Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Proximity Beacons

Anwendungsmöglichkeiten von Näherungssensoren am Beispiel einer Android Applikation

Bachelorarbeit

zur Erlangung des akademischen Grades

Bachelor of Science

im Studiengang

Media Systems

Vorgelegt von

Jan Hilbig

Matrikelnummer: 1859223

am 4. September 2015

an der

Hochschule für Angewandte Wissenschaften Hamburg

Fakultät Design, Medien und Information

Erster Gutachter: Prof. Dr. Andreas Plaß

Zweiter Gutachter: Prof. Dr. Torsten Edeler

Kurzfassung

Proximity Beacons sind kleine Sender, die als Näherungssensoren verwendet werden. Die auf Bluetooth Low Energy gestützte Technologie, ist für Entfernungen von kurzer bis mittlerer Distanz ausgelegt. Mobile Endgeräte, wie Smartphones und Tablet PCs, können die Signale der Sender empfangen und die Entfernung zu Ihnen ermitteln. Durch Einführung des von Apple entwickelten iBeacon Standards sollen Beacons nun für eine große Nutzerzahl zugänglich gemacht werden. Die Softwareentwicklung für diese Technik steht noch ganz am Anfang, wodurch die Einsatzmöglichkeiten noch nicht vollständig erschlossen sind. Durch die Entwicklung einer iBeacon Anwendung für Android werden Möglichkeiten und Grenzen von Bluetooth Näherungssensoren betrachtet.

Abstract

Proximity beacons are small Bluetooth transmitters - based on the Bluetooth Low Energy standard - used as proximity sensors for short to medium range distances. Mobile devices, such as smartphones or tablets, are able to receive these transmitted signals and calculate the distance to the emitter. To make this technology available to a broader clientele, Apple introduced the iBeacon standard. The capabilities of this technology are limited due to the early state of software development within this field. The potential and limitations of Bluetooth proximity sensors will be explored further by using the iBeacon standard in an Android application

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelor-Thesis mit dem Titel:

_____

selbständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

(Unterschrift)

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Smartphones und Tablet PCs sind aus der modernen Informationsgesellschaft kaum noch wegzudenken. Die als „Mobile Devices“ bezeichneten Geräte vereinen Funk- und Kommunikationstechnologien sowie eine Vielzahl von Sensoren in einen kleinen tragbaren Gerät. Die Softwareentwicklung formt für diese Geräte immer wieder neue Anwendungsmöglichkeiten, indem sie die verschiedenen Techniken geschickt miteinander verknüpft. Durch Apples Vorstoß, mit iBeacon einen Standard für Näherungssensoren am Markt zu etablieren, ist nun eine für Anwender und Entwickler noch unerschlossene Technik zugänglich gemacht worden.

Näherungssensoren sind eigentlich nichts Neues. Jedoch nutzen die neuen Beacon Standards die bereits sehr verbreitete Funktechnologie Bluetooth. Hierdurch wird es zum ersten Mal möglich, diese Technik der breiten Masse zugänglich zu machen. Hersteller von Mobile Devices müssen ihre Geräte zur Nutzung der Standards weder Modifizieren noch neue Hardware nachrüsten. Im Vergleich zu verwandten Technologien wie RFID und NFC bietet Bluetooth eine Reichweite von mehreren Metern statt weniger Zentimeter. Anbieter die Proximity Beacons nutzen wollen, kommen dadurch mit viel weniger Knoten aus um einen auf Annäherung basierenden Service flächendeckend zu installieren.

Proximity Beacons definieren kleine Abschnitte im Raum. Anstatt dass Nutzer sich auf einer Karte suchen, auf Buttons klicken oder Aktionen in Menüs auswählen, soll es Anwendungen hierdurch möglich sein, auf die physikalische Welt zu reagieren, ohne dass ein Eingreifen auf Nutzerseite notwendig ist.[1] Der Entwicklungsstand für mobile Apps die Beacons nutzen, ist in Deutschland und Europa noch am Anfang. Sowohl auf technischer wie auch auf kreativer Seite sind die Möglichkeiten von Proximity Beacons noch nicht erschlossen. Es gibt also noch viel Raum, um die Technik zu erforschen.

1.2 Zielsetzung

Mit der Erstellung eines experimentellen Software Prototyps werden Erfahrungen mit dem Android SDK der Firma Google und den Proximity Beacon Standard „iBeacon“ der Firma Apple gesammelt. Durch das Planen und umsetzen des Prototyps soll ein umfassender Einblick zur technischen Funktionsweise von Proximity Beacons entstehen. Ein besonderes Augenmerk wird hierbei auf die Anwendungsentwicklung im Android SDK gelegt.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit ist in drei Teile unterteilt. Sie beginnt mit den Grundlagen der Schlüsseltechnologie Bluetooth Low Energy , dem iBeacon Protokoll sowie einigen Hardwarespezifikationen der iBeacon Transmitter. Der zweite Teil ist ein kurzes Pflichtenheft für eine Beispielapplikation. Hier wurde der praktische Teil der Arbeit geplant und konzeptioniert. Im dritten Teil sind die notwendigen Arbeitsschritte zur Implementierung des Quellcodes im Android SDK dokumentiert.

1.4 Abgrenzung

Diese Arbeit zeigt einen konkreten Anwendungsfall für den Einsatz von Bluetooth Low Energy Proximity Beacons. Der Fokus liegt hierbei auf der Softwareentwicklung für Android Systeme ab Version 4.4 *(KitKat)* sowie Apples proprietären iBeacon Standard. Da der Begriff „iBeacon“ ein von der Firma Apple geprägter Markenname ist, wird in dieser Arbeit der Ausdruck Proximity Beacons oft als Synonym für iBeacon verwendet.

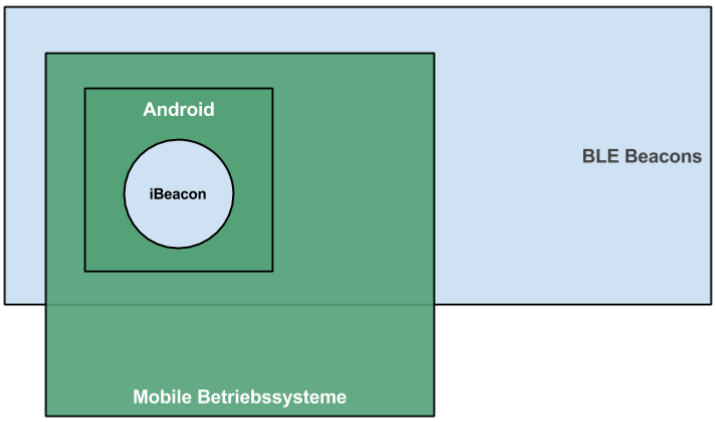

Abbildung 1.1: Themenabgrenzung

2 Grundlagen

Um zu verstehen wie Proximity Beacons funktionieren, ist es sinnvoll zunächst ihre Schlüsseltechnologie Bluetooth Low Energy (BLE) zu betrachten. Am Beispiel iBeacon wird gezeigt, wie ein Proximity Beacon Standard BLE auf Protokollebene nutzt. Zudem wird die Hardware der BLE Sender, den eigentlichen Beacons, betrachtet.

2.1 Bluetooth Low Energy

Bluetooth Low Energy ist eine Funktechnik zur Vernetzung von Geräten in geringen bis mittleren Distanzen. Es ist Teil der Bluetooth 4.0 Spezifikation die 2010 von der Bluetooth Special Interest Group (SIG) veröffentlicht wurde. BLE wird von der SIG unter anderem auch als Bluetooth Smart und Bluetooth Smart Ready vermarktet. Dabei können Bluetooth Smart Geräte ausschließlich mit dem neun 4.x Standard arbeiten während Bluetooth Smart Ready Geräte abwärtskompatibel zum klassischen Bluetooth 3.0 sind.

Mit der Veröffentlichung des Bluetooth 4.x Standards ist Bluetooth 3.0 jedoch nicht obsolet. Vielmehr wurde eine neue Protokoll Architektur für die BLE Funktionalität von der SIG in den Bluetooth Standard migriert. Die Technik wurde ursprünglich von Nokia unter den Namen „Wibree“ als Ergänzung zu Bluetooth entwickelt. Genau wie seine Vorgänger ab Version 2.1 arbeitet Bluetooth 4.x auf dem Frequenzband im 2,4 GHz Bereich.

Low Energy Geräte ab Bluetooth 4.0 sind speziell für Anwendungen ausgelegt, die in Intervallen geringe Datenmengen übertragen. Hierzu gehören unter anderem Sportsensoren (z.B. Pulsmesser), Smartwatches sowie Proximity Beacons. Die Low-Energy-Chips von Bluetooth 4.x benötigen nur eine kleine Batterie um für längere Zeit arbeiten zu können. Die tatsächliche Lebensdauer der Batterie hängt dabei vor allem von der Größe des Sendeintervalls ab.

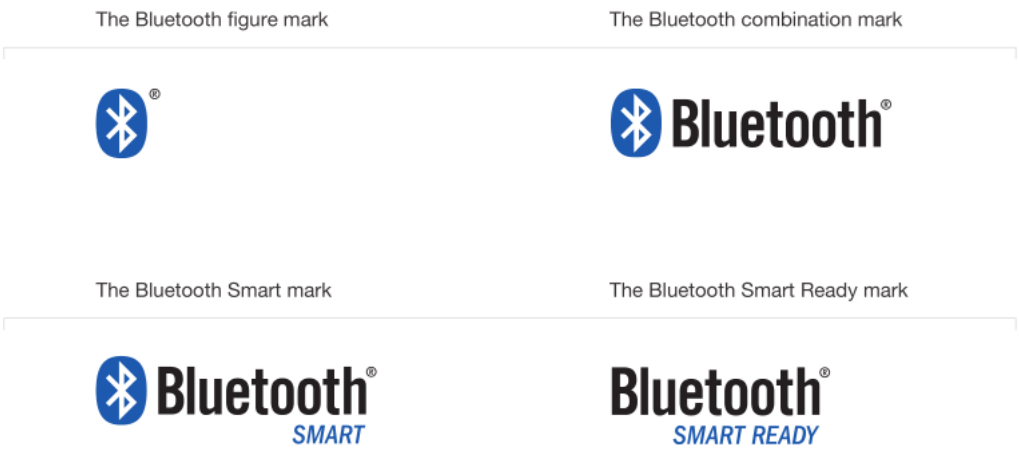

Abbildung 2.1: SIG Bluetooth Trademarks und Logos

2.1.1 Bluetooth Profile

Der Datenaustausch ist bei Bluetooth in sogenannten Bluetooth Profilen organisiert. Je nach Anwendungsfall wird dabei für jedes Gerät definiert, was notwendig ist um mit einem anderen Gerät zu kommunizieren. Ein Bluetooth Profil besteht dabei mindestens aus folgenden drei Punkten:

- Abhängigkeiten zu anderen Profilen
- Richtlinien für Benutzerschnittstellen
- Verwendete Bluetooth Protokolle

Für BLE Geräte, wie Proximity Beacons, sind jedoch nur zwei Profile von Bedeutung. Das Generic Access Profile (GAP) und das Generic Attribute Profile (GATT). Letzters wurde für die neue Protokollarchitektur von Bluetooth 4.x Low Energy spezifiziert.

2.1.2 Zugriffssteuerung durch GAP

Das Generic Access Profile steuert die Bekanntgabe *(Advertising)* und den Verbindungsaufbau *(Connection)* von Bluetooth Geräten. Es macht Geräte für die Außenwelt sichtbar. GAP definiert für BLE Geräte zwei unterschiedliche Rollen, **Central** und **Peripheral**.[2]

- **Peripheral** sind kleine Geräte die wenig Strom benötigen, Proximity Beacons zählen hierzu.

- **Central** sind Geräte denen etwas mehr Energie zur Verfügung steht. Sie steuern die Verbindung zu den Peripheriegeräten. In der Regel sind dies Smartphones oder Tablet PCs.

2.1.2.1 Verbindungsaufbau

Der Verbindungsaufbau für BLE Geräte wird in drei Schritten durchgeführt:

- Advertising

- Scan Request

- Scan Response

Bevor eine Verbindung zu einem BLE Gerät hergestellt wird, verschickt das Peripheriegerät in Intervallen zwischen 2ms - 10s **Advertising** Pakete. Die Kanaladresse für das Advertising ist Teil der BLE Spezifikation und daher immer gleich. Wird ein Advertising-Paket von einem Smartphone *(Central)* empfangen, antwortet dieses mit einem **Scan Request**. Das Peripheriegerät kann auf diese Aufforderung mit einer **Scan Response** antworten. Dies ist allerdings optional. Die Scan Response enthält vor allem die Kanaladresse über die das Smartphone eine Verbindung zum Peripheriegerät aufbauen kann. Der weitere Datenaustausch wird nun vom GATT geregelt. Bei einer bestehenden Verbindung werden keine weiteren Advertising Pakete vom Peripheriegerät verschickt.

2.1.2.2 BLE Broadcasting

Manchmal ist ein Verbindungsaufbau jedoch nicht gewollt. Dies ist genau dann der Fall, wenn ein Peripheriegerät Daten an mehrere Endgeräte gleichzeitig verschicken soll. Bei dem als BLE Broadcasting bezeichneten Verfahren werden die zu verschickenden Informationen im Datenteil der Advertising Pakete gekapselt. Genau diesen Ansatz verfolgen Proximity Beacon Technologien wie iBeacon.

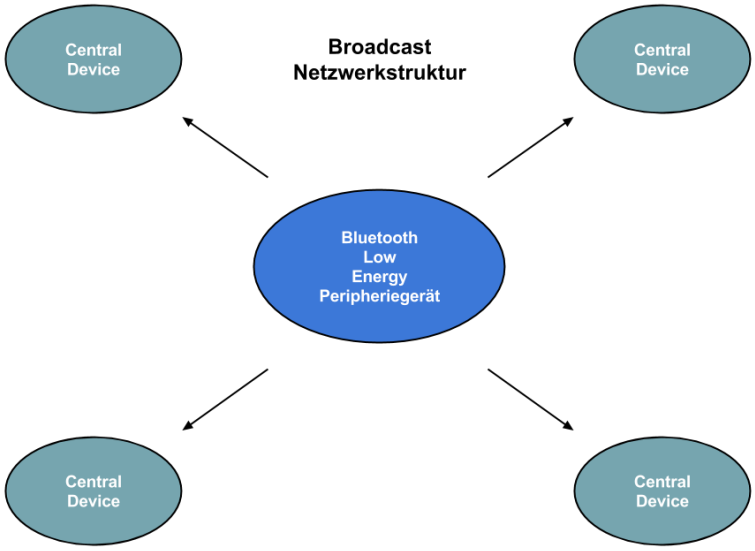

Abbildung 2.2: Bluetooth Low Energy Broadcasting

2.1.3 Datentransfer durch GATT

Das Generic Attribute Profile legt fest wie und was für Daten von BLE Geräten übertragen werden. Im Bluetooth Entwicklerportal (`https://developer.bluetooth.org/TechnologyOverview/Pages/Profiles.aspx#GATT`) stellt die SIG Hardwareherstellern und Softwareentwicklern eine Vielzahl dieser Profile zur Verfügung, die Grundlage für eigene Implementierungen sind. Ein BLE Peripheriegerät kann mit GATT immer nur eine einzelne Verbindung gleichzeitig aufbauen. Für die Gegenstelle gilt dieses jedoch nicht.

2.1.3.1 GATT Verbindungskonzept

Bei Verbindungen durch GATT agiert das Peripheriegerät als Server *(Slave)* und das Endgerät als Client *(Master)*. Der GATT Server stellt die für die Verbindung notwendigen Dienste bereit. Der Datenaustausch wird immer vom Master, also dem Client gestartet. Bei der Herstellung einer Verbindung wird vom Peripheriegerät ein Verbindungsintervall vorgeschlagen. Das Endgerät (Smartphone/Tablet-PC) wird infolgedessen versuchen zu jedem Intervall eine neue Anfrage an den Server zu schicken um zu überprüfen ob es neue Daten gibt. Wenn das Endgerät anderweitig beschäftigt ist, können Intervalle ausgelassen werden, ohne das die Verbindung abbricht.

2.1.3.2 GATT Profilarchitektur

GATT Profile basieren auf drei verschiedenen Objekttypen die **Profiles**, **Services** und **Characteristics** genannt werden.

- **Profiles** sind die eigentlichen GATT Profile welche von der Bluetooth SIG oder dem Hardwarehersteller des Peripheriegerätes entwickelt werden. Sie können einen oder mehrere Dienste bzw. Services enthalten.

- **Services** werden dazu verwendet um die zu übertragenen Daten in logische Einheiten zusammenzufassen. Sie enthalten eine oder mehrere Attribute (Characteristics).

- **Characteristic** ist ein einzelnes Attribut, wie z.B. die Sekunden, Minuten und Stunden einer Uhr. Characteristics sind die einzigen Parameter auf die Anwendungsentwickler später zugriff haben.

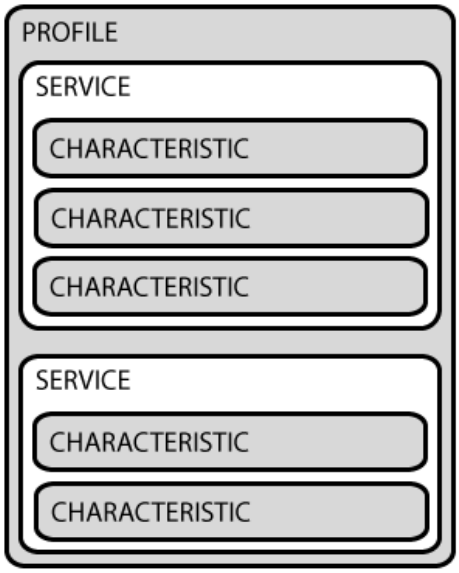

Abbildung 2.3: GATT Profilstruktur

2.2 Proximity Beacons Definition

Ein Proximity Beacon ist ein kleiner Bluetooth Sender, der anhand vorgegebener Spezifikationen in regelmäßigen, nach Bedarf konfigurierbaren, zeitlichen Abständen BLE Advertising Pakete verschickt. Durch Vorgaben eines Standards können diese von mobilen Empfängern als für sie relevante Pakete erkannt werden.

In diesem Zusammenhang von Näherungssensoren zu sprechen scheint zunächst etwas irreführend, da der eigentliche Sensor nicht der Beacon, sondern der Bluetoothempfänger des mobilen Endgerätes ist. Die Bezeichnung Beacon *(engl. Leuchtfeuer)* beschreibt die Funktionsweise der Sender ziemlich treffend. Bezogen auf dieses Bildnis, übermitteln die Sender, wie eine Art Leuchtturm, lediglich ihre Anwesenheit und Signalstärke in einer vordefinierten Entfernung, wodurch vom Empfänger die tatsächliche Entfernung zum Sender bestimmt werden kann.

2.3 Proximity Kategorien

Proximity Beacons haben einen Annäherungsmechanismus der den gemessenen Abstand zum Beacon in Kategorien unterteilt. Diese Kategorien können in Software Implementierungen als Auslöser für Aktionen wie z.B. Push-Nachrichten oder andere Verhaltensweisen benutzt werden. Die vier Entfernungskategorien sind:

- Immediate (Unmittelbar) : bis 50cm
- Near (Nahe) : bis 2m
- Far (Weit) : bis 30m
- Unknown/Out (Unbekannt) : außer Reichweite

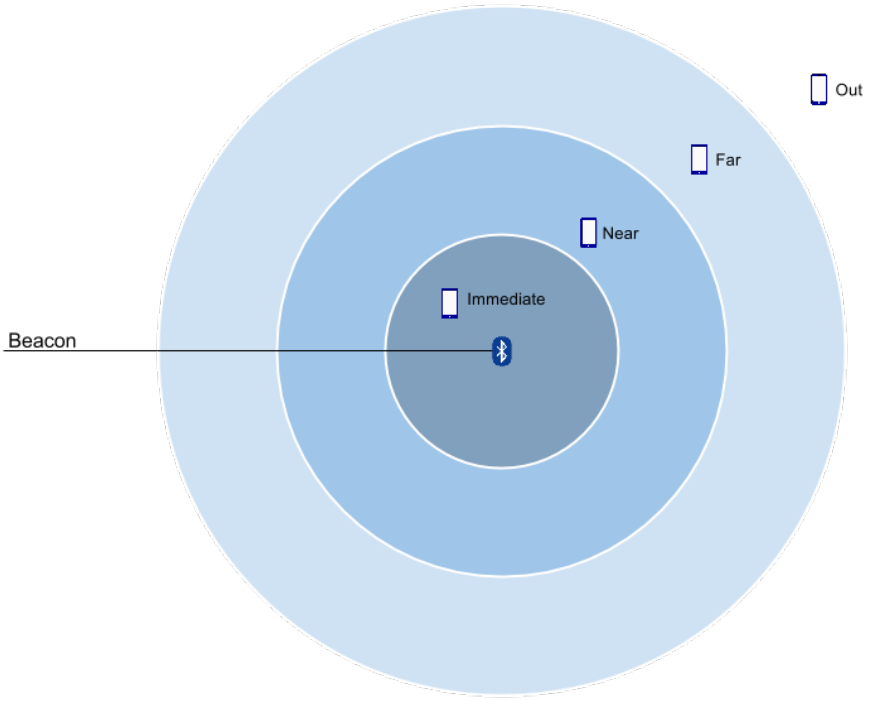

Abbildung 2.4: Entfernungskategorien

2.4 Der iBeacon Standard

Die Firma Apple hat mit **iBeacon** einen proprietären Standard für Innenraumnavigation entwickelt. iBeacon wird von iOS ab Version 7 und Android ab Version 4.3 unterstützt. Wie im Abschnitt BLE Broadcasting beschrieben, benutzt iBeacon Advertising Pakete um Informationen an eine Vielzahl von Empfängern zu verteilen. Der iBeacon Standard spezifiziert, wie BLE Beacons den Nutzdatenanteil eines BLE Advertising Pakets nutzen.

2.4.1 BLE Advertising Pakete

Ein komplettes BLE Advertising Paket kann bis zu 47 Byte lang sein und besteht aus:

- 1 Byte Präambel
- 4 Byte Kanaladresse
- 2-39 Byte Payload Data Unit (PDU)
- 3 Byte Fehlerkorrektur (CRC)

Die PDU hat ihren eigenen Header in dem Größe und Typ des Payloads angegeben werden. Die ersten 6 Byte des Payloads sind für die MAC Adresse reserviert.

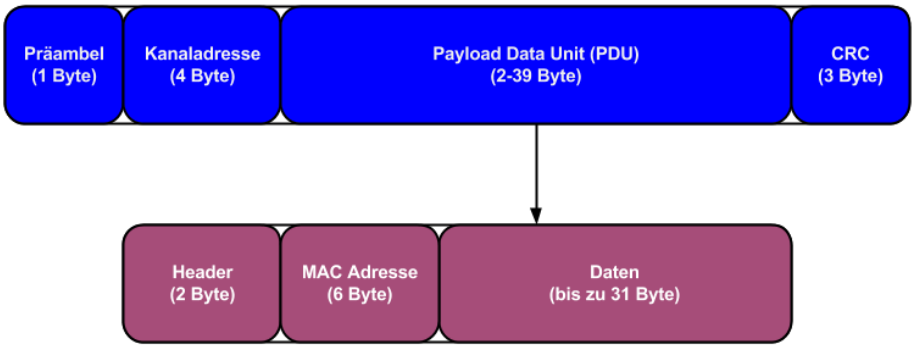

Abbildung 2.5: BLE Advertising Paketstruktur

2.4.2 iBeacon Pakete

Der Payload des BLE Advertising Paketes enthält schließlich das eigentliche iBeacon Paket, das 30 Byte groß ist. Es passt also gut in die übrigen 31 Byte des Payloads. Das Format des Paketes wurde von Apple festgelegt und macht das BLE Advertising Paket zu einem iBeacon Paket. Es enthält:

- 9 Byte iBeacon Prefix
- 16 Byte Proximity UUID
- 2 Byte Major
- 2 Byte Minor
- 1 Byte TX Power

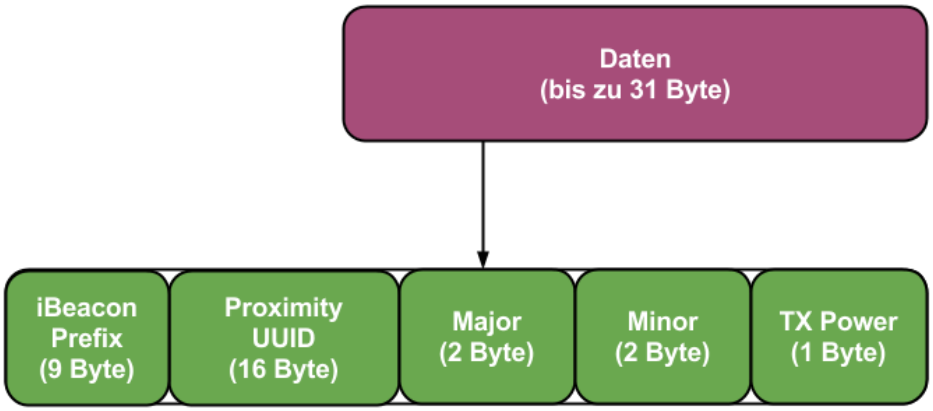

Abbildung 2.6: iBeacon Paket

2.4.2.1 iBeacon Prefix

Das iBeacon Prefix ist zum größten Teil von Apple festgelegt. Hier wird u.a. der Beacon Typ und die Hersteller ID angegeben. Das letzte Byte gibt zu dem die restliche Größe des Payloads an, der die für Entwickler relevanten Daten enthält.

2.4.2.2 Proximity UUID

Die Proximity UUID (UUID) ist ein 16 Byte langer Wert, der als 32 stellige Hexadezimalzahl dargestellt wird:

```
EBEFD083-70A2-47C8-9837-E7B5634DF523
```

Die UUID wird in der Regel dazu verwendet, den iBeacon eindeutig einer bestimmten Applikation zuzuweisen. So ist es Entwicklern möglich, einen für ihre Anwendung vorgesehenen iBeacon von anderen zu unterscheiden. Beispielsweise würden alle iBeacons einer Supermarktkette die gleiche UUID besitzen.

2.4.2.3 Major

Major und Minor sind vorzeichenlose Integer Werte zwischen 1 und 65535. Normalerweise werden iBeacons einer bestimmten Anwendung durch den Major Wert in Gruppen unterteilt. Um bei dem Beispiel der Supermarktkette zu bleiben, könnte man so Städte und Abteilungen identifizieren *(z.B. 100 : Berlin, 101 : Berlin-Fleisch, 102 : Berlin-Käse..., 200: Hamburg, 201 : Hamburg-Fleisch...)*.

2.4.2.4 Minor

Der Minor Wert wird dazu verwendet einen iBeacon eindeutig innerhalb seiner Gruppe zu identfizieren *(z.B. Major 101 also Berlin-Fleisch, Minor 01 : Wurstwaren, 02 : TK Geflügel, 03 : TK Rind...)*.

2.4.2.5 TX Power

Der TX Power Wert gibt an, wie stark das Signal des Beacons in einer Entfernung von einem Meter ist (z.B. -75dBm). Um die aktuelle Entfernung zum Beacon zu berechnen, wird dieser Wert mit der Signalstärke der zuletzt empfangenen BLE Pakete des Bluetooth-Empfängers vom Smartphone verglichen. Bei Bedarf kann der TX Power Wert manuell kalibriert werden.

2.5 Beacon Hardware

Beacon Hardware besteht aus einem Mikrocontroller, einem BLE Chipsatz sowie einer Batterie. Außerdem gibt es noch Varianten die über USB oder die Steckdose mit Strom versorgt werden. Auch das Smartphone selber kann als Proximity Beacon agieren, sofern dies vom Betriebssystem und dem verwendeten Software Development Kit (SDK) unterstützt wird. An dieser Stelle wird jedoch nur auf Batteriebetriebene „konventionelle“ iBeacons eingegangen.

Es gibt einige Faktoren, die die Lebensdauer der Batterie eines Proximity Beacons beeinflussen. Abhängig von dem verbauten Chipsatz und der verwendeten Firmware können diese über eine Konfigurationssoftware eingestellt werden. Dabei sind vor allem die Sendeleistung und das Advertising-Intervall interessant. Zu berücksichtigen ist auch, dass sich bei vielen iBeacon Modellen die Batterien nicht tauschen lassen.

Abgesehen vom äußeren Design unterscheidet sich die im Handel erhältliche Hardware im Wesentlichen an folgenden Merkmalen:

- Anzahl und Form der Batterien
- BLE Chipsatz
- Beacon Firmware

2.5.1 Batterietypen

Die Ladungskapazität der Batterien hat direkten Einfluss auf die maximale Betriebsdauer eines Proximity Beacons. Zudem haben die Anzahl und Form der Batterien großen Einfluss auf die Größe. Am Häufigsten sind Lithium Knopfzellen verbaut, die eine Ladungskapazität von bis zu 1000 mAh haben können. Einige Beacons verwenden auch Alkali-Batterien vom Typ AA, welche z.B. für TV Fernbedienungen oder Fotokameras üblich sind. Alkali-Batterien haben mit 2000 mAh eine wesentlich höhere Ladungskapazität. Dafür macht der Formfaktor die Beacons deutlich größer. Tabelle 2.1 zeigt die Batterietypen, die am häufigsten in Proximity Beacons verbaut werden.

Typ	Kapazität /mAh	Formfaktor	Ø /mm	Höhe /mm
CR2032	240	Knopfzelle	20	3,2
CR2450	620	Knopfzelle	24	5
CR2477	1000	Knopfzelle	24	7,7
AA	2000	Mignon	14,5mm	50,5

Tabelle 2.1: Proximity Beacon Batterien

2.5.2 BLE Chipsatz

Die Chips mit der BLE Funkeinheit werden derzeit am häufigsten von zwei großen Herstellen produziert, *Texas Instruments (USA)* und *Nordic Semiconductor (Norwegen)*. Hinzu kommen noch *BlueGiga (Finnland)* und *Gimbal/Qualcomm (USA)*, welche die Chips von Texas Instruments verwenden und diese mit einer eigenen Firmware versehen. Diese Chipsätze sind derzeit in ca. 95% aller angebotenen iBeacons verbaut: [3]

1. Texas Instruments: TI CC254x
2. Nordic Semiconductor: nRF51822
3. BlueGiga: BLE112 / BLE113
4. Gimbal (Qualcomm)

2.5.3 Beacon Firmware

Jeder Beacon hat seine eigene Firmware. Die Firmware enthält die Logik, die es der Hardware ermöglicht ihre Arbeit zu verrichten. Sie enthält die Implementierung des Proximity Beacon Standards wie z.B. iBeacon. Zusätzlich steuert die Firmware einige Eigenschaften, die direkten Einfluss auf die Lebensdauer der Batterie haben, die **Sendeleistung** und das **Advertising Intervall**.

2.5.3.1 Sendeleistung

Proximity Beacons übertragen ihr Signal mit einer festen voreingestellten Sendeleistung *(Tx Power)*, die in **dBm** *(Dezibel Milliwatt)* angegeben wird. Eine geringe Sendeleistung bedeutet zwar eine kürzere Reichweite jedoch eine längere Lebensdauer der Batterie, da weniger Strom verbraucht wird. Bei den meisten Beacons lässt sich die Sendeleistung mit einer proprietären Konfigurationssoftware auf Werte zwischen -30 dBm und 0 dBm einstellen. Abbildung 2.7 zeigt eine Annäherung an die zu erwartende Reichweite in Abhängigkeit zur Sendeleistung.

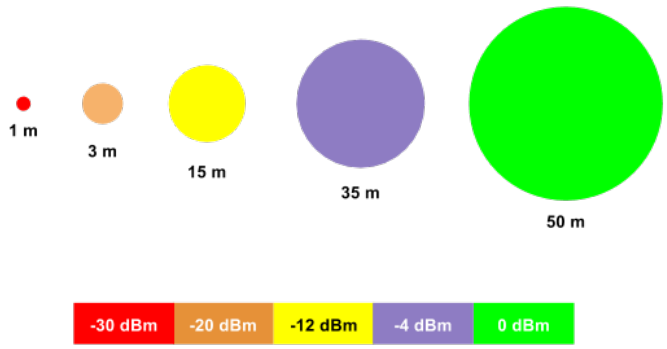

Abbildung 2.7: Sendeleistung und Reichweite von iBeacons

2.5.3.2 Advertising Intervall

Das Advertising Intervall bestimmt in welchen Zeitabständen Proximity Beacons ein Signal senden. Die Größe des Intervalls kann wie im *Abschnitt 2.1.2.1* beschrieben zwischen 2ms - 10s liegen. Je größer das Advertising Intervall ist, desto größer ist auch die Lebensdauer der Batterie. Nachteil von größeren Intervallen ist, das Anwendungen langsamer reagieren, wenn sie z.B. nur alle 2s ein Signal bekommen. Apple empfiehlt für iBeacons ein Advertising Intervall von 100ms. In der Praxis werden zugunsten der Betriebsdauer meist deutlich größere Intervalle gewählt.

3 Projektkonzept: MS-Seminar-Coach

Im folgenden Abschnitt wird beschrieben, was die Demoanwendung leisten soll. Er dient im Rahmen dieser Arbeit als Pflichtenheft im Sinne eines Softwareentwicklungsprozesses. Zu diesem Zweck wurde ein Anwendungsfall erschaffen, der nicht zwingend einen reellen Nutzen hat, aber auch nicht zu abstrakt ist.

3.1 Zielbestimmung

Die Anwendung MS-Seminar-Coach soll Studenten der Studiengangs Media Systems an der HAW Hamburg dabei helfen, sich in den Räumlichkeiten des Campus Finkenau zu orientieren und ihnen helfen ihr Studium zu organisieren. Es werden Proximity Beacons eingesetzt um zu überprüfen, ob ein Student während eines Seminars anwesend ist. Ist der Anwender zur geplanten Seminar-Zeit nicht im entsprechenden Raum, verschickt die Anwendung eine Erinnerungsnachricht. Bei Aufruf der Nachricht wird dem Anwender auf einem Raumplan angezeigt, wo das Seminar stattfindet. Anders als bei den meisten Proximity Beacon Anwendungen, wird also eine Aktion ausgelöst werden, wenn zu einer bestimmten Zeit ein bestimmter Beacon nicht in der Nähe ist. Die App soll lokal auf dem Smartphone laufen und den Studierenden zur anonymen Selbstkontrolle dienen.

3.1.1 Musskriterien

- Benutzer
  - kann Semester erstellen / löschen
  - kann Semester auswählen für Seminar-Ansicht
  - kann Seminare erstellen / löschen
  - kann Seminare auswählen für Kartenansicht
- App
  - eine Liste der erstellten Semester anzeigen
  - eine Liste der erstellten Seminare anzeigen
  - Seminare auf einem Gebäudeplan (Karte) anzeigen
  - **iBeacon Funktionen**
    - **Automatisches Ein- und Auschecken beim Betreten bzw. Verlassen eines Raumes**
    - **Anwesenheit mittels Auto-CheckIn überprüfen**
    - **Erinnerung wenn Seminar verpasst wird**

3.1.2 Wunschkriterien

- Benutzerfunktionen
  - Seminare nachträglich ändern

- Erweiterte Anwendungsfunktionen
  - keine simultanen Seminare möglich
  - Erweiterung der Coach-Funktionen durch
    * Auto Wecker bei Seminaren die frühmorgens stattfinden
    * Implementierung von Gamification Mechanismen (Belohnungssystem)

3.1.3 Abgrenzungskriterien

- keine Web-/Cloudanbindung

3.2 Produkteinsatz

3.2.1 Anwendungsbereiche

Studierende der Studiengänge Media Systems und Medientechnik an der HAW Hamburg können sich einen individuellen Vorlesungsplan erstellen. Studienanfänger können zudem die Kartenansicht der App als Orientierungshilfe verwenden.

3.2.2 Zielgruppen

Studierende am Department Medientechnik der HAW Hamburg.

3.2.3 Betriebsbedingungen

Aufgrund der iBeacon-Funktionen kann es keine Abwärtskompatibilität zu älteren Betriebssystemversionen geben.

- Betriebssystem: Android 4.4+
- Endgerät: *Bluetooth Smart* fähig
- Lebenszyklus: ständig
- Bluetooth: ständig

3.3 Umgebung

3.3.1 Software

- Betriebssystem
  - Android 4.4+ *(KitKat)*

3.3.2 Hardware

- Endgerät
  - Bluetooth Smart fähig
- Proximity Beacons
  - iBeacon Standard

3.4 Funktionen

3.4.1 Benutzerfunktionen

3.4.1.1 Semester-Liste anzeigen

Nach dem Starten der Anwendung bekommt der Nutzer eine Liste mit den von Ihm erstellten Semestern angezeigt. Es wird ein Infotext angezeigt falls noch keine Semester vorhanden sind.

3.4.1.2 Semester erstellen

Über ein „+“-Icon in der Titelleiste kann der Nutzer neue Semester hinzufügen. Nach einem Klick auf das „+“-Icon öffnet sich ein entsprechendes Formular. Zum Erstellen eines Semesters sind folgende Angaben erforderlich:

- Semester-Titel
- Erster Vorlesungstag *(Anfangsdatum)*
- Letzter Vorlesungstag *(Enddatum)*

3.4.1.3 Semester Löschen

Zum Löschen eines Semesters muss der Nutzer lange auf ein Semester-Eintrag klicken. Das Löschen muss über ein Dialog bestätigt werden.

3.4.1.4 Semester auswählen

Mit einem kurzen Klick auf ein Semester öffnet sich eine neue Ansicht, in der die Seminare des Semesters angezeigt werden.

3.4.1.5 Seminar-Liste anzeigen

Nach Auswahl eines Semesters wird eine Seminar-Liste angezeigt. Die Titelleiste zeigt in diesem Fall den Namen des ausgewählten Semesters. Jeder Eintrag soll zusätzlich einen Button zum Aufrufen der Karte haben. Falls die Liste leer ist, wird ein kurzer Infotext angezeigt.

3.4.1.6 Seminar erstellen

Über ein „+“-Icon in der Titelleiste kann der Nutzer Seminare hinzufügen. Nach Klicken auf das „+“-Icon öffnet sich das entsprechende Formular. Zum Erstellen eines Seminars sind folgende Angaben erforderlich:

- Seminar-Titel
- Wochentag des Seminars
- Start-Uhrzeit des Seminars
- End-Uhrzeit des Seminars
- Seminarraum

3.4.1.7 Gebäudeplan anzeigen

Der Gebäudeplan zeigt einen Umriss des Erdgeschosses der Finkenau 35, auf dem die Räume des Studiengangs Media Systems und deren Raumnummern eingezeichnet sind. Die Kartenansicht soll sich mittels Gesten steuern lassen (Pinch-Zoom / Panning). Points Of Interest (POIs) werden mit blinkenden Punkten markiert. Die Farben der Punkte sind in folgende Ereignisse kategorisiert:

- Positives Ereignis: Grün
  - Auto-CheckIn-Information

- Neutrales Ereignis: Blau
  - Seminarraum anzeigen
  - Auto-CheckOut-Information
- Negatives Ereignis: Rot
  - Verpassen eines Seminars

3.4.2 Erweiterte Anwendungsfunktionen

3.4.2.1 Auto-CheckIn-System

Die Anwendung soll sich beim Betreten eines Seminarraumes automatisch in diesem Ein- und beim Verlassen wieder Auschecken. Dem Nutzer wird dies mit einer Mitteilung bestätigt. Ein Klick auf die Mitteilung öffnet die Kartenansicht auf dem der betroffene Raum markiert ist. Das CheckIn-System benutzt Proximity Beacons zur Identifizierung der Räume.

3.4.2.2 Erinnerungsdienst

Befindet sich der Nutzer während eines Seminar Zeitraums nicht im richtigen Raum, verschickt die Anwendung eine Erinnerungsmitteilung. Ein Klick auf die Mitteilung öffnet die Kartenansicht, welche in diesem Fall anzeigt, wo gerade das Seminar verpasst wird.

3.5 Daten

Zusammenfassend hier nochmal sämtliche Daten, die in der Anwendung vorkommen.

3.5.1 Benutzerdaten

3.5.1.1 Semester

- Semester-Titel
- Anfangsdatum
- Enddatum

3.5.1.2 Seminar

- Seminar-Titel
- Wochentag
- Start-Uhrzeit
- End-Uhrzeit
- Seminarraum

3.5.2 Anwendungsdaten

3.5.2.1 Kartendaten

- SVG-Datei

3.5.2.2 Raum

- Raum-Titel
- Kartenkoordinaten
  - X-Koordinate
  - Y-Koordinate

3.5.2.3 Beacons

- ID
- UUID
- Major
- Minor

3.6 Leistungen

3.6.1 Fehlererkennung

Beim Erstellen von Einträgen durch den Nutzer sind keine inkonsistenten Eingaben möglich. Es dürfen keine Textfelder leer bleiben und zeitliche Angaben müssen Sinn ergeben, d. h. ein Ende darf nicht vor dem Beginn liegen etc. .

3.7 Benutzeroberfläche

3.7.1 Struktur

Im Folgenden wird die Struktur der Benutzeroberfläche (GUI) für die Benutzung der Anwendung gezeigt. Fehlerhafte Eingaben haben in der Regel ein Nicht-Setzen des Eingabefelds und die Ausgabe eines Android-Toasts zur Folge.

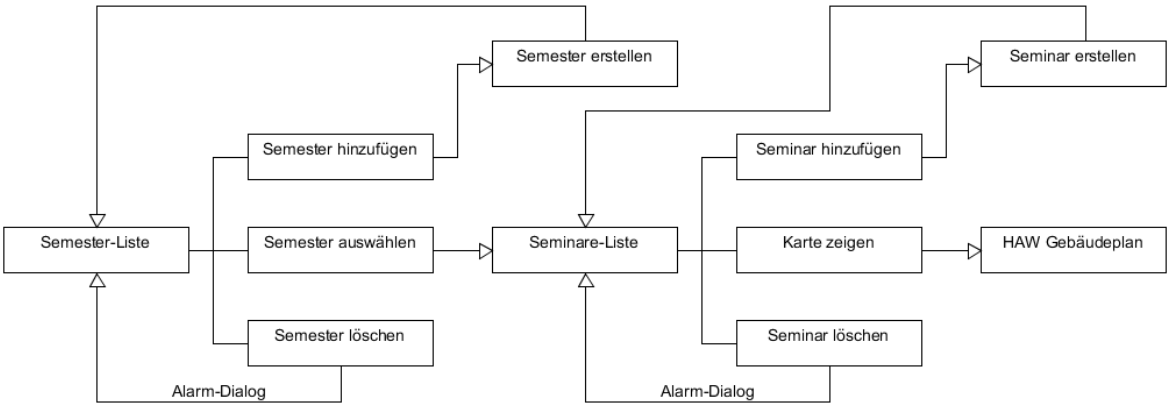

Abbildung 3.1: Flussdiagramm GUI Struktur

3.7.2 Farben

Die Farben des GUI sollen den HAW Farben entsprechend Blautöne sein. Auch der Gebäudeplan soll wenn möglich in Blautönen gehalten werden. Hierdurch soll ein optischer Wiedererkennungswert zur HAW entstehen.

Abbildung 3.2: MS-Seminar-Coach Farbpalette

4 Anwendungsentwicklung

In diesem Abschnitt ist die Entwicklung der Demo-Anwendung **MS-Seminar-Coach** dokumentiert. Es werden vor allem die technische Umsetzung sowie die Anforderungen und Lösungsansätze der einzelnen Softwarekomponenten beschrieben. Der komplette Quellcode der Anwendung befindet sich als Android Studio Projekt auf dem beiliegenden Datenträger. Die Anwendung wurde ausschließlich für Android Systeme ab Version 4.4+ (API Level 19) entwickelt. Auf Abwärtskompatibilität wurde keinen Wert gelegt, da die Schlüsseltechnologie BLE frühestens ab Version 4.3 (API Level 18) von Android unterstützt wird.

4.1 Software Tools

4.1.1 Entwicklungsumgebung

Als IDE für die Entwicklung der Anwendung wurde Android Studio, zuletzt in der Version 1.3 verwendet. Android Studio basiert auf der Java IDE „IntelliJ IDEA“ der Firma JetBrains aus Tschechien und ist Googles IntelliJ-Ableger speziell für die Entwicklung von Android Applikationen. Android Developer Tools (ADT) wird als Plug-In für Eclipse zwar noch weiterentwickelt, jedoch ist der Support von Googles Seite kaum vorhanden.

4.1.2 Grafikbearbeitung

Für die Erstellung des Gebäudeplans, dem Erdgeschoss der Finkenau 35, wurde das Vektor-Grafik-Programm *Adobe Illustrator CC* verwendet. Als Vorlage diente eine Grafik

von der Webseite der HAW Hamburg. Der Gebäudeplan ist, wie der Rest der Anwendung, in verschiedenen Blautönen gehalten, um CI der HAW mit einzubeziehen. Auf die Einzeichnung von Türen wurde zu Gunsten der Übersichtlichkeit bewusst verzichtet. Um nicht vom Rest der Anwendung abzuweichen, wurde für die Darstellung der Raumnummern im Gebäudeplan der Android eigene Standart-Font „Roboto“ verwendet, der im Internet frei verfügbar ist.

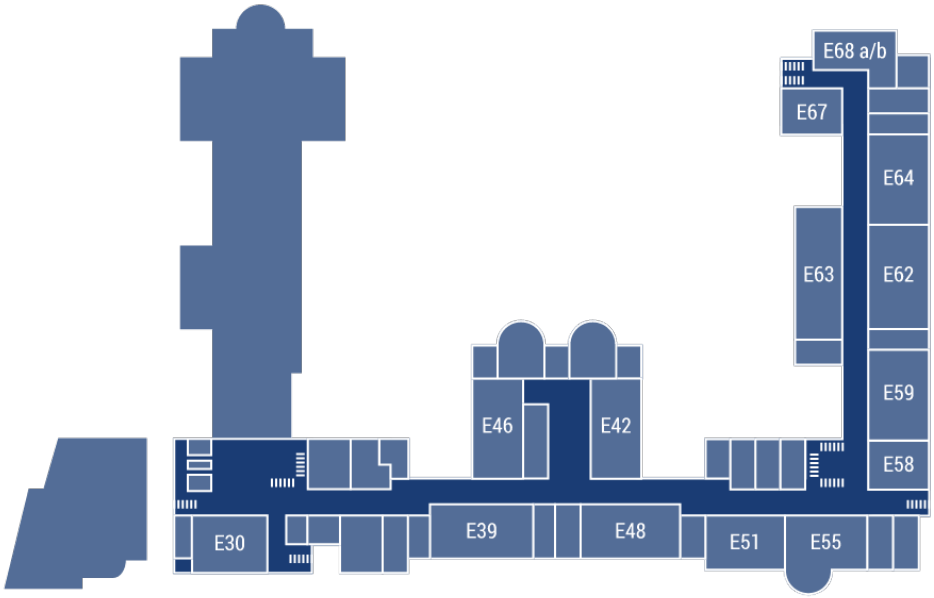

Abbildung 4.1: HAW Gebäudeplan: Finkenau 35, Erdgeschoss

4.1.3 Bibliotheken

Es gibt für das Android SDK eine Vielzahl von Drittanbieter Bibliotheken. Bei der Implementierung der Demo Anwendung für diese Arbeit wurde weitgehend auf den Einsatz solcher Bibliotheken verzichtet. Speziell für die Implementierung von iBeacon bietet fast jeder iBeacon-Anbieter sein eigenes SDK an, welche jeweils Vor- aber auch Nachteile haben. Die Anbieter SDKs verwenden häufig ein Backend mit Cloud-Anbindung, die Entwickler dazu zwingen ihre iBeacons bei dem jeweiligen Anbieter zu registrieren. Ein manuelles konfigurieren der Beacons ist dann meist nicht mehr möglich. Für die Implementierung

der iBeacon Funktionen wurde deshalb eine Open Source Bibliothek verwendet. Für die Konvertierung der Grafiken kommt zudem ein SVG-Parser zum Einsatz.

4.1.3.1 SVG Android Library

SVG Android ist ein SVG-Parser, der SVG Dateien *(Scaleable Vector Graphics)* in Android-Canvas-Pfade konvertiert. Die Bibliothek ist speziell dafür entwickelt worden, Grafiken aus Android in verschiedenen Bildschirmgrößen darstellen zu können. Auch das Problem unzureichender Pixeldichte beim Vergrößern einer Grafik (Zoom), tritt bei der Verwendung von Vektorgrafiken nicht auf.

Die Bibliothek unterstützt nur einen Teil der SVG Basic 1.1 Spezifikation des W3C (World Wide Web Consortium). Bei dem Erstellen der Vektorgrafik muss darauf geachtet werden, das die Vorgaben der Bibliothek streng eingehalten werden. Die in Illustrator erstellte Vektorgrafik des Gebäudeplans wurde als SVG-Datei exportiert. In den SVG Exportoptionen von Illustrator kann SVG Basic 1.1 als Zielformat ausgewählt werden.

Es wurde besonders Wert darauf gelegt, dass die Grafik des Gebäudeplans als Vektor-Datei vorliegt, um ein verpixeln der Grafik bei dem Heranzoomen in der Anwendung zu verhindern. So ist es möglich, den Gebäudeplan unabhängig vom Zoom-Faktor gestochen scharf abzubilden. Mit *SVG Android* kann die Grafik in dem Canvas Container der Anwendung später stufenlos transformiert werden *(scale, translate, rotate)* ohne das die Bildqualität beeinflusst wird.

Quelle: `https://code.google.com/p/svg-android/`

4.1.3.2 Android Beacon Library

Für die Implementierung der iBeacon Funktionen des MS-Seminar-Coach wurde die Open Source Bibliothek *Android Beacon Library* verwendet. Die Bibliothek wurde zum größten Teil von der Firma Radius Networks entwickelt und ist im Vergleich zu den meisten Anbieter eigenen SDKs, hervorragend dokumentiert. Auch der Support der Entwickler über über die Entwicklerplattform *StackOverflow* ist gut.

Ursprünglich wurde die *Android Beacon Library* als reine iBeacon Bibliothek für An-

droid Systeme entwickelt und hieß bei der ersten Veröffentlichung noch *iBeacon Android Library*. Auch die Funktionalität und Handhabung der Bibliothek ist weitgehend der Original-Implementierung aus Apples iOS SDK nachempfunden. Nach kurzer Zeit wurde das iBeacon Android Repository, vermutlich aus rechtlichen Gründen, von GitHub entfernt.

Radius Networks reagierte in dem es eine Open Source Spezifikation für Proximity Beacons veröffentlichte die *AltBeacon* heißt. Die Library wurde in *Android Beacon Library* umbenannt und erkennt mit der Voreinstellung nur Proximity Beacons welche den *AltBeacon* Standard nutzen. Die Bibliothek wurde mit einem Beacon-Parser ergänzt, der es ermöglicht beliebige Beacon Layouts hinzuzufügen. Um iBeacon Pakete erkennen zu können, muss das Layout der iBeacon Spezifikation nun vom Anwendungsentwickler selbst hinzugefügt werden.

Der Funktionsumfang der *Android Beacon Library* ist beachtlich und wächst weiter. Durch den Beacon-Parser könnten theoretisch sogar Beacons mit unterschiedlichen Spezifikationen in einer Anwendung verwendet werden. Detaillierte Informationen inkl. Java-Doc sind auf der Projekt Webseite einzusehen.

Quelle: `http://altbeacon.github.io/android-beacon-library/index.html`

4.2 Hardware

4.2.1 Testgerät

Die Anwendung wurde ausschließlich auf einem Motorola „Moto G“ der ersten Generation getestet. Auf dem Testgerät war Android 5.01 (API Level 21) installiert.

4.2.2 Proximity Beacons

Zur Umsetzung des Projekts stehen 10 *iBKS Beacons* (Modell: IBK10) der Firma *Accent-Systems* zur Verfügung. Beschafft wurden die Beacons über *Sensorberg* in Deutschland.

Sensorberg fungiert also nur als Dienstleister und bietet ein eigenes iBeacon SDK inklusive Cloud-Backend an.

4.2.2.1 Beacon Standorte

Es werden in 8 Seminarräumen des Studiengangs Media Systems der HAW Hamburg iBeacons installiert. 2 Beacons werden als Reserve behalten, falls in größeren Seminarräumen die Sendeleistung eines einzelnen Beacons unzureichend ist. In folgenden Räumen werden die iBeacons installiert:

- E39 : Seminarraum
- E42 : Seminarraum
- E46 : PC-Pool 2
- E48 : Seminarraum
- E59 : PC-Pool 3
- E62 : PC-Pool 4
- E63 : Digitaltechnik
- E64 : Netzwerk-Technik

4.2.3 Konfiguration der Beacons

4.2.3.1 Platzierung und Sendeleistung der Beacons

Jeder Raum soll wenn möglich mit nur einem Beacon versehen werden. Die iBKS Beacons lassen sich auf 4 verschiedene Sendeleistungen einstellen:

- Stufe 0 : 0.3m bis 5m (-23dBm)

- Stufe 1 : 25m bis 45m (-6dBm)
- Stufe 2 : 45m bis 60m (0dBm)
- Stufe 3 : 60m bis 120m (+4dBm)

Die Angaben der möglichen Entfernungen sind nur eine Richtlinie. Die präzise Reichweite hängt immer von der Umgebung ab, da das Signal von verschiedenen Faktoren, wie z.B. vorhandene WLAN Netzwerke oder reflektierenden Materialien wie Glas, beeinträchtigt werden kann. Auch die Leistung des Empfängers am Smartphone hat großen Einfluss auf die maximale Reichweite eines Beacons.

Damit das iBeacon Check-In-System möglichst genau funktioniert, werden die Beacons in der Mitte der Gebäude Außenwand platziert. Der Radius zum Einchecken in den Raum soll später ca. 5m betragen. Die Sendeleistung der Beacons wird auf -6dBm, also 25m bis 45m gesetzt. Die Signale der Beacons sind theoretisch also weit über die Grenze eines Raumes zu empfangen.

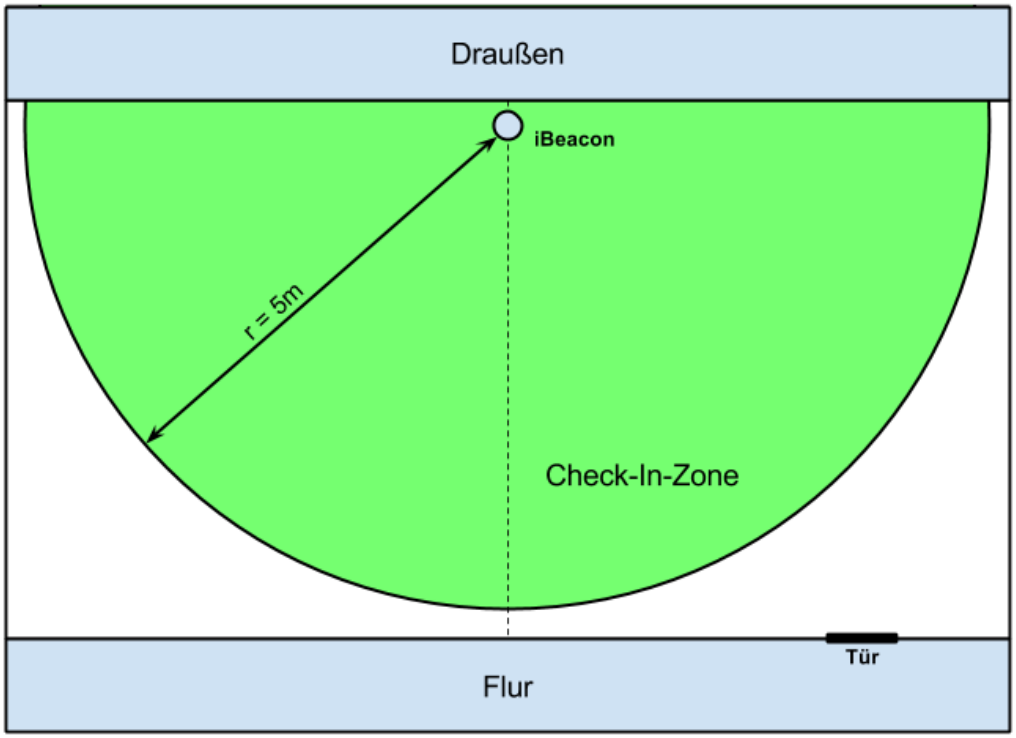

Abbildung 4.2: Versuchsaufbau iBeacon Platzierung

4.2.3.2 Advertising Intervall

Die Einstellung des Advertising Intervalls hat auf mehrere Bereiche der Anwendung Einfluss. Offensichtlich ist, dass wie bereits erwähnt, bei größeren Advertising Intervallen die Reaktionszeit der Anwendung steigt, aber dafür auch die Lebensdauer der Batterie erhöht wird. Auf Seite des Android Beacon SDKs muss man jedoch beachten, dass kleinere Advertising Intervalle eine präzisere Entfernungsbestimmung ermöglichen, da mehr RSSI-Samples (Received Signal Strength Indication) für die Berechnung der Entfernung gesammelt werden können. Für die MS-Seminar-Coach Anwendung heißt das, dass das automatische Check-In-System genauer wird, je kleiner das Advertising Intervall ist. Jedoch soll die Lebensdauer der Batterien möglichst über ein ganzes Semester andauern. Da genaue Erfahrungswerte fehlen, wird das Advertising Intervall zunächst auf 670ms gesetzt.

4.2.3.3 Identifier: UUID, Major, Minor

Die UUID für den MS-Seminar-Coach ist frei gewählt und wurde mit einem Online-Tool auf `https://www.uuidgenerator.net/` generiert. Die Webseite erzeugt bei jedem Nachladen der Seite eine zufällige UUID.

- Application UUID = **28415da5346b-459d-9795-0a30092ccd30**

Zu Gunsten der Übersichtlichkeit wird für die UUID in der nachfolgenden Tabelle einfach „AppID“ notiert. Der Major-Wert wird in diesem Fall benutzt, um das Stockwerk des Gebäudes zu identifizieren. Es wird der Wert „1“ für das Erdgeschoss gewählt, um gegebenenfalls die „0“ für das Untergeschoss verwenden zu können. Der Minor-Wert wird den Raum-Nummern entsprechend gesetzt. Da das *iBKS-Config-Tool* mit Hexadezimal-Werten arbeitet gibt es für den Minor-Wert zusätzlich eine entsprechende Spalte. Der Vollständigkeit halber, wurde zuletzt das Advertising Intervall und die Stufe der Sendeleistung notiert.

Beacon-Nr.	Raum	UUID	Maj.	Min.-Int	Min.-Hex	A.I./ms	Pwr.-Lv.
1	E39	AppID	1	39	27	670	1
2	E42	AppID	1	42	2A	670	1
3	E46	AppID	1	46	2E	670	1
4	E48	AppID	1	48	30	670	1
5	E59	AppID	1	59	3B	670	1
6	E62	AppID	1	62	3E	670	1
7	E63	AppID	1	63	3F	670	1
8	E64	AppID	1	64	40	670	1

Tabelle 4.1: MS-Seminar-Coach Beacon Konfiguration

4.2.3.4 Konfigurations-Apps

Die Konfiguration der iBeacons mittels der von *Sensorberg* zur Verfügung gestellten Konfigurations-App *(Sensorberg BeaCfg)* ist für experimentelle Zwecke leider unzureichend. Die Anwendung greift aktuell in jedem Fall auf das *Sensorberg* Cloud-Backend zu. Die Identifier, UUID, Major und Minor werden bei der Registrierung in der *Sensorberg* Cloud automatisch zugewiesen. Manuell lassen sich lediglich die Sendeleistung und das Advertising Intervall einstellen.

Um die Beacons dennoch nach seinen eigenen Vorstellungen zu konfigurieren, wurde ein simpler Trick angewandt und das *iBKS-Config-Tool*, die Konfigurations-App des Beacon Herstellers *Accent Systems* verwendet. Um mit dem Config Tool eine Verbindung zu einem Beacon herzustellen, muss bei diesem der Konfigurations-Modus aktiviert werden. Hierfür wird der Beacon geöffnet und ein Knopf auf der Platine betätigt. Anschließend kann das Tool eine Verbindung zu dem Beacon herstellen und die Services vom GATT-Profil erkennen. Bei dem Speichern der Configuration werden die neuen Characteristic-Werte auf den Beacon geschrieben (siehe Abschnitt 2.1.3.2 GATT Profilarchitektur).

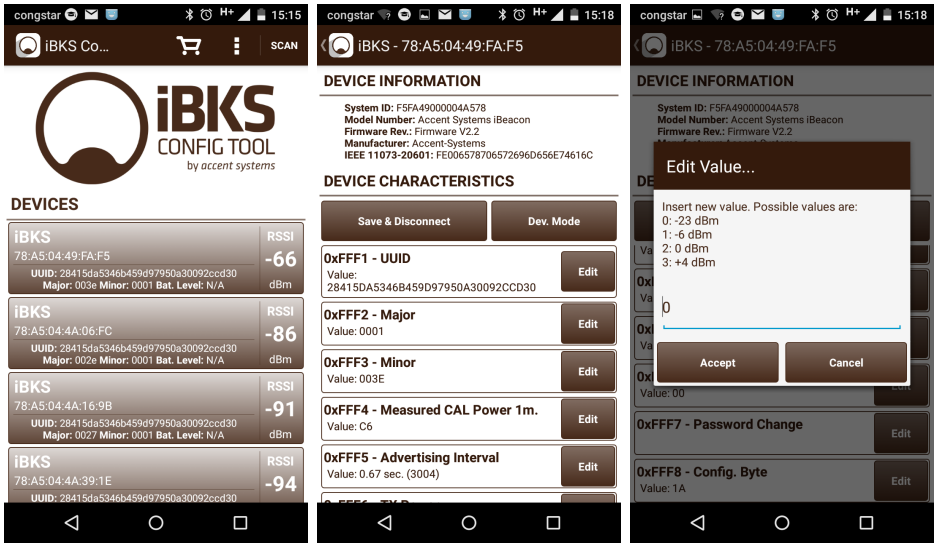

Abbildung 4.3: iBKS Config Tool

4.3 Softwareentwicklung

Im folgenden Abschnitt wird die Software Architektur sowie die Lösungswege einiger Anwendungskomponenten gezeigt. Um den Rahmen dieser Arbeit gerecht zu werden, wurde dies auf das Wesentliche beschränkt. Für diesen Abschnitt werden grundlegende Kenntnisse objektorientierter Programm-Paradigmen, Java sowie Basis wissen über das Android SDK vorausgesetzt.

4.3.1 Projektübersicht

Das Java-Paket mit dem Quellcode der Anwendung enthält folgende Klassen:

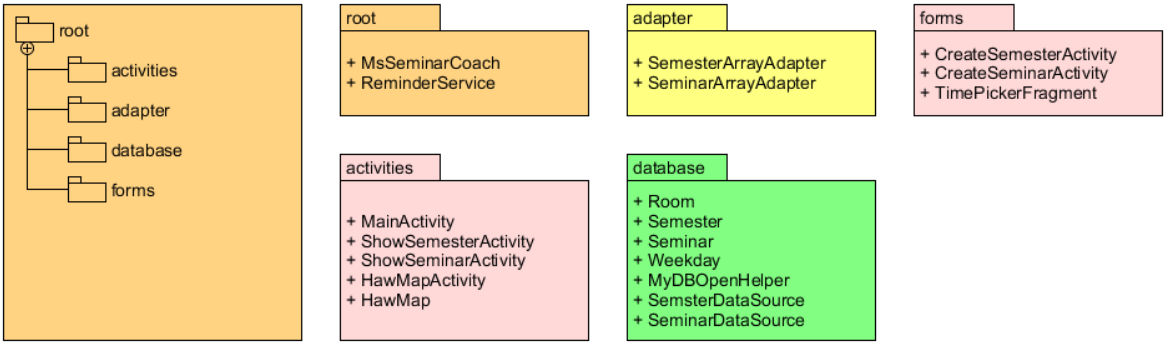

Abbildung 4.4: UML Paketdiagramm: de.janhilbig.hawcoursecoach

4.3.2 Anwendungsdaten

4.3.2.1 User-Daten und Asset-Daten

Alle Datenbank Klassen und Objekt-Modelle sind in dem Java-Pakage *de.janhilbig.hawcoursecoach.database* enthalten. Die Anwendungsdaten sind in User-Daten und Asset-Daten unterteilt worden. User-Daten sind diejenigen, die vom Nutzer selber eingegeben Werden. Beim MS-Seminar-Coach sind dies Semester und Seminare. Asset-Daten sind Daten welche die Anwendung kennen muss, aber vom Nutzer nicht verändert werden können. In diesem Fall die Seminarräume und Wochentage.

Die User-Daten werden von der Anwendung in einer SQLite Datenbank gespeichert, welche im Android SDK bereits enthalten ist. Für die Asset-Daten wurden lediglich Objekt-Modelle definiert, die eine statische Liste mit den benötigten Daten bereithalten. Die Modell-Klassen *Semester* und *Seminar* implementieren das Interface *Parcelable* des Android SDKs um die Objekte in ein Parcel-Objekt kapseln zu können. Dies ermöglicht es, individuelle Objekte als Intent-Extra zwischen Android Activities zu verschicken.

Code Beispiel:

```
...
Semester semester = new Semester(titel, anfang, ende);
Intent resultIntent = new Intent();
resultIntent.putExtra(NEW_SEMESTER, semester));
...
```

4.3.2.2 Datenbankzugriffe

Die Datenbankzugriffe sind in folgenden Klassen implementiert:

- MyDBOpenHelper
- SemesterDataSource
- SeminarDataSource

MyDBOpenHelper
MyDBOpenHelper erbt die Eigenschaften der Superklasse SQLiteOpenHelper die Teil des Android SDKs ist. Hier wird die SQLite Datenbank konfiguriert und die Relationen für Semester und Seminare erstellt.

SemesterDataSource
SemesterDataSource steuert die Zugriffe auf die Relation Semester. Sie enthält Methoden zum Erstellen, Löschen und Abfragen der Semester. Zusätzlich gibt es eine private Methode die den Datenbank-Cursor in ein Semester-Objekt konvertiert.

Abfragemethoden:

- `List<Semester> getAllSemester();`
- `List<String> getTitleById(long semesterID);`

SeminarDataSource

SeminarDataSource ist äquivalent zu SemesterDataSource. Sie hat jedoch etwas andere Abfragemethoden:

- `List<Seminar> getAllSeminars();`
- `List<Seminar> getSemesterSeminars(long semesterID);`

Abbildung 4.5: MS-Seminar-Coach Logo

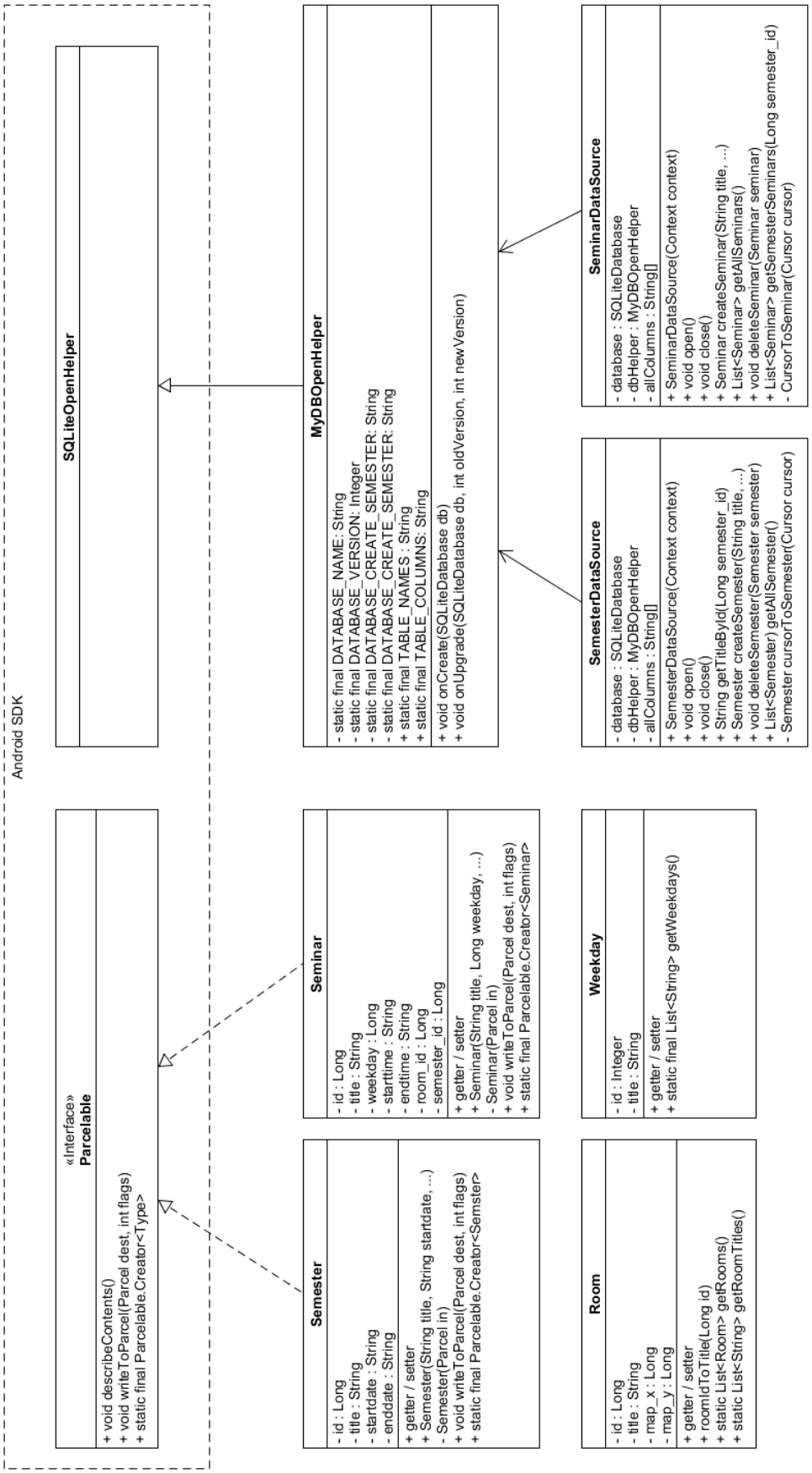

Abbildung 4.6: UML Klassendiagramm 1: Datenbank

4.3.3 Activities (GUI)

Die Navigation über das GUI ist genau nach der Konzeptvorgabe umgesetzt worden (siehe Abbildung 3.1). Grundlage für das funktionale Layout der Anwendung ist die Action-Bar und List-Views. Alle Activities erben die Action-Bar und zugehörige Funktionen von der Superklasse *ActionBarActivity* des Android SDK. Im MS-Seminar-Coach hat die Action-Bar zwei Funktionen:

- Anzeige des Namens der aktuellen Ansicht
- Menüfunktion für das Hinzufügen von Semestern und Seminaren

4.3.3.1 MainActivity

Die **MainActivity** ist bei dem MS-Seminar-Coach leer und nie zu sehen. Sie wird nur gebraucht um das Verhalten der App im Android Launcher zu beeinflussen. Sie vererbt über den Intent-Filter im Android Manifest den Applikationsnamen an alle anderen Activities. Andernfalls würde unter Android Lollipop anstatt des Applikationsnamens der Name der zuerst gestarteten Activiy im Android Launcher und Task-Manager stehen.

4.3.3.2 ShowSemesterActivity

Die Semester-Anzeige implementiert einen *ListView* zur Darstellung der in der Datenbank vorhandenen Semester. Der *ArrayAdapter* des List-Views wurde in der Klasse **SemesterArrayAdapter** so angepasst, dass dieser Semester-Objekte entsprechend formatiert und diese anzeigen kann. Die Einträge der Datenbank werden in einer Array-Liste zwischengespeichert, welche dann dem *ArrayAdapter* des List-Views zugewiesen wird.

```
dataSource = new SemesterDataSource(this);
        try {
            dataSource.open();
        } catch (SQLException dbException){
            Log.w("dbError", "Datenbankfehler: " + dbException);
        }
// Get semesters from database
values = (ArrayList<Semester>) dataSource.getAllSemester();

// Set SemesterArrayAdapter for listView
adapter = new SemesterArrayAdapter(this, values);
listView.setAdapter(adapter);
```

Für das Erstellen eines neuen Semesters wurde der Action-Bar ein Icon hinzugefügt. Beim Klicken des Icons wird die **CreateSemesterActivity** aufgerufen. Für das Auswählen und Löschen eines Semesters sind die Methoden für kurze und lange Klicks des List-View implementiert worden. Bei der Auswahl eines Semesters wird die Semester-ID der Position des List-Views an die **ShowSeminarActivity** übergeben.

4.3.3.3 ShowSeminarActivity

In der Seminar-Liste werden nur die Seminare des Ausgewählten Semesters angezeigt.

```
...
values = (ArrayList<Seminar>) dataSource.getSemesterSeminars(semester_id);
...
```

Mit der von **ShowSemesterActivity** übergebenen Semester-ID wird zusätzlich der Titel des Semesters in der Action-Bar gesetzt.

SeminarArrayAdapter
Die Seminar-Liste hat für jeden Eintrag ein Button über den der Raum des Seminars auf dem Gebäudeplan angezeigt werden kann. Hierfür übergibt der **SeminarArrayAdapter** die ID des Seminarraumes an die **HawMapActivity**.

```
...
// listener for row mapButton
final Button button = (Button) rowView.findViewById(R.id.seminar_button);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(getContext(), HawMapActivity.class);
        intent.putExtra(ROOM_ID, values.get(position).getRoom_id());
        intent.putExtra(COLOR_ID, 3);
        context.startActivity(intent);
    }
});
...
```

4.3.3.4 Konsistenzprüfung der Nutzereingaben

Beim Entgegennehmen der Nutzereingaben sollen nach Möglichkeit keine inkonsistenten Datensätze entstehen. Bei der Verwendung von Drop-Down-Listen (in Android *Spinner*) stellt dies kein Problem dar. Da der Nutzer Eingaben nur auswählen kann, können keine ungültigen Werte eingegeben werden. Bei Text-Eingabefeldern sollten die vom Nutzer eingegebenen Werte jedoch überprüft werden, bevor diese in der Datenbank gespeichert werden. Bei dem MS-Seminar-Coach müssen vom Nutzer zudem Zeitangaben gemacht werden mit denen später weitergearbeitet wird. Gibt der Nutzer hier falsche Werte ein, würde die Anwendung abstürzen. Zu diesem Zweck wurden in den verantwortlichen Formular-Activities **CreateSemesterActivity** und **CreateSeminarActivity** folgende Mechanismen implementiert:

- Deaktivieren des Speicher-Buttons bei Unvollständigkeit
- Erzwingen erweiterter Eingabemethoden *(DatePicker, TimePicker)*
- Erzwingen der Eingabereihenfolge
- Vergleichen der Zeitangaben

Um den Speicher-Button nur zu aktivieren wenn alle Felder ausgefüllt sind, wurde ein *TextWatcher* implementiert. Die Standard-Implementierung ist dafür so verändert worden, das der Methode nur noch die Text-Felder übergeben werden müssen. Hier ein Beispiel aus **CreateSemesterActivity**:

```
private void myTextWatcher(final EditText editText) {
editText.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(
                CharSequence s, int start, int count, int after) {
    // nothing to do here
    }
    @Override
    public void onTextChanged(CharSequence s, int start, int before, int
        count) {
        if (!isEmpty(semestertitel) && !isEmpty(semesteranfang) && !isEmpty
            (semesterende)){
            saveButton.setEnabled(true);
        } else saveButton.setEnabled(false);
    }
    @Override
    public void afterTextChanged(Editable s) {
    // nothing to do here
    }
});
}
```

Um den Nutzer bei der Eingabe der Zeiten zu zwingen, die *Picker* zu benutzen, wurden die KeyListener der Eingabefelder deaktiviert.

```
...
// User can't input corruptet date values
        semesteranfang.setKeyListener(null);
        semesterende.setKeyListener(null);
...
```

Das Erzwingen der Eingabereihenfolge und Vergleichen der Werte bei Datums- und Zeitangaben, sind Bedingungen des Eingabefelds für das Ende.

```
...
if (startDate != null) {
   if (!endDate.before(startDate)) {
   semesterende.setText(date);
   } else { Toast.makeText(getApplicationContext(), "Ende vor Anfang!?",
       Toast.LENGTH_SHORT).show();
     }
  } else { Toast.makeText(getApplicationContext(), "Bitte zuerst das
     Anfangsdatum angeben!", Toast.LENGTH_SHORT).show();
    }
...
```

4.3.3.5 HawMapActivity

Die **HawMapActivity** dient als Controller für die **HawMap**. Hier wird der *View* instanziiert und die Informationen der aufrufenden *Intents* an den *View* durchgereicht. Alles Ausschlaggebende passiert ausschließlich im *View* **HawMap** selbst.

4.3.3.6 HawMap

Die Klasse **HawMap** ist ein *SurfaceView*, der das Interface *Runnable* implementiert. *SurfaceView* ist eine Subklasse von *View*, die es ermöglicht die Zeichnungen des Views in einem zweiten GUI-Thread laufen zu lassen. Alle anderen Views der GUI werden also nicht von der **HawMap** beeinflusst und umgekehrt. Mit dem Interface *Runnable* wird das Verhalten des Threads, über die Activity-Callbacks (onResume(), onPause()) der **HawMapActivity**, gesteuert. Im Konstruktor der **HawMap** kommt der SVG-Parser (Android SVG Library) zum Einsatz.

```
...
// parse svg
SVG svg = SVGParser.getSVGFromResource(getResources(), R.raw.
    haw_map_neu_dunkel);
// get picture
picture = new Picture();
picture = svg.getPicture();
...
```

Pinch-Zoom und Panning

Der Pinch-Zoom und das Panning wurden mit dem *ScaleGestureDetector* implementiert. Für das Zoomen gibt es eine private Klasse **ScaleListener** die von *ScaleGestureDetector.SimpleOnScaleGestureListener* abgeleitet wurde. Für das Panning, also das Verschieben der Grafik, wird die Methode *onTouchEvents(MotionEvent ev)* überschrieben und die einzelnen Ereignisse ebenfalls an den *ScaleGestureDetector* weitergereicht.

```
...
@Override
    public boolean onTouchEvent(MotionEvent ev) {
        // Let the ScaleGestureDetector inspect all events.
        mScaleDetector.onTouchEvent(ev);

        final int action = ev.getAction();
        switch (action & MotionEvent.ACTION_MASK) {
            case MotionEvent.ACTION_DOWN: { ... }
            case MotionEvent.ACTION_MOVE: { ... }
            ...
        }
    }
...
```

So sehen die GUI-Activities in der fertigen Anwendung aus:

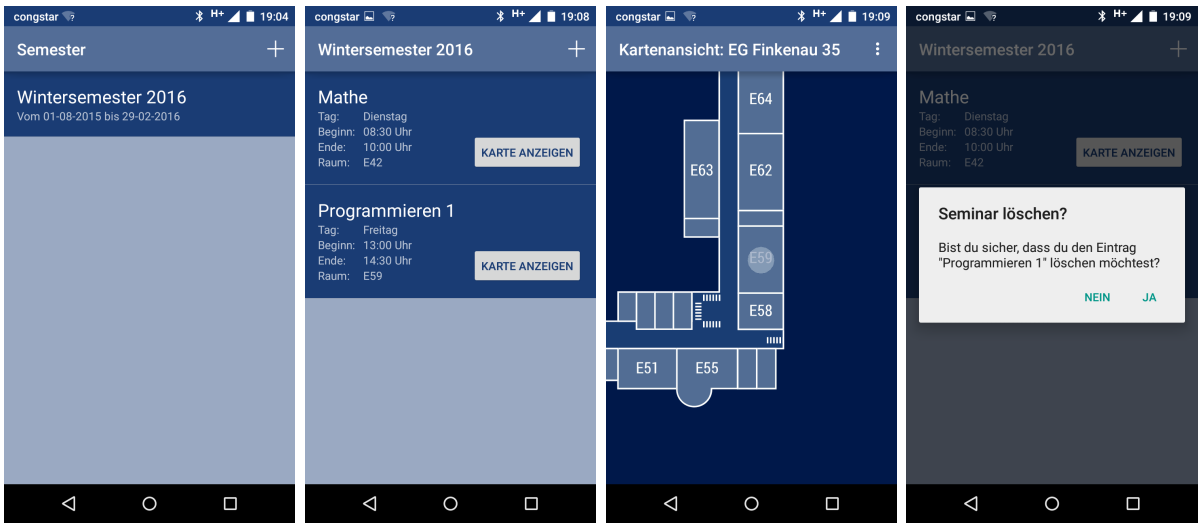

Abbildung 4.7: MS-Seminar-Coach Activities

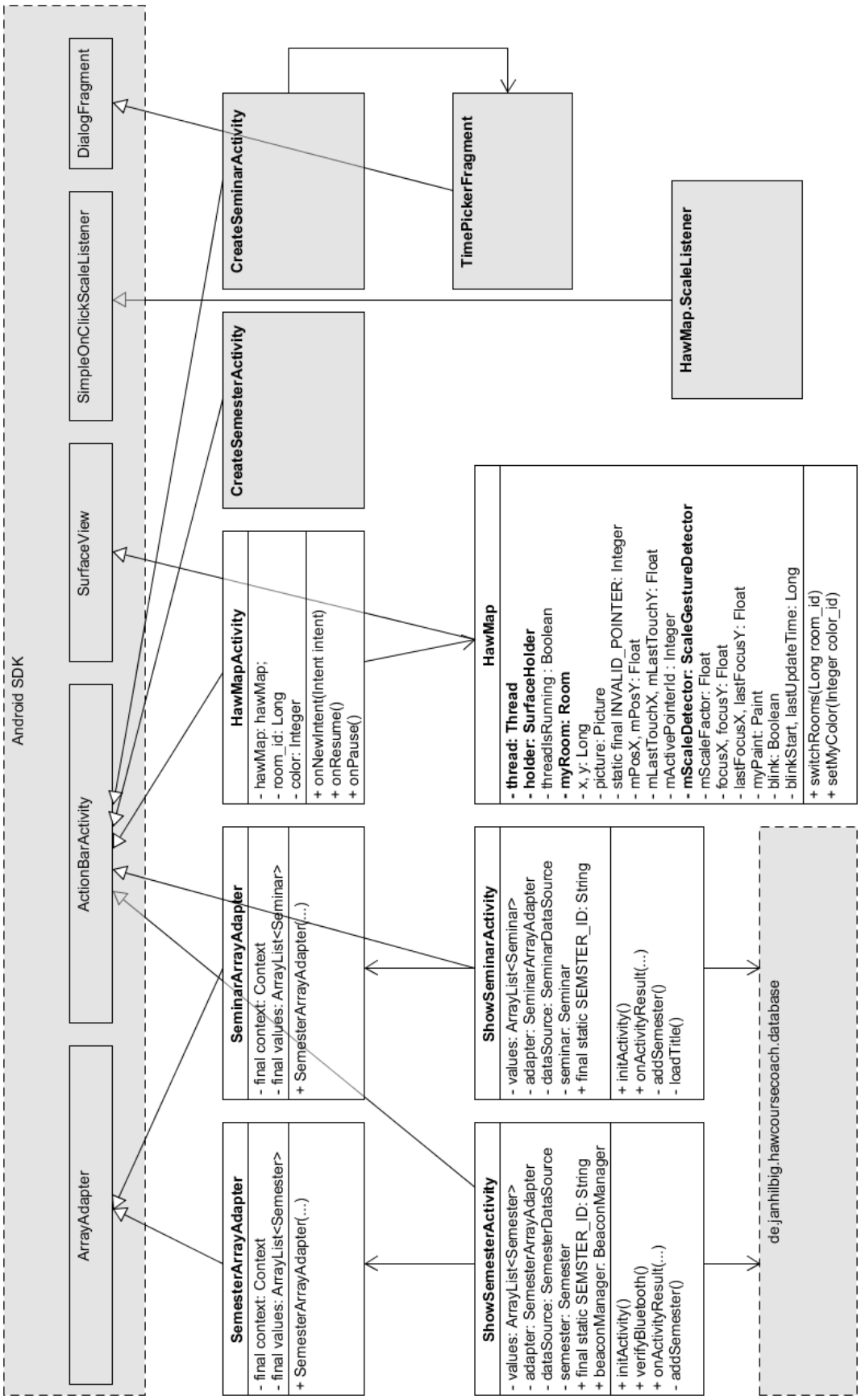

Abbildung 4.8: UML Klassendiagramm 2: Activities

4.3.4 Check-In-System

Das Check-In-System ist der Teil der Anwendung, in dem die iBeacon Funktionen implementiert sind. Bevor der Blick auf die eigentliche Implementierung gerichtet wird, müssen einige Hintergrundinformationen der Android Beacon Library bekannt sein, sowie grundsätzliche Problematiken beim Entwickeln von Software die Proximity-Beacon-Signale nutzt geklärt werden.

Die ursprüngliche Idee war es, das Check-In-System völlig autark vom Rest der Anwendung und, wenn möglich, auch im Hintergrund laufen zu lassen. Das heißt, dass das Check-In-System zu jeder Zeit registriert, ob der Nutzer sich in der Nähe eines Beacons aufhält oder nicht, unabhängig davon ob er die Anwendung aktuell gestartet hat. Zudem soll das System möglichst zeitnah reagieren, also den Check-In-Vorgang unmittelbar nach dem Betreten eines Raumes auslösen.

Proximity Beacons werden bekanntlich über ihre UUID, Major und Minor eindeutig identifiziert. In den gängigen Beacon SDKs werden diese *Identifier* meist mit einem *String* ergänzt und zu einer *Region* zusammengefasst. Definiert man in seiner Software später Beacons auf die reagiert werden soll, erstellt man also einzelne, bzw. eine Liste von *Regions*. Grundsätzlich gibt es bei Proximity Beacons SDKs zwei unterschiedliche Mechanismen, mit denen auf Beacon-Signale zugegriffen werden kann:

- *Monitoring*
- *Ranging*

Monitoring ist das schlichte Erkennen eines Beacons. Beim *Monitoring* werden die *Idenfier* der Beacons ausgelesen und mit den in der Software definierten *Regions* verglichen. Man bekommt über das *Monitoring* keine Information über die Entfernung zu einem Beacon. Der große Vorteil von *Monitoring* ist, dass hier auch energiesparende Background-Scans durchgeführt werden können. Die Anwendung muss also nicht zwangsweise gestartet worden oder aktuell geöffnet sein, um auf den Mechanismus zuzugreifen. Es scheint also nahe zu liegen ein Check-In-System, das als Hintergrund-Dienst stets verfügbar ist, auf der Basis von *Monitoring* zu implementieren.

In der Praxis müsste man die Position und Sendereichweite der Beacons dann so kalibrieren, dass ihr Signal nur innerhalb eines Raumes empfangen werden kann. Bezogen auf die

Testumgebung in der HAW müsste die Signalverteilung der Beacons dann ungefähr wie die roten Kreise in folgender Abbildung aussehen:

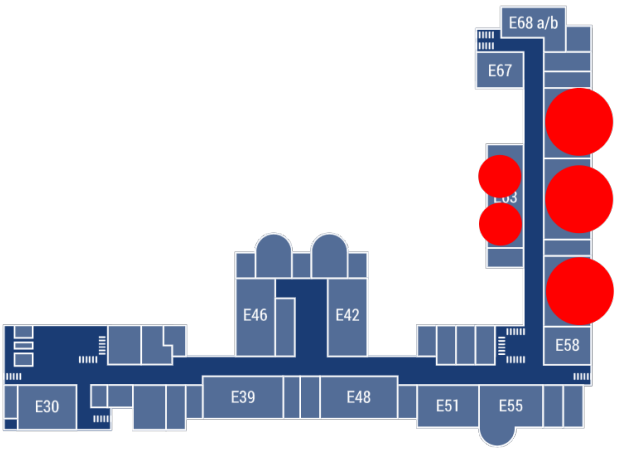

Abbildung 4.9: iBeacon Signalverteilung Szenario 1

Man sieht auf dem Bild deutlich, dass die Sendereichweite der Beacons und ihre Anzahl durch unterschiedliche Raumgrößen und Formen stark variieren müssen, um ein homogenes Feld innerhalb eines Raumes zu erzeugen, dass nicht in den Gängen oder angrenzenden Räumen empfangen werden kann. Die Anzahl der Beacons stellt technisch kein Problem dar, da sich die *Identifier* der Beacons beim Konfigurieren leicht klonen lassen. Ein solches Szenario kann durchaus funktionsfähig sein, wenn man die Sendereichweite der Beacons sehr genau kalibrieren könnte.

In der Praxis besteht jedoch das Problem, das die Sendereichweite der Beacons nur stufenweise kalibriert werden kann, und das zuvor durchgespielte Szenario dadurch unrealistisch wird. Die offensichtlichste Lösung ist die Sendereichweite eines Beacons zu vergrößern, wodurch man an vielen Orten im Gebäude mehrere Signale gleichzeitig empfängt. Das hat zur Folge, dass der Monitoring-Mechanismus nicht ausreichend ist, um ein Check-In-System zu realisieren. Eine realistische Signalverteilung im Gebäude sieht eher aus wie in Abbildung 4.10 dargestellt.

Um ein praxistaugliches Check-In-System zu implementieren, ist es somit notwendig, die Entfernung zum Beacon durch *Ranging* zu bestimmen. Der Ranging-Mechanismus kann nur genutzt werden, wenn die Anwendung geöffnet und im Vordergrund ist, da er im

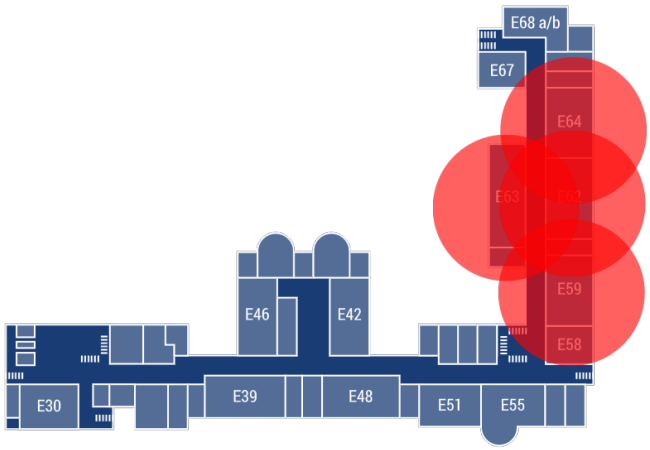

Abbildung 4.10: iBeacon Signalverteilung Szenario 2

Gegensatz zum Monitoring viel Strom verbraucht. Dadurch wird es für den Check-In-Vorgang notwendig, dass der Anwender die Anwendung kurz startet um in einen Raum einzuchecken. Das Check-In-System ist dadurch leider nicht mehr völlig unabhängig. Auch wenn der Nutzer die Anwendung „nur“ kurz öffnen muss.

Bei dem *Ranging* muss man als Entwickler vor allem beachten, das die Entfernung die das SDK zu einem Beacon berechnet in den meisten Fällen sehr ungenau ist. Radius Networks schreibt dazu in der Dokumentation der Android Beacon Library folgendes:

„When the library reports a beacon is 5 meters away, it may actually be anywhere between 2 meters and 10 meters away. At great distances, an estimate of 30 meters might really be anywhere from 20-40 meters. It is also important to understand that different Android device models may overestimate or underestimate distances because of Bluetooth antennas that have higher or lower gain than average devices.“ [4]

Man muss sich bewusst machen, dass man bei Funksignalen keine klare Grenze ziehen kann, bis wohin ein Signal empfangen werden kann. Es sollten bei der Implementierung von daher große Toleranzen einkalkuliert werden. Beim verwendeten Testgerät *Motorola Moto G* hat sich in Tests gezeigt, dass die von der Bibliothek ermittelten Entfernungen weit überschätzt werden. Soll diese eine Aktion bei 1,5 Metern auslösen, muss das Gerät tatsächlich ca. 40 cm bis 60 cm vom iBeacon entfernt sein um die Aktion auszulösen.

Für den Prototypen des MS-Seminar-Coach wurde daher die auslösende Entfernung für den Check-In-Vorgang leicht angehoben. Ausgecheckt wird erst, wenn das Beacon-Signal nicht mehr zu empfangen ist. Mann muss sich also deutlich von dem Proximity Beacon entfernen um den Check-In-Zustand der Anwendung nach erfolgreichem Einchecken zu ändern. Dies soll verhindern, dass die Anwendung in einem Grenzbereich andauernd ihren Zustand ändert. Folgende Abbildung soll dies nochmal verdeutlichen:

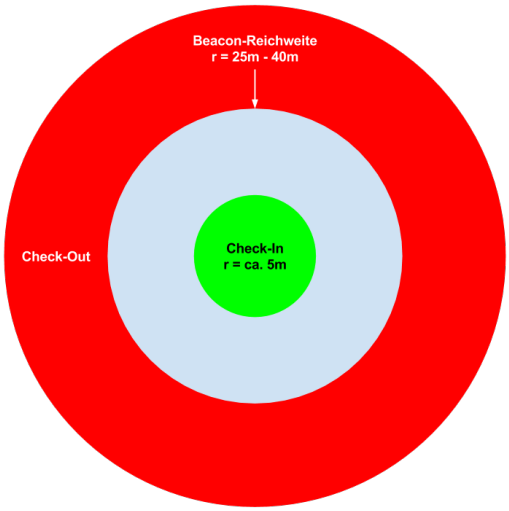

Abbildung 4.11: Check-In-System-Grenzen

Das Check-In-System speichert die Räume, in die eingecheckt wurde, in einer *ArrayListe*. Diese Liste wird von dem Alarm-Manager an den **ReminderService** übergeben, damit dieser überprüfen kann in welchem Raum man sich aufhält. In der Anwendung wird dies in der Application-Klasse **MsSeminarCoach** implementiert:

Check-In:

```
...
beaconManager.setRangeNotifier(new RangeNotifier() {
    @Override
    public void didRangeBeaconsInRegion(Collection<Beacon> beacons, Region
        region) {
        for (Beacon beacon : beacons) {
            if (beacon.getDistance() < 6.5) {
                // Perform distance-specific action
                if (!regionList.contains(region)) {
                    ...
                    checkInList.add(beaconId);
                    updateAlarmIntent(checkInList);
                }
...
```

Check-Out:

```
...
@Override
public void didExitRegion(Region region) {
   Region exitRegion = region;
   if (myRegions().contains(exitRegion)) {
      if (regionList.contains(exitRegion)) {
         ...
         checkInList.remove(beaconId);
         updateAlarmIntent(checkInList);
      }
   }
}
...
```

4.3.5 Reminder Service

Der Reminder-Service ist ein Hintergrunddienst, der regelmäßig überprüft, ob aktuell ein Seminar stattfindet und, falls ja, sich das Gerät auf dem die Anwendung ausgeführt wird, im Seminar-Raum befindet. Es müssen zunächst alle Datenbankeinträge überprüft und mit der aktuellen Zeit verglichen werden. Die Semester- und Seminardaten sind so ausgelegt, das die Überprüfung nach und nach eingegrenzt werden kann. Die Überprüfung findet in folgenden Schritten statt:

1. Systemzeit abfragen
2. Systemzeit mit Anfang und Ende der Semester vergleichen
3. Systemzeit mit den Seminar-Tagen vergleichen
4. Systemzeit mit Start- und End-Zeiten der Seminare vergleichen
5. Räume der Check-In-Liste mit den Seminarräumen vergleichen

Die Überprüfung des ReminderService wird einmal pro Minute durchgeführt. Im Quellcode der Implementierung des Check-In-Systems kann man sehen, dass jedes Mal wenn sich die Check-In-Liste ändert, auch das Alarm-Intent mit der Methode **updateAlarmIntent(ArrayList<String> checkInList)** aktualisiert wird.

```
...
public void updateAlarmIntent(ArrayList<String> checkInList) {
    Intent intent = new Intent(this, ReminderService.class);
    intent.putStringArrayListExtra(CHECK_IN_LIST, checkInList);
    alarmIntent = PendingIntent.getBroadcast(this, 0, intent, PendingIntent.
        FLAG_UPDATE_CURRENT);
}
```

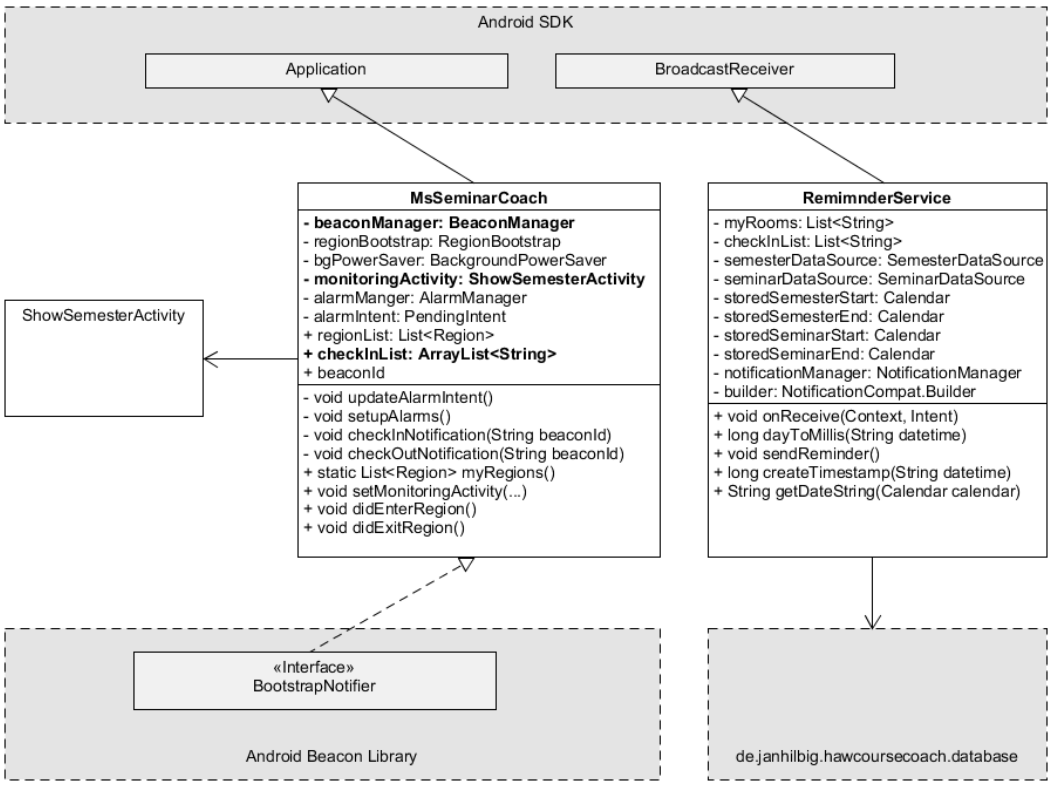

Abbildung 4.12: UML Klassendiagram 3: Application

5 Fazit

Die Umsetzung des Gesamtprojektes, eine iBeacon Anwendung für Android Systeme zu entwickeln, war recht erfolgreich, auch wenn der Prototyp der MS-Seminar-Coach Anwendung noch einige Schwächen hat. Die grundlegende Idee, ein Check-In-System zu entwickeln, das komplett autark als Hintergrunddienst funktioniert, konnte mit der iBeacon Technologie leider nicht vollständig umgesetzt werden. Es konnte aber gezeigt werden, dass es auch möglich ist eine kontextbezogene Anwendung ohne Internetanbindung zu realisieren.

Das Zusammenspiel aller beteiligten Proximity Beacon Komponenten, der Schlüsseltechnologie BLE, die Beacon Hardware und der SDKs wirkt noch relativ unausgereift. Die Technologie hat viele Schwächen, die Android Entwickler in ihren Anwendungen softwareseitig abfangen, bzw. kompensieren müssen. Auch der Funktionsumfang der Beacon SDKs lässt noch einige Wünsche offen. Es wäre z.B. wünschenswert, wenn der Monitoring-Mechanismus von iBeacon zumindest eine grobe Entfernungsschätzung beinhalten würde, damit iBeacon, auch wenn die Anwendung nicht geöffnet ist, sinnvoll genutzt werden kann.

Die Entfernungsmessungen der Proximity Beacons sind sehr ungenau. Um akkurate Entfernungsmessungen von einem iBeacon SDK zu bekommen, müssen die Advertising-Intervalle der Beacons sehr kurz sein. Infolgedessen halten die Batterien der Beacons nicht lange genug, um sie über längere Zeiträume einsetzen zu können, ohne die Batterie zu wechseln. Der von den Herstellern versprochene Vorteil, das BLE Geräte über einen längeren Zeitraum mit einer kleinen Batterie betrieben werden können, scheint daher zu verschwinden.

Die iBeacon Spezifikation selbst weist auch Schwächen auf. Die *Identifier (UUID, Major, Minor)* der Beacons können mit jedem BLE fähigem Smartphone ausgelesen und die Beacons dadurch leicht geklont werden. Die Datenstruktur der iBeacon Spezifikation ist

unflexibel. Natürliche Schwächen, die im Wesen von Funktechnologie stecken, könnten z.B. auch auf Protokollebene kompensiert werden.

Die Proximity Beacon Technologie steht noch ganz am Anfang. Nach Apple hat mittlerweile auch Google eine Spezifikation entwickelt, die „Eddystone“ heißt. Eddystone hat im Vergleich zu iBeacon schon einige Verbesserungen die vielversprechend scheinen. Die Idee, Smartphone Anwendungen auf die physikalische Welt reagieren zu lassen, ist wegweisend. Es ist nur eine Frage der Zeit bis BLE Chipsätze und Proximity Beacon Spezifikationen verbessert werden, um die Effizienz und Genauigkeit der Beacons zu steigern.

Kontextbezogene Anwendungen durch Proximity Beacons werden sicherlich in viele Bereiche unseres alltäglichen Lebens Einzug erhalten. Schon heute arbeiten viele Firmen daran, die Technik für ihre Zwecke einzusetzen. PayPal hat mit PayPal-Beacon z.B. ein Bezahlsystem auf Basis von Bluetooth Low Energy entwickelt. Ob sich in einigen Jahren ein bestimmter Proximity Standard durchgesetzt hat, oder wir mit vielen unterschiedlichen Spezifikationen, je nach Einsatzgebiet, arbeiten werden, bleibt abzuwarten. Sicher ist jedoch, dass Proximity Beacons eine große Innovation sind.

Anhang

Abkürzungsverzeichnis

ADT ..... Android Developer Tools

BLE ..... Bluetooth Low Energy

CI ..... Corporate Identity

CRC ..... Cyclic Redundancy Check

dBm ..... Dezibel Milliwatt

GAP ..... Generic Access Profile

GATT ..... Generic Attribute Profile

GUI ..... Graphical User Interface

MAC ..... Media Access Control

mAh ..... Milliamperestunden

NFC ..... Near Field Communication

PDU ..... Payload Data Unit

POI ..... Point Of Interest

RFID ..... Radio Frequency Identification

RSSI ..... Received Signal Strength Indicator

SDK ..... Software Development Kit

SIG ..... Bluetooth Special Interest Group

SVG ..... Scaleable Vector Graphics

USB ..... Universal Serial Bus

UUID ..... Universally Unique Identifier

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] Gast, Matthew S.: *Building Applications with iBeacon.* O'Reilly Media, 2014. 1

[2] Townsend, Kevin: *Getting Started with Bluetooth Low Energy.* O'Reilly Media, 2014. 6

[3] *The Hitchhikers Guide to iBeacon Hardware.* http://www.aislelabs.com/reports/beacon-guide/, 2014. [letzter Aufruf: 06.07.2015]. 15

[4] Networks, Radius: *Android Beacon Library Documentation.* https://altbeacon.github.io/android-beacon-library/distance-triggering.html, 2014, 2015. [letzter Aufruf: 01.09.2015]. 50

[5] Developer, Android: *Bluetooth Low Energy.* https://developer.android.com/guide/topics/connectivity/bluetooth-le.html, 2014. [letzter Aufruf: 23.06.2015].

[6] Developer, Android: *Graphics architecture.* http://source.android.com/devices/graphics/architecture.html, 2014. [letzter Aufruf: 31.08.2015].

[7] Kechler, Christoph: *UML 2.5, Das umfassende Handbuch.* Rheinwerk Verlag, 2015.

[8] Rumpe, Bernhard: *Modellierung mit UML.* http://mbse.se-rwth.de/book1/index.php?c=overview, 2011. [letzter Aufruf: 27.08.2015].

[9] Schnabel, Patrick: *Bluetooth Low Energy.* http://www.elektronik-kompendium.de/sites/kom/1805171.htm, 2014. [letzter Aufruf: 23.06.2015].

[10] SIG: *Bluetooth Brand, Trademarks and Logos.* `https://www.bluetooth.org/en-us/bluetooth-brand/bluetooth-brand`, 2014. [letzter Aufruf: 23.06.2015].

[11] Toulson, Simon: *Beacon Parameters.* `https://support.kontakt.io/hc/en-gb/articles/201620741-Beacon-Parameters-UUID-Major-and-Minor`, Oktober 2014. [letzter Aufruf: 24.08.2015].

[12] Warski, Alan: *How do iBeacons work?* `http://www.warski.org/blog/2014/01/how-ibeacons-work/`, 2014. [letzter Aufruf: 22.06.2015].