Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prof. B. Plattner
WS 2006/2007

Erweiterung von PolyPhone um eine Infrastruktur für SIP Instant Messaging-Dienste

Semesterarbeit
SA-2007-02
24. Januar 2007

Studentin: Claudia Brauchli
Betreuer: Michele De Lorenzi, Marcel Baur

Zusammenfassung

Seit März 2006 wird PolyPhone als Pilotversuch an der ETH Zürich durchgeführt. Die gesamte ETH-Community (Studierende, Mitarbeiter) haben die Möglichkeit, diese Dienstleistung zu nutzen. Die Kommunikation in PolyPhone, eine auf dem Session Initiation Protocol (SIP) basierende Dienstleistung, erfolgt via Sprache, Video und Kurzmeldungen. Die Idee, auch andere Dienstleistungen über PolyPhone anzubieten, entstand schon nach kurzer Zeit. In dieser Semesterarbeit wird erläutert, wie diese Idee umgesetzt werden kann. PolyPhone soll erweitert werden um Instant-Messaging (IM)-Dienste zur Verfügung zu stellen. Dienstanfragen sollen mit IM automatisch von einer Software-Applikation beantwortet werden (z.B. zur Abfrage einer Telefonnummer). Anhand von Anforderungen und Analysen werden zwei Konzeptvarianten vorgeschlagen und miteinander verglichen. Die Wahl fällt hier sehr leicht, da man die Implementation und das Testing sehr einfach halten will. Diese Semesterarbeit deckt die Analyse, das Konzept und Teile der Implementation ab, nicht aber die vollständige Implementation.

Abstract

Since March 2006 PolyPhone is accomplished as a pilot test at the ETH Zurich. The entire ETH community (students, employees) has the possibility to use this service. Communication in PolyPhone, based on the Session Initiation Protocol (SIP) service, effected via spoken language, video and short messages. The idea to offer different services over PolyPhone came up after short time. This thesis elucidates how this idea can be done. PolyPhone is to be extended, in order to make Instant Messaging (IM) services available. Service requests are to be answered automatically by IM by a software application (e.g. a request of a telephone number). On the basis of requirements and analysis, two concept versions are suggested and compared with each other. The choice is easy, since the implementation and testing want to be kept very simple. This thesis covers the analysis, the concept and parts of the implementation, but not the complete implementation.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 Ziel und Motivation der Arbeit

Ziel dieser Arbeit ist es, eine generische Plattform für das Erbringen von Instant Messaging [1] (IM)-Diensten zu erstellen. Diese Plattform wird in die vorhandene PolyPhone-Infrastruktur der ETH Zürich integriert. Der vorhandene PolyPhone-Client muss nicht angepasst werden, um diese neue Infrastruktur zu nutzen. Am Ende dieser Arbeit soll eine fertige Infrastruktur zur Verfügung stehen, mit welcher man neue Dienste anbieten kann. Diese PolyPhone-Dienste können von allen PolyPhone-Benutzern via Instant Messaging benutzt werden. Neben den vorhandenen PolyPhone-Funktionen wie Telefonie, Chat etc. werden nun auch verschiedene Dienste über den PolyPhone-Client angeboten. Mögliche Dienste sind Anfragen, abonnierte Mitteilungen, Chaträume, Multicast-Messages und Kalender. Die Idee, Dienste über PolyPhone anzubieten, liegt in der Vereinfachung der Informationsbeschaffenheit. Man muss nicht mehr via ETHZ[1], Google[2], Webseiten, Telefonbücher oder andere Auskunftsquellen mühsam Daten beschaffen, sondern kann mit PolyPhone auf einfache Art und Weise die gewünschten Informationen erhalten.

1.2 Was ist PolyPhone

PolyPhone [2] ist eine an der ETH Zürich entwickelte SIP-Infrastruktur[3]. Sie ermöglicht das Kommunizieren via Internet und steht allen Angehörigen der ETH Zürich frei zur Verfügung. Jeder Benutzer verfügt über eine eigene, persönliche Telefonnummer, welche über das Internet, aber auch über das gewöhnliche Telefonnetz erreichbar ist. Die Kommunikation zwischen Benutzern erfolgt via Sprache, Video und Textmitteilungen. Der PolyPhone-Proxy-Server enthält eine Liste aller angemeldeten PolyPhone-Benutzern mitsamt deren Präsenz-Status (Available/Away/Offline etc.).
Das PolyPhone-Dienstangebot muss in die technische Infrastruktur von PolyPhone der ETH Zürich [4] integrierbar sein.

[1] www.ethz.ch
[2] www.google.com
[3] Siehe Kapitel 1.3 für SIP-Definition
[4] Siehe Abbildung 1

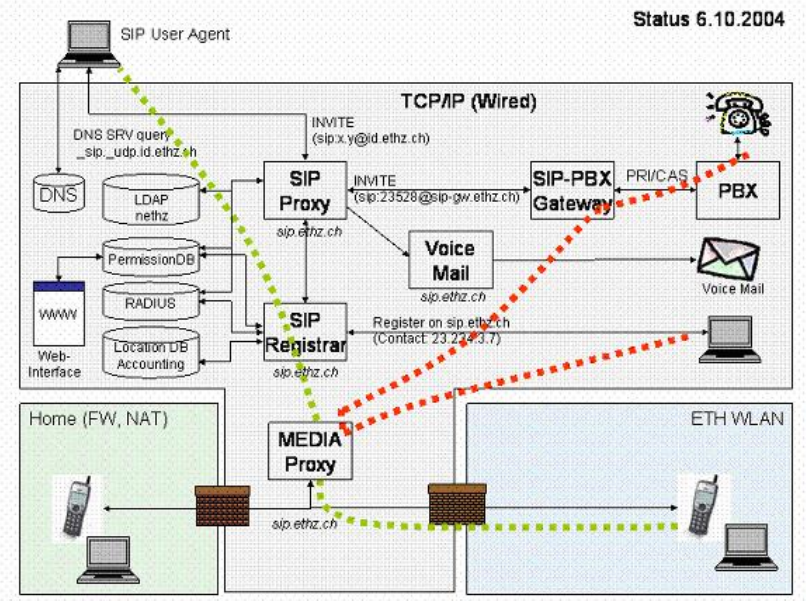

Abbildung 1: PolyPhone-Infrastruktur der ETH Zürich

.
Die PolyPhone-Infrastruktur besteht aus folgenden Komponenten:

SIP Proxy Server: Der verwendete Server ist ein SIP Express Router (SER) [34], welcher erweitert wurde, um existierende Services der ETH Zürich zu integrieren

SIP Registrar: Im SER Registrar sind alle SIP-Benutzer registriert, welche online sind

Feature Server: Zusatzdienste wie Voicemail, Echo, Musik, etc.

Jasomi Networks: PeerPoint, ein Session Border Gateway Controller für NAT-Traversal

LDAP n.ethz: Speichert Benutzernamen, Passwörter, E-Mails und interne Telefonnummern

SIP-PBX Gateway: Schnittstelle von Cisco zur Telefonanlage der ETH Zürich

1.3 Was ist SIP

Das Session Initiation Protocol (SIP) [3] ist ein Netzprotokoll auf der Applikationsschicht für den Aufbau einer Kommunikationssitzung zwischen zwei oder mehreren Teilnehmern. Eine Anfrage für eine Kommunikationssitzung wird an einen zentralen Computer gesendet, welcher eine Liste alle angemeldeten Benutzer der jeweiligen Plattform enthält. Ist der Empfänger angemeldet und online, wird die Anfrage an den Empfänger weitergeleitet. Für jeden SIP-Request wird eine SIP-Response mit den zu HTTP analogen Status-Codes zurückgesendet. Nach diesem Verbindungsaufbau via Proxy-Server steht nun ein Datenstrom zwischen den beiden Benutzern offen. Über diesen Strom können nun Telefonie, Videokonferenzen, Computerspiele etc. stattfinden. Jedoch wird dann nicht mehr das SIP Protokoll verwendet, sondern es kommt ein anderes Protokoll zum Zuge. Das Session Description Protocol (SDP) [7] wird verwendet, um zwischen den Endpunkten das Transportprotokoll auszuhandeln. Das Realtime Transport Protocol (RTP) [8] hat die Aufgabe, Multimedia-Datenströme (Audio, Video, Text etc.) zu transportieren und über UDP [6] zu versenden. Das Protokoll weist beabsichtigte Ähnlichkeiten mit den SMTP- [4] und HTTP- [5] Protokollen auf. Ähnlich zum Versenden von E-Mails „mailto:test@ethz.ch“ kann mit einer SIP-Adresse

sip:test@ethz.ch

ein Gesprächspartner eindeutig adressiert werden. Diese Bezeichnungen werden URI (Uniform Resource Identifier) genannt. Der Aufbau einer URI ist „schema:user@server“ wobei *schema* das verwendete Protokoll (z.B. SIP) beschreibt und *user* die Identifikation des users in einer Domainumgebung *server* ist. Häufig wird als PolyPhone-Bezeichnung lediglich der „user“ angegeben, welcher wiederum aus einer beliebigen Zeichenfolge zusammengesetzt ist.

1.4 Wie funktioniert SIP Instant Messaging

SIP wird normalerweise nur für den Kommunikationsaufbau benutzt. Danach wird über den Datenstrom mit anderen Protokollen kommuniziert. Um trotzdem kleinere Textnachrichten direkt via SIP zu verschicken, existiert bei SIP eine Erweiterung, genannt „MESSAGES“. Mit dieser Methode können solche Instant Messages (IM) [1] im Body einer SIP-Nachricht versendet werden. Die PolyPhone-Textnachricht basiert auf dieser SIP-Erweiterung. Es sind unabhängige SIP-Nachrichten, welche nur für den Benutzer einen logi-

schen Zusammenhang ergeben.
Als weitere Variante kann Extensible Messaging and Presence Protocol (XMPP) [11] benutzt werden, um Instant Messages mit SIP auszutauschen. XMPP ist ein XML-basiertes [12] Open Source Protokoll, welches IM und Präsenzinformationen enthält. Diese Technologie wird bei Jabber [13] verwendet. Eine andere weit verbreitete Variante für IM in SIP ist Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions (SIMPLE) [14], eine Konkurrenztechnologie zu XMPP.

1.5 Aufbau einer SIP-Instant Message

Der Header eines SIP-Packages mit der Erweiterung „Messages“ enthält unter anderem die folgenden Parameter (Beispiel siehe Tabelle 1):

- **MESSAGE:** Die SIP-Erweiterung von SIP Instant Messages, um zusätzlich IM im SIP-Body mitzusenden. Anschliessend nach „Message“ folgt die Absender-SIP-Adresse und die SIP-Versionsnummer (z.B. SIP/2.0)
- **From:** Die SIP-Adresse des Absenders (sip:name@host.com)
- **To:** Die SIP-Adresse des Empfängers
- Text, welcher als Message dem Empfänger geschickt werden kann

Für jedes Request-SIP-Paket (hier mit Erweiterung) wird ein Response SIP-Paket zurückgesendet. Das Response-SIP-Paket enthält neben den nötigen Informationen auch den Status-Code [15] des Request-SIP-Paketes. Der häufigste Status-Code ist „200 OK“, wonach die Meldung des Request-SIP-Paketes den Empfänger fehlerfrei erreicht hat.

1.6 Beispiel einer SIP-IM-Kommunikation

Um Kapitel 1.4 und Kapitel 1.5 besser zu veranschaulichen wird eine einfache SIP-IM-Kommunikation an einem Beispiel erläutert. Dafür benötigt man einen Absender mit der SIP-Bezeichnung „sip:alice@ethz.ch“, einen Empfänger mit der SIP-Bezeichnung „sip:bob@ethz.ch“ und einen zentralen Proxy-Server. Der Proxy-Server kann mittels einer DNS-Anfrage ermittelt werden. Dieser verfügt über eine Liste aller Benutzer, welche angemeldet sind und den

Abbildung 2: SIP-Nachricht von Alice an Bob

Status online haben. Alice sendet nun eine Nachricht „Hallo Bob. Gruss Alice“ via Proxy-Server an Bob. Um eine einzige Nachricht von Alice an Bob zu übermitteln werden vier unterschiedliche SIP-Pakete benötigt. In der Tabelle 1 sind diese vier SIP-Pakte mit kurzer Beschreibung ersichtlich. Abbildung 2 veranschaulicht die vier Schritte zwischen Alice, Bob und dem Proxy-Server.

Schritt	SIP	Erklärung
1	MESSAGE sip:alice@ethz.ch SIP/2.0 From: sip:alice@ethz.ch; To: sip:bob@ethz.ch Hallo Bob. Gruss Alice.	Die Nachricht von Alice wird in einem SIP-Paket an den Proxy-Server gesendet.
2	MESSAGE sip:alice@ethz.ch SIP/2.0 From: sip:alice@ethz.ch; To: sip:bob@ethz.ch; Hallo Bob. Gruss Alice.	Der Server leitet das unveränderte SIP-Paket an Bob weiter.
3	SIP/2.0 200 OK From: sip:bob@ethz.ch; To: sip:alice@ethz.ch;	Bob schickt dem Server sofort nach Eintreffen der Nachricht den Status-Code „200 OK“ zurück.
4	SIP/2.0 200 OK From: sip:bob@ethz.ch; To: sip:alice@ethz.ch;	Der Server leitet die unveränderte Antwort an Alice weiter.

Tabelle 1: Ausgetauschte SIP-Nachrichten

1.7 Aufbau der Arbeit

Die Kapitelaufteilung dieser Arbeit lehnt stark an die definierte Phasenaufteilung[5] der gesamten Arbeit an. Zu den fünf Phasen wurden noch mehrere Kapitel einbezogen, um dem Bericht eine gute und verständliche Struktur zu geben. Nachfolgend werden die einzelnen Kapitel und Anhänge kurz zusammengefasst:

Kapitel 1, Einführung: Es wird eine Basis für den Leser geschaffen, damit er die Aufgabenstellung und das weitere Vorgehen der Arbeit nachvollziehen kann. Unter anderem wird PolyPhone der ETH Zürich, SIP und Instant Messaging genauer erklärt.

Kapitel 2, Anforderungen: Die Anforderungen an die Infrastruktur und die Dienste werden in diesem Kapitel festgelegt. Die notwendigen Daten werden aus den Beispieldiensten aus dem ersten Unterkapitel und den Interviews aus Anhang A extrahiert.

Kapitel 3, Analyse: In der Analyse wird die Diensttypenaufteilung erklärt, der Messageflow erläutert und die möglichen Verteilungen der PolyPhone-Dienst-Nummern analysiert.

Kapitel 4, Umfeldanalyse: Es werden ähnliche IM-Systeme analysiert und mögliche Programmbibliotheken untersucht.

Kapitel 5, Konzept: Aus den Anforderungen und der Analyse werden mehrere Konzeptvarianten vorgestellt. Es werden die Vor- und Nachteile der Varianten erläutert und untereinander verglichen. Die Wahl für das zu implementierende Konzept wird begründet.

Kapitel 6, Implementation: Die ausgewählte Konzeptvariante wird implementiert. In diesem Kapitel werden wichtige Eigenschaften des Quellcodes genauer erläutert.

Kapitel 7, Evaluation: Die ausgewählte Konzeptvariante mit der Implementation wird evaluiert.

Kapitel 8, Quellenangabe: Im letzten Kapitel werden alle benutzten Informationsquellen aufgeführt.

[5] Siehe Anhang B.3

Anhang A, Interviews: Die Protokolle der geführten Interviews werden an dieser Stelle aufgeführt.

Anhang B, Aufgabenstellung: Die genaue Aufgabenstellung der Semesterarbeit kann hier nachgelesen werden.

Anhang C, Zeitplan: Der Wochenzeitplan der Semesterarbeit wird hier tabellarisch aufgeführt.

2 Anforderungen

Dieses Kapitel befasst sich mit den PolyPhone-Diensten, ihrer Anbietungsarten und ihrer Ausführungsarten. Dienste können grundsätzlich auf zwei verschiedene Arten angeboten werden. Die offizielle Variante beschreibt die zentrale Dienstanbietung mit einem dafür verantwortlichen Administrator. In der zweiten Variante wird die Möglichkeit erläutert, Dienste privat anzubieten. Unabhängig von der Dienstanbietung ergeben sich zwei weitere Varianten für die Dienstausführung. Ein Dienst kann über eine eigene PolyPhone-Bezeichnung verfügen oder einer Gruppe von Diensten mit gleicher PolyPhone-Bezeichnung angehören. Im zweiten Fall benötigt jeder Dienst zusätzlich ein Dienstkeyword, um die Dienste voneinander unterscheiden zu können. Um einige Beispiel für PolyPhone-Dienste wiederzugeben wird eine Auswahl möglicher Dienste im letzten Unterkapitel angeboten. Die Anregungen und Ideen für diese Dienste stammen grösstenteils aus den Interviews mit verschiedenen Personen. Diese Interviews sind im Anhang A mit Protokollen der Sitzungen aufgeführt.

2.1 Angebot von IM-Diensten

Grundsätzlich soll es möglich sein, Dienste auf zwei verschiedene Arten anzubieten. Beide Varianten werden in Folge vorgestellt und erklärt. Für den PolyPhone-Endbenutzer sollen sich keine Unterschiede zwischen den beiden Anbietungsarten zeigen.

2.1.1 Offiziell, zentral angebotene Dienste

Diese Dienste sollen zentral auf einem Server als Teil der PolyPhone-Infrastruktur angeboten und verwaltet werden. Ein dafür verantwortlicher Administrator wird alle zentralen Dienste verwalten. Insbesondere wird er Dienste aufschalten, modifizieren, löschen etc. Die offiziellen Dienste werden in einem Dienstverzeichnis aufgelistet und sind für jeden Benutzer abrufbar. Jeder dieser Dienste wird eine Dienstbeschreibung und Benutzungsanleitung bereitstellen. Generell ist jeder PolyPhone-Benutzer berechtigt, offizielle Dienste zu implementieren und für die Aufschaltung dem Administrator zu übergeben.

2.1.2 Privat angebotene Dienste

Privat angebotene Dienste können von jedem PolyPhone-Benutzer erstellt und auf einem Server (z.B. privater Rechner) angeboten werden. Diese Dienste laufen nicht unter den offiziellen PolyPhone-Diensten und werden somit nicht in einem Dienstverzeichnis angeboten. Jeder Benutzer ist selber verantwortlich für Dienste, welche unter seiner persönlichen PolyPhone-Nummer angeboten werden

2.2 PolyPhone-Bezeichnungen der Dienste

Ein PolyPhone-Dienst wird durch folgende Elemente identifiziert:[6]

Dienst: Bezeichnung[Dienstkeyword][Parameter]*

Die Bezeichnung eines Dienstes ist seine PolyPhone-Nummer, eine URI-SIP Adresse. Jeder Dienst besitzt mit Bestimmtheit eine Bezeichnung. Je nach Variante der Dienstbezeichnung muss ein Dienstkeyword existieren um den Dienst eindeutig identifizieren zu können. Unabhängig von der Dienstbezeichnung können Parameter eingesetzt werden, welche in ihrer Anzahl von 0 bis unendlich vielen reichen können. Die eindeutige Identifizierung eines Dienstes wird über die PolyPhone-Bezeichnung, welche auf zwei verschiedene Arten erfolgen kann, erreicht. Im folgenden Kapitel werden diese zwei Identifikationsarten als zwei verschiedene Ausführungsarten beschrieben.

2.3 Ausführungsarten der Dienste

Die Identifikation eines Dienstes bestimmt seine Ausführungsart. Bei der persönlichen Bezeichnung wird der Dienst direkt aufgerufen, wobei die gruppierte Bezeichnung indirekt mit einem zusätzlichen Dienstkeyword aufgerufen wird. Diese zwei Bezeichnungs- und Ausführungsarten werden in den folgenden zwei Unterkapitel erläutert.

[6] Data1[Data2][Data3]* ist ein regulärer Ausdruck mit folgender Bedeutung:
Data1: genau 1 Vorkommnis
[Data2]: 0 oder 1 Vorkommnis
[Data3]*: 0 bis unendlich viele Vorkommnisse

2.3.1 Persönliche Dienstbezeichnungen, direkte Ausführung

Ein Dienst kann eine eigene persönliche Bezeichnung besitzen. Somit ist der Dienst direkt identifizierbar und kann direkt ausgeführt werden. Es wird kein weiteres Dienstkeyword benötigt. Als Beispiel wird ein IM-Dienstaufruf für den Mensa-Dienst mit der PolyPhone-Bezeichnung

sip:Mensa@ethz.ch

, kurz „Mensa“, in Abbildung 3 veranschaulicht. Der Mensa-Dienst erwartet einen Parameter, welcher das Gebäude für den Menuvorschlag beschreibt. In diesem Beispiel wird „HG“ für das Hauptgebäude der ETH Zürich benutzt. Die Serverantwort erscheint im gleichen IM-Clientfenster.

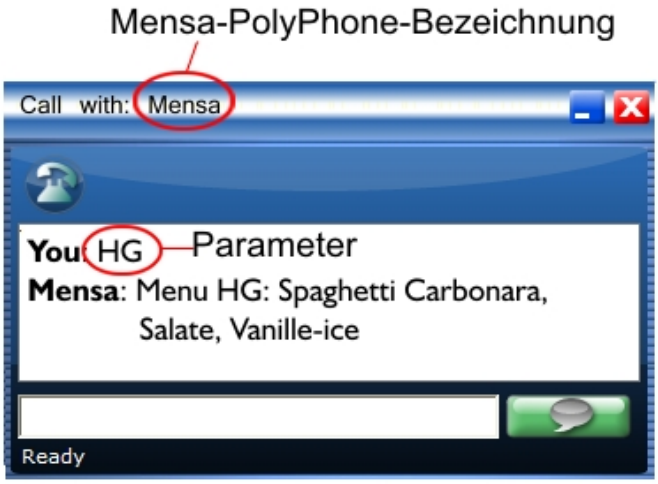

Abbildung 3: Beispiel eines Dienstaufrufes für einen Dienst mit eigener PolyPhone-Bezeichnung

2.3.2 Gruppierte Dienstbezeichnungen

Im Gegensatz zur ersten Variante werden hier mehrere Dienste mit der gleichen Bezeichnung versehen. Gehört ein Dienst einer Gruppe von Diensten mit gleicher PolyPhone-Bezeichnung an, so muss jeder Dienst ein Dienstkeyword für die Identifizierung vorweisen. In der Abbildung 4 wird ein Dienst „Mensa“ mit der allgemeinen Bezeichnung „443“ aufgerufen.

sip:443@ethz.ch

Um den Mensa-Dienst dieser Dienstgruppe anzusprechen benötigt man ein Dienstkeyword „Mensa“. Als Parameter wird „HG“ verwendet, um das Menu vom Hauptgebäude der ETH Zürich zu erhalten.

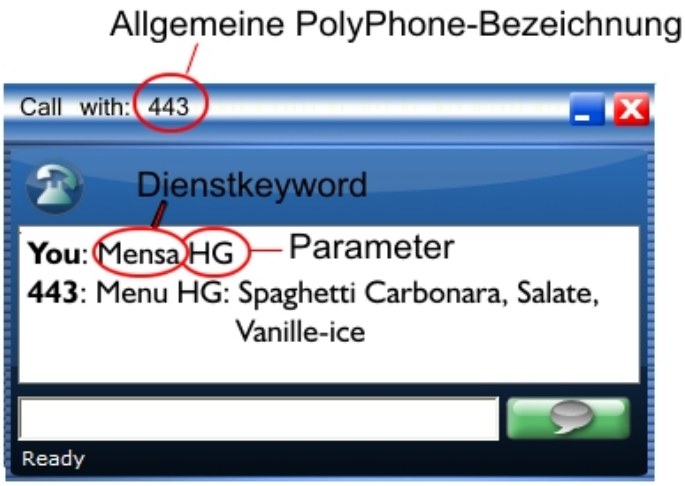

Abbildung 4: Beispiel eines Dienstaufrufes für einen Dienst mit einer allgemeinen PolyPhone-Nummer und einem eindeutigen Keyword

Kurz nach Absenden der IM-Nachricht beider Varianten erscheint im selben IM-Fenster die Antwort des aufgerufenen Dienstes. Der Benutzer kann nun weitere Dienste aus demselben IM-Fenster aufrufen. Diese beiden Varianten für die Bezeichnung von Diensten werden in Abbildung 5 dargestellt. Die drei Beispieldienste „Mensa, Adresse und Mulitcast“ werden auf der linken Seite mit einer eigenen, persönlichen Bezeichnung und auf der rechten Seite als Dienstgruppe dargestellt. Bei dieser Dienstgruppe muss jeder Dienst noch ein Dienstkeyword besitzten (hier jeweils der Dienstname Mensa, Adresse und Mulitcast) um die Dienste identifizieren zu können. Die erste Hierarchiestufe definiert die PolyPhone-Bezeichnung eines Dienstes und die zweite Hierarchiestufe wird für die Dienstkeywords oder Parameter verwendet.

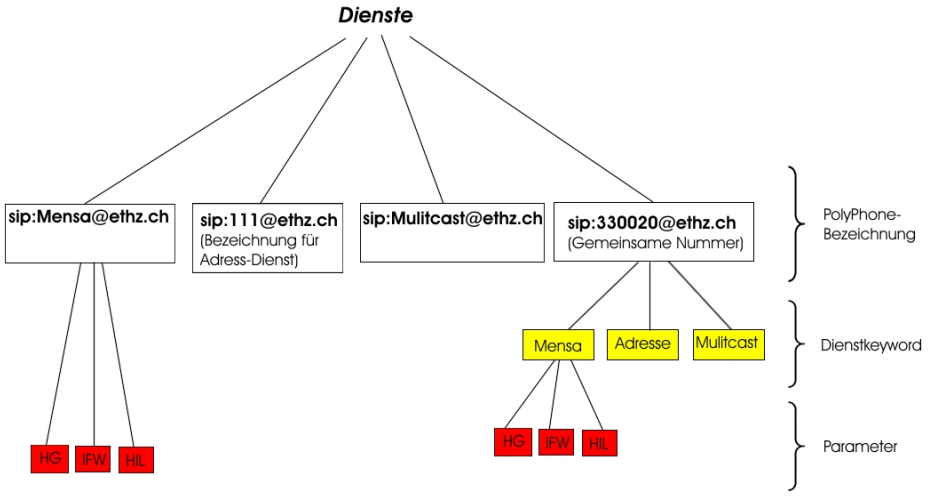

Abbildung 5: Verschiedene Arten der Vergabe von PolyPhone-Nummer an Dienste

2.4 Beispiele für mögliche Dienste

Nach den Gesprächen mit einigen Personen, welche direkt, indirekt oder gar nichts mit PolyPhone zu tun haben sind viele Anregungen, Ideen und Beispiele für mögliche Dienste zusammen gekommen. Die folgende Aufzählung ist nicht vollständig, aber sie enthält die meistgenannten und interessantesten Ideen.

Menu: Mit einem einfachen Keyword „Menu“ gefolgt von einem Gebäudenamen (ifw, hg, hpp, ...) erhält man von diesem Dienst die Menüwahl des nächsten Mittagessens. Dieser Dienst wird auf einem Dienstserver evaluiert, die Antwort in ein SIP-Paket übersetzt und dem Benutzer zurück gesendet. Der Server merkt sich keine Daten des Benutzers, ist somit stateless. Der Dienst holt sich die Daten direkt von der Mensa-Internetseite [16] wie es auch im myETH-Portal [17] gemacht wird.

Adressbuch: Nach dem Keyword „Adressbuch“ und einem Vor- und Nachnamen einer an der ETH eingeschriebenen oder tätigen Person bekommt man als Antwort die wichtigsten Daten dieser Person wie PolyPhone-Nummer, E-Mail-Adresse, ev. Mobiletelefonnummer, ev. Büronummer etc. Auch bei diesem Dienst muss sich der Server keine Daten des anfragenden Benutzers speichern. Die nötigen Daten können aus dem LDAP-Verzeichnis oder über das Web [9] geholt werden.

News: Will man die aktuellsten News eines Departements erhalten, schickt man das Keyword „News“ gefolgt vom jeweiligen Departemenkürzel (dinfk, eltech, ...) als IM dem Dienstserver. Jedes Departement hat eine Web-Seite mit ihren News, welche geparst wird und als SIP-Nachricht dem Benutzer zurück geschickt wird. Ein konkretes Beispiel wären die News vom Informatikdepartement [19].

Gebäude: Möchte man Informationen über ein bestimmtes Gebäude erhalten, kann man mit einer IM „**Gebäude rz**“ alle aktuellen Informationen dazu erhalten. Als aktuelle Informationen gelten Baustellen am und im Gebäude, Anzahl Vorlesungsräume inklusive Kapazität, Mensen, Studienplätze etc. Diese Palette kann individuell erweitert werden. Als Zusatz kann eine URL zum gewünschten Gebäudelageplan übergeben werden. Einige dieser Rauminformationen können aus der Rauminfo-Internetseite [20] stammen.

Buddylist: Ein Benutzer kann eine neue persönliche Buddylist definieren.

Als Message schickt er dem Server „buddylist create buddylistname“ und danach „addbuddy buddylistname nummer1, nummer2, nummer3, ..., nummerX“, um eigene Buddies in diese Liste zuzufügen.

Multicast-Messages: Mit einer definierten Buddylist kann ein Benutzer eine Multicast-Message an diese Liste schicken. Der Dienst „Multicast buddylistname Hallo, wie geht es euch? Gruss Buddy“ versendet die Message „Hallo, wie geht es euch? Gruss Buddy“ an alle Teilnehmer dieser Buddylist.

Chatroom: Dieser Dienst erlaubt eine Kommunikation zwischen mehreren Benutzern. Ein Benutzer definiert einen neuen Chatroom mit der Message „chatroom new chatroomname“. Mit dem Keyword „chatroom“ und dem Parameter “list“ erhält der Benutzer eine Liste von allen offenen Chaträumen und der Anzahl Personen, die diesen Chaträumen zur Zeit beigetreten sind. Mit einem weiteren Befehl „chatroom join chatroomname“ kann ein Benutzer einem ausgewählten Chatroom beitreten. Danach werden alle Messages, die diesem Chatraum gesendet werden, allen eingeschriebenen Benutzern weitergesendet. Sprich: Jeder beteiligte Benutzer eines Chatrooms sieht alle Messages der anderen Benutzern dieses Chatrooms und seine Messages werden auch allen Benutzer dieses Chatrooms geschickt. Mit dem Befehl „chatroom exit chatroomname“ kann ein Chatraum verlassen werden und mit „chatroom delete chatroomname“ kann ein Chatroom gelöscht werden, wenn sich keine Benutzer mehr darin aufhalten.

Kalender: Dieser Dienst erlaubt es einem Benutzer, seine persönlichen Kalendereinträge zu speichern (Vorlesungen, Seminare, persönliches). Für jeden Eintrag gibt es die Möglichkeit, einen Reminder zu setzen. Dadurch wird dem Benutzer vor dem Ereignis eine Nachricht als Erinnerung geschickt. Die Anbindung an Exchange/Outlook wäre eine Erweiterung des Kalenderdienstes.

Events: Um zu wissen, was an der ETH Zürich in nächster Zeit ansteht, gibt es den Dienst „Events“. Dieser liefert eine Liste von Events, welche in nächster Zeit an der ETH Zürich stattfinden werden. Auch hier kann man einen „Reminder“ setzen, um kurz davor mit einer Message daran erinnert zu werden. Unter [21] sind beispielweise die Events des VSETHs aufgelistet.

Parties: Im Gegensatz zum Dienst „Events“ liefert der Dienst „Parties“ nur Studentenparties wie Semesterbeginnparty, Bündnerparty, Sportsbar, Konzerte etc. Dieser Dienst wird von den Studenten sehr geschätzt, da es keine offizielle Homepage mit diesen Daten gibt. Bis jetzt laufen die Werbungen für solche Anlässe immer noch über Mund-zu-Mund Propaganden und Flyers.

Der VSETH kann als Ansprechpartner beigezogen werden. Auf ihrer Seite [18] gibt es eine Telefonnummer für die Auskunft über StuZ2-Parties [10].

Bus Zentrum-Hönggerberg: Für einige Studenten, Professoren und ETH-Mitglieder ist die Busverbindung vom ETH Hauptgebäude im Zentrum zum ETH Hönggerberg sehr von Bedeutung. Um die aktuelle Abfahrtszeit des ETH Direktbuses zu erhalten, kann der Dienst „Bus zentrumhönggerberg“ angefragt werden. Dieser liefert die nächsten fünf Busabfahrtszeiten in beide Richtungen. Die Abfahrtszeiten können von der ETH-Seite [22] oder direkt von der ZVV-Seite [23] genommen werden.

Rechner: Für kleinere oder mittlere Berechnungen kann dieser Dienst benutzt werden. Sein Spektrum von Rechnungsarten reicht von normaler Addition, Subtraktion bis hin zu Integral-, Differenzialrechnungen. Exponential-, Wurzelrechnungen sowie Modulo und Div (Ganzzahldivision) werden auch benutzbar sein. Es soll ein Rechner entstehen, der für alle gängigsten wissenschaftlichen Berechnungen zu gebrauchen ist.

E-Mail: Ein viel benutzter Dienst ist der E-Mail-Dienst. Dadurch wird es möglich, einem Benutzer eine Nachricht zu schicken, welche direkt als E-Mail dem Absender geschickt wird. „Email buddy Hallo Buddy. Wie ist das Wetter in Australien?“ sendet diese Nachricht als E-Mail dem Buddy. Dieser Dienst hat den Vorteil, dass der Absender diese Nachricht auf jedem Computer lesen kann. Er braucht nicht den PolyPhone-Client zu installieren, sondern muss nur seine E-Mails abrufen, um diese Message zu erhalten.

SMS: Ein weiterer nützlicher Dienst ist der SMS-Dienst. Wie beim E-Mail-Dienst wird es hier möglich sein, einem Benutzer direkt eine Nachricht als SMS zu senden. Der Vorteil ist, dass der Benutzer nicht online sein muss, um diese Nachricht zu erhalten.

Missed Calls: Um während seiner Abwesenheit alle eingegangenen Anrufen zu erhalten, kann der Dienst „MissedCalls“ aufgerufen werden. Die Antwortliste enthält die Buddynamen sowie das Datum und Uhrzeit, wann die einzelnen Anrufe stattgefunden haben.

Admintool: Es soll möglich sein, seine eigenen ETH-Angaben direkt über PolyPhone zu administrieren. Mit vordefinierten Befehlen können Passwörter angefragt, geändert werden, Adresse ändern etc. Jedoch muss hier noch die Sicherheit abgeklärt werden, da Daten wia SIP unverschlüsselt versendet werden.

Subscribe: Die Einschreibung für eine Vorlesung erfolgt mit „Subscribe Quantenphysik“. Nun bekommt der Benutzer 10 Minuten vor Beginn jeder Vorlesung des Faches „Quantenphysik“ eine Message zugeschickt. Dieser Dienst muss sich ein Benutzer abonnieren, und kann ihn jederzeit wieder abbestellen (mit „Unsubscribe Quantenphysik“).

2.5 Allgemeine Anforderungen

Da weder die PolyPhone-Infrastruktur noch der Administrator direkten Einfluss auf die privat angebotenen Dienste haben, werden gewisse Anforderungen bei den privaten Diensten nicht erfüllt sein. Dies ist die Angelegenheit des Anbieters. Zusammengefasst ergeben sich die folgenden Anforderungen an Dienste und Infrastruktur, damit es möglich ist, Dienste über PolyPhone anzubieten:

Verfügbarkeit: Es gibt zwei verschiedene Arten von Verfügbarkeit eines Dienstes:

1. **Physisch:** Ein Dienst kann physisch verfügbar sein, d.h. er ist auf dem Server gespeichert. Ist ein Dienst nicht physisch verfügbar, so existiert der Dienst nicht auf dem Server und kann nicht benutzt werden.
2. **Status:** Physisch verfügbare Dienste können bei dafür eingerichten PolyPhone-Clients den Präsenz-Status des Dienstes anzeigen. Als Status gelten: available, busy, offline etc.

Beschreibung: Zu jedem Dienst soll eine Beschreibung für seine Benutzung vorhanden sein. Es soll ein Mechanismus geben mit dem der Benutzer diese Beschreibung eines Dienstes abfragen kann (z.B. „help“).

Verzeichnis: Alle offiziellen Dienste werden in einem Dienstverzeichnis gespeichert. Dieses Verzeichnis muss bei jeder Dienständerung angepasst werden.

Syntaxen: Die PolyPhone-Infrastrukur zur Anbietung von Diensten muss zwei Syntaxen unterscheiden können. Entweder besitzt ein Dienst eine eigene Bezeichnung, und somit kein Dienstkeyword, oder ein Dienst besitzt eine allgemeine Bezeichnung und zusätzlich ein Dienstkeyword.

Anbieten: Das Interface zwischen der Infrastruktur und den Diensten soll so einfach als möglich gehalten werden. Infolgedessen wird die Implementierung eines Dienstes auch für einen Laien möglich sein.

3 Analyse

In der Untersuchung von der Dienstanbietung bei PolyPhone werden in Folge bestimmte Aspekte genauer untersucht und analysiert. Ein wichtiger Aspekt ist die Wahl der Dienstbezeichnung welche im Kapitel 2.2 erläutert wurde. In diesem Kapitel werden diese zwei Varianten analysiert. Eine Aufteilung der Dienste in Diensttypen wird im zweiten Unterkapitel mit den Zuständen „stateless, stateful und stateful mit Serverpush“ erläutert. Als Abschluss der Analyse wird der Messageflow mit „unicast, multicast und broadcast“ für Textnachrichten beschrieben.

3.1 Wahl der PolyPhone-Bezeichnung von Dienste

Eine PolyPhone-Bezeichnung in „sip:bezeichnung@ethz.ch“ setzt sich aus einer Zeichenfolge von Buchstaben und Zahlen zusammen. PolyPhone-Benutzer haben eine persönliche Telefonnummer als Bezeichnung
z.B. „sip:044658....@ethz.ch“. Somit kann jeder Benutzer aus dem Telefonnetz direkt angerufen werden. Dienste hingegen müssen nicht aus einem Telefonnetz aufrufbar sein, weshalb diese Bezeichnungen auch lediglich aus Buchstaben bestehen können. Sie müssen aber trotzdem gültige Bezeichner gemäss SIP-Standard sein. Analog zu den PolyPhone-Benutzern können auch Dienste persönliche Bezeichnungen besitzen oder aber einer Gruppe von Diensten mit gleicher Bezeichnung angehören.

3.1.1 Eine PolyPhone-Bezeichnung pro Dienst

Dienste mit einer eigene PolyPhone-Bezeichnung sind eindeutig bestimmbar. Damit der Dienst direkt mit seiner Bezeichnung aufrufbar ist, muss im PolyPhone-Client ein Buddy erstellt werden.
Vorteil: Es wird kein Dienst-Keyword mehr benötigt. Dank dem Buddy-Status (available, offline) ist direkt ersichtlich, ob ein gewünschter Dienst zur Verfügung steht oder nicht.
Nachteil: Damit man einen Dienst benutzen kann muss der zugehörige Dienstbuddy in die Liste aufgenommen werden. Dadurch kann die Buddyliste extrem gross werden.

3.1.2 Mehrere Dienste für eine PolyPhone-Bezeichnung

Eine Gruppe von Diensten wird mit der gleichen Bezeichnung adressiert. Dadurch muss ein Dienstkeyword existieren, um den einzelnen Dienst eindeutig bestimmen zu können.
Vorteil: Es muss nur ein einziger Dienstbuddy in die Liste eingefügt werden.
Nachteil: Zusätzlich zur PolyPhone-Nummer muss auch das Dienstkeyword bekannt sein um einen Dienst aufrufen zu können. Wenn der Buddy „available“ ist, ist noch nicht gewährleistet, dass auch alle Dienste zur Verfügung stehen. Es ist nicht direkt ersichtlich, ob die Dienste benutzbar sind oder nicht.

Da sich die Nachteile beider Varianten gegenseitig aufheben, ist die beste Lösung für PolyPhone ein Mix aus beiden Varianten. Somit kann ein Dienst eine eigene PolyPhone-Bezeichnung haben (z.B. für Departementsdienste, wichtige Dienste) oder einer Gruppe von Diensten mit gleicher PolyPhone-Bezeichnung angehören (zB. für kleinere Dienste).
Einziger Nachteil: Benutzer werden ein Durcheinander mit der Dienstbenutzung bekommen. Einige Dienste müssen neu in die Buddyliste hinzugefügt werden und bei anderen Diensten braucht es kein Dienst-Keyword.

3.2 Diensttypen

Aus den genannten Beispieldiensten erkennt man eine Aufteilung in drei Haupttypen, welche in Abbildung 6 dargestellt sind.

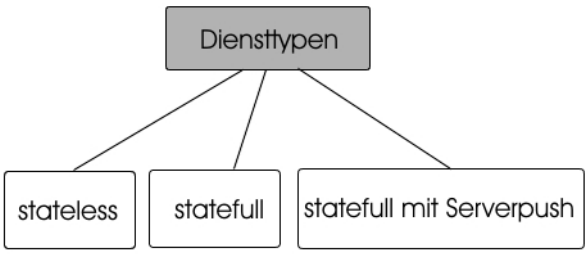

Abbildung 6: Die drei Hauptarten von Diensten

Stateless: Ist ein Dienst stateless, so versteht man darunter, dass sich der Dienstserver keinerlei Informationen über den Benutzer und seine Anfrage speichern muss. Jede Anfrage wird als unabhängige Transaktion ohne Bezug zu früheren Anfragen behandelt. Diese Dienstart verlangt keinen zusätzlichen Speicher, jedoch muss bei jeder Anfrage mehr Informationen mit gesendet werden, welche vom Server bei jeder Anfrage neu interpretiert werden muss. Ein Beispiel dafür ist der Menu-Dienst.

Stateful: Von stateful spricht man, wenn sich der Server Informationen einer Anfrage und seines Benutzers zwischen Anfragen speichert. Diese Daten werden meist in einer Datenbank gespeichert, wo sie jederzeit wieder abrufbar sind. Die Buddylist ist ein klassischer stateful-Dienst.

Stateful mit Ereignissen: Um zu einem gegebenen Zeitpunkt ein bestimmtes Ereignis aufzurufen, muss ein Benutzer zuerst solch einen Dienst abonnieren. Wie ein Weckdienst erinnert der Server den Benutzer für das entsprechende Ereignis. Jeder abonnierte Dienst muss vom Benutzer wieder abbestellt werden, um in Zukunft keine weiteren Meldungen vom Server zu erhalten.

Tabelle 2 zeigt die Aufteilung der möglichen Dienste aus Kapitel 2.4 in die drei Dienstarten stateless, stateful und stateful mit Ereignissen. Man erkennt schnell, dass die meisten Dienste von der Art „stateless“ sind. Bei solchen Diensten muss sich der Server keine Daten speichern und ist somit einfacher zu implementieren (Request, Response). Für stateless-Dienstarten gibt es unzählig weitere Dienstmöglichkeiten. Dieser Teil der Dienstliste wird sich bestimmt schnell vergrössern. Für stateful-Listen benötigt man mehr Know-how, um diese Dienste zu implementieren. Es wird mindestens eine Datenbank, ein Filesystem oder eine andere möglichkeit Informationen zu speichern, benötigt. Die nächste und schwierigere Stufe ist stateful mit Ereignissen. Der Server muss zusätzlich Ereignisse erkennen und dementsprechend reagieren.

stateless	stateful	stateful mit Ereignissen
Menu	Buddylist	Kalender
Adressbuch	Multicast	Subscribe
News	SMS	Missed Calls
Gebäude	Admintool	Chatroom
Event		
Parties		
Bus		
Rechner		
E-Mail		

Tabelle 2: Mögliche PolyPhone-Dienste, nach Dienstarten aufgeteilt

3.3 Message flow

Der Nachrichtenfluss (im englischen „Message flow“) kann durch mehrere Variationen ausgeführt werden. In Abbildung 7 werden drei Möglichkeiten visuell dargestellt.

Unicast: Unter unicast versteht man das Versenden einer Nachricht an genau eine Destination. Chat zwischen zwei Personen und Dienstaufrufe sind unicast.

Mulitcast: Von multicast spricht man, wenn die Nachricht an mehrere Destinationen gleichzeitig gesendet wird. Bei Konferenzen mit mehreren Beteiligten wird multicast benutzt.

Broadcast: Wird eine Nachricht von einer Destination an alle Destinationen im gleichen Netz versendet, so handelt es sich um broadcast. Bei PolyPhone kann dieser Fall vom Administrator oder vom Rektorat benutzt werden, um die gleiche Meldung an alle PolyPhone-Benutzer zu schicken.

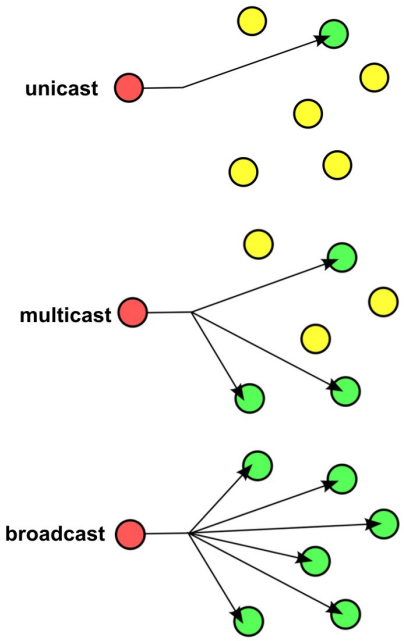

Abbildung 7: Mögliche Varianten von Senden einer Nachricht im gleichen Netzwerk

.

4 Umfeldanalyse

Dienste anzubieten ist für PolyPhone neu, existiert aber bereits in ähnlichen Kommunikationsplattformen. In diesem Kapitel werden bestehende Open Source- oder kommerzielle Plattformen analysiert, welche IM-Dienste anbieten. Man nennt diese Dienste auch IM-Bots (von Instant Messaging Robot). Darunter versteht man ein Computerprogramm, welches auf Anfragen wartet und diese selbständig beantwortet. Ist der Dienst einmal gestartet und beim SIP Proxy-Server angemeldet, arbeitet das Programm eigenständig.

4.1 Plattformspezifische IM-Dienste

Weltweit gibt es unzählige Messengers, aber nur ein kleiner Teil davon bietet Dienste über ihre Plattform an. Diese neue Dienstleistung wird sich in Zukunft schnell etablieren und viele Messenger-Plattformen mitziehen. PolyPhone wagt es als eine der ersten Messenger-Plattformen einen Fuss in eine relativ unbekannte Gegend zu setzten. Es wird sich in Zukunf herausstellen ob Erfolge mit Dienstangeboten zu erziehlen sind. In diesem Kapitel werden drei bekannte Messenger-Plattformen auf Dienstangebote analysiert.

4.1.1 AOL Instant Messenger

AOL Instant Messenger (AIM) [28] stellt IM-Bots als Buddies zur Verfügung. AOL spricht von „künstlicher Intelligenz“ für den Instant Messenger. Unter anderem kann man den AOLBuddy über Wetter, Filmzeiten, Spiele etc. anfragen. Im Internet aufgelistet unter **[29]** findet man eine sehr grosse Auswahl von Dienstbuddies, die zum AOL-Messenger hinzugefügt werden können. AIM verwendet nicht das SIP Protokoll wie es PolyPhone verwendet, sondern setzt auf das OSCAR protocol, wie es auch ICQ verwendet.

4.1.2 MSN Messenger

Die Software-Firma it-observer **[30]** wird bald ein Online-Tool erstellen um einfach und schnell eigene IM-Dienste (auch genannt Bots) für den Microsoft msn-Messenger zu erstellen. Durch den Zugriff auf Echtzeit-Daten werden diese Bots in Zukunft schnell Interesse bei den Usern wecken und ihre Entwicklung vorantreiben.

4.1.3 Habotat

Habotat [31] ist eine Open-Source-Software, mit welchem man selbständig neue Dienste (Bots) für verschiedene Instant Messaging Netzwerke aufsetzen kann - jedoch nicht für SIP-kompatible IM-Dienste. Habotat ist in Java geschrieben und kann auf allen gängigen Plattformen genutzt werden. Einige Bots sind schon implementiert und könnten für PolyPhone auch von Interesse sein:

- **ExecuBot**: Startet ein externes Programm und liefert den Output als IM zurück
- **Jot Bot:** Speichert kurze Textnotizen
- **Reminder Bot:** Erinnert den Benutzer an persönliche Ereignisse
- **URL Reminder:** Kann mehrere URL's für den Benutzer speichern

Diese Auswahl von Bots sind auf der Hauptseite von Habotat [31] mit einer genauen Erklärung aufgelistet. Benutzer-definierte Bots werden auf dem eigenen Benutzer-Rechner für andere Benutzer zur Verfügung gestellt. Jeder Dienst wird mit einer neuen Adresse auf dem Hauptserver angemeldet und ist somit lediglich ein normaler Buddy. Wird eine Message an einen Dienst-Buddy geschickt, wird sie vom Hauptserver an den jeweiligen Dienstanbieter weitergeleitet und auf diesem evaluiert. Die Antwort schickt der Dienstbuddy via Server dem Benutzer zurück. Diese Dienste sind jedoch nicht mehr aufrufbar, wenn der Dienstanbieter mit dem Rechner offline ist.

4.2 Plattformübergreifende IM-Dienste

4.2.1 Trillian

Trillian stellt eine Schnittstelle zu den gängisten Instant-Messaging-Plattformen her. ICQ, Yahoo Messenger, AIM und MSN Messenger können alle aus diesem einen Programm aus genutzt werden. Man kann unabhängig von den IM-Plattformen mit allen Freunden gleichzeitig kommunizieren. Trillian bietet aber nur ein Gateway zu anderen Plattformen an und ist selbst keine IM-Plattform. Der Benutzer benötigt einen persönlichen Account auf den jeweiligen IM-Plattformen. Verwendet der Gesprächspartner ebenfalls Trillian, können die Daten auch verschlüsselt versendet werden.

4.2.2 JBuddy

JBuddy Message Server [32] ist eine Enterprise Instant Messaging (EIM)-Plattform mit verschiedenen Gateways zu MSN, AIM, Jabber, ICQ etc. Benutzer von JBuddy sind unabhängig von den verschiedenen IM-Anbietern und können mit allen Freunden, Kunden, Arbeitskollegen in einem IM-Programm kommunizieren. Die IM's werden in Echtzeit und verschlüsselt versendet. JBuddy-Client und JBuddy-Server sind in Java programmiert und unterstützen somit alle gängigsten Plattformen. Die zentrale Administration aller Benutzer auf dem JBuddy-Server gibt mehr Flexibilität und Kontrolle als bei Peer-to-Peer Chat Software. Für Firmen mit Support-Hilfen kann der Instant Help Desk verwendet werden. Ein Kunde braucht nur den Helpbutton auf der Webseite zu drücken und es erscheint ein Java-Script Chat-Fenster. Darin kann der Kunde seine Fragen an die Firma stellen. Die Help-Desk-Software leitet diese IM an den nächst freien Mitarbeiter der Firma weiter. Falls der Kunde in einer Warteschlange ist, wird eine Rückmeldung geschickt, in welcher Position er sich gerade befindet. Auf der Herstellerseite von JBuddy [32] wird das Erstellen von eigenen IM-fähigen Applikationen oder Services genau erklärt.

4.3 SIP-Bibliotheken

Es gibt mehrere Bibliotheken, welche für SIP-Kommunikation verwendet werden können. Eine Bibliothek wird in den Sourcecode eingebunden und enthält eine Anzahl wichtiger implementierter Klassen. Es vereinfacht die Implementation. Je nach Vorgaben (z.B. die Programmiersprache) und Umgebungen wird ein anderes Wahlverfahren angewendet. Es werden nun vier wichtige Bibliotheken untersucht, welche für das Konzept in Frage kommen würden.

4.3.1 jSIP

Die jSIP-Library (=Jain-SIP) [24] hat einige Prototypen für SIP-Erweiterungen implementiert, im Speziellen für IM. Diese Open Source Bibliothek ist in Java geschrieben und enthält Klassen wie SipMessages, CallListener, Poller, UDPMessageSocket etc. JSIP ist nach eigenen Angaben eine der besten Java SIP-Libraries und ist in der Industrie und an Universitäten weit verbreitet.

4.3.2 osip

Osip [25] ist ein Teil des GNU-Projektes und wird hauptsächlich für SIP-Proxies (low layer) verwendet. Diese Bibliothek ist in C geschrieben, ist ziemlich umständlich und wird für Applikationen mit minimalem Speicherplatz benutzt. oSIP ist eine Implementation von SIP, welche für den Kommunikationsaufbau benötigt wird. Für zusätzliche Instant Messaging gibt es keine Erweiterung.

4.3.3 Sofia-SIP

Sofia-SIP [26] ist eine Open Source SIP-Library für Clientsoftware mit VoIP, IM und anderen Echtzeit- Kommunikationsservices. Diese Bibliothek wird in der C-Programmierumgebung verwendet. Instant Messaging wird mit der generellen Erweiterung „MESSAGES“ ermöglicht.

4.3.4 resiprocate

Diese in C++ geschriebene Bibliotheke wird vor allem für das Parsen und Erstellen von eigenen SIP-Nachrichten benutzt. Diese Bibliothek wird bereits im PolyPhone-Client verwendet. Instant Messaging wird über die Erweiterung „MESSAGES“ ermöglicht. Auf der Wiki-Seite [33] wird die Verwendung dieser Bibliothek mit IM genau erklärt.

5 Konzept

Änderungen an einer sich im Betrieb befindlichen Softwarearchitektur sind später nur mit hohem Aufwand möglich. Die Entscheidung über das Design ist somit eine der kritischsten und wichtigsten Punkte im Entwicklungsprozess einer Software. Es werden nun anhand der Analyse und den Anforderungen an Dienste und Infrastruktur die wichtigsten Komponenten für die Architekturen dargelegt. Im Anschluss werden mehrere vorgestellt und ihre Komponentenwahl erklärt. Für jede Variante wird die Grundidee der Architektur erläutert und die jeweiligen Vor- und Nachteile erwähnt. Im letzten Unterkapitel 5.4 wird eine Konzeptvariante für die Weiterentwicklung (Implementation) ausgewählt und ihre Wahl begründet.

5.1 Gesamtübersicht

Durch geeignetes zusammenfügen der verschiedenen Komponenten entstehen verschiedene Konzeptvarianten. Es wird von einer Referenzarchitektur ausgegangen, welche in Abbildung 8 dargestellt ist. Durch geeignetes Abändern, Verschieben und Ersetzen der einzelnen Komponenten entstehen verschiedene Konzeptvarianten mit verschiedenen Eigenschaften. Im Folgenden wird diese Referenzarchitektur vorgestellt und die einzelnen Komponenten erläutert.

5.1.1 Übersicht

In einer Referenzarchitektur wird die Hauptaufgabe der Dienstanbietung über PolyPhone mittels „Divide and Conquer“ in kleinere Aufgaben aufgeteilt, in so genannte Komponente. Diese Komponenten sind in sich geschlossene Funktionen und sind über einfachste Schnittstellen miteinander verbunden. Die Referenzarchitektur in Abbildung 8 bietet die ganze Funktionalität um Dienste über PolyPhone anbieten zu können. Eine Referenzarchitektur kann beliebig verändert und angepasst werden, ohne dass die Funktionalität beeinträchtigt wird. Der Informationsaustausch zwischen den Komponenten kann über verschiedene Arten stattfinden. Diese hängen von den Architekturen der Plattform ab, welche in Kapitel 5.3 mit zwei verschiedenen Varianten vorgestellt und erläutert werden.

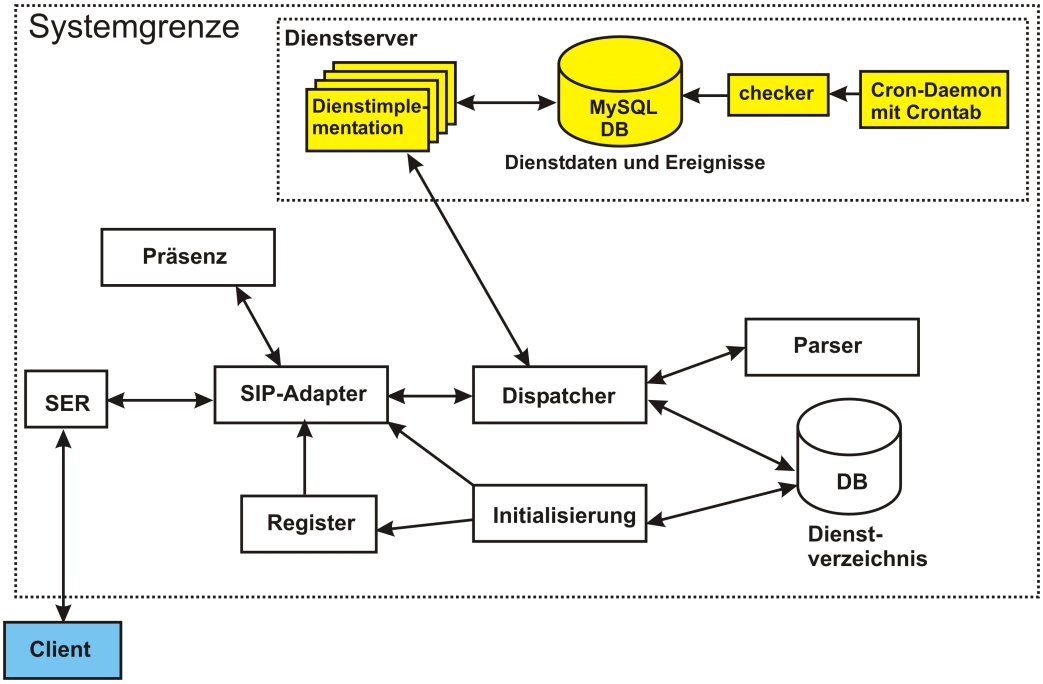

Abbildung 8: Referenzarchitektur mit den wichtigsten Komponenten

Die Architektur besteht aus folgenden Komponenten mit Kurzbeschreibungen:

- **Register:** Jeder Benutzer muss sich bei jedem Einloggen beim Registry (im SIP-SER integriert) anmelden. Diese Registrierung ist nur für ein bestimmtes Zeitintervall gültig. Der PolyPhone-Client des Benutzers meldet sich von Zeit zu Zeit und vor Ablauf des Zeitintervalls beim Registry neu an. Auch Dienste müssen sich beim Registry snmelden und im Zeitintevall neu registrieren.
- **Initialisierung:** Diese Komponente wird beim Start der Dienstplattform aufgerufen. Zur Initialisierung gehören das Verbinden der Plattform mit dem SER in der PolyPhone-Infrastruktur und das Registrieren auf dem Registrar (im SER).
- **SIP-Adapter:** Der SIP-Adapter ist das Tor zur Aussenwelt. Er ist für das Empfangen und Versenden von Anfragen/Antworten zuständig. Dazu werden die vom Benutzer erhaltene SIP-Anfrage in ein geeignetes Format umgewandelt, sowie Antworten für den Benutzer in SIP-Pakete umgewandelt.

- **Dispatcher:** Die Rolle des Dispatchers ist die Koordination aller Komponenten. Er nimmt Anfragen vom SIP-Adapter entgegen und sendet diesem die Antworten zurück, nachdem der Parser, die Dienstimplementation uws. bei der Umsetzung ihren Teil dazu beigetragen haben.
- **Parser:** Jedes SIP-Paket wird vom Parser analysiert und nach bestimmten Inhalten wie Empfängeradresse, Keyword und Parameter, durchsucht. Die gefundenen Parameter gibt er dem Dispatcher für die Weiterverarbeitung zurück.
- **Dienstimplementation:** Jeder Dienst ist durch seine Dienstimplementation definiert. Je nach Dienst wird ein Cron-Daemon für Ereignisse benötigt und/oder ein Speichermedium für stateful-Dienste.
- **Datenbanken:** Datenbanken werden für das Dienstverzeichnis und für Ereignisse und Daten der Dienste benötigt.
  - **Dienstverzeichnis:** Das Dienstverzeichnis enthält alle nötigen Angaben aller Dienste für die Lokalisierung der Dienstimplementationen. Der Dispatcher kann mit Hilfe dieser Datenbank jeden vorhandenen Dienst aufrufen.
  - **Dienstdaten und Ereignisse:** Bei stateful-Diensten müssen gewisse Benutzerdaten gespeichert werden und jederzeit abrufbereit sein. Für Ereignisse müssen zusätzliche Daten wie Ablaufzeit, Benutzer etc. gespeichert werden.
- **Checker:** Die Checker-Komponente überprüft bei jeder Ausführung alle Ereignisdatentabellen auf eintreffende Ereignisse. Wird eine neue Ereignisstabelle erstellt, muss dies unverzüglich der Checker-Komponente mitgeteilt werden, andernfalls wird diese Tabelle nicht berücksichtigt. Tritt ein zeitliches Ereignis ein wird die zuständige Ereignisfunktion innerhalb der Dienstimplementation aufgerufen und ausgeführt.
- **Cron-Daemon:** Mit dem Cron-Daemon können Ereignisse minutengenau gesteuert werden. Diese von UNIX erstellte Zusatzfunktion enthält eine Tabelle (CronTab) und überprüft diese jede Minute auf eintreffende Ereignisse. Bei der Verwendung von Cron-Daemon bei PolyPhone wird eine weitere Komponente, die Checker-Komponente, dazwischen geschaltet um die Verarbeitung der Ereignisse zu erleichtern.
- **Präsenz:** In der Präsenzkomponente werden die Präsenz-Status der Benutzer und der Dienste festgehalten. Dienste können nur bei Online-Präsenz angefragt werden. Wiederum kann es Dienste geben, die bei Statusänderungen der Benutzer reagieren.

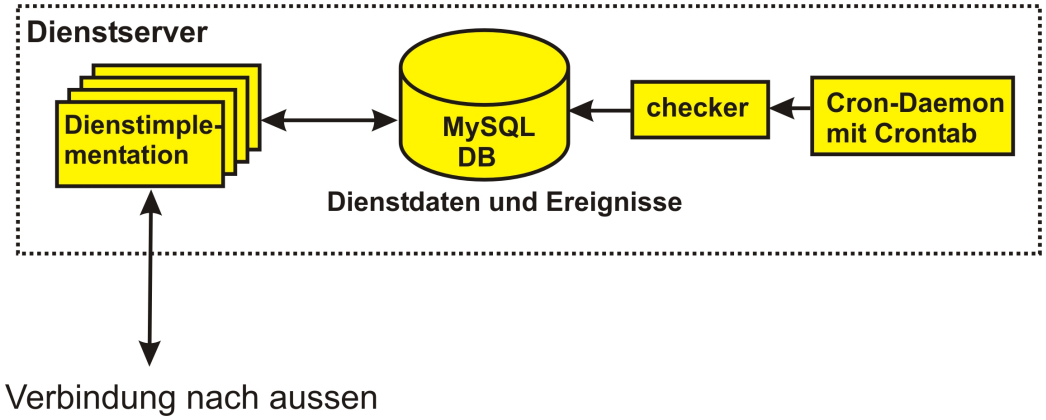

Abbildung 9: Dienstserver mit seinen vier Komponenten

Vier der erwähnten Komponenten stehen im direkten Zusammenhang zueinander und bilden den Dienstserver (Abbildung 9). Sie koordinieren ihren Programmablauf unabhängig von der gewählten Konzeptvariante und werden unverändert beibehalten. Es handelt sich hierbei um folgende vier Komponenten:

- Dienstimplementationen
- Datenbank für Dienstdaten und Ereignisse
- checker-Modul
- Cron-Daemon

Die vorgestellte Referenzarchitektur enthält die wichtigsten Komponenten um eine Plattform für interaktive IM-Dienste zu schaffen. Es müssen nicht alle Komponenten in den Konzeptvarianten[7] enthalten sein. Um die Entscheidung der Komponentenwahl treffen zu können, müssen die Aufgabenbereiche und die Wichtigkeiten der Komponenten bekannt sein. Die einzelnen Komponenten werden in den kommenden Unterkapiteln beschrieben.

[7] Siehe Kapitel 5.3

5.1.2 Komponente „Initialisierung“

Zu Beginn und nach jedem Neustart des Servers muss die Plattform initialisiert werden. Dazu gehört der Aufbau einer Verbindung zum SER sowie das Registrieren am SIP-Registrar. Dieser Registrar ist im SER der PolyPhone-Umgebung integriert und hält Registrierungen nur für einen beschränkten Zeitraum fest. Während diesem Zeitraum muss die Registrierung beliebig oft verlängert werden. Die Initialisierung muss somit sicherstellen, dass die Registrierung der Plattform auch über einen längeren Zeitraum konstant bleibt.

5.1.3 Komponente „SIP-Adapter“

Ein Adapter ist eine technische Einrichtung zur Anpassung verschiedener miteinander verbundenen Komponenten mit unterschiedlichen Schnittstellen. Ein Adapter verpackt und entpackt Daten in ein gewünschtes anderes Datenformat. Mögliche Formate sind beispielsweise SIP, XML oder HTTP. Eine Vereinfachung des SIP-Adapters ist in Abbildung 10 zu sehen. Der im SER implementierte SIP-Adapter konvertiert alle erhaltenen SIP-Pakete in ein neues Format um. Dieses Format wird „sip_msg“ genannt und enthält alle notwendigen Daten aus dem SIP-Paket[8]. Der SIP-Adapter leitet diese Struktur dem Dispatcher für die Weiterverarbeitung weiter. Nach Evaluierung dieses Requests auf der Dispatcher-Seite werden die Antwortelemente (Response) dem SIP-Adapter übergeben. Daraus erstellt der SIP-Adapter ein oder mehrere SIP-Pakete und sendet sie dem PolyPhone-Empfänger zurück.

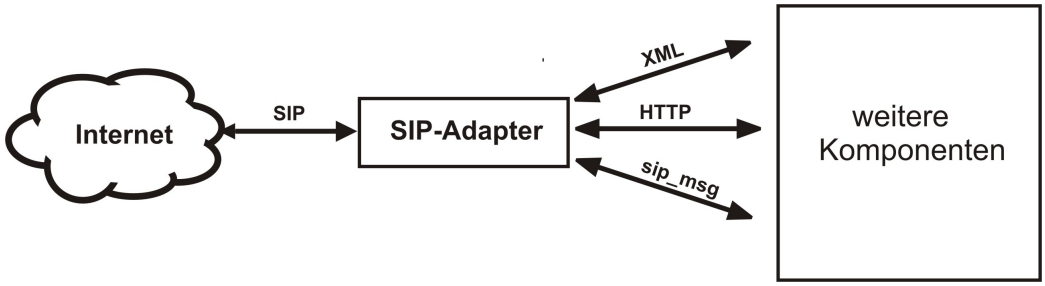

Abbildung 10: SIP-Adapter und seine Schnittstellen

[8]Siehe Kaptitel 6.2

5.1.4 Komponente „Dispatcher“

Der Dispatcher ist das Kernstück der SIP IM-Infrastruktur. Er kommuniziert mit dem Grossteil der Komponenten und koordiniert den Ablauf der Anfragen. Nach der erfolgreichen Initialisierung werden alle Dienstanfragen an den Dispatcher weitergeleitet. Vom SIP-Adapter nimmt er SIP-Pakete in der neuen Struktur „sip_msg“ entgegen. Diese Struktur hat keinen direkten Zugriff auf Dienstangaben im Body, wie zum Beispiel Keyword und Parameter. Die müssen zuerst extrahiert werden. Dazu wird eine neue Struktur erstellt, welche eine Erweiterung der sip_msg-Struktur ist und neu fünf weitere Strukturfelder für Dienstbezeichnung, Dienstkeyword und Parameter enthält. Mit der Parserkomponente, welche nicht mit dem im SER integrierte Parser zu verwechseln ist, werden die Dienstangaben herausgelesen und in die vorgesehenen Strukturparameter gespeichert. Dieser liefert dem Dispatcher die nötigen Informationen aus dem SIP-Paket, welche für die Abarbeitung der Anfrage benötigt werden. Nach Erhalt dieser Informationen überprüft der Dispatcher mit der Präsenz-Komponente, ob ein entsprechender Dienst vorhanden ist (Status = online). Bei positiver Antwort schaut der Dispatcher im Dienstverzeichnis nach dem Pfad zur gesuchten Dienstimplementation[9]. Der Dispatcher ruft nun die richtige Dienstimplementation auf, welche die Antwort als Rückgabewert liefert. Diese Antwort leitet der Dispatcher dem SIP-Adapter weiter, welcher die Antwort als SIP-Pakete dem Empfänger zurück sendet. Der genaue Ablauf einer Dienstanfrage wird in Kapitel 5.2 erläutert.

5.1.5 Komponente „Parser“

Der Parser zerlegt und analysiert den Eingabetext. Der Inhalt wird nach **Dienstbezeichnung, Absenderadresse, Dienstkeyword und Parameter** durchsucht. Diese Daten werden für die Weiterverarbeitung der Anfrage vom Dispatcher benötigt. Im SIP-Header wird die Absenderadresse herausgelesen. Aus dem SIP-Body werden Dienstkeyword und Parameter extrahiert. Die Message im SIP-Body ist die Eingabe des Benutzers für den Dienstaufruf. Jede Message besteht aus einer genauen Abfolge von Wörtern, welche mit Leerschlägen getrennt sind. Mehrere Leerschläge werden vom Parser ignoriert. Die Nachricht besteht aus einem optionalen Dienstkeyword und mehreren optionalen Parametern.

[9] Siehe Kapitel 5.1.8 Datenbanken, Dienstverzeichnis

5.1.6 Komponente „Dienstimplementation“

Jedem Dienst liegt eine Implementation zugrunde. Unabhängig von der Konzeptvariante wird ein Server für alle Dienstimplementationen erstellt. Ein Dienst wird direkt vom Dispatcher aufgerufen und mit den übermittelten Parameter ausgeführt. Es gibt zwei verschiedene Arten von Diensten, die stateful- und die stateless-Dienste, welche aber vom Dispatcher nicht unterschieden werden müssen. In einer Datenbank kann jeder Dienst eine eigene Tabelle für notwendige Daten anlegen. Diese stateful-Dienste speichern Angaben über die Benutzer. Für Dienste, welche auf Ereignisse reagieren, wird eine Ereignistabelle in der Datenbank angelegt. Sie wird vom Cron-Daemon[10] jede Minute auf Ereignisse überprüft.

5.1.7 Komponente „Cron-Daemon“

Der Cron-Daemon läuft im Hintergrund ab und steuert eine zeitliche Jobliste auf Unix- oder Unix-artigen Betriebsystemen. In einer Tabelle, genannt Crontab, können zeitliche Ereignisse mit einem auszuführenden Befehl festgelegt werden. Der Crontab enthält minutengenaue Einträge und wird häufig für die Archivierung von Logdaten eines Systemes benutzt. Diese Einträge werden selten geändert oder gelöscht und führen ihre Befehle zeitgemäss aus. Für Dienstereignisse, wie zum Beispiel ein Weckdienst, werden häufig neue Ereignisse erstellt und eingetroffene Ereignisse gelöscht. Dies kann schnell zu einer sehr grossen Liste von zeitlichen Ereignissen führen. Obwohl es möglich ist die Crontab immer wieder zu modifizieren, wird davon abgeraten. Die bessere und schnellere Variante für Ereignisse wird mit einer Ereignisdatenbank erzielt. Jeder Dienst, welcher Ereignisse anbietet, erhält eine eigene Ereignistabelle in einer Datenbank. Diese Tabellen werden von einem Programm „checker“ nach Ereignissen überprüft. Das „checker“-Programm selbst wird vom Cron-Daemon in kurzen Intervallen (z.B. einmal pro Minute) abgefragt und auch ausgeführt. Bei der Überprüfung der Ereignistabelle vergleicht das „checker“-Programm seine eigene Zeit mit der eingetragenen Ereigniszeit. Trift ein Ereignis ein, wird ein Programm, dessen Link in der Tabelle eingetragen ist, aufgerufen. In der Tabelle können noch weitere benutzereigene Parameter festgehalten werden. Diese Art von Ereignisüberwachung vereinfacht das Einfügen und Löschen von Ereignissen sehr. Der Aufwand, Ereignisse in der Crontab einzutragen, ist um einiges höher als Datenbankabfragen durchzuführen. Der Ressourcenverbrauch des „checkers“ ist dabei minimal.

[10] Siehe nächste Komponente

Im Crontab wird nur ein zusätzlicher Eintrag, nämlich die minutiöse Ausführung des „checkers“, eingefügt.

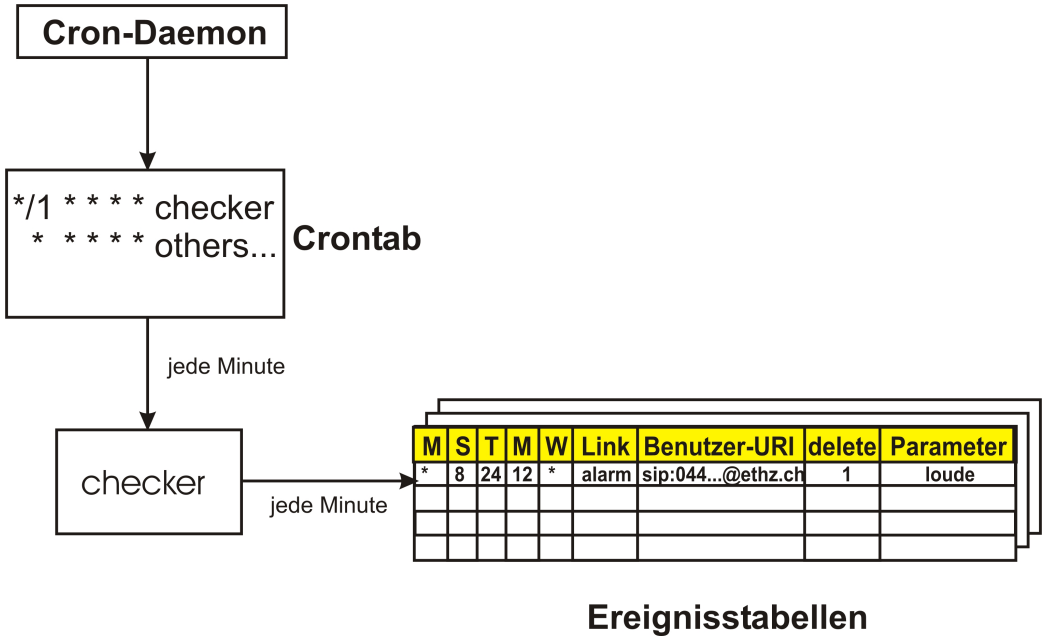

Abbildung 11: Ereignisabfragen mit Cron-Daemon

Die Ereignistabelle, welche in der Abbildung 11 auf der rechten Seite ersichtlich ist, enthält unter anderem folgende Tabellenfelder in der gegebenen Reihenfolge. Sie sind analog zu den Tabellenfelder der Einträge im Crontab.
M: Minutenzahl (0 - 59)
S: Stundenzahl (0 - 23)
T: Tag (1 - 31)
M: Monat (1 - 12)
W: Wochentag (0 - 6, Sonntag=0)
Ein Stern * wird für keine konkreten Angaben benutzt. Die Bezeichnung „*\5“ wird für Ereignisse verwendet, welche je nach Tabellenfeld alle 5 Minuten, 5 Stunden, 5 Tagen etc eintreffen sollen.

5.1.8 Komponente „Datenbanken“

In der Referenzarchitektur werden zwei verschiedene Datenspeicherungen benötigt. Zum einen muss ein Mechanismus für das Dienstverzeichnis vorhanden sein und zum anderen wird eine dienstspezifische Datenspeicherung benötigt.

Dienstverzeichnis:
Für die Speicherung des Dienstverzeichnisses eignet sich am besten eine relationale Datenbank. Es wird eine tabellarische Auflistung der Dienste benötigt. Der Dispatcher tätigt eine Anfrage an die Datenbank um nötige Informationen über einen Dienst zu erhalten. In der Tabelle 3 wird eine mögliche Struktur dargestellt. Neue Dienste werden in diese Tabelle eingefügt und sind neben der physischen Verfügbarkeit (Implementation) nun auch für den Benutzer verfügbar (Präsenz-Status).

ID	Pass-wort	PolyPhone -Bezeichnung	Implementation-Modul	Keyword	Betreiber
0	***	Mensa	mensa		22
1	***	111	auskunft		43
2	***	Mulitcast	mulitcast		11
3	***	0446580001	test	helloworld	22
integer	**string**	**string**	**string**	**string**	**integer**

Tabelle 3: Struktur des Dienstverzeichnisses

Die Spaltenbezeichnungen und die Datentypen der Felder werden folgend im Detail erklärt. Die letzte Zeile repräsentiert die Datentypen jeder Spalte.

ID: Die relationale Datenbank verteilt jedem Tabelleneintrag eine eindeutige ID, ein sogenannter Primarykey. Diese ID ist über die ganze Datenbank universell. Somit können Tabellen als Querverweise (Associations) mit Primarykeys erstellt werden.
Passwort: Nicht alle Dienste sind von Jedermann aufrufbar. Um den Zugriff nur autorisierten Benutzer zu Ermöglichen wird ein Passwort benötigt.
PolyPhone-Bezeichnung: Jede Dienstbezeichnung hat eine allgemeine oder eigene Dienstbezeichnung. Diese Bezeichnung kann aus Zahlen, Buchstaben oder anderen gängigen Zeichen bestehen.
Implementation: In diesem Feld wird der Pfad zur entsprechenden Implementation angegeben.
Keyword: Dienste mit allgemeiner Dienstbezeichnung benötigen ein Dienstkeyword zum den Dienst eindeutig bestimmen zu können.
Betreiber: Für nachträgliche Änderungen und Anpassungen wird eine Verlinkung zum Betreiber erstellt. Ein Betreiber hat ein oder mehrere Dienste erstellt und implementiert. Jeder Betreiber wird einmalig in eine Tabelle mit ID, Name, Vorname und PolyPhone-Nummer eingetragen. Seine ID (Primarykey) kann nun als Foreignkey in die Diensttabelle übernommen werden.

Dienstspezifische Datenspeicherung: Jeder stateful-Dienst benötigt ein Speichermedium für Angaben über Benutzer, Ereignisse etc. Dafür wird eine MySQL-Datenbank erstellt und jedem stateful-Dienst eine oder mehrere Tabellen zur Verfügung gestellt. Für Dienste mit Ereignis-Anbietungen wird eine zusätzliche diensteigene Tabelle erstellt. Diese Ereignistabellen werden minutiös von der checker-Komponente überprüft. Die Darstellungen der einzelnen Tabellen (bis auf die Ereignistabellen) sind unabhängig und können frei gewählt werden. Die Ereignistabelle wurde bei der Cron-Daemon-Komponente erläutert.

5.1.9 Komponente „Präsenz“

Mit der Präsenzkomponente verfügt die Plattform über eine Übersicht der aufgeschalteten, ansprechbaren Dienste und eingeloggten Benutzer. Die Präsenzverwaltung der Dienste wird für die genaue Fehlerbehandlung beim Status „nicht online“ gebraucht. „Nicht online“ bedeutet entweder „offline“ oder „nicht existieren“. Je nach Status wird eine entsprechende Fehlermeldung dem Benutzer zurückgesendet. Diese Überprüfung findet nach dem Parservorgang statt. Nur bei einer positiven Antwort, welche für Dienste mit dem Status „online“ zurückgegeben wird, fährt der Dispatcher ordnungsgemäss fort. Die Status-Angaben der Benutzer werden von möglichen Diensten benötigt. Es kann zum Beispiel von Interesse sein, die ganze online-Zeit eines Benutzers festzuhalten. Dazu wird der entsprechende Dienst auf die Präsenzkomponente zurückgreifen um die nötigen Informationen zu erhalten, respektive wird der zuständige Dienst informiert, wenn sich der Status eines eingeschriebenen Benutzers ändert.

5.2 Request-Response Ablauf

Wie in der Referenzarchitektur in Abbildung 8 ersichtlich ist durchläuft eine Anfrage mehrere Male den Dispatcher. Er koordiniert die ganze Abfrage und leitet die Frage an mehrere Module weiter. Der Ablauf einer Frage bis zur Antwort wird in Abbildung 12 schrittweise aufgezeigt.
Jede ankommende Frage eines Benutzers wird vom SIP-Adapter aufgenommen. Dieses SIP-Paket durchläuft als erstes den im SER vorhandenen Parser. Dieser konvertiert das SIP-Paket in die sip_msg-Struktur und übergibt dies dem Dispatcher. Um die nötigen Dienstdaten aus diesem Struct zu erhalten,

leitet ihn der Dispatcher dem Parser weiter. Nach dessen Durchlauf sind diese Daten im Struct enthalten. Nun kennt der Dispatcher den aufgerufenen Dienst. Zur Überprüfung dessen Präsenz befragt er das Präsenz-Modul. Nur bei positiver Antwort wird die Frage weiterverarbeitet. Andernfalls wird eine Fehlermeldung zurückgesendet. Nach erfolgter Präsenz-Anfrage holt der Dispachter die Implementationsadresse des Dienstes aus dem Dienstverzeichnis. Diese Implementation ruft der Dispatcher mit den nötigen Parametern auf und wartet auf dessen Antwort. Als Schlussaufgabe des Dispatchers ist es die Weiterleitung dieser Antwort an den SIP-Adapter, welcher die sip_msg-Struktur in ein SIP-Paket konvertiert und dem Benutzer zurück sendet.

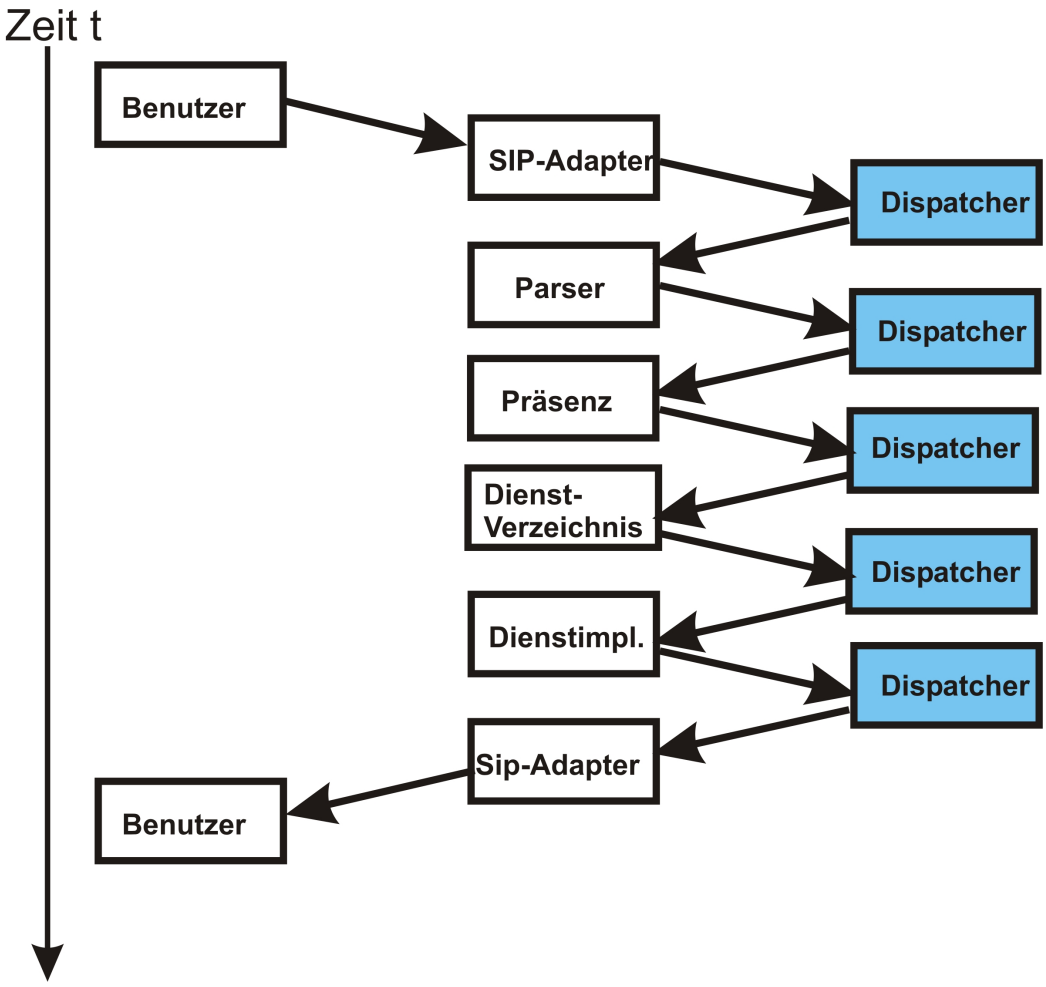

Abbildung 12: Abfolge von SIP-Requests

5.3 Konzeptvarianten

Von der Ausgangslage der Referenzarchitektur aus werden nun zwei Varianten für die Erweiterung der PolyPhone-Infrastruktur vorgestellt. In der ersten Konzeptvariante wird die vollständige zusätzliche Infrastruktur als Modul im SER implementiert. Dadurch fällt überflüssige Kommunikation zwischen SER und Dienstmodul weg. Für die Trennung der existierenden PolyPhone-Infrastruktur mit der neuen Dienstangebot-Infrastruktur wird eine externe Konzeptvariante vorgestellt. Die Unabhängigkeit vom SER benötigt jedoch zusätzliche Kommunikation. Diese Variante der Software kann auf einem externen Server für offizielle Dienste oder für private Dienste eines Benutzers auf einem privaten Rechner angeboten werden.

5.3.1 Variante 1: Dienstplattform als Modul im SER

Die Dienstplattform für Dienstangebote wird als Modul im SER eingebettet wobei der Dienstserver als eigenständiger Server gehalten wird. SER ist der SIP-Express-Router und der Proxy-Server der PolyPhone-Infrastruktur[11]. Die neue Architektur wird in Abbildung 13 dargestellt. Es werden alle vorgestellten Komponenten aus Kapitel 5.1.1 in die Architektur integriert, wobei der SIP-Adapter, die Präsenz und der Parser als Modul im SER schon vorhanden sind.

Die verwendeten Komponenten werden nun spezifisch für die ausgewählte Plattform definiert:

Initialisierung: Bei der ersten Aufschaltung der Software und nach jedem Neustart muss die Software neu initialisiert werden. Dafür muss auf dem SER ein Eintrag in der Konfigurationsdatei „ser.cfg“ in die Liste der zu ladende Module eingefügt werden. Nur bei einer erfolgreichen Registrierung ist die Plattform betriebsbereit. Vor der Initialisierung werden alle Dienstanfragen unbeantwortet zurückgeschickt. Erst nach der Initialisierung können die Dienstanfragen von dieser neuen Infrastruktur bearbeitet werden. Es muss eine stetige Neuregistrierung stattfinden, da eine Registrierung nur für einen gewissen Zeitabschnitt gültig ist.

[11] Siehe Abbildung 1

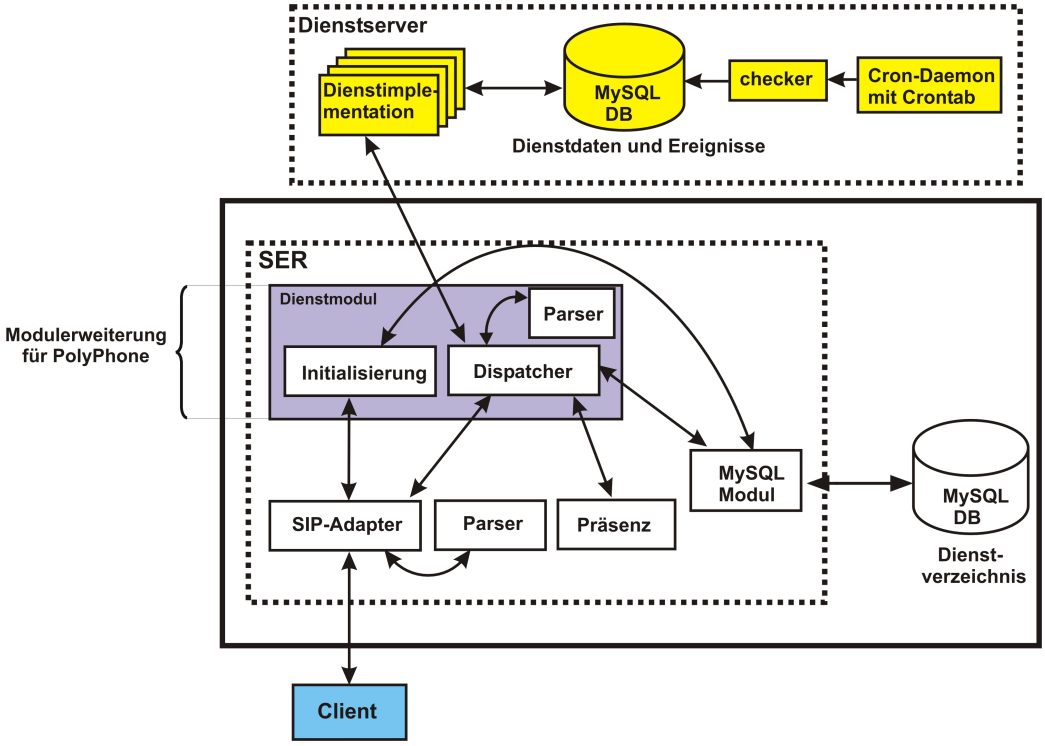

Abbildung 13: Variante 1: Plattform im SER integriert

SIP-Adapter: Der SIP-Adapter kann ohne weitere Anpassungen als Modul vom SER übernommen werden. Der SIP-Adapter empfängt alle Dienstanfragen. Bevor er die Nachricht weiterleitet, wird sie vom SER-Parser in ein sip_msg-struct konvertiert. Erst in dieser Form wird die Nachricht der Dienstarchitektur weitergeleitet. Erhaltene Antworten verpackt der SIP-Adapter in genormte SIP-Pakete und leitet sie dem Empfänger weiter.

Dispatcher: In dieser Variante benötigt der Dispatcher keine Ergänzungen zum Dispatcher der Referenzarchitektur.

Parser: Im SER ist schon ein Parser vorhanden, welcher aber den Body einer SIP-Nachricht nicht beachtet. Im SIP-Body sind alle nötigen Informationen, um eine Dienstanfrage zu starten. Aus diesem Grund wird ein kleiner Parser auf dem Dienstmodul implementiert, so wie er in der Referenzarchitektur vorgestellt wurde. Die Aufgabe dieses Parsers ist lediglich den SIP-Body nach Keywords und Parameter zu durchsuchen.

Präsenz: Das Präsenz-Modul wird vom SER zur Verfügung gestellt und muss nicht mehr neu implementiert werden. Während eines Dienst-Requests wird der Dispatcher dieses Modul nach dem Präsenz-Status des Dienstes anfragen.

Dienstverzeichnis: Für die Auflistung aller Dienste mit ihrer Lokalisierung wird die Datenbankstruktur, wie sie in der Referenzarchitektur erklärt worden ist, verwendet. Es fallen keine spezifischen Änderungen an.

Dienstserver: Die Dienstimplementationen sowie Cron-Daemon und Dienstdatenbank werden ausserhalb des SER auf einem eigenen Server angesiedelt. Grund dafür ist die klare Trennung vom SER-Proxy und den Dienstimplementationen. Es werden laufend neue Dienste erstellt, existierende Dienste abgeändert und gelöscht. Die dazugehörige MySQL-Datenbank wird einer ständigen Änderung ausgesetzt sein.

Bemerkung: Mit dieser Variante entfallen viele Doppelimplementationen. Die Module SIP-Adapter, Parser und MySQL sind im SER schon vorhanden und können vom Dienstmodul benutzt werden. Die Aufbau- und Testphase werden eine schwierige Situation darstellen da der SER und die PolyPhone-Infrastruktur sich schon in Betrieb befinden. Diese Plattform darf natürlich nicht ungewollt beeinträchtigt werden. Es dürfen keine gröberen Fehler geschehen, was zu einem Absturz der ganzen Infrastruktur führen könnte. Ist die Dienstinfrastruktur einmal vollständig im SER integriert, kann mit einem reibungsfreien Betrieb gerechnet werden. Diese Konzeptvariante verlangt als Programmiersprache C.

5.3.2 Variante 2: Dienstplattform auf externem Server

Die ganze Infrastruktur wird auf einem externen Server installiert und in Betrieb genommen. Extern bedeutet unabhängig von den bisherigen PolyPhone-Komponenten. Es wird ein Programm mit verschiedenen Klassen als Komponenten erstellt. Dies vereinfacht den Informationsaustausch zwischen den einzelnen Komponenten. Somit genügen Funktionsaufrufe innerhalb der Infrastruktur. Wie in Abbildung 14 ersichtlich wird die Infrastruktur nur an zwei Stellen mit anderen Programmen verbunden. Dort wird die Kommunikation mit Sockets realisiert. Der Ablauf eines Request wird in der Referenzarchitektur in Abbildung 12 erläutert. Innerhalb der Infrastruktur werden mit einfachen Funktionsaufrufen Informationen ausgetauscht. Zwischen SER-SIP-Adapter und Dienst-SIP-Adapter wird über TCP/IP und Sockets gesprochen. Im Gegensatz zur ersten Konzeptvariante müssen hier alle Komponenten von Grund auf neu implementiert werden. Der Server wird mit einem UNIX-Betriebsystem laufen. Gründe dafür ist die einfachere und schnellere Administration des Servers, auch über Fernwartung (SSH), die grösser Sicherheit, schnelle verfügbare Software-Updates, was wiederum für die Sicherheit spricht und herstellerunabhängig. Auch ist der Gebrauch von UNIX ressourcen-schonend, es braucht auf dem Server kein Gui, wie es bei Windows der Fall ist. Die Programmierumgebung und Programmiersprache kann hier frei gewählt werden (Java, C, C++ etc).

Initialisierung: Die Initialisierungs-Komponente muss sich bei dieser Variante intervallweise neu registrieren. Eine Registrierung ist nur für eine bestimmte Zeitdauer gültig, kann aber beliebig oft verlängert werden.

SIP-Adapter: Alle Pakete werden vom SER-SIP-Adapter konvertierte und an die externe Dienstplattform gesendet. Daher ist hier der Gebrauch eines eigenen SIP-Adapters überflüssig. Er wird aber trotzdem Verwendet für die Verbindung zum SER-SIP-Adapter der bestehenden PolyPhoneplattform. Die Initialisierung sowie der Dispatcher leiten ihre Pakete an den eigenen SIP-Adapter weiter. **Dispatcher:** Der Dispatcher, wie er in der Referenzarchitektur erläutert wurde, kann so übernommen werden. Der Informationsaustausch kann im Gegensatz zur ersten Konzeptvariante mit einfachen Funktionsaufrufen stattfinden. Dies vereinfacht die Implementierung dieser Komponente um einiges.

Parser: Obwohl jede Nachricht durch den SER-Parser gelangt, benötigt es trotzdem einen eigenen Parser für Dienstanfragen. Dieser analysiert den Body einer SIP-Nachricht, welcher unangetastet bleibt beim ersten Parser. Der

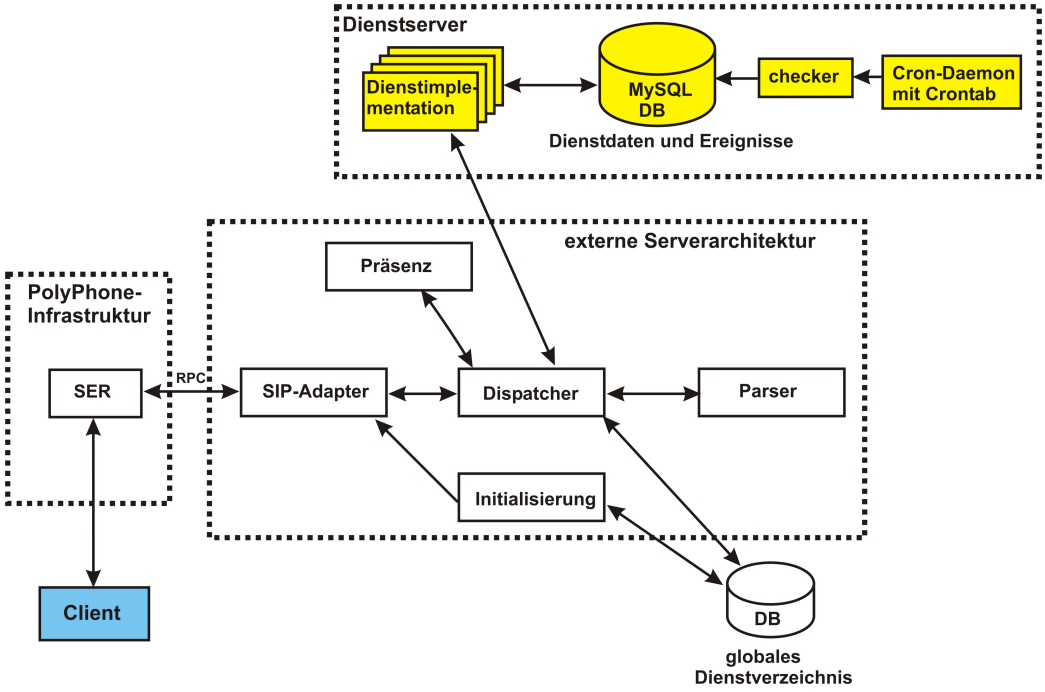

Abbildung 14: Externe Konzeptvariante

Parser in der Referenzarchitektur berücksichtigt dies und kann so übernommen werden.

Präsenz: Die Präsenzkomponente ist auf dem PolyPhone-Server im SER enthalten und muss von dort übernommen werden. Für jede Dienstanfrage wird deren Präsenz überprüft. Da die Präsenzen der Dienste und der Benutzer in der gleichen Komponente gehalten werden kann diese Komponente nicht ausgelagert werden.

Dienstverzeichnis: Falls keine Redundanz gewünscht wird und nur ein Dienstserver auf geschaltet wird, kann das Dienstverzeichnis in einer lokalen relationalen Datenbank auf dem Dienstserver gespeichert werden. Bei Verwendung von mehreren Dienstservern, muss das Dienstverzeichnis global gespeichert sein und von allen Dienstservern ansprechbar sein.

Dienstserver: Der Dienstserver kann unverändert von der Referenzarchitektur übernommen werden. Es müssen keine weiteren Anpassungen durchgeführt werden.

Privatanbieter: Eine weitere Möglichkeit bietet sich nun für Privatanbieter an. Die Software der externen Konzeptvariante kann auf jedem beliebigen Rechner, darunter versteht man auch private Rechner, installiert und betriebsbereit gemacht werden. Somit kann jeder PolyPhone-Benutzer eine eigene Dienstplattform anbieten. Alle Anfragen an Dienste dieser Plattform werden mit der Benutzer-PolyPhone-Nummer und einem Dienstkeyword identifiziert. Das Dienstverzeichnis wird global gespeichert sein um alle Dienstinformationen einheitlich und in einer Datenbank zu speichern.

Bemerkung: Die Implementierungsphase ist bei dieser Variante einfacher. Es können Schrittweise Komponenten implementiert und getestet werden (zum Beispiel mit Extrem Programming). Für die existierende PolyPhone-Infrastruktur stellt diese Variante keinerlei Gefahr dar. Es sind zwei getrennte Plattformen und sind nur via TCP/IP miteinander verbunden. Ein weiterer Vorteil ist der Aufbau der ganzen Infrastruktur. Es kann ein Programm mit mehreren Klassen erstellt werden um die gleiche Funktionalität zu erreichen. Der Informationsaustausch zwischen den Komponenten ist auf diese Art und Weise sehr einfach mit Funktionsaufrufen zu realisieren. Jedoch müssen auch SIP-Adapter und Parser neu implementiert werden. Diese Konzeptvariante lässt die Entscheidung für Redundanz offen. Es wäre möglich die Software auf mehreren Servern in Betrieb zu nehmen. Aber im Normalfall ist besitzt ein Server genug Ressourcen um alle Dienstabfragen zu bearbeiten.

5.4 Entscheidung

In der Implementierungsphase wird nur eine Konzeptvariante verwendet. Um abzuwägen, welche Variante die Bessere ist, wird mit Hilfe der nachfolgender Tabelle eine Übersicht geboten. Es werden einige Merkmale der beiden Varianten aufgelistet und mit den folgenden Zeichen gewichtet:

- -: Sehr schlecht

-: Schlecht

+: Gut

++: Sehr gut

	Variante 1	Variante 2
Testing	- -	++
Redundanz	-	+
Privat Angebot	-	+
Unabhängige Programmiersprache	-	+
Unabhängiges Betriebsystem	-	+
Implementierungs-einfachheit	-	+
Kleine Komponenten-anzahl	+	-

Tabelle 4: Entscheidung für Konzeptvariante

Für die Realisierung der Dienstanbietung über PolyPhone wird die zweite Konzeptvariante mit dem externen Server gewählt. Diese Wahl vereinfacht den Programmaufbau, die Implementierung, die Integrierung und das Testen. Obwohl mehrere Komponenten implementiert werden müssen, ist es dennoch die einfachere Variante. Da ein einziges Programm für die Dienstanbietung implementieren kann wird auch die Kommunikation zwischen den einzelnen Komponenten vereinfacht. Jede Komponente wird als eigenständige Klasse dargestellt.

6 Implementierung

Für die PolyPhone-Dienstanbietung wird die zweite Konzeptvariante als Grundlage dienen. Die Implementierung wurde noch nicht durchgeführt, enthält aber einige Klassen-Signaturen und technische Angaben, welche empfehlend verwendet werden sollen.

6.1 Technische Angaben zu Komponenten

Dispatcher: Der Dispatcher enthält vier Funktionen: main(), newRequest(), convToOld() und convToNew(). Die main-Funktion wird bei der Initialisierung aufgerufen. Die newRequest-Funktion wird vom SIP-Adapter aufgerufen. Der Dispatcher bleibt in dieser Funktion, bis er die Antwort dem SIP-Adapter zurückgesendet hat. Falls es nur eine Instanz dieser Klasse gibt ist der Dienst bis zur Rückantwort geblockt. Abhilfe wird durch die objektorientierte Programmierung gegeben. Für jede neue Anfrage wird eine Instanz dieser Klasse erzeugt. Nach Beendigung der Anfrage wird das Objekt wieder gelöscht. Die beiden Funktionen convToOld() und convToNew() konvertieren von der erhaltenen sip_msg-Struktur in die neue sip_new_msg-Struktur, welche fünf weitere Felder für Dienstname, Dienstparameter und SIP-Body enthält und umgekehrt. In der Tabelle 5 wird das Interface des Dispatchers beschrieben.

Parser Da der Parser keine Kenntnis über den Dienst und seine eventuellen Keywords und Parameter hat, behandelt er diese Angaben als „words“ (word1 bis wordn):

Message: [word]*

Dies vereinfacht die Arbeit des Parsers um einiges. Er liest das erste Wort bis zum ersten Leerschlag und speichert es in einer lokalen Variable word1. Die folgenden Leerschläge (falls der Benutzer mehrere eingegeben hat) ignoriert der Parser und beginnt beim zweiten Wort weiterzuparsen. Dieses wird in word2 gespeichert. Je nach Dienst gibt es eine unterschiedliche Anzahl von Wörtern. Der Parser überprüft nur die Syntax des Inhalts mit entsprechenden Fehler-Meldungen. Die Semantik wird nicht berücksichtigt und muss vom jeweiligen Dienst-Programm abgehandelt werden. Nach Ende der Durchlaufphase (parsen) werden die gewonnenen Daten in den fünf zusätzlichen Feldern der neuen sip_msg-Struktur gespeichert und dem Dispatcher zurückgegeben.

Funktion	Beschreibung
main()	Hauptfunktion, welche beim Start des Modules aufgerufen wird
newRequest(struct sip_msg_old* msg_old)	Diese Funktion wird vom SIP-Adapter aufgerufen um eine neue Anfrage zu übergeben. Der Rückgabewert ist die Antwort der Anfrage in der ursprünglichen sip_msg-Struktur. Diese Funktion bearbeitet eine Dienstanfrage bis zum Schluss, d.h. sie ruft folgende Module auf: Parser, Präsenz, MySQL-Modul und Dienst-implementation. Für die Konvertierungen von der ursprünglichen Struktur zur Neuen werden die beiden nachfolgenden Funktionen verwendet.
convToOld(struct sip_msg_old* msg_old)	Konvertiert von der erhaltenen sip_msg-Struktur in die neue sip_msg_new-Struktur, welche gleich als Rückgabewert übergeben wird.
convToNew(struct sip_msg_new* msg_new)	Konvertiert von der neuen Struktur in die ursprüngliche sip_msg-Struktur

Tabelle 5: Interface des Dispatchers

Funktion	Beschreibung
parse(struct sip_msg* msg) parse(struct sip_msg* msg, string sep)	Die parse-Funktion (mit oder ohne Trennzeichen „sep“) durchsucht die erhaltene ursprüngliche SIP-Nachricht nach Dienstinformationen. Gefundene Daten werden der Reihe nach in die bereitgestellten words gespeichert. Rückgabewert ist die sip_msg-Struktur.
get_body(struct sip_msg* msg)	Extrahiert den Body-Text aus einem SIP-Paket und wird von der parse()-Funktion aufgerufen.

Tabelle 6: Interface des Parsers für Caller (nach Aussen)

Wird nur eine Instanz des Parsers zur Laufzeit generiert, ist der Parser für den ganzen Vorgang geblockt. Im Falle einer objektorientierten Lösung können mehrere Instanzen generiert werden, welche nach Beendigung vom Garbage-Collector gelöscht werden. Tabelle 6 beschreibt das Interface der Parser-Komponente.
Um auf die ursprüngliche SIP-Nachricht zugreifen zu können wird eine Funktion „buf“ zur Verfügung gestellt. Wird sie auf ein SIP-Paket angewendet (sip_msg.buf) erhält man das unveränderten SIP-Paket.

Dienstserver: Die Kommunikations zwischen Dispatcher und dem Dienstserver wird über eine normale TCP-Verbindung statt finden. Die Portnummer kann eine beliebige Zahl > 1000 sein. In Abbildung 15 ist der Dienstserver mit seinem Server-Socket für die Kommunikation zum Dispatcher abgebildet.

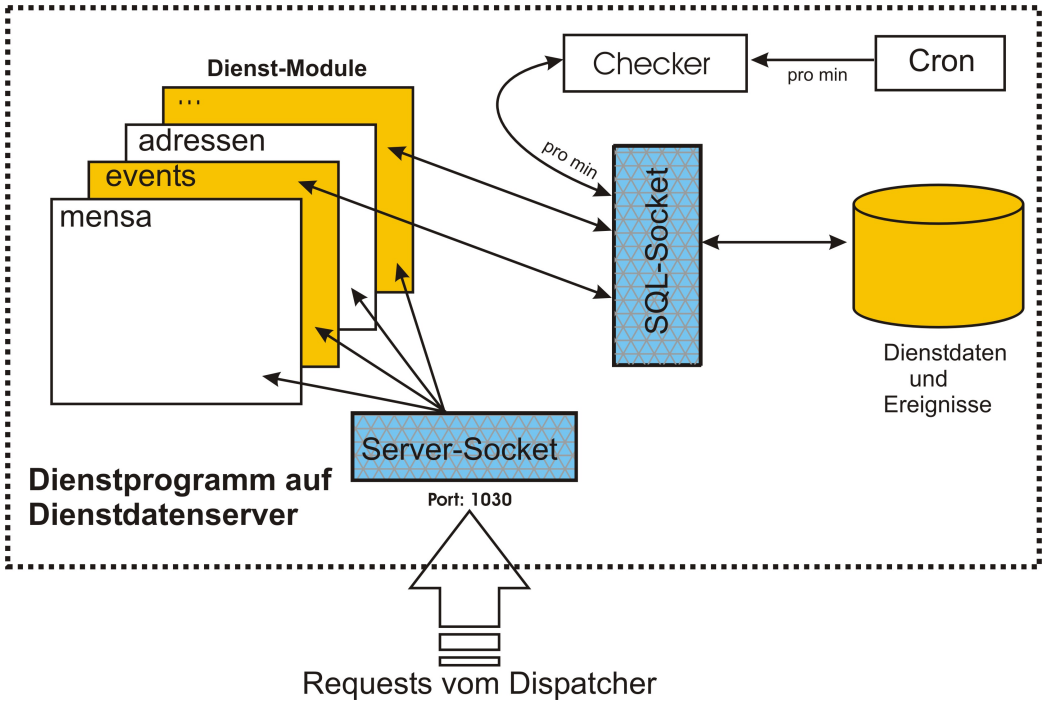

Abbildung 15: Architektur auf dem Dienstimplementations-Server

Um das Socket-Modul zu realisieren muss eine listener- und eine send-Funktion implementiert sein. Die listener-Funktion wird benötigt, um Anfragen des Dispatchers jederzeit entgegen nehmen zu können. Die send-Funktion dient lediglich für das Übermitteln der Antwort an den Dispatcher. Alle Anfragen werden über den gewählten Port >1000 empfangen und dem entsprechenden Dienst weitergeleitet. Zwischen den Diensten und der Datenbank wird ein SQL-Socket eingesetzt. Diese Datenbank wird von stateful-Diensten für die Speicherung von nötigen Informationen des anfragenden Benutzers benötigt. Jeder stateful-Dienst erhält eine oder mehrere Tabellen. Müssen zusätzlich noch Ereignisse erkannt werden erhält jeder Dienst eine eigene Ereignistabelle. Jede Dienstimplementation muss eine Hilfsfunktion „help“ enthalten. Diese Funktion reagiert bei jeder Fehlinterpretation der SIP-Message oder bei Eingabe des Parameters „help“ durch den Benutzer. Das Schema einer Dienstimplementation wird nun festgelegt:

Funktion	Beschreibung
execute(Parameterlist)	Führt Dienst aus
help()	Jeder Dienst enthält die „help“-Funktion, welche die Funkionalität des Dienstes zusammengefasst als Text im PolyPhone-Client wiedergibt.
ereignis()	Optional: Falls ein Dienst eine Ereignisüberwachung offeriert, so enhtält sie eine Ereignis-Funktion, die bei Eintreffen eines Ereignisses aufgerufen wird.

Tabelle 7: Interface eines Dienstes

6.2 sip_msg-Struktur

Der im der PolyPhone integrierte SIP-Adapter konvertiert alle ankommenden SIP-Pakete in eine „sip_msg“-Struktur um. Diese Struktur kann weiterverwendet werden um Dienste anzubieten. Die einzelnen Informationen wie Header, Absender-, Empfängeradresse etc. können einfach aus dieser Struktur gelesen werden. Die vollständigen Strukturdefinition kann bei iptel[12][37] nachgelesen werden. Dort werden noch mehrere hilfreiche Strukturen genau beschrieben.

[12]iptel stellt Informationen und Ressourcen rund um SIP-Kommunikation zur Verfügung

struct **sip_msg**	
unsigned int id;	*Nachrichten-ID, universell pro Prozess*
struct hdr_field* headers;	*SIP-Header*
struct ip_addr src_ip;	*Absenderadresse*
struct ip_addr dst_ip;	*Empfängeradresse*
string * buff	*Buffer mit ganzem SIP-Packet*

7 Evaluation

Die Anforderungen aus Kapitel 2 werden mit der gewählten Konzeptvariante abgedeckt. Im Folgenden werden die einzelnen Anforderungen anhand der Konzeptvariante ausgewertet:

Verfügbarkeit: Die physikalische Verfügbarkeit wird durch das Hochladen des implementierten Dienstes auf den Dienstserver gewährleistet. Dieser Dienst ist aber noch nicht betriebsbereit. Es muss noch einen Status zugefügt werden. In der Konzeptvariante wird das mit dem Dienstverzeichnis in der Datenbank und dem Präsenzmodul durchgeführt. Ersteres verlinkt den physischen Dienst mit der Plattform und das Zweit genannte ordnet dem Dienst einen gewissen Status zu. Mögliche Status sind: available, busy, offline.

Beschreibung: Mit jeder Implementierung eines Dienstes wird auch eine Beschreibung des Dienstes mitgeliefert. Als Standardfunktion „help“ liefert jeder Dienst eine ausführliche Beschreibung und Anleitung, wie man den aufgerufenen Dienst benutzen kann.

Verzeichnis: In der lokalen oder globalen Datenbank (je nach Redundanzstufe) wird ein Verzeichnis der Dienste festgehalten. Durch einen gegebenen Dienstnamen, der aus der SIP-Nachricht geparst wird, kann zur Implementation dieses Dienstes verlinkt werden.

Syntaxen: In der Dienstverzeichnisdatenbank wird die Syntax des Dienstaufrufes festgehalten. Dort wird erwähnt, ob der Dienst noch ein zusätzliches Keyword benötigt, oder ob er eine eigene PolyPhone-Nummer zugeteilt bekommen hat.

Anbieten: Ein Administrator wird die ganze Infrastruktur warten. Er ist auch zuständig für das Einfügen und Entfernen von Diensten. Somit kann jeder Benutzer einen Dienst implementieren und dem Administrator weiterleiten. Dieser ist dann zuständig für das Funktionieren der Dienste. Der Benutzer muss sich damit nicht umschlagen.

8 Schlussfolgerungen und offene Arbeiten

Die Anforderungen an die neue Dienstplattform wurden sehr ausführlich erarbeitet. Zusammen mit der Analyse und den Konzeptvarianten wurde diese Arbeit grösser als geplant. Es wurde viel Wert darauf gelegt, dass der wichtige Teil dieses Projekts (Konzept) gut strukturiert wird und als Grundlage für die Weiterführung des Projektes dienen kann. Die Implementation wurde in Angriff genommen, was aber beim Verbindungsaufbau zum SER sehr zeitintensiv war und einige Probleme mitsich brachte. Die Probleme lagen in der Verwendung eines fehlerhaften Clients. Aus diesem Grund wurde die Implementation nur im theoretischen Sinn zu Ende geführt und bedarf einer praktischen Implementierung. Es stellte sich heraus, dass die reine Implementation der vorgeschlagenen Konzeptvariante viel Zeit in Anspruch nimmt. Offene Arbeiten sind die ganze Implementation und Testen der gewählten Konzeptvariante.

9 Quellenangaben und Internetadressen

Literatur

[1] **Wikipedia:** Instant Messaging, IM
http://en.wikipedia.org/wiki/Instant_messaging

[2] **ETH Zürich:** PolyPhone
http://www.polyphone.ethz.ch

[3] **Wikipedia:** Session Initiation Protocol, SIP
http://en.wikipedia.org/wiki/Session_Initiation_Protocol

[4] **IETF:** Simple Mail Transfer Protocol, SMTP RFC 821
http://www.ietf.org/rfc/rfc0821.txt

[5] **IETF:** HTTP RFC 2616
http://www.ietf.org/rfc/rfc2616.txt

[6] **IETF:** User Datagram Protocol, UDP RFC 768
http://www.ietf.org/rfc/rfc768.txt

[7] **IETF:** Session Description Protocol, SDP RFC 4574:
ftp://ftp.rfc-editor.org/in-notes/rfc4574.txt

[8] **IETF:** Realtime Transport Protocol, RTP RFC 3550:
ftp://ftp.rfc-editor.org/in-notes/rfc3550.txt

[9] **ETH Zürich:** Adressbuch:
http://www.ethz.ch/people

[10] **ETH Zürich:** StuZ2 Partyraum:
http://www.vseth.ethz.ch/stuz2.php

[11] **Wikipedia:** Extensible Messaging and Presence Protocol, XMPP
http://en.wikipedia.org/wiki/XMPP

[12] **IETF:** Extensible Markup Language, XML RFC 3858:
ftp://ftp.rfc-editor.org/in-notes/rfc3858.txt

[13] **Wikipedia:** Jabber:
http://de.wikipedia.org/wiki/Jabber

[14] **Wikipedia** SIMPLE:
http://en.wikipedia.org/wiki/SIMPLE

[15] **IP-Phone-Wiki:** SIP-Status-Codes
http://wiki.ip-phone-forum.de/voip:sip:start

[16] **ETH Zürich:** Mensa:
http://www.mensa.ethz.ch

[17] **ETH Zürich:** myETH:
http://www.myeth.ethz.ch

[18] **ETH Zürich:** VSETH:
http://www.vseth.ethz.ch

[19] **ETH Zürich:** News des Informatikdepartements:
http://inf.ethz.ch/news/news

[20] **ETH Zürich:** Rauminformationen der ETH Zürich:
http://www.rauminfo.ethz.ch

[21] **ETH Zürich:** Agenda vom VSETH:
http://www.vseth.ethz.ch/agenda.php

[22] **ETH :** Bus Zentrum Hönggerberg:
https://www.dozent.ethz.ch/pages/de/verbindungen.jsp

[23] **ZVV-Fahrplan:** Bus Zentrum Hönggerberg:
http://fahrplan.zvv.cht

[24] **jSIP** Java Library for SIP-IM
http://jsip.sourceforge.net

[25] **GNU:** SIP Library in C:
http://www.gnu.org/software/osip/

[26] **Debian:** SIP Library in C:
http://packages.debian.org/unstable/libs/
libsofia-sip-ua-glib0

[27] **Sipfoundry:** SIP Library in C++:
http://www.sipfoundry.org/reSIProcate/

[28] **AOL:** AOL's Instant Messenger
http://aim.com

[29] **AOL:** Botlist:
http://aimtoday.aim.com/features/main_redesign.adp

[30] **it-observer:** Botlist
http://www.it-observer.com/news/5884/secure_developers_portal_instant_messaging_bots/

[31] **Habotat:** create your own buddies
http://hyperrealm.com/main.php?s=habotat

[32] **Zion** JBuddy Client, Server, HelpDesk
http://www.zion.com/

[33] **Resiprocate** Library for SIP inclusive IM:
http://www.zion.com/

[34] **OpenSer:**
http://www.openser.org/

[35] **ETH Zürich:** mySQL Server, Webaufbau:
http://mysqlweb.ethz.ch

[36] **ETH Zürich:** SMS-Projekt der ETH Zürich
http://www.sms.ethz.ch

[37] **IP Telefonie:** Strukturdefinitionen:
http://old.iptel.org/ser/doc/serdev/serdev.html#ROUTING-ENGINE

A Interviews

Durch die Interviews über PolyPhone und seine möglichen Dienste habe ich sehr viele hilfsreiche Informationen von meinen Interviewpartnern bekommen. Bei Fragen und dem Aufsetzen von Servern kann ich mich jederzeit an Francesco Piovano wenden. Armin Brunner ist der Projektleiter von PolyPhone und steht mir bei allfälligen Fragen zu PolyPhone jederzeit zur Verfügung. Ein ähnliches System (SMS-Tool mit Gateway und Connector) hat Herr Wittmann bereits implementiert und steht mir bei Fragen zu Gateway, Connector, Java und Linux zur Seite.

A.1 A. Brunner

Datum: 7. August 2006
Ort: RZ G16

Armin Brunner ist Projektleiter von PolyPhone. Er kennt sich aus mit dem Gebrauch von Servern und Infrastruktur. Falls ich in diesem Bereich Ideen und Vorschläge benötige, kann ich mich jederzeit mit ihm wieder in Verbindung setzen. Er erklärte mir die Idee mit den SIP-Nachrichten, welche mit der SIP-Extension „MESSAGES“ direkt in einem SIP-Paket versendet wird. Im Falle von Dienstanbietungen benötigt man definierte Schlüsselwörter, besser bekannt als Keywords. Um diese Keywords auseinander zuhalten benötigt man einen definierten Trenner. Als Trenner hat sich das Leerzeichen „ „ und der Punkt durchgesetzt. Es sei mir überlassen, wie ich das bei meiner Semesterarbeit definieren möchte. Des Weiteren erwähnte er den Keywordbaum, ein so genannter Entscheidungsbaum, welcher alle Keywörter hierarchisch darstellt. An der ETH kann der aber sehr komplex sein. Aber dass müsse man mit weiteren Personen zuerst absprechen wie dieser Keywordbaum aufgebaut sein sollte und ob überhaupt solch ein Entscheidungsbaum für mein Projekt nötig ist. Wir diskutierten auch über den Gateway, welcher vor die Service-Applikationen geschaltet ist. Der Gateway muss nicht wissen, ob der Dienst stateless / stateful oder eine andere Art von Dienst ist. Zusammen mit dem Gateway wird ein Connector benötigt. Als Referenz gab Herr Brunner mir die E-Mail-Adresse von Herr Wittmann, da dieser den HTTP-Connector für das ETH-SMS-Tool erstellt hat. Herr Brunner meint, dass ich mir das auch überlegen solle, ob ich den HTTP-Connector verwenden möchte. Es würde eine Schnittstelle ergeben, wobei man die PolyPhone-Dienste über PolyPhone und eventuell auch über das SMS-Tool verwenden könnte. Er fände eine Zusam-

menarbeit mit den SMS-Projektleuten sehr sinnvoll um eine Verschmelzung beider Projekte voranzutreiben. Für meine Semesterarbeit wäre dies aber ein zu grosser Aufwand, sollte aber als mögliche Ergänzung angesehen werden. Mit der Zusammenarbeit meint er, dass man vorhandene Dienste per SMS anfragen kann, da man ja nicht immer online mit PolyPhone ist. Somit würde sich eine Art Transparenz und Unabhängigkeit vom Internetzugang ergeben.

Armin Brunners Ideen für mögliche Dienste:

- Mulitcast messages mit Buddyliste
- Message an E-Mail-Adresse schicken
- Message an Mobiltelefon schicken

A.2 Professor B. Plattner

Datum: 7. August 2006
Ort: ETZ G89

Ich stellte meine Semesterarbeit Herr Plattner vor. Wir diskutierten hauptsächlich über mögliche Dienste und wie sie interpretiert werden. Es ist auch zu überlegen, wer das Zielpublikum für mögliche Dienstentwicklung ist. Ob man spezielle Programmierfähigkeiten braucht, oder ob später eine graphische Oberfläche benutzt werden kann. Der Katalog aller Dienste soll schnell und einfach für den Benutzer abrufbar sein. Entweder auf dem netz (z.B. www.dienste.polyphone.ethz.ch) oder direkt im PolyPhone-Client eingebaut. Im zweiten Fall muss der Client noch angepasst werden und immer wieder neu beim Benutzer aktualisiert werden. Herr Plattner würde eine weitere Phase bei der Arbeitsaufteilung einfügen, in welcher mögliche Dienstarten und Beispiele aufgelistet sind. Jedoch möchte ich die Phasenaufteilung so belassen, wie sie in der Aufgabenstellung vorliegt. Im Kapitel 2 habe ich ein neues Unterkapitel mit den Dienstarten und Beispielen erzeugt. Herr Plattner möchte bei den Meilenstein-Präsentationen (jeweils nach Beendigung der 2. und 3. Phase) auch dabei sein, wird aber die nächsten sieben Wochen abwesend sein.

Herrn Plattners Ideen für mögliche Dienste:

- Terminfindung zwischen mehreren Beteiligten
- Messages auf Mobiltelefone und umgekehrt schicken

- Dienste auch mittels SMS über das Mobiltelefon zu Ermöglichen

A.3 F. Piovano

Datum: 10. August 2006
Ort: HPI F25

Herr Piovano ist zuständig für Telekommunikation bei PolyPhone. Er kann mir jederzeit einen Server zur Verfügung stellen mit dem gewünschten Betriebsystem. Herr Piovano hatte die Idee von einem Alarmserver, einen so genannter Polling-Server. Dieser fragt bei einer Datenbank in einem vorgegebenen Intervall nach, ob irgendein Ereignis eintrifft (z.B. ein Weckdienst). Falls der Alarmserver solch ein Ereignis erkennt, meldet er dies dem zugehörigen Benutzer via Instant Messaging. Wir haben uns über die Diensttelefonnummern unterhalten und diskutiert, wie wir das lösen wollen. Eine Variante wäre für jeden Dienst eine neue Nummer der Form 044658... anzubieten. Die zweite und bessere Variante ist nur eine Telefonnummer für alle Dienste anbieten und mittels Keywords die einzelnen Dienste abrufen. So haben wir keine Nummernknappheit und die Dienste sind einfacher über logische Keywords abrufbar. Denkbar wäre auch eine Mischform beider Varianten. Nach der Analyse soll ich Francesco mitteilen, für welche Variante ich mich entschieden habe, damit er diese Änderungen auf dem Proxy-Server vornehmen kann. Er wird mir eine oder mehrere SIP-Telefonnummern für meine Dienste zur Verfügung stellen.

Herrn Piovanos Ideen für mögliche Dienste:

- Eine Nachricht mit allen PolyPhone-Anrufen, welche während der eigenen Abwesenheit unbeantwortet eingetroffen sind
- Direkt über PolyPhone-Dienste Zugriff auf Administrations-Tool zulassen, um alle eigenen Einstellungen und Daten direkt ändern zu können. Jedoch stellt sich das Problem, dass SIP-IM unverschlüsselt verschickt werden und somit die Passwörter ungesichert über das Netz gesendet werden.

A.4 A. Wittmann

Datum: 11. August 2006
Ort: CLU

Herr Wittmann ist beim SMS-Projekt [36] tätig und hat den HTTP-Connector implementiert. Die Stärken seiner Gruppe (Betriebsinformatik) sind vor allem in den Bereichen Java, Linux-Server, TomCat etc. Er schlägt mir vor, auch mit diesen Tools zu arbeiten, da ihre Gruppe viel Erfahrung damit hat und sie mir bei Fragen und Problemen gut weiterhelfen können. Der Gateway im SMS-Tool wird nächstens durch eine neue Version ersetzt, der HTTP-Connector jedoch bleibt erhalten. Ein Connector stellt eine Verbindung zwischen zwei unabhängigen Programmen her und fungiert als Interpreter, falls die beiden Software nicht das gleiche Interface definieren. Er hat mir die Idee mit dem Connector näher gebracht und mir gute Tipps und Ideen geliefert. Er erwähnte die Idee mit einem Keywordbaum. Falls man diese Idee realisieren möchte, muss man die Hierarchie innerhalb der ETH Zürich abklären und anerkennen lassen. Jedoch sei das eine sehr komplexe Angelegenheit, da die Hierarchiestufen noch nirgends festgelegt seien. Herr Wittmann erklärte mir die Verwendung des LDAP-Verzeichnisses. Für gewisse Dienste sei ein Zugriff auf die ETH-Datenbank vorgesehen, welches mit dem LDAP auf einfache Art und Weise erledigt werden kann.

Herrn Wittmanns Ideen für mögliche Dienste:

- Mobilenummern von ETH-Personen abfragen
- andere ETH-spezifische Personendetails abfragen

A.5 Student M. Bühler

Datum: 17. August 2006
Ort: IFW

Ich befragte meinen Mitstudenten (auch Informatik an der ETH Zürich) über das PolyPhone-Projekt. Er hat nicht direkt mit Polyphone zu tun, benutzt diese Infrastruktur aber als Kommunikationsmittel. Mit ihm besprach ich die Verwaltung der Dienste, wie sie gespeichert werden. Dadurch kamen fünf wesentliche Gruppen zum Vorschein wie man Daten beschaffen kann: HTTP, XML, LDAP, Polling und Extern. Er riet mir von einem Entscheidungsbaum

ab, da es in diesem Fall unnötig sei und nur viel zu viel Arbeit für den Endbenutzer mache. Nach vielen Pros und Contras zum Keywordbaum habe ich mich definitiv dagegen entschieden.

Herrn Bühlers Ideen für mögliche Dienste:

- Telefonbuch-Dienst um gewisse Angaben über Personen an der ETH Zürich nachzufragen (www.emd.ethz.ch)
- Informationen über ETH-Gebäude, mit Lageplan
- Busfahrplan ETH-Zentrum <–> ETH-Höngerberg

B Aufgabenstellung

PolyPhone, eine auf dem Session Initiation Protocol (SIP) basierende Dienstleistung, soll erweitert werden um später neue Instant-Messaging (IM)-Dienste zur Verfügung zu stellen. Dadurch kann jeder PolyPhone-Benutzer diese zusätzlichen Dienste, welche im Zusammenhang mit der ETH Zürich stehen (namentlich Campus-Leben, Lehre, Forschung, zentrale Dienstleistungen etc.), jederzeit benutzen. Am Ende dieser Semesterarbeit soll eine grundlegende Basis erstellt sein (Server, Datenbanken,. . .) und ein gut dokumentiertes Interface, damit Dritte noch weitere Dienste auf einfache Weise anbieten können. Ich werde zwei bis drei exemplarische Dienste implementieren um das erstellte System zu testen und danach der Öffentlichkeit zur Verfügung zu stellen (OpenSource).

B.1 Funktionalität

- Interface für Dienstanbieter öffentlich freigeben
- Dienste anbieten/verwalten/löschen
- Dienste benutzen
- Autorisierung der Dienste
- zwei bis drei Dienste zur Verfügung stellen

B.2 Technik

- Server
- Datenbanken
- Protokolle

B.3 Vorgehensweise

Diese Semesterarbeit ist in folgende 5 Phasen aufgeteilt. Nach den Phasen 2 und 3 wird je eine Präsentation als Meilenstein durchgeführt.

1. **Phase:** Aufnahme von Anforderungen der einzelnen Projektpartner. Mittels Interviews Ideen und Vorstellungen sammeln.
2. **Phase:** Analyse bestehender Systeme („Market Survey“ bzw. Suche nach „Related Work“).
3. **Phase:** Erstellen eines Konzepts für die Infrastruktur.
4. **Phase:** Implementierung des Konzepts.
5. **Phase:** Erstellen eines Berichts, welcher die Arbeit beschreibt.

C Zeitplan

	Woche		Milestones	Dokumentation
1	31.7 - 5.8	Interviews vereinbaren		
2	7.8 - 12.8	Interviews		A
		Dienste beschreiben		n
3	14.8 - 18.8	Charakter eines		a
		Dienstes definieren		l
		Anforderungen definieren		y
4	21.8 - 25.8	Marktanalyse:		s
		Vorhandene Libraries		e
		Vorhandene Dienste		
5	28.8 - 1.9	Marktanalyse		
6	4.9 - 8.9	Analyse	8.9 Milestone	
7	11.9 - 15.9	Beenden der Analyse		
8	18.9 - 22.9	Referenzarchitektur		K
		und Komponenten		o
		festlegen		n
9	25.9 - 29.9	Komponenten		z
		beschreiben		e
10	2.10 - 6.10	Anhand von Analyse		p
		und Referenzarchitektur		t
		Konzeptvarianten ersellen		
11	9.10 - 13.10			
12	16.10 - 20.10	Beginn der Implementation		Implementation
13	23.10 - 27.10	Aufbau einer Verbindung zum SIP-Server		
14	30.10 - 3.11	Semesterbeginn		
	22.01 - 26.01 2007	Abschluss und Vorbereitung der Präsentation.	Endpräsentation	

Tabelle 8: Zeitplan für Semesterarbeit