Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Andreas Herglotz

Konzeption und Realisierung eines
ubiquitären und kontextabhängigen Messengers

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Andreas Herglotz

Konzeption und Realisierung eines ubiquitären und kontextabhängigen Messengers

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Master Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft
Zweitgutachter: Prof. Dr. Gunter Klemke

Abgegeben am 6. Oktober 2008

Andreas Herglotz

Thema der Masterarbeit

Konzeption und Realisierung eines ubiquitären und kontextabhängigen Messengers

Stichworte

allgegenwärtig, kontextsensitiv, Web 2.0, Messenger, Open Source, Android, Smartphone

Kurzzusammenfassung

Diese Masterarbeit handelt von der Konzeption und prototypischen Realisierung eines ubiquitären und kontextabhängigen Messengers. Wichtige Aspekte dieses Messengers sind Open Source und User Generated Content. Als Grundlage der Realisierung dient die Android Plattform. Integrale Bestandteile des Messengers sind: Das Verschicken von Sofortnachrichten, die Verknüpfung mit sozialen Netzwerken, die Verortung der Freunde sowie die Verortung von Informationen (Annotationen). Abschließend findet eine Evaluierung dieser Arbeit statt.

Title of the Masterthesis

Design and Implementation of an Ubiquitous and Context-Aware Messenger

Keywords

ubiquitous, context-aware, web 2.0, messenger, open source, Android, smartphone, user generated content, community driven development

Abstract

This thesis deals with the design and implementation as proof of concept of an ubiquitous and context-aware messenger. Important aspects of this messenger are open source and user generated content. The Android platform serves as basis for the implementation. The core capabilities of this messenger are: instant messaging, social networking, location awareness of the buddies as well as location awareness of information (annotations). At the end an evaluation will be performed.

Vorwort und Danksagung

An dieser Stelle möchte ich mich kurz bei ein paar Menschen bedanken, die mir während des Studiums und während der Masterarbeit tatkräftig beiseite gestanden haben.

Zuerst möchte ich mich bei meinen Eltern und Großeltern bedanken, die mir immer, während des gesamten Studiums, mit Rat und Tat zur Seite standen.
Ein weiteres Dankeschön geht an die Firma c.a.r.u.s. IT, die mich während der meisten Zeit des Studiums unterstützt hat.
Zusätzlich möchte ich mich bei Jan Peter Tutzschke und Markus Dreyer für die anregenden und hilfreichen Diskussionen im Zusammenhang mit dieser Arbeit bedanken.
Zu guter letzt möchte ich mich bei Prof. Dr. Olaf Zukunft für die hervorragende Betreuung bedanken.

Inhaltsverzeichnis

1. Einleitung

Im November 2007 wurden 16.000 Personen im Alter von 16 bis 60 Jahren in 29 Ländern weltweit zum Thema Smartphone[1] befragt. Die Studie "Global Technology Insights 2007-08" [TNS (2008)] wurde von der TNS Infratest GmbH, die Mitglied der TNS Gruppe ist, durchgeführt. Diese Gruppe gehört zu einem der weltweit führenden Marktforschungs- und Beratungsunternehmen. Die Studie ergab, dass weltweit im Durchschnitt jeder achte Befragte ein Smartphone besitzt. In Deutschland sind es sogar 40% der Befragten, die ein Smartphone ihr Eigen nennen können.[2] Einen weiteren Beweis für die starke Verbreitung von Smartphones liefern die von Gartner [Gartner (2008)] veröffentlichten Zahlen zum Verkauf von Smartphones. Die Abbildung 1.1 zeigt den weltweiten Verkauf von Smartphones im zweiten Quartal 2008 und vergleicht diese Zahlen mit dem gleichen Zeitraum im Jahr 2007. Insgesamt wurden im zweiten Quartal 2008 über 32 Millionen Smartphones weltweit verkauft, was einem Zuwachs von 15,7% entspricht. Die gute Akzeptanz der Smartphones zeigt das enorme Poten-

Anbieter	Verkaufte Geräte			Marktanteil	
	Q2 2008	Q2 2007	Wachstum	Q2 2008	Q2 2007
Nokia	15,297 Mio.	14,151 Mio.	+8,1%	47,5%	50,8%
RIM	5,594 Mio.	2,471 Mio.	+126,4%	17,4%	8,9%
HTC	1,330 Mio.	0,605 Mio.	+119,6%	4,1%	2,2%
Sharp	1,328 Mio.	2,275 Mio.	-41,6%	4,1%	8,2%
Fujitsu	1,071 Mio.	0,877 Mio.	+22%	3,3%	3,2%
Andere	7,598 Mio.	7,472 Mio.	+1,7%	23,6%	26,7%
Summe	32,221 Mio.	27,854 Mio.	15,7%	100%	100%

Abbildung 1.1.: Weltweit verkaufte Smartphones nach Gartner, zweites Quartal 2008

tial, welches sich hinter mobilen Anwendungen für diese Geräte verbirgt. Eine weitere TNS Infrastruktur Studie [TNS2 (2008)] besagt, dass bereits 16% der Deutschen ab 14 Jahren mobiles Internet und seine Applikationen nutzen. Zusätzlich haben weitere 12% der Nicht-Nutzer ein Interesse an mobilem Internet bekundet. Für diese sind die entstehenden Kosten das größte Hemmnis.

[1] Smartphone: Die weit gefasste Definition eines Smartphones für diese Studie befindet sich im Glossar.

[2] Es wurden je Land ca. 500 ausgewählte Personen repräsentativ über das ganze Land verteilt befragt. Allerdings wurden in so genannten Schwellenländern nur Personen in den Metropolregionen befragt.

Ein bekanntes Smartphone-Beispiel, welches auf mobiles Internet und seine Applikationen setzt, ist das iPhone von Apple. Ebenfalls wegweisend ist das Open Source Projekt namens Android von Google. Dabei handelt es sich um ein Linux-Betriebssystem für Smartphones mit einer Java Virtual Machine und einem umfangreichen Framework. Das Framework ermöglicht auf leichte Weise auch komplexe Applikationserstellung. Dadurch werden einer offenen Entwicklergemeinschaft ganz neue Möglichkeiten bei der Erstellung mobiler Anwendungen ermöglicht.

1.1. Motivation

Die Vision dieser Masterarbeit beruht auf den zuvor beschriebenen Möglichkeiten im Umgang mit Smartphones und dem "Always On" Paradigma, immer mit dem Internet verbunden sein zu können. Das Ziel ist es mit diesen Geräten, jedem zu jeder Zeit und an jedem Ort Kommunikation mit seinen Freunden zu ermöglichen. Dabei soll die Art der Kommunikation über verschiedene Wege kein Hindernis darstellen.
Die Verortung spielt ebenfalls eine wichtige Rolle. Diese ermöglicht es prinzipiell zu erfahren, wo sich die eigenen Freunde tatsächlich aufhalten, um sich beispielsweise spontan treffen zu können. Ein weiterer Aspekt der Verortung ist das Annotieren jeder beliebigen Location. Der Open Source Gedanke macht es dabei möglich, die Software jedem bereitzustellen und individuell anzupassen. Das fängt mit dem einfachen Erstellen von Annotationen an und geht bis hin zur komplexen Erweiterung der Software. Eine detaillierte Beschreibung des Szenarios findet im Kapitel 3.1 statt.

Die einzelnen Bereiche der Vision wurden bereits mehrfach im Kontext des Web 2.0 entwickelt. Es gibt eine Vielzahl an proprietären Lösungen. Beispiele dafür sind Socialight [Socialight (2007)], Dodgeball [Dodgeball (2007)] und SLAM [SLAM (2003)]. Die Herausforderung besteht darin ein offenes System zu erstellen, welches das Beste aus bereits existierenden Lösungen vereint und jedem zur freien Verfügung stellt.
Es gibt noch einen weiteren Aspekt, der die Motivation, sich mit diesem Thema auseinanderzusetzen, erhöht. Auch nach der erfolgreichen Einführung des iPhones durch Apple gibt es noch kein Produkt am Markt, das sich durchsetzen konnte. Durch die Erstellung eines eigenen Open Source Projektes, welches selbst bereits existierende Open Source Quellen integriert, soll sich dieses Problems angenommen werden. Damit besteht das Ziel, eben kein weiteres proprietäres Produkt zu entwickeln, sondern ein Produkt für jedermann zur Verfügung zu stellen.

1.2. Ziele der Masterarbeit

Das Ziel der Masterarbeit ist es, einen mobilen ubiquitären kontextabhängigen Messenger zu erstellen. Ein elementarer Aspekt bei der Entwicklung ist die Integration des Open Source Ge-

danken. In der Analyse wird besonderer Wert darauf gelegt, eine gute Grundlage für die Open Source Entwicklung zu erstellen. Dazu sollen bereits existierende Systeme eingebunden und wiederverwendet werden. Ebenso wichtig ist die komponenten- und modulbasierende Architekturerstellung. Neben der Konzeption und Erstellung einer Software, die Open Source ist und daher von der Open Source Community weiterentwickelt werden kann, spielt auch die aktive Integration der Benutzer eine wichtige Rolle (User Generated Content). Die Entwicklung dieser Arbeit orientiert sich am Unified Process [Arlow und Neustadt (2005)].

1.3. Überblick über die Kapitel

Im ersten Kapitel, der Einleitung, wurde zum Thema hingeführt, die Motivation geschildert und Ziele definiert. Dieser Abschnitt gibt einen kurzen Überblick über die weiteren Kapitel dieser Arbeit. In Kapitel 2, den Grundlagen, werden die fundamentalen Themen, die für diese Arbeit wichtig sind, beschrieben und definiert. Zu diesen Themen gehört das Ubiquitous Computing, die Context Awareness, Social Software, Open Source, Benutzerintegration und die Android Plattform. Da sich für die Softwareentwicklung am Unified Process orientiert wird, enthält das Kapitel 3 die Analyse. Für die Analyse wird das Szenario vorgestellt sowie funktionale und nicht funktionale Anforderungen erfasst. Als letzten Teil der Analyse werden ähnliche Projekte kurz beschrieben und mit diesem verglichen. Kapitel 4 stellt aufbauend, auf der eben genannten Analyse, das Konzept vor. Das Konzept besteht aus einer fachlichen und einer technischen Architektur, die jeweils vom Abstrakten zum Konkreten beschrieben wird. Die Realisierung wird in Kapitel 5 beschrieben. Dabei werden neben den als „Proof of Concept“ realisierten Komponenten die Unterschiede zum Konzept dargestellt und nachfolgend das Testkonzept vorgestellt. Kapitel 6, die Evaluierung, bewertet die Ziele und Ergebnisse dieser Arbeit aus einer kritischen Distanz sowohl aus fachlicher als auch aus technischer Sicht. In der Zusammenfassung, Kapitel 7, wird das Wesentliche der Masterarbeit zusammengefasst. Dies beinhaltet den Stand der Entwicklung, aufgetretene und gelöste Probleme und gesammelte Erfahrungen. Im Ausblick werden noch Weiterentwicklungs- und Verbesserungsmöglichkeiten aufgezeigt.

2. Grundlagen

Dieses Kapitel beinhaltet die für diese Arbeit notwendigen Grundlagen. Dabei findet durch die einzelnen Unterkapitel eine Einordnung der in diesem Zusammenhang wichtigen Komponenten in den Gesamtkontext statt. Es wird Ubiquitous Computing als zu Grunde liegende Vision erklärt. Weiterhin werden die wichtigen Komponenten Context Awareness und Social Software der Vision behandelt. Anschließend wird die Vision mit dem Open Source Gedanken und der aktiven Benutzerintegration in Verbindung gebracht. Umgesetzt werden soll diese auf der Android Plattform. Abschließend wird das Kapitel zusammengefasst.

2.1. Ubiquitous Computing

Bereits im Jahre 1991 prägte Weiser den Begriff des Ubiquitous Computing. Sein visionärer Artikel “The Computer for the 21st Century” [Weiser (1991)] propagierte den allgegenwärtigen Computer. Dabei handelt es sich um die Entwicklung vom Mainframe Computer über den Personal Computer hin zu mobilen kleinen Computern und Mikroprozessoren. Zu Zeiten des Mainframe Computers arbeiteten viele Menschen gleichzeitig an einem Großrechner. In der Ära des PCs hatte jeder Nutzer üblicherweise ein Gerät. Mittlerweile ist es so, dass jeder Nutzer mit mehreren elektronischen Geräten ausgestattet oder von ihnen umgeben ist. Charakteristisch dafür ist die Integration dieser Geräte in das tägliche Leben. Dabei verschwindet die Technik quasi im Hintergrund und erleichtert den Alltag. Dieser Prozess findet immer noch auf unabsehbare Zeit statt.

Moore’s Law: Ein rasanter technischer Fortschritt wurde bereits von Moore 1965 prognostiziert. In dem von ihm entwickelten „Gesetz"[Moore (1965)] stellte er die These auf, dass sich alle 18 Monate die Leistungsfähigkeit von Prozessoren (bei abnehmendem Preis und sich reduzierender Größe) verdoppelt. Dieses traf bisher ziemlich genau zu und scheint trotz Kritikern wie beispielsweise Tuomi [Tuomi (2002)] noch einige Jahre zu gelten.

2.1.1. Definition

Mattern [Mattern (2005b)] setzt auf der Grundlage von Mark Weiser auf. Dabei identifiziert er wesentliche Technologien und elementare Voraussetzungen, die für das Ubiquitous Computing wichtig sind. Zum einen ist dieses die Kommunikation. Drahtlose Kommunikation als ein sehr wichtiger Faktor ist mittlerweile zum Beispiel in Form von WLAN, UMTS oder auch

Bluetooth weit verbreitet. Eine weitere Technologie, die in diesem Kontext eine immer wichtiger werdende Rolle spielen kann, ist RFID. Nahezu beliebige Gegenstände lassen sich mit sogenannten RFID-Tags versehen. Dabei lassen sich diese mit Informationen versehenen Tags in einer Entfernung von wenigen Metern über entsprechende Lesegeräte auslesen. Diese Technologie, WLAN, GPS oder die Zellen der Handynetze lassen sich zur Lokalisierung nutzen und stellen damit einen weiteren Schwerpunkt für Ubiquitous Computing dar.
Zum anderen unterscheidet Mattern im Ubiquitous Computing drei Kategorien bei der Nutzung dieser technischen Möglichkeiten:

Embedded Computing: Damit werden Alltagsgegenstände bezeichnet, die durch technische Geräte erweitert werden und Informationen verarbeiten können. Dies kann beispielsweise die Kaffeetasse sein, die dem Trinkenden mitteilt, dass der Kaffee bereits kalt ist und er sich besser neuen Kaffee holen sollte.

Wearable Computing: In diesem Fall ist von technischen Geräten die Rede, die direkt am Körper oder in der Kleidung getragen werden. Ein Beispiel ist der RFID-Chip beim Skifahren, durch den man automatisch, wenn man in die Nähe des Lifts kommt, seine Zugangsberechtigung mitteilt und den Lift passieren kann. Eine weitere Ausprägung des Wearable Computing ist es, den Gesundheitszustand mittels solcher Geräte zu überwachen.

Sensornetze: Die entsprechenden technischen Geräte reichern die Umwelt an, so dass man Informationen über die Umwelt im konkreten Fall erhält. Ein Beispiel für den Einsatz eines Sensornetzwerkes ist beim Feuerwehrmann-Szenario zu finden. Dieser legt die Sensoren aus, während er das Gebäude betritt. Will der Feuerwehrmann das Gebäude wieder verlassen, können die Sensoren ihm helfen den Weg zu finden, indem sie ihm beispielsweise Informationen über deren unmittelbare Temperatur mitteilen. Damit weiß er, ob es an einer bestimmten Stelle mittlerweile zu heiß geworden ist oder der Weg noch passierbar ist.

Zusammenfassend kann man sagen, dass beim Ubiquitous Computing kleine, leichte, energiesparsame Mikroprozessoren zum Einsatz kommen. Über drahtlose Kommunikation werden Informationen von der Umwelt aufgenommen und weitergegeben. Dieses geschieht häufig über Sensoren. Gemäß Neil Gershenfeld vom MIT [Gershenfeld (1999)] zitiert Mattern, dass ein „Internet der Dinge“ erschaffen wird. Die Realität wird durch diese smarten Gegenstände erweitert. Damit schneidet er ein weiteres Gebiet der Informatik an, die „Augmented Reality“. Dieses Thema wird im Folgenden nicht weiter vertieft. Zusätzlich spricht Mattern das Problem an, dass sich nicht jeder Fortschritt durch technologische Verbesserungen erzielen lässt, sondern beispielsweise auch neue und bessere Konzepte für die Mensch-Maschine-Interaktion wichtig sind. In diesem Bereich entspricht die Geschwindigkeit der Fortschritte laut Mattern leider nicht der des Moorschen Gesetzes.

Risiken: Mattern weist auch auf potentielle Risiken und Nebenwirkungen durch die Alltagsinformatisierung hin [Mattern (2005a)]. Jede Information für sich stellt kaum eine Bedrohung dar. Verknüpft man jedoch die Vielzahl an Informationen, die durch smarte Geräte entstehen, bedarf es nicht viel Phantasie, um sich vorstellen zu können, was mit diesen Informationen geschehen könnte. Dieses kann mit dem Ausspionieren des Lebenspartners beginnen, führt weiter über die Profilerstellung durch Firmen, um noch stärker personalisierte Werbung zu erstellen und kann bis hin zum Identitätsdiebstahl führen. Die möglichen Folgen des Ubiquitous Computing sind kaum absehbar. Andere offene Fragestellungen in dem Zusammenhang sind: Funktioniert der Alltag bald nur noch, wenn alle Geräte funktionieren und was passiert, wenn diese ausfallen? Entwickeln die smarten Gegenstände ein ungewolltes Eigenleben, welches eventuell zu Kontrollverlust führt? Wer ist für die Handlung smarter Gegenstände verantwortlich - der Besitzer oder der Hersteller? Anhand dieser Fragen wird deutlich, dass es eine Menge von wichtigen Fragen gibt, die beim Fortschreiten dieses Prozesses betrachtet und behandelt werden müssen.

2.1.2. Pervasive Computing

Neben dem Begriff des Ubiquitous Computing wurde von der Industrie, respektive von IBM, im Jahr 1998 ein weiterer Begriff eingeführt, das Pervasive Computing [Nieuwdorp (2007)]. Mattern beschreibt nach J. Burkhardt [J. Burkhardt (2001)] und U. Hansmann [Hansmann u. a. (2001)], dass dieser Begriff auf den gleichen Grundlagen aufbaut, aber ein etwas anderes Ziel verfolgt.

> "Während Weiser den Begriff „Ubiquitous Computing" eher in akademisch-idealistischer Weise als eine unaufdringliche, humanzentrierte Technikvision versteht, die sich erst in der weiteren Zukunft realisieren lässt, hat die Industrie dafür den Begriff „Pervasive Computing" mit einer leicht unterschiedlichen Akzentuierung geprägt. [...] Mit dem primären Ziel, diese eher kurzfristig im Rahmen von Electronic-commerce-Szenarien und web-basierten Geschäftsprozessen nutzbar zu machen." [Mattern (2001)]

Dadurch wird aus der reinen Vision des Ubiquitous Computing ein Praxisbezug geschaffen. Aufgrund dieses für diese Arbeit nicht relevanten Unterschiedes wird im Weiteren nur der Begriff des Ubiquitous Computing verwendet und als Synonym gebraucht.

2.1.3. Beispiel

Ein praktisches Beispiel für Ubiquitous Computing ist der weit verbreitete Einsatz von Handys, Smartphones und PDAs. Aktuell am Markt befindliche Geräte vereinen die Kommunikationsmöglichkeit per Sprache, die Datenübertragung per UMTS, die Nutzung von Ortsinformationen über GPS und mehr in sich. Dass die Vision von Weiser Realität annimmt, bildet die

Grundlage für diese Arbeit. Smartphones als leistungsfähige, mobile Computer sollen in diesem Zusammenhang nicht nur den Alltag erleichtern, sondern auch durch neue Möglichkeiten wie zum Beispiel dem Annotieren der Realität oder dem Nutzen von Kontextinformationen das tägliche Leben erweitern.

2.2. Context Awareness

Die Geschichte der Context Awareness startete laut Baldauf [Baldauf u. a. (2007)] im Jahr 1992 mit dem „Active Badge Location System". Dieses System konnte mithilfe der Infrarottechnologie Personen orten und Telefonate an diese weiterleiten [Want u. a. (1992)]. In der Literatur erschien dieser Begriff allerdings das erste Mal 1994 bei Schilit und Theimer [Schilit und Theimer (1994)]. Sie beschrieben Kontext als Position, Identitäten naher Personen, Objekte und Veränderungen zu diesen Objekten. 1997 bezeichneten Ryan u.a. [Ryan u. a. (1998)] Kontext als die Position, Umgebung, Identität und Zeit des Benutzers. Ein Jahr später definierte Dey [Dey (1998)] Kontext als den emotionalen Zustand des Benutzers, Mittelpunkt der Aufmerksamkeit, Position und Orientierung, Datum und Zeit, sowie Objekte und Menschen in der Umgebung des Benutzers. Eine weitere gute Definition gibt Brown im Jahr 1996 [Brown (1996)]: Kontext besteht aus den Elementen der Benutzerumgebung, von denen der Computer weiß.

Nachdem durch verschiedene Definitionen von Context Awareness ein erster Überblick zu diesem Thema gegeben wurde, soll in diesem Zusammenhang darauf hingewiesen werden, dass es sich hierbei wie beispielsweise bei dem Active Badge System um größtenteils stationäre Systeme handelt. Praktische Relevanz bekommt dieses Thema erst seit den letzten Jahren, als sich portable Computer und vor allem kabellose Kommunikation immer stärker etablierten. So wird auch die enge Verbindung zu Ubiquitous Computing deutlich. Aus diesem Grund sind die beiden Themen auch in dieser Arbeit eng miteinander verknüpft.

2.2.1. Definition

Ausgehend von den verschiedenen Definitionen zuvor wird an dieser Stelle eine Definition von Kontext und eine Definiton von Context Awareness gegeben, die die Grundlage für die weitere Arbeit bilden.

Kontext: Eine umfassende und trotzdem präzise Definition von Kontext liefern Dey und Abowd [Abowd u. a. (1999)]. Diese Definition lautet wie folgt:

> „Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.“

Dabei kann eine „Person“ ein einzelnes Individuum oder auch eine Gruppe sein. Unter „Place“ werden Räume, Gebäude, etc. verstanden. „Object“ ist die Bezeichnung für physikalische Objekte, Computerkomponenten, etc. Der Kontext wird in verschiedene Typen unterteilt: Activity, Identity, Location und Time. „Activity“ bedeutet soviel wie Status oder die spezifischen Eigenschaften einer Entity. Ein Beispiel ist die Temperatur und Helligkeit in einem Raum oder verschiedene Prozesse, die auf einem Gerät ablaufen. Die „Identity“ entspricht einem eindeutigen Schlüssel für eine Entity. Die Position oder nähere Umgebung einer Entity wird durch die „Location“ beschrieben. Der Typ „Time“ wird benutzt, um Situationen mithilfe eines Zeitstempels genau zu definieren oder eine Ordnung über Events erstellen zu können, etc.

Context Awareness: Ebenfalls liefern Dey und Abowd eine Definition Context Awareness. Diese lautet wie folgt:

> „A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.“

Diese Definition wird durch eine Taxonomie, die drei Hauptkriterien unterscheidet, konkretisiert. Dabei werden context-aware Applikationen unterschieden nach:

- „presentation of information and services to a user;
- automatic execution of a service;
- tagging of context to information for later retrieval.“

Anmerkung zur vorherigen Definition: Die folgende Definition von Chen und Kotz in [Chen und Kotz (2000)] steht nicht im Widerspruch zu der vorherigen Definition, sondern erläutert Kontext von einer weiteren Seite. Sie unterteilen Kontext nur in zwei Bereiche. Dadurch wird die vorherige Definition etwas deutlichder. Zum einen ist Kontext eine Menge von Umgebungszuständen und Einstellungen, die das Verhalten der Anwendung bestimmt. Zum anderen treten durch den Kontext Ereignisse auf, die den Benutzer interessieren. Das bedeutet eine Unterscheidung zwischen aktivem Kontext, der das Verhalten der Anwendung beeinflusst und damit kritisch für die Anwendung ist, und passivem Kontext, der zwar relevant ist, aber nicht kritisch. So ist die Position bei einer Anwendung, die den Benutzer navigiert, kritisch für die Funktion der Anwendung. Im Vergleich dazu sind Informationen zu besonderen Sehenswürdigkeiten zwar relevant, aber nicht kritisch. Dabei soll sich die Anwendung bei der aktiven Context Awareness automatisch an den erkannten Kontext durch Veränderung des Verhaltens anpassen. Das Ziel ist es, unnötige Benutzerinteraktionen zu vermeiden und die Technologie möglichst zu verstecken.
Diese Begriffsklärung soll später im Design bei der Erstellung der Architektur helfen, den Kontext beziehungsweise die Context Awareness besser in die Applikation integrieren zu können. Aufgrund des zeitlichen Rahmens der Masterarbeit wird für den Prototypen der Kontext

auf Ortsinformationen und Objekte in der Nähe beschränkt. „Objekte in der Nähe“ bezieht sich darauf, dass das System weiß, welche Personen sich in der Nähe befinden oder auch darauf, welche Objekte von Interesse in der Nähe sind. Zwei Beispiele dafür sind das Teleporting System [Bennett u. a. (1994)] und der Context-Aware Pager [Yan und Selker (2000)]. Weitere Informationen zu diesen Systemen sind in den Quellen nachlesbar.

2.2.2. Beispiele

Kurze Beispiele sollen zeigen inwiefern Kontext und Context Awareness in ubiquitären Umgebungen sinnvoll eingesetzt werden kann.

GPS Navigation: Die GPS Navigation ist mittlerweile sehr weit verbreitet und damit ein sehr bekanntes Beispiel für Context Awareness. Es geht darum, dass man sich irgendwo im Freien befindet und nicht genau weiß wo man ist, bzw. wie man zu einem bestimmten Ziel gelangt. Über Koordinaten, die vom Satelliten empfangen werden, kann die eigene Position bestimmt werden. Mit Hilfe einer Karte kann das Ziel ausgewählt und eine entsprechende Route berechnet werden. Dieses ist ein gutes Beispiel zur Demonstration wie Context Awareness den Alltag erleichtern kann.

Benutzerschnittstelle: Die Context Awareness kann auch herangezogen werden, um abhängig von äußeren Einflüssen die Kommunikation vom Gerät zum Benutzer zu variieren. Ist die Umgebung beispielsweise sehr laut, kann das Gerät die Lautstärke der Audioausgabe daran anpassen oder eventuell auf eine andere Art der Kommunikation umschwenken. Wird das Gerät beispielsweise stark bewegt, ist es unwahrscheinlich, dass der Benutzer kleine Schrift auf dem Display lesen kann, also könnte die Schrift vergrößert werden oder eine Audioausgabe des Textes erfolgen.

Gerätestatus: Der Benutzer möchte sich einen POI (Point of Interest) anzeigen lassen. Zu diesem POI sind beispielsweise ein Text, Sprachinformationen und ein Video hinterlegt. Der Benutzer hat das Gerät so konfigutiert, dass es weiß, dass keine UMTS-Flatrate vorhanden ist. Also wählt das Gerät, wenn nur eine UMTS-Verbindung möglich ist den Text aus, um die Kosten möglichst gering zu halten. Befindet sich der Benutzer in einem WLAN, kann die Sprachinformation und das Video heruntergeladen werden. Ist jetzt aber der Akku beinahe alle, wird nicht das Video angezeigt, sondern das Display verdunkelt und die Sprachinformation abgespielt.

2.3. Social Software

Von Beginn an war das Internet ein Medium, das nicht nur Maschinen, sondern auch Menschen miteinander verbunden hat [Breslin und Decker (2007a)]. Zu nennen sind beispielswei-

se E-Mail, Mailing Listen, Usenet und Bulletin Boards, die es Menschen ermöglichten, sich zu sozialen Online-Netzwerken zu verbinden. Dieses geschah typischerweise zu bestimmten Themen. Den Unterschied zu heute bildet die Tatsache, dass die Menschen damals nicht explizit soziale Netzwerke definierten. Dieses fand durch das Verhalten der Menschen implizit statt. Im Zuge des Web 2.0 entstand eine andere Auffassung von Online Communities. Sie erstellten explizit Verknüpfungen zwischen Menschen. Häufig geschah dieses auf Basis von Informationen, die aus Benutzerprofilen gewonnen werden konnten.

Relevanz: Heutzutage hat die Social Software einen hohen Stellenwert, in der von Computern geprägten Gesellschaft, erreicht. Die folgenden Zahlen von comScore [comScore (2007)] verdeutlichen das: MySpace.com zog im Juni 2007 114 Millionen Besucher an. Damit hatte MySpace einen Zuwachs von 72% in einem Jahr. Facebook wuchs sogar um 270% auf 52.2 Millionen Besucher. Das Pew Internet & American Life Project berichtete, dass 48% der Erwachsenen Webseiten wie YouTube aufgerufen haben.

2.3.1. Definition

Die Studie zeigt deutlich, wie wichtig Social Software in den letzten Jahren geworden ist. Daher wurde versucht, diesen Begriff zu definieren.

Übersicht zu Social Software: In einem ersten Schritt wird aufgeführt, welche Anwendungen zu Social Software gehören. Dazu ist zu sagen, dass laut Green und Pearson [Green und Pearson (2005)] jede Software, die Gruppenkommunikation ermöglicht, Social Software ist. Es wird allerdings eine deutliche Abgrenzung zu Groupware gezogen. Während bei Groupware die Menschen aus organisatorischen oder funktionellen Gründen in Gruppen unterteilt wurden, steht bei Social Software das Individuum und sein Verlangen deutlich stärker im Vordergrund. Verschiedene Typen von Social Software laut Green und Pearson sind:

- E-Mail
- Statische und datenbankbasierte Webseiten
- Diskussionsforen
- Internet Chat / Instant Messaging
- Video und Audio Streaming
- Video und Audio Konferenzen
- Weblog
- Wiki

- RSS
- Tagging Systeme
- Social Networking
- Virtuelle Welten
- Gadgets / Widgets

Eigenschaften von Social Software: Im Zusammenhang mit dieser Arbeit ist es sinnvoll Social Software detaillierter zu betrachten. Die zuvor aufgeführte Liste von Social Software Typen zeichnet sich durch verschiedene Eigenschaften aus. Diese Eigenschaften sind zum Beispiel:

Kommunikationsarten: Die Kommunikation kann synchron oder asynchron stattfinden. In einigen Fällen ist sogar beides möglich. E-Mail ist ein Beispiel für asynchrone Kommunikation, während Video und Audio Streaming Beispiele für synchrone Kommunikation sind.

Sichtbarkeit: Die Sichtbarkeit der Kommunikation kann entweder auf die Beteiligten begrenzt, auf definierte Freunde begrenzt oder für jedermann offen sein. E-Mails sind nur für die Beteiligten sichtbar. Bei einem Weblog hingegen kann jeder das Geschriebene lesen.

Kommunikationsformen: Es gibt verschiedene Formen der Kommunikation. Diese kann durch die Übermittlung von Text-, Audio- oder Videonachrichten geschehen. E-Mails beispielsweise bestehen heute in der Regel aus Textnachrichten, wobei sich die Instant Messenger weiterentwickelt haben und Audio- oder sogar Videokommunikation häufig integriert sind.

Erweiterung des Social Software Begriffs: Im Vergleich zu den Ursprüngen von Social Software ist eine Weiterentwicklung zu erkennen. Diese ist geprägt durch leistungsfähigere PCs und eine leistungsfähigere Infrastruktur. Aufgrund der heutigen Möglichkeiten, miteinander in Kontakt treten zu können, soll der Begriff Social Software etwas erweitert werden. Zum einen soll eine Kombination aus den verschiedenen Kommunikationsarten und -formen geschaffen werden. Zum anderen soll Social Software in Verbindung mit Context Awareness nicht mehr auf das Virtuelle begrenzt sein, sondern auch Menschen im realen Leben zueinander führen.

Social Software aus soziologischer Sicht: Im Folgenden wird eine Definition gegeben, die auf der Arbeit des Gesellschaftswissenschaftlers Schmidt [Schmidt (2006)] beruht: Demnach gibt es drei Einsatzbereiche, die bei Social Software unterschieden werden.

- „Informationsmanagement: Ermöglichung des Findens, Bewertens und Verwaltens von (online verfügbarer) Information.
- Identitätsmanagement: Ermöglichung der Darstellung von Aspekten seiner selbst im Internet.
- Beziehungsmanagement: Ermöglichung Kontakte abzubilden, zu pflegen und neu zu knüpfen."

Aus diesen drei Basisfunktionen entwickelte Schmidt [Schmidt (2006)] folgende Definition:

> „Social Software sind solche internetbasierten Anwendungen, die Informations-, Identitäts- und Beziehungsmanagement in den (Teil-) Öffentlichkeiten hypertextueller und sozialer Netzwerke unterstützen."

Die verschiedenen Bereiche und Sichtweisen der Social Software zeigen, dass es keine strikte Definition gibt. Vielmehr wird deutlich wie vielschichtig Social Software ist. Dennoch wurde der Begriff Social Software detailliert genug beschrieben, um nachfolgend die Ziele, die erreicht werden sollen, formulieren zu können.

2.3.2. Ziele der Social Software

Das Ziel der Social Software in diesem Zusammenhang ist es, die Möglichkeit zu schaffen soziale Kontakte nicht nur virtuell sondern auch real treffen zu können. Dadurch soll eine stärkere Bindung zwischen den Kontakten entstehen. Eines der beiden Probleme, die soziale Netzwerke mit sich bringen [Breslin und Decker (2007b)] ist, dass diese langweilig geworden sind, da viele Benutzer versuchen möglichst große Kontaktlisten zu haben oder einfach nur die Kontakte nach kuriosen Dingen durchsuchen. Das bedeutet, dass viele Kontakte keine wirkliche Beziehung zueinander besitzen. Mithilfe der Context Awareness soll diesem Problem entgegnet werden. Den losen Kontakten sollen durch die Unterstützung der Context Awareness die Möglichkeit gegeben werden, diese im realen Leben zu treffen und somit die Beziehung zu intensivieren. Auf diese Weise kann den Kontakten wieder mehr Wert beigemessen werden.
Es wird noch ein zweites Problem angesprochen. Bisher war es so, dass im Grunde jedes soziale Netzwerk separat existierte. Trat man also einem neuen Netzwerk bei, musste man sämtliche Profildaten neu eingeben und die Kontakte neu hinzufügen. Des Weiteren musste man in den verschiedenen Netzwerken jeweils die Profildaten und Kontakte pflegen, wodurch ein enormer Aufwand entstand. Aufgrund dieses Problems entstand die Idee eines Social Graph [Fitzpatrick (2007)] oder die Umsetzung von Google mit dem Projekt Open Social [OpenSocial (2008)]. Das Ziel beider Ansätze ist es, bereits existierende soziale Netzwerke miteinander zu verknüpfen. Dadurch müssen sowohl die Kontakte als auch das eigene Profil nur einmal gepflegt werden. Ebenso muss man sich auch nur einmal anmelden. Open Social stellt zusätzlich

für die Benutzer eine Application Programming Interface (API) zur Verfügung, um die sozialen Netzwerke erweitern zu können. Auf diese Weise kann ein Gadget einmal geschrieben werden und bei allen kooperierenden Netzwerken universal benutzt werden.

Instant Messenger: Vergleichbares gibt es schon beim Instant Messaging. Es gibt sogenannte Multi-Protokoll-Client Programme, die die Protokolle proprietärer Messenger wie ICQ, MSN, AIM, Yahoo, etc. miteinander verknüpfen. Dadurch kann man in einem Programm alle Freunde aus verschiedenen Messengern kontaktieren. Ein Problem dabei ist, dass sich die Anbieter nicht auf einen Standard geeinigt haben. Wird also ein Update eines Anbieters herausgegeben, kann es sein, dass keine Verbindung mehr mit dem Multi-Protokoll-Client aufgebaut werden kann, bis entsprechende Anpassungen an der Client-Software vorgenommen wurden. Gleiches gilt für die Unterstützung der Features. Ein Beispiel für einen solchen Multi-Protokoll-Client ist Miranda IM. Eine andere Lösung bietet Jabber. Dieses ist ein Protokoll, welches den XMPP-Standard umsetzt. Jabber Clients bieten keine direkte Unterstützung anderer Protokolle, sondern nutzen dabei sogenannte Transports um die Verbindung in andere Netzwerke zu erstellen. Bei Jabber ist dieses die Aufgabe des Jabber-Servers. Das bedeutet, dass die Änderungen nur am Server und nicht an jedem Client durchgeführt werden müssen.

2.4. Open Source

Die Entstehung von Organisationen, die die Rechte und Pflichten im Umgang mit Software definieren, beruht auf einem langen Entwicklungsprozess in der Welt der Computer. Bereits in den 1930er Jahren wurden bei IBM Projekte geschaffen, um Wissen in Bezug auf Software zu sammeln und mit Kunden zu teilen. Bis ins Jahr 1970 stellte IBM seine Software und den entsprechenden Quellcode zur Verfügung [Johnson (1998)]. Parallel dazu entwickelte sich zwischen 1960 und 1970 in akademischen US-Einrichtungen eine Form des Miteinanders (Hacker Culture). In dieser Kultur war es selbstverständlich, Software und den entsprechenden Code auszutauschen und dadurch zu verbessern. Damals war es üblich, den Quelltext mit den Systemen auszuliefern. Dadurch kamen Rückmeldungen und Verbesserungsvorschläge an die Hersteller zurück. Zu der Zeit wurden Software und Computer noch als eine Einheit betrachtet.

2.4.1. Historie

Am 23. Juni 1969 [Grad (2002)] führte IBM die Trennung von Software und Hardware ein und schuf somit einen neuen Wirtschaftsmarkt. Die Software wurde urheberrechtlich geschützt. Es wurden zugehörige Lizenzverträge eingeführt, der Code wurde nicht mehr offen gelegt und entgeltliche Leistungen wurden für die Software angeboten. Auf diese Weise entstanden Wartungsverträge. Ebenso entstanden neue Kosten für Weiterentwicklungen. In den folgenden Jahren entstanden so neue Firmen, die neue Lizenzen für ihre Software entwickelten. Es wurden Nutzen, Weitergabe und die Möglichkeit, die Software selbstständig zu verändern, stark

eingeschränkt. Um die Software vor dem unerlaubten Verändern zu schützen, wurde nur noch der maschinenlesbare Code ausgeliefert. Auf diese Weise entstand proprietäre Software, durch die die ursprünglich frei verfügbare Ressource Software künstlich eingeschränkt wurde.

Neuorientierung: Zusammengefasst gab es drei Möglichkeiten, sich mit der neuen Situation auseinanderzusetzen. Die eine ist von der Branche Abstand zu nehmen, die zweite sich mit den neuen Umständen abzufinden und die dritte dem entgegenzutreten und zu versuchen, eine entsprechende Community zu gründen, die die alten Prinzipien weiterhin vertritt.

2.4.2. Free Software Foundation

Am 27.09.1983 [FSF (1999)] verfasste Richard Stallman eine E-Mail mit der Überschrift „Free Unix!“. In dieser tat er seine Idee kund, eine Unix-Version namens GNU zu implementieren. Das Besondere dieser Unix-Version ist, dass er sie für jedermann frei zur Verfügung stellen möchte. Dabei fragt er jedermann um Mithilfe. Diese kann in verschiedenen Formen stattfinden, zum Beispiel durch Programme, Equipment oder auch Geld von privaten Personen oder Firmen. In dieser E-Mail wird seine Haltung gegenüber den proprietären Software-Lösungen deutlich. Nachdem diese Idee auf Zuspruch stoß, gründete Stallman eine gemeinnützige Stiftung, die Free Software Foundation (FSF) [Stallman (1998)] zum Zweck, bei der Förderung und Entwicklung freier Software zu helfen. In der GNU General Public License (GPL) werden die Rechte für „freie Software“ festgehalten. Diese Lizenz steht in Verbindung mit dem Copyleft-Prinzip. Dieses verhindert, dass aus GNU Software proprietäre Software gemacht wird. Es dreht das Copyright-Gesetz quasi um und ist damit ein Mittel um Software frei zu halten.

2.4.3. Open Source Initiative

Im Jahre 1998 veranlasste diese Ausarbeitung [Raymond (1997)] von Raymond, dass Netscape den Quellcode für ihren Browser offenlegten. Daraus entstand später das Mozilla-Projekt. In diesem Jahr gründeten Raymond, Perens und O’Reilly die Open Source Initiative (OSI) [OSI (2006)]. Ziel dieser Gründung war es, den Begriff „freie Software“ durch den Begriff „Open Source“ zu ersetzen. Der Grund der Begriffsersetzung ist, dass dieser ideologisch unbelasteter ist und sich somit besser vermarkten lässt. Dieser Begriff erscheint geschäftsfreundlicher. Entsprechend wurden Open Source Lizenzen geschaffen, die für die Wirtschaft interessanter sein sollten. Ein Beispiel dieser Lizenzen ist die Mozilla Public License.

2.4.4. Ergebnis

Es hat eine Spaltung in zwei Lager stattgefunden. Zum einen gibt es die Free Software Foundation (FSF), für die aus ideologischen Gründen die Freiheit bei freier Software stark im Vordergrund steht. Diesbezüglich wird von dieser oft zitiert:

„Denk an „freie Rede“, nicht an „Freibier“.“

Es geht bei ihnen nicht um den Gedanken, dass etwas frei im Sinne von gratis ist, sondern dass es frei im Sinne von Freiheit ist.
Zum anderen gibt die Open Source Initiative (OSI), die ihren Schwerpunkt stärker an der Wirtschaft orientiert. Trotz dieser Unterschiede und der berechtigten Frage, unter welcher Lizenz die Software veröffentlicht werden soll ist festzustellen, dass die zuvor erwähnten Gruppierungen inhaltlich Ähnliches aussagen. Diese Rechte werden im Folgenden einmal aufgeführt:

- das Recht, die Software nach Belieben weiterzugeben;
- das Recht, die Quelltexte zu erhalten;
- das Recht, die Software zu verändern und in veränderter Form weiterzugeben;
- das Recht, die Software für jeden Zweck einzusetzen.

Für diese Arbeit ist der sich größtenteils in der Ideologie befindliche Unterschied nicht von Bedeutung. Daher wird im Folgenden nur der Begriff Open Source benutzt, welcher nicht weiter von „freier Software“ unterschieden wird. Es gibt eine Vielzahl von Open Source Projekten. Dabei haben große Projekte zumeist eine eigene Webseite und kleinere werden durch Software-Entwicklungsmanagementsysteme wie zum Beispiel Sourceforge präsentiert. Ein Eingehen auf die Details der einzelnen Lizenzen würde hier leider den Rahmen der Arbeit sprengen. Wichtig in diesem Zusammenhang ist jedoch zu erwähnen, dass die Qualität von Open Source im Allgemeinen einen hohen Standard hat und starke Akzeptanz genießt, da sich prinzipiell jeder an der Entwicklung beteiligen kann und der offen gelegte Code ein stärkeres Vertrauen mit sich bringt [Scacchi (2007)]. Ein weiterer Vorteil von Open Source Software sind die schnelleren Reaktionszeiten bei Fehlern oder neuen Features [Mockus (2002)]. Ein Nachteil allerdings ist, dass es bisher kaum Wartungsverträge für Open Source Software gab. Dieses hatte zur Folge, dass Firmen die Produkte teilweise nicht einsetzten, weil keiner für Fehler haftbar gemacht werden konnte und Fehler somit nicht behoben werden mussten. Zurzeit wird das Angebot von Wartungsverträgen für Open Source Software jedoch größer.

2.5. Aktive Benutzerintegration

Bei der aktiven Benutzerintegration handelt es sich um eine Entwicklung, die besonders im Zusammenhang mit Web 2.0 in den Vordergrund getreten ist. Dabei geht es darum, den Benutzer von der reinen Benutzung des Programmes zur aktiven Teilnahme am Programm zu bewegen. Dazu wird im Folgenden eine Gruppierung der möglichen Benutzer vorgenommen.

2.5.1. Gruppierung

In diesem Abschnitt werden die potentiellen Nutzer in vier verschiedene Gruppen eingeteilt. Diese Unterteilung wird vorgenommen, um unterschiedliche Ambitionen der Benutzer festzustellen und damit im Design und der Entwicklung des Messengers speziell auf die einzelnen Gruppen eingehen zu können.

Consumer: Die Gruppe der Consumer ist die wahrscheinlich größte der vier Gruppen. Consumer nutzen das Programm, ohne eine aktive Teilnahme. Beispielsweise schauen sie sich den Inhalt, den andere erstellt haben, nur an.

Author: Diese Gruppe ist für die Erstellung des Inhaltes zuständig. Sie besitzt die Fähigkeit Einträge in Wikis oder Blogs vorzunehmen. Im Kontext des Web 2.0 wurde der Begriff User Generated Content (UGC) geprägt.

Programmer: Die aktive Beteiligung steigt in dieser Gruppe weiter an. Die Programmer erweitern bereits bestehende Anwendungen. Sie erstellen Plugins, Gadgets, Javascript-Anwendungen oder nutzen proprietäre Skriptsprachen, um bereits Existierendes zu erweitern. Zumeist werden von dieser Gruppe kleine Projekte durch einzelne Personen realisiert.

Wizard: Wizards sind Progammierer, die komplexe Software entwickeln. Unter Andrerem werden von ihnen Frameworks und Architekturen erstellt. Ein Beispiel sind die Entwickler der Mozilla-Foundation, die mit Firefox und Thunderbird einen konkurrenzfähigen Open Source Browser und ein E-Mail-Programm entwickelt haben.

2.5.2. Anforderungen der Benutzer

Dieser Abschnitt zeigt die Bedeutung der Benutzer auf und klärt anschließend die Anforderungen. Die Anforderungen müssen umgesetzt werden, damit der Wechsel der Consumer in die Gruppen Author und Programmer möglichst einfach und häufig vollzogen wird. Gelingt das, steigen Akzeptanz und Nutzbarkeit der Anwendung, so dass eine größere Benutzerzahl erreicht werden kann. Auf diese Weise entsteht eine Aufwärtsspirale.

Reed's Law: Zum besseren Verständnis der Aufwärtsspirale ist Reed's Law [Reed (1999)] zu erläutern. Dieses Gesetz behandelt die Bedeutung der Benutzer. Es stellt fest, dass der Wert eines „Group Forming Network“ (GFN: Ein Netzwerk, das Gruppen von mehr als zwei Personen ermöglicht miteinander zu interagieren) mit der Anzahl der Teilnehmer exponentiell ansteigt. Diese mathematische Grundlage zeigt deutlich die Bedeutung der Teilnehmer. Um diesen Wert zu maximieren, muss das Design der Anwendung darauf ausgelegt sein, dass möglichst viele der Consumer in die Gruppe Author oder Programmer gelangen, damit sie

selbst den Inhalt der Anwendung erstellen können. Je mehr Benutzer in diesen Gruppen sind, desto explosiver entwickelt sich das Programm weiter. Es bringt den großen Vorteil mit sich, dass die aktiven Benutzer genau das erstellen, was sie benötigen. Diese Tatsache ermöglicht es den Entwicklern, sich auf andere Aufgaben zu konzentrieren. Denn häufig ist es so, dass Technologieexperten den Inhalt selbst erstellen müssen ohne genau zu wissen, welches die Bedürfnisse der Benutzer sind.

Anforderungen: Die Anforderungen, um die zuvor genannten Ziele zu erreichen, werden nachfolgend aufgelistet.

Benutzerschnittstellen: Die Benutzerschnittstellen müssen so einfach handhabbar sein, dass sie von jedem (nicht nur von Computerexperten und Programmierern) bedienbar sind.

Direktes Feedback: Das direkte Ergebnis der Handlung spielt eine wichtige Rolle. Die direkte Rückmeldung (zur Laufzeit) ermöglicht es dem Nutzer bei Fehlern sofort korrigierend einzugreifen oder Verbesserungen vorzunehmen, ohne in eine separate Entwicklungsumgebung wechseln zu müssen. Durch diese schnellen Interaktionen und kurzen Wege wird die Motivation der Nutzer besser erhalten.

Wiederverwendung: Zur schnellen Ausbreitung und Anpassung ist es sehr hilfreich, wenn der Benutzer bereits bestehende Objekte, die andere erstellt haben, einfach kopieren und bei sich einfügen kann. Dazu ist wichtig, dass die Benutzer zu jeder Zeit ohne Hindernisse direkten Zugriff auf den Quellcode haben.

Rekombination und Modifikation: Sinnvolle und logische Erweiterungen der Wiederverwendung sind die Rekombination und die Modifikation bestehender Objekte. Dadurch können neue Anwendungen entstehen. Ebenfalls können die Benutzer auf diese Weise auf bestehende Bedürfnisse besser eingehen.

In dem Zusammenhang ist auch die Theorie der verteilten Erkenntnis (Distributed Cognition) relevant, welche besagt, dass menschliches Wissen und Erkenntnis nicht auf das Individuum begrenzt sind, sondern auch durch von Menschen erstellte Artefakte in unserer Umwelt verkörpert werden [Hagberg (1997)]. In Kombination mit Reed's Law ist es sehr wahrscheinlich, dass selbst durch den kleinsten Anstieg im Verhältnis der Benutzer, die auch „Authoren" sind, ein substantieller Anstieg des Nutzens der Software daraus resultiert.

2.5.3. Parallelen zum Web 2.0

Man kann die aktive Benutzerintegration als Errungenschaft des Web 2.0 bezeichnen. In dem Zusammenhang ist die Beschreibung von Web 2.0 von O'Reilly [O'Reilly (2005)] hilfreich. Er beschreibt die Schlüsselkomponenten des Web 2.0, in denen verdeutlicht wird, wie wichtig der aktive Benutzer geworden ist. Zu den Schlüsselfaktoren gehören zum Beispiel das Design für „Hackability und „Remixibility" und das implizite wie auch explizite Involvieren des

Benutzers einen Beitrag zur Anwendung zu leisten. Dieses Konzept setzt eine lose Kopplung und das Prinzip, die Komponenten möglichst klein und einfach zu halten, voraus. Auch für Echtzeittests sollen die Benutzer gewonnen werden. Zu diesen Prinzipien gehört besonders zu kooperieren, anstatt zu kontrollieren. Weiterhin spricht O'Reilly von der „perpetual beta", die ständigen Verbesserungen und Erweiterungen der Benutzer unterliegt.

2.5.4. Beispiele

Zwei Ansätze, die das Ziel verfolgen aus den Consumern Authoren zu machen, sind Croquet mit dem Projekt „Brie" [Stearns u. a. (2006)] und Google mit dem Projekt „Friend Connect". Brie besteht aus einer Sprach-Architektur und einem Framework. Es wurde designed, um jede vorstellbare Benutzerschnittstelle, die sich ein Author vorstellen kann, zu verwirklichen. Bei Friend Connect geht es in erster Linie darum, dass Besitzer von Webseiten diese durch einfaches Einbinden von sozialen Features für Freunde und andere Personen attraktiver machen. Dieses soll durch das Einfügen kleiner Codeschnipsel ermöglicht werden. Mit dem Slogan „No programming whatsoever." verdeutlicht Google, dass das Produkt für jedermann ist. Es müssen lediglich bestehende Codestücke kopiert und auf der eigenen Seite eingefügt werden. Den Rest übernimmt Friend Connect.
Diese beiden Projekte schaffen die Möglichkeit, eine neue „Kultur des Austausches" zu gründen. Findet man also eine Anwendung oder Funktion, die etwas macht was einem gefällt, kann man das Aussehen, die Handhabung und den Inhalt kopieren. Durch einfache Rekombination und Modifikation entsteht die Möglichkeit, die Komponenten neu zusammenzustellen und dadurch etwas zu erschaffen, was sich der Author so nicht vorstellen konnte.

2.6. Android

Bei Android handelt es sich um ein Projekt der Open Handset Alliance. Die Open Handset Alliance besteht aus über 30 Technologie- und Handyherstellern. Ziel ist es, die erste vollständige, offene und freie mobile Plattform zu entwickeln. Um frühzeitig die Möglichkeit zu schaffen, um diese Plattform eine offene Entwicklergemeinschaft aufbauen zu können, wurde ein „early look" in Form eines Android Software Development Kits (SDK) angeboten. Aktuell ist dieses in der Version 0.9 verfügbar. Diese Alliance besteht aus namenhaften Firmen wie T-Mobile, Intel, NVIDIA, HTC, Motorola, Samsung, Ebay, Google und vielen mehr. Erste Endgeräte sollen in der zweiten Jahreshälfte 2008 erscheinen. Bis Endgeräte im Handel erhältlich sein werden, wird der zur Verfügung gestellte umfangreiche Emulator bei der Anwendungserstellung genutzt. Der für diese Arbeit interessante Bereich ist die Software Plattform.

2.6.1. Software Plattform

Die Software Plattform für mobile Endgeräte besteht aus einem Betriebssystem, der Middleware und Applikationen. Mithilfe des SDKs können Softwareentwickler mit der Programmiersprache Java Anwendungen erstellen. Zusätzlich zu dem SDK und dem Emulator werden dem Entwickler noch die notwendigen APIs, eine umfassende Dokumentation und Beispielanwendungen zur Verfügung gestellt. Im Folgenden wird genauer auf die Softwarearchitektur eingegangen. Die Architektur ist mehrschichtig und wird in die Schichten Linux Kernel, Libraries, Android Runtime, Application Framework und Applications unterteilt.

Linux Kernel: Android basiert auf einem Linux Kernel in der Version 2.6. Auf diesem Linux laufen die Kern System Services (Sicherheit, Speichermanagement, Prozessmanagement, Netzwerk Stack und Treiber Model). Der Kernel dient weiterhin als Abstraktionsschicht zwischen der Hardware und der Software.

Android Runtime: Die zugrunde liegende Java Virtual Machine (JVM) namens Dalvik wurde für den Einsatz auf Embedded Systems entwickelt. Es läuft jede Android Applikation als eigener Process mit seiner eigenen Dalvik VM. Für die effiziente Nutzung wird ein extra Dateiformat (.dex) benutzt. Die VM arbeitet mit Registern. Dabei werden die Klassen mit einem Java Compiler übersetzt, die wiederum mit dem mitgelieferten „dx"Tool in das .dex-Format übertragen werden. Die Android Runtime besteht nicht nur aus der Dalvik VM, sondern auch aus den sogenannten Core Libraries. Das Meiste dieser Funktionalität der Core Libraries wird in der Java Programmiersprache zur Verfügung gestellt.

2.6.2. Framework

Nachdem Kernel und Runtime im vorherigen Abschnitt beschrieben wurden, wird an dieser Stelle auf die zur Verfügung stehenden Libraries und das darauf aufbauende Framework eingegangen. Auf dieser Basis können relativ einfach Anwendungen erstellt werden.

Libraries: In Android sind eine Menge von C/C++ Bibliotheken enthalten. Diese werden von verschiedenen Komponenten des Android Systems benutzt. Die darin enthaltenen Funktionalitäten werden dem Entwickler durch das Android Application Framework zur Verfügung gestellt. Auf einige dieser Bibliotheken wird im Folgenden kurz eingegangen. Es gibt unter anderem eine mächtige und leichtgewichtige relationale Datenbank namens SQLite, die allen Anwendungen zur Verfügung steht, 3D Bibliotheken basierend auf Open GL ES 1.0, eine 2D Graphik Maschine SGL, LibWebCore, eine moderne Webbrowser Maschine.

Application Framework: Auf den gerade genannten Grundfunktionalitäten setzt das Application Framework auf. Bei der Erstellung des Frameworks wurde Wert auf die einfache

Wiederverwendung der Komponenten gelegt. Jede Applikation kann seine Komponenten veröffentlichen und jede andere Applikation kann bei Bedarf diese Komponente einsetzen. Dieser Mechanismus erlaubt es auch existierende Komponenten durch andere zu ersetzen. Grundsätzlich werden allen Anwendungen durch das Application Framework bestimmte Services und Systeme zur Verfügung gestellt. Nachfolgend werden einige aufgeführt:

Views: Es werden eine Vielzahl an Views (auch erweiterbar) mitgeliefert, um das Frontend zu erstellen. Diese enthalten Listen, Textboxen, Buttons einen eingebetteten Browser und mehr.

Content Provider: Durch den Content Provider wird es ermöglicht von einer Applikation auf eine andere zugreifen zu können. Beispielsweise um in einer eigenen Anwendung auf die Kontakte aus dem Adressbuch zugreifen zu können.

Resource Manager: Durch den Resource Manager kann auf Ressourcen wie beispielsweise Bilder zugegriffen werden.

Notification Manager: Treten irgendwelche für den Nutzer interessanten Hinweise auf, werden diese durch den Notification Manager in der Statusleiste angezeigt.

Activity Manager: Der Activity Manager kümmert sich um die Verwaltung des Speichers. Er schließt Anwendungen und gibt damit Speicher frei, wenn dieser benötigt wird. Wenn man allerdings zurück zu einer vom Activity Manager geschlossenen Anwendung navigiert, wird diese mit den vorherigen Parametern neu gestartet, so dass der Benutzer bis auf eine kleine zeitliche Verzögerung davon nichts mitbekommt.

XMPP: Über eine Verbindung zu einem Server (GTalk) können sowohl GTalk Instant Messages geschickt als auch P2P Message Passing genutzt werden. Die Vorteile gegenüber SMS, welches auch möglich wäre, liegen darin, dass es schneller geht und kostenlos ist.

Applications: Aufbauend auf dem Application Framework werden die Applikationen erstellt. Bei der Installation des SDKs werden ein paar Beispielanwendungen mitgeliefert. Das gesamte Application Framework steht den Entwicklern während der Anwendungserstellung zur Verfügung.
Die nachfolgende Grafik (Abbildung 2.1) veranschaulicht die beschriebene Struktur.

2.6.3. Weitere mobile Plattformen

J2ME: J2ME (Java 2 Micro Edition) ist eine Java Version für mobile Geräte wie beispielsweise Handys. Es wurden verschiedene Standards, sogenannte Java Specification Request (JSR), entwickelt. Grundsäztlich laufen diese Anwendungen auch MIDlet genannt auf allen Java-fähigen Handys. Allerdings ist dabei zu beachten, dass die von dem MIDlet genutzten JSR auch von dem jeweiligen Handy unterstützt werden. Sun als Hersteller von J2ME hat

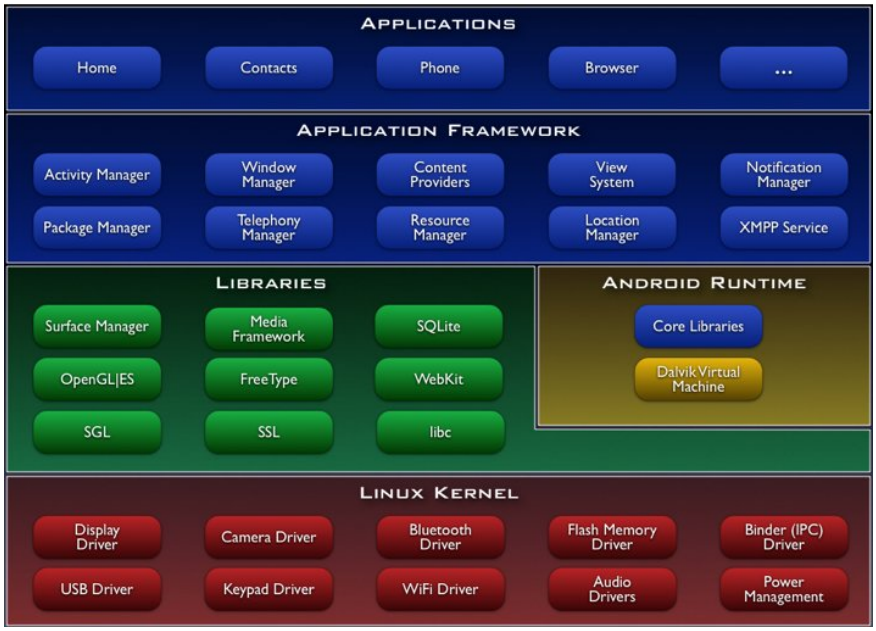

Abbildung 2.1.: Android Architektur [Android (2007)]

angekündigt, dass J2ME nicht weiterentwickelt wird, da die mobilen Geräte immer leistungsfähiger werden und somit in naher Zukunft J2SE auf diesen Geräten verfügbar sein werde.

.NET Compact Framework: Das .NET Compact Framework ist Teil des .NET Frameworks und speziell ausgerichtet für die Nutzung auf mobilen Endgeräten. Dabei wurde das .NET Framework um Klassen, die für mobile Endgeräte zu schwergewichtig oder nicht von Nutzen sind, reduziert. Programme, die mit diesem Framework entwickelt werden, laufen ausschließlich auf der mobilen Version des Microsoft Betriebssystems Windows. Allerding ist dieses Betriebsystem auf verschiedenen Smartphones, Pocket PCs und PDAs bereits installiert. Dadurch erreicht Microsoft eine gewisse Art der Plattformunabhängigkeit. Das Betriebssystem selbst ist nicht Open Source und auch nicht frei verfügbar. Meist ist es auf den Endgeräten bereits installiert und man erwirbt mit dem Kauf eines solchen Gerätes eine kostenpflichtige Lizenz des Betriebssystems.

iPhone: Das iPhone ist ein von Apple entwickeltes Smartphone. Als Betriebssystem dient ein angepasstes Mac OS X, welches ebenfalls nicht Open Source ist. Für das iPhone wurde eine mobile Variante des Cocoa-Frameworks namens Cocoa Touch entwickelt. Apple stellt Cocoa touch, eine integrierte Entwicklungsumgebung (Xcode) und einen Emulator für die Entwicklung auf dem Mac als SDK zur Verfügung. Die Entwickler haben die Wahl, ob die

Software zur freien Verfügung gestellt wird oder nicht. Entscheidet sich der Entwickler für die kostenpflichtige Variante, wird Apple mit 30% der Einnahmen beteiligt. Um die eigene Software über den App Store zu veröffentlichen, ist eine kostenpflichtige Lizenz notwendig. Bisher ist das iPhone das einzige Gerät, auf dem das Betriebssystem inklusive der Anwendungen laufen. Besonders die Bedienung und die Multitouchfähigkeit stechen bei dem iPhone positiv hervor.

2.6.4. Bewertung

Wie zuvor beschrieben gibt es verschiedene Möglichkeiten mobile Anwendungen zu erstellen. In einem vorangegangenen Projekt wurde sich mit J2ME auauseinander gesetzt. Die Entwicklung eines protoypischen Messengers gelang, auch wenn teilweise an die Grenzen von Java ME gestoßen wurde. Bei J2ME gibt es eine Reihe von Spezifikationen, die nicht bei jedem Handy umgesetzt sind. Dieses führte zu Problemen bei der Nutzung der Software auf unterschiedlichen Endgeräten. Ein Nachteil war weiterhin die schlechte Dokumentation. Das .Net Compact Framework hat den entscheidenen Nachteil, dass die Software nur auf Endgeräten läuft, die Windows Mobile, ein geschlossenes Betriebssystem, nutzen. Bei dem iPhone verhält sich dieses ähnlich, die Software ist nur auf dem iPhone oder dem iPod Touch betriebsfähig. Zudem sind die Freiheiten der Entwickler relativ stark eingeschränkt. Der große Vorteil bei Android ist, dass es Open Source ist und der Quellcode gut dokumentiert ist. Als Entwickler stehen einem damit alle Möglichkeiten für die Entwicklung zur Verfügung ohne seitens des Herstellers eingeschränkt zu werden. Eine erste Evaluierung des sich noch in der Betaversion befindlichen Software Development Kit (SDK) erschien erfolgversprechend. Da der in dieser Arbeit zu entwerfende Messenger im Kontext von Open Source steht, wurde sich für Android entschieden. Ein Nachteil bei Android besteht darin, dass es noch keine Endgeräte gibt, so dass bis zur Verfügbarkeit der Endgeräte der Emulator ausreichen muss. Da das Android-Projekt noch sehr neu ist, werden die zukünftigen Geräte aktuellen Anforderungen entsprechen.

3. Analyse

Im vorherigen Kapitel wurden die notwendigen Grundlagen geschildert. Darauf aufbauend wird in diesem Kapitel zunächst das Szenario beschrieben. Anschießend werden die an den Messenger zu stellenden funktionalen sowie nicht-funktionalen Anforderungen erläutert. Zuletzt wird dieser mit ähnlichen Projekten verglichen und entsprechende Unterschiede werden herausgestellt.

3.1. Szenario

Dieses Szenario enstand ausgehend von der Möglichkeit über stationäre Computer Sofortnachrichten (Instant Messaging) mit Freunden, Bekannten, Arbeitskollegen etc. austauschen zu können. Mobile und leistungsfähige Geräte sind weit verbreitet und mobiler Datenaustausch ist heute in zufriedenstellendem Maße möglich. Daher soll in einem ersten Schritt das Instant Messaging von der stationären auf die mobile Welt übertragen werden, wobei dieses Szenario sogar noch weiter geht.
Das Instant Messaging mit mobilen Geräten hat das Potential, die Kommunikation und das soziale Verhalten der Menschen zu prägen. Die hinzu gewonnene Komponente der Mobilität kann in diesem Zusammenhang in Form von Context-Awareness aktiv genutzt werden. Mit aktueller Technik ist es möglich, grundsätzlich jederzeit und überall eine Verbindung zum Internet und damit zum Rest der Welt aufnehmen zu können. Auf diese Weise soll ein ubiquitärer kontextabhängiger Messenger entstehen. Das folgende Szenario soll dies verdeutlichen:
Karl und Karla sind ein junges Paar, das sehr gerne kommuniziert und viel reist. Karl arbeitet in seiner Freizeit gerne mit Computern und hat schon das ein oder andere Plugin für die Freundesnetzwerke, die er benutzt, geschrieben. Karl versteht sich als „Programmer“. Seit Neuestem gibt es die Möglichkeit, dass er die Plugins, die er für seine Netzwerke schreibt, aufgrund einer Kooperation der Netzwerke miteinander, nur noch einmal schreiben muss. Dadurch muss er auch seine Kontaktdaten nur noch einmal pflegen. Die Zeit, die Karl jetzt weniger am Computer verbringt, nutzt Karla nun mehr, um mit ihren Freunden zu chatten. Ihre Freunde benutzen verschiedene Messenger. Glücklicherweise gibt es in diesem Bereich bereits Programme, die verschiedene Protokolle in einem Messenger zusammenfassen. Seitdem Karl und Karla für ihre Smartphones auch einen günstigen Datentarif besitzen, haben sich beide einen Instant Messenger auf den Geräten installiert. Den Messenger nutzen die beiden jetzt jederzeit und überall, um miteinander und mit anderen in Kontakt treten zu können.
Zu Karls Freude ist dieser Messenger Open Source und kooperiert auch mit seinen Netzwer-

ken. So haben beide ihre gesamten Freundeslisten nun auch auf ihren Smartphones. Karl stellt sich, vor ein Plugin für den Messenger zu entwickeln, damit erkannt wird, welche Kontakte aus seinem Adressbuch, seinen Freundesnetzwerken und seinen Buddylisten mehrfach vorhanden sind und diese automatisch zusammengefasst werden. Als er feststellt, dass die Idee seine Möglichkeiten überschreitet, fragt er in der Community nach, ob einer der „Wizards“ der Community diese Funktionalität nicht für ihn umsetzen könnte. Er selbst hat sich überlegt, ein Plugin zum Austausch von Dateien zu schreiben, um Bilder direkt mit seinen Freunden austauschen zu können.
An diesem Wochenende ist Karl in Hamburg. Es ist Samstag und Karl will eigentlich eine Stadtführung machen. Da fällt ihm die neue Funktionalität seines Smartphones ein. Er geht durch die Speicherstadt und an verschiedenen Sehenswürdigkeiten vorbei. Sobald er in die Nähe einer solchen Sehenswürdigkeit kommt, signalisiert sein Smartphone durch einen Hinweis, dass Zusatzinformationen, die sogenannten Annotationen, vorhanden sind. Diese kann er mit einem Klick sofort abrufen. Eine Weile später geht Karl an einem offensichtlich interessanten Gebäude vorbei, über das keine Zusatzinformationen vorhanden sind. Ein Schild an dem Gebäude besagt lediglich, wovor er sich befindet. Diese Informationen reichen ihm noch nicht aus. Er geht mit seinem Smartphone ins Internet und sucht nach weiterführenden Informationen. Er stellt sich nun direkt vor das Gebäude, da er weiß, dass seine Position von dem System erfasst wird. Mit einem Klick lädt er die recherchierten Informationen hoch. Dadurch hat er für alle anderen Benutzer des Systems eine neue Information bereitgestellt. Dies schreibt er auch gleich seiner Freundin über den Messenger, da ihr Status signalisiert, dass sie bereits telefoniert. Während Karla zu Hause über ihr Smartphone kostenlos mit einer Freundin telefoniert, liest sie die Nachricht, geht zum Computer, um sich dort den neuen Eintrag von ihrem Freund anzuschauen. Da sie zufällig mehr darüber weiß als Karl, editiert sie den Beitrag und stellt dadurch die vervollständigte Information allen zur Verfügung. Ganz unbewusst ist Karla von einer „Consumerin“ zu einer „Authorin“ geworden. Von diesem praktischen Feature erzählt Karla ihrer Freundin, dabei kommen die beiden auf den Gedanken, Karl überraschend zu besuchen. Beide machen sich auf den Weg. Mittlerweile ist es abends und Karl hat mit einem alten Bekannten eine Tour über den Hamburger Kiez geplant. Sein Freund benutzt diesen Messenger zwar nicht, aber da der Messenger auch die Kontakte mit dem Adressbuch des Smartphones synchronisiert, wählt Karl ihn wie alle anderen Kontakte einfach aus seiner Messengerbuddyliste aus. Dort wird ihm angezeigt, dass er ihm keine Sofortnachricht schicken kann. Alternativ wird ihm angeboten, seinem Freund eine E-Mail oder eine SMS zu schicken. Da Karl den Status seines Freundes nicht einsehen kann und demnach nicht weiß, ob dieser gerade online ist oder nicht, entscheidet er sich kurzerhand für die nicht kostenlose Lösung, ihm eine SMS zu schicken. Er hofft, dass sein Freund diese schneller bemerkt und sich bei ihm meldet. Ein paar Minuten später bekommt Karl eine Antwort. Die beiden verabreden sich auf die herkömmliche Art für 21 Uhr vor einem bestimmten Club. Sie betreten den Club, der ebenfalls an der Community teilnimmt und als Annotation eingetragen ist. Für diesen Abend bieten sie eine besonders vielversprechende Show an. Andere „Authoren“ haben diesen Club und diese Veranstaltung bereits mehrfach gut bewertet. Karla ist mit ihrer Freun-

din in Hamburg angekommen und hat sich von ihrem Smartphone zu Karls letzter bekannten Position navigieren lassen. Diese ist mittlerweile eine halbe Stunde alt. An der Position angekommen steht sie vor einem großen Club. Da Karla weiß, dass die Positionsfeststellung in Gebäuden schwierig oder eventuell gar nicht funktioniert, treffen die beiden die Annahme, dass sich Karl in dem Club befindet. Die beiden betreten den Club und geben nach 15 Minuten intensiver Suche auf. Der Club ist zu groß und zu voll. Karla zückt ihr Smartphone und schreibt Karl eine Nachricht, der sofort darüber in Kenntnis gesetzt wird. Er liest die Nachricht und ist einen kurzen Augenblick verwundert, dass er von seiner Freundin gebeten wird, an die Cocktailbar in dem Club zu kommen, indem er sich gerade befindet. Schon fällt ihm die Funktion seines Messengers ein, dass die Buddies ihre Position anderen mitteilen, wenn der eingestellte Status dies vorsieht. Entsprechend gibt es zur Bewahrung der Privatssphäre verschiedene Stati. Gegen Mitternacht vibriert das Smartphone noch einmal und teilt ihm mit, dass einer seiner Freunde zufällig in der Nähe ist. Spontan ruft er seinen Freund an und fragt, ob er nicht eine Kleinigkeit essen wolle. Später entscheiden sich alle für den Heimweg. Während Karl über sein Smartphone den Heimweg mit den öffentlichen Verkehrsmitteln prüft, gibt Karla die Adresse des Hotels in ihr Smartphone ein und lässt sich eine Route dahin berechnen. Gemeinsam entscheiden sie, den Weg zu Fuß zurückzulegen.
Abschließend zum Szenario muss der Vollständigkeit halber erwähnt werden, dass zwar von einer weitestgehend flächendeckenden Internetverfügbarkeit ausgegangen wird, aber dies noch nicht der Realität entspricht. Weltweit kann davon nicht die Rede sein und in den Industriestaaten sind häufig noch ländliche Gegenden nicht abgedeckt. Allerdings schreitet die Abdeckung immer weiter voran.

3.2. Funktionale Anforderungen

Dieser Abschnitt beschäftigt sich mit den funktionalen Anforderungen. Als Grundlage werden in einem ersten Schritt allgemein mithilfe des System-Use-Case-Diagramms die funktionalen Anforderungen aus Sicht der Benutzer beschrieben. Eine folgende Use-Case-Beschreibung wird die einzelnen Use-Cases näher erläutern. Durch ein zugehöriges Aktivitätsdiagramm werden die Anforderungen verfeinert, indem die möglichen Aktionen dargestellt werden. Abschließend werden die funktionalen Anforderungen aufgelistet und nach verschiedenen Stufen der Wichtigkeit unterschieden.

3.2.1. Systembeschreibung

Das folgende System-Use-Case-Diagramm verdeutlicht anschaulich auf einer hohen Ebene die Funktionalitäten. Das System im Konkreten ist der mobile ubiquitous context-aware Messenger (mucM). Das System-Use-Case-Diagramm (Abb.: 3.1) besteht aus fünf Aktoren und elf Use-Cases. Auf der linken Seite sind vier Aktoren zu sehen, wobei drei dieser Aktoren vom Consumer abgeleitet werden. Je größer die Ableitungshierarchie, desto mehr Use-Cases

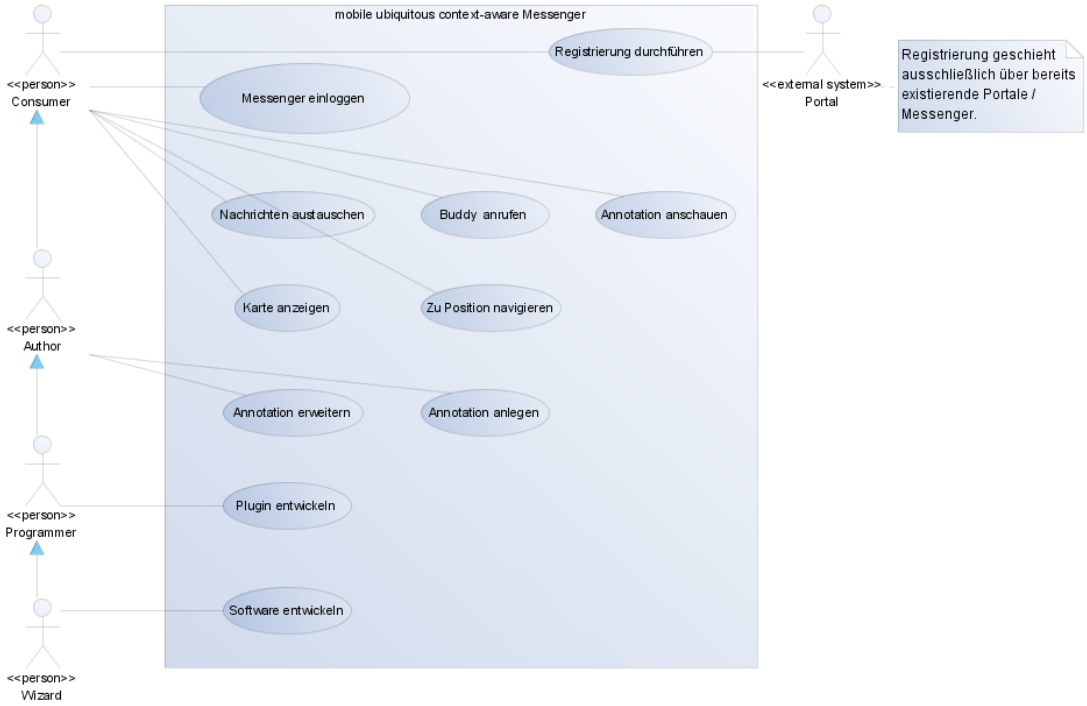

Abbildung 3.1.: Use Case: Mobile Ubiquitous Context-Aware Messenger

kann der Aktor ausführen. Auf der rechten Seite befindet sich ein Aktor eines fremden, aber kooperierenden Systems, der für die Registrierung zuständig ist. Die Use-Cases des mucM werden an dieser Stelle durch eine semi-formale Notation, in Form einer tabellarischen Beschreibung der einzelnen Use-Cases detaillierter, beschrieben. Exemplarisch sind nachfolgend zwei Use-Cases angeführt. Die anderen Use-Cases befinden sich im Anhang A.1.

Use Case	**Registrierung durchführen**
Beschreibung	Der Benutzer muss sich einmalig registrieren. Dies geschieht durch ein externes kooperierendes System, das Portal.
Akteure	Consumer, Author, Programmer, Wizard, Portal
Vorbedingungen	Es muss eine Internetverbindung vorhanden sein.
Normalablauf	1. Der Benutzer wählt ein kooperierendes Netzwerk aus. 2. Der Benutzer füllt die Pflichtfelder aus. 3. Der Benutzer folgt den weiteren Anweisungen. 4. Der Benutzer registriert sich.
Nachbedingungen	Es besteht ein Zugang, mit dem man sich beim mucM anmelden kann.
Fehlerfall	1. Besteht keine Internetverbindung, kann keine Registrierung durchgeführt werden. 2. Schlägt die Registrierung fehl, obliegt es dem externen System, wie verfahren wird. In der Regel kann der Benutzer versuchen, die Registrierung erneut durchzuführen.

Tabelle 3.1.: Use Case: Registrierung durchführen

Use Case	**Messenger einloggen**
Beschreibung	Der Benutzer meldet sich am mucM an.
Akteure	Consumer, Author, Programmer, Wizard
Vorbedingungen	Es muss eine Internetverbindung vorhanden sein. Der Benutzer muss registriert sein.
Normalablauf	1. Eine Synchronisation mit dem Adressbuch findet statt. 2. Eine Synchronisation mit den Netzwerkkontakten findet statt. 3. Eine Synchronisation der Buddylisten der Messenger findet statt. 4. Es werden die Buddies mit aktuellen Stati angezeigt. 5. Die Annotationen in der Nähe werden geladen.
Nachbedingungen	Der Benutzer befindet sich im mucM und kann alle Funktionen benutzen.
Fehlerfall	1. Besteht keine Internetverbindung, kann keine Anmeldung durchgeführt werden. 2. Ist der Benutzer nicht registriert, wird dieser darauf hingewiesen. 3. Ist das Passwort falsch, wird der Benutzer ebenfalls darauf hingewiesen. 4. Hat der Benutzer das Passwort vergessen, kann er sich das Passwort an seine hinterlegte E-Mail-Adresse schicken lassen.

Tabelle 3.2.: Use Case: Messenger einloggen

3.2.2. Ablauf des Messengers

An dieser Stelle wird als Hilfsmittel ein Aktivitätsdiagramm 3.2 eingesetzt, um die Use-Cases aus dem vorherigen Abschnitt zu verdeutlichen. Dabei gibt das Diagramm den Ablauf der Gesamtanwendung wieder. Das Diagramm ist aus Sicht des „Authors“ erstellt und spiegelt die einzelnen Aktionen und deren Zusammenhang bei der Nutzung des mucM wider.

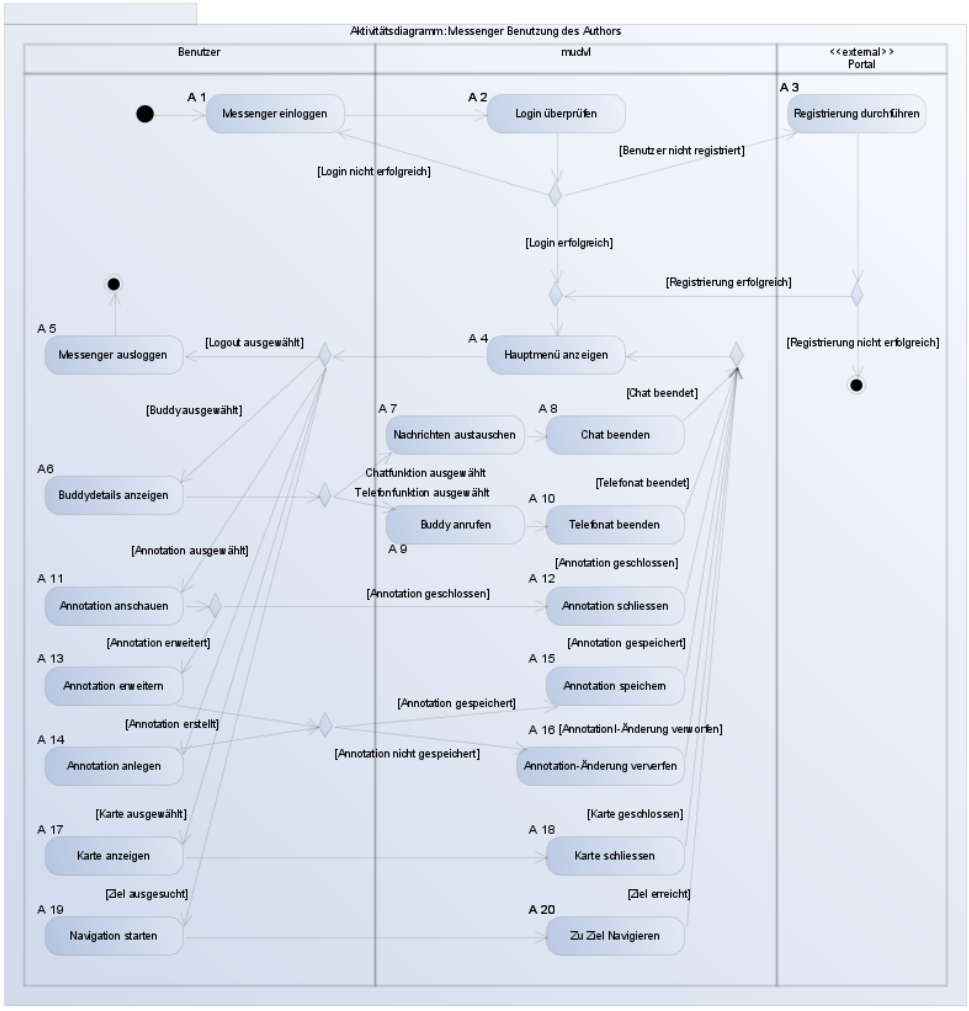

Abbildung 3.2.: Aktivitätsdiagramm: Mobile Ubiquitous Context-Aware Messenger

3.2.3. Beschreibung der funktionalen Anforderungen

Auf Basis des System-Use-Case-Diagrammes (Abbildung 3.1), der Beschreibung der einzelnen Use-Cases und der Darstellung der Aktionen durch das Aktivitätsdiagramm (Abbildung 3.2), können nun die Anforderungen aufgeschrieben werden. Dazu wird eine Unterteilung in essentielle (MUSS), empfohlene (SOLL) und zukünftige (WIRD) funktionale Anforderungen vorgenommen:

A 1 Der Benutzer muss die Möglichkeit haben, sich beim mucM einzuloggen.

A 2 Der mucM soll überprüfen, ob der Login des Benutzers richtig ist. Ist das Passwort falsch, kann der Benutzer erneut versuchen sich einzuloggen (siehe A 1).

A 3 Falls der Benutzer nicht registriert ist, soll ihm die Möglichkeit gegeben werden, sich zu registrieren. Wird keine Registrierung durchgeführt, wird der mucM beendet.

A 4 Falls der Login oder die Registrierung erfolgreich war, muss dem Benutzer das Hauptmenü angezeigt werden.

A 5 Falls der Benutzer die Aktion „Messenger ausloggen“ gewählt hat, muss der mucM den Benutzer abmelden und das Programm beenden.

A 6 Falls der Benutzer die Aktion „Buddydetails anzeigen“ gewählt hat, muss der mucM dem Benutzer die zugehörigen Details über den Buddy anzeigen.

A 7 Falls der Benutzer die Aktion „Nachrichten austauschen“ gewählt hat, muss der mucM einen Chat mit dem Buddy zur Verfügung stellen.

A 8 Wählt der Benutzer die Aktion „Chat beenden“ aus, muss der mucM den Chat beenden und wieder das Hauptmenü anzeigen.

A 9 Falls der Benutzer die Aktion „Buddy anrufen“ auswählt, muss der mucM ein Telefonat zwischen dem Benutzer und dem Buddy herstellen.

A10 Wählt der Benutzer die Aktion „Telefonat beenden“ aus, muss der mucM das Telefonat beenden und wieder das Hauptmenü anzeigen.

A11 Falls der Benutzer die Aktion „Annotation anschauen“ auswählt, muss der mucM die Details der Annotation darstellen.

A12 Wählt der Benutzer die Aktion „Annotation schließen“ aus, muss der mucM die aktuelle Ansicht schließen und wieder das Hauptmenü anzeigen.

A13 Falls der Benutzer die Aktion „Annotation erweitern“ auswählt, soll der mucM die Möglichkeit bieten, die gerade dargestellte Annotation zu verändern.

A14 Falls der Benutzer die Aktion „Annotation anlegen“ auswählt, soll der mucM eine neue Annotation anlegen.

A15 Falls der Benutzer die Aktion „Annotation speichern“ auswählt, soll der mucM die Annotation speichern und danach wieder das Hauptmenü anzeigen.

A16 Falls der Benutzer die Aktion „Annotation-Änderung verwerfen“ auswählt, soll der mucM ohne die Änderungen zu speichern das Hauptmenü anzeigen.

A17 Falls der Benutzer die Aktion „Karte anzeigen“ auswählt, muss der mucM die Karte anzeigen.

A18 Wählt der Benutzer die Aktion „Karte schließen“ aus, muss der mucM wieder das Hauptmenü anzeigen.

A19 Falls der Benutzer die Aktion „Navigation starten“ auswählt, wird der mucM ihn zum Ziel navigieren.

A20 Der mucM wird den Benutzer zum Ziel navigieren, bis dieses erreicht ist. Anschließend wird wieder das Hauptmenü angezeigt.

3.2.4. Fehlerfälle

Auf die möglichen Fehlerfälle wurde bereits bei der Beschreibung der einzelnen Use Cases eingegangen. Die Behandlung von Fehlern in mobilen Systemen ist ein sehr weites Thema und kann den Inhalt mehrerer Abschlussarbeiten füllen. Aus diesem Grund wird dieser Bereich nicht weiter vertieft.

3.3. Nicht-funktionale Anforderungen

Es gibt neben den funktionalen Anforderungen noch nicht-funktionale Anforderungen, die zu einer vollständigen Anforderungsanalyse dazu gehören. In Kundenprojekten sind die Aufgaben der nicht-funktionalen Anforderungen, die Zufriedenheit der Kunden zu erhöhen, die Rechtssicherheit zu stärken, die Produktivität zu steigern, die Planung abzusichern und die Spezifikationen zu vervollständigen. Für die Masterarbeit ist der letzte Punkt der Bedeutendste. Durch die Vervollständigung der Spezifikationen kann eine bessere Grundlage für die Konzepterstellung gegeben werden. Die nicht-funktionalen Anforderungen werden der Übersichtlichkeit halber in verschiedene Bereiche unterteilt. Grundsätzlich ist zu den nicht-funktionalen Anforderungen zu sagen, dass sie alle für einen reibungslosen Ablauf des Systems notwendig sind. Im Rahmen dieser Arbeit können aber nicht alle Anforderungen umgesetzt werden.

3.3.1. Technische Anforderungen

Unter technischen Anforderungen sind beispielsweise Hardware-, Architektur- oder Programmiersprachenanforderungen zu verstehen.

Programmiersprache: Aufgrund der weiten Verbreitung und der guten Kenntnis des Autors dieser Arbeit soll als Hauptprogrammiersprache Java gewählt werden.

Geräteunabhängigkeit: Es ist wichtig, dass sich der mucM möglichst auf verschiendenen Hardwareplattformen installieren lässt. Dadurch wird erreicht, dass der Messenger möglichst weit verbreitet werden kann.

Open Source: Möglichst alle Komponenten sollen Open Source sein. Das entspricht dem Gedanken der Community und ermöglicht es, wenn nötig, bis direkt an die einzelnen Hardwarekomponenten zu gelangen und diese ansprechen zu können.

Datenaustausch 1: Der mucM soll echtzeitfähig sein.

Datenaustausch 2: Um den Nachrichtenaustausch zwischen verschiedenen IM zu ermöglichen, soll der mucM verschiedene Protokolle unterstützen. Beispiele dafür sind Jabber, ICQ oder MSN.

Datenaustausch 3: Für die Telefonfunktion ist es notwendig, dass die Kommunikation in Echtzeit stattfinden kann.

Datenaustausch 4: Das Telefonieren soll über verschiedene Technologien ermöglicht werden (z.B.: VOIP).

Software 1: Das Framework soll in der gewählten Programmiersprache als Open Source vorliegen, um es individuell anpassen zu können. Der Open Source Aspekt ist auch hier sehr wichtig, damit das Projekt von jedermann frei genutzt und erweitert werden kann.

Software 2: Der mucM soll als Service auch im Hintergrund weiterlaufen können.

Software 3: Für die Interaktion mit verschiedenen sozialen Netzwerken soll eine allgemeine API eingesetzt werden, die möglichst viele Netzwerke benutzen.

Systemarchitektur: Um das System flexibel weiterentwickeln zu können (Open Source), sollen die einzelnen Komponenten gut gekapselt sein, um sie beispielsweise durch verschiedene Technologien umsetzen oder austauschen zu können. Die Erweiterbarkeit spielt damit auch eine wichtige Rolle für die zukünftige Entwicklung durch die Community.

Benutzer: Die Unterteilung von Consumer bis zum Wizard ist relativ dynamisch und von dem Interesse und den Fähigkeiten des Benutzers abhängig. Der Benutzer kann gleichzeitig mehrere Rollen einnehmen. Diese Möglichkeit ist sogar ziemlich wahrscheinlich.

Erstellt der Programmer beispielsweise ein neues Plugin für eine konkrete Anwendung, wird er die Anwendung sehr wahrscheinlich auch selbst benutzen und auch Inhalte dafür erstellen.

3.3.2. Anforderungen an die Benutzungsoberfläche

Dieser Teil der Anforderungen beschäftigt sich mit den Human-Machine Interface (HMI) Spezifikationen.

Bedienung 1: Der mucM soll mit einer möglichst intuitiven Bedienung ausgestattet werden. Zum einen soll er sich an der Bedienung des Endgerätes orientieren. Zum anderen soll jede Mensch-Maschine-Interaktion ohne Hilfsmittel durchgeführt werden können.

Bedienung 2: Die Menüführung startet im Hauptmenü, zu dem man durch wenige Interaktionen einfach zurückgelangen kann, um die Orientierung zu erleichtern. Eine grobe Vorstellung liefert die Abbildung 3.2.

Bedienung 3: Treten Ereignisse auf, die den Benutzer interessieren, so sollen diese auf der GUI angezeigt werden. Diese Notifikation, durch die der Benutzer genauere Informationen zu dem Ereignis bekommen kann, soll interaktiv sein.

Aussehen: Das Aussehen des mucM soll sich an der Systemumgebung der Endgerätes orientieren.

Performance: Die Reaktionszeiten auf eine Benutzerinteraktion mit dem mucM sollen soweit möglich geringer als eine Sekunde sein.

Internationalisierbarkeit: Damit der mucM auch weltweit eingesetzt und bedient werden kann, ist die Internationalisierung bei der Architekturerstellung ebenfalls zu berücksichtigen.

3.3.3. Qualitätsanforderungen

Die Qualitätsanforderungen treffen eine Aussage über die Güte eines Produktes. Dazu werden die Qualitätsanforderungen in verschiedene Bereiche unterteilt.

Funktionalität 1: Der mucM soll eine zuverlässige Nachrichtenübertragung ermöglichen und im Fehlerfall dem Benutzer die misslungene Übermittlung mitteilen. Dabei sollen 99,9% der Nachrichten übertragen werden.

Funktionalität 2: Der mucM soll die Kommunikation zwischen den verschiedenen Instant Messaging (IM) und sozialen Netzwerken ermöglichen.

Zuverlässigkeit 1: Falls die Verbindung unterbrochen wird und der Benutzer etwas geändert hat, das einer Verbindung bedarf, um die Änderung für alle zur Verfügung zu stellen, darf dies nicht zu Inkonsistenzen im System führen. Die Änderung oder Anfrage soll lokal zwischengespeichert werden, bis die Verbindung wiederhergestellt ist. Dann soll versucht werden die Änderung durchzuführen.

Zuverlässigkeit 2: Der mucM soll zu jeder Zeit und von überall aus funktionsbereit sein, soweit es die technischen Möglichkeiten wie beispielsweise GPS-Empfang zulassen.

Effizienz 1: Beim Instant Messaging sollen gemäß der bekannten Desktoplösungen so geringe Verzögerungszeiten wie möglich entstehen. Die Verzögerungen sollen unter einer Sekunde liegen.

Effizienz 2: Jede Aktivität im mucM soll so schnell wie möglich ausgeführt werden und darf maximal eine Sekunde dauern.

Änderbarkeit 1: Das System soll so gestaltet sein, dass Änderungen an einzelnen Komponenten des mucM keine Auswirkungen auf andere Systemteile haben.

Änderbarkeit 2: Einzelne Komponenten sollen so gekapselt sein, dass eine Realisierung der gleichen Funktionalität mit einer anderen Technologie keine Neuentwicklung der ganzen Komponente notwendig macht.

Anforderungen an den Schutz der Systemumgebung und an den Schutz des Systems 1: Jeder Benutzer des mucM soll sich bei der Anmeldung mittels Benutzernamen und Passwort authentifizieren. Das Single-Sign-On-Prinzip scheint eine gute Möglichkeit zu sein.

Anforderungen an den Schutz der Systemumgebung und an den Schutz des Systems 2: Für den Benutzer soll es verschiedene Stati geben. Zum einen sollen sie den Buddies den derzeitigen Status mitteilen und zum anderen soll die Privatsphäre in den verschienden Stati unterschiedlich stark gewahrt werden.

Wartbarkeit: Damit sich das Projekt von der Community möglichst gut weiterentwickeln lässt, ist eine hohe Anforderung an die gute Wartbarkeit unerlässlich.

Testbarkeit: Damit die Software stabil läuft, muss eine gute Testbarkeit gewährt sein.

3.3.4. Sicherheit

Der gesamte Datenaustausch soll über eine gesicherte Datenleitung erfolgen (End-to-End-Security). Dabei spielen verschiedene Bereiche eine Rolle. Vertraulichkeit (Privacy), damit nur die gewünschten Personen (lesenden) Zugriff auf die privaten Daten haben. Authentizität (Authenticity), damit festgestellt werden kann, ob der Kommunikationspartner auch wirklich

der gewünschte Kommunikationspartner ist. Dieses kann pro Nachricht oder pro Verbindung geschehen. Bei der Autorisation geht man einen Schritt weiter. Diese setzt nach der Authentifizierung an. Dabei wird der Zugriff für authentifizierte Teilnehmer beschränkt. Die Integrität (Integrity) sorgt dafür, dass die Daten so transportiert werden, wie sie übergeben werden. Es geht darum, Veränderungen durch Dritte zu erkennen (Entfernen, Verändern und Verdoppeln von Nachrichten). Für diese Reihe nicht-funktionaler Anforderungen gibt es eine Vielzahl an Lösungsansätzen, die besonders im mobilen Bereich teilweise noch in der Erforschung stecken. Da dieses Thema eine sehr hohe Komplexität hat, gleichzeitig aber nicht den Schwerpunkt dieser Arbeit darstellt, wird dieser Bereich nicht weiter behandelt. Es bleibt festzustellen, dass Sicherheit dennoch ein absolut wichtiges Thema ist, welches bei einem produktiven System auf jeden Fall umgesetzt werden muss.

3.4. Ähnliche Projekte

Dieser Abschnitt gibt einen kurzen Überblick über existierende Projekte, die ein ähnliches Ziel verfolgen oder ähnliche Funktionalität wie der mucM zur Verfügung stellen.

3.4.1. Socialight

Socialight ist ein Social Networking Service, der für den mobilen Einsatz optimiert wurde. Das Besondere daran ist, dass man Nachrichten an einem speziellen Ort hinterlassen kann. Diese Nachrichten werden von den Benutzern erstellt. Die Verortung findet mittels Location-based Services per GPS und Zelleninformationen statt. Dabei wird auf eine bereits bestehende Infrastruktur aufgebaut. Das Basiselement, genannt „Sticky Note“, ist eine digitale Nachricht mit Informationen zur Location. Diese Sticky Note wird mit Ortsangaben versehen. Man kann Bilder, Sounds, Videos oder Texte über das Handy oder die Webseite hochladen und den Freunden zur Verfügung stellen. Der Einsatz von Socialight ist weltweit möglich. Noch befindet sich Socialight in einer Betaversion. Es gibt eine Inbox, die mit Informationen gefüllt ist, die sich auf den Ort beziehen, an dem man sich gerade befindet. Es soll auch die Möglichkeit bestehen, sich mit anderen sozialen Netzwerken zu verbinden. Das bedeutet, dass man seine „Sticky Notes“ über ein entsprechendes Plugin in verschiedenen sozialen Netzwerken einbinden kann. Anstelle von 1zu1 Kommunikation oder über ein Bulletin Board entsteht auf diese Art eine Placebased Conversation.

3.4.2. Dodgeball

Dodgeball ist in 22 Städten in Amerika verfügbar. Dodgeball ist ein soziales Netzwerk, in dem man Freunde hinzufügen kann und Freunde von Freunden sehen kann. Das Besondere ist, dass dies mit dem Handy geht. Über SMS mit eigener Positionangabe wird den Freunden mitgeteilt, wo man sich befindet. Auch wird der Benutzer darüber informiert, wenn sich Freunde von den eigenen Freunden innerhalb der nächsten zehn Blocks aufhalten. Weiterhin

kann man fünf Freunde auswählen, bei denen man informiert wird, wenn man sich in deren Nähe befindet. Durch eine „Shout-SMS“ kann man allen seinen Freunden bei Dodgeball eine Nachricht schicken. Als letztes Feature bietet das Programm an, dass man den Namen einer Location eingibt, per SMS an das System schickt und wenn das System diese kennt, antwortet es mit einer SMS, die die Adresse zu der Location enthält.

3.4.3. SLAM

SLAM ist ein Microsoft Research Projekt. SLAM ist Social Software zur Kommunikation mit Gruppen und Freunden mit mobilen Geräten. Es ermöglicht Echtzeitkommunikation, Location-Awareness und Photo-Sharing. Die Anwendung läuft eigentlich nur auf Windows Mobile Smartphones. Besitzt jemand kein solches Gerät, kann in eingeschränkter Version durch die Nutzung von SMS an der Kommunikation teilgenommen werden. Das Kernkonzept ist ein „Slam“ - eine Gruppe von Leuten, die miteinander Nachrichten und Fotos austauschen können. Die Idee dahinter ist Gruppenkommunikation. Jede Nachricht an eine Gruppe geht an alle Mitglieder dieser Gruppe. Mögliche Szenarien für die Nutzung von SLAM sind das Ausmachen von Treffen in Gruppen in Echtzeit ohne vorherige Planung, der sofortige Austausch von Fotos mit Freunden einer Gruppe und Gruppenkommunikation in dem Sinne, dass man an die Gruppe eine Frage stellt und sich irgendjemand meldet. Für die SMS-Variante fallen nur die Kosten der SMS an und für die Smartphone-Variante fallen die Kosten für den Datentransfer an. Es wird ausdrücklich darauf hingewiesen, dass eine Datenflatrate sehr zu empfehlen ist. Bei der Smartphone-Variante kann man sich zusätzlich auf einer Karte den Aufenthaltsort eines Freundes anzeigen lassen. Die Positionsfeststellung funktioniert über die Zelleninformationen und benötigt kein GPS. Um nicht jederzeit jedem den aktuellen Aufenthaltsort mitzuteilen, gibt es verschiedene Einstellungsmöglichkeiten. SLAM läuft als Service im Hintergrund, damit das Progamm jederzeit über Neuigkeiten informieren kann.

3.4.4. Fring

Fring ist ein mobiler Internetservice und Social Software eines Startup Unternehmens, welcher den Zugang zu und die Interaktion mit sozialen Netzwerken unterwegs ermöglicht. Des Weiteren ermöglicht Fring kostenlose Telefonate und Instant Messaging mit allen Fring, Skype, MSN Messenger, Google Talk, ICQ, SIP, Twitter, Yahoo und AIM Freunden. Ausreichend ist eine Internetverbindung des Mobiltelefons. Die Buddyliste kann bei Fring zusätzlich durch die Kontakte aus dem eigenen Adressbuch des Telefons erweitert werden. Mit Fring kann man mehrere Chats gleichzeitig führen, Dateien empfangen und versenden und auch Add-Ons lassen sich dafür entwickeln. Beispielsweise sind durch die Fring API schon ein Facebook Add-On, ein Gmail Notifier, Orkut Social Network Add-On und vieles mehr entwickelt worden. Voice over IP (VoIP) ermöglicht es zusätzlich günstigere Telefonate zu führen, als über den Netzprovider. Die Einwahl in WiFi Hotspots geschieht automatisch genau wie das Anmelden bei den verschiedenen Messengern. Fring benötigt keine spezielle Hardware, le-

diglich eine Internetverbindung muss möglich sein. Damit Fring auch im Hintergrund laufen kann, läuft dieses Programm ebenfalls als Service.

3.4.5. Vergleich der Projekte

Nachdem die Hauptfunktionalitäten der einzelnen Beispielanwendungen aufgezeigt wurden, wird deutlich, dass die Idee dieser Abschlussarbeit keine vollkommen neue Idee ist. Jedoch ist offensichtlich, dass die Schwerpunkte der einzelnen Projekte immer nur einen Teil der Lösung für dieses Szenario ausmachen. Entweder sind verschiedene Messenger Protokolle integriert oder es wird Wert auf Annotation der Realität gelegt und eine Form der Verortung findet statt. Ein Projekt ist sogar örtlich begrenzt. Ziel dieses Projektes ist, diese Projekte zusammenzuführen. Fring ist ein sehr guter Ansatz dazu. Hier fehlt eigentlich nur die Verortung. Selbst eine Schnittstelle für die „Programmer“ gibt es, um Plugins oder Add-ons zu entwickeln. Genau an dieser Stelle ist bei diesem Projekt ein deutlicher Unterschied zu sehen. Dieses Projekt ist Open Source und ermöglicht es damit jedem „Wizard“, an der Software selbst mitzuentwickeln. Hinter diesem Projekt steckt keine Firma, die damit Geld verdienen will. Es geht hierbei darum, für alle Menschen mit entsprechenden Geräten die Kommunikation mit Freunden möglichst hindernisfrei zur Verfügung stellen zu können. Die folgende Tabelle vergleicht die Projekte miteinander und verdeutlicht dabei noch einmal die Unterschiede.

	Socialight	**Dodgeball**	**SLAM**	**Fring**	**mucM**
SMS	-	x	(x)	x	x
Instant Messaging	-	-	(x)	x	x
Telefonie	-	-	-	x	x
Soziale Netzwerke	-	(x)	(x)	x	x
Verschiedene Protokolle	-	-	-	x	x
UGC (Annotation)	x	x	x	(x)	x
Verortung der Buddies	-	x	x	-	x
Weltweit nutzbar	x	-	x	x	x
Plattformunabhängigkeit	x	x	(x)	x	x
Verortung	x	x	x	-	x
Offene API	-	-	-	x	x
Kommerziell	x	x	-	x	-
Open Source	-	-	-	-	x

Tabelle 3.3.: Vergleich der Projekte

3.5. Fazit

Nachdem für dieses System eine umfassende Analyse sowie ein Vergleich existierender Lösungen stattgefunden hat, soll an dieser Stelle vor der Konzepterstellung ein kleines Fazit gezogen werden. Bisher gibt es noch keinen mobilen, ubiquitären und kontextabhängigen Messenger, mit den hier vorgestellten Möglichkeiten. Es gibt lediglich Lösungen, die Teilbereiche abdecken. Eine weitere Besonderheit, die es in diesem Zusammenhang noch nicht gibt, ist der Open Source Aspekt. Dieser soll es möglich machen, den Messenger nach den Bedürfnissen der Benutzer weiterzuentwickeln und für jeden frei einzusetzen. Der Vergleich verschiedener Entwicklungsplattformen ergibt ein klares Bild. J2ME ist nach eigenen Erfahrungen eine gute Grundlage, kann aber neueren Plattformen und Endgeräten nicht mehr gerecht werden. Das .NET Compact Framework bietet viele Möglichkeiten bei der Softwareentwicklung, ist aber genauso wie das iPhone sehr stark an den Hersteller gebunden und entspricht nicht den Anforderungen an ein Open Source Projekt. Des Weiteren ist eine der nicht funktionalen Anforderungen gewesen, dass die Programmiersprache Java sein soll. Dies trifft auch bei beiden Plattformen nicht zu. Dadurch, dass das iPhone die am stärksten ausgeprägte proprietäre Plattform ist, grenzt es sich in einem weiteren Punkt von der weiteren Betrachtung für die Konzepterstellung ab. Auf diese Weise zeigt sich Android mit seinen Eigenschaften, die in Kapitel 2.6 bereits beschrieben wurden, als gute Grundlage bei der Konzepterstellung.

4. Konzept

Nach der Analyse des Systems und der Identifikation der funktionalen sowie nicht funktionalen Anforderungen folgt darauf aufbauend die Konzepterstellung. Ziel der Konzepterstellung ist es, für den zu entwerfenden mucM eine Software-Architektur zu erstellen. Die Software-Architektur soll dabei die gestellten Anforderungen aus der Analyse erfüllen. Bei der Erstellung der Architektur werden verschiedene Sichtweisen herangezogen, um das Konzept von allen notwendigen Seiten zu beschreiben. Dazu wird sich an der Literatur von [Starke (2005)] und [Arlow und Neustadt (2005)] orientiert. Aufgrund der Tatsache, dass dieses eine Masterarbeit und keine Produkterstellung ist, musste der Unified Process an einigen Stellen an die besondere Situation angepasst werden. Während der iterative und inkrementelle Prozess in gekürzter Fassung erhalten bleibt, sowie die Aufteilung in die verschiedenen Phasen, wird die Realisierung nur prototypisch als „Proof of Concept“ stattfinden. Daher ist die Testphase auch stark gekürzt.

Aufteilung des Konzeptes: Im folgenden Abschnitt wird ein Überblick über die konzeptionellen Grundlagen gegeben. Daran anschließend wird die Software-Architektur erstellt. Dabei wird eine Unterteilung in ein Fachkonzept und ein technisches Konzept vorgenommen. Durch diese Trennung werden eine geringe Kopplung und eine hohe Kohäsion erreicht. In einem ersten Schritt wird ein Fachkonzept entwickelt. Von diesem ausgehend wird in einem weiteren Schritt das technische Konzept entwickelt. Diese Unterteilung wird im Weiteren noch verfeinert.

4.1. Konzeptionelle Grundlagen

Nachfolgend wird beschrieben, welche Grundlagen aufbauend auf der Analyse notwendig sind, um ein erfolgreiches Konzept erstellen zu können. Um der Vielseitigkeit der funktionalen Anforderungen gerecht werden zu können, soll auf bereits existierende Systeme zurückgegriffen werden. Weiterhin haben die nicht funktionalen Anforderungen gezeigt, dass besonders für ein Open Source Projekt eine flexible Architektur mit guter Kapselung der Systeme und eine Abstraktion von der Technologie unbedingt notwendig sind (siehe Abschnitt 3.3.1). Zusätzlich wird dieses durch die Anforderungen an die Änderbarkeit, Wartbarkeit und Testbarkeit unterstützt (siehe Abschnitt 3.3.3).

4.1.1. Komponentenmodell

Es wird großer Wert auf die Erstellung eines Komponentenmodells gelegt, da Komponenten klar definierte Schnittstellen nach außen anbieten, die Implementierung verstecken und damit die Wiederverwendbarkeit und Austauschbarkeit erhöhen. Ein zusätzlicher Vorteil von häufig eingesetzten Komponenten ist, dass diese in der Regel stabiler sind. Daher sollen die einzelnen Systeme eigenständige Komponenten sein, wobei diese Komponenten wiederum selbst aus Komponenten oder aus Modulen bestehen können. Unter Modulen sind elementare Komponenten zu verstehen [Siedersleben (2004)].

4.1.2. Robustheit und Fehlertoleranz

Weiterhin wird ausgehend von den Anforderungen und dem mobilen Kontext viel Wert auf hohe Robustheit und Fehlertoleranz gelegt. Hierbei geht es insbesondere darum, dass bestimmte Teile der Anwendung temporär nicht zur Verfügung stehen und sich diese Probleme auch nicht direkt lösen lassen. Das muss bei der Konzepterstellung berücksichtigt werden. Es müssen also sogenannte Fallback-Lösungen entwickelt werden, die genutzt werden können, wenn ein Teil ausfällt. Dazu ist anzumerken, dass der Fehler eventuell nicht vollständig behoben werden kann, aber zumindest seine negativen Auswirkungen reduziert werden können. Im Folgenden werden dazu drei Hauptfehlerquellen aufgeführt.

Verbindungsabbruch: Der Verbindungsabbruch ist ein Beispiel für einen Fehler, der sich nicht verhindern lässt. Allerdings kann man versuchen, den Schaden für den Benutzer möglichst gering zu halten. Möchte der Nutzer zum Beispiel eine Nachricht verschicken und in dem Moment bricht die Verbindung ab, so kann das Programm anstelle davon, dass die Nachricht verloren geht, diese Nachricht zwischenspeichern bis wieder eine Verbindung besteht und dem Benutzer dann ermöglichen, die eingegebene Nachricht erneut zu verschicken. Ein weiteres Beispiel ist das Übertragen von Dateien. Wenn eine Datei nach einem gewissen Prozentsatz nicht mehr weiter übertragen werden kann, weil die Verbindung unterbrochen wurde, könnte sich der zuletzt bestätigte Zustand gemerkt werden und bei erneuter Verbindung dort fortgesetzt werden. Daher ist die Umsetzung einer partiellen Offline-Funktionalität sinnvoll. Möglichkeiten dafür sind ein Cache oder das Zwischenspeichern in einer Datenbank, oder das Nutzen eines Cache-Proxies.

Verlust des GPS-Satellitensignals: Auch der Verlust des Satellitensignals lässt sich in Gebäuden zum Beispiel nicht verhindern. Eine robuste Anwendung kann den Fehler aber in vielen Fällen stark einschränken. So kann beispielsweise die letzte bekannte Position gespeichert werden und von dieser ausgehend die örtlichen Dienste weiter zur Verfügung gestellt werden. Alternativ kann auch versucht werden auf weniger genaue Positionsdaten zurückzugreifen. So könnte anstelle von GPS-Daten die Identifizierung per Cell-ID, zum Beispiel beim U-Bahn fahren, einen deutlich genaueren Wert liefern als die zuletzt bekannte GPS-Position.

Das lässt sich auch kombinieren. Bei Verlust des Satellitensignals wird sich die letzte Position gemerkt. Allerdings wird diese nur solange beibehalten, bis die Cell-ID eine deutlich andere Position feststellt. Dann wird diese Position zur aktuellen Position.

Ausfall externer Systeme: Wenn externe Systeme ausfallen, hat der Anwendungsentwickler keinen Einfluss darauf. Dieser kann sich entweder auf die Zuverlässigkeit der externen Systeme verlassen oder, wenn es vergleichbare Alternativsysteme gibt, diese als Fallback-Lösung einbinden. Ein Beispiel dafür ist der Austausch von Google Maps mit Yahoo Maps, falls eines der Systeme überlastet ist. Ein weiteres Beispiel ist die parallele Nutzung von Geoinformationen aus verschiedenen Systemen. Dazu müssen allgemeingültige Schnittstellen bereitgestellt werden und die spezifischen Schnittstellen der Systeme müssen über einen Adapter darauf angepasst werden.

4.2. Fachliche Architektur

Bei einer Architektur gibt es immer verschiedene Sichtweisen, die beschrieben werden müssen. Auf der einen Seite wird ein Überblick über die fachlichen Komponenten gezeigt und diese im Anschluss detailliert beschrieben. Daraufhin folgt eine Beschreibung der fachlichen Architektur bis auf Klassenebene. Auf der anderen Seite wird die fachliche Verteilung gezeigt, an die sich die Darstellung des dynamischen Ablaufs der Komponenten anschließt. Zuletzt werden die notwendigen Schnittstellen definiert.

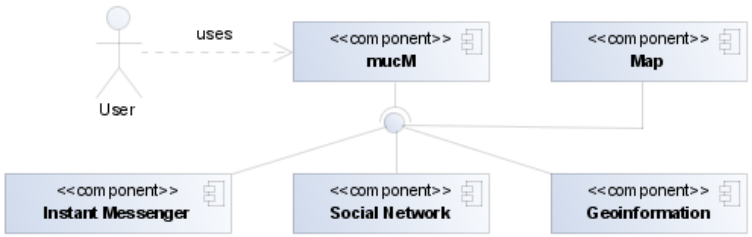

Abbildung 4.1.: Übersicht der Architekturbausteine des Gesamtsystems

Das Gesamtsystem wird aus fachlicher Sicht in Abbildung 4.1 dargestellt, wobei diese Abbildung als konzeptionelle Übersicht dient. Der Einstiegspunkt ist der `User`, der das System benutzt. Dabei besteht das System aus mehreren einzelnen Komponenten (externen Systemen), die als Ganzes die Funktionalität des Messengers bereitstellen. Die Komponente, mit der der `User` direkt in Kontakt tritt, ist der `mucM`. Dieser stellt die gesamten Messengerfunktionalitäten zur Verfügung. Um diese komplexe Aufgabe erledigen zu können, bedient sich diese Komponente weiterer Komponenten. Diese Komponenten sind `Instant Messenger`, `Social Network`, `Geoinformation` und `Map`. Die genaue Bedeutung der einzelnen

Komponenten wird im Folgenden näher beschrieben. Der Schwerpunkt bei der Entwicklung des Konzepts liegt auf der Hauptkomponente `mucM`, da diese selbst entwickelt wird. Diese Komponente wird in weitere Komponenten aufgeteilt. Die anderen Komponenten müssen zur Bereitstellung der Gesamtfunktionalität integriert werden.

4.2.1. Übersicht der fachlichen Komponenten

Die zuvor beschriebene Hauptkomponente `mucM` wird in vier weitere Komponenten aufgeteilt. Das verdeutlicht die folgende Abbildung 4.2.

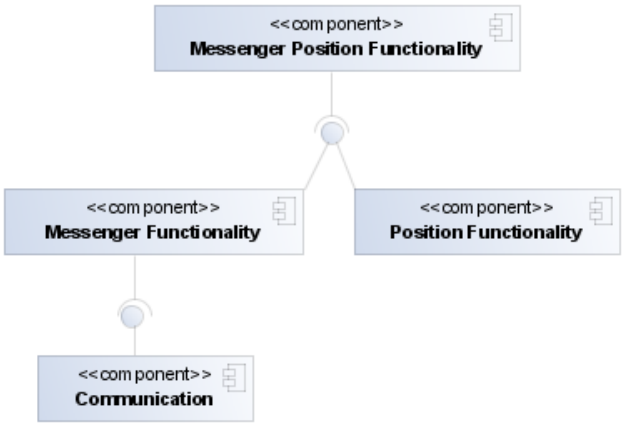

Abbildung 4.2.: Bausteinsicht der Hauptkomponente mucM

Während die Einzelheiten dieser vier Komponenten nachfolgend genau erläutert werden ist an dieser Stelle folgendes zu sagen: Bei der Unterteilung in die vier Komponenten bilden die zwei Komponenten `Messenger Functionality` und `Position Functionality` die Basis. Aus der Komposition dieser beiden Komponenten entsteht die Komponente `Messenger Position Functionality` mit erweiterter Funktionalität. Um die Komplexität gering zu halten, gibt es eine extra Komponente `Communication`, die für die Kommunikation, mit beispielsweise den Messengern und sozialen Netzwerken, zuständig ist. Daher wird diese Komponente von der Komponente `Messenger Functionality` genutzt. Somit entsteht ein modularer Aufbau, der die Wiederverwendbarkeit erhöht und gleichzeitig die Komplexität gering hält.

4.2.2. Details der fachlichen Komponenten

An dieser Stelle werden die auf zuvor sehr hoher Ebene beschriebenen Komponenten detaillierter dargestellt.

Messenger Functionality: Die Komponente `Messenger Functionality` besteht ihrererseits wiederum aus mehreren Modulen. Dies zeigt die folgende Abbildung 4.3.

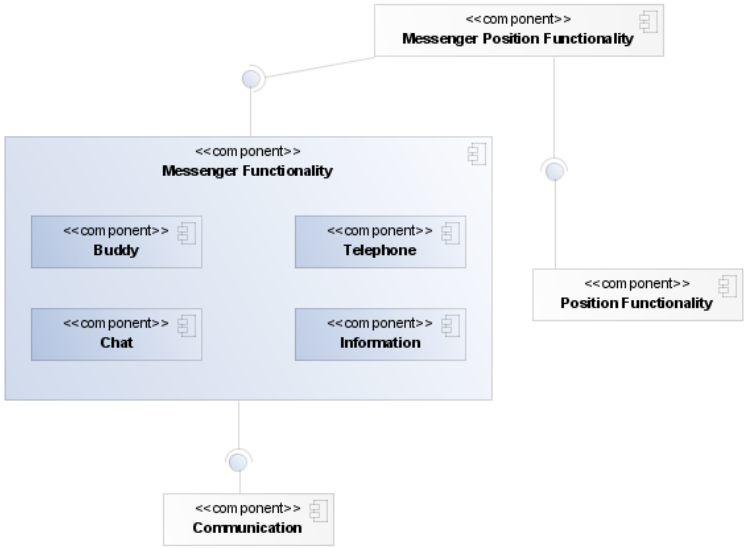

Abbildung 4.3.: Messenger Funktionalität der mucM-Komponente

Diese Komponente stellt die für diesen Messenger wichtigen Grundfunktionalitäten zur Verfügung. Das ist zum einen das `Buddy`-Modul, welches dafür zuständig ist Kontakte und das eigene Profil zu verwalten. Zum anderen gibt es zwei Module, die für die Kommunikation mit den Buddies zuständig sind. Dabei stellt das `Telephone`-Modul die Schnittstelle zwischen dem `User` und den Buddies bereit, um in telefonischen Kontakt treten zu können, während es die Aufgabe des `Chat`-Moduls ist durch eine Schnittstelle zum `User` Nachrichten mit allen Buddies austauschen zu können. Das `Information`-Modul hat die Aufgabe von den Benutzern erstellte Informationen für die Community bereitzustellen.

Communication: Für die Kommunikation, die der mucM ermöglichen soll, ist diese Komponente aus der Komponente `Messenger Functionality` separiert worden. Dadurch wird die Modularität und Flexibilität für Erweiterungen, die in diesem Bereich häufig auftreten, erhalten. Diese Komponente besteht ebenfalls aus weiteren Modulen, die die Abbildung 4.4 zeigt. Auf diese Weise wird die Kommunikation durch das Modul gekapselt. Das `Tele Communication`-Modul ist für die Telefonie zuständig und ermöglicht es mit den Buddies auf beliebige Art zu telefonieren. Das Modul `SN Communication` ist für die Kommunikation mit sozialen Netzwerken aller Art zuständig. Die gleiche Aufgabe übernimmt das Modul `IM Communication` für die Kommunikation mit Instant Messengern.

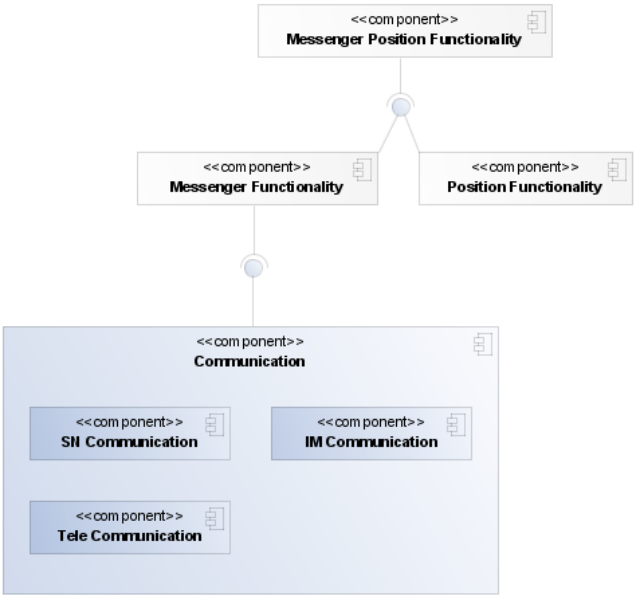

Abbildung 4.4.: Kommunikations-Komponente des mucM

Position Functionality: Die Erweiterung eines herkömmlichen Messengers zu einem kontext- abhängigen Messenger spiegelt diese Komponente wider. Die Komponente `Position Functionality` besteht aus zwei Modulen. Dieses zeigt Abbildung 4.5.

Die Komponente besteht zum einen aus einem `Position`-Modul. Durch die Abstraktion dieses Moduls ist es offen für verschiedene Positionsdaten. Dadurch wird es zum Beispiel ermöglicht, die Positionsdaten in Form von Koordinaten dem System zur Verfügung zu stellen. Auf der Möglichkeit Positionen feststellen zu können baut das `Map`-Modul auf. Dieses kann Positionen zur Anzeige bringen. Dieses Modul wurde ebenfalls abstrahiert, um beliebige Maps einbinden zu können.

Messenger Position Functionality: Die `Messenger Position Functionality`-Komponente entsteht aus der Komposition der zwei zuvor beschriebenen Basismodule. Das daraus entstandene neue Modul zeigt Abbildung 4.6.
Die neu entstandenen Module sind `BuddyPositition` und `Position`. Zum einen wird das `Buddy`-Modul durch die `Position Functionality`-Komponente erweitert. Zum anderen entsteht aus dem `Information`-Modul ebenfalls durch die Erweiterung mit der `Position-Functionality`-Komponente das neue `Position`-Modul. Beide neu-

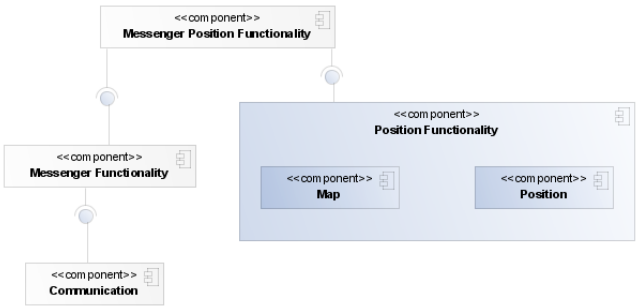

Abbildung 4.5.: Position Funktionalität der mucM-Komponente

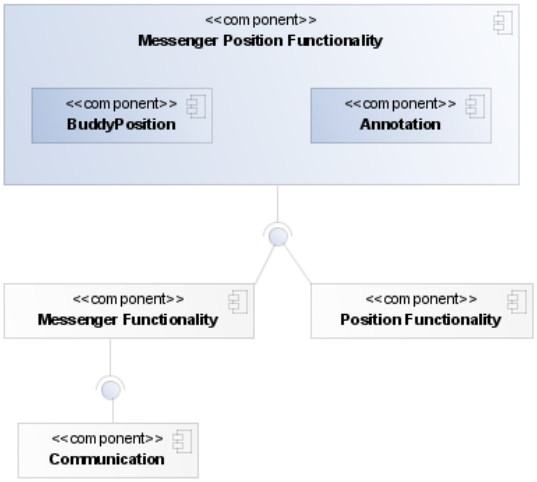

Abbildung 4.6.: Messenger-Position Funktionalität der mucM-Komponente

en Module ermöglichen auf diese Art eine Erweiterung der Standardmodule `Buddy` und `Information` um die Eigenschaft der Verortung.

4.2.3. Innenansicht der fachlichen Komponenten

Das Diagramm 4.7[3] zeigt die Innenansicht der zuvor beschriebenen fachlichen Komponente, der `mucM`-Komponente. Dafür wird zur Darstellung ein Klassendiagramm benutzt. Die darin enthaltenen fachlichen Klassen und Methoden veranschaulichen den strukturellen Aufbau. Bei genauer Betrachtung ist ersichtlich, dass sich nicht exakt an die Definition von Komponenten gehalten wurde. An dieser Stelle wurde die Definition etwas aufgeweicht. Es findet eine engere Kopplung mit dem Ziel statt, dass die „Komponenten“ spezieller aufeinander abgestimmt werden können und somit eine bessere Performance erreicht werden kann.
Die in dem Diagramm dargestellten fachlichen Methoden spiegeln eine Grundfunktionalität wider, die im Sinne der Anforderungen und des Konzeptes erweiterbar ist. Nachfolgend werden der Aufbau und die komplexen Methoden näher erläutert.

User: Die `User`-Klasse ist eine Spezifizierung der `Buddy`-Klasse und stellt damit besondere Funktionalitäten zur Verfügung, die über die einfachen Funktionalitäten des `Buddy` hinausgehen. Der `User` ist der `Buddy`, der am mucM angemeldet ist. Er hat damit zwei Erweiterungen. Zum einen kann der `User` seine persönlichen Daten ändern und zum anderen kann er einen bestimmten Status einstellen. Der Status ist bezugnehmend auf die Anforderungen besonders wichtig, da dem `User` eine gewisse Privatsphäre ermöglicht werden muss. `ProvideUserState` ermöglicht es, neben bereits vordefinierten Profilen, auch eigene Profile zu erstellen. Das ist eine sehr komplexe Angelegenheit. Es können kontextabhängige Profile erstellt werden. So ist zum Beispiel das Profil „AtWork“ an die Position des Arbeitsplatzes und an einen gewissen Radius gebunden und zeigt an, dass der `User` gerade arbeitet und nur in dringenden Notfällen gestört werden möchte. Das Profil „Invisible“ zum Beispiel zeigt nicht an wo der `User` sich gerade aufhält und kann manuell gewählt werden. Das Profil „Weekend“ stellt sich automatisch in der Zeit von Freitagabend bis Sonntagabend ein und zeigt den Buddies der Kategorie „Freunde“ die derzeitige Position und sein Vorhaben am Wochenende an.
Des Weiteren hat der `User` eine `BuddyList`, die alle seine Buddies enthält und ihm unter Anderem ermöglicht neue Buddies hinzuzufügen, alte zu löschen oder sie in eine andere Gruppe zu veschieben. Sobald ein `Buddy` aus der `BuddyList` seinen Zustand wechselt oder der `User` etwas ändert, wird eine Aktualisierung durchgeführt.

Account: Die `Account`-Klasse stellt die Schnittstelle der `Communication`-Komponente dar. Dabei muss jede `Messenger`-Komponente mindestens einen `Account`

[3] In der Messenger-Komponente sind zuvor Chat und Telefone aufgeführt worden. An dieser Stelle werden sie der Übersichtlichkeit wegen nicht angezeigt, da in dieser Komponente lediglich die Schnittstelle zum User geschaffen wird.

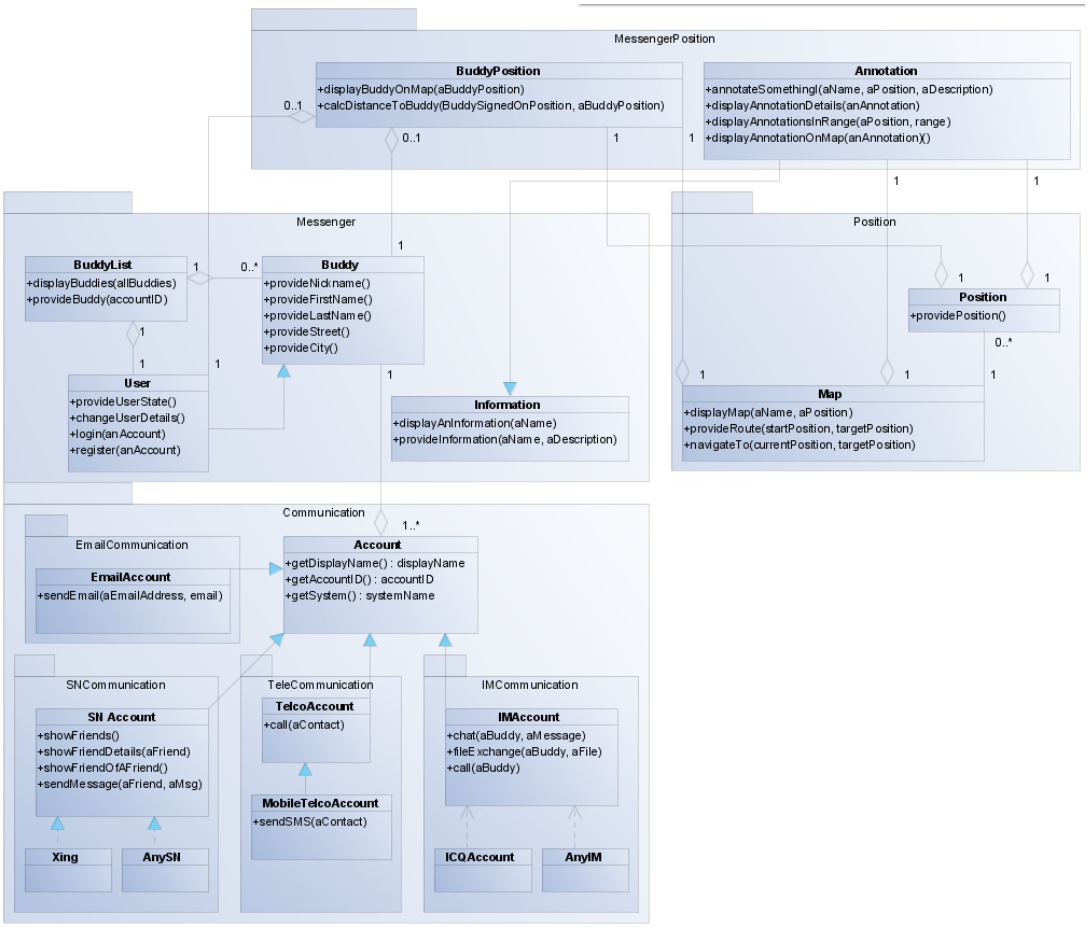

Abbildung 4.7.: Innenansicht zu den fachlichen Komponenten des mucM

besitzen, um die Kommunikation mit Anderen zu ermöglichen. Ein `Account` ist eindeutig definiert durch einen Anzeigenamen, eine ID und den Systemnamen. Der Anzeigename dient dem `User` lediglich zur Unterscheidung, ob es beispielsweise die private oder geschäftliche Telefonnummer ist. Die ID ist pro System eindeutig und identifiziert zusammen mit dem Systemnamen den `Account`. Ein Beispiel dafür ist eine Nummer (81643357) mit dem zugehörigen System ICQ. Entsprechend des `Account`-Typen bietet jeder `Account` eine spezielle Art von Methoden an. Beispiele für verschiedene `Account`-Typen sind E-Mail oder Instant Messaging.

Annotation: Die `Annotation`-Klasse ist die Erweiterung der `Information`-Klasse um die Verortung. Das erweiterbare Konzept sieht es vor, dass man alles mit einer Position annotieren, also mit Zusatzinformation versehen kann. Neben einer Position hat eine `Annotation` auch einen Namen und eine Beschreibung. Diese Beschreibung kann sehr vielfältig sein. Es kann von rein schriftlicher Beschreibung über Fotos hin zu Audiodateien oder Videos gehen. Diese Annotationen werden dann allen bereitgestellt. Während die Anzeige einer `Annotation` auf der Karte oder aller Annotationen in der Nähe relativ einfache Methoden sind, ist die Methode `displayAnnotationDetails` komplexer. Abhängig von dem gewählten Medium zur Beschreibung und vom Kontext des `User` sollen die Informationen entsprechend bereitgestellt werden. Ist zum Beispiel der Akkustand des Endgerätes des `User` sehr niedrig, soll darauf verzichtet werden ein Video abzuspielen es sei denn, der `User` wünscht dieses ausdrücklich.

4.2.4. Fachlicher Ablauf

Im Folgenden wird mit Hilfe von Sequenzdiagrammen fachlich das dynamische Verhalten modelliert. Bei der Modellierung des Ablaufes ist zu sagen, dass es sich jeweils um einen exemplarischen Ablauf der Komponenten und Module dieses Systems handelt. Aus Gründen der Übersichtlichkeitsbewahrung wird sich auf das Wesentliche konzentriert und von der Modellierung von Ausnahmen und Fehlern abgesehen. Bei der Beschreibung der folgenden Abläufe gilt, dass der `User` bereits am Messenger (mucM) angemeldet ist und damit Zugriff auf die einzelnen Komponenten hat. Der mucM läuft die gesamte Zeit, während die restlichen Komponenten nur temporär benutzt werden.

4.2.4.1. Ablauf der Geoinformation-Komponente

Das folgende Diagramm 4.8 zeigt den Ablauf der `Geoinformation`-Komponente und den zugehörigen Modulen.
Es gibt vier Abläufe, die der `User` mit dieser Komponente ausführen kann. Möchte der `User` die Annotationen in seiner Nähe anschauen (1), muss der `mucM` seine Position und den Umkreis, in dem er die Informationen erhalten möchte, kennen. Daraufhin wird eine Verbindung zur `Geoinformation`-Komponente hergestellt und die notwendigen Daten übergeben (2). Die Ergebnisse werden an den `mucM` zurückgegeben (3) und dem `User` zur Verfügung gestellt (4). Um detaillierte Informationen zu einer Annotation zu bekommen, wählt der `User` eine Annotation aus (5). Es wird erneut eine Verbindung zum `Geoinformation`-System aufgebaut und die notwendigen Informationen abgefragt (6). Diese Informationen werden über den `mucM` (7) an den `User` zurückgegeben (8). Soll die Annotation auf einer Karte angezeigt werden (9), muss diese an die `Map`-Komponente inklusive ihrer Position übergeben werden (10). Der Kartenausschnitt mit der Annotation wird dann zurückgegeben und dem Benutzer angezeigt (11, 12). Soll etwas annotiert werden, muss der `User` eine Annotation, bestehend aus dem Namen, einer Position und der Annotationsbeschreibung erstellen (13). Diese Daten

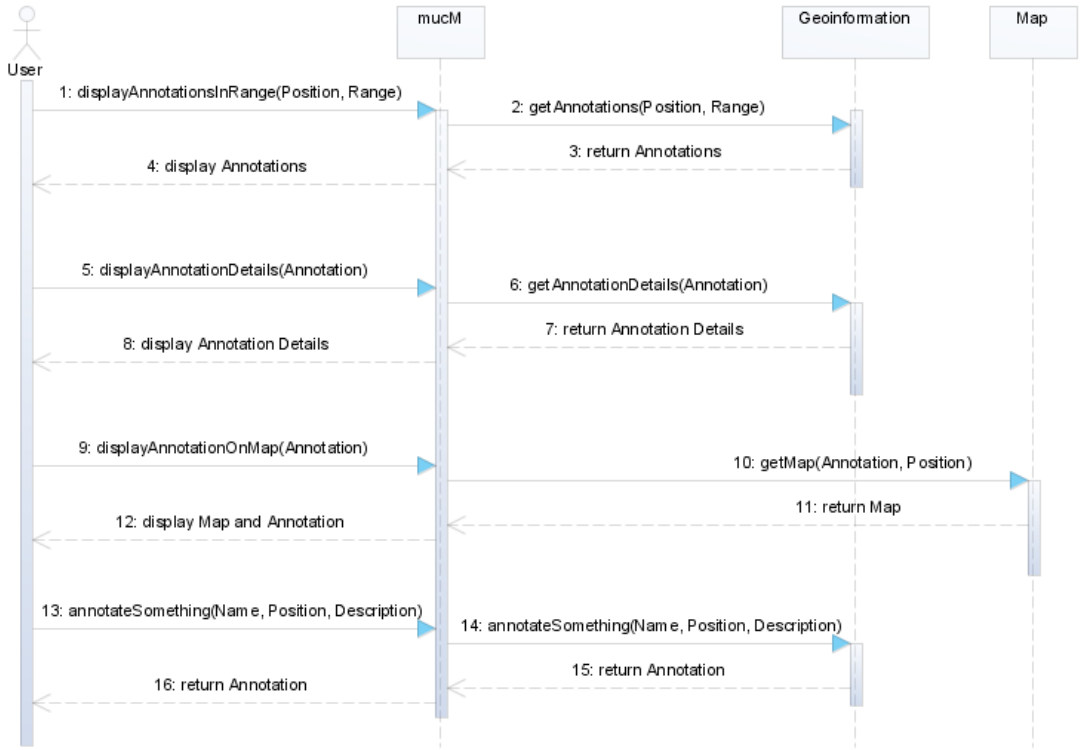

Abbildung 4.8.: Fachliches Sequenzdiagramm der Geoinformation-Komponente mit den zugehörigen Modulen

werden dann an das `Geoinformation`-System übertragen (14), um sie anderen zur Verfügung zu stellen. Die Annotation wird über den `mucM` an den `User` zurückgegeben (15, 16).

4.2.4.2. Ablauf der Messenger-Komponente

Das Diagramm 4.9 zeigt den Ablauf der `Messenger`-Komponente und den zugehörigen Modulen.
Die `Messenger`-Komponente besteht im Wesentlichen aus fünf Abläufen, die in dem Diagramm dargestellt sind. Dabei besteht eine ständige Verbindung zum `IM` und darüber zu seinen Buddies. Möchte der `User` seine Buddies anzeigen lassen (1), teilt der `mucM` dieses dem `IM` mit (2). Dieser übergibt die aktuellen Daten seiner Buddies wieder dem `mucM` (3), welcher wiederum die übergebenen Daten zur Anzeige bringt (4). Soll einer der Buddies auf der Karte dargestellt werden (5), stellt der `mucM` eine Verbindung zur `Map`-Komponente

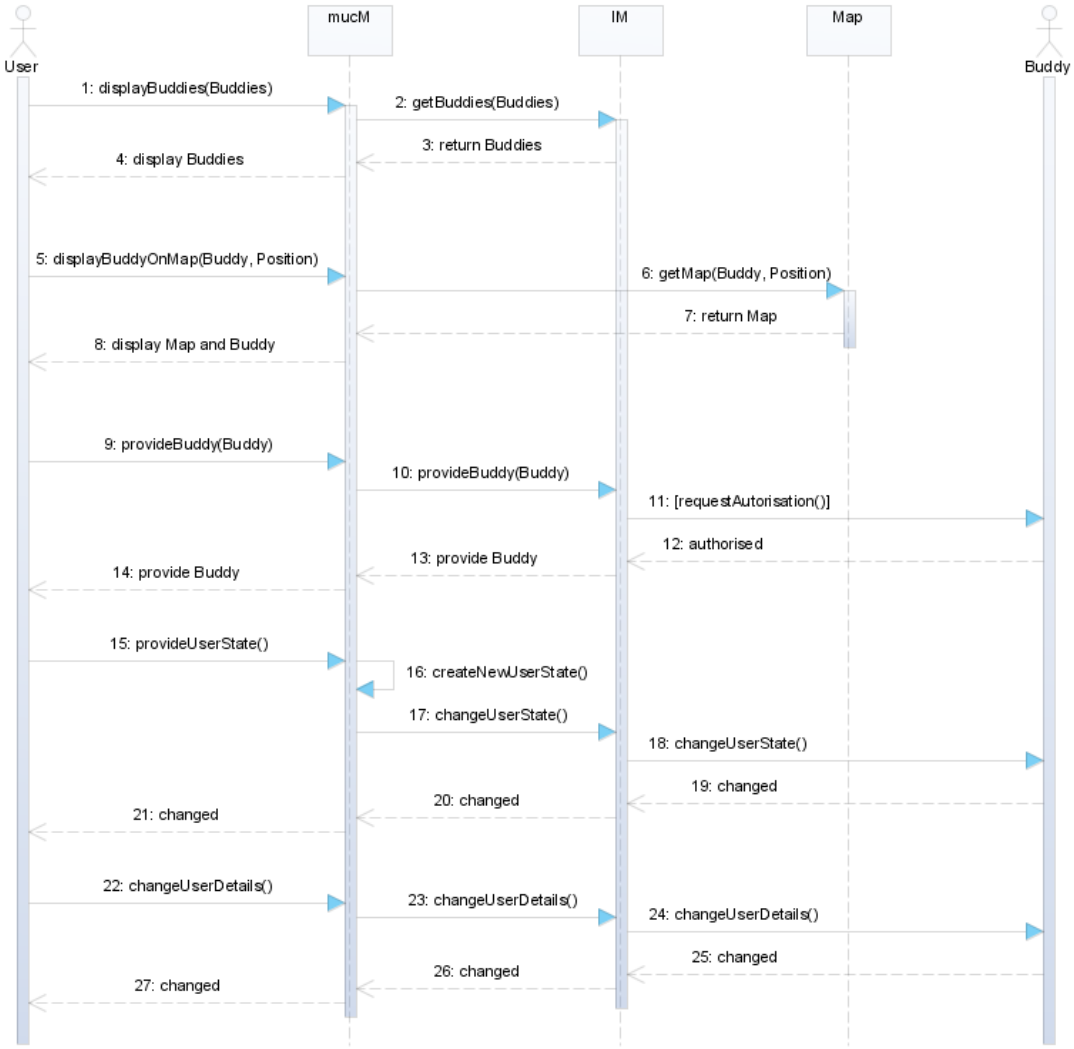

Abbildung 4.9.: Fachliches Sequenzdiagramm der Messenger-Komponente mit den zugehörigen Modulen

her und übergibt dieser den ausgewählten Buddy und seine Position (6). Die entsprechende Karte wird wieder an den mucM zurückgegeben (7) und zur Anzeige gebracht (8). provideBuddy(Buddy) (9) ermöglicht es, einen Buddy hinzuzufügen oder zu löschen. Dazu wendet sich der mucM an den IM und teilt diesem entweder mit, dass ein Buddy aus der Liste gelöscht wurde, oder dass ein Buddy hinzugefügt werden soll (10). Im zweiten Fall muss der IM den Buddy noch um Erlaubnis bitten (11). Der Buddy gibt eine Bestätigung oder eine Ablehnung an den IM zurück (12). Der IM teilt dem mucM das Ergebnis mit (13) und leitet dieses an den User weiter (14). provideUserState() (15) ermöglicht es dem User, einen neuen Status zu erstellen oder einfach nur den Status zu wechseln. Entsprechend kann beim mucM ein neuer Status erstellt werden (16). Auf jeden Fall muss die Statusänderung über den IM (17) den Buddies (18) mitgeteilt werden. Die erfolgreiche Statusänderung wird an den IM (19), den mucM (20) und den User (21) weitergeleitet. Ändert der User seine persönlichen Angaben (22), wird dies über den mucM (23) dem IM mitgeteilt. Dieser leitet die Änderungen an die Buddies (24) weiter. Diese Änderungen werden an den einzelnen Komponenten quittiert (25, 26, 27).

4.2.4.3. Ablauf der Communication-Komponente

Das Diagramm 4.10 zeigt den Ablauf der Communication-Komponente und den zugehörigen Modulen. Dieser Ablauf ist der Komplexeste, da die meisten Komponenten eingebunden werden müssen und eine ähnliche Funktionalität über verschiedene Komponenten stattfinden muss.
Möchte der User mit einem Buddy chatten (1), wird die Nachricht vom mucM an den IM unter Angabe des Buddy weitergeleitet (2). Dieser leitet die Nachricht dann weiter an den Buddy (3). Das Übertragen der Nachricht wird entsprechend von den beteiligten Komponenten quittiert (4, 5, 6). Vergleichbar ist der Ablauf auch mit dem Austausch von Dateien und dem Anrufen eines Buddies, daher wird dieses nicht extra aufgeführt. Schickt der User eine E-Mail an einen Buddy, geschieht dies unter Angabe der E-Mail-Adresse (7). Der mucM leitet die Nachricht an die Email-Komponente weiter (8), die diese Nachricht dem Buddy zustellt (9). Möchte der User seine Freunde aus den sozialen Netzwerken sehen (10), teilt der mucM das dem SN mit (11) und diese Komponente gibt die Freunde über den mucM (12) an den User (13) zurück. Analog dazu ist der Ablauf von showFriendOfAFriend und showFriendDetails. Einem Buddy aus den sozialen Netzwerken eine Nachricht zu schicken (14), hinterlegt die Nachricht von der mucM-Komponente bei der SN-Komponente (15). Einen Buddy über die Telefonfunktion anzurufen (16) geht über den mucM an die Telephone-Komponente (17) an den Buddy (18). Der Anruf wird beantwortet und dann geht es in umgekehrter Reihenfolge zurück bis zum User (19, 20, 21). Vergleichbar mit dem Anruf ist das Schicken einer SMS (22, 23, 24). Der Rückweg besteht allerdings lediglich aus einer Quittierung der Übermittlung (25, 26, 27).
Abschließend ist bei dieser Ablaufbeschreibung zu sagen, dass für die verschiedenen Kontakt-

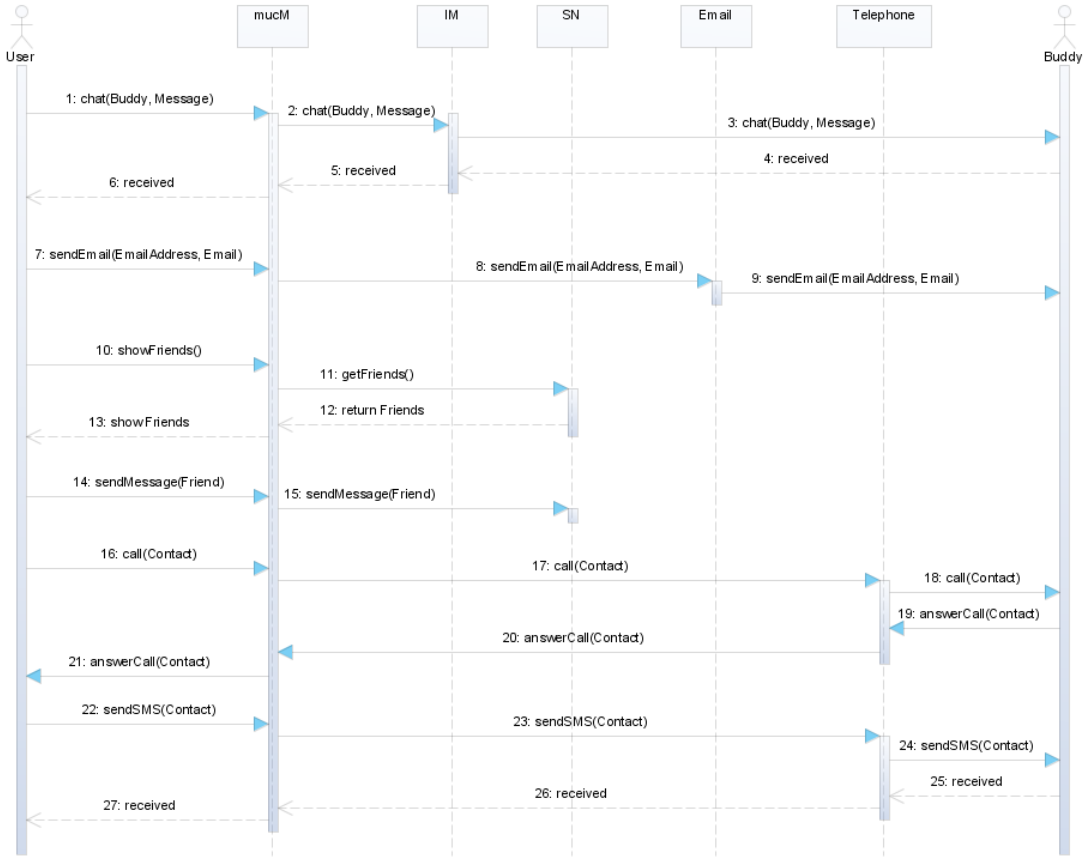

Abbildung 4.10.: Fachliches Sequenzdiagramm der Communication-Komponente mit den zugehörigen Modulen

aufnahmen verschiedene `Account`-Daten notwendig sind. Dies soll durch `Buddy`, `Friend` und `Contact` verdeutlicht werden.

4.2.4.4. Ablauf der Map-Komponente

Das Diagramm 4.11 zeigt den Ablauf der `Map`-Komponente und der zugehörigen Module. Die `Map`-Komponente wurde bisher von den anderen Komponenten mehrfach benutzt. An dieser Stelle wird der Ablauf der reinen `Map`-Komponente beschrieben. Jedes Mal, wenn etwas von dem `Map` abgefragt wird, muss erneut eine Verbindung aufgebaut werden.
Eine Grundfunktionalität der `Map` ist es, dem `User` eine bestimmte Position auf einer Karte anzuzeigen. Der `User` fragt die `Map` danach (1) und diese liefert ein entsprechendes Ergebnis (2). Möchte der `User` eine Strecke zwischen zwei Punkten erfahren, wird die `Map` mit zwei

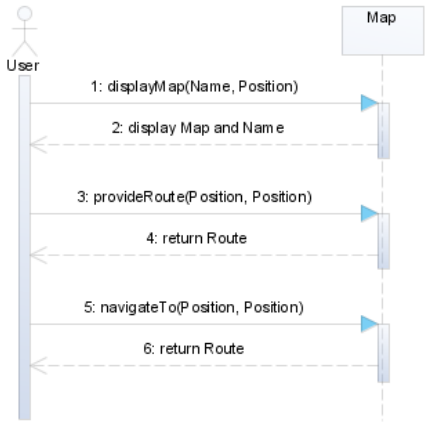

Abbildung 4.11.: Fachliches Sequenzdiagramm der Map-Komponente mit den zugehörigen Modulen

Positionen danach befragt (3). Entsprechend liefert die `Map` eine Strecke zurück (4). Soll zu einer bestimmten Postion navigiert werden, muss nur das Ziel angegeben werden. Der Startpunkt wird intern durch die Position des `User` gesetzt (5). Die `Map` liefert die entsprechende Strecke (6).

4.2.5. Fachliche Verteilung

Das Diagramm 4.12 zeigt die Verteilung der zuvor identifizierten und beschriebenen fachlichen Komponenten und Module. Dabei wird eine Unterscheidung in zwei Hauptbereiche vorgenommen. Es gibt einerseits das eigene System und andererseits die externen Systeme. Bei den externen Systemen handelt es sich um Lösungen, die zur Nutzung eingebunden werden. Das System selbst ist wieder in zwei Bereiche unterteilt. Der erste ist das Endgerät, welches die Schnittstelle zum Benutzer darstellt. Dieser wird als Client bezeichnet. Der zweite ist der Server, der zentrale Aufgaben übernimmt.
Aufgrund der eingeschränkten Ressourcen und Betriebsdauer soll der Client so wenig wie möglich belastet werden. Daher besteht dieser nur aus einer grafischen und der `UserPosition`-Komponente. Die `GUI`-Komponente wird auf dem Client benötigt, damit der Benutzer mit den einzelnen Komponenten des mucM interagieren kann. Die `UserPosition` -Komponente muss auf dem Client sein, da nur das mobile Endgerät selbst die Position des Benutzers kennt und zur Verfügung stellen kann. Der Client kommuniziert ausschließlich mit dem Server. Das hat den Vorteil, dass der Client nicht ständig Verbindungen zu verschiedenen Systemen aufbauen und gegebenenfalls aufrecht erhalten muss. An folgendem Beispiel lässt sich das verdeutlichen: Wenn der Client in seiner Buddyliste Buddies aus vier verschiedenen Messengern hat, muss zu all diesen Messengern ei-

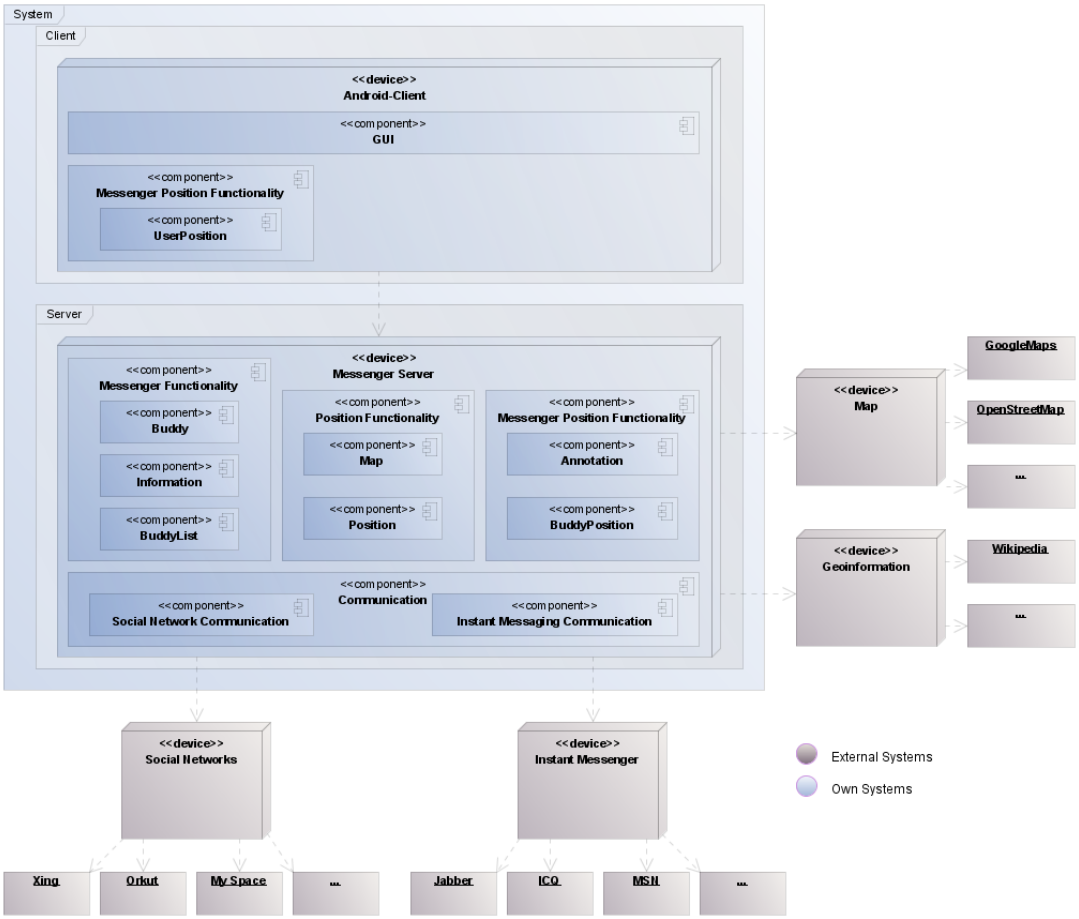

Abbildung 4.12.: Fachliches Verteilungsdiagramm des Gesamtsystems

ne Verbindung aufrechterhalten werden. Die Aufgabe kann aber auch der Server übernehmen und den Client damit deutlich entlasten. Auf diese Weise muss nur eine Verbindung aufrecht erhalten werden und der Server sorgt für eine geringere Datenlast. Der Server ist weiterhin als zentrale Instanz unerlässlich, da die von der Community erstellten Daten der Community auch wieder zur Verfügung gestellt werden müssen. Auf dem Server befinden sich die Komponenten `Messenger Functionality`, `Position Functionality`, `Messenger Position Functionality` und `Communication`. Die genauen Aufgaben dieser Komponenten wurden bereits in diesem Kapitel beschrieben. Über den Server des eigenen Systems wird auf die externen Systeme zugegriffen. Durch diese Aufteilung wird weiterhin erreicht, dass der Server die meiste Logik ausführen kann und nur noch die notwendigen Nutzdaten übertragen muss. Somit wird auch die Rechenlast auf dem Client verringert.

Ein Beispiel dafür ist die Berechnung einer Route. Bei dieser Verteilung muss der Client dem Server lediglich den Start- und Zielpunkt mitteilen. Der Server übernimmt die komplexe und rechenintensive Aufgabe der Berechnung der Route. Wenn das geschehen ist, gibt er lediglich die notwendigen Kartenausschnitte mit der fertigen Route an den Client zurück. Die externen Systeme sind die Instantiierungen der auf dem eigenen Server liegenden gekapselten Komponenten. Das funktioniert allerdings nur, weil diese externen Systeme offene Schnittstellen bereitstellen, um diese zu integrieren. Die externen Systeme sind in vier Kategorien entsprechend der Komponenten auf dem eigenen Server unterteilt: Social Networks, Instant Messenger, Geoinformation und Map.

4.2.6. Fachliche Schnittstellen der Komponenten

Werden externe Systeme (Komponenten) in ein System integriert, werden diese als Blackbox dargestellt. Es sind lediglich die benötigten und angebotenen Schnittstellen bekannt, wobei eine Schnittstelle aus mehreren Operationen bestehen kann. Nachfolgend werden die Schnittstellen der einzelnen Komponenten in Abbildung 4.13 dargestellt und beschrieben.

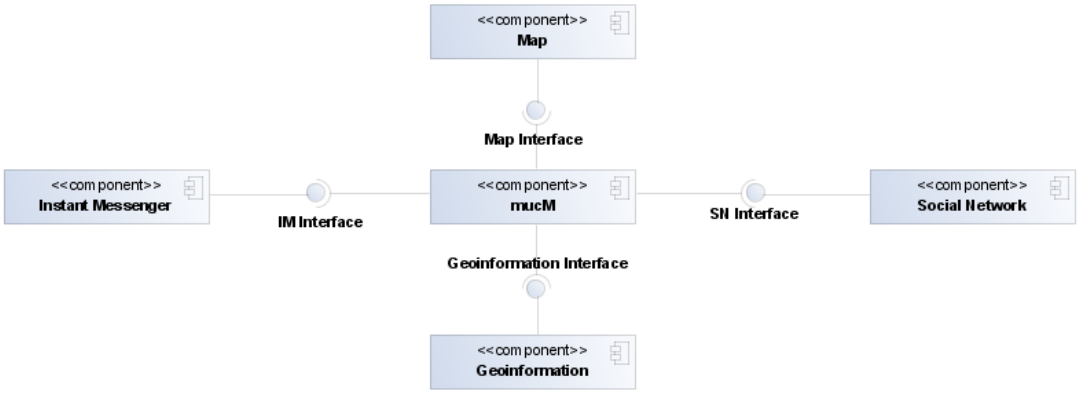

Abbildung 4.13.: Fachliche Schnittstellen zwischen den externen Komponenten und dem mucM

Das Diagramm verdeutlicht, dass die externen Systeme die Schnittstellen anbieten, die der `mucM` benötigt, um seine Funktionalität zur Verfügung stellen zu können. Diese angebotenen Schnittstellen und deren Methoden werden für jede Komponente beschrieben.

IM-Interface: Die `IM`-Schnittstelle stellt vier Methoden als Grundfunktionalität zur Verfügung:

- `provideChat(Buddy, Message)` stellt eine Chat-Verbindung zwischen dem `User` und einem `Buddy` her und überträgt die Nachricht.

- `provideBuddyStatus(Buddy)` stellt den Status eines Buddies zur Verfügung.
- `call(Buddy)` stellt eine Verbindung zum Telefonieren zwischen dem `User` und einem `Buddy` her.
- `fileExchange(Buddy, File)` ermöglicht es, eine Datei vom `User` zu einem `Buddy` zu schicken.

SN-Interface: Die `SN`-Schnittstelle stellt ebenfalls vier Methoden als Grundfunktionalität zur Verfügung:

- `provideFriends()` gibt alle Freunde in dem Freundesnetzwerk zurück.
- `provideFriendsOfAFriend(Friend)` gibt alle Freunde eines bestimmten Freundes zurück.
- `provideFriendDetails(Friend)` stellt die Details zu einem Freund zur Verfügung.
- `sendMessage(Friend, Message)` sendet eine Nachricht an einen Freund.

Geoinformation-Interface: Für die `Geoinformation`-Schnittstelle ist eine Methode ausreichend:

- `provideGeoinformation(Position, Range)` stellt die Geoinformationen in einem angegebenen Radius um die angegebene Position zur Verfügung.

Map-Interface: Die `Map`-Schnittstelle stellt drei Methoden zur Verfügung:

- `displayMap(Name, Position)` stellt den zu einer Position gehörenden Kartenausschnitt bereit und zeigt die entsprechende Position mit Namen an.
- `provideRoute(Position, Position)` berechnet eine Strecke und stellt diese als Karte dar.
- `navigateTo(Position, Position)` navigiert von der aktuellen Position zu einer Zielposition.

4.3. Technische Architektur

Aufbauend auf die entworfene fachliche Architektur wird an dieser Stelle die technische Architektur beschrieben. Die technische Architektur legt fest, wie die fachliche Architektur umgesetzt wird. Es wird beschrieben, wie die Komponenten tatsächlich verteilt werden, in welche Schichten der mucM aufgeteilt wird, wie die Kommunikation stattfindet und welche Protokolle zum Einsatz kommen.

4.3.1. Technische Verteilung

Das folgende Verteilungsdiagramm 4.14 zeigt entsprechend des fachlichen Verteilungsdiagrammes eine Client-Server Architektur. Bei den eigenen Systemen wird eine zusätzliche Unterscheidung vorgenommen. Neben den eigenen Systemen, die realisiert werden, gibt es welche, die nicht detaillierter betrachtet werden.
Damit der mucM auf möglichst vielen Endgeräten genutzt werden kann ebenso wie zur Demonstration, dass eine gute Kapselung vorliegt, sind drei verschiedene Arten von Clients vorgesehen. Die erste Möglichkeit ist ein Web-Client. Der ist plattformunabhängig und benötigt zur Ausführung lediglich einen Browser. Die zweite Möglichkeit ist ein Java-Client, der ebenfalls plattformunabhängig ist, aber zum Beispiel den Vorteil bietet, dass der mucM als Service laufen kann. Die Kommunikation dieser beiden Möglichkeiten lässt sich unter Anderem über Representational State Transfer (REST) umsetzen. Die beiden Clients dienen in diesem Zusammenhang nur der Veranschaulichung und werden daher nicht weiter behandelt, da stationäre Clients nicht Schwerpunkt dieser Arbeit sind. Sie zeigen, dass das System darauf ausgelegt ist, erweitert zu werden und verdeutlicht, dass der Server gekapselte, offene Schnittstellen zur Integration bereitstellt. Der dritte Client ist der Android-Client, der tatsächlich realisiert wird. Dieser soll über das XMPP-Protokoll mit dem eigenen Server kommunizieren. Warum sich das Protokoll besonders gut eignet, wird in diesem Abschnitt später geklärt.

4.3.1.1. Kommunikation allgemein

Grundsätzlich ist zu der Kommunikation zwischen den einzelnen Systemen Folgendes zu sagen: Die Kommunikation wird zwischen sämtlichen Schnittstellen auf dem Application-Layer implementiert. Da bei der Erstellung der Architektur sowohl der Modularität als auch der Erweiterbarkeit besondere Aufmerksamkeit zuteil wurde, können zur Kommunikation zwischen den Schnittstellen grundsätzlich alle Protokolle des Application-Layers genutzt werden. Dieses ist besonders wichtig, da die mobilen Protokolle noch in der Entwicklung stecken und sich keines der aktuellen Protokolle bisher durchsetzen konnte. Sobald das der Fall ist, kann man das Protokoll mit relativ geringem Aufwand austauschen. Im Folgenden werden Vorschläge gegeben, mit welchen Protokollen sich die Kommunikation gut realisieren lässt.

4.3.1.2. Client-Server Kommunikation

Wie zuvor schon angesprochen eignet sich für die Web- und Java-Clients die Einbindung als Webservice zum Beispiel per REST. Auf diese Weise werden im Zuge des Web 2.0 sehr viele Anwendungen im Internet bereitgestellt und eingebunden. Für den Android-Client bietet sich das offene XMPP-Protokoll an, da es ein auf XML basierendes „Near-Real-Time“-Protokoll ist. Es wurde extra für Instant Messaging und Presence Information entwickelt und ist zu einem Standard geworden. Es ist zudem erweiterbar, was den Anforderungen aus der Analyse sehr gut entspricht und unterstützt viele andere Features, die in einem Messenger benötigt werden. Das sind beispielsweise E-Mail, VOIP, Dateiübertragung, SMS und viele mehr. Auch

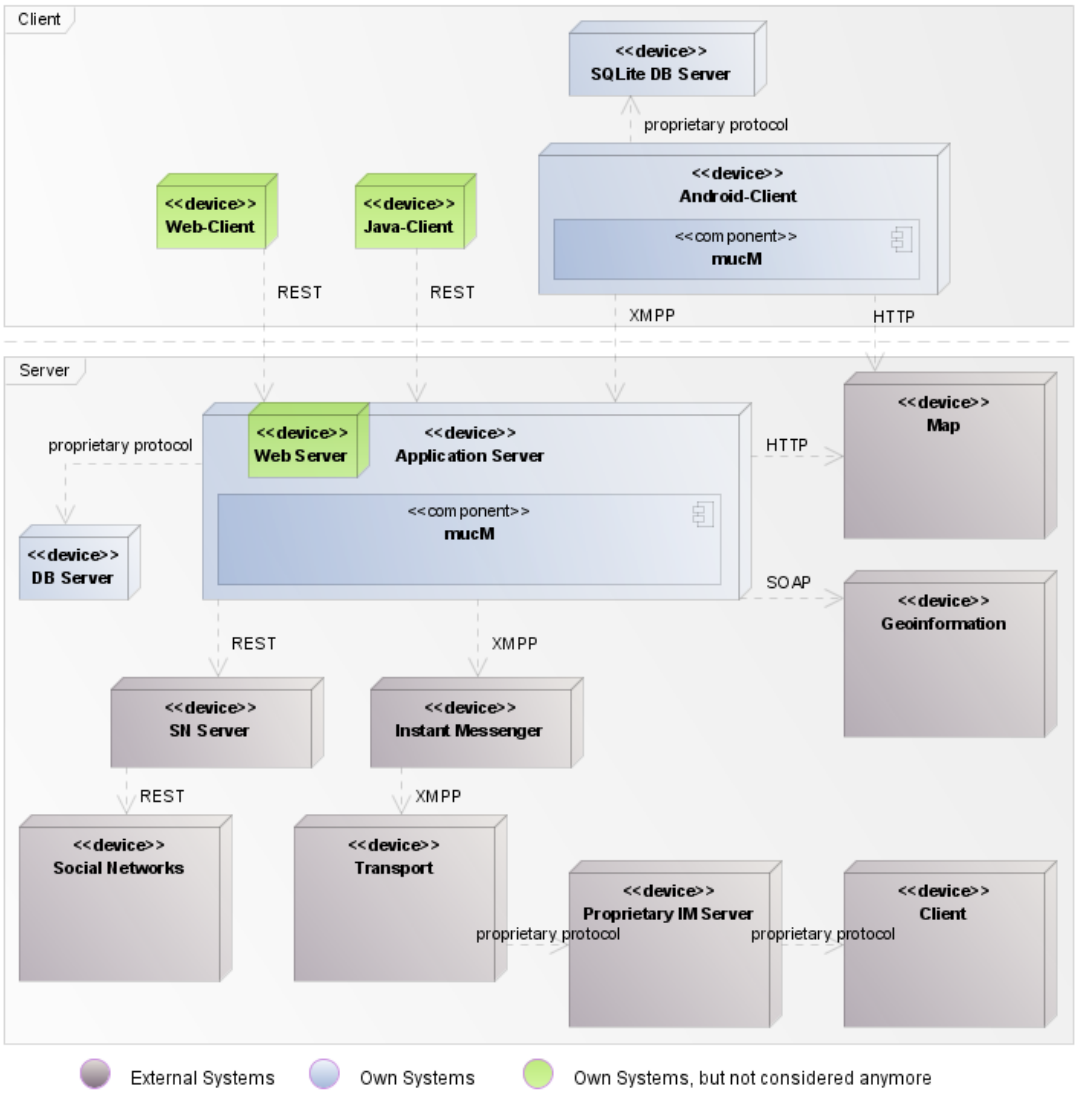

Abbildung 4.14.: Technisches Verteilungsdiagramm des Gesamtsystems

ein Geolocation Feature ist bereits in der Entwicklung. Zu erwähnen bleibt, dass das Netzwerk dezentral aufgebaut ist und jeder die Möglichkeit hat, seinen eigenen Server bereitzustellen. Verschlüsselung wird sowohl vom Client zum Server als auch von Server zu Server standardmäßig angeboten. Zusätzlich besteht die Möglichkeit einer Ende-zu-Ende-Verschlüsselung.

4.3.1.3. Kommunikation mit den externen Systemen

Die Schnittstellen der einzelnen externen Systeme sind sehr unterschiedlich. Diese variieren von Produkt zu Produkt. Daher wird die Kommunikation für jedes externe System separat beschrieben.

Map: Bei der Map-Komponente ist es wichtig, eine Karte zu benutzen, die möglichst vollständig, aktuell und frei verfügbar ist. Dazu gibt es verschiedene Möglichkeiten, wie Google Maps oder Open Street Map. In diesem Fall wurde sich für Google Maps entschieden, da bei Android Google Maps bereits integriert ist. Allerdings ist das Kartenmaterial austauschbar, indem einfach die Schnittstelle eines anderen Anbieters implementiert wird. Die Kommunikation mit Google Maps läuft über das HTTP-Protokoll mittels einer extra dafür bereitgestellten API. Wie bereits erwähnt, sieht Android die direkte Kommunikation mit Google Maps vor, aber auch eine Kommunikation über den eigenen Server als Vermittler ist möglich.

Geoinformation: Es gibt eine Vielzahl an Geoinformationssystemen, die verschiedene Informationen bereithalten. Es gibt beispielsweise Plattformen, die Bilder zu bestimmten Orten zur Verfügung stellen. Ein weiteres Beispiel ist Wikipedia, die bereits eine Vielzahl der Artikel mit den zugehörigen Koordinaten ausgestattet haben. Eine Möglichkeit, diese verschiedenen Systeme miteinander zu kombinieren, gibt es noch nicht. Daher besteht die Möglichkeit, entweder nur eins der existierenden Systeme einzubinden oder mehrere parallel einzubinden. Das bedeutet, dass für jedes neue System eine weitere Schnittstelle integriert werden muss. Für diesen konkreten Fall wurde sich dafür entschieden, exemplarisch Wikipedia zu integrieren. Die Wikipediaartikel lassen sich über Web Services einbinden und abrufen.

Social Network: Bei den sozialen Netzwerken gibt es die Möglichkeit, durch eine vereinheitlichte, offene Schnittstelle mehrere Netzwerke auf einmal zu integrieren. Google hat mit dem Projekt „Open Social“ die Möglichkeit geschaffen, dass sich alle sozialen Netzwerke anschließen und durch die Integration der „Open Social“ Schnittstelle miteinander verbinden lassen. Diese Schnittstelle ermöglicht es allerdings nicht nur soziale Netzwerke miteinander zu verbinden, sondern auch das selbe Plugin bei allen kooperierenden Netzwerken einsetzen zu können. Die Kommunikation mit den Netzwerken läuft über REST ab. Da nicht alle sozialen Netzwerke diese API integrieren, besteht die Überlegung, eine universellere API zu erstellen, um möglichst alle sozialen Netzwerke einbinden zu können. Dieser Gedanke wird nicht weiter verfolgt, da prinzipiell alle sozialen Netzwerke die freie „Open Social“ API einbinden können und damit das Problem bereits gelöst wäre.

Instant Messaging: Im Instant Messaging Bereich gibt es eine Vielzahl von Messengern mit einer Vielzahl von proprietären Protokollen. Auch hier gibt es die Möglichkeit, alle proprietären Protokolle der Messenger, die unterstützt werden sollen, einzeln zu integrieren. Durch diesen Ansatz kann jedes Protokoll detailliert integriert werden, wodurch sämtliche Funktionen des jeweiligen Messengers unterstützt werden. Dieses erfordert allerdings einen enormen Aufwand. Ein sinnvollerer Ansatz ist folgender: Zum einen werden von den proprietären Protokollen nur die Standardfunktionen unterstützt, da dieses in den meisten Fällen ausreicht. Zum anderen wird ein offenes, standardisiertes Protokoll verwendet, mit dem die anderen Protokolle kommunizieren können. Zuvor wurde das XMPP-Protokoll kurz beschrieben. Das Protokoll eignet sich sehr gut für dieses Konzept. Auf diese Weise kommuniziert der eigene Server mit nur einer Verbindung über nur ein Protokoll (XMPP) mit dem sogenannten Jabber Server. Der Jabber Server kommuniziert selbst auch über das XMPP-Protokoll mit den Jabber Clients. Um mit anderen proprietären Messengern kommunizieren zu können, dienen sogenannte Transports als Gateway. Die Kommunikation zu den Transports findet auch über das XMPP-Protokoll statt. Der Transport sorgt dafür, dass die XMPP Nachricht in das proprietäre Protokoll übersetzt wird. Diese proprietäre Nachricht wird nun zum proprietären Server geschickt. Der wiederum leitet die Nachricht an den proprietären Client weiter. Entsprechend andersherum funktioniert der Rückweg. Von dieser Übersetzung bemerken weder der proprietäre noch der Jabber Client etwas. Selbstverständlich geschieht die Übersetzung in proprietäre Protokolle nicht automatisch. Die Transports müssen pro proprietäres Protokoll entwickelt werden. Der positive Effekt für dieses Konzept ist, dass diese Aufgabe ausgelagert wird und somit in diesem Konzept nicht realisiert werden muss.

4.3.2. Anwendungsarchitektur

Nachfolgend wird die technische Anwendungsarchitektur beschrieben, die einen abstrakten Überblick über die Schichten und Komponenten gibt. Entsprechend der fachlichen Verteilung wird an dieser Stelle auch zwischen der Server- und der Client-Anwendung unterschieden.

4.3.2.1. Server-Anwendung

Der Server übernimmt so viele Teile der Geschäftslogik wie möglich, da in diesem Szenario prinzipiell von einer ständigen Verbindung ausgegangen wird. Aufgrund dieser Voraussetzung werden die Daten auf dem Server gespeichert. Abbildung 4.15 zeigt die Architektur.

Application Layer: Der `Application Layer` enthält die gesamte auf dem Server ausführbare Logik. Entsprechend der detaillierten Beschreibung der fachlichen Komponenten in Kapitel 4.2.2 werden die hier aufgeführten Komponenten auf dem Server ausgeführt. Die Messengerfunktionalität in Form von Buddy-, Information- und BuddyListfunktionalität wird

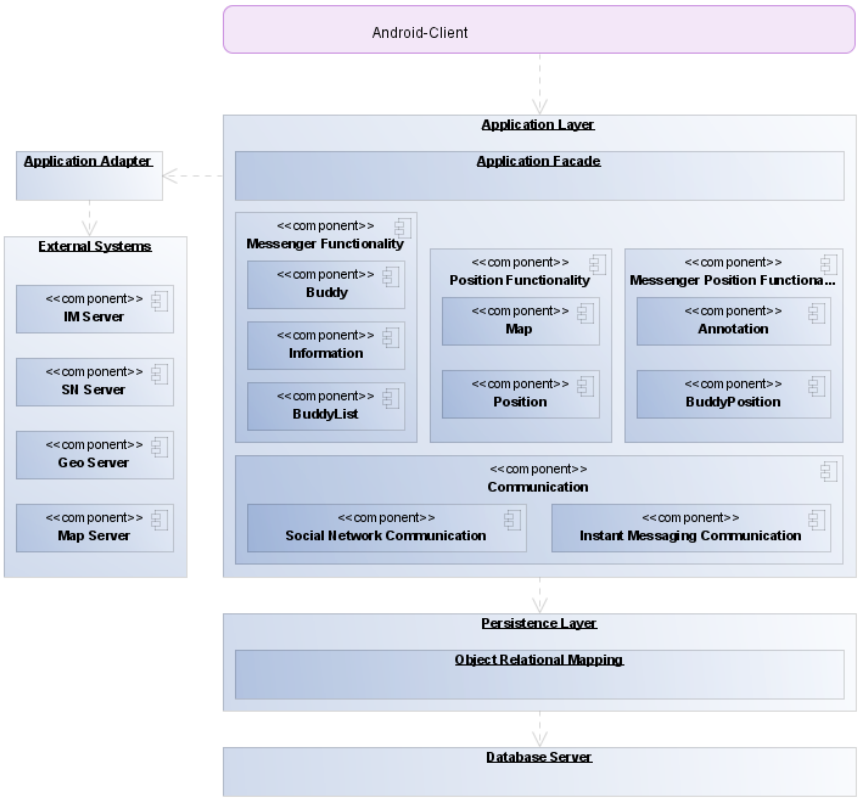

Abbildung 4.15.: Konzeptionelle Sicht auf die Zwei-Schichten-Architektur des Servers

bereitgestellt. Ebenfalls wird die Logik zur Bereitstellung der Map- und Positionsfunktionalitäten zur Verfügung gestellt. Auch die Funktionalitäten für die Annotation und Buddy-Position Bereitstellung sowie die Kommunikation mit den sozialen Netzwerken und Instant Messengern findet auf dem serverseitigen `Application Layer` statt. Die Anwendungslogik wird in der `Application Facade` gekapselt und durch eine einheitliche Schnittstelle nach außen dargestellt. Des Weiteren können auf diese Weise Teile der Anwendungslogik ausgetauscht werden ohne dass Probleme auftauchen, solange die neue Anwendungslogik die gleiche Schnittstelle implementiert. Diese Schnittstelle kann von den potentiellen Clients eingebunden werden. Um eine Verbindung zu den externen Systemen herstellen zu können, muss jedes System einen `Application Adapter` nutzen. Dieser ist dafür zuständig, die zumeist proprietären Schnittstellen, die die externen Systeme bereitstellen, an die einheitliche Schnittstelle der `Application-Facade` anzupassen. Sollen beispielsweise verschiedene `Map`-Systeme benutzt werden (von Google und von Yahoo), haben diese Komponenten in der Regel keine einheitliche Schnittstelle. Damit dennoch beide benutzt werden können, wird für

beide ein Adapter entwickelt. Entsprechend gilt dies für alle anderen externen Systeme. Auf diese Weise können beliebig viele externe Systeme, die die gleiche Funktionalität aufweisen, eingebunden werden. Damit lässt sich die Zuverlässigkeit, falls einmal ein externes System ausfällt, erhöhen. Wenn also die Google-Maps-Komponente nicht mehr zur Verfügung steht, kann einfach auf die Yahoo-Maps-Komponente zurückgegriffen werden, dadurch hat man an dieser Stelle keinen Single-Point-Of-Failure mehr.
Es gibt eine Sondersituation bei Android. Die Google-Maps-Komponente ist bereits in den Client integriert. Auf Grund der Kapselung und des einheitlichen Vorgehens soll dieser Zugriff nicht direkt geschehen, sondern über die `Application Facade`. Dabei wird der Aufruf einfach weitergereicht. Auf diese Weise bleibt das Konzept in sich konsistent und bevorzugt nicht ein bestimmtes externes System.

Persistence Layer: Der `Persistence Layer` ist zuständig für die Persistierung sämtlicher notwendiger Daten aus dem `Application Layer`. Beispielsweise müssen die BuddyListe oder die berechnete Route gespeichert werden. Es wird ein Mapping von transienten Daten in persistente Daten vorgenommen. Die dafür genutzte Funktionalität wird als Object-Relational Mapping (ORM) bezeichnet. Dadurch wird es ermöglicht, Objekte in einer relationalen Datenbank zu speichern und aus diesen Datensätzen wieder Objekte zu erstellen, ohne die Datenbankzugriffe explizit in Structured Query Language (SQL) programmieren zu müssen. Es wird auf diese Weise eine Unabhängigkeit gegenüber bestimmter SQL-Dialekte und damit gegenüber einer bestimmten Datenbank erreicht.

4.3.2.2. Client-Anwendung

Der Client ist als eine Drei-Schichten-Architektur konzipiert worden. Dabei liegt der Schwerpunkt des Clients auf der Präsentation der Daten und der Kommunikation mit dem Benutzer. Die folgende Abbildung 4.16 zeigt die clientseitige Architektur.

Presentation Layer: Der Benutzer hat direkten Zugriff auf den `Presentation Layer`, sowie auch der `Presentation Layer` mit dem Benutzer kommunizieren und Ereignisse mitteilen kann. An dieser Stelle werden Ereignisse durch Eingaben des Benutzers ausgelöst und fachliche Daten und Ereignisse angezeigt. Benutzereingaben erfolgen in Form von Tastatureingaben. Dabei besteht die Möglichkeit, eine virtuelle oder physische Tastatur zu nutzen. In dieser Schicht wird die Kommunikation durch Anzeige, Eingabe oder Ausgabe ermöglicht. Es wird das Kartenmaterial mit den entsprechenden Informationen angezeigt, welches schließlich bearbeitet werden kann. Auch können die Informationen zu den Annotationen angezeigt und verändert werden. Zuletzt kommt es hier zur Anzeige der BuddyDetails, über die der Benutzer alle notwendigen Angaben zu seinen Buddies erfährt. Die Weiterleitung der Benutzereingaben zum `Application Layer` erfolgt über die `Presentation Control`. Dort werden die entsprechenden Eingaben weiterverarbeitet.

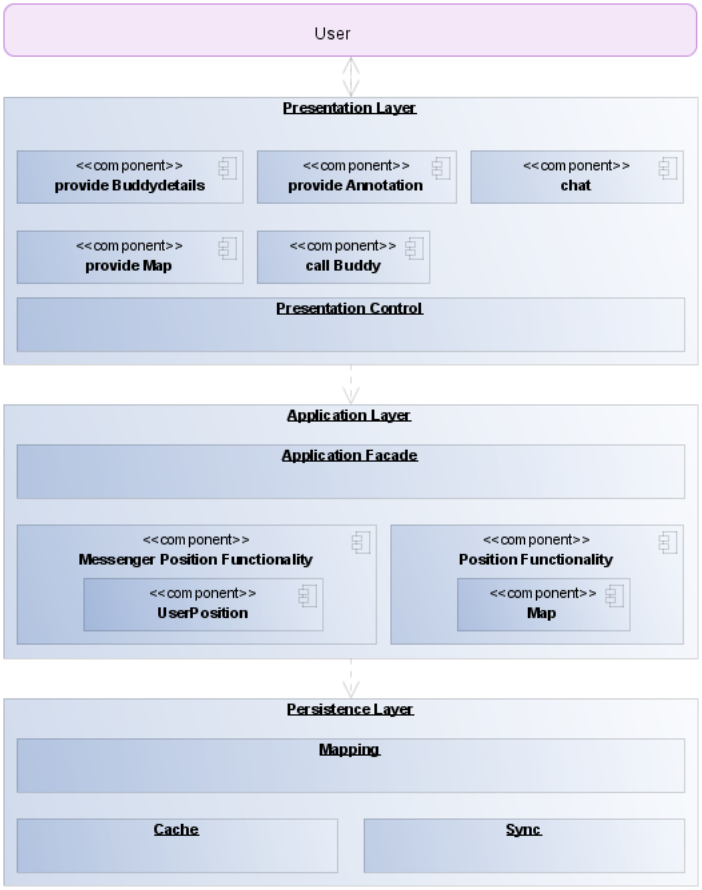

Abbildung 4.16.: Konzeptionelle Sicht auf die Drei-Schichten-Architektur des mobilen Clients

Application Layer: Der `Application Layer` des Clients fällt deutlich kleiner aus, da die meiste Logik auf dem Server ausgeführt wird. Zur Kapselung gibt es auch hier eine `Application Facade`, um den Zugriff fachlicher Abläufe von der Präsentationslogik zu ermöglichen. Die einzige, die auf dem Client in der Anwendungslogik stattfinden muss, ist die Feststellung der eigenen Position in der Komponente `UserPosition`. Da das Gerät selbst die Position über Sensoren feststellt, befindet sich auch auf dem Client die entsprechende Anwendungslogik. Zusätzlich liegt auf dem Client die `Map`-Komponente aus einem besonderen Grund. Der Android-Client stellt eine direkte Verbindung zu einer `Map`-Komponente her. Daher kann diese Verbindung auch benutzt werden. Das ist allerdings nicht verpflichtend.

Persistence Layer: Um den Anforderungen nach möglichst viele Aufgaben auf den Server auszulagern, um Ressourcen zu sparen und Batterielaufzeit zu schonen und unter der Annahme, immer mit dem Server verbunden zu sein, ist eine Persistenzschicht auf dem Client nicht zwingend erforderlich. Aber aufgrund der Tatsache, dass sich bei einer kabellosen und zusätzlich sehr mobilen Verbindung keine dauerhafte Verbindung ohne Abbrüche garantieren lässt, wird es entgegen der fachlichen Anforderungen als sinnvoll angesehen, eine kleine Persistenzschicht auf dem Client zu integrieren. Auch in diesem Fall ist ein Mapping der transienten Daten auf persistente Daten notwendig, allerdings nicht in dem Maße wie auf dem Server. Auf dem Client ist eine spezielle Datenbank vorhanden, die auf ein mobiles ressourcenbeschränktes System ausgelegt ist. Daher ist die Abstraktionsschicht zum Einbinden verschiedener Datenbanken nicht vordergründig. In diesem Fall ist es wichtiger, auf eine dem System angepasste Datenbank möglichst schnell mit speziellen und nicht mit generellen Methoden zugreifen zu können. Auf diese Weise hat der Benutzer auch die Möglichkeit, Daten ohne Verbindung zu verändern, daher wird zusätzlich eine Synchronisationsfunktionalität notwendig.

4.3.3. Technischer Ablauf

An dieser Stelle wird die dynamische Sicht der technischen Architektur beschrieben. Im Vergleich zur fachlichen Architektur variiert die Laufzeitsicht sehr stark abhängig von der tatsächlich verwendeten Technik. Daher wird nur exemplarisch dargestellt, wie diese bei der `Geoinformation-` / `Map`-Komponente und der `Messenger`-Komponente aussehen kann. Die Abbildung 4.17 zeigt das Verhalten der `Geoinformation-` bzw. `Map`-Komponente.

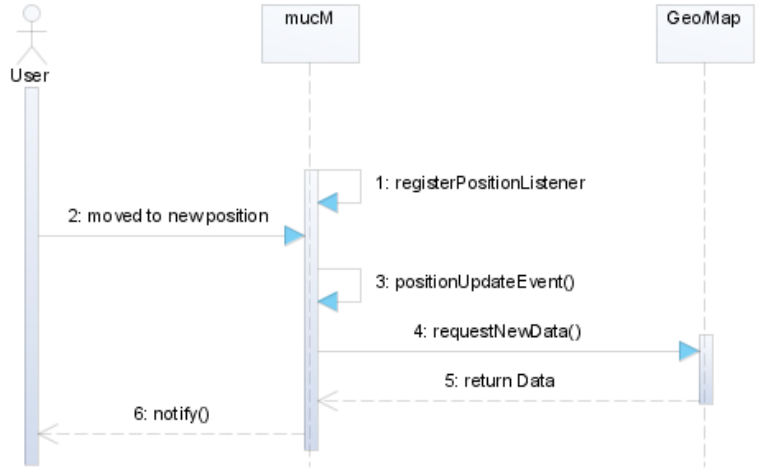

Abbildung 4.17.: Technisches Sequenzdiagramm der Geoinformation-/Map-Komponente und der zugehörigen Module

Zuerst muss sich der `mucM` für Positionsveränderungen registrieren (1). Wenn der `User` sich dann an eine neue Position begibt (2), wird ein `positionUpdateEvent` beim `mucM` ausgelöst (3). Dieser holt sich daraufhin die neuen Daten von der entsprechenden Komponente (4), welche die neuen Daten an den `mucM` zurückgibt (5). Der `User` wird darüber in Kenntnis gesetzt (6).
Die Abbildung 4.18 zeigt das Verhalten der `Messenger`-Komponente bezogen auf den Nachrichtenaustausch.

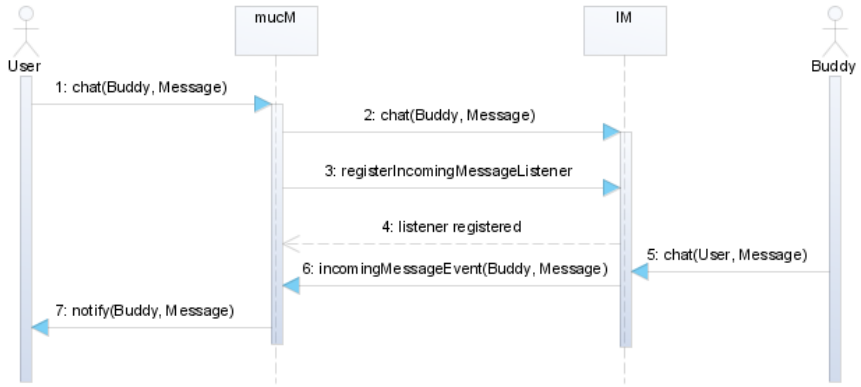

Abbildung 4.18.: Technisches Sequenzdiagramm der Messenger-Komponente und der zugehörigen Module

Entsprechend des fachlichen Diagrammes kann der `User` eine Nachricht an einen Buddy über den `mucM` (1) an den `IM` (2) schicken. Allerdings muss dieser auch eine Möglichkeit haben, diese Nachricht zu empfangen. Dazu ist Folgendes notwendig: Der `User` muss sich für eingehende Nachrichten beim `IM` (3) registrieren. Die erfolgreiche Registrierung wird zurückgemeldet (4). Nun ist der `User` bereit, Nachrichten zu empfangen. Schickt der `Buddy` nun eine Nachricht an den `User` über den `IM` (5), so wird ein entsprechendes Event ausgelöst (6). Auf diese Weise kann die Nachricht zugestellt werden (7).

4.3.4. Technische Schnittstellen

In Abschnitt 4.2.6 wurden die fachlichen Schnittstellen der externen Systeme definiert. Dabei gibt es für jedes externe System, für das es noch keine standardisierte Schnittstelle gibt, eine proprietäre technische Schnittstelle. Damit theoretisch jedes Geoinformationssystem oder jede `Map`-Komponente eingebunden werden kann, wird eine einheitliche technische Schnittstelle für diese Systeme definiert. Entsprechend der technischen Serveranwendung gibt es einen Adapter (Kapitel 4.3.2.1), der die proprietären technischen Schnittstellen in die einheitliche

technische Schnittstelle übersetzt. Da die fachlichen Schnittstellen aufeinander abgestimmt sind, können diese die einheitlichen technischen Schnittstellen direkt benutzen.

IM- und SN-Interface: Für die `IM`-Komponente soll XMPP benutzt werden, welches bereits als Standard anerkannt ist. Als Schnittstelle sind zwei Methoden notwendig, die nachfolgend erklärt werden.

- `createConnection()` stellt eine Verbindung zum Server her.
- `exchangeData(Receiver, Data)` Nachdem die Verbindung hergestellt wurde, können mit Hilfe dieser Methode jegliche Art von Daten an einen bestimmten Empfänger übertragen werden.

Bei genauer Betrachtung fällt auf, dass die `SN`-Komponente sich durch die gleiche Schnittstelle mit den gleichen Methoden realisieren lässt. Es müssen ebenfalls eine Verbindung zum externen System hergestellt werden und Daten im Kontext eines Freundes (Receiver) verarbeitet werden.

Geoinformation-Interface: Die technische Schnittstelle der `Geoinformation`-Komponente entspricht genau der fachlichen Schnittstelle dieser Komponente (siehe Abschnitt 4.2.6).

Map-Interface: Nach näherer Betrachtung verschiedener Map-Anbieter ist zu sagen, dass die Schnittstellen sehr unterschiedlich und auch sehr komplex sind. Daher wird die technische Schnittstelle stark abstrahiert. Auf diese Weise kann eine einfache Schnittstelle definiert werden und sie bleibt trotzdem erweiterbar.

- `displayOnMap(Position[], Data)` Auf diese Weise können ein oder mehrere Positionen angezeigt werden. Eine Position kann dabei aus Ortsangaben oder aus Längen- und Breitengrad bestehen. Unter Data sind zusätzliche Informationen zu verstehen. Ein Beispiel ist das Zoomlevel, das übergeben werden kann.

4.4. Fazit

Abschließend lässt sich zur Konzeption sagen, dass die funktionalen und nicht-funktionalen Anforderungen aus dem dritten Kapitel weitestgehend berücksichtigt wurden. Auch die Differenzierung zwischen dem fachlichem und dem technischem Konzept, wobei das technische Konzept aus dem fachlichen entwickelt wurde, bot eine gute Unterstützung zur Erstellung eines robusten, modularen und erweiterbaren Konzeptes. Die Trennung zwischen fachlicher und technischer Architektur, sowie der modulare Aufbau, sind ein gutes Fundament für eine verteilte Entwicklung in einer Open Source Community. Damit konnte eine gute Grundlage für eine erfolgreiche Realisierung geschaffen werden.

5. Realisierung

In den vorherigen Kapiteln wurde der Grundstein für eine erfolgreiche Realisierung gelegt. Nachdem die Anforderungen in Kapitel 3 gestellt wurden, wurde im darauf folgenden Kapitel 4 die fachliche und technische Architektur erstellt. Dieses Kapitel beschreibt schließlich die Komponenten, die tatsächlich prototypisch realisiert wurden.

5.1. Abweichungen vom Konzept

Es gibt häufig Unterschiede zwischen dem Konzept und der tatsächlichen Realisierung. An dieser Stelle werden die Abweichungen vom Konzept zu den umgesetzten Komponenten dargestellt, beschrieben und entsprechend begründet. Die Abbildung 5.1 zeigt den Inhalt der prototypischen Umsetzung. Darin werden die Verteilung und zu realisierende Komponenten berücksichtigt.

5.1.1. Eigene Serverkomponente

Aufgrund der Tatsache, dass ein Prototyp als „Proof of Concept“ erstellt werden sollte und es noch keine Android Endgeräte gibt, wurden die Ressourcenknappheit und die Batterielaufzeit der Endgeräte vernachlässigt. Daher wird die Funktionalität des Prototyps direkt auf dem Client stattfinden. Bei einer Realisierung über den Prototyp hinaus soll die eigene Serverkomponente entsprechend des Konzeptes umgesetzt werden, damit die beschriebenen Vorteile auch tatsächlich genutzt werden können. Zu diesem Zeitpunkt entsteht durch die Umsetzung der eigenen Serverkomponente kein nennenswerter Vorteil. Wenn die prototypische Umsetzung auf dem Client funktioniert, ist die Integration der Serverkomponente als erste gravierende Verbesserung umzusetzen. Gleichzeitig ist die reine Umsetzung auf dem Client als „Proof of Concept“ ausreichend. Wenn der Client als Fat-Client lauffähig ist, wird er auch als Thin-Client laufen, mit dem Unterschied, dass einiges performanter läuft und weniger Datenoverhead übertragen werden muss, weil der Server die rechenintensive Logik ausführen und die Nachrichten, sowie die Anzahl der Verbindungen auf ein Minimum reduzieren kann.

5.1.2. Persistenz

Da auf eine eigene Serverkomponente verzichtet wird, fällt die serverseitige Persistenz ebenfalls weg. Allerdings ist für den Client auch eine Persistenzschicht vorgesehen, die beispielhaft realisiert werden soll, um zu zeigen, dass das Konzept schlüssig ist. Als Beispiel dafür

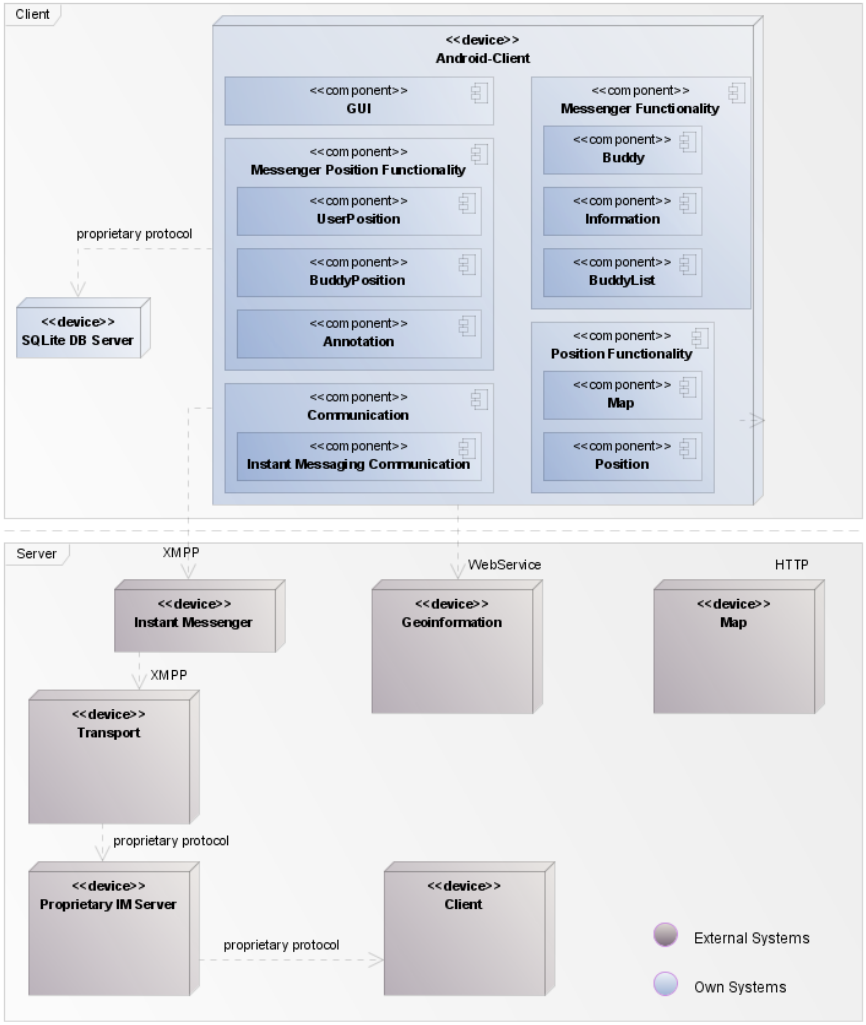

Abbildung 5.1.: Verteilung der realisierten Komponenten des Prototyps

dient das externe System Geoinformation, welches in diesem Zusammenhang realisiert werden soll, um eine Offline-Funktionalität zur Verfügung stellen zu können. Dazu werden lokal die Annotationen in der Nähe des Benutzers in der auf dem Client zur Verfügung stehenden `SQLite`-Datenbank gespeichert. Weitere Details dazu sind in Kapitel 5.2.4 zu finden.

5.1.3. Externe Systeme

Von den ursprünglich vier externen Systemen werden nur drei Systeme prototypisch umgesetzt. Die drei Systeme, die integriert werden, stellen das meiste der Grundfunktionalität, die zuvor beschrieben wurde, zur Verfügung. Da die Integration des vierten externen Systems keinen deutlichen Gewinn an Funktionalität mit sich bringt, wird aus zeitlichen Gründen auf eine Realisierung der `Social Network`-Komponente verzichtet. Es werden also folgende externe Systeme integriert: Der `Instant Messenger` als Hauptfunktionalität für die Kommunikation, das `Map`-System zur Darstellung der ortsgebundenen Informationen und damit auch das `Geoinformation`-System, welches ortsgebundene Informationen zur Verfügung stellt.

5.1.4. Anwendungslogik

Die gesamte Anwendungslogik muss aufgrund der Entscheidung, auf eine eigene Serverkomponente zu verzichten, auf dem Client umgesetzt werden. Das bedeutet, dass neben der sowieso vorgesehenen `BuddySignedOnPosition`- und der `Map`-Komponente sämtliche Serverkomponenten aus dem Application-Layer auf dem Client implementiert werden müssen. Es wird also als Basis eine rudimentäre `Messenger`-Funktionalität auf dem Client umgesetzt. Diese wird im Kapitel 5.2.1 ausführlich beschrieben. Für die Context-Awareness, die wie zuvor erwähnt für diesen Prototypen auf die Location- Awareness reduziert wird, wird die `Geoinformation`-Funktionalität umgesetzt. Diese wird in Kapitel 5.2.2 näher beschrieben. Um die Location-Awareness sinnvoll nutzen zu können, wird die `Map`-Funktionalität umgesetzt, die in Kapitel 5.2.3 verdeutlicht wird.

5.2. Konkretisierung

Nachdem beschrieben wurde, welche Komponenten und Funktionalitäten im Zuge einer prototypischen Entwicklung umgesetzt werden sollen, wird an dieser Stelle beschrieben, wie diese einzelnen Komponenten tatsächlich umgesetzt wurden.

5.2.1. Messenger

Ziel bei der Umsetzung der `Instant Messenger`-Komponente ist es laut Konzept, einen Server aufzusetzen, der das XMPP Protokoll unterstützt, um dieses mit einem Jabber Client

verbinden zu können, der ebenfalls das XMPP Protokoll spricht, da XMPP ein offener Standard für Instant Messenger ist.

5.2.1.1. Integration von XMPP

Google hat sich entgegen der ersten SDK-Versionen entschieden XMPP doch nicht als Protokoll für das Instant Messaging zu nutzen und auch in der aktuellen SDK-Version (0.9) kein proprietäres Protokoll auszuliefern. Dieses Problem wurde gelöst, indem eine bereits bestehende XMPP Client Library, die auf Java basiert, auf Android portiert wurde. Es bot sich die Open Source XMPP Client Library für Instant Messaging und Presence Smack von Jive Software an. Die offene Library musste angepasst werden, um auf Android lauffähig zu sein. Zum Einsatz kommt die smack.jar in der Version 3.04, in der hauptsächlich der MXParser, mit all seinen Abhängigkeiten durch den KXmlParser ersetzt wurde. Damit ist die smack.jar auf dem Android Client lauffähig und die XMPP Funktionalität steht zur Nutzung zur Verfügung.

5.2.1.2. XMPP Server

Entsprechend der Anforderungen muss auf einen XMPP Server aufgesetzt werden, um die verschiedenen Jabber Clients miteinander zu verbinden, aber auch um über die sogenannten Transports eine Verbindung zu anderen proprietären Instant Messengern möglich zu machen. Zum Einsatz kommt der XMPP Server Open Fire in der Version 3.6.0a. Dieser Server ist ebenfalls von Jive Software und wird sowohl als Open Source als auch kommerziell angeboten. Im Sinne dieser Masterarbeit wurde sich für die Open Source Variante entschieden. Die Installation ist mit Hilfe eines Menüs einfach durchführbar. Entsprechend der Anforderungen an die Sicherheit lässt sich bei der Konfiguration eine verschlüsselte Verbindung vom Client zum Server und vom Server zu anderen XMPP Servern einstellen. Zusätzlich für die Kommunikation mit proprietären Instant Messengern muss ein extra Plugin installiert werden, das sogenannte `IM Gateway`-Plugin. Damit lassen sich Verbindungen zu anderen proprietären Netzwerken konfigurieren.

5.2.1.3. Instant Messenger Client Entwicklung

Auf dem Client müssen entsprechend der Abblidung 5.1 die dort dargestellten Komponenten umgesetzt werden.

Instant Messaging Communication: Die `Instant Messaging Communication`-Komponente stellt die Kommunikation gekapselt zur Verfügung. Damit die Kommunikation auf dem Android Client während der gesamten Laufzeit des Messengers über verschiedene Activities hinweg zur Verfügung steht, muss die Kommunikation als Android-Service umgesetzt werden. Dies geschieht in mehreren Schritten.
Android stellt eine Android Interface Definition Language (`AIDL`) zur Verfügung, um die

Kommunikation mit den Services zu ermöglichen. In dem Interface `IJabberService` werden die Methoden zum An- und Abmelden an den Messenger Service zur Verfügung gestellt.

```
interface IJabberService {
    void registerMessengerCallback
        (IJabberServiceMessenger cbMessenger);
    void unregisterMessengerCallback
        (IJabberServiceMessenger cbMessenger);
}
```

Listing 5.1: Erstellen einer Schnittstelle zum Registrieren und Deregistrieren

In einer weiteren `AIDL` wird beispielsweise eine Methode zur Verfügung gestellt, die eingehende Nachrichten von einem User empfängt.

```
interface IJabberServiceMessenger {
    void incommingMessage(String user, String message);
}
```

Listing 5.2: Erstellen einer Schnittstelle zum Empfang von Nachrichten

Aus den AIDL Dateien werden automatisch Java Klassen generiert.

Erstellung einer Verbindung: Eine Klasse `XMPP`, die ein Singleton darstellt, stellt die Verbindung zum XMPP-Server her und sorgt durch einen privaten Konstruktor und der statischen Methode `getInstance()` dafür, dass es nur eine Instanz dieser Klasse gibt.

```
private static XMPP instance = new XMPP();
    private XMPP(){}
    public static XMPP getInstance() {
        return instance;
    }
```

Listing 5.3: Nutzung des Singleton-Patterns für genau eine Verbindung

Die Verbindung selbst wird über die Klasse ConnectionConfiguration erstellt. Die Verbindungsdaten bestehen aus der Serveradresse, dem Port, und einem Servicenamen.

```
ConnectionConfiguration connConfig =
    new ConnectionConfiguration(
        connectionSettings.getServer(),
        connectionSettings.getPort(),
        connectionSettings.getServiceName()
);
```

Listing 5.4: Erstellen einer Verbindung zum XMPP-Server

Nachdem die Verbindung zum Server hergestellt wurde, kann man sich als User mit dem zugehörigen Passwort anmelden.

Chat Activity: In den verschiedenen Aktivitäten wie beispielsweise das Darstellen einer BuddyListe oder ein Chat mit einem Buddy, muss sich die Activity jeweils an den Service binden und für den Empfang der Nachrichten registrieren.

```
1  bindService(new Intent(IJabberService.class.getName()),
2      mConnection, Context.BIND_AUTO_CREATE);
3  ...
4  private ServiceConnection mConnection = new ServiceConnection() {
5    public void onServiceConnected(
6      ComponentName className, IBinder service) {
7        mJabberService = IJabberService.Stub.asInterface(service);
8        mJabberService.registerMessengerCallback(mCallbackMessenger)
             ;
9      }
10 }
11 ...
12 private IJabberServiceMessenger mCallbackMessenger =
13  new IJabberServiceMessenger.Stub() {
14   public void incommingMessage(String user, String message)
15       throws RemoteException {
16     if(user.compareTo(buddy)==0){
17       mAdapter.add(message);
18     }
19   }
20 };
```

Listing 5.5: Bindung an einen Service

GUI: Die GUI wird in Android über XML-Dateien erstellt. Auf diese Weise lassen sich, ähnlich wie bei HTML, einfach verschachtelte Tags erstellen. Die GUI Erstellung wird durch eine Vielzahl von vordefinierten Elementen erleichtert. Die XML-Dateien müssen in einem bestimmten Verzeichnis abgelegt werden: `res/layout/`. Diese Elemente sind Unterklassen der `View`-Klassen in Android. In den Activity Klassen werden diese Views in der Startmethode `onCreate()` jeder `Activity` Klasse (ähnlich dem Konstruktor) mit folgendem Aufruf gestartet:

```
1  setContentView(R.layout.login)
```

Listing 5.6: Aufruf einer View in der onCreate-Methode einer Activity

Die Login View ist als ein Beispiel einer möglichen View im Anhang A.3 zu sehen. Dieses XML-Layout besteht aus einem `LinearLayout`, das ein `TableLayout` und ein weiteres `LinerLayout` enthält. Das `TableLayout` enthält fünf `TableRows`, die jeweils aus einem `LinearLayout` mit einer `TextView` und einem `LinearLayout` mit einem `EditText` bestehen. Das `LinearLayout` auf gleicher Ebene wie das `TableLayout` enthält seinerseits auch wieder zwei `LinearLayouts`, die jeweils einen `ImageButton` enthalten. Abbildung 5.2 verdeutlicht die etwas unübersichtliche Verschachtelung der XML-Elemente:

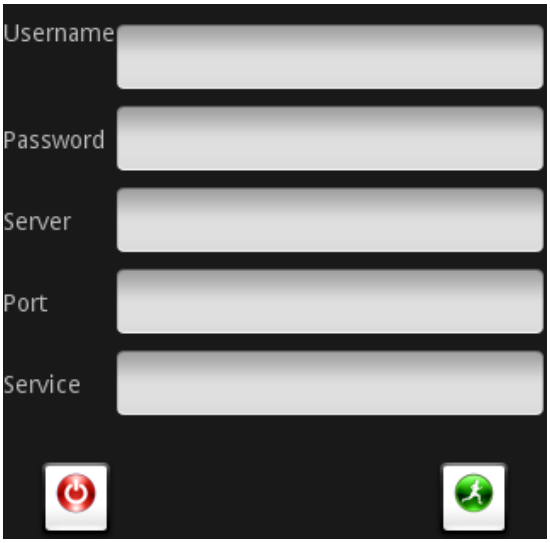

Abbildung 5.2.: Login Screen des Messengers

5.2.2. Geoinformation

Um die Context-Awareness des Messengers unter Beweis zu stellen, ist diese Komponente elementar. Beispielhaft für die Integration eines bereits existierenden, weit verbreiteten und akzeptierten Community-Projektes, das Geoinformationen bereitstellt, fiel die Entscheidung auf Wikipedia. Bei Wikipedia sind bereits viele Artikel zusätzlich mit Koordinaten versehen. Dadurch sind aus einfachen Informationen Geoinformationen geworden. Der zweite wichtige Faktor ist die Möglichkeit der Erweiterung durch die Community. Dies ist bei Wikipedia zweifelsohne gegeben.

5.2.2.1. Integration des Wikipedia-Services

Es gibt bereits Webservices, die zu einer bestimmten Position in der Nähe befindliche Geoinformationen bzw. Annotationen zur Verfügung stellen. Zusätzlich gibt es beispielsweise die Möglichkeit einer Volltextsuche nach Wikipedia Artikeln. Die erstgenannte Möglichkeit wurde hier umgesetzt.
Der Webservice wird über folgende URL mit zusätzlichen Parametern aufgerufen:

```
1 http://ws.geonames.org/findNearbyWikipedia?lat=53.55&lng=9.99&
    radius=10&maxRows=20
```

Listing 5.7: Aufruf des Webservices

Der erste Teil des Aufrufs `http://ws.geonames.org/` ist die Adresse, die den Webservice anbietet. Der nächste Teil `findNearbyWikipedia` ruft den entsprechenden Webservice auf. Der Webservice wird mit vier Parametern aufgerufen. Zuerst kommt der Längengrad `lat=53.55`, gefolgt von dem Breitengrad `lng=9.99`. Der nächste Parameter ist der Radius `radius=10`, der angibt, in welchem Radius (in Kilometern) um die angegebene Position Ergebnisse geliefert werden sollen. Der letzte Parameter `maxRows=20` setzt die maximale Anzahl der gelieferten Ergebnisse fest. Beim Aufruf dieses Webservices müssen diese Parameter abhängig vom Aufenthaltsort des Benutzers und der eingestellten Zusatzinformationen dynamisch übergeben werden. Die Ergebnisse werden in Form einer XML-Datei geliefert. Diese Datei besteht aus maximal so vielen Ergebnissen, wie der Benutzer es eingestellt hat. Ein Ergebnis entspricht dem Tag `<entry>...</entry>`. Eine solche Geoinformation besteht aus einigen weiteren Tags, wobei für diese Realisierung nur folgende Tags von Interesse sind: Der Tag `<title>...</title>` liefert den Titel der Geoinformation, der in einer Liste angezeigt wird. Als Details zu den Geoinformationen lässt sich der Tag `<summary>...</summary>` nutzen. Des Weiteren enthält ein Entry noch die Informationen, die übergeben wurden - `<lat>...</lat>`, `<lng>...</lng>` und `<distance>...</distance>`. Sehr nützlich ist der Tag `<wikipediaUrl>...</wikipediaUrl>`, durch den man zum Originalartikel gelangen kann. Im Anhang A.2 ist ein Auszug aus einer solchen XML-Datei zu lesen.

5.2.2.2. Parsen des Webservice-Ergebnisses

Wie zuvor beschrieben, liefert der Webservice eine XML-Datei zurück. Damit diese genutzt werden kann, muss sie geparst werden. Android stellt dafür einen bereits integrierten SAX-Parser zur Verfügung. Dadurch lassen sich die Daten so konvertieren, dass man eine Menge von einzelnen Geoinformationen erhält. Diese lassen sich dann, wie in Kapitel 5.2.4 beschrieben wird, als einzelne Geoinformationen speichern. An dieser Stelle wird anhand kurzer Codeausschnitte kurz beschrieben, wie das Parsen geschieht. Die vollständigen Methoden sind im Anhang A.2.2 zu finden. Zuerst muss der `SAXParser` initialisiert werden, der eine XML-

Datei lesen kann. Dann muss ihm der Link zu der XML-Datei übergeben werden, woraufhin die Datei geparst werden kann. Folgender Codeausschnitt verdeutlicht dies:

```
1     SAXParserFactory spf = SAXParserFactory.newInstance();
2     SAXParser sp = spf.newSAXParser();
3     XMLReader xr = sp.getXMLReader();
4     xr.setContentHandler(this);
5     xr.parse(new InputSource(url.openStream()));
```

Listing 5.8: Initialisierung des SAX-Parsers

Es werden drei Methoden genutzt, um die Tags, die genutzt werden sollen, zu finden und den Inhalt zu extrahieren.Die erste Methode `startElement(...)` sucht beispielsweise nach dem öffnenden Tag `<title>` und setzt dann das zugehörige Flag auf `true`:

```
1     if (name.trim().equals(title_tag)){
2         inTitle = true;
3     }
```

Listing 5.9: Die start-Methode sucht nach einem öffnenden Tag

Genauso wird mit allen weiteren Tags umgegangen. Die nächste Methode `endElement(...)` sucht das zugehörige schließende Tag `</title>` und setzt das zugehörige Flag auf `false`. Zusätzlich wird in dieser Methode überprüft, ob der Entry vollständig ist. Ist dies der Fall, wird er in der Datenbank gespeichert. Als letztes wird überprüft, ob die Anzahl der maximalen Geoinformationen erreicht wurde. Ist dies der Fall, wird an dieser Stelle abgebrochen.

```
1     if (name.trim().equals(title_tag)){
2         inTitle = false;
3     }
4     if (targetFlag == TARGET_FEED && currentFeed.url != null
5         && currentFeed.title != null) {
6             geoInformationDB.insertFeed(currentFeed.title,
                  currentFeed.url);
7     }
8     if (targetFlag == TARGET_ARTICLES && currentArticle.url !=
          null
9         && currentArticle.title != null) {
10        geoInformationDB.insertArticle(currentFeed.feedId,
              currentArticle.title,
11        currentArticle.url, currentArticle.latitude,
12        currentArticle.longitude, currentArticle.distance);
13        currentArticle.title = null;
14        currentArticle.url = null;
15        articlesAdded++;
```

```
16          if (articlesAdded >= ARTICLES_LIMIT){
17              throw new SAXException();
18          }
19      }
```

Listing 5.10: Die end-Methode sucht nach dem geschlossen Tag

Die dritte Methode `characters(...)` liest die entsprechenden Zeichen dazwischen aus und schreibt die zu den ausgewählten Tags gehörenden Werte in das entsprechende Objekt:

```
1           String chars = (new String(ch).substring(start, start +
                length));
2           if (inTitle){
3               currentArticle.title = chars;
4           }
```

Listing 5.11: characters-Methode fügt die Taginhalte ein

5.2.2.3. Auswertung der geparsten Geoinformationen

In den ersten zwei Schritten wurden die Geoinformationen von einem Webservice erhalten, geparst und in der Datenbank abgespeichert. Nun müssen die Daten zur Anzeige gebracht werden. Dazu wird eine androidspezifische ListView verwendet, mit der die Titel mit Entfernung zum derzeitigen Aufenthaltsort der einzelnen Geoinformationen aufgelistet werden. Durch die Auswahl einer Geoinformation wird die hinterlegte URL zu der Geoinformation aufgerufen. Dazu wird der Browser gestartet und die entsprechende Seite angezeigt. Dadurch, dass man sich auf der originalen Wikipedia Seite befindet, kann man sich entsprechend der Anforderungen für die Erweiterbarkeit durch die Community bei Wikipedia einloggen und den bestehenden Artikel verändern oder einen neuen hinzufügen. In einem weiteren Entwicklungsschritt wäre zu überlegen, ob diese Funktion direkt in die Anwendung integriert werden soll oder nicht. Zusätzlich lässt sich über die Auswahl einer Geoinformation über das Menü, sowohl der Ort der Geoinformation als auch der eigene Aufenthaltsort auf der Karte anzeigen.

5.2.2.4. Erweiterung der Geoinformationen

Bei der prototypischen Realisierung konnte sogar das Erweitern oder Hinzufügen von Geodaten ermöglicht werden. Das wurde über einen kleinen Trick ausgelagert. Dadurch dass auch die URL zu dem gesamten Wikipediaartikel geparst wird, lässt sich dieser auch direkt im von Android mitgelieferten Browser öffnen. Auf diese Weise kann man sich auf der Webseite einloggen und dort direkt die Artikel bearbeiten oder neue hinzufügen.

5.2.3. Map

Bei der Einbindung einer Map wurde sich aufgrund der bereits integrierten Funktionalität für Google Maps auch für diese entschieden. Die Integration dieser Komponente geschieht durch das Starten einer neuen Activity, in diesem Fall einer MapActivity. Mit dem folgenden Aufruf wird eine neue Activity gestartet und ihr wird ein Intent übergeben.

```
1 startActivityForResult(intentLocation, ACTIVITY_VIEW);
```

Listing 5.12: Starten einer Subactivity mit Übergabeparametern

Damit die Daten, wie beispielsweise die Positionsdaten und der Titel der Geoinformationen, die in der vorherigen Activity zur Verfügung standen, weiterhin zur Verfügung stehen, müssen diese gespeichert werden. Dazu wird in der vorherigen Activity ein sogenanntes Intent erstellt, das zu der neuen Activity gehört. In dem Intent wird ein Bundle erzeugt, das die notwendigen Informationen speichert. Dieses ist vergleichbar mit einer Map, die Key-Value Paare speichert.

```
1 Intent intentLocation = new Intent(this, GeoInformationMap.class);
2   intentLocation.putExtra("article_id",
3     articles.get(getSelectedItemPosition()).articleId);
4   intentLocation.putExtra("article_title",
5     articles.get(getSelectedItemPosition()).title);
6   intentLocation.putExtra("article_latitude",
7     articles.get(getSelectedItemPosition()).latitude);
8   intentLocation.putExtra("article_longitude",
9     articles.get(getSelectedItemPosition()).longitude);
```

Listing 5.13: Erstellung eines Intent zu einer Activity

In der neuen Activity, der MapActivity, gelangt man über den Context an das Intent und so an das Bundle, das die Informationen gespeichert hat.

```
1 Bundle extras = getIntent().getExtras();
2   article.articleId = extras.getLong("article_id");
3   article.title = extras.getString("article_title");
4   article.latitude = extras.getDouble("article_latitude");
5   article.longitude = extras.getDouble("article_longitude");
```

Listing 5.14: Zugriff auf Daten aus einer anderen Aktivität über das Bundle

Auf diese Weise hat man activityübergreifend die Möglichkeit, Daten zur Verfügung zu stellen. In der MapActivity muss eine MapView erzeugt werden. Entsprechend der Anforderungen kann man dieser MapView mehrere Overlays hinzufügen. Auf diesen Overlays lassen sich alle notwendigen Dinge darstellen. In diesem Fall die eigene Position, die der Freunde und die

Geoinformationen. Damit die Positionen (GPS-Koordinaten) dargestellt werden können, müssen der Map neben den Icons und dem Namen auch die Positionen in Form von sogenannten GeoPoints übergeben werden. Zur Anzeige müssen diese allerdings in Bildschirmkoordinaten konvertiert werden. Durch das Aufrufen bereits fertiger Methoden ist es relativ einfach möglich, zu einem bestimmten Punkt auf der Karte zu navigieren, auf der Karte zu scrollen, zu zoomen, zwischen Straßenkarte und Satellitenbild zu wählen und Vergleichbares. Um diese Funktionalitäten benutzen zu können, ist ein zur MapView gehörender MapController notwendig. Der nachfolgende Codeausschnitt veranschaulicht das gerade Beschriebene:

```
1   mapView = new MapView(this, apikey);
2   setContentView(mapView);
3   GeoInformationOverlay geoInformationOverlay =
4     new GeoInformationOverlay();
5   mapController = mapView.getController();
6   mapView.getOverlays().add(geoInformationOverlay);
7   mapController.animateTo(geoPoint);
8   mapController.setZoom(defaultZoomLevel);
9   mapView.setClickable(true);
10  mapView.setEnabled(true);
```

Listing 5.15: Zugriff und Nutzung einer Map

5.2.4. Persistierung

Am Beispiel der Geoinformationen soll die Persistierung verdeutlicht werden. Auf dem Android-Client kommt die bereits vorinstallierte SQLite Datenbank zum Einsatz. Diese Datenbank ist für den mobilen Einsatz besonders geeignet. Folgendes einfaches ER-Modell (Abbildung: 5.3) zeigt, wie die Geoinformationen persistiert werden müssen.

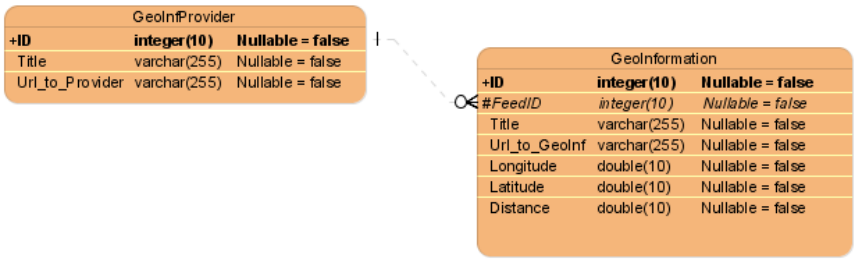

Abbildung 5.3.: ER-Modell für die Persistierung der Geoinformationen

Ein GeoInfProvider hat eine beliebige Anzahl von GeoInformationen. Ein GeoInfprovider ist spezifiziert durch eine eindeutige ID als Primary Key, den Titel des Providers und die URL, die

die XML-Datei mit den Geoinformationen liefert. Die Tabelle Geoinformation hat selbst auch eine ID als Primary Key und als Foreign Key die ID des GeoInfProviders, um jede Geoinformation genau einem Provider zuordnen zu können. Des Weiteren besteht eine Geoinformation aus einem Titel, einer URL zu dem Inhalt des Titels, Längen- und Breitengrad für die Position und derzeitige Entfernung zum Benutzer.
Mittels der Java Klasse SQLiteDatabase lassen sich eine Datenbank und Tabellen anlegen. Die Klasse Cursor hingegen stellt sämtliche Funktionen zum Einfügen, Auslesen und Verändern der Daten in den erstellten Tabellen zur Verfügung.

5.2.5. Bereitstellung

Open Source Software muss anderen bereitgestellt werden, damit sie genutzt und erweitert werden kann. Eine sehr verbreitete Plattform für Open Source Projekte ist Sourceforge. Diese Plattform eignet sich sehr gut zur Publikation von Projekten dieser Art. In diesem konkreten Fall bietet sich eine weitere Möglichkeit an, da bei der Realisierung eine Android Applikation erstellt wird. Google stellt extra einen Marktplatz zur Verfügung, den sogenannten „Android Market“ [Alliance (2008)], um Android Applikationen für andere bereitzustellen. Diese Möglichkeit wird auch für dieses Projekt genutzt. Um Software bereitzustellen, wird ein einfaches Konzept verfolgt, welches aus drei Schritten besteht. In einem ersten Schritt erfolgt eine Registrierung, dann kann man die Software hochladen und beschreiben und in einem letzten Schritt veröffentlichen. Den Entwicklern sollen zusätzliche Unterstützungen geboten werden. Die Android Endgeräte werden mit dem „Android Market“ ausgestattet sein.

5.3. Testkonzept

Das Testen ist ein fundamentaler Baustein, wenn es um die Entwicklung eines Softwareproduktes geht. Diese analytische Maßnahme, bestehend aus einem Testkonzept sowie dem systematischen Erstellen und Durchführen der Tests, wird zur Qualitätssicherung und zum Überprüfen der Anforderungen genutzt. Da es sich hierbei um eine verteilte Anwendung handelt, ist das Testen generell schwierig, da sich beispielsweise Objekte ändern, die man selbst nicht beeinflussen kann. In diesem Fall ist das Testen noch schwieriger, da diese Anwendung kontextabhängig ist. Das bedeutet, dass sich die kontextabhängigen Daten ständig dynamisch ändern und sehr schwer reproduzieren lassen. Das macht ein systematisches Testen sehr schwer. Wie damit umgegangen wird, wird nachfolgend beschrieben.

5.3.1. Unterschiede zum Testen im Unified Process

Wie bereits im ersten Kapitel erwähnt, orientiert sich diese Arbeit an dem Unified Process [Arlow und Neustadt (2005)]. Allerdings gibt es begründete Unterschiede zu diesem Standardvorgehen. Aufgrund der Tatsache, dass diese Arbeit eine Einzelleistung widerspiegelt und nur ein Prototyp entwickelt werden sollte, gibt es keine gesonderte Tester-Rolle. Auch

gibt es noch keine Benutzer, die das System testen können, weil es bisher zum einen nur einen Prototypen gibt und zum anderen noch keine Endgeräte verfügbar sind.
Das Testen im Unified Process ist einer der fünf Kernarbeitsabläufe und zeigt damit die Bedeutung des Testens für die Produktentwicklung. Während das Testen dort bereits in der zweiten Phase, der Entwurfsphase, beginnt, wird in dieser Arbeit davon abgewichen, da das Ziel eine prototypische Realisierung ist. Das Testen findet nur in eingeschränktem Maße bei der prototypischen Entwicklung statt.

5.3.2. Durchgeführte Tests

An dieser Stelle werden die tatsächlich durchgeführten Tests beschrieben. Das darauf folgende Unterkapitel beschreibt, was und wie zusätzlich noch getestet werden muss, aber im Rahmen dieser Arbeit nicht mehr geleistet werden kann.

Entwicklung eines Prototypen: Im Zusammenhang mit dieser Arbeit ist die Realisierung eines Prototypen der erste Test. Dieser Test dient als „Proof of Concept“. Er beweist trotz der Abweichungen zum Konzept zum Beispiel, dass sich die externen Komponenten tatsächlich integrieren und nutzen lassen. Damit konnte bewiesen werden, dass sich die `IM`-Komponente, die `Geoinformation`-Komponente und die `Map`-Komponente zusammen nutzen lassen und sogar neue Funktionalitäten bereitstellen. Dazu wird der nachfolgend beschriebene Mock genutzt.

Mock: Um den Prototypen testen zu können bevor es echte Daten gibt, wird ein Mock erstellt. Das bedeutet, dass echte Daten simuliert werden. Dadurch kann dem großen Problem, der sich dynamisch ändernden kontextabhängigen Daten, etwas entgegentreten.Auf diese Weise lassen sich statische Testdaten erzeugen, die ein systematisches Testen ermöglichen. Android hat dafür bereits extra Mock-Klassen mit entsprechenden Methoden vordefiniert. Mock-Objekte haben den zusätzlichen Vorteil, dass man leicht Objekte erschaffen kann, durch die die Tests fehlschlagen. Auf diese Weise können zuverlässig Fehlermodi getestet und auch Fehlerbehandlungsroutinen erstellt werden. Allerdings ist ein Test mit echten Daten im realen Einsatz trotzdem notwendig.
Zum Testen wurden Test-Buddies und -Geodaten erzeugt. Die erzeugten Test-Buddies bewiesen, dass die `IM`-Komponente in ihren elementaren Zügen funktioniert. Nachdem eine Verbindung zum XMPP-Server hergestellt werden konnte, konnten Nachrichten zwischen den Buddies ausgetauscht werden. Auch der Status der Buddies konnte abgerufen werden. Anhand simulierter Positionsdaten, um welche die Buddies erweitert wurden, konnten diese auf einer Karte angezeigt werden. Dadurch konnte die erfolgreiche Integration der `Map`-Komponente gezeigt werden. Der `User` selbst wurde auch mit Positionsdaten versehen, wodurch sich die Funktionsfähigkeit der `Geoinformation`-Komponente beweisen ließ. Abhängig von der Position des `User` wurden die Geoinformationen in seiner Nähe bezogen und angezeigt.

Funktionsfähigkeit bei Verbindungsabbruch: Auch die partielle Funktionsfähigkeit bei Verbindungsabbruch konnte gezeigt werden. Nachdem der User die Geoinformationsdaten in seiner Nähe erhalten hatte und die Verbindung abbrach, konnte er diese weiterhin anzeigen, da diese in einer Datenbank gespeichert werden.

Logging und Debugging: Zwei weitere Methoden um den Code und seinen Ablauf überprüfen zu können stellen das Logging und das Debugging dar. Auf diese Weise lässt sich überprüfen, ob beispielsweise der Code wie erwartet aufgerufen und ausgeführt wird, oder ob Exceptions geworfen werden. Für das Logging sind von Android bereits verschiedene Level vorgesehen, wodurch das effektive Arbeiten unterstützt wird. Diese sehr elementaren Möglichkeiten kommen in der Regel während der Entwicklung einzelner Komponenten zum Einsatz, bevor diese fertig gestellt sind und weiterführende Tests (Unit Tests, etc.) vollzogen werden können.

5.3.3. Testaufbau

Für zukünftige Tests ist ein strukturierteres und automatisierteres Vorgehen notwendig: Tests sind bei der Entwicklung eines Softwareproduktes in mindestens vier verschiedenen Stufen durchzuführen. Dabei besteht jeder Test aus verschiedenen Testphasen, die sich in Planung, Vorbereitung, Durchführung und Abschluss untergliedern. Um die Tests sinnvoll nutzen bzw. ausführen zu können, müssen diese automatisiert werden. Daher wird die Ausführung der Tests in den Buildprozess integriert.

Unit Test: Ein Unit Test ist ein Test auf unterster Ebene. Es werden Klassen und ihre Methoden getestet. Dazu kommt JUnit in der Version 3 auf dem Android Client zum Einsatz. Allerdings müssen Anpassungen vorgenommen werden, bevor sich die JUnit-Tests durchführen lassen, da in dem `android.jar` nicht die vollständigen JUnit Klassen vorliegen, sondern lediglich Stub Klassen. Ist das allerdings geschehen, lassen sich auf diese Weise die einzelnen Methoden überprüfen, ob sie die erwarteten Ergebnisse liefern und wie sie reagieren, wenn falsche Werte übergeben werden.

Integrationstest: Haben die Klassen den Unit Test erfolgreich bestanden, können die Komponenten im Zusammenspiel getestet werden. Dazu ist für jede Abhängigkeit zwischen zwei Komponenten eines Systems ein Testszenario zu entwickeln. Dieses Szenario muss sicherstellen, dass die Zusammenführung der beiden Komponenten als auch der Datenaustausch entsprechend der Spezifikation der Schnittstellen abläuft. Dazu werden Funktionstest als auch Schnittstellentests durchgeführt. In diesem Fall wurden die Mockdaten benutzt, um zu überprüfen, ob die `mucM`-Komponente mit den anderen Schnittstellen funktioniert.

Systemtest: In dieser Testphase wird das gesamte System auf seine Anforderungen hin überprüft. Das gilt sowohl für die funktionalen als auch für die nicht-funktionalen Anforderungen. In der Regel finden diese Tests lediglich mit Testdaten in einer Testumgebung statt. An dieser Stelle bietet es sich auch an Lasttests durchzuführen. Dabei ist es wichtiger die Stabilität als die Performance zu testen. Auf diese Weise lassen sich Deadlocks, Memoryleaks oder Dateninkonsistenzen finden. Zur Erstellung dieser Lasttests können die Unit-Tests wiederverwendet werden.

Abnahmetest: Der Abnahmetest bedeutet das Testen der ausgelieferten Software durch den Kunden. Dieser Test kann bzw. soll auf einem Produktivsystem mit echten Daten erfolgen. In diesem Fall gibt es keinen konkreten Kunden. Allerdings kann die Community als Kunde angesehen werden. Die gesamte Community verfügt in dem Fall selbst über echte individuelle Testdaten und kann damit den Messenger testen. Fehler können auf diese Weise manuell berichtet werden, sollten aber auch automatisch erstellt und an die Entwickler weitergeleitet werden.

5.3.4. Weitere Arten des Testens

Bei der Entwicklung einer Open Source Software bietet es sich an, bevor Änderungen eingecheckt werden, durch ein qualifiziertes Mitglied der Community ein Review durchzuführen. Zum Beispiel bei Mozilla ist das der Fall. Eine weitere Möglichkeit, die Mozilla auch nutzt, ist das Feedback der Nutzer. Es werden sogenannte Bugreports erzeugt, die automatisch die auftretenden Fehlermeldungen an das Entwicklerteam schicken. Eine weitere übliche Methode bei Open Source Projekten ist es ein frühes Release zu veröffentlichen und bewusst die User für das Melden von Fehlern einzubinden. Das wurde beispielsweise bei dem neuen Browser namens Chrome von Google so gemacht.

5.4. Fazit

Es gibt zwei Seiten der Realisierung, die zu betrachten sind. Im Sinne der Produktentwicklung befindet sich die Realisierung noch am Anfang. Es gibt viele Möglichkeiten der Verbesserung und auch der Integration weiterer Funktionalitäten. Das Konzept mit seinen Anforderungen konnte wegen des Umfanges und der technischen Voraussetzungen (Fehlen der Endgeräte) nicht eins-zu-eins umgesetzt werden, aber es gibt auch die andere Seite. Dieses Kapitel hat den Zweck, durch die Realisierung das Konzept zu beweisen, was gelungen ist. Es konnte genug umgesetzt werden, um zu zeigen, dass unter den aktuellen Voraussetzungen eine vollständige Realisierung des Konzeptes möglich ist. Sehr wahrscheinlich ist auch die Funktionsfähigkeit auf einem Endgerät, die allerdings nicht bewiesen werden konnte.

6. Evaluierung

In der Evaluierung wird diese Arbeit kritisch bewertet. Dabei wird beschrieben, ob die Anforderungen im Konzept und in der Realisierung erfüllt werden konnten oder ob es Mängel in einem der Bereiche gibt. Wenn es Mängel gibt, soll versucht werden diese durch alternative Lösungsansätze zu beheben.

6.1. Umsetzung der Anforderungen

An dieser Stelle wird überprüft, in welchem Maß die funktionalen und nicht-funktionalen Anforderungen sowohl im Konzept als auch in der Realisierung umgesetzt wurden.

6.1.1. Umsetzung der funktionalen Anforderungen

Das Use Case Diagramm 3.1 enthält alle grundsätzlichen funktionalen Anforderungen und wird daher als Referenz herangezogen.

6.1.1.1. Akteure

In diesem Diagramm werden vier Akteure aufgezeigt, die sich so direkt nicht im Konzept wiederfinden lassen.

Consumer, Author: Diese beiden Aktuere werden nicht unterschieden, weil der Übergang vom reinen Consumer zum Author fließend stattfinden soll. Wie zuvor erwähnt, sollen möglichst viele Consumer zu Authoren werden. Daher werden diese beiden Rollen im Konzept nicht unterschieden.

Programmer, Wizard: Der Programmer und der Wizard spielen bei der Konzepterstellung insofern eine Rolle, dass beide die Möglichkeit haben sollen den mucM erweitern zu können. Daher ist dieses Projekt zum einen Open Source, um Zugriff auf den Quelltext haben zu können. Zum anderen wurde besondere Rücksicht auf die komponentenbasierte Entwicklung gelegt. Auf diese Weise können unabhängig voneinander einzelne Bereiche gegen die Schnittstellen entwickelt werden.

6.1.1.2. Use Cases

Bezüglich der einzelnen Use Cases lässt sich Folgendes sagen:

Registrierung durchführen: Dieser Use Case findet implizit über die externen Systeme statt, die einen User-Account benötigen. Der User kann sich bei den einzelnen Systemen direkt registrieren.

Messenger einloggen: Das einloggen funktioniert so, dass sich der User mit einem Account aus der `Communication`-Komponente einloggen kann.

Nachrichten austauschen, Buddy anrufen, Annotation anlegen, Annotation erweitern: Ist der User eingeloggt, kann dieser den Anforderungen entsprechend diese Use Cases ausführen.

Karte anzeigen, Annotation anschauen, zu Position navigieren: Diese Use Cases sind ebenfalls im Konzept umgesetzt worden.

In der Realisierung wurden prototypisch die Use Cases „Messenger einloggen“, „Nachrichten austauschen“, „Annotation anschauen“, „Annotation anlegen“, „Annotation erweitern“, sowie „Karte anzeigen“ umgesetzt.

6.1.2. Umsetzung der nicht-funktionalen Anforderungen

Dieser Abschnitt beschreibt, welche nicht-funktionalen Anforderungen umgesetzt wurden und an welchen Stellen es Unterschiede zu den Anforderungen gibt.

6.1.2.1. Technische Anforderungen

Den Anforderungen entsprechend wurde Java als „Programmiersprache“ gewählt. Die „Geräteunabhängigkeit“ wurde durch die Auslagerung der meisten Logik auf einen Server erreicht, so dass die Entwicklung verschiedener Clients für andere Endgeräte relativ einfach möglich ist. Zusätzlich werden Schnittstellen für die verschiedenen Clients bereitgestellt. In Zukunft wird es verschiedene Smartphones von verschiedenen Herstellern geben, die die Android Plattform nutzen. Es sind alle Komponenten „Open Source“, mit Ausnahme der `Map`-Komponente, die allerdings weitestgehend ohne Beschränkungen genutzt werden kann und notfalls auch durch eine andere `Map`-Komponente ersetzt werden kann. Durch die `Communication`-Schnittstelle wurde es ermöglicht, bei der Erstellung der verschiedenen Accounts auch verschiedene Protokolle nutzen zu können. Damit wurde die Anforderung an den „Datenaustausch“ erfüllt. Zu den Anforderungen an die „Software“ lässt sich sagen, dass das genutzte Framework noch nicht als Open Source vorliegt, aber das soll noch dieses Jahr geschehen. Ebenfalls wurde der mucM als Service realisiert und durch die Kapselung in der `Communication`-Komponente die Möglichkeit eröffnet beliebige Netzwerke einzubinden.

Die „Systemarchitektur“ wurde wie erwartet sehr modular durch Komponenten erstellt, wodurch die notwendige Flexibilität erreicht wurde. Die Umsetzung der Anforderungen an den „Benutzer“ wurden bereits bei den funktionalen Anforderungen beschrieben.

6.1.2.2. Anforderungen an die Benutzungsoberfläche

Diese Anforderungen konnten soweit noch nicht getestet werden, da der Prototyp noch nicht weit genug entwickelt wurde. Das muss entsprechend nach weiterer Entwicklung nachgeholt werden.

6.1.2.3. Qualitätsanforderungen

Die Qualitätsanforderungen können bisher nur teilweise überprüft werden. „Funktionalität 1“ wurde noch nicht getestet. „Funktionalität 2“ wurde im Konzept umgesetzt und teilweise bereits erfolgreich realisiert. Sichere Aussagen zur „Zuverlässigkeit“ und „Effizienz“ können ebenfalls noch nicht getroffen werden, da es noch keine Tests im realen Umfeld gibt. Die Anforderungen an die „Änderbarkeit“ wurden im Konzept umgesetzt. Der erste Teil der „Anforderungen an den Schutz der Systemumgebung und an den Schutz des Systems“ wurde noch nicht umgesetzt, lässt sich aber als weitere Komponente integrieren. Der zweite Teil wurde erfolgreich in das Konzept integriert. Eine gute „Wartbarkeit“ ist durch die komponentenbasierte Entwicklung möglich. Die „Testbarkeit“ wurde in der Realisierung beschrieben und damit ebenfalls berücksichtigt.

6.1.2.4. Sicherheit

Die Anforderungen an die Sicherheit wurden für diese Arbeit ausgeklammert, allerdings wurde eine verschlüsselte Verbindung während des Nachrichtenaustausches erfolgreich integriert.

6.1.3. Tabellarischer Vergleich der Anforderungen

Die Tabelle 6.1 aus dem Vergleich der Projekte (siehe Abschnitt 3.4.5) veranschaulicht, welche Funktionalitäten der mucM bieten soll, welche davon bereits im Konzept umgesetzt wurden und was prototypisch realisiert wurde. Vergleicht man die Spalte der „Anforderungen“ mit der Spalte „Konzept“ ist zu sehen, dass die Anforderungen im Konzept tatsächlich berücksichtigt werden konnten. Aufgrund der begrenzten Zeit ist in der Spalte „Realisierung“ zu sehen, dass nicht alle Anforderungen realisiert werden konnten. SMS, Telefonie und soziale Netzwerke konnten aus zeitlichen Gründen nicht umgesetzt werden. Das Instant Messaging konnte erfolgreich realisiert werden, wobei nur das XMPP-Protokoll exemplarisch umgesetzt wurde, um mit Jabber-Clients zu kommunizieren. Annotationen lassen sich erstellen und anzeigen. Die Positionen der Buddies können auf einer Karte angezeigt werden. Da keine besondere Infrastruktur oder ähnliches notwendig ist, lässt sich der mucM weltweit einsetzen. Allerdings wird noch keine Mehrsprachigkeit unterstützt. Die Plattformunabhängigkeit und

	Anforderungen	Konzept	Realisierung
SMS	x	x	-
Instant Messaging	x	x	x
Telefonie	x	x	-
Soziale Netzwerke	x	x	-
Verschiedene Protokolle	x	x	(x)
UGC (Annotation)	x	x	x
Verortung der Buddies	x	x	x
Weltweit nutzbar	x	x	x
Plattformunabhängigkeit	x	x	x
Verortung	x	x	x
Offene API	x	x	(x)
Open Source	x	x	x

Tabelle 6.1.: Umsetzung der Anforderungen

die Verortung wurden ebenfalls umgesetzt. Die offene API steht noch nicht zur Verfügung. Der Prototyp muss erst weiter entwickelt werden, um sicherzustellen, dass die Schnittstellen, so wie sie definiert sind, auch bleiben können. Der Code des Prototypen wird entsprechend der Anforderungen Open Source sein und damit zur Weiterentwicklung zur Verfügung stehen.

6.2. Mögliche Verbesserungen

Fast immer gibt es, nachdem etwas erarbeitet wurde, Möglichkeiten etwas zu verbessern. Damit beschäftigt sich dieser Abschnitt zuerst im Detail und abschließend bezogen auf die gesamte Masterarbeit.

Innenansicht der fachlichen Komponenten: Für die Konzepterstellung bei der Innenansicht der fachlichen Komponenten wurde aufgrund der Hoffnung besserer Performance die Komponentendefinition etwas aufgeweicht. Im Nachhinein ist der größere Nutzen im Vergleich zu den Nachteilen, die diese Entscheidung gerade für die Entwicklung in einer Open Source Community mit sich bringt, nicht eindeutig. Sinnvoll wäre ein Performance Test, für die Variante aus dem Konzept und die Variante der strikten Einhaltung der Komponentendefinitionen. Ist kein deutlicher Unterschied zu erkennen, sollte die komponentenbasierte Entwicklung auch an dieser Stelle noch umgesetzt werden. Demnach würden die Komponenten `Messenger`, `Position`, `MessengerPosition` und `Communication` lose gekoppelt sein und eine separate Entwicklung gegen definierte Schnittstellen erleichtern. Erst in diesen Komponenten, genauer gesagt Modulen, findet die objektorientierte Entwicklung ihre Anwendung.

Technische Schnittstellen: Die technischen Schnittstellen zeigen sehr vereinfachte und allgemeine Schnittstellen zu den externen Systemen. Betrachtet man beispielsweise das Map-Interface so gibt es nur eine sehr allgemeine Schnittstelle zum Anzeigen mehrerer Positionen und einen weiteren Parameter, der prinzipiell alles enthalten kann. Tatsächlich bieten die verschiedenen Kartenanbieter deutlich komplexere Schnittstellen an. Daher müsste bei einer Weiterentwicklung die Überlegung angestellt werden, ob die Schnittstelle nicht konkretisiert und weitere Methoden hinzugefügt werden sollten. Diese Überlegung ist bei den anderen Schnittstellen ebenfalls anzustellen.

Realisierung: Da bisher nur ein Prototyp realisiert wurde, besteht insgesamt eine Menge von Verbesserungsvorschlägen. Ein Bereich soll dabei besonders betrachtet werden:
Dabei handelt es sich um die `Geoinformation`-Komponente, die mit einem Wikipedia Webservice realisiert wurde. Das Besondere bei dieser Realisierung ist die Simplizität. Zum einen kann der Service mit ein paar Parameterübergaben ziemlich einfach aufgerufen und an persönliche Vorlieben angepasst werden (Umkreis der Annotationen, Sprache, Anzahl der Annotationen, etc.). Zum anderen ist durch einen kleinen Trick sehr viel Arbeit erspart und ausgelagert worden. Beim Aufruf einer Annotation wird man auf die Wikipedia-Seite weitergeleitet, die in einem Browser angezeigt wird. Dadurch hat man automatisch die Möglichkeit, sich bei Wikipedia einzuloggen und die Annotation zu erweitern oder auch neue Annotationen zu erstellen. Das Ganze geschieht genau wie vom PC aus. Für diese geforderte Funktionalität musste nichts weiter getan werden, als die Weiterleitung auf die entsprechende Webseite.
Allerdings besteht auch hier die Überlegung, die Komponente in Zukunft stärker zu integrieren. Dadurch könnte man erreichen, dass nicht zusätzlich der Browser gestartet werden muss, der weitere Ressourcen in Anspruch nimmt. Vor allem ließen sich, bei einer stärkeren Integration in den mucM, während der Annotationserstellung beispielsweise die Koordinaten gleich mit übertragen.

Test: Das Ziel des Testens ist es Fehler zu finden, um die Qualität und die Benutzbarkeit zu erhöhen. Neben dem beschriebenen Testkonzept, dass die Grundlage für eine gute und stabile Software bildet, gibt es noch weitere Tests, die beispielsweise die Benutzbarkeit (Usability) testen. Die Benutzbarkeit ist für diese Art von Software besonders wichtig. Mithilfe des Mocks oder mit echten Nutzern, muss zum Beispiel getestet werden, bei wievielen Freunden oder Annotationen aktuelle Anzeigemöglichkeiten auf einem Smartphone keinen Sinn mehr machen. Befinden sich sehr viele Freunde oder Annotationen in der direkten Umgebung des Users, findet man vermutlich heraus, dass eine Anzeige auf einer Karte nicht mehr sinnvoll ist. Ist dies der Fall, müssen Alternativen gefunden werden. Eine Möglichkeit ist das Einblenden verschiedener Overlays, auf denen nur bestimmte Gruppierungen angezeigt werden, zum Beispiel nur die Freunde vom Sport oder nur die Annotationen, die der Kategorie Sehenswürdigkeiten angehören. Durch diese Art von Tests ergibt sich eventuell, dass das gewählte

Vorgehen bei der Entwicklung des Software nicht das Beste war. Auf diesen Aspekt wird im nächsten Abschnitt eingegangen.

Grundsätzliche Verbesserungen: Der Unified Process wurde in modifizierter Form als Grundlage für Erstellung der Masterarbeit genutzt. Dieser Prozess hat sich als geeignet erwiesen. Allerdings besteht die Möglichkeit, dass in diesem Fall eine andere Vorgehensweise, insbesondere, wenn aus dieser Arbeit ein Produkt entstehen soll, geeigneter ist. Der Nutzer steht in besonderem Maße im Vordergrund bei dieser Anwendung. Zum einen ist dieser Messenger ubiquitär und somit immer im Zugriff des Nutzers, dabei ist eine problemlose Nutzung des Programmes im Alltag unerlässlich. Zum anderen wird das Programm nicht nur benutzt, sondern es werden aktiv Inhalte erschaffen und das Programm kann sogar weiterentwickelt werden. Daher erscheint ein benutzerzentrierter Ansatz in dieser Situation sinnvoller. Das sogenannte „User Centered Design“ stellt von Anfang an den Benutzer in den Vordergrund, um eine sehr hohe Benutzbarkeit (Usability) erreichen zu können.

6.3. Fazit

Insgesamt lässt sich zur Evaluierung sagen, dass alle grundlegenden Anforderungen im Konzept erfüllt worden sind. Wobei die, die noch nicht erfüllt werden konnten, noch ausstehen und wahrscheinlich später auch erfüllt werden können. Die Realisierung zeigt, dass sich die wichtigsten und schwierigsten Bereiche umsetzen lassen. Damit lässt sich abschließend sagen, dass trotz einiger Möglichkeiten zur Verbesserung, die prinzipielle Durchführbarkeit (Proof of Concept) belegt werden konnte.

7. Zusammenfassung

Zum Abschluss dieser Arbeit wird ein Fazit erstellt und ein Ausblick gegeben.

7.1. Fazit

Das Ziel dieser Arbeit war es, einen ubiquitären kontextabhängigen Messenger mit einem Schwerpunkt auf Open Source zu konzipieren, realisieren und evaluieren. Dazu mussten verschiedene Systeme, die sonst für sich einzeln existieren und genutzt werden, miteinander verbunden werden. Der Anspruch dieser Arbeit besteht nicht nur darin diese verschiedenen Systeme zu verknüpfen, sondern auch gleichzeitig die besonderen Randbedingungen zu berücksichtigen. Diese Randbedingungen bestehen zum einen aus der Mobilität, die eingeschränkte Ressourcen, Akkulaufzeit, etc. mit sich bringt. Zum anderen sind es die Anforderungen, die erfüllt werden müssen, um den mucM als Open Source Projekt nutzen zu können. Ein sehr wichtiger Faktor dabei ist die Modularität. Diese ist notwendig, um unterschiedlich definierte Komponenten zu haben, die abgesehen von der gemeinsamen Schnittstelle unabhängig voneinander entwickelt werden können.
Exemplarisch wurde die Android Plattform gewählt, die aufgrund der Tatsache, dass diese Plattform gerade erst entwickelt wird, den hohen Anforderungen sehr gut entspricht. Mittlerweile ist die Version 1.0 des SDK verfügbar. Das erste Endgerät soll von der Firma HTC noch im Oktober 2008 in den USA erscheinen. Den veröffentlichten Informationen nach ist dieses Gerät für den ubiquitären Einsatz geeignet und verfügt über verschiedene Sensoren / Empfänger, wie beispielsweise GPS, um Context Awareness zu ermöglichen. Zusätzlich ist Android ein Open Source Projekt und passt damit sehr gut in die Anforderungen. Der Quellcode soll mit dem ersten Gerät veröffentlicht werden.
Das Konzept wurde so allgemein gehalten, dass es sich problemlos auf andere Systeme portieren lässt. Der Server bietet eine einheitliche Schnittstelle nach außen und beinhaltet beinahe die gesamte Logik. Dadurch werden die Clients so wenig wie möglich belastet und die vom Server bereitgestellte Funktionalität kann einfach wiederverwendet werden.
Die Realisierung weicht zwar leicht vom Konzept ab, aber zeigt dennoch, dass das Konzept den Anforderungen entspricht und realisierbar ist. Es wurde gezeigt, dass das Instant Messaging mit Presence Information funktioniert, dass die Buddies mit Positionen ausgestattet werden können und dass sich Annotationen zum Aufenthaltsort des Users anzeigen und verändern lassen. Grafisch können die Positionen auf einer Karte dargestellt werden.
Die Evaluierung ergab, dass der Machbarkeitsbeweis (Proof of Concept) erbracht werden

konnte, aber gleichzeitig auch noch viele Möglichkeiten der Verbesserung bleiben.
Damit lässt sich abschließend sagen, dass auf dem Emulator ein funktionstüchtiger Prototyp des übiquitären kontextabhängigen Messengers erfolgreich erstellt werden konnte.

7.2. Ausblick

Da bei dieser Arbeit nur ein Prototyp realisiert wurde, besteht der nächste Schritt darin das Konzept weiter umzusetzen. Als erstes muss die Serverkomponente entwickelt werden, die im Zuge der prototypischen Entwicklung erst einmal zurückgestellt wurde (momentan wird die gesamte Logik auf dem Android-Client ausgeführt). Dazu ist es sinnvoll im Sinne des Open Source Gedanken das Projekt und den Sourcecode auf einer öffentlichen Plattform Anderen bereitzustellen. Zum Bereistellen des Produktes eignet sich der in Kapitel 5.2.5 beschriebene Android Market sehr gut. Nach derzeitigen Informationen erscheint der Android Market aber keine Plattform für die Entwicklung eines Open Source Projektes zu bieten. Demnach bietet sich Sourceforge als sehr bekannte Open Source Plattform an. Allerdings ist es mit der reinen Erstellung eines Open Source Projektes noch nicht getan. Das Projekt muss publik gemacht werden, damit um das Projekt herum eine Community entstehen kann. Eine Möglichkeit dieses zu erreichen, ist eine gute Betaversion auf dem Android Market mit dem Hinweis auf Sourceforge zu veröffentlichen. Zusätzlich kann man das Projekt in verschiedenen Foren zu diesem Thema veröffentlichen und auf die Wirksamkeit des viralen Marketings hoffen. Bei der Erstellung des Open Source Projektes muss sich für eine Lizenz entschieden werden. Dabei ist es ratsam sich für eine der bekannten Lizenzen zu entscheiden, die von Google zugelassen sind. Um eine endgültige Entscheidung treffen zu können, müssen diese allerdings ausführlicher betrachtet werden.
Ein weitere Aufgabe die erledigt werden muss, ist die Anpassung des Prototypen in der Version 0.9 auf die aktuelle Version 1.0. Nach einem kurzen Einblick in die Änderungen, ist kein großer Aufwand nötig, um dies zu erreichen.
Bei der weiteren Realisierung müssen die Schnittstellen zu den externen Systemen verifiziert werden. Dabei soll überprüft werden, ob mit diesen Schnittstellen auch andere Systeme mit gleicher Funktionalität integriert werden können. Ein Beispiel ist Yahoo-Maps im Austausch mit Google-Maps.
Zu guter letzt muss der Schritt vom Emulator auf das Endgerät vollzogen werden. Auf bisherigen Erfahrungen basierend, wird diese Aufgabe einige Zeit in Anspruch nehmen. Allerdings besteht die Hoffnung, dass es nicht allzu viele Probleme gibt, wenn das Betriebssystem stabil läuft. Der Grund für diese Hoffnung ist die gute Kapselung, denn die Anwendung baut auf dem Application Framework auf, welches in das Betriebssystem integriert ist.

A. Anhang

A.1. Use-Case Beschreibung

Use Case	**Nachrichten austauschen**
Beschreibung	Der Benutzer chattet mit jemandem aus seiner Buddyliste.
Akteure	Consumer, Author, Programmer, Wizard
Vorbedingungen	Der Benutzer muss eingeloggt sein. Es muss mindestens eine Internetverbindung des Benutzers vorhanden sein, um die Nachricht abschicken zu können.
Normalablauf	1. Der Benutzer wählt einen Buddy aus. 2. Der Benutzer verfasst die Nachricht. 3. Der Benutzer schickt die Nachricht ab.
Nachbedingungen	Die Nachricht wird übertragen.
Fehlerfall	Ist die Internetverbindung bei dem Benutzer oder dem Buddy unterbrochen, wird die Nachricht übertragen, sobald die Internetverbindung wieder hergestellt ist.

Tabelle A.1.: Use Case: Nachrichten austauschen

Use Case	**Buddy anrufen**
Beschreibung	Der Benutzer ruft jemandem aus seiner Buddyliste an.
Akteure	Consumer, Author, Programmer, Wizard
Vorbedingungen	Der Benutzer muss eingeloggt sein. Es muss eine Verbindung der beiden Teilnehmer hergestellt werden. Der Buddy muss das Gespräch annehmen.
Normalablauf	1. Der Benutzer wählt einen Buddy aus. 2. Der Benutzer ruft den Buddy an. 3. Der Buddy nimmt den Anruf entgegen.
Nachbedingungen	Die Kommunikation per Sprache kann stattfinden.
Fehlerfall	Das Telefonat kann nicht hergestellt werden, wenn einer der beiden nicht online ist oder bereits telefoniert. Dies wird dem Benutzer mitgeteilt. Er muss es zu einem späteren Zeitpunkt erneut versuchen.

Tabelle A.2.: Use Case: Buddy anrufen

Use Case	**Karte anzeigen**
Beschreibung	Der Benutzer lässt sich eine Karte anzeigen. Initial ist die Position des Benutzers der Mittelpunkt der Karte.
Akteure	Consumer, Author, Programmer, Wizard
Vorbedingungen	Es ist eine Position des Benutzers bekannt. Es besteht eine Internetverbindung.
Normalablauf	1. Der Benutzer wählt „Karte anzeigen“ aus. 2. Der Benutzer wird als Mittelpunkt auf der Karte angezeigt. 3. Die Annotationen auf dem Kartenausschnitt werden angezeigt. 4. Buddies in der Nähe werden angezeigt. 5. Der Benutzer kann eine Annotation oder einen Buddy direkt auswählen.
Nachbedingungen	Die gesamte Funktionalität der Karte kann genutzt werden.
Fehlerfall	1. Die Karte wird zur Laufzeit heruntergeladen. Gelingt dies nicht, weil keine Internetverbindung besteht, wird der Benutzer darüber informiert. Ist die Internetverbindung wieder hergestellt, wird die Karte heruntergeladen und angezeigt. 2. Ist keine aktuelle Position des Benutzers vorhanden, wird initial die letzte bekannte Position genommen.

Tabelle A.3.: Use Case: Karte anzeigen

Use Case	**Zu Position navigieren**
Beschreibung	Der Benutzer gibt eine Position an und wird dorthin navigiert.
Akteure	Consumer, Author, Programmer, Wizard
Vorbedingungen	Die Position des Benutzers ist bekannt. Es besteht eine Internetverbindung.
Normalablauf	1. Der Benutzer wählt ein Ziel (Adresse, Buddy, Annotation) aus. 2. Die Route wird vom System berechnet. 3. Der Weg zum Ziel wird auf der Kartenansicht angezeigt.
Nachbedingungen	Der Benutzer wird zu seinem Ziel geführt.
Fehlerfall	1. Ist die Internetverbindung unterbrochen bevor die Route heruntergeladen werden konnte, wird die Navigation nicht gestartet. 2. Wird die Route plötzlich geändert und es besteht keine Internetverbindung, kann die Route nicht neu berechnet werden. 3. Ist keine aktuelle Position des Benutzers vorhanden, bleibt die Navigation an der letzten bekannten Position stehen.

Tabelle A.4.: Use Case: Zu Position navigieren

Use Case	**Annotation anschauen**
Beschreibung	Der Benutzer betrachtet eine Annotation in seiner Nähe.
Akteure	Consumer, Author, Programmer, Wizard
Vorbedingungen	Die Position des Benutzers ist bekannt. Es besteht eine Internetverbindung.
Normalablauf	1. Der Benutzer wählt Annotation aus. 2. Der Benutzer schaut sich die Informationen über die Annotation an.
Nachbedingungen	Der Benutzer erhält Zusatzinformationen zu der ausgewählten Annotation.
Fehlerfall	1. Wechselt der Benutzer seine Position und hat keine Internetverbindung, können die Zusatzinformationen der Annotationen in der Nähe des Benutzers nicht nachgeladen werden. Der Benutzer erhält erst die neuen Zusatzinformationen, wenn die Internetverbindung wieder besteht. 2. Ist keine aktuelle Position des Benutzers bekannt, werden auch keine neuen Zusatzinformationen zu neuen Annotationen heruntergeladen. Dies geschieht erst, wenn die Position aktualisiert wurde.

Tabelle A.5.: Use Case: Annotation anschauen

Use Case	**Annotation anlegen**
Beschreibung	Der Benutzer legt eine neue Annotation an.
Akteure	Author, Programmer, Wizard
Vorbedingungen	Die Position des Benutzers ist bekannt. Es besteht eine Internetverbindung. An der Position gibt es noch keine Annotation.
Normalablauf	1. Der Benutzer stellt Informationen zu einer Annotation zusammen. 2. Der Benutzer begibt sich an die Position der Annotation. 3. Der Benutzer lädt die Informationen hoch.
Nachbedingungen	Der Benutzer hat eine neue Annotation erstellt.
Fehlerfall	1. Ist zum Zeitpunkt des Hochladens keine Internetverbindung vorhanden, kann die neue Annotation auch nicht angelegt werden. 2. Ist keine aktuelle Position vorhanden, wartet das System, bis eine aktuelle Position vorhanden ist.

Tabelle A.6.: Use Case: Annotation anlegen

Use Case	**Annotation erweitern**
Beschreibung	Der Benutzer ändert eine bereits existierende Annotation.
Akteure	Author, Programmer, Wizard
Vorbedingungen	Es besteht eine Internetverbindung. Der Benutzer hat eine Annotation ausgewählt.
Normalablauf	1. Der Benutzer wählt eine existierende Annotation aus. 2. Der Benutzer passt die Informationen an. 3. Der Benutzer lädt die Informationen hoch.
Nachbedingungen	Der Benutzer hat eine bestehende Annotation erweitert.
Fehlerfall	1. Ist während des Hochladens keine Internetverbindung vorhanden, kann die Annotation nicht erweitert werden. 2. Hat vorher jemand anderes die Annotation verändert, kann die Annotation erst nach einem Update verändert werden, um Inkonsistenzen zu verhindern.

Tabelle A.7.: Use Case: Annotation erweitern

Use Case	**Plugin entwickeln**
Beschreibung	Der Benutzer erweitert die Funktionalität des mucM.
Akteure	Programmer, Wizard
Vorbedingungen	Der Benutzer hat Erfahrung mit dem Plugindevelopment. Das Plugin lässt sich mit der API verwirklichen.
Normalablauf	1. Dem Benutzer fällt eine fehlende Funktionalität auf. 2. Der Benutzer entwickelt das Plugin. 3. Der Benutzer stellt anderen das Plugin zur Verfügung.
Nachbedingungen	-
Fehlerfall	-

Tabelle A.8.: Use Case: Plugin entwickeln

Use Case	**Software entwickeln**
Beschreibung	Der Benutzer möchte an der Entwicklung des mucM teilnehmen.
Akteure	Wizard
Vorbedingungen	Der Benutzer hat Erfahrung mit Softwaredevelopment in einer Community. Der mucM ist modular aufgebaut.
Normalablauf	1. Die Community stimmt über die nächsten Aufgaben ab. 2. Der Benutzer entwickelt den mucM im Team weiter. 3. Ein neues Release des mucM wird den Benutzer bereitgestellt.
Nachbedingungen	-
Fehlerfall	-

Tabelle A.9.: Use Case: Software entwickeln

A.2. Geoinformation

A.2.1. Webservice XML-Datei

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<geonames>

<entry>
<lang>en</lang>
<title>St. Nikolai, Hamburg</title>
<summary>The Gothic Revival St. Nikolai´s Church (German: St.-
   Nikolai-Kirche) was formerly one of the five Lutheran ''
   Hauptkirchen'' (main churches) in (...)</summary>
<feature>landmark</feature>
<countryCode>DE</countryCode>
<population>0</population>
<elevation>0</elevation>
<lat>53.5475</lat>
<lng>9.9906</lng>
<wikipediaUrl>http://en.wikipedia.org/wiki/St._Nikolai%2C_Hamburg
   </wikipediaUrl>
<thumbnailImg>http://www.geonames.org/img/wikipedia/55000/thumb
   -54786-100.jpg</thumbnailImg>
<distance>0.2803863296919791</distance>
</entry>

<entry>
<lang>en</lang>
<title>St. Petri, Hamburg</title>
<summary>, Germany. It is named after the Christian Apostle Peter,
    who the Catholic Church believes to be the first Pope. The
   church is located on Mönckebergstrasse and marks the highest
   point iAnnotationamburg´s Old Town. (...)</summary>
<feature>landmark</feature>
<countryCode>DE</countryCode>
<population>0</population>
<elevation>0</elevation>
<lat>53.5503</lat>
<lng>9.9964</lng>
<wikipediaUrl>http://en.wikipedia.org/wiki/St._Petri%2C_Hamburg</
   wikipediaUrl>
<thumbnailImg>http://www.geonames.org/img/wikipedia/145000/thumb
   -144424-100.jpg</thumbnailImg>
<distance>0.42317768710950754</distance>
</entry>
```

```
33
34  </geonames>
```

Listing A.1: Ergebnis des Webservice Aufrufs

A.2.2. Parsen mit dem SAX-Parser

```
1  public void createFeed(Context ctx, URL url, String title) {
2          Log.d(LOG_TAG, "createFeed-start");
3          try {
4              //defining that where are looking for a feed
5              targetFlag = TARGET_FEED;
6              geoInformationDB = new GeoInformationDB(ctx);
7              currentFeed.url = url;
8              currentFeed.title = title;
9              //creating a SAXParserFactory to create a SAXParser to
                   create a XMLReader!
10             SAXParserFactory spf = SAXParserFactory.newInstance();
11             SAXParser sp = spf.newSAXParser();
12             XMLReader xr = sp.getXMLReader();
13             //setting this class as ContentHandler and passing the
                   xml
14             //stream to the XMLReader using the supplied url.
15             xr.setContentHandler(this);
16             xr.parse(new InputSource(url.openStream()));
17
18         } catch (IOException e) {
19             Log.e(LOG_TAG, "createFeed" + e.toString());
20         } catch (SAXException e) {
21             Log.e(LOG_TAG, "createFeed" + e.toString());
22         } catch (ParserConfigurationException e) {
23             Log.e(LOG_TAG, "createFeed" + e.toString());
24         }
25         Log.d(LOG_TAG, "createFeed-end");
26     }
```

Listing A.2: Initialisierung des Sax-Parsers

```
1      public void startElement(String uri, String name, String qName
           ,
2              Attributes atts) {
3          Log.d(LOG_TAG, "startElement-start");
4          if (name.trim().equals("title"))
```

```
            inTitle = true;
        else if (name.trim().equals("entry"))
            inItem = true;
        else if (name.trim().equals("wikipediaUrl"))
            inLink = true;
        else if (name.trim().equals("lat"))
            inLat = true;
        else if (name.trim().equals("lng"))
            inLng = true;
        else if (name.trim().equals("distance"))
            inDistance = true;
        Log.d(LOG_TAG, "startElement-end");
    }
```

Listing A.3: Suchen und Markieren des entsprechenden Start-Tags

```
    public void endElement(String uri, String name, String qName)
            throws SAXException {
        Log.d(LOG_TAG, "endElement-start");
        if (name.trim().equals("title"))
            inTitle = false;
        else if (name.trim().equals("entry"))
            inItem = false;
        else if (name.trim().equals("wikipediaUrl"))
            inLink = false;
        else if (name.trim().equals("lat"))
            inLat = false;
        else if (name.trim().equals("lng"))
            inLng = false;
        else if (name.trim().equals("distance"))
            inDistance = false;
        // Check if looking for feed, and if feed is complete
        if (targetFlag == TARGET_FEED && currentFeed.url != null
                && currentFeed.title != null) {
            // We know everything we need to know, so insert feed
               and exit
            geoInformationDB.insertFeed(currentFeed.title,
               currentFeed.url);
            throw new SAXException();
        }
        // Check if looking for article, and if article is
           complete
        if (targetFlag == TARGET_ARTICLES && currentArticle.url !=
            null
                && currentArticle.title != null) {
```

```
26              geoInformationDB.insertArticle(currentFeed.feedId,
                    currentArticle.title,
27                      currentArticle.url, currentArticle.latitude,
28                      currentArticle.longitude, currentArticle.
                            distance);
29              currentArticle.title = null;
30              currentArticle.url = null;
31              // Lets check if we've hit our limit on number of
                    articles
32              articlesAdded++;
33              if (articlesAdded >= ARTICLES_LIMIT)
34                  throw new SAXException();
35          }
36          Log.d(LOG_TAG, "endElement-end");
37      }
```

Listing A.4: Suchen und Markieren des entsprechenden End-Tags

```
1       public void characters(char ch[], int start, int length) {
2           Log.d(LOG_TAG, "characters-start");
3           String chars = (new String(ch).substring(start, start +
                length));
4           try {
5               // If not in item, then title/link refers to feed
6               if (!inItem) {
7                   if (inTitle)
8                       currentFeed.title = chars;
9               } else {
10                  if (inLink)
11                      currentArticle.url = new URL(chars);
12                  if (inTitle)
13                      currentArticle.title = chars;
14                  if(inLat)
15                      currentArticle.latitude = Double.parseDouble(
                            chars);
16                  if(inLng)
17                      currentArticle.longitude = Double.parseDouble(
                            chars);
18                  if(inDistance)
19                      currentArticle.distance = Double.parseDouble(
                            chars);
20              }
21          } catch (MalformedURLException e) {
22              Log.e(LOG_TAG, "characters" + e.toString());
23          }
```

```
24          Log.d(LOG_TAG, "characters-end");
25      }
```

Listing A.5: Auslesen und Speichern der Zeichen zwischen den zusammengehörigen Tags

A.3. GUI Layout

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
       android"
3      android:orientation="vertical" android:layout_width="
           fill_parent"
4      android:layout_height="fill_parent" android:gravity="center">
5      <TableLayout
6          xmlns:android="http://schemas.android.com/apk/res/android"
7          android:layout_width="wrap_content"
8          android:layout_height="wrap_content" android:
               stretchColumns="1">
9
10         <TableRow android:layout_width="fill_parent">
11             <LinearLayout
12                 xmlns:android="http://schemas.android.com/apk/res/
                       android"
13                 android:orientation="vertical" android:
                       layout_width="fill_parent"
14                 android:layout_height="fill_parent"
15                 android:gravity="center_vertical">
16                 <TextView android:layout_width="wrap_content"
17                     android:layout_height="fill_parent"
18                     android:text="@string/login_username_label" />
19             </LinearLayout>
20             <LinearLayout
21                 xmlns:android="http://schemas.android.com/apk/res/
                       android"
22                 android:orientation="vertical" android:
                       layout_width="fill_parent"
23                 android:layout_height="fill_parent" android:
                       gravity="right">
24                 <EditText android:id="@+id/login_username"
25                     android:layout_width="fill_parent"
26                     android:layout_height="wrap_content"
27                     android:singleLine="true" />
28             </LinearLayout>
```

```
29          </TableRow>
30          <TableRow android:layout_width="fill_parent">
31              <LinearLayout
32                  xmlns:android="http://schemas.android.com/apk/res/
                        android"
33                  android:orientation="vertical" android:
                        layout_width="fill_parent"
34                  android:layout_height="fill_parent"
35                  android:gravity="center_vertical">
36                  <TextView android:layout_width="wrap_content"
37                      android:layout_height="wrap_content"
38                      android:text="@string/login_password_label" />
39              </LinearLayout>
40              <LinearLayout
41                  xmlns:android="http://schemas.android.com/apk/res/
                        android"
42                  android:orientation="vertical" android:
                        layout_width="fill_parent"
43                  android:layout_height="fill_parent" android:
                        gravity="right">
44                  <EditText android:id="@+id/login_password"
45                      android:layout_width="fill_parent"
46                      android:layout_height="wrap_content"
47                      android:singleLine="true"
48                      android:password="true" />
49              </LinearLayout>
50          </TableRow>
51                  <TableRow android:layout_width="fill_parent">
52              <LinearLayout
53                  xmlns:android="http://schemas.android.com/apk/res/
                        android"
54                  android:orientation="vertical" android:
                        layout_width="fill_parent"
55                  android:layout_height="fill_parent"
56                  android:gravity="center_vertical">
57                  <TextView android:layout_width="wrap_content"
58                      android:layout_height="wrap_content"
59                      android:text="@string/login_server_label" />
60              </LinearLayout>
61              <LinearLayout
62                  xmlns:android="http://schemas.android.com/apk/res/
                        android"
63                  android:orientation="vertical" android:
                        layout_width="fill_parent"
64                  android:layout_height="fill_parent" android:
                        gravity="right">
```

```
65                  <EditText android:id="@+id/login_server"
66                      android:layout_width="fill_parent"
67                      android:layout_height="wrap_content"
68                      android:singleLine="true"/>
69              </LinearLayout>
70          </TableRow>
71                  <TableRow android:layout_width="fill_parent">
72              <LinearLayout
73                  xmlns:android="http://schemas.android.com/apk/res/
                        android"
74                  android:orientation="vertical" android:
                        layout_width="fill_parent"
75                  android:layout_height="fill_parent"
76                  android:gravity="center_vertical">
77                  <TextView android:layout_width="wrap_content"
78                      android:layout_height="wrap_content"
79                      android:text="@string/login_port_label" />
80              </LinearLayout>
81              <LinearLayout
82                  xmlns:android="http://schemas.android.com/apk/res/
                        android"
83                  android:orientation="vertical" android:
                        layout_width="fill_parent"
84                  android:layout_height="fill_parent" android:
                        gravity="right">
85                  <EditText android:id="@+id/login_port"
86                      android:layout_width="fill_parent"
87                      android:layout_height="wrap_content"
88                      android:singleLine="true"/>
89              </LinearLayout>
90          </TableRow>
91                  <TableRow android:layout_width="fill_parent">
92              <LinearLayout
93                  xmlns:android="http://schemas.android.com/apk/res/
                        android"
94                  android:orientation="vertical" android:
                        layout_width="fill_parent"
95                  android:layout_height="fill_parent"
96                  android:gravity="center_vertical">
97                  <TextView android:layout_width="wrap_content"
98                      android:layout_height="wrap_content"
99                      android:text="@string/login_service_label" />
100             </LinearLayout>
101             <LinearLayout
102                 xmlns:android="http://schemas.android.com/apk/res/
                        android"
```

```
103                 android:orientation="vertical" android:
                        layout_width="fill_parent"
104                 android:layout_height="fill_parent" android:
                        gravity="right">
105                 <EditText android:id="@+id/login_service"
106                     android:layout_width="fill_parent"
107                     android:layout_height="wrap_content"
108                     android:singleLine="true"/>
109             </LinearLayout>
110         </TableRow>
111     </TableLayout>
112     <LinearLayout android:layout_width="fill_parent"
113         android:layout_height="wrap_content" android:padding="20sp
                "
114         xmlns:android="http://schemas.android.com/apk/res/android"
                >
115         <LinearLayout android:layout_width="wrap_content"
116             android:layout_height="wrap_content"
117             xmlns:android="http://schemas.android.com/apk/res/
                    android"
118             android:orientation="vertical">
119             <ImageButton android:id="@+id/login_exit"
120                 android:layout_width="wrap_content"
121                 android:layout_height="wrap_content"
122                 android:text="@string/login_exit" android:src="
                        @drawable/exit_24"
123                 android:gravity="left" />
124         </LinearLayout>
125         <LinearLayout android:layout_width="fill_parent"
126             android:layout_height="wrap_content"
127             xmlns:android="http://schemas.android.com/apk/res/
                    android"
128             android:gravity="right">
129             <ImageButton android:id="@+id/login_login"
130                 android:layout_width="wrap_content"
131                 android:layout_height="wrap_content"
132                 android:text="@string/login_login" android:src="
                        @drawable/login_24" />
133         </LinearLayout>
134     </LinearLayout>
135 </LinearLayout>
```

Listing A.6: Beispiel einer mit XML erstellten GUI

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Glossar

Augmented Reality
: Augmented Reality bezeichnet die um beispielsweise virtuelle Tags erweiterte Realität.

Gadget
: Ein kleines nicht eigenständiges Computerprogramm, welches eine bestimmte Umgebung benötigt. Diese stellt entsprechend Grundfunktionen und Ressourcen zur Verfügung. Ein Gadget wird zum Beispiel in eine Webseite eingebunden.

Instant Messaging
: Instant Messaging bezeichnet den direkten Textnachrichtenaustausch. Dieser wird im Allgemeinen als chatten bezeichnet. Um die Nachrichten sofort zu übertragen, wird das Push-Verfahren verwendet. Dazu müssen sie direkt oder über einen Server verbunden sein.

Presence Information
: Unter Presence Information versteht man die Möglichkeit, einen bestimmten Status einzunehmen, während man zum Beispiel einen Instant Messenger benutzt. Dieser Status zeigt zum Beispiel die Bereitschaft zum Kommunizieren an. Mögliche Stati sind: Verfügbar, abwesend, unsichtbar. Zusätzlich lassen sich solche Stati noch mit persönlichen Nachrichten anreichern.

Proof of Concept
: Proof of Concept ist ein Machbarkeitsbeweis, der meist mit Hilfe eines Prototypen belegt, dass die prinzipielle Durchführung eines Projektes möglich ist.

Smartphone
: Als Smartphone definiert TNS dabei alle Handymodelle, die neben einem Internetzugang mindestens zwei Anwendungen aus E-Mail Funktionalität, PDA-Funktion, drahtlose Datenübertragung (WLAN), Microsoft Office Kompatibilität und Touchscreen aufweisen.

Ubiquitous Computing
: Bedeutet übersetzt in die deutsche Sprache „Allgegenwärtige Datenverarbeitung“. Mehr Details sind in dem entsprechenden Kapitel Ubiquitous Computing 2.1 zu erfahren.

XMPP

Das Extensible Messaging and Presence Protocol (kurz: XMPP) ist ein Internetstandard für XML-Routing, im Moment wird es primär für Instant Messaging eingesetzt. XMPP ist XML-basiert und bildet die Grundlage des Jabber-Protokolls.

Abkürzungsverzeichnis

API	Application Programming Interface.
FSF	Free Software Foundation.
GPL	General Public License.
HMI	Human-Machine Interface.
IM	Instant Messaging.
JSR	Java Specification Request.
JVM	Java Virtual Machine.
mucM	mobile ubiquitous context-aware Messenger.
OSI	Open Source Initiative.
REST	Representational State Transfer.
SDK	Software Development Kit.
SQL	Structured Query Language.
UGC	User Generated Content.
VoIP	Voice over IP.

Literaturverzeichnis

[Raymond 1997] *The Cathedral and the Bazaar*. Eric S. Raymond. 1997. – URL http://www.catb.org/~esr/writings/cathedral-bazaar/. – Abrufdatum 01.07.2008

[Stallman 1998] *Das Gnu Projekt*. Richard Stallman. 1998. – URL http://www.gnu.org/gnu/thegnuproject.de.html. – Abrufdatum 01.07.2008

[FSF 1999] *Initial Announcement*. Free Software Foundation. 1999. – URL http://www.gnu.org/gnu/initial-announcement.html#f1. – Abrufdatum 01.07.2008

[SLAM 2003] *SLAM*. Microsoft Coperation. 2003. – URL http://www.msslam.com/. – Abrufdatum 30.04.2008

[Fring 2006] *Fring it's fringing freedom!* Fring. 2006. – URL http://www.fring.com/. – Abrufdatum 21.07.2008

[OSI 2006] *History of the OSI*. Open Source Initiative. 2006. – URL http://opensource.org/history. – Abrufdatum 02.07.2008

[Dodgeball 2007] *dodgeball.com*. Crowley, Dens. 2007. – URL http://www.dodgeball.com/. – Abrufdatum 30.04.2008

[comScore 2007] *Social Networking Goes Global*. comScore. 2007. – URL http://www.comscore.com/press/release.asp?press=1555. – Abrufdatum 19.06.2008

[Socialight 2007] *Socialight*. Kamida. 2007. – URL http://socialight.com/. – Abrufdatum 30.04.2008

[Android 2007] *What is Android?* Google. 2007. – URL http://code.google.com/android/what-is-android.html. – Abrufdatum 21.06.2008

[Gartner 2008] *Gartner Says Worldwide Smartphone Sales Grew 16 Per Cent in Second Quarter of 2008*. Gartner. 2008. – URL http://www.gartner.com/it/page.jsp?id=754112. – Abrufdatum 11.09.2008

[TNS 2008] *Global Technology Insights 2007-08*. TNS Infratest. 2008. – URL http://www.tns-infratest.com/presse/pdf/Presse/20080304_TNS_Infratest_GTI_200708.pdf. – Abrufdatum 21.05.2008

[OpenSocial 2008] *Open Social*. Google. 2008. – URL http://code.google.com/apis/opensocial/. – Abrufdatum 01.07.2008

[TNS2 2008] *TNS Infratest Studie zur aktuellen Nutzung des mobilen Internets*. TNS Infratest. 2008. – URL http://www.tns-infratest.com/presse/pdf/Presse/20080911_TNS_Infratest_MobilesInternet.pdf. – Abrufdatum 15.09.2008

[Abowd u. a. 1999] ABOWD, Gregory D. ; DEY, Anind K. ; BROWN, Peter J. ; DAVIES, Nigel ; SMITH, Mark ; STEGGLES, Pete: Towards a Better Understanding of Context and Context-Awareness. In: *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. London, UK : Springer-Verlag, 1999, S. 304–307. – URL http://portal.acm.org/citation.cfm?id=743843. – ISBN 3540665501

[Alliance 2008] ALLIANCE, Open H.: *Android Market*. http://android-developers.blogspot.com/2008/08/android-market-user-driven-content.html. 2008. – URL http://android-developers.blogspot.com/2008/08/android-market-user-driven-content.html. – Abrufdatum 31.08.2008

[Apple 2008] APPLE: *Apple iPhone - Wikipedia*. http://de.wikipedia.org/wiki/Iphone. 2008. – URL http://de.wikipedia.org/wiki/Iphone. – Abrufdatum 01.08.2008

[Arlow und Neustadt 2005] ARLOW, Jim ; NEUSTADT, Ila: *UML 2.0 and the Unified Process. Practical Object-Oriented Analysis and Design*. Addison-Wesley, July 2005

[Baldauf u. a. 2007] BALDAUF, Matthias ; DUSTDAR, Schahram ; ROSENBERG, Florian: A survey on context-aware systems. In: *Int. J. Ad Hoc Ubiquitous Comput.* 2 (2007), Nr. 4, S. 263–277. – ISSN 1743-8225

[Bennett u. a. 1994] BENNETT, Frazer ; RICHARDSON, Tristan ; HARTER, Andy: Teleporting - Making Applications Mobile. In: *Proceedings of 1994 Workshop on Mobile Computing Systems and Applications*. Santa Cruz, December 1994. – URL citeseer.ist.psu.edu/bennett94teleporting.html

[Bien 2006] BIEN, Adam: *Enterprise Architekturen - Leitfaden für die effiziente Software-Entwicklung*. entwickler.press, 2006. – ISBN 3-935042-99-x

[Breslin und Decker 2007a] BRESLIN, John ; DECKER, Stefan: The Future of Social Networks on the Internet: The Need for Semantics. In: *IEEE Internet Computing* 11 (2007), Nr. 6, S. 86–90. – ISSN 1089-7801

[Breslin und Decker 2007b] BRESLIN, John ; DECKER, Stefan: The Future of Social Networks on the Internet: The Need for Semantics. In: *IEEE Internet Computing* 11 (2007), Nr. 6, S. 86–90. – ISSN 1089-7801

[Brown 1996] BROWN, P. J.: The Stick-e Document: a Framework for Creating Context-aware Applications. In: *Proceedings of EP'96, Palo Alto*, also published in it EP–odd, June 1996, S. 259–272. – URL http://www.cs.kent.ac.uk/pubs/1996/396

[Chen und Kotz 2000] CHEN, Guanling ; KOTZ, David: A Survey of Context-Aware Mobile Computing Research. Hanover, NH, USA : Dartmouth College, 2000. – Forschungsbericht. – URL http://portal.acm.org/citation.cfm?id=867843

[Chris Rupp 2007] CHRIS RUPP, u.a.: *Requirementsengineering und Management*. Carl Hanser Verlag München, 2007. – ISBN 978-3-446-40509-7

[Dey 1998] DEY, Anind K.: Context-aware computing: The CyberDesk project. In: *AAAI 1998 Spring Symposium on Intelligent Environments*. Palo Alto : AAAI Press., 1998, S. 51–54. – URL http://www.cc.gatech.edu/fce/cyberdesk/pubs/AAAI98/AAAI98.html

[Fitzpatrick 2007] FITZPATRICK, Brad: *Thoughts on the Social Graph*. 2007. – URL http://bradfitz.com/social-graph-problem/. – Abrufdatum 03.07.2008

[Gershenfeld 1999] GERSHENFELD, Neil: *When Things Start to Think*. Henry Holt and Company, Inc., 1999. – ISBN 0805058745

[Grad 2002] GRAD, Burton: A Personal Recollection: IBM's Unbundling of Software and Services. In: *IEEE Ann. Hist. Comput.* 24 (2002), Nr. 1, S. 64–71. – ISSN 1058-6180

[Green und Pearson 2005] GREEN, David T. ; PEARSON, John M.: Social Software and Cyber Networks: Ties That Bind or Weak Associations within the Political Organization? In: *HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 5*. Washington, DC, USA : IEEE Computer Society, 2005, S. 117.2. – ISBN 0-7695-2268-8-5

[Hagberg 1997] HAGBERG, Sean: Edwin Hutchins, Cognition in the Wild. In: *Minds Mach.* 7 (1997), Nr. 3, S. 456–460. – ISSN 0924-6495

[Hansmann u. a. 2001] HANSMANN, Uwe ; NICKLOUS, Martin S. ; STOBER, Thomas: *Pervasive computing handbook*. New York, NY, USA : Springer-Verlag New York, Inc., 2001. – ISBN 3-540-67122-6

[J. Burkhardt 2001] J. BURKHARDT, u.a.: *Pervasive computing. Grundlagen und Anwendungen*. Addison-Wesley, 2001

[J2ME 2007] J2ME: *Java Platform, Micro Edition - Wikipedia*. http://de.wikipedia.org/wiki/J2ME. 2007. – URL `http://de.wikipedia.org/wiki/J2ME`. – Abrufdatum 01.08.2008

[Johnson 1998] JOHNSON, Luanne (.: A View From the 1960s: How the Software Industry Began. In: *IEEE Ann. Hist. Comput.* 20 (1998), Nr. 1, S. 36–42. – ISSN 1058-6180

[Mattern 2001] MATTERN, Friedemann: Ubiquitous Computing – Der Trend zur Informatisierung und Vernetzung aller Dinge. In: ROSSBACH, Gerhard (Hrsg.): *Mobile Internet, Tagungsband 6. Deutscher Internet-Kongress, dpunkt-Verlag*. September 2001, S. 107–119. – (Aktualisierte Version von Übiquitous Computing", erschienen in: Internet @ Future, Jahrbuch Telekommunikation und Gesellschaft 2001)

[Mattern 2005a] MATTERN, Friedemann: Allgegenwärtige Informationsverarbeitung – Technologietrends und Auswirkungen des Ubiquitous Computing. (2005)

[Mattern 2005b] MATTERN, Friedemann: Allgegenwärtige und verschwindende Computer. In: *Praxis der Informationsverarbeitung und Kommunikation (PIK)* 28 (2005), Januar, Nr. 1, S. 29–36

[Microsoft 2007] MICROSOFT: *.NET Compact Framework - Wikipedia*. http://de.wikipedia.org/wiki/.NET_Compact_Framework. 2007. – URL `http://de.wikipedia.org/wiki/.NET_Compact_Framework`. – Abrufdatum 01.08.2008

[Mockus 2002] MOCKUS, Audris: Two case studies of open source software development: Apache and Mozilla. In: *ACM Trans. Softw. Eng. Methodol.* 11 (2002), Nr. 3, S. 309–346. – ISSN 1049-331X

[Moore 1965] MOORE, G. E.: Cramming More Components onto Integrated Circuits. In: *Electronics* 38 (1965), April, Nr. 8, S. 114–117. – URL `http://dx.doi.org/10.1109/JPROC.1998.658762`

[Nieuwdorp 2007] NIEUWDORP, Eva: The *ervasive* discourse: an analysis. In: *Computers in Entertainment* 5 (2007), Nr. 2

[O'Reilly 2005] O'REILLY, Tim: *What Is Web 2.0? Design Patterns and Business Models for the Next Generation of Software*. www.oreilly.com. September 2005

[Reed 1999] REED, D. P.: That Sneaky Exponential— Beyond Metcalfe's Law to the Power of Community Building. In: *Context Magazine* (1999), Spring. – URL `http://www.contextmag.com/archives/199903/digitalstrategyreedslaw.asp`

[Ryan u. a. 1998] RYAN, N. S. ; PASCOE, J. ; MORSE, D. R.: Enhanced Reality Fieldwork: the Context-aware Archaeological Assistant. In: GAFFNEY, V. (Hrsg.) ; LEUSEN, M. van

(Hrsg.) ; EXXON, S. (Hrsg.): *Computer Applications in Archaeology 1997*. Oxford : Tempus Reparatum, October 1998 (British Archaeological Reports). – URL http://www.cs.kent.ac.uk/pubs/1998/616

[Scacchi 2007] SCACCHI, Walt: Free/open source software development: recent research results and emerging opportunities. In: *ESEC-FSE companion '07: The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering*. New York, NY, USA : ACM, 2007, S. 459–468. – ISBN 978-1-59593-812-1

[Schilit und Theimer 1994] SCHILIT, B. N. ; THEIMER, M. M.: Disseminating active map information to mobile hosts. In: *Network, IEEE* 8 (1994), Nr. 5, S. 22–32. – URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=313011

[Schmidt 2006] SCHMIDT, Jan: Social Software. Onlinegestütztes Informations-, Identitäts- und Beziehungsmanagement. In: *Forschungsjournal Neue Soziale Bewegungen* (2006), Nr. 2, S. 37–47

[Siedersleben 2004] SIEDERSLEBEN, Johannes: *Moderne Software-Architektur*. City : Dpunkt.Verlag GmbH, 2004. – ISBN 3898642925

[Starke 2005] STARKE, Gernot: *Effektive Softwarearchitekturen*. Carl Hanser Verlag München, 2005. – ISBN 3-446-22846-2

[Stearns u. a. 2006] STEARNS, Howard ; GARGUS, Joshua ; SCHUETZE, Martin ; LOMBARDI, Julian: Simplified Distributed Authoring Via Component-based Object Construction and Deconstruction in Collaborative Croquet Spaces. In: *C5 '06: Proceedings of the Fourth International Conference on Creating, Connecting and Collaborating through Computing*. Washington, DC, USA : IEEE Computer Society, 2006, S. 79–87. – ISBN 0-7695-2563-6

[Tuomi 2002] TUOMI, Ilkka: The Lives and Death of Moore's Law. In: *First Monday* 7 (2002), Nr. 11. – URL http://dblp.uni-trier.de/db/journals/firstmonday/firstmonday7.html#Tuomi02

[Want u. a. 1992] WANT, Roy ; HOPPER, Andy ; FALCAO, Veronica ; GIBBONS, Jonathan: The active badge location system. In: *ACM Trans. Inf. Syst.* 10 (1992), Nr. 1, S. 91–102. – ISSN 1046-8188

[Weiser 1991] WEISER, Mark: The Computer for the 21st Century. In: *Scientific American* (1991), 02/1991. – URL http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html

[Yan und Selker 2000] YAN, Hao ; SELKER, Ted: Context-aware office assistant. In: *IUI '00: Proceedings of the 5th international conference on Intelligent user interfaces*. New York, NY, USA : ACM, 2000, S. 276–279. – ISBN 1-58113-134-8

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 6. Oktober 2008	
Ort, Datum	Unterschrift