

Evaluierung einer dynamischen
Informationsanzeige basierend auf
Location Awareness

Studienarbeit 1

Verfasser:	Christoph Gebhardt, Christoph Marquis
Studiengang:	Informationstechnik
Bearbeitungszeitraum:	4 Monate
Kurs:	TIT07INA
Betreuer:	Wolfgang Weyand

Erklärung

Wir versichern hiermit, dass wir die vorliegende Arbeit mit dem Thema

„Darstellung von Informationen in Echtzeit mit Location Awareness“

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet haben.

Stuttgart, den 18. Februar 2010

Christoph Gebhardt

Christoph Marquis

Zusammenfassung

In dieser Studienarbeit wird ein Konzept zur Darstellung von standortbezogenen Umgebungsinformationen auf einem mobilen Endgerät vorgestellt. Es geht darum, die statischen Informationen heutiger Navigationssysteme durch dynamische Inhalte zu ergänzen.

Dabei werden generelle Entscheidungen zur Systemarchitektur getroffen und verschiedene Anwendungsfälle für das System untersucht.

Des Weiteren wird ein konkreter Prototyp entwickelt, der die Basisfunktionen einer Location-Awareness-Anwendung implementiert. Dadurch werden das Konzept validiert und diesbezügliche Weiterentwicklungen ermittelt.

Abstract

This student research project presents a concept for displaying location based environment information. The goal is to extend the static information provided by today's navigation systems through dynamic contents. In this context, general decisions concerning the systems architecture are made, and different use cases are investigated.

Furthermore a concrete prototype is developed, which implements the base functions of a location awareness application. In that way, the concept is validated, and potential further developments are identified.

Inhaltsverzeichnis

ERKLÄRUNG	2
ZUSAMMENFASSUNG	3
ABSTRACT	3
INHALTSVERZEICHNIS	4
1 EINLEITUNG	7
1.1 EINFÜHRUNG IN DAS THEMA	7
1.2 AUFGABENSTELLUNG	7
1.3 SZENARIO	8
2 THEORETISCHE/TECHNISCHE GRUNDLAGEN	10
2.1 GPS	10
2.1.1 <i>Allgemein</i>	10
2.1.2 <i>Positionsbestimmung</i>	12
2.1.3 <i>Assisted GPS</i>	16
2.2 OPENSTREETMAP	19
2.2.1 <i>Allgemein</i>	19
2.2.2 <i>Einbindung von OpenStreetMap-Karten</i>	21
3 ANALYSE	23
3.1 LOCATION-AWARENESS-SYSTEMARCHITEKTUR	23
3.1.1 <i>Fat Client</i>	23
3.1.2 <i>Client-Server-System</i>	25
3.1.3 <i>Auswahl eines Systems</i>	27

3.2	ANWENDUNGSFÄLLE	28
3.2.1	<i>Display</i>	28
3.2.2	<i>Mobiles Endgerät</i>	29
3.2.3	<i>Administration</i>	32
4	KONZEPT	34
4.1	SYSTEMARCHITEKTUR DES PROTOTYPEN	34
4.2	SOFTWARE-DESIGN DES CLIENT	36
4.2.1	<i>Gesamter Client</i>	36
4.2.2	<i>GPS-Daten-Verwaltung</i>	38
4.2.3	<i>Hauptprogramm</i>	40
4.3	SERVERABLÄUFE	42
4.3.1	<i>Allgemein</i>	42
4.3.2	<i>POI-Behandlung</i>	43
4.3.3	<i>Session-Handling</i>	45
4.4	DATENBANKKONZEPT	46
4.5	ADMINISTRATIONS-INTERFACE	48
5	ENTWICKLUNG DES PROTOTYPS	51
5.1	ENTWICKLUNG DES CLIENT	51
5.2	SERVERSEITIGE PROGRAMMIERUNG	55
5.2.1	<i>Testsystem</i>	55
5.2.2	<i>Server-Skript</i>	56
5.3	TESTFÄLLE	59
5.3.1	<i>Testen der GPS-Daten-Ermittlung</i>	59
5.3.2	<i>Testen der Webserver-Grundfunktionalität</i>	61
5.3.3	<i>Testen des Location-Awareness-Prototyps</i>	62
6	SCHLUSSBETRACHTUNG UND AUSBLICK	64

ANHANG A – LITERATURVERZEICHNIS	66
BUCHQUELLEN.....	66
INTERNETQUELLEN	67
ANHANG B – ABBILDUNGSVERZEICHNIS	68
ANHANG C – GLOSSAR UND ABKÜRZUNGSVERZEICHNIS	69
ANHANG D - ANLAGEN.....	72

1 Einleitung

1.1 Einführung in das Thema

In der heutigen Zeit ist der Wettstreit um die Gunst des Kunden wichtiger denn je. Im täglichen Leben ist jeder Mensch einer noch nie dagewesenen Informationsflut ausgesetzt. Diese erreicht uns über alle erdenklichen Medien wie Fernsehen, Rundfunk, Internet, usw. Allerdings erscheint uns ein Großteil dieser Informationsangebote unnützlich, weil wir sie am falschen Ort und/oder zur falschen Zeit aufnehmen.

Genau diesen Kritikpunkt der Informationsbereitstellung greift das Thema Location Awareness auf. Dieser große Titel beschreibt die ortsbezogene Bereitstellung von Informationen und Diensten. „Bei solchen ,ortsbezogenen Diensten‘ kennt der Dienstanbieter die Position des Nutzers und bietet automatisch nur die Informationen, die für seine aktuelle Position relevant sind.“ ([COMMERCE 02] S.386)

Es wird also versucht, dem Kunden nur Informationen (z. B. kulturellen oder kommerziellen Inhalts) über Standorte zu vermitteln, die er auch binnen kurzer Zeit erreichen kann. So werden insgesamt zwar weniger Menschen angesprochen, diese sind aber aufgrund ihrer Nähe zum Werbenden potenzielle Kunden.

1.2 Aufgabenstellung

Bei dem Thema „Darstellung von Informationen in Echtzeit mit Location Awareness“ geht es um die Entwicklung eines Konzepts zur Ermittlung und Darstellung standortbezogener Informationen anhand der Positionsdaten eines Navigationssystems.

Im Gegensatz zum relativ statischen Informationsgehalt heutiger Navigationsanwendungen soll eine Location-Awareness-Applikation dynamische Umgebungsinformationen bereitstellen.

Das Ziel ist es, den Systemkontext vorzustellen, verschiedene Anwendungsfälle zu ermitteln und das Gesamtsystem zu konzeptionieren.

Anschließend soll das Konzept durch die Entwicklung eines Softwareprototyps validiert werden. Anhand dieses Prototyps wird die Realisierbarkeit der verschiedenen ermittelten Use Cases überprüft.

1.3 Szenario

Man stelle sich vor, man fährt abends mit der S-Bahn in die Innenstadt, ohne genauere Pläne zu haben. Durch das Fenster sieht man das Kino, welches sich in der Nähe der nächsten S-Bahn-Station befindet. Kurze Zeit später zeigen Displays im Innenraum der Bahn das aktuelle Kinoprogramm mit der dazugehörigen Anzahl an noch verfügbaren Eintrittskarten. Da das aktuelle Kinoprogramm nichts für den eigenen Geschmack beinhaltet, entschließt man sich, weiter in Richtung Stadtmitte zu fahren. Die Bahnstrecke führt weiterhin am Theaterhaus sowie an einigen Diskotheken vorbei. Auch deren Abendprogramme erscheinen auf den vorher genannten Displays, sobald sich der Zug den Objekten nähert. Auf Grund der erhaltenen Informationen über die verschiedenen Lokalitäten entscheidet man sich, ins Theaterhaus zu gehen.

Ein weiteres Szenario:

Im Urlaub befindet man sich in einer fremden Großstadt. Man ist ortsfremd und auf der Suche nach den örtlichen Sehenswürdigkeiten. Man startet eine Applikation auf dem Handy. Diese erkennt über einen GPS-Empfänger die aktuelle Position des Anwenders und sucht in einer Datenbank nach interessanten Objekten, die sich in der Umgebung befinden. Über die Filterfunktion grenzt der Anwender alle Objekte aus, die nicht zu den Sehenswürdigkeiten gehören. Durch einen Klick auf eines der übrig gebliebenen Objekte werden genauere Informationen darüber angezeigt. Unter anderem kann sich der Anwender eine Wegbeschreibung zum gewünschten Objekt anzeigen lassen. Nach Besichtigung einiger Sehenswürdigkeiten verspürt man Hunger. Man startet die Applikation erneut und nutzt dann die Filterfunktion „Gastronomie“. Daraufhin werden in der Nähe liegende Restaurants, Cafés oder Bars angezeigt.

Diese beiden Szenarien sollen Anwendungsfälle für die Darstellung ortsbezogener Informationen beschreiben. Das Ziel dieser Studienarbeit ist die Entwicklung eines Systems, durch das Informationen in der beschriebenen Art und Weise dargestellt werden können.

2 Theoretische/Technische Grundlagen

2.1 GPS

2.1.1 Allgemein

GPS ist ein Satelliten-Navigationssystem, der offizielle Name NAVSTAR-GPS steht für „Navigation System with Timing and Ranging – Global Positioning System“.

Die Idee für das Projekt kam vom U.S. Department of Defense (DOD), das 1973 das Joint Program Office (JPO) mit der Entwicklung eines Weltraum-Navigationssystems beauftragte. Obwohl GPS ursprünglich zu militärischen Zwecken entwickelt wurde, entschieden der US-Kongress und der Präsident, dass es auch zivil eingesetzt werden können soll.

W.Woods definierte GPS 1985 relativ genau mit dem Satz:

„The Navstar Global Positioning System (GPS) is an all-weather, spacebased navigation system under development by the Department of Defense (DoD) to satisfy the requirements for the military forces to accurately determine their position, velocity, and time in a common reference system, anywhere on or near the Earth on a continuous basis” ([GPS 01], S.11).

GPS übermittelt seinen Benutzern zwei Werte: einerseits den eigenen Standort mit genauer geographischen Länge, Breite und Höhe (über NN) und andererseits die genaue aktuelle Zeit als koordinierte Weltzeit (Universal Time Coordinate (UTC); eine international eindeutige Referenzzeit). Aus diesen Werten können dann weitere Informationen wie beispielsweise die Geschwindigkeit errechnet werden.

Dazu schließt das System von den bekannten Positionen der Satelliten auf die unbekannte Position eines Objekts (vgl. [GPS 01], S.11). Um die vier zu ermittelnden Werte (geographische Länge, Breite, Höhe und Zeit) zu errechnen, muss ein GPS-Empfänger mit vier Satelliten gleichzeitig in Verbindung treten. Jeder der Satelliten besitzt zur genauen Zeitbestimmung vier Atomuhren. Ein Satellit versendet ein Signal mit dem Zeitpunkt der Absendung, so kann ein synchronisierter Empfänger den Abstand zu den Satelliten anhand der Sendezeit und der Geschwindigkeit des Signals berechnen (nähere Informationen: siehe Kapitel 2.1.2).

Das GPS-Gesamtsystem setzt sich aus drei Segmenten zusammen, dem Weltall-Segment, das aus den Satelliten besteht, die Signale senden, dem Kontroll-Segment, welches das System steuert, sowie dem Benutzer-Segment, das die Werte mit vielen verschiedenen Anwendungen verwendet und auswertet.

Das Weltraum-Segment besteht aus insgesamt 31 Satelliten, davon gewährleisten 24 den GPS-Betrieb, während die anderen zur Ausfallsicherheit beitragen.

Die 24 Satelliten im Betrieb sind auf sechs verschiedenen Umlaufbahnen mit einer Neigung von 55° gegenüber dem Äquator verteilt (4 Satelliten pro Bahn), wobei jeder Satellit mit seinem Wirkungsbereich jeweils ca. $1/3$ der Erdoberfläche abdeckt. Dadurch wird sichergestellt, dass an jeder Position der Erde mindestens vier Satelliten sichtbar sind und dass so die Positionsbestimmung möglich ist.

Die Aufgaben des Kontroll-Segments sind auf drei verschiedene Bestandteile aufgeteilt.

Die Master Control Station (MCS) sammelt die Verfolgungsdaten der Satelliten von den Monitoring Stations (MS) und berechnet daraus anhand des Kalman-Filters die voraussichtliche Satellitenumlaufbahnen und die Clock-Parameter. Die schon erwähnten Monitoring Stations empfangen die Signale aller Satelliten, die sich in ihrer Sicht befinden, und messen so die Entfernungen zu ihnen aus. Die dritte Aufgabe übernehmen die Ground Control Stations (GCS), sie sind die Kommunikationsverbindung zu den Satelliten. Sie übertragen die ermittelten Ephemeriden (Tabellen, welche die Umlaufbahn eines Satelliten beschreiben) über Bodenantennen und synchronisieren die Zeit der Satelliten untereinander.

Das Benutzer-Segment lässt sich in zwei verschiedene Arten von Benutzern unterteilen, die zivilen Nutzer und die militärischen Nutzer. „Das zivile Signal SPS (Standard Positioning Service) ist von der Allgemeinheit frei nutzbar, während das militärische Signal PPS (Precise Positioning Service) nur von autorisierten Stellen genutzt werden darf“ ([GPS 09], S. 11). SPS ist mit dem C/A-Code verschlüsselt, es hat eine Frequenz von 1575,42 MHz, der sogenannten L1-Frequenz. PPS verwendet das P(Y)-Signal auf der L2-Frequenz von 1227,60 MHz.

GPS hat für den militärischen Benutzer den Vorteil, dass der Empfänger kein Signal sendet, durch das er seine Position verraten könnte, sondern nur ein Signal empfängt und trotzdem die genaue Position ermitteln kann. Typische Anwendungen, die im militärischen, aber auch immer mehr im zivilen Bereich verwendet werden, sind die Routen-Navigation, das Orten, die Positionierung sowie die genaue Bestimmung der Zeit.

Im zivilen Bereich gibt es noch diverse weitere Anwendungen, und mit der Entwicklung von Smartphones oder UMTS-fähigen Netbooks tritt ein ganz neues Segment von GPS-Anwendungen auf den Markt, sogenannte Location Based Services (LBS). „Als *standortbezogene Dienste (engl. Location Based Services, LBS) werden Dienstleistungen oder Angebote bezeichnet, die sich auf den aktuellen Standort des Benutzers (Mobilfunkteilnehmers, ausgerüstet mit einem Handy) beziehen*“ ([GPS 09], S. 150).

2.1.2 Positionsbestimmung

Wie schon erwähnt, bestimmt ein GPS-Empfänger vier verschiedene Werte, die geographische Länge, Breite und Höhe sowie die genaue aktuelle koordinierte Weltzeit. Um diese vier Unbekannten mathematisch berechnen zu können, braucht man bekannter Weise vier Gleichungen. Diese werden von den vier Satelliten geliefert, die ein Empfänger zur genauen Bestimmung seiner Position benötigt. Um die Position des Empfängers aus den Positionen der Satelliten erschließen zu können, wird das gleiche Prinzip wie zur Bestimmung der Entfernung von einem Gewitter verwendet. Bei einem Gewitter misst man die Zeit zwischen dem gesehenen Blitzschlag und dem gehörten Donner und multipliziert dieses Ergebnis mit der Schallgeschwindigkeit. GPS funktioniert in ähnlicher Weise: der Satellit sendet mit dem Signal immer seine aktuelle Zeit, ein synchronisierter Empfänger kann so durch den Vergleich der Ankunfts- und Absende-Zeit die Laufzeit ermitteln. Des Weiteren weiß man, dass der Satellit mit einer Frequenz von 1575,42 MHz sendet und dass sich das Signal mit Lichtgeschwindigkeit ausbreitet. Die Laufzeit für einen Empfänger auf Meeresspiegelniveau direkt unter dem Satelliten beträgt daher 67,3 ms. Für jeden Kilometer, den das Signal zusätzlich überwinden muss, werden

3,33 μs zusätzlich benötigt. Durch die einzelnen Laufzeitlängen der Signale der vier Satelliten und deren Positionen kann der Empfänger nun seine Position berechnen. Sie befindet sich dort, wo sich die Signalausbreitungskegel unterhalb der Satelliten schneiden.

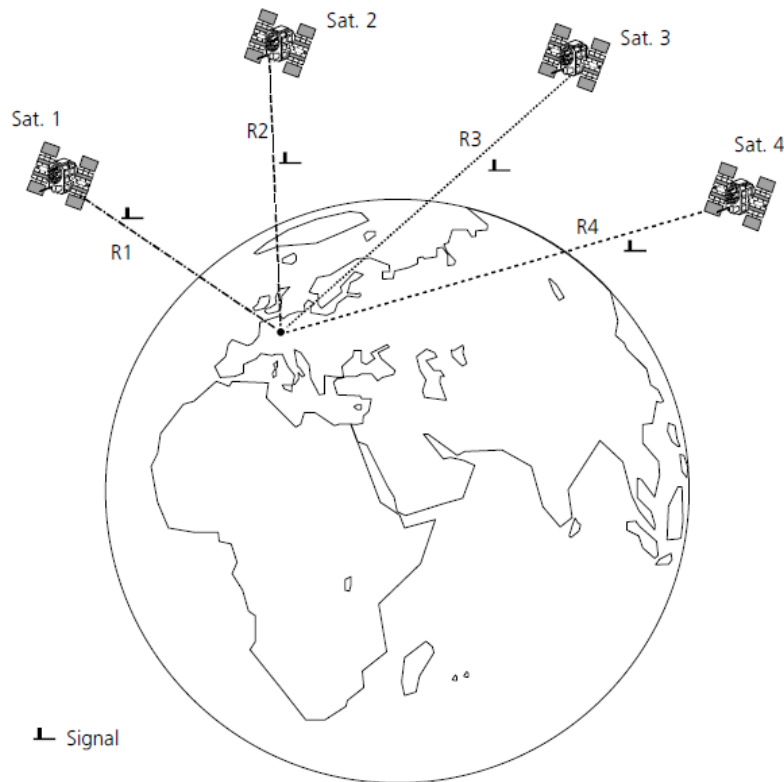


Abbildung 2-1: Positionsbestimmung aus Signallaufzeit

Quelle: [GPS 09], S.19

Es stellt sich nun die Frage, weshalb zum Bestimmen von drei Positionen vier Satelliten benötigt werden. Dies ist der Fall, da die GPS-Empfänger-Zeit nicht exakt mit der des Satelliten synchronisiert ist. Allgemein gilt: „*Werden unsynchronisierte Empfänger-Uhren verwendet, muss die Anzahl der Zeitsender um Eins grösser sein, als die Anzahl der unbekannt Dimensionen*“ ([GPS 09], S. 15). Trotz des vierten Satelliten besteht immer noch ein Zeitfehler

bei der Bestimmung der Laufzeit. Dieser spielt in die Berechnung der Entfernung von Satellit zu Empfänger ein, weshalb man in diesem Zusammenhang oft von „Pseudorange“ spricht.

Die genaue mathematische Berechnung der Position aus den vier verschiedenen Satelliten-Signalen läuft folgendermaßen ab: Man stellt zuerst eine Formel für den Zusammenhang von tatsächlicher Entfernung zur Pseudoentfernung auf.

$$\Delta t_{detected} = \Delta t + \Delta t_0$$

$$PSR = \Delta t_{detected} * c = (\Delta t + \Delta t_0) * c$$

$$PSR = R + \Delta t_0 * c$$

R: tatsächliche Entfernung zwischen Satellit und Empfänger

PSR: fehlerhafte Entfernung zwischen Satellit und Empfänger („Pseudorange“)

c: Lichtgeschwindigkeit

Δt : Laufzeit zwischen Satellit und Empfänger

Δt_0 : Differenz zwischen Zeit der Satelliten und des Empfängers

Im kartesischen Koordinatensystem lässt sich die Entfernung zum Satelliten folgendermaßen berechnen:

$$R = \sqrt{(x_{Sat} - x_{Empf})^2 + (y_{Sat} - y_{Empf})^2 + (z_{Sat} - z_{Empf})^2}$$

Eingesetzt in die Formel für die Pseudoentfernung ergibt sich folgendes Ergebnis:

$$PSR = \sqrt{(x_{Sat} - x_{Empf})^2 + (y_{Sat} - y_{Empf})^2 + (z_{Sat} - z_{Empf})^2} + \Delta t_0 * c$$

Dies gilt für die Pseudoentfernung aller vier Satelliten (i=1...4).

Um diese Wurzelgleichung zu linearisieren, wird nun die Taylor-Formel angewendet, wobei man nur das 1.Glied der Näherung in Betracht zieht:

$$f(x) = f(x_0) + f'(x_0) * \Delta x$$

Anschließend berechnet man x_{Empf} , y_{Empf} und z_{Empf} nicht direkt, sondern schätzt diese.

Für die geschätzten Koordinaten gilt:

$$x_{Empf} = x_{Ges} + \Delta x$$

$$y_{Empf} = y_{Ges} + \Delta y$$

$$z_{Empf} = z_{Ges} + \Delta z$$

Dabei sind Δx , Δy und Δz die Fehler der geschätzten gegenüber den tatsächlichen Werten.

Für die Entfernung der vier Satelliten ($i=0\dots4$) gilt jetzt demzufolge:

$$R_i = \sqrt{(x_{Sat_i} - x_{Ges})^2 + (y_{Sat_i} - y_{Ges})^2 + (z_{Sat_i} - z_{Ges})^2}$$

Wenn man nun diese Gleichung in die Formel für Pseudoentfernungen einsetzt und auf die resultierende Gleichung die Taylor-Formel anwendet, erhält man folgendes Ergebnis:

$$PSR_i = R_i + \frac{\delta(R_i)}{\delta x} * \Delta x + \frac{\delta(R_i)}{\delta y} * \Delta y + \frac{\delta(R_i)}{\delta z} * \Delta z + c * \Delta t_0$$

Leitet man diese Gleichung partiell ab, erhält man:

$$PSR_i = R_i + \frac{x_{Ges} - x_{Sat_i}}{R_i} * \Delta x + \frac{y_{Ges} - y_{Sat_i}}{R_i} * \Delta y + \frac{z_{Ges} - z_{Sat_i}}{R_i} * \Delta z + c * \Delta t_0$$

Nach dem Umstellen der Gleichung können nun die gesuchten Fehler bestimmt werden:

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta t_0 \end{bmatrix} = \begin{bmatrix} \frac{x_{Ges} - x_{Sat_1}}{R_1} & \frac{y_{Ges} - y_{Sat_1}}{R_1} & \frac{z_{Ges} - z_{Sat_1}}{R_1} & c \\ \frac{x_{Ges} - x_{Sat_2}}{R_2} & \frac{y_{Ges} - y_{Sat_2}}{R_2} & \frac{z_{Ges} - z_{Sat_2}}{R_2} & c \\ \frac{x_{Ges} - x_{Sat_3}}{R_3} & \frac{y_{Ges} - y_{Sat_3}}{R_3} & \frac{z_{Ges} - z_{Sat_3}}{R_3} & c \\ \frac{x_{Ges} - x_{Sat_4}}{R_4} & \frac{y_{Ges} - y_{Sat_4}}{R_4} & \frac{z_{Ges} - z_{Sat_4}}{R_4} & c \end{bmatrix}^{-1} * \begin{bmatrix} PSR_1 - R_1 \\ PSR_2 - R_2 \\ PSR_3 - R_3 \\ PSR_4 - R_4 \end{bmatrix}$$

Anhand der errechneten Fehler Δx , Δy und Δz werden nun neue Werte geschätzt. Diese werden wieder in die Gleichung eingesetzt. Das Verfahren wird iterativ solange durchgeführt, bis der Fehler kleiner als die gestellte Anforderung ist (Berechnung vgl. [GPS 09], S. 76-80).

2.1.3 Assisted GPS

Durch die Integration von GPS-Empfängern in Smartphones hält, wie schon erwähnt, eine neue Art von GPS-Anwendungen, die sogenannten Location Based Services, Einzug auf dem Markt der mobilen Anwendungen. Jedoch ist die Integration eines GPS-Empfängers in ein mobiles Endgerät aus folgenden Gründen nicht ganz unproblematisch.

Bei einer Unterbrechung der Kommunikation vom Satellit zum GPS-Empfänger, die länger als zwei Stunden dauert, kann das GPS-Endgerät bis zu mehreren Minuten brauchen, um die aktuellen Bahndaten der Satelliten neu zu empfangen und eine erste Position zu berechnen. Des Weiteren ist die GPS-Positionsbestimmung sehr rechenintensiv, was zu einem erhöhten Stromverbrauch des mobilen Endgeräts führt und lange Akkulaufzeiten verhindert. Es kommt hinzu, dass ein GPS-Empfänger in dichtbesiedelten städtischen Gebieten oft keine einwandfreie Sicht auf vier GPS-Satelliten hat, was die Positionsbestimmung mit klassischen GPS-Empfängern unmöglich macht.

Aus diesen Gründen wurde Assisted GPS (AGPS) entwickelt. Anthony LaMarca und Eyal de Lara definieren AGPS folgendermaßen:

“Assisted GPS or A-GPS is a hybrid positioning technique that combines the Global Positioning System (GPS) and the mobile phone network” ([LOCATIONSYSTEMS 08], S. 52).

Bei dieser Technologie werden die über einen Satelliten erhaltenen GPS-Signale um Daten ergänzt, die über das Mobilfunknetz übermittelt werden. Diese sogenannten Aiding-Daten stammen von einem festinstallierten Empfänger oder einer Monitoring Station (MS) in der Nähe des mobilen GPS-Geräts, sie enthalten die präzisen Bahndaten der Satelliten, deren Zeitinformation sowie weitere Werte, die zur Berechnung der aktuellen Position nötig sind. Anhand dieser Daten, die aus dem Mobilfunknetz schneller empfangen werden als über die Satelliten, wird die Initialisierungszeit eines GPS-Empfängers nach längerer Kommunikationsunterbrechung erheblich reduziert. Des Weiteren muss der Empfänger nicht umständlich aus den über das GPS-Signal empfangenen Daten die Satellitenkonstellation berechnen, sondern erhält diese über die Mobilfunkschnittstelle, wodurch Rechenzeit und damit auch Energie gespart wird. Der prinzipielle Aufbau von AGPS wird in folgender Abbildung erklärt.

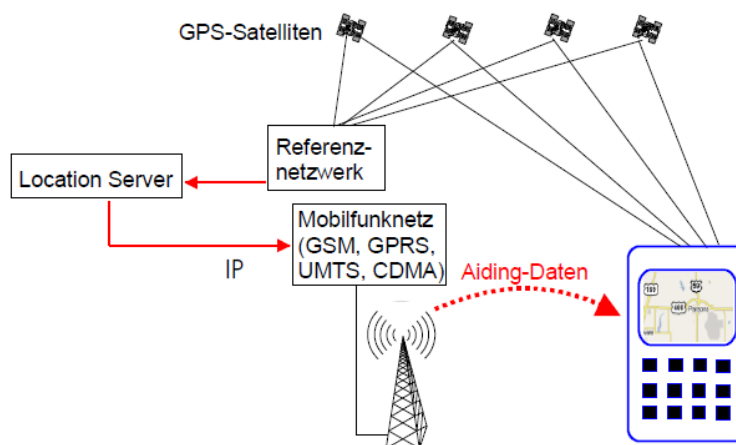


Abbildung 2-2: Systemaufbau Assisted GPS

Quelle: [GPS 09], S. 113

Ein fest installiertes Referenznetzwerk (MS) in der Nähe des mobilen Endgeräts empfängt das GPS-Signal von den Satelliten und übermittelt die Daten an einen Location Server. Dieser sendet die Werte über das Internet an das Mobilfunknetz weiter, welches die Daten über UMTS oder GSM an das Mobiltelefon übermittelt.

Somit hat das Smartphone zu dem eigenen empfangenen GPS-Signal noch die Daten eines Referenznetzwerks. Dadurch ist es dem Gerät möglich, die Position zu bestimmen, auch wenn weniger als vier Satelliten in Sichtweite sind. Jedoch nimmt die Genauigkeit der Positionsbestimmung ab, je mehr Referenznetzwerke und je weniger Satelliten beteiligt sind. Folgende Werte sind einer Tabelle des AGPS-Empfänger gpsOne der Firma Qualcomm entnommen. Sie zeigt den Fehler der Positionsbestimmung bei verschiedenen Kombinationen von Satelliten und GPS Monitoring Stations.

	Fehler (in Metern)	
	67 %	95 %
4 Satelliten	11	18
3 Satelliten + 1 MS	19	35
2 Satelliten + 2 MS	88	253
1 Satellit + 3 MS	139	345

Tabelle 1: Fehler des Qualcomm-Empfängers gpsOne bei unterschiedlichen Signalstärken

Quelle: [GPSONE 00], S.334-335

AGPS hat zwei unterschiedliche Systemarten, nämlich zum einen das netzwerkbasierte AGPS und zum anderen das handheld-basierte AGPS. Beim netzwerkbasierten AGPS übermittelt das Smartphone die von ihm ermittelten GPS-Daten, unabhängig davon, wie viele Satelliten in Sicht sind, an den Location Server. Dieser berechnet die aktuelle Position anhand der übermittelten Daten und der Satelliten-Signale, die er vom Referenznetzwerk erhalten hat. Anschließend sendet er das Ergebnis an das mobile Endgerät zurück, welches die Daten an sein GPS-Modul weitergibt. Der Vorteil dieses Verfahrens besteht darin, dass die eigentliche Berechnung auf dem

Location Server stattfindet, wodurch der Handheld Ressourcen spart. Nachteilig ist jedoch, dass der Location Server seinerseits die Position des Smartphones besitzt, was hinsichtlich des Datenschutzes als äußerst kritisch zu betrachten ist. Beim handheld-basierten AGPS empfängt das GPS-Modul die Daten vom Referenznetzwerk und berechnet anhand dieser und der eigenen empfangenen GPS-Werte seine aktuelle Position. Hierbei ist es von Vorteil, dass die Position des GPS-Moduls nur dem Handheld selbst bekannt ist und an keine Dritten weiter gegeben wird. Nachteilig ist, dass die Positionsberechnung auf dem mobilen Endgerät stattfindet und damit dessen Ressourcen beansprucht.

2.2 OpenStreetMap

2.2.1 Allgemein

Um dem Anwender nicht nur zu vermitteln, welche Points Of Interest sich in der näheren Umgebung befinden, sondern auch, wo sich diese Objekte befinden, eignen sich Karten sehr gut. Den Weg zu einem bestimmten Gebäude anhand einer Karte zu finden, ist wesentlich einfacher, als die Weginformation einem Fließtext zu entnehmen. Die Einbindung von Karten in Web-Applikationen oder Webseiten ist heutzutage ein weit verbreitetes Mittel. Dafür stehen einige Anbieter von Kartenmaterial zur Verfügung. Neben vielen anderen kommerziellen Anbietern stellt z. B. auch Google ein riesiges Kartenmaterial mit umfangreicher API zur Verfügung. Diese Karten sind allerdings für bestimmte Anwendungen lizenzpflichtig. Daher wird in dieser Studienarbeit das Kartenmaterial der OpenStreetMap-Stiftung betrachtet und verwendet. Allerdings wird die Einbindung von Karten erst im nächsten Semester im Prototyp realisiert. Trotzdem soll an dieser Stelle schon auf den Stand der Technik eingegangen werden.

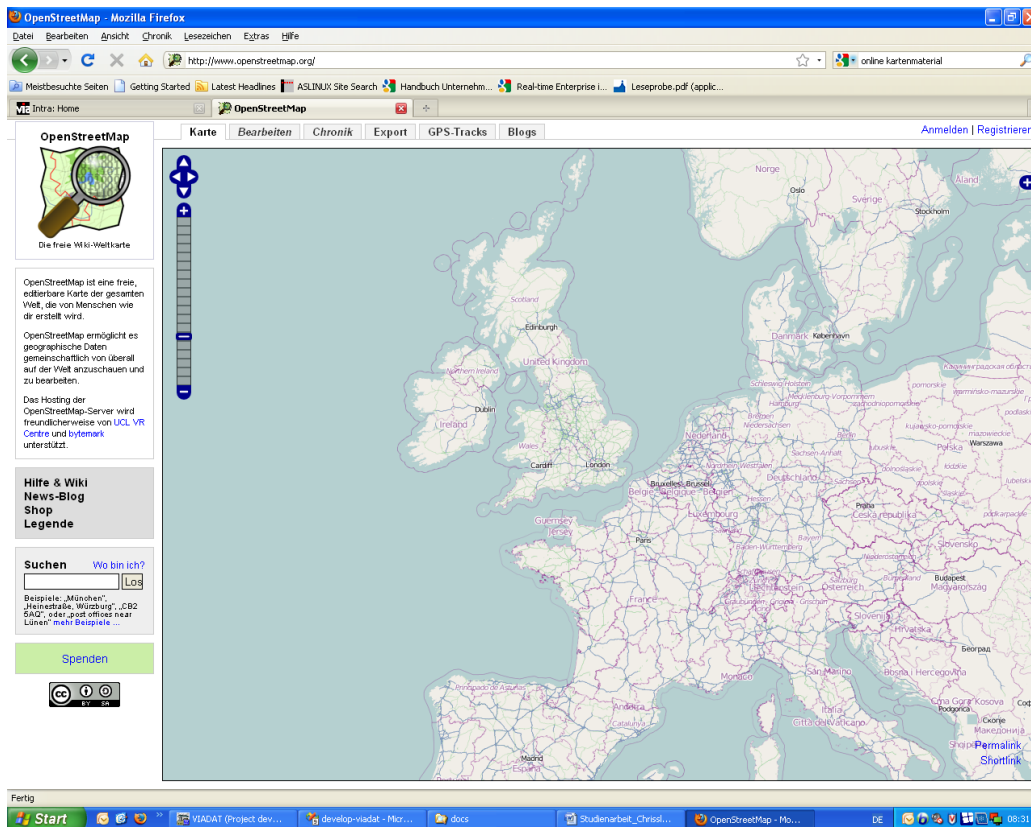


Abbildung 2-3: Startseite der OpenStreetMap-Homepage

Quelle: [OSM]

Hinter dem Namen OpenStreetMap verbirgt sich ein Projekt, in dem frei verfügbare Geodaten erstellt werden. Mit Hilfe dieser Daten können dann Welt- und andere Spezialkarten generiert werden. Eine bessere Beschreibung findet man im Computer-Lexikon 2009:

„Das Projekt OpenStreetMap (OSM) hat sich zum Ziel gesetzt, weltumspannend exakte Stadt- und Wegpläne bereitzustellen, die von jedermann frei genutzt und bearbeitet werden können. Die Daten werden mit Unterstützung freiwilliger Helfer auf der ganzen Welt gesammelt, die die Strecken mit GPS-Empfängern aufzeichnen, am heimischen PC aufbereiten und auf den Zentralserver des Projekts hochladen. Auf diese Weise entstehen präzisere Landkarten als

kommerzielle Anbieter anbieten, die zudem noch frei von Urheberrechten sind.“ ([LEXIKON 08] S. 594)

Dabei basieren die Karten auf einer wiki-ähnlichen Datenbank, in der die geografischen Informationen hinterlegt werden. Die fertigen Kartendaten werden im OSM-Format ausgetauscht. Das ist ein XML-Format, dessen Syntax den Ausgaben der OpenStreetMap-API entspricht. Das gesamte Projekt unterliegt der Creative Commons Attribution-ShareAlike 2.0 Lizenz und ist damit frei verfügbar.

2.2.2 Einbindung von OpenStreetMap-Karten

Für den Prototyp wird vorgesehen, die ortsbezogenen Informationen im HTML-Format darzustellen. Deshalb wird hier die Einbindung von OpenStreetMap-Karten in eine HTML-Webseite beschrieben. Es wäre auch möglich, die Karten über XML einzubinden und mit Hilfe eines eigenen Interpreters darzustellen. Diese Möglichkeit findet hier allerdings keine Betrachtung.

Neben erforderlichen Kenntnissen in den Techniken HTML und CSS sind auch Erfahrungen im Umgang mit JavaScript hilfreich.

Zur Einbindung einer zoombaren OpenStreetMap-Karte in eine Webseite kann man sich den notwendigen Quellcode vom OpenStreetMap-Wiki herunterladen. Diesen fügt man in die gewünschte Webseite ein. Nachdem man die Links zu den JavaScript-Dateien und zu den Stylesheets an die eigene Installation angepasst hat, kann man mit der Konfiguration der eigentlichen Kartendarstellung beginnen. Über die Parameter Längengrad (lon) und Breitengrad (lat) wird die Position eingestellt, auf welche die Karte später zentriert. Der Parameter zoom spezifiziert, wie nahe an die Erdoberfläche herangefahren wird. Über weitere Funktionen, welche die OSM-API mitbringt, können dann noch Punkte auf der Karte markiert und diesen Markierungen weitere Texte und Bilder zugefügt werden. Anschließend werden die CSS-Dateien erstellt. Auch hierfür steht der Code im Internet zum Download bereit. Die Stylesheet-Dateien können dann im Bedarfsfall noch an die Anforderungen der Webseite angepasst werden. Zuletzt

müssen noch erforderliche JavaScript-Dateien eingebunden werden. Der Code ist im Internet abrufbar.

Nachdem alle notwendigen Komponenten eingerichtet wurden, kann die editierte HTML-Datei geöffnet werden. Dabei ist es nicht notwendig, sich einen lokalen Webserver einzurichten oder die Files auf einen echten Server zu kopieren. Die Anzeige der Karten geschieht ohne Verwendung serverseitiger Codes. Die HTML-Seite kann also testweise einfach im Browser aufgerufen werden. Wurde vorher alles korrekt konfiguriert, sollte die Karte im Browser angezeigt werden. Der Fokus der Karte sollte sich auf der mit Hilfe von Längen- und Breitengrad eingestellten Position befinden. Außerdem ist es möglich, die Kartenansicht live zu verschieben bzw. aus der Ansicht heraus- bzw. in sie hineinzuzoomen.

Neben dem Anzeigen der Karten mit einigen Markierungen ist es auch möglich, Routen zu berechnen und diese in die Karte zeichnen zu lassen. In dieser Arbeit wäre es beispielsweise denkbar, den Weg von der aktuellen Position zum gewünschten Point Of Interest in der Karte anzuzeigen und die Wegfindung somit noch einfacher zu machen. Diese Anwendung soll in der weiteren Bearbeitung dieses Studienarbeit-Themas im nächsten Semester betrachtet und deren Umsetzung geprüft werden.

3 Analyse

3.1 Location-Awareness-Systemarchitektur

Zur Realisierung des in der vorliegenden Studienarbeit behandelten Themas gilt es zunächst eine grundlegende Entscheidung bezüglich der Architektur des Gesamtsystems zu treffen. Dafür kommen in erster Linie zwei Systemarchitekturen in Frage. Diese werden nachfolgend beschrieben und hinsichtlich der Verwendung in dieser Studienarbeit bewertet. Die Bewertung erfolgt auf Grund folgender Kriterien:

- **Verfügbarkeit:**
Unter welchen Bedingungen ist die Anwendung für den User verfügbar, bzw. nicht verfügbar.
- **Zugriffszeit:** Wie schnell kann auf die Komponenten der Anwendung zugegriffen werden.
- **Datenbasis:** Wo befinden sich die Datenhaltung und damit auch die POI-Berechnung, welche direkt auf diese zugreift.
- **Verteilung von Änderungen:**
Wie einfach können Änderungen (in der Datenbank und in zentralen Programmabläufen) auf den Clients verteilt werden.
- **Rechenlast:** In welcher Art und Weise wirkt sich die Architektur auf die Rechenlast der physischen Instanzen aus.

3.1.1 Fat Client

Bei einem Fat-Client-System wird im Allgemeinen „*die gesamte Anwendungslogik inklusive Datenbankzugriff auf dem Client realisiert. Die Datenbank wird als reiner Datenspeicher benutzt.*“ ([DATENBANKEN 2007], S. 155)

Abbildung 3-1 zeigt das typische Modell eines Fat Client.

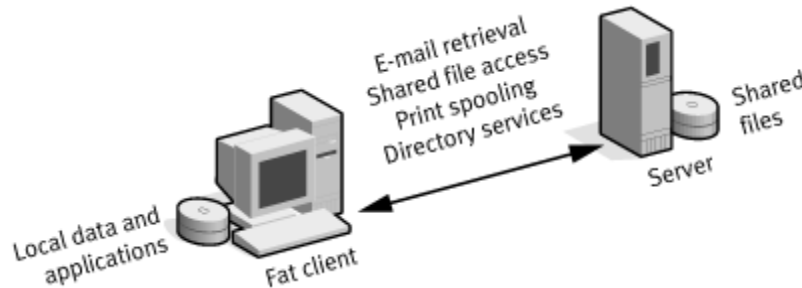


Abbildung 3-1: Aufbau einer Fat-Client-Architektur

Quelle [FAT-CLIENT]

Als Fat Clients werden im Allgemeinen Desktop-PCs bezeichnet. Diese hängen zwar vielleicht in einem gemeinsamen Netzwerk (z. B. gemeinsames Firmennetzwerk) und erledigen ähnliche Aufgaben, benötigen jedoch alle eine eigene Softwareinstallation. Jeder Rechner besitzt ein eigenes Betriebssystem und hat daher auch einen gewissen Bedarf an Arbeits- und Festplattenspeicher. Lediglich eine Datenbank, falls vorhanden, stellt einen zentralen Punkt im Netzwerk dar, auf den alle Clients zugreifen.

Allerdings würden wir im Falle dieser Studienarbeit von der allgemeinen Definition abweichen. Bei einer Realisierung des Systems als Fat Client würde komplett auf den Server verzichtet und die Datenbank zusammen mit der eigentlichen Applikation auf dem Zielsystem installiert werden. Insofern wäre es eigentlich richtig, von einer Monolith-Architektur zu sprechen. Darauf bezieht sich auch die nachfolgende Zusammenstellung der Vor- und Nachteile dieser Architektur. Die Vorteile eines Fat Client liegen in dieser Arbeit vor allem in der bedingungslosen Verfügbarkeit. Die Datenbank wäre auch ohne Internetverbindung abrufbar. Ein weiterer Vorteil ist die Zugriffszeit. Anwendung und Datenbank liegen auf demselben System, und die Zugriffszeiten sind deshalb optimal.

Der Nachteil dieser Lösung ist die Wartbarkeit. Bei Installation der Anwendung muss die Datenbank auf das Zielsystem übertragen werden. Bei sehr großen Datenbanken kann der

Speicherbedarf dabei enorm anwachsen. Wenn sehr viele dieser Anwendungen in Umlauf gekommen sind, müsste eine Möglichkeit gefunden werden, alle diese Datenbanken synchron upzudaten, nachdem neue Objekte in das System aufgenommen worden sind.

3.1.2 Client-Server-System

Im Gegensatz zum Fat Client steht die Client-Server-Architektur. Diese „besteht aus zwei logischen Teilen:

- *einem oder mehreren Clients, der die Services oder Daten des Servers in Anspruch nimmt und somit anfordert*
- *einem Server, der Services oder Daten zur Verfügung stellt*

Zusammen bilden beide ein komplettes System mit unterschiedlichen Bereichen der Zuständigkeit, wobei diese Zuständigkeiten oder Rollen fest zugeordnet sind, entweder ist ein Prozess ein Client oder ein Server.“ ([SYSTEME 02], S. 29)

Die Abbildung illustriert den Aufbau eines Client-Server-Systems. Es ist zu erkennen, dass die Clients nur über Hardware zur Ein- und Ausgabe verfügen. Die Applikationen, Speicher und Rechenleistung werden vom zentralen Server zur Verfügung gestellt.

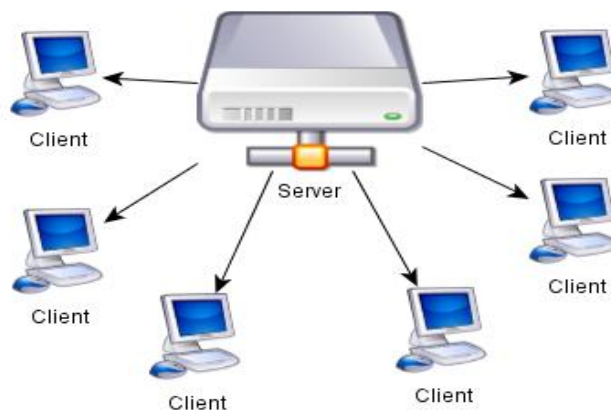


Abbildung 3-2: Aufbau einer Client-Server-Architektur

Quelle: [CLIENT-SERVER]

Als Beispiel für Client-Server-Anwendungen kann man jede Aktion betrachten, bei der zwei Rechner über das Internet kommunizieren. Dabei gibt es immer einen Host, der Dienste bereitstellt und auf die Anfragen reagiert (Server), und einen Host, der Anfragen stellt und die Antworten verarbeitet (Client). Konkrete Beispiele für Dienste, die von Servern bereitgestellt werden, sind bspw. die Bereitstellung von Datenbankzugriffen oder die Übermittlung von E-Mails.

Die Realisierung des Systems als eine Client-Server-Anwendung bietet den Vorteil, dass alle Benutzer des Systems immer auf die gleiche Datenbasis zugreifen. Die Informationen zu den Objekten liegen zentral auf einem Webserver. Die Clients schicken ihre GPS-Daten an diesen Server. Dort werden die in der Nähe gelegenen Objekte in der Datenbank gesucht und an den Client zurückgeschickt. Damit wäre dem Client Rechenlast genommen (keine GPS-Verarbeitung). Außerdem würden neue Datenbankeinträge allen Teilnehmern sofort zur Verfügung stehen.

Der Nachteil ist allerdings, dass ein Ausfall des Webserver oder der Datenbank das gesamte System betreffen würde. Die Anwendung würde auf keinem einzigen Client mehr laufen. Außerdem ist für eine komplikationsfreie Funktion der Anwendung eine Internetverbindung auf den Clients notwendig.

3.1.3 Auswahl eines Systems

Vor der Auswahl für eines der beiden Systeme sei noch erwähnt, dass im Fall dieser Studienarbeit leicht von den Definitionen abgewichen wird.

Tabelle 2 fasst noch einmal die wichtigsten Eigenschaften der beiden beschriebenen Architekturen zusammen.

	Client-Server-Architektur	Fat Client
Verfügbarkeit	Abhängig von Verfügbarkeit des Webservers	Abhängig von Verfügbarkeit des Fat Client
Zugriffszeit	Abhängig von Internetverbindung und Datenbank	Optimal, da Datenbank und Anwendung auf der gleichen physischen Instanz
Datenbasis	Zentrale Datenbasis für alle Clients	Muss auf jedem System mitgeliefert werden
Einspielen und Verteilen von Änderungen	Sehr einfach, nur an einer Stelle notwendig	Schwierig, alle Systeme müssen aktualisiert werden
Rechenlast	Rechenlast verteilt sich auf Client und Server	Physische Instanz muss die komplette Rechenlast stemmen

Tabelle 2: Vergleich von Client-Server-Architektur und Fat Client

Nach der Auswertung aller Vor- und Nachteile der beiden Möglichkeiten fällt die Entscheidung für eine Realisierung des Systems als eine Client-Server-Anwendung. Vor allem die schwierige Verwaltung der Datenbank und deren Aktualisierung bei der Fat-Client-Lösung geben den Ausschlag für die Entscheidung. Außerdem stellen die Nachteile der Client-Server-Architektur, also die notwendige Internetverbindung und die Ausfallproblematik des Webservers, heute keine großen Schwierigkeiten mehr dar. In den Ballungsgebieten ist der Ausbau der Mobilfunknetze so

weit fortgeschritten, dass stets eine Internetverbindung zur Verfügung steht. Für Webserver garantieren die Anbieter heutzutage eine Verfügbarkeit von 99,9 %.

Es sei erwähnt, dass der Prototyp eine Mischform der beiden Systeme sein wird. Aus den obigen Definitionen wird ersichtlich, dass bei der reinen Client-Server-Architektur lediglich die Darstellungs- und eine Kommunikationsschicht auf dem Client liegt. Im vorliegenden Fall wird allerdings auch ein Teil der Applikationslogik auf dem Client implementiert. Trotzdem wird im weiteren Verlauf nur noch von einem Client-Server-System ausgegangen. Die Lösung des Fat Client findet im Folgenden keine Beachtung mehr.

3.2 Anwendungsfälle

3.2.1 Display

Ein Anwendungsfall dieses Systems auf einem einzelnen Display wurde bereits im Kapitel 1.2 beschrieben. Das Display würde dann beispielsweise in einem Stadtbus oder einer S-Bahn hängen. In diesem Fall ist keine Interaktion mit dem System möglich. Es werden lediglich nacheinander Informationen zu den verschiedenen Objekten angezeigt.

Da die Informationen von einem Webserver geladen werden, brauchen die Displays eine dauerhafte Internetverbindung. Desweiteren brauchen die Displays einen GPS-Empfänger oder müssen zumindest an einen solchen angeschlossen sein. Dem Aufruf des Webservers werden dann die aktuellen GPS-Daten mitgegeben. Auf dem Server werden daraus nahe liegende Objekte berechnet. Die dazugehörigen Abläufe werden später beschrieben. Die komplette Darstellung wird vom Webserver übernommen. Der Client ruft den Webserver in einem voreinstellbaren Intervall immer wieder auf und überträgt die aktuelle GPS-Position. Somit wird garantiert, dass immer Informationen zu Objekten angezeigt werden, die auch wirklich in der Nähe liegen. In einer Weiterentwicklung wäre es wünschenswert, auch noch eine Information über die Richtung zu erhalten, in die sich das Display (bzw. das Fahrzeug, in dem es installiert ist)

bewegt. Dadurch wäre es möglich, Informationen zu Objekten schon anzuzeigen, noch bevor sie am Streckenrand erscheinen. Dabei gilt es auch zu prüfen, inwieweit auch die Geschwindigkeit des Client in die Berechnung einfließen kann.

Die vorliegende Studienarbeit konzentriert sich allerdings in erster Linie auf die Entwicklung einer Anwendung für mobile Endgeräte. Die Anwendung im Display findet damit vorerst keine weitere Betrachtung. Es besteht allerdings die Möglichkeit, dieses Thema im nächsten Semester noch einmal aufzugreifen.

3.2.2 Mobiles Endgerät

Im Gegensatz zum Display erlaubt das mobile Endgerät eine Benutzer-Interaktion, wodurch der Anwendung grundsätzlich andere Anforderungen gestellt werden.

Ein Benutzer möchte keine statische Applikation, die ihm eine Standard-Ausgabe generiert, sondern vielmehr eine Anwendung die unterschiedliche Einstellungsmöglichkeiten hat, um sie an die eigenen Bedürfnisse anpassen zu können. Anhand dieser Vorgaben wurden verschiedene Anforderungen ermittelt, die man als Benutzer an eine Location-Awareness-Anwendung stellt. In folgendem Use-Case-Diagramm sind die einzelnen Anwendungsfälle dargestellt.

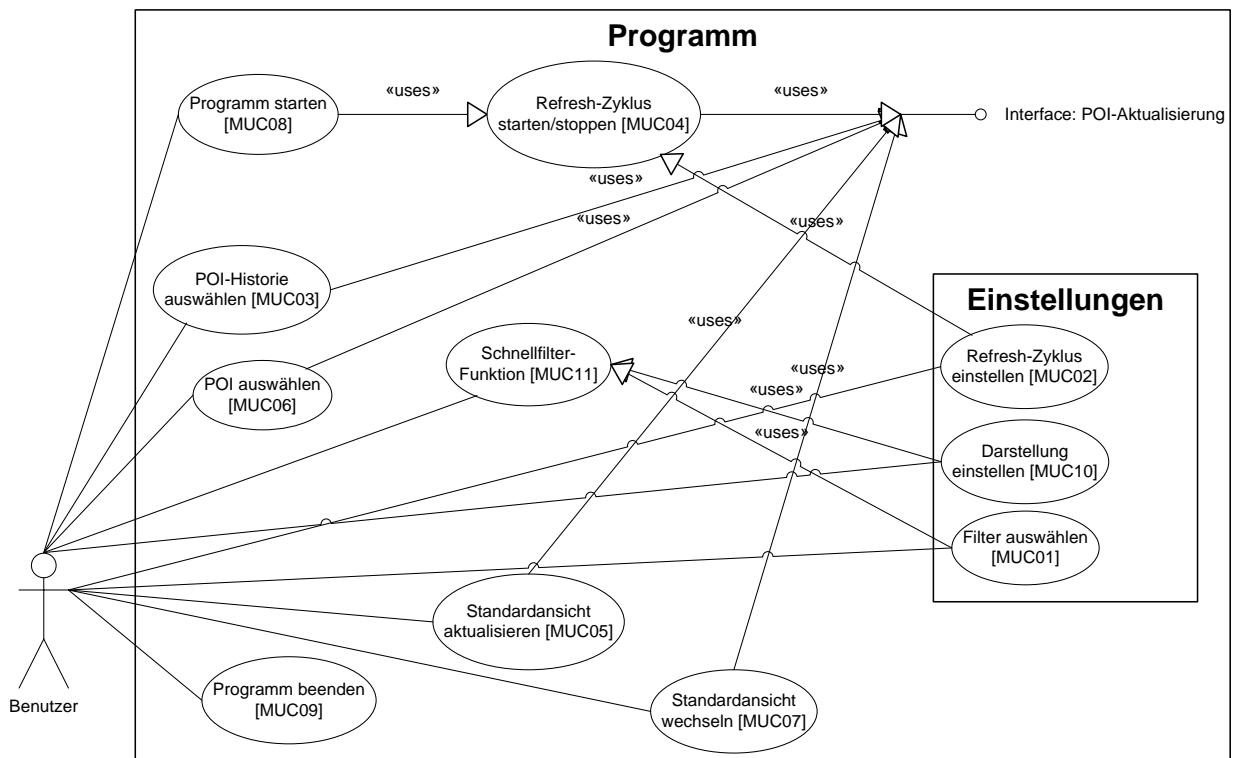


Abbildung 3-3: Use-Case-Diagramm des mobilen Endgeräts

Der Benutzer soll das Programm über verschiedene Einstellungen individualisieren können. Außerdem hat der Benutzer Interaktionsmöglichkeiten, die es ihm erlauben, detailliertere Informationen zu einem POI anzufordern.

Der Grundgedanke der Anwendung besteht darin, dass der Benutzer auf einer Standardansicht die POIs angezeigt bekommt, die sich in seiner unmittelbaren Nähe befinden. Will er nähere Informationen zu einem POI erhalten, so kann er diese durch Anklicken auswählen, wobei sich dann eine Detailansicht öffnet (Use Case MUC06). Von dieser Maske kann er wieder in die Standardansicht wechseln (Use Case MUC07). Der Use Case „POI-Historie auswählen“ (Use Case MUC03) beschreibt die Möglichkeit, die zuletzt aufgerufenen POIs in einer Liste abzuspeichern. Durch abermaliges Auswählen wird wiederum in die Detailansicht des POI gewechselt.

Des Weiteren besitzt die Anwendung für das mobile Endgerät die Möglichkeit, die angezeigten POIs zu filtern. Das bedeutet, es werden nicht mehr alle in der Nähe des Benutzers befindlichen POIs angezeigt, sondern nur solche, die den aktuell ausgewählten Filterkriterien entsprechen. Ein Beispiel für eine Filter-Kategorie könnte beispielsweise „Restaurant“ sein.

Um die Filterkriterien definieren zu können, gibt es zwei unterschiedliche Vorgehensweisen. Einerseits kann man im Dialog Einstellungen einen individuellen Filter festlegen, der über die Programm-Laufzeit hinweg auf dem System abgespeichert und bei jedem Start des Programms neu angezogen wird (MUC01). Zum Anderen besteht die Möglichkeit, im Anwendungsfenster über eine Schnellfilter-Leiste den Filter anzupassen (MUC11). Diese Einstellungen haben jedoch nur zur Laufzeit Gültigkeit. Außerdem werden sie durch das Verändern des Filters im Einstellungsdialog implizit angepasst.

Im Einstellungsdialog kann man ebenfalls festlegen, ob man die Schnellfilter-Liste dargestellt sehen möchte oder nicht (MUC10).

Zu guter Letzt besteht die Möglichkeit, im Einstellungsdialog die Länge des Refresh-Zyklus festzulegen. Der Refresh-Zyklus bestimmt die Zeit zwischen der automatischen Aktualisierung der POIs in der Standardansicht. Dabei kann die automatische Aktualisierung auch abgeschaltet werden, wodurch implizit der Use Case „Refresh-Zyklus starten/stoppen“ (MUC04) aufgerufen wird. Zusätzlich zur automatischen Aktualisierung kann die Standardansicht über einen Button manuell aktualisiert werden (MUC05).

Eine genaue Beschreibung der einzelnen Use Cases befindet sich im Anhang (siehe ...).

3.2.3 Administration

Die Objekte, zu denen sich Anwender Informationen ansehen können, sind in einer Datenbank hinterlegt. Diese Datenbank liegt dann im Normalfall mit auf dem Webserver, auf dem auch die POI-Berechnung durchgeführt ist. Da zu diesem Server allerdings nur der Administrator einen Zugang hat, gilt es auch anderen Anwendern eine Möglichkeit zu schaffen, um neue Objekte in die Datenbank einzutragen.

Als Beispiel soll hier ein Restaurantbetreiber in der Stuttgarter Innenstadt dienen. Der Gastronom erfährt von der neuen Werbeplattform „Location Awareness“ und entscheidet sich, auch für sein Lokal in dieser Form Werbung betreiben zu wollen. Er ruft die entsprechende Internetseite auf und kann sich registrieren. Er hat nun die Möglichkeit, ein neues Objekt (sein Restaurant) in die Datenbank einzutragen. Dazu muss er dem Objekt einen Namen geben und die exakten Positionsdaten in Form von Längen- und Breitengrad eingeben. Weiterhin kann er sein Objekt in eine Kategorie einordnen. Je nach Wahl kann er dann für diese Kategorie typische Informationen in das System eintragen. Mit einem Klick auf "Speichern" wird das Objekt in die Datenbank übernommen. Wünschenswert wäre eine noch einfachere Eingabe der Position über eine Karte. Damit würde ein einfacher Klick auf die Stelle, an der sich das Objekt befindet genügen. Diese Möglichkeit wird im nächsten Teil der Studienarbeit auf ihre Realisierbarkeit geprüft.

Nach einiger Zeit zieht das Restaurant um. Der Besitzer kann sich jetzt unter seinem Namen in das System einloggen und die Informationen zu den von ihm hochgeladenen Objekten bearbeiten. Ein Klick auf "Speichern" genügt, und das Restaurant ist jetzt unter der neuen Adresse im System eingetragen.

Der Administrator des Location-Awareness-Systems kann sich ebenfalls über das Admin-Kennwort an der Plattform anmelden. Er hat die Möglichkeit, alle Objekte, die in der Datenbank stehen, einzusehen, zu validieren oder zu verändern, neue Objekte hinzuzufügen oder Objekte aus dem System zu löschen.

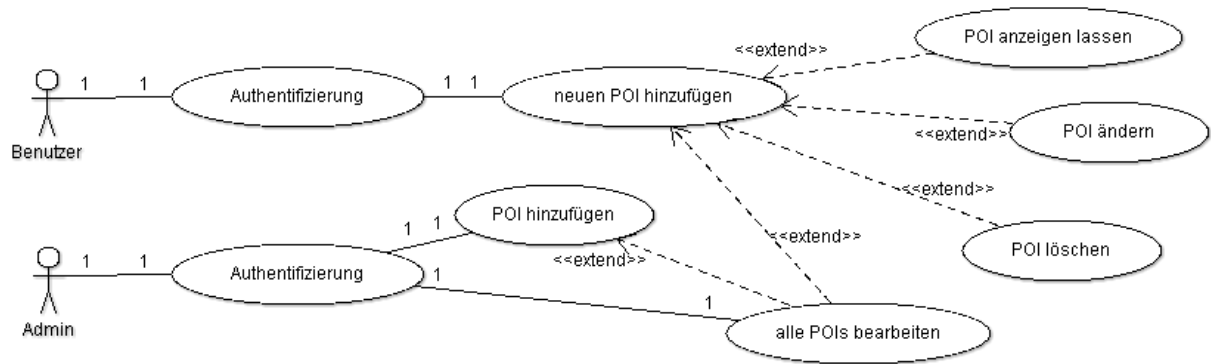


Abbildung 3-4: Use-Case-Diagramm der Administrationsplattform

4 Konzept

4.1 Systemarchitektur des Prototypen

Bei einer Location-Awareness-Anwendung geht es darum, Umgebungsinformationen anhand der von einem GPS-Empfänger ermittelten Positionsdaten auszuwählen. Dabei sollen solche POIs auf dem Client erscheinen, die sich in geringer Entfernung vom Benutzer befinden.

Auf Grund der leichteren Wartbarkeit der Umgebungsinformationen und der höheren Praktikabilität basiert der Prototyp der Location-Awareness-Anwendung auf einer Server-Client-Architektur. Außerdem hat man sich dazu entschieden, den Client für ein mobiles Endgerät zu entwickeln, da sich dem System durch Benutzer-Interaktion viel mehr Möglichkeiten bieten. Zudem hat eine Anwendung für ein mobiles Endgerät in Zeiten des Smartphone-Booms sicherlich eine höhere Marktrelevanz als eine Display-Applikation.

Das Gesamtsystem hat die Aufgabe, die Positionsdaten des Client mit den Umgebungsinformationen, die auf dem Server hinterlegt sind, in Zusammenhang zu bringen.

Der Client ist der mobile Teil der Anwendung, dessen Position über die dargestellten Umgebungsinformationen entscheidet. Dazu ermittelt die Applikation die aktuellen GPS-Daten über den Empfänger des mobilen Endgeräts und sendet diese an den Location Awareness Server. In dessen Datenbank sind die Umgebungsinformationen abgelegt, die anhand der Positionsdaten ausgewählt werden, wobei nur POIs in unmittelbarer Nähe der ermittelten Position von Interesse sind. Diese werden dann zurück an den Client gesendet und dem Benutzer über die grafische Oberfläche dargestellt.

Die folgende Abbildung zeigt die Architektur des Gesamtsystems.

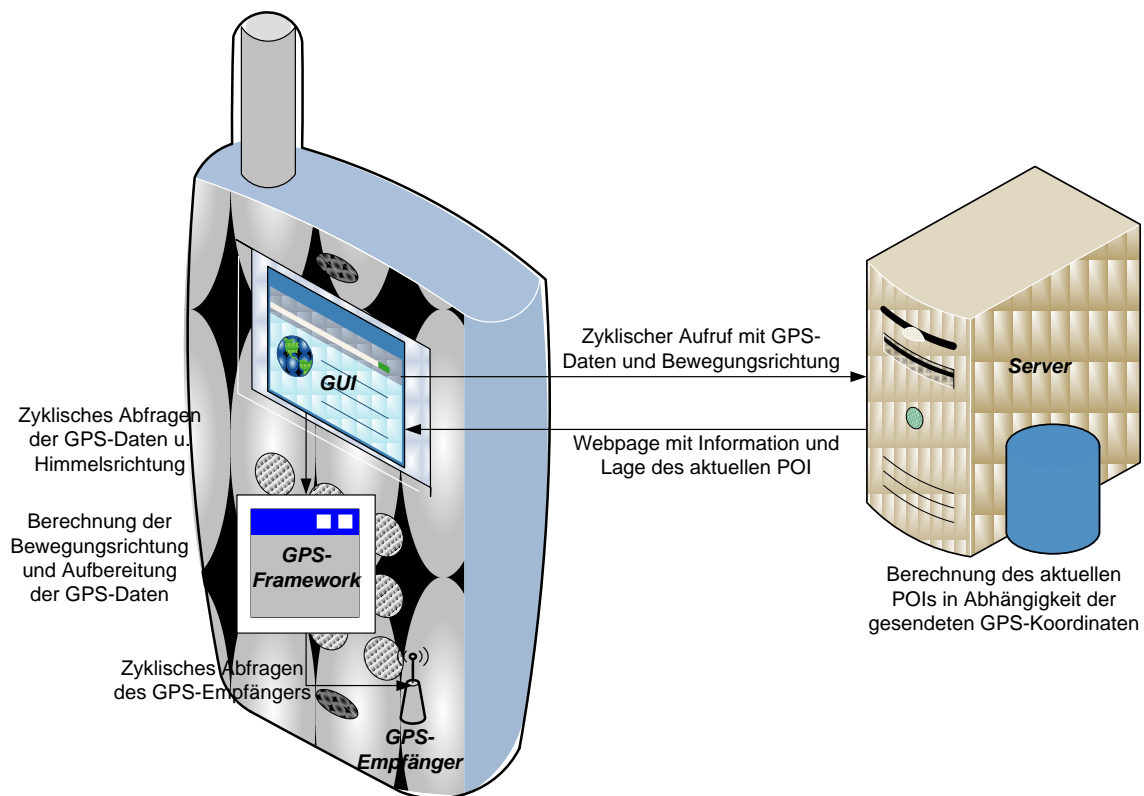


Abbildung 4-1: Systemarchitektur des Prototypen

Wie in der Abbildung zu sehen ist, ist die Location-Awareness-Anwendung in einzelne Module gegliedert, welche die unterschiedlichen Systeme voneinander abgrenzen.

Eine Komponente des Systems ist der Server. Er ist ein klassischer Webserver, der auf dem Bundle Apache beruht und den man über einen http-Aufruf ansprechen kann. Er beantwortet den http-Request durch das Senden einer Meta-Daten-Datei, die auf Client-Seite interpretiert werden muss. In diesem Fall bedeutet es, dass der Client dem Server über eine http-Anforderung seine GPS-Daten und Bewegungsrichtung übermitteln muss. Der Server berechnet dann anhand der Werte die aktuellen POIs und sendet sie an den Client zurück.

Um dem Benutzer die erhaltenen Umgebungsinformationen anzuzeigen, hat der Client eine grafische Oberfläche, in die ein Webbrowser integriert ist. Dieser ist nicht nur für die

Interpretation und die Darstellung der Daten zuständig, sondern sendet auch den http-Request an den Server.

Damit der Webbrowser die aktuellen Positionsdaten an den Server senden kann, muss die Position des Client aus dem GPS-Empfänger des Geräts ermittelt und so aufbereitet werden, dass der Server die Daten interpretieren kann. Dazu greift die GUI auf das GPS-Framework zu. Dieses liest den GPS-Treiber über eine standardisierte Schnittstelle aus und berechnet aus den letzten ermittelten Werten eine Bewegungsrichtung, um eine bessere Eingrenzung des zur Ermittlung der POIs relevanten Bereichs vornehmen zu können. Anschließend werden die Werte formatiert, so dass der Webserver sie interpretieren kann.

Durch die klare Abgrenzung der einzelnen Systemkomponenten voneinander hat die Anwendung eine hohe Modularität. Ohne weiteres können der Webserver oder der GPS-Treiber ausgetauscht werden. Als kritisch ist der Aufbau der http-Verbindung durch die Benutzer-Schnittstelle zu betrachten. Hier würde eine Veränderung der Verbindungstechnologie zwangsläufig zu Anpassungen der GUI führen.

4.2 Software-Design des Client

4.2.1 Gesamter Client

Wie in der Darstellung der Systemarchitektur zu sehen ist (siehe Abbildung 4-1), besteht der Client aus drei Modulen, nämlich erstens der GUI, die das Hauptprogramm der Anwendung ist, zweitens dem GPS-Framework, das für die Ermittlung und Formatierung der Positionsdaten sowie für die Berechnung der Bewegungsrichtung zuständig ist, und drittens dem GPS-Treiber, der die Hardware ausliest.

Das Ziel der Architektur ist die Modularität des Systems. Die einzelnen Komponenten sollen austauschbar sein, ohne dass die Software des gesamten Client angepasst werden muss.

Dazu sind klare Schnittstellen zwischen den einzelnen Modulen definiert. Die folgende Abbildung stellt das Klassendiagramm eines Prototyps dar, welches die Abgrenzung der

Komponenten untereinander zeigt. Dabei handelt es sich um eine mögliche Referenzimplementierung.

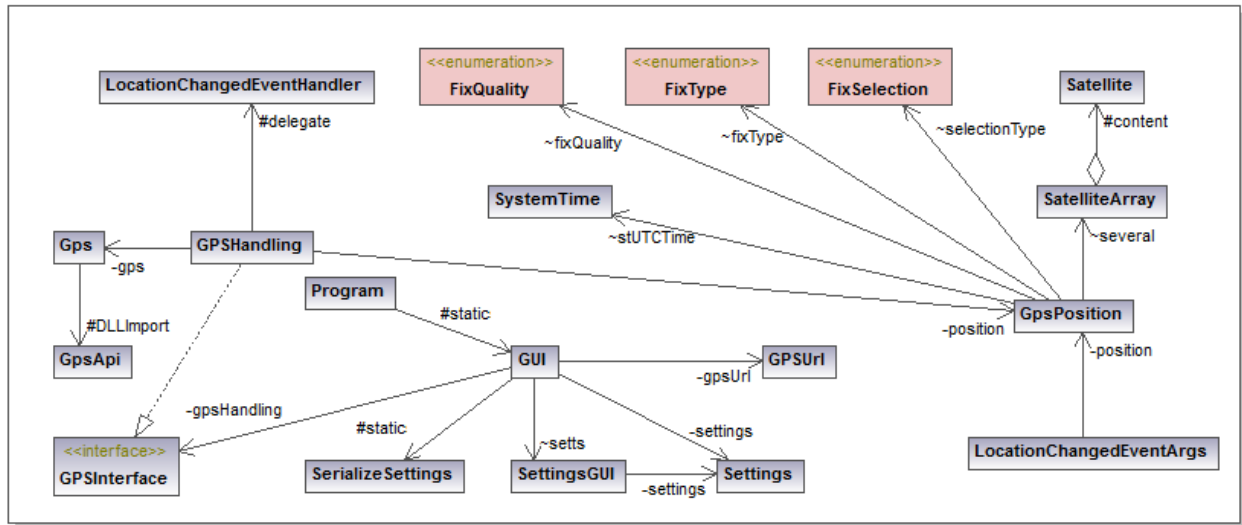


Abbildung 4-2: Klassendiagramm Client

Die GUI greift über die Schnittstelle GPSInterface auf die Werte der GPS-Daten- Verwaltung (Klasse GPSHandling) zu. Das Auslesen des GPS-Treibers läuft über das Einbinden der GpsApi, einer von Windows Mobile angebotenen DLL, welche unabhängig von der zugrundeliegenden Hardware arbeitet.

In den zwei folgenden Unterkapiteln werden die beiden Hauptaufgaben des Location Awareness Client spezifiziert, nämlich zum einen die GPS-Daten-Ermittlung und -Verarbeitung sowie zum anderen die Darstellung der Umgebungsinformationen und die Verwaltung der Anwendung (die Dokumentation des Client befindet sich im

Anhang D - Anlagen).

4.2.2 GPS-Daten-Verwaltung

Wie schon erwähnt, bietet Windows Mobile eine hardwareunabhängige Schnittstelle zum GPS-Empfänger in Form einer DLL-Datei (gpsapi.dll). Sie beinhaltet vier Methoden, um den GPS-Receiver zu verwalten. Diese erledigen folgende Aufgaben:

- Positionsdaten-Ermittlung starten
- Positionsdaten-Ermittlung beenden
- GPS-Position ermitteln
- GPS-Gerätestatus ermitteln

Die Methoden übergeben oder übernehmen jeweils Pointer. Über sie hat man Zugriff auf die Daten-Struktur des GPS-Empfängers.

Um aus der Struktur die tatsächlichen Werte der aktuellen Position als Breitengrad und Längengrad zu erhalten, wird in der Anwendung ein Microsoft GPS Framework verwendet. Dieses verschiebt die ermittelten Daten aus der Struktur des Empfängers in die Anwendung. Die folgende Abbildung zeigt das Gefüge und Zusammenspiel der einzelnen Klassen der GPS-Daten-Verarbeitung:

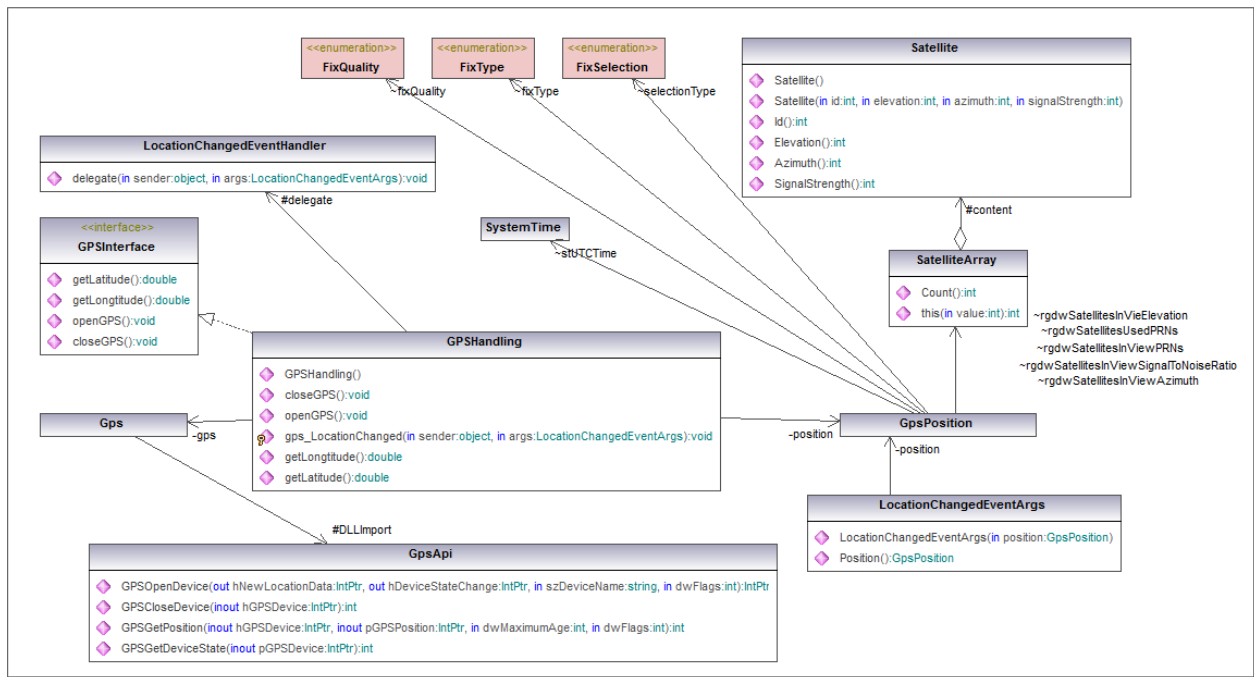


Abbildung 4-3: Klassendiagramm der GPS-Datenverarbeitung

Man sieht die Klasse Gps, welche die gpsapi.dll importiert. Sie verwaltet den GPS-Empfänger über einen Pointer und verschiebt dessen Daten in eine Struktur der Klasse GpsPosition. Eine Instanz der Klasse GpsPosition enthält die Informationen über den Längengrad und Breitengrad der aktuellen Position. Außerdem hat sie einen Array, der alle für den Empfänger sichtbaren Satelliten beinhaltet, deren genaue Eigenschaften wiederum in der Klasse Satellit spezifiziert werden.

Des Weiteren enthält die Instanz verschiedene Enum-Typen. Diese geben Auskunft über die Qualität des Signals (FixQuality), den Auswahl-Typ (2D- oder 3D-Position, FixType) und die Art der Auswahl (automatische oder manuelle Auswahl zwischen 2D- oder 3D-Signal, FixSelection).

Im Prototyp stellt die Klasse GPSHandling stellt das Framework dem Hauptprogramm zur Verfügung. Dazu wird die Event-Methode gps_LocationChanged einer Instanz der Klasse Gps

implementiert, welche ausgelöst wird, sobald sich die Empfänger-Position verändert. Innerhalb dieser Methode wird die GPS-Position (Instanz der Klasse GpsPosition) ermittelt.

Das Hauptprogramm der Referenzimplementierung greift über die Schnittstelle GPSInterface auf die Methoden der Klasse GPSHandling zu. Es erhält über die Methoden getLatitude und getLongitude die ermittelten Werte von Breiten- und Längengrad der exakten momentanen Position als Gleitkommawert.

Des Weiteren kann das Hauptprogramm die Positionsbestimmung über die Methoden openGPS und closeGPS starten und stoppen.

4.2.3 Hauptprogramm

Der Einstiegspunkt des Location-Awareness-Prototyps ist die Main-Methode der statischen Klasse Program. Hier wird eine Instanz der Klasse GUI initialisiert und der Methode Application.Run übergeben. Sie startet die grafische Oberfläche und stellt das Ausführen der Anwendung sicher, bis der Benutzer diese über eine Eingabe beendet. Das Objekt der Klasse GUI repräsentiert den Hauptdialog und verwaltet dadurch die Anwendung.

Die folgende Abbildung zeigt das detaillierte Klassendiagramm der Referenzimplementierung des Hauptprogramms.

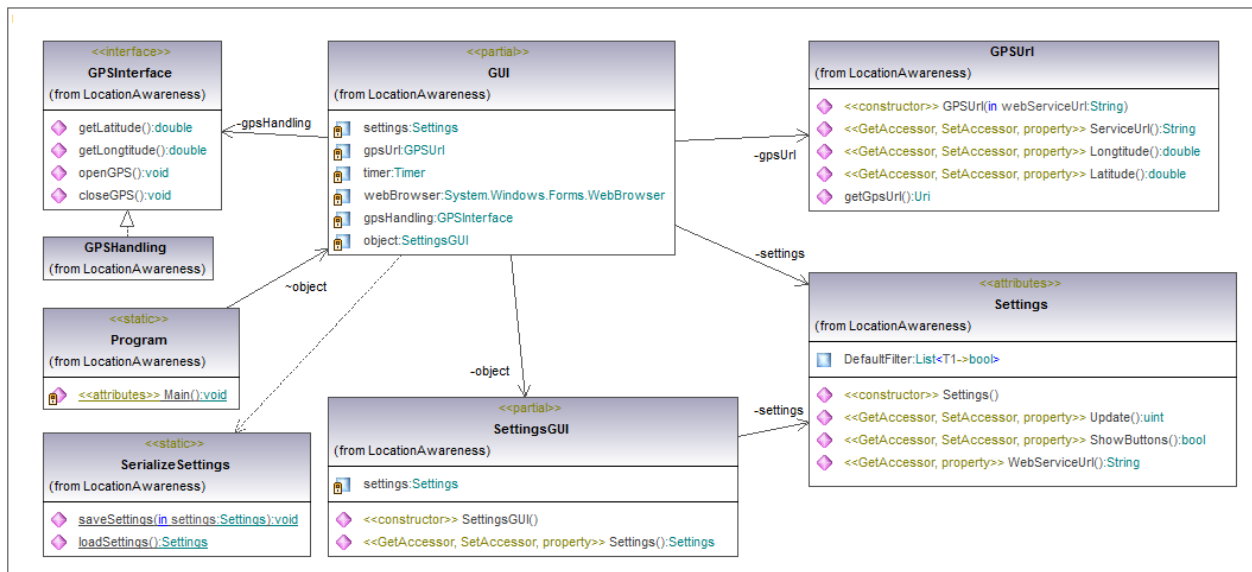


Abbildung 4-4: Detailliertes Klassendiagramm des Hauptprogramms

Wie man im Diagramm sieht, ist die GUI das zentrale Element. Sie delegiert die verschiedenen Ereignisse an die Instanzen der entsprechenden Klassen.

Um vom Server die aktuellen Umgebungsinformationen zu erhalten, greift sie über das GPSInterface auf die GPS-Daten-Verwaltung zu und ermittelt Längen- und Breitengrad. Diese werden einer Instanz der Klasse GPSUrl übergeben, welche im Konstruktor eine eindeutige Url erwartet, die den Location Awareness Server identifiziert und den http-Aufruf beinhaltet. Das Objekt fügt nun die Positionsdaten in diesen String ein und gibt ihn als Instanz der Klasse Uri zurück.

In der GUI wird das Uri-Objekt dem Webbrowser zur Verfügung gestellt, dieser übergibt sie einem http-Request und erhält als Antwort die Umgebungsinformationen.

Des Weiteren kann der Benutzer die Anwendung über einen separaten modalen Dialog SettingsGUI an seine eigenen Bedürfnisse anpassen. Dazu wird eine Instanz des Dialogs in GUI initialisiert und bekommt das Objekt der Klasse Settings übergeben, welches die aktuellen Einstellungen der Anwendung repräsentiert. Der Dialog zeigt dem Benutzer die aktuellen Einstellungen an und bietet ihm die Möglichkeit, diese zu verändern. Nach dem Schließen der Maske werden die getroffenen Einstellungen durch das Hauptprogramm übernommen.

Die Einstellungen, die der Benutzer treffen kann, sind durch die Eigenschaften der Klasse Settings festgelegt. Er kann über `WebServiceUrl` den Aufruf des referenzierten Webservice verändern, über die Eigenschaft `Update` den Zyklus festlegen, wie oft die Anwendung neue POIs anfordert, und über `ShowButtons` die Darstellung des Dialogs GUI anpassen.

Des Weiteren kann über die Liste `DefaultFilter` der Standard-Filter der Anwendung festgelegt werden. Über die Filter-Einstellungen kann der Benutzer, wie schon erwähnt, die Art von POIs (z. B. Restaurants) festlegen, welche vom Server angefordert werden. Der Filter wird über einen separaten `http-Request` an den Server übermittelt.

Um die Einstellungen auch über die Laufzeit des Programms hinweg festlegen zu können, gibt es die statische Klasse `SerializeSettings`, die Objekte der Klasse `Settings` auf dem statischen Speicher des Client hinterlegt. Dazu bietet sie zwei Methoden an:

Zum einen existiert die Funktion `saveSettings`, die von GUI vor Beenden der Anwendung aufgerufen wird und die aktuellen Einstellungen serialisiert, die von einer Instanz der Klasse `Settings` repräsentiert werden. Zum anderen besteht die Funktion `loadSettings`, die beim Laden der Anwendung aufgerufen wird und die die auf dem ROM-Speicher hinterlegten Einstellungen des letzten Programmaufrufs deserialisiert.

4.3 Serverabläufe

4.3.1 Allgemein

Wie bereits weiter oben beschrieben, ruft der Client den Webserver über eine URL auf. Dieser Aufruf enthält die exakte Position des Client in Form von Längen- und Breitengrad. Die Aufgabe des Servers ist es nun, die Positionsdaten weiterzuverarbeiten. Das heißt im Detail, dass zunächst einmal in der Datenbank nachzuschlagen ist, ob sich POIs in der Nähe des Client befinden. Wenn das der Fall ist, müssen die Objekte nach ihrer Nähe zum Client geordnet werden. Danach wird diese sortierte Liste in Form einer HTML-Datei zur Ausgabe gebracht. Entscheidet sich der Client, für eines der Objekte genauere Informationen sehen zu wollen (durch Anklicken dieses

Objekts), so stellt der Server diese Informationen in Form einer neuen HTML-Seite dar. Möglichkeiten, diese Informationen auf dem Server zu hinterlegen, werden in Kapitel 4.3.3 genauer erläutert.

Diese Vorgänge wiederholen sich dann immer wieder, bis der Client keine Positionsdaten mehr an den Server schickt. Im nachfolgenden Sequenzdiagramm ist ein solcher Ablauf dargestellt.

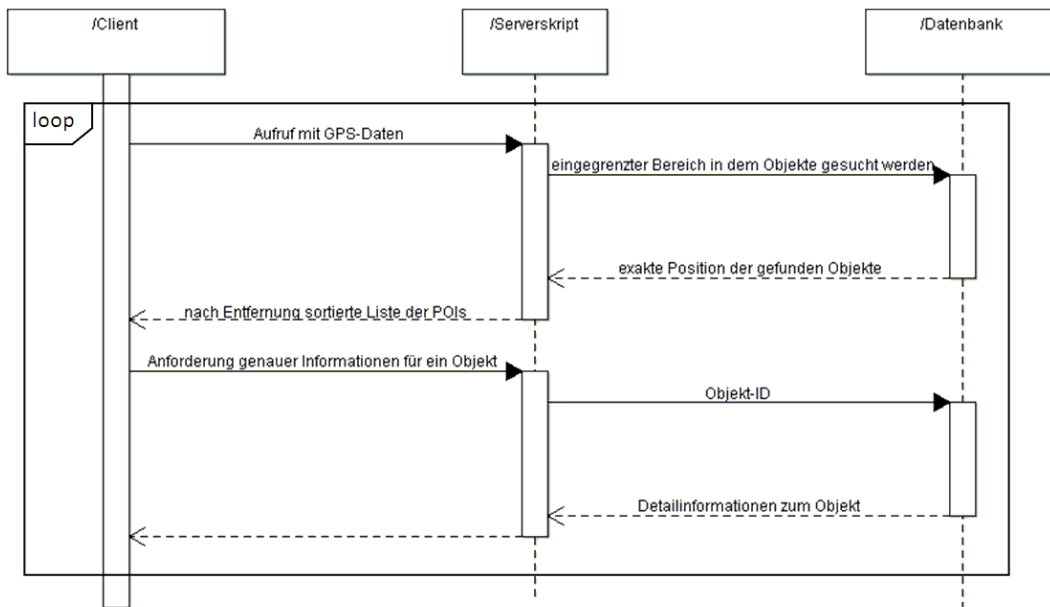


Abbildung 4-5: Sequenzdiagramm einer kompletten Serveranfrage

4.3.2 POI-Behandlung

Wie oben beschrieben, bekommt der Webserver beim Aufruf des Client dessen exakte GPS-Koordinaten mitgeliefert. Die bestehen aus dem Breitengrad (Latitude) und dem Längengrad (Longitude). Das auf dem Webserver aufgerufene PHP-Skript muss nun aus diesen Daten einen Umkreis berechnen, in dem nach Points Of Interest in der Datenbank gesucht werden soll. Dazu wird auf die beiden Positionsdaten des Client einfach ein fester Wert (Radius) hinzuaddiert bzw. subtrahiert:

$$\begin{array}{lll} \text{Längengrad:} & \text{untere Grenze} & = \text{Client-Längengrad} - \text{Radius} \\ & \text{obere Grenze} & = \text{Client-Längengrad} + \text{Radius} \\ \text{Breitengrad:} & \text{untere Grenze} & = \text{Client-Breitengrad} - \text{Radius} \\ & \text{obere Grenze} & = \text{Client-Breitengrad} + \text{Radius} \end{array}$$

Mit diesen Grenzen wird dann auf die Datenbank zugegriffen. Es gilt alle Objekte zu selektieren, deren Position sich innerhalb dieser Grenzen befindet. Werden keine passenden Objekte gefunden, wird dies dem Client mitgeteilt und auf den nächsten Aufruf mit neuen Client-Positionsdaten gewartet. Sollte entsprechende Objekte in der Datenbank vorliegen, werden deren Daten weiterverarbeitet. Vor der Ausgabe müssen die gefundenen Objekte noch nach ihrer Entfernung zum Client geordnet werden. Hierfür gilt es einen geeigneten Algorithmus zu entwerfen. Dieser wird im ersten Teil der Studienarbeit wohl noch recht einfach konzipiert und sollte dann in der weiteren Bearbeitung verbessert werden.

Die Anforderungen an den Sortieralgorithmus lauten:

- Berechnung der Entfernung zum Client
- exakte Sortierung der Objekte nach Entfernung zum Client
- möglichst geringer Rechenaufwand
- zuverlässige Sortierung auch bei nahe beieinander liegenden Objekten

Für diesen ersten Teil der Arbeit wäre ein Algorithmus ausreichend, der in den meisten Fällen eine richtige Sortierung vornimmt. Diese ist momentan noch nicht ausschlaggebend, da es vorerst vor allem um die Konnektivität zwischen Client und Server sowie zwischen Server und Datenbank geht. Nach erfolgreicher Sortierung der Objekte wird eine HTML-Seite erstellt, welche dann auf dem Client zur Ausgabe kommt.

Der Benutzer hat jetzt die Möglichkeit, sich genauere Informationen zu den Objekten anzeigen zu lassen. Durch einen Klick auf das gewünschte Objekt wird eine erneute Anfrage an den Webserver abgeschickt. Diese enthält die ID des Objekts, für das Informationen angefordert werden. Der Webserver erzeugt aus der Objekt-ID eine neue Anfrage an die Datenbank und setzt

sie ab. Das genaue Aussehen dieser Anfrage hängt von der Datenbankstruktur ab, welche weiter unten beschrieben wird. Die von der Datenbank gelieferten Informationen werden dann wieder zur Ausgabe an den Client gesendet.

4.3.3 Session-Handling

Für die vorgesehene Filterfunktion, über die der Anwender die angezeigten Objekte auf bestimmte Kategorien reduzieren kann, sollen PHP-Sessions eingesetzt werden.

„Sessions speichern temporäre Daten über Ihre Besucher und eignen sich besonders, wenn diese Daten von außerhalb Ihres Servers nicht zugänglich sein sollen. Sie sind eine Alternative zu Cookies, wenn der Benutzer Cookies auf seinem Computer deaktiviert hat, da PHP URLs automatisch umschreiben kann, um eine Session-ID für Sie zu übertragen. [...]

„Eine Session ist eine Kombination aus einer serverseitigen Datei, die alle Daten enthält, die Sie speichern wollen, und einem clientseitigen Cookie, das eine Referenz auf die Serverdaten enthält.“

([PHP 06], S. 184)

Der allgemeine Ablauf einer Session-Behandlung in PHP gliedert sich wie folgt. Über den Befehl `session_start()` wird überprüft, ob der Besucher ein Session-Cookie gesendet hat. Ist dies der Fall, lädt PHP die Session-Daten. Anderenfalls wird eine neue Session-Datei auf dem Server erzeugt und eine ID an den Besucher zurückgesendet. Dieser wird danach mit der neuen Datei assoziiert. Über die nun gültige Session können jedem User globale Variablen zugeordnet werden. Die Session-Daten werden im Session-Superglobal-Array `$_SESSION` gespeichert.

Jede Variable besetzt, zusammen mit ihrem Wert, ein Element in diesem Array. Eine neue Variable wird folgendermaßen angelegt:

```
$_SESSION['var'] = $val
```

```
Bsp.: $_SESSION['Name'] = 'Ralf'
```

Danach können die Variablen verwendet und verändert werden. Gelöscht werden Session-Daten über die Funktion `unset(<Variable>)`. Eine Session wird erst beendet, wenn der Benutzer den

Browser schließt oder im PHP-Code die beiden Funktionen `$_SESSION = array()` und `session_destroy()` aufgerufen werden (vgl. [PHP 06], S.184 f.)

Dieses Session-Handling lässt sich sehr gut in das Kategorie-Konzept des Location-Awareness-Systems übertragen. Sendet der User zum ersten Mal eine Anfrage mit seiner GPS-Position an den Webserver, so wird eine neue Session erzeugt und in dieser eine neue Variable „*kat*“ angelegt. Die variable ist zunächst noch leer, und dem User werden alle Objekte angezeigt, die sich in seiner Nähe befinden. Dabei wird keine Rücksicht auf die zugehörige Kategorie genommen. Entscheidet sich der Benutzer dann, diese Liste nach einer Kategorie zu filtern, wird die Kategorie einmalig an den Server übertragen. Dies könnte auf dem gleichen Weg wie bei der in Kapitel 4.3.2 beschriebenen POI-Behandlung über GET-Variablen realisiert werden. Mit der somit übertragenen Information wird die Session-Variable `$_SESSION['kat']` gefüllt. Von nun an werden bei der Datenbankabfrage nicht mehr alle Objekte selektiert, die sich in der Nähe des Users befinden, sondern nur noch diejenigen, die auch der entsprechenden Kategorie zugeordnet sind. Diese Einschränkung bleibt solange aktiv, bis der Benutzer einen neuen Filter auswählt bzw. die Filterfunktion deaktiviert. Erst dann wird die Session-Variable neu gesetzt und die Datenbankabfrage dementsprechend angepasst.

4.4 Datenbankkonzept

Die Datenbank wird zunächst sehr einfach strukturiert sein. Für diesen ersten Teil der Studienarbeit genügt eine Datenbank, die eine Tabelle enthält. In dieser werden die Objekte hinterlegt. Zu Testzwecken wird diese Tabelle vorerst nur wenige Datensätze enthalten. Die Datensätze benötigen Attribute zur eindeutigen Kennung des Objekts, für den Namen, für den Breitengrad und für den Längengrad. Diese Datenbankstruktur reicht vorerst absolut aus, um einige Tests durchzuführen. Die genaue Struktur der Datenbank sowie die dazu durchgeführten Tests werden im Kapitel 5 genauer beschrieben.

Für die Weiterentwicklung des Prototyps im nächsten Semester ist es sinnvoll, die Datenbankstruktur zu erweitern. Dies gilt besonders in Bezug auf die Detailinformationen, die der Server dem Client zur Verfügung stellen soll. Dafür gibt es zwei grundlegende Möglichkeiten. Zum einen wäre es möglich, in der Datenbank lediglich eine URL zu hinterlegen. Hinter dieser Adresse steht dann eine weitere Webseite, die Informationen zum entsprechenden Objekt beinhaltet. Diese Webseite wird dann vom Webserver zur Anzeige gebracht, entweder als komplett neuer Seitenaufruf oder in einem Frame. Das bedeutet aber, dass jedes Objekt, das in die Datenbank aufgenommen wird, auch zwingend eine weitere Webseite erfordert, welche die Informationen dazu anbietet. Diese Seite müsste dann aber von den Verantwortlichen des jeweiligen Objekts administriert werden, da sie außerhalb des eigentlichen Location-Awareness-Systems liegt. Eine solche Konstellation könnte allerdings Probleme mit sich bringen, falls die Seite von den jeweiligen Betreibern nicht dauerhaft gepflegt wird. In einem anderen Fall könnte die Seite von den Verantwortlichen komplett von Netz genommen werden. Dann würde die URL in der Datenbank ins Leere zeigen, was zu Fehlfunktionen in der Anwendung führen könnte. Außerdem würde es diese Methode mit sich bringen, dass die Detailinformationen der Objekte wahrscheinlich sehr viele unterschiedliche Designs hätten. In einer geschlossenen Anwendung, wie es hier der Fall ist, wünscht man sich aber eher ein durchgehendes Design, das eine schnelle Orientierung auf den verschiedenen Oberflächen ermöglicht. Dieser Aspekt wäre dann nicht gewährleistet.

Zum anderen besteht die Möglichkeit, die Informationen mit in der Datenbank zu hinterlegen. Als Weiterentwicklung des Prototyps ist es angedacht, die Objekte nach bestimmten Filtern kategorisieren zu lassen. Dazu könnten die Kategorien mit in der Datenbank hinterlegt werden. Weiterhin wäre es dann auch möglich, nicht bloß die Information, zu welcher Kategorie ein Objekt gehört, in die Datenbank zu schreiben, sondern für jede Kategorie eine eigene Tabelle anzulegen. Dies wäre durchaus sinnvoll, da Objekte, die in dieselbe Kategorie eingeordnet sind, im Normalfall auch in etwa dieselben Informationen anzeigen. So wird z. B. ein Restaurant in der Regel die allgemeinen Öffnungszeiten, besondere Angebote und die Kontaktdaten in der Detailinformation zeigen wollen. Die Detailansicht für örtliche Sehenswürdigkeiten wird

normalerweise immer die Eintrittspreise, Öffnungszeiten und die Anschrift vermitteln. So könnten für alle Kategorien Standardoberflächen entwickelt werden, deren Inhalte dann einheitlich in der Datenbank hinterlegt werden können. Zusätzlich wäre es möglich, Stylesheets für die Kategorien anzulegen, um die Ausgabe möglichst gut an die Struktur der Detailinformationen anzupassen. Dem Anwender würden damit immer ähnliche Oberflächen geboten, Anordnung und Bedienung über alle Ansichten und Kategorien hinweg gleich bleiben. Das folgende ER-Diagramm zeigt eine Tabellenstruktur mit den Kategorien Gastronomie und Attraktion, auf welche die Objekte verweisen können. Auf diese Art und Weise ist denkbar, für jede Kategorie eine Tabelle anzulegen.

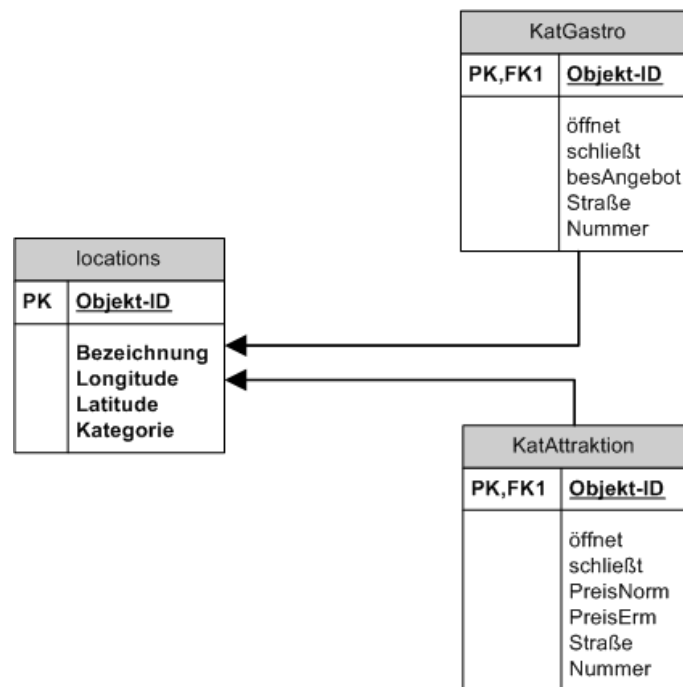


Abbildung 4-6: ER-Diagramm der erweiterten Datenbankstruktur mit Kategorien

4.5 Administrations-Interface

Für den vorläufigen Prototyp des Systems wurden die Objekte noch von Hand, also mit Hilfe von SQL-Anweisungen, in die Datenbank eingetragen. Im nächsten Schritt soll der Webserver dem

Administrator ein Interface zur Verwaltung des Systems bieten. Über diese Web-Oberfläche kann die Datenbank, in der alle Objekte mit ihren GPS-Positionen erfasst sind, administriert werden. Somit ist es möglich, neue Objekte in die Datenbank einzufügen, alte Objekte zu löschen oder die Informationen zu bereits erfassten Objekten zu verändern.

Weiterhin soll auch potenziellen Kunden die Möglichkeit gegeben werden, Informationen über ihre Einrichtungen in die Datenbank hochzuladen. Dies verlangt eine Benutzerverwaltung, so dass jeder Benutzer, der Objekte in die Datenbank eingefügt hat, auch bloß diese Objekte löschen oder verändern kann. Es darf Benutzern nicht gestattet sein, Informationen anderer Objekte zu manipulieren.

Dafür muss das oben beschriebene Datenbankmodell noch erweitert werden. Es bedarf einer weiteren Tabelle *User*, in der alle angemeldeten Benutzer gespeichert werden. Außerdem bedarf es eines weiteren Attributs *User-ID* in der Tabelle *locations*, um jedem Objekt einen Benutzer zuordnen zu können. Das erweiterte ER-Diagramm ist in der folgenden Abbildung dargestellt.

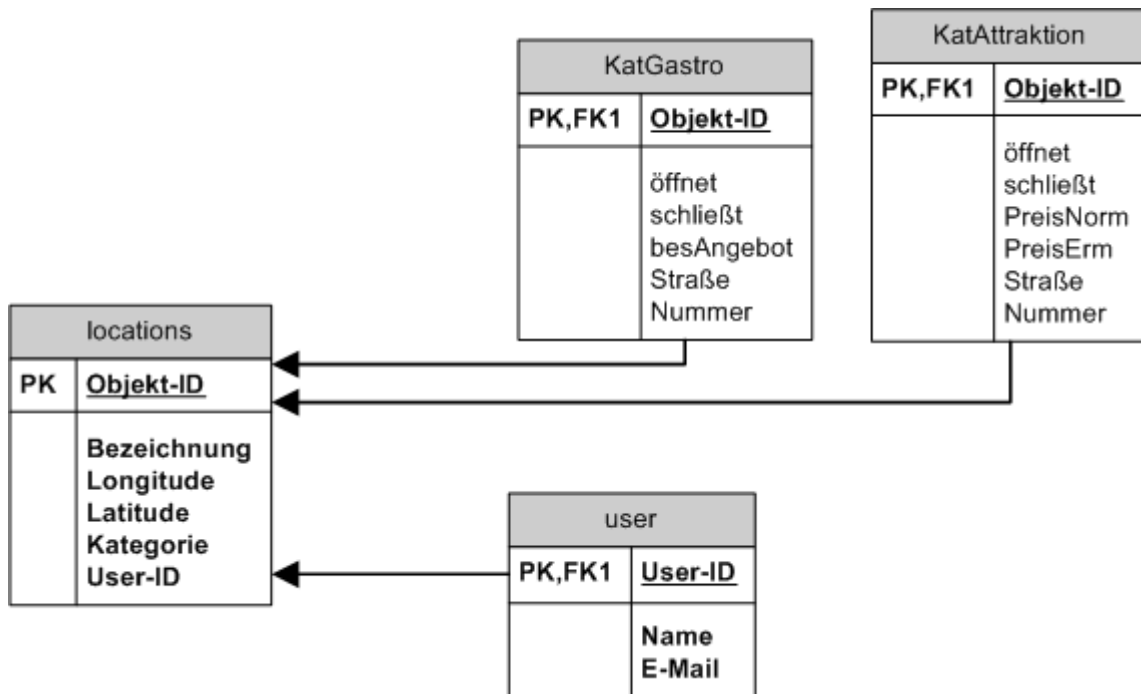


Abbildung 4-7: ER-Diagramm der Datenbankstruktur mit Benutzer-Verwaltung

Um die Konfiguration der Datenbank für die Benutzer und den Administrator möglichst einfach zu halten, werden einige Anforderungen an die Administrationsplattform gestellt. Das Design der Oberfläche soll übersichtlich strukturiert werden, so dass eine intuitive Bedienung möglich ist.

Die Administration soll einfach und schnell von der Hand gehen, ohne dass es notwendig ist, sich durch unzählige Menüs zu klicken.

Die Administrationsseite liegt auf dem Webserver und ist über eine Passwortabfrage geschützt.

Dadurch wird unbefugten Personen der Zutritt zur Administrationsplattform verweigert.

Für die Datenbank wird ein spezieller User angelegt und der Zugriff ebenfalls durch ein Passwort geschützt.

5 Entwicklung des Prototyps

5.1 Entwicklung des Client

Der Prototyp ist für das Betriebssystem Windows Mobile 6 Professional entwickelt worden. Dieses basiert auf dem .Net Compact Framework 2.0 Service Pack 2, einer umfangreichen Klassenbibliothek, die den Entwickler beim Implementieren für Windows-Mobile-Anwendungen unterstützt. Über das Framework lassen sich verschiedene Programmierschnittstellen des Mobiltelefons ansprechen, die beispielsweise den Zugriff auf die Telefonfunktionen (z. B. SMS versenden) und die Gerätekomponenten wie die Kamera oder den GPS-Empfänger ermöglichen. Im Entwicklungsprozess wurden die im Entwurf festgelegten Klassen und Funktionen implementiert. Dabei ist zu erwähnen, dass nicht alle Funktionalitäten, die in den vorausgehenden Kapiteln (siehe Kapitel 3.2.2, 4.1, 4.2) genannt werden, im Prototypen des Client umgesetzt wurden. Das Filtern ist zwar im Klassendiagramm berücksichtigt und auf der Oberfläche implementiert worden, jedoch steht keine Logik dahinter. Ähnliches gilt für eine auswählbare POI-Chronik und die Berechnung der Bewegungsrichtung aus den letzten empfangenen GPS-Positionen.

Die Oberfläche des Hauptdialogs wurde anhand der Benutzeranforderungen im Kapitel Anwendungsfälle gestaltet und mit den Funktionalitäten der Klasse GUI ausgestattet. Das Ergebnis zeigt die folgende Abbildung.



Abbildung 5-1: Anwendungsdialo des Prototypen

Der Dialog zeigt die ohne Logik hinterlegte Quickfilter-Button-Leiste und die Chronik-ComboBox. Die große weiße Fläche in der Mitte ist der Webbrowser. Über das Menü am unteren Bildrand kann man unter anderem den Einstellungsdialog aufrufen (über Programm → Einstellungen). Dieser ist ein modaler Dialog und hat folgende Darstellung.



Abbildung 5-2: Einstellungsdialog des Prototypen

Die einzelnen Eigenschaften der Klasse Settings können über drei TabPages getroffen werden. Über den Button OK werden die getroffenen Einstellungen in die Hauptanwendung übernommen.

Um die Programm-Einstellungen auf dem statischen Speicher des Client hinterlegen zu können, wird die .Net-Klasse XmlSerializer verwendet. Sie kann Objekte einer Klasse serialisieren bzw. deserialisieren, die als XmlRootAttribute gekennzeichnet ist (siehe Listing 1).

```
/// <summary>
/// This class is an objectiv representation
/// of the choosen settings by the user
/// </summary>
[XmlRootAttribute("Settings", IsNullable=false)]
public class Settings
{
    ...
}
```

Listing 1: Klasse Settings als serialisierbar gekennzeichnet

Dazu muss man den Methoden `serialize` oder `deserialize` das entsprechende Objekt und einen Stream auf eine Datei übergeben. Ein Beispiel einer XML-Datei, die eine Instanz der Klasse `Settings` repräsentiert, ist in der folgenden Abbildung dargestellt.

```
<?xml version="1.0" ?>
<Settings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <DefaultFilter>
    <boolean>true</boolean>
    <boolean>>false</boolean>
    <boolean>true</boolean>
    <boolean>>false</boolean>
    <boolean>true</boolean>
    <boolean>>false</boolean>
    <boolean>>false</boolean>
    <boolean>>false</boolean>
    <boolean>true</boolean>
  </DefaultFilter>
  <Update>10</Update>
  <ShowButtons>true</ShowButtons>
  <WebServiceUrl>http://192.168.0.102/locawa/index.php?lat={0}&long={1}</WebServiceUrl>
</Settings>
```

Listing 2: Serialisiertes Settings-Objekt

Die einzelnen Eigenschaften der Klasse sind jeweils als eigenes XML-Element dargestellt, deren Inhalt den Wert wiedergibt.

Des Weiteren sieht man in Listing 2 die eindeutig formatierte URL, die den Webservice spezifiziert. Anstelle der tatsächlichen Werte für Längen- und Breitengrade stehen die Platzhalter `{0}` und `{1}`. Diese werden in der Klasse `GPSUrl` mit Hilfe der statischen Methode `String.Format` durch die tatsächliche Position ersetzt (siehe Listing 3).

```
/// <summary>
/// Returns the URL for the webserver, with the actual gps data.
/// </summary>
public Uri getGpsUrl()
{
    return new Uri(String.Format(serviceUrl, latitude, longitude));
}
```

Listing 3: getGpsUrl-Methode

Beim Aufruf der Methode übergibt man zuerst den formatierten String und anschließend die Werte, welche die Platzhalter ersetzen sollen. Der Rückgabewert ist ebenfalls eine Zeichenkette, die in der Methode `getGpsUrl` in eine Instanz der Klasse `Uri` gewandelt und dem Webbrowser der Anwendung übergeben wird.

Dieser ist ein instanziiertes Objekt der Klasse `WebBrowser`, einem Control von .Net, durch das man einen Browser in seine eigene grafische Anwendung einbinden kann. Er baut die Kommunikation zum Server auf und stellt die erhaltenen HTML-Inhalte grafisch dar.

Dazu übergibt man dessen Eigenschaft `Url` ein Objekt der Klasse `Uri`, wodurch das Control einen `http-Request` an den in der URL spezifizierten Server schickt. Die erhaltene Antwort wird dann auf der Oberfläche des Control interpretiert und dargestellt.

Im Fall der Location-Awareness-Anwendung wird in der URL die aktuelle Position an den Server übermittelt, dieser schickt anschließend eine Liste der POIs, die sich am nächsten zur gesendeten Position befinden.

5.2 Serverseitige Programmierung

5.2.1 Testsystem

Da es in diesem ersten Teil der Entwicklung lediglich darum geht, Grundfunktionalitäten zu testen, wurde auf die Verwendung eines echten Webservers verzichtet. Stattdessen werden die Tests auf einer lokalen Installation durchgeführt. Dabei wird der Webserver durch eine Installation der Produktreihe XAMPP simuliert. „*XAMPP ist ein Komplettpaket, das aus Apache, MySQL, PHP, phpMyAdmin und diversen anderen Komponenten besteht.*“ ([PHP II], S. 47) Somit wird ein kompletter Webserver auf dem lokalen Rechner simuliert. Nachdem über das XAMPP-Control-Panel die Dienste des Webservers und der zugehörigen Datenbank gestartet wurden, kann der Server über den Browser aufgerufen werden. Dies geschieht über den Aufruf der Adresse „localhost“. In der Dateistruktur der XAMPP-Installation befindet sich ein Verzeichnis „htdocs“. Dort kann man eigene HTML-Dateien bzw. PHP-Skripte hinterlegen.

Diese Files lassen sich dann wie normale Webseiten über den Browser aufrufen. Über das mitgelieferte Tool phpMyAdmin ist es auf einfache Art und Weise möglich, Datenbanken zu erstellen und zu verwalten. Diese können dann auch sehr einfach über die PHP-Skripte angesprochen werden.

Dieses Testsystem ist vorläufig völlig ausreichend. Client und Server laufen zwar hier auf demselben Rechner, trotzdem sieht der Client den Webserver aber als externes System, da er wie ein entfernter Webserver über eine URL angesprochen wird. Somit lässt sich die Konnektivität zwischen Client und Server unter sehr realistischen Voraussetzungen testen.

Ebenso lässt sich die Konnektivität zwischen Webserver und Datenbank testen. Damit ist es möglich, einen kompletten Durchlauf durchzuspielen, wie im Sequenzdiagramm in Kapitel 4.3.1 beschrieben.

Eine genaue Beschreibung der Testfälle erfolgt in Kapitel 5.3.2.

5.2.2 Server-Skript

Zur Erstellung einer ersten Version des Prototyps wurde ein PHP-Skript erstellt, welches die Anfragen des Client bearbeitet. Wie im Konzept im Kapitel 4.3 beschrieben, ruft der Client den Server unter Übergabe seiner GPS-Koordinaten auf. Diese werden über GET-Variablen der Programmiersprache PHP übertragen. Das Skript übernimmt diese Daten in lokale Variablen und führt damit die Berechnung des Umkreises durch, in dem nach POIs gesucht werden soll. Diese Berechnung läuft ebenfalls ab wie zuvor beschrieben. Es wird einfach ein Offset auf die beiden GPS-Koordinaten aufaddiert bzw. subtrahiert und somit die Grenzen festgelegt. Über die Dimensionierung des Offsets wird bestimmt, wie groß der Bereich sein soll, in dem gesucht wird. Mit diesen Grenzen wird eine Abfrage an die Datenbank abgesetzt und somit nach Objekten gesucht. Sollten keine Objekte-Bereich gefunden werden, wird eine entsprechende Ausgabe erzeugt und das Skript an dieser Stelle beendet. Wurden Objekte gefunden, so beginnt die Berechnung der Abstände zum Client. Für diesen ersten Prototyp erzeugt die Berechnung noch keine Entfernung in Metern, sondern errechnet nur die mittlere Abweichung von Längen- und

Breitengrad der Objekte gegenüber denen des Client. Die zugehörige Formel sieht folgendermaßen aus:

$$Abstand = \frac{(LongC - LongO) + (LatC - LatO)}{2}$$

LongC = Längengrad des Client

LongO = Längengrad des Objekts

LatC = Breitengrad des Client

LatO = Breitengrad des Objekts

Dieser Abstand wird für alle gefundenen Objekte berechnet und diese Werte eben dann nach genau diesem Kriterium sortiert. Zuletzt erfolgt die Ausgabe der Objekte. Sie werden in einer Liste ausgegeben, welche mit dem Objekt beginnt, das dem Client am nächsten liegt. Der Gesamt Ablauf des Skripts ist in folgendem Struktogramm illustriert.

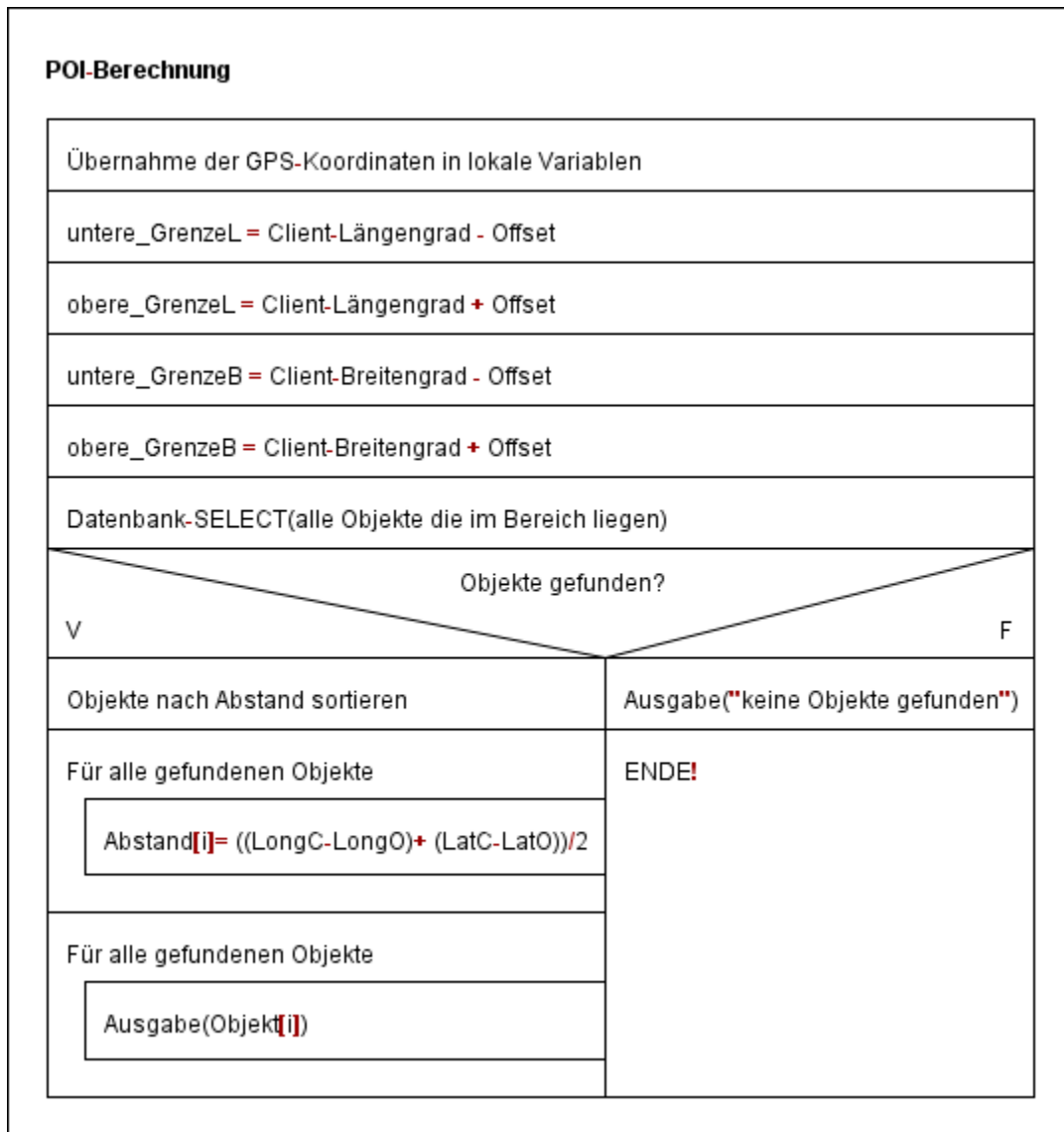


Abbildung 5-3: Ablauf der POI-Behandlung

5.3 Testfälle

5.3.1 Testen der GPS-Daten-Ermittlung

Die geschriebene Software des Client kann man auf dem Emulator der Entwicklungsumgebung Visual Studio 2008 testen. Dieser emuliert die Hardware eines Smartphones und basiert auf einem der Windows-Mobile-Betriebssysteme. Im Falle des Location Awareness Client handelt es sich dabei wie erwähnt um Windows Mobile 6 Professional.

Um nun die GPS-Daten-Ermittlung des Client auf dem Emulator testen zu können, muss man die GPS-Daten simulieren. Dies geschieht anhand des GPS-Simulators FakeGPS, der für Windows Mobile 6 entwickelt wurde. Er simuliert Routen, die ihm im NMEA-Format übergeben werden, indem er die Datensätze nach und nach auf die Datenstruktur des GPS-Empfängers verschiebt. NMEA steht für National Marine Electronics Association und ist ein standardisiertes Format zur Ausgabe der wichtigsten Empfängerinformationen.

Um eine Route in diesem Format zu erhalten, markiert man diese auf Google Earth und speichert sie ab. Die Route wird dadurch in dem Google-spezifischen Format KML abgelegt. Anhand eines GPS-Konvertierers kann man dieses Format nun in den NMEA-Standard wandeln. Das KML-Format enthält zwar nicht alle Informationen, die für die NMEA-Datensätze benötigt werden, jedoch reichen sie aus, um eine Route komplett zu definieren.

Das Ziel beim Testen der GPS-Daten-Ermittlung ist es, die vom Empfänger ermittelten Positionsdaten als formatierte URL zu erhalten. Dazu sollten die Daten von der Speicher-Struktur des Empfängers in das Microsoft-GPS-Framework verschoben und von dort über das GPSInterface an die eigentliche Anwendung weitergegeben werden. Anschließend soll der ermittelte Längen- und Breitengrad in den String geparkt werden, der den Webservice spezifiziert.

Dabei sollen zyklisch bei jedem Aufruf des GPS-Treibers iterierende Werte erhalten werden. Zu Testzwecken wird dazu die URL auf der Oberfläche der Anwendung angezeigt (siehe Abbildung 5-4).



Abbildung 5-4: Testen der GPS-Daten-Ermittlung

Im konkreten Testfall hat man in Google Earth eine Route festgelegt, die innerhalb Stuttgarts vom Hauptbahnhof zum Rotenbühlplatz verläuft, und die Ausgabe mit den tatsächlichen Werten verglichen. Das Ergebnis war eine einwandfreie Ermittlung und Formatierung der dem Simulator übergebenen Werte.

5.3.2 Testen der Webserver-Grundfunktionalität

Zum Testen der Grundfunktionalität des Servers wurde dieser einfach über einen Browser aufgerufen. Der URL wurden dabei beispielhafte GPS-Daten beigefügt. Es wurden zwei verschiedene Szenarien getestet.

Szenario 1:

Der Webserver wird mit validen GPS-Daten aufgerufen, zu denen auch Objekte in der Datenbank abgelegt sind. Es wäre zu erwarten, dass diese Objekte in der Datenbank gefunden werden und geordnet nach ihrer Entfernung zu den übergebenen GPS-Koordinaten angezeigt werden. Dieses Ergebnis wurde vom Testfall erzeugt, welcher somit bestanden ist.

Die Sortierung der Objekte nach ihrer Entfernung zum Client (dessen Daten hier von Hand eingegeben wurden) wurde bei mehreren Testläufen korrekt durchgeführt. Selbst wenn die Objekte nahezu identische Entfernungen zum Client hatten, wurde die richtige Reihenfolge erkannt.

Szenario 2:

Der Server-URL werden GPS-Daten mitgegeben, für die keine Objekte in der Datenbank vorhanden sind. Da keine Objekte in der Datenbank gefunden wurden, wird eine Meldung mit der entsprechenden Information ausgegeben, und das Skript wird beendet. Auch dieses Ergebnis entspricht der Erwartungshaltung. Damit gilt auch dieser Test als bestanden.

5.3.3 Testen des Location-Awareness-Prototyps

Um nun die komplette Anwendung zu testen, muss auf Client-Seite nur noch der konkrete Aufruf des Webservice dem in die Oberfläche integrierten Webbrowser übergeben werden. Dazu wurden zwei klassischen Testverfahren der Softwareentwicklung verwendet: der Black-Box- und der White-Box-Test.

Zuerst wurde der White-Box-Test durchgeführt. Es wurde gezielt eine Route simuliert für die keine Einträge in der Datenbank vorhanden sind und eine Route, die in entsprechender Entfernung zu den POIs verläuft. Dabei gilt, dass der Server dem Client keinen Eintrag für eine übermittelte Position zurückgibt, die über 500m von einem POI der Datenbank entfernt ist. Nach dem Start der Simulation wurden clientseitig Fehler bezüglich der Reihenfolge der Übermittlung von Längen- und Breitengrad festgestellt. Auf Server-Seite wurde bei Routen für die keine POIs in der Datenbank vorhanden sind, keine korrekte Fehlermeldung ausgegeben. Nach Korrektur der Mängel, verliefen die Tests für beide Szenarien erfolgreich.

Beim Black-Box-Test wurden in die Datenbank des Servers drei POIs eingetragen: der Stuttgarter Hauptbahnhof, einen Club in der Stuttgarter Stadtmitte („Keller Club“), der IT-Standort der DHBW Stuttgart.

Nun wurde anhand von FakeGPS die Route vom Stuttgarter Hbf zum Rotebühlplatz simuliert und die Ausgabe mit den zu erwartenden Ergebnissen verglichen.

Die Vergleiche lieferten ein durchweg positives Ergebnis. Im Testverlauf sah man die korrekte Sortierung der Entfernungen zu den einzelnen POIs zu jedem Zeitpunkt. Folgende Abbildung stellt einen Screenshot des Testverlaufs dar.



Abbildung 5-5: Black-Box-Test des Location-Awareness-Prototyps

Weiterführende Tests, wie beispielsweise Performance- oder Stress-Tests, machen in diesem Teil der Studienarbeit noch keinen Sinn, da die Anwendung auf einem Emulator und der Server auf dem Localhost läuft. Die Aufbauzeit eines angeforderten HTML-Dokuments oder auch die Er- und Übermittlung von GPS-Daten ist daher optimal.

Aussagen über die Performance können erst durch das Testen der Anwendung in einer konkreten Umgebung getroffen werden.

6 Schlussbetrachtung und Ausblick

Das Thema der vorliegenden Arbeit war die Konzeptionierung eines Prototyps zur Bereitstellung ortsbezogener Dienste. Szenario 2 in Kapitel 1.3 beschreibt eine Situation, in der ein solches System Anwendung findet. Ausgehend von dieser Anwendungsanalyse wurden Konzepte zur Realisierung eines solchen Systems erstellt. Dabei wurden sowohl nicht-funktionale Aspekte wie die Wahl der System-Architektur als auch funktionale Aspekte analysiert. Auf Grundlage dieser Analyse konnte ein Gesamtkonzept erstellt werden, welches später in kleinere Module zerlegt wurde. Auch für diese Module wurden Lösungsvorschläge erarbeitet und entsprechend ihrer Realisierbarkeit bewertet.

Als Ergebnis dieser Arbeit liegt ein Gesamtkonzept für ein komplettes System vor, welches die Grundfunktionalität gewährleistet, die im oben genannten Szenario verlangt wird. Das Konzept umfasst den client- und den serverseitigen Aufbau des Systems. Dazu wurde bereits ein erster Prototyp erstellt und auf Grundanforderungen getestet.

Im nächsten Teil dieser Studienarbeit steht die Erweiterung des Prototyps im Fokus. Dieser soll weitere Funktionalitäten erhalten. Es gilt die Detailansicht für die einzelnen Objekte zu realisieren und zu prüfen, in welcher Form sich die Informationen am besten speichern lassen. Ein weiterer wichtiger Punkt ist der Aufbau einer Administrationsplattform, über welche die Datenbank des Systems auf einfache Art und Weise verwaltet werden kann. Dieser Punkt umfasst auch die Realisierung des Benutzerkonzepts.

Neben diesen Neuerungen gibt es auch noch einige Erweiterungen, deren Realisierbarkeit erst noch geprüft werden muss. Einer dieser Punkte ist die Einbindung von Kartenmaterial mit der konkreten Darstellung einer Route zum Point of Interest. Eine solche Funktion würde die Anschaulichkeit des Systems enorm erhöhen. Sollte dies nicht durchführbar sein, muss dem Nutzer eine andere Wegbeschreibung geliefert werden. Ein weiteres Feature wäre die Erkennung der Bewegungsrichtung des Client. Dies macht es möglich, dem Client Objekte anzuzeigen, an denen er in Kürze vorbeikommen wird, und andere, die hinter ihm liegen, aus der Ausgabeliste zu entfernen. Außerdem soll im nächsten Teil der Studienarbeit eine komplette Umstrukturierung

der Datenübertragung geprüft werden. Diese würde dann über XML statt wie bisher über HTML laufen, was die gesamte Applikation noch viel dynamischer machen würde.

Wie man sieht, bietet dieses Thema also noch großes Erweiterungspotenzial. In der vorliegenden Arbeit wurde der Grundstein für die Entwicklung eines voll funktionsfähigen Prototyps gelegt.

Allgemein lässt sich sagen, dass die in dieser Arbeit gewonnenen Erkenntnisse eindeutig darauf hinweisen, dass das Thema Location Awareness in der Zukunft eine große Rolle spielen wird.

Die Bereitstellung ortsbezogener Informationen und Dienste bietet sowohl dem Werbenden als auch dem Kunden viele Vorteile, und durch immer billiger werdende mobile Endgeräte könnte hier in den nächsten Jahren ein beträchtlicher Markt entstehen.

Anhang A – Literaturverzeichnis

Buchquellen

[DATENBANKEN 07] Thomas Kudrass, Taschenbuch der Datenbanken, Hanser Fachbuchverlag, 2007.

[LEXIKON 08] Peter Winkler, Computer-Lexikon 2009, Markt und Technik, 2008.

[COMMERCE 02] Günter Silberer / Jens Wohlfahrt / Thorsten Wilhelm, Mobile Commerce: Grundlagen - Geschäftsmodelle - Erfolgsfaktoren, Dr. Th. Gabler GmbH, 2002.

[PHP I 06] Paul Hudson, PHP in a nutshell, O`Reilly Verlag, 2006.

[PHP II 08] Michael Kofler / Bernd Öggl, PHP 5.3 & MySQL 5.1: Grundlagen - Programmieretechniken - Beispiele, Addison-Wesley, 2008.

[SYSTEME 02] Günther Bengel, Verteilte Systeme, Vieweg Verlag, 2002.

[GPS 09] Jean-Marie Zogg, GPS und GNSS: Grundlagen der Ortung und Navigation mit Satelliten, u-blox AG, 2009.

[GPS 01] B. Hoffmann-Wellenhof / H. Lichtenegger / J.Collins, GPS – Theory and Practice, Springer-Verlag, 2001.

[LOCATIONSYSTEMS 08] Anthony LaMarca / Eyal de Lara, Location Systems: An Introduction to the Technology behind Location Awareness, Morgan & Claypool, 2008.

[GPSONE 00] Samir Soliman u.a., gpsOne: A hybrid position location system, in: IEEE (Hrsg.), IEEE Sixth International Symposium on Spread Spectrum Techniques and Applications, Parsippany (New Jersey, USA), 2000, S. 330-335.

Internetquellen

[CLIENT-SERVER] <http://www.herlemann-it-loesungen.de/client-server.html>, 02.02.2010

[FAT-CLIENT] http://cis.cuyamaca.net/draney/214/web_server/client.htm, 02.02.2010

[OSM] <http://www.openstreetmap.org>, 02.02.2010

Anhang B – Abbildungsverzeichnis

Abbildung 2-1: Positionsbestimmung aus Signallaufzeit	13
Abbildung 2-2: Systemaufbau Assisted GPS	17
Abbildung 2-3: Startseite der OpenStreetMap-Homepage	20
Abbildung 3-1: Aufbau einer Fat-Client-Architektur	24
Abbildung 3-2: Aufbau einer Client-Server-Architektur	25
Abbildung 3-3: Use-Case-Diagramm des mobilen Endgeräts	30
Abbildung 3-4: Use-Case-Diagramm der Administrationsplattform	33
Abbildung 4-1: Systemarchitektur des Prototypen	35
Abbildung 4-2: Klassendiagramm Client	37
Abbildung 4-3: Klassendiagramm der GPS-Datenverarbeitung	39
Abbildung 4-4: Detailliertes Klassendiagramm des Hauptprogramms	41
Abbildung 4-5: Sequenzdiagramm einer kompletten Serveranfrage	43
Abbildung 4-6: ER-Diagramm der erweiterten Datenbankstruktur mit Kategorien	48
Abbildung 4-7: ER-Diagramm der Datenbankstruktur mit Benutzer-Verwaltung	49
Abbildung 5-1: Anwendungsdiallog des Prototypen	52
Abbildung 5-2: Einstellungsdialog des Prototypen	53
Abbildung 5-3: Ablauf der POI-Behandlung	58
Abbildung 5-4: Testen der GPS-Daten-Ermittlung	60
Abbildung 5-5: Black-Box-Test des Location-Awareness-Prototyps	63

Anhang C – Glossar und Abkürzungsverzeichnis

AGPS

Assisted GPS – GPS wird mit Daten aus dem Mobilfunknetz ergänzt.

DOD

Department of Defense – US Verteidigungsministerium.

GCS

Global Control Station – Bestandteil des Kontroll-Segments von GPS. Kommunikationsverbindung zu den Satelliten.

GPS

Global Positioning System – Globales Satelliten-Navigationssystem zur Bestimmung der Position und der exakten aktuellen Zeit.

GSM

Global System for Mobile Communications – Mobilfunkstandard für kabellose Telefonie und Versendung von Kurznachrichten.

JPO

Joint Program Office – Amt des DOD. Koordiniert Programme des amerikanischen Militärs.

KML

Keyhole Markup Language – Von Google entwickelte Auszeichnungssprache zur Darstellung von GPS-Daten.

LBS

Location Based Services – Dienste die in Abhängigkeit der Position des Benutzers angeboten werden.

MCS

Master Control Station – Bestandteil des Kontroll-Segments von GPS. Sammelt die von den Monitoring Stations ermittelten Verfolgungsdaten der Satelliten an zentraler Stelle.

MS

Monitoring Station – Bestandteil des Kontroll-Segments von GPS. Misst Pseudorange aller Satelliten die sich in Sicht befinden. Alle MS bilden das Verfolgungsnetzwerk der GPS-Satelliten.

NMEA

National Marine Electronics Association – Ein standardisiertes Format zur Ausgabe der wichtigsten GPS-Empfängerinformationen.

OSM

OpenStreetMap – Eine freie Weltkarte, die aus Daten erstellt wird, welche von ehrenamtlichen Mitarbeitern mit GPS-Geräten gesammelt werden.

POI

Point of Interest – Punkt der Umgebung welcher für den Benutzer von Interesse ist.

Pseudorange

Näherung der exakten Entfernung von GPS-Satellit und Empfänger. Aufgrund des Zeitfehlers von Empfänger und Satellit, einzig messbare Distanz.

UMTS

Universal Mobile Telecommunications System - Mobilfunkstandard für kabellose Telefonie und Versendung von Kurznachrichten. Höhere Übertragungsrate als GSM.

UTC

Universal Time Coordinate – Breiten und Längengrad unabhängige, koordinierte Weltzeit.

Anhang D - Anlagen

Der Anhang umfasst drei Dokumente:

- Die Use-Cases des Projekts
- Die Software-Dokumentation des Clients als HTML-Seiten
- Die Software-Dokumentation des GPS-Framework als HTML-Seiten