Bachelorarbeit

Anton Prawiro

Entwicklung eines mobilen Verabredungssystems
auf Basis von WLAN-Ortung

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Anton Prawiro

Entwicklung eines mobilen Verabredungssystems auf Basis von WLAN-Ortung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Ing Birgit Wendholt
Zweitgutachter : Prof. Dr. rer.nat. Gunter Klemke

Abgegeben am 25. Mai 2010

Anton Prawiro

Thema der Bachelorarbeit

Entwicklung eines mobilen Verabredungssystems auf Basis von WLAN-Ortung.

Stichworte

Verabredungssystem, Positionsbestimmung, Person Tracking, Indoor Lokalisierung.

Kurzzusammenfassung

Innerhalb dieser Bachelorarbeit wird ein Indoor Verabredungssystem für Smartphones mit Hilfe von vor Ort vorhandener Infrastruktur (z.B. WLAN) für die Lokalisierung und ortsgebundener Dienste für das Aufspüren von Verabredungsbeteiligten realisiert.

Anton Prawiro

Title of the paper

Meeting assistent system oriented on indoor areas using WLAN infrastructure.

Keywords

Meeting assistent system, positioning system, person tracking, indoor positioning system.

Abstract

This bachelor thesis evolves around an indoor meeting assistant system, intended to be use in Smartphone. It uses the available infrastructure for the positioning. It also uses location-based-services to detect other parties concerned.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Danksagung

Hiermit möchte ich mich bei all denen bedanken, die mir bei der Erstellung dieser Arbeit direkt oder indirekt geholfen haben.

Zuerst bedanke ich mich an Gott, der mir sowohl in meinen guten als auch in schlechten Zeiten geholfen hat. Nur durch seine große Gnade kann ich mein Studium zum Ende bringen.

Ich möchte mich bei meiner Betreuerin Frau Prof. Dr. Ing. Birgit Wendholt, die mich zielorientiert unterstützt und mir konstruktive Anmerkungen gegeben hat.
Herr Prof. Dr. rer. nat. Gunter Klemke möchte ich dafür danken, dass er als mein zweiter Prüfer sofort zugesagt hat.

Danach möchte ich mich auch bei meiner Freundin, Grace Jaya, bedanken, für alle ihre Unterstützungen von Anfang dieser Arbeit an.

Und natürlich alle meine Freunde, die mir während der Erstellung dieser Bachelorarbeit unterstützt haben, möchte ich bedanken. Besonders Daniel Plappert, der mir mit Grammatik- und Satzaufbauproblemen viel geholfen hat!

1. Einleitung

1.1. Motivation

Heutzutage wird angestrebt, alle möglichen Aspekte in unserem Leben, die wiederholt auftreten bzw. voraus gesehen werden können, zu automatisieren. Damit soll erreicht werden, dass sich die Menschen nicht mehr um derartige Bereiche kümmern müssen und auf andere wichtige Aufgaben konzentrieren.
Eine Verabredung gehört zu solchen Aspekten, die automatisiert werden kann.
Viele Smartphones haben einen Terminkalender, der einen Termin erfassen kann und den Benutzer zu einem bestimmten Zeitpunkt an den Termin erinnert.
Allerdings muss der Benutzer, wenn er zu dem Zeitpunkt vor Ort ist, dann noch seinen Partner manuell kontaktieren bzw. fragen, wo sein Partner sich befindet. Den genauen Ort bzw. Raum muss er danach gegebenenfalls noch suchen.

Lösungen für solche Szenarien sind für den Outdoor-Bereich bereits mehrfach vorhanden. Das bekannteste Beispiel für Outdoor-Positionsbestimmung ist wohl die Autonavigation, die mit Hilfe der Galileo Satelliten die Position bestimmt. Für die Positionsbestimmung werden mindestens drei solcher Satelliten gebraucht. Eine weitere Methode ist A-GPS (Assisted GPS). Dabei wird zusätzlich zu den GPS-Signalen das GSM-Signal verwendet, um möglichst hohe Genauigkeit bei der Positionsbestimmung zu erreichen.

Die genannten Methoden helfen allerdings nur wenig beim Einsatz im Indoor-Bereich, da die Mehrzahl von den GPS Signalen von den Wänden absorbiert oder reflektiert werden und dadurch ungenaue Ergebnisse liefern. Das GSM-Signal alleine liefert für die Positionsbestimmung ebenfalls nur ungenaue Ergebnisse.
Positionsbestimmung bestimmt im Wesentlichen die eigene Position. Damit eine Person X die Position einer anderen Person Y in einem automatisierten Vorgang erkunden kann, stößt die Positionsbestimmung allerdings an ihre Grenzen. Für solche Fälle muss eine sogenannte Personensuche verwendet werden.

Ein Beispiel: Zwei Personen (X und Y) haben sich in einem Gebäude A verabredet. Person Y wartet auf Person X in diesem Gebäude. Wenn Person X vor dem Gebäude steht, muss Person X Person Y benachrichtigen, dass Person X sich vor dem Gebäude steht und auch eventuell fragen, wo Person Y zurzeit sich befindet. Es kann auch sein, dass Person Y in ständiger Bewegung ist, z.B. Person Y geht in einem anderen Zimmer bzw. zu einem anderen Stockwerk. Hiermit soll nun erreicht werden, dass möglichst, bei einer Verabredung, die wartende Person (im Beispiel Person Y) über das Ankommen der erwarteten Personen (im Beispiel Person X) automatisch

benachrichtigt wird. Gleichzeitig soll die Person X (jederzeit) über die Position der Person Y informiert werden.

Für die Automatisierung wird zudem noch eine Positionsbestimmung oder Ortung benötigt. Positionsbestimmung ist eine Ermittlung des Ortes in Bezug zu einem gewissen Bezugspunkt (Bezugssystem) [WikiOrt].

Es gibt bereits mehrere Projekte, Software-Lösungen und auch Firmen, die im Bereich der Ortsmessungs- oder Positionsbestimmung arbeiten, von denen sich die meisten mit der Outdoor-Positionsbestimmung beschäftigen.
Für die Indoor-Positionsbestimmung müssen jedoch andere Methoden zum Einsatz kommen, die genauere Ergebnisse liefern. Gängige Methoden sind RFID[1]-[Finkenzeller, K 2003], Funksignal-, Ultraschallsignal-, Bluetooth- und WLAN-gestützte Positionsbestimmung.

1.2 Zielsetzung

Für die Automatisierung eines Verabredungsablaufs soll eine Anwendung auf einem Windows Mobile basiertem Smartphone entwickelt werden.
Die Anwendung dient dem Benutzer als Verabredungsassistent, in dem:

- die wartende Person über das Ankommen der erwarteten Person informiert wird.
- die erwartete Person über die Position der wartenden Person informiert wird.
- die erwartete Person über mögliche Positionsänderung der wartenden Person benachrichtigt wird.
- alle mögliche Vorgänge auf Basis der Push-Technologie realisiert werden, d.h., dass der Benutzer nicht aktiv das Ereignis (das Ankommen der erwarteten Person) abfragen muss.

Die vier genannten Bedingungen sollen für den Indoor Bereich umgesetzt werden.

Die Positionsbestimmung und Personensuche werden in den nächsten Kapiteln anhand von Beispielen der existierenden Arbeiten bzw. Software-Lösungen genauer unter die Lupe genommen. Die besten Methoden und Anwendung sollen als Basis für diese Arbeit dienen.

Als letztes wichtiges Ziel, das nicht außer Acht gelassen werden sollte, ist die Benutzerfreundlichkeit zu nennen. Der Benutzer kann nicht zu jeder Zeit auf sein Smartphone achten, deshalb sollen alle automatisierbaren Vorgänge identifiziert und umgesetzt werden, die den Benutzer im Umgang mit der Anwendung maximal entlasten.
Innerhalb dieser Rahmbedingungen bewegt sich diese Bachelorarbeit.

---

[1] Radio Frequency Identity.

Ziel dieser Arbeit ist also die Entwicklung eines mobilen persönlichen Assistenten für die Unterstützung von Verabredungsszenarien in Gebäuden.

1.3 Gliederung der Arbeit

Diese Arbeit wird wie folgt gegliedert:

- *Kapitel 1 – Einleitung*
  Dieses Kapitel erläutert die Motivation, die dieser Arbeit zugrunde liegt und benennt eine erste Gliederung.

- *Kapitel 2 – Vergleichbare Arbeiten*
  Dieses Kapitel benennt andere verwandte Arbeiten, die bereits untersucht wurden bzw. sich im Einsatz befinden. Diese werden nach ihren Funktionen systematisiert, verglichen und am Ende wird die Arbeit, die möglichst viele offene Anforderungen abgedeckt hat, genommen.

- *Kapitel 3 – Analyse*
  Dieses Kapitel beschreibt die Anwendungen anhand von Use-Case Diagrammen, die Vorbedingungen, sowie die technischen und nicht-technischen Anforderungen dieser Arbeit.

- *Kapitel 4 – Design und Realisierung*
  Dieses Kapitel behandelt die Problembeschreibung der Arbeit, die Technologien und auch die Software, die für die Realisierung benötigt werden Unter Anwendung der UML-Methode wird die Problemlösung entwickelt.

- *Kapitel 5 – Fazit*
  Dieses Kapitel dient als Ausblick und Zusammenfassung dieser Arbeit.

2. Vergleichbare Arbeiten

Computergestützte Verabredungsszenarien zwischen zwei oder mehreren Personen, auch als Person Finder oder Friend Finder bezeichnet, müssen mindestens zwei wesentlichen Aspekte untersuchen oder berücksichtigen:

- wie man sich orten lässt.
- wie man unterstützt durch mobile Computer oder Smartphones Verabredungen ortsbasiert unterstützen kann und dabei ortsabhängige Events nutzt (z.B. eine Person befindet sich im Umkreis von einem Kilometer oder eine Person betritt ein Gebäude).

Beide Punkte gehören eigentlich zu den Anwendungsbeispielen des Location-Based-Services (LBS). Schiller und Voisard definieren Location-Based-Services wie folgt:

> *The term location-based services (LBS) is a recent concept that denotes applications integrating geographic location (i.e., spatial coordinates) with the general notion of services.*
>
> [Schiller, u.a. 2004]

In dieser Arbeit wird das Konzept der LBS mit Fokus auf mobilen Anwendungen untersucht. Bezüglich der Ortung sind zwei verschiedene Bereiche zu nennen: Outdoor und Indoor Ortung. Outdoor Ortung basiert auf GPS (s. Kapitel 1.1), GSM (zellbasiert), WLAN oder die Kombinationen von GPS, GSM und WLAN. Outdoor Ortung gilt heute als „state of the art“: Alle modernen Smartphones enthalten heutzutage eine Kombination aus den genannten Verfahren.

Im Gegensatz zur Outdoor Ortung basiert Indoor Ortung auf anderen Infrastrukturen, wie z.B. WLAN, Bluetooth, Funksignalen. Heutzutage ist sie fast ausschließlich in spezialisierten Anwendungsbereichen anzutreffen und noch nicht im Massenmarkt verbreitet. Insbesondere die Verwendung der WLAN Infrastruktur ist sehr weit verbreitet, z.B.: fast alle modernen Handys oder Smartphones sind mit WLAN-Empfängern ausgestattet, ebenso fast alle modernen Gebäude mit WLAN-Sendern (Access Point bzw. AP).
Im Kapitel 2.1 sollen nun die bereits existierenden Arbeiten in Bezug der Indoor-Ortung genauer betrachtet werden.

Die Personensuche ist ein ortsabhängiger, ereignisbasierter Mechanismus. Der Mechanismus ist ein durchzuführender Vorgang, der durch ortsgebundene Ereignisse getriggert wird. Das Muster kann bspw. sein: Person A (die erwartete Person) betritt

das Gebäude, Person B (die wartende Person) wechselt das Stockwerk, und vieles mehr. Über der Personensuche wird im Kapitel 2.2 beschreiben.
Dieses Kapitel stellt Beispiele für ortsbasierte mobile Personenfinder vor und zeigt die unterschiedlichen Ansätze zur Realisierung eines Benachrichtigungsverfahrens für ortsbasierte Ereignisse.

Damit sich zwei Personen in einem Gebäude finden können lässt sich ein Person Finder um kartenbasierte Führung, besser bekannt als Indoornavigation, erweitern. Es gibt in diesem Bereich bereits viele verschiedenen Lösungen. Einige von ihnen sind in Form wissenschaftlicher Arbeiten an der HAW zu finden: Thomas Pfaff [Pfaff 2007] und Borys Kogan [Kogan 2009].

Indoornavigation selbst soll allerdings nicht Bestandteil dieser Arbeit werden. Eine derartige Anwendung müsste Gegenstand einer separaten Lösung sein. Dieses Kapitel konzentriert sich dagegen auf die Location-Based-Services, die als Personenfinder bekannt sind (Friend Finder Anwendung).

2.1. Ortung

Positions- oder Ortsbestimmung (abk. Ortung) ist eine Ermittlung des Ortes in Bezug zu einem gewissen Bezugspunkt (Bezugssystem). In diesem Fall wird über die Bestimmung des eigenen Standortes [WikiOrt] geredet.
Diese soll in dieser Arbeit in Verfahren zur kontinuierlichen Ortung (s. Kapitel 2.1.1) und solche zur partiellen Ortung (s. Kapitel 2.1.2) unterschieden werden.

2.1.1. Kontinuierliche Ortung

Unter kontinuierlicher Ortung versteht man, dass man sich jederzeit über seine aktuelle Position informieren kann. Der Benutzer erhält die Signale, die für die Positionsberechnung nötig sind, ununterbrochen von einem Signalsender bzw. mehreren Signalsendern. Ein solcher Signalsender kann im Form von WLAN Access Points (z.B. bei WiPoD), Beacon (s. Kapitel 2.1.1.2), Bluetooth vorhanden sein.

Da die Anwendung bei kontinuierlichem Verfahren mit den benötigten Signalen ununterbrochen beliefert wird, erhält man eine dementsprechend höhere Genauigkeit für die Positionsbestimmung als bei der partiellen Ortung (s. Kapitel 2.1.2). Nachteilig wirkt sich aus, dass die Anwendung stark von den Signalen abhängig ist. Das ist der Fall, wenn die vorhandene Signale in umliegende Umgebung nicht zuverlässig sind, z.B. die Signale werden von den Wänden stark absorbiert oder reflektiert, liefert die Anwendung schlechte bzw. ungenaue Ergebnisse.

Im nächsten Unterkapitel werden ein paar konkrete Beispielanwendungen behandelt, die kontinuierliche Ortung benutzen.

2.1.1.1. WiPoD

WiPoD [Gümüskaya, u.a. 2005] steht für Wireless Positioning System based on 802.11 WLAN Infrastructure. Das System wurde an der Fatih Universität entwickelt. Wie der Name schon sagt, benutzt die Arbeit auf WLAN gestützte Positionsberechnung. Die Software sammelt sämtliche Signale von vorhandenen Access Points, speichert die Dateien und berechnet die aktuelle Position einer Person. Anhand dieser Eigenschaften kann WiPoD die vorhandene WLAN Infrastruktur nutzen, daher keine neue (bestimmte) Infrastruktur eingebaut werden muss.
WiPoD besteht aus 2 Komponenten, dem WiPoD Spotter und dem WiPoD Tracker, Ersterer wird für die Kalibrierung und Zweiterer als die eigentliche Anwendung beim Benutzer verwendet. Beide Komponenten verwenden eine kartenbasierte Benutzeroberfläche, die die physikalische Umgebung abbildet.

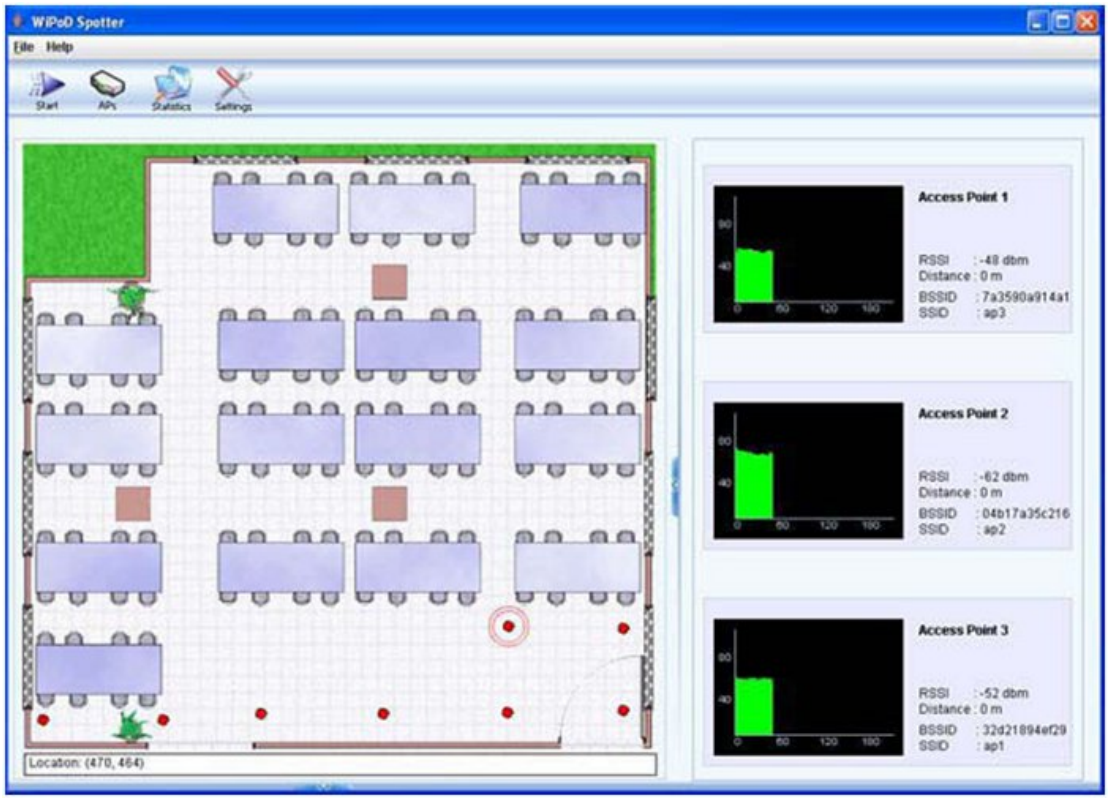

Abbildung 2.1 WiPoD Spotter

Mit Hilfe des WiPoD Spotters wird das gesamte System kalibriert. Die Kalibrierung findet statt, indem sich ein Benutzer auf bestimmten Stellen bewegt. Diese Stellen werden auf der Karte notiert (Referenzpunkte einsetzen, die mit roten Punkten gekennzeichnet wird), wo er sich gerade befindet. Zu gleicher Zeit werden Signale vom WiPoD Spotter (s. Abbildung 2.1) gesammelt, wie zum Beispiel die SSID (Service Set Identifier), BSSID (Basic Service Set Identifier), RSS (Receive Signal

Strength) und noch weitere Informationen. Zusätzlich wird die reale Position des WLAN Senders vom Server gesendet. Danach speichert der WiPoD Spotter die Koordinaten und die sämtliche Signalinformationen von allen Acces Points, die von dem Spotter empfängt werden können (s. Abbildung 2.1 rechts). Dieser Vorgang wird auf mehreren Stellen und an mehreren Tagen wiederholt. Diese Daten werden danach auf dem Server gespeichert.

Zur Ortung ist WiPoD Tracker zuständig. Dessen Benutzeroberfläche ähnelt der Abbildung 2.1, es fehlt nur die Anzeige der Signalstärke. Der Tracker benutzt entweder KNN[2] oder Triangulation Algorithmus. Beide Algorithmen berechnen die Position anhand der Signalstärke, die von mindestens drei unterschiedlichen Access-Points ausgesendet werden. Die Position des Access-Points muss bekannt sein.
Eine weitergehende ausführliche Erklärung über beiden Algorithmen bieten [Flink, U 2008] und [KNN].
Die benötigte Karte und die Referenzpunkte werden vor der Benutzung vom Server beim Erstseinsatz heruntergeladen.

Für die Installation vom WiPoD wird keine zusätzliche Hardware bzw. Infrastruktur benötigt, was den Aufwand zum Aufbau vom WiPoD erheblich erleichtert. Die Abweichung der Genauigkeit des WiPoDs liegt innerhalb eines Bereiches von einigen Metern, was diese Software zu einer guten Basis für eine mögliche Location-Based-Service Software macht, darauf man entwickeln kann.

2.1.1.2. IMAPS

Bei IMAPS [Gregor, S 2006] handelt es sich um eine Positionsbestimmung innerhalb von Gebäuden, welche auf dem von MIT entwickelten Cricket System[3] [Cricket] basiert.
Zum Betreiben der Infrastruktur wird spezielle Hardware benötigt, die Beacons und die Listener.

[2] K-Nearest Neighbor
[3] Cricket Systems benutzt zwei spezielle Hardware für die Positionsbestimmung, Sender und Empfänger. Für genauere Beschreibung von Cricket Systems, bitte auf die vorhandene Referenz aufschlagen.

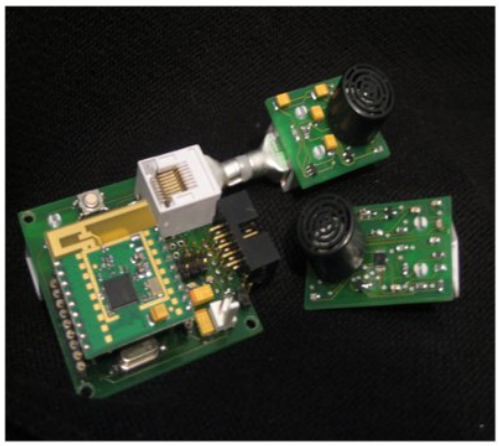

Abbildung 2.2 Beacon und Listener

Der Beacon übernimmt die Aufgabe eines Senders und der Listener übernimmt die Aufgabe eines Empfängers. Der IMAPS Listener benutzt TDOA (Time Difference Of Arrival) Verfahren [DesJardins, G 1993], um eigene Position zu bestimmen. Die Zeitdifferenz, die hier gemeint ist, ist der Zeitabstand zwischen dem Empfangen von Funksignal und Ultraschallsignal, die zeitgleich vom Beacon in regelmäßigen Zeitabständen gesendet werden. Funksignale sind schneller als Ultraschallsignale, das würde bedeutet, obwohl beide Signale zur gleichen Zeit gesendet werden, empfängt der Listener zuerst die Funksignale und erst danach die Ultraschallsignale.

Beide Signale werden für die Positionsberechnung benötigt, wobei das Funksignal unter anderem die Positionsdaten des Beacons enthält. Die Signale werden danach vom Listener gesammelt. Für die Berechnung der Position eines Listeners braucht IMAPS die Distanz zwischen den Beacons und dem Listener.

Um eine höhere Genauigkeit bei der Positionsbestimmung zu erhalten, sollten möglichst viele Beacons zur Verfügung gestellt und sie geschickt platziert werden.
Ein möglichst dichtes Netz von Beacons bedeutet allerdings auch, dass dementsprechend mit höheren Anschaffungskosten gerechnet werden muss.

Abbildung 2.3 Mögliche Beaconsplatzierungen

Bei der Installation der Infrastruktur muss unbedingt darauf geachtet werden, wie weit die Beacon von einander platziert werden. Denn die Positionsberechnung ist abhängig vom Winkel zwischen den Beacons und dem Listener.

Der Vorteil von IMAPS liegt in seiner hohen Genauigkeit der Positionsberechnung, die im Millimeter Bereich ist, was eigentlich sehr ideal ist, Anwendungen auf IMAPS zu bauen.
Nachteil liegt im höheren Schwierigkeitsgrad des Aufbaus und der Wartung der Infrastruktur.

2.1.1.3. Indoor Navigation with Minimal Infrastructure (INMI)

In diesem Unterkapitel wird angestrebt, Positionsbestimmung ohne eine aufwendige Infrastruktur zu ermöglichen.
Bei der INMI [Merico, u.a. 2007] wird ein sog. „Dead Reckoning“ für die Positionsberechnung benutzt, wobei die Geschwindigkeit, Zeit und Richtung für die Berechnung verwendet wird, wobei zwischen den Zeitintervallen die Geschwindigkeit als konstant angenommen wird [Doerfler, R 1993]. Damit wird erreicht, dass der Client nicht so sehr vom Server abhängig ist, was selbstverständlich auch die Genauigkeit der Positionsbestimmung beeinträchtigt. Um diesen Nachteil zu minimieren bzw. zu tolerieren, müssen sich die Clients in bestimmten Zeitintervallen mit dem Server synchronisieren.

Anderer Ansatz dieser Arbeit ist die Kosten der zu verwendenden Infrastruktur zu minimieren. Da aber die Positionsberechnung der Dead Reckoning ungenau ist, müssen zusätzlich andere Wege in Betracht gezogen werden. Möglich ist z.B., dass der Benutzer seine reale Umgebung auf Basis von Bildern, die auf seinem mobilen Gerät angezeigt werden, erkennen soll. Das Bild wird vom Benutzer gewählt und das System erkennt die Position des Benutzers anhand seiner vorausgegangenen Wahl.

Abbildung 2.4 Beispielfoto auf der Benutzeroberfläche

In Abbildung 2.4 ist ein Beispielbild gezeigt, das auf dem mobilen Gerät des Benutzers erscheint. Die Bilder werden auf dem Server gespeichert und automatisch bei Benutzung heruntergeladen. Da die Bilder anhand von POIs[4] gemacht wurden, ist es relativ einfach sie in der realen Umgebung wieder zu erkennen. POIs sind Orte oder Merkmale, die für den Benutzer Bedeutung haben können oder einfach zu merken sind. Im Vorteil versteckt sich gleichzeitig ein Nachteil. Der Benutzer kann seine reale Umgebung anhand eines Bildes auf dem mobilen Gerät dann nur schwer erkennen, wenn er sich in einer ungünstigen Position befindet bzw. die Umgebung aus dem falschen Blinkwinkel betrachtet.

2.1.1.4. MagicMap

Bei MagicMap [MagicMap] handelt es sich um eine Indoor Positionsbestimmung mit Hilfe gängiger Kurzstrecken-Funkstandards, wie Bluetooth, RFID oder WLAN. Wegen der Vielfältigkeit an Funkstandards, kann man MagicMap in vielen unterschiedlichen Umgebungen einsetzen, egal ob eine oder gleich mehrere bestehende Infrastrukturen vorhanden sind. MagicMap besitzt in vielen Aspekten große Ähnlichkeit zum WiPoD. MagicMap wie WiPoD benutzen WLAN zur Positionsberechnung, sind kartenbasierte Anwendungen und sind zum mobilen Einsatz gedacht. Da MagicMap aber auf einem modularen Konzept basiert, besitzt MagicMap im Gegensatz zu WiPoD den Vorteil, dass es einfacher ist, das System zu erweitern.

[4] Point-of-interest.

In der ersten Version enthielt MagicMap fünf wesentliche Module:

- Einen GUI[5] Modul für das Anzeigen von Position (auf der Karte).
- Einen Stumbler[6] für das Sammeln aller Signalinformationen, die für die Berechnung nötig sind.
- Einen Server für die Verteilung der Daten zwischen Client und das Speichern der Referenzposition.
- Eine P2P-Kommunikation[7] für den Datenverkehr zwischen Client auch in infrastrukturlosen Umgebungen.
- Eine Positioning Engine für die eigentliche Positionsberechnung.

Beim letzten Update (ab Version 1.0) wurde ein wichtiges Modul in MagicMap integriert, der sogenannte Tracker. Der Tracker hat die Aufgabe, die Positionsdaten zu beobachten und bei vorgegebenen Mustern voreingestellte Aktionen zu triggern. Über den Tracker wird in Kapitel 2.2.3 genauer erklärt.

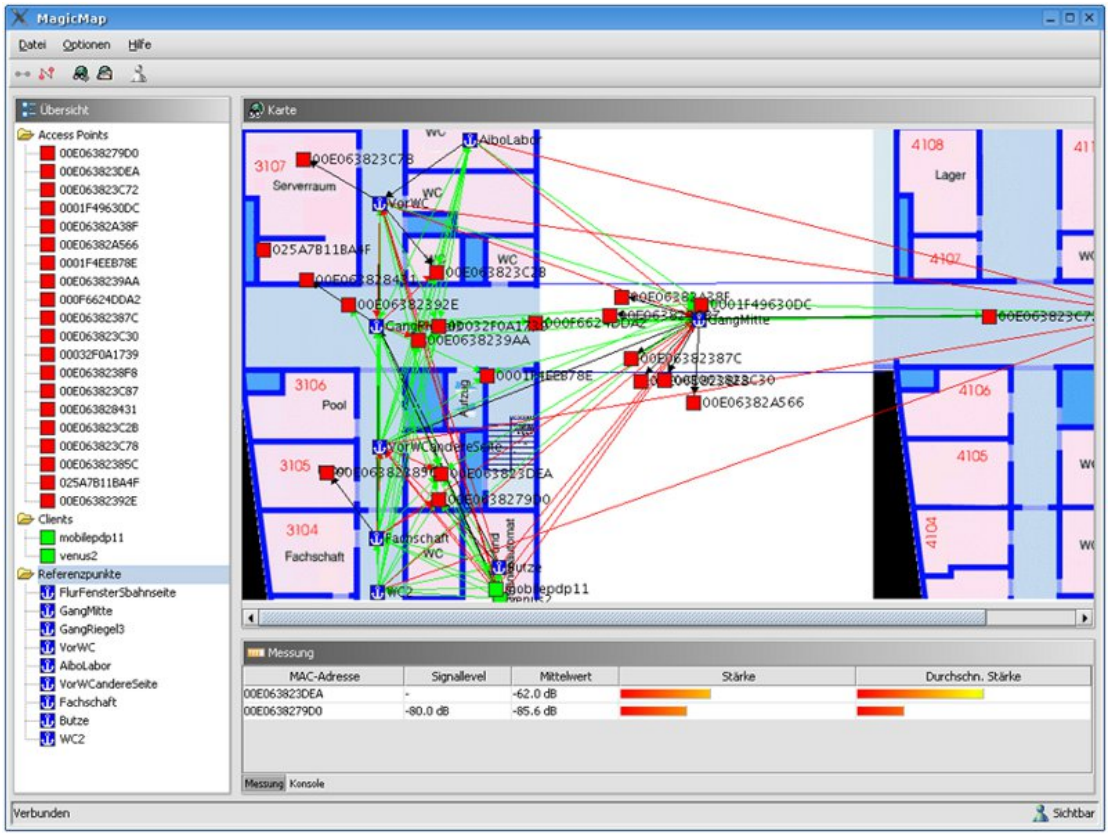

Abbildung 2.5 MagicMap Benutzeroberfläche

[5] Graphical User Interface.
[6] Ein Software, das für das Aufspüren von WLAN Signalen zuständig.
[7] Peer-to-Peer Kommunikation ist eine Kommunikation unter Gleichen, hier bezogen auf Endanwendungen, Rechner, Smartphones und so weiter. Weitere Informationen sind zu finden in [Schoder, u.a. 2002].

Wie in Abbildung 2.5 zu sehen, werden bei MagicMap alle möglichen Objekte, die sich auf derselben Karte befinden, angezeigt (als Standardkonfiguration). Dies kann nachträglich angepasst werden.
Die Genauigkeit von MagicMap variiert zwischen weniger als 1m und 10m. Das Ganze hängt von der Voreinstellung (Kalibrierung) vor dem Einsatz ab. Je feiner man kalibriert (viele Angaben von Referenzpunkten, Signalquellenpositionen, usw.), desto genauer wird die Positionsbestimmung.
Die Nachteile sind unter anderem, dass MagicMap nur bis Version 0.9 ein Open Source Projekt war. Alle späteren Versionen sind leider nicht mehr frei zugänglich (Shareware). Weiterhin sollte der Einsatz von MagicMap auf einem gut ausgestatteten Smartphone erfolgen, da die Berechnung auf dem Client passiert und zugleich mit dem Server synchronisiert wird.

2.1.1.5. Andere Projekte

Neben den oben dargestellten vier Arbeiten (Kapitel 2.1.1.1 – 2.1.1.4), gibt es noch viele nennenswerte andere Projekte, wie zum Beispiel:

Ekahau [Ekahau] : funktioniert im Prinzip wie MagicMap, ist eine kartenbasierte Anwendung, hat eine Genauigkeit im Bereich von einem Meter bei der Positionsbestimmung, die Berechnung findet im Server statt, was dazu führt, dass das System nicht gut skalierbar ist. Der größte Nachteil ist, dass es ein kommerzielles System ist und damit mit großen Anschaffungskosten verbunden ist.

RADAR [Bahl, u.a. 2000] : wurde von Microsoft entwickelt, benutzt Funksignale wie IMAPS, ist eine kartenbasierte Anwendung, benutzt K-Nearest Neighbour Berechnungsverfahren, die Genauigkeit der Positionsbestimmung liegt bei drei Meter. Es besitzt aber deutliche Nachteile: zum einen ist es nicht für Multi-User ausgelegt, zum anderen ein Forschungssystem.

2.1.2. Partielle Ortung

Unter partieller Ortung ist zu verstehen, dass man seine Position nur innerhalb eines bestimmten Bereichs exakt bestimmen kann. Zwischen den Bereichen ist über die aktuelle Position keine Aussage möglich.

2.1.2.1. Indoor Navigation on Distributed Stationary Display Systems

Die Idee hinter dieser Arbeit besagt, die vorhandene Infrastruktur (öffentliche Bildschirme, Daten, Systeme, usw.) als ein Hilfsgerät zur Positionsbestimmung zu

verwenden. Für die Positionsbestimmung bei Indoor Navigation on Distributed Stationary Display Systems [Ruppel, u.a. 2009] muss eine Vielzahl von Gegenständen in die Berechnung einbezogen werden, wie z.B.: die räumliche Struktur des Gebäudes, benutzbare Eingänge, unpassierbare oder nur im Notfall benutzbare Wege (Fahrstuhl beim Feuer oder Notausgänge).

Es wird angenommen, dass heutzutage in vielen öffentlichen Einrichtungen schon Bildschirme vorhanden sind. Als stationäre Geräte benötigen diese Bildschirme im Gegensatz zu ihren mobilen Pendants keine dynamische Positionsbestimmung, die sowohl auf dem mobilen Gerät als auch auf dem Server stattfinden müsste. Aus diesem Grund kann eine relativ hohe Genauigkeit erreicht werden. Denn der Benutzer muss bei der Benutzung einer solchen Anwendung direkt vor der Infrastruktur (s.Abbildung 2.6) stehen bzw. sehr nah sein.
Ein anderer Vorteil wäre es zu erwähnen, dass der Benutzer seine Umgebung gut erkennen kann, da auf dem Bildschirm ein einfacher Stockwerkplan und darauf die POIs und auch ein Weg zu den nächst gelegenen Einrichtungen (z.B: öffentlicher Bildschirm oder wie in Abbildung 2.6) angezeigt werden.

Zurzeit wird ein solches Indoor Navigation on Distributed Stationary Display System im Münchener Flughafen eingesetzt. Dort kann das System viele Passagiere bspw. bei der Suche nach dem Anschlussflug unterstützen. Bei diesen Geräten kommt ein RFID-Kartenleser zum Einsatz, so dass das System schnell auf die Anforderung des Benutzers reagieren kann. Dies wird erreicht, indem die für den Passagieren notwendige Fluginformationen und mögliche POI zwischen Ankunft- und Abflugterminal auf dem Server mit einer eindeutigen Nummer gespeichert werden. Diese Nummer passt zur Nummer, die in jeder RFID-Karte gespeichert ist. Jedes Mal, wenn die Karte gelesen wird und nachdem die passenden Daten zur ID-Karte auf dem Server gefunden wurden, werden die benötigten Informationen sofort angezeigt.

Abbildung 2.6 Beispiel eines öffentlichen Bildschirmes

2.2. Personensuche

Personensuche ist ein Teil der Location-Based-Service Anwendung, bei dem zwei oder mehrere Personen beteiligt und wechselseitig am Aufenthaltsort des Anderen interessiert sind. Ein Beispiel wäre: Suche alle Freunde in meiner Liste innerhalb des elften Stockwerkes. Das kann realisiert werden, z.B.: durch Anzeigen der Position von Freunden auf der Karte[8] des elften Stockwerkes oder etwa mit Hilfe einer Nachricht über den Aufenthalt der Freunden[9]. Wobei für diese Arbeit vor allem das zweite Beispiel interessant ist.

Hier muss beachtet werden, auf welche Weise man eine solche Benachrichtigung bekommen kann.
Zurzeit gibt es zwei verschiedene Verfahren, um den aktuellen Aufenthaltsort eines Benutzers zu erhalten, die sind:

- *Push Dienste*
  Ein Benutzer bekommt Informationen[10], ohne dass der Benutzer aktiv nachfragen muss. Z.B.: wenn die erwartete Person das Gebäude eintritt, bekommt diese Person eine Nachricht, wo sich die wartende Person befindet. Normalerweise muss sich der Benutzer vorher für den Dienst registrieren.

- *Pull Dienste*
  Im Gegensatz zu Push Dienst muss der Benutzer selbst die Informationen aktiv nachfragen. Z.B.: wenn die erwartete Person das Gebäude betritt, schickt diese Person eine Anfrage zum System, wo sich die wartende Person befindet.

Im folgenden Unterkapitel werden einige existierende Arbeiten vorgestellt und im Bezug auf Push und Pull Lösung untersucht.

2.2.1. IFEN-Indoor

Das Ziel von IFEN-Indoor [Erwin, Loehnert 2008] ist es, den Einsatz der bestehenden Galileo-Satelliten (GPS Satellit), die vorher nur im Outdoor-Bereich eingesetzt wurden, ebenso im Indoor-Bereich zu verwenden. Um die Galileo-Satelliten für die Berechnung der Position zu verwenden, wird eine klare und direkte Sichtbarkeit zwischen dem Empfängergerät[11] und dem Satellit benötigt. Deshalb ist es leider aus leicht ersichtlichen Gründen nicht ohne weiteres möglich, die Galileo-Satelliten für die Berechnung im Indoor-Bereich zu verwenden.

---

[8] Bei dieser Arbeit wird generell eine kartenbasierte Benutzeroberfläche benutzt.
[9] Die Inhalt der Nachricht könnte z.B. lauten: Herr Weiß ist in 11. Stock.
[10] In diesem Zusammenhang ist die Nachricht die Rede sein.
[11] Z.B. Smartphone mit GPS Unterstützung.

Um das trotzdem zu ermöglichen, müssen spezielle Technologien eingesetzt werden. Z.B. mit Hilfe von speziellen hoch sensiblen Signaltechnologien und entsprechendem Signalempfänger. Beide Technologien wurden von IFEN entwickelt.
IFEN-Indoor befindet sich zurzeit noch im Forschungsstufe. Sein späterer Einsatzbereich soll in sicherheitssensitiven und/oder professionellen logistischen Bereichen werden.

Alle Berechnungen und Ortungen finden beim Client statt. Da die Positionsbestimmung im Indoor Bereich komplizierter ist, als im Outdoor Bereich (z.B. wegen Notwege, Aufzug, unzugängliche Wege, usw.), müssen diese zusätzlichen Informationen über das Gebäude gespeichert werden.
Dazu gehören auch Informationen über Materialien, sowie deren Durchlässigkeit, da solche Signalen bei ihrem Weg durch Wände, Türen, usw. abgeblockt bzw. verfälscht werden. Was die Genauigkeit angeht, liegt die Positionsbestimmung der IFEN-Indoor im Bereich von drei Metern.

IFEN-Indoor wurde entwickelt, um eine ortsabhängige, ereignisbasierte Anwendung zu ermöglichen. Dazu stützt sich das System auf die Pull Methoden Variante, d.h. der Benutzer muss sich aktiv für ein Ereignis abfragen.
Ein Administrator sitzt im Hintergrund des IFEN-Indoor-Systems, um eine mögliche Abfrage von einem Benutzer zu verarbeiten. Wenn eine Abfrage ankommt, bearbeitet der Administrator alle möglichen Informationen bezüglich der Position des Benutzers, um sie danach an den Benutzer weiterzuleiten. Die Rolle des Administrators kann von Menschen oder Maschinen übernommen werden.

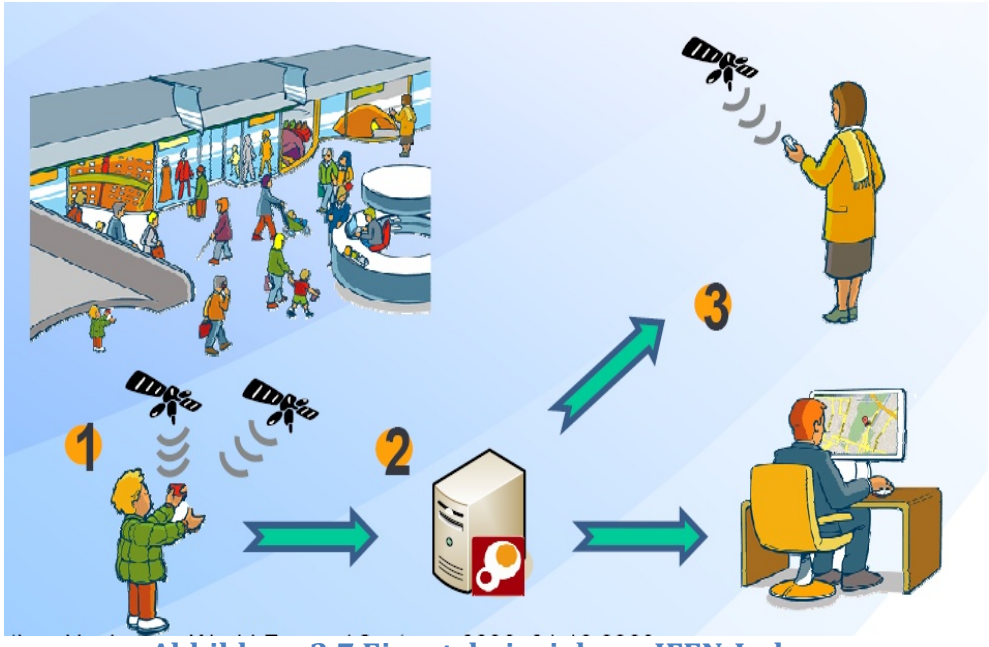

Abbildung 2.7 Einsatzbeispiel von IFEN-Indoor

In Abbildung 2.7 ist ein mögliches Szenario abgebildet. Auf einer Messe wurde ein Kind unabsichtlich von den Eltern getrennt. Das Kind trägt ein Empfängergerät bei sich, das mit dem IFEN-Indoor System verbunden ist. Mit Hilfe dieses Gerätes wird die Position des Kindes an den Server gesendet. Die Eltern stellen eine Anfrage[12] an den Server. Dieser Server[13] schickt die Information bezüglich der Position des Kindes auf das Gerät der Eltern.

Über diesen Anwendungsfall und weitere Anwendungsfälle, die von IFEN-Indoor entwickelt wurden, gibt es zurzeit nur wenige Informationen, deswegen kann hier nicht tiefer und ausführlicher darauf eingegangen werden.

2.2.2. FriFi – GPS Friend Finder and Free SMS Style Chat

FriFi [FriFi] ist eine Beispielanwendung für die ortsabhängige ereignisbasierte Anwendung. Im FriFi bekommt der Benutzer automatisch eine Warnung auf sein Smartphone, wenn sich eine für ihn interessierende Person in seinem Umkreis auftaucht. Es ist nur für den Outdoor-Bereich geeignet, da die Anwendung Galileo-Satelliten für die Positionsbestimmung benutzt und außerdem nur das Kartenmaterial von Google Maps nutzt. Da Google Maps jedoch nur eine normale Weltkarte beinhaltet, kann schon aus diesem Grund kein Stockwerkplan angezeigt werden, der für den Indoor-Bereich Einsatz unerlässlich ist.
Der andere Nachteil ist, dass FriFi nicht plattformunabhängig (nur für Iphone) ist und keine Möglichkeit besteht, eigene Anwendungen auf FriFi weiter zu entwickeln.

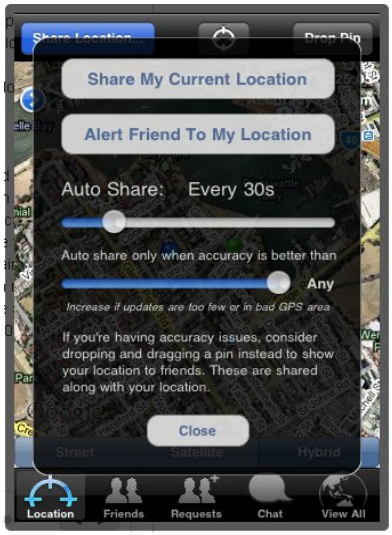

Abbildung 2.8 Benutzeroberfläche von FriFi

[12] Z.B. „Zeigt mir die Position meines Kindes“.
[13] In diesem Fall wird die Anfrage von einem menschlichen Administrator bearbeitet.

Abbildung 2.8 zeigt ein Beispielbild, das FriFi dem Benutzer anbietet. Der Benutzer besitzt die Möglichkeit, über den „Alert Friend To My Location“ Knopf ein bestimmtes Ereignis zu abonnieren. Wenn eine Person, die in der Freundliste des Benutzers steht, auf die Karte auftaucht, bekommt der Benutzer auf seinem Gerät eine automatische Warnung. Dies ist ein Vorteil im Gegensatz von IFEN-Indoor, da der Benutzer nicht aktiv ein bestimmtes Ereignis nachfragen muss.

2.2.3. MagicMap Tracker

In diesem Unterkapitel wird die MagicMap Anwendung ein weiteres Mal diskutiert. Dies liegt darin begründet, dass MagicMap anhand seiner modularen Systemsarchitektur in der Lage ist, mit weiteren interessanten Anwendungen ausgebaut zu werden. Ein solches zusätzliches Modul ist der Tracker. Der Tracker übernimmt die Aufgabe eines Push Diensts. Der Benutzer gibt dem Tracker mit Hilfe der Benutzeroberfläche (Abbildung 2.9) alle Ereignisse ein, die für ihn interessant sind, sowie den dazugehörigen Vorgang, welcher durchgeführt werden muss, sobald das Ereignis eintritt.

Abbildung 2.9 Benutzeroberfläche für das Eintragen vom Ereignis und dessen durchzuführende Vorgang

Hinter dem Tracker verbirgt sich die Jadex Event Stream Processing Architektur, JESPA [JESPA]. JESPA ist eine verteilte, Agenten-basierte Architektur zur Verarbeitung von Ereignisströmen. Die eigentliche Engine, die für die Verarbeitung von hunderttausenden Ereignissen pro Sekunde zuständig ist, nennt sich Esper [ESPER]. JESPA wird von MagicMap gesammelten Messwerten versorgt, aus denen benutzerdefinierte komplexe Ereignismuster extrahiert werden.

Der Tracker benutzt unter anderem die folgende MagicMap-Schnittstellen:

- die Adapter zu Aufzeichnung lokal empfangener Signalstärken und weitere Sensordaten
- die Schnittstelle zur Positionsberechnung
- die Webservice-Schnittstelle, über die ein MagicMap-Client Objektdaten mit anderen Clients austauscht.

Ein konkretes Beispiel für den Tracker wäre z.B.: Ein Benutzer hat ein Ereignismuster definiert, und zwar "Benachrichtige mich, wenn Herr Mustermann vor dem Eingang steht". Der Benutzer braucht nur noch in diesem Zeitpunkt zu registrieren und interessiert sich nicht, wann Herr Mustermann ankommt (Push Dienst Verfahren). Wenn der gemeinte Herr Mustermann vor dem Eingang, bekommt der Benutzer eine Benachrichtigung (der Benutzer bekommt die Nachricht geschoben). Das ist ein bedeutender Vorteil des Trackers.

2.3. Zusammenfassung

In diesem Kapitel wurden mehrere vorhandene Technologien vorgestellt und in Betracht gezogen, die für diese Arbeit von Nutzen sein könnten. Für den Anfangsteil des Szenarios (s. Kapitel 3), in dem die erwartete Person das Gebäude betritt und der wartenden Person eine Erinnerung gesendet wird, reicht durchaus die partielle Ortung aus. Danach müsste die kontinuierliche Ortung zum Einsatz kommen (genauere Erklärung ist im Kapitel 3), da die partielle Ortung die Position nur innerhalb eines bestimmten Bereichs bestimmen kann. In diesem Sinne ist kontinuierliche Ortung vom Vorteil verglichen zur partiellen Ortung, da kontinuierliche Ortung die Aufgabe der partiellen Ortung abdeckt.

Im Kapitel 2.2 sind die Beispielanwendungen, welche die kontinuierlichen Ortung benutzen und noch dazu die zweite Anforderung[14] erfüllen, zu sehen, daher werden nachfolgend nur die genannten drei Beispielanwendungen betrachtet. IFEN-Indoor verwendet die Pull Dienste[15], FriFi und MagicMap Tracker verwendet die Push Dienste. Die Anwendung soll der Einfachheit für den Benutzer halber die Push Dienste bieten.

FriFi ist für den Outdoor-Bereich Einsatz gedacht, benutzt Google Maps als Benutzeroberfläche und verwendet nur GPS-Signale zur Positionsbestimmung.
Außerdem ist FriFi ein Endprodukt und leider keine Architektur, mit deren Hilfe weitere Anwendungen entwickelt werden können. Deshalb besteht bei FriFi leider keine Möglichkeit, um den Einsatz für den Indoor-Bereich nachzurüsten bzw. weiterzuentwickeln.

---

[14] Wie man den Anderen finden kann.
[15] Aus dem Szenario im Abbildung 2.7.

IFEN-Indoor setzt sich im Indoor-Bereich Einsatz durch und benutzt WLAN-Signale und ein speziell entwickeltes Signal zur Positionsbestimmung. Allerdings hat IFEN-Indoor einen Nachteil, und zwar, der Benutzer muss die für ihn interessanten Informationen aktiv nachfragen.

MagicMap (inkl. Tracker) beschäftigt sich, wie IFEN-Indoor im Indoor Bereich Einsatz und kann viele verschiedene Signalarten zur Positionsbestimmung verwenden, einschließlich WLAN-Signale. Mit Hilfe des Tracker-Moduls ist MagicMap zudem in der Lage, einen Push Dienst zu bieten.

Hiermit ist es MagicMap möglich, alle offene Anforderungen zu erfüllen. Deshalb dient MagicMap als Basis dieser Arbeit.

3. Analyse

In diesem Kapitel werden die Anforderungen an das System analysiert. Diese Anforderungen werden in zwei Unterkapitel geteilt, funktionale Anforderungen und nicht-funktionale Anforderungen.
Anhand des Beispielszenarios (s. Kapitel 3.2) wird der Umfang dieser Arbeit eingegrenzt sowie die funktionalen und nicht-funktionalen Anforderungen erläutert.
Die funktionalen Anforderungen (s. Kapitel 3.3) sollen mit Hilfe von Use-Case Diagrammen und der Unified Modeling Language (UML) modelliert und analysiert werden.
Die nicht-funktionalen Anforderungen (s. Kapitel 3.4) werden anhand der Software-Qualität analysiert.
Beiden Anforderungen werden danach in kommenden Kapiteln näher und detaillierter erläutert.

3.1. Anmerkungen

In diesem Unterkapitel werden die Vorbedingungen erwähnt und Definition einiger notwendiger Begriffe geliefert.

Die Vorbedingungen sind:

- Zur Vereinfachung werden die erwartete Person als Person A und die wartende Person als Person B bezeichnet.
- Um das System zu benutzen, muss der Benutzer sich erst einmal an das System registrieren.
- MagicMap CE-Client[16] muss auf dem Smartphone installiert sein.

Folgende Hardware werden benötigt:

- *Access Point(s)*[17]
  Access Points werden benutzt, um die WLAN-Signale, die für Positionsberechnung nötig sind, auszustrahlen. Außerdem wird die Access Points für die Kommunikation zwischen Client und Server benutzt.

[16] MagicMap Client für Windows Mobile Smartphone
[17] Auch bekannt unter der Abkürzung AP

- *Smartphones*
  Smartphones werden für die Positionsbestimmung benötigt. Außerdem dienen die Smartphones zur Groborientierung. In diesem Fall ist es zur Anzeige der Nachricht „Person A betritt das Gebäude“ oder „Person B ist im 11. Stockwerk“.

- *Öffentlicher Bildschirm*
  Öffentliche Bildschirme sind zur Feinorientierung gedacht. Feinorientierung bedeutet das Anzeigen der Position der Person B auf einem Grundriss eines Stockwerkes mit Hilfe eines Tokens. Jedes Stockwerk ist mindestens mit einem solchen Bildschirm ausgerüstet.

Technische Begriffe, die in diesem Kapitel benutzt werden, sind:

- Token
  Token ist eine Markierung. Ein Token wird benutzt um eine Person auf einem Grundriss, der auf einem öffentlichen Bildschirm angezeigt ist, zu identifizieren bzw. zu unterscheiden. Z.B.: ein farbiger Punkt.

- Ereignismuster
  Ereignismuster ist eine Datei im XML-Format. Das Muster beschreibt, was ein Tracker (s. Kapitel 2.2.3) genau tun soll, wenn ein bestimmtes Ereignis auftritt. Die Auswertung der Datei wird intern im Tracker geregelt. Die genauere Erklärung folgt im Kapitel 4.

3.2. Mögliches Szenario

Ein mögliches Szenario, das für diese Arbeit am besten geeignet ist, ist eine Verabredung zwischen zwei Personen, z.B. im HAW Informatik-Gebäude. Das Gebäude besitzt vierzehn Stockwerke mit ca. zwanzig Räumen. In jedem Stockwerk ist ein öffentlicher Bildschirm vorhanden, der vor den Aufzugtüren montiert ist.
Person A hat sich mit Person B an der HAW verabredet. Beide Personen tragen die Verabredung in ihrem Kalender ein.

Person B ist auf die Ankunft von Person A an der HAW vorbereitet. Person B schaltet sein Smartphone ein und startet die Verabredungsanwendung. Person B muss ab diesem Punkt nur noch auf das Ankommen der Person A warten.

Nach einiger Zeit kommt die Person A an der HAW und betritt sofort das Gebäude. Person B bekommt sofort eine Nachricht, die sie darüber informiert, dass Person A das Gebäude betreten hat. Zugleich bekommt Person A eine Nachricht, im welchen Stockwerk sich Person B zurzeit befindet. Z.B.: „Person B ist in 11. Stockwerk“. Person A bekommt zusätzlich ein Token von Person B zugesandt.

Person A steigt dann in den Aufzug ein und fährt in den elften Stock. Sobald Person A aus dem Aufzug aussteigt, sieht Person A den öffentlichen Bildschirm vor sich und tritt vor.

Sobald Person A vor diesem Bildschirm steht, wird ihm ein Grundriss des Stockwerkes auf dem Bildschirm angezeigt. Als Markierung für die Position der Person B wird ein Token im Grundriss angezeigt.
Person A merkt sich die Position und geht in dem Raum, in dem sich Person B gerade befindet.

Anhand dieses Szenario werden die Anforderungen der Anwendung erstellt (s. Kapitel 3.3 und Kapitel 3.4).

3.3. Funktionale Anforderungen

Funktionale Anforderungen beschreiben, was ein System mindestens leisten bzw. erledigen soll.
Mit Hilfe des Anwendungsfalldiagramms (s. Kapitel 3.3.1) wird eine Übersicht über die möglichen Dienste bzw. Aufgaben des Systems erläutert. Danach werden die beteiligten Akteure (s. Kapitel 3.3.2) gelistet und die jeweilige Anwendungsfälle (s. Kapitel 3.3.3) einzeln ausführlich erklärt.

3.3.1. Anwendungsfalldiagramm

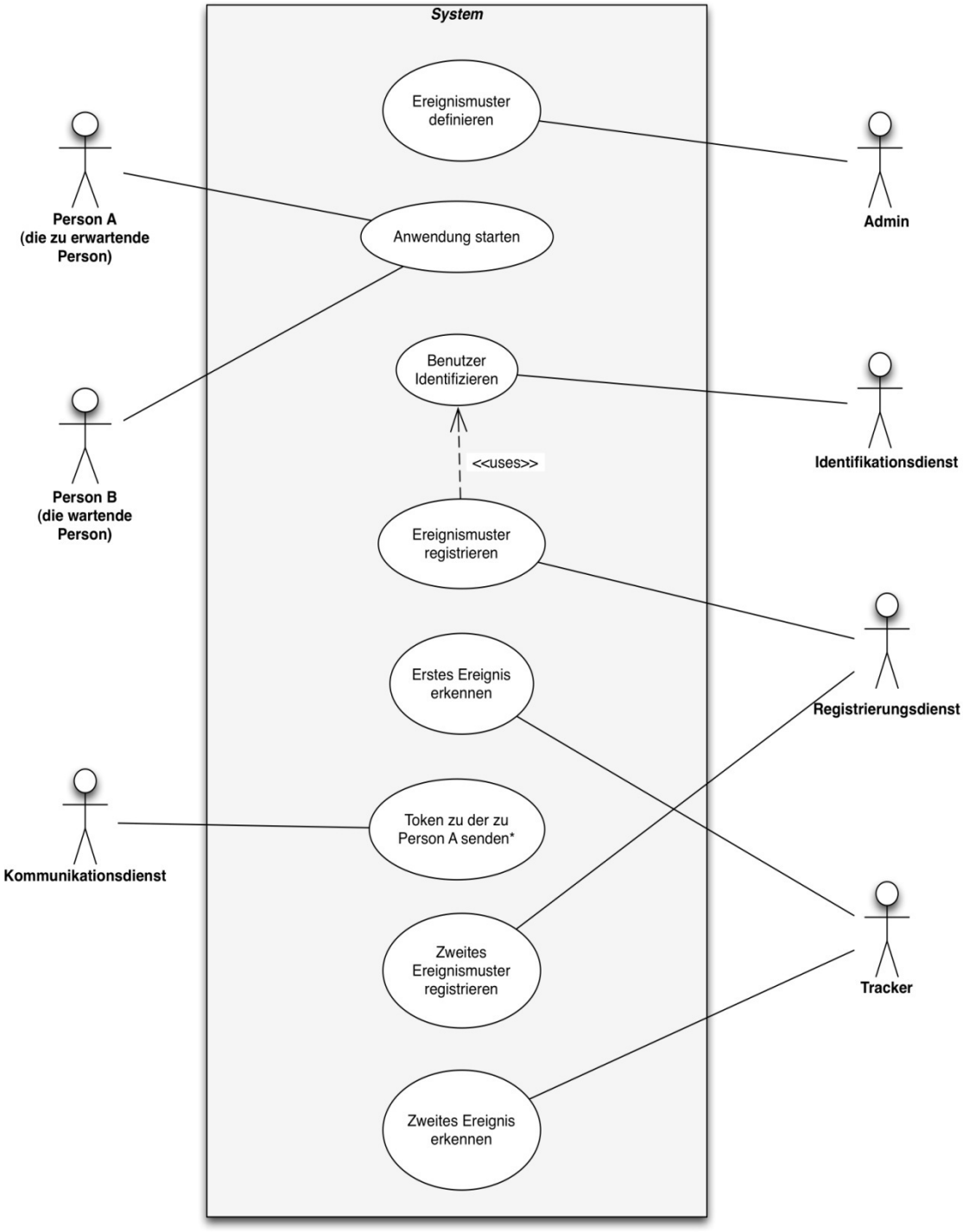

Abbildung 3.1 Use-Case Diagramm

In Abbildung 3.1 ist zu sehen, dass für dieses Szenario sieben beteiligte Akteure benötigt werden. Der Admin ist ein Akteur, der das Ereignis definiert, sodass der Tracker das Ereignis verstehen kann. Person A ist der Akteur, auf den Person B wartet. Der Tracker bietet einen ortsbasierten Überwachungsdienst, in dem der Tracker auf ein bestimmtes einzutretendes Ereignis wartet. Dieser Identifikationsdienst hat die Aufgabe, eine Person, die sich gerade an das System angemeldet hat, zu identifizieren, in dem die persönlichen mit den technischen Daten (MAC-Adresse) verknüpft werden. Dem Registrierungsdienst kommt die Aufgabe zu, die Ereignismuster zu initialisieren. Der Kommunikationsdienst hat eine wichtige Aufgabe, und zwar, das Tokens zu Person A zu senden.

Person A und Person B haben schon vorher verabredet, sich am Tag X zu treffen. Beiden Personen haben diese Verabredung auch in ihren Smartphones eingetragen. Sobald Person B MagicMap CE-Client gestartet hat, sammelt MagicMap CE-Client alle benötige Acces Points Signale und berechnet die eigene Position. Diese Positionsinformation wird mit allen anderen Informationen[18] in einer Datei zum Server gesendet. Wenn der Identifikationsdienst die Information in der Datei erkennt, gleicht sie der Identifikationsdienst mit seiner Datenbank ab. Die zugehörigen Informationen werden zum Registrierungsdienst weitergeleitet.
Der Registrierungsdienst instanziiert das Ereignismuster anhand der Informationen vor. Anschließend wird das Ereignismuster zum Tracker registriert.

Ereignismuster, die in dieser Arbeit nötig sind, sind:

- *Erkennung, wenn Person A das Gebäude betritt.*
  Wenn dieses der Fall ist, sendet der Tracker Person A eine Nachricht, in welchem Stockwerk sich Person B befindet, und Person B eine Nachricht, dass Person A das Gebäude betritt.

- *Erkennung, wenn Person A sich innerhalb eines bestimmten Bereichs und Stockwerks auf der Karte befindet (in der Nähe des Bildschirms).*
  Für unser Szenario bedeutet dies innerhalb eines ein Meter Bereich im Umkreis des Bildschirms. Sobald sich Person A dort befindet, blendet der Tracker die Position der Person B auf dem Bildschirm ein.
  Vorausgesetzt Person A befindet sich im richtigen Stockwerk.

Nach der Registrierung der Ereignismuster wartet nun der Tracker, bis ein Ereignis auftaucht und eventuell nachhinein seine Aufgabe ausführt.
Wenn Person A das Gebäude betritt wird dieses Ereignis vom Tracker erkannt. Dieser führt nun die im Ereignismuster registrierte Aktionsfolge aus (Nachricht zu beiden Beteiligten senden).

Wenn der Kommunikationsdienst, der in das Smartphone der Person B installiert wurde, diese Nachricht erkannt hat, schickt er ein Token zur Person A. Dieses Token

[18] Unter anderem: die eigene MAC-Adresse sowie der Name der anderen Beteiligten.

ist eindeutig und dient als eine Markierung für die Position der Person B später auf dem Bildschirm.
Zu gleicher Zeit wird das zweite Ereignismuster instanziiert und für die Registrierung am Tracker bereitgestellt. Der Tracker wartet dann auf das nächste Auftreten des Ereignismusters.

Person A geht in das genannte Stockwerk (in diesem Fall mittels eines Aufzugs). Person A steigt aus dem Aufzug in das richtige Stockwerk. Vor dem Aufzug befindet sich ein Bildschirm und Person A nähert sich diesem an.
Wenn Person A im richtigen Stockwerk in der Nähe des Bildschirms steht, erkennt der Tracker das zweite Ereignis. Das Ereignis bewirkt eine Aktion: der Grundriss des Stockwerkes erscheint auf dem Bildschirm. In diesem Grundriss ist die Position der Person B mit dem Token von Person B markiert.

Die genaue Beschreibung der Akteure und die Anwendungsfälle werden nachher in den Kapiteln 3.3.2 und 3.3.3 beschreiben.

3.3.2. Akteure

Anbei werden die Akteure, die in Abbildung 3.1 gezeichnet sind, kurz erläutert.

Akteur	Beschreibung
Admin	Der Admin muss zuerst das Ereignismuster definieren, sodass das oben genannte Szenario gestartet werden kann.
Person A	Diese Person ist die erwartete Person. Die Person interessiert sich für die Position der anderen Person (Person B).
Person B	Diese Person ist die wartende Person. Die Person interessiert sich für das Ankunft der gewarteten Person (Person A).
Tracker	Ein wichtiges Modul, das die Rolle eines Beobachters übernimmt. Der Tracker ist ein MagicMap[19] Plugin, das allen Datenströmen lauscht, mit denen die MagicMap Anwendung in kontinuierlichen Zeitintervallen versorgt wird. Der Tracker arbeitet anhand eines Ereignismusters. Dieses Muster wird aus einem Datenstrom gefiltert. Wenn der Tracker das Muster erkannt, wird die vorgegebene Aktionsfolge durchgeführt werden.

[19] MagicMap unterscheidet sich mit MagicMap CE-Client und nicht auf dem Smartphone des Benutzer installiert, sondern auf einem Rechner

Identifikationsdienst	Ein Dienst, der einen Benutzer identifiziert. Der Identifikationsdienst ist ein MagicMap-Plugin und lauscht allen Datenströmen (wie der Tracker). Der Identifikationsdienst hat die Aufgabe, der Benutzer anhand der MAC-Adresse oder Namen zu identifizieren, indem er die MAC-Adresse bzw. Namen des Beteiligten mit der auf dem Server eingetragene MAC-Adresse bzw. Namen verglichen werden. Anschließend werden die restlichen Informationen herausgegeben, z.B.: die Telefonnummer.
Kommunikationsdienst	Ein Dienst, der für das automatische Senden des Tokens zwischen den Beteiligten zuständig ist.
Registrierungsdienst	Ein Dienst, der für die Vorbereitung und die Registrierung und Instanziierung des Ereignismuster zuständig ist.

Tabelle 3.1 Akteure

3.3.3. Anwendungsfälle

In diesem Unterkapitel werden die einzelnen Funktionen des Systems (s. Abbildung 3.1) in tabellarischer Form detailliert und systematisch beschrieben. Im Anschluss daran werden die Vorbedingungen zu jede Use Case erläutert. Die Nachbedingungen, die nach der korrekten Beendigung des Use Cases passieren sollen, werden auch aufgelistet.
Anschließend werden mögliche Ausnahmen, die eventuell auftreten könnten, beschrieben und einzeln betrachtet, sowie die beteiligten Akteuren und Auslöser des Use Cases eingetragen.

Use Case 1: Ereignismuster definieren	
Beschreibung des Ablaufs	Am Anfang definiert der Admin das Ereignismuster, so dass der Tracker das Ereignis, welches im Ereignismuster beschrieben wurde, bearbeiten kann.
Akteure	Admin.
Vorbedingungen	Keine.
Nachbedingungen	Der Tracker kann das Ereignismuster bearbeiten.
Auslöser	Beim Aufbau des Systems definiert ein Admin das Ereignismuster.
Ausnahmen	Keine.

Tabelle 3.2 Ereignismuster definieren

Use Case 2: Verabredungsszenario starten	
Beschreibung des Ablaufs	In diesem Use Case werden viele Vorgänge durchgeführt. Sobald der Benutzer die Anwendung gestartet und seinen Benutzernamen und Passwort eingegeben hat, wird die Positionsberechnung durchgeführt und danach zum Server übergetragen. Dabei wird auch der Name von Person A und von Person B gesendet
Akteure	Benutzer.
Vorbedingungen	Beide Beteiligten müssen sich vor der erste Nutzung beim System anmelden bzw. registrieren.
Nachbedingungen	Positionsdaten der Person an den Server gesendet.
Auslöser	Der Benutzer schaltet die Anwendung ein.
Ausnahmen	Wenn keine WLAN-Verbindung vorhanden ist, wird dies dem Benutzer angezeigt.

Tabelle 3.3 Verabredungsszenario starten

Use Case 3: Benutzer identifizieren	
Beschreibung des Ablaufs	Sobald der Identifikationsdienst die Namen der beteiligten Personen aus den Verabredungsdaten gelesen hat, findet der Dienst die MAC-Adresse und schreibt die zugehörige MAC-Adresse in das Ereignismuster ein. Der Benutzer wird später vom Tracker anhand von dessen MAC-Adresse erkannt.
Akteure	Identifikationsdienst.
Vorbedingungen	Die Namen der beteiligten Personen sind im Datenpaket zu sehen.
Nachbedingungen	Die passenden Informationen sind gefunden worden.
Auslöser	Registrierungsdienst braucht die technischen Größen (MAC-Adresse), so dass ein Ereignismuster instanziiert werden kann.
Ausnahmen	Wenn die Namen bzw. die MAC-Adresse nicht bereit auf dem Server vorhanden sind, muss dies dem Benutzer mitgeteilt werden.

Tabelle 3.4 Benutzer identifizieren

Use Case 4: Ereignismuster registrieren	
Beschreibung des Ablaufs	Sobald das Ereignismuster instanziiert wurde[20], kann das erste Ereignismuster mit dem Ereignis „Person A betritt das Gebäude“ umgehend vom Tracker gelesen werden. Der Tracker erkennt das Ereignismuster und speichert es.
Akteure	Registrierungsdienst.
Vorbedingungen	Alle Informationen, die das Ereignismuster benötigt, sind vorhanden.
Nachbedingungen	Erstes Ereignismuster ist beim Tracker registriert.
Auslöser	Der Registrierungsdienst erkennt die Informationen zur Instanziierung des Ereignismusters.
Ausnahmen	Keine.

Tabelle 3.5 Ereignismuster registrieren

Use Case 5: Erstes Ereignis erkennen	
Beschreibung des Ablaufs	Sobald der Tracker das Ereignis erkannt hat, wird der Tracker bestimmte Aktionen, die ihm in dem Ereignismuster eingegeben worden sind, durchführen. In diesem Fall wird eine Nachricht zu beiden Personen gesendet. Die Nachricht lautet z.B.: Person B bekommt eine Nachricht „Person A ist angekommen“ und Person A bekommt eine Nachricht „Person B befindet sich im 11. Stockwerk“.
Akteure	Tracker.
Vorbedingungen	Das Ereignismuster ist schon registriert worden. Person A hat das Gebäude betreten.
Nachbedingungen	Beide beteiligte Personen bekommen eine Nachricht.
Auslöser	Person A betritt das Gebäude.
Ausnahmen	Keine.

Tabelle 3.6 Erstes Ereignis erkennen

[20] Werden im Kapitel 4 ausführlich erklärt.

Use Case 6: Token zur Person A senden	
Beschreibung des Ablaufs	Sobald die Nachricht zur Person A gesendet wurde, sendet der Kommunikationsdienst umgehend das Token zur Person A, das für die Erkennung der Position von Person B auf dem Bildschirm zuständig ist.
Akteure	Kommunikationsdienst
Vorbedingungen	Person A hat das Gebäude betreten und Person B hat die Nachricht „Person A betritt das Gebäude“ bekommen. Die E-Mail bzw. Telefonnummer von Person A ist bekannt.
Nachbedingungen	Person A bekommt das Token und anhand dieses Tokens kann er später die Position von Person B auf dem Bildschirm erkennen.
Auslöser	Der Kommunikationsdienst erkennt, dass Person B eine Nachricht bekommen hat.
Ausnahmen	Keine.

Tabelle 3.7 Token zur Person A senden

Use Case 7: Zweites Ereignismuster registrieren	
Beschreibung des Ablaufs	Sobald das erste Ereignis eingetreten ist, wird das zweite Ereignismuster (mit Ereignis „Person A steht vor dem Bildschirm“) instanziiert und umgehend dem Tracker mitgeteilt. Der Tracker erkennt das Ereignismuster und speichert es.
Akteure	Registrierungsdienst.
Vorbedingungen	Alle Informationen, die für die Instanziierung des Ereignismusters sind, sind vorhanden.
Nachbedingungen	Das zweite Ereignismuster ist registriert.
Auslöser	Der Registrierungsdienst sendet die Ereignismuster.
Ausnahmen	Keine.

Tabelle 3.8 Zweites Ereignismuster registrieren

Use Case 8: Zweites Ereignis erkennen	
Beschreibung des Ablaufs	Wenn Person A vor dem Bildschirm steht, wird auf dem Bildschirm ein Grundriss angezeigt. In diesem Grundriss erscheint ein Token. Das Token symbolisiert die Stelle, wo Person B sich zurzeit befindet.
Akteure	Tracker.
Vorbedingungen	Person A steht vor dem Bildschirm im richtigen Stockwerk.
Nachbedingungen	Die genaue Position der Person B wird auf dem Bildschirm angezeigt.
Auslöser	Die Person A ist in der Nähe von Bildschirm (steht innerhalb von 1m Bereich im Umkreis des Bildschirms).
Ausnahmen	Person A ist in dem falschen Stockwerk, dann auf dem Bildschirm des Stockwerkes wird nichts angezeigt.

Tabelle 3.9 Zweites Ereignis erkennen

Das Einblenden des Tokens auf einem Grundriss auf dem Bildschirm wird in dieser Arbeit nicht weiter entwickelt. Dies soll als zukünftiges und weiteres Arbeitthema eingestuft werden.
In dieser Arbeit wird die Position allen Benutzern, die sich auf dem bestimmten Stockwerk befinden (z.B.: Stockwerk 11), auf dem Bildschirm dieses Stockwerkes angezeigt.

3.4. Nicht-funktionale Anforderungen

Anders als funktionale Anforderungen wird das System bei nichtfunktionalen Anforderungen als Ganzes betrachtet. Während bei funktionalen Anforderungen jede einzelne Funktionen genau betrachtet wird, verändert sich der Fokus bei nicht-funktionalen Anforderungen, z.B.: wie etwas gemacht wird, ob das System für den Einsatz geeignet und weitere mehr.

Abbildung 3.2 Umfang der Software-Qualität

Martin Glinz hat in seinem Vorlesungsfolien die Software-Qualität [Glinz, M 2005] in sechs Eigenschaften[21] unterteilt (s. Abbildung 3.2). Diese Unterteilung beschreibt die nicht-funktionale Anforderungen. Im folgenden Unterkapiteln werden die nicht-funktionalen Anforderungen im Bezug auf die Anwendung, die aus dieser Arbeit entstehen soll, genauer erläutert.

3.4.1. Funktionalität

Ein System soll nicht nur viele Funktionen anbieten, sondern muss auch richtige Ergebnisse liefern können. Dieses kann mit Hilfe von umfangreichen Tests erreicht werden. Der Test bezieht sich nicht auf einzelne Module, sondern als alles.

Es ist auch wichtig, wenn das System mit anderen Systemen gekoppelt wird, dass das System auch trotzdem immer noch richtige Ergebnisse liefert und geprüft werden kann. Damit soll erreicht werden, dass das System ein hoher Grad der Erweiterbarkeit erzielen kann.

[21] Diese Unterteilung ist eine von vielen möglichen Unterteilungen. Die Unterteilungen sind nicht immer auf genau diese Weise definiert. Möglich wäre es auch einige zusätzliche Eigenschaften miteinzubeziehen oder andere wegzulassen.

3.4.2. Zuverlässigkeit

Zuverlässigkeit oder Stabilität bedeutet, dass ein System in bestimmten Zeitintervallen seine Funktionen oder die ihm gestellten Anforderungen korrekt ableistet. Diese Eigenschaft ist leider nicht unmittelbar messbar.

Das System soll sichergestellt werden, dass das Ergebnis korrekt geliefert werden, selbst ein Fehler eingetreten ist, z.B.: mittels redundanter Komponenten. So kann das System die Störungen in einem bestimmten Intervall vernachlässigen und in einer Log-Datei für kommende Wartungen speichern. Vor allem ist dies wichtig bei einem Verhalten, das einen schweren Fehler verursachen kann, z.B.: ungewolltes Löschen von irgendeinem wichtigen Datei. Hier muss das System dem Benutzer eine verständliche und gut platzierte Warnmeldung anzeigen.

Zuletzt soll der Benutzer die Möglichkeit haben, nach dem Auftritt von (schweren) Fehlern das System wiederherzustellen.

3.4.3. Benutzbarkeit

Ein System soll nicht nur zuverlässig sein, sondern muss für den Benutzer auch verständlich sein. Z.B.: Ein Benutzeroberfläche mit zu vielen Auswahlknöpfe, zu stark gekürzte Informationen und noch vieles mehr. Diese bringen nur wenig Sinn für den Benutzer und führen dazu, dass der Benutzer das System nicht mehr richtig bedienen kann. Da das System auch für durchschnittliche Menschen[22] gedacht, spielt diese Eigenschaft eine große Rolle.

Das System muss außerdem intuitiv und einfach zu benutzen. Wenn das System doch noch kompliziert, muss das System so gestattet werden, so dass der Benutzer die Möglichkeit über das System zu erlernen, z.B.: mit verständlicher Hilfsinformation, einem Benutzerhandbuch oder ähnlichem.

Genauso einfach und intuitiv muss bereits die Installation des Systems ablaufen. Hiermit soll es ermöglicht werden, mit nur wenigen Schritten das System betriebsbereit einzurichten. Das ideale Fall bestünde darin, das System ohne Installation auf dem Client zum Laufen zu bringen.

[22] Keine technisch hochbegabten Menschen oder Personen mit begrenztem Verständnis von Technik.

3.4.4. Effizienz

Effizienz oder Performance ist eine Eigenschaft, die für den Benutzer eine große Rolle spielt. Z.B.: Eine Positionsbestimmung sollte nicht länger als fünf Sekunde geschehen, am besten nicht einmal eine Sekunde. Ein Zeitintervall von großer als fünf Sekunden belastet die Beweglichkeit des Benutzers. Ein Beispiel dafür: so dass ein Benutzer seine Position genau erfahren will, muss er jedes Mal für fünf Sekunde erstmal stillstehen und warten.

Besonders bei mobilen Geräten (PDAs oder Smartphones) ist diese Eigenschaft sehr wichtig. Dieser Nachteil kann minimiert werden, indem alle rechenintensiven Aufgaben, die viel Resourcen benötigen, möglichst nicht auf dem Client lagern oder die Kommunikation zwischen Server und Client nur noch auf das wesentliche bzw. nötigste beschränken.

Da mobile Geräten alle mit Akku betrieben werden, ist es auch von Wichtigkeit, dass es das System schafft, innerhalb dieser Akkulebensdauer die auszuführenden Tätigkeiten zu leisten.

3.4.5. Änderbarkeit

Änderbarkeit oder Wartbarkeit oder Erweiterbarkeit beschäftigt sich mit den Aspekten, z.B.: wie einfach bzw. wie schwer das System zu verbessern bzw. zu erweitern. Deshalb ist es auch wichtig, dass das System aus vielen kleinere Modulen aufzubauen. Der Hintergedanken besteht darin, dass ein einfaches und kleineres Modul weniger Aufwand für eine Entwicklung bzw. eine spätere Erweiterungen als ein großes Modul benötigt. Da bei einem modularen System gezielt in einem bestimmten Modul weitere Funktionen erweitern lässt.

Zum Beispiel in der Fall wenn ein System jetzt ein neuer Typ von Token[23] verwendet werden soll. Bei einem modularen System kann gezielt nur dasjeniges Modul, welches für die Kommunikation zwischen Server und Client verantwortlich ist, erweitert werden. Derartige Änderungen sind bei einem nicht-modularen System in der Regel mit einem hohen Aufwand verbunden.
Bei dieser Architekturart ist es ratsam, das System mit genügender und guter Dokumentation zu unterstützen.

[23] Marke bzw. Hilfsbezeichnung.

3.4.6. Übertragbarkeit

Mit Übertragbarkeit oder Portierbarkeit ist zu verstehen, dass das System möglichst plattformunabhängig[24] ist. Das bedeutet, um dieses zu erreichen, soll das System auch von einer plattformunabhängigen Programmiersprache entwickelt wird. Die am meisten und besten geeignete Sprache ist z.B. Java Programmiersprache, da Java von vielen bzw. fast allen gängigen Betriebssystem unterstützt und von vielen Entwicklern verwendet wird.

Es soll ebenfalls beachtet werden, sich für die Kommunikation zwischen Server und Client auf einen bestimmten Standard zu einigen, so dass die Fehler bei der Kommunikation minimiert werden können.
Als mögliche Lösung ist eine XML-basierte Austauschnachricht für die Kommunikation denkbar.

Anderer Punkt, die auch betrachtet werden kann, ist der Schwierigkeitsgrad der Installation eines Systems. Hiermit soll erreicht werden, dass bei allen Betriebssystemen das gleiche Verfahren für die Installation des Systems bzw. der gleiche Aufwand benötigt wird.

3.4.7. Sicherheit

Martin Glinz hat in seiner Folie die Sicherheitseigenschaft innerhalb der Funktionalitätseigenschaft behandelt, diese stellt aber aus Sicht des Autors eine wichtige Eigenschaft der Software-Qualität dar und wird deshalb als eigenes Unterkapitel behandelt.

Da das System die Position von allen Clients speichern bzw. anzeigen kann, muss sichergestellt werden, dass diese Informationen ausschließlich den berechtigten Clients zur Verfügung gestellt werden. Außerdem soll die Abfrage der Position von anderen Clients nur in bestimmten zuvorvereinbarten Zeitintervallen gestattet werden.

3.5. Zusammenfassung

In diesem Kapitel wurden die Anforderungen an das System aufgelistet. Anhand dieser Anforderungen soll später das Endsystem entwickelt werden, das möglichst viele dieser Anforderungen berücksichtigen soll. Ganz wichtig sind die funktionalen Anforderungen zu beachten.
Das System wird von einem Administrator beim Aufbau eingerichtet. Zu seinen Aufgaben zählen, Token und Ereignis zu definieren und persönliche Dateien in einer Datenbank zu speichern. Das System muss ebenfalls in der Lage sein, die Ankunft

[24] Z.B.: Linux OS, Mac OS und so weiter.

bzw. die aktuelle Position einer Person zu erkennen und die Position einer Person mittels eines Tokens anzuzeigen.
Die nicht-funktionalen Anforderungen werden soweit wie möglich in der Entwicklung des Systems berücksichtigt.
Funktionalität kann anhand eines Unit-Tests erfolgen, um mögliche Fehler abzudecken. Dieser kann mit Hilfe eines Tools wie JUnit realisiert werden. Zuverlässigkeit wird anhand von Warnmeldungen angestrebt. Übertragbarkeit ist aufgrund der Vielzahl von Programmiersprachen, die für verschiedene Smartphone speziell entwickelt wurden, nur schwer zu erfüllen. Dies ist z.B.: Objective-C bei Iphone, C# bei Windows Mobile Smartphones, und vieles mehr.
Benutzbarkeit, Effizienz und Sicherheit werden nicht berücksichtigt und für die weitere Stufe der Entwicklung vorgesehen.

4. Design und Realisierung

In diesem Kapitel wird das technische Konzept genauer betrachtet und als eine Anleitung zur Entwicklung der Anwendung. Das Design der Anwendung wird auch in Bezug auf Kapitel 3 erklärt.
Die System- sowie die Komponentenarchitektur werden in Kapitel 4.1 näher betrachtet und später die technische Erklärung von Begriffen, auf die in Kapitel 3 noch nicht richtig eingegangen wurde, ausführlicher erklärt.

Letztendlich wird im Kapitel 4.3 das Designkonzept aus Kapitel 4.1 unter den fachlichen und technischen Anforderungen im Rahmen einer gegebenen Zielarchitektur realisiert.

4.1. Architektur

Andrew S. Tanenbaum hat in seinem Verteiltes System Buch [Tanenbaum, u.a. 2003] fünf Ansätze der Kompetenzverteilung für ein Client-Server-System beschreiben, die sind:

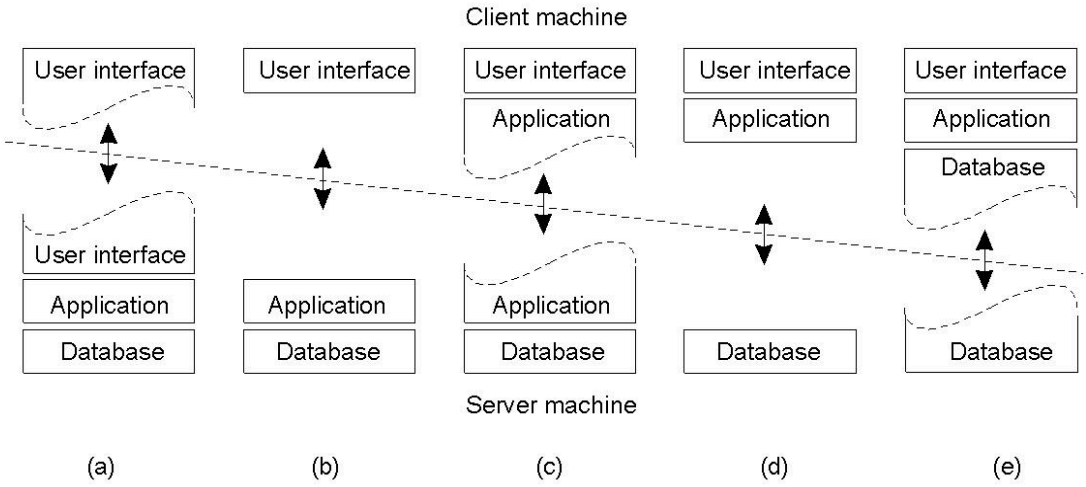

Abbildung 4.1 Client-Server-System

Die jeweiligen Ansätze lauten:

- *Thin Client (a)*
  Bei (a) wird der Client nur noch zur Anzeige von Informationen dienen. Alles andere (Anwendung und Datenspeicherung) ist auf dem Server vorhanden.

- *Thin Client (b)*
  Bei (b) liegt die ganze Benutzeroberfläche auf dem Client. Das bedeutet, dass der Client eine grafische Benutzeroberfläche zur Verfügung stellt. Die anderen Komponenten sind auf dem Server vorhanden.

- *Rich Client (c)*
  Bei (c) bekommen sowohl der Client als auch der Server die Anwendung. D.h. der Client hat die Möglichkeit bestimmte (eventuell einfachere) Aufgaben zu übernehmen und die rechenintensiveren Aufgaben dem Server zu überlassen.

- *Rich Client (d)*
  Der Client bekommt bei (d) die gesamte Software der Anwendung. Dies sind normalerweise ein PC[25], der mit einem angeschlossenen Datenbank-Server verbunden ist.

- *Fat Client (e)*
  Bei (e) bekommt der Client zusätzlich zu (d) auch die Datenbankteile. Das ist z.B. ein PC, mit dem der Client einen Cache über die besuchten Webseiten.

Die Entwicklung in dieser Arbeit benutzt den Rich Client (d) Ansatz, da der Server hierbei nur benutzt wird, um die Daten zu speichern. Die Positionsberechnung und alle andere Geschäftsanwendungsfälle finden auf dem Client statt.

4.1.1. Systemarchitektur

Die Systemarchitektur ist aus vielen vernetzten Komponenten aufgebaut. Die einzelnen Komponenten haben unterschiedliche Aufgaben zugeteilt.
Abbildung 4.2 beschreibt die Systemarchitektur, wie sie in diesem Kapitel genauer erklärt wird.

Das Smartphone dient den Benutzer in dieser Anwendung zur Positionsbestimmung, zur Speicherung von Verabredungsdaten und zur Kommunikation zwischen den Benutzern (Versenden und Empfangen von Token).
Die anderen Komponenten des verteilten Systems z.B. MagicMap-Server, Tracker, usw. sind für den Benutzer unsichtbar. Der Unterschied zwischen den MagicMap CE-Client und den MagicMap-Client wird im Kapitel 4.2 genauer ausgeführt.

[25] Abkürzung für Personal Computer; auf deutsch: der Rechner

Öffentliche Bildschirme werden hier für das Anzeigen der Informationen eingesetzt. Der Bildschirm (s. Abbildung 4.2) muss nicht unbedingt an dem Rechner, auf dem MagicMap-Client und Tracker betrieben wird, angeschlossen werden. D.h. es würde schon reichen, wenn der Bildschirm mit einem Rechner, auf dem ein MagicMap-Client betrieben wird, angeschlossen wird.

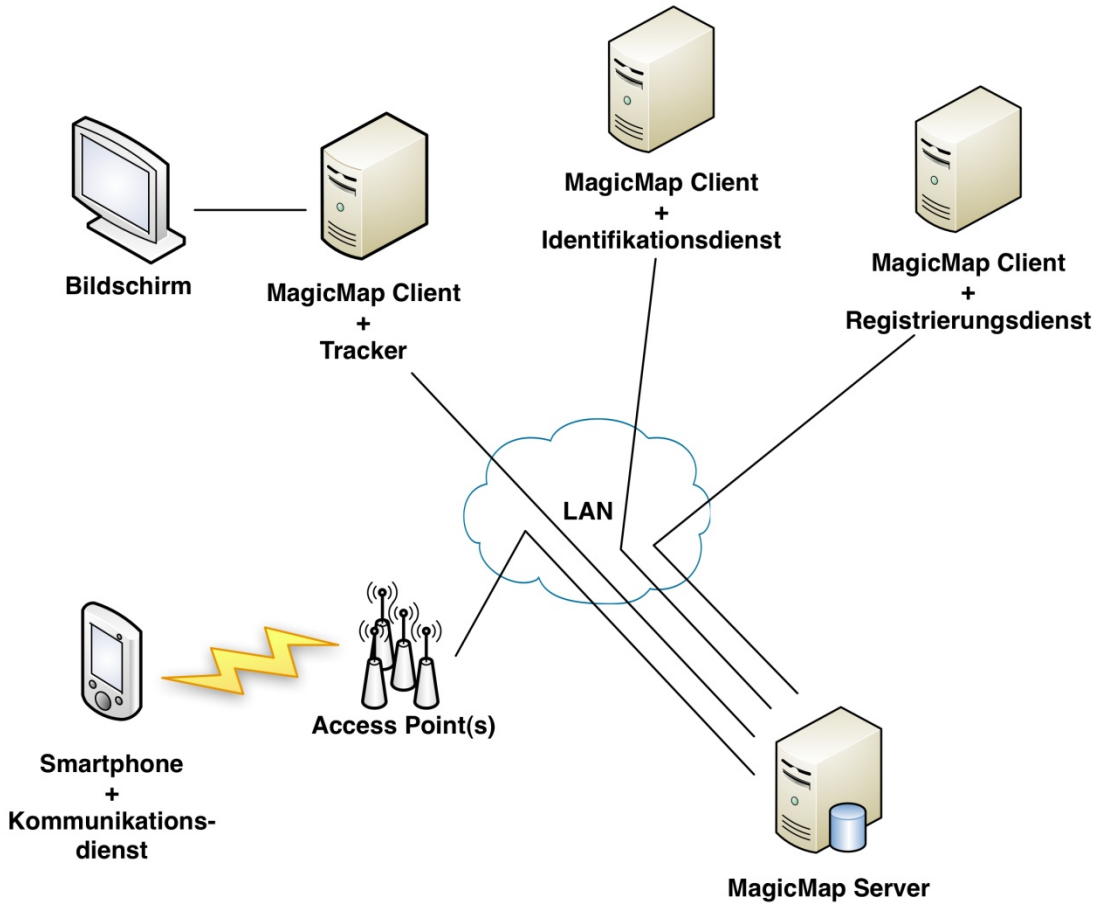

Abbildung 4.2 Systemarchitektur

Für die Kommunikation zwischen den verteilten Komponenten werden eine Funk-Datenübertragungsmethode, WLAN, und eine kabelgebundene Datenübertragungsmethode, LAN, benutzt. Wie in Kapitel 2.1.1.4. beschrieben, kann MagicMap, das als Basisanwendung für diese Arbeit dient, andere Methode für die Kommunikation[26] benutzen. Die werden aber hier ausgeschlossen und für die weitere Entwicklung nicht benutzt.
Aus der Anforderung im Kapitel 3.3 wird außerdem ein GSM Netz für die Kommunikation zwischen den Benutzern benötigt.

[26] Z.B.: Bluetooth oder Port-to-Port.

Die Dienste werden auf den Client installiert (sowohl beim Rechner als auch beim Smartphone). Die jeweiligen Clients kommunizieren untereinander durch den Server. Das bedeutet, dass keine direkte Kommunikation zwischen den Clients möglich ist.

4.1.2. Softwarearchitektur

Der Begriff „Softwarearchitektur“ beschreibt eine strukturierte oder hierarchische Anordnung der Systemkomponenten sowie ihre Beziehungen untereinander [Balzert, H 2001]. Das Gesamtsystem wird in Komponenten zerlegt. Somit können die Beziehungen zwischen den Komponenten auch dargestellt werden.
Diese Zerlegung verringert möglicherweise auftretende Probleme, d.h. deren Aufspüren wird vereinfacht.

Diese Softwarearchitektur wird ebenfalls durch nicht-funktionale Anforderungen (siehe Kapitel 3.4) bestimmt.
Spätere Änderungen bei der Softwarearchitektur sind mit hohem Aufwand verbunden. Somit gelten die in diesem Rahmen getätigten Entscheidungen als die kritischsten und wichtigsten Punkte im Entwicklungsprozess einer Software.

4.1.2.1 Drei-Schichten-Architektur-Modell

Die Softwarearchitektur des Systems wird als Drei-Schichten-Architektur gegliedert wie in Abbildung 4.3 dargestellt.

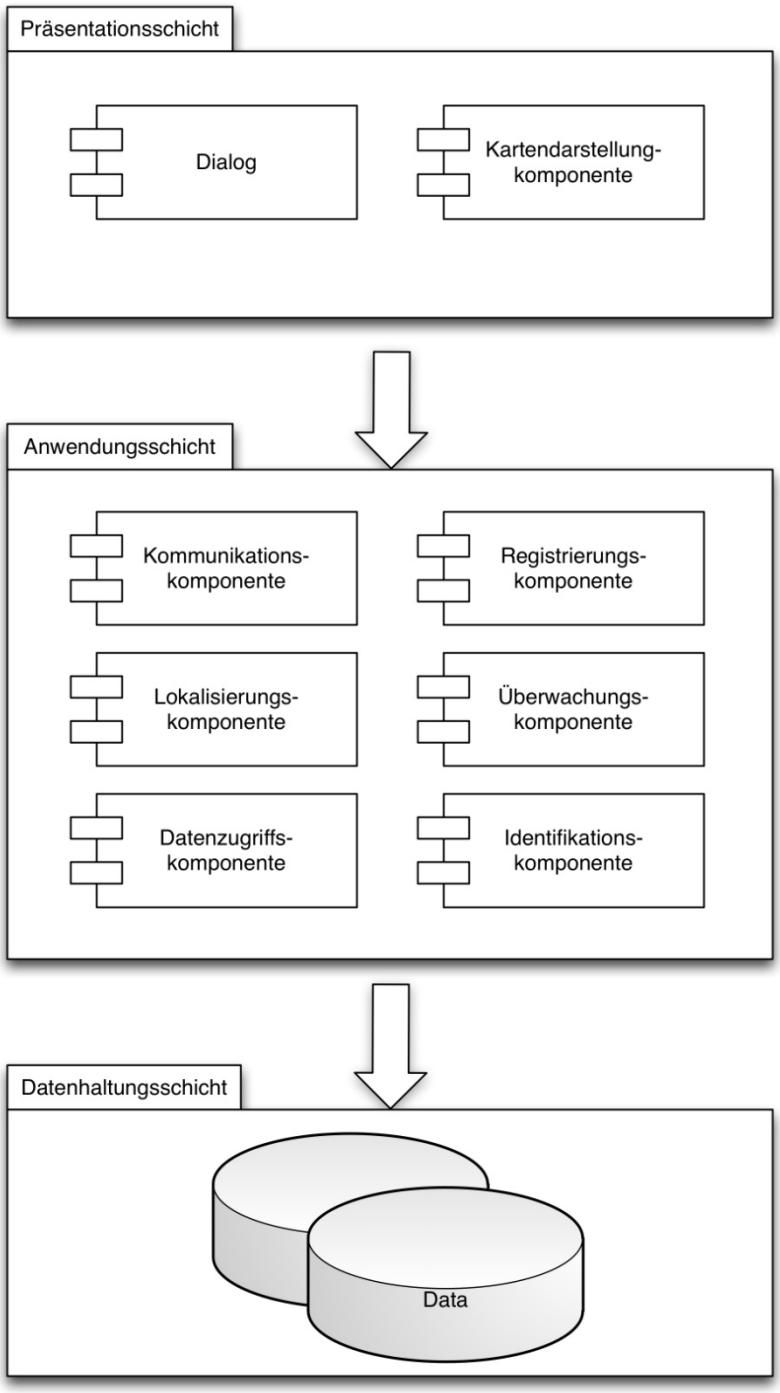

Abbildung 4.3 Drei-Schichten-Architektur

Die Drei-Schichten-Architektur besteht aus Präsentationsschicht, Anwendungsschicht und Datenhaltungsschicht. Die Komponenten werden auf diese Weise unterteilt, da angestrebt wird, dass nur die Komponenten der oberen Schicht auf ihre jeweils unterliegenden Schichten zugreifen dürfen. Ein Zugriff in umgekehrter Richtung ist nicht vorgesehen.

Die Präsentationsschicht besteht aus einer Dialog- und Kartendarstellungskomponente. Die Dialogkomponente stellt eine Schnittstelle zum Benutzer her (z.B. das Anzeigen der Anmeldungsschnittstelle) und nimmt die Abfrage des Benutzers an. Die Kartendarstellungskomponente ist für das Anzeigen von Kartenmaterial sowie zusätzlicher Informationen, z.B. Token, zuständig.
Die Entwicklung dieser Schicht spielt eine große Rolle, da sie eine direkte Schnittstelle zum Benutzer herstellt und die Anforderung aus Kapitel 3.4.3 berücksichtigen muss.

Die Anwendungsschicht besteht aus sechs wichtigen Komponenten.

- Die Kommunikationskomponente spielt bei MagicMap eine große Rolle, und zwar, für den Nachrichtaustausch zwischen den einzelnen Clients, zwischen Client und den verfügbaren Diensten[27], zwischen den einzelnen Diensten und zuletzt zwischen Client oder Diensten und den Servern. MagicMap stellt als Standard zwei verschiedene Verfahren für den Kommunikationswege zur Verfügung, die sind, Port-to-Port Kommunikation und Kommunikation über den Server. In dieser Arbeit wird die Kommunikation über den Server eingesetzt. Die Kommunikation zwischen den Clients läuft über das GSM-Netz (für den Versand des Tokens anhand MMS). Dieses (kostenpflichtige) Verfahren kann umgegangen werden, in dem das Token per Email gesendet wird. Diese Komponente wird auf MagicMap CE-Client entwickelt und deshalb wird nicht als Plugin realisiert, da MagicMap CE-Client nicht nach Plugin-Entwurfsmuster entworfen.

- Die Lokalisierungskomponente wird von MagicMap bereitgestellt. Diese Komponente hat die Aufgabe, die eigene Position zu berechnen. Da die Funktionalität des MagicMap bereits in Kapitel 2.1.1.4 erörtert wurde, deshalb soll dies hier nur kursorisch erwähnt werden.

- Die Datenzugriffkomponente koordiniert Datenzugriffe auf und vom MagicMap-Server. Der Server selbst ist eine reine Datenbank mit Webservices zum Annehmen von Anfragen des Benutzers. In dieser Datenbank werden die Positionsdaten aller MagicMap-Clients gespeichert. Die Komponente synchronisiert die Position des Clients mit dem Server, was nicht anders bedeutet als das Eintragen von Positionsdaten auf dem Server. Danach wird der MagicMap-Client die Position aller anderen MagicMap-Client herunterladen. Somit kennt jeder Client die Position des anderen.

[27] Z.B.: Tracker, Identifikationsdienst, Registrierungsdienst und viele weitere.

- Die Überwachungskomponente hat die Aufgabe Zustandsänderungen der Clients zu beobachten. Dies kann z.B. die Positionsänderung eines Clients sein oder der Wechsel eines Stockwerks. Diese Änderungen werden als Ereignis von der Überwachungskomponente wahrgenommen. Z.B.: Person A befindet sich zuerst auf Position (0,0) einer Karte. Dann bewegt sich diese Person auf eine andere Position (10,7), wobei eine Position zwischen (7,7) und (10,8) vorher als Eingang definiert wurde. Diese Positionsänderung wird dann als „Person A betritt das Gebäude“ von der Überwachungskomponente interpretiert. Eine derartige Komponente ist als Plugin im Kapitel 4.1.2.2.1 vom MagicMap verfügbar.

- Die Identifikationskomponente übersetzt den Namen des Clients zu einer MAC-Adresse und ist in der Lage auch in umgekehrter Richtung zu übersetzen. Jede Positionsdatei wird mit einem Namen des Clients versehen und der von ihm erwarteten Person versehen. Die Komponente hat Zugriff auf eine Datenbank, in der die Benutzerdaten vor dem Benutzung der Anwendung schon mal gespeichert wurde. Diese wird als Plugin im Kapitel 4.1.2.2.2 realisiert.

- Die Registrierungskomponente ist für die Instanziierung eines Ereignismusters zuständig. Instanziierung bedeutet Ausfüllen eines Ereignismusters, das von der Überwachungskomponente benutzt wird. Diese wird als Plugin im Kapitel 4.1.2.2.3 realisiert.

Die Datenhaltungsschicht besteht aus Datenbankinstanzen, auf der die Positionsdaten gespeichert werden. Die Instanzen werden nach Benutzungsart in zwei Kategorien unterteilt. Die erste ist die Datenbank im MagicMap-Server, die andere ist für die Speicherung der Personendaten des Clients zuständig. Die letztere Variante könnte auch anhand einer einfachen Liste realisiert werden. In dieser Arbeit wird allerdings mit einer echten Datenbank gearbeitet.

4.1.2.2. Plugin Entwurfsmuster

MagicMap-Client als Basis für diese Bachelorarbeit benutzt das Plugin-Entwurfsmuster [Plugin Pattern]. Dieses Entwurfsmuster erweitert die Funktionalität von existierende Klassen bzw. Methoden, für weitere gezielte Aufgaben.
Außerdem wird in diesem Entwurfsmuster angestrebt, eine einfache Aufteilung von Schnittstelle einer Software, die benutzt bzw. nicht benutzt werden dürfen, zu erzielen.

Das bedeutet, MagicMap-Client hat Schnittstellen zur Verfügung gestellt und diese wird von den neuen Plugins verwendet werden.

Dieses Entwurfsmuster besitzt einen Vorteil, dass die Basisanwendung unabhängig vom Plugin weiter entwickelt werden kann, solange die Entwicklung nicht die veröffentliche Schnittstelle betrifft.
Die Rolle der einzelnen Plugins wird in den kommenden drei Unterkapiteln erörtert.

Die Beispiele sind hier der Tracker, Identifikationsdienst, Registrierungsdienst und werden in nächstes Kapitel ausführlicher erklärt.
Hier muss erwähnt werden, dass der MagicMap CE-Client leider nicht nach Plugin Entwurfsmuster entworfen wurde. Dieses wurde in Kapitel 4.1.2.1 kurz erwähnt.

4.1.2.2.1. Tracker

Der Tracker im Kapitel 2.2.3 ist ein Teil von MagicMap. Der Tracker wurde als Plugin entwickelt, um Ereignisse zu überwachen. Ereignisse sind in dieser Arbeit als Zustandsänderung der Position eines Objektes definiert, z.B. wenn eine Person einen bestimmten Raum betritt. Der Tracker überwacht die Ereignisse bezüglich einer Karte, d.h. Ereignisse müssen auch im Bezug zu einer Karte definiert werden. Z.B.: Person A befindet sich vor dem Bildschirm des elften Stockwerkes. Dieses Ereignis wird in einem Ereignismuster beschrieben.

Außer das Ereignis hat das Ereignismuster eine Aktionsfolge. Diese wird ausgeführt, wenn ein bestimmtes Ereigniss eingetretten ist. Die Aktionsfolge kann z.B. die Visualisierung im MagicMap Benutzeroberfläche beeinflussen und somit kann die Anzeige eines Objektes besonders markiert werden.

Der Tracker besteht aus drei Teilen: einer Benutzeroberfläche zur Definition eines zu überprüfenden Ereignismusters, einer Zustandsüberwachung und einer Aktionsausführung.
Zustände werden über die Esper Event Pattern Language[28] definiert und mit JESPA verarbeitet.

```
SELECT 'B' as id, 'A ist im Eingangsbereich' as msg
FROM PhysicalObjectEvent
WHERE id ='A' AND x in [400:500] AND Y in [400:500]
```

Abbildung 4.4 Esper Event Pattern Language

Auf den ersten Blick erscheint Esper EPL als einer normalen SQL-Anweisung zum Verwechselnd ähnlich. Bei der Auswertung allerdings verhält sie sich völlig unterschiedlich.

[28] Wird nachfolgend als Esper EPL gekürzt.

Im „Select“ Bereich wird definiert, alles was in den Nachrichtaustausch geschrieben werden wird. Dieser Bereich ist die eigentliche Aktionsfolge des Ereignismusters.

Im „From“ Bereich wird der Name des Ereignisstroms angegeben, über den der Tracker mit Daten versorgt wird. Als Standard ist „PhysicalObjectEvent“ vorgegeben, es sei denn, dass mehr als ein Ereignisstrom benötigt wird. Z.B.: Außer den Positionsdaten werden auch Temperatursensordaten benötigt.

Der „Where“ Bereich definiert das eigentliche Ereignis. In Abbildung 4.4 ist ein Beispiel eines Ereignismusters beschrieben: „Person A betritt das Gebäude“. In diesem Fall stehen x und y für zwei diagonale Punkte eines Rechtecks[29] in einem (zwei-dimensionalen) kartetischen Koordinatensystem einer Karte.

4.1.2.2.2. Identifikationsdienst

Der Identifikationsdienst wird als Plugin von MagicMap realisiert. Dieses Plugin ist für die Übersetzung von Informationsdateien, mit der MagicMap versorgt wird, zuständig. Die Information beinhaltet nicht nur die Positionsdaten, sondern auch unter anderem die Namen der beteiligten Personen in diesem Szenario. Der Identifikationsdienst vergleicht diese Namen mit den in seiner Datenbank registrierten Personendaten (MAC-Adresse und Telefonnummer). Die MAC-Adresse wird vom Tracker benötigt, da der Tracker die Positionsänderungen auf einer bestimmten Karte anhand von MAC Adresse überwacht.

[29] Im Bezug des Beispiels sind die Punkte: x = (400,400) und y = (500,500).

4.1.2.2.3. Registrierungsdienst

Der Registrierungsdienst wird auch als Plugin von MagicMap realisiert und hat zur Aufgabe, das Ereignismuster zu initialisieren.

```
<eventsubscription version="0.1">
    <request
        desc="A soll im Eingangsbereich benachrichtigt werden, dass B im 11.Stock ist">
        <eventpattern>
            SELECT 'B ist im 11.Stock' as msg, 'A' as id
            FROM PhysicalObjectEvent
            WHERE id='A' AND x in [400:500] AND y in [400:500]
        </eventpattern>
    </request>
    <request
        desc="Sobald B im 3.Stock vorm Monitor steht, soll die Position von A durch C in den Fokus geholt werden">
        <eventpattern>
            SELECT 'true' as focus, 'B' as id
            FROM PhysicalObjectEvent
            WHERE id='A' AND x in [100:200] AND y in [100:200]
        </eventpattern>
    </request>
</eventsubscription>
```

Abbildung 4.5 Ereignismuster im XML Format

Abbildung 4.5 zeigt das Beispiel eines Ereignismusters, das vom Registrierungsdienst erzeugt wurde und vom Tracker gelesen werden kann. Das hier vorliegende Ereignismusterbeispiel wäre aus dem Use Case 7 (s. Tabelle 3.8) entstanden.

Der Eventsubscription-Tag beschreibt die verwendete Versionsnummer des Eregisnismusters. Innerhalb des Tags wird das eigentliche Ereignis beschrieben. In diesem Beispiel sind zwei Ereignisse definiert, wobei jedes Ereignis in einen Request-Tag geschrieben wird. Im Request-Tag wird danach mittels der Esper Event Pattern Language (Esper EPL) die Aktionsfolge definiert, welche in einem Eventpattern-Tag eingebettet wird.

Das Ereignismuster wird dynamisch erzeugt, d.h. der Eventpattern-Bereich kann modifiziert werden. Je nach der Position einer Person[30] könnte der Bereich anders aussehen. Z.b.: SELECT ´B ist im 7. Stock´ as msg, ´A´ as id oder SELECT ´C ist im 7.Stock´ as msg, ´B´ as id. Wobei im Falle des letzten Beispiel angenommen wird, dass C sich im siebten Stock befindet und auf B wartet.

[30] Im Abbildung 4.5 ist Person B auf dem 11.Stock als Beispiel genommen.

4.2. Grundlegende Konzepte

In diesem Kapitel werden die wichtigen Begriffe, die im nächsten Kapitel erwähnt werden, ausführlich erklärt.

Die MagicMap Software als Basis dieser Arbeit besteht aus drei Teilanwendungen, dem MagicMap CE-Client, dem MagicMap-Client und MagicMap-Server, wobei nur MagicMap CE-Client und MagicMap-Client weiterentwickelt werden. MagicMap CE-Client unterscheidet sich vom MagicMap-Client in einem wichtigen Punkt: der MagicMap CE-Client wird nicht nach dem Plugin-Entwurfsmuster entwickelt, weswegen unterschiedliche Vorangehensweisen bei der Entwicklung der Komponenten auf dem MagicMap CE-Client und dem MagicMap-Client benutzt werden müssen.

Wichtig bleibt zu erwähnen, dass der MagicMap CE-Client und MagicMap-Client in der Lage sind, die Positionsdaten aller auf dem Server angemeldeten Clients zu lesen und ihre eigenen Positionsdaten an ihn zu senden. Jeder Client kommuniziert nur mit dem MagicMap-Server. Diese Vorgehensweise trägt einen großen Vorteil in sich: Sämtliche Clients und Plugin sind vollständig entkoppelt.

In MagicMap-System wird die Kommunikation zwischen den Clients so realisiert, dass jeder Client seine eigene Nachricht an seine eigenen Positionsdaten anheftet und zum MagicMap-Server sendet. Ein Client, der sich für diese Nachricht interessiert, kann die Nachricht dann auswerten. Dadurch wird angestrebt, dem Client die Illusion zu geben, als ob er direkt mit den anderen Clients anspricht. Die Nachricht wird über eine Parameter, von Typ AttributeDTO, der Positionsdaten zum Server gesendet.

Genauere Erklärungen über diese zwei Arten von Clients sowie deren Umsetzung in dieser Arbeit werden in Kapitel 4.3 erklärt.

4.3. Realisierung

In diesem Kapitel wird über die Realisierung des Systems mehr detailliert betrachtet. In Abbildung 4.2 wird ein Überblick über die Systemkomponenten von MagicMap gegeben. Die Verteilung der Module aus Kapitel 4.1 auf diese Systemkomponenten wird im Folgenden vorgenommen.
Das Basissystem gliedert sich wie folgt:

- MagicMap CE-Client
  Diese Basissoftware wird auf dem Smartphone installiert. Darin wird der Kommunikationsdienst entwickelt. Das Token, das für die Anzeige der Benuzterposition zuständig ist, wird mit Hilfe des Kommunikationsdienstes zu anderen Beteiligten versandt.

- MagicMap-Client
  Diese Basissoftware wird auf einem Rechner mit Java Runtime als Laufzeitumgebung installiert. Dieser Client wird nach dem Plugin-Entwurfsmuster entwickelt. Der Identifikationsdienst sowie der Registrierungsdienst sind auf diesem Client als Plugin installiert.

- MagicMap-Server
  Diese Basissoftware wird auf einem Rechner installiert und von der HAW zur Verfügung gestellt. Auf dem MagicMap-Server wird nicht weiteres entwickelt, da der MagicMap-Server nur für die Speicherung der Positionsdaten in einer Datenbank zuständig ist.

Hier ist auch erwähnenswert, dass die jeweiligen Dienste[31] sowohl vereinzelt auf unterschiedlichen Rechnern als auch auf einem einzigen Rechner installiert werden können. In dieser Arbeit werden die drei Dienste auf unterschiedlichen Rechner installiert.

Da die Kommunikation zwischen den Clients zum großen Teil über den MagicMap-Server laufen (s. Kapitel 4.1.1), könnte ein großer Datenverkehr im Netzwerk entstehen, das ein so genanntes Flaschenhals-Problem hervorrufen könnte.
Dieses Problem wird umgegangen, indem die jeweilige Abfrage zum Server asynchron abläuft. Damit kann der jeweilige Client weiterarbeiten, selbst wenn ein Flaschenhals-Problem auftreten sollte.

4.3.1. Technische Voraussetzungen

Die Realisierung dieser Arbeit hat einige Voraussetzungen festgelegt, unter anderem Hardware- und Software-Voraussetzungen, die in Kapitel 4.3.1.1 und 4.3.1.2 aufgelistet werden.

4.3.1.1. Hardware

- zwei Smartphones HTC TyTN Pro:
  - WLAN 802.11b Empfänger.
  - Windows Mobile 6.0 Betriebssystem.
  - ausgestattet mit MagicMap CE-Client.

[31] Der Identifikationsdienst, der Registrierungsdienst und der Kommunikationsdienst.

- Bildschirm (32 Zoll):
  - mit einem PC verbunden.
  - mit einem Touchscreen ausgestattet.
  - angeschlossen auf dem PC mit MagicMap-Client.
  - verbunden mit dem HAW LAN.

- Rechner:
  - Dual Core Intel Prozessor mit Windows Server 2003 R2 Betriebssystem.
  - Datenbankinstanzen für ID Speicherung.
  - ausgestattet mit MagicMap-Client.
  - MagicMap-Client mit Tracker, Identifikationsdienst und Registrierungsdienst.

- Server Rechner:
  - ausgestattet mit MagicMap-Server.
  - Datenbankinstanzen für die Speicherung von Positionsinformation.

4.3.1.2. Software

Zur Realisierung der Anwendung auf dem Benutzergerät (MagicMap CE-Client) wird mit Hilfe des Microsoft Visual Studio 2008 [VS] IDE[32] verwendet, da die Anwendung auf C# basiert ist und Visual Studio momentan das beste IDE zur Entwickelung von C# basierten Anwendungen ist.

Die Anwendung auf dem Rechner (MagicMap-Client) wird mit IntelliJ IDE [IntelliJ] entwickelt, da die Anwendung auf Java basiert ist. Weiterhin vorgeschrieben ist Java 1.6 Runtime als Laufzeitumgebung.

Während der Entwicklung werden praktisch keine Veränderungen an der Benutzeroberfläche durchgeführt. Nur am Ende soll MagicMap-Client in der Lage sein, die Benutzerposition mittels ihrer unterschiedlichen Tokens anzuzeigen.

4.3.2. Realisierung der Anwendung auf dem Rechner

In diesem Unterkapitel wird sich auf die Realisierung der Rechneranwendung konzentriert. Dafür diskutiert die Arbeit den funktionalen Umfang (s. Kapitel 4.3.2.1), danach die Sequenzdiagramme (s. Kapitel 4.3.2.2) und letztendlich die Klassendiagramm (s. Kapitel 4.3.2.3).

[32] Integrated Development Environment (eine Anwendung bzw. ein Werkzeug zur Entwicklung von Software)

4.3.2.1. Funktionaler Umfang

Die gestellte Anforderungen (s. Kapitel 3) werden im funktionalen Umfang bei der Entwicklung der Plugins soweit wie möglich berücksichtigt bzw. implementiert.

In MagicMap-Client gibt es die Möglichkeit, die Anwendung mittels eines Plugins weiterzuentwickeln. Die verfügbaren Methoden sind z.B. startPlugin(), stopPlugin(), connect(), loadMap(). Die connect()-Methode ist die einzige Methode, die für die Entwicklung der Funktionalität in dieser Arbeit benutzt wird. Diese Methode lauscht auf Veränderungen der Positionsdaten aller am MagicMap-Server angemeldeter Benutzer.

Die Plugins kommunizieren mit Hilfe einer PositionObject-Klasse. Diese Klasse besitzt eine freie Variable, die AttributeDTO (s. Kapitel 4.2) heißt. Deshalb müssen sich die anwendungsspezifischen Daten, die zwischen dem Plugin versendet werden, vom Typ AttributeDTO ableiten. Genaueres dazu in Kapitel 4.3.2.3.

MagicMap-Client als Basis dieser Arbeit liefert folgende Funktionalitäten:

- Die der Positionsdaten zu sammeln, die Positionsdaten nach MagicMap-Server zu übertragen und die Positionsdaten der anderen Benutzern zu holen und speichern.
- Der Tracker-Plugin wird getrennt geliefert und dient in dieser Arbeit als Überwachungsdienst (s. Abbildung 4.3).
- Anzeigen von der Positionen von am MagicMap-Server angemeldeten Benutzern.

Damit die Anwendung die gestellten Anforderungen erfüllen kann, müssen mindestens folgende Funktionalitäten eingebaut werden:

- Erkennung der MagicMap-Benutzer.
- Automatische Registrierung eines Verabredungstermins beim MagicMap-Tracker.
- Benachrichtigung aller Beteiligten eines Verabredungstermins, wenn mindestens zwei Beteiligte anwesend sind[33].

Die Funktionalitäten werden anhand von Sequenz- und Klassendiagrammen im nächsten Kapitel erläutert.

[33] Wenn mehr als zwei Beteiligte vorhanden sind, werden jedes Mal die wartende Person und die neu angekommene Person eine Nachricht bekommen.

4.3.2.2. Sequenzdiagramm

Das Sequenzdiagramm des Trackers wird in diesem Kapitel nicht genauer erklärt, da keine Änderungen bzw. Verbesserungen beim Tracker implementiert wurden.
In diesem Kapitel werden die zwei in dieser Arbeit entwickelten Plugins anhand eines Sequenzdiagramms erklärt.

4.3.2.2.1. Identifizierungsdienst

Das Sequenzdiagramm in Abbildung 4.6 beschreibt den Ablauf eines Plugins am Beispiel des Identifizierungsdiensts. Hier werden der Einfachheit halber nur noch die wichtigsten Komponenten, der Methodenaufruf und die Lebenszyklen aufgelistet.

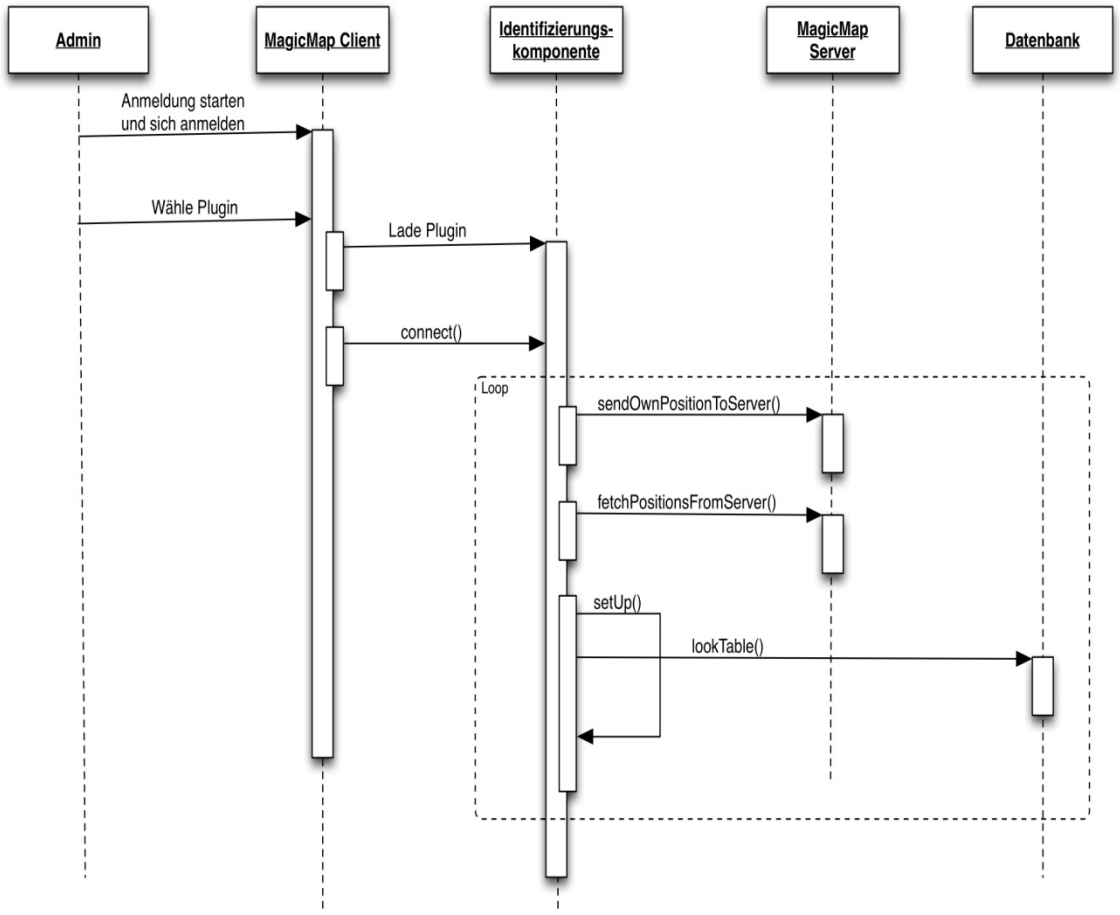

Abbildung 4.6 Sequenzdiagramm des Identifikationsdiensts

Der Identifizierungsdienst besitzt eine wichtige Aufgabe. Es muss in der Lage sein, angemeldete Benutzern zu erkennen. Der Benutzer, anders als der Tracker, erkennt die anderen Benutzer anhand ihrer Namen. Der Tracker erkennt den Benutzern dagegen anhand von ihrer MAC-Adresse.

Der MagicMap-Client wird auf einem Rechner installiert, der bspw. in einem Rechenzentrum betrieben werden kann.
Wird der MagicMap-Client von einem Administrator gestartet und ist erfolgreich mit dem MagicMap-Server verbunden, können nun die Plugin geladen werden. In Abbildung 4.6 wird ein Identifizierungsdienst auf dem MagicMap-Client geladen.
Nachdem der Identifizierungsdienst gestartet ist, ruft der MagicMap-Client die connect()-Methode auf. Diese Methode ist eine wichtige Schnittstelle für diese Arbeit, da der Ablauf des Identifizierungsdiensts von ihr angetriggert wird.
Der Identifizierungsdienst bereitet die Positionsdaten vor, die wichtige Daten enthält, namentlich die Verabredungsdaten. Die Verabredungsdaten beinhalten die Liste der Benutzern, die an einem Verabredungsszenario teilnehmen.

Am Anfang sind die Verabredungsdaten leer. Diese werden zusammen mit Positionsdaten zum Server gesendet. Das passiert innerhalb des sendOwnPositionToServer()-Methodensaufrufs. Danach wird die fetchPositionsFromServer()-Methode aufgerufen. Dieser Aufruf bewirkt, dass die Position aller anderen Benutzer vom Server geholt und anschließend in einem Cache gespeichert werden.

Der Identifizierungsdienst ruft dann umgehend die setUp()-Methode auf. Innerhalb dieses Aufrufs werden alle vom Server abgeholten Positionsdaten einzeln überprüft. Die Positionsdaten beinhalten unter anderem ihre eigenen Namen, die Namen der erwarteten Person(en). Wobei der Identifizierungsdienst sich nur für die Namen der Beteiligten interessiert.

Innerhalb der setUp()-Methode wird die lookTable()-Methode aufgerufen. Diese Methode vergleicht die Namen mit der Datenbank. Wurde ein passender Eintrag gefunden, werden diese Daten[34] geholt und in Verabredungsdaten gespeichert.
Diese Daten werden von einem Registrierungsdienst benötigt.

Die connect()-Methode wird immer wieder aufgerufen, deshalb werden die Positionsdaten beim nächsten Aufruf zum Server gesendet.

[34] Telefonnummer, Email Adresse und so weiter.

4.3.2.2.2. Registrierungsdienst

Das Sequenzdiagramm in Abbildung 4.7 beschreibt den Ablauf eines Identifizierungsdiensts.

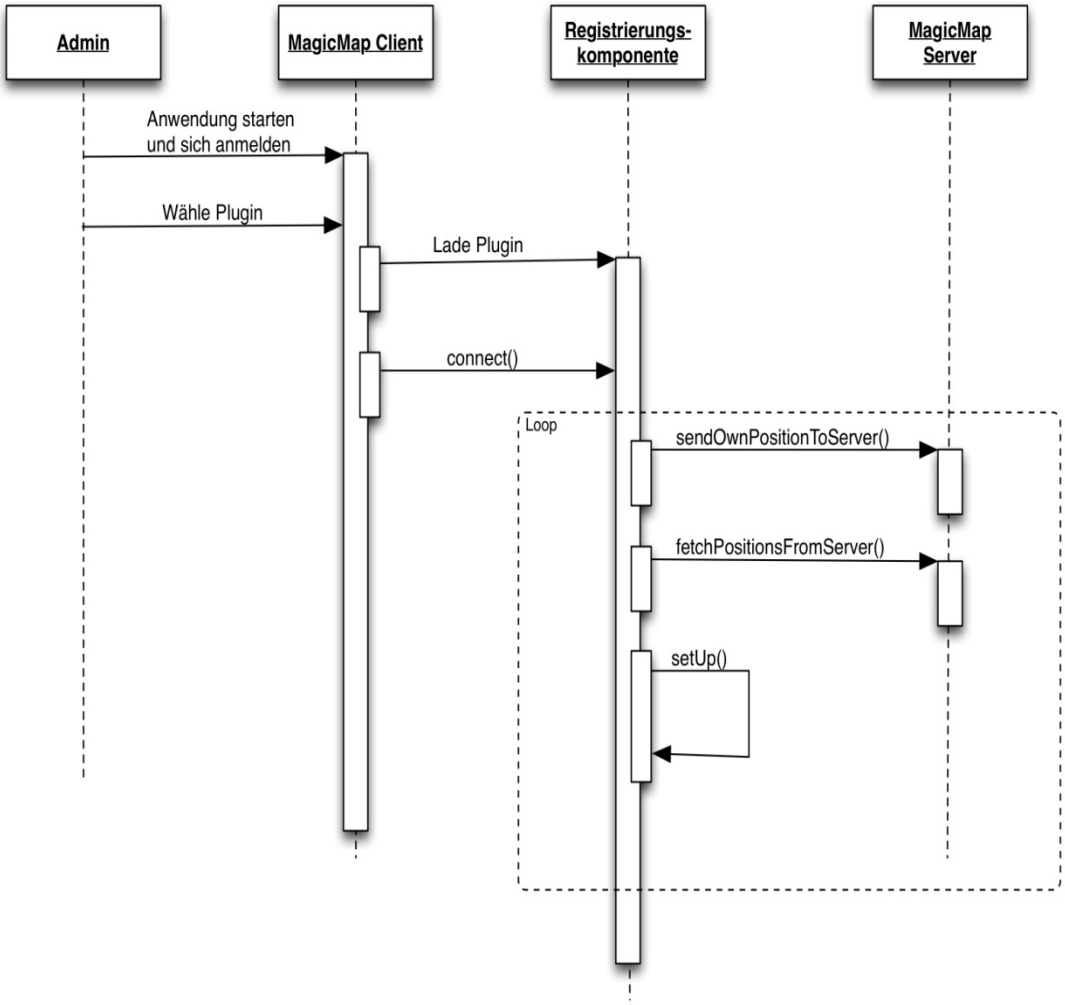

Abbildung 4.7 Sequenzdiagramm des Registrierungsdiensts

Der Registrierungsdienst besitzt ein ähnliches Diagramm wie der Identifikationsdienst. Auch der Ablauf ist vom Aussehen her fast identisch.
Der Registrierungsdienst kann auch im selben MagicMap-Client, der den Identifikationsdienst als Plugin betreibt, gestartet werden.

Zuerst startet der Administrator die Anwendung und meldet sich an. Nach erfolgreicher Anmeldung wählt der Administrator den Registrierungsdienst als Plugin.
Nachdem der Registrierungsdienst gestartet wurde, ruft MagicMap-Client die connect()-Methode auf, womit der Registrierungsdienst angetriggert wird.

Innerhalb dieser Methode werden unter andere die Ereignismuster (s. Abbildung 4.5) initialisiert und für die Registrierung beim Tracker vorbereitet.
Der Registrierungsdienst bereitet alle benötigen Positionsinformationen vor und speichert diese in Positionsdaten ab. In der sendOwnPositionToServer()-Methode sendet der Registrierungsdienst die Positionsdaten zum Server.
Die fetchPositionsFromServer()-Methode wird im Nachhinein aufgerufen, um alle Positionsdaten, die auf dem Server gespeichert wurden, zu holen. Diese werden in einem Cache gespeichert.

Letztendlich wird die setUp()-Methode aufgerufen. Dieser Methodenaufruf ist der wichtigste Teil des gesamten Plugin-Lebenszyklus. In dieser Methode wird nur nach den Positionsdaten des Identifikationsdiensts gesucht. Alle anderen Positionsdaten sind für den Registrierungsdienst unwichtig.
Der Registrierungsdienst filtert die Verabredungsdaten, die in Positionsdaten zu finden sind.

Falls mehr als zwei Beteiligte (N-Personen) in einem Verabredungsszenario vorhanden sind, werden N – 1 Ereignismuster initialisiert und für die Registrierung beim Tracker vorbereitet.
Die Positionsdaten werden im nächsten sendOwnPositionToServer()-Methodenaufruf dann zum Server gesendet.

Wenn das erste Ereignis vom Tracker erkennt wurde (s. Tabelle 3.6), wird ein anderes Ereignismuster, dessen Anzahl N – 1 ist, erzeugt. In diesem Vorgang braucht der Registrierungsdienst zusätzlich die aktuelle Position der wartenden Person.

4.3.2.3. Klassendiagramm

In diesem Kapitel werden nur die wichtigsten Klassen, die Beziehungen zwischen den Klassen und den Methoden aufgelistet[35].
Das Klassendiagramm wird in jeweils ein Klassendiagramm des Identifizierungsplugins und ein Klassendiagramm des Registrierungsplugins unterteilt.

Obwohl beide Dienste für verschiedene Aufgabe verantwortlich sind, benutzen diese Klassen dennoch ein paar gleiche Klassen. Das bedeutet, dass die beiden Plugins z.B. die PositionDTO- oder die SOAPServerManager-Klasse benutzen.

[35] Eine vollständige Auflistung würde den Rahmen dieser Arbeit sprengen.

Eine kurze Auflistung der benutzten Klassen, die nach dem Plugin-Entwurfsmuster für die in dieser Arbeit entwickelten Plugins zur Verfügung stehen:

- AbstractPlugin: eine abstrakte Klasse. Alle entwickelten Plugins müssen sich von dieser Klasse ableiten, sodass die Plugins von MagicMap benutzt werden können.

- AttributesDTO: eine Klasse, die zurzeit dem Entwickler zur Verfügung gestellt wurde. Diese Klasse darf nach Bedarf für die Entwicklung dieser Arbeit verwendet werden, und zwar, als Platzhalter für die Kommunikation zwischen den Clients (s. Kapitel 4.2).

- PositionDTO: eine Klasse, die in dieser Arbeit als Positionsinformation bezeichnet wird. PositionDTO wird zusammen mit AttributesDTO als Positionsdaten in dieser Arbeit bezeichnet.

- SOAPServerManager: diese Klasse beinhaltet alle Webservice Interfaces, die vom MagicMap-Server zur Vefügung gestellt werden. Diese Klasse besitzt eine wichtige Methode, die das Versenden eigener Positionsdaten zum Server bewirkt. Diese Methode ist createOrUpdatePosition().

- Controller: Diese Klasse wird von den entwickelten Plugins nicht direkt benutzt. Diese Klasse wird für die Instanziierung der PollHandler- und von der SOAPServerManager-Klasse als Parameter benötigt.

- PollHandler: Diese Klasse wird wie die Controller Klasse von den entwickelten Plugins nicht direkt benutzt. Diese Klasse wird für die Instanziierung der ServerPoller Klasse als Parameter benötigt.

- ServerPoller: Diese Klasse ist für den Abruf der vom Server geholten Positionsdaten zuständig.

Anhand von Klassendiagrammen (s. Kapitel 4.3.2.3.1 und Kapitel 4.3.2.3.2) wird das Konzept des Plugin-Frameworks (s. Kapitel 4.1.2.2) und deren Beziehungen mit den entwickelten Plugins deutlich gemacht.

Das Klassendiagramm des Trackers wird hier nicht weiter erklärt (s. Kapitel 4.3.2.2).

4.3.2.3.1. Identifizierungsdienst

Abbildung 4.8 zeigt das Klassendiagramm des Identifizierungsplugins. Hier werden nur die wichtigste Klassen und Methoden aufgelistet.
Die grau gefärbten Klassenobjekte sind aus dieser Arbeit entstanden.

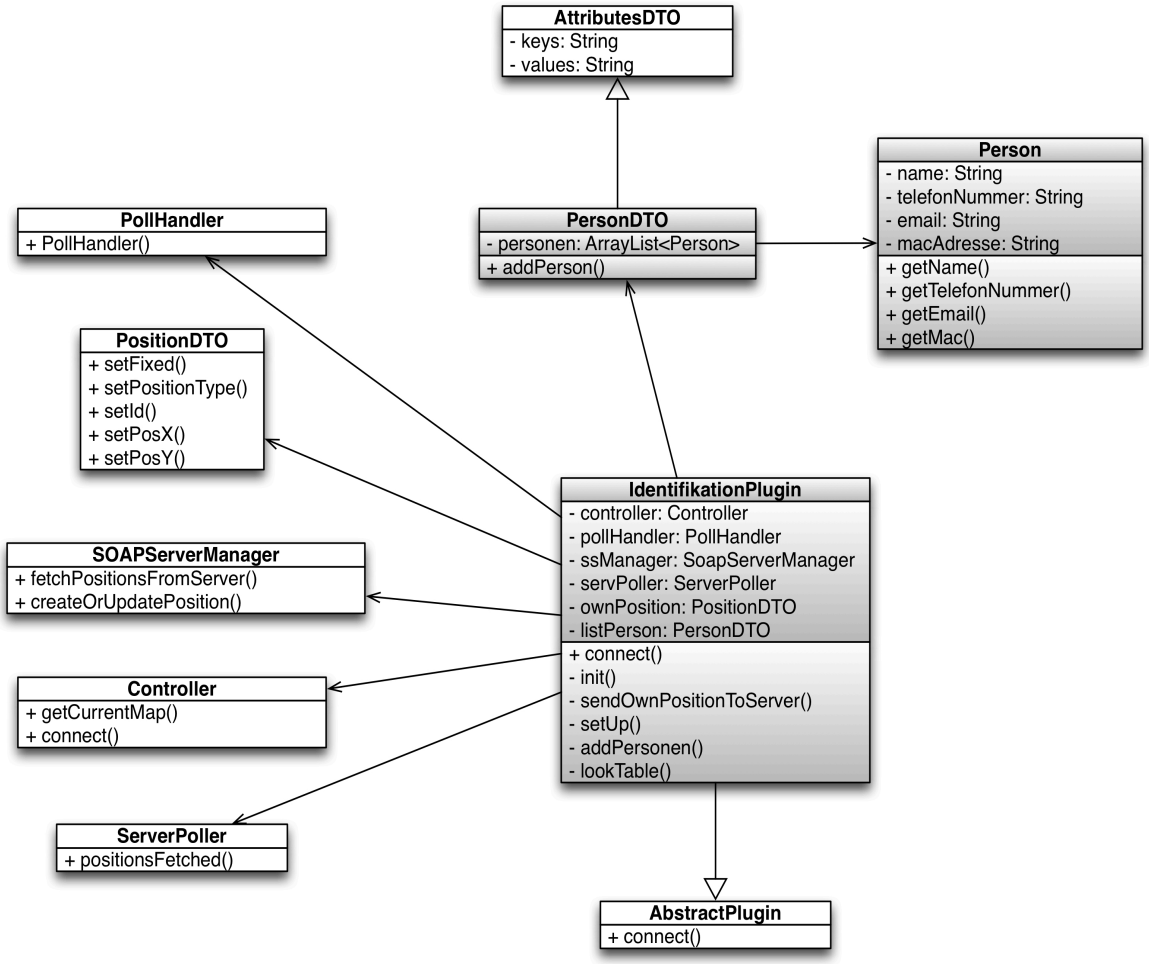

Abbildung 4.8 Klassendiagramm des Identifizierungsplugins

[Person]
Die Person-Klasse ist eine Klasse, in der die Informationen einer Person gespeichert werden. Diese Klasse beinhaltet nur die benötige Getter- und Settermethoden.

[PersonDTO]
Die PersonDTO-Klasse ist eine Unterklasse der AttributesDTO-Klasse und wird mittels einer PositionObject-Klasse[36] zum Server gesendet. In dieser Klasse sind die Instanzen der Person-Klassen gespeichert.
Die PersonDTO-Klasse muss eine Unterklasse der AttributesDTO Klasse sein, weil alle zusätzlichen Informationen, die zum Server mit Hilfe einer PositionObject-Klasse gesendet werden, in einer unbelegten Variable, deren Typ AttributesDTO ist, gespeichert werden (s. Kapitel 4.2).

[36] Diese Klasse ist eine Realisierung von Positionsdaten.

[IdentifikationPlugin]
IdentifikationPlugin ist eine Unterklasse der AbstractPlugin-Klasse und implementiert die wichtigste Methode der AbstractPlugin-Klasse, die connect()-Methode. Diese Methode wird vom MagicMap-Client[37] in einem bestimmten Intervall aufgerufen.

Die connect()-Methode ruft sendOwnPositionToServer()- und setUp()-Methode. Die sendOwnPositionToServer()-Methode bewirkt, dass die eigenen Positionsdaten zuerst zum Server gesendet werden. Das wird realisiert, in dem die createOrUpdatePosition()-Methode von SOAPServerManager-Klasse aufgerufen wird.
Die setUp()-Methode sammelt zuerst alle Positionsdaten vom Server. Diese Positionsdaten beinhalten unter anderem den Namen einer Person und den Namen seines eventuellen Verabredungspartners. Diese Informationen werden in der addPersonen()-Methode als Liste von Personennamen gespeichert und für das RegistrierungPlugin[38] vorbereitet.

In der addPersonen()-Methode wird die lookTable()-Methode aufgerufen. Dieser Methodenaufruf ist dafür zuständig, dass die Namen mit denen in der Datenbank des Identifikationsdiensts verglichen werden. Wird ein passender Eintrag gefunden, werden die MAC-Adresse (und eventuell andere Informationen) geholt und in einer Person-Klasse gespeichert.

[37] Wird hier nicht abgebildet.
[38] Und auch für jedes weitere Plugin, das diese Informationen braucht.

4.3.2.3.2. Registrierungsdienst

Abbildung 4.9 zeigt das Plugin des Registrierungsdienstes. Hier werden, wie schon im letzten Kapitel, nur die wichtigsten Klassen und Methoden aufgelistet
Die grau gefärbten Klassenobjekte sind aus dieser Arbeit entstanden.

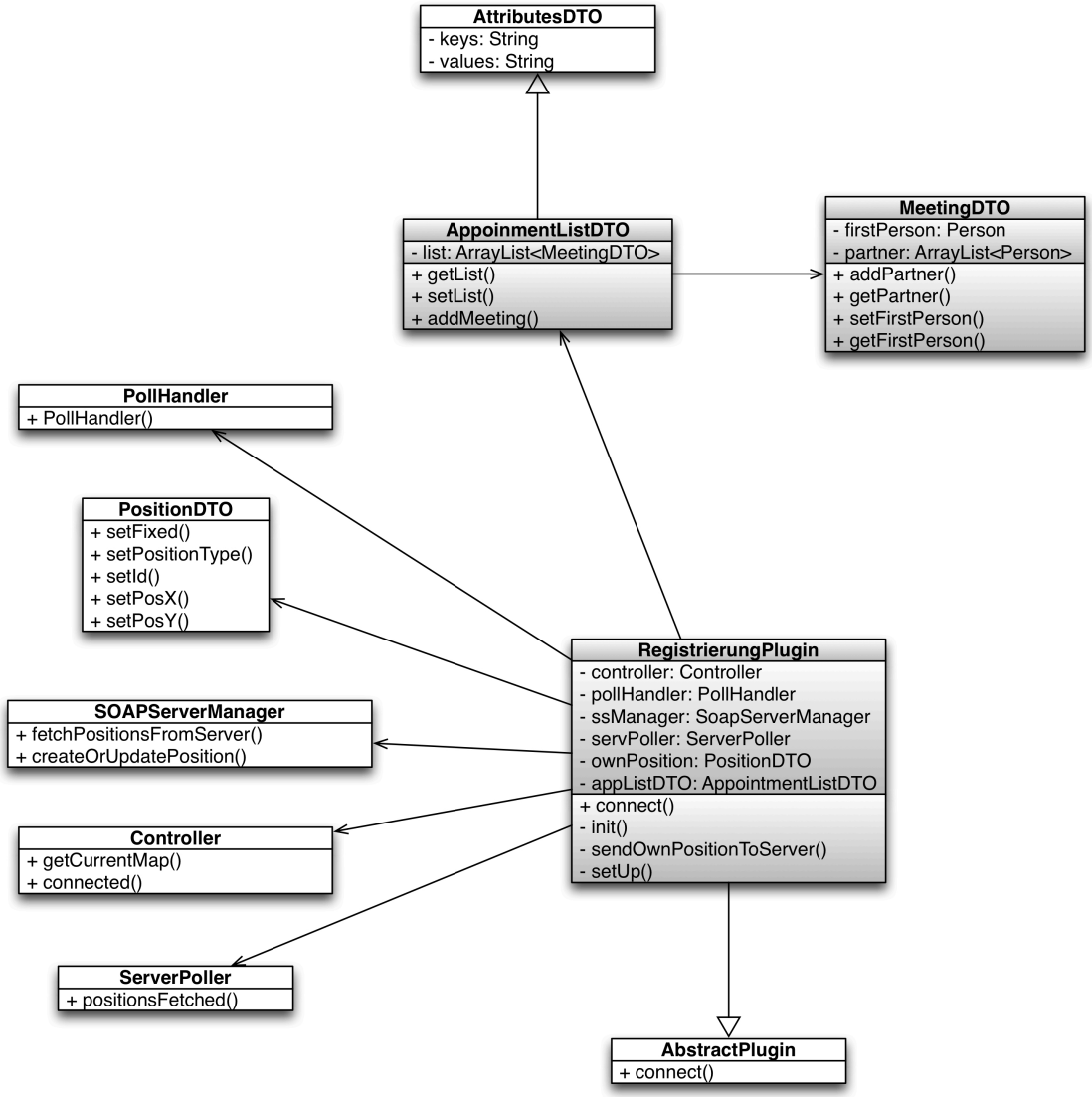

Abbildung 4.9 Klassendiagramm des Registrierungsplugins

[AppoinmentListDTO und MeetingDTO]
Die MeetingDTO-Klasse bezeichnet eine Verabredung zwischen zwei oder mehreren Personen. Die AppoinmentListDTO-Klasse ist in dieser Arbeit entwickelt worden, da eine Person in einer bestimmten Zeit[39] mehrere Verabredungstermine haben kann. Die

[39] Z.B.: an einem Tag hat eine Person B gleich zwei verschiedene Termine.

AppointmentListDTO-Klasse ist eine Unterklasse des AttributesDTO- und wird mittels einer PositionObject-Klasse zum Server gesendet.

[RegistrierungPlugin]
Das RegistrierungPlugin ist eine Unterklasse der AbstractPlugin-Klasse und unterliegt deshalb den gleichen Bedingungen und Regeln, wie jede Unterklassen der AbstractPlugin-Klasse (s. Kapitel 4.3.2.3.1).

Die connect()-Methode funktioniert wie die IdentifikationPlugin Methode (s. Kapitel 4.3.2.3.1).
Die setUp()-Methode sammelt zuerst alle Positionsdaten von Server. Danach werden die Positionsdaten des Identifikationsdiensts herausgesucht, da die Namen von Personen, die für die Initialisierung eines Ereignismusters benötigt werden, dort gespeichert sind.
Wenn die Namen vorhanden sind, wird unverzüglich das Ereignismuster initialisiert (s. Kapitel 4.1.2.2.3) und dieses zum Server gesendet.

4.3.3. Realisierung der Anwendung auf dem Benutzergerät

Ähnlich wie in Kapitel 4.3.2. wird hier auch die Realisierung der Benutzeranwendung zusammengefasst. Erst einmal wird über den funktionalen Umfang (s. Kapitel 4.3.3.1), danach über das Sequenzdiagramm (s. Kapitel 4.3.3.2) und letztendlich über das Klassendiagram (s. Kapitel 4.3.3.3) diskutiert.

In diesem Fall dient der MagicMap CE-Client als Basis für die Entwicklung. Dieser besitzt aber den Nachteil, und zwar, dass sie nicht nach dem Plugin-Entwurfsmuster entwickelt wurde. Daraus folgt, dass die Entwicklung auf dem Source-Code der Anwendung direkt passieren muss.

4.3.3.1. Funktionaler Umfang

In Kapitel 4.3.2.1 werden die gestellten Anforderungen im funktionalen Umfang auf dem Benutzergerät soweit wie möglich implementiert.

Der Benutzer wird von der Anwendung während des Ablaufes unterstützt. Dabei ist es wichtig, dass der Benutzer den Termin in seine Terminkalendar-Anwendung einträgt. In diesem Fall muss der Benutzer nur noch die Anwendung starten und sich einloggen.

Nach dieser Anmeldung wird der Benutzer nicht mehr mit irgendeiner Interaktion mit dem Smartphone belästigt, außer die ihm gesandte Nachrichten zu lesen. Der Benutzer bekommt die Nachrichten auf sein Smartphone zugesendet. Je nach Aufenthalt des Benutzers kann der ortsgebundene Dienst verschiedene Nachrichten

erzeugen, z.B.: Person A[40] bekommt unterschiedliche Nachrichten, wenn sie sich vor dem Eingang oder auf irgendeinem beliebigen Stockwerk befindet.

MagicMap CE-Client als Basis der Anwendung der Arbeit liefert folgende Funktionalitäten:

- ein Dialogfenster, das zur Anmeldung des Benutzers am MagicMap-Server dient.
- Positionsdaten zu sammeln, nach MagicMap-Server zu übertragen und die Positionsdaten der anderen Benutzern zu holen und speichern.

Damit die Anwendung die gestellten Anforderungen erfüllen kann, müssen mindestens folgende Funktionalitäten eingebaut werden:

- den Termin automatisch zu prüfen und die Informationen der Beteiligten zu sammeln.
- den Termin automatisch für den Tracker bereitzustellen.
- das Token zu den Anderen zu übertragen.

Um die Funktionalitäten zu realisieren, müssen einige neue Methoden und Komponenten[41] entwickelt werden.

[40] Das Beispiel ist auf Kapitel 3 bezogen (Person A ist die erwartete Person)
[41] In diesem Kapitel ist nur eine neue Komponente (Kommunikationskomponente) entwickelt worden.

4.3.3.2. Sequenzdiagramm

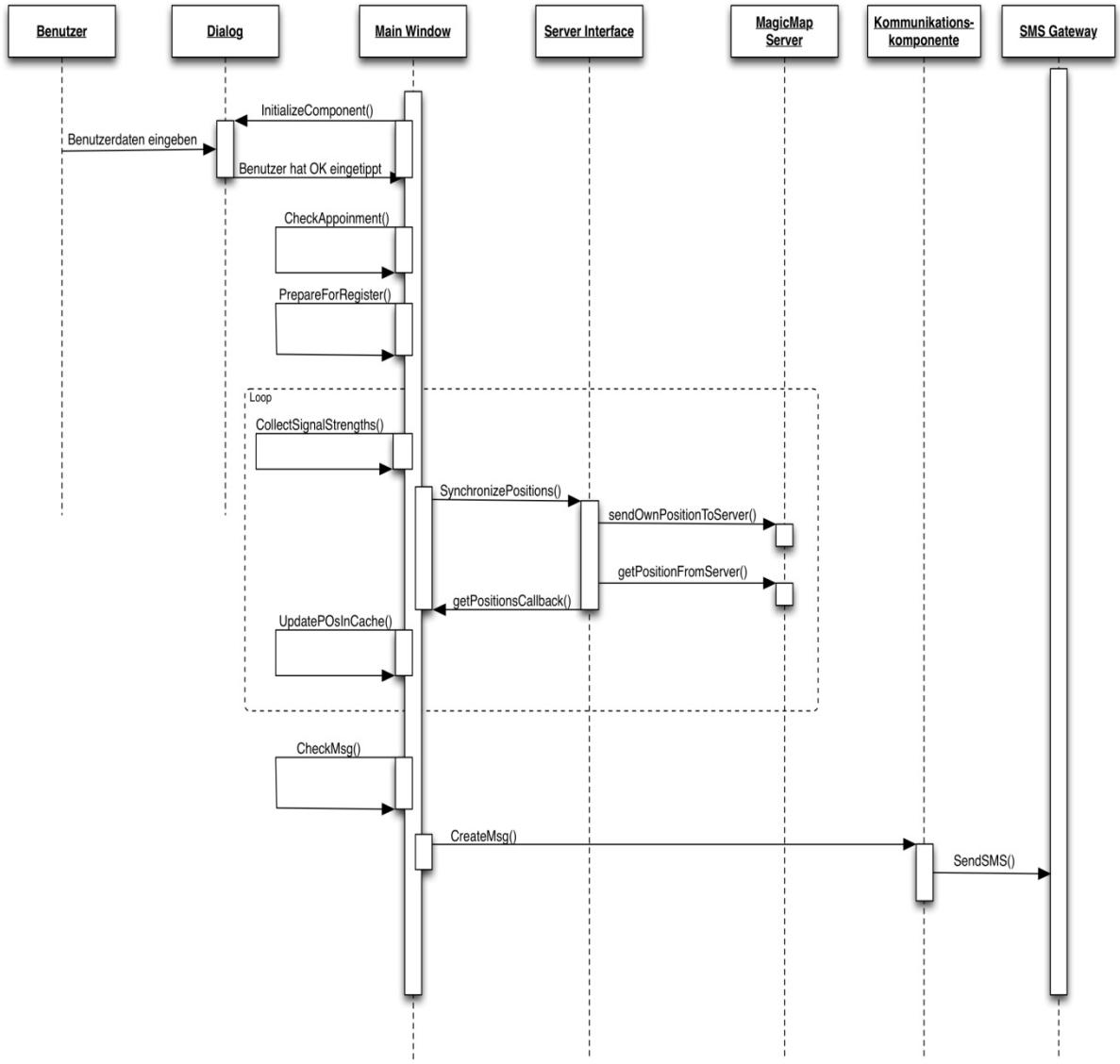

Abbildung 4.10 Sequenzdiagramm von der Anwendung auf Benutzergerät

Das Sequenzdiagramm in Abbildung 4.10 zeigt den Ablauf, der auf dem CE-Client passiert ist.

SMS Gateway wird gerade jetzt erwähnt, da es ein externer Dienst ist. Dieser Dienst wird von einem beliebigen SMS Gateway Provider gewählt, z.B.: mit Hilfe von NowSMS [NowSMS] oder LogixMobile [LogixMobile].

Sobald der Benutzer die MagicMap CE-Client Anwendung gestartet hat, wird das in der Abbildung 4.6 abgebildete Szenario gestartet. Zuerst ruft das MainWindow die InitializeComponent()-Methode auf. Der Aufruf bewirkt unter anderen, dass das Dialogfenster dem Benutzer angezeigt wird.
Der Benutzer gibt nun seine Login-Daten ein. Diese werden zum MainWindow weitergeleitet. Danach ruft das MainWindow die CheckAppointment()-Methode auf, um den Termin auf dem Smartphone zu suchen. Innerhalb dieser Methode werden auch weitere benötige Informationen[42] gesammelt. Danach wird die PrepareForRegister()-Methode aufgerufen. Innerhalb dieser Methode werden die Informationen in einem Appointment-Objekt instanziiert.

Das Appointment-Objekt wird nachher in der CollectSignalStrengths()-Methode zusammen mit sämtlichen anderen Positionsdaten in einem PositionObject-Objekt instanziiert.
Die SynchronizePositions()-Methode ist dafür zuständig, das PositionObject-Objekt zum Server zu übertragen. Innerhalb der Methode wird unter anderen die Sync()-Methode vom Server Interface aufgerufen. Dieser Aufruf ergibt einen fortlaufenden Methodenaufruf, sendOwnPositionToServer(). Diese Methode ist eine Webservice Methode, die eine Speicherung des PositionObject-Objekts in der MagicMap-Server Datenbank bewirkt.
Die getPositionsFromServer()-Methode wird ebenfalls auch in dieser Sync()-Methode aufgerufen. Sie ist auch dafür zuständig, dass die Position der anderen Benutzern vom MagicMap-Server geholt wird.
Nachdem die Positionen von sämtlichen Benutzern gesammelt wurden, wird letztendlich die getPositionsCallback()-Methode aufgerufen. Dieser Aufruf bewirkt, dass das MainWindow über die eventuell aufgetauchten Veränderungen von Positionsdaten informiert wird.

Danach werden diese Positionsdaten in einem Cache[43] gespeichert. Es handelt sich um einen flüchtigen Speicher, d.h. beim nächsten Start der Anwendung sind die Daten nicht mehr vorhanden. Der Vorgang passiert in der UpdatePOsInCache()-Methode.
Die CollectSignalStrengh()-, SynchronizePositions()- und UpdatePOsInCache()-Methoden sind Threads, deshalb wird sie, anders als alle anderen Methoden, immer wieder aufgerufen werden.

Bis hierhin läuft der Vorgang sowohl für die wartenden als auch für die erwartete Person identisch ab. Die wartende Person muss zusätzlich noch in der Lage sein, das Token, welches zu Erkennung der Position des Senders dient, zu versenden. Das

---

[42] Es wird immer angenommen, dass der Benutzer den Termin in sein Smartphone einträgt. Der Inhalt sind der Name des Benutzers und der der erwarteten Person(en).
[43] Dies ist eine Methode, um Inhalte, die bereits einmal vorlagen, beim nächsten Zugriff schneller zur Verfügung zu stellen. Die Inhalte können auf einem dauerhaften oder flüchtigen Speicherplatz gespeichert werden [Cache].

bedeutet, die Ausführung der CheckMsg()- und CreateMsg()-Methoden. Die erwartete Person muss anschließend nur noch auf diese Nachricht warten.

CheckMsg() prüft in bestimmten Zeitabständen, ob eine Nachricht vom Tracker angekommen ist. Wenn das MainWindow in Laufe der Zeit die Nachricht von einem Tracker bekommt, wird die CreateMsg()-Methode aufgerufen. Daraus folgt, dass der Kommunikationsdienst ein MMS-Muster erzeugt. Innerhalb des MMS-Musters ist ein Token gespeichert.

Wenn der Vorgang vollständig ausgeführt ist, erhält die erwartete Person das Token auf ihrem Smartphone.

4.3.3.3. Klassendiagramm

Im Abbildung 4.11 werden nur die wichtigste Klassen, die Beziehungen zwischen den Klassen und die Methoden aufgelistet.

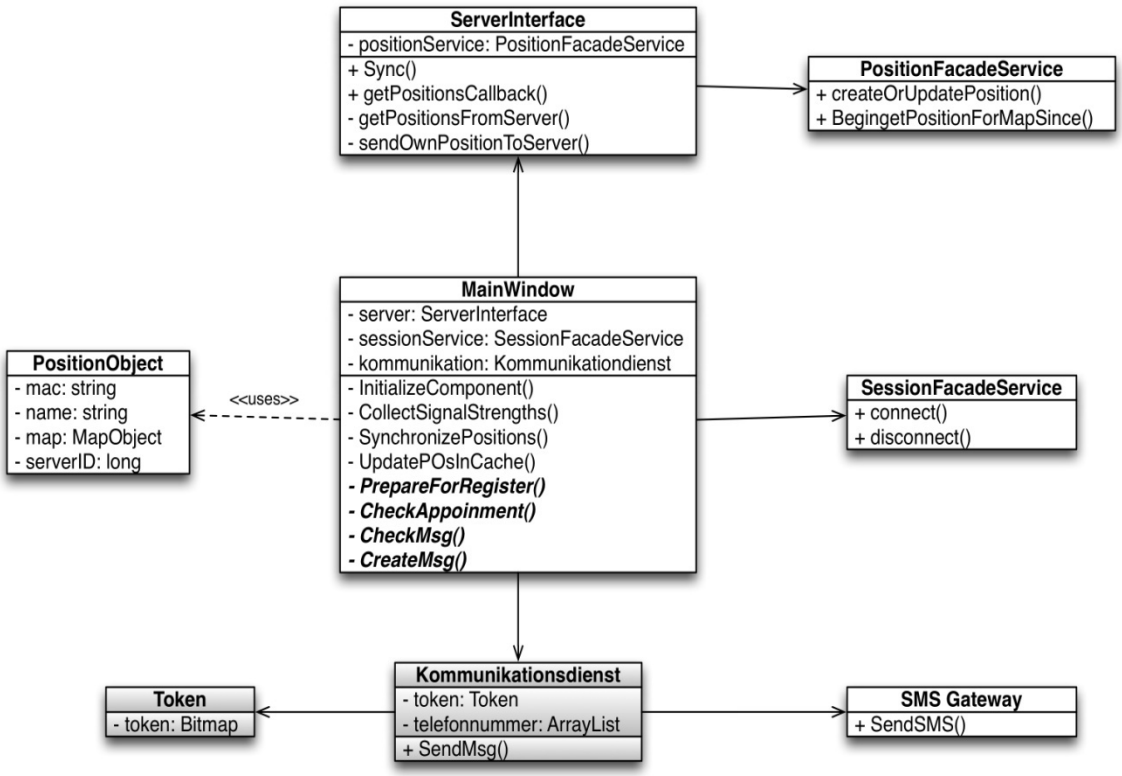

Abbildung 4.11 Klassendiagramm von der Anwendung auf Benutzergerät

Die grau gefärbte Klassenobjekte sind in dieser Arbeit entwickelt worden, ebenso wie die Methodennamen, die mit fettkursiv Schriftart geschrieben wurden.

Zuordnung der Klassen auf Komponenten (s. Abbildung 4.3):

- Lokalisierungskomponente: MainWindow
- Datenzugriffskomponente: ServerInterface
- Externer Service: PositionFacadeService, SessionFacadeService, SMS Gateway
- Hilfskomponente (wird nicht abgebildet): PositionObject, Token

[MainWindow]
Die Klasse MainWindow stellt in der mobilen Anwendung (CE-Client) eigentlich nicht nur die Lokalisierungskomponente dar. Diese Klasse ist leider ungünstig entworfen worden und deshalb besitzt die MainWindow Klasse fast alle wichtige Methoden des MagicMap CE-Clients.

Die CollectSignalStrengths()-Methode dient zum Polling der vorhandenen WLAN-Signalstärke. Diese Signale werden intern zu Positionsinformation umgewandelt und daraufhin zum Server gesendet. Innerhalb dieser Methode wird die UpdatePOsInCache()-Methode aufgerufen. Der Aufruf bewirkt die Aktualisierung der Positionsinformation anderer CE-Clients.
Für den Versand der Positionsdaten beauftragt die MainWindow-Klasse die ServiceInterface Klasse durch die SynchronizePositions() Methode. Innerhalb dieser Methode wird die eigentliche Sync()-Methode der ServerInterface-Klasse aufgerufen.
Die PrepareForRegister()-Methode ist für die Erstellung der zur Registrierung benötigen Informationen zuständig: die Telefonnummer, eigener Name und die Namen der anderen Beteiligten.

Die CheckMsg()-Methode überprüft das Ankommen einer Nachricht, die von einem Tracker gesendet wird (s. Tabelle 3.6). Wenn die Nachricht angekommen ist, wird die CreateMsg()-Methode aufgerufen. Dieser Aufruf ruft die SendMsg()-Methode aus der Kommunikationsdienst-Klasse.

[ServerInterface]
Die ServerInterface-Klasse ist für die Kommunikation zwischen dem CE-Client und dem Server zuständig.
Diese Klasse hat mindestens zwei wichtige Methoden, die von der MainWindow Klasse aufgerufen werden, die sind: Sync()- und getPositionsCallback().
Die Sync()-Methode synchronisiert alle Objekte zwischen dem Server und dem Client. Innerhalb dieser Methode gibt es zwei Methoden, die den Synchronisationsvorgang realisieren, die sind: sendOwnPositionToServer() und getPositionsFromServer().
Die sendOwnPositionToServer()- und getPositionsFromServer()-Methode funktionieren wie die Methode des Identifikationsdienstes bzw. Registrierungsdienstes. Die sendOwnPositionToServer()-Methode sendet die

Positionsdaten zum Server, in der die createOrUpdatePosition()-Methode des PositionFacadeServices aufgerufen wird.

Die getPositionsFromServer()-Methode holt die Positionsdaten aller anderen Benutzer vom MagicMap-Server ab. Diese Methode hat einen asynchronen Rückruf, der als die getPositionsCallback()-Methode realisiert wird. Dieser Aufruf bewirkt, dass die Ausführung andere Funktionen des CE-Clients soweit wie möglich davon nicht beeinträchtigt werden. Der asynchrone Rückruf wird in einer BegingetPositionForMapSince()-Methode aufgerufen.

[Kommunikationsdienst]
Die Kommunikationsdienst-Klasse ist für die Kommunikation zwischen den CE-Clients zuständig.
Diese Klasse besitzt eine wichtige Methode, die SendMsg() heißt. Diese Methode bewirkt, dass eine MMS-Nachricht zu einem anderen CE-Client gesendet wird, in dem der Kommunikationsdienst den externen Service benutzt.
Die Klasse erzeugt einen zufälligen Token, der die Position des CE-Clients auf einer MagicMap-Karte eindeutig anzeigt.
Der wichtigste Inhalt der MMS-Nachricht ist der Token.

4.4. Zusammenfassung

Im Kapitel 4.1 wurde die Architektur der Anwendung erklärt. Die Anforderungen, die in Kapitel 3 aufgelistet wurden, bilden das Gerüst für dieses Kapitel. Da MagicMap vom Werk aus nach Rich Client Ansatz entwickelt wurde, bewirkt diese Entscheidung indirekt, dass die hier vorgestellte Anwendung auch den Rich Client Ansatz weiter benutzt. Für die Softwarearchitektur werden die Komponenten des Systems in einem Drei-Schichten-Architektur-Model entwickelt. Diese Schichtenaufteilung ist für die Entwicklung dieser Arbeit am Besten geeignet, da daraus ein besserer Gesamtüberblick gezogen werden kann. Das bedeutet, es ist einfacher zu entscheiden, wo die jeweiligen Schichten entwickelt werden müssen. In diesem Fall werden die Präsentations- und Anwendungsschicht auf der Clientanwendung und die Datenhaltungsschicht auf dem Server entwickelt.

Die Wahl des Entwurfsmusters (Plugin-Entwurfsmuster) wurde vom MagicMap als Basis dieser Arbeit vorgegeben. Dieses Entwurfsmuster besitzt unter anderem folgende Vorteile: das Hinzufügen neuer Komponenten wird erleichtert, das Projektdesign wird vereinfacht und die Entwicklung eines Plugins bleibt von der Entwicklung des Hauptsystems unabhängig. Dieses Entwurfsmuster entspricht der Vorgabe aus Kapitel 3.4.5.

Nach der Vorgabe in Kapitel 4.1 wurde die Realisierung der Anwendung in Kapitel 4.3 beschrieben. In Kapitel 4.3 wurde der Realisierungsschritt genauer unter die Lupe genommen. Während des gesamten Entwicklungsprozesses wurde der Entwurf des Systems aus Kapitel 4.1 berücksichtig.

Die im Rahmen dieser Arbeit entstandenen Dienste wurden nach dem Installationsort[44] unterteilt und teilweise erklärt. Die Funktionalitäten einzelner Diensten wurden anhand vom Sequenzdiagramm und Klassendiagramm genauer erörtert, wobei durch Sequenzdiagrammen der Arbeitsablauf eines Dienstes und durch Klassendiagrammen die jeweiligen Klassen, deren Methoden und Beziehungen erläutert wurden.

[44] Entweder auf dem Smartphone oder auf dem Rechner.

5. Fazit

Für die Entwicklung dieser Arbeit wurden zuerst die benötigten Grundlagen recherchiert und erläutert. Danach wurden die zurzeit am Markt vorhandenen vergleichbaren Arbeiten ausgesucht und bezüglich der Techniken der jeweiligen Arbeiten, die benutzt worden sind, erklärt. Am Ende wurden die kontinuierliche Ortung und die Push Dienste als Grundlage für den mobilen Verabredungsassistenten ausgewählt.

Des Weiteren wurde die zu entwickelnde Arbeit nach den funktionalen und nicht-funktionalen Anforderungen untersucht. Die funktionalen Anforderungen wurden anhand der UML-Sprache definiert und modelliert. Dafür wurden Use-Case Diagramm benutzt und jede Use-Case in einem tabellarischer Form genauer betrachtet.
Die nicht-funktionalen Anforderungen wurden für den Entwurf des Systems miteinbezogen. Wobei hier der Sicherheitsaspekt aus zeitlichen Gründen nicht betrachtet wurde.

Der grundlegende Entwurf eines Softwaresystems ist unter Einhaltung der funktionalen Anforderungen abgeschlossen und in einem vorhandenen Framework für Indoor Lokalisierung integriert. Dazu wurde insbesondere die Plugin Architektur mit der losen Koppelung zwischen Plugin verwendet.

Das Ergebnis dieser Arbeit besteht aus drei wichtige Diensten, Identifizierungs-, Registrierungs- und Kommunikationsdienst. Diese drei Dienste sind auf MagicMap-Client und MagicMap CE-Client verteilt und mit Hilfe diesen Diensten kann das Szenario, das im Kapitel 3.2 vorgestellt wurde, realisiert werden.

5.1. Auftretende Probleme

Die Probleme, die während der Entwicklung dieser Arbeit aufgetreten sind, werden im Folgenden aufgelistet.

MagicMap lieferte während des Tests leider ungenaue Positionsbestimmung. Dieses Problem könnte von den Wänden des Gebäudes, der Qualität des WLAN Signalempfänger des Smartphone oder der Qualität des WLAN Sender verursacht werden.

Der aktuelle Version des Tracker ist leider noch nicht in der Lage, das Ereignismuster des MagicMap-Servers auswerten. Das soll aber im kommenden Version beheben sein.

Die Endanwendung ist leider ungetestet, da die Kommunikation zwischen den Autor dieser Arbeit und den Entwicklern der Basissoftware MagicMap nicht einwandfrei gelaufen ist. Die Schwierigkeit der Anschaffung eines eigenen Servers hat auch dazu beigetragen, dass die Endanwendung ungetestet ist.

5.2. Ausblick

Diese Arbeit hat an sich viele weitere Entwicklungsmöglichkeiten. Diese sind aber wegen kurzer Entwicklungszeit nur schwer zu erreichen. Außerdem gibt es unendliche Möglichkeiten, die gemacht werden könnte. Eine Auflistung solche Entwicklungsmöglichkeiten werden den Rahmen dieser Arbeit sprengen. Deshalb wird ein paar Möglichkeiten erörtert.

Ein Beispiel dafür ist die Integration der MMS-Kommunikation des Systems. Der Benutzer bekommt zurzeit das Token in seinem normalen Posteingangsfach.
Das Anzeigen einer Nachricht innerhalb der laufenden Anwendung würde für den Benutzer die Bedienung der Anwendung vereinfachen, wobei das Wechseln zwischen die Anwendungen[45] verringert wird.

Ein zweites Beispiel ist die Automatisation der Erkennung des Stockwerkes. Es soll erreicht werden, dass der Benutzer vollständig von manuellem Wechsel zwischen Karten befreit wird. Zur Zeit ist das System noch nicht in der Lage, wenn der Benutzer das Stockwerk wechselt, automatisch den Kartenkontext anzupassen.

Ein anderes Beispiel ist eine Möglichkeit den eingetragenen Verabredungstermin dynamisch und automatisch zu ändern. Das könnte in dem Fall passieren, wenn die wartende Person aus irgendeinem Grund das Gebäude nach der Registrierung verlassen musste. In diesem Fall sollte das System diese Veränderungen wahrnehmen und eventuell die Rolle ändern, die erwartete Person als die wartende Person und umgekehrt um zu definieren.

Die letzten beiden Entwicklungsmöglichkeit setzen eine Erweiterung des Trackers voraus. Deshalb können diese Möglichkeiten nur in Zusammenarbeit bzw. Einstimmung mit dem Autor des Trackers erarbeitet werden.

[45] In diesem Fall, zwischen das MagicMap CE und Posteingangfach des Smartphone.

Literaturverzeichnis

[Bahl, u.a. 2000] Bahl, P; Padmanabhan, V: RADAR: An In-Building RF-based User Location and Tracking System ~ URL: http://research.microsoft.com/en-us/um/people/padmanab/papers/infocom2000.pdf; Letzter Zugriff: 1. Dezember 2009

[Balzert, H 2001] Balzert, H: Lehrbuch der Software-Technik. 2. Auflage, Spektrum Akademischer Verlag, 2001

[Cache] Cache ~ URL: http://de.wikipedia.org/wiki/Cache; Letzter Zugriff: 5. April 2010

[Cricket] The Cricket Indoor Location System ~ URL: http://nms.lcs.mit.edu/projects/cricket/; Letzter Zugriff: 15. Januar 2010

[DesJardins, G 1993] DesJardins, G: TDOA/FDOA technique for locating a transmiter ~ URL: http://www.google.de/patents?hl=de&lr=&vid=USPAT5570099&id=QngkAAAAEBAJ&oi=fnd&dq=TDOA&printsec=abstract#v=onepage&q=&f=false

[Doerfler, R 1993] Doerfler, R. E. : Dead Reckoning: Calculating Without Instruments, Gulf Publishing Company, 1993

[Ekahau] Ekahau Positioning Engine ~ URL: http://ekahau.com/; Letzter Zugriff: 15. Januar 2010

[Erwin, Loehnert 2008] Erwin, Loehnert: INDOOR – Galileo/GPS Indoor Navigation & Positioning with Particular Respect to Security-Sensitive Applications ~ URL: http://www.indoor-navigation.de/publications/IFEN-INDOOR-Presentation-Systems08.pdf; Letzter Zugriff: 15. Februar 2010

[ESPER] EsperTech Event Stream Intelligence ~ URL: http://esper.codehaus.org/; Letzter Zugriff: 17. März 2010

[Finkenzeller, K 2003] Finkenzeller, K: RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification, John Wiley & Sons, 2003

[Flink, U 2008] Flick, U: Triangulation: Eine Einführung, VS Verlag, 2008

[FriFi] FriFi ~ URL: http://www.frifi.net; Letzter Zugriff: 2. Februar 2010

[Glinz, M 2005] Glinz, M: Nicht-funktionale Anforderungen ~ URL: www.ifi.uzh.ch/req/ftp/ses/kapitel_12.p.pdf; Letzter Zugriff: 20. Februar 2010

[GoogleMap] Google Maps ~ URL: http://maps.google.de

[Gregor, S 2006] Gregor, S: Entwicklung einer Hardwareplattform für die Ermittlung von Positionsdaten innerhalb von Gebäuden ~ URL: http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/gregor.pdf; Letzter Zugriff: 15. Januar 2010

[Gümüskaya, u.a. 2005] Gümüskaya, H; Hakkoymaz, H: WiPoD Wireless Positioning System based on 802.11 WLAN Infrastructure ~ URL: http://www.waset.org/journals/waset/v9/v9-22.pdf; Letzter Zugriff: 15. Januar 2010

[IntelliJ] Best Java IDE :: Do more high quality code in less time with IntelliJ IDEA ~ URL: http://www.jetbrains.com/idea/; Letzter Zugriff: 10. April 2010

[JADEX] JADEX: A Short Overview ~ URL: http://vsis-www.informatik.uni-hamburg.de/getDoc.php/publications/212/jadex_node.pdf; Letzter Zugriff: 17. März 2010

[JESPA] MagicTracker-Plugin ~ URL: http://wiki.informatik.hu-berlin.de/nomads/index.php/MagicTracker-Plugin#JESPA; Letzter Zugriff: 17. März 2010

[KNN] K-Nearest Neighbour ~ URL: http://www.cs.utsa.edu/~bylander/cs6243/nearest-neighbor.pdf; Letzter Zugriff: 20. Februar 2010

[LogixMobile] mCore .NET SMS Library ~ URL: http://www.logixmobile.com/products/mcorelib/index.asp; Letzter Zugriff: 2. April 2010

[MagicMap] MagicMap: Ein System zur kooperativen Positionsbestimmung über WLAN ~ URL: http://www2.informatik.hu-berlin.de/rok/MagicMap/index.htm; Letzter Zugriff: 9. März 2010

[Merico, u.a. 2007] Merico, D; Bisiani, R: Indoor Navigation with Minimal Infrastructure ~ URL: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4167831; Letzter Zugriff: 1. Dezember 2009

[Mäs u.a. 2006] Mäs, Stephan; Reinhardt, Wolfgang; Wang, Fei: Conception of a 3D Geodata Web Service for the Support of Indoor Navigation with GNSS ~ URL: http://www.indoor-navigation.de/publications/3DGeoInfo_Paper_StephanMaes.pdf; Letzter Zugriff: 15. Januar 2010

[NowSMS] Now SMS/MMS Gateway and MMSC ~ URL: http://www.nowsms.com/; Letzter Zugriff: 2. April 2010

[Plugin Pattern] Design Patter – Plugin Pattern ~ URL: http://msdn.microsoft.com/en-us/library/cc511899.aspx; Letzter Zugriff: 17. März 2010

[Ruppel, u.a. 2009] Ruppel, P; Gschwandtner, F; Schindhelm, C; Linnhoff-Popien, C: Indoor Navigation on Distributed Stationary Display Systems ~ URL: http://ieeexplore.ieee.org/search/wrapper.jsp?arnumber=5254282; Letzter Zugriff: 15. Januar 2010

[Schoder, u.a. 2002] Schoder, D; Teichmann, R; Fischbach, K: Peer-to-Peer: Ökonomische, technologische und juristische Perspektiven, Springer, Berlin, 2002

[Tanenbaum, u.a. 2003] Tanenbaum, A. S.; Steen, M. van: Verteilte Systeme, Pearson Studium, 2003

[VS] Visual Studio ~ URL: http://www.microsoft.com/germany/visualstudio/products/team/visual-studio-ultimate.aspx; Letzter Zugriff: 10. April 2010

[WikiOrt] Ortsbestimmung ~ URL: http://de.wikipedia.org/wiki/Ortsbestimmung; Letzter Zugriff: 2. Februar 2010

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 25. Mai 2010	
Ort, Datum	Unterschrift