter` oder dem `PolyPhonePushlet` *senden*.
Die benötigten Puffer wie z.B. `messages` werden durch die Klasse `RingBuffer` zur Verfügung gestellt. Diese sind dem Consumer-Producer-Modell entsprechend Thread-sicher *synchronisiert*. Produzierende Klassen greifen über das implementierte Interface `EnQueueable` zu. Über die Methode `getMessageHistory()` können die zuletzt gerouteten Nachrichten abgerufen werden.
Bei der Instanzierung muss ein *Identifier* mitgegeben werden, praktischerweise bietet sich hier die uPortal-Session-ID an. Damit ist es dem `SipAdapter` oder dem `PolyPhonePushlet` zu jedem späteren Zeitpunkt möglich, durch die statische Methode `getInstance(String ID)` auf die *richtige* Instanz zuzugreifen.

SipAdapter

Diese Klasse ist das Bindeglied zur jSIP-Library. Mittels der `register()`-Methode wird der Benutzer beim SIP-Proxy angemeldet und, wie bei PolyPhone erforderlich, authentifiziert. Falls nötig wird auf eine Umleitung mit einer Wiederregistrierung beim alternativen Server vorgenommen.
Es wird eine anonyme Klasse des Typs `CallListener` mit den zwei Methoden `responseReceived()` und `requestReceived()` definiert, welche auf Events von jSIP je nach Message-Typ reagiert. Im Falle einer SIP-Nachricht vom Typ `MESSAGE` wird diese Nachricht dem `EventRouter` mit einem `enqueue()` übergeben.

In der anderen Richtung ist die Methode `send()` für das Verschicken von Instant Messages an den SIP Proxy verantwortlich.

Message

Diese Klasse kapselt eine Instant Message mit den Angaben über die *Richtung* (eingehend oder ausgehend), die *SIP-Adresse* des Kommunikationspartners (kann Sender oder Empfänger sein), die *Zeitangabe* und natürlich den *Inhalt*. Im statischen Kontext wird eine XML-Serialisierung mit der Methode `toXMLString()` vorgenommen. Im Pushlet-Kontext kommt `toString()` der erbenden Klasse `Event` aus dem Pushlet-Framework zum Zuge. Dazu werden im Konstruktur die Felder der Superklasse entsprechend gesetzt.

PolyPhoneUser

Kapselt alle benutzerrelevanten Daten und implementiert das Interface `IBuddyQueryable` (siehe A.1.2 für Details). *PolyPhone-Nummer*, *Benutzername* (für die Authentifizierung), *PolyPhone-Passwort* und *uPortal-Session-ID* können abgefragt oder gesetzt werden.

PolyPhonePushlet

Diese Klasse erweitert das ursprüngliche Pushlet mit PolyPhone-spezifischen Erweiterungen (siehe weiter unten). Aus Sicht des Tomcat-Container läuft dieser Servlet unter einem eigenen Kontext und wird beim Starten des Portals aktiviert. Die weiteren Anpassungen schlagen sich nieder in den JavaScripts, die einen grossen Beitrag zur Funktionalität leisten.

4.3 Erweiterungen an Bibliotheken

4.3.1 myETH

Zwecks besserer Portabilität wurde auf die Verwendung von Klassen möglichst verzichtet, die speziell für myETH (Package `ch.ethz.ethworl.uportal`) entwickelt worden sind, und zwar zugunsten der offiziellen uPortal-Distribution. Der PolyPhone-Channel implementiert das Interface `IChannel` direkt, ohne von Klassen wie `BaseChannel` oder `StateAwarePersistentUDChannel` zu erben. Diesen Weg haben andere Channel-Implementation eingeschlagen, wie z.B. der SMS-Channel. Dieser Weg ist jedoch eine Sackgasse, da jeder wesentliche Upgrade des darunterliegenden uPortal-Frameworks unweigerlich eine Reimplementation der eingefügten Änderungen erfordern würde.

4.3.2 jSIP

In der Version 0.8 geht die Bibliothek davon aus, dass der Benutzername eines `SipClient` gleich zu setzten ist mit dem *Authentifizierungsnamen*. Dem ist aber nicht so, denn bei PolyPhone ist der Benutzername die PolyPhone-Nummer, der Authentifizierungsname muss jedoch dem des n.ethz-Accounts entsprechen.

Aus diesem Grund wurden die Klassen `SipUser` und `SipRegister` dahingehend erweitert. Die entsprechenden Änderungen sind im Quellcode markiert.

4.3.3 Pushlet

Die Methode `doRequest()` wird überschrieben, um das Mapping zwischen uPortal- und Pushlet-Session-ID vorzunehmen. Diese können mit `getPushletID()` abgefragt werden, oder mit `getNotifierID()` im Falle des Signalisationskanals.

4.4 Bildschirmfotos

Der Instant Messenger (IM) bietet mindestens zwei Ansichtsmodi (Views); *normal* und *edit* (siehe Abb. 4.5 und 4.6). Im ersten Modus wird die Textanzeige über eingetroffene und abgehende Textnachrichten, die nötigen Felder fürs Verfassen von neuen Nachrichten und eine Spalte mit den Buddies und ihrem Präsenzstatus dargestellt.
Im zweiten Modus (edit) ist die Schaltfläche sichtbar, mit welcher sich einerseits die Einstellungen des Benutzers (PolyPhone-Nummer und -Passwort) vornehmen lassen, als auch die Buddyliste verwalten lässt.

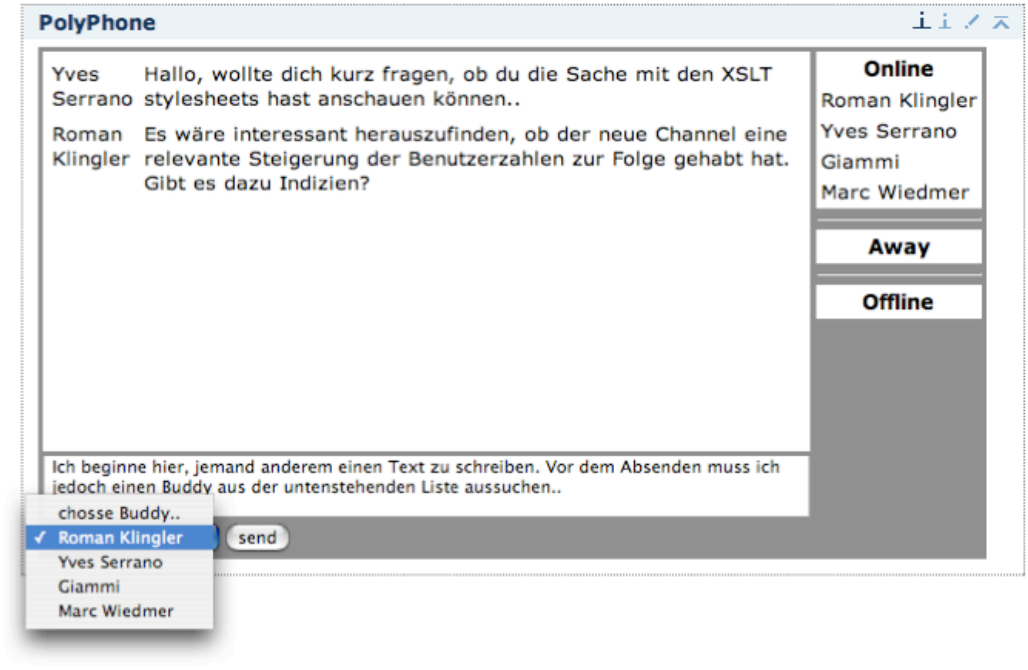

Abbildung 4.5: Benutzersicht des PolyPhone IM Channels im *normal* Modus

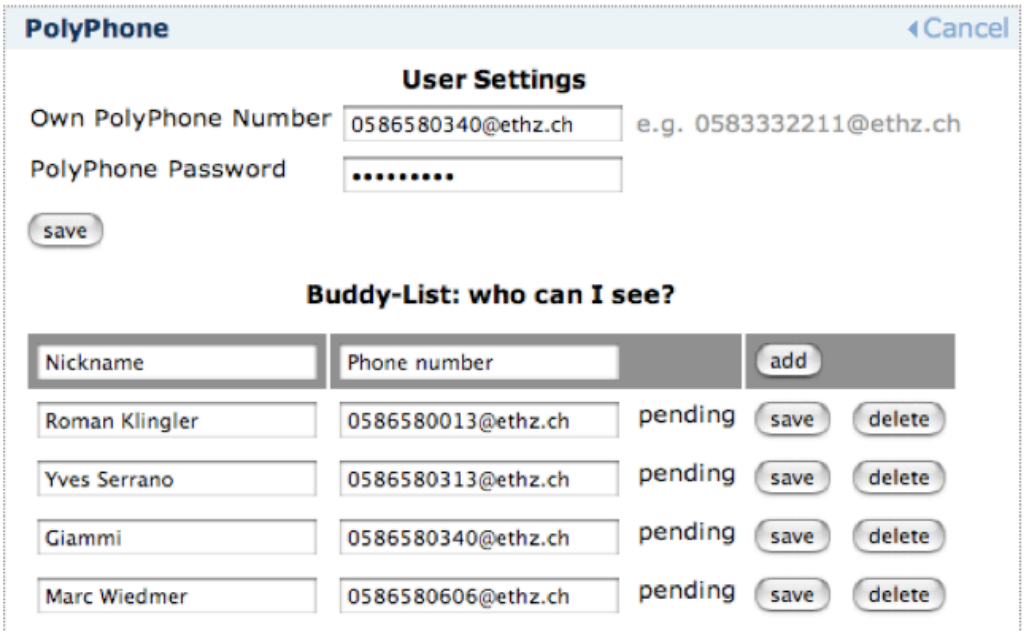

Abbildung 4.6: Benutzersicht des PolyPhone IM Channels im *edit* Modus

5 Betriebskonzept

5.1 Infrastruktur

Für den Betrieb der SIP-Infrastruktur sind die Informatikdienste verantwortlich. Die operationelle Verantwortung für das Portal liegt bei myETH selbst, welches mit seinen eigenen Ressourcen nicht nur die Plattform und die entsprechenden Maschinen wartet, sondern auch eigene Channels entwickelt und den laufenden Ansprüchen anpasst (siehe Abb. 5.1).

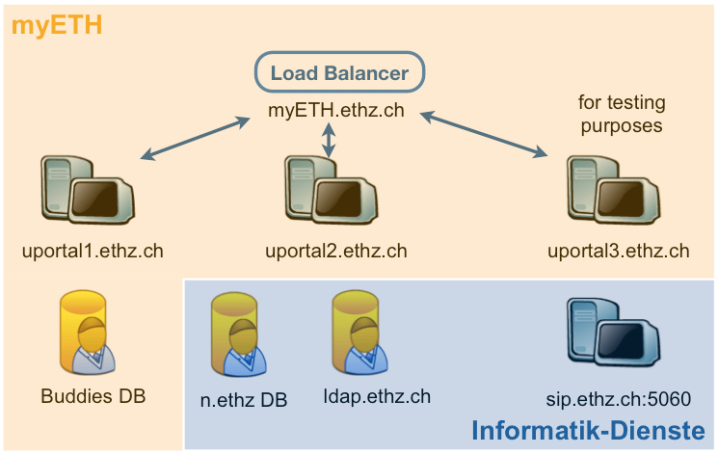

Abbildung 5.1: Aufteilung der betrieblichen Verantwortung der verwendeten Server

5.2 Wartung und Weiterentwicklung

Wie wir bereits schon unter Kapitel 2.2 festgestellt haben, sind verschiedene Parteien in diesem Projekt involviert. Jede dieser Parteien könnte prinzipiell das Projekt am Ende übernehmen. Viele Gründe sprechen jedoch dafür, dass myETH das Projekt am Schluss adoptiert. Zum einen verfügt die Gruppe über eigene Ressourcen für die Entwicklung von Channels, zum anderen ist hier die Überlappung zwischen Aufgabengebiet von myETH und den Aspekten dieser Arbeit deutlich am grössten.
ETH World hat sein Aufgabengebiet im Explorationsbereich und die Informatik-Dienste sind eher für den Unterhalt der Infrastruktur denn für die auf ihr laufenden Applikationen ausgerichtet. Zudem endet das ETH World Projekt im Jahr 2005 und wird in einer anderen Form weitergeführt.

5.3 Rechtliche Implikationen

Sämtliche Komponenten (Libraries) unterstehen entweder der GPL [12] oder der LGPL [13]. Die beiden Lizenzarten können miteinander kombiniert werden und es enstehen keine weiteren Pflichten bezüglich Offenlegung[1] des Codes.

[1] diese ist gemäss Lizenzbestimmung nur dann zwingend, falls diese Software veröffentlicht werden sollte. Siehe dazu http://www.gnu.org/licenses/gpl-faq.html#GPLRequireSourcePostedPublic

6 Deployment

6.1 Installation

6.1.1 Java Kompilate

Das Package `ch.ethz.ethworld.uportal.channels.polyphone` muss ins uPortal-Verzeichnis `../WEB-INF/classes/` kopiert werden. Hierhin kommen auch sämtliche verwendeten XSLT-Stylesheets (Dateien mit dem Suffix `.ssl` oder `.xsl`).

6.1.2 Java Bibliotheken

Die jSIP-Library wurde proprietär abgeändert, darum muss das mitgelieferte Package `org.mitre.jsip` und nicht die als jar-File frei verteilte Version installiert werden. Weiterhin müssen folgende Bibliotheken unter `../WEB-INF/lib/` verfügbar gemacht werden:

- `jace.jar`
- `pushlets.jar` mit den entsprechenden Properties-Dateien

6.1.3 JavaScripts

Im Ordner `../static/` muss sich ein Unterordner `pushlet` befinden mit folgenden Dateien:

- `polyphone.js`
- `edit.js`

- `xmlhttprequest.js`
- `xparse.js`
- `notifier.js`

6.2 Konfiguration

6.2.1 web.xml

Diese Konfigurations-Datei für den myETH-Servlet (uPortal) muss um folgenden Eintrag ergänzt werden:

```
<servlet>
   <servlet-name>PPPushlet</servlet-name>
   <servlet-class>
     ch.ethz.ethworld.uportal.channels.polyphone.pushlet.PolyPhonePushlet
   </servlet-class>
   <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
   <servlet-name>PPPushlet</servlet-name>
   <url-pattern>/pushlet/srv</url-pattern>
</servlet-mapping>
```

6.2.2 XSLT Stylesheets

Diese Dateien enthalten eingebetteten JavaScript-Code, in denen absolute Pfade vorkommen, um die JavaScript-Dateien zu referenzieren. Sie müssen den effektiven URLs entsprechen und angepasst werden.

6.2.3 Pushlet

Die Dateien `sources.properties` und `pushlet.properties` müssen sich im gleichen Ordner wie die Pushlet-Library befinden.

6.3 Log-Dateien

Log-Informationen werden primär in der Datei `Portal.log` und `catalina.out` des Tomcat-Log-Ordners abgelegt.

7 Diskussion

7.1 Schwierigkeiten bei der Implementierung

Ein Merkmal dieser Arbeit ist zweifelsohne der gewaltige Mix an Technologien. Das Zusammenfügen der einzelnen Komponenten hat sich als zu meisternde, aber nicht zu unterschätzende Aufgabe erwiesen.

In besonderem Masse musste Know-how im Bereich Tomcat/uPortal gewonnen werden, bevor die zweite Hürde, die Einbettung in die SIP-Architektur, angegangen werden konnte. Mangels hinreichender Dokumentation musste die Mechanik von uPortal anhand des Quelltextes nachvollzogen werden oder durchs Studium des schon vorhandenen SMS-Channels. Auch die jSIP ist nur rudimentär, sprich mit Annotationen im Quelltext und mittels einer Referenzapplikation [10], dokumentiert.

Besondere Aufmerksamkeiten kam dabei dem Wirken der Java-Klassenlader zu, wodurch sich in einer ersten Phase die Interprozess-Kommunikation zwischen dem uPortal-Servlet und dem Pushlet schwierig gestaltete. Erst nach Entschlüsselung der genauen Vererbungskette der verschiedenen Klassenlader ergab sich die Lösung, dass das Pushlet in den uPortal-Kontext eingefügt werden *muss*.

Eine betriebliche Hürde beim Entwickeln dieser Applikation war die Entwicklungsumgebung selbst: viele der Server, auf denen die Software läuft oder interagiert, sind business-kritisch und erlauben folglich nur restriktiv Zugriff. Deshalb war ich nicht in der Lage, Einsicht in die Log-Dateien zu erlangen. Selbst der mir zur Verfügung gestellte Entwicklungsserver war bei den Zugriffsrechten derart eingeschränkt, dass beispielsweise das Neustarten des Tomcats nicht

erlaubt war. Erst nach Überwindung dieser betrieblichen Hindernisse konnte der Entwicklungsprozess an Geschwindigkeit gewinnen, weil dann Probleme schneller analysiert und aufgedeckt werden konnten.

7.2 Architektur-Beschränkungen

Es gibt einige konzeptionelle Faktoren, die die Skalierbarkeit dieser Lösung limitieren. Durch das Hinzufügen des PolyPhone-Channels muss jeder Web Browser neben der bereits bestehenden Verbindung eine zusätzliche *persistente* HTTP-Verbindung zum Tomcat-Server behalten, damit er mittels des Pushlet-Frameworks über Updates benachrichtigt wird.
Eine weitere Limitierung stellt die im Pushlet verwendete Sitzungsverwaltung dar. Die generierten Sitzungsbezeichner (Session Identifiers) sind relativ klein und weisen eine Struktur auf, die eher auf menschliche Lesbarkeit abzielt denn auf Ausschöpfung des ASCII-Zeichensatzes. Dies hat zur Folge, dass bei einer steigenden Zahl von Benutzern die Kollisionsgefahr für diese Bezeichner zunimmt. Um diese Beschränkung zu beseitigen, müsste der Quelltext analysiert und abgeändert werden, was grundsätzlich möglich ist.

Die jetzige Implementierung generiert viele Threads. Jeder Benutzer erhält einen eigenen `EventRouter`-Thread, der auf dem von der SIP-Bibliothek verwalteten Socket auf eintreffende Ereignisse wartet. Ein weiterer Thread sorgt für eine periodische Wiederregistrierung. Somit stehen jedem angemeldeten Benutzer drei Threads gegenüber. Früher oder später wird eine solche Lösung an die Grenze der von der Virtual Machine zur Verfügung gestellten Threads stossen – ganz abgesehen vom zu erwartenden Performance-Einbruch. Abhilfe können sogenannte Threadpool schaffen oder ein eigenes Verfahren zum Aufteilen der Rechenressourcen.

7.3 Weiterführende Arbeiten

Diese Applikation kann in mehreren Dimension weiter wachsen. Auf der einen Seite gibt es, wie eben dargelegt, genügend Raum und Notwendigkeit für Performance-Optimierungen. Auf der anderen Seite gibt es noch viele Wünsche an Funktionalität, die ungestillt geblieben sind. So etwa das Integrieren von Voice-Fähigkeiten, oder eine Integration von XCAP für die Verwaltung von Buddylisten.

Performance-Verbesserungen	**Funktionalität-Erweiterungen**
Threadzahl senken	Voice-Kommunikation
Pushlet-Bezeicher erweitern	XCAP
	Integration des SMS-Dienstes

7.4 Fazit

Nach mehrmonatiger Auseinandersetzung mit der Thematik bleibt das Gefühl, dass viele Parameter die Güte und Brauchbarkeit dieser Lösung bestimmen. Diese Fragilität kam vor allem bei der SIP-Anbindung zum Vorschein, bei welchem jeder Wechsel des SIP-Proxys eine Software-Regression zur Folge hatte. Jeder Proxy wendet andere, unterschiedlich strenge Verhaltensregeln an, mit denen jSIP unterschiedlich gut klar kommt.

Aufgrund der in Kapitel 7.1 beschriebenen Schwierigkeiten war ich nicht in der Lage, derart viel Funktionalität zu implementieren wie es sich auf den ersten Blick angeboten hätte. Am Ende der Semesterarbeit ist nur Instant Messaging komplett realisiert, Präsenz ist zum grössten Teil vorbereitet aber nicht aktiv, und Voice wurde schon sehr früh aus der Liste der Implemetierungspunkte gestrichen. Gemessen an den Anforderungen aus Kapitel 2.3 sind die ersten zwei Punkte (2.3.1 und 2.3.2) vollständig erfüllt, der nächste (2.3.3) nur teilweise und der letzte (2.3.4) gar nicht.

Persönlich muss ich herausstreichen, dass sich diese Semesterarbeit als ein intensives und lehrreiches Projekt herausgestellt hat. Die dazugewonnene Expertise in den verwendeten Technologien haben gewiss dazu beigetragen. Eine besondere Qualität dieses Projekts war jedoch die Kommunikation mit den verschiedenen involvierten Parteien und die projekttechnischen Aspekte: Das Abklären der Bedürfnisse, die Entwicklung eines übergreifenden Konzepts und die anschliessende Implementierung wurden stets von Milestone-Präsentationen begleitet, die jeweils eine Projektphase abschlossen und dem Projekt eine aufbauende Struktur verliehen. Das Resultat dieser Vorgehensweise äussert sich letztlich auch darin, dass keine Designentscheidung rückgängig gemacht werden mussten und das vorgeschlagene Konzept aufgegangen ist.

Schlussendlich, nach allen Mühen und Freuden, hat diese Implementierung bewiesen, dass sie ihrem Anspruch gerecht wird: nämlich PolyPhone auf intuitive und einfach Weise in myETH verfügbar zu machen. Sollten genügend Ressourcen auf seine Weiterentwicklung aufgewendet werden, hat PolyPhone in myETH das Potential, sich zu einem äussert brauchbaren und beliebten Instrument zu entwickeln.

A Implementierung

A.1 Java Interfaces

A.1.1 IBuddy

Ein *Buddy* wird aus Sicht eines Benutzers über dieses Interface angesprochen. Die implementierende Klasse ist `Buddy`.

```
public interface IBuddy {
   public static final int AUTH_DENIED = 0;
   public static final int AUTH_PENDING = 1;
   public static final int AUTH_APPROVED = 2;

   public String getSipNumber();
   public int getAuthStatus();
   public void setPolyPhoneNumber(String number);
   public void setAuthStatus(int status);
   public String getNickName();
}
```

A.1.2 IBuddyQueryable

Über dieses Interface werden Buddies des Benutzers aus einer persistenten Quelle (Datenbank, XCAP o.ä.) abgerufen oder modifiziert.

```
public interface IBuddyQueryable {
   public IBuddy[] getBuddies();
   public boolean addBuddy(IBuddy b);
   public boolean deleteBuddy(String key);
   public boolean modifyBuddy(String key, String attributeName,
                              String attributeValue );
}
```

A.1.3 EnQueueable

```
public interface EnQueueable {
   public void enqueue(Message m);
   public void enqueue(Presence p);
   public void enqueue(FeedBack f);

   public void dequeue();
}
```

A.2 PostgreSQL Tabellen

A.2.1 Schemas

BuddyList

```
CREATE TABLE Buddy (
   nUser VARCHAR(64) PRIMARY KEY NOT NULL
   buddyNumber VARCHAR(64) NOT NULL
   buddyNickName VARCHAR(64)
   authStatus INTEGER DEFAULT 0
)
```

PolyPhoneUser

```
CREATE TABLE PolyPhoneUser (
   nUser VARCHAR(64) PRIMARY KEY NOT NULL
   polyPhoneNumber VARCHAR(64) NOT NULL
   password VARCHAR(64) NOT NULL
)
```

A.2.2 SQL Abfragen

Folgende SQL-Abfragen stammen aus der Java-Klasse `PolyPhoneUser`:

BuddyList

```
SELECT_BUDDIES = "SELECT * FROM BuddyList
                    WHERE nUser=UID";

INSERT_BUDDY = "INSERT INTO BuddyList (nUser, buddyNumber, buddyNickName, authStatus)
                    VALUES (UID, SipNummer, NickName, AuthStatus)";

UPDATE_BUDDY = "UPDATE BuddyList
                    SET attributeName = attributeValue
                    WHERE nUser=uid AND buddynumber=SipNummer";

DELETE_BUDDY = "DELETE FROM BuddyList
                    WHERE nUser=UID AND buddyNumber=SipNummer";
```

PolyPhoneUser

```
INSERT_USER_ENTRY = "INSERT INTO PolyPhoneUser(nUser, polyPhoneNumber, password)
                    VALUES (UID, num, pass)";

UPDATE_POLYPHONE_NUMBER = "UPDATE PolyPhoneUser
                    SET PolyPhoneNumber= num
                    WHERE nUser=UID";

UPDATE_PASSWORD = "UPDATE PolyPhoneUser
                    SET password= pass
                    WHERE nUser=UID";

SELECT_USER_INFO = "SELECT * FROM PolyPhoneUser
                    WHERE nUser=UID";
```

A.3 XSLT Style sheets

- CPolyPhone.ssl
- normal.xsl
- edit.xsl
- about.xsl
- help.xsl

- CNotifier.ssl
- notifier.xsl

Literaturverzeichnis

[1] *myETH*, das Community-Portal der ETH Zürich,
`https://myeth.ethz.ch`

[2] *ETH World*, Technologie-Exploration an der ETH Zürich,
`http://www.ethworld.ethz.ch`

[3] *PolyPhone* Internet-Telefonie und mehr für die Community der ETH Zürich,
`http://polyphone.ethz.ch`

[4] *uPortal* Tomcat basiertes, freies, von Hochschulen entwickeltes Portal,
`http://www.uPortal.org`

[5] *Apache Tomcat* Open-Source Container für Java Servlets,
`http://jakarta.apache.org/tomcat/`

[6] *XSLT:* Extensible Stylesheet Language Transformation,
`http://www.w3.org/TR/xslt`

[7] *XCAP:* XML Configuration Access Protocol,
`http://www.jdrosen.net/papers/draft-ietf-simple-xcap-07.txt`

[8] *AJAX:* Asynchronous JavaScript and XML,
`http://en.wikipedia.org/wiki/AJAX`

[9] *Pushlets:* HTTP-basiertes Publish/Subscribe-Framework, von Just van den Broecke, Just Objects B.V.,
`http://www.pushlets.com/`

[10] *jSIP:* Java Library für SIP, entwickelt von der MITRE Corp., basiert auf *dissipate* von Billy Biggs,
`http://jsip.sourceforge.net/`

[11] *Jain-Sip:* Java Library für SIP, entwickelt vom *National Institute of Standards and Technology* (NIST),
`https://jain-sip.dev.java.net/`

[12] *GPL:* GNU General Public License,
`http://www.gnu.org/copyleft/gpl.html`

[13] *LGPL:* GNU Lesser General Public License,
`http://www.gnu.org/copyleft/lesser.html`

[14] Session Initiation Protocol, RFC 3261,
`http://rfc.net/rfc3261.html`

[15] Session Initiation Protocol (SIP) Extension for Instant Messaging, RFC 3261,
`http://rfc.net/rfc3428.html`

[16] The Message Session Relay Protocol, IETF Draft,
`http://www.softarmor.com/wgdb/docs/draft-ietf-simple-message-sessions-11.txt`

[17] Session Description Protocol, RFC 2327,
`http://rfc.net/rfc2327.html`

[18] RTP: A Transport Protocol for Real-Time Applications, RFC 1889,
`http://rfc.net/rfc1889.html`