ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Optimierung von Geschäftsprozessen mithilfe mobiler Endgeräte und unter Berücksichtigung der zur Verfügung stehenden Sensoren

Masterarbeit an der Universität Ulm

Vorgelegt von:
Philip Geiger
philip.geiger@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert
Prof. Dr. Peter Dadam

Betreuer:
Marc Schickler
Rüdiger Pryss

2014

„Optimierung von Geschäftsprozessen mithilfe mobiler Endgeräte und unter Berücksichtigung der zur Verfügung stehenden Sensoren“
Fassung vom 12. Oktober 2014

Inhaltsverzeichnis

1 | Einleitung

Moderne mobile Endgeräte wie Smartphones und Tabletcomputer werden immer leistungsstärker. Gleichzeitig wird der Energieverbrauch kontinuierlich gesenkt und damit die Laufzeit verlängert. Das führt dazu, dass die Geräte nicht nur zum täglichen Gebrauch für Telefonate, Instant Messaging oder der Verwendung des Internets herangezogen, sondern nun auch zur Nutzung von aufwendigen mobilen Geschäftsanwendungen verwendet werden können. Die meisten mobilen Endgeräte besitzen neben diesen Eigenschaften eine große Anzahl eingebauter Sensoren, die Benutzern oder Geschäftsanwendungen Aufschluss darüber geben können, in welchem Kontext sie sich aktuell befinden. Der Kontext kann die Position eines Benutzers im Freien oder geschlossenen Räumen, die Blickrichtung, Fahr- oder Laufgeschwindigkeit oder die gesundheitliche Verfassung sein. In einen Kontext fließen aber auch der momentane Fokus eines Benutzers, materielle Objekte oder andere Benutzer aus der direkten Umgebung mit ein [24]. Die Kombination von sowohl leistungsstarken als auch energieeffizienten mobilen Endgeräten und den immer zahlreicher vorkommenden Sensoren lässt sich nutzen, um Geschäftsanwendungen mit Sensorik zu verbinden und damit Geschäftsprozesse mithilfe von *Context Aware Applications* zu optimieren und automatisieren.

Eine Context Aware Application ist eine Anwendung, die sich anhand der aufgenommenen Daten von Sensoren über den aktuellen Kontext des Benutzers oder des mobilen Geräts bewusst ist und durch diesen relevante Informationen an den Benutzer weitergibt oder selbstständig Aktionen durchführt. Dabei ist zu beachten, dass die Relevanz der Informationen von der aktuellen Aufgabe und Intuition des Benutzer abhängt [25].

Der Kontext kann, wie oben beschrieben, beispielsweise die aktuelle Position des Benutzers oder eines mobilen Endgeräts sein. Nach [26] ist die Definition von Kontext jede Information, welche dazu verwendet werden kann, die Situation einer Entität zu charakteri-

sieren. Dabei ist eine Entität eine Person, ein Ort oder ein Objekt, das für die Interaktion zwischen Benutzer und Anwendung, inklusive der Anwendung und dem Benutzer selbst, als relevant angesehen wird. Diese Definition beschreibt den Kontext einer Context Aware Application aus der Sicht eines Entwicklers bzw. der mobilen Anwendung als interne Repräsentation. Andere Definitionen aus anderen Perspektiven werden z.B. in [80] und [27] gegeben.

Sensoren, die zur Kontextbestimmung verwendet werden, lassen sich in *physische* Sensoren, *virtuelle* Sensoren, *logische* Sensoren und das *Benutzerprofil* einteilen. Physische Sensoren sind meist kleine Hardwareprodukte, die physische Daten wie Position, Beschleunigungskräfte, Bewegungsgeschwindigkeit, Licht, Lautstärke, Temperatur und viele andere physische Werte messen können. Virtuelle Sensoren erhalten ihre Kontextinformationen von anderen Anwendungen oder Eigenschaften des Geräts, das die Anwendung ausführt. Dies können unter anderem eingetragene Termine in einem Terminkalender sein, Daten von entfernten Informationsquellen oder dem Benutzerprofil des Geräts. Logische Sensoren sind höher angesiedelte und abstrakte Sensoren, die ihre Kontextinformationen durch Kombination von physischen und virtuellen Sensoren erhalten. Es könnte z.B. die Position, welche durch einen GPS-Sensor erhalten wurde, mit einem virtuellen Kalendereintrag und der aktuellen Uhrzeit verbunden werden, um damit den Benutzer über einen möglichen Termin im weiteren Verlauf des Tages an der aktuellen Position zu erinnern. In jeden Typ der Sensoren und deren bestimmte Kontexte fließen noch die Eigenschaften eines Benutzers mit ein. Beispiele dafür sind präferierte Benutzereinstellungen, Eigenschaften des Benutzers oder die Zugehörigkeit zu einer bestimmten Organisationseinheit.

Neben unterschiedlichen Arten von Sensoren, die für die Bestimmung des Kontexts der Context Aware Application notwendig sind, wird in diesen Anwendungen zwischen *expliziter* und *impliziter* Interaktion unterschieden. Explizite Interaktion ist die wohlbekannte physische Interaktion durch Maus, Tastatur oder Multi-Touch-Gesten mit einem Gerät und der darauf ausgeführten Anwendung. Context Aware Applications können zusätzliche über implizite Interaktionen mit dem Gerät und der Anwendung interagieren. Ein Smartphone kann sich beispielsweise durch den gegebenen Kontext, der aus der aktuellen Position und einem zur aktuellen Uhrzeit stattfindenden Kalendereintrag besteht, durch implizite Interaktion automatisch in einen lautlosen Zustand versetzen. Da implizite Interaktionen meistens nicht vom Anwender wahrgenommen werden, muss er von der Anwendung beispielsweise durch eine Vibration des Geräts über die verborgene Aktion informiert werden.

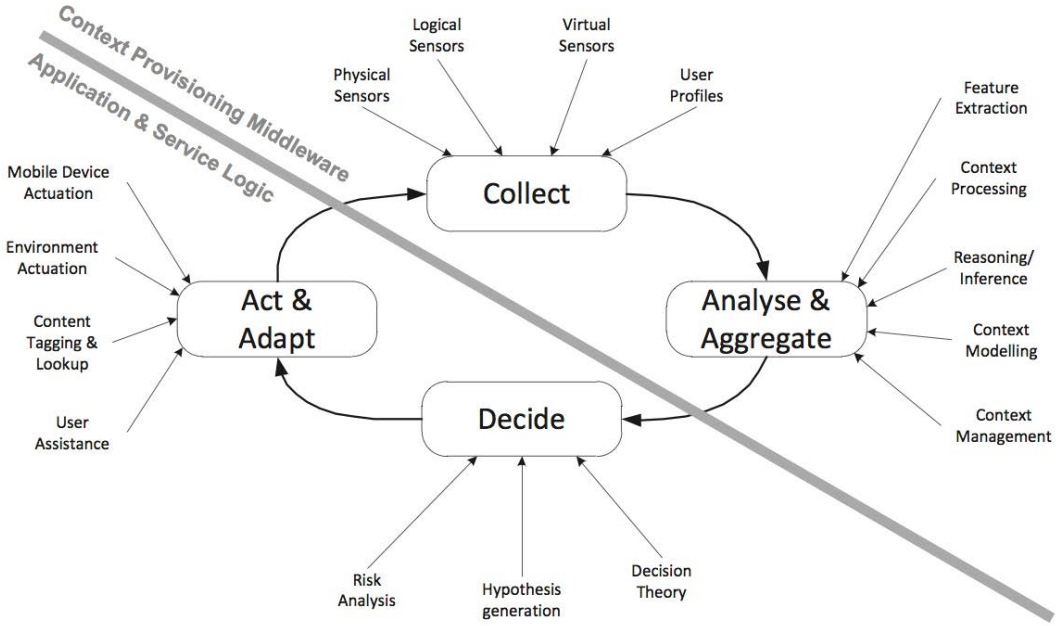

Abbildung 1.1: Systemkreislauf einer Context Aware Application nach [59]

Der komplette Systemkreislauf einer Context Aware Application ist in Abbildung 1.1 abgebildet. In ihr ist zu erkennen, dass unterschiedliche Arten und Typen von Sensoren abgefragt werden. Dadurch erhaltene Daten werden durch eine Vorverarbeitung an die internen Systemeingenschaften angepasst. Aus ihnen werden *Features* extrahiert und durch Kombinationen mehrere Sensoren - der sog. *Sensor Fusion* - der Kontext bestimmt. Durch ihn kann die Anwendung Entscheidungen treffen und im letzten Schritt Aktionen durchführen.

Die Context Aware Application, die im Verlauf der Arbeit entwickelt und vorgestellt wird, erhält ihre Daten zur Kontextbestimmung aus Sensoren zur Positionsbestimmung, dem aktuellen geöffneten Anwendungsfall der Anwendung und der Zugehörigkeit eines Benutzers zu einer bestimmten Organisationseinheit.

1.1 Zielsetzung

Ziel der Arbeit ist es, einen weitreichenden Überblick über einige auf mobilen Endgeräten verfügbare physische Sensoren zu geben und ihre Verwendung zur Bestimmung des Kontexts eines Benutzers aufzuzeigen. Der Schwerpunkt vorgestellter physischer Sensoren

wird auf denen liegen, die zur Positionsbestimmung des Benutzers herangezogen werden können. Darauf aufbauend werden Szenarien und Geschäftsprozessen erläutert, die durch den Einsatz mobiler Sensorik optimiert und automatisiert werden. Das Ziel, einen tiefen Einblick in die Kombination von Geschäftsprozessen mit Geschäftsanwendungen und mobiler Sensorik zu geben, wird durch die Implementierung eines Krankenhaus-Prozesses erreicht. Anhand des Prozesses und seiner Implementierung soll dargestellt werden, wie unter Verwendung von Bluetooth LE, Apples iOS, einer Serverarchitektur und zusätzlichen Komponenten eine Context Aware Application Geschäftsprozesse optimieren und automatisieren kann. Das resultierende Softwaresystem soll beispielsweise das Auffinden und Einsehen von Patientenakten und Aufgabenübersichten anhand von Kontexten optimieren, die durch die momentane Position eines Benutzers und seiner Zugehörigkeit zu einer Organisationseinheit bestimmt werden. Auch fehlerhafte Patientenverlegungen sollen durch die Anwendung verhindert und Medikamentenbestände einzelner Medikamente automatisiert dokumentiert und aktualisiert werden. Für die Umsetzung einer solchen Geschäftsanwendung wird das Ziel gesetzt, eine geeignete Systemarchitektur zu erarbeiten, Sensoren und ihre Daten zu untersuchen, Schnittstellen bereitzuhalten und einen geeigneten Einsatz mobiler Sensorik zu finden.

1.2 Aufbau der Arbeit

In Kapitel 2 werden Sensoren vorgestellt, die zur Positionsbestimmung eines Benutzers oder mobilen Endgeräts verwendet werden können. Dabei werden ebenfalls einige Methoden und Algorithmen erläutert, die auf verschiedene Arten die Position berechnen und unterschiedlich genaue Ergebnisse liefern. Da ein Kontext auch über andere Methoden als über die Positionsbestimmung ermittelt werden kann, werden im gleichen Kapitel weitere Sensoren kurz skizziert. Die in der Arbeit entwickelte Context Aware Application ermittelt ihren Kontext mittels Positionsbestimmung. Die Positionsbestimmung basiert wiederum auf der Funktechnologie Bluetooth LE und dem Konzept der von Apple vorgestellten iBeacons. Aus diesem Grund werden in Kapitel 3 iBeacons vorgestellt und sowohl das zugrundeliegende Protokoll als auch die Funktechnologie Bluetooth LE erläutert. Während der Entwicklung und Implementierung der Context Aware Application spielen Raspberry Pis eine besondere Rolle. Die kleinen mobilen Platinen werden als iBeacons eingesetzt. Deshalb werden sie in Kapitel 4 skizziert und das Vorgehen beschrieben, wie sie als iBeacons eingesetzt werden können und was dafür nötig ist. In Kapitel 5 werden einige

Geschäftsprozesse und Szenarien vorgestellt, die mithilfe von iBeacons und Kontextbestimmung optimiert und automatisiert werden können. Dabei wird zu Beginn der bereits erwähnte Krankenhaus-Prozess skizziert und ausführlich auf Verbesserungen untersucht. Der Krankenhaus-Prozess und sein Verbesserungspotential wird die Grundlage für die in Kapitel 6 vorgenommene Anforderungsanalyse sein. Darin werden Anhand der Problematiken des Krankenhaus-Prozesses Anforderungen definiert und gesammelt, die durch das in der Arbeit entwickelte und implementierte Softwaresystem umgesetzt werden sollen. Basierend auf den Anforderungen wird in Kapitel 7 der Architekturentwurf des Softwaresystems und das Design involvierter Komponenten vorgestellt. In Kapitel 8 werden die Implementierungen wichtiger Konzepte einzelner Komponenten aufgezeigt und die verwendeten Technologien und Hilfsmittel kurz erwähnt. Kapitel 9 präsentiert die Idee, ein Workflowmanagementsystem an das entwickelte Softwaresystem anzukoppeln und den technischen Entwurf dieser Kopplung. Die mobile Context Aware Application des Softwaresystems wird in Kapitel 10 durch Screenshots veranschaulicht und die Bestimmung des Kontextes in verschiedenen Anwendungsfällen beschrieben. In Kapitel 11 werden einige Erkenntnisse dargelegt, die während dem Testen der Anwendung erlangt wurden. Ferner wird die Anwendung hinsichtlich spezieller Anforderungen der Anforderungsanalyse bewertet. In Kapitel 12 wird die Arbeit und ihr Ergebnis nochmals zusammengefasst und ein Ausblick gegeben.

1.3 Verwandte Arbeiten

Sowohl mobile Geschäftsanwendungen als auch die Kopplung dieser mit mobilen Sensoren zur Positionsbestimmung wurden in [42–44] bearbeitet. In diesen Arbeiten lag der Fokus auf *Mobile Augmented Reality* und der Positionsbestimmung über GPS. Daraus entstand ein Framework namens *AREA* (Augmented Reality Engine Application) das für die mobilen Betriebssysteme Android und iOS konzipiert und implementiert wurde. Auf der Basis dieser Arbeiten wurde in [78] ein Workflowmanagementsystem mit dem entwickelten Framework verbunden, um ortsbezogene Geschäftsanwendungen mittels Positionsbestimmung außerhalb von Gebäuden über GPS zu entwickeln. Ein Framwork zur Positionsbestimmung innerhalb von Gebäuden mithilfe von WLAN-Netzwerken und sogenannten Heatmaps wurde in [5] entwickelt. Es wurde in [81] verwendet, um in Verbindung mit AREA ein Navigationssystem für die Navigation innerhalb von Gebäuden herzustellen. Eingenschaften von Bluetooth 4.0 LE, was in dieser Arbeit ebenfalls unter Einbezug von iBeacons ver-

wendet werden wird, wurden in [48] untersucht. In [67, 68] wurde mit dem Konzept *MEDo* untersucht, wie Geschäftsprozesse in Krankenhäusern durch die Integration von Workflowmanagementsystemen mit mobilen Endgeräten optimiert und vereinfacht werden können. Die prozessgesteuerte Sammlung von Daten mit mobilen Endgeräten wurde in [79] behandelt. Das Konzept mobile Endgeräte als ausführende Clients in eine Prozessumgebung einzubetten und zu modellieren wurde in [65, 66, 69] basierend auf *MARPLE* [77] behandelt und untersucht. Generelle Eigenschaften, Problematiken, Vorteile und das Vorgehen bei der Entwicklung von mobilen Anwendungen für mobile Endgeräte wurde anhand der Anwendung *DBIScholar* in [77] betrachtet. Die Entwicklung mobiler Patientenakten ohne Prozesssteuerung und ohne Sensorsensibilität wurde in [30, 56] entwickelt. Aus der Wirtschaft lässt sich das Unternehmen *Ekahau* [29] nennen. Es bietet Systeme an, die auf WLAN-Signalstärken und RFID basieren, um Geschäftsprozesse im Gesundheitswesen mittels Sensorik und Nachverfolgung von Personal und Gegenständen zu unterstützen.

Aufbauend auf den Ideen der vorgestellten verwandten Arbeiten werden in dieser Arbeit mehrere dieser Konzepte miteinander verbunden. So werden unterschiedliche Sensoren zur Positionsbestimmung genauer untersucht und eine geeignete Sensortechnologie zur Positionsbestimmung innerhalb von Gebäuden ausgewählt. Sie wird auf Bluetooth LE basieren und sowohl zur Positionsbestimmung als auch für NFC-ähnliche Interaktionen herangezogen. Die Sensortechnologie wird dann in einer Context Aware Application verwendet, um unter anderem eine Patientenakte automatisiert anzuzeigen. Ferner wird die Kombination aus Sensortechnologie und Context Aware Application mit einem Workflowmanagementsystem verbunden, um eine automatisierte prozessorientierte mobile Anwendung zu entwickeln, die Daten prozessorientiert erfassen kann, die aktuelle Position des mobilen Geräts in den Prozess einfließen lässt und einen konkreten Krankenhaus-Prozess implementiert.

2 | Mobile Sensorik

In diesem Kapitel werden Sensoren und Systeme vorgestellt, die mittlerweile in vielen mobilen Geräten vorhanden sind, durch deren Einsatz ein Kontext eines Benutzers bzw. eines mobilen Geräts bestimmt werden kann. Dabei liegt der Schwerpunkt auf Sensoren und Systemen, die für die Bestimmung der Position herangezogen werden können.

Begonnen wird mit GPS-Sensoren, die sich vor allem zur Positionsbestimmung außerhalb von Gebäuden eignen, aber auch zur Bestimmung der horizontalen Bewegungsgeschwindigkeit verwenden lassen. Danach wird auf WLAN eingegangen, wobei einige Methoden und Herangehensweisen vorgestellt werden, die die Bestimmung der Position ermöglichen. Mobilfunknetze und Bluetooth sind ebenfalls Systeme, die auf vielen mobilen Endgeräten vorhanden sind. Aus diesem Grund werden beide Systeme kurz skizziert, wobei beim Mobilfunk eine spezifische Methode zur Positionsbestimmung vorgestellt wird. Wie weiter unter dargestellt werden wird, unterscheidet sich die Methodik zur Positionsbestimmung bei Mobilfunknetzen und Bluetooth jedoch nur in kleinen Teilen von der, die bei WLAN verwendet wird. Da die Implementierung des Krankenhaus-Prozesses auf Bluetooth LE und den darauf basierenden iBeacons basiert, wird in diesem Übersichtskapitel nicht näher auf Bluetooth eingegangen. Eine detaillierte Beschreibung findet sich im Kapitel 3.1 über die iBeacon-Technologie.

Nachdem die primären Sensoren und Systeme zur Positionsbestimmung eines mobilen Endgeräts vorgestellt wurden, werden weitere Sensoren, Systeme und Technologien aufgelistet und kurz diskutiert, die Aufschluss über den Kontext geben können.

2.1 GPS

GPS (*Global Positioning System* [63]) ist das wohl bekannteste System zur Bestimmung der Position. GPS ist, anders als die folgenden Sensoren und Systeme, ausschließlich zur Bestimmung der Position entwickelt worden. Um die Position zu bestimmen, wird ein *GPS-Sensor* benötigt. Ein solcher ist zur heutigen Zeit in so gut wie jedem mobilen Endgerät vorhanden. Dieser Sensor stellt eine Verbindung zu mindestens drei bzw. vier der sich in der Erdumlaufbahn befindlichen Satelliten her, mithilfe derer sich die Position auf einige Meter genau berechnen lässt. Eine genauere Position kann über *DGPS* erreicht werden. Herkömmliche mobile Geräte verwenden jedoch GPS-Sensoren, die nur die oben genannte Genauigkeit erreichen. Die Positionsbestimmung über GPS funktioniert im Grunde so, wie die Trilateration bei WLAN, die in Kapitel 2.2.2 genauer vorgestellt und in Abbildung 2.3 veranschaulicht wird. Da die einzelnen Satelliten auf festen Umlaufbahnen die Erdkugel umkreisen, kann jederzeit die exakte Position eines Satelliten bestimmt werden. Die Entfernung zwischen Satellit und GPS-Empfänger wird wie in Kapitel 2.2.2 über Signallaufzeiten zwischen Satellit und Empfänger bestimmt. Sobald eine Verbindung zu drei Satelliten hergestellt wurde, die Positionen dieser Satelliten bekannt sind und die Entfernungen über Signallaufzeiten berechnet wurden, kann über eine Trilateration die Position des Empfängers bestimmt werden. Wird zusätzlich ein Verbindung zu einem vierten Satelliten hergestellt, so kann außerdem die Höhe über Normalnull des Empfängers berechnet werden.

GPS kann darüber hinaus verwendet werden, um die horizontale Bewegungsgeschwindigkeit und Kompassdaten zu bestimmen. Für letzteres werden dazu aufeinanderfolgende Positionen mit der Differenz der Zeitstempel verrechnet.

Eine Position, die durch GPS berechnet wurde, wird in einem der folgenden Formate mit Breiten- und Längengraden angegeben[1]:

- Grad mit Dezimalstellen:
  $48.40108°N$, $9.98761°E$
- Grad mit Bogenminuten und Dezimalstellen:
  $48°24.065'N$, $9°59.257'E$
- Grad mit Bogenminuten, Bogensekunden und Dezimalstellen:
  $48°24'3.9''N$, $9°59'15.4''E$

[1] Beispielhafte Koordinaten für die Magirus-Deutz-Straße in Ulm.

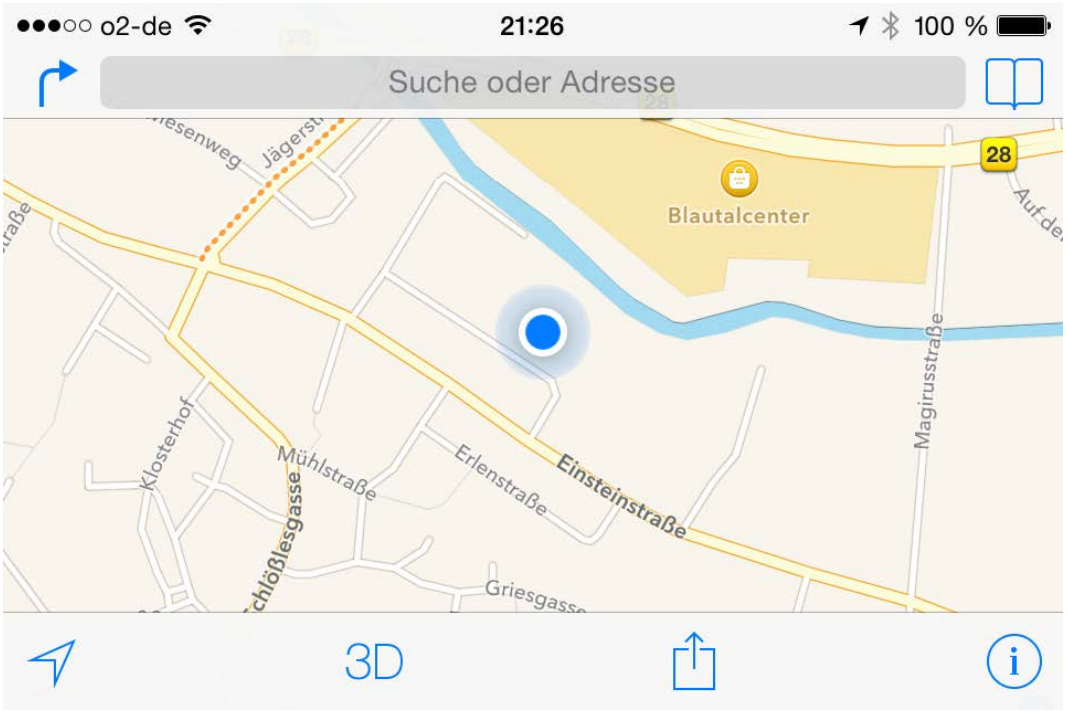

Abbildung 2.1: Ein Beispiel für die GPS-Positionsbestimmung auf der Erdoberfläche

GPS findet vor allem in Anwendungen und Geschäftsprozessen außerhalb von Gebäuden und unter freiem Himmel Gebrauch, da für eine exakte oder möglichst genaue Positionsbestimmung Sichtkontakt mit Satelliten hergestellt werden muss. Deshalb kommt es auch zu Verbindungsabbrüchen und Ungenauigkeiten der berechneten Koordinaten, wenn GPS zur Bestimmung der Position innerhalb von Gebäuden oder in einer Umgebung mit vielen abschirmenden Objekten verwendet wird.

Der Vorteil gegenüber den folgenden Systemen und Sensoren zur Positionsbestimmung ist der, dass keinerlei Wissen über die Umgebung des Benutzers oder den Anwendungskontext vorhanden sein muss, da GPS-Koordinaten direkt mit Positionen und Koordinaten auf der Erdkugel in Verbindung gesetzt werden. Deswegen wird GPS vor allem in Anwendungen zur Navigation und Positionsbestimmung auf der direkten Erdoberfläche verwendet. Eine solche Anwendung mit Karte und Position des mobilen Geräts ist in Abbildung 2.1 abgebildet.

2.2 WLAN

WLAN steht für *Wireless Local Area Network* und ist, anders als GPS, nicht für die Bestimmung der Position entwickelt worden, sondern für das Einrichten eines Computernetzwerkes ohne Kabelverbindung, wie es z.B. bei Ethernet-Verbindungen notwendig wäre. Im Folgenden wird die genaue Funktionsweise von WLAN nicht vorausgesetzt und nicht weiter diskutiert. An dieser Stelle werden nur Methoden und Herangehensweisen aufgezeigt, welche eine Positionsbestimmung durch WLAN ermöglichen.

Ein WLAN besteht im wesentlichen aus einem oder mehreren *Routern*, *Access Points* und *Clients*. Für die Positionsbestimmung sind in erster Linie die Koordinatoren des Netzwerks, also Router und Access Points notwendig. Da die Koordinatoren zur Aufrechterhaltung des Netzwerkes in bestimmten Zeitabständen Datenpakete versenden, die sogenannten *Beacons*, können auf die Daten innerhalb dieser Pakete Algorithmen angewendet werden, um die Position des Benutzer zu bestimmen.

Anders als bei GPS, werden die mit folgenden Methoden und Herangehensweisen erhaltenen Koordinaten und Positionen nicht direkt auf Positionen in der realen Welt, also der Erdoberfläche, angewendet. Es müssen immer eigene Karten erstellt werden oder die Koordinaten und Positionen auf fiktive Orte innerhalb der Anwendung übersetzt werden.

2.2.1 WLAN Signal Strength Map

Eine einfache aber ungenaue Herangehensweise, um mit einem WLAN-Signal die Position des mobilen Endgeräts zu bestimmen, ist die sogenannte WLAN *Signal Strength Map* oder *Heatmap*. Hierbei wird die Signalstärke von einem Koordinator, also einem Router oder Access Point, aus gemessen und verfolgt. Ein Testgerät wird mit einem Koordinator gekoppelt. Ausgehend von diesem Koordinator wird die Umgebung abgelaufen und die Signalstärke an beliebigen Punkten gemessen und vermerkt. Die gemessenen Signalstärken müssen und dürfen nur für diesen Koordinator verwendet werden. Befinden sich in einem Raum oder Gebäude mehrere Koordinatoren, so kann das Vorgehen für jeden dieser Koordinatoren wiederholt werden. Wichtig ist aber, dass für jeden Koordinator die eigenen zugehörigen Signalstärken vermerkt werden. Anhand der Signalstärken der einzelnen Koordinatoren an spezifischen Positionen kann nun eine Signal Strength Map erstellt werden und fiktiven Positionen zugeordnet werden.

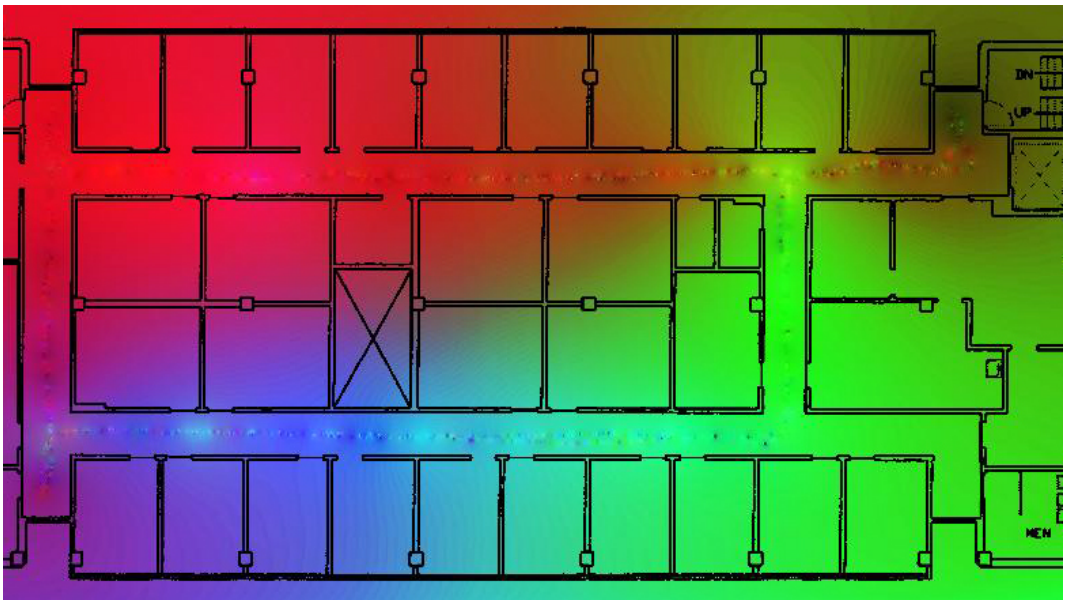

Abbildung 2.2: Beispiel für eine WLAN Signal Strengt Map mit eingefärbten Signalstärken [50]

In Abbildung 2.2 ist eine beispielhafte Signal Strength Map dargestellt. Zu dessen Berechnung wurde ein einzelner Koordinator verwendet. Man kann nun erkennen, dass die Signalstärke nah am Router (grün) stark ist, wohingegen sie mit zunehmender Entfernung immer geringer wird. Zur Bestimmung der Position des mobilen Endgeräts kann nun die Signalstärke an einer beliebigen Stelle in Reichweite des Netzwerks gemessen und mit den zuvor bestimmen Signalstärken verglichen werden.

Wie man leicht erkennen kann, eignet sich diese Herangehensweise bei der Verwendung eines einzelnen Koordinators nur zur ungenauen Positionsbestimmung, da viele Positionen bedingt durch Wände und eventuelle Störungen des Signals die gleichen Signalstärken ausweisen und damit eine exakte Positionsbestimmung unmöglich ist. Um die Genauigkeit erheblich zu verbessern, müssen, wie oben beschrieben, mehrere Koordinatoren eingesetzt werden. Die Kalibrierung und Messung der Signalstärken muss über mehrere Tage und Wochen durchgeführt werden, da die Signalstärken über diese Zeit gemittelt werden müssen. Weiter sollte darauf geachtet werden, dass während der Kalibrierung wenig Kommunikation im Netzwerk herrscht, da ansonsten Interferenzen durch andere Anwender im Netzwerk auftreten.

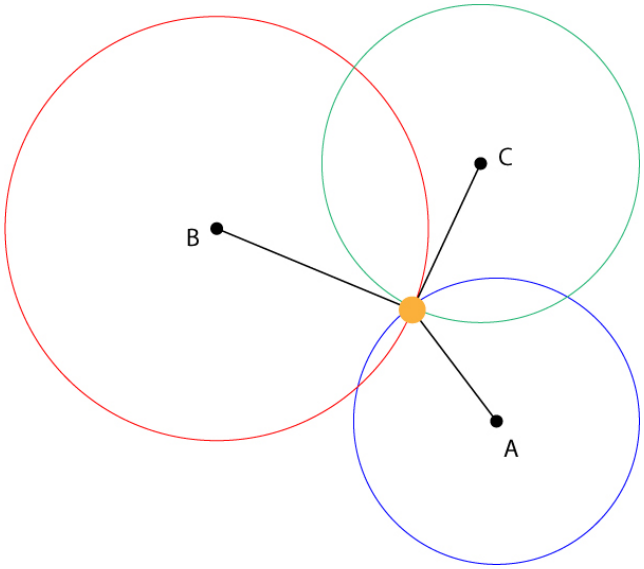

Abbildung 2.3: Tilateration zur Bestimmung der Position

An dieser Stelle kann das Unternehmen *Ekahau* [29] erwähnt werden. Dieses Unternehmen bietet eine Software an, mit der Signal Strength Maps in relativ kurzer Zeit und ohne großen Aufwand erstellt und kartographiert werden können.

2.2.2 WLAN Trilateration

Eine andere Herangehensweise, mit der sich genauere Positionen bestimmen lassen, ist die WLAN *Trilateration*. Hierbei müssen mindestens drei Koordinatoren in der Nähe des Benutzer liegen, welche an fixen und zuvor bekannten Position stationiert sind. Um die Position des mobilen Endgeräts zu bestimmen, werden die fixen Positionen der drei Koordinatoren und die Entfernung des mobilen Endgeräts zu jeweils einem der Koordinatoren [85] benötigt. Sobald die Entfernungen berechnet wurden und bekannt sind, werden sie mit den fixen Positionen der Koordinatoren verrechnet, wodurch die Position des mobilen Endgeräts bestimmt werden kann. Dargestellt wird diese Herangehensweise in Abbildung 2.3, worin zu erkennen ist, dass die Position des mobilen Endgeräts der Schnittpunkt der Kreise der drei Koordinatoren ist. Die Kreise besitzen als Radius die Entfernung zwischen mobilem Endgerät und dem jeweiligem Router.

Die Schwierigkeit dieser Herangehensweise liegt beim Finden der Entfernung zwischen dem mobilen Endgerät und einem Koordinator. Eine Möglichkeit sie zu berechnen, gelingt

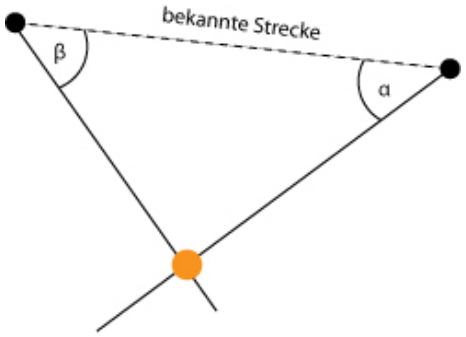

Abbildung 2.4: Triangulation zur Bestimmung der Position

über signaltheoretische Funktionen oder die kalibrierte Signalstärke eines Koordinators. Auf letzteres wird in Kapitel 3 über die iBeacon-Technologie nochmals genauer eingegangen. Eine andere Möglichkeit die Entfernung zu bestimmen, basiert auf hoch synchronisierten feinauflösenden Uhren zwischen dem mobilen Endgerät und den Koordinatoren. Hierbei senden die Koordinatoren ihre Pakete mit Zeitstempeln an das mobile Endgerät, das aus mehreren aufeinanderfolgenden Paketen mit geeignet Algorithmen die Entfernung berechnet.

2.2.3 WLAN Triangulation

Die *Triangulation* funktioniert ähnlich wie die Trilateration, basiert jedoch auf der sogenannten Kreuzpeilung. Die Entfernungen zwischen den einzelnen Koordinatoren und deren fixen Positionen sind als bekannt vorausgesetzt. Um die Position des mobilen Geräts zu bestimmen, werden die Winkel der ankommenden Signale gemessen. Die Position des mobilen Geräts ist somit der Schnittpunkt der beiden Geraden, welche den zuvor berechneten Winkel zu den Koordinatoren besitzen. Abbildung 2.4 veranschaulicht dieses Vorgehen nochmals.

Der Vorteil gegenüber der Trilateration ist der, dass zur Berechnung nur zwei Koordinatoren benötigt werden [85]. Die Schwierigkeit bzw. ein wichtiger Nachteil der, dass eine Möglichkeit gefunden werden muss, den Winkel des eingehenden Signals zu berechnen.

2.2.4 WLAN Cell of Origin

Eine weitere sehr einfache, aber auch ungenaue Herangehensweise, ist die Bestimmung der Position über die Methode *Cell of Origin*. Dabei wird jedem Koordinator eine Zone zugewiesen und vermerkt. Das mobile Endgerät verbindet sich nun immer mit dem Koordinator, der die beste Signalqualität und Signalstärke aufweist. Anhand einer Identifikation des Koordinators kann dann herausgefunden werden, in welcher Zone sich das mobile Endgerät befindet. Nämlich irgendwo in der Empfangszone des verbunden Koordinators. Die Genauigkeit der Positionsbestimmung hängt dabei maßgeblich von der Größe der Zonen ab und dies wiederum von der Anzahl der Koordinatoren und deren Abständen zueinander.

2.3 Mobilfunk

An dieser Stelle will erwähnt werden, dass jedes Smartphone, auf welchem Geschäftsanwendungen ausgeführt werden könnten, die Möglichkeit besitzt, sich mit einem Mobilfunknetz zu verbinden. Auch mit einem Mobilfunknetz lässt sich die Position mit den gleichen Methoden und Herangehensweisen wie im WLAN bestimmen. Die Methoden unterscheiden sich lediglich in den verwendeten Protokollen der verschiedenen Technologien und in den jeweiligen signaltheoretischen Eigenschaften der Mobilfunknetze (z.B. GSM, GPRS, UMTS, LTE).

Die Triangulation, die in Kapitel 2.2.3 vorgestellt wurde, besitzt die Schwierigkeit, den Winkel zwischen einem mobilen Endgerät und den Koordinatoren zu berechnen. Im Mobilfunk besitzen die Koordinaten (die Mobilfunkmasten) drei gerichtete Antennen, die jeweils einen Winkel von 120° abdecken (s. Abbildung 2.5). Die Triangulation und Positionsbestimmung im Mobilfunk gelingt nun wie folgt [19].

Zuerst muss herausgefunden werden, mit welcher der drei gerichteten Antennen ein mobiles Endgerät verbunden ist. Damit kann bereits jetzt angegeben werden, in welchem der drei Sektoren eines Koordinators sich das mobile Gerät befindet. Danach kann, um eine genauere Position des mobilen Geräts zu erhalten, die Entfernung zu diesem Koordinator berechnet werden und erhält einen Bogen von 120° mit der berechneten Entfernung um den Koordinator (s. Abbildung 2.6).

Werden nun noch zwei weitere Koordinatoren hinzugefügt und in allen drei Koordinatoren der gleiche Ablauf durchgeführt, so kann die Position eines mobilen Endgerät relativ ge-

Abbildung 2.5: Ein Mobilfunkmast mit seinen in drei Richtungen gerichteten Antennen [19]

nau bestimmt werden. Wird die Entfernung zu den Koordinatoren weggelassen, kann eine Position innerhalb eines kleinen Sektors berechnet werden. Wird zusätzlich die Entfernung hinzugenommen, so kann die Position durch Mobilfunk innerhalb von Städten, in denen eine hohe Dichte an Mobilfunkmasten vorhanden ist, auf etwa 5 bis 10 Meter genau bestimmt werden. Abbildung 2.7 zeigt das Resultat nach der Bestimmung der Position anhand dreier Koordinatoren und der zusätzlich Berechnung und Kenntnisnahme der Entfernung zwischen Koordinator und mobilem Endgerät.

2.4 Bluetooth

Bluetooth ist ein Netzwerk, ähnlich wie WLAN, das Primär zur Übertragung von kleineren Daten verwendet wird. Ein weiteres Ziel von Bluetooth ist das Steuern von entfernten Geräten oder die Übertragung von Sprache und Ton (z.B. über kabellose Headsets). Bluetooth fällt in die Kategorie der *WPAN*s (*Wireless Personal Area Network*) und hat seine Vorzüge vor allem bei der Erstellung von *Ad-hoc*-Netzwerken. Die Positionsbestimmung über Bluetooth kann im Grunde über die gleichen Methoden und Herangehensweisen bewerkstelligt werden. Bei Bluetooth ist jedoch kein Router oder Mobilfunkmast als Koordinator notwendig, sondern nur einzelne Bluetooth-Geräte.

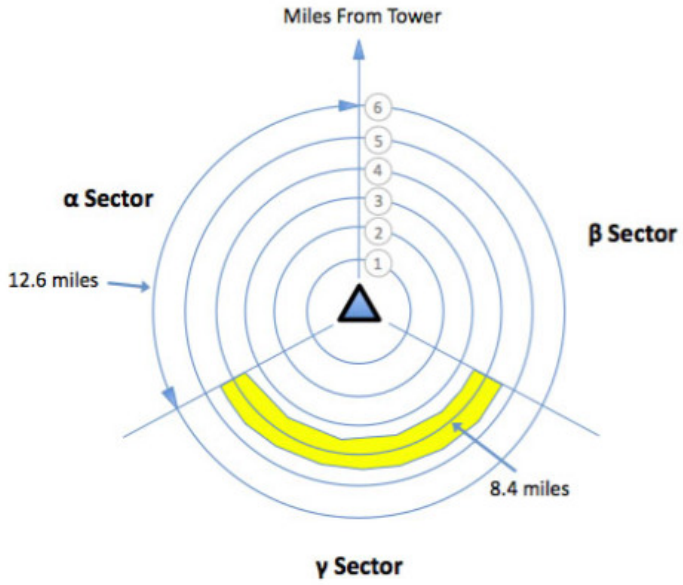

Abbildung 2.6: Erster Schritt der Triangulation im Mobilfunknetz [19]

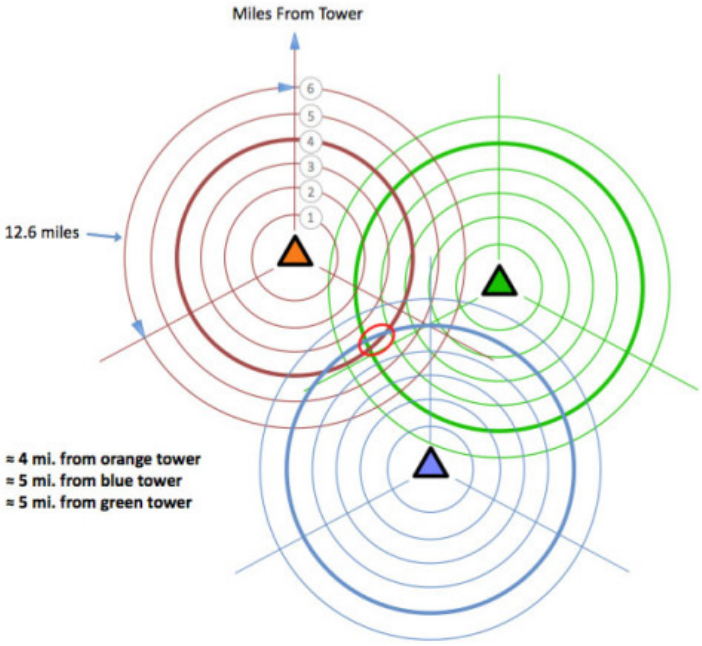

Abbildung 2.7: Resultat der Positionsbestimmung im Mobilfunknetz [19]

Die Implementierung des Krankenhaus-Prozesses, der später im Detail vorgestellt wird, basiert auf *Bluetooth LE*. Bluetooth LE besitzt Fähigkeiten, mit deren Hilfe es möglich ist, auf eine andere Art und Weise die Position von mobilen Geräten stromsparend und vor allem auch besonders gut innerhalb von Gebäuden zu bestimmen. Wie diese Methodik funktioniert, wird im Zusammenhang mit iBeacons im Kapitel über die iBeacon-Technologie genauer diskutiert.

Neben der Positionsbestimmung lässt sich Bluetooth als energiesparende Trägertechnologie zur Übertragung von angeschlossenen und verbundenen Sensoren verwenden. Beispiele für Sensoren, deren Daten über Bluetooth übertragen werden können, sind z.B. Gesundheitsdaten von Puls- und Blutdrucksensoren oder Schrittzählern. Im Bluetooth-Standard sind für die Übertragung von Sensordaten schon einige Profile implementiert, die sich von den Sensoren als einfache Schnittstelle verwenden lassen. Ein Beispiel für ein solches Profil befindet sich innerhalb des *GATT*-Profils [40, 41], das *HRP* (Heart Rate Profile) [53].

2.5 Weitere Sensoren

Bis zu diesem Punkt wurden einige der wichtigsten Sensoren vorgestellt, die in den meisten mobilen Geräten vorhanden sind. So können durch GPS, WLAN, Mobilfunk und Bluetooth viele Anwendungen, die Geschäftsprozesse implementieren, bereits weitestgehend optimiert werden. Bevor im nächsten Kapitel auf die iBeacon-Technologie eingegangen wird, welche die Grundlage für die Implementierung des Krankenhaus-Prozesses liefert, sollen an dieser Stelle weitere Sensoren mobiler Geräte vorgestellt werden, die sich für Optimierung, Automatisierung oder Positionsbestimmung eignen.

2.5.1 RFID

RFID steht für *Radio-Frequency Identification* und ist eine Technologie für berührungslose Kommunikation zwischen Sender und Empfänger. Sie dient der Identifikation und Positionsbestimmung von Objekten [82, 83]. Zur Kommunikation werden Radiowellen verwendet. Ein RFID-Tag ist ein sogenannter *Transponder* von sehr geringer Größe (s. Abbildung 2.8), mit dem ein Objekt eindeutig identifiziert werden kann. Transponder kommen in unterschiedlichen Arten vor und lassen sich in *passive* und *aktive* Transponder einteilen. Passive

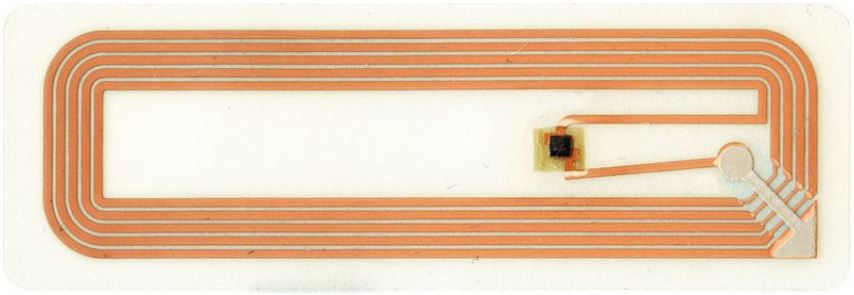

Abbildung 2.8: Ein Transponder, der bei RFID zum Einsatz kommt [76]

Transponder besitzen keinerlei Energiequelle und werden lediglich durch die Funksignale des Abfragegeräts durch Induktion mit Energie versorgt. Sobald ein Abfragegerät nahe genug am Transponder ist, wird ein Energiefeld aufgebaut und der Transponder antwortet dem Lesegrät mit den gewünschten Informationen, wie z.B. der ID des Transponders. Aktive Transponder hingegen besitzen eine eigene Energieversorgung und senden ohne Einwirkung eines Energiefeldes eines Lesegeräts Informationen aus. Durch die integrierte Energiequelle können Entfernungen von mehreren Kilometern erreicht werden.

Je nachdem auf welcher Frequenz Transponder und Lesegerät operieren, werden unterschiedliche Distanzen zur Kommunikation erreicht. Die Frequenzen werden im Zuge der Entwicklung von Anwendungen durch der vorliegenden Anwendungsfall ausgewählt. Im Folgenden werden die Frequenzen und die Reichweiten aufgelistet [35]

- Langwellen-Frequenz mit $30 - 300$ kHz: $0,5$ m
- Kurzwellen-Frequenz mit $3 - 30$ MHz: $0,5$ m
- Dezimeterwellen-Frequenz mit $0,3 - 3$ GHz: $3 - 6$ m
- Mikrowellen-Frequenz mit mehr als 3 GHz: 10 m

Nach *ISO* kann RFID in folgende Klassen eingeteilt werden [35]

- Close-coupling systems mit einer Reichweite von bis zu einem Zentimeter (ISO 10536)
- Proximity remote-coupling systems mit einer Reichweite von mit zu zehn Zentimetern (ISO 15693, ISO 18000-2, ISO 18000-3)
- Vicinity remote-coupling systems mit einer Reichweite von bis zu einem Meter (ISO 14443, ISO 18000-3)

- Long-range systems mit einer Reichweite von einem bis zu zehn Metern(ISO 18000-4, ISO 18000-6, ISO 18000-7)

Neben der Identifikation und dem Auffinden von Objekten kann durch RFID die Position eines mobilen Geräts auf einer fiktiven Karte bestimmt werden. Hierzu müssen die Transponder an fixen Punkten platziert werden und eindeutig mit Positionen auf einer Karte verknüpft werden. Befindet sich das mobile Gerät dann in einem Bereich nah an einem Transponder, so kann die Position über dessen eindeutige Position und Identifikation bestimmt werden. RFID eignet sich aber aufgrund der eher kurzen Distanzen und der selten vorhandenen Sensorik in mobilen Geräten eher weniger für die Positionsbestimmung.

2.5.2 NFC

NFC steht für *Near Field Communication* und ist eine Erweiterung bzw. Spezialisierung von RFID. Wie der Name bereits beschreibt, wird NFC ausschließlich für die Kommunikation über sehr kurze Distanzen von einigen Millimetern bis wenigen Zentimetern verwendet. Dies hat den Vorteil, dass eventuelle datenschutzkritische Szenarien vermieden werden, die bei RFID stattfinden könnten. Hierzu kann z.B. das *Tracking* von Personen hinzugerechnet werden. Mit RFID ist es möglich, Personen in bestimmten Umgebungen wiederzuerkennen und damit ein Persönlichkeitsbild zu erstellen, da auch Systeme mit größeren Distanzen existieren. Durch NFC wird dies verhindert, indem die Kommunikation eben nur bei sehr geringem Abstand möglich ist.

NFC eignet sich aufgrund seiner Eigenschaften nicht für die Positionsbestimmung eines mobilen Endgeräts. Denn Jedes Mal, wenn die Position bestimmt werden sollte, ein fix positioniertes NFC Gerät direkt kontaktiert werden müsste und weiter eine sehr große Anzahl an fixen NFC-Geräten notwendig wäre, um eine relativ genaue Position zu bestimmen.

2.5.3 ZigBee

ZigBee ist ähnlich wie Bluetooth ein Netzwerk, das ohne komplexe Infrastruktur und ad-hoc erstellt werden kann. Die Technologie wurde mit dem Ziel entworfen, große Sensor-Netzwerke aufzubauen und durch eher niedrige Datenraten und geringen Energieverbrauch Geräte mit langen Laufzeiten herzustellen. Ein weiterer Unterschied gegenüber Bluetooth ist der, dass ZigBee-Netzwerke aus mehreren hundert Geräten bestehen können, was auf

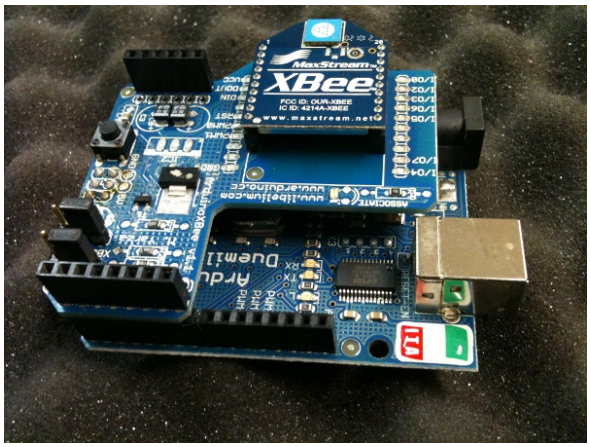

Abbildung 2.9: Ein ZigBee-Modul, aufgesteckt auf einem Arduino [86]

einer Adressierung mit meistens 16 oder 32 Bit beruht. Bluetooth-Netzwerke bestehen jedoch nur aus einem Master und 7 Slaves bzw. 255 geparkten[2] Geräten, wobei die Adressierung zwischen Master und Slaves bei 3 Bit liegt und geparkte Geräte mit 8 Bit adressiert werden. Dadurch lassen sich sehr große Sensor-Netzwerke erstellen, auch über größere Distanzen hinweg. Durch ein solches Netzwerk lassen sich die unterschiedlichsten Sensoren miteinander kombinieren und damit z.B. *Homeautomation* durchführen. Beispielsweise können ein Lichtsensor, ein Bewegungssensor und Zeitsensor miteinander verbunden werden, um bei geeignetem Kontex, z.B. wenn keine Personen den Bewegungsmelder auslösen, es eine Uhrzeit am späten Abend ist und es außerhalb des Raums dunkel ist, das Licht im Raum auszuschalten. Da Zigbee-Endgeräte die meiste Zeit keine Daten versenden, um damit so viel Strom wie möglich zu sparen, ist die Positionsbestimmung über Triangulation, Trilateration, Heatmaps, etc. nicht bzw. kaum möglich. ZigBee ist also ähnlich wie Bluetooth eine energiesparende Trägetechnologie, über die Sensordaten übertragen werden können. Sie eignet sich aber im Gegensatz zu Bluetooth weniger zur Positionsbestimmung, aufgrund der langen Inaktivität von von ZigBee-Modulen und dem nicht-implementierten Protokoll auf mobilen Endgeräten.

ZigBee-Module lassen sich in drei Kategorien einteilen - Koordinatoren (ZC), Router (ZR) und Endgeräte (ZED). Ein ZC ist der Initiator und gleichzeitig die Wurzel des Netzwerks. An einem ZC können mehrere ZR angehängt werden, welche die eigentlichen Anwendungs-

[2] Geparkte Bluetooth-Endgeräte befinden sich zwar im gleichen PAN, können aber nicht an der Kommunikation und Interaktion teilnehmen. Möchte ein geparktes Endgerät an der Kommunikation teilnehmen, so muss es zu einem Slave werden. Die geht jedoch nur dann, wenn im PAN noch keine 7 Slaves aktiv sind.

Abbildung 2.10: Ein Marker, der über eine Kamera erkannt werden kann

funktionen ausführen und Daten von anderen ZR oder ZED durch das Netzwerk führen. ZEDs sind die Blätter eines solchen Netzwerks und können auch nur als Blätter integriert werden. Diese Geräte bestehen meist aus einem Sensor und ansonsten so wenig Hardware wie möglich, um somit eine sehr hohe Batterielaufzeit zu gewährleisten.

In Abbildung 2.9 ist ein aufgestecktes ZigBee-Modul auf einem *Arduino* abgebildet. Das ZigBee-Modul selbst kann wiederum um weitere Sensoren erweitert werden.

2.5.4 Kamera

Durch die eingebaute Kamera in mobilen Geräten lässt sich zum einem über geeignete Vorgehensweisen die Position bestimmen und zum anderen ist es möglich, die Kamera z.B. als Bewegungssensor zu verwenden.

Eine einfache Möglichkeit die Position des mobilen Endgeräts über die Kamera zu bestimmen, ohne auf den GPS-Sensor oder andere Technologien zur Positionsbestimmung zurückzugreifen, gelinkt über das optische Scannen von Markern. Die Funktionsweise ähnelt der von RFID, die in Kapitel 2.5.1 vorgestellt wurde. Die Marker werden an fixen Orten positioniert und haben entweder ihre Position innerhalb der Markerstruktur codiert oder aber besitzen einen eindeutigen mit diesem Ort verknüpften codierten Bezeichner. Dieser Bezeichner kann z.B. mit einem WebService übersetzt und mit einer zugehörigen Position in Verbindung gebracht werden. Ein Marker mit einer binären Codierung von 16 Bit als schwarze und weiße Quadrate innerhalb des Markers ist in Abbildung 2.10 dargestellt.

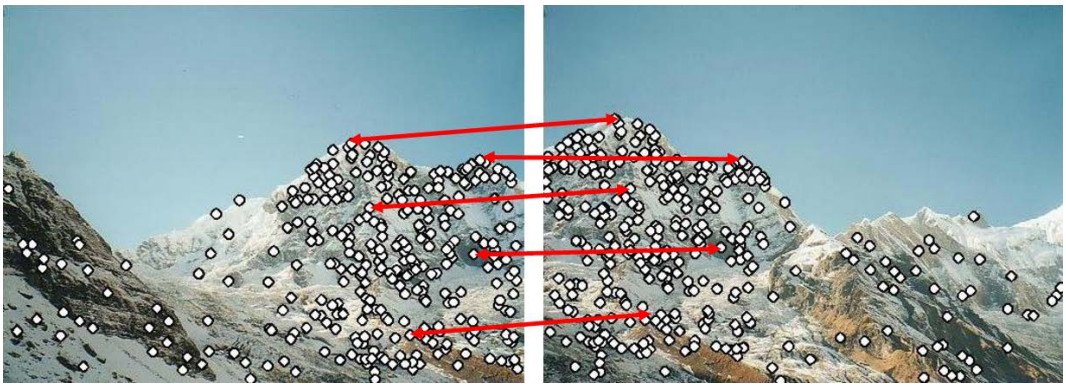

Abbildung 2.11: Vergleich von Features von aufgenommenem und hinterlegtem Bild [34]

Eine weitere aber weitaus komplexere Möglichkeit eine Positionsbestimmung über die Kamera durchzuführen, gelingt über das Vergleichen des aufgenommenen Bildes mit Bildern in einer Datenbank. Dieses Schema ist in Abbildung 2.11 dargestellt und läuft wie folgt ab.

Über die Kamera des mobilen Endgeräts wird ein Foto oder ein Video aufgenommen. In diesem Bild werden nun *Features* extrahiert, z.B. sehr markante Punkte oder andere signaltheoretische Eigenschaften des Bildes. Algorithmen die dazu verwendet werden können, sind unter anderem aus dem Bereich der *Edge Detection* der *Sobel*-Operator [58] oder der *Laplace*-Filter [57]. Diese generieren anhand des aufgenommenen Bildes eine Kantendetektion. Kanten in einem Bild sind im Grunde nichts anderes, als Hoch-, Tief und Wendepunkte der Signalfunktion des Bildes. Weitaus markantere Punkte im Bild können über Algorithmen aus dem Bereich der *Interest Point Detection* gefunden werden. Hierzu zählen der *Moravec*-Operator [49] und der *Harris*-Detektor [23]. Beide Herangehensweisen detektieren nicht alle Kanten in einem Bild, sondern vor allem Ecken, da diese weitaus markanter und mehr Informationen über das Bild bereitstellen als Kantendetektoren.

Nachdem die markanten Punkte aus einem Bild extrahiert wurden, müssen diese in einer zur Verarbeitung geeigneten Form dargestellt werden, um diese später mit anderen Bildern vergleichen zu können. Eine weit verbreitete Möglichkeit der Darstellung und Beschreibung von Features ist *SIFT* (Scale Invariant Feature Transform). Die so erhaltenen Features werden nun mit den Features von Bildern in einer Datenbank verglichen. Wird dabei ein Bild bzw. ein Bild mit einer gewissen Anzahl von übereinstimmenden Features gefunden und hat das Bild in der Datenbank eine Position z.B. als GPS-Koordinaten hinterlegt, so kann dem Benutzer mitgeteilt werden, wo er sich ungefähr befindet.

2.6 Zusammenfassung

In diesem Kapitel wurden einerseits die wichtigsten Sensoren zur Kontextbestimmung vorgestellt, die auf so gut wie allen mobilen Geräten vorhanden sind, und zum anderen einige Sensoren und Technologien, die dafür zwar verwendet werden können, aber weniger verbreitet sind. Da in der Implementierung des Krankenhaus-Prozesses der Kontext die aktuelle Position des Benutzers und des mobilen Geräts beinhaltet, soll an dieser Stelle nochmals auf die Technologien zur Positionsbestimmung eingegangen werden.

Zur Positionsbestimmung eignen sich die Sensoren GPS, WLAN, Bluetooth, die Mobilfunktechnologie, RFID und die Kamera. GPS und die Mobilfunktechnologie besitzen ihre Stärken vor allem in der Positionsbestimmung in Gebieten außerhalb von Gebäuden. Sobald eine Position innerhalb von Gebäuden bestimmt werden muss, wird die Position auf Grund von Abschirmung durch Wände und anderen Gegenständen ungenau. Deshalb müssen innerhalb von Gebäuden andere Sensoren verwendet werden. Wie gezeigt wurde, eignen sich dafür RFID und die Kamera, indem RFID-Tags und Marker mit fixen Position innerhalb des Gebäudes in Verbindung gebracht werden. Dies funktioniert aber nur, solange der Benutzer in Kontakt mit RFID-Tags oder Markern tritt. Soll die Position auch ohne direkten Kontakt innerhalb von Gebäuden bestimmt werden, können WLAN und Bluetooth herangezogen werden. Beide Technologien haben eigentlich die Aufgabe, Daten in einem Netzwerk zu übertragen. Durch deren ausgestrahlten Signale und Eigenschaften kann als Nebeneffekt aber auch die Position bestimmt werden. Bluetooth hat weiterhin noch die positive Eigenschaft, dass aufgrund des neuen Bluetooth LE Standards sehr wenig Energie verbraucht wird und durch die iBeacon-Technologie zusätzlich zur Positionsbestimmung auch direkter Kontakt mit einem solchen iBeacon erkannt werden kann. Dadurch können zum einen Positionsbestimmungen durchgeführt werden und zum anderen NFC- oder RFID-ähnliche Anwendungen implementiert werden. Aus diesem Grund und weil die Technologie von Bluetooth LE und den iBeacons relativ neu ist, wird der Krankenhaus-Prozess, welcher weiter unten vorgestellt und implementiert wird, auf Bluetooth und iBeacons zur Positionsbestimmung basieren. Im nächsten Kapitel werden deshalb die iBeacon-Technologie, die Funktionsweise und ihre Eigenschaften genauer vorgestellt.

3 | iBeacon-Technologie

Ein *iBeacon* [3, 46] ist eine auf der Funktechnologie *Bluetooth 4.0* [16] und insbesondere auf der *LE*-Spezifikation basierende Technologie, welche im Jahr 2013 von *Apple* [1] vorgestellt wurde und seitdem in den Geräten und Betriebssystemen von Apple aber auch in Geräten und Betriebssystemen anderer Hersteller implementiert wird.

iBeacons sind kleine mobile und vor allem energiesparende Endgeräte, die meist nicht auf eine kabelgebundene Stromanbindung angewiesen sind, sondern ihre Energie aus im iBeacon eingebauten Batterien entnehmen. Aufgrund der dadurch ermöglichten Mobilität und der Energiezufuhr aus Batterien ist es möglich, dass diese Endgeräte frei in Räumen und an schwer zugänglichen Orten positioniert werden können.

iBeacons tun nun nichts anderes, als periodisch bestimmte Datenpakete zu versenden und über eine spezifische Funkfrequenz zu publizieren. Diese Datenpakete werden von einem Empfänger verwendet, um dessen Position zu bestimmen, weitere Informationen über den iBeacon zu erfahren, sich mit diesem iBeacon über Bluetooth für weiteren Datenaustausch zu verbinden oder benachrichtigt zu werden, wenn ein bestimmter iBeacon in der Nähe aufgefunden wurde [1]. Anzumerken ist an dieser Stelle, dass sowohl der iBeacon als auch der Empfänger den Bluetooth 4.0 LE Standard implementieren müssen. Aus der Sicht des Datenschutzes ist es für iBeacons nicht möglich, sich ein Bild vom Nutzer zu machen oder Persönlichkeitsinformationen zu sammeln, da iBeacons lediglich ihre eigenen Daten versenden, es aber ohne Zutun des Empfängers nicht möglich ist, Informationen über Empfänger oder deren Benutzer zu erhalten.

Da der Krankenhaus-Prozess, welcher später in dieser Arbeit vorgestellt und implementiert wird, auf iBeacons, der Positionsbestimmung und der Informationsabfrage basiert, werden im Folgenden die Grundzüge von Bluetooth 4.0 LE, der Aufbau von iBeacons, deren spezifische Nachrichten und die Nutzung dieser Nachrichten erläutert.

3.1 Bluetooth LE

Bluetooth LE (*Bluetooth Low Energy*, *BLE*), auch *Bluetooth Smart* genannt, ist eine Erweiterung der klassischen Bluetooth-Spezifikation um besonders energiesparende Funktionen und Verhaltensweisen [15]. BLE wurde parallel zur eigentlichen Weiterentwicklung von Bluetooth durch *Nokia* initiiert, um über die bereits existierende Spezifikationen eine energiesparende Erweiterung zu entwickeln. Das daraus entstandene Produkt wurde 2006 vorgestellt und erhielt den Namen *Wibree* [52]. Im Jahr 2010 wurde Wibree in die Spezifikation von Bluetooth 4.0 mit aufgenommen und seitdem als Bluetooth LE verbreitet [52].

3.1.1 Unterscheidung von Endgeräten

Da nicht alle Endgeräte BLE unterstützen und Geräte, die BLE nutzen wollen jedoch nur mit BLE-konformen Endgeräten kommunizieren können, werde Endgeräte wie folgt voneinander differenziert [14]:

- **Bluetooth Smart Ready** Endgeräte kommunizieren sowohl über klassisches Bluetooth als auch über BLE
- **Bluetooth Smart** Endgeräte kommunizieren nur über BLE, d.h. mit Bluetooth Smart- oder Bluetooth Smart Ready-Endgeräten
- **Classic** Endgeräte kommunizieren über klassisches Bluetooth mit Bluetooth Smart Ready- oder anderen Classic-Endgeräten.

3.1.2 Frequenz und Datendurchsatz

BLE funkt und operiert im *ISM*-Band (*Industrial, Scientific, Medical*) wie klassisches Bluetooth [9]. Das ISM-Band ist ein frei zugängliches und unlizensiertes Frequenzband mit einer Breite von 2.4 GHz bis 2.485 GHz [55]. Da dieses Frequenzband frei zugänglich ist, funken sowohl viele Endgeräte als auch andere Protokolle und Technologien wie WLAN, RFID oder Mikrowellenherde auf diesen Frequenzen, was unter Umständen und der Anwesenheit vieler Endgeräte zu Interferenzen und damit zu Ungenauigkeiten, Fehlübertragungen und weiteren Problemen führen kann. Solche Probleme werden später im Krankenhaus-Prozess und dessen Beurteilung vorgestellt. Um Interferenzen zu vermeiden bzw. zu minimieren, wird wie im klassischen Bluetooth Standard ein *Frequency Hopping Spread Spec-*

trum Algorithmus verwendet [9, 37]. Dabei wird das ISM-Band bei BLE in 40 Kanäle mit jeweils einer Breite von 2 MHz aufgeteilt [11]. Die Kommunikation, basierend auf einer Berechnung zwischen Sender um Empfänger, wird nicht auf einem einzelnen Kanal, sondern auf einer mehr oder weniger zufälligen Abfolge von Kanälen durchgeführt. Drei dieser Kanäle sind als *Advertisement Channel* markiert. Auf diesen Kanälen machen sich Geräte sichtbar, indem sie Datenpakete publizieren, die von anderen Geräten auf diesen Kanälen empfangen werden können. Die restlichen 37 Kanäle sind *Data Channel*, über die eine normale Kommunikation durchgeführt werden kann, die dem klassischen Bluetooth ähnelt [13].

Da BLE in der Spezifikation eine besonders energiesparende Kommunikation vorsieht, ist der theoretische Datendurchsatz von BLE mit 1 MBit/s niedriger als der des klassischen Bluetooth [10]. Anzumerken ist auch, dass BLE nicht vorsieht, dass Endgeräte an sich besonders energiesparend sind, sondern Profile und Kommunikation innerhalb von BLE auf eine energiesparende Art und Weise ausgelegt sind.

3.1.3 Kommunikation

BLE basiert auf *GAP* (*Generic Access Profile*) wie auch das klassische Bluetooth. GAP ist die Grundlage für alle anderen Profile in allen Bluetooth-Spezifikationen und definiert, wie sich Geräte über Funk gegenseitig finden und verbinden können [38, 45]. GAP ist also unter anderem dafür zuständig, dass ein Endgerät überhaupt gefunden werden kann. Weiter werden Bluetooth-Geräte nach GAP in zwei Klassen eingeteilt, in *Master* und *Slaves*. Die Kommunikation zwischen zwei verbundenen (oder gekoppelten) Geräten basiert auf *Polling*, das durch einen Master initiiert wird. Ein Master, der eine Verbindung zu bis zu 7 Slaves besitzen kann, pollt seine Slaves bei Bedarf nach Daten. In BLE wird diese Begrenzung auf 7 Slaves und damit einer Adressierung von 2 Bit nicht vorgeschrieben. Ein Master ist meist ein Endgerät mit hohen Rechenkapazitäten, z.B. ein Smartphone oder ein PC. Slaves hingegen sind Endgeräte, die meist wenig Rechenkapazitäten vorweisen und über Batterien betrieben werden (z.B. Sensoren oder iBeacons). Weiter befinden sich die Geräte laut GAP entweder in einem *Advertising Mode* (*AM*) oder in einem *Scanning Mode* (*SM*), wobei respektive *Advertising Data Packages* (*AD*) oder *Scan Response Data Packages* (*SRD*) versendet werden. In beidem Modi werden Pakete mit einer Größe von bis zu 47 Byte in den oben genannten Advertisement Channels versendet [12]. Der nach dem Standard verpflichtend zu implementierende AM versendet periodisch alle 20 ms bis

zu 10 Sekunden ein solches Paket. Wenn ein Gerät im SM ein solches Paket erhält, kann es, falls auf diesem Gerät implementiert, ein SRD an das aufgefundene Endgerät senden, um damit wiederum ein AD zu erhalten, in welchem beispielsweise weitere Informationen eingebunden sind. Je kleiner das Intervall zwischen den ADs ist, desto schneller kann ein solches Gerät gefunden werden. Dadurch nimmt jedoch gleichzeitig der Energiebedarf drastisch zu.

Sobald zwei Geräte über GAP Informationen ausgetauscht haben, können die Geräte nach einer Kopplung über *GATT* (*Generic Attribute Profile*, [40, 41]) und die oben genannten 37 Data Channels kommunizieren und Daten austauschen [39]. Die Datenpakete in GATT besitzen im Gegensatz zu GAP keine Limitierung auf 47 Byte, sodass es möglich ist, größere Datenmengen auszutauschen.

iBeacons sind meist einfache Endgeräte in der Kategorie der Slaves und befinden sich im AM. Sie senden also in festgelegten periodischen Intervallen ADs als *Broadcast* über einen der drei Advertisement Channels an alle Geräte, die sich in Reichweite[1] und im SM befinden. Aus diesem Grund sollen die versendeten Pakete und deren Struktur im Folgenden Kapitel genauer untersucht werden.

3.2 Nachrichteninhalt und Aufbau

In Kapitel 3.1.3 wurde beschrieben, dass iBeacons meist zu der Geräteklasse der Slaves zählen. Sie senden aus dem AM heraus Datenpakete mit einer Größe von 47 Byte an alle möglichen Empfänger im SM, die sich innerhalb der Reichweite zum sendenden Gerät befinden. Deshalb werden diese Datenpakete in diesem Kapitel genauer untersucht.

3.2.1 Aufbau

Die 47 Bytes des AD lassen sich in vier Bereiche einteilen [12]. Am Anfang des Pakets ist eine Präambel mit einer Größe von 1 Byte zu finden. Die Präambel hat die Form `01010101`, falls das erste Bit des darauffolgendes Blocks an der ersten Stelle eine `0` gesetzt hat, ansonsten `10101010`. Auf die Präambel folgt die *Access Address* mit einer Größe von 4 Byte. ADs werden immer mit `0xD6BE898E` markiert. Datenpakete wie z.B. SRDs und die darauf

[1] Bluetooth-Signale legen meist eine Entfernung von bis zu 100 Metern zurück. Je nach Art der zugrundeliegenden Hardware sind auch höhere und niedrigere Distanzen möglich.

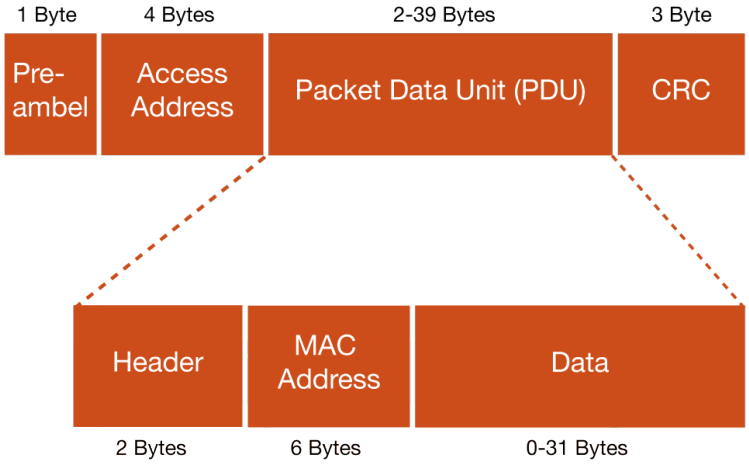

Abbildung 3.1: Aufbau eines Bluetooth LE Advertising Data-Pakets

folgende Antwort mit einem zufälligen Wert pro Verbindung. Auf diesen Block folgen die eigentlichen Daten (*PDU*, *Packet Data Unit*) des Pakets, mit einer Größe zwischen 2 und 39 Bytes. Auf den Inhalt der PDU wird ein *CRC* (*Cyclic Redundancy Check*, *Zyklische Redundanzprüfung*, [64]) mit einer Größe von 3 Byte angewandt und der CRC selbst ans Ende des AD angehängt.

Die PDU des AD mit einer Größe von 2 bis 39 Bytes ist laut Standard mit einem Header mit einer Länge von 2 Byte, der Advertiser Address bzw. MAC-Adresse des Senders mit einer Größe von 6 Bytes und den eigentlichen Daten (*Payload*), die eine Länge von 0 bis 31 Bytes besitzen können, ausgestattet [12]. Der Aufbau eines AD und der PDU ist in Abbildung 3.1 dargestellt. Innerhalb des Payloads der PDU sind einige für iBeacons spezifische Daten untergebracht. Dadurch ist es Frameworks wie z.B. *CoreBluetooth* oder *CoreLocation* von Apple möglich, die sendenden iBeacons zu identifizieren und mit Informationen in Verbindung zu bringen.

Ein Beispiel für ein AD, dass von einem iBeacon gesendet wurde, könnte wie folgt aussehen (alle Informationen liegen in Hexadezimaldarstellung vor):

```
aa d6 be 89 8e 40 25 00 02 72 c5 d7 9d 1e 02 01 1a 1a ff 4c 00 02 8e 14 41 1d 20
    5d 3a 42 8e a2 8f ea db de dd 79 00 00 00 01 c4 d8 ce
```

Diese Hexadezimalzahlen lassen sich nun anhand der Struktur eins AD folgend einteilen.

```
aa                  // Präambel
d6 be 89 8e         // Access Address
```

```
---------------------- PDU --------------------------
40 25               // Header der PDU
00 02 72 c5 d7 9d   // MAC-Adresse der PDU
1e 02 01 1a 1a ff 4c 00 02 15 8e 14 41 1d 20 5d 3a 42 8e a2 8f ea db de dd 79 00
    00 00 01 c4     // Payload der PDU
-----------------------------------------------------
d8 ce 12            // CRC
```

Als nächstes müssen die Daten innerhalb der PDU genauer untersucht werden. Wurde das AD von einem iBeacon aus gesendet, dann besitzt der Payload immer die gleiche Struktur und lässt sich grob in fünf wichtige Bestandteile auflösen: *Präfix*, *UUID*, *Major*, *Minor* und *RSSI* (s. Abbildung 3.2) [46, 51, 84]. Wenn der Payload der PDU nach diesem Schema aufgeteilt wird, erhält man folgende Daten.

```
----------------------------- Präfix ------------------------------
1e    // Anzahl der Bytes (30) die im Payload folgen
02    // Anzahl der Bytes (2) im ersten Block
01    // Flag des Advertising Data Typ
1a    // steht binär für 00011010 und setzt folgende Bluetooth-Werte
      // bit 0 (aus): LE Limited Discoverable Mode
      // bit 1 (an): LE General Discoverable Mode
      // bit 2 (aus): BR/EDR Not Supported
      // bit 3 (an): Simultaneous LE and BR/EDR to Same Device Capable  (
          controller)
      // bit 4 (an): Simultaneous LE and BR/EDR to Same Device Capable (host)
1a    // Anzahl der Bytes (26) im zweiten Block
ff    // Herstellerspezifischer Advertising Data Typ
4c 00 // Code zur Herstelleridentifikation (Apple)
02    // Erstes Byte als Indikator für einen iBeacon
15    // Zweites Byte als Indikator für einen iBeacon
-------------------------------------------------------------------
e2 c5 6d b5 df fb 48 d2 b0 60 d0 f5 a7 10 96 e0 // UUID
00 00 // Major-Wert
00 01 // Minor-Wert
c4    // RSSI (kalibrierte Signalstärke im Zweierkomplement
```

Erhält ein Framework oder Endgerät nun über BLE ein solches AD, so kann es anhand dieser Daten eine Zuordnung zu einem iBeacon und eine eventuelle Positionsbestimmung

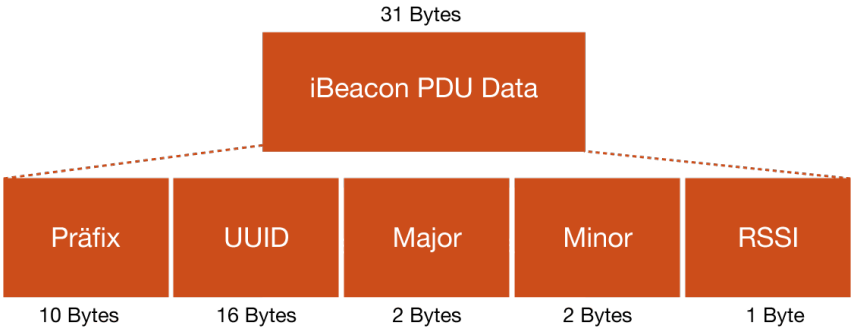

Abbildung 3.2: Aufbau des Payloads einer PDU eines iBeacons

vornehmen. Dabei sind die UUID, der Major- und Minor-Wert und der Wert im RSSI-Feld von großer Bedeutung [46]. Aus diesem Grund werden diese Daten in den folgenden Kapiteln weiter vertieft und deren Nutzen beschrieben.

3.2.2 UUID

Eine *UUID* (*Universal Unique Identifier*) ist eine eindeutige 16 Byte lange Zeichenfolge, welche dazu verwendet wird, einen iBeacon eindeutig zu identifizieren. Aufgrund der Länge und Hinzunahme des *Gerburtstagsparadoxon* ist die Wahrscheinlichkeit, dass ein iBeacon ungewollt und zufällig die gleiche UUID wie ein anderer iBeacon hat, vernachlässigbar klein [71]. iBeacons der gleichen Institution wie z.B. dem Krankenhaus im Krankenhaus-Prozess haben meist die gleiche UUID. Damit kann eine Anwendung dahingehen konfiguriert werden, dass sie nur iBeacons mit einer oder wenigen eindeutigen UUIDs weiter betrachtet und zur Verarbeitung verwendet.

3.2.3 Major und Minor

Anhand der beiden Felder *Major* und *Minor*, welche beide eine Breite von 2 Byte haben, können UUIDs von iBeacons einer Institution weiter verfeinert werden. Jedes dieser Felder kann also einen Wert zwischen 0 und 65535 annehmen. iBeacons, die einer spezifischen UUID angehören, können also weiter in $65536 * 65536 = 4294967296$ unterschiedliche und eindeutig zur einer UUID zuordbare iBeacons eingeteilt werden. Der Major-Wert unterteilt dabei die UUID auf der ersten Ebene in 65536 weitere Subebenen, welche durch den Minor-Wert wiederum um die gleiche Anzahl an Subebenen eingeteilt werden können. Ei-

ne Institution, die eine eindeutige UUID inne hat, könnte durch den Major-Wert einzelne und örtlich verteilte Dienststellen identifizieren (z.B. Major=1: München, Major=2: Ulm) und durch den Minor-Wert jeweils in diesen Dienststellen bestimmte Organisationseinheiten oder Stockwerke (z.B. Minor=0: Erdgeschoss, Minor=1: 1. Stock). Wie UUIDs, Major- und Minor-Werte in expliziten Szenarien verwendet werden, wird in Kapitel 5 beschrieben

3.2.4 RSSI

Durch den Wert im *RSSI*-Feld (*Received Signal Strength Indication*), welcher eine Größe von 1 Byte besitzt und im Zweierkomplement angegeben ist, ist es der Anwendung bzw. dem Empfangsgerät möglich, die Entfernung zum iBeacon zu messen, von dem die Daten gesendet wurden. Anhand dieser Entfernung bzw. der Signalstärken können mithilfe von Trilateration genaue Positionen innerhalb von Gebäuden berechnet werden und vordefinierte Entfernungen zu iBeacons bestimmt werden, die ebenfalls eine weitere Unterscheidung von iBeacons zulässt. Wie dies genau funktioniert und welche Anwendungsfälle darauf basieren können, wird im nächsten Kapitel beschrieben.

3.3 Positionsbestimmung

Bevor die Entfernung zu einem iBeacon möglichst genau berechnet werden kann, muss dessen RSSI-Wert bzw. die initiale Signalstärke des Bluetooth-Senders kalibriert werden. Dazu muss mit einem Empfängergerät die Signalstärke aus einer Entfernung von einem Meter zum iBeacon gemessen werden. Der dadurch erhaltene Wert, welcher auf iPhones in *dBm* (Dezibel Milliwatt) angegeben und berechnet wird, muss im iBeacon hinterlegt werden und mit jedem AD im Payload der PDU mitgeliefert werden. Anhand des kalibrierten und mitgelieferten Wertes ist es der Anwendung möglich, unter Hinzunahme der eigentlichen empfangenen Signalstärke die Entfernung zu bestimmen.

Die Berechnung der Entfernung zwischen iBeacon und Empfänger wird anhand dieser Werte mit folgender Rechnung durchgeführt [7]. S_r ist die auf dem Empfängergerät empfangene Signalstärke RSSI und n die kalibrierte und normalisierte Signalstärke des iBeacons, welcher im Payload mitgeliefert wird. Dann kann die Entfernung r durch folgende Formel berechnet werden:

$$S_r = -10 * \lambda * log_{10}(r) + n \tag{3.1}$$

$$\Leftrightarrow r = 10^{-(S_r)/10*\lambda} * n \tag{3.2}$$

wobei λ der Verlust der Signalstärke ist und $\lambda = 2$ für den Verlust durch die Luft ohne Wände oder andere Störgegenstände gesetzt wird.

Hat das Gerät auf Empfängerseite Kontakt zu drei iBeacons, so kann mithilfe der in Kapitel 2.2.2 vorgestellten Trilateration eine relativ exakte Position in einem definierten Raum über Bluetooth und die oben beschriebene Formel berechnet werden. Befinden sich die iBeacons immer am selben fixen Standort, so ist auch eine gröbere Bestimmung der Position über die Methodik der Cell of Origin möglich, welche sich durch die Berechnung der Entfernung über die Signalstärke sogar verbessern lässt.

Apple stellt seit iOS 7 in seinen Frameworks eine Reihe von Konstanten bereit, die die Entfernung zu einem iBeacon anhand der RSSI-Werte beschreiben. Die Konstante *Immediate* sagt aus, dass das Endgerät sehr nah (einige cm) am iBeacon positioniert ist. *Near* beschreibt eine relativ nahe Entfernung zwischen 1 und 3 Metern zum iBeacon. Als *Far* wird alles bezeichnet, was zwischen 3 und ca. 100 Metern Entfernung liegt. Die vierte Konstante *Unknown* sagt aus, dass die Entfernung aufgrund möglicher Störquellen, Interferenzen oder sonstigen Ungenauigkeiten nicht berechnet werden kann. Es wird zwar ein iBeacon gefunden, dessen genaue Entfernung zu bestimmen ist aber im Moment nicht möglich. Über diese Konstanten ist es der Anwendung und dem Benutzer folglich möglich zu bestimmen, wie weit entfernt dieser sich von einem solchen iBeacon aufhält bzw. ob sich ein interessanter iBeacon in der Nähe befindet, welcher für eine explizite Anwendung von Interesse sein könnte.

An dieser Stelle soll zur Bestimmung der Position und der Entfernung zwischen iBeacon und Endgerät erwähnt werden, dass Störsignale und Interferenzen mit anderen Signalen und Geräten aber auch Wänden, Gegenständen und Menschen die Bestimmung des RSSI und die Kalkulation der Entfernung stark beeinflussen können, woraus falsche und ungenaue Werte resultieren können.

3.4 Geräte

Seitdem Apple die iBeacon-Technologie 2013 vorgestellt hatte, starteten Unternehmen das Vorhaben, iBeacon-Geräte zu entwickeln und kommerziell zu vermarkten Auch während dieser Arbeit wurden neue Gräte vorgestellt und auf dem Markt angeboten. Da Apple seit Februar 2014 [2] zusätzlich eine Zertifizierung [54] für iBeacon-Geräte für Unternehmen zur Verfügung stellt und die Unternehmen und deren Geräte immer bekannter werden, werden an dieser Stelle einiger dieser Geräte vorgestellt.

Eines der ersten Unternehmen, das die iBeacon-Technologie in eigene Gräte implementierte, ist *Radius Networks* [70]. Radius Networks leistete einen großen Beitrag zum Reengineering-Prozess des iBeacon-Protokolls, bevor Apple die Spezifikation für zertifizierte Unternehmen freigab. Das Unternehmen entwickelte ebenfalls eines der ersten Frameworks für *Android*, um iBeacons auf Smartphones und Tablets mit *Google*s Betriebssystem zu portieren. Es bietet die unterschiedlichsten Arten von iBeacons an, z.B. USB-Sticks, iBeacons auf Raspberry PIs basierend und iBeacons die direkt in Steckdosen gesteckt werden können.

Estimote [31] bietet iBeacons in Dreierpackungen in verschiedenen Farben an, welche zusätzlich im Outdoor-Bereich aufgrund von Wasserfestigkeit eingesetzt werden können. Ein zerlegter iBeacon von Estimote ist in Abbildung 3.3 zu sehen. Wie dabei zu erkennen ist, ist der iBeacon kaum größer als eine Knopfbatterie. Das Unternehmen geht momentan sogar noch weiter und bietet seit kurzen iBeacons mit einer Höhe von nur 3 mm an, die neben der eigentlichen BLE- und iBeacon-Technologie einen Beschleunigungssensor und Temperatursensor beinhalten [33].

Ein weiterer Big Player im Bereich der iBeacon-Hersteller ist *Gimbal* [47]. Gimbal bietet wie auch Estimote und Radius Networks eigene SDKs und APIs zur Erstellung von Anwendungen und Konfiguration der iBeacons an. Damit ist es dem Entwickler oder Anwender möglich, UUID, Major, Minor und RSSI explizit zu bestimmen. Gimbal bietet eine sehr kleine aber weniger langlebige und eine größere aber mit höherer Energiekapazität ausgerüstete Versionen von iBeacons an.

Kontakt.io [60] bietet ebenfalls iBeacons, SDKs und APIs an. Das Unternehmen geht jedoch einen Schritt weiter und portierte das iBeacon-Protokoll auf die WLAN-Funktechnik, um weitere Arten von Anwendungen implementieren zu können. Dieser auf WLAN basierender iBeacon kann zusätzlich als Master von vielen Bluetooth-iBeacons verwendet wer-

Abbildung 3.3: Ein zerlegter iBeacon von Estimote [32]

den, um letztere vollkommen ohne direkten Kontakt und aus der Ferne konfigurieren und jederzeit anpassen zu können.

Neben diesen größeren Unternehmen existieren noch kleinere bzw. unbekanntere Unternehmen, die iBeacons in den unterschiedlichsten Konfigurationen, Hardwarespezifikationen und Größen vermarkten. Hierzu zählen z.B. *RedBearLab* [74] und *Bleu* [8].

Auch ein herkömmlicher Raspberry Pi kann mit geeigneten Hilfsmitteln in einen vollständigen iBeacon verwandelt werden. Da die Entwicklung der iBeacon-Verbreitung zu Beginn dieser Arbeit noch nicht so weit war und es einen tieferen Einblick in die technische Funktionsweise eines iBeacons gibt, wird im folgenden Kapitel dargestellt, wie eine Raspberry Pi in einen iBeacon verwandelt wird. So konfigurierte iBeacons werden neben iBeacons von Gimbal im Krankenhaus-Prozess verwendet und zur Bewertung in späteren Kapiteln hinzugezogen.

4 | Raspberry Pi

In der Implementierung des Krankenhaus-Prozesses werden unter anderem *Raspberry Pi*s als iBeacons verwendet. Da diese zusätzlich eine geeignete Grundlage zum Verständnis von iBeacons beitragen, werden sie in diesem Kapitel kurz vorgestellt. Weiter werden die Konfiguration eines Raspberry Pis hin zu einem iBeacon erläutert und die notwendigen Komponenten dargelegt.

4.1 Komponenten

Der Raspberry Pi [72] (s. Abbildung 4.1) ist ein kleiner voll funktionsfähiger Computer, der von der *Raspberry Pi Foundation* entwickelt wurde. Das Ziel war es, einen kleinen und günstigen Computer zu entwickeln, um jüngeren Menschen die Möglichkeit zu geben, sich mit Computer-Technik und Softwareentwicklung bzw. Programmierung beschäftigen zu können.

Seit der Veröffentlichung des ersten Raspberry Pis wurden drei Versionen vorgestellt, *Model A*, *Model B* und *Model B+*. In dieser Arbeit wurden ausschließlich Modelle des Typ B verwendet.

Um diesen benutzen zu können, benötigt man neben dem eigentlich Raspberry PI eine SD-Karte, welche als Festplatte dient, und ein Betriebssystem. Seit der Veröffentlichung wurden einige Betriebssysteme entwickelt und für den Anwender verfügbar gemacht. Die am weitesten verbreitete Linux-Distribution für das Gerät nennt sich *Raspbian* [73]. Raspbian wird auch in dieser Arbeit verwendet. Neben dem bekanntesten Betriebssystem existieren noch einige andere, wie z.B. *Risc OS*, *Arch Linux* oder *Pidora*. Um dem Benutzer eine einfache Installation des gewünschten Betriebssystems zu ermöglichen, kann *NOOBS* (*New*

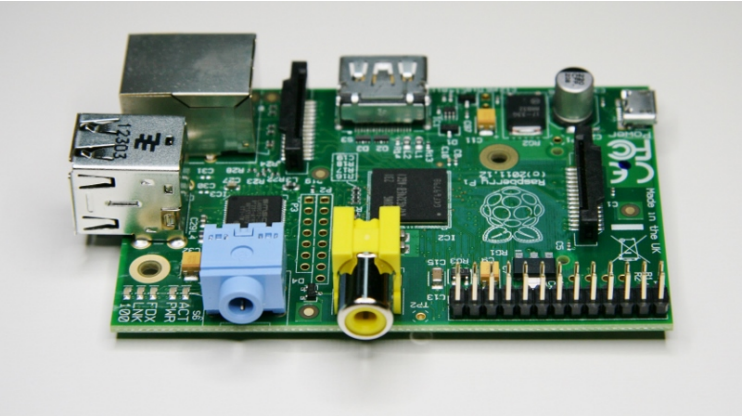

Abbildung 4.1: Raspberry Pi der Version Model B

Out Of the Box Software [62]) verwendet werden. Damit muss der Anwender sich nicht darum kümmern, die passende Distribution zu finden und auf der SD-Karte zu konfigurieren. Mit NOOBS ist es ihm möglich, durch eine Auswahl direkt das gewünschte Betriebssystem zu installieren.

Des Weiteren wird in dieser Arbeit ein WLAN-Adapter verwendet, um den Raspberry Pi bequem kabellos über entfernten *SSH*-Zugriff zu konfigurieren und zu administrieren.

Um aus dem Gerät einen iBeacon zu machen, wird ein Bluetooth LE-Dongle benötigt. Wichtig ist, wie im nächsten Kapitel aufgezeigt wird, dass der Dongle zum Bluetooth-Treiber *BlueZ* kompatibel ist. In dieser Arbeit wurde der kompatible *Plugable USB-BT4LE* verwendet.

Nachdem alle Komponenten gesammelt wurden und Raspbian auf dem Raspberry Pi installiert wurde, kann die Konfiguration zu einem iBeacon gestartet werden.

4.2 Konfiguration als iBeacon

Anhand der Informationen über die iBeacon-Technologie und Bluetooth LE aus Kapitel 3 ist es nun möglich, aus einem Raspberry Pi mit den oben angegeben Komponenten basierend auf Raspbian einen iBeacon zu erstellen. In diesem Kapitel soll der Prozess vom entfernten Einloggen über SSH bis zur Erstellung von Start- und Stop-Skripten erläutert werden.

4.2.1 Setup

Sobald man die IP des Raspberry Pis konfiguriert hat, ist es möglich sich einfach über SSH mit diesem zu verbinden. Dazu wird der initiale Benutzername `pi` und das Passwort `raspberry` von Raspbian benötigt.

```
philip-macbook:~ Philip$ ssh pi@192.168.2.119
pi@192.168.2.119's password: *********
```

Nachdem nun eine Verbindung zum Raspberry Pi besteht, müssen vor der Konfiguration zum iBeacon einige Bibliotheken für den Zugriff auf USB-Geräte und Hilfsbibliotheken installiert werden. Es empfiehlt sich ebenfalls, ein Update des Betriebssystems durchzuführen. Wichtig ist, dass diese Operationen mit Root-Rechten durchgeführt werden. Dazu kann nun wie folgt vorgegangen werden.

```
pi@raspberrypi ~ $ sudo -i
root@raspberrypi:~#
```

Danach werden Aktualisierungen für das Betriebssystem heruntergeladen und die notwendigen Bibliotheken installiert. Dazu werden folgende Befehle benötigt.

```
apt-get update
apt-get install libusb-dev
apt-get install libdbus-1-dev
apt-get install libglib2.0-dev --fix-missing
apt-get install libudev-dev
apt-get install libical-dev
apt-get install libreadline-dev
```

Nachdem die Aktualisierung abgeschlossen und die Bibliotheken installiert wurden, muss ein Bluetooth-Treiber installiert werden. *BlueZ* eignet sich hierfür sehr gut, da er Bluetooth LE implementiert und dadurch ein iBeacon erstellt werden kann. Zur Installation des Treibers müssen folgende Befehle eingegeben werden. Dadurch wird der Treiber heruntergeladen, entpackt und durch ein Make-Skript installiert.

```
wget https://www.kernel.org/pub/linux/bluetooth/bluez-5.9.tar.xz
tar xvJf bluez-5.9.tar.xz
cd bluez-5.9
./configure --disable-systemd --enable-library
make
```

```
make install
```

Das Herunterladen und Installieren des Treibers kann bis zu einer Stunde dauern. Nachdem die Installation abgeschlossen wurde, sollte der Raspberry Pi neu gestartet werden. Hierzu muss folgender Befehl verwendet werden.

```
shutdown -r now
```

Nun wurden alle wichtigen Vorkehrungen getroffen, um die eigentliche Konfiguration zu einem iBeacon durchzuführen. Als erstes muss nun der Identifikator des Bluetooth-Dongles herausgefunden werden. Hierzu wird der Befehl `hciconfig` eingegeben. Neben dem Identifikator `hci0` werden die MAC-Adresse als `BD Address`, die *MTU*s der beiden Protokolle *ACL* und *SCO*, der Status als `DOWN` und die übertragenen (*tx*) und empfangenen (*rx*) Bytes über ACL und SCO angezeigt.

```
root@raspberrypi:~# hciconfig
hci0: Type: BR/EDR  Bus: USB
  BD Address: 00:02:72:C5:D7:9D  ACL MTU: 1021:8  SCO MTU: 64:1
  DOWN
  RX bytes:827 acl:0 sco:0 events:32 errors:0
  TX bytes:433 acl:0 sco:0 commands:33 errors:0
```

Da der Bluetooth-Dongle momentan nicht aktiviert ist, muss dieser über dessen Identifikator und folgenden Befehl aktiviert werden, sodass sich dieser danach im Status `UP RUNNING` befindet.

```
root@raspberrypi:~# hciconfig hci0 up
root@raspberrypi:~# hciconfig
hci0: Type: BR/EDR  Bus: USB
  BD Address: 00:02:72:C5:D7:9D  ACL MTU: 1021:8  SCO MTU: 64:1
  UP RUNNING
  RX bytes:881 acl:0 sco:0 events:41 errors:0
  TX bytes:583 acl:0 sco:0 commands:42 errors:0
```

Nachdem der Bluetooth-Dongle aktiviert wurde, kann er durch folgende Befehle dazu gebracht werden, die in Kapitel 3.1.2 beschrieben ADs auf den Advertisement Channels zu versenden. Mit dem ersten Befehl wird der Payload des AD gesetzt. Dieser ist wie oben beschrieben hexadezimal angegeben und besteht unter anderem aus der UUID, Major, Minor und kalibriertem RSSI-Wert. In diesem Fall wird die UUID als `8e 14 41 1d 20 5d 3a`

`42 8e a2 8f ea db de dd 79` gesetzt, gefolgt von Major `00 9c`, Minor `00 00` und RSSI-Wert `c4`. Durch den zweiten Befehl und `leadv 3` wird angegeben, das keine Verbindungen über diesen Kanal zugelassen werden. Soll der iBeacon Verbindungen für weiterführende Anwendungsszenarien zulassen, so muss der Befehl mit `leadv 0` angegeben werden. Um das Broadcasten abzuschalten, muss der letzte Befehl lediglich mit `noleadv` eingegeben werden.

```
root@raspberrypi:~#  hcitool -i hci0 cmd 0x08 0x0008 1e 02 01 1a 1a ff 4c 00 02
    15 8e 14 41 1d 20 5d 3a 42 8e a2 8f ea db de dd 79 00 9c 00 00 c4
root@raspberrypi:~# hciconfig hci0 leadv 3
```

4.2.2 Skripte

Da ein Raspberry Pi auf eine direkte Stromquelle angewiesen ist und durch die Trennung von dieser herunterfährt, muss nach jedem Neustart der Bluetooth-Dongle neu gestartet, der Payload eingetragen und der Broadcast eingerichtet werden. Um diesen Vorgang zu automatisieren, wurden für die im Krankenhaus-Prozess verwendeten iBeacons Konfigurationsdateien und Skripte erstellt, die an dieser Stelle vorgestellt werden sollen.

iBeacon Config

Als erstes wird die Konfigurationsdatei `ibeacon.conf` (s. Listing 4.1) erstellt. In dieser werden die Variablen für den Identifikator des Bluetooth-Dongles, die UUID, Major, Minor und den kalibrierten RSSI-Wert deklariert und ihnen die notwendigen Werte zugewiesen. In diesem Fall die bereits oben verwendeten Werte für den Payload und der Identifikator `hci0`.

```
1 export DEVICE=hci0
2 export UUID="8e 14 41 1d 20 5d 3a 42 8e a2 8f ea db de dd 79"
3 export MAJOR="00 9c"
4 export MINOR="00 00"
5 export POWER="c4"
```

Listing 4.1: ibeacon.conf

Start-Skript

Als nächsten wurde ein Shell-Skript zum Starten des Bluetooth-Dongles und Broadcasten des AD mit dem gewünschten Playload erstellt (s. Listing 4.2). Wichtig ist hierbei in Zeile 1 der Befehl `#!/bin/sh`, womit das Skript als ein solches gekennzeichnet wird. In Zeile 2 werden die zuvor erstelle iBeacon-Konfiguration und ihre Variablen eingebunden. Die folgenden Zeilen sind aus den vorherigen Kapitel bekannt und sind dafür zuständig, den gewünschten Bluetooth-Dongle zu starten und den Payload zu setzen. Wichtig ist es, dem Skript die Rechte zur Ausführung über `chmod 777 start` zu geben.

```
1 #!/bin/sh
2 . ./ibeacon.conf
3 echo "Launching iBeacon..."
4 sudo hciconfig $DEVICE up
5 sudo hciconfig $DEVICE noleadv
6 sudo hcitool -i hci0 cmd 0x08 0x0008 1e 02 01 1a 1a ff 4c 00 02 15 $UUID $MAJOR
      $MINOR $POWER
7 sudo hciconfig $DEVICE leadv 3
8 echo "start: complete!"
```

Listing 4.2: start

Stop-Skript

Um die Möglichkeit zu haben, den Bluetooth-Dongle über einen einzigen kurzen Befehl zu beenden, wurde das `stop`-Skript erstellt (s. Listing 4.3). Wie dem Skript zum Starten des iBeacons muss auch das Stop-Skript die Rechte für die Ausführung über `chmod 777 stop` erhalten.

```
1 #!/bin/sh
2 . ./ibeacon.conf
3 echo "Disabling iBeacon..."
4 sudo hciconfig $DEVICE noleadv
5 echo "stop: complete"
```

Listing 4.3: stop

Init-Skript

Damit das erstellte Start-Skript beim Starten und das Stop-Skript beim Abschalten des Raspberry Pi ausgeführt wird, wurde während der Entwicklung des Krankenhaus-Prozesses ein Init-Skript (s. Listing 4.4) als Service erstellt. Dieses Skript besteht im Grunde aus einer Fallunterscheidung, in der überprüft wird, ob der gewünschte Service gestartet, gestoppt oder neu gestartet werden soll. Das Skript muss im Verzeichnis `/etc/init.d/` erstellt werden, um vom Betriebssystem als Service benutzt werden zu können.

```
#!/bin/bash
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:$PATH
DESC="iBeacon Application Software"
PIDFILE=/var/run/ibeacon.pid
SCRIPTNAME=/etc/init.d/ibeacon

case "$1" in
start)
        printf "%s" "start ibeacon..."
        cd /home/pi
        ./start
;;
stop)
        printf "%s" "stop ibeacon..."
        cd /home/pi
        ./stop
;;
restart)
        $0 stop
        $0 start
;;
*)
        echo "use: $0 {start|stop|restart}"
        exit 1
esac
```

Listing 4.4: /etc/init.d/ibeacon

Damit dieses Skript nun als Service eingetragen und benutzt werden kann, müssen zwei weitere Befehle durchgeführt werden. Das Skript benötigt Rechte zur Ausführung wie schon das Start-Skript und das Stop-Skript. Weiter muss das Skript als Service durch den Dienst `update-rc.d` registriert werden. Dazu muss im Befehl dem Service der Name des Skripts im `init.d`-Verzeichnis mitgeteilt werden. Der Parameter `defaults` besagt, dass das Skript beim Starten und Stoppen aufgerufen wird.

```
root@raspberrypi:~# chmod 777 /etc/init.d/ibeacon
root@raspberrypi:~# update-rc.d ibeacon defaults
```

4.3 Zusammenfassung

In diesem Kapitel wurde der Raspberry Pi vorgestellt und beschrieben, wie man ihn als einen iBeacon konfigurieren kann. Dazu wird ein Bluetooth-Treiber benötigt, der Bluetooth LE implementiert. Für die Implementierung des Krankenhaus-Prozesses wurde dazu BlueZ verwendet, da dieser sowohl weitverbreitet und bekannt ist als auch vor allem regelmäßig gewartet wird. Fernen ist ein Bluetooth-Dongle vorausgesetzt, der die Spezifikation Bluetooth 4.0 und Bluetooth LE unterstützt. Dazu wurde ein Plugable USB-BT4LE verwendet, da dieser von BlueZ empfohlen und als unterstützt gekennzeichnet ist. Mit den erstellten Skripten und Konfigurationsdateien ist es weiter möglich, den Raspberry Pi ohne weiteres Eingreifen vom Start an als iBeacon zu verwenden, sodass eine Neukonfiguration nach einem Neustart nicht notwendig ist. Die auf diese Weise erstellen iBeacons wurden während der Implementierung verwendet. Dabei traten keinerlei Fehler im Protokoll oder auf dem Raspberry Pi selbst auf, sodass diese Herangehensweise als empfehlenswert angesehen werden kann.

Bevor nun das Design und die eigentliche Implementierung des mehrfach genannten Krankenhaus-Prozesses vorgestellt wird, sollen im nächsten Kapitel einige Beispiele von Geschäftsprozessen und anderen Szenarien betrachtet werden, welche durch die Nutzung der vorgestellten iBeacons - wobei natürlich auch Raspberry Pis verwendet werden können - und dem Wissen über den Kontext und die Position von Anwendern oder anderen Gegenständen optimiert oder auch automatisiert werden können. Die Anwendungen dieser Szenarien besitzen folglich einige Eigenschaften der oben beschrieben Context Aware Applications.

5 | Geschäftsprozesse und iBeacons

Mit dem Wissen aus den vorherigen Kapiteln über Context Aware Applications und ihre Eigenschaften, verfügbare Sensoren und insbesondere die Funktionsweise und Nutzung von iBeacons, werden in diesem Kapitel einige Szenarien vorgestellt, die iBeacons - die natürlich auch wie oben vorgestellt, als Raspberry Pis implementiert sein können - und mobile Endgeräte für die Optimierung und teilweise Automatisierung von Geschäftsprozessen verwenden.

Dazu wird zunächst der schon mehrfach erwähnte Krankenhaus-Prozess vorgestellt. Es wird darauf eingegangen, welche Probleme in vielen Krankenhäusern existieren können und wie diese durch den Kontext der Position verbessert werden können. Weiter werden einige Anwendungsfälle beschrieben, die ebenfalls durch die Nutzung von mobilen Endgeräten und iBeacons umgesetzt werden können. Diese Anwendungsfälle werden die Grundlage für die Implementierung des Krankenhaus-Prozesses in späteren Kapiteln sein.

Auf die Vorstellung des Krankenhaus-Prozesses folgt ein Einblick in einen typischen Lagerhaus-Prozess. Auch hierbei werden einige Anwendungsfälle erläutert, die durch den Einsatz mobiler Geräte, dem Kontext der Position von Mitarbeitern und mobilen Endgeräten und iBeacons optimiert werden können.

Um einen weitreichenden Überblick möglicher Szenarien zu erhalten, werden ebenfalls einige sowohl bereits existierende und implementierte Szenarien vorgestellt, die iBeacons und mobile Geräte verwenden, als auch fiktive Anwendungsfälle vorgestellt. Dabei liegen die Schwerpunkte jedoch nicht auf der Optimierung und Automatisierung von Ge-

schäftsprozessen, sondern auf der Verbesserung von mobilen Anwendungen ihrem Verhalten.

5.1 Krankenhaus-Prozess

Der Krankenhaus-Prozess ist ein typischer Geschäftsprozess aus dem Gesundheitswesen in Krankenhäusern und wird die Grundlage für die Implementierung einer Context Aware Application zur Optimierung von Geschäftsprozessen durch den Einsatz von iBeacons legen. Bevor die Implementierung und das Design vorgestellt werden, soll der Krankenhaus-Prozess in diesem Kapitel erläutert werden. Eine modellierte Darstellung des Geschäftsprozesses nach dem Formalismus der *BPMN* (*Business Process Model and Notation*) ist in Abbildung 5.1 abgebildet.

5.1.1 Beschreibung

Am Krankenhaus-Prozess sind drei Organisationseinheiten innerhalb des Krankenhauses beteiligt. Dazu gehören *Stationsärzte*, *Krankenpfleger* und auf einen Fachbereich bzw. medizinischen Bereich spezialisierte *Fachärzte*. Jede dieser Organisationseinheiten besitzt verschiedene Aufgabenbereiche und Teilaufgaben, die im Folgenden beschrieben werden.

Ein Stationsarzt ist für Visiten auf seiner aktuellen Station zuständig. Stationen können räumlich getrennte Gebiete oder Stockwerke sein. Im vorliegenden Krankenhaus-Prozess sollen die Stationen aus unterschiedlichen Stockwerken bestehen. Sobald ein Stationsarzt mit der Visite beginnt, benötigt er die zum Patienten passende Patientenakte. Anhand der Patientenakte und der darin enthaltenen Informationen über die Historie von Behandlungen, Medikationen oder Untersuchungen, untersucht der Stationsarzt den Patienten. Nach der Untersuchung kann er in diesem Geschäftsprozess entweder nichts weiteres unternehmen, Medikationen mit verschiedenen Medikamenten verordnen oder die Patienten von Fachärzten behandeln lassen. Den Ablauf einer solchen Visite unternimmt der Stationsarzt für jeden Patienten auf seiner aktuell zugeteilten Station.

Ein Krankenpfleger ist ebenfalls einer expliziten Station zugeteilt und kann vom Stationsarzt für einen Patienten aufgegebene Behandlungen und Medikationen durchführen. Da ein Krankenpfleger auf einer bestimmten Station arbeitet, soll er nur Behandlungen und Medikationen an Patienten durchführen, die sich momentan auf derselben Station befinden. Je

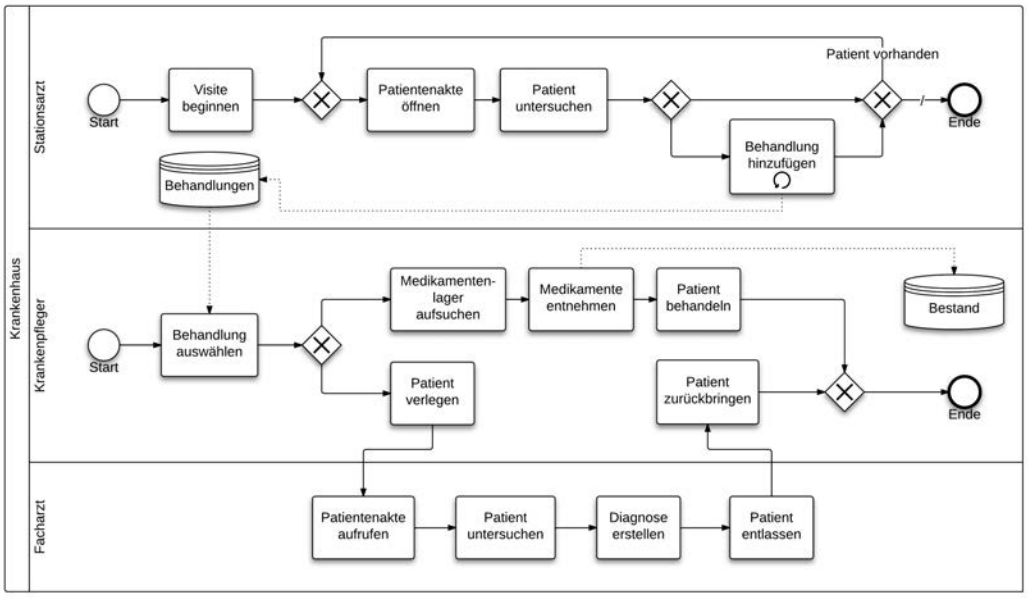

Abbildung 5.1: Modellierung des Krankenhaus-Prozesses nach BPMN

nach dem, für welche Art von Aufgabe sich der Krankenpfleger entscheidet, läuft der Geschäftsprozess unterschiedlich ab. Bei einer Medikation sucht der Krankenpfleger ein Medikamentenlager auf und entnimmt diesem die in der Medikation aufgelisteten Medikamente. Mit den entnommenen Medikamenten wird der passende Patient darauffolgend medikatös behandelt und die Aufgabe durch einen Eintrag in der Patientenakte als durchgeführt markiert. Bei einer fachärztlichen Untersuchung ist der Krankenpfleger dafür zuständig, den Patienten auf eine andere Station zum Facharzt zu verlegen und nach der durchgeführten Behandlung wieder zurück in sein Krankenzimmer zu bringen. Hierbei muss in der Patientenakte vermerkt werden, dass sich der Patient zum Zeitpunkt der Behandlung durch den Facharzt nicht in seinem Krankenzimmer aufhält, sonder sich bei einem bestimmten Facharzt auf einer bestimmten Station befindet.

Ein Facharzt ist für die Durchführung von fachärztlichen Behandlungen zuständig. Er empfängt die von einem Pfleger zur Behandlung verlegten Patienten und untersucht diese der Reihe nach. Sobald er die fachärztliche Untersuchung abgeschlossen hat, erstellt der Facharzt eine Diagnose und meldet der Station des Patienten, dass die Behandlung abgeschlossen wurde und der Patient zurück in sein Krankenzimmer verlegt werden kann.

5.1.2 Probleme und Verbesserungen

Der so beschriebene Geschäftsprozess besitzt einige Anfälligkeiten für Fehler und Unstimmigkeiten. Weiter können Zeitverluste durch das Heraussuchen von auf Papier basierenden Dokumenten und der Patientenakte auftreten. Um den Geschäftsprozess mithilfe der oben vorgestellten Konzepte von iBeacons, Context Aware Applications und mobilen Endgeräten zu optimieren und automatisieren, wurden die folgenden Anwendungsfälle herausgearbeitet und später implementiert.

Visite

Die Visite im vorliegenden Geschäftsprozess verursacht das Problem, dass der Stationsarzt die zum Patienten passende Patientenakte suchen muss. Um eine Visite bei einem Patienten durchführen zu können, muss der Patient in seinem Krankenzimmer anwesend sein. Befindet er sich z.B. bei einer fachärztlichen Untersuchung, so ist er nicht anwesend und die Visite kann hier nicht durchgeführt werden. Diese Information erhält der Stationsarzt jedoch erst bei einem Blick auf die zuvor gesuchte Patientenakte.

Um diese Probleme zu beseitigen, kann die Nutzung von mobilen Geräten, mobilen Anwendungen und iBeacons zur Positionsbestimmung des Stationsarztes und zur Identifizierung von Patienten verwendet werden. Dazu erhält jeder Patient einen eindeutigen Wert durch Major- und Minor-Werte der iBeacons zugewiesen. Wie sich diese Werte im Detail zusammensetzen, wird im Kapitel über das Design der Implementierung aufgezeigt. Weiter wird jedes Krankenzimmer mit einem iBeacon ausgerüstet, welcher ebenfalls eindeutig identifizierbar ist. Aufgrund der auf der Station befindlichen iBeacons an Krankenzimmer und derer von Patienten, ist es dem Stationsarzt nun möglich, über sein mobiles Gerät und der darauf installierten Anwendung herauszufinden, auf welcher Station er sich gerade befindet. Hierzu wird der iBeacon herangezogen, der räumlich am nächsten zum Stationsarzt gelegen ist. Dies kann leicht über das vorgestellte Konzept der Kalkulation der Entfernung über den RSSI-Wert durchgeführt werden. Anhand der eindeutigen Identifizierung des iBeacons kann nun die aktuelle Station herausgefunden und aufgrund dessen können dementsprechend alle Patienten aufgelistet werden, die ihr Krankenzimmer auf dieser Station haben. Durch dieses Vorgehen muss der Stationsarzt nicht mehr zuerst die Patientenakte eines Patienten finden, um über dessen An- oder Abwesenheit informiert zu werden, sondern sieht auf einen Blick welche Patienten auf der aktuellen Station anwe-

send sind. Über direkten Kontakt mit einem iBeacon an einem Krankenzimmer, also einer NFC- oder RFID- ähnlichen Interaktion, kann die Auflistung der Patienten weiter verfeinert werden. Dadurch kann der Stationsarzt schon vor dem Betreten des Krankenzimmers Informationen über die darin liegenden Patienten abfragen. Da jeder Patient ebenfalls einen eindeutig identifizierbaren iBeacon besitzt, kann der sich Stationsarzt weiter durch direkten Kontakt mit einem solchen iBeacon die Patientenakte des korrespondierenden Patienten anzeigen lassen.

Durch diese Nutzung von iBeacons in Verbindung mit einer auf einem mobilen Gerät befindlichen Patientenakte, entfällt das Aufsuchen von Patientenakten und weiter die Informationssuche über die An- oder Abwesenheit von Patienten. Über die zuvor beschriebene Liste von Patienten auf der aktuellen Station kann ebenfalls eine digitale und mobile Patientenakte abgerufen werden, auch ohne direkten Kontakt mit dem iBeacon eines Patienten suchen zu müssen. Der direkte Kontakt jedoch kann eventuelle Verwechslungen der Patientenakten und Patienten durch eindeutige Identifikation und Kopplung beider verhindern. Die vom Stationsarzt definierten Behandlungen und Medikationen werden weiter in einer Datenbank verwaltet, auf die andere Organisationseinheiten wie z.B. Krankenpfleger zugreifen können. Damit werden die Daten zentral gespeichert.

Im späteren Verlauf dieser Arbeit wird der Schwerpunkt auf dem Umgang und der Anbindung von iBeacons liegen und nicht auf der Implementierung einer mobilen Patientenakte. Das Design und eine Implementierung einer solchen Patientenakte kann in [30, 56, 61] weiter nachgelesen werden.

Aufgabenübersicht

Die von einem Stationsarzt definierten fachärztlichen Untersuchungen und Medikationen werden zentral in einer Datenbank gelagert. Krankenpfleger sind nur für Aufgaben und Patienten auf ihrer aktuellen Station zuständig. Anstatt nun die einzelnen Patientenakten nach Aufgaben zu durchsuchen, können diese durch die Nutzung von iBeacons leichter eingesehen werden. Durch die eingesetzten iBeacons an Krankenzimmern und Patienten kann die aktuelle Station des Krankenpflegers durch Positionsbestimmung wie im Anwendungsfall der Visite herausgefunden werden. Durch die dadurch bestimmte aktuelle Station und die Hinterlegung von Aufgaben in einer zentralen Datenbank, werden die vom Krankenpfleger begonnen aber noch nicht abgeschlossenen und offenen Aufgaben angezeigt. Durch die

Vermeidung des Auffindens von Patientenakten können Aufgaben effizienter und schneller gefunden und ausgeführt werden.

Medikation

Beim Ablauf der Medikation, die von einem Krankenpfleger ausgeführt wird, tritt ein sehr großes Problem in Krankenhäusern auf: die fehlende Dokumentation des Medikamentenbestands in Medikamentenlagern. Entnimmt ein Krankenpfleger die notwendigen Medikamente aus dem Lager, muss dieser - sofern die Möglichkeit überhaupt besteht - die Anzahl der entnommenen Medikamente papierbasiert zu hinterlegen.

Die Problematik der fehlenden Dokumentation des Medikamentenbestands in Medikamentenlagern lässt sich durch das Anbringen von eindeutig zugeordneten iBeacons an besagten Medikamentenlagern beseitigen und automatisieren. Hierzu ist weiter notwendig, dass die Bestände der einzelnen Lager zentral in einer Datenbank hinterlegt werden. Sobald ein Krankenpfleger nun Medikamente aus einem bestimmten Lager entnimmt, tut er dies mit mit der geöffneten Aufgabe der Medikation in seiner mobilen Anwendung auf einem mobilen Endgerät. Die Aufgabe enthält die Anzahl der zu verabreichenden Medikamente. Um die Entnahme der Medikamente aus dem Medikamentenlager abzuschließen, führt der Krankenpfleger mit geöffneter Aufgabe eine direkte Kontaktaufnahme mit dem iBeacon des Medikamentenlagers durch. Durch den Kontakt wird in der zentralen Datenbank der Bestand anhand der in der Aufgabe der Medikation gelisteten Medikamente automatisch aktualisiert. Da Medikamente nur auf eine solche Weise aus Medikamentenlagern entnommen werden dürfen, wird der Bestand jedes mal für das jeweilige Lager aktualisiert. Dadurch ist der Bestand von Medikamenten jeder Zeit dokumentiert. Fernen können Bestellungen von neuen Medikamenten *just in time* durchgeführt werden.

Fachärztliche Untersuchung

Eine fachärztliche Untersuchung wird initial von einem Krankenpfleger gestartet, indem der Patient hin zum Facharzt verlegt wird. Im dargestellten Geschäftsprozess wird an keiner Stelle vermerkt, ob sich der Patient in seinem Krankenzimmer oder z.B. momentan in einer fachärztlichen Untersuchung auf einer anderen Station befindet. Weiter können Wartezimmer von Fachärzten überfüllt sein, wodurch dem Facharzt leicht der Überblick über anwesende oder noch nicht eingetroffene Patienten verloren gehen kann.

Um diesen Problemen entgegenzusetzen und den Ablauf zu optimieren, werden zunächst zusätzliche iBeacons für Fachärzte hinzugefügt. So erhält jeder Facharzt an seinem Behandlungszimmer bzw. Büro einen eigenen iBeacon, der mit Zimmer- und Stationsnummer verknüpft wird.

Möchte ein Krankenpfleger einen Patienten wegen einer fachärztlichen Untersuchung verlegen, so sucht dieser mit der geöffneten Aufgabe den direkten Kontakt zum iBeacon des Patienten. Dadurch wird die Aufgabe gestartet und der Patient als nicht anwesend markiert. Der Stationsarzt weiß dadurch, dass der Patient z.B. während der Visite momentan nicht zur Verfügung steht. Sobald der Patient beim Facharzt angekommen ist, wird der iBeacon am Behandlungszimmer des Facharztes mit der geöffneten Aufgabe durch den Krankenpfleger berührt. Dadurch wird in der Patientenakte hinterlegt, dass sich der Patient auf der Station und im Behandlungszimmer der Facharztes befindet. Ferner erhält der Facharzt eine neue Aufgabe für eine fachärztliche Untersuchung, da der Patient sich nun bei ihm befindet. Nachdem die Behandlung durch den Facharzt durchgeführt wurde, markiert er die Aufgabe als erledigt, wodurch Krankenpfleger die Aufgabe erhalten, den Patienten vom Facharzt zurück in dessen Krankenzimmer zu verlegen. Dabei wird auf die gleiche Art und Weise vorgegangen wie bei der Verlegung hin zum Facharzt. Beim Starten der Abholung wird der iBeacon des Patienten berührt, wodurch die Aufgabe gestartet wird. Zum Abschließen der Aufgabe wird ebenfalls mit geöffneter Aufgabe der iBeacon des Krankenzimmers des Patienten berührt, damit die Aufgabe abgeschlossen wird und in der Patientenakte die Anwesenheit des Patienten in seinem Krankenzimmer dokumentiert wird.

Starten und Stoppen von Aufgaben

Im vorherigen Kapitel über die Aufgabenübersicht wurde erläutert, dass Krankenpfleger eine Liste von Aufgaben passend zu ihrer aktuellen Station in ihrer mobilen Anwendung angezeigt bekommen. Um diese Aufgabenübersicht weiter zu verbessern, wird der Status der Aufgaben vermerkt und dokumentiert. Dabei werden aber je nach angemeldetem Krankenpfleger nur die offenen oder von diesem bereits gestartete aber noch nicht abgeschlossene Aufgaben angezeigt. Die Liste wird nun dadurch verkürzt, dass von anderen Krankenpflegern in Ausführung befindliche Aufgaben ausgeblendet werden und dadurch nicht doppelt ausgeführt werden können.

Das Starten und Stoppen und damit das Vermerken des Status von Aufgaben wird durch den direkten Kontakt mit iBeacons durchgeführt. So muss eine Verlegung eines Patienten

damit beginnen, dass dessen iBeacon mit der geöffneten Aufgabe berührt wird. Dabei wird überprüft, ob der zu verlegende Patient der richtige ist, indem der iBeacon und dessen gekoppelte Informationen des Patienten mit dem Patienten in der Aufgabe verglichen werden. Dadurch wird sichergestellt, dass der richtige Patient verlegt oder mit Medikamenten versorgt wird und nur bei korrekter Übereinstimmung die Aufgabe gestartet oder gestoppt werden kann.

Bevor in den folgenden Kapiteln das konkrete Design und die Implementierung des durch mobile Endgeräte und iBeacons optimierten und automatisierten Krankenhaus-Prozesses vorgestellt wird, sollen noch weitere Szenarien dargestellt werden, die durch mobile Sensorik verbessert werden können.

5.2 Lagerhaus-Prozess

Der Lagerhaus-Prozess, der im folgenden beschrieben wird, ist ebenfalls ein typischer Geschäftsprozess, der in Lagerhäusern vorkommt. Auch dieser lässt sich durch geeigneten Einsatz von iBeacons und mobilen Endgeräten optimieren. Der Prozess ist in BPMN in Abbildung 5.2 abgebildet. Im Folgenden soll auch dieser kurz beschrieben und dessen Optimierungspotential erläutert werden.

5.2.1 Beschreibung

Der Geschäftsprozess des Lagerhauses wird durch drei Organisationseinheiten ausgeführt und lässt sich wie folgt beschreiben.

Ein *Vorarbeiter* erstellt neue Aufträge mit der Aufgabe, dass Artikel von einem *Lagerarbeiter* aus einem bestimmtem Regal entnommen und zu einem *Verarbeiter* geliefert werden sollen, die letzterer für seine weiteren Arbeiten benötigt. Die Aufträge werden vom Vorarbeiter in einem Computersystem erstellt und meist ausgedruckt. Ein Lagerarbeiter stellt sich für eine dieser Aufgaben zur Verfügung und holt den Auftrag beim Vorarbeiter ab. Nachdem dieser den Auftrag abgeholt hat, macht er sich auf den Weg in das angegebene Lager. Sobald dieser das richtige Lager aufgefunden hat, macht sich der Lagerarbeiter auf die Suche nach dem im Auftrag vermerkten Regal und sucht dort wiederum nach dem angegeben Artikel. Nach der Entnahme des Artikels macht sich der Lagerarbeiter auf den Weg

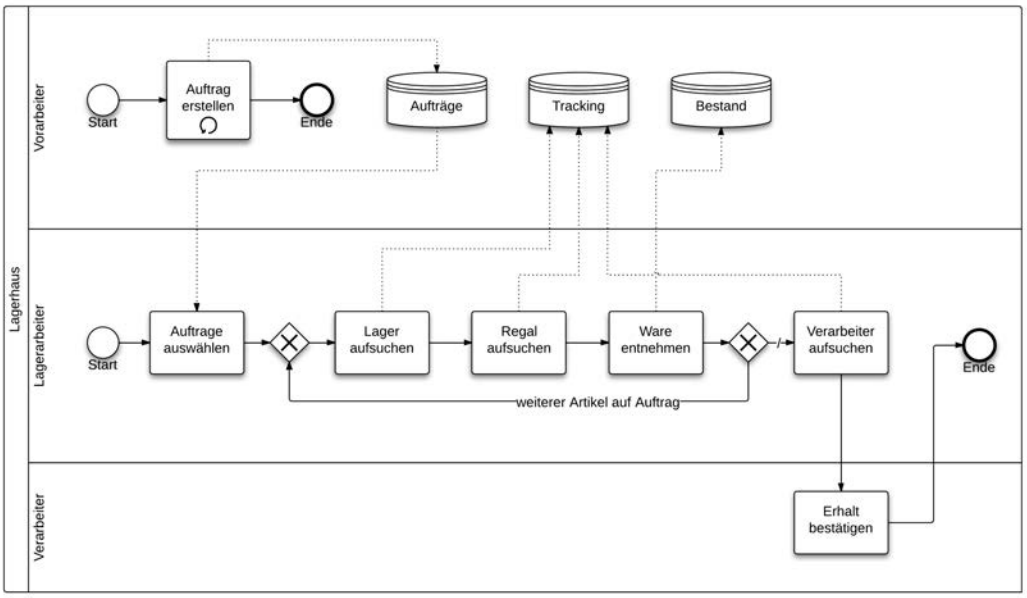

Abbildung 5.2: Modellierung des Lagerhaus-Prozesses nach BPMN

zu dem auf dem Auftrag angegebenen Verarbeiter oder sucht nach weiteren auf dem Auftrag vorhandenen Artikeln. Wenn der Lagerarbeiter alle Artikel zusammengesammelt hat, begibt er sich zum Verarbeiter und übergibt ihm die Artikel zur weiterer Verarbeitung. Der Verarbeiter bestätigt auf dem Auftrag den Erhalt der Artikel, womit der Prozess beendet ist und der Lagerarbeiter sich einen neuen Auftrag zur Bearbeitung beim Vorarbeiter abholt.

5.2.2 Probleme und Verbesserungen

Auch in diesem Geschäftsprozess können Verbesserungen durch den Einsatz mobiler Endgeräte, mobiler Anwendungen und der Nutzung von iBeacons erreicht werden. Diese Verbesserungen werden im Folgenden vorgestellt.

Mobile Aufträge

Der Vorarbeiter erstellt Aufträge an einem Computer und druckt diese nachträglich zur Bearbeitung durch einen Lagerarbeiter aus. Um die Aufträge schneller erstellen und für Lagerarbeiter verfügbar zu machen, kann eine mobile Anwendung erstellt werden, welche auf einem mobilen Endgerät eines Lagerarbeiters ausgeführt wird. Der Lagerarbeiter kann

damit verfügbare Aufträge über eine Datenbank abfragen und von seiner momentanen Position aus annehmen, ohne dabei direkten Kontakt zum Vorarbeiter suchen zu müssen. Die mobile Anwendung ist außerdem so konfiguriert, dass sie Daten von iBeacons empfangen und damit für weiteren Optimierungen verwendet werden kann.

Indoor-Navigation

Falls bestimmte Artikel nicht immer an der selben Stelle gelagert werden oder neue Mitarbeiter die Lager und die Aufbewahrungsorte von Artikeln noch nicht kennen, kann eine Indoor-Navigation durch den Einsatz von iBeacons zur Verfügung gestellt werden. Hierzu werden je nach Granularität der Navigation iBeacons für ganze Lager, einzelne Regale oder gar einzelne Artikel in Regalen erstellt und konfiguriert. Die iBeacons müssen so konfiguriert werden, dass sie erstens eine fixe Position erhalten und zweitens eindeutig mit einem Lager, einem Regal oder einem Artikel in Verbindung gebracht werden können. Durch dieses Vorgehen kann anhand einer fiktiven und auf den Anwendungsfall angepassten Karte und durch den Einsatz mobiler Endgeräte mit Triangulation von Signalstärken je nach Granularität der aufgestellten iBeacons die Position bestimmt und eine Navigation zum gewünschten Artikel durchgeführt werden. Neben der Triangulation können auch die anderen in Kapitel 2 vorgestellten Algorithmen zur Indoor-Navigation bzw. der Positionsbestimmung innerhalb von Gebäuden herangezogen werden.

Nachverfolgen und spontane Aufträge

Mit den aufgestellten und konfigurierten iBeacons, die zur Optimierung des Geschäftsprozesses durch Indoor-Navigation herangezogen wurden, können außerdem die Positionen der Lagerarbeiter nachverfolgt werden. Die mobile Anwendung auf dem mobilen Endgerät bestimmt kontinuierlich die Position der Lagerarbeiters über die gleichen Algorithmen, die während der Indoor-Navigation angewendet werden können. Die Position wird in einer Datenbank korrespondierend zum Lagerarbeiter dokumentiert. Falls ein Vorarbeiter nun einen dringenden Auftrag zur Verarbeitung erhalten oder erstellt hat, ist es diesem möglich, anhand der Positionen von Lagerarbeitern in der Datenbank einen geeigneten für die Bearbeitung auszuwählen. Ein geeigneter Lagerarbeiter kann ein solcher sein, der sich aktuell sehr nah am dringend benötigen Artikel befindet oder aber erst vor kurzem einen neuen Auftrag angenommen hat und womöglich noch nicht allzu viele Artikel zusammengebracht

hat. An dieser Stelle soll erwähnt werden, dass ein solches Vorgehen zum Nachverfolgen und Bestimmen der Position von Mitarbeitern natürlich mit Arbeitsrecht und Datenschutz in Einklang gebracht werden muss, da eine dauerhafte Nachverfolgung von Mitarbeitern das Persönlichkeitsrecht verletzen könnte (übermäßig viele Toilettenbesuche, Pausen, etc.).

Dokumentation des Bestands

Ähnlich wie im Krankenhaus-Prozess können auch im Lagerhaus-Prozess Automatismen durch die Nutzung von iBeacons und mobilen Endgeräten erzeugt werden. Hierzu können die iBeacons an Artikeln angebracht werden. Nach der Entnahme berührt der Lagerarbeiter den iBeacon mit dessen mobilen Endgerät und dem geöffneten Auftrag. Da die iBeacons so konfiguriert sind, dass sie einen bestimmten Artikel in einem bestimmten Regal und Lager identifizieren, können die Bestände in diesem Regal in einer Datenbank nach Entnahme automatisch aktualisiert werden, womit auch in diesem Geschäftsprozess Nachbestellungen von Artikeln just in time durchgeführt werden können. Ferner ist es möglich, die Lagerarbeiter durch die vorgestellte Indoor-Navigation zu einem anderen Regal zu lotsen, falls die Artikel in mehreren Regalen vorhanden sind, aber eines oder mehrere dieser Regale bereits keine Artikel vorrätig haben.

Identifikation bei Übergabe

Möchte ein Lagerarbeiter seinen Auftrag abschließen und die Artikel an einen Verarbeiter übergeben, so kann dieser Prozessschritt ebenfalls durch iBeacons optimiert werden. Dazu besitzt der Verarbeiter einen kleinen iBeacon, den er z.B. in der Hosentasche tragen kann und so konfiguriert ist, dass er den Verarbeiter eindeutig identifiziert. Der Lagerarbeiter kann seinen Auftrag dann nur abschließen, wenn er direkten Kontakt zum iBeacon des Verarbeiters herstellt und letzterer mit dem im Auftrag vermerkten Verarbeiter übereinstimmt. Durch den so durchgeführten Abschluss kann ferner der Status des Auftrags sofort und automatisiert in der Datenbank vermerkt werden. Dadurch werden wiederum womöglich lange Wege zurück zum Vorarbeiter und der direkte Kontakt vermieden, wodurch Zeit eingespart und effizienter gearbeitet werden kann.

5.3 Mobiles Bezahlen und Bestellungen

Die beiden zuvor beschriebenen Geschäftsprozesse kommen so tagtäglich in den unterschiedlichen Geschäftsdomänen vor. In diesem Kapitel wird nun ein fiktives Szenario zur Bestellung von Waren, dem anschließenden mobilen Bezahlen und der Abholung aus einem Lager für ein Warenhaus erläutert. Das Szenario ist in Abbildung 5.3 schematisch abgebildet. Auch hierbei werden mobile Endgeräte, mobile Anwendungen und iBeacons zur Realisierung verwendet.

Das Szenario besteht aus einem Warenhaus mit ausgestellten Artikeln, einer Station für das Bezahlen und einem Gebäude zur Abholung eventuell gekaufter Artikel. Innerhalb des Warenhauses werden zwei unterschiedliche Arten von iBeacons platziert, welche aber die gleichen UUIDs besitzen, um damit das Warenhaus zu identifizieren. Rot dargestellt sind iBeacons, die ein ausgestelltes Produkt und dessen Preis repräsentieren. Grün abgebildet ist ein iBeacon an der Station für die mobile Bezahlung. Zusätzlich wird in diesem Szenario eine mobile Anwendung vorausgesetzt, die mit den aufgestellten iBeacons umgehen, einen Warenkorb aufbauen und zur mobilen Bezahlung verwendet werden kann.

Sobald ein potentieller Kunde mit der installierten mobilen Anwendung in die Nähe des Warenhauses kommt, kann dieser über die im Inneren aufgestellten iBeacons über mögliche Angebote oder aktuelle Produkte über sein mobiles Endgerät informiert werden. Dazu empfängt die mobile Anwendung das Signal eines oder mehrerer iBeacons mit der passenden UUID aus dem inneren des Warenhauses und überprüft anhand der Berechnung der Entfernung, ob sich der potentielle Kunde noch außerhalb des Warenhauses befindet (z.B. über die Konstante *Far* des iBeacon-Frameworks von Apple). Die beschriebe Notifikation wird aber nur dann auf dem mobilen Endgerät des Kunden aufgerufen, wenn er sich außerhalb des Warenhauses befindet.

Betritt der Kunde das Warenhaus und möchte bestimmte ausgestellte Produkte kaufen, so berührt er mit seinem mobilen Endgerät und der geöffneten mobilen Anwendung den jeweiligen iBeacon. Dadurch wird das Produkt in einer benutzerdefinierten Ausführung, wie z.B. der Schuhgröße, in den mobilen Warenkorb gelegt. Läuft der Kunde weiter durch den Laden, könnte der Benutzer weitere Notifikationen über Angebote zu nahe gelegenen Artikeln erhalten (z.B. über die Konstante *Near* des iBeacon-Frameworks). Der Kunde kann seinen Einkauf abschließen, indem er den iBeacon an der Station für mobiles Bezahlen berührt. Dieser Schritt ähnelt sehr dem bekannten mobilen Bezahlen über NFC und mobile Endge-

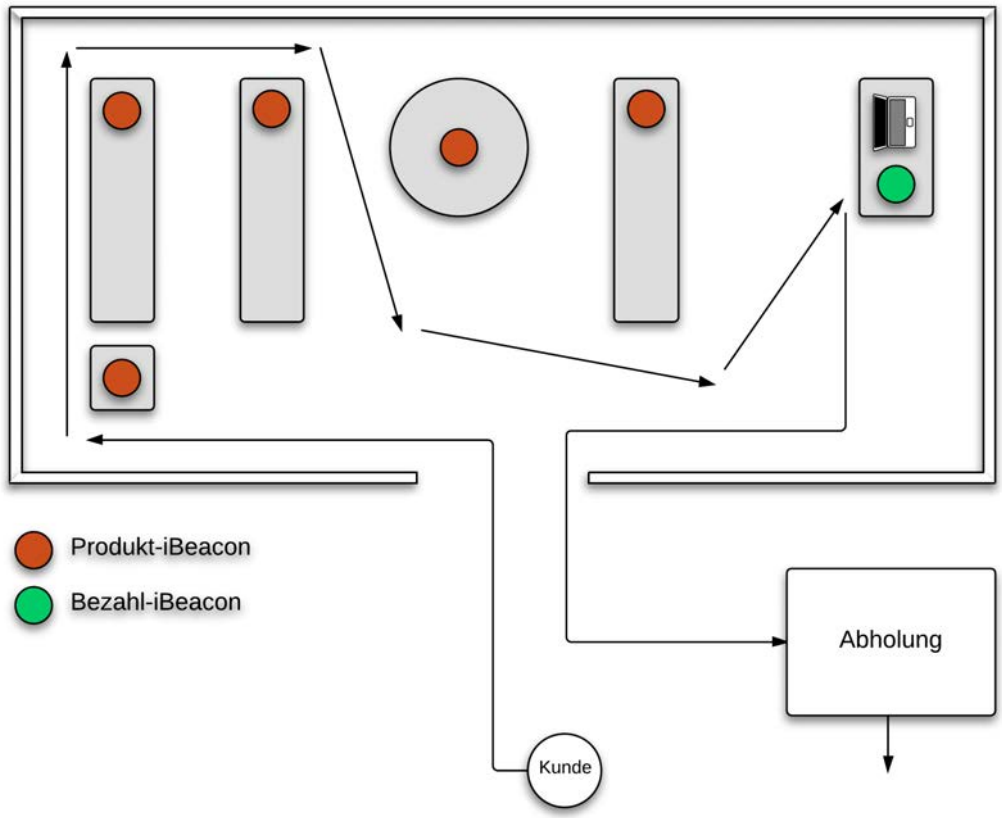

Abbildung 5.3: Mobiles Bezahlen und Bestellungen mit iBeacons und mobilen Endgeräten

räte. Die Bezahlung der Artikel aus dem Warenkorb wird mit den in der Anwendung hinterlegten Informationen zur Bezahlung durchgeführt. Dazu könnte eine Kreditkartennummer oder ein PayPal-Konto hinterlegt sein. Sobald der Kunde seinen Warenkorb bezahlt hat, werden die gewünschten Artikel im Lager gesammelt und zur Abholung bereit gelegt.

Dieses Szenario hat durch die Nutzung von Context Awareness der mobilen Anwendung und dem Gebrauch von iBeacons mehrere Vorteile. Der potentielle Kunde kann bereits während des Vorbeilaufens am Warenhaus mögliche tagesaktuelle Angebote einsehen, ohne den Laden betreten zu müssen. Weiter muss er ausgewählte Artikel nicht durch das Warenhaus tragen, sondern kann die bezahlten Artikel gesammelt an einer Stelle abholen. Bargeld zur Bezahlung gewünschter Artikel ist ebenfalls nicht notwendig, da die Bezahlung über das mobile Endgerät durchgeführt wird. Für das Warenhaus selbst besteht der Vorteil, dass Artikel nicht in unterschiedlichen Varianten ausgestellt werden müssen, da diese vom

Kunden über die mobile Anwendung personalisiert werden können (z.B. die Schuhgröße). Dadurch ist eine kleinere Ladenfläche zur Präsentation von Artikeln ausreichend. Weiter entfällt für das Warenhaus das direkt angeschlossene Lager, da der Kunde außerhalb des Warenhauses, womöglich auch weiter entfernt vom Warenhaus, seine Artikel gesammelt und personalisiert abholen kann.

5.4 Weitere Szenarien

Neben den vorgestellten Geschäftsprozessen und Szenarien gibt es noch viele andere Möglichkeiten die Vorteile von iBeacons und Context Aware Applications zu nutzen. So wurden iBeacons auf der *CeBIT 2014* verwendet, um an Ständen mit Jobangeboten in der Nähe befindlichen Nutzern von der passenden mobilen Anwendung diese Jobangebote als Notifikation zukommen zu lassen [6]. Die britische Fluggesellschaft *EasyJet* verwendet iBeacons in ihrer Anwendung, um Passagieren an Flughäfen in der Nähe von Zoll- und Passkontrollen nötige Informationen auf ihre mobilen Endgeräte zukommen zu lassen [28]. Die Supermarktketten *Carrefour* und *Nisa* nutzen iBeacons für *Business Intelligence* in Einkaufswägen, indem sie die Laufwege und Aufenthaltsdauern an Ständen und Artikeln von Kunden nachverfolgen [18]. Dadurch ist es den Ketten möglich z.B. das *Product Placement* in ihren Filialen zu verbessern. Apple verwendet iBeacons in *Apple Stores*, um ähnliche Szenarien zu unterstützen, die in Kapitel 5.3 vorgestellt wurden [36].

Weitere Anwendungsfälle wären das automatische Einschalten eines Weckers auf einem mobilen Endgerät, sobald es nahe an einen iBeacon auf einem Nachttisch abgelegt wird. Dazu könnten als weitere Kontextinformationen die aktuelle Tageszeit und der Wochentag hinzugezogen werden, um je nach Kontext eine andere Weckzeit einzustellen. Auch *Home Automation* kann durch iBeacons erreicht werden. So könnte sich eine Alarmanlage einschalten, wenn das mobile Geräte eine gewisse Entfernung zu einem iBeacon hat oder aus dessen Reichweite verschwindet. Sobald das mobile Gerät wieder in der Nähe des iBeacons befindlich ist, kann sich die Alarmanlage automatisch wieder ausschalten. Weiter kann sich ein Garagentor automatisch öffnen, wenn sich das Automobil dem iBeacon des Garagentors nähert. Ein Schlüsselbund könnte mit einem kleinen iBeacon verbunden und mit einer Anwendung gekoppelt werden, sodass die Anwendung auf dem mobilen Gerät anfängt zu klingen falls der Schlüsselbund verloren wurde und sich die Distanz zwischen iBeacon und mobilem Gerät vergrößert.

Wie in diesem Kapitel dargestellt wurde, existieren sehr viele Geschäftsprozesse, Szenarien und Anwendungsfälle, die durch die Nutzung von iBeacons und Context Aware Application optimiert und automatisiert werden können. Da die Technik der iBeacons noch relativ neu ist, sich aber bis zum Zeitpunkt des Verfassens dieser Arbeit sehr rasant verbreitete und immer bekannter wurde, werden auch in Zukunft immer mehr Ideen konzipiert und realisiert werden, die sowohl das tägliche Leben als auch Geschäftsprozesse erleichtern.

Um nun einen tiefen Einblick in die Implementierung einer solchen mobilen Context Aware Application zur Optimierung und Automatisierung von Geschäftsprozessen mithilfe von iBeacons zu erlangen, wird in den folgenden Kapiteln die Anforderungsanalyse, der Entwurf und die Implementierung des weiter oben vorgestellten Krankenhaus-Prozesses vorgestellt. Dabei wird auf alle kritischen Systemteile eingegangen, die für die Realisierung des Szenarios benötigt werden.

6 | Anforderungsanalyse

In diesem Kapitel wird die Anforderungsanalyse für die Context Aware Application vorgestellt, die mithilfe von iBeacons und der Kontextbestimmung den oben vorgestellten Krankenhaus-Prozess optimieren und automatisieren soll. Diese Anwendung wird den Namen *Medical Beacons* erhalten und wird neben der mobilen Anwendung selbst, aus einem REST-Service, einer Datenbank und den bereits oben vorgestellten Raspberry Pis als iBeacons bestehen.

6.1 Anforderungen an Medical Beacons

Medical Beacons soll aus einem mobilen Client bestehen, der auf einem mobilen Endgerät ausgeführt werden kann und die Kommunikation über Bluetooth LE mit iBeacons beherrscht. Das zu entwickelnde Endprodukt soll die in Kapitel 5.1 erläuterten Probleme durch die Nutzung der vorgestellten Konzepte und Methoden beseitigen. Um Medical Beacons implementieren zu können, werden zwei Arten von iBeacons verwendet. Die erste Art von iBeacons ist für die Identifikation von Patienten zuständig. Dafür erhält jeder Patient des Krankenhauses einen eigenen eindeutig identifizierbaren iBeacon. Die zweite Art wird für die Identifikation von Zimmern herangezogen. So erhält jedes Krankenzimmer, jedes Medikamentenlager und jede Praxis von Fachärzten einen eindeutigen iBeacon zugeteilt.

Medical Beacons soll drei Bereiche für drei Organisationseinheiten mit unterschiedlichen Aufgabenbereichen und Berechtigungen bereitstellen (s. Abbildung 6.1). Dazu soll sich ein Anwender zu Beginn am System anmelden können. Anhand der eingegeben Informationen wird einer der drei Bereiche innerhalb der Anwendung bereitgestellt. Die drei Organisationseinheiten und deren Bereiche lassen sich in Stationsärzte, Krankenpfleger und

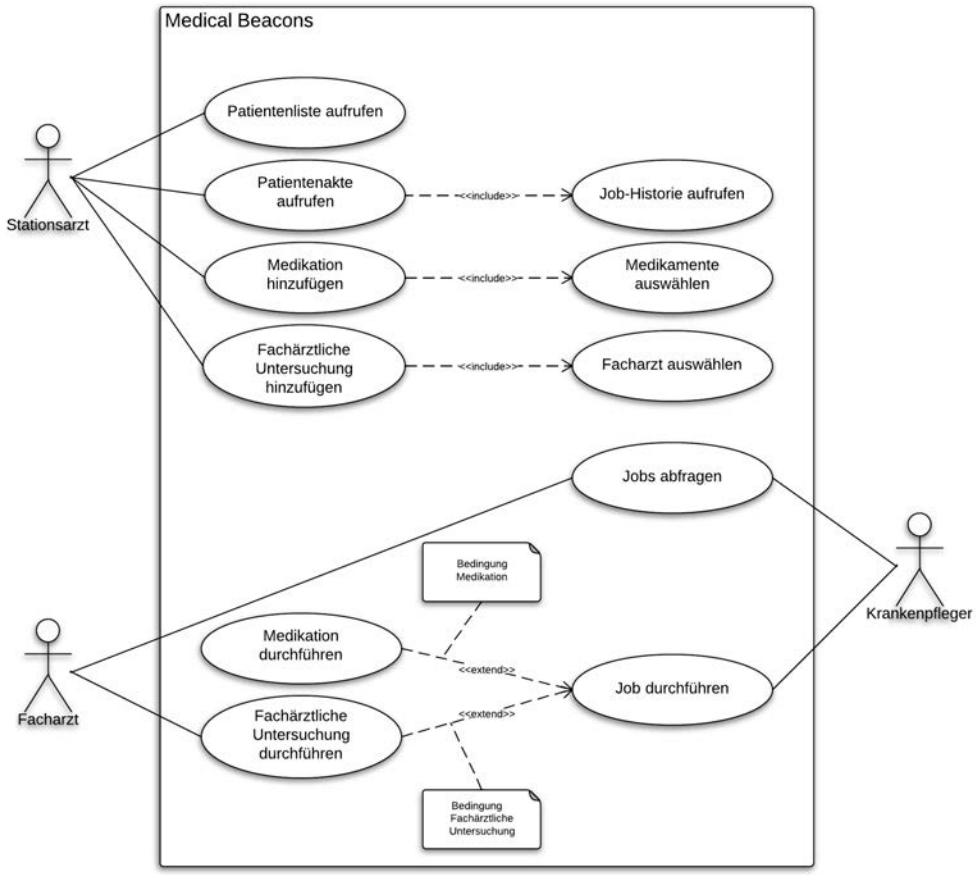

Abbildung 6.1: Anwendungsfalldiagramm des Krankenhaus-Prozesses

Fachärzte einteilen. Jede dieser Organisationseinheiten stellt unterschiedliche Anforderungen an Medical Beacons.

6.1.1 Stationsarzt

Einem Stationsarzt soll es möglich sein, eine Patientenübersicht der Patienten auf der aktuellen Station einzusehen. Dabei soll die aktuelle Station über die Bestimmung der Position des Stationsarztes über iBeacons ermittelt werden und sich bei Bewegung und Eintritt in eine andere Station automatisch aktualisieren. Ein Stationsarzt soll in der Lage sein, über die Patientenliste einen expliziten Patienten auszuwählen und dessen Patientenakte einzusehen. Die Patientenakte soll hierbei Informationen über den Patienten bereitstellen und die

bisher durchgeführten und verordneten fachärztlichen Untersuchungen und Medikationen darstellen. Ein Stationsarzt soll über diese Patientenakte weitere fachärztliche Untersuchungen und Medikationen hinzufügen können, die von Krankenpflegern und Fachärzten ausgeführt werden sollen. Beim Hinzufügen einer Medikation sollen durch den Stationsarzt Medikamente und bei der Erstellung einer fachärztlichen Untersuchung ein Facharzt ausgewählt werden können. Um dem Stationsarzt eine bessere Übersicht über die Patienten auf der aktuellen Station zu gewähren und damit die Visite besser planbar zu machen, soll die Patientenliste einen Hinweis über die An- oder Abwesenheit von Patienten geben. Die Patientenliste soll sich über den direkten Kontakt mit einem iBeacon an einem Krankenzimmer soweit verfeinern lassen, dass nur Patienten innerhalb des Zimmers angezeigt werden. Neben der Expliziten Auswahl eines Patienten über die Patientenliste, um dessen Patientenakte einzusehen, soll die Patientenakte durch direkten Kontakt mit dem iBeacon eines Patienten automatisch aufgerufen werden können.

6.1.2 Krankenpfleger

Einem Krankenpfleger soll es durch Medical Beacons möglich sein, anhand der über iBeacons bestimmten Station, eine Aufgabenübersicht als Liste zu erhalten. In dieser Übersicht sollen alle offenen und vom angemeldeten Krankenpfleger bereits gestarteten Aufgaben angezeigt werden. Die Aufgaben lassen sich durch Medikationen und Hin- und Rückverlegungen für fachärztliche Untersuchungen unterscheiden. Wenn der Krankenpfleger eine Medikation auswählt und diese noch nicht gestartet wurde, so soll es ihm möglich sein, die Aufgabe zu starten. Die Medikation soll durch die Entnahme der in der Beschreibung genannten Medikamente aus einem Medikamentenlager und dem direkten Kontakt mit dem iBeacon am Medikamentenlager und der in der Anwendung geöffneten Aufgabe abgeschlossen werden. Wählt der Krankenpfleger eine Verlegung zu einem Facharzt aus, so soll er die Aufgabe nur starten können, indem er den iBeacon des Patienten mit dem mobilen Endgerät und der geöffneten Anwendung berührt. Indem dabei der Patient mit dem Patienten der Aufgabe verglichen wird, soll sichergestellt werden, dass der richtige Patient verlegt wird. Deshalb soll die Aufgabe nur gestartet werden können, wenn die Patienten übereinstimmen. Die Aufgabe soll abgeschlossen werden können, indem wiederum der iBeacon des Facharztes an dessen Zimmer mit dem mobilen Gerät und geöffneter Anwendung berührt wird. Falls der Facharzt, der über den iBeacon identifiziert wird, mit dem Facharzt in der Aufgabenbeschreibung übereinstimmt, soll die Aufgabe korrekt abgeschlos-

sen werden. Die Rückverlegung soll ähnlich ausgeführt werden können. Zum Starten der Aufgabe soll der iBeacon des Patienten berührt werden und zum Abschließen der iBeacon des passenden Krankenzimmers des Patienten. Auch hierbei darf die Aufgabe nur gestartet oder abgeschlossen werden können, wenn die jeweiligen Informationen übereinstimmen.

6.1.3 Facharzt

Einem Facharzt soll es durch Medical Beacons möglich sein, eine Aufgabenliste der fachärztlichen Untersuchen anzeigen zu lassen. Dabei sollen nur Aufgaben angezeigt werde, deren Patienten sich bereits im Wartezimmer befinden. Möchte der Facharzt eine Aufgabe starten, so muss der iBeacon des wartenden Patienten mit dem mobilen Endgerät und der geöffneten Anwendung berührt werden. Stimmt der Patient mit dem in der Aufgabe angegebenen Patienten überein, so wird sie gestartet. Die Aufgabe soll abgeschlossen werden können, indem wiederum der iBeacon des Patienten berührt wird.

6.1.4 Weitere Anforderungen

Auf den jeweiligen Aufgabenlisten von Fachärzten und Krankenpflegern sollen nur Aufgaben aufgelistet werden, welche noch nicht von anderen Anwendern gestartet worden sind oder bereits als abgeschlossen markiert wurden. Sobald ein angemeldeter Facharzt oder Krankenpfleger eine Aufgabe auswählt und sie startet, soll sie an den angemeldeten Anwender gekoppelt werden und als gestartete Aufgabe in dessen Liste angezeigt werden. Für andere Anwender soll diese vom angemeldeten Anwender gestartete Aufgabe nicht mehr angezeigt und auswählbar sein. Ferner sollen vom angemeldeten Anwender gestartete Aufgaben von selbigem abgebrochen werden können. Dadurch soll sie auch wieder für andere Anwender sichtbar und auswählbar gemacht werden. Um dieses Verhalten bereitstellen zu können, müssen entsprechende Definitionen für den Status einer Aufgabe erarbeitet werden. Das Starten und Abschließen von Aufgaben soll über direkte Berührungen des mobilen Endgeräts mit iBeacons vonstatten gehen, um fehlerhafte Verlegungen, Verlegungen falscher Patienten oder Untersuchungen falscher Patienten auszuschließen. Ferner sollen Informationen über Patienten, im speziellen deren An- oder Abwesenheit im Krankenzimmer, in einer Datenbank vermerkt werden. So soll ein Patient als abwesend markiert werden, wenn er sich bei einer fachärztlichen Untersuchung oder auf dem Weg zu dieser befindet. Sobald der Patient in sein Krankenzimmer rückverlegt wurde, soll er

wieder als anwesend markiert werden. Um einen Überblick über verfügbare Medikamente in den unterschiedlichen Medikamentenlagern des Krankenhauses erhalten zu können, sollen bei Abschluss einer Medikation und dem Berühren des iBeacons am Medikamentenlager automatisch die Bestände der Medikamente aktualisiert werden. Die eingesetzten iBeacons müssen durch eine leicht erweiterbare Herangehensweise konfiguriert werden können, die sich zur eindeutigen Identifikation von Patienten, Krankenzimmer und Praxen von Fachärzten eignet.

Die einzelnen funktionalen Anforderungen der Anwendungsfälle und die notwendigen technischen Anforderungen an Medical Beacons sind nochmals in Tabelle 6.1 mit Erläuterungen versehen aufgelistet.

Tabelle 6.1: Tabellarische Aufstellung der Anforderungen

Anforderung	Beschreibung
Mobiles Endgerät	Zur Kommunikation mit iBeacons wird ein Bluetooth LE-fähiges mobiles Endgerät benötigt.
Organisationseinheiten	Es werden drei unterschiedliche Organisationseinheiten mit unterschiedlichen Anwendungsfällen und Berechtigungen benötigt (Stationsärzte, Krankenpfleger, Fachärzte).
iBeacon-Arten	Jeder Patient erhält einen eindeutigen den Patienten identifizierenden iBeacon. Jedes Krankenzimmer, jedes Medikamentenlager und jede Praxis eines Facharztes einen den Raum identifizierenden iBeacon.
iBeacon-Konfiguration	Die iBeacons müssen leicht erweiterbar konfiguriert werden, Patienten und Räume eindeutig identifizieren und die Position über deren Konfiguration ermittelbar sein.
Positionsbestimmung	Über die aufgestellten iBeacons und deren Konfiguration soll die aktuelle Station ermittelt werden können.

Direkter Kontakt mit iBeacons	Die Anwendung soll direkten Kontakt mit iBeacons erkennen können, um z.B. Aufgaben zu starten und zu beenden oder Patientenlisten und Patientenakten aufzurufen.
Aufgabenstatus	Um Aufgabenlisten für Fachärzte und Krankenpfleger erstellen zu können, müssen Aufgaben mit einem aussagekräftigen Status versehen werden.
Medikamentenbestand	Der Medikamentenbestand soll für jedes Lager separat dokumentiert und automatisch aktualisiert werden.
An- und Abwesenheit von Patienten	Die An- und Abwesenheit von Patienten in deren Krankenzimmern soll dokumentiert werden.
Login	Es soll ein Login angeboten werden, um die Anwendung auf eine der Organisationseinheiten anzupassen.
Patientenliste aufrufen	Ein Stationsarzt soll eine Patientenliste von Patienten auf seiner aktuellen Station einsehen können.
Zimmerliste aufrufen	Über den direkten Kontakt mit einem iBeacon eines Krankenzimmers sollen die Patienten innerhalb des Krankenzimmers angezeigt werden.
Patientenakte aufrufen	Über einen Listeneintrag oder direkten Kontakt mit einem iBeacon eines Patienten soll eine Patientenakte eingeblendet werden. Die Patientenakte enthält Informationen über den Patienten und eine Auflistung von Behandlungen.
Medikation hinzufügen	Über die Patientenakte soll vom Facharzt eine Medikation hinzugefügt werden können.
Medikamente auswählen	Bei Hinzufügen einer Medikation sollen Medikamente und deren Anzahl angegeben werden können.
Fachärztliche Untersuchung hinzufügen	Über die Patientenakte soll vom Facharzt eine fachärztliche Untersuchung hinzugefügt werden können.

Facharzt auswählen	Beim Hinzufügen einer fachärztlichen Untersuchung soll ein Facharzt ausgewählt werden können.
Aufgabenliste aufrufen	Krankenpfleger: Offene und vom angemeldeten Anwender gestartete Aufgaben der aktuellen Station sollen aufgelistet werden. Facharzt: Offene und vom angemeldeten Facharzt begonnene Aufgaben sollen aufgelistet werden, offene aber nur, wenn der Patient im Wartezimmer angekommen ist.
Medikation durchführen	Bei der Entnahme von Medikamenten aus einem Medikamentenlager sollen die in der Medikation angegebenen Medikamente im Bestand aktualisiert werden.
Fachärztliche Untersuchung durchführen	Eine Fachärztliche Untersuchung beginnt mit der Verlegung des Patienten vom Krankenzimmer zum Facharzt, geht über zur Untersuchung durch den Facharzt und endet mit der Rückverlegung vom Facharzt ins Krankenzimmer des Patienten.

7 | Entwurf

In diesem Kapitel wird anhand der zuvor gesammelten Anforderungen der Entwurf der Context Aware Application Medical Beacons vorgestellt. Medical Beacons besteht nicht nur aus der eigentlichen mobilen Anwendung auf einem mobilen Endgerät, sondern zusätzlich aus einem Server und einer angehängten Datenbank. All diese Komponenten verwenden ein Organisationsmodell, um die unterschiedlichen Anwendertypen voneinander zu unterscheiden. Zusätzlich wird eine einheitliche Konfiguration und Kodierung der iBeacons benötigt, um die oben genannten Anforderungen umsetzen zu können. Wie die einzelnen Komponenten aufgebaut sind und das Gesamtsystem arbeitet, wird in den folgenden Kapitel erläutert.

7.1 Architektur

In Abbildung 7.1 ist die Architektur des Systems Medical Beacons und dessen Komponenten dargestellt, welche einer *SOA* (*Service-orientierte Architektur*) sehr ähnelt [17]. Wie in der Abbildung zu erkennen ist, besteht das Systems aus *iBeacons*, einem *mobilen Client* und einem *Server*. Die iBeacons sind so konfiguriert, dass sie Patienten und unterschiedliche Arten von Räumen im Krankenhaus eindeutig identifizieren. Wie diese Konfiguration und die dazu notwendige Kodierung aussieht, wird in Kapitel 7.3 weiter erarbeitet.

Die von den iBeacons über Bluetooth LE versendeten Pakete, werden vom mobilen Client empfangen und zur weiteren Verarbeitung an einen *iBeacon-Service* versendet. Letzterer macht die Informationen innerhalb der Pakete in einfach zu verstehenden Datenstrukturen publik und speichert diese ab. Die eigentliche *Anwendung*, welche die oben gesammelten Anwendungsfälle umsetzen soll, verwendet diese Datenstrukturen, um damit Informationen über den aktuellen Kontext in Erfahrung zu bringen. Dazu gehört z.B. die Bestimmung

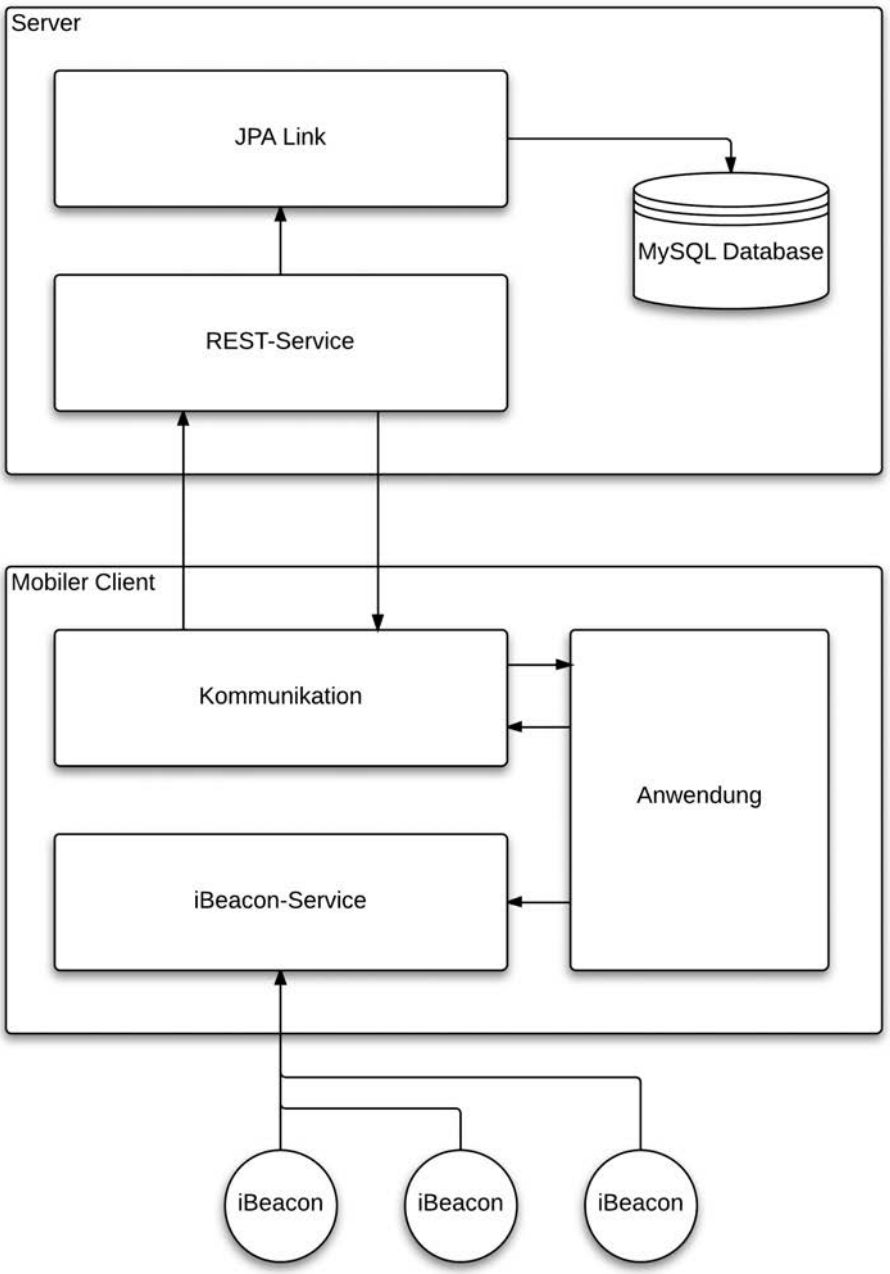

Abbildung 7.1: Übersicht und Kommunikation der einzelnen Komponenten von Medical Beacons

der aktuellen Station oder die Abfrage einer Patientenakte über direkten Kontakt mit einem Patienten. Der mobile Client besitzt des Weiteren eine Schnittstelle zur *Kommunikation* mit einem entfernten Server, um von ihm die notwendigen Daten zum Anzeigen einer Patientenakte oder Auflistung von Aufgaben der aktuellen Station zu erhalten.

Der Server nimmt über einen *REST-Service* Anfragen von mobilen Clients entgegen, um die gewünschten Ressourcen an diese auszuliefern. Dazu verwendet der REST-Service eine *Datenbank*, in der die notwendigen Daten abgelegt werden. Die Datenbank enthält Informationen über die unterschiedlichen Organisationseinheiten, Aufgaben und deren Status und z.B. Medikamentenbestände. Um eine direkte Eingabe und Implementierung von SQL-Statements zu verhindern, wird zwischen der Kommunikation des REST-Services mit der Datenbank ein *JPA-Link* eingefügt. Durch diesen ist es möglich, auf den Relationen der Datenbank mit objektorientierten Methoden und all den damit einhergehenden Vorteilen zu arbeiten.

7.2 Organisationsmodell

Das Organisationsmodell von Medical Beacons besteht aus vier verschiedenen Organisationseinheiten, die jeweils unterschiedliche Anwendungsfälle realisieren und Berechtigungen besitzen.

Ein *Stationsarzt* kann Patientenlisten anhand der aktuellen Station oder eines Krankenzimmers abfragen und Patientenakten einsehen. Weiter ist es nur ihm möglich, neue Medikationen und fachärztliche Untersuchungen hinzuzufügen. Ein *Krankenpfleger* besitzt die Berechtigungen, Aufgabenlisten anhand der aktuellen Station einzusehen und ist dafür zuständig, Medikationen und Verlegungen zu und von fachärztlichen Untersuchungen durchzuführen. Der *Facharzt* wiederum besitzt die Berechtigung, eine Aufgabenliste über seine durchzuführenden fachärztlichen Untersuchungen einzusehen und diese durchzuführen. Sowohl die drei Organisationseinheiten als auch deren Berechtigungen und Anwendungsfälle sind bereits aus Kapitel 6 bekannt. Durch einen Anmeldevorgang auf dem mobilen Client wird zwischen diesen unterschieden, wodurch die Anwendung die unterschiedlichen Anwendungsfälle und Berechtigungen angepasst bereitstellt.

Durch den Entwurf kommt noch die vierte Organisationseinheit der Patienten hinzu. Diese besitzen zwar keine Berechtigungen oder Anwendungsfälle im System selbst, werden jedoch für Patientenlisten, Medikationen und fachärztliche Untersuchungen benötigt. Eine

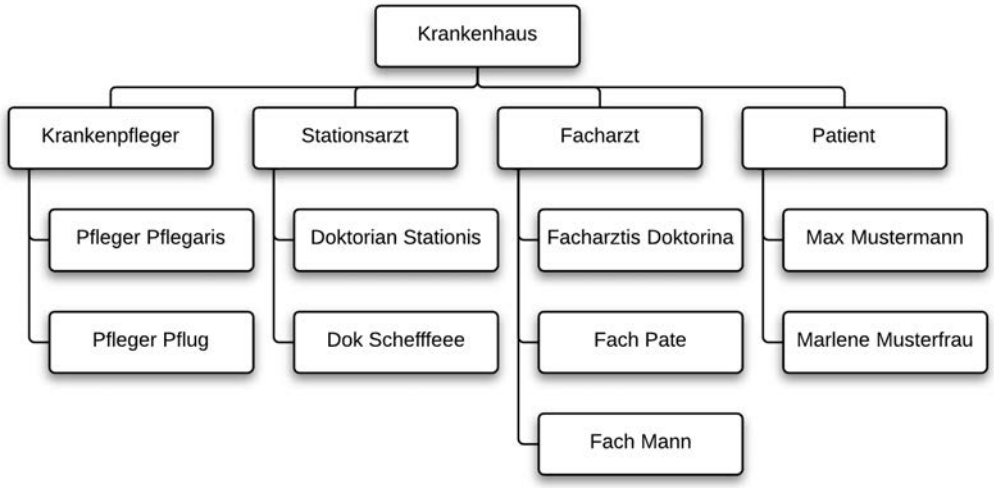

Abbildung 7.2: Die vier Organisationseinheiten von Medical Beacons mit mögliche Agenten

Übersicht der vier Organisationseinheiten und beispielhaften Agenten, die so teilweise in einer Testumgebung verwendet wurden, ist in Abbildung 7.2 abgebildet.

7.3 iBeacon-Kodierung

Durch die konfigurierbaren Werte von UUID, Major und Minor im Payload eines ADs, können die speziellen Anforderungen der eindeutigen Identifikation von Patienten, Krankenzimmern, Stationen und Medikamentenlagern umgesetzt werden. Dazu soll im Folgenden die Art und Weise vorgestellt werden, wie diese Informationen kodiert und interpretiert werden.

Im Krankenhaus-Prozess und der Implementierung von Medical Beacon werden unterschiedliche Arten von iBeacons eingesetzt. Dazu gehören Raspberry Pis, die durch die bereits vorgestellte Herangehensweise und Konfiguration als iBeacons verwendet werden. Neben diesen wurden sehr kleine iBeacons von Radius Networks verwendet, die den Vorteil haben, dass diese - anders als Raspberry Pis - keine direkte kabelgebundene Stromversorgung benötigen. Zusätzlich zu diesen Geräten wurde ein iPhone verwendet und eine kleine iBeacon-Anwendung entwickelt, mit welcher sowohl ADs gesendet als auch empfangen werden können. Der Vorteil der iPhone-Anwendung ist der, dass sie eine effiziente

und einfache Änderung der Informationen von UUID, Major und Minor ermöglicht. In Kapitel 8.1 wird genauer auf die Implementierung dieser hilfreichen Anwendung eingegangen. All diese Geräte besitzen jedoch dieselbe, im Folgenden vorgestellte Kodierung.

Die UUID ist der Bezeichner, den iBeacons versenden müssen, wenn sie ein in Medical Beacons und innerhalb des Krankenhaus-Prozesses eingesetzt werden. Die mobile Anwendung verarbeitet nur Informationen von iBeacons weiter, die als UUID `8e14411d-205d-3a42-8ea28feadbdedd79` gesetzt haben. Die Informationen innerhalb des Major- und Minor-Wertes, die mit der genannten UUID versehen sind, können nun verwendet werden, um Patienten, Krankenzimmer, Stationen und Medikamentenlager eindeutig zu identifizieren.

Anhand des Major-Wertes sollen sowohl die korrespondierenden Stationen als auch auf diesen Stationen befindliche Krankenzimmer, Praxen von Fachärzten und Medikamentenlager bestimmt werden. Dazu wird der Major-Wert so aufgeteilt, dass die Stellen vor den Einer- und Zehnerstellen eine Station identifizieren und der gesamte Major-Wert selbst die Zimmernummer. Ein empfangener Major-Wert von `202` identifiziert also Zimmer 202 auf Station 2. Der Major-Wert `1249` identifiziert Station 12 und das auf dieser Station befindliche Zimmer 1249. Durch diese Herangehensweise können also durch die Größe von 2 Byte des Major-Wertes 654 Stationen und jeweils 100 Zimmer eindeutig identifiziert werden[1]. Die Zimmer werden ferner so eingeteilt, dass die Zimmer 90 bis 99 für Medikamentenlager reserviert und nur die Zimmer 00 bis 89 für Krankenzimmer und Praxen von Fachärzten verwendet werden.

Anhand des Minor-Wertes kann der Major-Wert weiter verfeinert werden. In Medical Beacons wird der Minor-Wert für die Identifikation von Patienten herangezogen. Ein Patient ist also über den Major-Wert, bestehend aus Station und Zimmernummer, und dem Minor-Wert bestimmt. Der Minor-Wert wird als Bettnummer interpretiert, sodass ein Krankenzimmer auf einer bestimmten Station eine gewisse Anzahl an Betten besitzt. Ein Patient kann nun durch die Daten eines iBeacons wie folgt identifiziert werden. Empfängt die Anwendung den Major-Wert `202` und den Minor-Wert `1`, so handelt es sich beim empfangenen iBeacon um den Patienten auf Station 2 in Zimmer 02 in Bett 1. Durch eine Anfrage an einer Datenbank mit diesen Werten, kann dann der eindeutig identifizierbare Patient und dessen Informationen abgefragt werden. Wichtig ist dabei, dass jedem stationären Patienten eine Station, ein Zimmer und ein Bett zugewiesen wird. Diese Zuweisung muss bei

[1] Durch 2 Byte ist 65535 als Wert darstellbar. Da durch 65535 und die oben genannte Herangehensweise auf Station 655 aber nur 36 Zimmer identifiziert werden können, wird die Grenze bei Station 654 gesetzt.

einer dauerhaften Verlegung in ein anderes Zimmer erneuert werden, jedoch nicht bei einer zeitlich begrenzten Verlegung wegen einer fachärztlichen Untersuchung.

Durch den direkten Kontakt mit einem iBeacon wird mit der bisherigen Kodierung ein Patient identifiziert. Da aber zusätzlich auch der direkte Kontakt mit iBeacons an Krankenzimmern oder Medikamentenlagern sichergestellt werden soll, müssen die Minor-Werte weiter differenziert werden. Dazu erhalten iBeacons, die bestimmte Zimmer identifizieren sollen, den Wert 0 zugewiesen. Dadurch müssen Patienten und damit einhergehend die Bettnummer bei 1 beginnen. Der empfangene Major-Wert von `202` und Minor-Wert von 0 identifiziert also das Zimmer 02 auf Station 2. Da eine 0 im Minor-Wert enthalten ist, ist diese iBeacon nur mit einem Zimmer aber keinem Patienten verknüpft.

Die dafür notwendige Fallunterscheidung lässt sich durch Anwendung einfacher mathematischer Operationen auf den Major-Wert realisieren. Um zum einen die aktuelle Station durch den nächsten iBeacon und zum anderen die Station eines Zimmers oder eines Patienten zu erhalten, wird wie in Formel 7.1 vorgegangen. Dabei wird die Integer-Division auf digitalen Rechnersystemen genutzt, indem der Major-Wert durch 100 dividiert wird.

$$\begin{aligned} Def. : Major/100 &= Station \\ 202/100 &= 2 \\ 1349/100 &= 13 \end{aligned} \tag{7.1}$$

Um die Zimmernummer aus dem empfangenen Major-Wert zu extrahieren, wird der Modulo-Operator angewendet. Die Einer- und Zehnerstelle des Major-Werte erhält man durch Formel 7.2.

$$\begin{aligned} Def. : Major\%100 &= Zimmer \\ 202\%100 &= 02 \\ 1349\%100 &= 49 \end{aligned} \tag{7.2}$$

Tabelle 7.1: iBeacon-Kodierung eines Patienten

UUID	
8e14411d-205d-3a42-8ea28feadbdedd79	
Major	Minor
202	3
Beschreibung	
Patient in Bett 3 in Zimmer 02 auf Station 2	

Tabelle 7.2: iBeacon-Kodierung eines Krankenzimmers oder einer Praxis

UUID	
8e14411d-205d-3a42-8ea28feadbdedd79	
Major	Minor
1342	0
Beschreibung	
Krankenzimmer oder Praxis mit Zimmer 42 auf Station 13	

Um einen Überblick über die unterschiedlichen Konstellationen von Major- und Minor-Werten und deren Interpretation zu erhalten, werden in den Tabellen 7.1, 7.2, 7.3, und 7.4 einige Beispiele dargestellt.

7.4 Mobiler Client

Der mobile Client besteht aus einer mobilen Anwendung, die auf einem mobilen Endgerät installiert ist. Das mobile Endgerät muss in der Lage sein, über Bluetooth LE zu kommunizieren, um die Informationen sendender iBeacons empfangen und verarbeiten zu können.

Tabelle 7.3: iBeacon-Kodierung eines Medikamentenlagers

UUID	
8e14411d-205d-3a42-8ea28feadbdedd79	
Major	Minor
393	0
Beschreibung	
Medikamentenlager mit Zimmer 93 auf Station 3	

Tabelle 7.4: Undefinierte iBeacon-Kodierung

UUID 25351F16-82F4-4E23-B38A-038DC75E4D9C	
Major 383	Minor 4
Beschreibung Falsche UUID; Medikamentenlager 83 auf Station 3, aber Minor ist nicht 0	

Der mobile Client besteht aus drei großen Komponenten zur Kommunikation mit dem Server, einem iBeacon-Service für den Empfang und die Verarbeitung der Informationen von iBeacons und der eigentlichen Anwendung, welche den Kontext ermittelt und die Anwendungsfälle der einzelnen Organisationseinheiten anbietet.

Der Kontext in Medical Beacons wird vom aktuellen Anwendungsfall und den unterschiedlichen Interaktionen mit iBeacons abhängen. Es existieren zwei unterschiedliche Arten von Interaktionen. Zum einen der direkte Kontakt mit einem iBeacon und zum anderen die einfache Anwesenheit des mobilen Endgeräts innerhalb der Reichweite eines iBeacons. Befindet sich der Stationsarzt im Anwendungsfall, um eine Patientenliste zu erhalten, so besteht der Kontext aus dem Anwendungsfall *Patientenliste aufrufen* und der Entfernung zum nächst gelegenen iBeacon und dessen Station. Anhand dieses Kontextes wird dann die passende Patientenliste dargestellt. Befindet er sich im gleichen Anwendungsfall, führt jedoch einen direkten Kontakt mit einem iBeacon durch, ändert sich der Kontext dahingehen, dass die Context Aware Application weiß, dass anhand der direkten Interaktion und des genannten Anwendungsfall eine Patientenliste eines Krankenzimmers angezeigt werden soll. Die einzelnen Anwendungsfälle und ihre jeweiligen Interaktionen zur Kontextbestimmung werden in Kapitel 10 und der Präsentation der mobilen Anwendung weiter erläutert.

7.4.1 iBeacon-Kommunikation

Das Sequenzdiagramm in Abbildung 7.3 stellt den Entwurf des Nachrichtenaustauschs und dessen Abfolge dar, die für das Empfangen und Verarbeiten von Informationen der iBeacons notwendig sind, um die Anwendungsfälle aus den Anforderungen umsetzen zu können. Ein *iBeacon* versendet ein konfiguriertes AD mit dem oben definierten Payload als Broadcast. Das mobile Endgerät empfängt das AD und gibt es an die mobile Anwen-

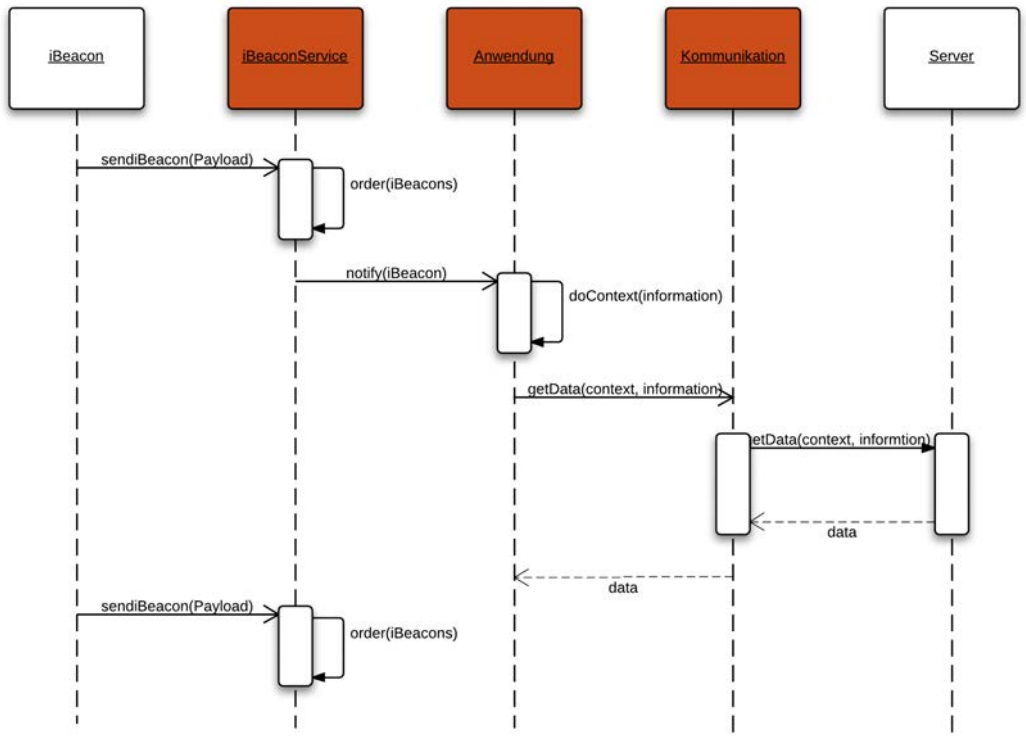

Abbildung 7.3: Sequenzdiagramm der Komponenten

dung weiter. Der *iBeaconService* filtert nun die ankommende Signale dahingehend aus, dass nur die Informationen weiter verarbeitet werden, die die passende UUID beinhalten. Danach sortiert er die iBeacons aufsteigend nach ihrer Entfernung. Die so sortierten iBeacons und deren Informationen werden vom iBeaconService an die eigentliche *Anwendung* weitergereicht. Diese bestimmt anhand des aktuellen Anwendungsfalls und der erhalten Informationen den Kontext. Der Kontext könnte das Anzeigen einer Patientenliste auslösen, falls ein Stationsarzt den Anwendungsfall *Patientenliste anzeigen* ausgewählt hat. In diesem Fall wird die Station anhand des iBeacons mit der geringsten Entfernung bestimmt. Befindet sich unter den iBeacons in diesem Anwendungsfall ein solcher, zu dem direkter Kontakt besteht, die Distanz also wenige Zentimeter kurz ist, so ändert sich der Kontext dahingehend, sodass die Patientenliste eines Krankenzimmers oder eine Patientenakte eingeblendet wird. Der Kontext wird dann durch den Minor-Wert und der damit stattfindenden Fallunterscheidung weiter verfeinert. Wurde der Kontext von der Anwendung korrekt bestimmt, so sendet sie diesen und die gewünschten Informationen an die Komponente

zur *Kommunikation*. Letztere startet wiederum einen synchronen Aufruf mit dem Kontext und den Informationen an einen *Server*. Der Server antwortet darauf mit den gewünschten Daten, die von der Kommunikation empfangen und an die eigentliche Anwendung weitergereicht werden. Die Anwendung ist dann in der Lage, je nach Kontext und Anwendungsfall die gewünschte Aktion automatisch durchzuführen.

7.4.2 Klassenstruktur

In Abbildung 7.4 ist ein vereinfachtes Klassendiagramm des mobilen Clients von Medical Beacons abgebildet. In diesem Klassendiagramm sind ebenfalls die drei Komponenten iBeaconService, Kommunikation und Anwendung vorhanden, die in Kapitel 7.1 bereits kurz vorgestellt wurden. In diesem Kapitel sollen neben diesen Komponenten auch weitere vorgestellt werden, die für die Implementierung notwendig sind.

Innerhalb des Moduls *Controller* sind drei Klassen vorhanden. Die Klasse `iBeaconService` ist dafür zuständig, die Daten von iBeacons zu empfangen und zu verarbeiten. Die `Kommunikation` wird benötigt, um Informationen an den Server zu senden und von diesem zu empfangen. Dazu zählt z.B. das Empfangen einer Patientenliste, einer Patientenakte oder dem Erstellen einer neuen Medikation. Weiter sind die zwei Interfaces `NetworkController` und `iBeaconService` notwendig, die von den meisten Klassen der eigentlichen Anwendung implementiert werden. Die Klassen innerhalb der Anwendung, die diese Interfaces Implementierung, sind in der Lage sich bei den beiden genannten Klassen zu registrieren und sowohl Notifikationen über iBeacons als auch über abgeschlossene Downloads und Uploads zu erhalten. Der `LoginController` wird benötigt, damit sich die einzelnen Organisationseinheiten in der Anwendung anmelden und damit die unterschiedlichen Anwendungsfälle ausführen können.

Die Klassen innerhalb der Anwendung lassen sich grob in die drei Bereiche *Doctor*, *Nurse* und *Specialist* einteilen und entsprechen den drei Organisationseinheiten, die Anwendungsfälle in den Anforderungen definiert haben. Jede dieser Klassen besitzt eine korrespondierende `View` auf dem mobilen Gerät und ist für die unterschiedlichen Anwendungsfälle zuständig. Aufgrund der Übersichtlichkeit wurden Assoziationen zwischen diesen Klassen und deren `Views` ausgeblendet.

Die Klasse `DoctorStartController` ist der Controller, der nach der erfolgreichen Anmeldung eines Stationsarztes aufgerufen wird. Diese Klasse kann durch die Informatio-

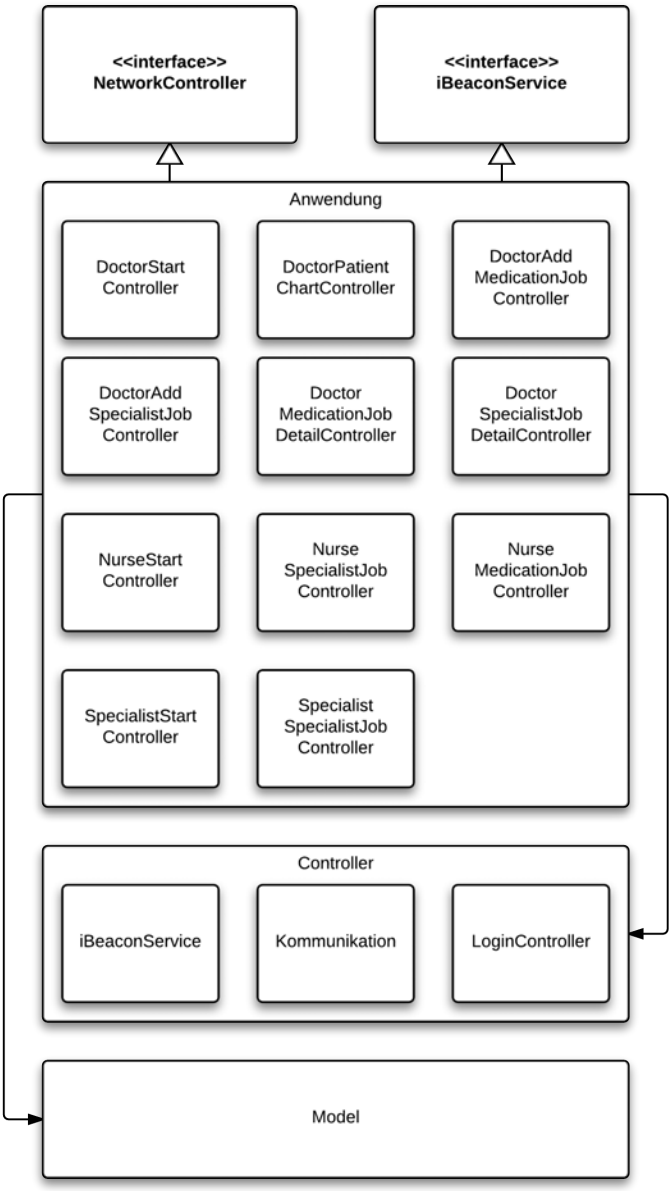

Abbildung 7.4: Vereinfachtes Klassendiagramm des mobilen Clients

nen der Klasse `iBeaconService` die aktuelle Station des Stationsarztes ermitteln und dadurch eine Patientenliste anzeigen. Weiter erkennt die Klasse direkten Kontakt mit iBeacons von Krankenzimmer und Patienten, um dementsprechend eine Patientenliste ei-

nes Krankenzimmers oder die Patientenakte eines Patienten einzublenden. Über die Auswahl eines Patienten aus der Patientenliste oder über direkten Kontakt mit einem iBeacon eines Patienten wird die Klasse `DoctorPatientChartController` aufgerufen. Diese bildet die Patientenakte des Patienten ab und bietet die Möglichkeiten über direkte Interaktion mit dem Stationsarzt eine neue Medikation oder fachärztliche Untersuchung hinzuzufügen. Hierzu werden die beiden Klassen `DoctorAddMedicationJobController` und `DoctorAddSpecialistJobController` und deren korrespondierenden `Views` benötigt. Während dem Hinzufügen werden die Funktionen aus der Klasse `Kommunikation` und die Implementierung des entsprechenden Interfaces benötigt. Weiter kann der Stationsarzt aus der Klasse `DoctorPatientChartController` heraus bereits hinzugefügte Medikationen und fachärztliche Untersuchungen durch die Klassen `DoctorPatientMedicationJobDetailController` und `DoctorPatientSpecialistJobDetailController` einsehen.

Durch eine erfolgreiche Anmeldung als Krankenpfleger wird die Klasse `NurseStartController` aufgerufen. Dieser Klasse ist es möglich, anhand der Informationen des `iBeaconServices` und den Funktionen der Klasse `Kommunikation` die offenen und vom angemeldeten Krankenpfleger gestartete Medikationen und fachärztlichen Untersuchung auf der aktuellen Station aufzulisten. Durch die Auswahl einer Medikation wird die Klasse `NurseMedicationJobController` aufgerufen und durch die Auswahl einer fachärztlichen Untersuchung die Klasse `NurseSpecialistJobController`. Durch diese werden Informationen zu den jeweiligen Aufgaben eingeblendet und Funktionen zum Starten und Beenden dieser bereitgestellt. Wie in den Anforderungen definiert, können diese Klassen direkten Kontakt zu iBeacons erkennen, was zum verwechslungsfreien Starten und Beenden von Aufgaben führt und z.B. den Bestand eines expliziten Medikamentenlagers aktualisiert.

Mit der erfolgreichen Anmeldung als Facharzt wird die Klasse `SpecialistStartController` aufgerufen. Diese listet fachärztliche Untersuchungen auf, die vom angemeldeten Facharzt durchgeführt werden können. Dabei werden jedoch nur Aufgaben mit Patienten angezeigt, die Bereits in die Praxis des Facharztes verlegt wurden. Durch die Auswahl einer Aufgabe wird die Klasse `SpecialistSpecialistJobController` und dessen `View` aufgerufen. Diese kann den direkten Kontakt zu iBeacons feststellen, was zum Starten und Beenden von Aufgaben notwendig ist.

Die Klassen innerhalb des Moduls *Model* sind ein objektorientiertes Abbild der Datenstrukturen, die im Server und der angehängten Datenbank abgelegt sind. Diese objektorientierten Abbildungen werden von den einzelnen Controllern und Anzeigen zur Interaktion

verwendet. Bevor der Entwurf der Datenhaltung und Datenstruktur vorgestellt wird, wird im folgenden zunächst der Server vorgestellt.

7.5 Server

Der Server besitzt als öffentliche Schnittstelle nach außen einen *REST*-Service (*Representational State Transfer*). Ein REST-Service besitzt einige wichtige Eigenschaften:

- REST-Services publizieren einfache Schnittstellen, die nur aus dem Aufruf einer URI bestehen. Jede valide URI eines REST-Services zeigt auf genau eine Ressource, die durch verschiedene HTTP-Methoden entweder ausgeliefert, aktualisiert, erstellt oder gelöscht werden können.
- Die durch die URIs verlinkten Ressourcen können in unterschiedlichen Repräsentationen ausgegeben werden (wie z.B. JSON oder XML).
- Der REST-Service selbst speichert keinerlei Zustandsinformationen einer Kommunikation zwischen Service/Server und Client. Alle notwendigen Informationen werden in den Ressourcen selbst und durch HTTP-Response-Codes abgebildet.
- Die Kommunikation mit einem REST-Service basiert vollständig auf dem HTTP-Protokoll und dessen Methoden (GET, POST, PUT, DELETE).

Durch die nicht notwendige Zustandsspeicherung, eine einfache Schnittstelle und die ressourcenorientierte Arbeitsweise, sind REST-Services sehr leichgewichtig. Über die oben genannten Methoden des HTTP-Protokolls kann dem REST-Service während der Kommunikation mitgeteilt werden, wie mit der Ressource an der aufgerufenen URI vorgegangen werden soll.

- **GET** liefert die an der aufgerufenen URI gelegene Ressource aus, falls sie vorhanden ist.
- **POST** erstellt die im Body des Requests mitgelieferte Ressource an der aufgerufenen URI.
- **PUT** aktualisiert die an der aufgerufenen URI gelegene Ressource durch die Ressource im Body des Requests, falls diese vorhanden ist. Sollte die Ressource nicht vorhanden sein, so wird eine neue Ressource erstellt.

- **DELETE** löscht die Ressource an der aufgerufenen URI.

Das Konzept eines REST-Services als Schnittstelle nach außen wurde aus zwei Gründen in Medical Beacons gewählt. Zum einen wird die Implementierung des mobilen Clients auf iOS basieren. Auf dieser Plattform ist das Anbinden eines schwergewichtigen SOAP-Services und seiner Definition in einer WSDL aufgrund fehlender Bibliotheken und Konverter nur durch die händische Implementierung von SOAP-Nachrichten möglich. Deshalb bietet es sich an dieser Stelle an, auf REST zu setzen. Eine REST-Kommunikation basiert wie oben erwähnt nur aus dem Aufruf einer URI über das HTTP-Protokoll und der Manipulation von ausgelieferten Ressourcen. Da dies im Grunde nicht anderes ist, als das aufrufen einer Website und das Aufrufen von Websites auf allen mobilen Plattformen unterstützt wird, wird dieses leichtgewichtige Protokoll verwendet. Zum anderen deshalb, weil die Daten, die vom mobilen Client benötigt werden, hauptsächlich Ressourcen in der Form von Entitäten sind, weswegen die ressourcenorientierte Arbeitsweise eines REST-Services sehr gut passt. Die Entitäten und deren Strukturen werden in Kapitel 7.6 weiter behandelt.

Die notwendigen Ressourcen werden vom REST-Service im JSON-Format ausgeliefert. JSON ist ein leicht verständliches und leicht kodierbares Format, das weniger Ballast mit sich führt als z.B. XML (z.B. XML-Schema). Aufgrund des nicht vorhanden Ballastes und der Einfachheit, eignet sich JSON sehr gut für den Austausch von Ressourcen, wie es der REST-Service tut und es in Medical Beacons notwendig ist. Bei den ausgelieferten Ressourcen handelt es sich beispielsweise um Patientenakten, fachärztliche Untersuchungen oder Medikamente. Ferner werden auf so gut wie allen Plattformen geeignete und gut dokumentierte JSON-Konverter angeboten, wodurch eine gewisse Interoperabilität der ausgetauschten Ressourcen sichergestellt wird.

In Listing 7.1 ist die Ressource eines Patienten im JSON-Format abgebildet. Es beinhaltet Informationen über den Patienten, wie z.B. die Station, das Krankenzimmer und das Bett, um den Patienten über einen iBeacon identifizieren zu können. Ferner werden die Medikationen und fachärztlichen Untersuchungen mitgeliefert, die für diesen Patienten durch einen Stationsarzt erstellt wurden. Im Beispiel sind diese Listen in Zeilen 12 und 13 leer, da noch keine Behandlungen definiert wurden.

Der REST-Service in Medical Beacons soll die folgende API anbieten, um die Anwendungsfälle und Anforderungen umsetzen zu können. Dabei ist die URL definiert als *localhost:8080/MedicalBeacons/* und die URIs der einzelnen Ressourcen und die dazugehörigen HTTP-Methoden wie folgt.

```
{
  "patientId":1,
  "firstname":"Philip",
  "lastname":"Geiger",
  "gender":"male",
  "birthday":"12.10.1989",
  "admission":"01.04.1923",
  "isInRoom":0,
  "floorNumber":1,
  "roomNumber":101,
  "bedNumber":1,
  "medicationJobs":[],
  "specialistJobs":[]
}
```

Listing 7.1: patient.json

GET: patients/list Liefert alle Patienten des Krankenhauses aus.

GET: patients/id/{patientId} Liefert die Ressource eines Patienten über dessen id aus.

GET: patients/station/{stationId}/room/{roomId}/bed/{bedId} Liefert die Ressource eines Patienten über dessen zugehörige Station, Zimmernummer und Bettnummer aus. Wird verwendet, um einen Patienten anhand seines iBeacons zu identifizieren.

GET: patients/station/{stationId}/room/{roomId} Liefert alle Patienten aus dem angegeben Krankenzimmer aus. Wird verwendet um über einen iBeacon die Patienten eines Krankenzimmers aufzulisten

GET: patients/station/{stationId} Liefert alle Patienten einer Station aus.

POST: patients/id/{patientId}/jobs/medicationjob Die über den Request mitgelieferte Medikation wird einem Patienten hinzugefügt.

POST: patients/id/{patientId}/jobs/specialistjob Die über den Request mitgelieferte fachärztliche Untersuchung wird einem Patienten hinzugefügt.

GET: jobs/all Liefert alle Medikationen und fachärztlichen Untersuchungen aus.

GET: jobs/medicationjobs Liefert alle Medikationen aus

GET: jobs/specialistjobs Liefert alle fachärztlichen Untersuchungen aus

GET: jobs/nurse/station/{stationId}/employeeid/{employeeId} Liefert alle Medikationen und fachärztlichen Untersuchungen von Patienten auf der aktuellen Station aus, die vom angegebenen Krankenpfleger begonnen oder beendet werden können.

GET: jobs/specialist/employeeid/{employeeId} Liefert alle fachärztlichen Untersuchungen aus, die vom angegeben Facharzt begonnen oder beendet werden können.

PUT: jobs/medicationjobs Die über den Request mitgelieferte Medikation, wird aktualisiert. Wird z.B. verwendet, um den Status einer Medikation zu ändern.

PUT: jobs/specialistjobs Die über den Request mitgelieferte fachärztliche Untersuchung, wird aktualisiert. Wird z.B. verwendet, um den Status einer fachärztlichen Untersuchung zu ändern.

POST: jobs/medicationjobs Die über den Request mitgelieferte Medikation, die einen Patienten enthalten muss, wird hinzugefügt.

POST: jobs/specialistjobs Die über den Request mitgelieferte fachärztliche Untersuchung, die einen Patienten enthalten muss, wird hinzugefügt.

GET: employees/list Liefert alle Mitarbeiter des Krankenhauses aus.

GET: employees/type/{type} Liefert alle Mitarbeiter des Krankenhauses mit dem angegeben Typ aus. Wird z.B. verwendet, um dem Stationsarzt die Möglichkeit zu bieten, einen Facharzt für eine fachärztliche Untersuchung auszuwählen.

GET: drugs/list Liefert alle Medikamente aus. Wird z.B. verwendet, um dem Stationsarzt die Möglichkeit zu bieten, Medikamente für eine Medikation auszuwählen.

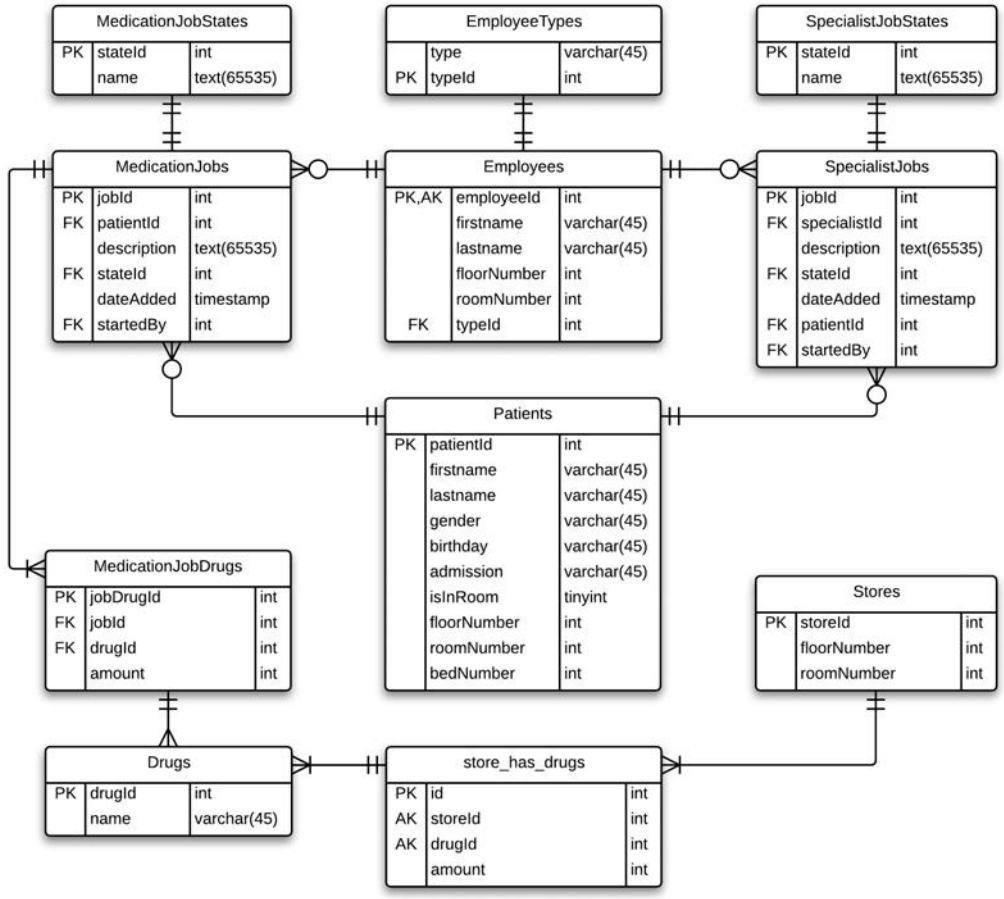

Abbildung 7.5: ER-Diagramm des Datenmodells von Medical Beacons

7.6 Datenmodell

Das im Folgenden beschriebene Datenmodell wird so sowohl objektorientiert in Form von Klassen im mobilen Client und in Relationen in einer MySQL-Datenbank auf dem Server gehalten als auch vom REST-Service im JSON-Format ausgeliefert. In Abbildung 7.5 ist das Modell als ER-Diagramm abgebildet.

Die Entität *Patients* besteht neben den typischen Informationen vor allem aus Attributen für die Station, das Krankenzimmer und die Bettnummer. Durch *floorNumber*, *roomNumber* und *bedNumber* kann ein iBeacon dahingehenden konfiguriert werden, dass er über diese

Tabelle 7.5: Status-Kodierung von Medikationen

Wert	Beschreibung	Ausführbar von
1	Offen	Krankenpfleger
2	In Bearbeitung	Krankenpfleger
3	Abgeschlossen	-

Attribute einen Patienten eindeutig identifiziert. Weiter besitzt ein Patient das Attribute *isInRoom*, mit dem ausgedrückt werden kann, ob der Patient im Krankenzimmer ist oder sich bei einer ärztlichen Untersuchung befindet.

Ein Patient besitzt keine oder mehrere Medikationen, die als Entität *MedicationJobs* dargestellt sind. Eine Medikation besitzt genau einen Patienten, der über dessen *id* identifiziert wird. Neben einer Beschreibung und einem Zeitstempel besitzt die Entität ein Attribut für den Status und eines für den Krankenhausmitarbeiter, der die Aufgabe gestartet hat. Das Attribut *stateId* wird benötigt, um offene Aufgaben von bereits gestarteten zu differenzieren, das Attribut *startedBy* um gestartete Aufgaben den entsprechenden Mitarbeiter zuzuordnen. Durch die Kombination der beiden Attribute ist es möglich, einem angemeldeten Anwender nur offene oder von ihm gestartete Medikationen anzuzeigen. Der Status der Medikation ist durch die Entität *MedicationJobStates* modelliert. Die verschiedenen Status und ihre Beschreibungen sind in Tabelle 7.5 aufgelistet.

Zu einer Medikation gehört auch immer mindestens ein Medikament, das als *MedicationJobDrugs* modelliert wird. Diese Entität besteht aus dem eigentlichen Medikament und dessen verordnete Anzahl.

Das eigentliche Medikament ist die Entität *Drugs* und besitzt einen Namen. Da es eine Anforderung ist, die Bestände der im Krankenhaus verteilten Medikamentenlager zu dokumentieren, existiert die Entität *Stores*. Ein Medikamentenlager besitzt immer eine Station und eine Raumnummer, durch die das Lager eindeutig identifiziert werden kann. Ein solches Medikamentenlager besitzt Medikamente mit einer bestimmten Anzahl. Diese Information ist als *store_has_drugs* modelliert und besitzt die Attribute *drugId* und *amount*. Amount gibt dabei die Anzahl an und die drugId verweist auf die oben vorgestellte Entität *Drugs*.

Neben einer Medikation kann ein Patient auch keine oder mehrere fachärztliche Untersuchungen besitzen, was als *SpecialistJobs* modelliert ist. Auch eine fachärztliche Untersuchung besitzt die beiden Attribute *startedBy* und *stateId*, wodurch die Aufgabe ei-

Tabelle 7.6: Status-Kodierung von fachärztlichen Untersuchungen

Wert	Beschreibung	Ausführbar von
1	Offen	Krankenpfleger
2	Unterwegs zur Untersuchung	Krankenpfleger
3	Auf Untersuchung wartend	Facharzt
4	In Untersuchung	Facharzt
5	Auf Abholung wartend	Krankenpfleger
6	Auf Rückweg	Krankenpfleger
7	Abgeschlossen	-

nem Krankenhausmitarbeiter zugeordnet und auf Aufgabenlisten korrekt angezeigt werden kann. Die Status der fachärztlichen Untersuchung sind ebenfalls als eigene Entität *SpecialistJobStates* modelliert und sind in Tabelle 7.6 aufgelistet. Dabei ist zu beachten, dass je nach Status eine andere Organisationseinheit die fachärztliche Untersuchung starten, durchführen oder abschließen darf. Weiter besitzt eine fachärztliche Untersuchung einem vom Stationsarzt zugewiesenen Facharzt (*specialistId*), der die Untersuchung durchführen soll.

Die Entitäten *MedicationJosb* und *SpecialistJobs* besitzen beide das Attribut *startedBy*, letztere auch das Attribut *specialistId*, die auf die Entität *Employees* verweist. Ein Mitarbeiter besitzt neben den typischen Informationen vor allem die Attribute *typeId*, *floorNumber* und *roomNumber*. Der Typ beschreibt, welcher Organisationseinheit der Krankenhausmitarbeiter zugehörig ist und ist in *EmployeeTypes* modelliert. Handelt es sich bei dem Mitarbeiter um einen Facharzt, so müssen die beiden Attribute *roomNumber* und *floorNumber* gesetzt sein. Dies ist notwendig, um den Krankenpfleger über den Raum zu informieren, in den der Patient verlegt oder abgeholt werden soll. Weiter wird diese Information benötigt, um, wie oben beschrieben, eine fehlerhafte Verlegung durch den Kontakt mit einem iBeacon und dem Vergleichen der Werte auszuschließen.

8 | Implementierung

In diesem Kapitel soll nun die Implementierung vorgestellt werden, die anhand der Anforderungen und des Entwurfs durchgeführt wurde. Wie im Entwurf bereits beschrieben, besteht Medical Beacons aus einem Server mit angehängter Datenbank und einem mobilen Client. In den nächsten Kapiteln werden ausgewählte Beispiele der Implementierung beider Komponenten erläutert, die wichtige Konzepte zur Umsetzung der Anforderungen aufzeigen. Ferner wird die Implementierung einer iOS-Anwendung aufgezeigt, die ebenfalls als iBeacon eingesetzt werden kann.

8.1 iPhone iBeacon App

Sowohl für einfaches Testen der Anwendung als auch schnelles Erstellen und Konfigurieren von iBeacons, wurde eine iPhone iBeacon Applikation entwickelt. Die Anwendung soll im Krankenhaus-Prozess als iBeacon eingesetzt werden können und Felder für die Konfiguration des Major- und Minor-Wertes bereitstellen. Das Resultat dieser Entwicklung ist in Abbildung 8.1 abgebildet. Für die Entwicklung dieser Anwendung wurde iOS in der Version 7.1 verwendet. Seit Version 7 werden von Apple Frameworks angeboten, um iBeacons zu erstellen oder Signale zu empfangen. In diesem Kapitel soll erläutert werden, wie mithilfe dieser Frameworks iBeacons erstellt werden. Wie dieses Framework zum Empfangen von Signalen verwendet werden kann, wird in Kapitel 8.3 und der Vorstellung der Implementierung des mobilen Client veranschaulicht. Um die Anwendung verwenden zu können, wird ein iPhone, iPad oder iPod benötigt, der Bluetooth 4.0 LE unterstützt. Dazu gehören iPhones ab der Version 4s, iPads ab der 3. Generation und iPods ab der 5. Generation. Auch das aktuelle iPad mini in der ersten Version unterstützt Bluetooth 4.0 LE und ist damit in

Abbildung 8.1: iPhone iBeacon App zur schnellen Konfiguration eines iBeacons

der Lage, als iBeacon konfiguriert zu werden. Ferner wurde für die Implementierung die Entwicklungsumgebung *Xcode* in der Version 5.1.1 verwendet.

Das Kernstück der iPhone iBeacon App ist der `AdvertiseViewController`. Ein Auszug aus diesem Controller ist in Listing 8.1 zu sehen. Nicht abgebildet sind die Life-Cycle-Methoden und die Header-Datei. Der Controller ist dafür zuständig, die iBeacon-Konfiguration vorzunehmen und die View zu erzeugen und zu verwalten, die in Abbildung 8.1 zu sehen ist.

Die Methode `initBeacon` wird immer dann aufgerufen, wenn der Controller aufgerufen wird oder der Nutzer den Button zur Aktualisierung der Daten drückt. In dieser wird der Major- und Minor-Wert aus den jeweiligen Textfeldern extrahiert. Weiter wird eine eindeutige UUID erzeugt. Diese ist die bereits bekannte UUID, die zur Identifikation von iBeacons innerhalb des Krankenhaus-Prozesses verwendet wird. In Zeile 22 wird dann eine `beaconRegion`

```
#import "AdvertiseViewController.h"

@interface AdvertiseViewController ()
@property (strong, nonatomic) CLBeaconRegion *beaconRegion;
@property (strong, nonatomic) NSDictionary *beaconPeripheralData;
@property (strong, nonatomic) CBPeripheralManager *peripheralManager;
@end

@implementation AdvertiseViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    [self initBeacon];
    [self transmitBeacon];
}

- (void)initBeacon {
    int major = _majorLabel.text.intValue;
    int minor = _minorLabel.text.intValue;
    NSUUID *uuid = [[NSUUID alloc] initWithUUIDString:@"8e14411d-205d-3a42-8ea2-8
        feadbdedd79"];
    _beaconRegion = [[CLBeaconRegion alloc] initWithProximityUUID:uuid major:
        major minor:minor identifier:@"de.uni-ulm.ibeacon"];
}

- (void)transmitBeacon{
    _beaconPeripheralData = [_beaconRegion peripheralDataWithMeasuredPower:[
        NSNumber numberWithInt:-53]];
    _peripheralManager = [[CBPeripheralManager alloc] initWithDelegate:self queue
        :nil options:nil];
}

- (void)peripheralManagerDidUpdateState:(CBPeripheralManager *)peripheral {
    if (peripheral.state == CBPeripheralManagerStatePoweredOn) {
        [_peripheralManager startAdvertising:_beaconPeripheralData];
        [self setLabels];
    } else if (peripheral.state == CBPeripheralManagerStatePoweredOff) {
        [_peripheralManager stopAdvertising];
    }
}

- (IBAction)updateIBeacon:(id)sender {
    [_peripheralManager stopAdvertising];
    [self initBeacon];
    [self transmitBeacon];
}

@end
```

Listing 8.1: AdvertiseViewController.m

erstellt, welcher die UUID, Major-Wert, Minor-Wert und einen Identifier zur Identifikation der Region übergeben wird. Um die `beaconRegion` vom Typ `CLBeaconRegion` verwenden zu können, muss das Framework `CoreLocation` importiert werden.

Darauffolgend wird die Methode `transmitBeacon` aufgerufen. In dieser wird ein `NSDictionary` mit den Daten der zuvor erstellten Region befüllt. Gleichzeitig wird die kalibrierte Signalstärke (RSSI) des iBeacons übermittelt. Das so befüllte `NSDictionary` mit dem Namen `beaconPeripheralData` stellt damit den Payload des AD des iBeacons dar. In der zweiten Zeile dieser Methode wird ein `CBPeripheralManager` initiiert. Dieser Manager ist dafür Zuständig, Bluetooth-Services über GATT zu verwalten. Ferner wird dem Manager als `delegate` der eigene Controller übergeben. Dieser muss dazu das Protokoll `CBPeripheralManagerDelegate` implementieren.

Sobald der Manager seinen Status ändert, wird die delegate-Methode `peripheralManagerDidUpdateState` aufgerufen. Falls sich der Status des Managers auf *aktiviert* geändert hat, soll der Manager mit dem Broadcasten des ADs mit dem zuvor erstellten Payload beginnen. Intern erstellt der Manager dazu einen Bluetooth-Service, der über GATT verwaltet wird. Das eigentliche Broadcasten der Informationen des Services, wird über GAP und Bluetooth LE bewerkstelligt (s. Kapitel 3.1).

8.2 Server

Die Implementierung des Servers besteht aus einem *Glassfish Container* der Version 4.0, der den REST-Service und dessen Komponenten beinhaltet und aus einer *MySQL*-Datenbank in der Version 5.6.14, welche die notwendigen Daten beinhaltet. Der REST-Service besteht aus einer *JAX-RS* API, einem Controller und einem *JPA*-Model (*Java Persistence API*). Das JPA-Model wird verwendet, um eine objektorientierte Sicht auf die relationalen Daten der Datenbank zu erhalten. Eine weitere Komponente des REST-Services sind die *JSON-Types*. Diese konvertieren die unterschiedlichen Datentypen und Entitäten der Datenbank in ein JSON-konformes Format, welche die REST-API zum Austausch von Ressourcen verwendet. In diesem Kapitel werden für einige dieser Komponenten Auszüge aus der konkreten Implementierung aufgezeigt und erläutert. Eine Übersicht über die verwendeten Komponenten kann in Abbildung 8.2 eingesehen werden.

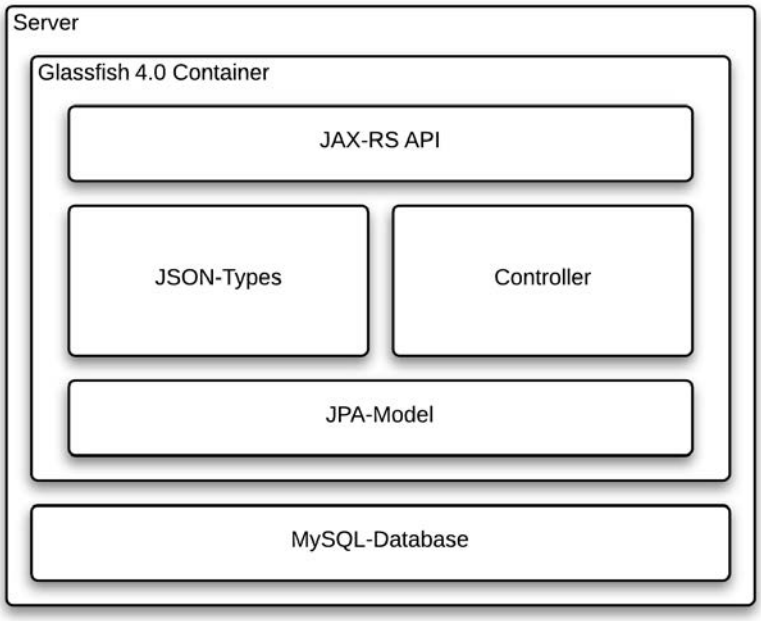

Abbildung 8.2: Komponenten und Aufbau der Implementierung des Servers

8.2.1 JAX-RS API

In Listing 8.2 ist ein Auszug aus dem *PatientService* der JAX-RS API gelistet. Um als REST-Schnittstelle verwendet werden zu können, muss eine Klasse nach dem JAX-RS-Standard zwei Annotationen erhalten. Durch `@Stateless` wird aus der Klasse eine zustandslose *Seasson Bean* generiert. Das bedeutet, dass die Klasse nicht an eine bestimmte Session eines Clients gebunden ist, sondern für alle Clients identisch zur Verfügung steht. Die zweite Annotation, die benötigt wird, ist der Pfad, an dem der Service publiziert wird. In `@Path` wird ein zum Wurzelverzeichnis relativer Pfad angegeben. In diesem Fall ist der Service unter *localhost:8080/MedicalBeacons/patients* aufrufbar. Durch die Annotation `@Inject` wird einer der Controller injiziert, in diesem Fall der `PatientController`. Die Controller sind für Abfragen und Änderungen von Ressourcen zuständig und werden später genauer erläutert. Eine injizierte zustandslose Session Bean wird auf diese Weise automatisch vom Container instanziiert.

Um nun Ressourcen über diese so definierten REST-Schnittstelle abzufragen oder zu erstellen, werden weitere Annotationen und Methoden benötigt. In Zeile 15 wird die Metho-

```
1  package de.uulm.medicalbeacons.services;
2
3  import java.util.List;...
4
5  @Stateless
6  @Path("/patients")
7  public class PatientService {
8
9    @Inject
10   private PatientController patientController;
11
12   @GET
13   @Path("/id/{patientId}")
14   @Produces({ MediaType.APPLICATION_JSON })
15   public Response getPatientChart(@PathParam("patientId") int patientId) {
16     PatientType r = patientController.getPatientChartById(patientId);
17     return Response.ok(r).build();
18   }
19
20   @POST
21   @Path("/id/{patientId}/jobs/medicationjob")
22   @Consumes({ MediaType.APPLICATION_JSON })
23   public Response addMedicationJob(@PathParam("patientId") int patientId,
         MedicationJobType newJob) {
24     patientController.addMedicationJob(patientId, newJob);
25     return Response.created(null).build();
26   }
27 }
```

Listing 8.2: PatientService.java

de `getPatientChart` definiert. Eine Methode in einem REST-Service ist dafür zuständig, Ressourcen zu versenden oder zu empfangen. Durch die Annotation `@GET` wird beschrieben, dass diese Methode bei einem Request mit der HTTP-Methode GET aufgerufen werden soll. Ferner soll die Methode aufgerufen werden, wenn die relative URI */patients/id/{patientId}* angefordert wird, was durch die Annotation `@Path` in Zeile 13 beschrieben wird. Da patientId in der Annotation in geschweiften Klammen steht, wird patientId als Parameter definiert. D.h. in der URI zur angeforderten Ressource muss die Id des Patienten angegeben werden. Valide relative Pfade zu einer Ressource wären also z.B. */patients/id/1* oder */patients/id/4711*. Dieser annotierte Parameter kann auf einen Parameter der Methodendefinition projiziert werden. Hierzu wird vor den gewünschten Parameter in der Methode die Annotation `@PathParam` mit dem gewünschten Parameter aus der URI gesetzt. Damit kann der Parameter aus der URI innerhalb der Methode verwendet werden. Durch die Typi-

sierung des Parameters in der Methodendefinition ist es zusätzlich möglich, falsche Eingaben zu filtern (wie z.B. Buchstaben). Innerhalb der Methode wird in Zeile 16 der Controller verwendet, um den Patienten mit der übergeben Id zu finden und daraus einen JSON-Type zu erstellen. Dieser JSON-Type kann dann automatisch durch das return-Statement in einen JSON-String konvertiert werden und in den Body der HTTP-Response abgelegt werden, indem die Methode zusätzlich mit `@Produces` in Zeile 14 und dem entsprechenden Typ annotiert wird. Soll anstatt JSON eine XML generiert werden, so muss der Typ in Zeile 14 ausgetauscht werde. Wichtig ist hierbei, dass die JSON-Types eine bestimmte Klassenstruktur vorweisen, die weiter unten genauer erläutert wird.

Die Methode in Zeile 23 wird verwendet, um neue Ressourcen hinzuzufügen, in diesem Fall Medikationen. Um Ressourcen empfangen zu können und sie an einer bestimmten URI erstellen zu können, muss die Methode ebenfalls mit Annotationen versehen werden. Die Annotation `@POST` markiert die Methode in der Weise, dass sie durch einen HTTP-POST-Request an der in `@Path` angegebenen URI aufgerufen wird. Zusätzlich erhält die Methode die Annotation `@Consumes`, durch die bestimmt wird, welche Art von Typ die Methode empfangen kann. Einhergehend mit der Annotation muss in der Methodendefinition ein Parameter mit dem passenden Typ angegeben sein. In diesem Fall der `MedicationJobType`-Typ. Die Ressource im Body des POST-Requests wird dann in diesen Parameter gekapselt und ist somit innerhalb der Methode verwendbar. In Zeile 24 wird die Ressource dann mithilfe des Controllers in der Datenbank abgelegt. Durch das return-Statement in Zeile 25 kann dem Client durch die unterschiedlichen HTTP-Status-Codes mitgeteilt werden, ob der Request erfolgreich war oder ob Fehler aufgetreten sind.

8.2.2 JSON-Types

Die JSON-Types werden vom REST-Service und dessen Controllern verwendet, um aus den Ressourcen der Datenbank JSON-Strings zu generieren. Außerdem werden sie benötigt, um aus empfangenen und vom Client via POST oder PUT versendete JSON-Strings in Java-Objekte umzuwandeln. Für jede Ressource, mit welcher der REST-Service umgehen können soll, muss ein entsprechender JSON-Type definiert werden. JSON-Types sind im Grunde nichts anderes als simple *POJO*s (*Plain Old Java Object*), die aus privaten Attributen, Getter- und Setter-Methoden und einem Standard-Konstruktor bestehen. Um aus diesen POJOs JSON-Strings oder XML-Dokumente automatisiert generieren zu lassen, müssen diese mit Annotationen versehen werden.

```
1  package de.uulm.medicalbeacons.datatypes;
2
3  import javax.xml.bind.annotation.XmlRootElement;
4  import javax.xml.bind.annotation.XmlType;
5
6  @XmlRootElement
7  @XmlType(propOrder = { "drugId", "name"})
8  public class DrugType {
9
10   private int drugId;
11   private String name;
12
13
14   public int getDrugId() {
15     return drugId;
16   }
17
18   public void setDrugId(int drugId) {
19     this.drugId = drugId;
20   }
21
22   public String getName() {
23     return name;
24   }
25
26   public void setName(String name) {
27     this.name = name;
28   }
29 }
```

Listing 8.3: DrugType.java

In Listing 8.3 ist ein solcher JSON-Type für Medikamente dokumentiert. Wie zu erkennen ist, besteht diese Klasse nur aus Attributen und Zugriffsmethoden. Oberhalb der Klassendefinition in Zeile 6 muss die Annotation `XmlRootElement` angegeben werden. Durch diese Annotation weiß der Container, dass dieses POJO in JSON-Strings oder XML-Dokumente umgewandelt werden kann. Durch die optionale Annotation `XmlType` mit dem Parameter `propOrder` kann die Reihenfolge der Attribute im generierten Dokument erzwungen werden. In diesem Fall soll zuerst die `drugId` und dann `name` im generierten Dokument gelistet werden.

8.2.3 Controller

Die Controller der REST-Services sind dafür zuständig, Ressourcen aus der Datenbank abzufragen, zu ändern oder durch einen Client übergebene in der Datenbank abzulegen.

Auch die Überprüfung von Statusänderungen von Medikationen und fachärztlichen Untersuchungen seitens der Anwender wird innerhalb der Controller durchgeführt. Um einen Überblick über die unterschiedlichen Aufgaben der Controller zu erhalten, werden im Folgenden drei Beispiele von Implementierungen in Controllern und deren Anwendungsfälle vorgestellt.

```
1  @Stateless
2  public class EmployeeController {
3
4    @PersistenceContext(unitName="MedicalBeacons")
5    private EntityManager em;
6
7    public List<EmployeeType> getAllEmployees() {
8      List<Employees> employees = getAllEmployeeEntities();
9      if(employees == null) {
10       return null;
11     }
12
13     List<EmployeeType> responses = new ArrayList<EmployeeType>();
14     for(Employees employee : employees) {
15       EmployeeType response = createEmployeeResponse(employee);
16       responses.add(response);
17     }
18     return responses;
19   }
20
21   private List<Employees> getAllEmployeeEntities() {
22     Query q = em.createQuery("select m from Employees m");
23     if(q.getResultList().size() < 1) {
24       return null;
25     }
26
27     List<Employees> employees = new ArrayList<Employees>();
28     for(Object o : q.getResultList()){
29       employees.add((Employees)o);
30     }
31     return employees;
32   }
33 ...
34 }
```

Listing 8.4: EmployeeController.java

In Listing 8.4 ist ein Auszug eines übersichtlichen Controllers dargestellt, der für das Abfragen von Mitarbeitern des Krankenhauses aus der Datenbank mithilfe des JPA-Models verwendet wird. Letzteres wird über die Annotation in Zeile 4 in die Variable in Zeile 5

injiziert. Über einen `EntityManager` können Queries durchgeführt werden und in Klassen des JPA-Models überführt werden. Dazu muss in der Annotation das entsprechende JPA-Modell angegeben werden. Dieses lässt sich im Glassfish Container konfigurieren und verweist auf eine spezifische Datenbank. In der Methode `getAllEmployeeEntities` in Zeile 25 ist das Abfragen aus der Datenbank zu erkennen. In Zeile 26 wird ein Query auf dem `EntityManager` und dessen JPA-Model erstellt, das einen ähnlichen Aufbau wie SQL besitzt. In Zeile 27 wird dieses Query ausgeführt. Falls die Abfrage mindestens ein Element besitzt, so wird ab Zeile 31 jedes dieser Elemente durchlaufen und in eine Klasse des JPA-Models umgewandelt (s. Zeile 33). In der Methode `getAllEmployees`, welche unter anderem von der JAX-RS API heraus aufgerufen wird, um die Ressourcen abzufragen, werden die Objekte des JPA-Models in JSON-Types (erkennbar an der Endung Types im Klassennamen) in Zeile 15 umgewandelt. Dies geschieht im Grunde über das Kopieren der Attribute vom JPA-Objekt hin zum JSON-Objekt.

In Listing 8.5 ist die Methode `updateSpecialistJobs()` aus dem `JobController` abgebildet. Diese Methode wird vom REST-Service aus aufgerufen, falls ein Krankenhausmitarbeiter eine Statusänderung an einer fachärztlichen Untersuchung vorgenommen hat. In dieser Methode wird in Zeile 6 festgestellt, ob sich der Status wirklich geändert hat. Weiter sind für fachärztliche Untersuchungen sieben verschiedene Status definiert. Deswegen wird in Zeile 10 überprüft, ob sich der Identifikator der Aufgabe innerhalb des Intervalls befindet. Wenn der Status von einem laufenden Status auf einen offenen Status zurückgesetzt werden soll, so wird in Zeile 17 der Mitarbeiter aus der Aufgabe entfernt, der sie davor gestartet hatte. Dies muss durchgeführt werden, damit sie wieder von anderen Krankenhausmitarbeitern eingesehen und folglich gestartet werden kann. Eine Anforderung war nämlich, dass nur offene Aufgaben oder vom gerade angemeldeten Mitarbeiter gestartete angezeigt werden.

Eine fachärztliche Untersuchung lässt sich in drei Teilaufgaben einteilen. Der Status mit 1 definiert, dass die Aufgabe offen ist. 3 definiert, dass der Patient zum Facharzt verlegt wurde und damit nun offen für den Facharzt und die Untersuchung ist. 5 bedeutet, dass die Untersuchung abgeschlossen wurde und nun offen für die Rückverlegung ist (siehe dazu auch Kapitel 7.6). Handelt es sich beim neuen Status um einen anderen Wert, so wurde die Aufgabe oder eine Teilaufgabe der fachärztlichen Untersuchung begonnen, wodurch der Krankenhausmitarbeiter, der die Aufgabe gerade begonnen hat, in das entsprechende Feld eingetragen wird. In Zeile 22 wird letztendlich noch die An- oder Abwesenheit des Patienten anhand des neuen Status gesetzt. Dadurch ist es dem Stationsarzt möglich einzusehen,

```
1  public SpecialistJobType updateSpecialistJob(SpecialistJobType job) {
2      SpecialistJobs old = em.find(SpecialistJobs.class, job.getJobId());
3      if(old == null) {
4        return null;
5      }
6      if(old.getState().getStateId() == job.getState().getStateId()) {
7        return null;
8      }
9
10     if(job.getState().getStateId() < 1 || job.getState().getStateId() > 7) {
11       return null;
12     }
13
14     old.setState(em.find(SpecialistJobStates.class, job.getState().getStateId()))
           ;
15
16     if(job.getState().getStateId() == 1 || job.getState().getStateId() == 3 ||job
           .getState().getStateId() == 5) {
17       old.setStartedBy(null);
18     } else {
19       old.setStartedBy(em.find(Employees.class, job.getStartedBy().getEmployeeId
             ()));
20     }
21
22     switch (job.getState().getStateId()) {
23       case 2:
24       case 3:
25       case 4:
26       case 5:
27       case 6:
28         old.getPatient().setIsInRoom(0);
29         break;
30       case 1:
31       case 7:
32         old.getPatient().setIsInRoom(1);
33         break;
34     }
35
36     em.persist(old);
37     return createSpecialistJobResponse(old);
38   }
```

Listing 8.5: JobController.java: updateSpecialistJob

welche Patienten an- oder abwesend sind. Auch dies ist eine definierte Anforderung. In Zeile 36 wird die fachärztliche Untersuchung noch über den `EntityManager` aktualisiert, wodurch der Status in der Datenbank abgeändert wird.

In Listing 8.6 ist die Methode `updateMedicationJobs()` aus dem `JobController` abgebildet. Diese wird aufgerufen, wenn der Status einer Medikation geändert werden soll. Diese Methode funktioniert ähnlich wie die zum Aktualisieren einer fachärztlichen Untersuchung. Besonderes Augenmerk liegt hierbei aber auf der Aktualisierung des Medikamentenbestandes. Dazu wird in Zeile 19 überprüft, ob der neue Status den Identifikator 3 aufweist, was bedeutet, dass die Aufgabe abgeschlossen wurde. Weiter wird überprüft, ob das Zimmer des iBeacons, der laut Anforderung beim Abschließen einer Medikation berührt werden muss, wirklich ein Medikamentenlager ist (Medikamentenlager haben die Zimmernummern 89 bis 99). Falls dies zutrifft, wird das Medikamentenlager über den `EntityManager` aus der Datenbank extrahiert. In Zeile 27 wird folglich der Medikamentenbestand des Medikamentenlagers anhand der in der Medikation angegebenen Medikamente und deren Anzahl aktualisiert und weiter unten in Zeile 44 letztlich in der Datenbank angepasst.

8.2.4 JPA-Model

Durch JPA lässt sich eine objektorientierte Sicht auf eine relationale Datenbank erstellen. Dazu gibt es grundsätzlich zwei Herangehensweisen. Bei der einen wird zunächst das JPA-Modell mit dessen Klassen und Assoziationen erstellt. Aufgrund der Klassenstruktur kann dann die zugrundeliegende relationale Struktur der Datenbank generiert werden. Die andere Herangehensweise basiert darauf, eine Klassenstruktur auf einer bereits existierenden relationalen Datenstruktur aufzubauen. Letztere wurde während der Implementierung von Medical Beacons durchgeführt. Wie dabei vorgegangen wird und wie die Klassen eines JPA-Models aufgebaut sind, wird im Folgenden dargestellt.

In Listing 8.7 ist die Klasse eines Medikamentes abgebildet. Sie ist das objektorientierte Abbild der dazugehörigen Tabelle der relationalen Datenbank. Da im Falle von Medical Beacons die Datenbank bereits existiert und das JPA-Model darauf aufgesetzt wird, müssen einige Dinge beachtet werden. Zunächst muss der Klassenname mit dem Namen der korrespondierenden Tabelle übereinstimmen. Durch `@Entity` in Zeile 8 wird die Klasse als Entität markiert, wodurch zur Laufzeit alle Instanzen dieser Klasse auf die Tabelle innerhalb der Datenbank abgebildet werden. Mit der Annotation in Zeile 11 wird der *Primary Key* gesetzt. Auch dieser muss denselben Namen und denselben Typ wie in der Tabelle vorweisen. Durch Zeile 12 kann konfiguriert werden, dass der Primary Key automatisiert generiert und beim Einfügen einer neuen Entität automatisch inkrementiert wird. Alle anderen Attribute, im Beispiel das Attribut `name`, müssen mit gleichem Namen und gleichem Typ in der

```
public MedicationJobType updateMedicationJob(MedicationJobType job, int station,
    int room) {
    MedicationJobs old = em.find(MedicationJobs.class, job.getJobId());
    if(old == null) {
      return null;
    }

    if(job.getState().getStateId() == old.getState().getStateId()) {
      return null;
    }

    if(job.getState().getStateId() < 1 || job.getState().getStateId() > 3) {
      return null;
    }

    if(job.getState().getStateId() == 3 && (station < 1 || room < 1)) {
      return null;
    }

    if(job.getState().getStateId() == 3 && (station > 0 && room > 89)) {
      Stores store = em.createQuery("select s from Stores s where s.floorNumber=:
          station and s.roomNumber=:roomId", Stores.class)
          .setParameter("station", station)
          .setParameter("roomId", room)
          .getSingleResult();
      if(store == null) {
        return null;
      }
      for(MedicationJobDrugs drug : old.getMedicationJobDrugs()) {
        em.createQuery("update StoreHasDrugs s set s.amount = (s.amount- :amount)
            where s.store.storeId = :storeId and s.drug.drugId=:drugId")
          .setParameter("storeId", store.getStoreId())
          .setParameter("drugId", drug.getDrug().getDrugId())
          .setParameter("amount", drug.getAmount())
          .executeUpdate();
      }
    }

    old.setState(em.find(MedicationJobStates.class, job.getState().getStateId()))
        ;

    if(job.getState().getStateId() == 1) {
      old.setStartedBy(null);
    } else {
      old.setStartedBy(em.find(Employees.class, job.getStartedBy().getEmployeeId
          ()));
    }

    em.persist(old);

    return createMedicationJobResponse(old);
  }
```

Listing 8.6: JobController.java: updateMedicationJob

Tabelle vorkommen. Da die Klassen eines JPA-Models ähnlich wie die JSON-Types POJOs darstellen müssen, werden zu jedem Attribut die passenden Setter- und Getter-Methoden und ein Standard-Konstruktor bereitgestellt.

```
1  package de.uulm.medicalbeacons.model;
2
3  import javax.persistence.Entity;
4  import javax.persistence.GeneratedValue;
5  import javax.persistence.GenerationType;
6  import javax.persistence.Id;
7
8  @Entity
9  public class Drugs {
10
11   @Id
12   @GeneratedValue(strategy=GenerationType.IDENTITY)
13   private int drugId;
14   private String name;
15
16
17   public int getDrugId() {
18     return drugId;
19   }
20
21   public void setDrugId(int drugId) {
22     this.drugId = drugId;
23   }
24
25   public String getName() {
26     return name;
27   }
28
29   public void setName(String name) {
30     this.name = name;
31   }
32 }
```

Listing 8.7: Drugs.java: updateMedicationJob

Da Tabellen und Entitäten in einer relationalen Datenbank auch Beziehungen zueinander haben, müssen diese Ebenfalls in den Klassen des JPA-Modells modelliert werden. In Listing 8.8 ist eine *One-To-One*-Relation zu sehen, die über die Annotation `@OneToOne` erstellt werden kann. Dabei besteht eine Relation zwischen einer Medikation und einem Krankenhausmitarbeiter, der die Aufgabe gestartet hat. Über die Annotation `@JoinColumn` wird der dafür notwendige Fremdschlüssel und dessen Name angegeben, der mit dem Pri-

märschlüssel der anderen Tabelle verknüpft ist (in diesem Fall der Primärschlüssel eines Krankenhausmitarbeiters).

In Listing 8.9 ist eine *One-To-Many*-Relation dargestellt. Um eine solche Relation zu realisieren, wird ein Attribut, dass eine Liste sein muss, mit `@OneToMany` markiert. Innerhalb der Annotation wird beschrieben, dass die Relation über den Fremdschlüssel `medicationJob` der anderen Klasse oder Tabelle die einer Medikation zugehörigen Medikamente erhält. Besonderes Augenmerk liegt hierbei auf `fetch=FechType.LAZY`. Durch diesen Parameter wird die Performanz des JPA-Modells stark verbessert, indem die dazugehörigen Medikamente erst dann aus der Datenbank abgefragt werden, wenn diese durch einen Zugriff auf `medicationJobDrugs` benötigt werden. Ansonsten werden nur Attribute einer Medikation abgefragt, ohne ihre Relationen nachzuverfolgen. In Listing 8.10 ist der umgekehrte Fall einer *Many-To-One* Relation abgebildet. Bei einer solchen wird ebenfalls in der `@ManyToOne`-Annotation angegeben, dass die Relationen erst bei Benutzung derselbigen aus der Datenbank abgefragt werden. In `@JoinColumn` wird angegeben, über welchen Fremdschlüssel der Klasse die Relation hergestellt werden soll (in diesem Fall `jobId`). Wie bei diesen Relationen zu erkennen ist, bestehen sie ebenfalls aus Klassen und Objekten, was den Umgang mit Relationen über objektorientierte Methoden stark vereinfacht.

```
1 @OneToOne
2 @JoinColumn(name="startedBy")
3 private Employees startedBy;
```

Listing 8.8: @OneToOne

```
1 @OneToMany(mappedBy="medicationJob", fetch=FetchType.LAZY)
2 private List<MedicationJobDrugs> medicationJobDrugs;
```

Listing 8.9: @OneToMany

```
1 @ManyToOne(fetch=FetchType.LAZY)
2 @JoinColumn(name="jobId")
3 private MedicationJobs medicationJob;
```

Listing 8.10: @ManyToOne

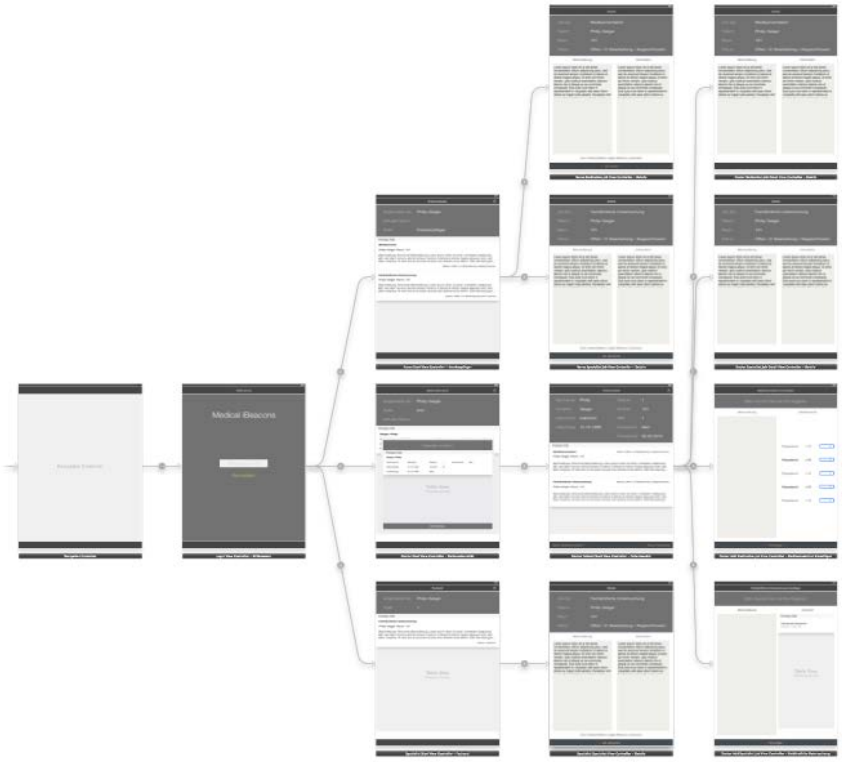

Abbildung 8.3: Storyboard der mobilen Anwendung

8.3 Mobiler Client

Die Implementierung des mobilen Clients der Context Aware Application wurde auf einem iPad der dritten Generation und iOS in der Version 7.1 durchgeführt. Es wurden ausschließlich die von Apple zur Verfügung gestellten Frameworks und Bibliotheken für den Umgang mit iBeacons, der Kommunikation mit dem Server und der Generierung von JSON-Strings verwendet. Weiter wurde die von Apple bereitgestellte Entwicklungsumgebung Xcode in der Version 5.1.1 benutzt. Die Benutzerschnittstelle der mobilen Anwendung wurde mithilfe der in Xcode bereitgestellten Storyboards implementiert. Das resultierende Storyboard ist in Abbildung 8.3 abgebildet. Da das Herzstück der mobilen Anwendung das Empfangen von Signalen von iBeacons und Verarbeitung jener ist, wird in diesem Kapitel die Implementierung dieser Konzepte kurz vorgestellt.

8.3.1 Empfangen von Signalen

In Listing 8.11 ist der `BeaconRegionMonitor` der mobilen Anwendung zu sehen. Dieser Entspricht dem im Entwurf vorgestellten iBeaconService und ist für das Empfangen von Signalen, der Zuordnung der Signale und einer ersten Bearbeitung dieser zuständig. Um Signale und deren Payloads von iBeacons zu empfangen, werden Funktionen aus dem bereitgestellten Framework `CoreLocation` verwendet [20]. Das Framework beinhaltet einen `CLLocationManager`, der anhand einer übergebenen Region nach angegebenen iBeacons suchen und deren Signale an die Anwendung weitergeben kann. Die Region beinhaltet eine UUID und einen Identifier, der nur zur internen Verwaltung und eventuellen Differenzierung mehrerer Regionen benötigt wird. Die UUID muss mit der UUID der iBeacons im Krankenhaus übereinstimmen. Denn nur iBeacons mit bekannten UUIDs werden an die Anwendung weitergegeben. Dieses Vorgehen und das anschließenden Starten des `LocationManagers` ist in der Methode `initRegion` einzusehen.

Das Starten des `LocationManagers` bewirkt vorerst nur, dass die Anwendung informiert wird, wenn ein iBeacon mit der passenden UUID in der Umgebung gefunden wurde. Erst durch die delegate-Methode in Zeile 17, nachdem das Starten des `LocationManagers` bestätigt wurde, wird durch die Methode `startRangingBeaconsInRegion` angegeben, dass er die gefundenen iBeacons der definierten Region weiter bearbeiten und insbesondere deren Entfernungen über den im Payload angegeben RSSI-Wert berechnen soll.

Sobald der `LocationManager` die Entfernungen zu bekannten, in der in der Nähe befindlichen iBeacons berechnet hat, wird die Methode `didRangeBeacons` in Zeile 21 aufgerufen. Dieser werden die gefundenen iBeacons samt derer Daten als Parameter übergeben. Die so der Anwendung übermittelten iBeacons werden in dieser Methode gefiltert und sortiert. So werden in Zeile 25 alle iBeacons herausgefiltert, die eine unbekannte und nicht berechenbare Entfernung aufgrund von möglichen Störquellen aufweisen. In Zeile 31 und 32 werden die übriggeblieben iBeacons mit kalkulierten Entfernungen aufsteigend sortiert. In Zeile 34 wird überprüft, ob sich unter den iBeacons ein solcher befindet, zu dem direkter Kontakt besteht. Ist dies der Fall, so wird das dem delegate des `BeaconRegionMonitors` mitgeteilt. In Zeile 42 werden dem delegate zusätzlich alle gefunden iBeacons übermittelt. Das delegate ist in diesem Fall einer der Controller der mobilen Anwendung, der anhand der übermittelten iBeacons einen Kontext bestimmen kann und Anwendungsfälle umsetzt.

```objc
@implementation BeaconRegionMonitor
-(id)init {
    if(self = [super init]) {
        _locationManager = [[CLLocationManager alloc] init];
        _locationManager.delegate = self;
        [self initRegion];
    }
    return self;
}

- (void)initRegion {
    NSUUID *uuid = [[NSUUID alloc] initWithUUIDString:@"8e14411d-205d-3a42-8ea2-8feadbdedd79"];
    _beaconRegion = [[CLBeaconRegion alloc] initWithProximityUUID:uuid identifier:@"de.uni-ulm.ibeacon"];
    [_locationManager startMonitoringForRegion:_beaconRegion];
}

- (void)locationManager:(CLLocationManager *)manager didStartMonitoringForRegion:(CLRegion *)region {
    [_locationManager startRangingBeaconsInRegion:_beaconRegion];
}

- (void)locationManager:(CLLocationManager *)manager didRangeBeacons:(NSArray *)beacons inRegion:(CLBeaconRegion *)region {
    NSMutableArray *tmp = [NSMutableArray arrayWithArray:beacons];
    _rangedBeacons = [NSMutableArray array];

    for(CLBeacon *beacon in tmp) {
        if (beacon.proximity != CLProximityUnknown) {
            [_rangedBeacons addObject:beacon];
        }
    }

    NSSortDescriptor *sortDescriptor1 = [[NSSortDescriptor alloc] initWithKey:@"accuracy" ascending:YES];
    [_rangedBeacons sortUsingDescriptors:@[sortDescriptor1]];

    for(CLBeacon *beacon in _rangedBeacons) {
        if (beacon.proximity == CLProximityImmediate) {
            if([(NSObject*)_delegate respondsToSelector:@selector(didRangeImmediateBeacon:)]) {
                [_delegate didRangeImmediateBeacon:beacon];
            }
            break;
        }
    }
    if([(NSObject*)_delegate respondsToSelector:@selector(didRangeBeacons:)]) {
        [_delegate didRangeBeacons:beacons];
    }
}
@end
```

Listing 8.11: BeaconRegionMonitor.m

Die einzelnen Controller, deren Anwendungsfälle und Kontexte werden in Kapitel 10 anhand von Screenshots der resultierenden Anwendung vorgestellt. Um als delegate des `BeaconRegionMonitors` fungieren zu können, muss das in Listing 8.12 gelistete Protokoll implementiert werden.

```
1 @protocol BeaconRegionMonitorDelegate
2 @optional
3 - (void) didRangeBeacons:(NSArray *)beacons;
4 - (void) didRangeImmediateBeacon:(CLBeacon *)beacon;
5 @end
```

Listing 8.12: BeaconRegionMonitorDelegate

8.3.2 Verwendung von Signalen

In Listing 8.13 ist die Implementierung des genannten Protokolls innerhalb des `DoctorStartViewControllers` dargestellt. Der Controller ist dafür zuständig, einem Stationsarzt Patientenlisten der aktuellen Station, Patientenlisten von Krankenzimmern oder Patientenakten anzuzeigen. Dazu werden die im Protokoll definierten Methoden wie folgt implementiert. Wird der Controller über `didRangeBeacons` über umliegende iBeacons informiert, welche aufsteigend nach deren Entfernung übergeben werden, so wird zuerst überprüft, ob die Liste der iBeacons überhaupt befüllt ist. Weiter wird ein globaler `NetworkController` auf dessen Bereitschaft überprüft. Ist letzterer momentan nicht beschäftigt und die Liste der iBeacons besitzt mindestens einen iBeacon, so wird in Zeile 12 die aktuelle Station ermittelt. Dazu wird der Major-Wert des ersten iBeacons in der Liste, der aufgrund der Sortierung der am nächsten gelegene ist, extrahiert und anhand des Wertes die Station ermittelt. Wenn es sich dabei um eine korrekte und neue Station handelt, so werden die Patienten am Server abgefragt, die auf der aktuellen Station liegen.

Der Controller wird vom `BeaconRegionMonitor` über berührte oder wenige Zentimeter entferne iBeacons durch die Methode `didRangeImmediateBeacon` notifiziert. Über sie erhält der Controller die Informationen zum iBeacon. In den Zeilen 26 bis 28 werden die Station des iBeacons, dessen Raumnummer und Bettnummer aus dem Payload extrahiert. Nun werden anhand einer Fallunterscheidung zwei Anwendungsfälle unterschieden. Ist die Bettnummer des iBeacons keine Null, so wurde ein iBeacon eines Patienten erkannt. Aus diesen Grund wird folglich die Patientenakte des Patienten am Server abgefragt und dem

Stationsarzt angezeigt. Ist die Bettnummer eine Null, so ist aufgrund der definierten Kodierung von iBeacons bekannt, dass ein iBeacon eines Zimmers berührt wurde. Dementsprechend werden die Patienten aus dem dazugehörigen Krankenzimmer am Server abgefragt.

```
- (void) didRangeBeacons:(NSArray *)beacons {
    if(beacons.count < 1)  {
        [self clearPatientsOfStation];
        _stationLabel.text = @"Keine Station gefunden.";
        return;
    }

    else if(_networkController.isBusy) {
        return;
    }

    int station = ((CLBeacon *)beacons.firstObject).major.intValue / 100;

    if(_currentStation == -1 || _currentStation != station) {
        _currentStation = station;
        _stationLabel.text = [NSString stringWithFormat:@"%i", _currentStation];
        [self fetchPatientsOfStation:_currentStation];
    }
}

- (void) didRangeImmediateBeacon:(CLBeacon *)beacon {
    if (_networkController.isBusy) {
        return;
    }

    int station = beacon.major.intValue / 100;
    int room = beacon.major.intValue;
    int bed = beacon.minor.intValue;

    if (bed != 0) {
        [self fetchPatientOfStation:station room:room andBed:bed];
    }
    else {
        [self fetchPatientsOfStation:station room:room];
    }
}
```

Listing 8.13: DoctorStartViewController.m

Anhand dieses Beispieles lässt sich nochmal der Zusammenhang zwischen Anwendungsfall und Kontext zur Ausführung einer bestimmten Aktion erläutern. Über die unterschiedlichen Controller der mobilen Anwendung werden Anwendungsfälle abgebildet und teilweise gebündelt. Der gerade angesprochene `DoctorStartViewController` implementiert An-

wendungsfälle, um Patientenlisten der aktuellen Station, Patientenlisten von Krankenzimmern und Patientenakten aufzurufen und anzuzeigen. Der Kontext wird bestimmt, indem zwischen den Interaktionen mit iBeacons differenziert wird. Ein direkter Kontakt bzw. eine Berührung mit einem iBeacon führt dazu, dass die Bettnummer des iBeacons überprüft wird. Ist diese Nummer eine Null, so kann der Kontext dahingehend bestimmt werden, dass eine Berührung mit einem iBeacon stattgefunden hat, dieser iBeacon ein Zimmer identifiziert und sich der Benutzer innerhalb der oben beschrieben Anwendungsfälle befindet. Dadurch weiß nun die Context Aware Application, dass der Stationsarzt die Patientenliste eines Krankenzimmers einsehen möchte und führt diese Aktion automatisch durch.

Die Kontextbestimmung und die Kommunikation über das oben beschriebene Protokoll findet so in jedem Controller innerhalb des mobilen Clients statt. Wobei die Kontextbestimmung natürlich je nach Anwendungsfall und Interaktion mit iBeacons angepasst werden muss. Bevor die implementierten Anwendungsfälle und deren Kontextbestimmungen und Interaktionen in Kapitel 10 vorgestellt werden, soll im nächsten Kapitel die Idee der Verbindung mit einem Workflowmanagementsystem erläutert werden.

9 | Verbindung mit Workflowmanagementsystem

In diesem Kapitel wird Idee die vorgestellt, Medical Beacons und insbesondere den Server mit einem Workflowmanagementsystem (WfMS) zu verbinden. Hierzu wird nach der Vorstellung der Idee der daraus resultierende Entwurf vorgestellt. Ferner werden Probleme erläutert, zu denen es während der Implementierung und Anbindung des WfMS an Medical Beacons kam.

9.1 Idee

In Kapitel 6 während der Anforderungsanalyse und in Kapitel 7.6, in der das Datenmodell von Medical Beacons vorgestellt wurde, wurde beschrieben, dass Medikationen und fachärztliche Untersuchungen unterschiedliche Status haben. Diese Status sagen aus, ob eine dieser Aufgaben auf einer Aufgabenliste angezeigt werden soll und welche Organisationseinheit die entsprechende Aufgabe beginnen, ausführen oder abschließen kann. Bisweilen werde solche Überprüfungen nur manuell durch Programmcode durchgeführt. Für Szenarien, in denen eine Reihe von Aufgaben nacheinander und von unterschiedlichen Organisationseinheiten begonnen, durchgeführt und abgeschlossen werden sollen, eignen sich besonders gut WfMS. Durch solche ist es einem Entwickler möglich, einen Prozess mit Aufgaben (Aktivitäten) zu modellieren und ihnen Organisationseinheiten zuzuordnen, die für die Ausführung der jeweiligen Aufgabe berechtigt sind. Zusätzlich können Aufgabenlisten automatisch von einem WfMS generiert werden, was bis hier hin ebenfalls lediglich über Programmcode geschieht.

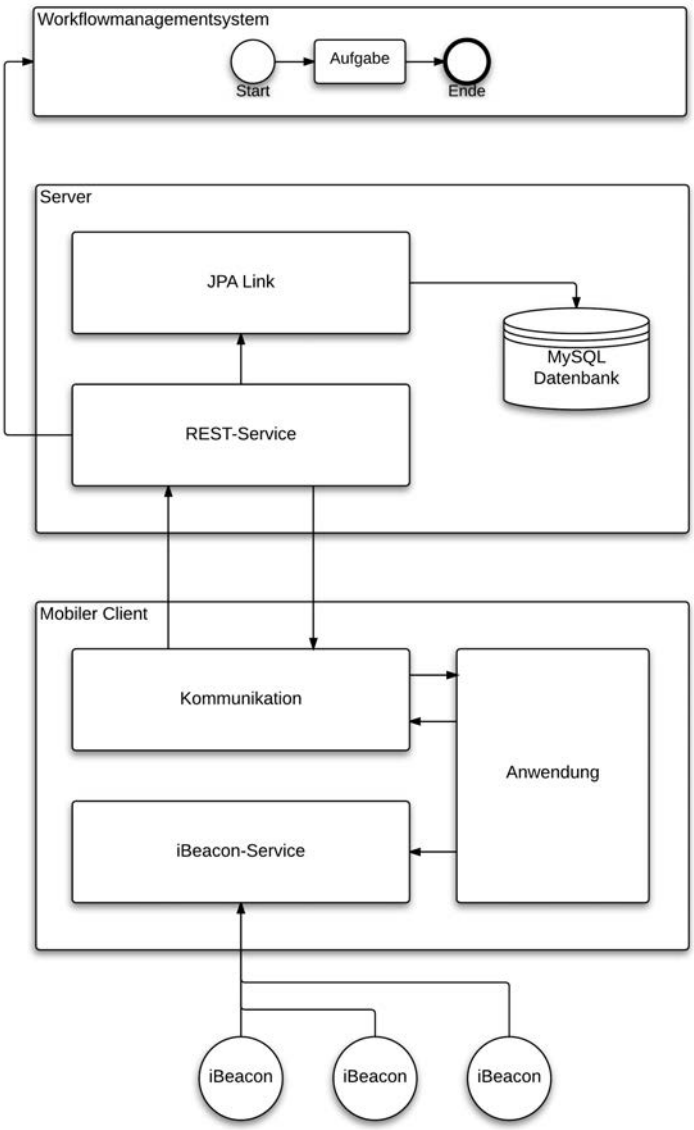

Abbildung 9.1: Die um ein Workflowmanagementsystem erweiterte Architektur von Medical Beacons

Die Idee ist es nun, ein WfMS an den REST-Service über SOAP und *JAX-WS* anzubinden. Dadurch entfällt das manuelle Setzen von Status der einzelnen Aufgaben und die Überprüfung von berechtigten Organisationseinheiten innerhalb der einzelnen Controller des REST-Services. In Abbildung 9.1 ist der bereits bekannte Aufbau von Medical Beacons

und die Erweiterung um ein WfMS dargestellt. Wie zu erkennen ist, soll die Erweiterung nur den REST-Service selbst betreffen. Alle anderen Komponenten bleiben von ihr unberührt, sodass vor allem der mobile Client weiterhin die leichgewichtige Schnittstelle des REST-Service verwenden kann.

9.2 Entwurf

Für das Design und den Versuch der Implementierung wurde als WfMS *Aristaflow* verwendet, das im Laufe von Forschungsarbeiten an der Universität Ulm entwickelt wurde und mittlerweile im kommerziellen Einsatz ist [4, 21, 22, 75].

Aristaflow bietet ein Werkzeug für die Modellierung von Organisationseinheiten an. Die bereits in der Datenbank befindlichen Krankenhausmitarbeiter aus den Organisationseinheiten der Stationsärzte, Krankenpfleger und Fachärzte wurden mit ihren Primärschlüsseln direkt übernommen. Der Primärschlüssel wird benötigt, um bereits zugeordnete Aufgaben nicht zu verlieren.

In Abbildung 9.2 ist der in Aristaflow modellierte und entwickelte Krankenhaus-Prozess dargestellt. Er besteht aus mehreren Aktivitäten und den dazugehörigen Organisationseinheiten. Eine Instanz des Prozesses kann von einem Stationsarzt erstellt und gestartet werden. Nur der Stationsarzt, der die Instanz gestartet hat, kann die erste Aktivität *Job hinzufügen* starten und ausführen. Anstatt den kompletten REST-Service zu verändern und Aristaflow alle Parameter einer neuen Aufgabe zu übergeben, wird die Aufgabe weiterhin innerhalb des REST-Services und durch das JPA-Modells erstellt. Nur der dadurch generierte Identifikator, d.h. der Primärschlüssel, und die Art der Aufgabe, d.h. Medikation oder fachärztliche Untersuchung, wird dem Prozess in Aristaflow übergeben. Die nächste Aktivität wird automatisch ausgeführt und es wird überprüft, welche Art von Aufgabe hinzugefügt wurde. Anhand dieser Entscheidung werden unterschiedliche Prozessabläufe generiert. Handelt es sich um eine Medikation, so wird die Aktivität *Medikation durchführen* aktiviert. Sie ist nur von Mitgliedern der Organisationseinheit der Krankenpfleger ausführbar. Andererseits wird die Aktivität *Patienten hinbringen* aktiviert. Sie ist ebenfalls nur von Krankenpflegern ausführbar. Ist diese Aktivität abgeschlossen wird *Untersuchung durchführen* aktiviert, die nur von Fachärzten ausgeführt werden kann. Danach folgt die Aktivität *Patienten zurückbringen*, was wiederum nur von einem Krankenpfleger ausgeführt werden kann. Sobald

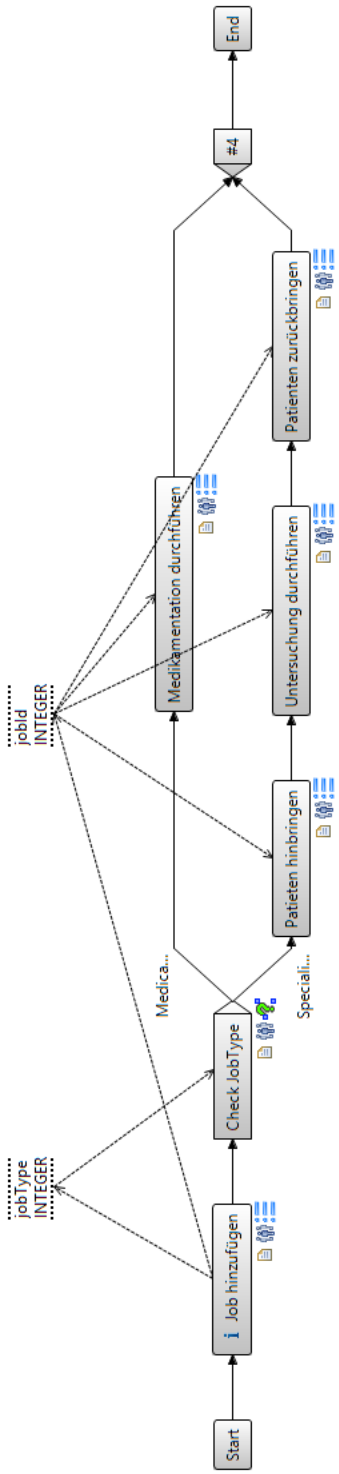

Abbildung 9.2: In Aristaflow modellierter Prozess von Medical Beacons

die Aktivität *Medikation durchführen* oder *Patienten zurückbringen* abgeschlossen wurden, endet die Prozessinstanz automatisch.

Sobald eine Aktivität von einem Krankenhausmitarbeiter aus der entsprechenden Organisationseinheit gestartet wurde, wird sie an diesen gebunden. Das Abfragen von Aufgabenlisten funktioniert anhand der WfMS nun wie folgt. Ein Krankenhausmitarbeiter erfragt am REST-Service eine Liste von Aufgaben, die von ihm bereits begonnen wurden oder offen sind. Dazu erhält der REST-Service den Identifikator des Krankenhausmitarbeiters. Der REST-Service wiederum erfragt über eine SOAP-Schnittstelle der WfMS alle Aktivitäten ab, die bereits zwar aktiviert, aber noch nicht gestartet wurden. Zusätzlich zu diesen Aktivitäten werden die vom Krankenhausmitarbeiter gestarteten Aktivitäten abgefragt. Das WfMS antwortet darauf mit den Aktivitäten, die der Krankenhausmitarbeiter starten kann oder von ihm bereits gestartet wurden. Dazu benötigt er den Identifikator des Krankenhausmitarbeiters und vergleicht ihn mit dem Organisationsmodell. Über die Zuordnung zwischen Aktivitäten und Organisationsmodell werden nur die gewünschten Aktivitäten zurückgeliefert. Da der REST-Service jedoch keine Kenntnis über die Aktivitäten innerhalb des WfMS besitzt, liefert letzteres die zu Beginn der Prozessinstanz übergebenen Primärschlüssel der Aufgabe mit. Mit diesen Primärschlüsseln kann der REST-Service dann den dazu konformen JSON-String generieren, indem die Aufgaben mittels des JPA-Modells ermittelt und konvertiert werden. Möchte ein Krankenhausmitarbeiter den Status einer Aufgabe ändern, so ruft der REST-Service die entsprechende Aktivität im WfMS auf und startet oder beendet sie je nach Art der Statusänderung.

Die Vorteile durch die Nutzung einer WfMS sind also, dass keine manuelle Änderung des Status einer Aufgabe vorgenommen werden muss, sondern das WfMS dies automatisch durchführt. Ferner können Aufgabenlisten durch das WfMS anhand eines einfachen Organisationsmodells ebenfalls automatisch und ohne manuelle Überprüfung von Berechtigungen erstellt werden.

Leider konnte die Idee und ihr Entwurf so nicht in Medical Beacons implementiert werden, da es zu Problemen kam, die die Anbindung verhinderten. Die Probleme sollen im folgenden Kapitel kurz skizziert werden.

9.3 Probleme

Eines der aufgekommenen Probleme hat mit dem Abfragen des bei der Instanziierung des Prozesses übergebenen Primärschlüssels eines Auftrags zu tun. In der hier vorgestellten Architektur von Medical Beacons benötigt der REST-Service den Primärschlüssel, um den dazugehörigen Auftrag aus der Datenbank abzufragen. Datenelemente einer Prozessinstanz lassen sich bei Aristaflow aber nicht extrahieren. Das Problem konnte jedoch dadurch umgangen werden, indem der Primärschlüssel in die Beschreibungen von Aktivitäten mit aufgenommen wird. Beim Abfragen der offenen oder gestarteten Aktivitäten wird dann folglich der dazugehörige Primärschlüssel in der Beschreibung ausgegeben. Das Vorgehen eignet sich in diesem konkreten Anwendungsfall für die Abfrage von Datenelementen. Sobald aber eine größerer Anzahl davon abgefragt werden soll, müssen komplexe Methoden entwickelt werden, um die jeweiligen Datenelemente in der Beschreibung auseinanderhalten zu können.

Das Problem, an dem die Implementierung scheiterte, ist die SOAP-Schnittstelle von Aristaflow. An dieser Stelle soll erwähnt werden, dass der Prozess über das proprietäre Protokoll mit den in Aristaflow ausgelieferten Werkzeugen sehr gut funktionierte. Die Anbindung an den REST-Service muss aber über Aristaflows SOAP-Schnittstelle durchgeführt werden.

Das erste Problem ist, dass die XML-basierten SOAP-Nachrichten Elemente besitzen, die als optional markiert sind. Der Inhalt eines solchen Elements darf aber nicht leer oder mit *null* befüllt sein. Daraus folgt, dass optionale Elemente, die von einer Anwendung nicht befüllt werden können oder sollen, komplett aus der SOAP-Nachricht entfernt werden müssen. Während der Implementierung wurde *wsimport* verwendet, um aus der WSDL des SOAP-Services die korrespondierenden Java-Klassen zu erzeugen. Nun trat das Problem auf, dass in den einzelnen erzeugten Klassen die Attribute und Elemente händisch entfernt werden mussten, die während des Nachrichtenaustauschs nicht befüllt werden sollten oder konnten. Leider konnte Aristaflow die so erzeugten SOAP-Nachrichten nicht mehr verarbeiten. Während des Verfassens dieser Arbeit konnte nicht nachvollzogen werden, warum die SOAP-Nachrichten nicht mehr erkannt wurden. Potentielle Fehlerquellen können wsimport, die SOAP-Implementierung JAX-WS oder der Aristaflow-Server sein.

Um dem Fehler auf die Spur zu kommen, wurde das Werkzeug *SOAP UI* verwendet. Mit diesem können, anhand der WSDL des SOAP-Services von Aristaflow, händisch XML-

basierte SOAP-Nachrichten erstellt werden. Beim Nachrichtenaustausch mit den händisch erstellen Nachrichten traten jedoch weitere Probleme auf, welche die Anbindung des WfMS letztendlich scheitern lies. Um eine Aktivität einer Prozessinstanz zu starten, muss sich ein Anwender aus dem Organisationsmodell mit einem Passwort anmelden. Nach der korrekten Anmeldung erhält man vom SOAP-Service eine *Session Id*. Diese muss nun bei allen weiteren Interaktionen mit dem Service vom Anwender mitgesendet werden. Insbesondere dann, wenn eine Aktivität gestartet werden soll. Die Session Id identifiziert laut Dokumentation den angemeldeten Benutzer. Beim Starten einer Aktivität verknüpft Aristaflow sie mit dem angemeldeten und die Aktivität startenden Anwender, sodass nur dieser Anwender die Aktivität abbrechen oder abschließen kann. Soll die Aktivität während der gleichen Session abgeschlossen oder abgebrochen werden, funktioniert der Ablauf wie beschrieben. Meldet sich der Anwender aber ab und danach wieder an, so kann er die Aktivität nicht mehr abbrechen oder abschließen. Der Verdacht, es könnte an der neuen Session Id liegen, die durch die neue Anmeldung neu generiert wurde, konnte aber nicht bestätigt werden. Auch mit der alten Session Id, die zum Starten der Aktivität verwendet wurde, lies sich diese nicht mehr abbrechen oder abschließen. Meldete man sich mit einem Administrator (*supervisor*) ein, so konnte auch dieser nicht mehr mit der Aktivität interagieren. Da sich Anwender auf dem mobilen Client und dem Server aber an- und abmelden oder der Aristaflow-Server einem Neustart unterzogen werden könnte, wodurch ebenfalls eine Neuanmeldung notwendig wäre, wurde die Implementierung und die vergebliche Fehlersuche an dieser Stelle abgebrochen. Es will an dieser Stelle aber nochmals ausdrücklich angemerkt werden, dass über das proprietäre Kommunikationsprotokoll und die mitgelieferten Werkzeuge alles umgesetzt werden konnte. Allein die Anbindung an den REST-Service über die SOAP-Schnittstelle von Aristaflow lies sich aufgrund der beschrieben Probleme nicht umsetzen.

10 | Präsentation

Abbildung 10.1: Ansicht für die Anmeldung

In in diesem Kapitel sollen nun die einzelnen Anwendungsfälle und die Interaktionen mit iBeacons der entwickelten mobilen Anwendung anhand von Screenshots dargestellt und erläutert werden. Dabei wird die mobile Anwendung zunächst aus der Sicht eines Stationsarztes, danach aus der Sicht eines Krankenpflegers und abschließend aus der Sicht eines Facharztes vorgestellt. Bevor die einzelnen Organisationseinheiten ihre spezifischen Teilanwendungen einsehen können, müssen sie sich erst über Ihre Mitarbeiternummer in der mobilen Anwendung anmelden. Anhand der Mitarbeiternummer und einer entsprechenden Antwort des Servers wird die dazugehörige Teilanwendung geladen. Die Ansicht zum Anmelden ist in Abbildung 10.1 abgebildet.

10.1 Stationsarzt

In Abbildung 10.2 ist der Startbildschirm eines Stationsarztes eingeblendet und wird in der Implementierung von `DoctorStartViewController.m` realisiert. Die Anwendungsfälle, die von dieser Ansicht angeboten werden, sind das Anzeigen aller Patienten auf der aktuellen Station, das Anzeigen von Patienten eines bestimmten Krankenzimmers und das manuelle oder automatisierte Einblenden einer Patientenakte. Um den richtigen Anwendungsfall auszuführen, wird der Kontext hier wie folgt bestimmt.

Befindet sich der Stationsarzt in der Reichweite von iBeacons, ohne direkten Kontakt mit einem solchen aufzunehmen, wird die Station des am nächsten gelegenen iBeacon ermittelt. Anhand der Station werden dann die Patienten der aktuellen Station aufgelistet. Wie in Abbildung 10.2 zu erkennen, werden die Patienten in einer Liste unter anderem mit ihren Namen, der Station-, Zimmer- und Bettnummer und der An- oder Abwesenheit aufgezeigt. Über die Information der An- und Abwesenheit ist es dem Stationsarzt möglich, seine Visite besser zu planen.

Stellt der angemeldete Stationsarzt in dieser Ansicht direkten Kontakt mit einem iBeacon her, so ändert sich der Kontext der Anwendung innerhalb dieser Ansicht dahingehend, dass entweder eine Übersicht der Patienten innerhalb eines Krankenzimmers eingeblendet oder eine Patientenakte aufgerufen wird. Dazu ist die Überprüfung des Payloads und insbesondere des Major- und Minor-Wertes des iBeacons nötig.

Wurde ein iBeacon eines Krankenzimmers berührt, so wird die Ansicht aus Abbildung 10.3 eingeblendet. Dabei handelt es sich um die gleiche Liste wie in der Stationsübersicht. Der Unterschied ist der, dass nur Patienten eingeblendet werden, die im gewünschten Zimmer ihr festes Krankenbett haben. Auch hier wird die wichtige Anforderung der An- und Abwesenheit von Patienten angezeigt.

Berührt der Stationsarzt in dieser Ansicht hingegen einen iBeacon eines Patienten, so ändert sich der Kontext und die Anwendung weiß, dass sie die Patientenakte des gewünschten Patienten aufrufen muss. Aus Gründen der Benutzerfreundlichkeit und Effizienz der Anwendungsfälle, kann die Patientenakte auch direkt über die Auswahl eines Listeneintrags aus einer der Listen eingeblendet werden. Die Ansicht der Patientenakte ist in 10.4 abgebildet und wird in der Implementierung von `DoctorPatientChartViewController.m` realisiert. In der Patientenakte werden Informationen über den ausgewählten Patienten angezeigt. Neben der Einsicht bereits erstellter fachärztlichen Untersuchungen und Medika-

tionen, die in einer Liste unter den Daten des Patienten mit Informationen und dem Status angezeigt werden, kann der Stationsarzt durch die beiden Schaltflächen *Neuer Medikamentenjob* und *Neuer Facharztjob* neue Aufgaben hinzufügen.

Das Hinzufügen einer neuen Medikation ist in Abbildung 10.5 abgebildet und wird in der Implementierung von `DoctorAddMedicationJobViewController` realisiert. Dem Stationsarzt ist es in dieser Ansicht möglich sowohl die benötigten Medikamente und deren Anzahlen zu bestimmen als auch eine Beschreibung für die Medikation einzugeben. Mit dem Auswählen der Schaltfläche *Hinzufügen* wird die Medikation hinzugefügt und auf der Aufgabenliste eins Krankenpflegers eingeblendet.

Eine fachärztliche Untersuchung wird über die Ansicht in Abbildung 10.6 erstellt. Für die Implementierung der Ansicht ist die Klasse `DoctorAddSpecialistJobViewController` zuständig. Dem Stationsarzt ist es möglich, eine Beschreibung einzugeben und einen passenden Facharzt aus der Liste auszuwählen. Durch das Auswählen der Schaltfläche *Hinzufügen* wird die fachärztliche Untersuchung hinzugefügt und auf der Aufgabenliste eines Krankenpflegers eingeblendet. Durch die Bestimmung eines Facharztes beim Erstellen der fachärztlichen Untersuchung wird gleichzeitig festgelegt, in welches Zimmer der Patient verlegt werden muss. Nämlich in die Praxis des Facharztes. Nach der Verlegung des Patienten wird die Aufgabe dann auf der Aufgabenliste des ausgewählten und damit zuständigen Facharzt aufgelistet.

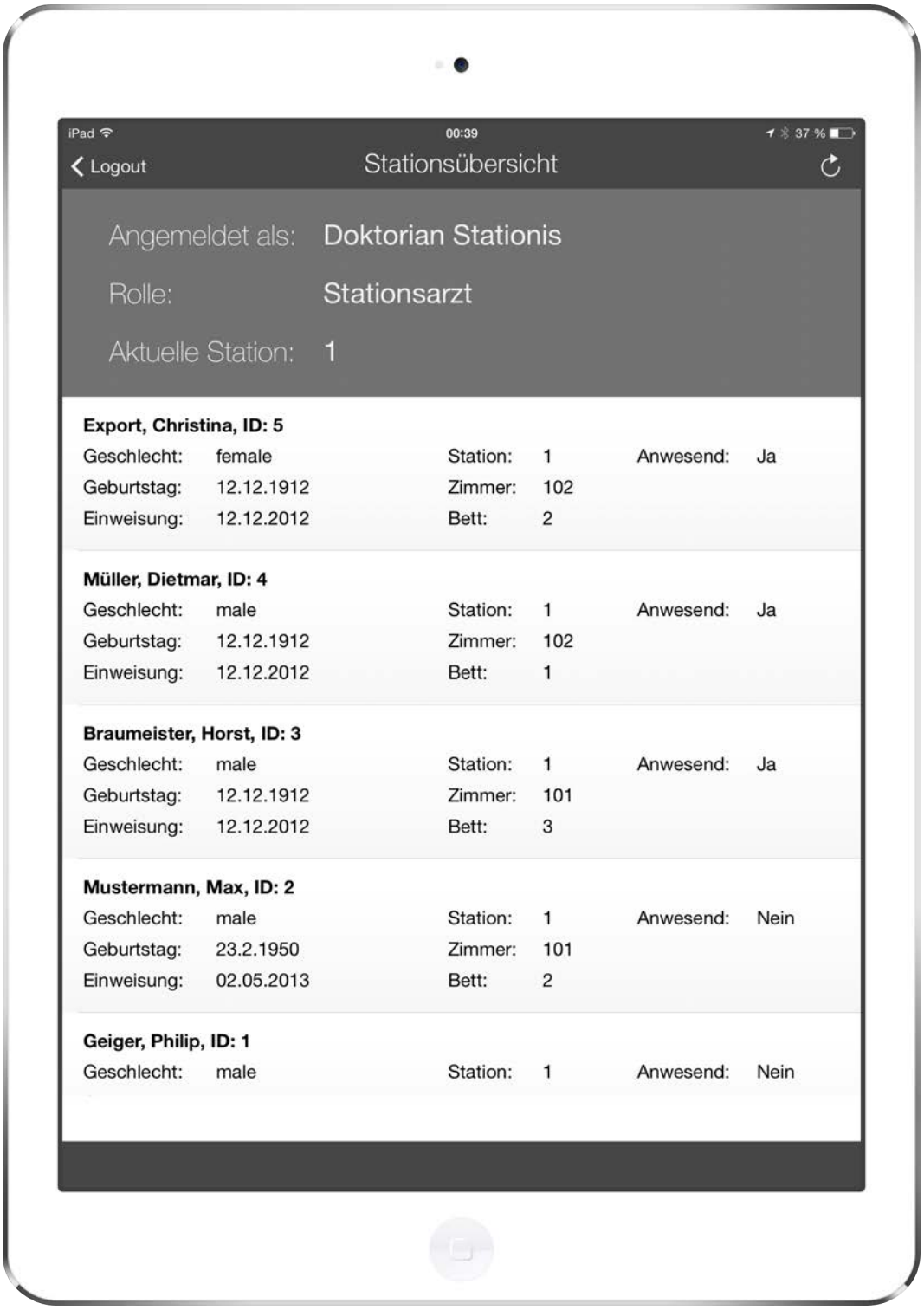

Abbildung 10.2: Stationsübersicht eines Stationsarztes

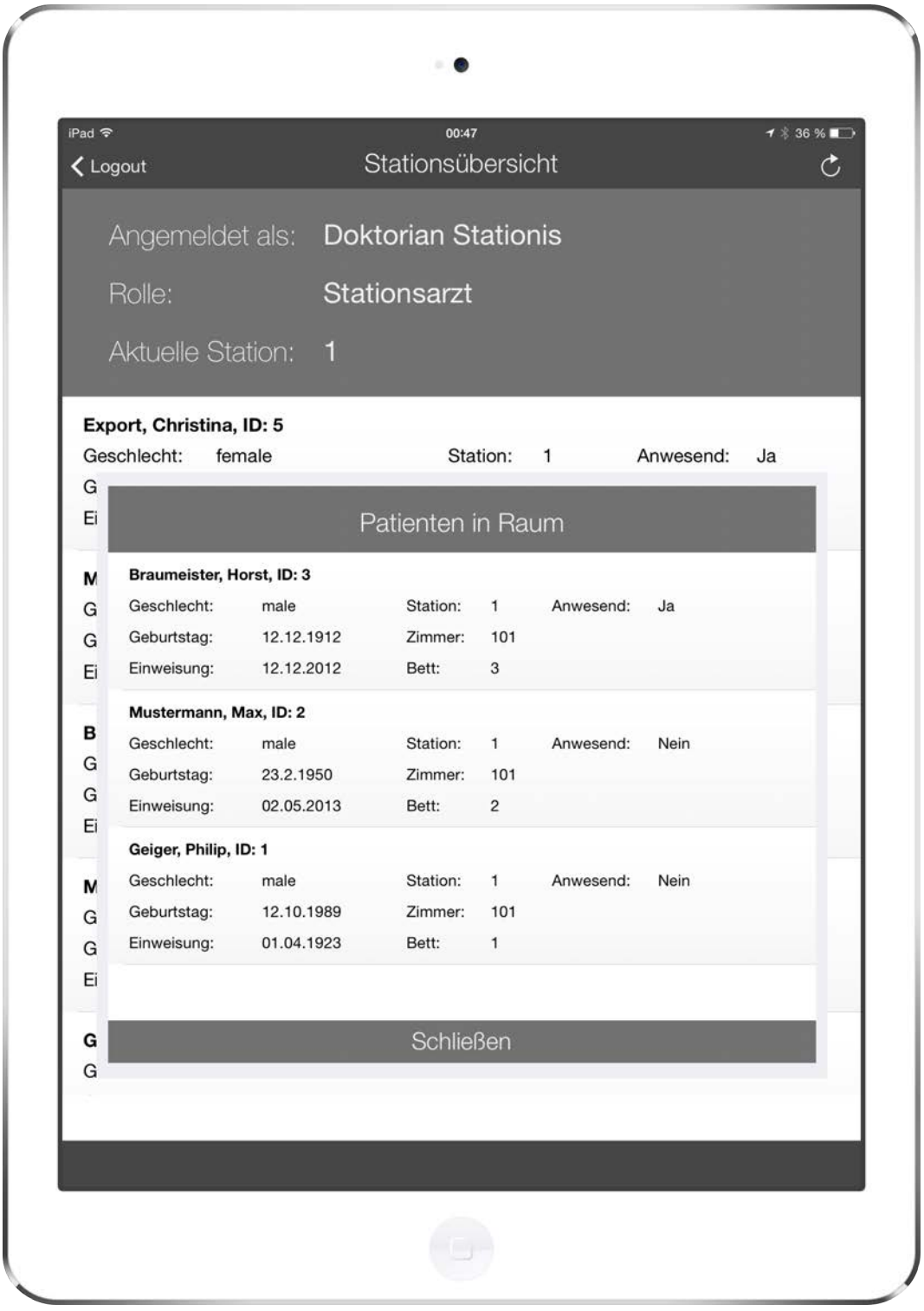

Abbildung 10.3: Übersicht der Patienten in einem ausgewählten Krankenzimmer

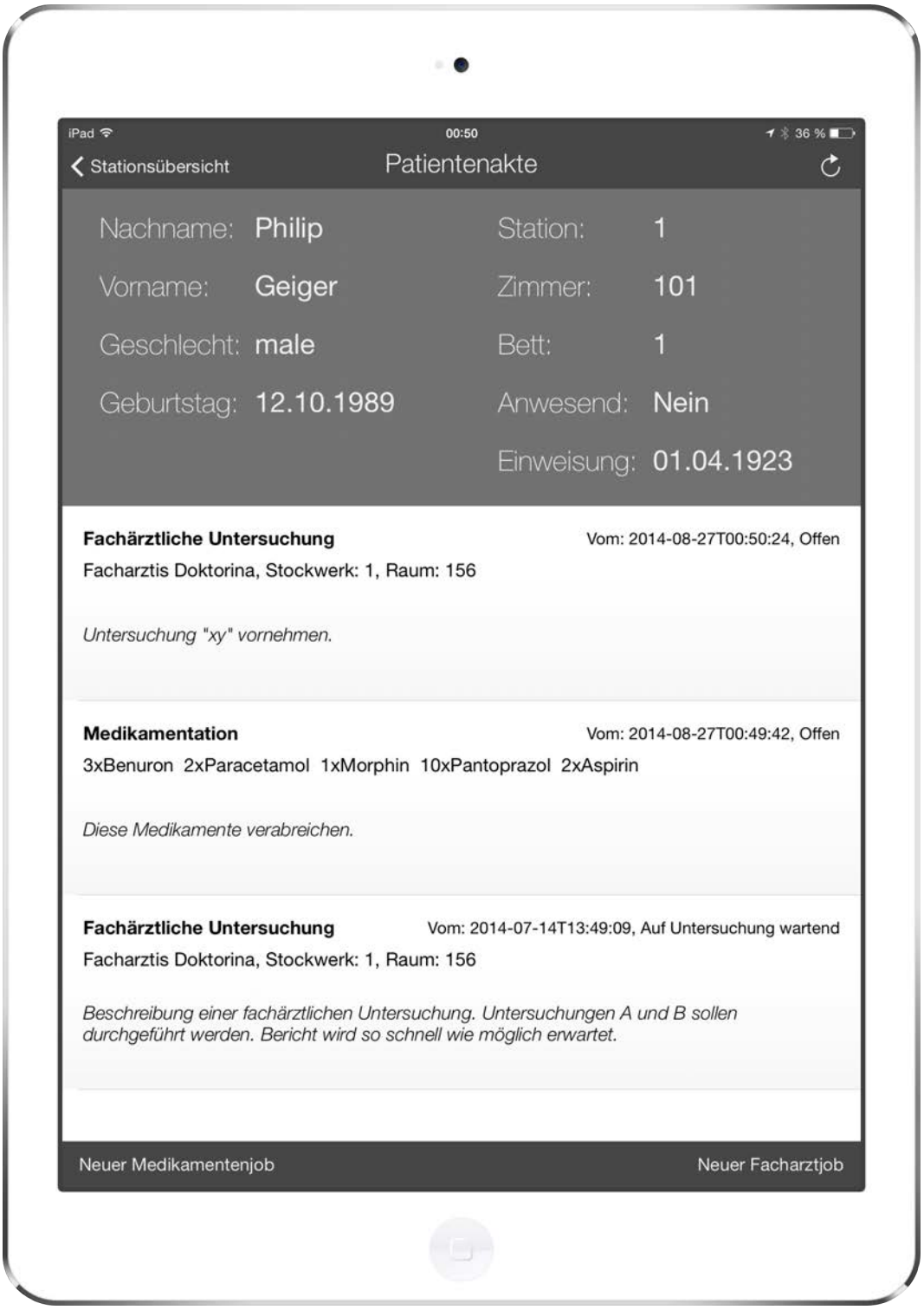

Abbildung 10.4: Ansicht der Patiententakte

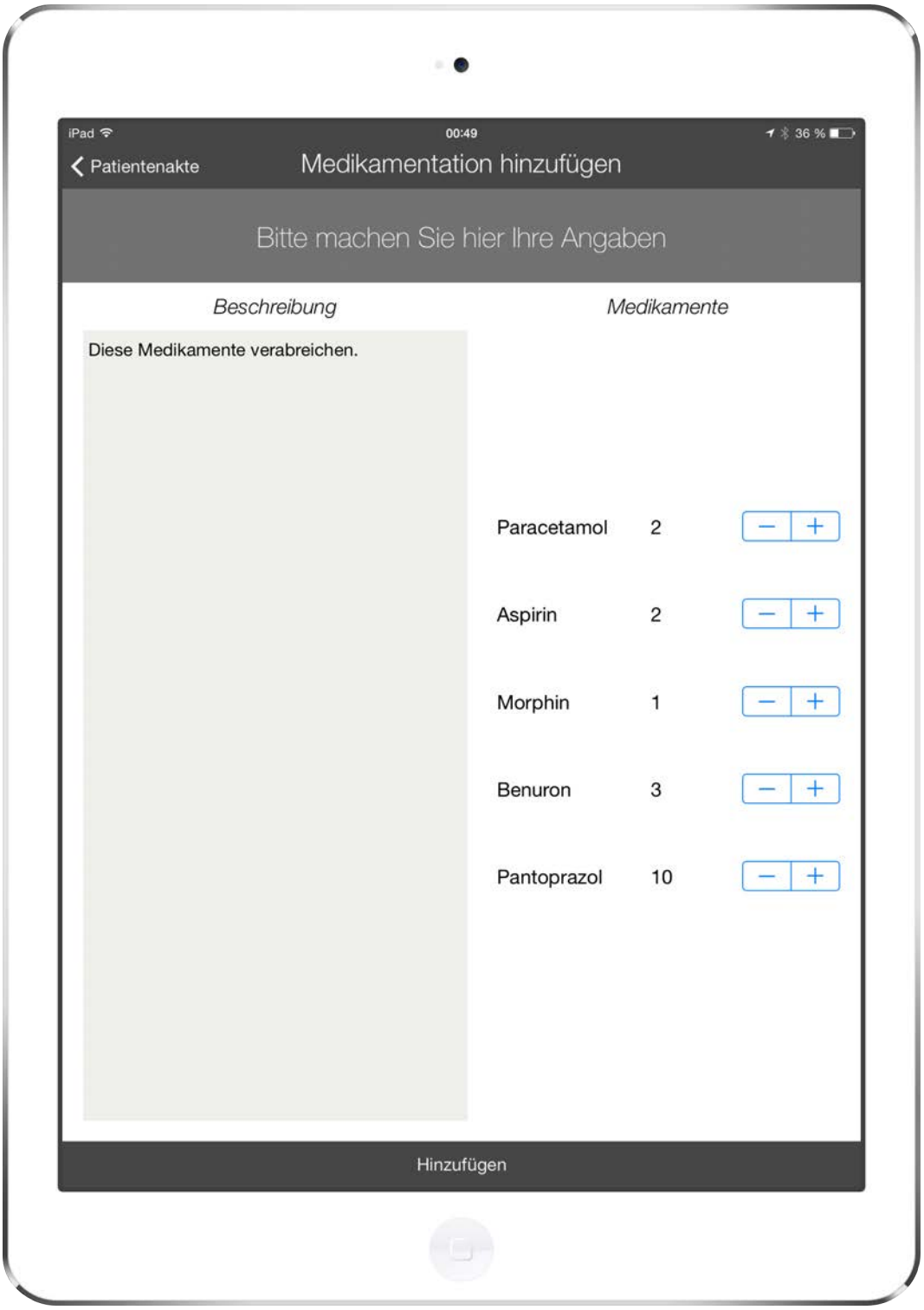

Abbildung 10.5: Ansicht, um eine Medikation hinzufügen

Abbildung 10.6: Ansicht, um Fachärztliche Untersuchung hinzufügen

10.2 Krankenpfleger

Ein angemeldeter Krankenpfleger erhält die Ansicht aus Abbildung 10.7, die von `NurseStartViewController` implementiert wird. Der Kontext lässt sich anhand der aktuellen Ansicht und einem in Reichweite befindlichen iBeacon bestimmen. Durch das Extrahieren der Station des am nächsten gelegenen iBeacons werden alle Aufgaben auf der aktuellen Station aufgelistet. Neben offenen Aufgaben der aktuellen Station werden ebenfalls vom Krankenpfleger bereits gestartete Aufgaben angezeigt. Die gestarteten Aufgaben können auch von einer anderen Station sein. Das ist nötig, weil ein Krankenpfleger dauerhaft auf einer anderen Station arbeiten könnte, aber noch offene Aufgaben von anderen Stationen zu erledigen hat.

Durch die Auswahl einer offenen Medikation wird die Ansicht in Abbildung 10.8 eingeblendet, die in der Implementierung von `NurseMedicationJobViewController` realisiert wird. In ihr werden einige Daten zum zugehörigen Patienten, eine Beschreibung der Medikation und die notwendigen Medikamente eingeblendet. Durch die Schaltfläche *Job starten* lässt sich die Medikation vom angemeldeten Krankenpfleger beginnen und wird gleichzeitig an ihn gebunden. Dadurch wird die gestartete Medikation nicht mehr in der Aufgabenliste anderer Krankenpfleger angezeigt.

Nach dem Starten einer Medikation ändern sich einige Informationen der aktuellen Ansicht (s. Abbildung 10.9). Zum einen ändert sich der Status auf *In Bearbeitung*, zum anderen wird dem Krankenpfleger die Information bereitgestellt, dass er zum Abschließen der Medikation den iBeacon des Medikamentenlagers berühren muss, aus dem er die Medikamente entnommen hat. Ferner ändert sich die untere Schaltfläche zu *Job abbrechen*, womit der Krankenpfleger die Medikation abbrechen kann und gleichzeitig wieder für andere Krankenpfleger verfügbar wird. Wird die Medikation korrekt abgeschlossen, werden die Medikamentenbestände des Medikamentenlagers anhand der gelisteten Medikamente innerhalb der Aufgabe aktualisiert.

Mit der Auswahl einer offenen fachärztlichen Untersuchung aus der Aufgabenliste wird die Ansicht aus Abbildung 10.10 aufgerufen. Diese Ansicht wird von der Klasse `NurseSpecialistViewController` implementiert. Durch sie werden dem Krankenpfleger sowohl einige Patientendaten als auch eine Beschreibung und Informationen zur Aufgabe angezeigt. Da eine fachärztliche Untersuchung mit der Verlegung des Patienten beginnt, wird dem Krankenpfleger angezeigt, zu welchem Facharzt der Patient gebracht werden muss.

Damit verhindert wird, dass ein falscher Patient verlegt wird, muss der iBeacon des zu verlegenden Patienten berührt werden, um die Aufgabe zu beginnen.

Wurde die Aufgabe korrekt begonnen, ändert sich die Ansicht (s. Abbildung 10.11). Zum einen ändert sich auch hierbei der Status, zum anderen wird dem Krankenpfleger angezeigt, dass er den iBeacon der Praxis des Facharztes berühren muss. Um eine Verlegung zu einem falschen Facharzt auszuschließen, wird der Kontext so ausgelegt, dass die Werte des berührten iBeacons mit der Station und Zimmernummer aus der Aufgabe übereinstimmen müssen.

Nicht abgebildet ist die Rückverlegung eines Patienten in sein Krankenzimmer nach einer fachärztlichen Untersuchung. Diese Aufgabe läuft aber nach dem gleichen Prinzip ab. Zum Beginnen einer Rückverlegung muss der dazugehörige iBeacon eines Patienten berührt werden. Das ist nötig, um sicherzustellen, dass der richtige Patient vom Facharzt abgeholt wird. Um die Rückverlegung einer fachärztlichen Untersuchung abzuschließen, muss der iBeacon am Krankenzimmer des Patienten berührt werden. Auch dadurch soll sichergestellt, dass der Patient in der Aufgabe mit dem Zimmer übereinstimmt, um damit eine fehlerhafte Rückverlegung zu vermeiden.

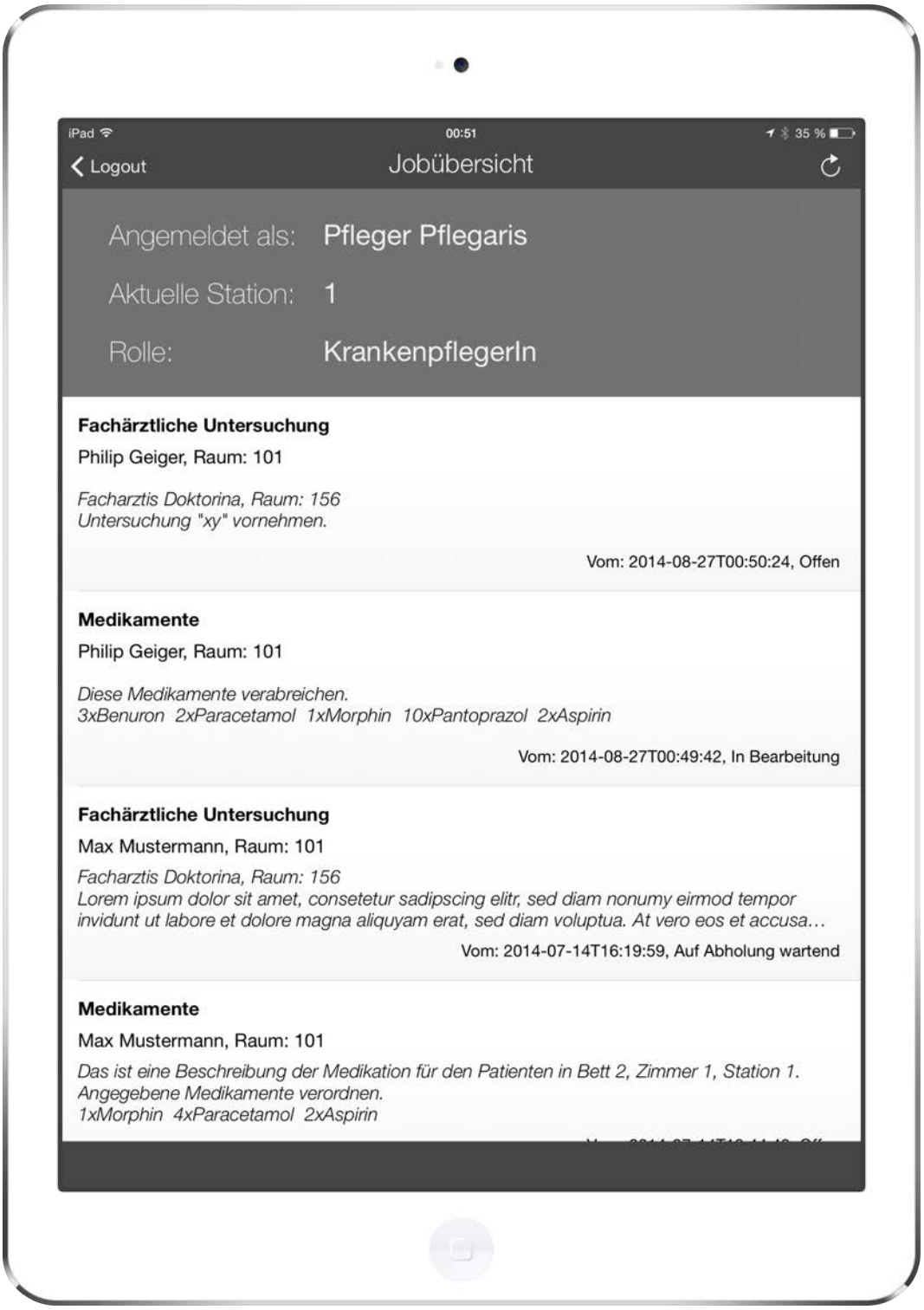

Abbildung 10.7: Aufgabenübersicht eines Krankenpflegers

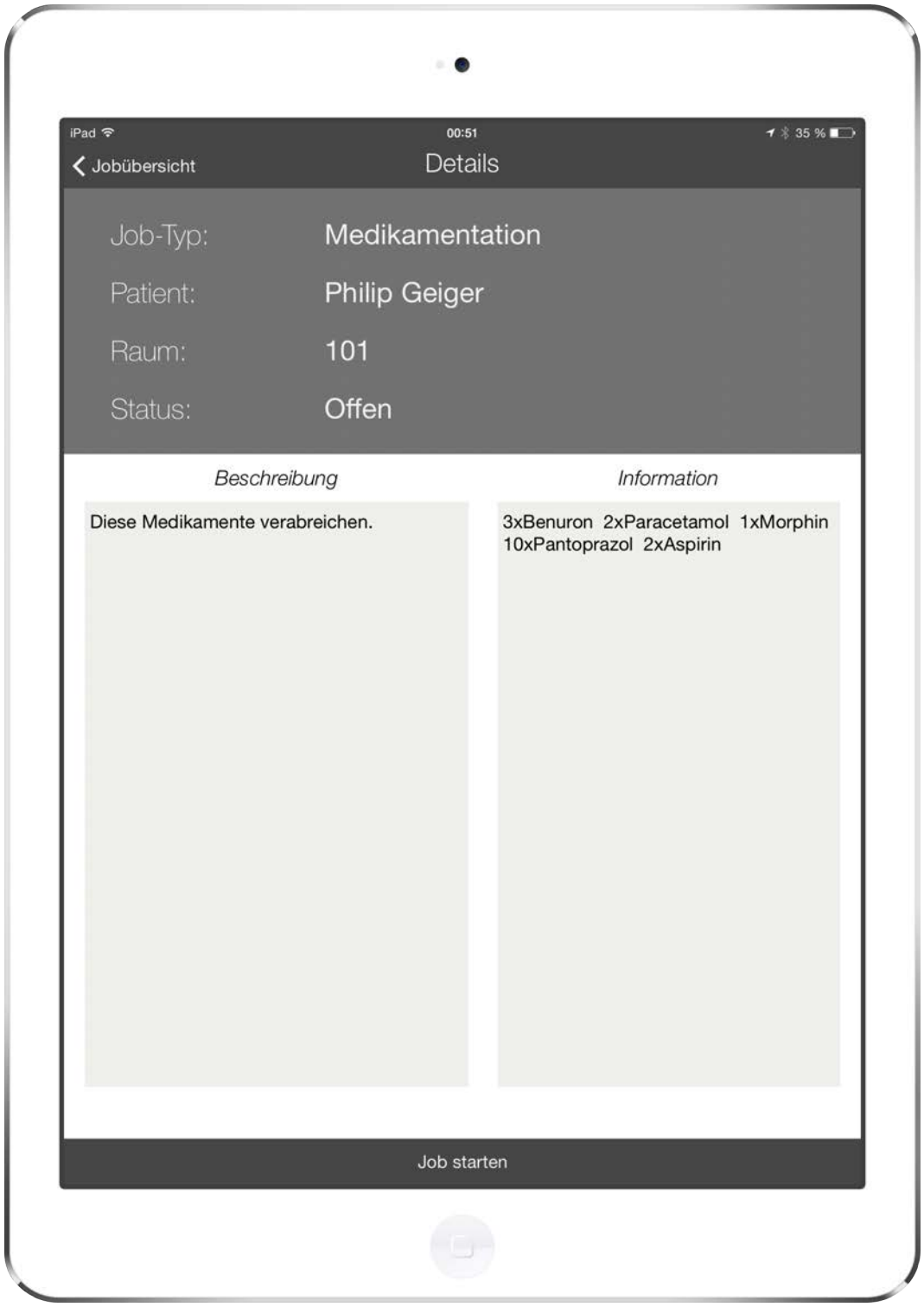

Abbildung 10.8: Ansicht einer offenen Medikation

Abbildung 10.9: Ansicht einer begonnen Medikation

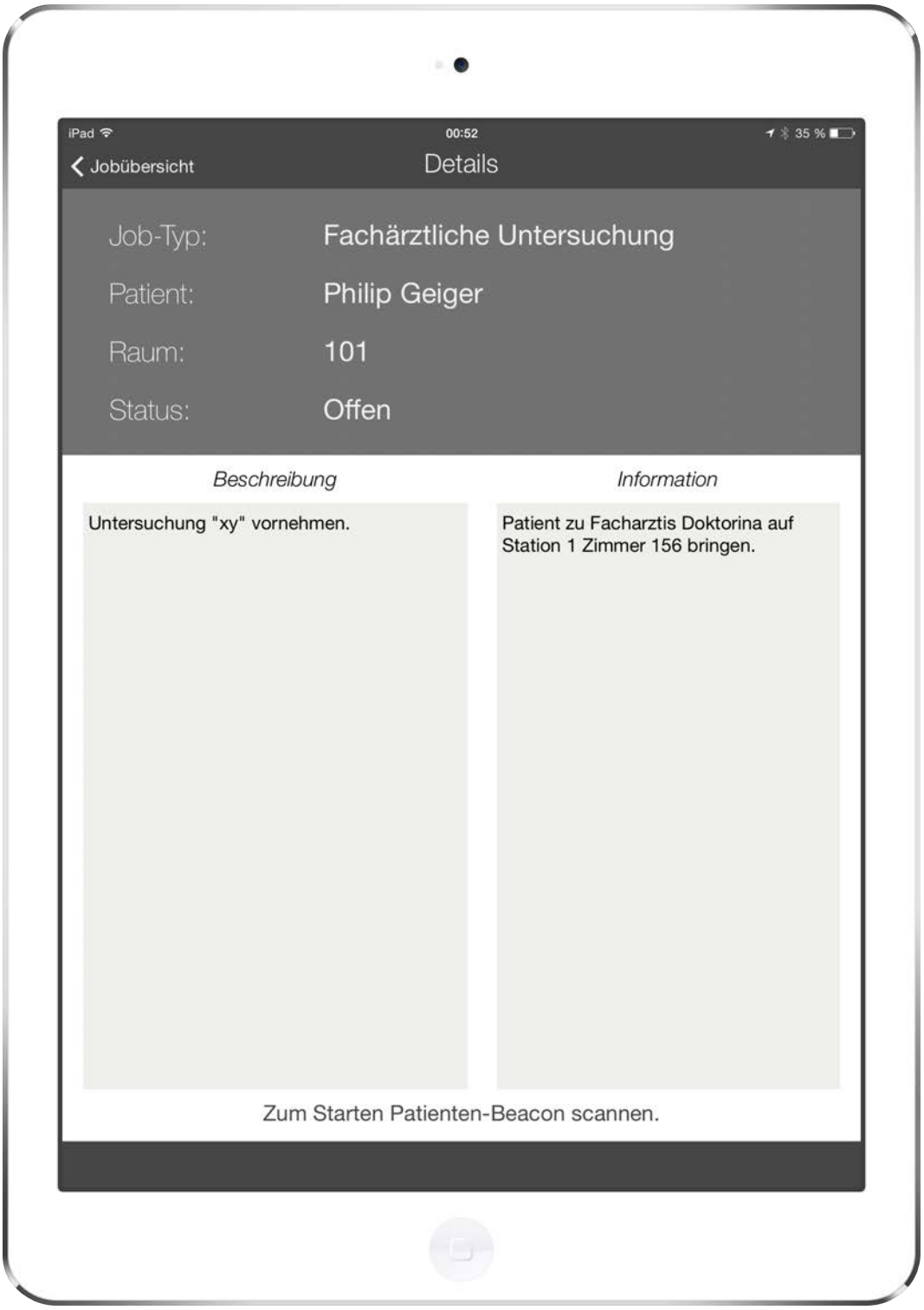

Abbildung 10.10: Ansicht einer offenen fachärztlichen Untersuchung aus der Sicht eines Krankenpflegers

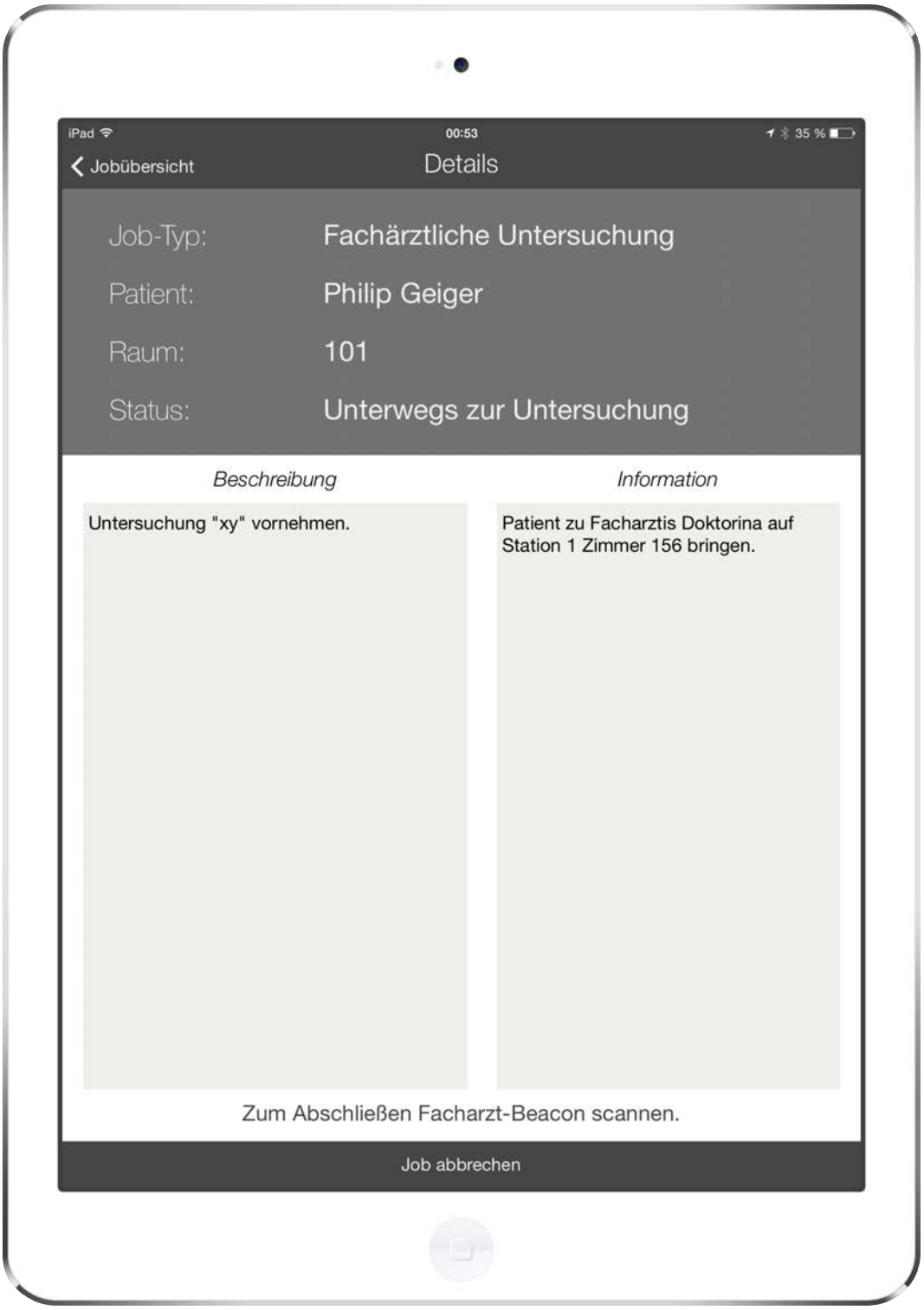

Abbildung 10.11: Ansicht einer begonnenen fachärztlichen Untersuchung aus der Sicht eines Krankenpflegers

10.3 Facharzt

Ein angemeldeter Facharzt erhält die Ansicht aus Abbildung 10.12, die von `Specialist-StartViewController` implementiert wird. In dieser Ansicht werden alle offenen und vom angemeldeten Facharzt begonnenen fachärztlichen Untersuchungen aufgelistet. Eine Aufgabe ist dabei für den Facharzt offen, wenn der Patient die Praxis des Facharztes erreicht hat und die Verlegung durch einen Krankenpfleger abgeschlossen wurde. Eine solche Aufgabe erhält den Status *Auf Untersuchung warten*.

Wählt ein Facharzt eine offene fachärztliche Untersuchung aus, so erhält der die Ansicht, die in Abbildung 10.13 abgebildet ist. Sie wird von `SpecialistSpecialistViewController` implementiert und liefert dem Facharzt einige Patientendaten, eine Beschreibung und Informationen zur Aufgabe. Um die Aufgabe zu beginnen, muss der passende iBeacon des Patienten berührt werden. Damit soll an dieser Stelle sichergestellt werden, dass der richtige Patient angenommen und danach untersucht bzw. behandelt wird.

Wurde die Aufgabe vom Facharzt korrekt begonnen, so ändert sich die Ansicht so ab, wie sie in Abbildung 10.14 abgebildet ist. Der Status der fachärztlichen Untersuchung ändert sich in *In Untersuchung*. Ferner ändert sich der Text der Information und die Anweisung den iBeacon des Patienten zu berühren, um die Aufgabe abzuschließen, wird eingeblendet. Der Kontext ergibt sich aus der geöffneten Aufgabe und dem berührten iBeacon. Die Anwendung weiß nach der Berührung mit dem passenden iBeacon, dass die Aufgabe abgeschlossen werden soll und schließt diese darauffolgend ab.

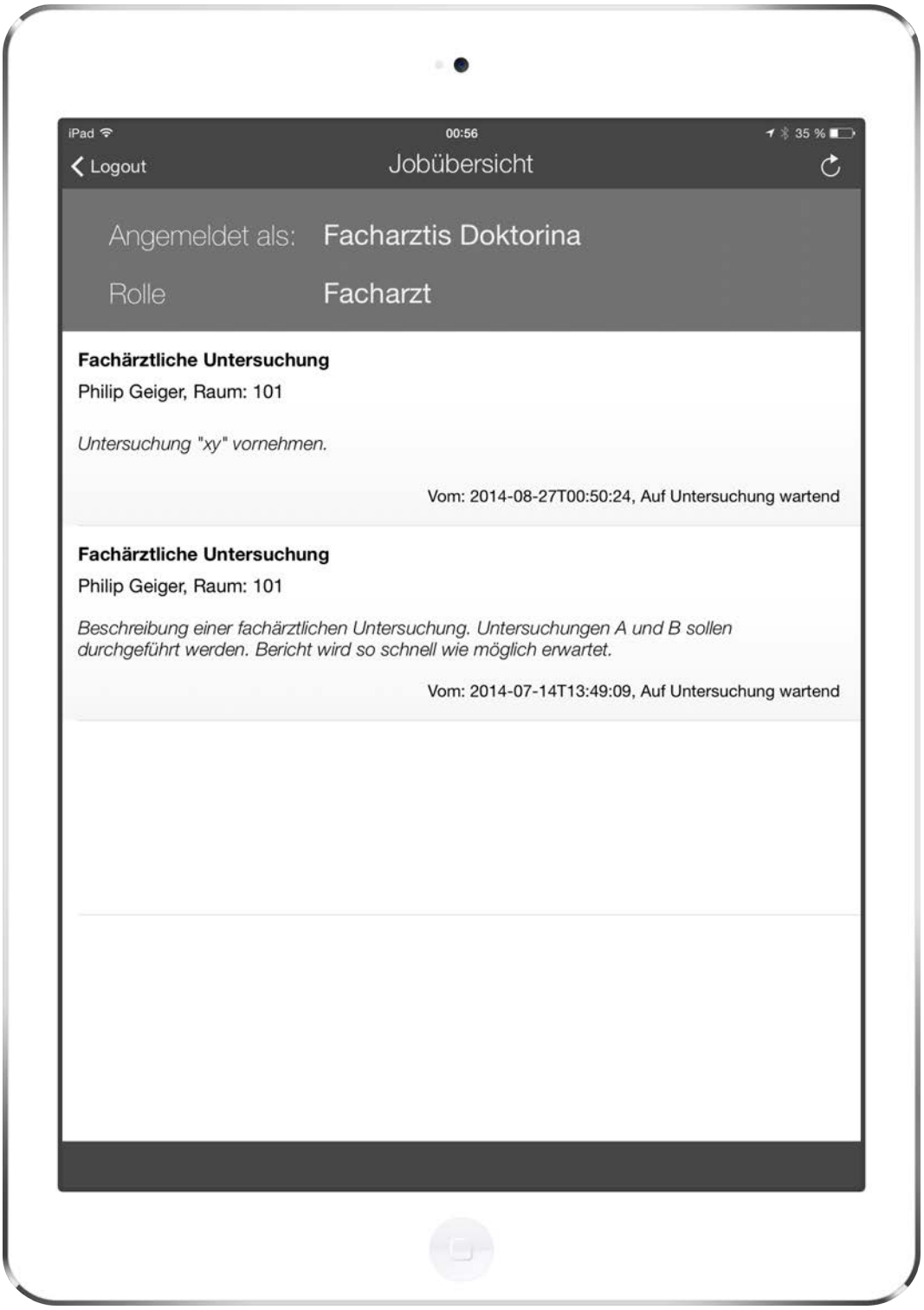

Abbildung 10.12: Aufgabenübersicht eines Facharztes

Abbildung 10.13: Ansicht einer offenen fachärztlichen Untersuchung aus der Sicht eines Facharztes

Abbildung 10.14: Ansicht einer begonnen fachärztlichen Untersuchung aus der Sicht eines Facharztes

11 | Bewertung

Die Implementierung von Medical Beacons wurde anhand des vorgestellten Krankenhaus-Prozesses und den aufgestellten Anforderungen durchgeführt. In diesem Kapitel werden kurz einzelne Anforderungen wiederaufgenommen und mit dem entwickelten System verglichen.

Es sollten iBeacons eingesetzt werden, die sowohl leicht zu konfigurieren und erweiterbar sind als auch Patienten, Stationen und Zimmer eindeutig identifizieren können. Dazu wurden iBeacons in der Form von Raspberry Pis, iBeacons von Radius Networks und eine iPhone iBeacon App für das einfache Durchführen von Tests verwendet. Mit der Kodierung von Station und Zimmernummer im Major-Wert des iBeacons können sehr viele Stationen und bis zu 100 Zimmer pro Station eindeutig identifiziert werden. Diese Anzahl sollte für jedes Krankenhaus groß genug sein. Der Minor-Wert eines iBeacon bestimmt die Bettnummer innerhalb eines Zimmers auf einer Station. Auch dieser Wert ist groß genug, um alle Betten in einem Zimmer zu identifizieren. Durch die vorgestellte Art der Kodierung wird einer Erweiterung um zusätzliche Gebiete des Krankenhaus sichergestellt, da sowohl im Major- als auch im Minor-Wert genügend Werte während der Implementierung unbesetzt geblieben sind.

Die unterschiedlichen Interaktionen mit iBeacons zur Kontextbestimmung wurden ebenfalls in den einzelnen Anwendungsfällen berücksichtigt. Nämlich zum einen durch einen direkten Kontakt mit einem iBeacon, um beispielsweise Aufgaben zu beginnen oder abzuschließen, und zum anderen die einfache Anwesenheit einer iBeacons, um die aktuelle Station eines Anwenders zu bestimmen.

Ferner wurden alle Anwendungsfälle für die Optimierung und Automatisierung des Geschäftsprozesses im Krankenhaus umgesetzt, die in der Anforderungsanalyse definiert und gesammelt wurden.

Die Anwendung wurde unter verschiedenen Bedingungen getestet, wodurch einige Erkenntnisse über die Genauigkeit und Verlässlichkeit von iBeacons gesammelt werden konnten. Werden die einzelnen iBeacons zu nah nebeneinander aufgestellt, so kommt es zu Interferenzen und die Anwendung kann die Entfernungen der iBeacons nicht mehr genau berechnen. Dies kann sogar dazu führen, dass die Interferenzen so stark werden, dass die Anwesenheit eines iBeacon überhaupt nicht mehr bestimmt werden kann, da die empfangenen Signale für eine korrekte Rekonstruktion zu sehr gestört bzw. verrauscht sind. Werden die iBeacon mit genügend Abstand platziert, so funktioniert die Bestimmung der Entfernung sehr gut. Es muss jedoch bedacht werden, dass Wände, WLAN-Signale (die auf der gleichen Frequenz arbeiten) oder sogar anwesende Personen die Berechnung der Distanz verfälschen können.

Der direkte Kontakt zwischen mobilem Endgerät und einem iBeacon wird erkannt, wenn sich beide nur wenige Zentimeter voneinander entfernt aufhalten. Hierzu müssen die iBeacons jedoch erst kalibriert werden, um den RSSI-Wert für jeden iBeacon zu setzen. Da die Signalausbreitungen der Bluetooth-Signale aber sehr stark von der Art und Weise der Platzierung von iBeacons abhängt, kann dies unter Umständen bei einer hohen Anzahl eingesetzter iBeacons einen erheblichen Arbeitsaufwand bedeuten. Und selbst dann ist eine 100%ige Sicherheit für die Erkennung eines direkten Kontakts nicht gegeben. Sobald dem iPad, das zum Testen der Anwendung verwendet wurde, eine handelsübliche Schutzhülle angelegt wurde, wurden die Signalstärken sofort abgeschwächt und eine neue Kalibrierung hätte stattfinden müssen.

Die Idee, das Setzen und Überprüfen von Status der Medikationen und fachärztlichen Untersuchen über eine WfMS zu vereinfachen, wurde versucht mit Aristaflow umzusetzen. Leider scheiterte die Implementierung aber an der fehlerhaften SOAP-Kommunikation.

Es lässt sich zusammenfassen, dass alle Anwendungsfälle und Anforderungen korrekt und effizient umgesetzt wurden. Auch die Kommunikation und das Empfangen von Bluetooth-Signalen der iBeacons funktioniert sehr gut, wodurch sich Stationen, Patienten und Krankenzimmer eindeutig identifizieren und bestimmen lassen. Leider nimmt die Zuverlässigkeit der empfangenen Signale drastisch ab, sobald sich Störsignale und andere Störquellen im umliegenden Bereich des mobilen Endgeräts befinden. Damit einhergehend nimmt auch die eindeutige Bestimmung der Station über die Positionsbestimmung ab. Dies lässt sich jedoch nicht auf eine fehlerhafte Implementierung von Medical Beacons zurückführen, sondern auf die Eigenschaften und Gegebenheiten von drahtlosen Kommunikationssystemen.

12 | Zusammenfassung und Ausblick

In dieser Arbeit wurden Sensoren vorgestellt, die auf mobilen Endgeräten vorhanden sind und zur Optimierung und Automatisierung von Geschäftsprozessen verwendet werden können. Dabei wurde der Schwerpunkt auf Sensorik zur Positionsbestimmung gelegt. Insbesondere Bluetooth LE und das darauf aufbauende Konzept der iBeacons wurde ausgiebig erläutert, da dieses sowohl zur Positionsbestimmung innerhalb und außerhalb von Gebäuden als auch zur direkten Kontaktaufnahme herangezogen werden kann, wie es mit RFID oder NFC möglich ist. Damit stellen iBeacons mehrere verfügbare Konzepte in einem einzigen System zur Verfügung.

Aufgrund der vielfältigen Einsatzmöglichkeiten von iBeacons wurde ferner erläutert, wie ein solcher iBeacon mithilfe eines Raspberry Pis selbst konzipiert, implementiert und konfiguriert werden kann. So konfigurierte Raspberry Pis wurden dann währen der Implementierung des Krankenhaus-Prozesses verwendet. Da im Laufe der Erarbeitung dieser Arbeit immer mehr Unternehmen entstanden, die vorgefertigte und konfigurierbare iBeacons anbieten, wurden einige bekanntere Unternehmen und deren Produkte kurz vorgestellt.

Wie das Konzept der iBeacons in Geschäftsprozessen und anderen Szenarien verwendet werden kann, um die Abläufe und das Benutzererlebnis zu verbessern, wurde anhand von Beispielen, wie dem beschriebenen Krankenhaus-Prozess, einem Lagerhaus-Prozess, dem Szenario des mobilen Bestellens und Bezahlens und einigen bereits implementierten Anwendungsszenarien aufgezeigt.

Auf der Basis des diskutierten Krankenhaus-Prozesses wurde das System Medical Beacons konzipiert und entwickelt, das mithilfe konfigurierter iBeacons einige Geschäftsabläu-

fe optimiert und automatisiert. Dazu wurde eine ausführliche Anforderungsanalyse durchgeführt, um herauszufinden, an welchen Stellen Bedarf an Verbesserungen besteht. Mit den so gesammelten Anforderungen wurde ein Architekturentwurf entwickelt, welcher aus einem Server, einer mobilen Anwendung auf einem mobilen Endgerät und den eigentlichen iBeacons besteht. Die Herausforderung der Implementierung bestand darin, eine geeignete, einfach erweiterbare und verständliche Kodierung der Major- und Minor-Werte der iBeacons zu finden. Mit der entwickelten Kodierung war es schließlich möglich, einzelne Stationen, Zimmer und Patienten eindeutig über iBeacons zu identifizieren und ferner die aktuelle Station eines Anwenders über die Methoden des Positionsbestimmung zu bestimmen. Die Idee, Medical Beacons selbst über ein WfMS zu optimieren und das Setzen von notwendigen Status zu vereinfachen, scheiterte leider an Problemen mit der SOAP-Schnittstelle von Aristaflow. Die Idee selbst und der durchgeführte Entwurf konnten aber Aufschluss darüber geben, wie Medical Beacons damit optimiert hätte werden können.

Während ausgiebigen Tests von Medical Beacons und insbesondere der Kommunikation der mobilen Anwendung mit den aufgestellten Beacons wurde herausgefunden, dass zueinander nah gelegene iBeacons sich gegenseitig durch Interferenzen stören können, wodurch eine genaue Positionsbestimmung und vor allem das Diagnostizieren einer direkten Berührung mit einem iBeacon verhindert werden kann. Dem entsprechend müssen eingesetzte iBeacons mit einer gewissen Distanz zueinander platziert werden. Ferner sollte auf eine Umgebung mit wenigen Störquellen wie WLAN Signalen oder anderen Bluetooth-Anwendungen gesetzt werden, da die meisten Funktechnologien für den privaten oder geschäftlichen Gebrauch auf den gleichen Frequenzen des ISM-Spektrums arbeiten, wodurch die gleichen Effekte wie bei zueinander zu nah gelegenen iBeacons auftreten können. Zusätzlich müssen die eingesetzten iBeacons kalibriert werden. Das bedeutet, dass die Signalstärke aus einem Meter Entfernung gemessen werden und der dadurch erhaltene RSSI-Wert in den Payload des iBeacons eingetragen werden muss. Empfänger, die den Payload empfangen, können anhand des eingetragenen RSSI-Werts und der aktuell empfangenen Signalstärke die Entfernung zum iBeacon berechnen. Da die Signalausbreitung und Signalstärke von der Platzierung abhängig ist, muss jeder eingesetzte iBeacon separat und am besten in dessen realer Umgebung kalibriert werden. Weiter wurde herausgefunden, dass das Signal von Bluetooth LE sogar durch handelsübliche Schutzhüllen an mobilen Endgeräten wie einem iPad zur Abschwächung und Verfälschung des empfangenen Signals bzw. der Signalstärke führen kann.

Somit wurde aufgezeigt, dass sich viele Sensoren und vor allem iBeacons sehr gut für die Optimierung und Automatisierung von Geschäftsprozessen eignen, viele Sensoren aber anfällig gegenüber Störungen sein können.

Während der Implementierung des mobilen Clients von Medical Beacons wurde der Schwerpunkt auf das korrekte Erkennen von iBeacons, der Kontextbestimmung und der automatischen Ausführung von Anwendungsfällen gelegt. Die implementierte Patientenakte weist nur die wichtigsten Funktionalitäten auf und viele Prozessabläufe und Anwendungsfälle aus Krankenhäusern werden momentan noch nicht beachtet. Dadurch kann Medical Beacons in weiteren Arbeiten um echte Patientenakten und weitere Prozessabläufe erweitert werden. Ferner könnte in zukünftigen Arbeiten das Nachverfolgen von Mitarbeitern implementiert werden, wie es im Lagerhaus-Prozess kurz skizziert wurde. Mit den damit erhaltenen Daten der Position von Mitarbeitern könnten weitere Szenarien gefunden und umgesetzt werden. Da die Anbindung eines WfMS in dieser Arbeit nicht vollständig implementiert werden konnte, können in weiteren Arbeiten die Ursachen des Problems untersucht und ggf. behoben werden. Gerade weil die Anbindung eines WfMS hohes Potential für eine noch bessere automatisierte Ausführung des vorgestellten Geschäftsprozesses bietet, sollte dieses Ziel weiter verfolgt werden. Auch ein Test unter realen Bedingungen in einem Krankenhaus oder einer Arztpraxis kann durchgeführt werden, um die Funktionsweise zu validieren und die Akzeptanz von Ärzten und Krankenpflegern zu evaluieren. Da WLAN in den meisten Krankenhäusern vorhanden ist, könnte weiter untersucht werden, wie WLAN und iBeacons (bzw. Bluetooth LE) miteinander verbunden werden könnten, um eine noch bessere Positionsbestimmung und weitere Anwendungsfälle umzusetzen.

Da iBeacons sowohl Positionsbestimmungen als auch NFC-ähnliche Interaktionen anbieten, eigenen sie sich hervorragend für Szenarien, die über die vorgestellten hinaus reichen. Bisher wurde hauptsächlich die Seite betrachtet, auf welcher der Client passive iBeacons in seiner Umgebung nutzt. Aber auch die umgekehrte Seite bietet einiges an Potential für weitere Anwendungen und Arbeiten. So könnte der Client aus einem aktiven funkenden Bluetooth LE-Modul bestehen und z.B. ein stationärer Computer den passiven iBeacon darstellen. Nähert sich nun ein Anwender mit einem solchen aktiven Client dem Computer, so kann der Computer automatisiert entsperrt werden und eine für den Anwender typische Anwendung starten. Ferner wurde in dieser Arbeit nur die Kommunikation über ADs betrachtet. Ein Client und ein iBeacon können sich aber nach dem Empfangen der ADs auch verbinden und damit eine erweiterte Kommunikation durchführen, wodurch eine höhere Anzahl an Informationen ausgetauscht werden kann, als es ein einzelnes AD zulässt.

Die in dieser Arbeit vorgestellten iBeacons lassen sich nur durch händisches Eingreifen konfigurieren. Weitere Arbeiten könnten untersuchen, inwiefern ein Netzwerk aus iBeacons aufgebaut werden kann, um die iBeacons automatisiert und ohne direkten Kontakt zu konfigurieren. Auch eine dynamische und autonome Konfiguration des AD von iBeacons kann in weiterführenden Arbeiten untersucht werden, wodurch iBeacons je nach Situation und Gegebenheit unterschiedliche Nachrichten versenden könnten. Gerade die Kombination aus WLAN und iBeacons könnte für eine benutzerfreundliche Massenkonfiguration weiter betrachtet werden. Um ein besserer Verständnis für die schwankende Signalstärke und Signalqualität von iBeacons zu erhalten, können in weiteren Arbeiten Feldversuche durchgeführt werden, welche Aufschluss darüber geben sollten, welche Objekte die Signale stören und wie z.B. über eine dynamische Berechnung und Adaption des RSSI-Wertes eines einzelnen iBeacons dagegen vorgegangen werden kann.

Mit den Erkenntnissen aus dieser Arbeit können also noch einige Verbesserungen an der Erkennung von Signalen, Erweiterungen von Medical iBeacons, weiterführende Implementierungen der vorgestellten Szenarien und Geschäftsprozesse, Untersuchungen noch nicht betrachteter Netzwerkstrukturen von iBeacons, das Hinzunehmen von weiteren Sensortechnologien und die Implementierung von aktiven statt passiven Clients vorgenommen und umgesetzt werden.

Literatur

[1] *Apple iBeacon*. [Online; aufgerufen 16.08.2014]. URL: https://developer.apple.com/ibeacon/.

[2] *Apple Starts iBeacon Product Certification under Made for the iPhone Program*. [Online; aufgerufen 17.08.2014]. URL: http://appleinsider.com/articles/14/02/25/apple-starts-ibeacon-product-certification-under-made-for-iphone-program.

[3] *Apple Trademark List*. [Online; aufgerufen 16.08.2014]. URL: http://www.apple.com/legal/intellectual-property/trademark/appletmlist.html.

[4] *AristaFlow - AristaFlow Next generation Business Process Management*. [Online; aufgerufen 24.08.2014]. URL: http://www.aristaflow.com.

[5] Bachmeier, A.: Masterarbeit. Wi-Fi based indoor navigation in the context of mobile services. 2013.

[6] *Beacon technology at CeBIT 2014*. [Online; aufgerufen 23.08.2014]. URL: http://www.cebit.de/en/news-trends/trends/mobile/articles/beacon-technology-at-cebit.xhtml.

[7] Bielawa, T. M.: Position location of remote bluetooth devices. Diss. Virginia Polytechnic Institute und State University, 2005.

[8] *Bleu*. [Online; aufgerufen 17.08.2014]. URL: http://bleu.io.

[9] *Bluetooth Basics*. [Online; aufgerufen 16.08.2014]. URL: http://www.bluetooth.com/Pages/Basics.aspx.

[10] *Bluetooth LE Overview*. [Online; aufgerufen 16.08.2014]. URL: http://www.summitdata.com/blog/ble-overview/.

[11] *Bluetooth Low Energy*. [Online; aufgerufen 16.08.2014]. URL: http://www.litepoint.com/wp-content/uploads/2014/02/Bluetooth-Low-Energy_WhitePaper.pdf.

[12] *Bluetooth Low Energy Technology Training*. [Online; aufgerufen 16.08.2014]. URL: http://www.imd.uni-rostock.de/ma/gol/lectures/embedded/Literatur/Low%20Energy%20Training.pdf.

[13] *Bluetooth Low Energy Version 4.0*. [Online; aufgerufen 16.08.2014]. URL: http://home.eng.iastate.edu/~gamari/CprE537_S13/project%20reports/Bluetooth%20LE.pdf.

[14] *Bluetooth SIG Extends Bluetooth Brand, Introduces Bluetooth Smart Marks*. [Online; aufgerufen 16.08.2014]. URL: http://www.bluetooth.com/Pages/Press-Releases-Detail.aspx?ItemID=138.

[15] *Bluetooth Smart*. [Online; aufgerufen 16.08.2014]. URL: http://www.bluetooth.com/Pages/Bluetooth-Smart.aspx.

[16] *Bluetooth Specification Adopted Documents*. [Online; aufgerufen 16.08.2014]. URL: https://www.bluetooth.org/en-us/specification/adopted-specifications.

[17] Buchwald, S., Bauer, T., Pryss, R.: IT-Infrastrukturen für flexible, service-orientierte Anwendungen - ein Rahmenwerk zur Bewertung. In: *13. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW'09)*. Lecture Notes in Informatics (LNI) P-144. Koellen-Verlag, 2009, S. 524–543.

[18] *Carrefour and Nisa track shoppers via Bluetooth beacons in trolleys and baskets*. [Online; aufgerufen 23.08.2014]. URL: http://www.nfcworld.com/2014/06/26/329979/carrefour-nisa-track-shoppers-via-bluetooth-beacons-trolleys-baskets/.

[19] *Cell Tower Triangulation – How it Works*. [Online; aufgerufen 12.08.2014]. URL: http://wrongfulconvictionsblog.org/2012/06/01/cell-tower-triangulation-how-it-works/.

[20] *Core Location Framework Reference*. [Online; aufgerufen 24.08.2014]. URL: https://developer.apple.com/library/ios/documentation/CoreLocation/Reference/CoreLocation_Framework/_index.html.

[21] Dadam, P., Reichert, M., Rinderle-Ma, S., Lanz, A., Pryss, R., Predeschly, M., Kolb, J., Ly, L. T., Jurisch, M., Kreher, U., Goeser, K.: From ADEPT to AristaFlow BPM Suite: A Research Vision has become Reality. In: *Proceedings Business Process Management (BPM'09) Workshops, 1st Int'l. Workshop on Empirical Research in Business Process Management (ER-BPM '09)*. LNBIP 43. Springer, 2009, S. 529–531.

[22] Dadam, P., Reichert, M., Rinderle-Ma, S., Göser, K., Kreher, U., Jurisch, M.: Von ADEPT zur AristaFlow BPM Suite - Eine Vision wird Realität: Correctness by Construction und flexible, robuste Ausführung von Unternehmensprozessen. In: UIB-2009-02. Ulm: University of Ulm, 2009.

[23] Derpanis, K. G.: The harris corner detector. In: *York University* (2004).

[24] Dey, A. K.: Context-aware computing: The CyberDesk project. In: *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*. 1998, S. 51–54.

[25] Dey, A. K.: Providing architectural support for building context-aware applications. Diss. Georgia Institute of Technology, 2000.

[26] Dey, A. K.: Understanding and using context. In: *Personal and ubiquitous computing* 5.1 (2001), S. 4–7.

[27] Dourish, P.: What we talk about when we talk about context. In: *Personal and ubiquitous computing* 8.1 (2004), S. 19–30.

[28] *EasyJet is latest airline testing iBeacons for speedier airport experiences*. [Online; aufgerufen 23.08.2014]. URL: http://www.mobilemarketer.com/cms/news/content/18174.html.

[29] *Ekahau - Business Intelligence Through Location*. [Online; aufgerufen 12.08.2014]. URL: http://www.ekahau.com.

[30] Enenkiel, K.: Diplomarbeit: healthHistory - Konzeption & Implementierung einer mobilen Patientenakte. 2011.

[31] *estimote*. [Online; aufgerufen 17.08.2014]. URL: http://estimote.com.

[32] *Estimote - iBeacon*. [Online; aufgerufen 17.08.2014]. URL: http://media.tumblr.com/90987ae16391f981ab81d8ebe4d501a1/tumblr_inline_mqvtmyR50P1qz4rgp.png.

[33] *Estimote Wants To Pioneer 'Nearables' With New Stickers Beacon Hardware*. [Online; aufgerufen 21.08.2014]. URL: http://techcrunch.com/2014/08/21/estimote-wants-to-pioneer-nearables-with-new-stickers-beacon-hardware/.

[34] *Feature Matching*. [Online; aufgerufen 12.08.2014]. URL: http://graphics.cs.cmu.edu/courses/15-463/2005_fall/www/463.html.

[35] Finkenzeller, K.: *RFID Handbook*. Wiley Online Library, 2003.

[36] *First look: Using iBeacon location awareness at an Apple Store*. [Online; aufgerufen 23.08.2014]. URL: http://appleinsider.com/articles/13/12/06/first-look-using-ibeacon-location-awareness-at-an-apple-store.

[37] *Frequency Hopping Spread Spectrum*. [Online; aufgerufen 16.08.2014]. URL: http://de.wikipedia.org/wiki/Frequency_Hopping_Spread_Spectrum.

[38] *GAP . Introduction to Bluetooth Low Engery . Adafruit Learning System*. [Online; aufgerufen 16.08.2014]. URL: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap.

[39] *GATT . Introduction to Bluetooth Low Engery . Adafruit Learning System*. [Online; aufgerufen 16.08.2014]. URL: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt.

[40] *GATT Profile Overview*. [Online; aufgerufen 12.08.2014]. URL: https://developer.bluetooth.org/TechnologyOverview/Pages/GATT.aspx.

[41] *GATT Specification Documents*. [Online; aufgerufen 12.08.2014]. URL: https://developer.bluetooth.org/gatt/Pages/GATT-Specification-Documents.aspx.

[42] Geiger, P.: Bachelorarbeit: Entwicklung einer Augmented Reality Engine am Beispiel des iOS. 2012.

[43] Geiger, P., Pryss, R., Schickler, M., Reichert, M.: *Engineering an Advanced Location-Based Augmented Reality Engine for Smart Mobile Devices*. Technical Report UIB-2013-09. Ulm: University of Ulm, 2013.

[44] Geiger, P., Schickler, M., Pryss, R., Schobel, J., Reichert, M.: Location-based Mobile Augmented Reality Applications: Challenges, Examples, Lessons Learned. In: *10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps*. 2014, S. 383–394.

[45] *Generic Access Profile*. [Online; aufgerufen 16.08.2014]. URL: https://www.bluetooth.org/en-us/specification/assigned-numbers/generic-access-profile.

[46] *Getting Started with iBeacon*. [Online; aufgerufen 16.08.2014]. URL: https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf.

[47] *Gimbal*. [Online; aufgerufen 17.08.2014]. URL: https://www.gimbal.com.

[48] Goosen, C. A.: Bachelorarbeit: Design and Implementation of a Bluetooth 4.0 LE Infrastructure for Mobile Devices. 2014.

[49] Harris, C., Stephens, M.: A combined corner and edge detector. In: *Alvey vision conference*. Bd. 15. Manchester, UK. 1988, S. 50.

[50] *Heatmap*. http://robotics.usc.edu/~ahoward/projects/wifi/usc-sal200-rss-5cm.jpg. [Online; aufgerufen 12.08.2014].

[51] *How do iBeaconsWork*. [Online; aufgerufen 16.08.2014]. URL: http://www.warski.org/blog/2014/01/how-ibeacons-work/.

[52] *How Stuff Works - Wibree*. [Online; aufgerufen 16.08.2014]. URL: http://www.howstuffworks.com/wibree.htm.

[53] *HRP Profile*. [Online; aufgerufen 12.08.2014]. URL: https://developer.bluetooth.org/TechnologyOverview/Pages/HRP.aspx.

[54] *iBeacon for Developers - Apple Developer*. [Online; aufgerufen 17.08.2014]. URL: https://developer.apple.com/ibeacon/.

[55] *International Telecommunication Union - Frqeuently Asked Questions*. [Online; aufgerufen 16.08.2014]. URL: http://www.itu.int/ITU-R/terrestrial/faq/index.html.

[56] Jablonski, M.: Bachelorarbeit: Technische Konzeption und Realisierung einer mobilen Anwendung zur Verwaltung von Patientendaten für Ärzte am Beispiel von Android. 2013.

[57] Jähne, B.: *Digitale Bildverarbeitung*. Bd. 3. Springer, 1989.

[58] Kanopoulos, N., Vasanthavada, N., Baker, R. L.: Design of an image edge detection filter using the Sobel operator. In: *Solid-State Circuits, IEEE Journal of* 23.2 (1988), S. 358–367.

[59] Knappmeyer, M., Liaquat Kiani, S., Reetz, E., Baker, N., Tönjes, R.: Survey of context provisioning middleware. In: *IEEE Communication Surveys and Tutorials* 15.3 (2013), S. 1492–1519.

[60] *kontak.io*. [Online; aufgerufen 17.08.2014]. URL: http://kontakt.io.

[61] Langer, D.: Masterarbeit: MEDo - Mobile Technik und Prozessmanagement zur Optimierung des Aufgabenmanagements im Kontext der klinischen Visite. 2012.

[62] *NOOBS*. [Online; aufgerufen 18.08.2014]. URL: https://github.com/raspberrypi/noobs/blob/master/README.md.

[63] Parkinson, B. W., Spilker, J. J.: *Progress In Astronautics and Aeronautics: Global Positioning System: Theory and Applications*. Bd. 2. Aiaa, 1996.

[64] Peterson, W. W., Brown, D. T.: Cyclic codes for error detection. In: *Proceedings of the IRE* 49.1 (1961), S. 228–235.

[65] Pryss, R., Musiol, S., Reichert, M.: Collaboration Support Through Mobile Processes and Entailment Constraints. In: *9th IEEE Int'l Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom'13)*. IEEE Computer Society Press, 2013.

[66] Pryss, R., Tiedeken, J., Reichert, M.: Managing Processes on Mobile Devices: The MARPLE Approach. In: *CAiSE'10 Demos*. 2010.

[67] Pryss, R., Langer, D., Reichert, M., Hallerbach, A.: Mobile Task Management for Medical Ward Rounds - The MEDo Approach. In: *1st Int'l Workshop on Adaptive Case Management (ACM'12), BPM'12 Workshops*. LNBIP 132. Springer, 2012, S. 43–54.

[68] Pryss, R., Mundbrod, N., Langer, D., Reichert, M.: Supporting medical ward rounds through mobile task and process management. In: *Information Systems and e-Business Management* (2014).

[69] Pryss, R., Tiedeken, J., Kreher, U., Reichert, M.: Towards Flexible Process Support on Mobile Devices. In: *Proc. CAiSE'10 Forum - Information Systems Evolution*. LNBIP 72. Springer, 2010, S. 150–165.

[70] *Radius Networks*. [Online; aufgerufen 17.08.2014]. URL: http://www.radiusnetworks.com.

[71] *Random UUID Probability of Duplicates*. [Online; aufgerufen 16.08.2014]. URL: http://en.wikipedia.org/wiki/Universally_unique_identifier#Random_UUID_probability_of_duplicates.

[72] *Raspberry Pi*. [Online; aufgerufen 17.08.2014]. URL: http://www.raspberrypi.org.

[73] *Raspbian*. [Online; aufgerufen 18.08.2014]. URL: http://www.raspbian.org.

[74] *RedBearLab*. [Online; aufgerufen 17.08.2014]. URL: http://redbearlab.com.

[75] Reichert, M., Dadam, P., Rinderle-Ma, S., Lanz, A., Pryss, R., Predeschly, M., Kolb, J., Ly, L. T., Jurisch, M., Kreher, U., Goeser, K.: Enabling Poka-Yoke Workflows with the AristaFlow BPM Suite. In: *Proc. BPM'09 Demonstration Track*. CEUR Workshop Proceedings 489. 2009.

[76] *RFID Tag*. [Online; aufgerufen 12.08.2014]. URL: http://upload.wikimedia.org/wikipedia/commons/6/6f/Transponder2.jpg.

[77] Robecke, A., Pryss, R., Reichert, M.: DBIScholar: An iPhone Application for Performing Citation Analyses. In: *CAiSE Forum-2011*. Proceedings of the CAiSE'11 Forum at the 23rd International Conference on Advanced Information Systems Engineering Vol-73. CEUR Workshop Proceedings, 2011.

[78] Schäuffele, S.: Bachelorarbeit: Integration von "Location-based Mobile Augmented Reality Tasksïn eine Business Process Management Umgebung. 2014.

[79] Schobel, J., Schickler, M., Pryss, R., Maier, F., Reichert, M.: Towards Process-Driven Mobile Data Collection Applications: Requirements, Challenges, Lessons Learned. In: *10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps*. 2014, S. 371–382.

[80] Tolmie, P., Pycock, J., Diggins, T., MacLean, A., Karsenty, A.: Unremarkable computing. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 2002, S. 399–406.

[81] Waldenmaier, T.: Bachelorarbeit: Entwicklung einer In-House-AR-Navigation unter Verwendung von AREA und Wifinder. 2014.

[82] Want, R.: An introduction to RFID technology. In: *Pervasive Computing, IEEE* 5.1 (2006), S. 25–33.

[83] Want, R., Fishkin, K. P., Gujar, A., Harrison, B. L.: Bridging physical and virtual worlds with electronic tags. In: *CHI*. Bd. 99. 1999, S. 370–377.

[84] *What is the iBeacon Bluetooth Profile*. [Online; aufgerufen 16.08.2014]. URL: http://stackoverflow.com/questions/18906988/what-is-the-ibeacon-bluetooth-profile.

[85] Zhang, D., Xia, F., Yang, Z., Yao, L., Zhao, W.: Localization technologies for indoor human tracking. In: *Future Information Technology (FutureTech), 2010 5th International Conference on*. IEEE. 2010, S. 1–6.

[86] *ZigBee Modul*. [Online; aufgerufen 12.08.2014]. URL: http://upload.wikimedia.org/wikipedia/commons/f/f8/Arduino_xbee.JPG.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Name: Philip Geiger Matrikelnummer: 700561

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Philip Geiger