Konzeption und Implementierung eines intelligenten Fahrauskunftssystems

Bachelorarbeit

zur Erlangung des akademischen Grades „Bachelor of Science“ an der Hochschule für Technik und Wirtschaft Berlin

Erstellt von:	Benjamin Burzan
Matrikelnummer:	520089
Studiengang:	Angewandte Informatik (B)
Hochschule:	Hochschule für Technik und Wirtschaft
1. Betreuer:	Prof. Dr. Jürgen Sieck
2. Betreuer:	Dipl.-Inf. (FH) Eileen Kühn

Eingereicht am: 03.01.2011

Danksagung

An dieser Stelle möchte ich all denen Personen danken, die mich während der Erstellung dieser Arbeit unterstützt haben.

- Im Besonderen, Stephan Bergemann
- Eileen Kühn, für die Unterstützung als 2. Betreuer
- Prof. Dr. Jürgen Sieck, für die kontinuierliche Unterstützung
- Meiner Familie und allen Korrekturlesern
- Meinen Kommilitonen, die mit in Finnland waren, Rick und Günni

Inhaltsverzeichnis

1 Einleitungen

1.1 Motivation

Viele öffentliche Personennahverkehrsunternehmen bieten auf ihren Webseiten Auskunfts- und Fahrplandienste für den Fahrgast an. Der Fahrgast gibt seinen Startpunkt an und bekommt einen oder mehrere Routenvorschläge zum gewünschten Zielort angezeigt. Diese Lösung hat sich bewährt, basiert jedoch auf einem statischen Fahrplan. Die Fahrplanauskunft auf der Webseite funktioniert nur soweit, wie die Verkehrsmittel fahrplangemäß abfahren bzw. ankommen. Tauchen Störungen im Betriebsablauf auf, muss der Fahrgast besonders darauf achten, mit welcher Bahn oder welchem Bus er zum gewünschten Zielort gelangen kann. Speziell für Touristen ist dies problematisch, da sie prinzipiell aufgrund geringer Orts- oder Sprachkenntnisse Schwierigkeiten haben, sich im öffentlichen Verkehrsnetz zurecht zu finden.

Die Vision des intelligenten Fahrauskunftssystems ist, eine Technik zu schaffen, die es dem Fahrgast ermöglicht, situationsgerecht Fahrinformationen zu beziehen und ihn gleichzeitig von der Konsultation der Fahrpläne oder Auskunftssysteme zu befreien. Der potentielle Fahrgast teilt dem System über sein mobiles Gerät (z.B. Smartphone) mit, zu welchem Zielbahnhof oder welcher Zielstation er geleitet werden möchte. Im zweiten Schritt sucht das System anhand des Standortes des Fahrgastes nach einer geeigneten Verbindung. Diese wird dem Fahrgast nicht direkt mitgeteilt, sondern genutzt, um den Fahrgast zu informieren, wenn die Bahn oder der Bus sich nähert. Auf dem Bildschirm des mobilen Gerätes erscheint eine Meldung „Bitte steigen Sie ein“, oder äquivalent dazu „Bitte steigen Sie in Bahn XY um“.

Der Fahrgast müsste sich nicht mehr mit dem Fahrplan befassen oder gar darauf achten, ob die nächste Bahn relevant ist. Dies ist besonders hilfreich in fremden Städten oder wenn zusätzlich Sprachbarrieren eine Hürde darstellen.

Um nun die Vision des intelligenten Fahrauskunftssystems in Teilen umsetzen zu können, werden wichtige Aspekte definiert und in einer Zielstellung im nächsten Abschnitt formuliert.

1.2 Zielsetzung

Ziel der Arbeit ist, eine benutzerfreundliche und funktionierende Software prototypisch zu entwickeln. Dabei ist es wichtig, dass die Kernfunktion (das Auswählen des Zielortes) und die Benachrichtigungsfunktion funktionieren. Die Schwerpunkte liegen in der Kommunikation zwischen mobilem Gerät und öffentlichem Verkehrsmittel. Hierbei spielt die Evaluation der RFID- und GPS-Technik eine entscheidende Rolle und muss mit Sorgfalt betrachtet werden.

Aus der Definition der Schwerpunkte erschließen sich wichtige Aspekte für die Konzeption des Entwurfs der Kommunikation zwischen Client und Server bzw. Client und Verkehrsmittel. Ein Aspekt ist die Modellierung eines Protokolls, das individuell den Anforderungen entsprechend schlank (möglichst wenig Overhead[1]) aufgebaut sein soll. Ein anderer Aspekt richtet sich an die geeignete Wahl der Ortungstechnik. Diese muss möglichst genaue Positionsdaten liefern oder Näherungen möglichst früh erkennen können.

Zur Erfüllung der Zielsetzung, eine benutzerfreundliche und funktionierende Software prototypisch zu entwickeln, soll ein Client entworfen werden, der die definierten Kernfunktionen beherrscht. Teil des Prototyps ist das Backend (Server) und ein System im öffentlichen Verkehrsmittel.

Wie diese Komponenten aufgebaut und implementiert werden können, wird in dieser Arbeit ausführlich dargestellt. Der nächste Abschnitt soll jedoch zunächst den Aufbau dieser Bachelorarbeit behandeln.

1.3 Aufbau der Arbeit

Die Arbeit ist in sieben Kapitel unterteilt. Nach der Einleitung im Kapitel 1 werden im Kapitel 2 grundlegende Techniken und Begrifflichkeiten erläutert sowie einzelne softwaretechnische Aspekte des Prototypen evaluiert.

Im dritten Kapitel werden hardwarespezifische Fassetten bewertet und Anwendungsszenarien des Fahrauskunftssystems näher erläutert. Basierend auf diesen Erkenntnissen wird mit dem Entwurf fortgefahren. In diesem vierten Kapitel ist das Gesamtsystem in den Teilbereichen Datenmodellierung, Protokollaufbau, Funktionsweise und Benutzeroberflächenentwurf detailliert beschrieben.

Nachdem die konzeptionellen Punkte der Arbeit abgeschlossen sind, wird im Kapitel 5 mit der Implementierung des Prototypen begonnen und parallel dessen programmierspezifische Funktionsweise dokumentiert.

Abschließend werden im Kapitel 6 Tests der Funktionstüchtigkeit des Prototypen durchgeführt und dessen Ergebnisse dokumentiert, sowie anschließend im letzten Kapitel 7 die Arbeit zusammenfassend betrachtet und ein Ausblick darüber gegeben, wie das intelligente Fahrauskunftssystem zukünftig weiterentwickelt werden könnte.

[1] Zusatzinformationen, die zur Übermittlung von Daten benötigt werden und nicht als Nutzdaten zählen.

2 Grundlagen

Dieses Kapitel vermittelt wichtige technische Grundlagen für den weiteren Verlauf der Arbeit.

Zuerst werden softwarebasierte Techniken zur Entwicklung von Webanwendungen und unidirektionalen Kommunikationsflüssen erläutert. Anschließend beschreibt das Kapitel hardwarebasierte Kommunikations- und Ortungssysteme, wobei speziell OpenBeacon[2] und GPS[3] thematisiert wird.

2.1 Software

2.1.1 Adobe Flash

Die erste Version von Flash, damals von Macromedia entwickelt, wurde 1997 veröffentlicht. Das proprietäre Browser-*Plugin* der Firma Adobe (Adobe übernahm Macromedia im Jahre 2005) ist eine weitverbreitete, meist als Browser-*Plugin* zu findende Software. Auf Grund seiner Fähigkeit, alle Arten von kodierten Videos abzuspielen, wurde Flash schnell populär. Primär jedoch sind interaktive Inhalte wie aufwändige Webseiten, Spiele oder andere Webapplikationen der Fokus dieser Technik.

Eine weitere Stärke von Flash liegt in der Tatsache, dass der definierte Funktionsumfang über alle Plattformen hinweg gleich umgesetzt ist. Der Entwickler kann sich darauf verlassen, dass sein Programm auf allen flashunterstützen Plattformen entsprechend gleich funktioniert.

Der Dialekt *ActionScript* der Scriptsprache *ECMAScript* dient der Entwicklung von Flash-Inhalten. Diese mit Flash Version 4 integrierte Sprache ermöglicht die Entwicklung interaktiver Inhalte und komplexer browserbasierter Webanwendungen.

Das folgende Listing[4] zeigt ein einfaches „Hallo Welt“-Beispiel in *ActionScript* 3.0.

---

[2] Freie RFID-Implementierung

[3] Global Positioning System

[4] Ausdruck eines Quelltextes

```
1       package com.beispiel
2       {
3               import flash.text.TextField;
4               import flash.display.Sprite;
5
6               public class HalloWelt extends Sprite
7               {
8                       public function HalloWelt()
9                       {
10                              var txtHallo:TextField = new TextField();
11                              txtHallo.text = "Hallo Welt";
12                              addChild(txtHallo);
13                      }
14              }
15      }
```

Listing 2-1: „Hallo Welt“-Beispiel in ActionScript 3.0

In die Kritik geriet Flash wegen einer fehlenden performanten Implementierung für mobile Geräte mit Touchscreen (Jobs, 2010). Zusätzlich problematisch ist die Bedienung mit dem Finger. Über viele Jahre wurden Flashanwendungen für Maus und Tastatur entwickelt. Seit dem Durchbruch der Smartphones und dem mobilen Internet müssen die für Maus und Tastatur optimierten Oberflächen von Grund auf neu durchdacht werden, da sie mit Fingergesten nur schwer zu bedienen sind.

Des Weiteren fehlt es an ressourcenschonenden Browser-*Plugins*. Diese Problematik ist bei Betriebssystemen wie Mac OS X oder diversen Linux-Distribution besonders nachteilig. Adobe Flash ist kein offizieller Webstandard und wurde nicht offen spezifiziert (Adobe Systems, Inc., 2008).

2.1.2 HTML5

Die Version 5 der *Hypertext Markup Language* wird eine komplett neu geschriebene Spezifikation des HTML-Webstandards sein. Wichtige bzw. prominente Neuerungen von HTML5 sind folgende:

- Canvas-Element

  Ermöglicht das dynamische Umsetzen von 2D-Grafiken im Webbrowser

- Geolocation API

  Bietet Ortungsfunktionen und abstrahiert die Wahl der Positionierungsmethode

- Media Playback

  Fügt neben dem Audio- und Video-Tag eine Funktion hinzu, die es ermöglicht, das Abspielen von Medien zu regeln.

- Offline Storage / Local Storage

  Webapplikation können lokal gespeichert, ohne Internetanbindung genutzt werden und Informationen in einer lokalen Datenbank ablegen.

- WebSocket API

  Mittels dieser Programmierschnittstelle können TCP basierte bidirektionale Verbindungen zwischen Webanwendungen realisiert werden.

Ein valides W3C HTML5-Dokument muss nicht mehr mit einem <*html*>- oder <*body*>-Tag eingeschlossen sein. Es wird lediglich ein <*!DOCTYPE html*>-Tag als Indikator von HTML5 benötigt.

```
1       <!DOCTYPE html>
2       <title>Hallo Welt Beispiel</title>
3       <p>Hallo Welt</p>
```

Listing 2-2: „Hallo Welt“-Beispiel in HTML5

2.1.2.1 Ziele

Es wurden drei Hauptziele definiert, die HTML5 in seiner vollendeten Fassung erfüllen soll (W3C, 2007).

2.1.2.1.1 Kompatibilität:

Bereits existierende Webseiten in HTML 4 oder früheren Versionen sollen in Zukunft mit einem HTML5-Parser dieselben Ergebnisse liefern, wie sie es auch schon in der Vergangenheit getan haben. Im Falle eines neuen, für ältere HTML-Versionen unbekannten Elements, ist HTML5 so konzipiert, dass eine alternative Methode im Design integriert ist.

Ein Fokus liegt außerdem in der Beibehaltung bereits implementierter Funktionen. Das Rad soll nicht neu erfunden werden. Bereits verwendete Praktiken werden nicht unterbrochen, selbst wenn sie als unschön gelten. Neuentwicklungen bestimmter Funktionen werden nur einen kleinen Teil ausmachen. Die Verbesserung und Weiterentwicklung steht im Vordergrund.

2.1.2.1.2 Zweckmäßigkeit

Änderungen in der Spezifikation werden vornehmlich vorgenommen, um gegenwärtige Probleme im Web in Hinblick auf die Zukunft zu lösen.

Per Design wird auf Sicherheit geachtet, indem beispielsweise eine Methodik eingeführt wird, die *XSS*[5] sicher mittels *Cross-document messaging* gestaltet.

Die Spezifikation des *DOM*[6] soll für *XML*[7]-, *XHTML*[8]- und HTML-Dokumente konsistent funktionieren und so geringfügig wie möglich abweichen.

2.1.2.1.3 Interoperabilität

Der letzte Punkt, die Interoperabilität, fokussiert die Einfachheit und Eindeutigkeit von definiertem Verhalten. Potentiell auftretende Fehler werden von niedrigen Instanzen abgefangen, um ein angenehmes Nutzungserlebnis (*User Experience*) zu bewahren.

Außerdem ermöglichen spezielle Methodiken unterschiedliches Anzeigen von HTML. Dies kommt besonders Nutzern mit Beeinträchtigungen zugute, da HTML in interpretierter Form auf verschiedenste Weise wahrgenommen werden kann.

Durch die konsequente Implementierung von *Unicode* wird der Internationalisierung Rechnung getragen.

2.1.3 HTML5 in Verbindung mit CSS3 und JavaScript

HTML in der Version 5 sorgt nicht alleine für interaktive Inhalte. Hierfür kann in Kombination mit *CSS*[9] und JavaScript eine vielversprechende Alternative zu Flash, Silverlight und JavaFX mit offenen Standards und Spezifikationen geschaffen werden. Im Web sind viele Demonstrationen zu finden, die die Möglichkeiten dieser Technik aufzeigen (Melanson, 2010).

Aktuelle moderne Browser unterstützen bereits viele Funktionen der neuen HTML und CSS Version. Mit optimierten *JavaScript-Engines* kann die Ausführung von interaktiven Inhalten beschleunigt werden.

---

[5] Cross-Site Scripting

[6] Document Object Model

[7] Extensible Markup Language

[8] Extensible Hypertext Markup Language

[9] Cascading Style Sheets

2.1.4 Weitere Frameworks für Rich Internet Applications

Die beiden folgenden Frameworks sind nur für einzelne mobile Plattformen verfügbar. Aus diesem Grund wird folgend nur grundlegend auf diese eingegangen.

In direkter Konkurrenz zu Flash steht unter anderen das 2007 herausgegebene Silverlight der Firma Microsoft. Das Browser-*Plugin* ist für Windows, Mac OS X und Linux verfügbar und steht für den Internet Explorer, Mozilla Firefox und für Safari bereit. Die Software ist proprietär, jedoch ist es möglich, mit der freien Entwicklungsumgebung Moonlight Silverlight Applikationen zu entwickeln. Eine mobile Implementierung gibt es bisher nur für das Microsoft Betriebssystem Windows Phone 7 (Microsoft Corporation, 2010).

JavaFX der Firma Oracle (ehemals Sun Microsystems) ist als Browser-Erweiterung für Windows, Linux, Solaris und Mac OS X verfügbar. Dieses *Framework* ist wie Flash und Silverlight zur Entwicklung von plattformübergreifenden *Rich Internet Applications* geeignet. JavaFX funktioniert prinzipiell auf allen Plattformen auf denen auch Java SE *Runtime* läuft. Die mobile Variante JavaFX Mobile ist jedoch nur für Android und Windows Mobile verfügbar.

2.1.5 Vergleich

In der folgenden Tabelle werden Techniken zur Entwicklung von Webanwendungen aufgeführt und miteinander nach verschiedenen Kriterien verglichen.

Technologie	Funktionalität	Verbreitung[10]	Unterstützte Plattformen		Gesamt
			Desktop	Mobil	
HTML5	+	- -	++	++	**+**
Flash	++	++	+	-	**+**
JavaFX	+	+	+	-	**-**
Silverlight	+	-	+	-	**-**

++ sehr gut, + gut, - befriedigend, - - ungenügend

Tabelle 2-1: Vergleich Web-Technologien

2.1.6 Verbreitung

An dieser Stelle wird die Verbreitung (auch als Marktanteil bezeichnet) der in diesem Kapitel erläuterten *Rich Internet Application Frameworks* aufgeführt. Dies kann für Flash, JavaFX und Silverlight mit Hilfe von statistischer Erhebungen ermittelt werden, da diese drei Technologien als *Plugin* im Webbrowser installiert sind und dies von Webseiten geprüft werden kann. Somit lässt sich die

[10] Siehe 2.1.6 Verbreitung

Verbreitung der einzelnen Frameworks feststellen. Für HTML5 wird die Ermittlung des Marktanteils schwieriger, dennoch kann durch Korrelation verschiedenster Daten die Verbreitung von HTML5 gemesen werden.

2.1.6.1 Plugins für Rich Internet Applications

Technologie	Erscheinungsjahr	Unterstützte Plattformen	Verbreitung in Prozent
Flash	1997	Windows, Mac OS X, Linux, Solaris, Android	96,73
JavaFX	2007	Windows, Mac OS X, Linux, Solaris	79,39
Silverlight	2007	Windows, Mac OS X, Linux, Windows Phone 7	51,43

Tabelle 2-2: Faktenübersicht Web-Technologien (Stat Owl, 2010)

2.1.6.2 HTML5

Auf Grund des Fakts, dass HTML-*Interpreter* direkt im Browser implementiert sind, ist es nicht einfach, die Verbreitung festzustellen. Im nun folgenden Abschnitt werden verschiedene Statistiken zusammengefügt, um eine ungefähre Verbreitung von HTML5 ermitteln zu können.

Als Basis dient eine Statistik der verwendeten Browser im Internet. Es ist dabei wichtig, dass eine Aufschlüsselung nach Browserversionen dargestellt wird, da HTML5 von Version zu Version fortschreitend implementiert wird. Auf Grund der Schwere der Identifizierbarkeit wird bei der Analyse „Other“ außer Acht gelassen. Zu „Other“ gehören unter anderen Browser wie Konqueror, Lynx, Mozilla bzw. Firefox-Abarten und ältere Versionen der populären fünf Browser.

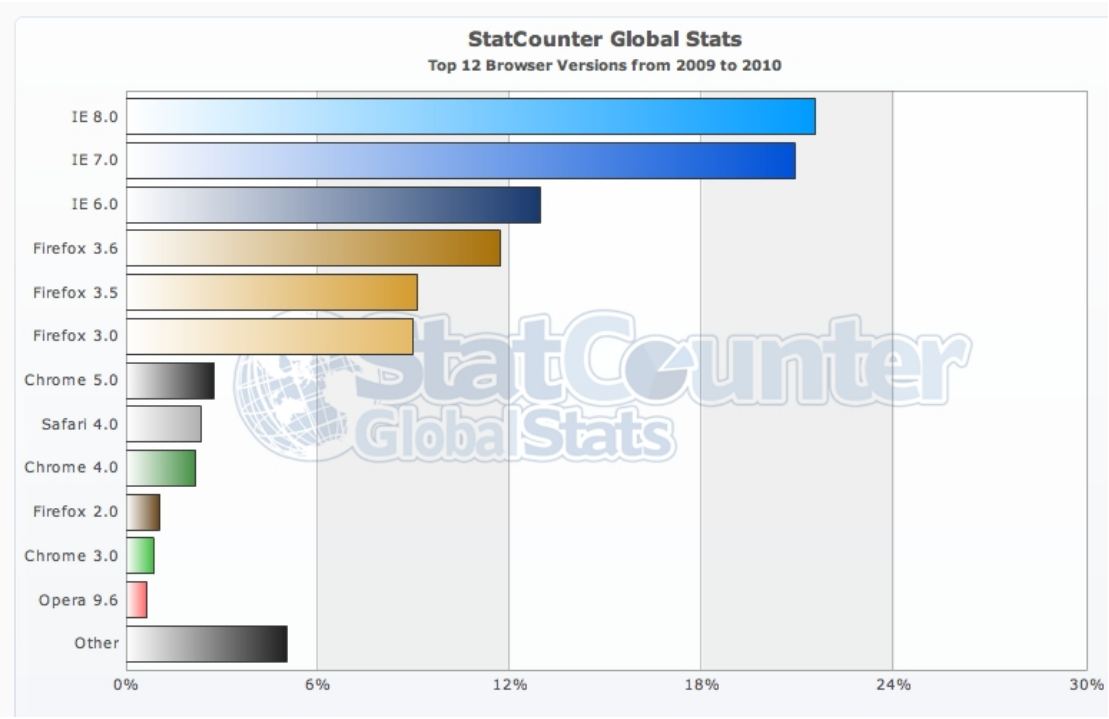

Abbildung 2-1: Top 12 Browser Versionen (StatCounter, 2010)[11]

In der folgenden Übersicht werden die Stände der Implementierungen von HTML5 in den jeweiligen Browsern tabellarisch dargestellt. Die Prozentzahlen geben an, wie viele der neuen HTML5-*Features* bereits von den HTML-*Engines* der Browser interpretiert werden können. Die fett markierten Zeilen deuten auf die aktuelle (*Release*)-Version des jeweiligen Browsers hin (Stand 22. September 2010).

[11] Genaue Angaben sind im Anhang browser_version-ww-yearly-2009-2010-bar.csv zu finden.

Browser	Version	HTML5 Unterstützung in Prozent	Marktanteil in Prozent
Chrome	4.0	83	2,18
	5.0	88	2,76
	6.0	**88**	**0,41**
	7.0	96	0,01
	8.0	96	0,00
Firefox	3.0	42	8,98
	3.5	77	9,11
	3.6	**83**	**11,69**
	4.0	96	0,01
Internet Explorer	6.0	13	12,92
	7.0	19	20,88
	8.0	**27**	**21,54**
	9.0	81	0,00
Safari	3.2	46	0,36
	4.0	83	2,41
	5.0	**88**	**0,64**
	5.*	96	0,00
Opera	10.1	52	0,01
	10.5	69	0,00
	10.6	**77**	**0,26**
	10.7	77	0,00

Tabelle 2-3: Webbrowser-Marktanteile mit HTML5-Unterstützung (Fyrd)

Anhand der Browser-Statistik und des Fortschritts der HTML5-Implementierung ist es möglich, eine ungefähre Abschätzung zur Verbreitung zu machen. Bisher hat kein Browserhersteller HTML5 in vollem Umfang implementiert. Dies ist dem Fakt geschuldet, dass die Spezifikation des Standards sich in der Bearbeitungsphase befindet und noch nicht von dem World Wide Web Consortium[12] verabschiedet worden ist. Voraussichtlich Ende 2010 wird eine vorläufige Referenzversion der Spezifikation erscheinen (Plehegar, 2010).

Der Richtwert von mindestens 70 Prozent Implementierungsfortschritt (Stand September 2010) soll der Abschätzung der Verbreitung dienen. Anschließend wird der Marktanteil der Browser, die den Richtwert einhalten oder überschreiten, zusammenaddiert. Demnach ergibt sich eine Verbreitung von HTML5 von 29,48 Prozent.

[12] Abk. W3C

2.1.7 Zusammenfassung

Die anhaltende Kritik an Adobe Flash und die Forderungen nach performanten Implementationen für diverse mobile Geräte und Betriebssysteme verhilft HTML5 zu weiterer Aufmerksamkeit. Über 50 Prozent des mobilen Webbrowser-Marktanteils wird von einem Browser gehalten, der auf einem Betriebssystem läuft, das Flash nicht unterstützt (Apple Inc., 2010).

Anhand der Verbreitung und des derzeitigen Standes der Spezifikation ist allerdings zu sehen, dass HTML5 noch lange nicht alltagstauglich ist. Meist wird daher als *Fallback*[13] auf bereits etablierte Techniken wie Flash oder Java zurückgegriffen. Dies zeigt, dass gegenwärtig weiterhin seitens der Webseitenbetreiber auf Flash gesetzt wird. Jedoch kann sich dies zukünftig ändern, wenn die Spezifizierung von HTML5 abgeschlossen und diese in allen Browsern vollständig implementiert ist.

Technologie	Verbreitung in Prozent	Unterstützte Plattformen
Flash	96,73	Windows, Mac OS X, Linux, Solaris, Android
HTML5[14]	29,48	Windows, Mac OS X, Linux, BSD, Android, iOS, WebOS, Symbian, Maemo, Java, Windows Mobile, Windows CE, Qt, Playstation 3
HTML5/CSS3/ JavaScript	29,48	Siehe HTML5
JavaFX	79,39	Windows, Mac OS X, Linux, Solaris
Silverlight	51,43	Windows, Mac OS X, Linux, Windows Phone 7

Tabelle 2-4: Verbreitung von Web-Technologien (Stat Owl, 2010)

[13] Ausweichmöglichkeit

[14] Unterstützte Plattformen von HTML5 resultieren aus Browsern die auf WebKit und Gecko-basieren.

Wäre die Verbreitung der Maßstab für die Wahl der Entwicklungsumgebung, ist Flash, mit 96,73% Marktanteil auf Desktop-Computern, der beste Kandidat. Die zu entwickelnde Software soll auf mobilen Geräten zum Einsatz kommen. Aus diesem Grund spielen vorrangig die unterstützten Plattformen eine primäre Rolle. Der noch im frühen Stadium befindlichen HTML5-Technologie fehlt es momentan an der Verbreitung. Der Trend geht jedoch klar zu offenen Webstandards welche in Zukunft besser von den Browserherstellern unterstützt werden und somit Einzug in die fünf populären Desktop-Browser (Chrome, Firefox, Internet Explorer, Safari, Opera), sowie in die mobilen Browser (Android Browser, Blackberry Browser, Firefox for mobile, Internet Explorer Mobile, Mobile Safari, Opera Mobile) halten werden. In dieser Arbeit wird für die Entwicklung der mobilen Plattform aus den vorher genannten Gründen auf HTML5 gesetzt.

2.2 Push-Dienste

Die Push-Technologie beschreibt eine Methodik, die es ermöglicht, den Informationsfluss vom Sender zum Abonnenten zu steuern. Dabei bekommt der Empfänger die Informationen direkt geliefert, ohne dass diese separat abgefragt werden müssen. Das englische Wort *push* steht für drücken oder schieben. Verdeutlicht gesagt: der Empfänger bekommt die Nachricht zugeschoben.

Im Gegensatz dazu steht die *Pull*-Technologie. Hierbei werden Informationen unmittelbar auf Wunsch des Empfängers vom Sender abgefragt und übersendet. Diese aktive Methode der Informationsbeschaffung ist vergleichbar mit einem Anruf bei der Telefonauskunft oder Surfen im World Wide Web mittels eines Browsers.

Ein weitverbreiteter Anwendungsfall des *Pushes* ist beispielsweise ein Zeitungsabonnement oder der sogenannte E-Mail-*Push*. Vor einigen Jahren wurde besonders im Bereich der internetfähigen Mobilfunktelefone (als *Smartphones* bekannt) die *Push-Notification* (*Push*-Benachrichtigung) sehr populär.

In den folgenden Kapiteln wird näher auf HTTP-basierende Push-Verfahren und auf die Push-Notifikations-Dienste der *Smartphones*-Betriebssystem Hersteller Apple und Google eingegangen,

2.2.1 HTTP-Push

HTTP-Push bezeichnet einen Mechanismus, Daten vom Webserver zum Webbrowser zu senden. Hierfür existieren vier Methoden.

- Common Gateway Interface (CGI)

  Definiert einen Standard für den Datenaustausch zwischen externer Software und Webserver. Dies ermöglicht einem Programm, eine bidirektionale Kommunikation zum Client aufzubauen.

- MIME[15] Typ *multipart/x-mixed-replace*

  Mittels dieses MIME-Typs interpretiert der Webbrowser die Webseite als „änderbar“. Dies erlaubt dem Webserver, eine neue Version der Webseite zum Client zu senden. Dieser MIME-Typ wird vom Microsoft Internet Explorer nicht unterstützt.

- Server-Sent-Events (SSE)

  Die *JavaScript*-Programmierschnittstelle *EventSource* bietet im Rahmen des HTML5-Standards die Fähigkeit, ereignisbasierte Push-Nachrichten von einem Server zu empfangen (Hickson, 2010).

- WebSocket API[16]

  Mit Hilfe von *WebSockets* können bidirektionale TCP-Verbindungen zwischen Client und Webserver hergestellt werden. In Kombination mit bspw. *Ajax*[17] können vom Webserver gesendete Daten direkt auf die Weboberfläche *gepusht* werden.

2.2.2 Push-Benachrichtigung auf Smartphones

Mit sogenannten Push-Benachrichtigungen ist es möglich, den Benutzer des mobilen Gerätes in Form einer Popup-Nachricht oder Symbolik auf Ereignisse hinzuweisen. Diese Ereignisse können von auf dem Smartphone installierten aber nicht laufenden Applikationen stammen. In dem folgenden Abschnitt werden zwei Techniken verschiedener Hersteller von mobilen Betriebssystemen erläutert. Im Kern funktionieren diese sehr ähnlich.

---

[15] Multipurpose Internet Mail Extensions

[16] Englisch: Application programming interface, Deutsch: Programmierschnittstelle

[17] Asynchronous JavaScript and XML

Das Grundprinzip der Benachrichtigungs-Dienste teilt sich in drei Bereiche auf.

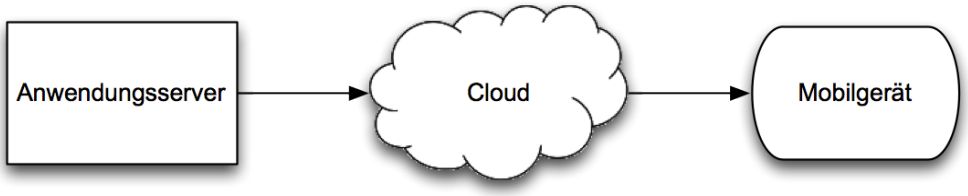

Abbildung 2-2: Grundprinzip Benachrichtigungs-Dienste

- Der Anwendungsserver generiert die zu versendenden Informationen und wird deshalb auch als *Provider* bezeichnet.
- Die Cloud (deutsch Wolke) hält die Verbindung zum mobilen Gerät und dient als Zwischeninstanz. Der Name *Cloud* bezeichnet den jeweiligen Dienst des Betreibers der Infrastruktur.
- Der Empfänger der Nachricht ist schlussendlich die auf dem Mobilgerät befindliche Applikation.

2.2.2.1 Apple Push Notification Service (APN)

Der Apple Push Notification Dienst ist die elementare Zentrale für die Benachrichtigungsfunktion für mobile Geräte mit Apples Betriebssystem *iOS*. Die Möglichkeit, Benachrichtigungen von installierten Applikationen zu empfangen, besteht seit der Version 3 des damals noch *iPhone OS* benannten Betriebssystems. Jedes Gerät kommuniziert durch eine persistente, verschlüsselte Verbindung mit dem APN-Dienst. Wird eine Benachrichtigung für eine nicht laufende Applikation empfangen, gibt das Betriebssystem diesen Hinweis an den Benutzer weiter. Der Provider (Anwendungsserver), der die Nachrichten generiert und die Übersendung initiiert, hält außerdem eine persistente verschlüsselte Verbindung zum APN-Dienst. Sollten für einen bestimmten Benutzer oder eine Gruppe neue Benachrichtigungen anfallen, sendet der Provider diese durch den als Kanal fungierenden APN-Dienst. Anschließend leitet dieser die Informationen mittels *Push* an die Benutzer weiter.

2.2.2.1.1 Übertragung der Benachrichtigung

Die Hauptaufgabe des Apple Push Notification Dienstes besteht darin, ein Benachrichtigungspaket vom Provider zum Gerät zu leiten. Dieses Benachrichtigungspaket beinhaltet neben den Informationen (*Payload*) auch die Gerätenummer (*Device Token*). Die Gerätenummer spielt eine wichtige Rolle bei der Authentifizierung der Benachrichtigung beim APN-Dienst und der Zuordnung des Gerätes, das die Nachricht empfangen soll.

Auf folgender Abbildung ist die einseitige Datenvermittlung vom Provider zum Gerät illustriert.

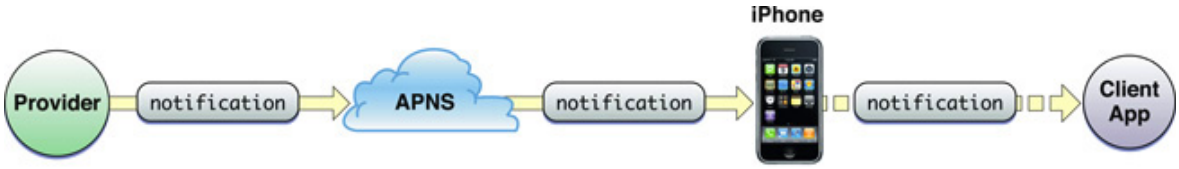

Abbildung 2-3: Datenvermittlung von Provider zu iOS-Gerät (Apple, 2010)

In diesem Fall wird eine 1:1-Verbindung gezeigt. In der Praxis handelt es sich jedoch um n:m-Beziehungen. Das bedeutet, der APN-Dienst ist meist mit mehreren Providern und Geräten verbunden wie folgend zu sehen ist.

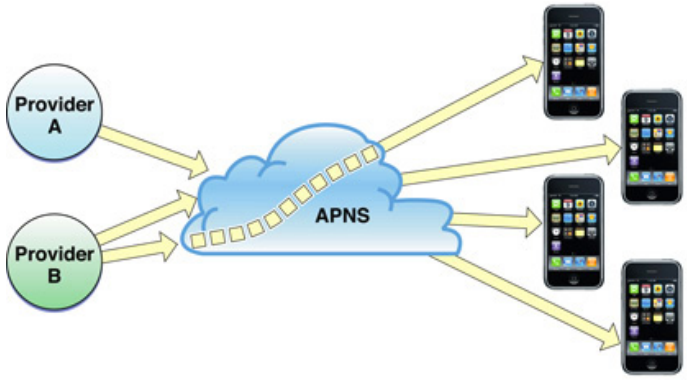

Abbildung 2-4: APN-Provider n:m-Beziehungen iOS-Geräte (Apple, 2010)

2.2.2.1.2 Sicherheit

Die Kommunikationsschnittstelle des APN-Dienstes fungiert als Mittelsmann zwischen Provider und Gerät. Hierfür stellt der Dienst auf zwei Ebenen eine Art von Authentifizierung für den Provider, dem Gerät und der Kommunikation bereit.

- Connection Trust (Authentifizierung auf Verbindungsbasis)

  Durch den APN-Dienst ist sicherzustellen, dass der vom Dienst legitimierte Provider der ist, der er angibt zu sein. Auf der anderen Seite muss das Gerät validiert sein, um mit dem APN-Dienst kommunizieren zu dürfen.

- Token Trust

  Das die richtige Nachricht des Providers, auch beim richtigen Gerät ankommt, wird mit der Gerätenummer sichergestellt. Da diese Nummer dem Gerät und dem Provider bekannt ist, ist es die Aufgabe des APN-Dienstes, die Benachrichtigung an die jeweils vom Provider angegebene Gerätenummer zu versenden.

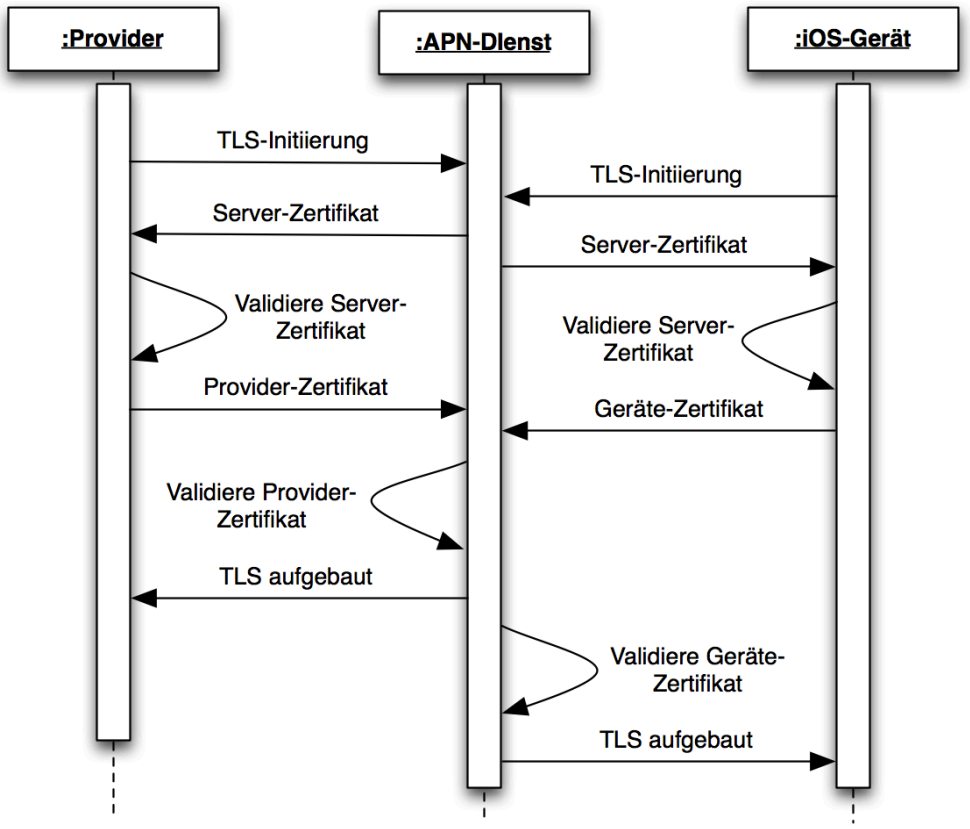

Abbildung 2-5: Sequenzdiagramm Kommunikationssicherheit (Apple, 2010)

Der Provider kommuniziert mit dem APN-Dienst über eine *TLS*[18]-gesicherte *Peer-To-Peer*-Verbindung. Anhand der Validierung der Zertifikate bei einer *Certificate-Authority* kommt eine authentifizierte verschlüsselte Verbindung zustande. Das *iOS*-Gerät wendet die gleiche Methodik an, um mit dem APN-Dienst zu kommunizieren.

Der Aufbau dieser persistenten Verbindung gilt für eine auf dem Gerät installierte Applikation. Jede weitere Applikation muss jeweils eine separate Verbindung zum APN-Dienst aufbauen.

Im nächsten Abschnitt soll nun das Push-Verfahren des Android Betriebssystems erläutert werden. Im Speziellen wird auf den Lebenszyklus einer Nachricht eingegangen und daran das gesamte System beschrieben.

[18] Transport Layer Security

2.2.2.2 Android Cloud to Device Messaging (C2DM)

Das Android Cloud to Device Messaging Framework der Firma Google erlaubt es, Benachrichtigungen an Android-Geräte zu senden. Auf dem Gerät installierte Applikationen können diese Benachrichtigungen empfangen und auswerten ohne dass diese zunächst aktiv sein müssen.

Aufgrund der Limitierung der Nachrichten auf 1024 Bytes ist das Framework nicht dafür ausgelegt, Daten (*Payload*) zum Gerät zu *pushen*. Es sollen vielmehr Informationen gesendet werden, die das Gerät bzw. die Applikation darauf hinweisen, dass *Payload* verfügbar ist (z.B. über eine mitgelieferte URL[19]). Google limitiert außerdem die Anzahl von Nachrichten, die an bestimmte Applikationen oder Android-Geräte gesendet werden können.

Alle Aspekte wie Verwaltung der Nachrichtenwarteschlangen (*Message Queuing*) und Auslieferung der Nachrichten (*Delivery*) werden vom C2DM-Dienst übernommen. Dennoch garantiert der Dienst keine zuverlässige Zustellung der Nachrichten.

2.2.2.2.1 Lifecycle Flow

Dieser Abschnitt beschreibt den Lebenszyklus einer Nachricht, der in drei Phasen eingeteilt ist. Die Grundlage für das Senden und Empfangen von Nachrichten ist ein *ClientLogin Authorization Token*. Dieser bildet die Basisauthentifizierung der Android-Applikation gegenüber der Google-Dienste. Eine Applikation, die vom C2DM-Dienst Gebrauch machen möchte, muss diesen *Token* im Vorfeld für jede Applikation aushandeln und auf dem Applikationsserver speichern.

2.2.2.2.1.1 Aktivierung von C2DM (*Enabling C2DM*)

Eine laufende Applikation registriert sich für den Empfang von Nachrichten. Dieser Prozess unterteilt sich in drei Schritte:

1. Die Applikation, die Nachrichten empfangen möchte, sendet eine Absichtserklärung (*Registration Intent*) an den C2DM-Dienst. Diese Nachricht enthält die *Application ID* und *Sender ID*.
2. Nachdem die Registrierung erfolgreich war, sendet der C2DM-Dienst eine Bestätigung mit einer *Registration ID* an die Applikation zurück.
3. Um den Registrierungsvorgang abzuschließen, sendet die Applikation die *Registration ID* an den Applikationsserver der dritten Instanz.

2.2.2.2.1.2 Versenden einer Nachricht (*Sending a message*)

Ein Applikationsserver einer dritten Instanz sendet eine Nachricht an ein registriertes Gerät. Folgende fünf Schritte werden durchgeführt:

[19] Uniform Resource Locator

1. Der Applikationsserver verschickt die zu sendende Nachricht an den C2DM-Dienst.
2. Kann die Nachricht aufgrund von Inaktivität des Android-Gerätes nicht direkt zugestellt werden, wird sie in eine Warteschlange eingereiht.
3. Später, wenn das Android-Gerät sich aktiv schaltet, verschickt der C2DM-Dienst die Nachricht erneut.
4. Innerhalb des Android-Gerätes wird die empfangene Nachricht an die Applikation geleitet. Dies führt dazu, dass die Applikation „aufwacht“.
5. Abschließend kann die Nachricht von der Applikation verarbeitet werden.

2.2.2.2.1.3 Empfangen einer Nachricht (*Receiving a message*)

Eine Android-Applikation empfängt eine Nachricht von einem C2DM-Server und verarbeitet diese in drei Schritten:

1. Zur Verarbeitung der empfangenen Nachricht wird vom Android-System der Kern der Nachricht (ein Schlüssel und Inhalt der Nachricht) vom Header etc. entfernt.
2. Das System leitet den Schlüssel und den Inhalt der Nachricht an die Zielapplikation weiter.
3. Anhand des Schlüssels kann die Zielapplikation auf den Inhalt zugreifen und diesen weiterverarbeiten.

Die Applikation ist jederzeit in der Lage, das Empfangen von Nachrichten zu deaktivieren (Google Inc.).

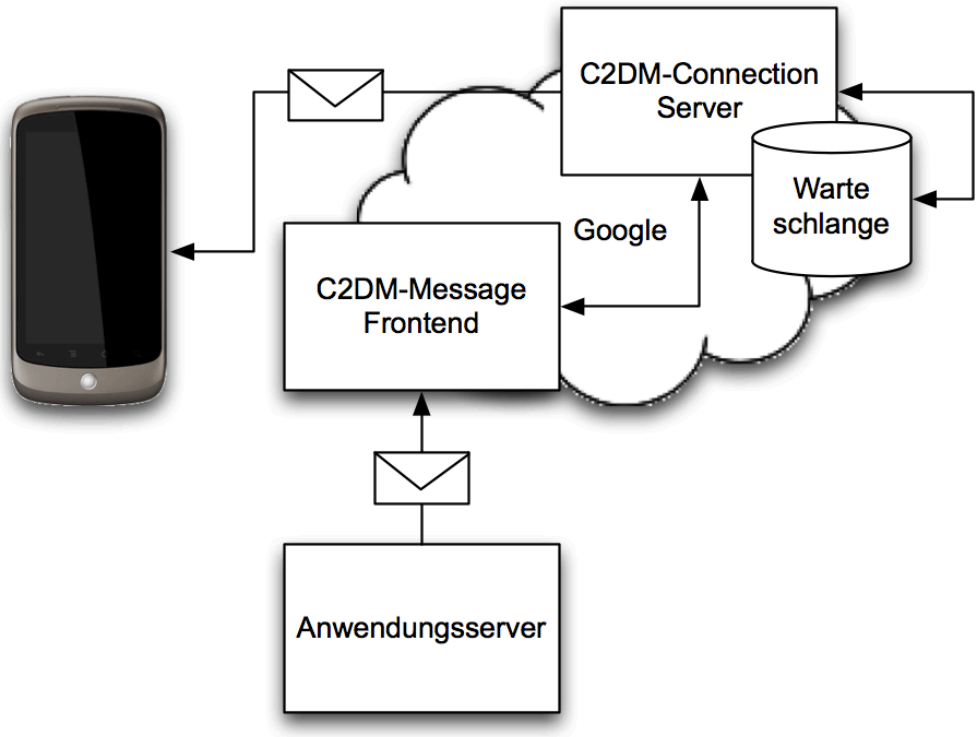

Abbildung 2-6: Weg einer Android Push-Nachricht: (Beckmann, 2010)

2.2.3 Zusammenfassung

Jedes der in diesem Kapitel vorgestellten Push-Verfahren hat seine Stärken, Schwächen und speziellen Anwendungsfälle. Aus diesem Grund scheint es nicht sinnvoll, die Push-Verfahren gegenüberzustellen und zu vergleichen. Es gilt vielmehr zu kategorisieren. Zum einen sind Push-Verfahren wie Apple Notification Service oder Android Cloud to Device Messaging plattformabhängig und zum anderen durch einen Provider (den Hersteller) in ihrer Funktionsweise und Zuverlässigkeit reglementiert. Für die Betrachtung einer plattformunabhängigen Implementierung von Push-Benachrichtigungen bleibt nur das HTTP-Push-Verfahren. Dieses kann mit Hilfe der in bereits vielen Deskop-Browsern implementierten *WebSocket* API realisiert werden. In dieser Arbeit wird aufgrund der noch im experimentellen Stadium befindlichen *Server-Sent-Events* des HTML5-Standards, die *WebSocket* API vorgezogen.

	Plattformabhängig	**Plattformunabhängig**
Benachrichtigung per Push-Nachricht auf Smartphone	• Apple Notification Service • Android Cloud to Device Messaging	
Benachrichtigung Webbasiert		• HTTP-Push

Tabelle 2-5: Übersicht Push-Technologien

2.3 OpenBeacon-Technologie

OpenBeacon ist ein von der Firma Bitmanufaktur ins Leben gerufenes Open Source-Projekt. Ziel des Projektes ist eine quelloffene Hard- und Softwareplattform für den drahtlosen Austausch im internationalen, lizenzfreien 2,4-GHz-ISM-Frequenzband[20] zu entwerfen und umzusetzen (Bergemann, 2010).

Für die Erfüllung dieses Zieles wurden RFID-Tags mit speziell angepasster Firmware entwickelt, die in der Lage sind, mit einer klassischen RFID-Basisstation Informationen auszutauschen. Zusätzlich können die RFID-Tags untereinander (Peer-To-Peer) kommunizieren und sind so fähig, *Ad-hoc*-Netzwerke aufzubauen.

RFID steht für *radio-frequency identification* und ermöglicht eine automatische Lokalisierung und Identifizierung von Gegenständen oder Lebewesen. Die Identifikation geschieht mit Hilfe eines Transponders (Tag) und einem Lesegerät (Antenne). Die Kommunikation der beiden Komponenten funktioniert drahtlos und findet in Frequenzbereichen von 30–500 kHz (LF[21]) für geringe Reichweiten bis zur Mikrowellen-Frequenz von 2,4–2,5 GHz und Frequenzen größer gleich 5,8 GHz (SHF[22]) für höhere Reichweiten (ca. 10 Meter), statt.

Die Kopplung zwischen Transponder und Lesegerät erfolgt üblicherweise mittels induktiver magnetischer Felder, die vom Lesegerät erzeugt werden. Dieses Nahfeld durchdringt die Spulen der Antenne und des Transponders. Durch Induktion in der Antennenspule wird die erzeugte Spannung in gleichgerichteter Form an die Spule des Transponders übertragen. Der Transponder beeinflusst das wirkende Feld der Antennenspule, was wiederum einen messbaren Spannungsabfall in der Generatorspule erzeugt und es somit ermöglicht, Daten zu übertragen. Die Kopplung innerhalb der Mikrowellen-Frequenzen, die sogenannte Fernfeldkopplung, erfolgt über elektromagnetische Felder.

(Elektronik-Kompendium.de)

Bevor im nächsten Abschnitt die OpenBeacon-spezifischen Geräte erläutert werden, wird nachstehend auf die drei Arten von RFID-Transpondern eingegangen.

---

[20] Industrial, Scientific and Medical Band

[21] Englisch: Low Frequency

[22] Super High Frequency

- Passive RFID-Transponder

  Die passiven RFID-Tags benötigen für den Betrieb externe Stromzufuhr. Mittels des Funksignals des RFID-Erfassungsgerätes wird Spannung in der Spule induziert und ein Kondensator geladen. Dies ermöglicht die Speicherung von Spannung, um eine Rückantwort ohne bestehendem Funksignal des Erfassungsgerätes zu übermitteln. Die Leistung des Antwortsignals bestimmt jedoch die Reichweite.

- Aktive RFID-Transponder

  Durch eine eigene Energieversorgung wie zum Beispiel durch eine Batterie, haben aktive RFID-Transponder eine üblicherweise höhere Reichweite (einige Kilometer laut Wikipedia (Wikipedia, 2010)). Sie nutzen die Energie, um das Rücksignal zu erzeugen und den Mikrochip zu betreiben.

- Semi-aktive RFID-Transponder oder auch Semi-passive RFID-Transponder

  Im Unterschied zur der ähnlichen Funktionsweise der passiven Transponder, besitzen die semi-aktiven oder semi-passiven RFID-Transponder keinen eigenen Sender, sondern reflektieren die Signale über Seitenbänder zum Lesegerät zurück. Dies ermöglicht ein stromsparenden Betrieb und hohe Reichweiten (bis zu 100 Meter (rfid-b2b.de)).

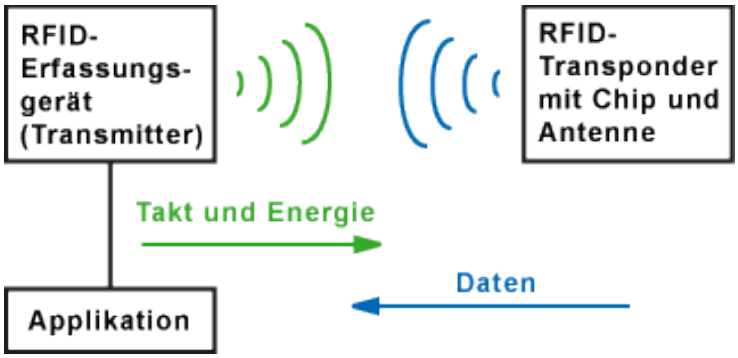

Abbildung 2-7: Übersicht RFID-Kommunikation (Elektronik-Kompendium.de)

Das OpenBeacon-Projekt nutzt Standard RFID-Hardware mit einigen Anpassungen, wie zum Beispiel die Verwendung eigener Firmware. Dies ermöglicht die Implementierung von sogenannten *Proximity-Tags*, die untereinander kommunizieren können. In dieser Arbeit wird jedoch die Reichweite von aktiven RFID-Tags im Mittelpunkt stehen. Das OpenBeacon-Projekt bietet auch hierfür gute Ansätze und somit eine technische Grundlage für die Konzeption des intelligenten Fahrauskunftssystems.

2.3.1 OpenBeacon-Geräte

- OpenBeacon-Tag

  Der OpenBeacon-Tag (siehe Abbildung 2-8 Bild 1) besteht aus einem 2,4-GHz-Transceiver zur bidirektionalen Kommunikation und einem reprogrammierbaren Prozessor. Die Stromversorgung wird durch eine CR2032-Knopfzelle sichergestellt.

- RFID-Antenne

  Es existieren verschiedene OpenBeacon-RFID-Antennen:

  - OpenBeacon USB[23] (siehe Abbildung 2-8 Bild 2)
  - OpenBeacon WLAN[24] (siehe Abbildung 2-8 Bild 3)
  - OpenBeacon Ethernet EasyReader PoE[25] II (siehe Abbildung 2-8 Bild 4)

  Alle drei Antennen empfangen die Signale des RFID-Tags und leiten diese per UDP[26] an einen angeschlossenen Computer weiter.

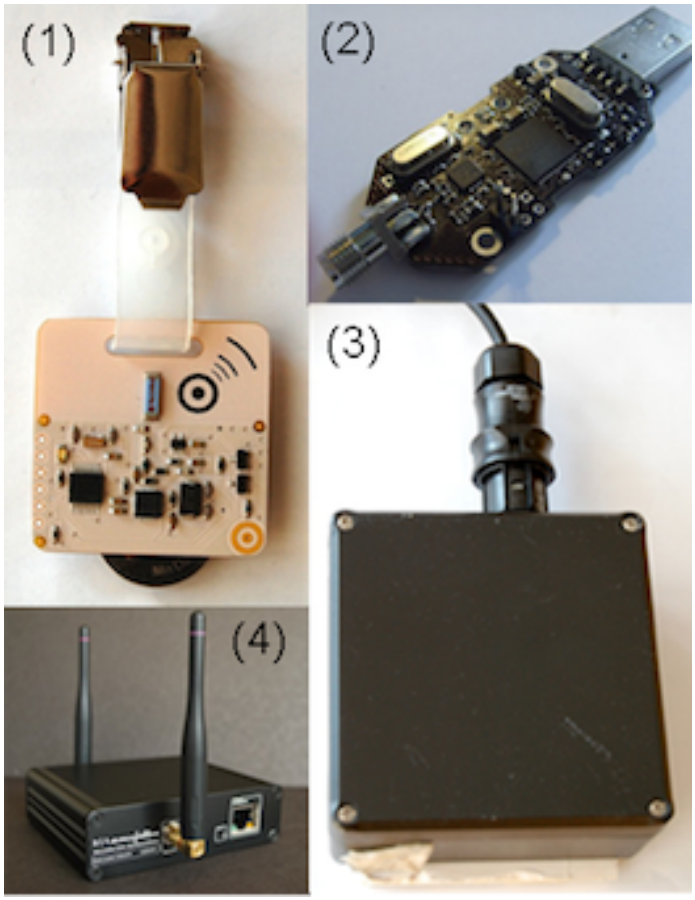

Abbildung 2-8: OpenBeacon-Geräte

[23] Universal Serial Bus
[24] Wireless Local Area Network
[25] Power over Ethernet
[26] User Datagram Protocol

2.3.2 OpenBeacon-System

Das OpenBeacon-System definiert sich als einen Satz von Hard- und Software, der mit Hilfe von *OpenBeacon* eine bestimmte Funktion implementiert. Aufgrund des hohen Grades an Flexibilität sieht vermutlich jedes OpenBeacon-System unterschiedlich aus. Die meisten Systeme werden jedoch dem Master-Slave-Schema folgen.

Dabei dienen eine oder mehrere *Base Stations* als Master und tauschen Informationen mit aktiven OpenBeacon-Tags, den Slaves, aus. Diese Informationen können anschließend an einen Server zur Auswertung weitergesendet werden. Mittels dieser Konstellation kann ein *Location Tracking* der RFID-Tags durchgeführt werden. Dafür senden die RFID-Transponder in definierten Zeitabständen *Beacon*-Pakete an die in der Nähe befindlichen *Base Stations*. Anhand der Signalstärke kann anschließend die Entfernung des Tags errechnet werden. Diese Technik wurde das erste Mal auf dem 23. Chaos Communication Congress unter dem Namen Sputnik eingesetzt (OpenBeacon, 2010).

Das zweite Schema funktioniert ohne die Hilfe von *Base Stations*. Bei dem OpenBeacon-P2P-System kommunizieren alle in Reichweite befindlichen aktiven RFID-Tags untereinander. Es ist sogar möglich, ganze Mesh-Netzwerke[27] aufzuspannen.

Ein weiteres Verfahren, um Objekte zu identifizieren oder positionieren, liegt in dem Einsatz von GPS. Diese Technologie wird in anschließenden Abschnitten detailliert erläutert. Außerdem werden Erweiterung und ähnliche Systeme dieser Technologie beschrieben und verglichen.

[27] Bei diesem Netzwerktyp sind alle Knoten direkt oder indirekt mit allen anderen Knoten verbunden.

2.4 Positionssysteme

Heutzutage werden Systeme zur Positionsbestimmung in vielen Bereichen des alltäglichen Lebens eingesetzt. Dazu zählen elektronische Touristenführer, Auto-Navigationsgeräte oder Geocaching-Spiele. Diese Systeme müssen unterschiedlichen Anforderungen wie Genauigkeit, Verfügbarkeit oder Kosten erfüllen. In diesem Kapitel werden das vom amerikanischen Verteidigungsministerium entwickelte GPS und dessen Erweiterungen, sowie das Projekt Galileo der Europäischen Union thematisiert.

2.4.1 GPS

GPS steht für *Global Positioning System* und hat den offiziellen Namen NAVSTAR, der für *Navigation System for Timing and Ranging* steht. Das System wurde in den 1970er Jahren vom amerikanischen Verteidigungsministerium entwickelt und dient zur Positionsbestimmung und Zeitmessung. Die Technik besteht aus drei Segmenten:

- Weltraumsegment (Satelliten)
- Kontrollsegment (Kontrollstationen)
- Benutzersegment (GPS-Empfänger)

Das Weltraumsegment besteht aus einer Konstellation von nominell 24 Satelliten (National Space-Based Positioning, Navigation, and Timing Coordination Office), die in ca. 20200 km Höhe die Erde umkreisen. Die Satelliten senden ihre momentanen Positions- und Zeitdaten an feststehende, *„sich auf der Erdoberfläche, in der Erdatmosphäre oder in niederen Umlaufbahnen bewegenden“* Empfänger (kowoma.de, Die geschichtliche Entwicklung des GPS-Systems, 2008). Mit einer Geschwindigkeit von 3,9 km pro Sekunde kreisen die Satelliten um die Erde. Die Umlaufzeit beträgt 11 Stunden und 58 Minuten. Bei einer *Inklination* von 55 Grad zur Äquatorebene verlaufen die Umlaufbahnen von 55 Grad nördlicher Breite bis 55 Grad südlicher Breite.

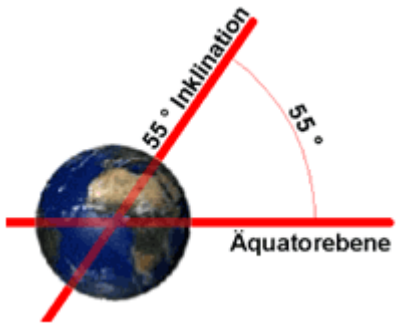

Abbildung 2-9: Bild Inklination (kowoma.de, 2005)

Dies hat den Vorteil, dass zum einen die Umlaufbahnen nördlich genug verlaufen, um eine Positionsbestimmung an den Polen durchführen zu können und zum anderen *„führt diese Anordnung auch zu einer relativ stabilen Konstellation, da Störfaktoren (z.B. Gravitationsfelder, Sonnenwinde) im Mittel auf alle Satelliten gleich einwirken“* (kowoma.de, 2005).

Aufgrund der *Inklination* von 55 Grad befinden sich die Satelliten an den Polen nicht mehr direkt über den Empfängern, sondern näher am Horizont. Die Positionsbestimmung ist jedoch weiterhin möglich, kann jedoch ungenau sein.

Die GPS-Kontrollstationen der Erde stellen das Kontrollsegment dar. Diese bestehen aus Monitor- und Master-Control-Station und liegen vollständig in der Hand der US-Armee. Die Monitor-Stationen sind vereinfacht gesagt GPS-Empfänger, die alle im Sichtbereich befindlichen Satelliten folgen und ihre Signal- und Bahndaten sammeln. Diese Daten werden an die Master-Control-Stationen weitergeleitet und dort analysiert, um Informationen über Uhren und Bahnen aller GPS-Satelliten zu erhalten. Bei eventuellen Fehlfunktionen können Korrekturdaten via *Uplink* (*„S-Band Signal (S-Band: 2000 – 4000 MHz)“* (kowoma.de, 2008)) an die Satelliten zurückgesendet werden.

Das Benutzersegment besteht aus dem GPS-Empfänger des Endverbrauchers. Diese erfreuen sich immer größerer Beliebtheit. Aufgrund des geringen Preises und des hohen Nutzens im geschäftlichen sowie im privaten Bereich konnte sich diese Technik schnell durchsetzen. Die Anwendungsfälle reichen vom modernen Flottenmanagement großer Logistikunternehmen bis hin zum Sightseeing-Navigator für Touristen.

2.4.1.1 Ausgesendete Signale und deren Aufbau

Die Entwicklung des Signalaufbaus war eine komplexe Aufgabe. Insbesondere mussten folgende Kriterien erfüllt werden, um allen Anforderungen der GPS-Infrastruktur gerecht werden zu können:

- Passive Positionsbestimmung
- Entfernungs- und Geschwindigkeitsbestimmung
- Navigationsnachrichten an Satelliten
- Simultane Erfassung mehrerer Satellitensignale
- Bereitstellung von Korrekturen für *ionosphärische* Verzögerungen
- Störungsunempfindlichkeit gegenüber Interferenzen und Mehrwegeffekte

(kowoma.de, 2008)

Wenn Radiowellen auf die *Ionosphäre* der Erde treffen, erfahren sie je nach Stärke eine Reflexion oder werden verzögert. Aufgrund des Wellenlängenabstandes von ca. 20 Prozent kann mit Hilfe der gleichzeitigen Auswertungen der zwei Trägerfrequenzen L1 und L2 das Verzögerungsverhalten in der *Ionosphäre* erfasst und korrigiert werden.

2.4.1.1.1 Trägerfrequenzen und gesendete Daten

Jeder Satellit überträgt gleichzeitig zwei im Mikrowellenbereich befindliche Trägerfrequenzen. Diese werden als L1 und L2 bezeichnet und liegen auf dem L-Band, das zwischen 1000 und 2000 MHz funkt und damit im Mittelwellenbereich liegt.

- L1 funkt auf 1575,42 MHz
- L2 funkt auf 1227,60 MHz

Die Wahl der Trägerfrequenzen unterlag mehreren Kriterien (Prinz D. T., GPS in Geowissenschaften, 2009):

- Frequenz muss unter 2000 MHz liegen, da sonst Richtantennen zum Empfang erforderlich sind.
- *Ionosphärische* Verzögerungen sollten so gering wie möglich gehalten werden. Diese sind bei einem Frequenzbereich von kleiner als 100 MHz und größer 10 GHz sehr hoch.
- Die Größe der Bandbreite muss für die zu übertragenden Daten gut gewählt sein.

Zivile GPS-Empfänger nutzen L1-Frequenzen, die sowohl den *C/A-Code (Coarse Acquisition)* und den *P-Code (Precise)* übertragen. Dies wird auch als *Standard Positioning Service (SPS)* bezeichnet. Für militärische Zwecke existiert das *Precise Positioning System (PPS)* welches auf dem L2-Band funkt und nur den P-Code überträgt.

2.4.1.1.2 Codemodulation

Für die Informationsübertragung der C/A- und P-Codes wird eine Folge von zufälligen positiven und negativen Impulsen moduliert, die als Pseudozufallsrauschen oder *Pseudo Random Noise Code (PRN)* bezeichnet werden.

Diese Impulse sind binäre Zeichen und können daher als Bits benannt werden. Sie werden in einer verhältnismäßig großen Länge als determinierte Folge gesendet (Prinz D. T., GPS in Geowissenschaften, 2009). Das übertragende Rauschen, das zufällig wirkt, unterliegt in Wirklichkeit einer Regel, die benutzt wird, um den Datenstrom zu entschlüsseln. Empfangsgeräte kennen diese Regel und nutzen sie, um die Daten zu entschlüsseln. Zusätzlich kann über den *PRN-Code* ein Satellit mit seiner Nummer identifiziert werden.

In der folgenden Abbildung ist zu erkennen, wie die GPS-Sendefrequenzen (L1 und L2) mit dem *PRN-Code* augenscheinlich verschlüsselt beziehungsweise mit der Grundfrequenz von 10,23 MHz, der Atomuhr des Satelliten, moduliert werden.

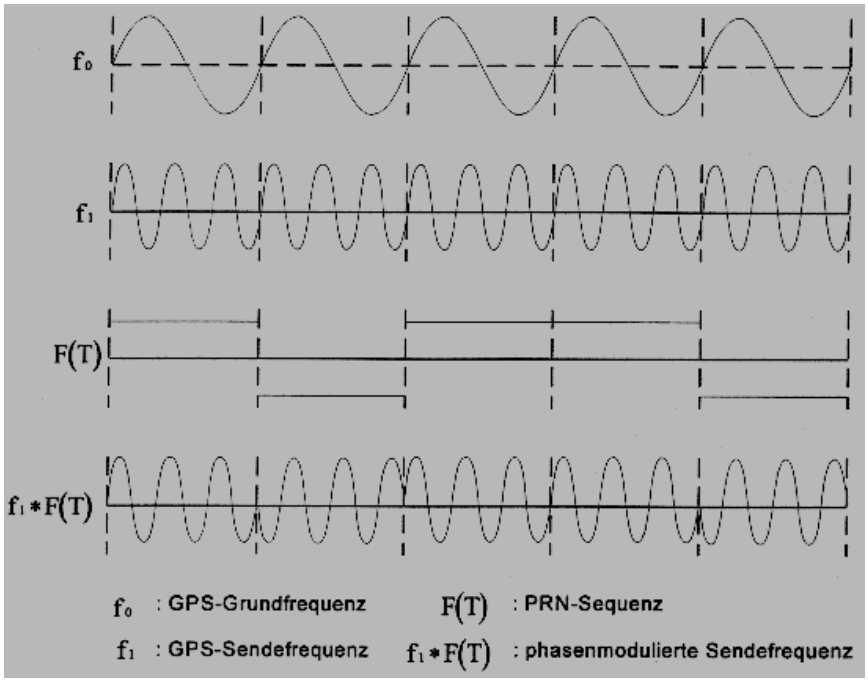

Abbildung 2-10: GPS-Frequenzen (Prinz D. T., 2009)

2.4.1.2 Positionsbestimmung

GPS-Satelliten senden kontinuierlich Informationen wie Position, Umlaufbahn (zuzüglich die der anderer Satelliten), Satellitennummer und die Zeit, wann diese Informationen gesendet wurden, zur Erde.

Ein GPS-Empfänger ist in der Lage, anhand der gesammelten Daten seine Position mit Hilfe von mindestens drei Satelliten zu berechnen.

Für die Ortung vergleicht der GPS-Empfänger als Erstes „*die Zeit, zu der das Signal ausgesandt wurde mit der Zeit, zu der das Signal empfangen wurde*“ (kowoma.de, Bestimmung der Position beim GPS-System - Laufzeitmessung, 2007).
Im zweiten Schritt werden drei weitere Satelliten benötigt, um mit Hilfe des Messverfahrens *Trilateration*, das auf Entfernungs- bzw. Abstandsmessungen beruht, eine genaue Ortung durchzuführen.

Es gilt grundsätzlich zu unterscheiden:

- Zweidimensionale Positionsbestimmung (*2D position fix)*
- Dreidimensionale Positionsbestimmung (*3D position fix)*

Letztere, bei der die Höhe über der Erdoberfläche mit einbezogen wird, kann nur mittels vier oder mehr Satelliten erfolgen.

Eine sehr rudimentäre Positionsbestimmung (Entfernungsbestimmung) funktioniert bereits mit einem Satelliten. Hierbei wird die Differenz zwischen Sende- und Empfangszeit der Pakete ermitteln. So kann anhand der Sendegeschwindigkeit, die im Vakuum 299 792 458 Meter/Sekunde beträgt, die Laufzeit des Signals bzw. den Abstand zwischen Satellit und Empfänger errechnet werden.

Dies geschieht mit folgender Formel:

$$\textbf{\textit{Entfernung}} = \textbf{\textit{Laufzeit}} * \textbf{\textit{Geschwindigkeit}}$$

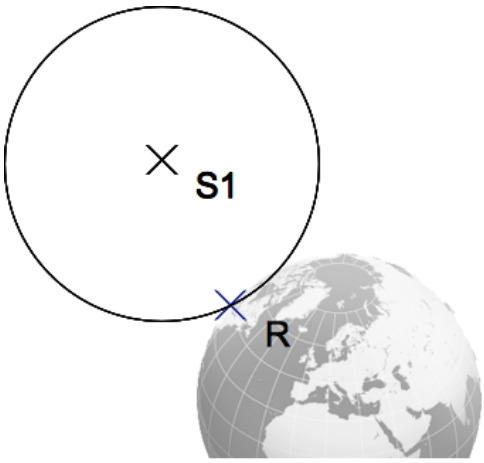

Abbildung 2-11: Positionierung 1 Satellit (Allthingsdistributed)

Da bekannt ist, dass der Empfänger *R* sich auf der Erdoberfläche befindet, können anhand der Entfernung und der Schneidepunkte der beiden Kreise (die Erde wird als zweiter Kreis betrachtet) mindestens zwei mögliche Positionen ermittelt werden. Um diese genau zu berechnen, wird ein weiterer Satellit hinzugezogen und eine erneute Entfernungsmessung durchgeführt.

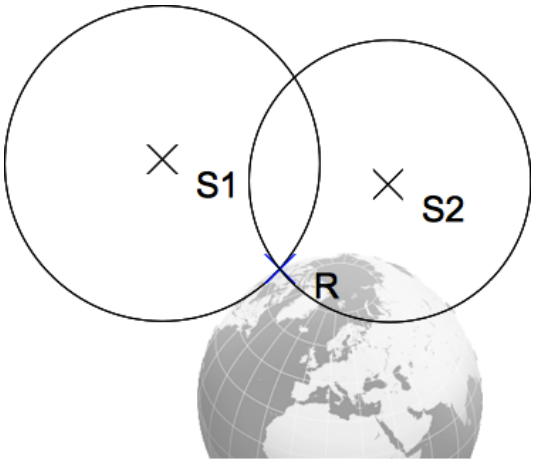

Abbildung 2-12: Positionierung 2 Satelliten (Allthingsdistributed)

Die Position des Empfängers *R* kann mittels der Satelliten *S1* und *S2* genau bestimmt werden. Auf den ersten Blick könnte man meinen, dass keine *Trilateration* durchgeführt wurde.

Auf den zweiten Blick ist jedoch zu erkennen, dass die Erde als dritter Kreis fungiert. Dies ist möglich, da bekannt ist, dass sich der Empfänger auf der Erdoberfläche befindet. Die Berechnung von Höhen kann allerdings nur mit mindestens drei Satelliten durchgeführt werden.

In der Praxis stößt diese optimistische Berechnung an ihre Grenzen, da die GPS-Empfänger aus Kostengründen keine Atomuhren besitzen. Somit sind die Uhren der Satelliten mit den Uhren der Empfänger nicht synchron. Anhand der Zeitstempel der gesendeten Daten kann die Entfernung zwischen Empfänger und Satellit errechnet werden. Durch die Übermittelung der Daten entstehen jedoch Latenzen, die Ungenauigkeiten der Ortung verursachen. Folgende Illustration verdeutlicht den Sachverhalt:

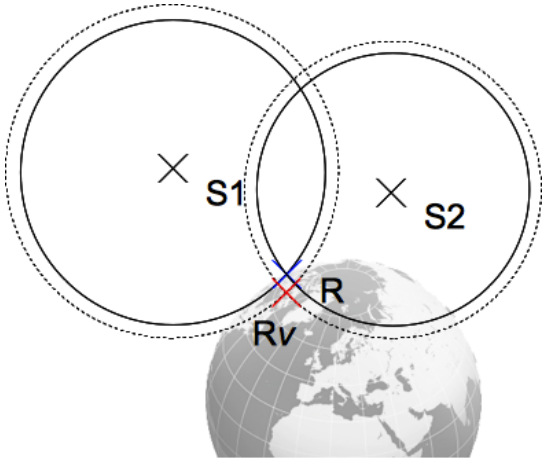

Abbildung 2-13: Positionierung 2 Satelliten mit Latenz (Allthingsdistributed)

Aufgrund der Asynchronität zwischen der Uhr im Satelliten und der im GPS-Empfänger kommt es zu Berechnungsfehlern der Entfernungen. Die mit gestrichelten Linien dargestellten Kreise, werden als Pseudoentfernungen (oder *Pseudoranges*) bezeichnet. Diese ermitteln sich wie folgt:

$$\textbf{\textit{Pseudoentfernung}} = (\textbf{\textit{Laufzeit}} + \textbf{\textit{Uhrenfehler}}) * \textbf{\textit{Geschwindigkeit}}$$

Der Punkt *Rv* hebt die fehlerhafte berechnete Position hervor. Es ist deutlich zu erkennen, dass die Asynchronität der Uhren eine unkorrekte Positionsbestimmung zur Folge hat. Bei bereits einem Uhrenfehler von „*1/100 Sekunde* […], *macht die GPS-Navigation eine Fehlbestimmung der Position um ca. 3000 km aus*“ (kowoma.de, Bestimmung der Position beim GPS-System - Laufzeitmessung, 2007).

Mit Hilfe eines dritten Satelliten können die Pseudoentfernungen korrigiert werden. Betrachten wir nun diesen Fall erneut und errechnen zuerst die Entfernung unter der Voraussetzung, dass alle Uhren synchron sind. Danach werden die Pseudoentfernungen ermittelt. Zu erkennen ist die Problematik, dass drei Schnittpunkte *Rv* das Ergebnis der Ortung sind und somit keinen eindeutigen Punkt ergeben.

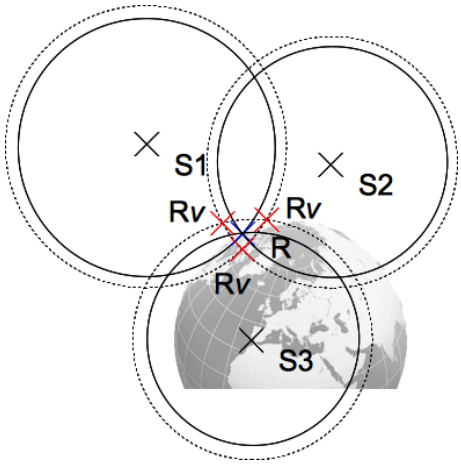

Abbildung 2-14: Positionierung 3 Satelliten mit Latenz (Allthingsdistributed)

Durch Verschieben der Zeit der Empfängeruhr kann diesem Problem entgegengewirkt werden. Die Uhr des GPS-Empfängers synchronisiert sich anhand dieser Methode mit der Atomuhr des Satelliten, um eine Ortungsgenauigkeit von 10 Metern zu erreichen.

Bei der dreidimensionalen Positionsbestimmung wird ein weiterer Satellit hinzugezogen. Dieser ist nötig, um neben Länge und Breite die Höhe bestimmen zu können. Bei der zweidimensionalen Positionsbestimmung wurde davon ausgegangen, dass sich der GPS-Empfänger auf der Erdoberfläche auf Höhe des Meeresspiegels befindet. Mit Hilfe des vierten Satelliten kann zusätzlich eine genaue Ortung auf Bergen oder tiefen Orten der Erde stattfinden. Dies geschieht ähnlich wie bei der zweidimensionalen Positionsbestimmung.

2.4.1.3 Genauigkeit und Fehlerquellen

Die erreichbare Genauigkeit von GPS kann nicht pauschal angegeben werden. Verschiedene Faktoren hängen von der Ortungsschärfe ab. Zum einen die seit dem Jahr 2000 abgeschaltete *Selective Availability*, wodurch GPS-Empfänger künstlich ungenaue Informationen empfingen. Zum anderen die Satellitengeometrie, die besonders ins Gewicht fällt, wenn sich Satelliten in „schlechter" Geometrie, also z.B. sehr horizontal zur eigenen Position, befinden. Der meist in Großstädten auftretende *Mehrwegeeffekt (Multipath)*, bei dem durch Reflexion an Gebäuden oder Erhöhungen die empfangenen Signale verzögert werden, führt auch zu Ungenauigkeiten bei der Positionsbestimmung. Zudem kommen weitere Faktoren hinzu, wie die bereits erwähnten atmosphärischen Effekte (*ionosphärische* Verzögerungen) oder die bei der Positionsbestimmung auftretende Uhrenungenauigkeit.

Folgende Tabelle zeigt einen Überblick über Störungsfaktoren und ihren Einfluss auf die Genauigkeit der Ortung.

Faktor	Einfluss
Störungen durch die Ionosphäre	ca. 5 Meter
Schwankungen der Satellitenumlaufbahnen	um 2,5 Meter
Uhrenfehler der Satelliten	etwa 2 Meter
Mehrwegeeffekt	ungefähr 1 Meter
Störungen durch die Troposphäre	ca. 0,5 Meter
Rechnungs- und Rundungsfehler	rund 1 Meter

Tabelle 2-6: Faktoren für GPS-Fehlerquellen (kowoma.de, 2008)

In den meisten Fällen ist jedoch eine Ortungsgenauigkeit von ca. 10 Metern zu erwarten.

2.4.1.3.1 Fehlerkorrektursysteme

Drei Fehlerkorrektursysteme erweitern GPS und verbessern die Genauigkeit der Positionsbestimmung. Um fehlerhafte Signale für die Positionierung ausschließend zu können, kann mittels RAIM[28] eine Integritätsprüfung der Signale durchgeführt werden. Das Differential Global Positioning System[29] ermöglicht eine signifikante Verbesserung der Positionsbestimmung, in dem landgestützte Referenzstationen Korrektursignale zurück an den GPS-Empfänger senden. Das dritte System besteht aus 25 Bodenstationen, die GPS-Signale überwachen und zwei an den Küsten der USA befindliche Referenzstationen, die diese Signale auswerten und Korrekturdaten bereitstellen. Es wird als Wide Area Augmentation

[28] Receiver Autonomous Integrity Monitoring

[29] Akk. D-GPS

System bezeichnet[30] und überträgt im Gegensatz zu D-GPS die Korrekturdaten an zwei geostationäre Satelliten.

Auf anderen Kontinenten sind ähnliche Systeme im Einsatz. In Europa das *EGNOS (Euro Geostationary Navigation Overlay Service - Europäischer Geostationärer Zusatz-Navigationsdienst)* und im asiatischen Raum das japanisches System namens *MSAS (Multi-Functional Satellite Augmentation System)*. Im Aufbau befindet sich das indische *GAGAN (GPS Aided Geo Augmented Navigation)*.

Folgende Tabelle gibt einen Überblick über die zu erwartenden Genauigkeiten der genannten Systeme.

2.4.1.3.2 Übersicht über die zu erwartende Genauigkeit

System	Genauigkeit
Ursprüngliches GPS-System mit aktivierter SA[31]	ca. 100 Meter
Typische Positionsgenauigkeit ohne SA	ungefähr 15 Meter
Typische Differential-GPS(DGPS)-Genauigkeit	etwa 3 bis 5 Meter
Typische Genauigkeit mit aktiviertem WAAS/EGNOS	rund 1 bis 3 Meter

Tabelle 2-7: Übersicht von Genauigkeiten GPS-Systeme (kowoma.de, 2007)

2.4.1.4 A-GPS

Beim *Assisted Global Positioning System* (kurz *A-GPS*) werden weitere Quellen wie z.B. das Mobilfunknetz zur schnelleren Standortbestimmung herangezogen.

Das Problem beim herkömmlichen GPS ist, dass es für den unregelmäßigen kurzen Gebrauch nicht konzipiert worden war. Speziell die erste Positionsbestimmung ist zeitaufwändig. Sollte das Gerät mehrere Tage keine Ortung mehr durchgeführt haben, müssen die Bahndaten der Satelliten erst aktualisiert werden. Da dies zeitaufwändig ist, wird beim *A-GPS* auf die Funkzellen des Telefonnetzes zurückgegriffen.

[30] Abk. WAAS

[31] *Selective Availability*

2.4.1.4.1 Funktionsweise

Die Lokalisierung eines im *GSM(Global System for Mobile Communications)*-Netz eingebuchten Telefons mittels Funkzelle ist zwar ungenauer als GPS, dennoch im Zweifel schneller, da die nötigen Parameter zur Ortung bereits bekannt sind. Die Ortung findet mittels dreier Funkzellen statt. Durch Messungen der Signallaufzeiten ist, ähnlich wie bei der GPS-Ortung, eine Positionierung möglich. Die Höheninformationen können dabei nicht erhoben werden. Diese ermittelte Position wird nun verwendet, um den Suchbereich für GPS-Satelliten einzugrenzen. Mittels des Almanachs, einer Liste mit Bahndaten von GPS-Satelliten, und der groben Position des GPS-Empfängers kann gezielt in einem Himmelsabschnitt nach Satelliten gesucht werden. Durch diese Eingrenzung ist eine schnellere und flexiblere Methode geschaffen worden, die in den meisten Fällen die Ortung im Alltag komfortabler gestaltet.

2.4.2 Galileo

Das Projekt der Europäischen Union zur satellitenbasierten Positionsbestimmung ähnelt im Aufbau dem bereits behandelten US-amerikanischen GPS und wurde primär für den zivilen Einsatz konzipiert.

Das im Jahre 2007 neuausgeschriebene Projekt (Spiegel Online, 2007) wird zusammen mit der Europäischen Weltraumorganisation (ESA) durchgeführt und soll voraussichtlich im Jahre 2014 (euronews, 2010) starten.

Galileo wird auf 27 Satelliten (zuzüglich drei als Ersatz) basieren, welche von verschiedenen Kontroll- und Wartungsstationen auf der Erde überwacht werden. Ähnlich wie beim GPS, allerdings ein wenig höher, werden die Satelliten ca. 23.260 km entfernt von der Erde, kreisen. Die Genauigkeit der Ortung beläuft sich auf vier Metern. Zusätzliche Faktoren wie ein Mobilfunknetz können die Präzision der Positionsbestimmung erhöhen.

Derzeit sind zwei Galileo-Satelliten aktiv und in der Lage, mit dem US-amerikanischen GPS III zusammenzuarbeiten (ainonline.com, 2009). In den nächsten Jahren werden weitere Satelliten der ESA und EU folgen, so dass in Zukunft auch eine Ortung ohne GPS möglich sein wird.

In dem nächsten Kapitel werden Anwendungsfälle definiert, in denen GPS eine wichtige Rolle für die Positionierung einnehmen kann. Denn OpenBeacon steht als weiteres Verfahren zur Diskussion. Es werden beide Technologien evaluiert, um zu ergründen, welche in den weiteren Kapiteln der Arbeit Verwendung finden wird.

3 Anforderungsanalyse

Dieses Kapitel behandelt die Anwendungsfälle des intelligenten Fahrauskunftssystems und deren technische sowie konzeptionelle Hintergründe. Anschließend werden Kommunikationstechnologien auf Basis von Messungen und gegebenen Einschränkungen evaluiert, um abschließend die Anforderungen an das Gesamtsystem definieren zu können.

3.1 Anwendungsfälle

In diesem Abschnitt werden drei Szenarien beschrieben, die zum einen grundlegende Funktionalität des intelligenten Fahrauskunftssystems beschreiben und zum anderen Fehlerfälle mit einbeziehen und berücksichtigen.

Die Szenarien werden detailliert beschrieben und anschließend mittels *UML(Unified Modeling Language)-Use-Case*-Diagramms[32] visualisiert.

3.1.1 Szenario 1: Fahrgast stellt Anfrage und benutzt vom System errechnetes ÖPNV-Mittel.

Dieses Szenario beschreibt den Hauptanwendungsfall des Systems: Der User stellt eine Verbindungsanfrage und wird nur durch Benachrichtigungen zum Ziel geführt. Detailliert geschieht dies wie folgt:

Der Nutzer des Dienstes verbindet sich über eine Webseite zum Fahrauskunftssystem. Dort gibt er seine RFID-Identifikationsnummer an und den Zielbahnhof, zu dem er geleitet werden möchte. Zuerst verbindet sich die Webseite zu einem Fahrinformationssystem, um eine Strecke zum Zielbahnhof zu erhalten. Aus diesen Informationen werden der Endbahnhof und die Liniennummer über die Webseite zum Server übertragen. Der Server speichert die Client- und Verbindungsinformationen zwischen.

Ein ÖPNV-Mittel sendet kontinuierlich alle in Reichweite befindlichen User-Identifikationen an den Server, der anschließend prüft, ob diese zwischengespeichert wurden und ob sie auf das ÖPNV-Mittel warten, das sich momentan in Reichweite befindet. Ist dies der Fall, sendet der Server eine Benachrichtigung an den Nutzer, der anschließend das ÖPNV-Mittel benutzt.

Abschließend löscht der Server den Eintrag der Verbindungsanfrage, da davon ausgegangen wird, dass der Nutzer das ÖPNV-Mittel schon benutzt und bereits auf dem Weg ist.

[32] Anwendungsfalldiagramm

3.1.2 Szenario 2: Fahrgast muss umsteigen, um den Zielbahnhof erreichen zu können.

Der zweite Anwendungsfall thematisiert die Problematik des Aus- und Umsteigens. In den meisten Fällen wird der Zielbahnhof des Fahrgastes nicht auf der Strecke des erstbestiegenen ÖPNV-Mittels liegen. Das heißt, der Fahrgast muss aus dem ÖPNV-Mittel aussteigen und in ein anderes wechseln. Sehr oft kommt es, speziell in Großstädten, vor, dass eine längere Wegstrecke (mehr als 300 Meter) zur nächsten Station zurückgelegt werden muss. An dieser Stelle wird der Fahrgast zur nächsten Station bzw. Haltestelle navigiert. Ist der Fahrgast an der nächsten Station angekommen, wird nach dem im Szenario 1 beschriebenen Prinzip fortgefahren und der Fahrgast benachrichtigt, wenn sich das vom Fahrplandienst errechnete ÖPNV-Mittel nähert.

Diesem Szenario stellen sich einige Probleme in den Weg. Zum einen muss der Server alle Umsteigepunkte speichern und stets prüfen, ob der Fahrgast bereits an diesen angekommen ist. Zum anderen besteht die Schwierigkeit, die Navigation unkompliziert für den Fahrgast zu gestalten.

Eine Lösung dieser Probleme könnte mit der Kombination von RFID und GPS realisiert werden. Wenn der Fahrgast in ein ÖPNV-Mittel steigt, beobachtet der Server die Position des ÖPNV-Mittels, da bekannt ist, dass der Fahrgast sich in diesem befindet. Die an den Server übermittelten Positionsdaten werden ausgewertet und es wird geprüft, ob der Fahrgast sich dem Umsteigepunkt nähert. Ist dies der Fall, benachrichtigt der Server den Client darüber, dass der Umstieg bevorsteht. Zusätzlich können Positionsdaten des Ausstiegspunktes und der nächsten Haltestelle bzw. des nächsten Bahnsteigs übermittelt werden, die der Client auf einer Karte mit passender Wegbeschreibung angezeigt bekommt. Da eine Wegbeschreibung auf einer Karte innerhalb von Bahnhöfen suboptimal ist, könnte mittels Angabe von Gleisnummer eine kartenlose Wegbeschreibung erreicht werden.

Aufgrund der Komplexität dieses Problems ist das „Szenario 2: Fahrgast muss umsteigen, um den Zielbahnhof erreichen zu können.“ in dieser Arbeit nicht berücksichtigt. Außerdem wird die Fähigkeit des Clients, Fahrinformationen von dritten ÖPNV-Betrieben zu erhalten, nicht implementiert.

3.1.3 Szenario 3: Fahrgast stellt Anfrage, wartet aber nicht, sondern entfernt sich vom Wartepunkt.

Das dritte Szenario beschäftigt sich mit der Frage, was passiert, wenn ein Nutzer sich für eine Fahrauskunft angemeldet hat, sich aber dann vom Wartepunkt (Bahnsteig, Haltestelle, etc.) entfernt.

Jede Anmeldung eines Nutzers wird mit einem Zeitstempel versehen. Wenn innerhalb einer bestimmten Zeit (z.B. zwei Stunden) die Nutzeridentifikation nicht in Reichweite eines Verkehrsmittels war, wird der Eintrag auf dem Server automatisch gelöscht. Dem Nutzer ist es jedoch freigestellt, eine weitere Fahrauskunftsanfrage zu stellen.

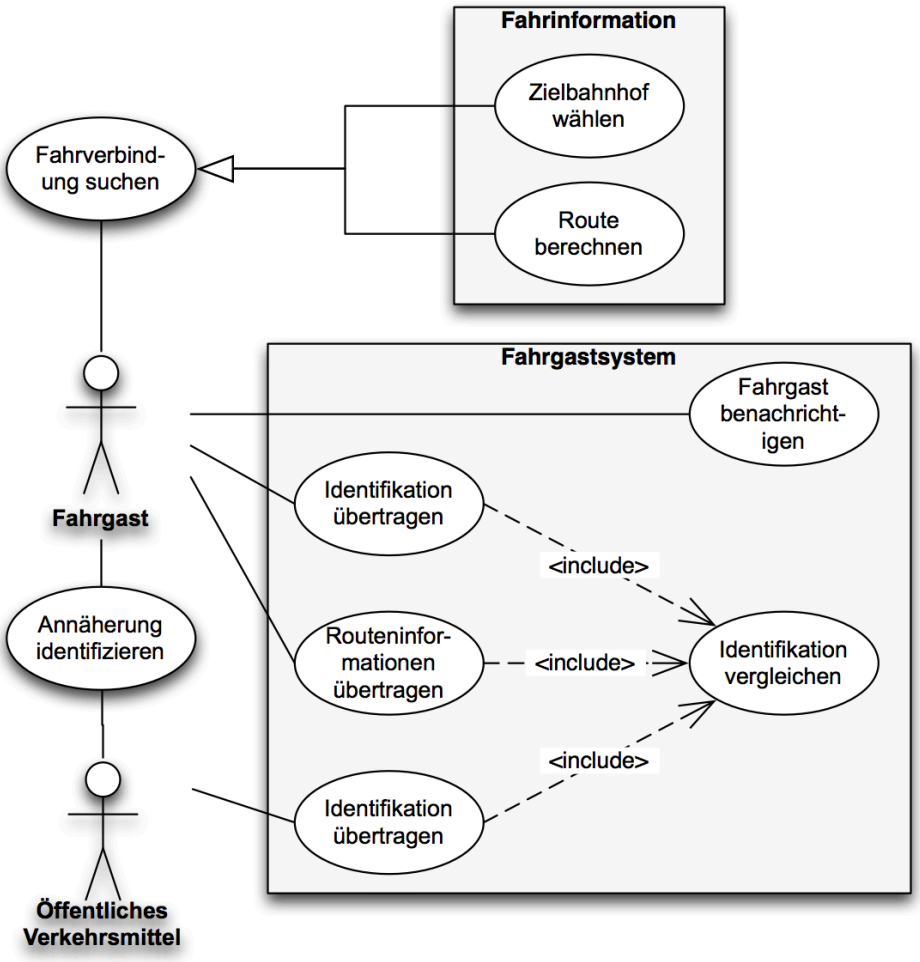

Abbildung 3-1: Anwendungsfalldiagramm Szenario 1 und 2

3.2 Rahmenbedingungen für die Entwicklung des Prototypen

Für die Rahmenbedingungen der Entwicklung einer prototypischen Client-Server-Implementierung sind folgende Kriterien relevant:

- Änderbarkeit und Erweiterbarkeit
  - Modularisierung der Funktionalitäten
  - Dokumentation der Schnittstellen (*Interfaces*)
- Benutzbarkeit
  - Gestaltung einer klar strukturierten Benutzeroberfläche
- Effizienz
  - Netzwerkkommunikation mit geringem Overhead
- Funktionalität
  - Einsatzfähigkeit für gängigen ÖPNV[33] (Schienenbahn, Omnibus, Fähre)
  - Plattformunabhängig
- Wiederverwendbarkeit
  - Objektorientierte Programmierung
  - Abgrenzung von Funktionalitäten in Klassen
  - Plattformunabhängigkeit durch portablen, interpretierbaren Quellcode
- Zuverlässigkeit

Anhand dieser Kriterien gilt es, bei der Implementierung eine geeignete Programmiersprache und -umgebung zu wählen.

[33] Öffentlicher Personennahverkehr

3.3 Leistungsanforderungen an Hard- und Software

Die Anforderungen an die Kommunikationsinfrastruktur zwischen ÖPNV und Server, ÖPNV und User sowie Server und User stellen eine Herausforderung dar. Hierfür werden zwei Systeme zur Kommunikation bzw. zur Ortung der Komponenten evaluiert.

3.3.1 Positionierung durch GPS

Bei Lokalisierung des Users und des ÖPNV-Mittels über GPS bzw. A-GPS werden die berechneten Positionsdaten (*Latitude*[34] und *Longitude*[35]) an den Server übermittelt, der diese auswertet und so die Entfernung der beiden Akteure ermitteln kann.

Dieses Übermitteln der Positionsdaten führt das ÖPNV-Mittel kontinuierlich durch. Der Server, der die Daten empfängt, kann diese jedoch nicht direkt verarbeiten. Aufgrund von Verzögerungen im Sendevorgang besteht die Problematik, dass das ÖPNV-Mittel sich bereits an einer anderen Position befindet, als der Server annimmt. Um dieses Problem zu lösen, müssen Einflussfaktoren auf den Sendevorgang beachtet und in die Auswertung der Position einbezogen werden. Die Differenz der aktuellen Serverzeit und der Zeit des ÖPNV-Mittels bei der Versendung der Position müssen mit der Durchschnittsgeschwindigkeit summiert werden, um eine fehlerbereinigte Position des ÖPNV-Mittels bestimmen zu können.

	Position ÖPNV
+	Server Zeit
-	ÖPNV-Zeit
+	ÖPNV-Durchschnittsgeschwindigkeit
=	Aktualisierte Position ÖPNV

Anhand der berechneten Position des ÖPNV-Mittels und der Position des Users kann der Server den Abstand zwischen ÖPNV und User errechnen und bei Annäherung den Benutzer informieren.

[34] geographische Breite

[35] geographische Länge

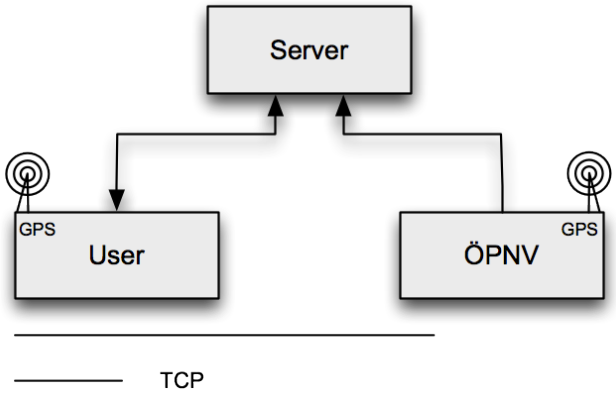

Abbildung 3-2: Aufbau Kommunikationsstruktur GPS

3.3.1.1 Problematik

Im Rahmen dieser Arbeit durchgeführte Messungen haben ergeben, dass eine genaue Ortung per GPS innerhalb des U-Bahn-Bereiches nicht möglich ist. Eine Positionierung mittels A-GPS könnte in Betracht gezogen werden. An dieser Stelle tritt jedoch ein zweites Problem auf, denn in der Berliner U-Bahn sind unterirdisch Funkzellen angebracht, die als *Repeater* oberirdischer Funkzellen dienen. Somit ortet sich das Mobilgerät in dem Gebiet, in dem die oberirdische Funkzelle lokalisiert ist. In einem Experiment, indem eine Positionierungsanfrage im U-Bahnhof „Lipschitzallee“ der Berliner U-Bahnlinie 7 durchgeführt wurde, zeigte sich, dass im Ergebnis der U-Bahnhof „Zwickauer Damm“ als Position errechnet wurden ist. Dies deutet darauf hin, dass dort eine oberirdische Funkzelle angebracht ist. Die Größe des hellblauen Kreises dokumentiert die hohe Ungenauigkeit der Ortungsbestimmung.

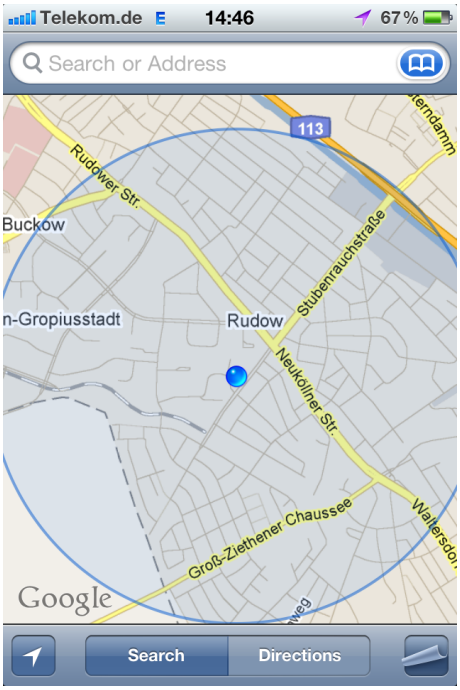

Abbildung 3-3: Positionsbestimmung mittels Google Maps auf einem iPhone

In Städten wie München ist gar keine Ortung möglich, da keine GSM-Netze innerhalb des U-Bahn-Bereiches existieren.

3.3.1.2 Lösungsansatz

Aufgrund der Problematik im U-Bahn-Bereich kann die GPS-Technologie nicht flächendeckend für alle ÖPNV-Mittel genutzt werden.

Es bietet sich jedoch eine Hybridlösung aus RFID für den U-Bahn-Bereich und GPS für Bus, Tram, S-Bahn, etc. an.

3.3.2 Positionierung durch OpenBeacon

Der User kommuniziert beim Anmeldevorgang bidirektional mit dem Server. Unter anderem werden dabei die RFID-Tag-ID und Identifikationsnummer des ÖPNV-Mittels übertragen. Die User – ÖPNV-Mittel-Identifizierung geschieht über direkten Kontakt via RFID. Die RFID-Antenne des ÖPNV-Mittels empfängt die in Reichweite befindlichen aktiven RFID-Tags des Users und sendet diese unmittelbar zum Server. Dieser vergleicht die empfangenen Identifikationsnummern mit den bereits vom User registrierten. Ist eine Identifikationsnummern auf ein ÖPNV-Mittel registriert und der RFID-Tag in Reichweite des ÖPNV-Mittels, dann benachrichtigt der Server den User.

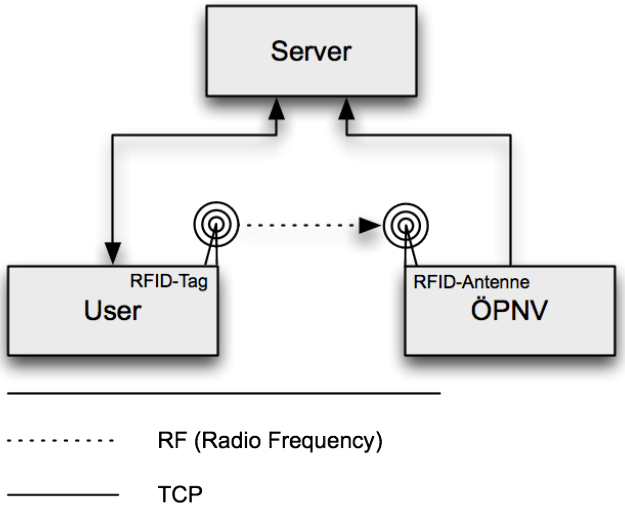

Abbildung 3-4: Aufbau Kommunikationsstruktur RFID

3.3.2.1 Problematik

Im Rahmen dieser Arbeit durchgeführte Messungen haben ergeben, dass die Reichweite der aktiven RFID-Tags auf ca. 8–10 Meter beschränkt ist. Da der menschliche Körper aufgrund seines hohen Wassergehaltes Wellen (elektromagnetische Wellen) absorbiert, kommt es bei größeren Menschenmengen zu geringeren Reichweiten. Die mit sechs Personen durchgeführten Messungen haben gezeigt, dass wenn alle Personen zwischen RFID-Tag und RFID-Antenne stehen, die Reichweite des Signals auf 4–6 Meter absingt.

3.3.2.2 Lösungsansatz

Die Reichweite von 4–6 Metern ist prinzipiell ausreichend, um eine Identifikation zwischen ÖPNV-Mittel und User durchführen zu können. Sehr wichtig ist jedoch die Geschwindigkeit, mit der die Identifikation des RFID-Tags, die Auswertung auf dem Server und die Benachrichtigung des Users stattfinden muss.

3.3.3 Resultierende Anforderungen

Die Tatsache, dass die Identifizierung zwischen User und ÖPNV-Mittel mit RFID-Technologie universell für alle Arten des ÖPNVs einsetzbar ist, macht diese Technologie zum geeigneten Mittel für die Konzeption und Implementierung des dynamischen Fahrauskunftssystems. Daraus resultierende Anforderungen werden folgend speziell für das ÖPNV-Mittel und allgemein beschrieben.

3.3.3.1 Öffentliche Personennahverkehrsmittel

Als Voraussetzung für den ordnungsgemäßen Betrieb benötigt das ÖPNV-Mittel einen Computer, softwareseitig den im ROBERTA-System befindlichen Proximity-Server sowie den Journey-Client. Der Proximity-Server wurde im Rahmen der Bachelorarbeit von Stephan Bergemann entwickelt und dient als Software-Schnittstelle zur Hardware des OpenBeacon-Projekts (Bergemann, 2010).

Auf der Hardwareseite wird eine OpenBeacon-RFID-Antenne benötigt, die über einen *PoE*-Switch oder WLAN-Router mit dem Computer verbunden ist. Für die Kommunikation zum Server erfordert der Computer einen Internetzugriff oder minimal eine Verbindung zum Server.

3.3.3.2 Allgemein

	User	Server	ÖPNV
Hardware	• Zugang zum Internet mittels verbautem GSM[36]-, UMTS[37]- oder CDMA[38]-Modul • Für präzise Ortung GPS-Modul • RFID-Tag (Transponder)	• Verbindung mittels Ethernet zum Internet	• Verbindung zum Internet • Mindestens eine RFID-Antenne
Software	• Browser, der HTML5-Features, Geolocation[39], WebSockets[40] sowie JavaScript unterstützt	• Webserver mit aktiviertem PHP-Modul • Journey-Software	• Qt-Framework • Proximity-Server • Journey-Software

Tabelle 3-1: Allgemeine Hard- und Software-Anforderungen

[36] Global System for Mobile Communications

[37] Universal Mobile Telecommunications System

[38] Code Division Multiple Access

[39] Funktion zur Ermittlung von geographischen Ortungsinformationen

[40] Schnittstelle zur bidirektionalen Verbindung mit Socket-Server oder WebSocket-Server

4 Entwurf

Die Zielstellung des Entwurfs ist, eine System- und Softwarearchitektur zu konzeptionieren, die Hardware zur Kommunikation abstrahiert, sodass diese ohne Anpassung der Softwarearchitektur ausgetauscht werden kann. Um dies zu realisieren, sind die Komponenten modular aufgebaut und mit Schnittstellen versehen, sowie die Systemarchitektur als Drei-Schichten-Architektur[41] konzipiert.

In diesem Kapitel werden Softwareschicht, Datenmodell und das verbindende Protokoll erläutert.

4.1 Konzeption der Softwarearchitektur

Die Architektur des Gesamtsystems unterteilt sich in drei Hauptkomponenten.

- User-Komponente
- Server-Komponente
- ÖPNV-Komponente

Die User-Komponente besteht aus dem Mobilgerät und dem dazugehörigen RFID-Tag. Die Server-Komponente, die einen Webserver und den Journey-Server enthält, arbeitet als Mittelsmann und implementiert die Geschäftslogik. Zum einen bietet die vom Webserver ausgelieferte Webseite die Schnittstelle zur User-Komponente und zum anderen fungiert der Journey-Server als *Socket*-Server, der gemäß dem Protokollentwurf spezifizierte Pakete erwartet. Das ROBERTA-System liefert die RFID-Funktionalität, ist innerhalb der ÖPNV-Komponente gekapselt und steht unidirektional mit der User- und Server-Komponente in Verbindung. Dies geschieht über definierte Schnittstellen und gewährleistet eine saubere Trennung der Komponenten.

[41] Englisch: three tier architecture

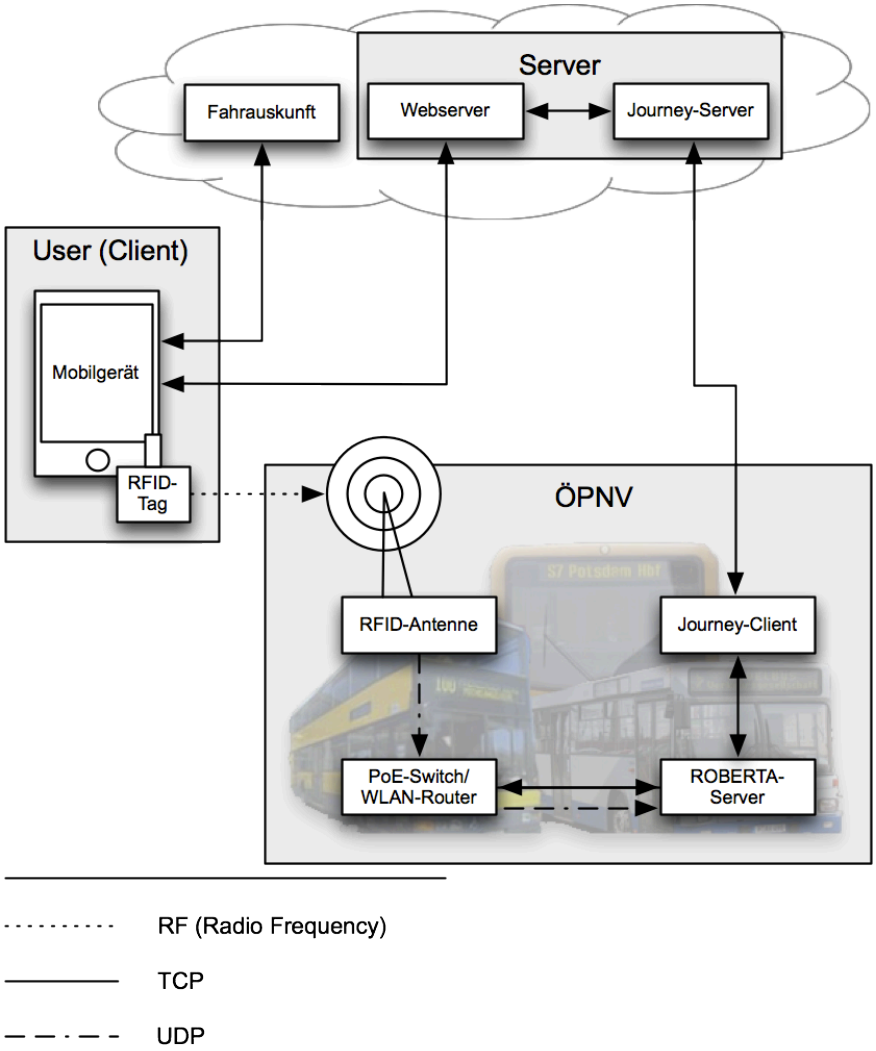

Abbildung 4-1: Aufbau Gesamtsystem (Stahnsdorf.de, 2010)

4.2 Entwurfsmuster

Um flexibel Änderungen und Erweiterungen der Komponenten oder deren Austausch zu ermöglichen, wird die Software dem Model-View-Control-Entwurfsmuster (Modell-Präsentation-Steuerung) entsprechend umgesetzt. Wie auf der Abbildung 4-2: Entwurfsmuster ersichtlich, ist die Steuerungs-Komponente durch den Server abgebildet. Dieser kontrolliert den Datenfluss zwischen dem Client (User-Komponente) und dem ÖPNV-Mittel. Die User-Komponente stellt die Präsentationsschicht dar. Diese kann mit der Modell-Schicht unidirektional kommunizieren und ist so in der Lage Daten, wie in diesem Fall die RFID-Tag-ID, an das Modell übermitteln zu können.

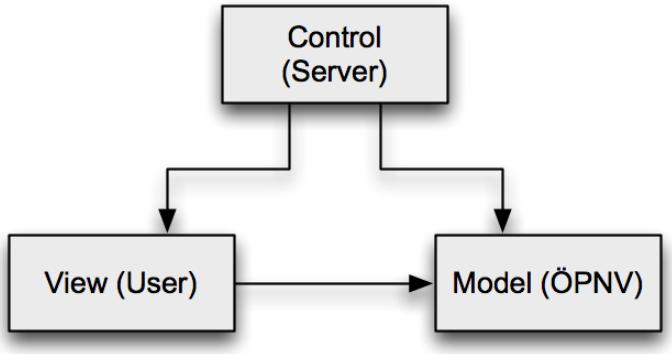

Abbildung 4-2: Entwurfsmuster

4.3 Datenmodell

Mittels des *Entity-Relationship-Modells*[42] sollen die Daten und deren Relationen zueinander, die innerhalb der Komponenten gespeichert sind, beschrieben werden. Für die Objekt-Notation wird IDEF1X als Modellierungssprache verwendet, wobei jedoch die Relations-Notation nicht der Modellierungssprache folgt, sondern eine individuelle Notation verwendet. Diese hebt besonders die Primär- und Fremdschlüssel hervor sowie die Eigenschaften der Relationen.

Die folgenden Datenmodelle sind auf die Anforderungen der verschiedenen Komponenten angepasst und entsprechend aufgeführt. Zusätzlich zum *ERM* werden Attribute der Tabelle beschrieben.

4.3.1 User-Komponente

Der Benutzer der User-Komponente soll der Webseite seine RFID-Tag-Identifikationsnummer übergeben können. Diese Identifikationsnummer wird lokal auf dem Client gespeichert. Dies hat den Vorteil, dass der Benutzer die Kontrolle über diese Information behält und nicht bei jedem Besuch der Webseite aufgefordert wird, die Nummer neu einzugeben.

Tabelle	Attribut	Beschreibung
UserData	TagID	*„Die TagID ermöglicht eine 1:1-Zuordnung eines Tags“* zu einem Client (Bergemann, 2010).

Tabelle 4-1: User-Komponente – Beschreibung des Datenmodells

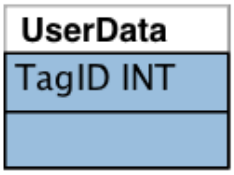

Abbildung 4-3: ERM User-Komponente

[42] Abk. ERM

4.3.2 Server-Komponente

Die Schwierigkeit bei dem Entwurf des Datenmodells bestand in der Eingrenzung auf die potentiellen Funktionen. Zusätzlich muss es schlank konzipiert sein, damit Abfragen in geringer Zeit durchlaufen können. Das Datenmodell, wie es in Abbildung 4-4 visualisiert ist, spezialisiert sich auf die Kernfunktionen der Anwendungsfälle. Das bedeutet, dass es prinzipiell möglich wäre, Fuhrpark oder Linieninformationen mit weitergehenden Daten abzubilden. Dies würde vermutlich in einem Datenmodell enden, das viele Aspekte beachtet, jedoch nicht mehr auf Kernfunktionalität und Rahmenbedingungen Rücksicht nimmt.

Tabelle	Attribut	Beschreibung
Linie	ID	Identifikationsnummer zur eindeutigen Zuweisung eines Datensatzes
	LinienNr	Die LinienNr beschreibt die vom Verkehrsunternehmen vergebene Linienbezeichnung.
	StationStart	Startstation des ÖPNV-Mittels
	StationEnde	Endstation des ÖPNV-Mittels
PublicTransport	PublicTransportID	Eigenständige Identifikationsnummer des ÖPNV-Mittels
PublicTransportType	PublicTransportTypeID	Identifikationsnummer des Typs
	Type	Das Attribut „Typ“ beschreibt die Art des ÖPNV-Mittels, zum Beispiel Bus, U-Bahn, Tram, etc..
User	UserID	Jeder verbundene Client bekommt eine Identifikationsnummer zugewiesen, anhand derer er eindeutig identifiziert werden kann. Diese Nummer ist serverseitig generiert, um ausschließen zu können, dass durch einen manipulierten Clientidentifikator die Zuordnung der verbundenen Clients Inkonsistenzen in der Datenbank hervorrufen kann.

	IP	Die IP-Adresse dient als Basis zur Kommunikation mit dem Client auf TCP-Ebene.
	TagID	*„Die TagID ermöglicht eine 1:1-Zuordnung eines Tags“* zu einem Client (Bergemann, 2010).
	Endbhf	Endbahnhof, den der Benutzer im Client angegeben hat
	TIMESTAMP	Zeitstempel, mit dem Einträge zeitlich zugeordnet werden können. Einträge, die älter als z.B. ein Tag sind, können gelöscht werden, da der Fahrgast vermutlich nicht mehr wartet.

Tabelle 4-2: Server-Komponente – Beschreibung des Datenmodells

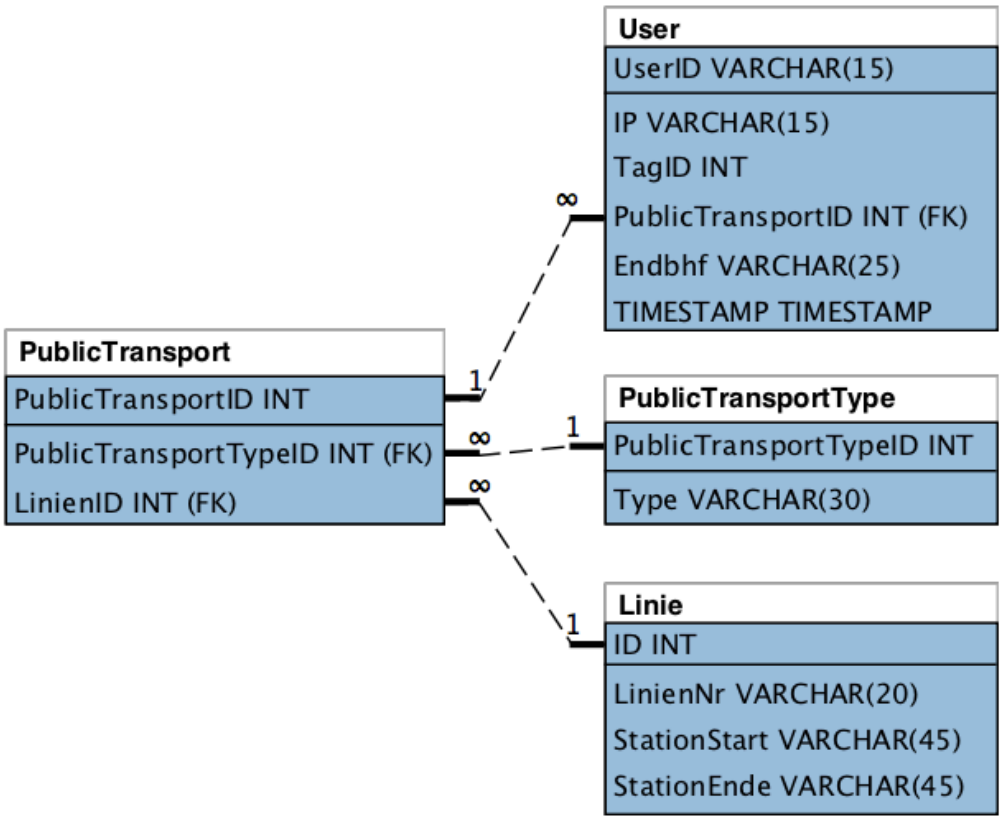

Abbildung 4-4: ERM Server-Komponente

4.3.3 ÖPNV-Komponente

Zur Identifikation des ÖPNV-Mittels muss der Fahrer bei Fahrtantritt die Endstation eingeben. Die Identifikationsnummer des Fahrzeugs (Zug, etc.) ist fest definiert und wird in der Tabelle mit der Endstation ergänzt.

Tabelle	Attribut	Beschreibung
PublicTransportLine	PublicTransportID	Eigenständige Identifikationsnummer des ÖPNV-Mittels.
	StationEnde	Endstation des ÖPNV-Mittels.

Tabelle 4-3: ÖPNV-Komponente – Beschreibung des Datenmodells

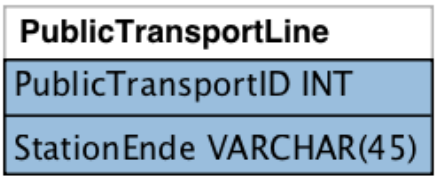

Abbildung 4-5: ERM ÖPNV-Komponente

4.4 Protokollentwurf

Die Konzeption des Protokolls soll die Anforderungen bei der Kommunikation zwischen den Komponenten abbilden. Jede von der RFID-Antenne empfangene RFID-Tag-Identifikationsnummer wird an den Journey-Server zur Auswertung gesendet. Kommt die Auswertung zum Ergebnis, dass ein bestimmter RFID-Tag bereits für ein Verkehrsmittel angemeldet ist, dann ist der Client – das Mobilgerät – zu benachrichtigen. Daher ist es wichtig, während der Kommunikation zwischen User-, Server- und ÖPNV-Komponente so wenig Daten wie möglich zu übertragen. Um den Besonderheiten der verschiedenen Komponenten gerecht zu werden, differenziert das Protokoll zwischen diesen. Die folgenden Abbildungen zeigen die unterschiedlichen Protokollaufbauten mit ihren dazugehörigen Kommunikationswegen (in Rot dargestellt).

4.4.1 Protokollaufbau Journey-Client – ROBERTA-Server

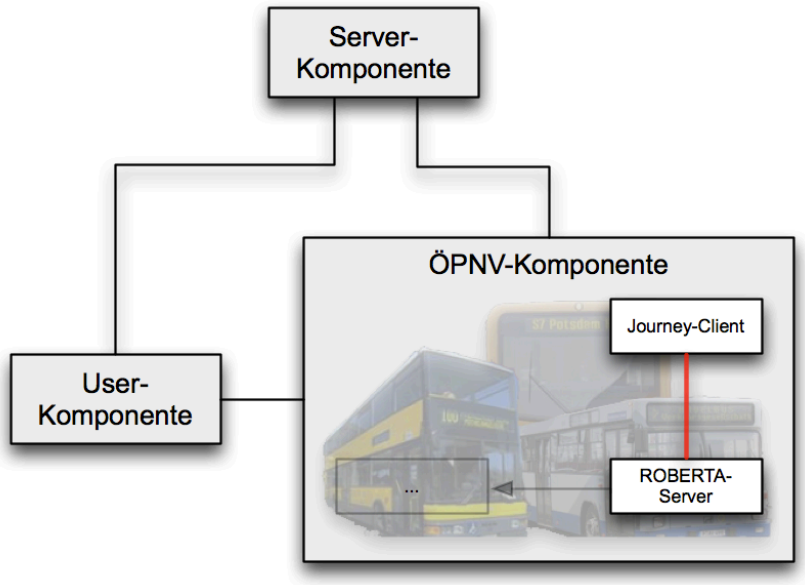

Abbildung 4-6: Kommunikationsabschnitt ROBERTA-Server – Journey-Client

Die vom ROBERTA-Server erhaltenen Daten sind sogenannte Tracker-nachrichten. Diese werden explizit bei Verbindungsaufbau angefordert, da diese Pakete Informationen darüber geben, welche RFID-Tags von einer bestimmten RFID-Antenne empfangen werden. Die folgende Abbildung beschreibt den Aufbau des 10 Byte großen Paketes.

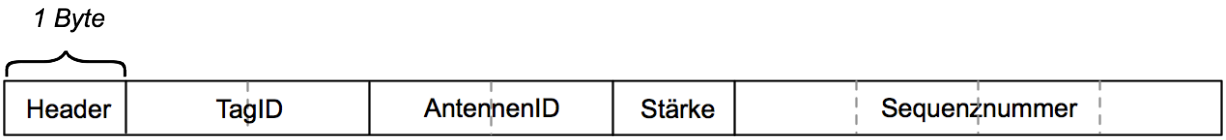

Abbildung 4-7: Trackernachricht (Bergemann, 2010)

4.4.2 Protokollaufbau Server – ÖPNV

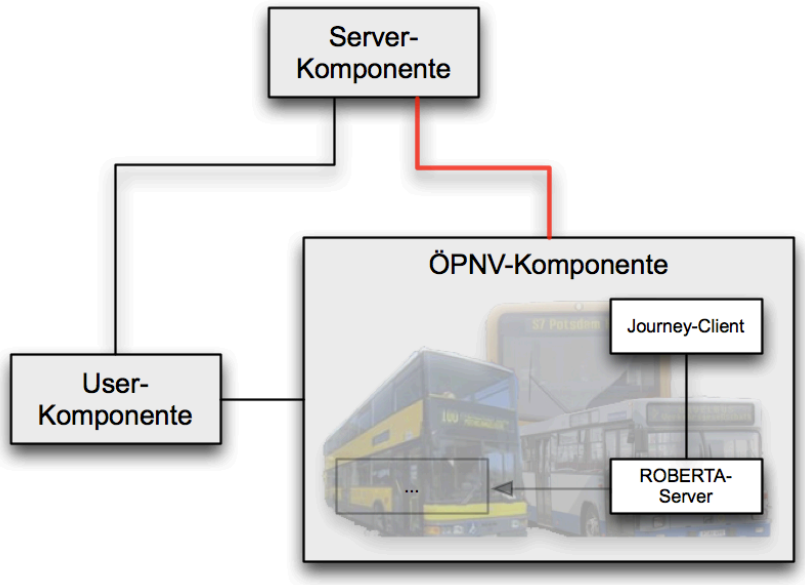

Abbildung 4-8: Kommunikationsabschnitt Server – ÖPNV

Der Journey-Client, der die Schnittstelle zur Server-Komponente darstellt, wickelt die Verbindung und Kommunikation zum ROBERTA-Server ab. Dabei sendet der Journey-Client beim Verbindungsaufbau mit der Server-Komponente ein „Initial-Paket“ mit der *PublicTransportID* und dem Endbahnhof (25 Byte). Aus den empfangenen Paketen (siehe Kapitel 4.4.1 Protokollaufbau Journey-Client – ROBERTA-Server) des ROBERTA-Servers wird die *TagID* von den restlichen Informationen des Paketes getrennt, da diese für die weitere Verarbeitung des Journey-Servers nicht relevant sind. Das auf 4 Byte minimierte Paket (inklusive *PublicTransportID*) wird anschließend zum Journey-Server übermittelt.

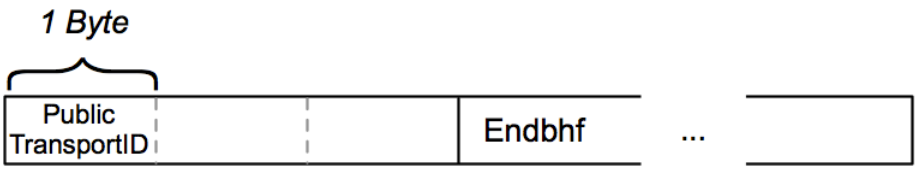

Abbildung 4-9: Protokollaufbau Server – ÖPNV (Initial-Paket)

Abbildung 4-10: Protokollaufbau Server – ÖPNV (Übermittlung der Tag-ID)

4.4.3 Protokollaufbau Server – User

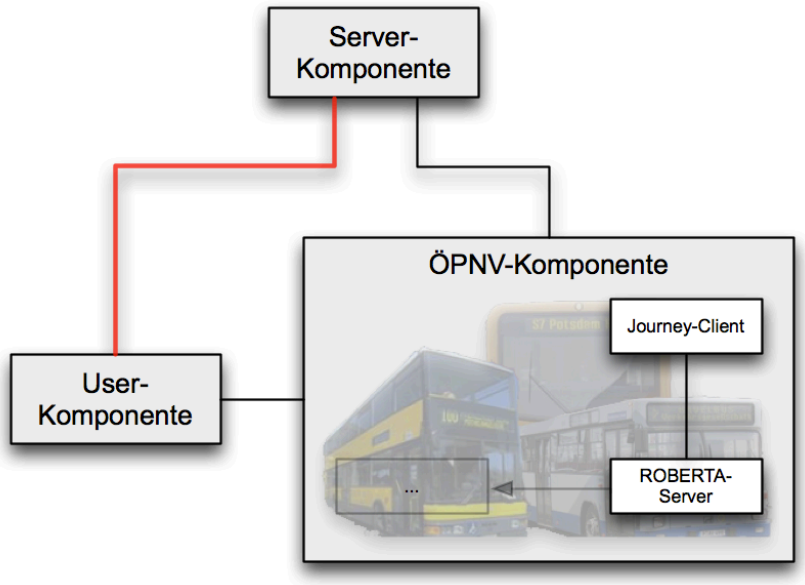

Abbildung 4-11: Kommunikationsabschnitt Server – User

Die User-Komponente sendet die *LinienNr*, den *Endbhf* (Endbahnhof) und die *TagID* zur Server-Komponente, die die Informationen auswertet und temporär speichert. Für den Endbahnhof sind 25 Byte im Protokoll reserviert.

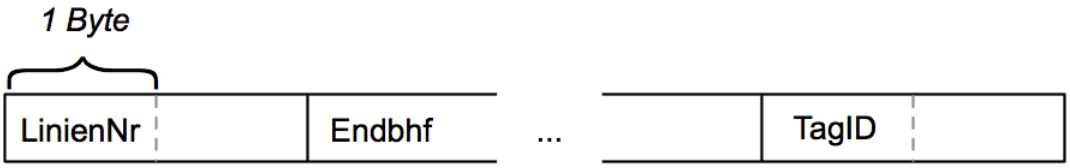

Abbildung 4-12: Protokollaufbau Server - User

4.5 Funktionalität

Es soll nun der Ablauf eines initialen Verbindungsaufbaus inklusive einer Client-Anfrage beschrieben werden. Dies wird mittels eines *UML*-Sequenzdiagramms dokumentiert und beschreibt Szenario 1 „Fahrgast stellt Anfrage und benutzt vom System errechnetes ÖPNV-Mittel“ und Szenario 3 „Fahrgast stellt Anfrage, wartet aber nicht, sondern entfernt sich vom Wartepunkt“ aus technischer Sicht.

Zu Beginn kontaktiert der Journey-Client den ROBERTA-Server und registriert sich für den Empfang von Trackernachrichten. Der ROBERTA-Server übermittelt anschließend kontinuierlich Trackernachricht, die RFID-Tag-IDs enthalten, die in Reichweite der angeschlossenen RFID-Antenne liegen. Diese Nachrichten werden vom Journey-Client aufbereitet, in dem Informationen entfernt werden, die für den weiteren Verlauf des System nicht relevant sind. Anschließend wird die *PublicTransportationID* des ÖPNV-Mittels angehangen und das Paket weiter zum Server gesendet. Eine Datenbankabfrage des Servers, ermittelt die *LinienNr* des ÖPNV-Mittels, anschließend prüft eine Routine, ob eine RFID-Tag-ID bereits für diese *LinienNr* registriert ist. An dieser Stelle wird der Client benachrichtigt, der sich für eine *LinienNr* mit seiner RFID-Tag-ID registriert hat. Das Sequenz-diagramm (Abbildung 4-13) zeigt, dass die Anfragen vom Mobilgerät asynchron stattfinden. Dies bedeutet, dass der Journey-Server auf Anfragen reagieren kann, bevor der Verbindungsaufbau vom Journey-Client zum ROBERTA-System statt-gefunden hat.

Nachdem der Fahrgast in das ÖPNV-Mittel eingestiegen ist, wird die registrierte RFID-Tag-ID gelöscht. Dies geschieht auch, wenn der Fahrgast nicht in das ÖPNV-Mittel einsteigt und sich vom Bahnhof oder von der Haltestelle entfernt hat.

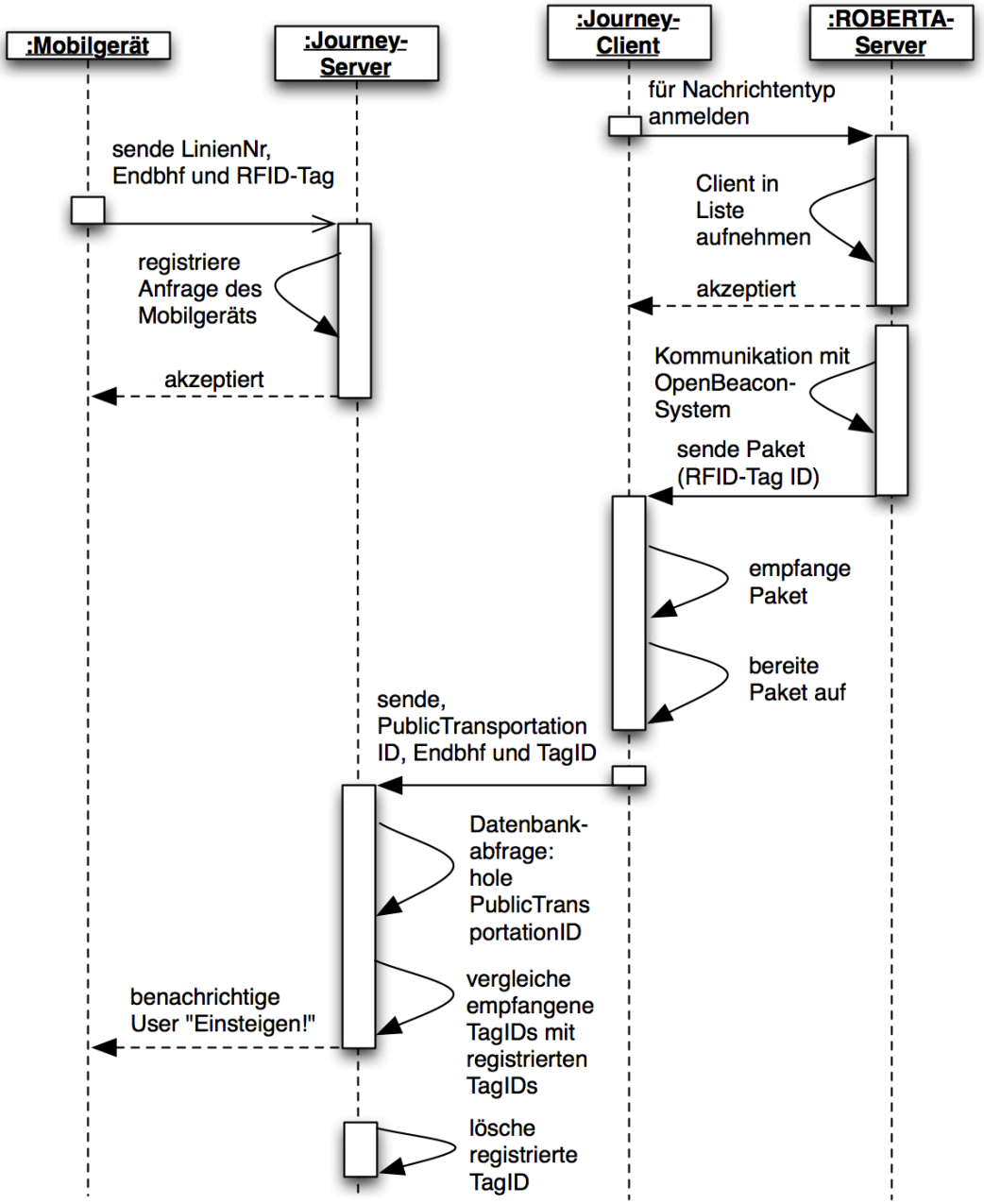

Abbildung 4-13: Sequenzdiagramm – Verbindungsaufbau und Kommunikation

4.6 Benutzerschnittstelle

Die Benutzerschnittstelle ist eine Software-Komponente, die dem Benutzer durch die Verwaltung von Ein- und Ausgaben die Interaktion mit dem Computersystem erlaubt.

Die View (Präsentationsschicht) des MVC-Entwurfsmusters stellt diese Schicht dar und ist in verschiedenen Formen in der User-, Server- und ÖPNV-Komponente enthalten.

- User-Komponente

  Ein- und Ausgabe von Informationen finden auf der vom Webserver ausgelieferten Webseite statt.

- Server-Komponente

  Die gesamte Steuerung von Web- und Journey-Server geschieht über die Kommandozeile. Die Datenbankanwendung „*SQLite*“ stellt zur Bedienung eine Kommandozeilenversion bereit.

- ÖPNV-Komponente

  Aufgrund der Bündelung einzelner Softwaresysteme enthält diese Komponente mehrere Benutzerschnittstellen. Das ROBERTA-System bietet eine kommandozeilenbasierte Oberfläche. Der Journey-Client ermöglicht dem Benutzer, den Endbahnhof bzw. die Endstation des ÖPNV-Mittels eingeben zu können. Dies geschieht beim Starten der Software über die Kommandozeile.

Der Benutzerschnittstelle der User-Komponenten wurde in dieser Arbeit besonders viel Aufmerksamkeit gewidmet. Viele Applikation bieten sinnvolle Funktionalitäten, können jedoch diese Stärke aufgrund der umständlichen Bedienung nicht ausspielen. Die Benutzeroberfläche der Webseite sollte so einfach wie möglich gestaltet werden und sich auf die Kernfunktionalität (dynamische Fahrauskunft) des Programmes konzentrieren. Beim ersten Start des Programmes (Aufruf der Webseite) wird der Benutzer aufgefordert, seine RFID-Tag-Identifikationsnummer einzutragen. Diese wird anschließend gespeichert und muss zukünftig nicht mehr eingegeben werden.

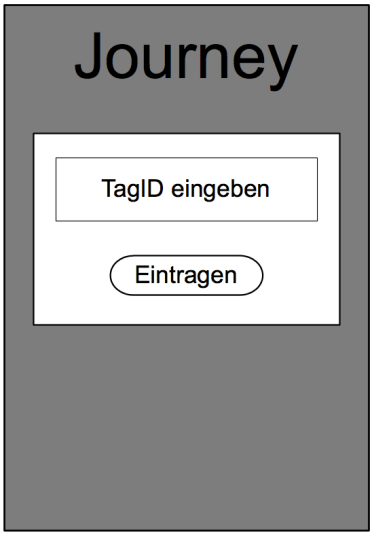

Abbildung 4-14: Benutzerschnittstelle GUI[43]-Prototyp: TagID-Eingabe

[43] Graphical user interface

Danach kann der Benutzer die Kernfunktionalität, das dynamische Benachrichtigen bei Näherung des ÖPNV-Mittels, nutzen. Dazu muss der potentielle Fahrgast die gewünschte Endstation eingeben.

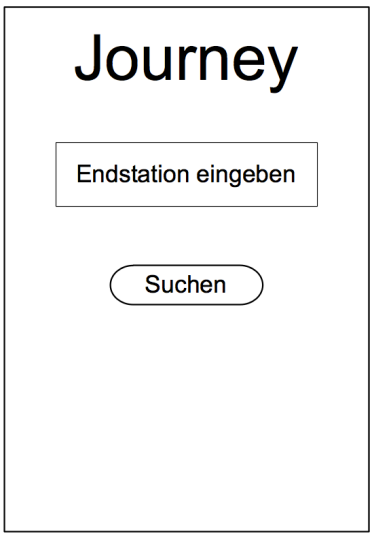

Abbildung 4-15: Benutzerschnittstelle GUI-Prototyp: Endstation-Eingabemaske

Anschließend sperrt das Programm die Oberfläche für weitere Eingaben und blendet ein Benachrichtigungsfeld ein, über das der Benutzer Informationen erhält. In diesem Feld bekommt der Benutzer die Aufforderung, in das ÖPNV-Mittel einzusteigen, zeitgleich ertönt ein Audiosignal. Sobald der Fahrgast in das ÖPNV-Mittel gestiegen ist, kann die Applikation beendet werden. Diese Funktionalität bietet die Oberfläche nicht, da dies über den Browser, in dem das Programm läuft, oder über das Gerät getan werden kann.

Abbildung 4-16: Benutzerschnittstelle GUI-Prototyp: Pop-Up für Meldungen

5 Implementierung

Dieses Kapitel beschreibt die Implementierung konkreter Problemstellungen aus dem Entwurf und zeigt programmiertechnische Besonderheiten auf. Einleitend schildert der erste Abschnitt die Wahl der Programmierumgebung und die Eingrenzung der zu implementierenden Funktionalitäten. Abschließend soll die Softwarearchitektur unter Aspekten der Benutzerschnittstelle, Kommunikation und Datenhaltung beleuchtet werden.

5.1 Funktionalität

In diesem Abschnitt werden die zu implementierenden Funktionalitäten beschrieben. Im Rahmen dieser Arbeit soll neben der Konzeption, ein Prototyp entwickelt werden. Dieser wird aufgrund der Komplexität der Anforderungen das Szenario 1 (3.1.1 Szenario 1: Fahrgast stellt Anfrage und benutzt vom System errechnetes ÖPNV-Mittel.), mit Einschränkungen und das Szenarios 3 (3.1.3 Szenario 3: Fahrgast stellt Anfrage, wartet aber nicht, sondern entfernt sich vom Wartepunkt) implementieren. Die erwähnten Einschränkungen wirken sich auf die Benutzung des Clients aus. Der Benutzer wird nicht in der Lage sein, mittels der Angabe des Zielbahnhofes eine Verbindungsauskunft erzeugen zu können. Da diese Funktionalität fehlt, muss der Benutzer neben dem Zielbahnhof, die gewünschte Liniennummer eingeben.

Das Szenario 2 „Fahrgast muss umsteigen, um den Zielbahnhof erreichen zu können“ und das Abfragen von Fahrinformationen wird wie bereits begründet, nicht im Rahmen dieser Arbeit entwickelt.

Die nächsten Abschnitte thematisieren die detaillierte Vorgehensweise zur Entwicklung des Prototypen unter Berücksichtigung der in diesem Abschnitt eingegrenzten Funktionalitäten.

5.2 Programmierumgebung und Datenhaltung

Zur Disposition stehen mehrere Programmiersprachen für die Serverimplementierung, da der ROBERTA-Server in C++ und die Firmware des OpenBeacon-Projektes in C geschrieben ist. Zusätzlich existiert bereits ein ROBERTA-Client in Java auf dessen Codebasis zurückgegriffen werden kann (Lenz, 2010). Die Entscheidung fiel jedoch auf die Programmiersprache PHP[44]. Die Gründe dafür liegen zum einen in der Möglichkeit plattformunabhängig serverseitige Applikationen zu entwickeln und zum anderen in der Lösung des Problems, wie ein Datenaustausch zwischen einem Webbrowser im Client und einem Server-*Socket* implementiert werden kann. Aufgrund der hohen Skalierbarkeit von PHP-Applikationen steigt der Ressourcenbedarf nur gering bei wachsender Eingabe-

[44] PHP Hypertext Preprocessor

menge (Viele Clients und ÖPNV-Mittel) an. Für weiteren Performancegewinn können in PHP geschriebene Programme in *C++*-Quellcode übersetzt werden. Ein *C++-Compiler* (z.B. g++) überträgt diesen anschließend in Maschinensprache.

Auch bei der Wahl der Programmiersprache des Clients steht die Plattformunabhängigkeit im Vordergrund. Dazu bietet sich eine Webapplikation, die im Browser funktioniert, an und grenzt dabei die Auswahl der Programmiersprachen ein. Diese Eingrenzung ist jedoch kein Negativaspekt, da browserlauffähige Programmiersprachen durchaus mit mächtigen Programmiersprachen wie C, C++ oder Java in Hinblick auf Funktionalität mithalten können. Ein gutes Beispiel für eine fortgeschrittene Applikationen im Web stellt der Google Mail Dienst[45] dar. Aufgrund der Funktionen *WebSockets*, *GeoLocation und Local Storage* dient HTML5 als Basis des Web-Clients. Zusätzlich werden mit JavaScript und CSS die Funktionalitäten erweitert.

Die Anforderung an die Datenhaltung der Komponenten ist sehr verschieden, sodass für jede Komponente eine unterschiedliche Wahl der Datenbanksysteme getroffen worden ist. Wegen der geringen zu speichernden Datenmenge ist für die User- und ÖPNV-Komponente kein komplexes Datenbanksystem nötig. Mit Hilfe von *SQLite* in der ÖPNV-Komponente und HTML5 *Local Storage* in User-Komponente können die einfachen Relationen in strukturierter Form persistent gespeichert werden. Sehr einfach für Benutzer und Entwickler gestaltet sich *Local Storage*. Der Benutzer der User-Komponente (Client) hat keinen Einrichtungs- oder Installationsaufwand, da diese Funktionalität mit einem aktuellen Browser mitgeliefert wird. Der Abschnitt 5.3.1.3 Datenhaltungsschicht beschreibt die Nutzung von *Local Storage* aus Sicht des Entwicklers.

Innerhalb der ÖPNV-Komponente kommt *SQLite* zum Einsatz. Die Datenbank besteht aus einer Datei und kann portabel z.B. auf einem USB-Stick mitgeführt werden. Dies hat den Vorteil, dass ÖPNV-Mittel und Computer-System nicht miteinander verknüpft sind. Bei Ausfall des Computers, muss nur die *SQLite*-Datenbank auf einen anderen Computer übertragen werden.

In der Server-Komponente muss ein komplexeres Datenmodell abgebildet werden. Hierfür eignet sich das Datenbankmanagementsystem *MySQL*. Es kann innerhalb von Datenbanken Relationen zwischen Tabellen abbilden. Zusatzprogramme wie *MySQLWorkbench* erlauben eine textbasierte oder grafische Erzeugung von ER-Modellen. Diese können anschließend direkt in das Datenbanksystem eingefügt werden.

[45] https://mail.google.com/

Zusammenfassend verdeutlicht folgende Tabelle die komponentenbezogene anforderungsorientierte Wahl der Programmiersprache und des Datenhaltungssystems.

Komponente	Programmiersprache	Datenhaltung
User	HTML5/JavaScript/CSS	HTML5 Local Storage
Server	PHP	MySQL
ÖPNV	PHP	SQLite

Tabelle 5-1: Übersicht Programmiersprache und Datenhaltung

5.3 Umsetzung der Softwarearchitektur

In den kommenden Abschnitten werden nun komponentenbezogene Implementierungsdetails beschrieben.

5.3.1 User-Komponente

Die User-Komponente beinhaltet eine Webseite bzw. Webapplikation. Diese ist gemäß des MVC-Modells aufgebaut, wobei der Controller die Kommunikation zum Server verwaltet, die lokale *SQLite*-Datenbank die Datenhaltungsschicht darstellt und die View mittels HTML-Elementen aufgebaut ist. Im Folgenden soll detailliert auf Besonderheiten hinsichtlich Implementierung der Schichten eingegangen werden.

5.3.1.1 Präsentationsschicht

Die Präsentationsschicht der User-Komponente beschreibt die Benutzerschnittstelle des Clients. Diese ist mit den Programmiersprachen HTML5, JavaScript und CSS realisiert. Auf der Benutzeroberfläche eingesetzten UI[46]-Komponenten entsprechen dem des HTML-Standards und sollen dem Benutzer ein gewohntes Aussehen vermitteln.

Für iOS basierte mobile Geräte gibt es die Möglichkeit dem Browser mitzuteilen, dass es sich um eine Webapplikation handelt, die im Vollbildmodus angezeigt werden soll. Zusätzlich wird die die *Pinch-To-Zoom*[47] Funktion deaktiviert.

```
1       <meta name="viewport" content="user-scalable=no, width=device-width, initial-
scale=1.0, maximum-scale=1.0"/>
2       <meta name="apple-mobile-web-app-capable" content="yes" />
3       <meta name="apple-mobile-web-app-status-bar-style" content="black" />
```

Listing 5-1: Client – Definition des Vollbildmodus für iOS-Geräte

[46] User Interface

[47] Diese Funktion ermöglicht das Vergrößern eines Bildschirmabschnittes mittels einer Fingergeste.

5.3.1.2 Steuerungsschicht

Die Hauptaufgabe der *Control*-Schicht des MVC-Modelles besteht in der Verwaltung der Verbindung zum Server. Diese Verbindung ist mit Hilfe der HTML5-*WebSocket* API (siehe 2.2.1 HTTP-Push) realisiert. Der *WebSocket* gibt vier Status zurück auf diese das Programm reagieren kann, aber nicht muss. Den Aufbau der Verbindung, löst die Programmierschnittstelle sehr elegant, indem eine *WebSocket*-Server Adresse an ein neu zu erstellendes Objekt übergeben wird und anschließend das Objekt den Zustand der Verbindung mittels numerischer Werte, übergibt. Alternativ können die sogenannten *readyState*[48] auch über Ereignisse verwaltet werden. Nachdem die Verbindung erfolgreich aufgebaut wurde, kann mittels des *readyState onopen* an den Server Daten übermittelt werden. Dies geschieht über das *send*-Attribut des *Socket*-Objektes. Die zusendenden Informationen werden, mit vorheriger Prüfung, aus den <*input*>-Elementen gewonnen und direkt an den Server übermittelt.

```
1       function connect() {
2               var host = "ws://www.example.com:8090/JourneyServer.php";
3               try {
4                               socket = new WebSocket(host);
5                               //Verbindung zu Host hergestellt
6                               socket.onopen = function(msg) {
7                               //Sende Nachricht an Server
8                               socket.send(LinienNr+EndBhf+TagID);
9                               };
10                              //Host antwortet
11                              socket.onmessage = function(msg) {
12                              //Verarbeite Serverantwort
13                              };
14                              //Verbindung wurde geschlossen
15                              socket.onclose = function(msg) {
16                              //Zeige (Fehler)meldung
17                              };
18                      } catch(ex) { alert(ex); }
19              }
```

Listing 5-2: Client – WebSocket-Verbindungsverwaltung

Eine weitere Aufgabe der *Control*-Schicht ist das Lokalisieren des Clients über die HTML5-*Geolocation* API. Das Betriebssystem (oder der Browser) fordert die Zustimmung des Benutzers, um Positionsdaten erheben zu dürfen. Anschließend werden die Positionsdaten in der Funktion *handlePos()* verarbeitet und mittels *Reverse Geocoding* der Google *Geocoding* API in eine visuell lesbare Adresse umgewandelt.

[48] http://dev.w3.org/html5/websockets/#the-websocket-interface

```
1       //Positionsdaten werden erhoben
2       navigator.geolocation.getCurrentPosition(handlePos,handleErr);
3       //In positions-Objekt befinden sich Latitude und Longitude
4       function handlePos(position) {
5               var lat = position.coords.latitude;
6               var lng = position.coords.longitude;
7               var geocoder = new google.maps.Geocoder();
8               var latlng = new google.maps.LatLng(lat, lng);
9               //Führe Reverse Geocoding durch
10              if (geocoder) {
11                      geocoder.geocode({'latLng': latlng}, function(results, status) {
12                      //results beinhaltet die Adresse
13                      });
14              }
15      }
```

Listing 5-3: Client – Geolocation API in Kombination mit Google Maps API

5.3.1.3 Datenhaltungsschicht

Die Datenhaltung der User-Komponente (Client) beläuft sich bei der Speicherung eines Wertes, die Identifikationsnummer des RFID-Tags. Hierfür wird die HTML5-Funktion *Local Storage* (siehe 2.1.2 HTML5) genutzt und der Wert lokal auf dem Client in einer *SQLite*-Datenbank gespeichert. Die *Local Storage* Programmierschnittstelle gestaltet die Speicherung von Werten sehr komfortabel. So kann mit *localStorage.setItem('AttributName', 'Wert')* ein Eintrag erstellt werden. Dies geschieht in der Funktion *saveTagID()*. Diese wird ausgeführt nachdem der Benutzer einen RFID-Tag eingegeben hat und die Fahrauskunftssuche startet. Beim nächsten Besuch der Webseite wird nachgesehen, ob eine Tag-ID bereits gespeichert worden ist. Nachdem vollständigen Laden der Webseite führt die Funktion *restoreTagID(),* die Funktion *localStorage.getItem('TagID')* aus. Dies erlaubt das Wiederherstellen der gespeicherten Tag-ID. Im Falle des Nichtvorhandenseins der Tag-ID wird der Benutzer aufgefordert diese einzutragen. Ist die Tag-ID bereits vorhanden, wird diese in das Tag-ID-Textfeld geschrieben. Mit Hilfe von *Local Storage* wird der Benutzer der Webseite nur einmal aufgefordert seine RFID-Tag Identifikationsnummer einzugeben.

```
1       function saveTagID() {
2               //Speichert TagID aus TagID Text Feld
3               if (localStorage.setItem('TagID', tagIDField.value)) {}
4       }
```

Listing 5-4: Client – Speicherung der TagID mittels Local Storage

```
1       function restoreTagID() {
2               //Sofern TagID bereits eingegeben,dann schreibe in TagID Feld
3               if (localStorage.getItem('TagID')) {
4                       tagIDField.value = localStorage.getItem('TagID');
5               }
6       }
```

Listing 5-5: Client – Wiederherstellung der TagID mittels Local Storage

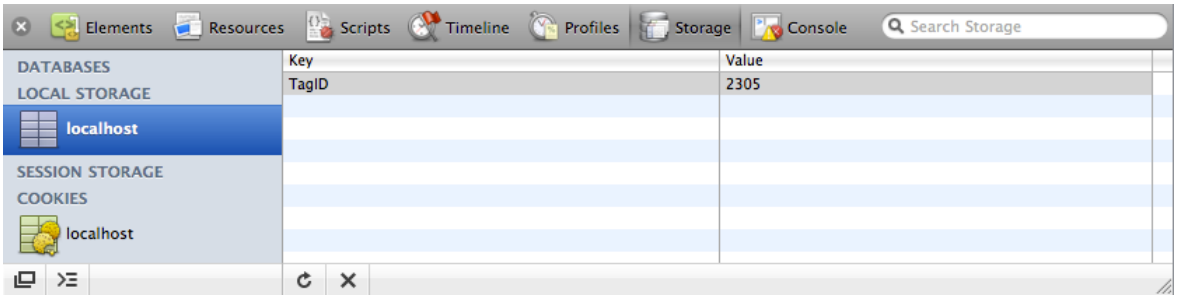

Abbildung 5-1: Safari Web Inspector

5.3.2 Server-Komponente

Ein Apache Webserver, ein *MySQL* Server und der im Rahmen dieser Arbeit implementierte Journey-Server sind Bestandteile der Server-Komponente. Gemäß des MVC-Modells besteht diese aus einer Präsentationsschicht, in Form eines Kommandozeilen-Zugriffs und einer Steuerungsschicht in der Kommunikations- und Datenbankverwaltung geregelt ist sowie das Auswerten der registrierten und vom ÖPNV-Mittel gesendeten RFID-Tag Identifikationsnummern. Außerdem enthält das Modell eine Datenhaltungsschicht in der Informationen persistent in einer *MySQL*-Datenbank abgelegt sind.

Im Folgenden soll detailliert auf Besonderheiten hinsichtlich der Implementierung der Schichten eingegangen werden.

5.3.2.1 Präsentationsschicht

Die Benutzeroberfläche des Servers beschränkt sich auf die Ausgabe von *Debug*-Informationen. Diese geben den Inhalt empfangener Pakete des Journey-Clients und vom Server generierte Identifikationsnummer der Clients aus.

5.3.2.2 Steuerungsschicht

Die Aufgaben der Steuerungsschicht unterteilen sich in Verwaltung der Kommunikation und den Abgleich der RFID-Tag Identifikationsnummern.

5.3.2.2.1 Kommunikation

In der Kommunikationsschicht der Server-Komponente laufen die Verbindungen der User (Clients) und der ÖPNV-Mittel zusammen. An dieser Stelle ist es besonders wichtig, dass ankommenden Verbindung ordnungsgemäß verwaltet werden. Dazu zählt das Öffnen und Schließen einer Verbindung, aber auch das Lesen der gesendeten Daten bzw. das Antworten auf diese.

In der Umsetzung stellt dies ein *socket_select*-Aufruf dar (siehe Listing 5-6: Server – Verbindungsverwaltung mittels socket_select()). Dieser Aufruf nimmt als Parameter ein *Socket*-Array entgegen und prüft, ob sich der Status eines *Sockets* geändert hat oder ein neuer *Socket* hinzugekommen ist. *Socket_select* gibt die Anzahl der Änderungen der *Sockets* zurück. Ist diese größer 1, beginnt der Server vom *Socket* Daten zu lesen bis die gesendete Datengröße 0 Bytes beträgt. Abschließend beendet der Server die Verbindung zum *Socket* und entfernt diesen aus dem Array.

Die sogenannte *Callback*-Funktion ermöglicht der Netzwerkschicht „im Hintergrund zu operieren“. Das Programm springt nach dem Aufruf der Netzwerkfunktionalität zurück zu einer im Aufruf zusätzlich definierten Methode. Innerhalb dieser definierten Methode können empfangene Daten bearbeitet und weitere Programmlogiken implementiert werden. Außerdem bearbeitet die Methode weitere Controller-Funktionalitäten, wie die Datenbankverwaltung und den Abgleich der vom ÖPNV-Mittel gesendeten Tag-IDs. Einen detaillierten Einblick dieser Funktionen soll der nächste Abschnitt bieten.

Zuvor zeigt der folgende Auszug aus dem Quelltext der Server-Komponente wie die Verwaltung der Client-Verbindung geschieht. Anschließend wird die *WebSocket*-Funktionalität des Servers erläutert.

```
1       public function run() {
2               //Starte Schleife für Verbindungsverwaltung
3               while(true) {
4                       $changed = $this->sockets;
5                       //Verwaltung von Sockets
6                       socket_select($changed, $write=NULL, $except=NULL,NULL);
7                       //Socket mit Statuswechsel wird bearbeitet
8                       foreach($changed as $socket) {
9                               //Verbinde zu Socket und lese Daten
10                              $this->connect($client);
11                              $bytes = @socket_recv($socket, $buffer, 2048, 0);
12                                      if($bytes == 0) {
13                                              //Verbindungsabbau wenn keine Daten mehr von
Client auf Socket geschrieben werden
14                                              $this->disconnect($socket);
15                                              //Rufe Callback Funktion auf und übergebe
Gesendete Daten etc.
16                                              if ($this->callback) {
17                                                      call_user_func($this->callback, $user,
$buffer, $this);
18                                              }
19                                      }
20                      }
21              }
22      }
```

Listing 5-6: Server – Verbindungsverwaltung mittels socket_select()

Um mittels eines Browsers Verbindungen zu dem Server aufnehmen zu können, muss ein *WebSocket-Handshake* implementiert werden. Dieser *Handshake* ermöglicht durch einem im *WebSocket*-Protokoll definierten Authentifizierungsalgorithmus eine bidirektionale persistente TCP-Server-Client-Verbindung. Ein Beispiel für einen Server-*Handshake* sieht folgendermaßen aus (angenommen Server und Client werden lokal auf einem Computer ausgeführt):

HTTP/1.1 101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Origin: http://127.0.0.1
Sec-WebSocket-Location:ws://127.0.0.1:8090/Implementierung/HTML5Page/↵
RFID/JourneyServer.php

Die PHP-Implementierung dieses *Handshakes* ist aus dem quelloffenen Projekt „phpwebsocket“ entnommen und entsprechend im Quelltext der Server Implementierung gekennzeichnet (Laurent, 2010).

```
1       //...Prüfe zuvor, ob Handshake schon durchgeführt wurde
2       //Bereite Paket vor
3       $upgrade  = "HTTP/1.1 101 Web Socket Protocol Handshake\r\n" .
4                       "Upgrade: WebSocket\r\n" .
5                       "Connection: Upgrade\r\n" .
6                       "WebSocket-Origin: " . $headers['Origin'] . "\r\n" .
7                       "WebSocket-Location: ws://" . $headers['Host'] . $resource . "\r\n"
.
8                       "\r\n";
9       //Führe Handshake durch
10      socket_write($this->socket, $upgrade.chr(0), strlen($upgrade.chr(0)));
11      //... weiter in der Methode..
```

Listing 5-7: Server – WebSocket Handshake

5.3.2.2.2 Abgleich der RFID-Tag Identifikationsnummern

Um zu erkennen, ob sich ein Fahrgast in der Reichweite eines ÖPNV-Mittel befindet, das für den Fahrtantritt relevant ist, vergleicht der Server die registrierten RFID-Tag-IDs mit den vom ÖPNV-Mittel in Reichweite befindlichen Tag-IDs. Wenn ein Benutzer sich für eine Fahrauskunft anmeldet, so sendet er seine Tag Identifikationsnummer, den gewünschten Zielbahnhof und die Liniennummer an den Server, der diese Anfrage registriert und in der Datenbank hinterlegt.

Dieser Abgleich ist einfach aufgebaut, sodass dieser mit möglichst hoher Geschwindigkeit durchgeführt werden kann. Zuerst werden alle registrierten RFID-Tags per Abfrage an die Datenbank geladen und in den Speicher der Anwendung gelegt. Dies hat den Vorteil, dass die Daten der Datenbank nur einmal erfragt werden. Dies spart Ressourcen und vor allem Zeit. Mittels eines *Datenbanktriggers* werden Änderungen an der Tabelle, in der die registrierten RFID-Tag Identifikationsnummer gehalten werden, überwacht, sodass nur Abfragen durchgeführt werden, wenn Datenänderungen stattgefunden haben. Diese Lösung ermöglicht dem Server den Abgleich mit geringer Verzögerung durchzuführen.

Vom ÖPNV-Mittel empfangene RFID-Tag Identifikationsnummern werden instantan mit den registrierten Tags verglichen. Im Falle einer Übereinstimmung wird die zum RFID-Tag gehörende *UserID* abgefragt und über den *Socket* der Verbindung eine HTTP-Push Nachricht übermittelt.

Diese Nachricht informiert den Benutzer in das ÖPNV-Mittel einzusteigen und löscht dessen Registrierung aus der Datenbank. Dies geschieht über eine Zwischenschicht, die mit der Programmlogik und der Datenbankverwaltung verbunden ist. Die detaillierte Funktionsweise wird im folgenden Abschnitt geschildert.

```
1       public function processData($user, $msg, $server){
2               //Parse Client Paket
3               //...
4               //Bekomme PublicTransportationID aus LinienNr und EndBhf
5               $User->PublicTransportID = DAOFactory::getPublicTransportDAO()-
>getIdByLinienNrAndEndbhf($UserlinienNr, $userEndbhf);
6
7               //Registriere Benutzeranfrage
8               DAOFactory::getUserDAO()->insert($User);
9
10              //Parse ÖPNV Paket
11              //...
12              $oepnvLinienNr = DAOFactory::getPublicTransportDAO()-
>getLinienNrByID($oepnvPulicTransportationID);
13              $oepnvEndBhf = DAOFactory::getLinienDAO()-
>getEndBhfByID($oepnvPulicTransportationID);
14
15              //Wenn Ergebnis leer, dann TagID nicht für diese ÖPNV-Mittel registriert
16              $oepnvUserID = DAOFactory::getUserDAO()->getIdByTagIDAndEndBhf($oepnvTagID,
$oepnvEndBhf);
17              //Wenn doch, dann hole Socket
18              $oepnvUserSocket = $server->getSocketByUserId($oepnvUserID);
19              //Sende Nachricht per HTTP-Push
20              $server->send($oepnvUserSocket, json_encode("Bitte steigen in die Linie
".$oepnvLinienNr." Richtung \".$oepnvEndBhf.\" ein"));
21              //Entferne Eintrag
22              DAOFactory::getUserDAO()->delUserByTagID($oepnvTagID);
```

Listing 5-8: Server – Abgleich von RFID-Tag-IDs

5.3.2.3 Datenhaltungsschicht

Das relationale Datenbankverwaltungssystem *MySQL* bietet die Grundlage zur persistenten Speicherung von Daten der Server-Komponente. Um die konzeptionellen Unterschiede zwischen der relationalen Datenbank und der objektorientierten Programmiersprache zu vereinen, bildet eine objektrelationale Abbildung[49] die Vermittlungsebene zwischen diesen beiden Schichten. Sie besteht aus Datentransferobjekt, Fabrikmethoden und Datenzugriffsobjekt. Die folgenden drei Abschnitte sollen diese Entwurfsmuster allgemein erläutern und spezifisch aufzeigen an welchen Stellen diese in der Implementierung eingesetzt werden.

5.3.2.3.1 Data transfer object (DTO)

Mit Hilfe des Datentransferobjekts können mehrere Daten in einem Objekt gebündelt werden, so dass anhand eines einzigen Aufrufs im Programm auf den kompletten Datensatz zugegriffen werden kann. Besonders bei der Übertragung von Daten in langsamen Netzwerken entfaltet dieses Entwurfsmuster seine Vorteile, da das Objekt alle relevanten Daten bündelt und gesammelt übermittelt

[49] Object-Relational Mapping

werden kann. Der folgende Codeabschnitt zeigt die Klasse *Linie,* die mittels primitiver Datentypen die Tabelle *Linie* inklusive Spalten und deren Bezeichnung in eine objektorientierte Notation überführt. In der nächsten tieferliegenden Schicht werden mit Hilfe von Fabrikmethoden weitere Funktionalitäten dem Datentransferobjekt hinzugefügt.

```
1       class Linie {
2               var $ID;
3               var $LinienNr;
4               var $StationStart;
5               var $StationEnde;
6       }
```

Listing 5-9: Server – Definition des Datentransferobjekts

5.3.2.3.2 Factory Method[50]

Die Fabrikmethode (Factory Method) beschreibt ein Entwurfsmuster mit dem ein Objekt, nicht wie üblich durch ein Konstruktor, sondern durch den Aufruf einer Methode, erzeugt wird. Der folgende Codeausschnitt zeigt eine Fabrikklasse mit einer Fabrikmethode zum Erzeugen eines Datenzugriffobjektes für das Datentransferobjekt *Linie*. Die Erzeugung und anschließende Rückgabe des *LinieMySQLDAO* Objektes ermöglicht dem aufrufenden Code, das Ausführen von Anweisungen an dem Objekt *Linie*, welche anschließend als *MySQL*-Anweisungen auf der Datenbank ausgeführt werden. Die Definition dieser Befehle findet in der darunterliegenden Schicht, der DAO-Schicht, statt.

```
1       class DAOFactory {
2               public static function getLinieDAO(){
3                       return new LinieMySQLDAO();
4       }
```

Listing 5-10: Server – Implementierung der Fabrikmethode

5.3.2.3.3 Data access object (DAO)

Anhand dieses Entwurfsmusters kann die Datenquelle gekapselt werden, sodass beim Austausch der Datenquelle an dem aufrufenden Code keine Modifikation vorgenommen werden müssen. Dies miniminiert den Portierungsaufwand und trennt die Programmlogik von den technischen Details der Datenspeicherung. Nachdem nun das Datentransferobjekt (*DTO*) für die Tabelle *Linie* und dazugehörige Fabrikmethode erstellt worden ist, wird die Schnittstelle (*Interface*) zum Datenzugriffobjekt der Tabelle *Linie* erstellt und anschließend definiert (implementiert).

[50] Fabrikmethode

```
1       interface LinieDAO {
2               public function insert($Linie);
3       }
```

Listing 5-11: Server –Schnittstellendefinition der Datenbankabfragen

Das folgende Codebeispiel zeigt Implementierung der Funktionalität für das Einfügen von Daten in die Tabelle *Linie*.

```
1       class LinieMySQLDAO implements LinieDAO {
2               public function insert($Linie) {
3                       //MySQL-INSERT Befehl zum Einfügen der Daten
4                       $sql = 'INSERT INTO Linie (ID, LinienNr, StationStart, StationEnde)
VALUES (?,?,?,?)';
5                       $sqlQuery = new SqlQuery($sql);
6                       //Sequenziell werden “?” mit Daten ersetzt
7                       $sqlQuery->set($Linie->ID);
8                       $sqlQuery->set($Linie->LinienNr);
9                       $sqlQuery->set($Linie->StationStart);
10                      $sqlQuery->set($Linie->StationEnde);
11                      //INSERT-Befehl wird ausgeführt
12                      $ID = $this->executeInsert($sqlQuery);
13                      $Linie->ID = $ID;
14              //ID der eingefügten Zeile wird zurückgegeben
15              //Ermöglicht Kontrolle des Datensatzes
16              return $ID;
17              }
18      }
```

Listing 5-12: Server – Implementierung der Datenbankabfragen

An der Stelle im Programmcode, an der der Datenzugriff durchgeführt werden soll, kann mittels der Verkettung von *DTO*, *Factory Method* und *DAO* sehr sauber in der Programmlogik ein Datensatz hinzugefügt werden.

```
1       //Objekt wird instanziiert und Attribute werden gesetzt
2       $Linie = new Linie();
3       $Linie->LinienNr = "163";
4       $Linie->StationStart = "S-Adlershof";
5       $Linie->StationEnde = "S-Schoeneweide";
6       //An generischer Factorymethode wird Methode “insert” des spezifischen Datenobjekts
“Linie” aufgerufen
7       DAOFactory::getLinieDAO()->insert($Linie);
```

Listing 5-13: Server – Datenbankzugriff in Programmlogik

5.3.3 ÖPNV-Komponente

In der ÖPNV-Komponente stellt der Journey-Client die Schnittstelle zwischen dem ROBERTA-Server und dem Journey-Server dar. Der Einsatz im ÖPNV-Mittel erfordert die Eingabe des Endbahnhofs über die Präsentationsschicht. Die Steuerungsschicht funktioniert analog zum Kommunikationsteil der Steuerungsschicht des Servers. Die Speicherung von Informationen wird in der Datenhaltungsschicht organisiert. In den folgenden Abschnitten wird hinsichtlich der Implementierung auf Besonderheiten eingegangen.

5.3.3.1 Präsentationsschicht

Nach dem Programmstart des Journey-Clients wird der Benutzer aufgefordert den anzufahrenden Endbahnhof bzw. die anzufahrende Endstation anzugeben. Anschließend werden über die Kommandozeile *Debug*-Informationen über den Verbindungsstatus ausgegeben.

```
1       public function promptForLastStop() {
2               fwrite(STDOUT, "Bitte geben Sie ihre Endstation ein: ");
3               $lastStop = trim(fgets(STDIN));
4               fwrite(STDOUT, "Gute Fahrt nach \"$lastStop\"\n");
5               ProximityServerClient::$this->lastStop = $lastStop;
6       }
```

Listing 5-14: ÖPNV – Abfrage Endbahnhof/Endstation

5.3.3.2 Steuerungsschicht

Die Steuerungsschicht der ÖPNV-Komponente ist der Kommunikationsimplementierung der Server-Komponente sehr ähnlich. Mit *socket_select* wird die Verbindungsverwaltung zwischen dem ROBERTA-Server und der Server-Komponente realisiert. Der Journey-Client arbeitet als Vermittler zwischen dem ROBERTA-Server und der Server-Komponente. Ankommende Pakete werden verarbeitet und direkt an die Server-Komponente weitergeleitet.

Der Journey-Client erweitert einen abstrakten *Socket*-Server und erbt dessen Funktionalitäten, um ROBERTA-Server spezifische Eigenheiten zu implementieren. Während des Verbindungsaufbaus versendet der Client ein Anmeldepaket. Dieses Paket registriert den Journey-Client für den Empfang von Trackernachrichten (siehe 4.4.1 Protokollaufbau Journey-Client – ROBERTA-Server). Anschließend werden die empfangenen Pakete verarbeitet und direkt an die Server-Komponente weitergeleitet. Aus Gründen der Geschwindigkeit werden die Daten nicht (permanent) gespeichert.

```
1       public function processData($msg, $server) {
2               //Verarbeite ankommende Pakete
3               $msg = ProximityServerClient::mergePacketIntoArray($msg);
4               $msg = ProximityServerClient::hex2dec($msg);
5               //Sende nach Verarbeitung Pakete an Server
6               ProximityServerClient::send("0," \
                .ProximityServerClient::$line.",".$msg, \
                ProximityServerClient::$journeyCon);
7       }
```

Listing 5-15: ÖPNV – Steuerungsschicht Paketverarbeitung

5.3.3.3 Datenhaltungsschicht

Im Entwurf des Datenmodelles ist beschrieben, dass die Identifikationsnummer, die fest im spezifischen Gerät definiert ist, und die Endstation in der Datenhaltungsschicht verwahrt werden. Für diesen Zweck sind weder Relationen oder mehrere Tabelle nötig, so dass aufgrund der geringen Komplexität diese Informationen in einer *SQLite*-Datenbank gespeichert werden können.

Während des Programmstarts wird der Fahrer aufgefordert den Endbahnhof, der angefahren werden soll, einzugeben. Dieser Endbahnhof wird mittels eines *SQL INSERT* in die Datenbank geschrieben.

Im nächsten Kapitel soll nun der konzeptionierte und implementierte Prototyp zur Anwendung kommen. Es wird an Beispielen gezeigt wie die Komponenten zusammenarbeiten und wie der Benutzer mit dem System interagiert.

6 Evaluierung und Demonstration

In den folgenden Abschnitten soll der entwickelte Prototyp in Hinblick auf Plattformunabhängigkeit, Benutzerfreundlichkeit und Identifizierung sowie Benachrichtigung des Clients, bewertet werden. Anschließend demonstriert der letzte Teil des Kapitels, die Funktionsfähigkeit des Prototyps anhand eines Beispielszenarios.

6.1 Evaluierung des Systems

6.1.1 Plattformunabhängigkeit

Die Entwicklung eines plattformunabhängigen Clients stand zuerst nicht primär im Vordergrund. Während der Evaluierung der Web-Technologien zeichnete sich jedoch die Möglichkeit ab, die Client-Software, durch Verwendung einer plattformunabhängigen Technologie, einer breiteren Masse an mobilen Geräten zur Verfügung stellen zu können. Dabei wurde anschließend bewusst auf den in der Spezifizierung befindlichen Webstandard HTML5 gesetzt. Speziell auf dem Markt der mobilen Geräte wird von vielen Herstellern dieser Standard in die Webbrowser der mobilen Geräte implementiert.

Der entwickelte Client benutzt HTML5-spezifische Funktionen, die der Browser unterstützen muss. Zu diesen Funktionen zählen *GeoLocation, Local Storage* und *WebSockets*. Zur Zeit (Stand Dezember 2010) werden diese Funktionen nur von Desktop-Browsern und vom mobile Safari auf *iOS* 4.2 unterstützt. Die mobilen Betriebssysteme Android 2.3, Windows 7 Phone und webOS 1.4.5 unterstützen keine *WebSockets*. Aus diesem Grund kann die Lauffähigkeit momentan auf diesen Betriebssystemen nicht ermöglicht werden, da die Hersteller diese offen spezifizierte Funktionalität noch nicht in die Browser (bzw. in die Betriebssysteme) integriert haben.

Die grafische Benutzeroberfläche des Clients wurde für Geräte mit einer Displaygröße von drei bis fünf Zoll konzipiert. Außerdem sind alle Elemente der Oberfläche groß genug, um per Fingergeste bedient werden zu können. An Computern, mit einem größeren Bildschirm und klassischer Mausbedienung kann die Software ebenso problemlos genutzt werden. Die Darstellung der Benutzeroberfläche ist mittels *Meta-Tags für iOS*-Geräte optimiert. Auf anderen Plattformen werden diese Meta-Tags ignoriert.

Eine plattformunabhängige Applikation muss meist den kleinsten gemeinsamen funktionalen Nenner bedienen. Dies ist ein klarer Nachteil von pattform-unabhängiger Entwicklung. Zusätzlich differiert bei nativen Applikation das plattformspezifische *Look and feel*[51] zu den pattformunabhängigen Web-Applikation. Einige Smartphone-Hersteller, zum Beispiel Apple bieten spezielle *Frameworks* an mit denen eine nativanmutende Web-Applikation entwickelt werden kann. Im nächsten Abschnitt werden Aspekte des Oberflächenentwurfes beschrieben und Besonderheiten erläutert

6.1.2 Benutzerfreundlichkeit

Während des Entwurfes der Benutzeroberfläche wurde darauf geachtet, die Struktur möglichst einfach zu halten. Die Anordnung der GUI-Elemente leitet sich von der Beschaffenheit der überwiegend quadratförmigen Smartphone-Bild-schirme ab. Daraus resultierend sind die Elemente untereinander gegliedert. Die Größen der Elemente und dessen Abstände sind auf Finger-Bedienung angepasst und mit ausreichender Höhe und Länge gestaltet.

Beim Starten der Applikation wird der Benutzer zunächst aufgefordert seine RFID-Tag-ID einzugeben. Diese Abfrage erscheint nur erstmalig, denn die Tag-ID wird im Browser gespeichert und beim erneuten Öffnen der Applikation wieder-hergestellt. Sollte sie wechseln, kann diese mittels drücken auf „TagID“ geändert werden. Dazu öffnet sich der gleiche Dialog, der auch bereits beim erstmaligen Starten erschien. Dies hat den Vorteil, dass der Benutzer diesen Dialog bereits kennt und sich an keine neue Oberfläche gewöhnen muss.

Eine Meldung erscheint bei jedem Start der Applikation, die erfragt, ob Positions-daten erhoben werden dürfen. Diese Abfrage ist betriebssystemspezifisch und kann meist für zukünftiges Erscheinen automatisch unterdrückt werden. Im unteren Teil der Applikation wird der momentane Aufenthaltsort des Benutzers (in Form der Adresse) schwarz hinterlegt angezeigt. Bewusst wurde eine vom Hintergrund abweichende Farbe gewählt. Dem Benutzer soll bewusst sein, dass das Gerät die aktuelle Position kennt und diese auch verwendet.

Im Rahmen der Entwicklung des Prototyps standen Aspekte der Barrierefreiheit nicht im Fokus der Entwicklung, sodass an dieser Stelle Potential für Erweiterung besteht.

[51] Aussehen und Handhabung

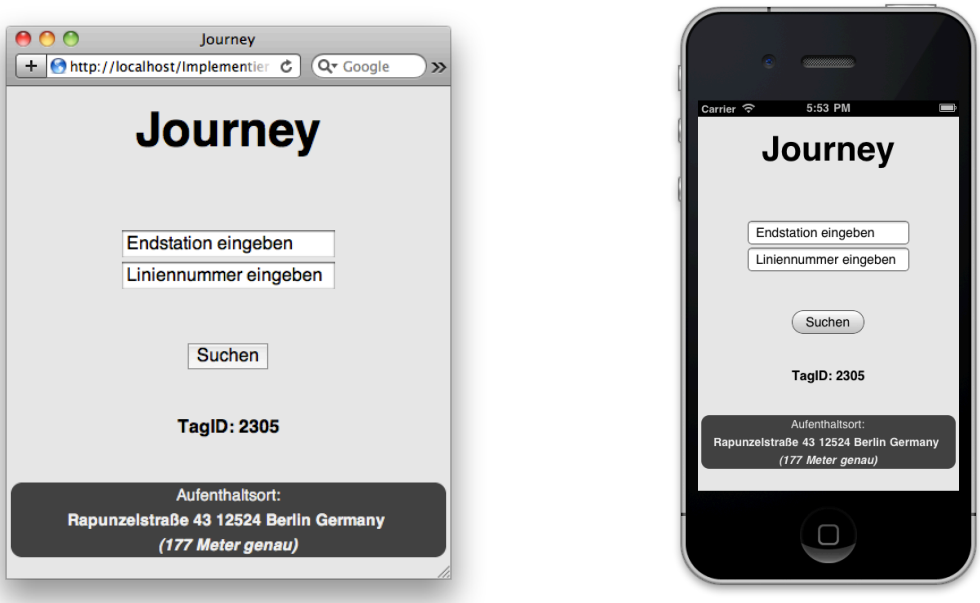

Abbildung 6-1: Vergleich grafische Oberfläche Safari – mobile Safari

6.1.3 Identifizierung des Clients und Benachrichtigung

Neben dem Auswählen des Zielortes, war die Möglichkeit den Client zu benachrichtigen, eine Kernanforderungen an die Software. Damit verbunden ist die Identifizierung des Clients durch das ÖPNV-Mittel.

Das ROBERTA-System, das auf dem OpenBeacon-System aufsetzt, bietet die Grundlage für die Client-Identifizierung per RFID-Technologie. Die RFID-Antenne wird von ÖPNV-Mittel mitgeführt, um in Reichweite befindliche RFID-Tags zu empfangen und dessen ID über den Journey-Client zum Server zu senden. Dies geschieht fortlaufend über eine TCP-*Socket*-Verbindung. Empfängt der Server eine RFID-Tag-ID, wird umgehend geprüft, ob diese ID registriert wurde und für das ÖPNV-Mittel bestimmt ist, dass die Tag-ID zum Server gesendet hat. In diesem Falle wird der Client über eine HTTP-Push-Nachricht informiert. Diese in der *WebSocket* API implementierte Funktionalität ermöglicht es direkt an verbundene Clients Pakete zu *pushen*. Auf dem Client wird die vom Server gesendete Nachricht direkt auf der grafischen Oberfläche mittel *Ajax* angezeigt und somit der Benutzer benachrichtigt.

6.2 Demonstration des Systems

Der folgende Abschnitt soll den Aufbau des Gesamtsystems erläutern und anschließend das Szenario 2: Fahrgast muss umsteigen, um den Zielbahnhof erreichen zu können.) an einem praktischen Fallbeispiel beschreiben.

6.2.1 Aufbau

Für die Demonstration des Systems werden auf der Hardwareseite ein OpenBeacon Ethernet EasyReader PoE II, ein PoE-Switch (in diesem Fall ein Netgear Pro Safe Poe Switch FS108P) verwendet. Ein Server, auf dem der Proximity-Server und der Journey-Client ausgeführt werden, ist mit dem PoE-Switch verbunden. Dieser Teil des Aufbaus simuliert das ÖPNV-Mittel.

Die Server-Komponente besteht aus einem weiteren Server auf dem der Journey-Server und der *MySQL* Server betrieben werden. Dieser ist mit der ÖPNV-Komponente und der User-Komponente verbunden.

Die User-Komponente besteht aus einem Smartphone (in diesem Fall ein iPhone) mit installierter Journey-Webapplikation und zwei OpenBeacon-RFID-Tags (Tag-ID 1320 und 1432).

6.2.2 Durchführung

Nach dem der Proximity-Server ausgeführt wurde, kann der Journey-Client mit anschließender Eingabe des Endbahnhofes gestartet werden. RFID-Tags können nun mit der RFID-Antenne des ÖPNV-Mittels in Kontakt treten.

Abbildung 6-2: Programmstart Journey-Client

In dem nächsten Schritt verbindet sich der Journey-Client mit dem Server, der Datenbank und Webseite bereitstellt.

Der Benutzer öffnet auf seinem Client die Webapplikation, erlaubt die Erhebung von Positionsdaten und gibt anschließend seine Tag-ID „1432“, den gewünschten Endbahnhof „S Karlshorst Bhf“ und die Liniennummer „2342“ ein.

Mittels drücken auf „Suchen“ sendet der Benutzer die Anfrage zum Server, der diese in seiner Datenbank vermerkt. Nun ist der Benutzer aufgefordert zu warten bis das ÖPNV-Mittel mit der Liniennummer „2342“ eintrifft.

Das ÖPNV-Mittel, das derzeit noch einige hundert Meter vom Benutzer entfernt ist, empfängt nun eine andere Tag-ID „1320“ und übermittelt diese zum Server. Der Server prüft, ob diese Tag-ID für das ÖPNV-Mittel „2342“ registriert wurde. Dies ist nicht der Fall, sodass der Server diese Anfrage verwirft und auf weitere wartet. Nun nähert sich das ÖPNV-Mittel „2342“ dem Benutzer und empfängt dessen Tag-ID „1432“. Diese sendet der Journey-Client zur Prüfung zum Server. Dieser erkennt, dass RFID-Tag für dieses ÖPNV-Mittel registriert wurde und benachrichtigt den Benutzer.

Nachdem der Benutzer in das ÖPNV-Mittel eingestiegen ist, wird die vom Client gestellte Anfrage auf dem Server gelöscht.

Abbildung 6-3: Abfrage Positionserhebung

Abbildung 6-4: Erstmalige Eingabe der Tag-ID

Abbildung 6-5: Fahrinformationsanfrage

Abbildung 6-6: „Bitte Warten“-Meldung

Abbildung 6-7: Benachrichtigung zum Einsteigen

7 Zusammenfassung und Ausblick

In diesem Kapitel soll die Arbeit zusammenfassend betrachtet, sowie die geschaffene Lösung evaluiert werden. Abschließend gibt dieses Kapitel einen Überblick über potentielle Erweiterungen des Systems.

Die Vision der Arbeit war eine Technik zu schaffen, die es dem Fahrgast ermöglicht, situationsgerecht Fahrinformationen zu beziehen und ihn gleichzeitig von der Konsultation der Fahrpläne oder Auskunftssysteme zu befreiten. Als Ziel sollte dazu eine benutzerfreundliche und funktionierende Software im prototypischen Zustand entwickelt werden.

Dabei war es wichtig, dass die Kernfunktion, das Auswählen des Zielortes und die Benachrichtigungsfunktion funktionieren. Die Schwerpunkte lagen in der Kommunikation zwischen mobilem Gerät und öffentlichem Verkehrsmittel. Hierbei wurden RFID- und GPS-Technik evaluiert, um ein geeignetes Verfahren zu finden, das es ermöglicht Annäherung zwischen ÖPNV-Mittel und Client zu erkennen. Ob dieses Ziel erreicht werden konnte wird im folgenden Abschnitt erläutert.

7.1 Soll-/Ist-Analyse

Eine textuelle Gegenüberstellung soll aufzeigen an welchen Stellen das implementierte System Schwächen aufweist bzw. Potentielle aufzeigt. Die Zielstellung galt während der gesamten Arbeit als richtungsweisend. An einigen Stellen mussten jedoch gewisse Probleme aufgrund der Komplexität aus dem Fokus genommen werden. Wo dies geschah, wird im folgenden Szenario beschrieben.

Der potentielle Fahrgast sollte dem System über sein mobiles Gerät (z.B. Smartphone) mitteilen, zu welchem Zielbahnhof oder Zielstation er geleitet werden möchte. Aufgrund der Schwierigkeit Fahrinformationen durch Dritte ÖPNV-Betriebe zu erhalten, muss der Fahrgast neben der Liniennummer zusätzlich den Endbahnhof (des ÖPNV-Mittels) eingeben. Nachdem der Client auf dem Server registriert ist, prüft der Server, ob das ÖPNV-Mittel, das der Fahrgast benutzen möchte, sich im unmittelbaren Umkreis befindet. Ist dies der Fall, reagiert der Server und benachrichtigt den Client. An dem Punkt, an dem der Fahrgast in das Verkehrsmittel einsteigt, stößt das implementierte System an seine Grenzen. Die Funktionalitäten der Benachrichtigung beim Aussteigen und Umsteigen sind nicht implementiert.

Das formulierte Ziel konnte somit nur eingeschränkt erfüllt werden. Auf Grund des modularen Aufbaus des Prototyps ist jedoch eine Erweiterung mit geringem Aufwand möglich. Zusätzlich wurde das System in einer Form entworfen, die eine Kombination aus RFID- und GPS-Technik ermöglicht oder durch andere Technologien ersetzt werden könnte. Im nächsten Abschnitt werden weitere für das System verwendbare Technologien erläutert.

7.2 Erweiterungen

Mögliche Erweiterungen gehen in Richtung eines „Fahrtassistenten“, der vom Einsteigen, Umsteigen, bis hin zum Aussteigen auf der gesamten Fahrt den Fahrgast unterstützt.

Für die Realisierung dieses Systems gibt es vermutlich zwei grundlegende Architekturentwürfe. Eine Methode ist das auch in dieser Arbeit verwendete Client-Server-Modell. An dieser Stelle können weitere Technologien zur Ortung des Benutzers und des ÖPNV-Mittels zum Einsatz kommen. Als Beispiel sind zu erwähnen Bluetooth, das *Ad-hoc*- und *Push*-Funktionalitäten bietet (Pritlove, 2010), NFC[52] und WLAN. Informationen von Systemen wie DAISY[53] oder DELFI[54], die aktuelle Fahrinformationen und Verbindungsauskünfte bereitstellen, können mit den Daten des intelligenten Fahrauskunftssystems kombiniert werden. Dadurch könnte das in der Arbeit entwickelte System, höhere Zeitgenauigkeiten erreichen.

Eine serverlose *Peer-To-Peer*-Variante wäre auch vorstellbar. Die Logik, die der Server des Client-Server-Modells implementiert, müsste bei diesem Prinzip auf Client und ÖPNV-Mittel aufgeteilt werden. Zusätzlich müssten neue Verfahren entwickelt werden, die die Identifizierung zwischen Client und ÖPNV-Mittel steuern.

[52] Near Field Communication

[53] Dynamische Auskunfts- und Informationssystem

[54] Durchgängige elektronische Fahrplaninformation

8 Anhang

8.1 Abbildungsverzeichnis

8.2 Listingverzeichnis

8.3 Literaturverzeichnis

(kein Datum). Abgerufen am 10. 12 2010 von http://www.ietf.org/old/2009/proceedings/04mar/slides/lemonade-1/sld2.htm

W3C. (2007 йил 26-11). *HTML Design Principles*. Retrieved 2010 йил 13-09 from HTML Design Principles: http://www.w3.org/TR/html-design-principles/

Wikipedia. (15. 12 2010). *RFID - Wikipedia*. Abgerufen am 15. 12 2010 von RFID - Wikipedia: http://de.wikipedia.org/w/index.php?title=RFID&oldid=82551263#Energieversorgung

Adobe Systems, Inc. (2008). *Adobe - Player Licensing : File Format Specification FAQ*. Retrieved 2010 йил 14-09 from Adobe - Player Licensing : File Format Specification FAQ: http://web.archive.org/web/20080401025101/http://www.adobe.com/licensing/developer/fileformat/faq/#item-1-8

ainonline.com. (2009 йил 01-03). *In-development satnav services could ecplise GPS: AINonline.* Retrieved 2010 йил 14-09 from In-development satnav services could ecplise GPS: AINonline: http://www.ainonline.com/news/single-news-page/article/in-development-satnav-services-could-eclipse-gps-20002/

Allthingsdistributed. (n.d.). Retrieved 2010 йил 25-09 from Allthingsdistributed: http://www.allthingsdistributed.com/images/globe-europe.jpg

Apple Inc. (2010 йил 08-04). *iPhone OS 4 Event*. (A. Inc., Editor, & Apple Inc.) Retrieved 2010 йил 24-09 from iPhone OS 4 Event: http://blog.quantcast.com/quantcast/2010/09/august-2010-mobile-os-share.html

Apple. (08. 03 2010). *Local and Push Notification Programming Guide: Apple Push Notification Service*. Abgerufen am 10. 10 2010 von Local and Push Notification Programming Guide: Apple Push Notification Service: http://developer.apple.com/library/ios/#documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/ApplePushService/ApplePushService.html%23//apple_ref/doc/uid/TP40008194-CH100-SW9

Beckmann, B. (05. 01 2010). *Google stellt Handy „Nexus One“ vor* . Abgerufen am 18. 12 2010 von Google stellt Handy „Nexus One“ vor : http://pics.computerbase.de/2/7/8/9/9/1_m.png

Bergemann, S. (2010). *Besucherinteraktion auf Veranstaltungen mit der OpenBeacon-Technologie.* Berlin: Stephan Bergemann.

euronews. (2010 йил 07-01). *Galileo services foreseen in 2014 - technology : europa, europe | euronews.* Retrieved 2010 йил 14-09 from Galileo services foreseen in 2014 - technology : europa, europe | euronews: http://www.euronews.net/2010/01/07/galileo-services-foreseen-in-2-greater-than4/

Elektronik-Kompendium.de. (kein Datum). *RFID - Radio Freuency Identification*. Abgerufen am 15. 12 2010 von http://www.elektronik-kompendium.de/sites/kom/0902021.htm: http://www.elektronik-kompendium.de/sites/kom/0902021.htm

Fyrd. (n.d.). *When can I use ...* Retrieved 2010 йил 24-09 from When can I use ...: http://caniuse.com/#agents=trident,gecko,webkit_saf,webkit_chr,presto&cats=HTML5&statuses=rec,pr,cr,wd

Google Inc. (kein Datum). *Android Cloud to Device Messaging Framework - Google Projects for Android*. Abgerufen am 11. 12 2010 von Android Cloud to Device Messaging Framework - Google Projects for Android: http://code.google.com/intl/en/android/c2dm/index.html

Hickson, I. (01. 11 2010). *Server-Sent-Events*. Abgerufen am 10. 12 2010 von Server-Sent-Events: http://dev.w3.org/html5/eventsource/

Jobs, S. (2010 йил 29-04). *Thoughts on Flash*, . ( ) Retrieved 2010 йил 26-09 from Thoughts on Flash: http://www.apple.com/hotnews/thoughts-on-flash/

kowoma.de. (2007 йил 29-11). *Bestimmung der Position beim GPS-System - Laufzeitmessung.* Retrieved 2010 йил 14-09 from Bestimmung der Position beim GPS-System - Laufzeitmessung: http://www.kowoma.de/gps/Positionsbestimmung.htm

kowoma.de. (2007 йил 18-06). *Erreichbare Genauigkeit bei GPS-Positionsbestimmungen*. Retrieved 2010 йил 25-09 from Erreichbare Genauigkeit bei GPS-Positionsbestimmungen: http://www.kowoma.de/gps/Genauigkeit.htm

kowoma.de. (2008 йил 18-11). *Das GPS Kontrollsegment - Die Bodenstationen und das Benutzersegment - Die Empfänger*. Retrieved 2010 йил 18-09 from Das GPS Kontrollsegment - Die Bodenstationen und das Benutzersegment - Die Empfänger: http://www.kowoma.de/gps/Bodenstationen.htm

kowoma.de. (2005 йил 30-11). *Die Umlaufbahnen der GPS Satelliten*. Retrieved 2010 йил 18-09 from Die Umlaufbahnen der GPS Satelliten: http://www.kowoma.de/gps/Umlaufbahnen.htm

kowoma.de. (2008 йил 13-02). *Die Frequenzen und ausgesendete Signale der GPS-Satelliten*. Retrieved 2010 йил 18-09 from Die Frequenzen und ausgesendete Signale der GPS-Satelliten: http://www.kowoma.de/gps/Signale.htm

kowoma.de. (2008 йил 30-09). *Die geschichtliche Entwicklung des GPS-Systems.* Retrieved 2010 йил 14-09 from Die geschichtliche Entwicklung des GPS-Systems: http://www.kowoma.de/gps/Geschichte.htm

kowoma.de. (2008 йил 13-02). *Fehlerquellen bei der GPS Positionsbestimmung*. Retrieved 2010 йил 25-09 from Fehlerquellen bei der GPS Positionsbestimmung: http://www.kowoma.de/gps/Fehlerquellen.htm

kowoma.de. (2005 йил 30-11). *Inklination der Umlaufbahnen*. Retrieved 24 йил 2010-09 from Inklination der Umlaufbahnen: http://www.kowoma.de/gps/inklination.gif

Laurent, R. (24. 11 2010). *phpwebsocket - Project Hosting on Google Code*. Abgerufen am 29. 12 2010 von phpwebsocket - Project Hosting on Google Code: http://code.google.com/p/phpwebsocket/source/browse/trunk/%20phpwebsocket/server.php

Leiba, B. (06 1997). *RFC 2177 - IMAP4 IDLE command*. Abgerufen am 10. 12 2010 von RFC 2177 - IMAP4 IDLE command: http://tools.ietf.org/html/rfc2177

Lenz, M. (2010). *Entwicklung eines NFC-gesteuerten mobilen Informationssystems für Modeschauen.* Berlin: Mathias Lenz.

National Space-Based Positioning, Navigation, and Timing Coordination Office. (n.d.). *Global Positioning System.* Retrieved 2010 йил 13-09 from Global Positioning System: http://gps.gov/systems/gps/

NGA. (2008 йил 31-12). *NGA GPS Division Home Page*. Retrieved 2010 йил 24-09 from NGA GPS Division Home Page: http://earth-info.nga.mil/GandG/saththml/StationMap.gif

Melanson, M. (2010 йил 30-08). *ReadWriteWeb Google Shows Off Chrome, HTML 5 With Interactive Music "Experience"*. Retrieved 2010 йил 13-09 from ReadWriteWeb Google Shows Off Chrome, HTML 5 With Interactive Music "Experience": http://www.readwriteweb.com/archives/google_shows_off_chrome_html5_with_interactive_mus.php?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+readwriteweb+%28ReadWriteWeb%29

Microsoft Corporation. (2010). *Silverlight for Windows Phone : The Official Microsoft Silverlight Site*. Retrieved 2010 йил 13-09 from Silverlight for Windows

Phone : The Official Microsoft Silverlight Site: http://www.silverlight.net/getstarted/devices/windows-phone/

Microsoft Inc. (kein Datum). *Exchange Server Protocol Documents*. Abgerufen am 10. 12 2010 von Exchange Server Protocol Documents: http://msdn.microsoft.com/en-us/library/cc425499%28EXCHG.80%29.aspx

OpenBeacon. (kein Datum). *OpenBeaconUSB*. Abgerufen am 15. 12 2010 von OpenBeaconUSB: http://www.openbeacon.org/images/OpenBeaconUSB.jpg

OpenBeacon. (kein Datum). *OpenBeaconPoE2*. Abgerufen am 15. 12 2010 von OpenBeaconPoE2: http://www.openbeacon.org/images/PoEII-s.jpg

OpenBeacon. (06. 09 2010). *Sputnik - OpenBeacon*. Abgerufen am 15. 12 2010 von Sputnik - OpenBeacon: http://www.openbeacon.org/Sputnik

Plehegar. (2010 йил 26-02). *HTML Working Group*. Retrieved 2010 йил 22-09 from HTML Working Group: http://www.w3.org/2007/03/HTML-WG-charter.html#deliverables

Prinz, D. T. (2009). *GPS in den Geowissenschaften*. Retrieved 2010 йил 24-09 from GPS in den Geowissenschaften: http://ivvgeo.uni-muenster.de/Vorlesung/GPS_Script/Bilder/Grundlagen/abb2_3_1.gifDr.

Prinz, D. T. (2009). *GPS in Geowissenschaften*. Retrieved 2010 йил 20-09 from GPS in Geowissenschaften: http://ivvgeo.uni-muenster.de/Vorlesung/GPS_Script/grundlagen_signale.html

Prinz, D. T. (2009). *GPS in Geowissenschaften*. Retrieved 2010 йил 26-09 from GPS in Geowissenschaften: http://ivvgeo.uni-muenster.de/Vorlesung/GPS_Script/grundlagen_signale.html

Pritlove, T. (26. 10 2010). *CRE169 Bluetooth - Chaosradio Podcast Network*. (T. Pritlove, Produzent, & Metaebene) Abgerufen am 26. 12 2010 von CRE169 Bluetooth - Chaosradio Podcast Network: http://chaosradio.ccc.de/cre169.html

Spiegel Online. (2007 йил 04-05). *EU-Navigationssystem: Milliarden-Desaster erschüttert Satellitenprojekt Galileo - SPIEGEL ONLINE - Nachrichten - Wirtschaft.* Retrieved 2010 йил 14-09 from EU-Navigationssystem: Milliarden-Desaster erschüttert Satellitenprojekt Galileo - SPIEGEL ONLINE - Nachrichten - Wirtschaft: http://www.spiegel.de/wirtschaft/0,1518,481124,00.html

Stahnsdorf.de. (05. 12 2010). *Stahnsdorf.de*. Abgerufen am 05. 12 2010 von Stahndorf.de: http://www.stahnsdorf.de/images/opnv.jpg

Stat Owl. (2010 йил 22-09). *Web Browser Plugin Market Share - Global Usage*. Retrieved 2010 йил 22-09 from Web Browser Plugin Market Share - Global Usage: http://statowl.com/plugin_overview.php

StatCounter. (2010 йил 22-09). *Top 12 Browser Versions from 2009 to 2010 | StatCounter Global Stats*. Retrieved 2010 йил 22-09 from Top 12 Browser Versions from 2009 to 2010 | StatCounter Global Stats: http://gs.statcounter.com/#browser_version-ww-yearly-2009-2010-bar

rfid-b2b.de. (kein Datum). *aktive Responder UHF.* Abgerufen am 15. 12 2010 von aktive Responder UHF: http://www.rfid-b2b.de/aktiveTransponder_UHF.htm

8.4 Tabellenverzeichnis

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Ort, Datum

Unterschrift