Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Sebastian Wilkes

Ein Verzeichnisdienst für kontextgebundene Informationen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Sebastian Wilkes

Ein Verzeichnisdienst für kontextgebundene Informationen

Bachelorarbeit eingereicht im Rahmen der Prüfungsordnung Technische Informatik 2008

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Martin Becke
Zweitgutachter: Prof. Dr. Birgit Wendholt

Eingereicht am: 12. Juli 2018

Sebastian Wilkes

Thema der Arbeit
Ein Verzeichnisdienst für kontextgebundene Informationen

Stichworte
Kontext, Beacon, Signale, Wi-Fi, Bluetooth, Verzeichnisdienst, Standortbasierter Dienst

Kurzzusammenfassung
Kontextbewusste Anwendungen bilden eine immer bedeutendere Schnittstelle zwischen der physikalischen und digitalen Welt. Neben der Zeit, der Identität und der Aktivität wird auch der Standort des Nutzers in die Anwendungslogik miteinbezogen. Möglichkeiten für Nutzer zu einem kontextgebundenen Informationsaustausch mit anderen Nutzern am Standort, existieren jedoch nur in eingeschränkter Form. Klassische Methoden im Umgang mit Kontextinformationen, wie die Auswertung der Position mittels GPS-Koordinaten oder die Auswertung von Beacons, liefern nur begrenzte Einsatzmöglichkeiten. Die berechneten Koordinaten von Nutzern sind für einen kontextgebundenen Informationsaustausch ungeeignet, da im Vorfeld Gültigkeitsbereiche definiert werden müssen, um sie mit den Koordinaten von potenziellen Empfängern vergleichen zu können. Beacons, die durch ihren Senderadius einen Gültigkeitsbereich schaffen, stehen aufgrund von proprietären Informationen der Anbieter häufig nur für einen begrenzten Nutzerkreis zur Verfügung.

Diese Arbeit präsentiert das Design und die Implementierung von einem signalbasierten Verzeichnisdienst zum Austausch kontextgebundener Informationen. Durch den Dienst können mobile Anwendungen Signale nutzen, um anderen Empfängern kontextrelevante Informationen dauerhaft bereitzustellen. Der Verzeichnisdienst verzichtet auf eine Positionsbestimmung und ermöglicht den Austausch von Kontextinformationen, stattdessen anhand eindeutiger Signalinformationen aus lokalen Signalquellen, wie u. a. WLAN-Accesspoints oder Bluetooth-Beacons. Durch eine einheitliche und eindeutige Signalbeschreibung sind Nutzer in der Lage Signalinformationen als Schlüssel zum Informationsaustausch vor dem Hintergrund eines örtlichen Kontexts zu nutzen.

Dazu werden im ersten Schritt Verfahren zur Kontextermittlung verglichen. Im zweiten Schritt wird der Verzeichnisdienst entwickelt. Um das System hinsichtlich Skalierbarkeit, Effizienz und Datenverfügbarkeit zu optimieren, wird eine entsprechende Architektur entwickelt, die ein geeignetes Datenbanksystem und eine geeignete Web-API integriert.

Der Dienst bietet die Grundlage für eine Kommunikation zwischen den Empfängern eines lokalen Signals. Eine Schnittstelle des Betriebssystems zu den Signalinformationen ist

eine essentielle Voraussetzung für die Nutzung des Dienstes auf der Anwendungsebene. Die Möglichkeiten Signalinformationen auszulesen, sind bei iOS stärker eingeschränkt als bei Android, was bei der Entwicklung berücksichtigt werden muss. Gegenüber der Nutzung von GPS-Koordinaten ist eine starke Abhängigkeit zum Senderadius und ein erhöhtes Ausfallrisiko der Signalquelle zu beachten.

Das Konzept hat das Potenzial, sowohl den Kontakt zwischen fremden Menschen zu fördern, als auch die individuelle Relevanz der Information für den Alltag der Nutzer zu erhöhen. Im Rahmen eines abschließenden Experiments werden die Bedingungen für den signalbasierten Informationsaustausch an öffentlichen Standorten untersucht.

Sebastian Wilkes

Title of the paper

A directory service for context-related information

Keywords

Context, Signals, Beacon, Wi-Fi, Bluetooth, Directory service, Location-based service

Abstract

Context-aware applications are becoming an important interface between the physical and digital world. In addition to time, identity and activity, the user's location is included in the application logic. However, opportunities for users to contextually exchange information with other users at the location exist only to a limited extent. Classic methods for context determination, such as the evaluation of the position using GPS coordinates and the use of beacon information, provide only limited application possibilities. The calculated coordinates of users are not suitable for a context-based exchange of information, since in order to be able to compare them with the coordinates of potential recipients, areas of validity must be defined. Beacons that create a range of validity due to their sender radius are often only available to a limited group of users due to provider proprietary information.

This work presents the design and implementation of a signal-based directory service for exchanging contextual information. The service allows mobile applications to use signals to provide contextually relevant information. The service dispenses with a position determination and allows the exchange of contextual information instead by using unique signal information from local signal sources such as Wi-Fi access points or Bluetooth beacons. Through a unified and unambiguous signal description, users are able to use signal information as a key to exchange information within the local context.

For this purpose, in the first step, different methods for context determination are compared. In the second step, the directory service is developed, which allows applications to provide information to the receivers of a signal. To optimize the system in terms of scalability, efficiency and data availability, an appropriate architecture is being developed integrating a suitable database system and a suitable web api.

The service provides the basis for communication between the receivers of a local signal. An interface of the operating system to the signal information is an essential requirement for the use of the service at the application level. The possibilities to read out signal information are more limited on iOS than on Android, what needs to be considered in the development. Compared with the use of GPS coordinates there is a strong dependency to the transmitter radius and an increased risk of failure of the signal source.

The concept has the potential to increase the contact between strangers as well as to increase the individual relevance of the information for the everyday life of users. The work concludes with an experiment examining the conditions for the signal-based information exchange at public sites.

Inhaltsverzeichnis

1. Einleitung **1**
- 1.1. Motivation . . . 1
  - 1.1.1. Kontextsensitivität . . . 3
- 1.2. Konzeptskizze . . . 5
- 1.3. Ziele und Abgrenzung . . . 5
- 1.4. Aufbau der Arbeit . . . 7

2. Grundlagen **9**
- 2.1. Arten von standortbezogenen Diensten . . . 9
- 2.2. Standortkategorien . . . 10
- 2.3. Kontextebenen . . . 11
  - 2.3.1. Beispiel HAW Hamburg . . . 11
- 2.4. Möglichkeiten zur Ermittlung des örtlichen Kontexts . . . 14
  - 2.4.1. Kontextermittlung durch Positionsbestimmung . . . 14
  - 2.4.2. Kontextermittlung durch Auswertung von Beacons . . . 18
- 2.5. Zusammenfassung . . . 21

3. Verwandte Arbeiten **22**
- 3.1. MagicMap . . . 22
  - 3.1.1. Bewertung . . . 22
- 3.2. Place Lab und POLS . . . 23
  - 3.2.1. Place Lab Architektur . . . 24
  - 3.2.2. POLS Software . . . 25
  - 3.2.3. Bewertung . . . 25
- 3.3. Standort APIs . . . 26
  - 3.3.1. Bewertung . . . 28
- 3.4. Google Beacon Platform . . . 29
  - 3.4.1. Eddystone . . . 29
  - 3.4.2. Nearby API . . . 29
  - 3.4.3. Proximity Beacon API . . . 31
  - 3.4.4. Physical Web . . . 31
  - 3.4.5. Bewertung . . . 31
- 3.5. Zusammenfassung . . . 32

4. Analyse und Design **34**
- 4.1. Anforderungen an die signalbasierte Kontextermittlung . . . 34

4.2. Anforderungen an das Gesamtkonzept . . . 35
4.2.1. Anforderungen an den Anwendungsserver . . . 35
4.2.2. Anforderungen an die graphische Nutzerschnittstelle . . . 36
4.2.3. Schritte für das Empfangen und Senden von Informationen . . . 36
4.3. Mögliche Szenarien für den signalbasierten Informationsaustausch . . . 37
4.4. Anforderungsanalyse für den Verzeichnisdienst . . . 38
4.4.1. Grundfunktion . . . 38
4.4.2. Stakeholder . . . 38
4.4.3. Use Cases . . . 38
4.4.4. Nicht-Funktionale Anforderungen . . . 39
4.4.5. Anforderungen an die Signalbeschreibung . . . 40
4.4.6. Anforderungen an die Kontextbeschreibung . . . 41
4.4.7. Begriffsdefinitionen . . . 44
4.4.8. Funktionale Anforderungen . . . 44
4.4.9. Anforderungen an die Verzeichnisstruktur . . . 45
4.5. Systementwurf . . . 46
4.5.1. Architekturmodell . . . 46
4.5.2. Datenmodell . . . 47
4.5.3. Datenbanksystem . . . 48
4.5.4. Kommunikation . . . 53
4.5.5. Bearbeitung von Anfragen . . . 59
4.5.6. Skalierbare Serverarchitektur . . . 62
4.6. Zusammenfassung . . . 65

5. Implementierung **67**
5.1. Erzeugung einer verteilten Datenbank mit Cassandra . . . 67
5.2. Erstellung von Methoden zur Bearbeitung der API-Anfragen . . . 68
5.3. Erstellung eines Node.js Clusters über die verfügbaren Kerne . . . 69

6. Diskussion **71**
6.1. Experiment zum Empfang von Beacons an öffentlichen Standorten . . . 71
6.1.1. Thema . . . 71
6.1.2. Vermutung . . . 71
6.1.3. Versuchsaufbau . . . 71
6.1.4. Durchführung . . . 76
6.1.5. Ergebnisse . . . 76
6.1.6. Bewertung . . . 79
6.2. Übersicht zur Eignung von Signalinformation zum Informationsaustausch . . . 80

7. Fazit **81**
7.0.1. Ausblick . . . 82

A. Appendix **83**
A.1. Programmcode des Verzeichnisdientes 83
A.1.1. Programmcode zur Initialisierung 83
A.1.2. Programmcode zur Bearbeitung von API-Anfragen 84
A.2. Inhalt der CD 91

Glossar **100**

Listings

3.1. Mozilla Location Service: Beispiel für ein HTTP-Request 27
3.2. Mozilla Location Service: Beispiel für ein HTTP-Response 28

4.1. REST-API: Beispiel einer API-Referenz mittels JSON Hyper-Schema [1] 55
4.2. REST-API: Hinzufügen eines Eintrags eines Eigentümers in einer Ebene 55
4.3. REST-API: Entfernen eines Eintrags eines Eigentümers in einer Ebene 56
4.4. REST-API: Anfrage der Einträge eines Eigentümers in einer Ebene 56
4.5. REST-API: Anfrage der Einträge aller Eigentümer in einer Ebene 56
4.6. REST-API: Anfrage der Eigentümer in einer Ebene anhand von IDs 57
4.7. REST-API: Anfrage aller Eigentümer in einer Ebene 57
4.8. REST-API: Löschen eines Eigentümers in einer Ebene 57
4.9. REST-API: Löschen einer Ebene 58

5.1. Cassandra: Konfiguration des Keyspace 67
5.2. Cassandra: Erstellung der Verzeichnis-Tabelle 68
5.3. Node.js Cluster Module: Definition der Cluster Klasse [2] 69
5.4. Node.js Cluster Module: Definition der Worker Klasse [2] 70

A.1. Programmcode: Installation verwendeter Bibliotheken 83
A.2. Programmcode: Konfiguration des Servers 83
A.3. Programmcode: Definition des Namensraums 83
A.4. Programmcode: Definition der Tabelle 83
A.5. Programmcode: Erstellung des Keyspace und der Tabelle 84
A.6. Programmcode: Bearbeitung der Anfrage nach Einträgen aller Eigentümer in einer Ebene 84
A.7. Programmcode: Bearbeitung der Anfrage nach Einträgen bestimmter Eigentümer in einer Ebene 85
A.8. Programmcode: Bearbeitung der Anfrage zum Hinzufügen eines Eintrages von einem Eigentümer in einer Ebene 86
A.9. Programmcode: Bearbeitung der Anfrage zum Entfernen eines Eintrages von einem Eigentümer in einer Ebene 87
A.10. Programmcode: Bearbeitung der Anfrage zum Entfernen eines Eigentümer in einer Ebene 88
A.11. Programmcode: Bearbeitung der Anfrage zum Entfernen einer Ebene 89
A.12. Programmcode: Bearbeitung der Anfrage zum Abruf einer vom Eigentümer in einer Ebene gesetzten Bezeichnung 90

A.13. Programmcode: Bearbeitung der Anfrage zum Aktualisieren einer vom Eigentümer in einer Ebene gesetzten Bezeichnung 90

Abbildungsverzeichnis

1.1. Konzeptskizze zur Verwendung des Verzeichnisdientes 6

2.1. Einordnung von Standortbeschreibungen in Kontextebenen 12
2.2. Auszug Lageplan HAW Hamburg [3] 13
2.3. Einordnung von Standortbeschreibungen am Beispiel der HAW Hamburg . . . 14

3.1. Architektur des MagicMap Clients 23
3.2. Hauptkomponenten in der Place Lab Architektur [4] 24
3.3. POLS Demonstration aus dem Jahr 2005 [5] 25

4.1. Veranschaulichung einer graphischen Nutzerschnittstelle 36
4.2. Abfragehierarchie innerhalb einer Signalbeschreibung 41
4.3. Informationsaustausch auf gleicher Kontextebene 42
4.4. Abfragehierarchie innerhalb einer Kontextbeschreibung 43
4.5. Relationsmodell 46
4.6. Schichtenmodell 47
4.7. Datenmodell 47
4.8. Spaltenfamilien-Datenmodell (in Anlehnung an [6]) 49
4.9. Cassandra Datenmodell 52
4.10. Aktivitätsdiagramm zum Erstellen oder Aktualisieren eines Eintrages 59
4.11. Aktivitätsdiagramm zum Lesen von Einträgen 60
4.12. Aktivitätsdiagramm zum Löschen eines Eintrages 61
4.13. Node.js Event Schleife [7] 63
4.14. Architektur mit Lastverteilung auf verfügbare Kerne der CPU 64
4.15. Architektur mit Proxy zur Lastverteilung auf mehrere Maschinen 65

5.1. Sequenzdiagramm: Anfrage für alle Einträge eines Eigentümers in einer Ebene 69

6.1. Hinweis zum Wi-Fi-Hotspot auf dem Roncalliplatz in Köln [8] 73
6.2. Ansicht auf den Starbucks vom Vorplatz des Hauptbahnhofs in Köln 74
6.3. Messpunkte auf dem Roncalliplatz in Köln 75
6.4. Messpunkte im Starbucks am Hauptbahnhof in Köln 75

Tabellenverzeichnis

2.1. Arten von standortbezogenen Diensten [9] 9
2.2. Ergebnisse eines Place Lab Experiments zur Nutzerzeit Abdeckung [4] 16
2.3. Verfahren zur Positionsbestimmung (eigene Darstellung) 18
2.4. Möglichkeiten von iOS und Android IDs unbekannter Geräte auszulesen . . . 20

4.1. Signalbeschreibung 40
4.2. Kontextbeschreibung 41

6.1. Senden und Empfangen von Wi-Fi-Signalinformationen am Roncalliplatz . . . 77
6.2. Senden und Empfangen von iBeacons und Eddystone-Beacons am Roncalliplatz 77
6.3. Senden und Empfangen von Wi-Fi-Signalinformationen im Starbucks 78
6.4. Senden und Empfangen von iBeacons und Eddystone-Beacons im Starbucks . 78
6.5. Übersicht zur Eignung von Signalinformation zum Informationsaustausch . . 80

1. Einleitung

1.1. Motivation

Das Smartphone nimmt eine immer bedeutendere Rolle im alltäglichen Leben ein. Während im Jahr 2015 der Anteil der Deutschen, die täglich unterwegs das Internet nutzten noch bei 18 Prozent lag, waren es im Jahr 2017 bereits 30 Prozent [10]. Dabei ermöglicht die Verfügbarkeit von Informationen eine flexiblere Planung und Spontanität [11]. Mit diesem Trend gewinnen situative, kontextgebundene Informationen an Bedeutung, was die Entwicklung von kontextbewussten Anwendungen vorantreibt. Der Standort ist ein Bestandteil des primären Kontexts und trägt neben der Zeit, der Identität und der Aktivität zum Kontextbewusstsein der Anwendung bei [12].

Das Standortbewusstsein wird dabei über Methoden der Positionsbestimmung oder der Signalidentifizierung hergestellt. Eine verbreitete Methode zur Positionsbestimmung ist die Auswertung von GPS-Signalen [13]. Das Satellitennavigationssystem GPS [1] erzielt eine hohe Genauigkeit, ermöglicht eine Positionsbestimmung überall auf der Erde und wird durch Umwelteinflüsse nur geringfügig beeinflusst. Nachteilig ist dagegen der schlechte Empfang innerhalb von Gebäuden, der eine Positionsbestimmung verhindert [14]. Aus diesem Grund haben sich Alternativen etabliert, die auf den Empfang von Informationen aus lokalen Signalquellen basieren.

Um Signale empfangen zu können, verfügen moderne Smartphones über mehrere Schnittstellen. Neben dem Empfang von GPS-Daten, ermöglichen sie dem Nutzer den Zugang zu Netzwerken, darunter das Wide Area Network (WAN), in dem beispielsweise der Mobilfunk eingesetzt wird, das Wireless Local Area Network (WLAN), das in der Regel eine Verbindung zum Internet herstellt, und das Personal Area Network (PAN), zu dem Bluetooth, die Infrarot-Übertragung IrDA [2] sowie NFC [3] und RFID [4] gehören. Im Wesentlichen unterscheiden sich die

[1] GPS: Global Positioning System
[2] IrDA: Infrared Data Association
[3] NFC: Near-Field-Communication
[4] RFID: Radio-Frequency-Identification

Signaltypen neben den Protokollspezifikationen in ihrer Datenübertragungsrate und ihrem Senderadius [15].

Diese Signalquellen können neben GPS zur Positionsbestimmung eingesetzt werden, so dass sowohl außerhalb als auch innerhalb von Gebäuden standortbezogene Dienste angeboten werden können. Zur Ermittlung von Koordinaten mit Hilfe von eindeutigen Signalinformationen existiert bereits eine breite Auswahl an Lösungen. Es ist möglich, Daten aus unterschiedlichen Signalquellen an Dienste, wie z. B. *Mozilla Location Service* [5] zu senden, die daraufhin eine Positionsbestimmung durchführen und entsprechende Koordinaten zurückliefern [16].

Koordinaten liefern zwar eine Erkenntnis über den Standort des Nutzers, jedoch erfordert die Zuordnung zu einem spezifischen örtlichen Kontext einen erhöhten Aufwand. Nur mit Hilfe im Vorfeld festgelegter Gültigkeitsbereiche, z. B. in einem kartesischen Koordinatensystem durch Geofencing [6] kann geprüft werden, ob die Angaben einem Kontext zugewiesen werden können. Die Notwendigkeit dieser Vorarbeit macht einen spontanen standortbezogenen Informationsaustausch zu einem aufwendigen Schritt.

In diesem Fall bietet sich die Nutzung von Beacons [7] an, da diese bereits einen Gültigkeitsbereich durch den Senderadius definieren und aufgrund eindeutiger Signalinformationen einen Schlüssel für den Austausch zwischen den Empfängern liefern.

Die Nutzung von Beacons kommt bereits in proprietären Anwendungen zum Einsatz, um Nutzer mit gezielten Informationen bezüglich bestimmter Angebote zu versorgen [18]. Es existiert jedoch kein anwendungsübergreifendes Verzeichnis für Kontextinformationen, das nicht auf der Auswertung von Koordinaten basiert und den unmittelbaren Austausch zwischen den Empfängern ermöglicht.

In dieser Arbeit werden die Vor- und Nachteile der Methoden zur Kontextermittlung gegenübergestellt. Auf der Grundlage der Erkenntnisse wird ein signalbasierter Verzeichnisdienst entwickelt, mit dem mit geringem Aufwand signalbezogene Kontextinformationen bereitgestellt werden können. Der Dienst vereinfacht die Entwicklung kontextsensitiver Anwendungen, in dem er einen einfachen Weg zum standortbezogenen Informationsaustausch zwischen Signalempfängern ermöglicht und die vorhandene Signalinfrastruktur ausnutzt. Diese Arbeit beschreibt die Analyse, das Design, die Implementierung des Verzeichnisdienstes und ermittelt die Bedingungen zur Nutzung von Signalinformationen an öffentlichen Standorten.

[5] Mozilla Location Service, https://www.mozilla.org (aufgerufen am: 26.06.2018)

[6] Geofencing beschreibt eine Methode zur Positionsbestimmung durch das Überschreiten einer definierten Begrenzung [17]

[7] Beacon (engl. für Leuchtfeuer) beschreibt ein Signal, das regelmäßig ausgesendet wird

In den folgenden Abschnitten dieses Kapitels werden zum Verständnis die Begriffe *Standortbewusstsein* und *Kontext* gegenübergestellt und die Motivation zum standort- und kontextbasierten Informationsaustausch näher erläutert.

1.1.1. Kontextsensitivität

Es wurde in der Motivation bereits erwähnt, dass der Standort ein wesentlicher Bestandteil des Kontextes ist. In diesem Abschnitt wird eine grundlegende Definition des Begriffs „Kontext" dargestellt und erläutert, welche Bedeutung die Sprache auf das Standort- und Kontextbewusstsein hat. Diese Klärung ist wichtig, um zu verstehen, wie mehrere Nutzer sich mit Hilfe einer Beschreibung, die einen Standort mit einem Kontext verknüpft, verständigen können.

Was ist ein Kontext?

„Der Kontext ist jede Information, die benutzt werden kann, um die Situation einer Entität zu beschreiben. Eine Entität ist eine Person, ein Standort oder ein Objekt, das als relevant für die Interaktion zwischen einem Nutzer und einer Anwendung sowie für den Nutzer und für die Anwendung selbst, angesehen wird. Dabei sind Standort, Identität, Aktivität und Zeit die primären Kontexttypen, um die Situation einer bestimmten Entität zu beschreiben." [12]

Vereinfacht dargestellt, bezieht sich der Kontext demnach auf die Situation eines Standorts, einer Person oder eines Objekts sowie deren Beziehungen zueinander. So ist es einerseits möglich den Kontext von Personen anhand der Standortinformation zu beschreiben. Andererseits kann beispielsweise auch die Beziehung zu einem Objekt Aufschluss über den Standort und den Kontext von Personen liefern.

In der Praxis vereinen Anwendungen im Rahmen einer Nutzer-Authentifizierung bereits die Information über die Identität, die Aktivität und die Zeit. Abhängig von der Kernfunktion der Anwendung ist der Standortbezug entweder eine Voraussetzung, optional oder nicht vorhanden.

Im nächsten Abschnitt wird erläutert, wie es zu einem Standortbewusstsein kommt und welche Vorteile sich daraus ergeben.

Standortbewusstsein und Sprache

„Visuelle, auditive und taktile Reize, aber auch Erfahrung unser eigenen Bewegung im Raum (Propriozeption), führen dazu, dass wir eine räumliche Repräsentation aufbauen, die dann wiederrum als Grundlage weiterer kognitiver Prozesse fungiert. Mit anderen Worten: Sprechen über

die gegenständliche Welt setzt komplexe Übertragungsprozesse zwischen sprachlichen und nicht sprachlichen Repräsentationen voraus.“ [19]

In der Sprache prägen Begriffe die Vorstellung von Realität [20]. Die einfachste Form einen räumlichen Kontext zu beschreiben, ist der Begriff ‚Raum‘, der einen dreidimensionalen geometrischen Körper beschreibt [21].

In der Sprache wird der Raumbegriff mit der Beschreibung von Tätigkeiten und Nutzungen kombiniert, wie z. B. bei dem Begriff ‚Wohnraum‘. Der Begriff ‚Wohnzimmer‘ zeigt als Teilbeschreibung eines Wohnraums eine weitere Eingrenzung der räumlichen Dimension durch die Verwendung von Sprache. Begriffe wie z. B. ‚Küche‘ können in diesem Zusammenhang als alternative Synonyme für die Beschreibung von Räumen mit speziellen Nutzungen angesehen werden, die ebenfalls dem Kontext des Wohnraums zugeordnet werden.

Das Beispiel zeigt, dass das Standortbewusstsein einer Person von den Begriffen der Standortbeschreibung abhängt. Begriffe erzeugen bestimmte Assoziationen die den Kontext implizit beschreiben.

Als ein weiteres Beispiel kann ein Flughafen genannt werden. Das Gebäude, das Rollfeld sowie die Start- und Landebahn werden für einen Besucher dabei u. a. als Kontext des Flughafens wahrgenommen, da dieser den Flächen während der unterschiedlichen Phasen des Aufenthalts begegnet. Das Gate und das dortige Warten auf das Boarding stellen dabei einen untergeordneten Kontext dar, da die Nutzung der Fläche auf eine neue Aktivität ausgelegt ist.

In beiden Beispielen ergibt sich der Kontext aus der mit dem Standort verbundenen Tätigkeit und kann sehr grob bis sehr fein aufgelöst sein. Mit anderen Worten: Umso mehr Kontextbezug die Standortbeschreibung aufweist, desto konkreter wird die Vorstellung von dem Umfeld und den Tätigkeiten, die mit dem Standort verknüpft sind, übermittelt.

In einer standortbewussten Anwendung können kontextrelevante Informationen sowohl von der Anwendung aufbereitet sein, wie es zum Beispiel bei standortbezogenen Wetterdaten der Fall ist, als auch Nachrichten von anderen Nutzern im selben Kontext darstellen. In beiden Fällen ist es wichtig, dem Nutzer den Kontextbezug der Information über eine geeignete Beschreibung zu vermitteln.

Standortbezogener Informationsaustausch

Menschen, die sich im selben Kontext befinden, haben in der Regel ein gemeinsames standortbezogenes Interesse und einen gemeinsamen Informationsbedarf. Ein Beispielszenario liefert das Gate am Flughafen; eine Gruppe von Menschen wird über Lautsprecheransagen und digitale Anzeigetafeln über Informationen bezüglich ihres Fluges informiert. Die Menschen der Gruppe, das Gate und der anstehende Flug bilden u. a. Bestandteile eines Kontexts (s. Abschnitt 1.1.1).

Die Menschen stellen damit potenzielle Interessenten für kontextgebundene Informationen dar. Die Möglichkeiten zum Austausch beschränken sich jedoch im Wesentlichen auf die verbale Kommunikation. Gängige soziale Plattformen wie Twitter [8] und Facebook [9] bieten hier keine digitale Alternative, da der Standortbezug von Nachrichten lediglich mit verfügbaren Adressen hergestellt werden kann, was in diesem Fall eine zu grobe Auflösung bedeuten würde. Dazu kommt erschwerend, dass dem Austausch über soziale Netze eine Freundschaftsanfrage bzw. ein bewusstes ‚Folgen' einer Person vorangehen muss, um Nachrichten zu abonnieren.

1.2. Konzeptskizze

Die in Abbildung 1.1 dargestellte stark vereinfachte Konzeptskizze, zeigt zwei Aktoren, die mit Hilfe der verfügbaren Informationen in einem gemeinsamen Kontext mit Hilfe eines zentralen Verzeichnisdienstes Daten austauschen. Aktor 1 sendet Daten mit einer eindeutigen Information aus dem Kontext an den Verzeichnisdienst. Andere Aktoren können daraufhin mit Hilfe der eindeutigen Information die hinterlegten Daten abrufen. Die Daten können z. B. Informationen zum Zugriff auf entfernte Ressourcen beinhalten.

1.3. Ziele und Abgrenzung

Die Möglichkeit zum kontextgebundenen Informationsaustausch stellt eine Ergänzung des bisherigen Angebots für den sozialen Austausch dar, der sich schwerpunktmäßig auf ortsbezogene, kontextgebundene Inhalte fokussiert.

Diese Arbeit beantwortet die Frage, welche eindeutige Information aus einem Kontext für den kontextgebundenen Informationsaustausch geeignet ist und wie die Bereitstellung von Informationen über einen zentralen Verzeichnisdienst umgesetzt werden kann. Damit grenzt sich der Dienst von Lösungen ab, die einen Informationsaustausch zwischen Nutzern ermöglicht, die sich in der Nähe zueinander befinden. Die Information wird mit dem Kontext verknüpft und steht anderen Empfängern zur Verfügung.

Es wird die Annahme getroffen, dass sich eine für den Kontext eindeutige Information anhand von Signalen ableiten lässt. Außerdem wird angenommen, dass das Wissen über den Standort und den Kontext über eine deskriptive Standortbeschreibung vom Anbieter der Information vermittelt und ohne Koordinaten beschrieben werden kann.

[8] Twitter, https://twitter.com (aufgerufen am: 21.03.2018)
[9] Facebook, https://facebook.com (aufgerufen am: 21.03.2018)

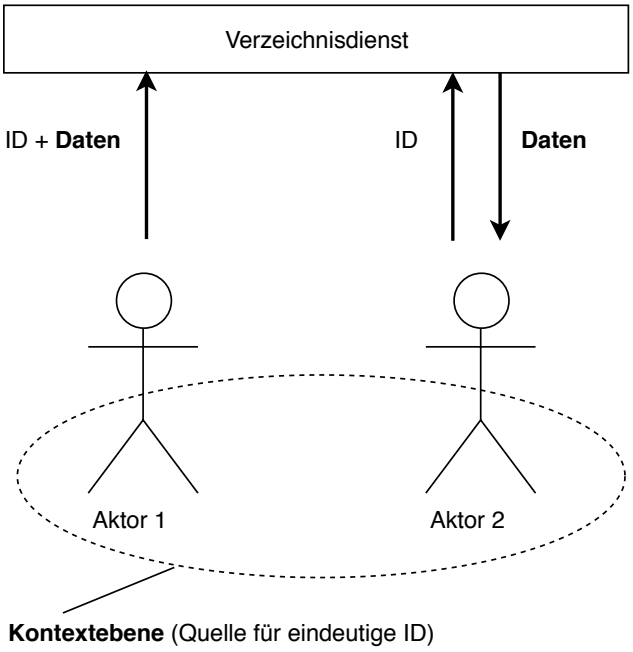

Abbildung 1.1.: Konzeptskizze zur Verwendung des Verzeichnisdientes

Damit ein Informationsaustausch zwischen mobilen Endgeräten stattfinden kann, werden in der Konzeption sowohl die Möglichkeiten zum Empfang relevanter Informationen für Android- als auch für iOS-Geräte untersucht. Andere Betriebssysteme werden nicht berücksichtigt. Zum Verständnis wird ein Gesamtkonzept dargestellt, in das der Verzeichnisdient eingeordnet wird. Es wird keine Implementierung einer Anwendung zur Nutzung der Schnittstelle des Verzeichnisdientes vorgestellt.

Der Beitrag dieser Arbeit lässt sich in drei Teilleistungen unterteilen:

Analyse zu den Eignungen von Signalinformationen zum Informationsaustausch Es werden Verfahren analysiert, die eine Ermittlung des örtlichen Kontexts ermöglichen. Die Erkenntnisse liefern die Grundlage für die Konzeption des kontextgebundenen Informationsaustauschs.

Entwicklung eines kontextgebundenen Verzeichnisdienstes Es wird eine Verzeichnisstruktur entwickelt, die eine Speicherung von IDs und Kontextinformationen ermöglicht.

Der Verzeichnisdienst ermöglicht das Bereitstellen und Abfragen von Kontextinformationen in einem verteilten System.

Experiment zu den Bedingungen an öffentlichen Standorten Durch ein Experiment werden die Bedingungen für den kontextgebundenen Informationsaustausch an öffentlichen Standorten untersucht. Es werden vorhandene Bedingungen ausgewertet und mit Alternativen verglichen.

1.4. Aufbau der Arbeit

Kapitel 2: Grundlagen - Das nächste Kapitel vermittelt ein grundlegendes Verständnis für standortbasierte Dienste und ordnet die Arbeit in den Themenbereich ein. Dazu werden die Begriffe Standort, Position und Kontext und deren Zusammenhänge erläutert. Es wird die Beziehung zwischen dem Standort und sich überlagernden Kontextebenen an einem Beispiel erklärt. Um das Verständnis für die vorgestellten verwandten Arbeiten zu erleichtern, werden Verfahren dargestellt, die für die Kontext- und Standortermittlung in standortbasierten Diensten relevant sind.

Kapitel 3: Verwandte Arbeiten - In diesem Kapitel werden verwandte Arbeiten dargestellt, die durch das Auswerten von Beacons eine Positionsbestimmung erreichen, Informationen referenzieren oder unabhängig vom Kontext eine Kommunikation zwischen Nutzern ermöglichen. Die Arbeiten werden kurz dargestellt und bewertet. Es folgt eine Zusammenfassung und Abgrenzung zum Thema dieser Arbeit.

Kapitel 4: Analyse und Design - In diesem Kapitel wird die Analyse und das Design des Verzeichnisdienstes dargestellt. Zunächst wird die Methode der Kontextermittlung festgelegt und daraus resultierende Anwendungsszenarien aufgezeigt. Es folgt die Analyse der Anforderungen für den Verzeichnisdienst. Dabei werden relevante Signal- und Kontexteigenschaften festgelegt und in einer Verzeichnisstruktur gebündelt. Im Systementwurf wird das Architektur- und Datenmodell festgelegt und ein Konzept zur Persistierung, Skalierung und Kommunikation entwickelt. Um die Voraussetzungen für eine nachvollziehbare Implementierung zu schaffen, wird im Verlauf des Kapitels die Wahl der Technologien begründet.

Kapitel 5: Implementierung - In diesem Kapitel wird die Implementierung des Designs beschrieben. Es werden die relevanten Programmabschnitte erläutert, die für den Betrieb des Servers,

der Nutzung des Datenbanksystems und zur Bearbeitung der API-Anfragen notwendig sind.

Kapitel 6: Diskussion - In diesem Kapitel werden in einem Experiment die Bedingungen für den signalbasierten Informationsaustausch an öffentlichen Standorten untersucht.

Kapitel 7: Fazit - In diesem Kapitel wird die Thesis kurz zusammengefasst und reflektiert in welchem Maß die aufgestellten Anforderungen erfüllt werden. Im Ausblick werden Möglichkeiten vorgeschlagen, das Thema weiter zu erforschen.

2. Grundlagen

Um das Verständnis für die Hintergründe dieser Arbeit zu erleichtern, werden in diesem Kapitel grundlegende Informationen zu standortbezogenen Diensten bereitgestellt. Dazu wird eine Einordnung des Themas in das Spektrum standortbezogener Dienste vorgenommen, die Begriffe Standort, Position und Kontext und deren Zusammenhänge erläutert und die Beziehung zwischen dem Standort und sich überlagernden Kontextebenen an einem Beispiel erklärt. Abschließend werden Verfahren dargestellt, die für die Kontextermittlung in standortbasierten Diensten genutzt werden.

2.1. Arten von standortbezogenen Diensten

Die Arten der unterschiedlichen standortbezogenen Dienste lassen sich wie folgt zusammenfassen:

Kategorie	**Hauptnutzen**
Trackingdienste	Ortung von Personen oder Objekten
Navigationsdienste	Ortung und Navigation zu fixen Zielen oder zu Anbietern betimmter Dienste oder Produkte
Informationsdienste	Bereitstellung ortsbezogener Informationen, ggf. mit Möglichkeit zur direkten Reaktion
Kommunikationsdienste	Erleichterung der Kommunikation zwischen Privatpersonen oder Personal
Unterhaltungsdienste	Schaffung eines Unterhaltungswertes durch Anpassung an den Aufenthaltsort des Nutzers
Transaktionsdienste	Initiierung und/oder Ausführung ökonomischer Transaktionen

Tabelle 2.1.: Arten von standortbezogenen Diensten [9]

Die in der Einleitung geschilderte Motivation machte bereits deutlich, dass der Fokus dieser Arbeit auf der standortbasierten Bereitstellung von Informationen liegt, der den Informationsaustausch zwischen Empfängern ermöglicht. Damit kann der Dienst den Bereichen *In-*

formationsdienste und *Kommunikationsdienste* zugeordnet und von den anderen Kategorien abgegrenzt werden.

Um das Verständnis für die vorgestellten verwandten Arbeiten und die Einordnung dieser Arbeit zu erleichtern, werden in den folgenden Abschnitten Verfahren dargestellt, die für die Kontext- bzw. Standortermittlung in standortbasierten Diensten relevant sind. Da diese Verfahren mit unterschiedlichen Standortbeschreibungen arbeiten, werden die Unterschiede im folgenden Abschnitt kurz vorgestellt.

2.2. Standortkategorien

Bei einem Standort unterscheidet Küpper [22] in drei Kategorien; dem deskriptiven, räumlichen und netzwerkbezogenen Standort.

Deskriptive Standortbeschreibung Eine deskriptive Standortbeschreibung bezieht sich auf geographische Gegebenheiten, die entweder eines natürlichen Ursprungs sind, wie z. B. Berge, Flüsse und Landschaften oder künstlicher Natur, wie z. B. Länder, Straßen, Gebäude und Räume. So kann man einerseits als Standort den Gipfel eines Berges beschreiben und anderseits sich mit Hilfe von Adressen aus Straßennamen und Hausnummern orientieren.

Räumliche Standortbeschreibung Als räumlicher Standort kann ein dreidimensionaler Punkt im euklidischen Raum verstanden werden. Üblicherweise wird dafür der Begriff Position verwendet. Die Position wird in der Regel mit Hilfe von zwei- oder dreidimensionalen Koordinaten in Form eines Vektors beschrieben, in denen jeder Wert die Position in einer Dimension darstellt. Koordinaten liefern die Grundlage für ein akkurate und hochpräzise Standortbeschreibung und sind in der Navigation unerlässlich. Um eine Mensch-Maschine-Schnittstelle zu schaffen, werden deskriptive Standortbeschreibungen auf Koordinaten abgebildet und ermöglichen damit die Realisierung standortbasierter Anwendungen.

Netzwerkbezogene Standortbeschreibung In Kommunikationsnetzen werden Knoten in hierarchischen Topologien mit Hilfe von eindeutigen Adressen beschrieben. Ein Beispiel sind IP-Adressen im Internet oder Telefonnummern im Mobilfunknetz.

Zusammenhang zwischen den Standortbeschreibungen Eine wichtige Funktion von standortbezogenen Diensten ist die Zuordnung zwischen den unterschiedlichen Kategorien [22] von Standorten. Eine Positionsbestimmung, die einen räumlichen oder netzwerkbezogenen Standort liefert, muss einer deskriptiven Standortbeschreibung zugeordnet werden, um

für den Nutzer interpretierbar zu sein. Auf der anderen Seite müssen deskriptive Standortbeschreibungen in räumliche oder netzwerkbezogene Beschreibungen übersetzt werden, um sie in Relation zu anderen Standorten zu stellen, zum Beispiel um die Entfernung zwischen zwei Standorten zu berechnen [23].

Viele Dienste basieren auf einer Positionsbestimmung und verknüpfen die ermittelte Position oder netzwerkbezogene Information mit den Kontextinformationen. Dabei werden räumliche Datenbanken wie z. B. ein *Geographic Information System* (GIS) oder netzwerkbezogene Datenbanken wie OpenCellId [1] verwendet.

Deskriptive Standortbeschreibungen, die einen Kontext (s. Abschnitt 1.1.1) beschreiben, können als sich zum Teil überlagernde Ebenen beschrieben werden. Im folgenden Abschnitt wird der Begriff der Kontextebene eingeführt und am Beispiel der Hochschule für Angewandte Wissenschaften (HAW) in Hamburg erläutert.

2.3. Kontextebenen

Eine Kontextebene beschreibt in dieser Arbeit eine Standortbeschreibung, die Kontextinformationen enthält. Mit der Hilfe einer Kontextebene können zusätzliche Informationen mit dem Kontext eines Standorts in Verbindung gebracht werden. Bei einem Informationsaustausch lässt sich die thematische Einordnung durch den Bezug zu einer Kontextebene herstellen.

Dabei ist eine Kontextebene in den meisten Fällen in mehrere übergeordnete Kontextebenen eingebettet (s. Abbildung 2.1).

Die Position bildet als genauste Standortangabe den Kern der Beschreibungsmöglichkeiten eines Standorts. Anhand der Positionsdaten können die übergeordneten Kontexte ermittelt werden. Standortbasierte Dienste die Koordinaten verwenden, verfügen über ein breites Spektrum von Einsatzmöglichkeiten. Die Beschreibung eines Kontexts lässt hingegen in der Regel keinen Rückschluss auf die genauen Positionsdaten zu und kann damit nicht unmittelbar für die Darstellung auf Karten eingesetzt werden.

Damit wird eine Korrelation zwischen der Standortgenauigkeit und dem Kontextumfang deutlich. Umso ungenauer die Standortbeschreibung, desto größer ist die Anzahl von kontextbezogenen Entitäten, die in die Beschreibung miteinbezogen werden (s. Abschnitt 1.1.1)

2.3.1. Beispiel HAW Hamburg

Um die Theorie an einem Praxisbeispiel zu veranschaulichen, wird in diesem Abschnitt ein Lageplan der HAW Hamburg betrachtet.

[1] OpenCellId von Unwired Labs, https://opencellid.org (aufgerufen am: 21.03.2018)

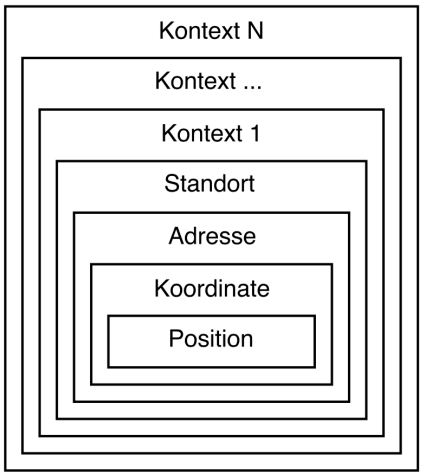

Abbildung 2.1.: Einordnung von Standortbeschreibungen in Kontextebenen

Der Lageplan in Abbildung 2.2 zeigt kontextrelevante Informationen zur HAW Hamburg. Zu sehen sind unterschiedliche Straßennamen und Hausnummern, Gebäudebezeichnungen, Hinweise auf öffentliche Verkehrsmittel und Angebote wie ein Fitnessstudio und ein Copyshop. Die zusammenhängende Grundfläche der Gebäude der HAW ist grün markiert. Bestimmte Gebäudenutzungen wie Bibliotheken oder die Mensa sind gekennzeichnet.

In diesem Beispiel soll verdeutlicht werden, wie die unterschiedlichen Kontextebenen zusammenhängen und welche Bedeutung das für die Identifizierung des Standorts und des Kontexts hat.

Die Abbildung 2.3 zeigt, wie sich der Kontext der Mensa in übergeordnete Kontextebenen einordnet. Die Farbwahl orientiert sich am Lageplan aus Abbildung 2.2. Spricht man über die „Mensa HAW Hamburg“, werden mehrere Kontextebenen vereint; die Mensa, die Hochschule und die Stadt Hamburg. Die Beschreibung vermittelt eine konkrete Vorstellung über die Nutzung, das Thema und implizit über den Standort. Dabei hängt die Zuordnung des Standortes von dem Hintergrundwissen der jeweiligen Person ab. Personen, die bereits am Standort waren, haben eine konkretere Vorstellung von dem Standort als andere. Bei einem standortbezogenen Informationsaustausch zwischen Nutzern im Kontext der genannten Mensa-Beschreibung sind beispielsweise Themen wie Öffnungszeiten, Angebote, Preise und Bewertungen des Essens denkbar.

Das Beispiel zeigt, dass eine Verständigung über einen Kontext nicht ausschließlich auf der Grundlage von Koordinaten erfolgen kann. Erst eine kontextbezogene visuelle oder deskriptive Beschreibung eines Standorts ermöglicht die Interpretation seitens der Nutzer und stellt damit

Abbildung 2.2.: Auszug Lageplan HAW Hamburg [3]

die wesentliche Grundlage für standortbasierte Dienste dar. Das Beispiel zeigt auch, dass es zweitrangig ist, ob die Standortbeschreibung durch eine Positionsbestimmung mit Hilfe einer Koordinate oder auf anderem Wege zugeordnet werden konnte.

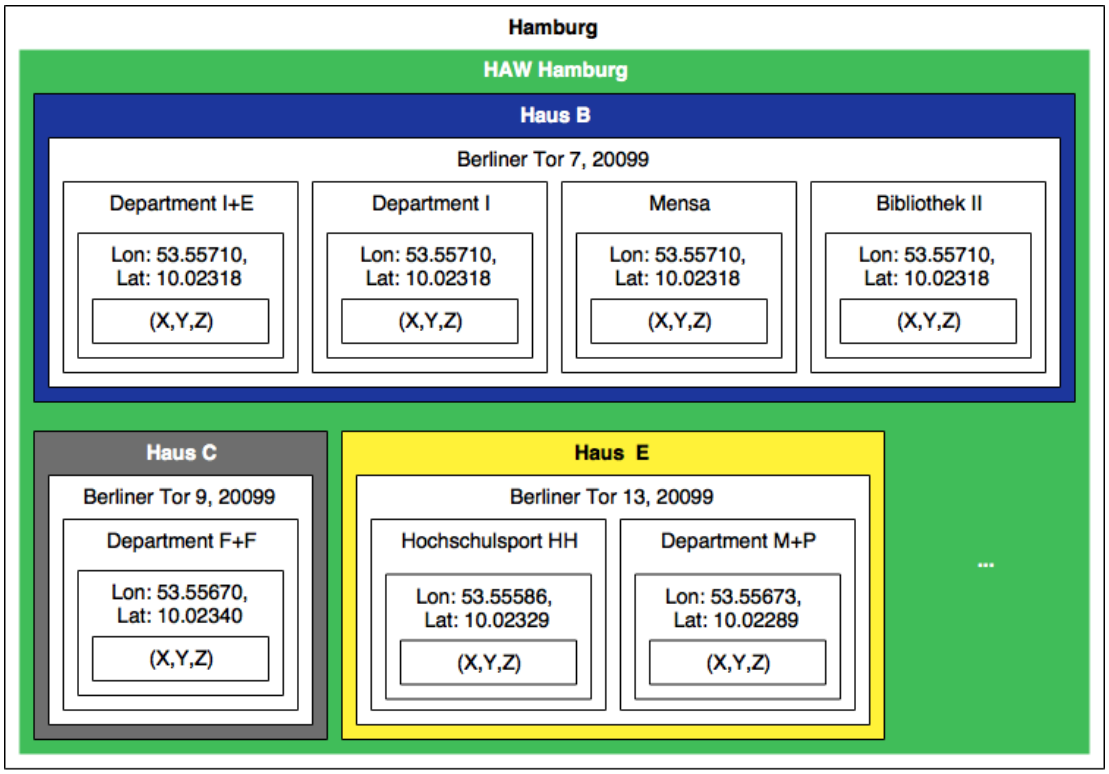

Abbildung 2.3.: Einordnung von Standortbeschreibungen am Beispiel der HAW Hamburg

2.4. Möglichkeiten zur Ermittlung des örtlichen Kontexts

Eine Gegenüberstellung der Methoden zur Kontextermittlung hinsichtlich ihrer Eigenschaften und Anforderungen im Umgang mit Signalen, liefert wichtige Erkenntnisse für den Entwurf des Verzeichnisdienstes.

2.4.1. Kontextermittlung durch Positionsbestimmung

Auf Geofencing [17] basierende Verfahren prüfen, ob sich eine Koordinate innerhalb der Fläche eines zuvor definierten Polygons aus Koordinatenpunkten befindet.

Dadurch können die mit dem Polygon verknüpften standort- oder kontextbezogenen Informationen der Koordinate zugeordnet und bei einer Anfrage zurückgeliefert werden. Neben der Definition von eigenen Bereichen, lassen sich mit Reverse Geocoding APIs [2] Koordinaten in deskriptive Standortbeschreibungen (s. Abschnitt 2.2), wie z. B. Straßennamen, übersetzen.

[2] OpenStreetMap Reverse Geocoding API, http://nominatim.openstreetmap.org (aufgerufen am: 03.07.2018)

Die Google Android Developer Plattform empfiehlt bei der Nutzung von Geofencing, die Genauigkeit auf ein Minimum zu beschränken. Je höher die Genauigkeit der Positionsbestimmung ist, desto höher ist der Akkuverbrauch des ausführenden Geräts [24]. Die damit einhergehende Ungenauigkeit hinsichtlich der Positionsbestimmung, wirkt sich entsprechend auf den Standortbezug des Kontextes aus. Ein Vorteil beim Geofencing ist, dass der Bereich überall festgelegt und sofort ohne Hardwareeinsatz verwendet werden kann [25].

Sind die mit dem Polygon verknüpften kontextbezogenen Informationen eindeutig, können sie als Informationsschlüssel für den Informationsaustausch verwendet werden. Damit stellt die Nutzung eines Geofencing-Verfahrens grundsätzlich eine Möglichkeit dar, einen kontextgebundenen Informationsaustausch auf der Grundlage von eindeutigen standortbezogenen Informationen zu realisieren. Die Anforderung an die Anwendung beschränkt sich dabei auf die Ermittlung der Koordinaten. Im folgenden Abschnitt werden daher relevante Methoden zu Positionsbestimmung erläutert.

Positionsbestimmung über Satelliten

Die Satellitentechnik hat die Positionsbestimmung in der Vergangenheit revolutioniert und bis heute geprägt. Zunächst wurden nur im militärischen Bereich Funksignale von Satelliten zur Positionsbestimmung verwendet. Später wurden Signale für die Zivilgesellschaft freigegeben [26]. Neben dem amerikanischen GPS [3] werden auch das russische GLONASS [4], und das europäische GALILEO [5] sowie andere Systeme aus Asien von mobilen Geräten unterstützt [27]. Um die Signale zur Positionsbestimmung zu nutzen, ist keine Anmeldung oder Datenaustausch mit dem Satelliten erforderlich. Damit sind Satelliten-Daten universell einsetzbar und attraktiv für den Einsatz in mobilen Anwendungen. Nachteilig beim Einsatz von satellitengestützter Positionsbestimmung sind hohe Investitionskosten. Zudem können Signalstörungen, z. B. durch Signalreflektionen von Gebäuden die Positionsgenauigkeit negativ beeinflussen [28].

Positionsbestimmung über Mobilfunk

Um hohe Installationskosten zu vermeiden, können bereits vorhandene drahtlose Netzwerke verwendet werden. Die Auswertung von Netzwerksignalen erlaubt zudem die Positionsbestim-

[3] GPS, Global Positioning System: Satellitensystem des amerikanischen Militärs, https://www.gps.gov (aufgerufen am: 06.06.2018)

[4] GLONASS, Global Navigation Satellite System: Satellitensystem des russischen Militärs

[5] Galileo ist ein europäisches globales Satellitennavigations- und Zeitgebungssystem unter ziviler Kontrolle. Das System befindet sich im Aufbau und ist teilweise im Einsatz, https://www.gsa.europa.eu/european-gnss/galileo/galileo-european-global-satellite-based-navigation-system (aufgerufen am: 06.06.2018)

mung in Gebäuden, was mit einer satellitengestützten Positionsbestimmung nicht möglich ist [14]. Bei den zellularen Netzwerken, wie das Mobilnetz GSM [6] oder UMTS [7] , kann aus der Identifikation der Funkzelle die Position des Endgerätes ermittelt werden kann.

Positionsbestimmung über WLAN

Das Funknetz WLAN [8] bietet weitere Möglichkeiten zur Positionsbestimmung. Auch hier kann auf eine bestehende Infrastruktur zugegriffen werden. Gegenüber der Positionsbestimmung über Mobilfunk und Satelliten ermöglicht die WLAN-Nutzung eine kostengünstige Positionsbestimmung mit hoher Präzision und besserer Abdeckung von sowohl außerhalb als auch innerhalb von Gebäuden liegender Bereiche. Die Humboldt-Universität zu Berlin hat beispielsweise mit der Softwarelösung MagicMap bereits 2004 ein mobiles System entwickelt, das auf einer WLAN-Signalstärken-Auswertung basiert (s. Abschnitt 3.1).

Vergleich Signalabdeckung

Um die Positionsbestimmung via GPS, GSM und WLAN (802.11) miteinander zu vergleichen, haben Intel Research, die University of California San Diego und die University of Washington unter Verwendung der Place Lab Software (s. Abschnitt 3.2) ein Experiment durchgeführt. Es wurden drei Testpersonen mit unterschiedlichen Berufen ausgewählt, deren Position mit jedem Verfahren für jeweils zehn Stunden laufend bestimmt wurde. Die nachfolgende Tabelle zeigt die Ergebnisse der Studie:

	GPS		**GSM**		**802.11**	
Testsubjekt	Abdeckung	Ø-Ausfall	Abdeckung	Ø-Ausfall	Abdeckung	Ø-Ausfall
Immunologe	12.8%	68 min	100%	-	87.7%	1.6 min
Hausfrau	0.6%	78 min	98.7%	2 min	95.8%	1 min
Verkäufer	0%	171 min	100%	-	100%	-
Durchschnitt	4.5%	105 min	99.6%	1 min	94.6%	1.3 min

Tabelle 2.2.: Ergebnisse eines Place Lab Experiments zur Nutzerzeit Abdeckung [4]

Die Ergebnisse dieses Experiments bestätigen, dass bei satellitengestützter Positionsbestimmung via GPS insbesondere durch den schlechten Empfang in Gebäuden mit einer geringen Abdeckung und hohen Ausfallzeiten zu rechnen ist. GSM und WLAN haben eine bessere Abdeckung und geringe Ausfallzeiten und können daher unter den Bedingungen der Studie

[6] GSM: Global System for Mobile Communications
[7] UMTS: Universal Mobile Telecommunications System
[8] WLAN: Wireless Local Area Network, IEEE Funktstandard 802.11

eine höhere Zuverlässigkeit garantieren. Die Studie zeigt außerdem, dass die Verfügbarkeit von GSM- und WLAN-Signalen in einem vergleichbaren urbanen Umfeld hoch ist.

Bei der Bewertung der Ergebnisse muss berücksichtigt werden, dass sich die Testpersonen während des Experiments in einem urbanen Umfeld aufhalten, so dass die Vorteile von satellitengestützter Positionsbestimmung in abgelegeneren Gebieten ohne Mobilfunk- und WLAN-Empfang nicht zum Tragen kommen. Es werden in den Ergebnissen keine Heuristiken berücksichtigt, mit denen Positionsbestimmungen verbessert oder Ausfallzeiten durch Schätzungen überbrückt werden können.

Im nächsten Abschnitt werden die zugrundeliegenden Verfahren zur Positionsbestimmung vorgestellt.

Verfahren zur Positionsbestimmung

Die unterschiedlichen Verfahren zur Positionsbestimmung [29] [30] orientieren sich in erster Linie an den Anforderungen an die Genauigkeit. Dabei sind je nach Verfahren unterschiedliche Parameter für die Berechnungen notwendig. Um das Ergebnis positiv zu beeinflussen, werden teilweise Daten weiterer Sensoren hinzugezogen und mögliche Fehlerpotenziale berücksichtigt.

- **Proximity Sensing (Prox.):** Die Position eines Mobilgeräts wird von den Koordinaten der Basisstation abgeleitet. Dabei wird für das Mobilgerät bei Empfang des Signals der Basisstation die Position der Basisstation angenommen. Alternativ kann die Basisstation die Annäherung über ein Signal des Mobilgeräts erkennen. Da bei diesem Verfahren eine Identifizierung der Signalquelle stattfinden muss, wird es auch für das Auslösen von kontextbezogenen Aktionen genutzt (s. Abschnitt 2.4.2).

- **Entfernungsmessung (Entf.):** Durch die Messung der Entfernungen zu Basisstationen in Form von absoluten Entfernungen (zirkulär) oder Entfernungsänderungen (hyperbolisch) wird die Position des zu bestimmenden Ziels berechnet. Die Entfernung ergibt sich dabei entweder aus Zeitmessungen oder der Signalstärke des empfangenen Signals (engl.: received signal strength (RSS)). Die Messung von drei Basisstation wird auch Trilateration genannt.

- **Winkelmessung (Wink.):** Durch die Messung der Winkel mittels Reihen aus Antennen an der Basisstation oder am Mobilgerät, wird die Position des Mobilgeräts berechnet. Das Fehlerpotential ergibt sich über die Ungenauigkeit der Winkelmessung. Die Messung von drei Basisstation wird auch Triangulation genannt.

- **Inertiale Messung (Inertial):** Eine inertiale Messung bedeutet eine Kombination aus unterschiedlichen Sensordaten, die es ermöglicht aus der zuletzt bekannten Position des Mobilgeräts die aktuelle Position abzuleiten.
- **Hybride Ansätze (Hybrid):** Es ist möglich verschiedene Verfahren zu kombinieren, um die Schwächen einzelner Verfahren zu Gunsten einer höheren Genauigkeit auszugleichen. Ein Beispiel ist die Erweiterung von Proximity Sensing mit einer Entfernungs- oder Winkelmessung. Ein Vorteil dieser Variante ist, dass die Messung weiterhin von nur einer Basisstation abhängt. Mit mehreren Basisstationen erreichen die Entfernungs- oder Winkelmessungsverfahren jedoch weiterhin eine deutlich höhere Genauigkeit.

Die aufgeführten Verfahren zur Positionsbestimmung lassen sich anhand folgender Kriterien unterscheiden:

	Prox.	**Entf.**	**Wink.**	**Inertial**	**Hybrid**
Berechnungsaufwand	sehr gering	mittel	hoch	hoch	hoch
Anzahl Signalquellen	1	>= 2	>= 2	>= 2	>= 1
Sensordatenfusion	nein	nein	nein	ja	möglich
Genauigkeit	gering	hoch	hoch	mittel	mittel/sehr hoch
Fehlerpotential	nein	ja	ja	ja	ja
Bewegungssensitiv	nein	ja	ja	ja	ja

Tabelle 2.3.: Verfahren zur Positionsbestimmung (eigene Darstellung)

2.4.2. Kontextermittlung durch Auswertung von Beacons

Um Beacons identifizieren zu können, senden sie in regelmäßigen Abständen die Adresse der Signalquelle z. B. in Form einer MAC-Adresse [31] und andere Informationen, wie z. B. IDs aus. Nutzer müssen sich im Sinne des Proximity Sensings (s. Abschnitt 2.4.1) innerhalb eines bestimmten Radius zur Signalquelle befinden, um die Informationen empfangen zu können. Sind Kontextinformationen in Bezug zur Signalquelle bekannt, können diese mit der signalbezogenen Adresse oder ID in einer Datenbank gespeichert und abgerufen werden.

Es existieren unterschiedliche Signaltypen, die für den Nutzer mit Hilfe entsprechender Scanvorgänge eines Mobilgeräts [9] sichtbar gemacht werden können.

Proximity Sensing wird z. B. im Marketing eingesetzt, um proprietäre Informationen am Verkaufspunkt (engl.: Point of sale) zu empfangen. Diese proprietären Informationen aus lokalen Signalquellen können in der Regel nur von spezifischen Anwendungen interpretiert

[9] Mobilgerät: tragbares Endgerät wie Smartphone und Laptop

werden. Damit ist die Nutzbarkeit von derartigen Informationen für andere Anwendungen stark eingeschränkt.

WLAN-Accesspoints versenden Beacons, um Empfängern die Verfügbarkeit des Netzwerks anzuzeigen. Diese können für unterschiedliche Verfahren zur Standortermittlung herangezogen werden [32]. Daneben existieren u. a. Bluetooth- oder RFID-Beacons zur Bereitstellung von standortbezogenen Signalinformationen [33].

Der Gültigkeitsbereich einer Information wird durch die Reichweite des Signals definiert. Der Senderadius von WLAN-Accesspoints fällt gegenüber Bluetooth Low-Energie Beacons größer aus, so dass durch die Wahl des Signaltyps eine unterschiedliche Abdeckung erreicht werden kann. Im Gegensatz zu Wi-Fi-Accesspoints können Bluetooth-Beacons so konfiguriert werden, dass mehrere Signalquellen dieselbe ID aussenden.

Die Betriebssysteme Android [10] und iOS [11] bieten Anwendungsentwicklern Schnittstellen für einen zum Teil eingeschränkten Zugriff auf Netzwerkinformationen an. Während bei Android detaillierte Informationen zu WLAN-Accesspoints mit Hilfe der *WifiManager* Klasse ausgelesen werden können [34], wird bei iOS für den vollen Zugriff eine Erlaubnis für die Nutzung der *NEHotspotHelper* Klasse [35] benötigt. Ohne entsprechende Erlaubnis ist bei iOS der Zugriff auf Informationen aus der aktuellen WLAN-Verbindung über die Methoden der Bibliothek *CaptiveNetwork.h* beschränkt [36].

Auch die MAC-Adressen aus den empfangenen Signalen von Bluetooth-Geräten können bei Android über den *BluetoothLeScanner* [12] ausgelesen werden. Bei iOS muss dagegen eine Kopplung mit dem Bluetooth-Gerät bestehen, damit über das *Core Bluetooth* Framework [13] die MAC-Adresse ausgelesen werden kann.

Neben der eindeutigen MAC-Adresse, können alternativ statische IDs aus Bluetooth-Beacons, die das iBeacon [14] - oder Eddystone [15] -Format unterstützen zur Identifizierung des Kontextes eingesetzt werden. Dazu kann für Android-Anwendungen z. B. die Bibliothek *Android Beacon*

[10] Android Betriebssystem von Google, https://www.android.com (aufgerufen am: 23.03.2018)
[11] iOS Betriebssystem von Apple, https://www.apple.com/de/ios/ios-11/ aufgerufen am: 23.03.2018
[12] BluetoothLeScanner, https://developer.android.com/reference/android/bluetooth/le/BluetoothLeScanner (aufgerufen am: 28.06.2018)
[13] Core Bluetooth Framework, https://developer.apple.com/documentation/corebluetooth (aufgerufen am: 28.06.2018)
[14] iBeacon, https://developer.apple.com/ibeacon/ (aufgerufen am: 15.06.2018)
[15] Eddystone, https://github.com/google/eddystone (aufgerufen am: 15.06.2018)

Library [16] verwendet werden. Für iOS-Anwendungen eignet sich die Bibliothek proximitykit-ios [17].

Administratoren von Bluetooth-Beacons können mehrere Signalquellen mit der gleichen eindeutigen ID konfigurieren. Damit kann eine Information über mehrere Signalquellen an unterschiedlichen Standorten bereitgestellt werden.

Bei der Verwendung von iBeacons können gleiche kontextbezogene UUIDs [18] verwendet und die Major und Minor-Werte zur Identifikation des Beacons genutzt werden. Im Eddystone-Format wird für den Kontextbezug der Namespace und zur Identifikation des Beacons die Instance ID genutzt.

Die Proximity Beacon API [37] der Google Beacon Plattform (s. Abschnitt 3.4) erlaubt das Registrieren von iBeacons und Eddystone-Beacons. Nach einer Registrierung durch den Administrator der Beacons ist es möglich, über Anhänge (engl.: Attachments) Daten mit dem Beacon zu verknüpfen. Mit der Nearby Messaging API [38] stellt Google eine Schnittstelle für iOS- und Android-Geräte zur Verfügung, mit welcher berechtigte Anwendungen Attachments empfangen können. Die Nearby Notifications API [39] ermöglicht Android-Geräten ab der Android-Version 4.4, das Empfangen von Beacon-bezogenen Benachrichtigungen mit Verweisen zu Android-Anwendungen oder Webseiten und benötigt dafür keine Installation einer Anwendung.

System	WLAN-MAC	BLE-MAC	iBeacon-UUID	Eddystone-UID	Attachment
Android	Ja	Ja	Ja	Ja	Ja
iOS	Nein	Nein	Ja	Ja	Ja

Tabelle 2.4.: Möglichkeiten von iOS und Android IDs unbekannter Geräte auszulesen

Die Auswertung von Bluetooth-Beacons führt im Vergleich zum Geofencing zu einem geringeren Akkuverbrauch beim Mobilgerät. Ein weiterer Vorteil liegt in der Möglichkeit die Signale innerhalb von Gebäuden zu nutzen. Dafür muss jedoch im Gegensatz zum Geofencing Hardware am Standort platziert werden und eine entsprechende Erlaubnis vorhanden sein. Beacons eignen sich für den Einsatz in kleineren Bereichen und nicht für die Abdeckung großer Bereiche. Sie können in sich bewegenden Fortbewegungsmitteln eingesetzt werden, in denen eine Positionsbestimmung schwierig ist [25].

[16] AltBeacon GitHub Projekt: android-beacon-library, https://github.com/AltBeacon/android-beacon-library (aufgerufen am: 28.06.2018)
[17] Proximity Kit von RadiusNetworks, https://github.com/RadiusNetworks/proximitykit-ios (aufgerufen am: 28.06.2018)
[18] UUID: Universally Unique Identifier

2.5. Zusammenfassung

Ein Nachteil von GPS gegenüber netzwerkgestützter Positionsbestimmung ergibt sich insbesondere durch den eingeschränkten Empfang des GPS-Signals in Gebäuden. Ein Nachteil von koordinatenbasierter Positionsbestimmung ist, dass die Positionsbestimmung in sich bewegenden Fortbewegungsmitteln eine hohe Abtastrate oder aufwendige Abschätzungen erforderlich macht. Wird die Information ausschließlich genutzt, um einen Kontextbezug zum Fortbewegungsmittel herzustellen, ist ein Proximity Sensing zur Auswertung der Signalinformationen einer mitgeführten Signalquelle besser geeignet.

Um standortbezogene Informationen bereitzustellen kann entweder eine Koordinate mittels Geofencing einem Kontextbereich zuordnet oder die Signalinformation eines Beacons ausgenutzt werden. Da die Verfahren sehr unterschiedliche Vor- und Nachteile haben, lassen sie sich effektiv kombinieren. Während Geofencing insbesondere für größere Bereiche eingesetzt werden kann, lassen sich kleinere Bereiche innerhalb von Gebäuden oder Fahrzeugen mit Beacons abdecken.

Während beim Geofencing eine Koordinate auf unterschiedlichen Wegen ermittelt werden kann, hängen die Möglichkeiten zur Auswertung von Signalinformationen stark von den erlaubten Zugriffen der Software der jeweiligen Empfangsgeräte ab.

Wie in Tabelle 2.4 dargestellt, werden die Möglichkeiten zum Empfang von eindeutigen Signalinformationen aus unbekannten Quellen von den Einschränkungen des iOS-Systems bestimmt. Ein Austausch zwischen iOS- und Android-Geräten unter gleichen Voraussetzungen ist demnach nur auf der Grundlage einer übereinstimmenden, statischen ID aus der jeweiligen Verbindung zu einem Wifi-Accesspoint, der Kopplung zu einem Bluetooth-Gerät, den Informationen von Bluetooth-Beacons oder dem Inhalt von Attachments möglich.

3. Verwandte Arbeiten

Um das Thema dieser Arbeit in den Stand der Forschung einordnen zu können, werden relevante Projekte und Angebote dargestellt.

3.1. MagicMap

MagicMap [40] ist ein kooperatives, hybrides System zur Positionsbestimmung mobiler Geräte, das 2004 an der Humboldt-Universität zu Berlin entwickelt wurde. Die Positionsbestimmung basiert dabei auf Trilateration (s. Abschnitt 2.4.1) und Auswertung der WLAN-Signalstärke. MagicMap unterstützt WLAN, RFID und ZigBee [1], ist jedoch darauf ausgelegt auch andere verfügbare Funknetzwerk-Technologien zu integrieren. MagicMap ist eine reine Softwarelösung, die bei mobilen Geräten außer einer standardmäßigen WLAN-Ausstattung keine weitere Hardware benötigt. Die WLAN-Accesspoints können beliebig verteilt sein und es sind keine Eingriffe an der Hardware oder Software erforderlich.

Die Besonderheit ist das Peer-to-Peer-Netzwerk, das den Austausch von relevanten Daten unter allen Teilnehmern ermöglicht. Mit Hilfe des Netzwerks lassen sich Fehler aufgrund schwankender Sende- und Empfangsleistungen korrigieren und die Kalibrierungszeit verringern. Teilnehmer des Peer-to-Peer-Netzwerks können ihren Standort freigeben und anderen Nutzern auf deren Karte anzeigen lassen. Bei Versuchen auf dem Universitätscampus Adlershof in Berlin erreichte die Positionsbestimmungen eine Genauigkeit zwischen fünf und zehn Metern.

Die Software steht zum freien Download zur Verfügung [41].

3.1.1. Bewertung

MagicMap bietet ein Beispiel für das intelligente Ausnutzen vorhandener Infrastruktur, wodurch eine hohe Abdeckung und hohe Genauigkeit erreicht werden kann. Gleichzeitig stellt das Konzept hohe Anforderungen an die Community, die zur Optimierung des Systems beitragen muss. Zudem wird für Berechnungen eine leistungsstarke System-Infrastruktur und es

[1] ZigBee, http://www.zigbee.org (aufgerufen am: 02.07.2018)

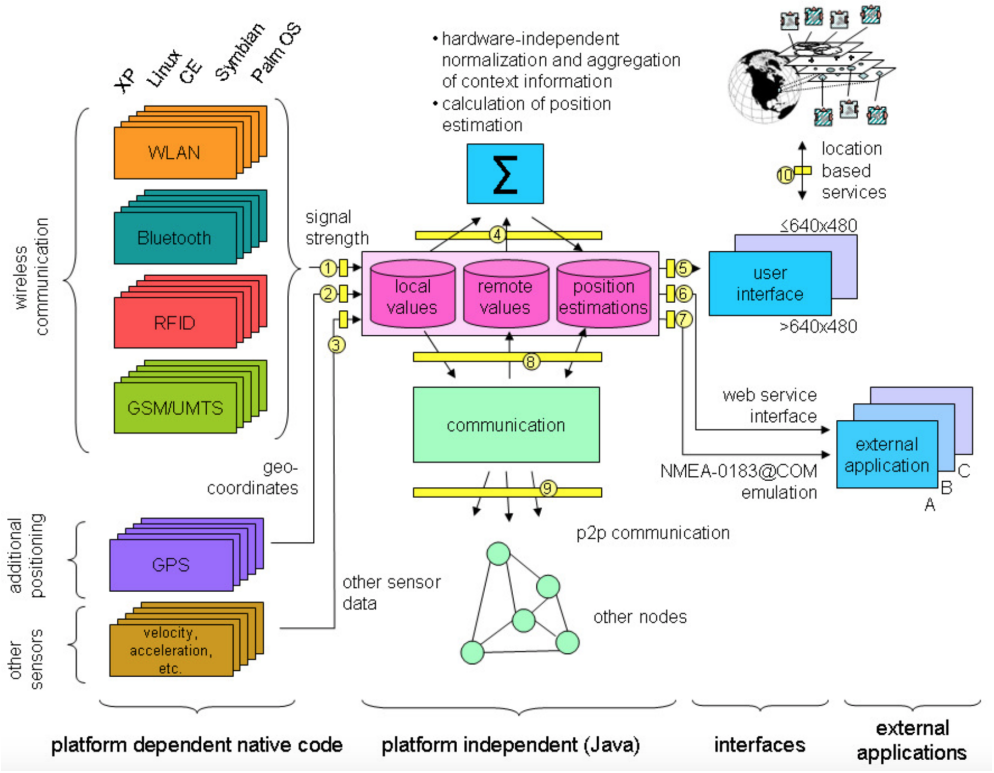

Abbildung 3.1.: Architektur des MagicMap Clients

müssen mindestens drei Accesspoints vorhanden sein, damit eine genaue Positionsbestimmung stattfinden kann.

3.2. Place Lab und POLS

Das durch Intel unterstützte Forschungsprojekt Place Lab [42] beschäftigte sich mit einer Positionsbestimmung mit Hilfe von Beacons. Place Lab war ein Open-Source Projekt das in Form des Spin-Offs POLS [43] weitergeführt wurde [44].

Das Open-Source Projekt Place Lab entwickelte ein neues Positionsermittlungsverfahren, das mobilen Geräten erlaubt sich selbst zu orten, ohne dafür zusätzlichen Hardware-Einsatz oder Assistenz des Mobilfunkanbieters zu benötigen. Empfangen werden Signale von WLAN-Accesspoints, GSM-Sendemasten und fest installierten Bluetooth-Beacons.

Die Leitziele des Projekts waren das Erreichen einer maximalen Abdeckung durch die Nutzung von vorhandener Infrastruktur und eine einfache Integration und Nutzung [45].

Place Lab nutzt die verfügbaren Schnittstellen zum Signalempfang von Mobilgeräten, um eindeutige IDs von Beacons, wie die MAC-Adresse in der Umgebung zu empfangen. Die Koordinaten der Signalquellen werden in einem zuvor heruntergeladenen lokalen Verzeichnis anhand der empfangenen IDs gesucht. Die gefundenen Koordinaten liefern die Grundlage für die Positionsberechnung des Mobilgeräts.

3.2.1. Place Lab Architektur

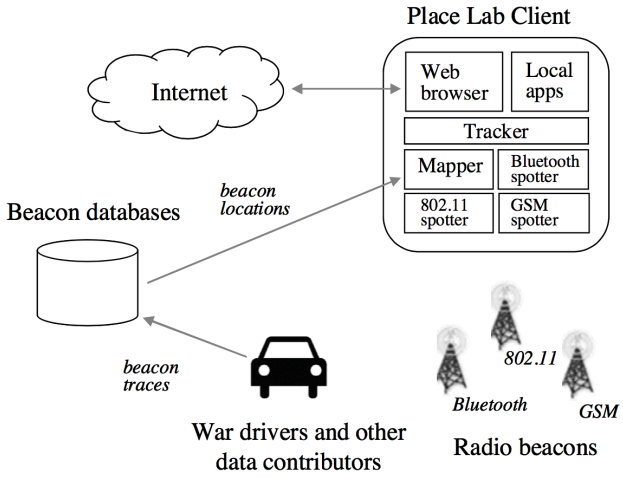

Abbildung 3.2.: Hauptkomponenten in der Place Lab Architektur [4]

Die Hauptkomponenten der Place Lab Architektur in Abbildung 3.2 sind Spotter, Mapper und Tracker. Spotter sind HAL-Komponenten, die eine Abstraktion für die Sensorik liefern. Mapper sind statische Datenbanken, die Positionsdaten auf Basis von Signalinformation für Tracker bereitstellen. Tracker erzeugen Positionsschätzungen auf Basis der durch die Spotter eingehenden Signalinformationen, mit denen die persistierten Positionsdaten der Mapper abgefragt und verarbeitet werden.

Die notwendigen Positionsdaten (hier: *beacon locations*) werden von unterschiedlichen Beacon-Datenbanken abgefragt. Der Place Lab Client hat dabei die Wahl über die geographischen Abdeckung der Daten. Es können Daten von einzelnen Städten aber auch der ganzen Welt heruntergeladen werden. Die Datengrundlage der Mapper kann aus eigenen Datenbanken oder aus einer heruntergeladenen Datenbank von Online-Datenquellen wie z. B. Wigle.net [2] bestehen. Das System hängt stark von den in Abbildung 3.2 dargestellten Datenzulieferern (engl.:

[2] Wigle.net, https://www.wigle.net (aufgerufen am: 08.05.2018)

Data contributers) ab, da ohne eine Datengrundlage keine Positionsbestimmung durchgeführt werden kann. Auch sich ändernde IDs der Geräte oder Standortänderungen von Signalquellen müssen in der Datengrundlage aktualisiert werden.

3.2.2. POLS Software

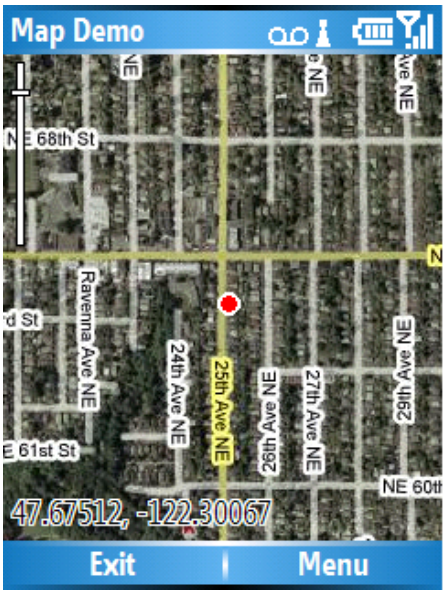

Abbildung 3.3.: POLS Demonstration aus dem Jahr 2005 [5]

POLS (Privacy-Observant Location System) [5] ist ein von Intel unterstütztes Standortsystem, das als Spin-Off des Place Lab Projekts entstanden ist und auf dem gleichen Konzept basiert. Es ermöglicht eine Positionsbestimmung für mobile Applikationen auf dem Gerät. Das Ziel von POLS ist ein einfacher Einstieg für standortbasierte Dienste auf mobilen Geräten.

Seit dem Pre-Release im Jahr 2006 ist keine Weiterentwicklung mehr veröffentlicht worden. Die Software befindet sich weiterhin im Beta-Status. Ein wesentlicher Nachteil der Software besteht darin, dass sie Zugriff auf den Kernel des Betriebssystems benötigt. Um den Kernelzugriff zu ermöglichen, ist ein Entsperren des Geräts notwendig (Root Access), womit in der Regel ein Verlust der Garantie durch den Hersteller einhergeht.

3.2.3. Bewertung

Insbesondere die Berechnung der Position auf dem Gerät mittels lokaler Datenbanken, wodurch der Datenschutz und die Offlinefähigkeit verbessert wird, stellt ein Alleinstellungsmerkmal des

Konzepts dar. Eine vollständige Unabhängigkeit von Daten aus dem Internet ist nicht gegeben, da die Standortdaten der Signalquellen regelmäßig aktualisiert werden müssen.

Es wird die bestehende Infrastruktur unterschiedlicher Sensortypen ausgenutzt, wodurch hohe Kosten vermieden und eine hohe Abdeckung (s. Abschnitt 2.2) erreicht wird. Wichtig für die Abfrage von Koordinaten in einer Datenbank ist die Verwendung einer eindeutigen Signalinformation, wie die MAC-Adresse der Signalquelle.

Das Konzept von Place Lab und POLS ist darauf ausgerichtet, anhand der Signalinformationen Koordinaten zu bestimmen, mit dem Ziel eine möglichst genaue Positionsbestimmung zu erreichen. Die Berechnung der Position ist davon abhängig, dass ein Eintrag zu der empfangenen ID vorhanden ist und die dort hinterlegte Koordinate korrekt ist. Ein Kontextbezug anhand einer Signalinformation herzustellen ist nicht die Kernaufgabe der Software und müsste auf der Grundlage der Koordinaten z. B. durch Reverse Geocoding (s. Abschnitt 2.4.1) erfolgen.

Nach einem vergleichbaren Prinzip funktionieren auch im Internet verfügbare Standort APIs, die eine Menge von Signalinformationen entgegennehmen und eine Koordinate zurückliefern. Zwei Beispiele für Standort APIs werden im folgenden Abschnitt vorgestellt.

3.3. Standort APIs

Standort APIs nehmen Signalinformationen per Anfrage entgegen, berechnen anhand der Angaben möglichst genau die Koordinaten und liefern diese in der Antwort zurück. Die Signalinformationen müssen dabei bestimmte Schemata berücksichtigen. Die Dienste von Unwired Labs LocationAPI [46] und Mozilla Location Service (MLS) [16] ermöglichen kostenfreie Lokalisierungsanfragen bis zu einer täglichen Begrenzung.

Während die Standort API von Unwired Labs nur Mobilfunk- und WLAN-Signale unterstützt, können bei MLS auch Informationen von Bluetooth-Beacons verarbeitet werden. MLS ist eine Instanz des Open-Source Projekts Ichnaea [3].

Die Dokumentationen der in Abschnitt 3.3 vorgestellten Standort APIs weisen darauf hin, dass Anforderungen an den Datenschutz erfordern, dass mindestens jeweils zwei MAC-Adressen von WLAN oder Bluetooth Signalquellen für die Bestimmung der Position notwendig sind.

"The combination of a MAC address and a location is considered personal data in some jurisdictions, as it contains data about the owner of a Wi-Fi network. [...] Lookups of individual Wi-Fi records are prevented by the service by requiring a combination of two matching nearby Wi-Fi

[3] Ichnaea, Github Projekt, https://github.com/mozilla/ichnaea (aufgerufen am: 06.06.2018)

networks in queries." Laut Mozilla gilt diese Einschränkung für Wi-Fi oder Bluetooth basierte Standortdienste [47].

Ein Beispiel für eine Anfrage mit allen möglichen Feldern wird in der API Dokumentation von Ichnaea [48] wie folgt dargestellt:

HTTP Request:

```
POST https://location.services.mozilla.com/v1/geolocate?key=<API_KEY>
 {
    "carrier": "Telecom",
    "considerIp": true,
    "homeMobileCountryCode": 208,
    "homeMobileNetworkCode": 1,
    "bluetoothBeacons": [
        {
            "macAddress": "ff:23:45:67:89:ab",
            "age": 2000,
            "name": "beacon",
            "signalStrength": -110
        }
    ],
    "cellTowers": [{
        "radioType": "wcdma",
        "mobileCountryCode": 208,
        "mobileNetworkCode": 1,
        "locationAreaCode": 2,
        "cellId": 1234567,
        "age": 1,
        "psc": 3,
        "signalStrength": -60,
        "timingAdvance": 1
    }],
    "wifiAccessPoints": [{
        "macAddress": "01:23:45:67:89:ab",
        "age": 3,
        "channel": 11,
        "frequency": 2412,
        "signalStrength": -51,
        "signalToNoiseRatio": 13
    }, {
```

```
        "macAddress": "01:23:45:67:89:cd"
    }],
    "fallbacks": {
        "lacf": true,
        "ipf": true
    }
}
```

Listing 3.1: Mozilla Location Service: Beispiel für ein HTTP-Request

HTTP Response:

```
{
    "location": {
        "lat": -22.7539192,
        "lng": -43.4371081
    },
    "accuracy": 100.0
}
```

Listing 3.2: Mozilla Location Service: Beispiel für ein HTTP-Response

3.3.1. Bewertung

Die Standort APIs bieten neben dem indirekten Zugriff auf große Datenbanken mit Positionsdaten von Beacons auch die Berechnung des eigenen Standorts an. Die Anforderung an mobile Anwendungen die den Service nutzen, beschränkt sich daher auf das Sammeln und Versenden der Signalinformationen. Dazu bietet MLS entsprechende Software für die Integration in Anwendungen an [49]. Unter den gängigen Betriebssystemen Android und iOS wird jedoch nur Android unterstützt. MLS weist darauf hin, dass Apple's iOS ein geschlossenes System sei, dass bewusst den Zugang zu Informationen zu Mobilfunk und Wi-Fi Netzwerken verhindere. Eigene Versuche bestätigten diese Aussage nur zum Teil. Auch wenn der Zugang zur Liste der empfangenen WLAN-Signale grundsätzlich nicht gestattet ist, konnten die Netzwerkinformationen der aktuellen Netzwerkverbindung ausgelesen werden, was eine Abfrage von signalbasierten Kontextinformationen über den Verzeichnisdienst ermöglicht.

3.4. Google Beacon Platform

Die Google Beacon Platform [50] dient dazu Anhänge (engl.: Attachments) mit individuellen Beacons zu verknüpfen und diese über Anwendungen zu verwalten. Die dafür bereitgestellten Protokolle, Schnittstellen und Dienste werden in diesem Abschnitt erläutert.

3.4.1. Eddystone

Eddystone [51] ist ein offenes Beacon Format von Google, das mit Hilfe von Schnittstellen über Android- und iOS-Geräte interpretiert werden kann. Das Format beinhaltet vier unterschiedliche „Frame types“ (engl. für Rahmentypen).

- Die **Eddystone-UID** (Eddystone Unique Identifier) ist eine ID, die den Beacon eindeutig identifiziert und daher genutzt werden kann, um signalbezogene Daten von Servern anzufragen.
- Die **Eddystone-EID** (Eddystone Ephemeral Identifier) ist eine verschlüsselte ID und stellt eine Alternative zur UID dar, die höhere Sicherheitsanforderungen erfüllt. Die EID ändert sich in einem konfigurierbaren Zeitraum von wenigen Minuten, Stunden oder Tagen und kann nur mit Hilfe eines externen Dienstes wie der Google Beacon Plattform aufgelöst werden. Damit hat der Administrator die Kontrolle über die Nutzbarkeit der ausgesendeten ID und verhindert die unautorisierte Verwendung durch fremde Anwendungen.
- Die **Eddystone-URL** [4] liefert die Grundlage für das Physical Web Projekt (s. Abschnitt 3.4.4) in dem es eine URL, die zu einer mit TLS [5] verschlüsselten Webseite leitet, beinhaltet.
- Die **Eddystone-TLM** (Eddystone Telemetry) beinhaltet Informationen über den Beacon. Die Informationen können das Batterie-Level, Daten von Sensoren oder andere relevante Informationen für Beacon Administratoren enthalten. Eine Kombination mit einem identifizierenden „Frame type“, ist in diesem Fall notwendig.

3.4.2. Nearby API

Die Nearby API [52] umfasst drei untergeordnete Schnittstellen um eine Interaktion mit umliegenden Geräten oder Personen zu ermöglichen: Nearby Messaging, Nearby Connections und Nearby Notifications.

[4] URL: Unique Resource Locator
[5] TLS, Transport Layer Security: ein Netzwerkprotokoll zur sicheren Übertragung von Daten

Nearby Messaging stellt Publish- und Subscribe-Methoden zur Verfügung, die auf Nähe basieren. Eine mobile Anwendung kann Informationen veröffentlichen, die von Teilnehmern in der Nähe empfangen werden können. Mit der Schnittstelle können iOS- und Android-Geräte Anhänge (engl: Attachments) empfangen, die einem Beacon mit Hilfe der Google Beacon Platform zugewiesen wurden [38].

Nearby Connections ist eine Peer-to-Peer-Netzwerk-API, mit der mobile Anwendungen, unabhängig von der Netzwerkkonnektivität andere Geräte erkennen, sich verbinden und Daten austauschen können. Die API ermöglicht die Erstellung von Mehrspieler-Spielen oder Anwendungen, mit denen offline Daten mit anderen Nutzern geteilt werden können [53].

Nearby Notifications ist eine Funktion, mit der Entwickler eine mobile mehrere Einträge zu Android-Anwendungen oder Webseiten mit einem Bluetooth-Beacon verknüpfen und kontextbezogene Benachrichtigungen erstellen können, auch wenn keine App installiert ist. Die Funktion ist für Android-Geräte ab der Android-Version 4.4 verfügbar und mit iOS nicht kompatibel [39].

Der Gegensatz zur Motivation dieser Arbeit entsteht beim Informationsaustausch über die Nearby Messaging API kein Bezug örtlichen Standort. Um den Unterschied zu verdeutlichen wird in der folgenden Auflistung der Mechanismus zum Informationsaustausch über API erläutert.

Es sind insgesamt fünf Schritte notwendig, um eine Nachricht an umliegende Teilnehmer zu senden:

1. Die veröffentlichende Anwendung macht eine Serveranfrage, um der Nachricht temporär einen eindeutigen zeitbezogenen Token zur Kopplung zuzuweisen.

2. Das veröffentlichende Gerät nutzt eine Kombination aus Bluetooth, Bluetooth Low Energy, Wi-Fi und Ultraschall um den Token für andere in der Nähe befindliche Geräte zu veröffentlichen.

3. Das veröffentlichende Gerät wartet ebenfalls auf Tokens von anderen Geräten.

4. Eine abonnierende Anwendung verknüpft ihr Abonnement mit einem Token und nutzt die in Punkt 2 beschriebenen Technologien, um diesen Token zum veröffentlichenden Gerät zu senden.

5. Empfängt ein Gerät den Token des anderen Geräts, sendet es den Token an den Server. Wenn zwei Geräte jeweils mit dem gleichen Token verknüpft und die API-Keys der Anwendungen kompatibel sind, werden die Nachrichten an die Geräte verteilt.

3.4.3. Proximity Beacon API

Die Proximity Beacon API [37] ermöglicht die Interaktion mit Google's Cloud Platform [6] für die Verwaltung von Beacons. Mit Hilfe der Schnittstelle können Informationen mit Beacons verknüpft, verwaltet und mit anderen Google Produkten kombiniert werden.

3.4.4. Physical Web

Das Physical Web [54] ist ein offener Ansatz, um Interaktionen mit physikalischen Objekten und Orten herzustellen, indem über Bluetooth-Beacons Eddystone-URLs ausgesendet werden.

Nutzer der Physical Web-Anwendung [7] [8] können über das Signal eines Bluetooth-Beacons eine Liste von URLs empfangen und verarbeiten. Dazu muss der Bluetooth-Beacon und der Empfänger das Eddystone Format unterstützen (s. Abschnitt 3.4.1).

Drei Schritte sind notwendig um URLs über das Physical Web für andere Nutzer die Physical Web-kompatible Geräte nutzen, aufmerksam zu machen.

1. Es müssen Bluetooth-Beacons vorhanden sein oder ein Android Smartphone, das mit Beacon-Emulatorsoftware als Bluetooth-Beacon genutzt werden kann.

2. Es müssen URLs festgelegt werden, die über die Bluetooth-Beacons gesendet werden sollen. Außerdem muss die Sende-Häufigkeit und Reichweite des Signals eingestellt werden.

3. Die Bluetooth-Beacons müssen in einem physikalischen Raum platziert werden.

Im Gegensatz zur Nearby Notifications API erlaubt das Physical Web nur eine begrenzte Anzahl von URLs und keine ergänzenden Informationen.

3.4.5. Bewertung

Google deckt durch die Kombination der unterschiedlichen, zusammenwirkenden Lösungen einen Großteil der Anforderungen zur standort- und nähe-basierten Informationsübertragung ab. Dabei konzentrieren sich die vorgestellten Lösungen auf zwei Prinzipen:

1. Bereitstellung von standort- oder objektbezogenen Informationen über Bluetooth-Beacons

[6] Google Cloud Platform, https://console.cloud.google.com (aufgerufen 02.07.2018)
[7] Physikal Web im Google Playstore, https://play.google.com/store/apps/details?id=physical_web.org.physicalweb&hl=de (aufgerufen am: 02.07.2018)
[8] Physikal Web im Appstore, https://itunes.apple.com/de/app/physical-web/id927653608?mt=8 (aufgerufen am: 02.07.2018)

2. Informations- bzw. Nachrichtenaustausch mit Nutzern in der näheren Umgebung über Funk- und Ultrasoundtechnologien

Trotz der vielfältigen Lösungen, ist es nicht möglich den in dieser Arbeit vorgestellten Ansatz umzusetzen. Es ist nicht vorgesehen ein Beacon-Signal als Grundlage für einen Informationsaustausch zu nutzen.

Es kann nur ein Administrator über die Verwaltungsebene der Google Beacon Platform eine Information in Form eines Attachments oder einer Nearby Notification über das Signal eines Beacons veröffentlichen. Eine spontane Verknüpfung von Informationen zu einem Signal durch Nicht-Administratoren ist nicht möglich.

Eine Anwendung, dessen Nutzer durch den Empfang einer eindeutigen ID im Sendebereich eines Beacons, einen Informationsaustausch einleiten können, kann mit Hilfe der hier dargestellten Lösungen nur begrenzt realisiert werden. Es kann zwar eine signalbasierte Benachrichtigungsfunktion genutzt werden, jedoch enthält diese Form der Benachrichtigung keine eindeutigen Informationen über den Beacon. Da nur Administratoren Benachrichtigungen modifizieren können, müssten diese eine eindeutige ID in der Benachrichtigung platzieren, die den Kontext referenziert und Nutzern einen kontextgebundenen Informationsaustausch gestattet.

Das Angebot von Google beschränkt sich auf die Nutzung von Bluetooth Beacons, was eine Verwendung der Software für andere Signaltypen wie Wi-Fi-Signale ausschließt.

3.5. Zusammenfassung

Die präsentierten Arbeiten zeigen, dass die Auswertung von Beacons erfolgreich zur Positionsbestimmung und zur Bereitstellung von Informationen eingesetzt wird.

Keine der verwandten Arbeiten nutzt den Sendebereich eines Beacons aus, um ihn als Kontextebene (s. Abschnitt 2.3) zu interpretieren. Bei den vorgestellten Methoden werden Signalinformationen entweder zur Positionsbestimmung genutzt oder vor dem Nutzer verborgen. Keine der Arbeiten erlaubt den Empfängern eines Signals, eindeutige Informationen zum Informationsaustausch bzw. zur Bereitstellung von kontextgebundenen Informationen zu nutzen.

Die Lösungen von MagicMap und POLS sowie die Standort APIs unterscheiden sich im Verfahren, das der Positionsbestimmung zugrunde liegt und in den Anforderungen an den Datenaustausch innerhalb des Systems. Sie haben gemeinsam, dass sie in der Lage sind, die vorhandene Infrastruktur aus unterschiedlichen Funksignalquellen zur Positionsbestimmung auszuwerten, ohne Software- oder Hardwarekonfiguration vornehmen zu müssen.

Bei den vorgestellten Angeboten von Google ist der Besitz und die Administrierung von Bluetooth-Beacons ein Kriterium für die Bereitstellung von signalbezogenen Informationen.

Als Ergebnis liefern die Verfahren MagicMap, POLS sowie die Standort APIs eine Koordinate, welche sich jedoch nur mit Hilfe von Reverse Geocoding auf Basis von Geofencing in eine Kontextebene einordnen lässt.

Mit Hilfe der Google Nearby API ist zwar der Informationsaustausch zwischen Nutzern in der näheren Umgebung über Funk- und Ultraschalltechnologien der jeweiligen Mobilgeräte möglich, jedoch existiert kein Bezug zum Kontext des Standorts. Die angebotenen Benachrichtigungsfunktionen enthalten keine eindeutigen Signalinformationen, die als Informationsschlüssel genutzt werden könnten.

Das folgende Kapitel analysiert, wie der signalbasierte Kontextbezug anhand eines Verzeichnisdienstes hergestellt werden kann und präsentiert ein Design, mit dem ein Informationsaustausch zwischen den Empfängern eines Signals möglich ist.

4. Analyse und Design

Mit Hilfe eines Verzeichnisdienstes soll das Nutzererlebnis in mobilen Anwendungen durch den signalbasierten Austausch von Kontextinformationen verbessert werden. Es sollen öffentlich zugängliche Signalinformationen genutzt werden können, wie sie in Cafés, öffentlichen Einrichtungen, öffentlichen Verkehrsmitteln und an öffentlichen Plätzen vorzufinden sind.

In diesem Kapitel werden die Anforderungen an den Entwurf und die Nutzung des kontextgebundenen Verzeichnisdienst analysiert und ein Systementwurf präsentiert.

4.1. Anforderungen an die signalbasierte Kontextermittlung

Die Verfahren, die Signale zur Positionsbestimmung verarbeiten, liefern im Ergebnis eine Koordinate mit unterschiedlicher Genauigkeit. Das Wissen über die Position einzelner Mobilgeräte beinhaltet jedoch keine Information, die für die Gruppierung von Mobilgeräten nutzbar wäre. Nur auf der Grundlage einer Gruppierung von Mobilgeräten kann ein standortbasierter Informationsaustausch stattfinden.

Dazu bedarf es der Definition eines Gültigkeitsbereichs, z. B. in Form eines koordinatenbasierten Polygons (Geofencing), mit dessen Hilfe geprüft werden kann, ob sich Mobilgeräte in dem Bereich befinden oder nicht. Diese Vorgehensweise ist aufgrund der benötigten Vorbereitung, für Szenarien in denen ein spontaner Austausch stattfinden soll, ungeeignet. Auch für den Bezug zu Bereichen innerhalb sich bewegender Fortbewegungsmittel, ist der Einsatz von Koordinaten nicht sinnvoll.

Die Auswertung von Beacons hat aufgrund des Bezugs zu einer Signalquelle im Hinblick auf einen Informationsaustausch den Vorteil, dass ein Empfangsbereich existiert, durch den Mobilgeräte binär als Empfänger oder Nicht-Empfänger klassifiziert werden können.

Befinden sich Mobilgeräte im Empfangsbereich, empfangen sie alle die selben Signalinformationen. Darin enthalten sind in der Regel eindeutige Informationen, die als Schlüssel für einen Informationsaustausch genutzt werden können.

Das Verfahren stellt minimale Anforderungen an die vorhandene Infrastruktur und ist daher als Grundlage für die Konzeption des Verzeichnisdienstes geeignet.

4.2. Anforderungen an das Gesamtkonzept

Um die Nutzbarkeit des Dienstes nicht von der Installation oder Konfiguration von Signalquellen abhängig machen zu müssen, soll die bestehende Signalinfrastruktur ausgenutzt werden können. Insbesondere Wi-Fi-Accesspoints, die für den Zugang zum Internet in Betrieb genommen wurden, stellen durch ihre globale Verbreitung und der kontextbezogenen Signalabdeckung eine attraktive Signalquelle dar. Daneben sollen jedoch auch andere Signaltypen wie z. B. Bluetooth-Beacons genutzt werden können. Die Signalinformationen müssen auf eine eindeutige ID reduziert werden können.

Um die kontextgebundenen Informationen aus dem Verzeichnis abrufen zu können, müssen Anwendungen einen entsprechenden Informationsschlüssel verwenden. Dieser muss in Form von Signalinformationen über die jeweilige Schnittstelle des Betriebssystems abgerufen und von einem Anwendungsserver mit Hilfe des Verzeichnisdientes verarbeitet werden.

Ein einzelner Informationsschlüssel, wie z. B. die MAC-Adresse eines Geräts, darf nicht in Kombination mit Positionsdaten von Nutzern gespeichert werden, da sonst die datenschutzrechtlichen Einschränkungen der vorgestellten Standort-APIs (s. Abschnitt 3.3) gelten.

Wie bereits in der Einleitung deutlich gemacht, wird keine Implementierung einer möglichen Software zur Anwendung des Verzeichnisdienstes diskutiert. In den folgenden Abschnitten werden jedoch zum Verständnis des Gesamtkonzepts die wesentlichen Anforderungen an die Nutzung des Verzeichnisdienstes verdeutlicht.

4.2.1. Anforderungen an den Anwendungsserver

Mobile Anwendungen benötigen nach dem Client-Server-Modell [55] , insbesondere zum Austausch von Daten mit anderen Nutzern, eine gemeinsame Schnittstelle in Form eines anwendungsspezifischen Servers.

Der Anwendungsserver nutzt den Verzeichnisdienst, um die kontextgebundenen Informationen des Clients mit den mitgelieferten Signalinformationen für andere Nutzer des Verzeichnisdienstes bereitzustellen.

Die Mindestanforderungen an den Anwendungsserver bestehen aus:

- Weiterleitung von Anfragen mobiler Anwendungen an den Verzeichnisdienst
- Auflösung von Kontext-IDs über Schnittstellen zu Datenquellen

Zu den optionalen Anforderungen gehören u. a. folgende Funktionen:

- Nutzerverwaltung
- Asynchrone Benachrichtigungsfunktion (z. B. bei der Änderung von Daten)

4.2.2. Anforderungen an die graphische Nutzerschnittstelle

Die Anforderungen an die graphische Nutzerschnittstelle beschränken sich auf zwei Mindestanforderungen:

- Präsentation der kontextbezogenen Informationen der Anwendung
- Eingabemöglichkeit zur Erstellung und Versand von kontextbezogenen Informationen

Abbildung 4.1.: Veranschaulichung einer graphischen Nutzerschnittstelle

4.2.3. Schritte für das Empfangen und Senden von Informationen

Es lassen sich die folgenden Schritte für das Empfangen von Informationen zusammenfassen:

- Die mobile Anwendung erhält Signalinformationen einer Signalquelle
- Die mobile Anwendung sendet Signalinformationen an den Anwendungsserver
- Der Anwendungsserver fragt Signalinformationen bei dem Verzeichnisdienst ab
- Der Anwendungsserver erhält Kontextinformationen (in Form von IDs)

- Der Anwendungsserver stellt Daten anhand der Kontext-IDs zusammen
- Der Anwendungsserver sendet kontextgebundene Daten an die mobile Anwendung
- Die mobile Anwendung präsentiert die kontextgebundenen Daten

Es lassen sich die folgenden Schritte für das Senden von Informationen zusammenfassen:

- Die mobile Anwendung stellt eine graphische Nutzerschnittstelle für die Erstellung von Informationen zur Verfügung
- Die mobile Anwendung sendet Signalinformationen und Informationen des Nutzers an den Anwendungsserver
- Der Anwendungsserver generiert eine UUID für die Kontextinformation
- Der Anwendungsserver sendet die Kontext-ID und die Signalinformationen an den Verzeichnisdienst
- Der Verzeichnisdienst persistiert die empfangenen Daten und stellt diese zur Verfügung

Im folgenden Abschnitt werden Szenarien skizziert, die eine Vorstellung von potenziellen Einsatzmöglichkeiten des Verzeichnisdienstes vermitteln und Beispiele für Kontextinformationen liefern.

4.3. Mögliche Szenarien für den signalbasierten Informationsaustausch

Um den potentiellen Mehrwert durch den Informationsaustausch von Empfängern eines Signals zu verdeutlichen, werden in der folgenden Auflistung Beispiele für Einsatzbereiche skizziert.

1. **Personenverkehr (Nah- und Fernverkehr)**
   Über eine Signalquelle in einem Verkehrsmittel, wie z. B. ein WLAN-Accesspoint oder ein Bluetooth-Beacon, könnten hilfreiche Informationen, wie z. B. die Haltestellen und Ankunftszeiten mitgeteilt werden. Dabei könnten Reisende mit Hilfe des signalbasierten Austauschs ein gemeinsames Interesse feststellen.

2. **Öffentliche Institutionen**
   In öffentlichen Institutionen wie z. B. Bildungsstätten könnten über den Verzeichnisdienst standort- oder raumbezogene Informationen unter Lehrern, Schülern, Professoren oder

Studenten ausgetauscht werden. Dies könnte eine Ergänzung zum klassischen Email-Verteiler darstellen und die Organisation verbessern.

3. **Austausch von Kontaktdaten**
   Nutzer könnten Kontaktdaten mit Signalinformationen verknüpfen, so dass diese interessierten Empfängern zur Verfügung stehen. Dieser Austausch von Kontakten könnte als eine digitale Alternative zum Austausch von Visitenkarten Verwendung finden.

4.4. Anforderungsanalyse für den Verzeichnisdienst

4.4.1. Grundfunktion

Der Dienst ermöglicht, dass ein Empfänger eines Signals Informationen in einem Verzeichnis ablegen kann und andere Empfänger des gleichen Signals diese Informationen abrufen können.

4.4.2. Stakeholder

Die Stakeholder des Verzeichnisdienstes sind Anwendungsentwickler, deren Anwendung auf einem kontext- bzw. standortbezogenen Informationsaustausch zwischen Signalempfängern basiert oder um eine entsprechende Funktion erweitert werden kann.

Daneben haben Nutzer von mobilen Anwendungen, die eine Affinität zur Bereitstellung oder Konsumierung von Inhalten in sozialen Netzwerken haben, ein mutmaßliches Interesse am Einsatz des Verzeichnisdienstes.

Die Anzahl der Nutzer von potenziellen Anwendungen bestimmt die Anzahl der Anfragen an den Verzeichnisdienst und ist damit eine relevante Größe für die Anforderungen an das System. Ein Richtwert für die Systemanforderung geben die weltweit 2,2 Milliarden monatlich aktiven Nutzer der sozialen Plattform Facebook [1] im März 2018 [56]. Da die Nutzung des Verzeichnisdienstes im Gegensatz zu Facebook von den Standorten der Signalquellen abhängt, ist eine regionale Vermarktung möglich. Damit kann die Zielgruppe zunächst auf 30 Millionen Facebook-Nutzer in Deutschland (Stand: September 2017) beschränkt werden [57].

4.4.3. Use Cases

Die Anwendungsfälle (engl.: Use Cases) des Verzeichnisdienstes lassen sich auf die folgenden Aufrufe beschränken:

- Erstellen eines Datensatzes

[1] Facebook, https://facebook.com (aufgerufen am: 07.06.2018)

- Lesen eines Datensatzes
- Aktualisieren eines Datensatzes
- Löschen eines Datensatzes

4.4.4. Nicht-Funktionale Anforderungen

Um die Funktionsfähigkeit des Dienstes bei einem schnellen Anstieg der Anzahl der Zugriffe zu gewährleisten, müssen im Vorfeld Vorkehrungen getroffen werden, welche die Skalierbarkeit des Systems garantieren.

Im Hinblick auf eine Veröffentlichung des Dienstes in weitere Länder, muss das System das Hinzufügen weiterer Regionen ohne Beeinträchtigungen in der Funktionsfähigkeit ermöglichen.

Dabei soll der Dienst, wenn möglich, auf Basis von Open-Source-Angeboten und eigener Programmierung realisiert werden.

Maßnahmen zur Skalierung des Systems werden konzeptionell berücksichtigt, so dass im Zuge erhöhter Nachfrage eine Implementierung erfolgen kann. Ein vorzeitiger Einsatz aller im Design vorgesehenen Optionen ist nicht vorgesehen.

Der Dienst muss in seiner Struktur möglichst auf seinen Zweck zugeschnitten sein. Daten müssen einfach und schnell verarbeitet werden und den Nutzern in Form einer einfachen, verständlichen Schnittstelle zur Verfügung stehen. Dabei stellt die Skalierbarkeit der Datenspeicherung und -verwaltung eine maßgebliche Anforderung an den Dienst.
Es lassen sich folgende Kernanforderungen zusammenfassen:

1. Das System muss die Verwaltung unterschiedlicher Signalinformationen ermöglichen.
2. Das System muss die Skalierung des Systems zur Laufzeit ermöglichen.
3. Das System muss den parallelen Zugriff von unterschiedlichen Anwendungen auf Ressourcen über eine gemeinsame Schnittstelle ermöglichen.
4. Die Verfügbarkeit der Daten über eine Schnittstelle im Internet muss gewährleistet werden können.
5. Die Bearbeitungszeit des Dienstes führt zu keiner merklichen Verzögerung beim Nutzer.
6. Das System ist beschränkt auf die Funktionalität als Verzeichnisdienst.
7. Das System ist zustandslos (engl.: stateless).

Über den Verzeichnisdienst werden kontextgebundene Informationen bestimmten Signalinformationen zugeordnet. Daher muss definiert werden, welche Informationen für die Beschreibung eines Signals und eines Kontexts relevant sind.

4.4.5. Anforderungen an die Signalbeschreibung

Daten, die durch Beacons versendet werden, liefern in der Regel nicht nur Informationen die das aussendende Gerät eindeutig identifizieren, sondern auch Beschreibungen, die für den Nutzer einen lesbaren und verständlichen Hinweis auf die Signalquelle geben. Der Signaltyp ergibt sich bereits durch die Funkfrequenz und dem Protokoll, über das die Daten übermittelt werden.

Über das Verzeichnis muss ein Signal eindeutig identifizierbar sein. Auch wenn eine eindeutige ID zur Identifizierung eines Signals grundsätzlich ausreicht, kann die Abfrageeffizienz gesteigert werden, indem eine Filterung anhand von Metadaten vorgenommen wird. Daher wird die Menge der zu vergleichenden Einträge durch die Angaben des Signaltyps eingegrenzt.

Das folgende Beispiel in Tabelle 4.1 veranschaulicht, wie Beschreibungen von unterschiedlichen Signaltypen in einem Schema zusammengefasst werden. Neben Wi-Fi und Bluetooth-Signalinformationen ist auch der Einsatz von CIDs [2] aus dem Mobilfunknetz denkbar.

Typ	**ID**	**Bezeichnung**
„WLAN“	MAC	SSID
„Bluetooth LE“	MAC	Geräte-Name
„Eddystone“	UUID	Geräte-Name
„GSM“	CID	„CID“
...	...	...

Tabelle 4.1.: Signalbeschreibung

Durch den Typ der Signalquelle kann eine erste Eingrenzung erfolgen. So kann ein Mobilgerät die Anfrage auf die Schnittstelle beschränken, mit der die Signalinformation empfangen wurde.

[2] CID, eine Abk. für Cell Identification (dt. Mobilfunkzellenidentifikation), ist eine eindeutige Kennzahl im GSM-Netz, die einer Basisstation zugeordnet ist.

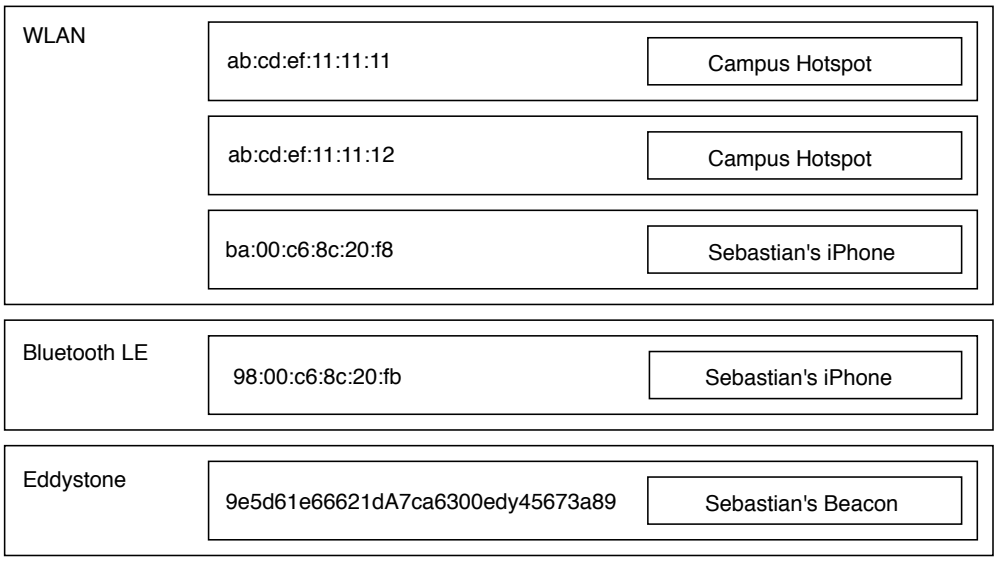

Abbildung 4.2.: Abfragehierarchie innerhalb einer Signalbeschreibung

4.4.6. Anforderungen an die Kontextbeschreibung

Informationen müssen in Kontextebenen (s. Abschnitt 2.3) eingeordnet werden. Innerhalb einer Kontextebene muss eine Unterscheidung zwischen den Eigentümern der bereitgestellten Informationen möglich sein. Einem Eigentümer kann eine Menge von jeweils eindeutigen Einträgen zugeordnet werden. Ein Eintrag liefert durch eine eindeutige ID eine Referenz zu einer externen Datenquelle. Je nach Anwendung können Hyperlinks mit Verweis auf die URL zur Anfrage der ID oder andere Metainformation, wie Zeitstempel oder Versionsstempel sinnvoll sein. Daher existiert die Möglichkeit die ID des Eintrages um Informationen im JSON-Dateiformat zu ergänzen.

Ebene	**Eigentümer**	**Einträge**
Ebene A	UUID	{UUID:{}, ...}
Ebene B	UUID	{UUID:{}}
...	...	...

Tabelle 4.2.: Kontextbeschreibung

Der Verzeichnisdienst kann von einer Anwendung direkt oder über ein anwendungsspezifisches Gateway angesprochen werden. Anwendungen können sich auf eigene Kontextebenen beziehen, die nur im Rahmen der Anwendung relevant sind, oder Kontextebenen nutzen, die anwendungsübergreifende Bedeutung haben.

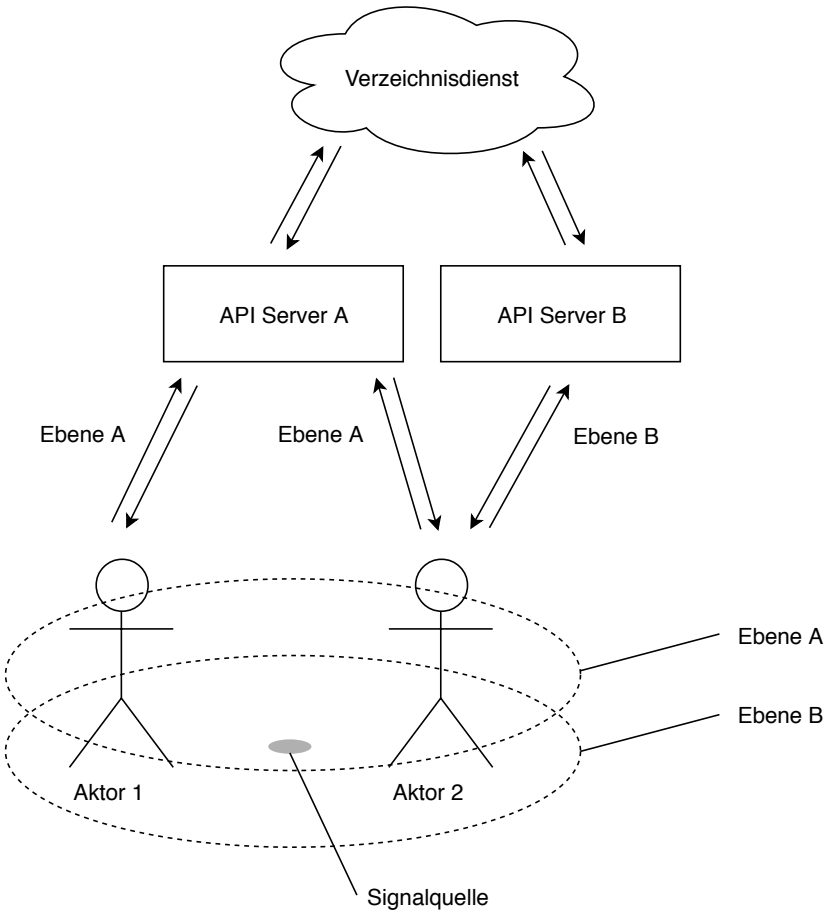

Abbildung 4.3.: Informationsaustausch auf gleicher Kontextebene

Das Beispiel in Abbildung 4.3 zeigt zwei Aktoren, die sich im Senderadius einer Signalquelle befinden. Die Aktoren verwenden unterschiedliche Anwendungen auf ihren Mobilgeräten. Während die Anwendung von Aktor 1 nur Ebene A kennt, sind der Anwendung von Aktor 2 Ebene A und Ebene B bekannt. So kann auf Basis der Signalinformation über die Ebene A ein Austausch zwischen Aktor 1 und Aktor 2 stattfinden und Aktor 2 darüber hinaus Daten der Ebene B abrufen. Ob die Aktoren die bereitgestellten Informationen des jeweils anderen abrufen können, kann zusätzlich über den Anwendungsserver A kontrolliert werden.

In Anlehnung an das Szenario aus Abbildung 4.3 zeigt Abbildung 4.4 ein Beispiel für mögliche Daten. Aktor 1 mit der UUID 3f35d20c62654dca8a2dcd63cfbe9661 hat drei Einträge in der Ebene A bereitgestellt. Aktor 2 mit der UUID 95fa6d04fd1d4ebe95872adf47903081 hat in der

Ebene A und der Ebene B jeweils einen Eintrag veröffentlicht. Im Gegensatz zu Aktor 1 kann Aktor 2 auch den Eintrag abrufen, der von einem dritten Aktor in Ebene B hinterlegt wurde.

Auf die Hinterlegung von optionalen Meta-Daten zu den jeweiligen Einträgen wurde in diesem Beispiel verzichtet.

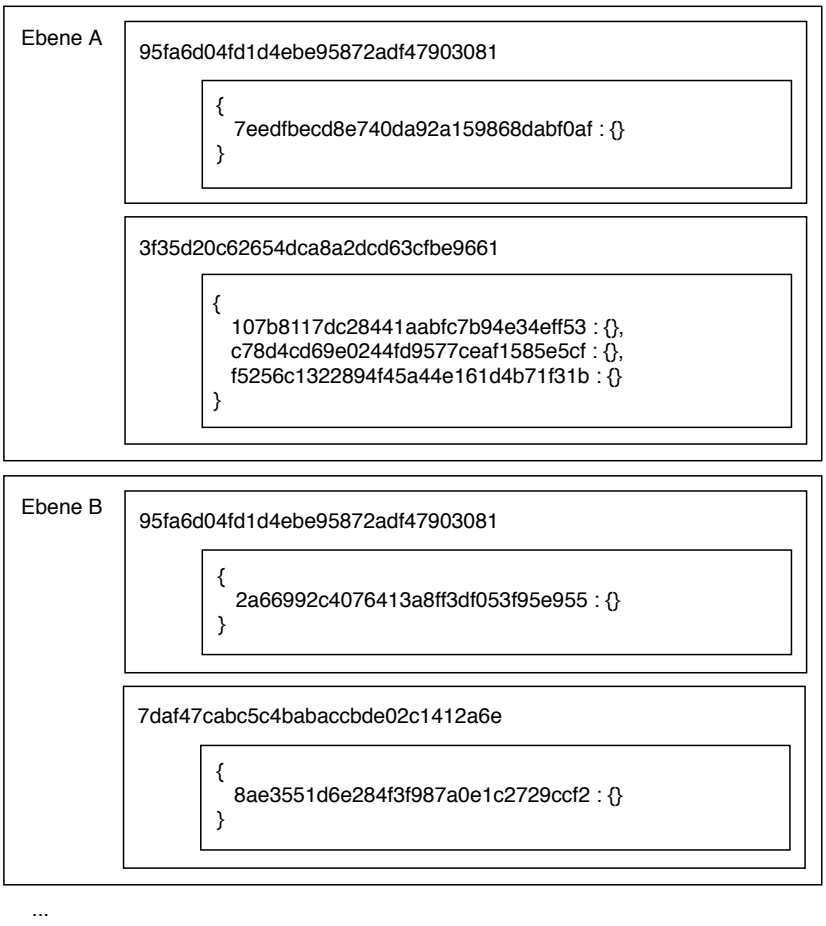

Abbildung 4.4.: Abfragehierarchie innerhalb einer Kontextbeschreibung

4.4.7. Begriffsdefinitionen

In der folgenden Auflistung werden relevante Begriffe erläutert, die sich aus der Anforderungsanalyse zu den Signal- und Kontextinformationen ergeben und für die Beschreibung der funktionalen Anforderungen genutzt werden.

- Signal-ID: Die Signalinformation beschreibt einen Wert, der für ein oder mehrere Signale eindeutig ist.
- Ebene: Die Ebene beschreibt die Kontextebene und die Art der Kontextinformation.
- Eigentümer: Der Eigentümer ist der Sender und Administrator der Kontextinformation.
- Eintrag-ID: Die Eintrag-ID beschreibt einen Wert, der für einen Eintrag eindeutig ist.
- Eintrag: Der Eintrag setzt sich aus einer Kontext-ID und weiteren Informationen zusammen.
- Kontextinformation: Eine Kontextinformation umfasst Eigentümer und deren Einträge.
- Signalinformation: Eine Signalinformation umfasst die Signal-ID und weitere Informationen zur Signalquelle.

4.4.8. Funktionale Anforderungen

1. Das System stellt eine Schnittstelle zum Abruf der verwalteten Ressourcen zur Verfügung
2. Der Dienst stellt existierende, Kontextinformationen zur Verfügung, die zu einer Signalinformation und Ebene hinterlegt wurden.
3. Der Dienst beschreibt und identifiziert eine Signalinformation eindeutig.
4. Der Dienst beschreibt und identifiziert eine Kontextinformation eindeutig.
5. Der Dienst ermöglicht die Zuordnung von Kontextinformationen zu einer Signalinformation.
6. Der Dienst ermöglicht die Zuordnung von Kontextinformationen zu einer Ebene.
7. Der Dienst ermöglicht die Zuordnung von einer Menge von Kontextinformation zu einem Eigentümer.

8. Der Dienst ermöglicht die Zuordnung von einer Menge von Eigentümern zu einem Kontextebene.
9. Der Dienst ermöglicht die Zuordnung von einer Menge von Ebenen zu einer Signalinformation.
10. Der Dienst ermöglicht die Speicherung von weiteren Informationen zu einem Signal.
11. Der Dienst ermöglicht das Löschen von Einträgen.
12. Der Dienst entfernt Eigentümer ohne Einträge.
13. Der Dienst entfernt Ebenen ohne Eigentümer.
14. Der Dienst entfernt Signale ohne Ebenen.

Um die funktionalen Anforderungen umsetzen zu können, müssen Abhängigkeiten zwischen den genannten Attributen geklärt und in einer Verzeichnisstruktur zusammengefasst werden.

4.4.9. Anforderungen an die Verzeichnisstruktur

Um die relationalen Eigenschaften zwischen den Attributen zu beschreiben, eignet sich das in Abbildung 4.5 dargestellte Relationsmodell. Das normalisierte Modell gibt Aufschluss über die grundsätzlichen Abhängigkeiten. Eine Implementierung in dieser Form vermeidet Redundanzen und gestattet eine hohe Flexibilität im Hinblick auf mögliche Abfragen. Daten können auch ohne Angabe des primären Schlüssels anhand von Fremdschlüsselbeziehungen eingegrenzt werden.

Jedoch muss dieses Modell im Hinblick auf die Skalierbarkeit und die Abfrageeffizienz kritisch bewertet werden. Die vielen Relationen sind nachteilig für die Verarbeitungsgeschwindigkeit von Abfragen und der Datenverfügbarkeit, da viele Tabellenzugriffe für die Abfrage der kontextbezogenen Informationen notwendig sind.

Ein denormalisierter Aufbau in Form einer einzigen Tabelle erleichtert die Aufteilung und Replikation von Daten über ein Cluster. Daten werden als Aggregat, das ausschließlich über einen primären Schlüssel angesprochen wird, auf einem Knoten gespeichert.

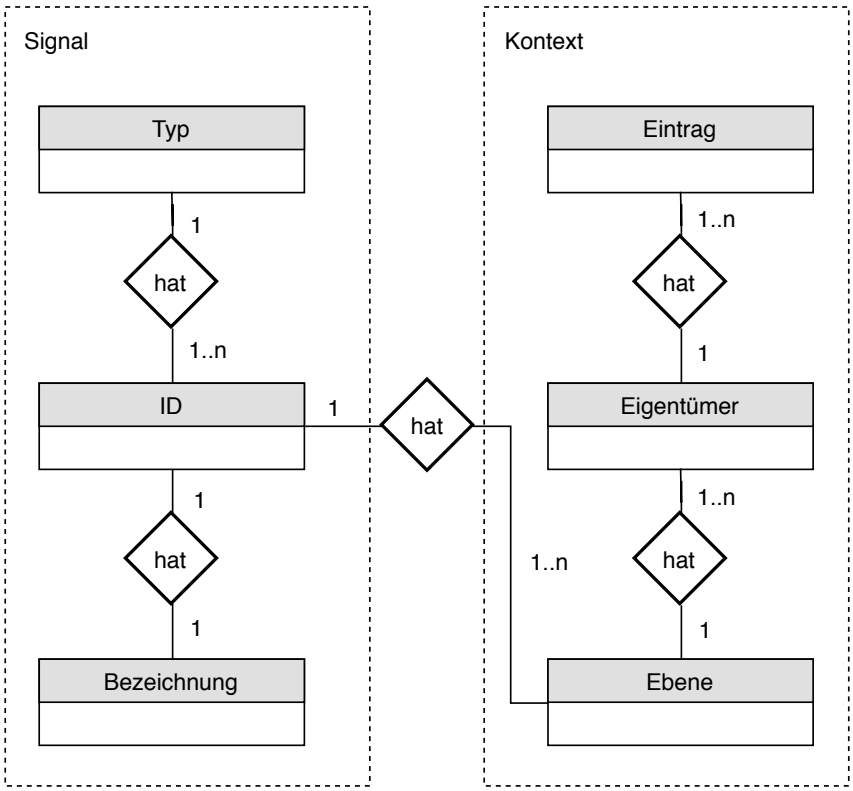

Abbildung 4.5.: Relationsmodell

4.5. Systementwurf

4.5.1. Architekturmodell

Ausgehend von den Ergebnissen der Anforderungsanalyse in Abschnitt 4.4, eignet sich die Konzeption des Verzeichnisdienstes als Microservice. Als Microservice bleibt der Verzeichnisdienst auf die zustandslose Verzeichnisfunktion beschränkt. Das vereinfacht die Verteilung der Funktion auf mehrere parallel ausführende Instanzen, wodurch eine Lastverteilung und Skalierung vereinfacht wird [58].

Die Abbildung 4.6 zeigt das Schichtenmodell der Gesamtarchitektur, bei der ein Anwendungsserver im Sinne der *Service Orchestrierung* [59] als Konsument den Microservice beansprucht. Die dargestellten Pfeile verdeutlichen, dass die Zugriffe nur in eine Richtung erfolgen und der Verzeichnisdienst nicht an weitere Dienste gebunden ist. Gegenüber dem Architekturparadigma einer serviceorientierten Architektur (Abk.: SOA), werden Abhängigkeiten zu anderen Diensten vollständig vermieden [59].

Über die Präsentationsschicht des Verzeichnisdienstes, können anhand von URLs Ressourcen angefragt werden. In der Logikschicht werden die Anfragen verarbeitet. Dabei werden Anfragen an die Datenerhaltungsschicht geschickt.

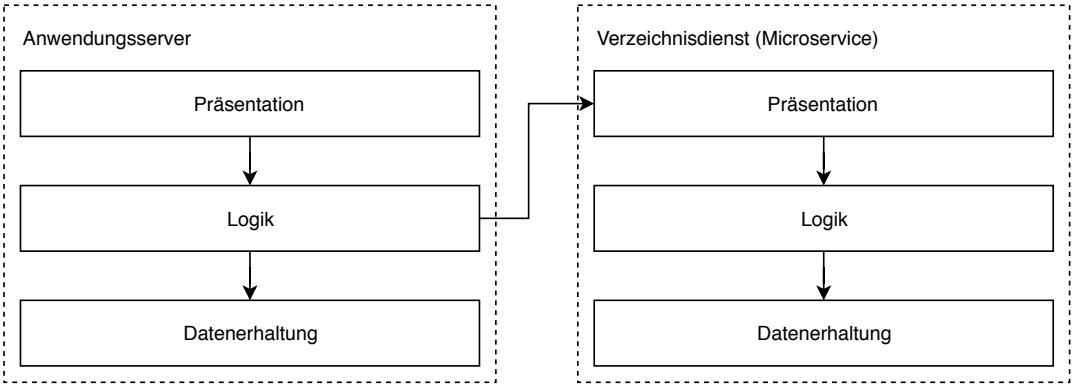

Abbildung 4.6.: Schichtenmodell

4.5.2. Datenmodell

Die in der Verzeichnisstruktur vorhandenen Attribute können in eine eindeutige Schlüssel- und eine Werte-Menge aufgeteilt werden, so dass keine zusätzliche Fremdschlüsselbeziehung aufgebaut werden muss.

Das Datenmodell in Abbildung 4.7 stellt eine Schlüssel-Wert Beziehung dar, dessen Aufbau sich an der Hierarchie des Relationsmodells in Abbildung 4.5 im Abschnitt 4.4.9 orientiert.

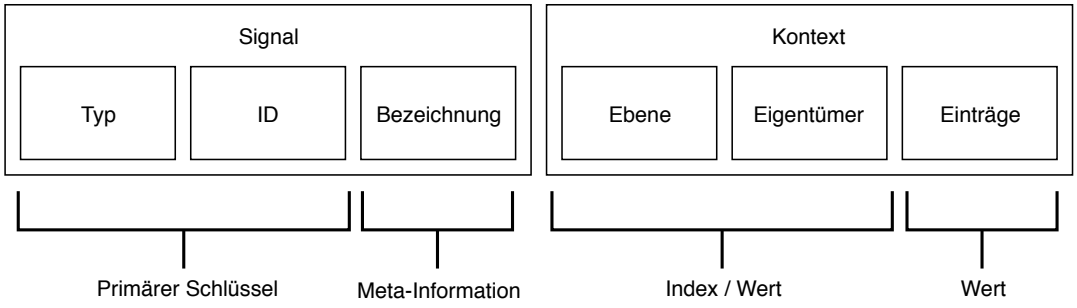

Abbildung 4.7.: Datenmodell

Die Signalinformationen *Typ* und *ID* stellen die Schlüssel-Attribute dar, die zur eindeutigen Abfrage von Wert-Attributen vorgesehen sind. Die für eine Abfrage relevanten Werte-Attribute

umfassen Metainformationen, wie die *Bezeichnung* der Signalquelle und die Kontextinformationen aus *Ebene*, *Eigentümer* und *Einträge*. Der definierte Zugriff über einen eindeutigen Schlüssel ermöglicht es, Datensätze auf Partitionen in einem Cluster zu verteilen und unmittelbar abzurufen [60].

Ein weiteres Hilfsmittel zur Steigerung der Zugriffsgeschwindigkeit ist die Indexierung der Werte-Attribute *Ebene* und *Eigentümer*, wodurch Einfluss auf die Sortierung der Datensätze genommen wird. Die Schlüsseldefinition und die Indexierung ermöglichen durch die optimale Sortierung der Datensätze einen effizienten Zugriff auf die Kontextinformationen.

4.5.3. Datenbanksystem

Das Datenmodell für den Verzeichnisdienst weist folgende relevante Eigenschaften auf:

- Es existiert ein einziger primärer Schlüssel in Form der Signalinformation
- Es existieren keine Fremdbeziehungen zu Werten anderer Schlüssel

Der Einsatz einer relationalen Datenbank ist aufgrund fehlender Fremdschlüssel nicht sinnvoll. Sind wenig oder keine Fremdbeziehungen zu anderen Datenbankeinträgen vorhanden, kann mit Hilfe von NoSQL-Datenbanken im Rahmen der BASE-Konsistenzanforderungen eine hohe Datenverfügbarkeit gewährleistet und die Schemafreiheit innerhalb der Datensätze ausgenutzt werden [61].

Im freien Angebot kommen Dokument-Datenbanken wie CouchDB [3] , Schlüssel-Wert Datenbanken wie Riak [4] oder spaltenbasierte Datenbanken wie Cassandra [5] in Frage. Graphdatenbanken wie Neo4J [6] bilden relationale Abhängigkeiten ab und sind in diesem Fall nicht relevant.

Die relevanten NoSQL-Datenbanken haben gemeinsam, dass sie einen Datensatz in Form eines Aggregats [62] mit Hilfe eines eindeutigen Schlüssels abspeichern. Ein Aggregat bildet eine transaktionale Konsistenzgrenze, d. h. dass Operationen auf ein Aggregat atomar stattfinden und ACID-Bedingungen [63] erfüllen [64]. Grundsätzlich sind die genannten Datenbanken für die Speicherung der Daten des Verzeichnisdienstes geeignet.

Für den Verzeichnisdienst wird in Anbetracht der potentiell hohen Nutzerzahlen eine hoch performante Datenbank benötigt. Zudem bietet es sich an, die Hierarchie des Datenmodells sinnvoll auszunutzen.

[3] CouchDB, http://couchdb.apache.org (aufgerufen am: 02.07.2018)
[4] Riak von Basho, http://basho.com/products/ (aufgerufen am: 02.07.2018)
[5] Cassandra, http://cassandra.apache.org (aufgerufen am: 02.07.2018)
[6] Neo4J, https://neo4j.com (aufgerufen am: 02.07.2018)

Im Gegensatz zu anderen NoSQL-Datenbanken können mithilfe von Spaltenfamilien-Datenbanken Datensätze anhand ihrer Attribute gruppiert werden [65]. In dieser Arbeit ist eine Gruppierung sinnvoll, um die Hierarchie der Kontextinformationen zu Gunsten der Abfragegeschwindigkeit auszunutzen.

Durch den eindeutigen und effizienten Zugriff auf Aggregate im Cluster einer Spaltenfamilien-Datenbank wird eine hohe Datenverfügbarkeit auch in Situationen in denen Schlüssel häufig und in kurzen Zeitabständen abgefragt oder modifiziert werden, gewährleistet [66].

Spaltenfamilien-Datenbanken

Spaltenfamilien-Datenbanken [65] gehören zur Gruppe der Aggregat-orientierten NoSQL-Datenbanken [6]. Aggregat-orientierte NoSQL-Datenbanken referenzieren Aggregate mit eindeutigen Schlüsseln und unterliegen den BASE-Konsistenzbedingungen. Bei Spaltenfamilien-Datenbanken werden Werte in Spalten gespeichert, die zu Spalten-Familien gruppiert werden können. Eine Spaltenfamilie ist eine Abbildung von Daten und ermöglicht einen schnellen Zugriff auf die dazugehörigen Spalten.

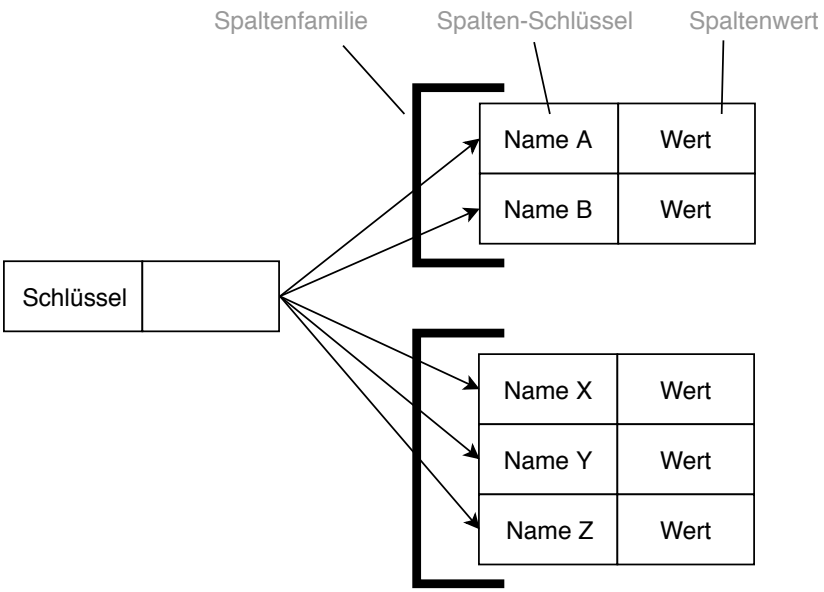

Abbildung 4.8.: Spaltenfamilien-Datenmodell (in Anlehnung an [6])

Im Gegensatz zu relationalen SQL-Datenbanken [67] werden die Daten bei Spaltenfamilien-Datenbanken ausschließlich über den Reihenschlüssel abgefragt. Spalten-Familien ermöglichen

eine weitere Eingrenzung der Abfrage. Durch die Definition der Reihenschlüssel und Spalten-Familien ergibt sich ein unveränderliches Abfragemuster [66].

Bei Spaltenfamilien-Datenbanken sind Transaktionen, die nur Änderungen innerhalb der Daten einer Reihe betreffen, atomar und ACID-konform [6]. Da der Verzeichnisdienst keine Aggregat-übergreifenden Änderungen vornimmt, sind demnach alle relevanten Operationen atomar.

Cassandra ist das einzige frei nutzbare, Spaltenfamilien-Datenbanksystem, weshalb ein Vergleich zu einem alternativen Produkt nicht stattfinden kann. Im folgenden Abschnitt werden die Eigenschaften des Datenbanksystems erläutert.

Cassandra

Cassandra ist ein partitionstolerantes Spaltenfamilien-Datenbanksystem, das als dezentrales Peer-to-Peer System entwickelt wurde [68]. Es wird von renommierten Unternehmen für den Umgang mit großen Datenmengen genutzt [69] [70]. Darunter soziale Plattformen wie Instagram [7] und Reddit [8]

Es gibt keinen Master-Knoten. Jeder Knoten im Cluster kann Lese- oder Schreiboperation ausführen. Die Reihenschlüssel können auf mehrere Knoten verteilt werden. Die dem Reihenschlüssel zugeordneten Spalten werden auf dem selben Knoten gespeichert. Cassandra lässt sich horizontal durch das Hinzufügen weiterer Knoten skalieren. Dabei kann ein Serverausfall vermieden werden. Anfragen können beantwortet, während neue Knoten hinzugefügt werden [71].

Im Umgang mit möglichen Inkonsistenzen wird bei Cassandra ein optimistisches Verfahren angewendet, in welchem Inkonsistenzen auftreten können und bei Eintritt behoben werden [72]. Es kann zwischen eventueller und strenger Konsistenz gewählt werden. Das Konsistenzlevel kann je nach Anforderungen der Anwendung eingestellt werden. Bei hohen Anforderungen an die Konsistenz kann auf ein Quorum der Knoten, auf eine bestimmte Anzahl oder auf alle Knoten gewartet werden, bevor die Operation als erfolgreich gilt. Umso niedriger die Konsistenzanforderungen sind, desto höher ist die Datenverfügbarkeit, aufgrund der schnelleren Verarbeitung von Anfragen.

Schnelle Schreibvorgänge

Cassandra zeichnet sich durch die schnelle Ausführung von Schreiboperationen aus. Bei Schreiboperationen werden Daten im „commit log“ und „memtable“ des Knotens eingetragen

[7] Instagram von Facebook, https://instagram.com (aufgerufen am: 14.05.2018)
[8] Reddit, https://reddit.com (aufgerufen am: 14.05.2018)

und dem Client eine Rückmeldung gesendet, dass die Operation erfolgreich war. Erst danach werden die Daten periodisch in eine neue Tabelle („SSTable“) gespeichert und auf weitere Knoten repliziert. Bei jeder Änderung werden neue Tabellen erzeugt. Die alten Tabellen werden in einer sogenannten „compaction“-Phase entfernt. Bei Leseoperationen mit geringem Konsistenzlevel werden auch nicht aktualisierte Daten zurückgesendet. In einem solchen Fall aktualisiert die Datenbank die Daten nach der Antwort an den Client für zukünftige Anfragen („read-repair“-Mechanismus). Ausstehende Aufträge zur Bearbeitung von Anfragen werden auf andere Knoten verteilt, so dass diese bei einem Ausfall des zuständigen Knoten übernommen werden können („hinted handoff“-Mechanismus) [66].

Consistent Hashing mit virtuellen Knoten

Zur Verbesserung der Ausfallsicherheit und Verfügbarkeit wird bei Cassandra Consistent Hashing mit virtuellen Knoten eingesetzt. Beim Consistent Hashing wird die Adressberechnung sowohl für die Knotenadressen als auch für die Speicheradressen der Objekte verwendet. Bei Cassandra ist ein solches Objekt das Aggregat einer Reihe. Es gibt demnach einen gemeinsamen Adressraum, in dem die Objekte auf dem Knoten mit der nächst größeren Adresse gespeichert werden. Das ermöglicht flexible Rechnerstrukturen, bei denen jederzeit Knoten hinzugefügt oder entfernt werden können. Änderungen haben nur Auswirkungen auf die Objekte in unmittelbarer Nähe zu dem veränderten Knoten im Ring. Wird ein Knoten hinzufügt, müssen nur die Objekte des nächsten Knotens verlagert werden, deren Adressen unterhalb der neuen Knotenadresse liegen. Wird ein Knoten entfernt, werden die Objekte auf den nächst höheren Knoten verlagert [73].

Die Ausfallsicherheit und Verfügbarkeit wird erhöht, indem Knoten repliziert werden. Dazu werden die Kopien der Objekte mit Versionsnummern versehen und auf dem Ring eingetragen. Cassandra verwendet zudem virtuelle Knoten, um die Objekte gleichmäßiger auf die Knoten zu verteilen. Die Knoten erhalten ebenfalls Versionsnummern [74], um sie auf dem Ring zu synchronisieren.

Partition und Schlüssel

Die Aufgabe des Partition Key’s (engl. für Partitionsschlüssel) ist es die Partition zu identifizieren, auf der das Objekt gespeichert wurde. Eine Partition befindet sich auf einem Knoten im Ring, dem ein bestimmter Wertebereich zugeordnet ist.

Ein Partitioner (engl. für Partitionierer) kann so konfiguriert werden, dass dieser entweder die Partitionsschlüssel gleichmäßig auf die Knoten verteilt oder in einer bestimmten Reihenfolge ablegt. Die geordnete Erstellung von Partition Key’s kann dazu führen, dass ein Knoten unter

stärkerer Belastung steht als ein anderer, weshalb die gleichmäßige Verteilung über willkürliche Partition Key's empfohlen wird.

Wenn Daten gelesen oder geschrieben werden, wird der Hashwert des Partitionsschlüssels berechnet und mit den Wertebereichen der Knoten verglichen (s. Abschnitt 4.5.3). Liegt der Hashwert im Wertebereich eines Knotens befindet sich dort die gesuchte Reihe.

Die Verteilung der Datensätze auf die Knoten im Cluster ist in dieser Arbeit hauptsächlich abhängig von der ID des Signals. Hierbei wirkt sich insbesondere die willkürliche Erzeugung von UUIDs positiv auf die gleichmäßige Verteilung im Cluster aus.

Der Partitionsschlüssel ist der erste Eintrag in der Definition des Primary Key's (engl. für Primärer Schlüssel). Ein Partitionsschlüssel, der mehrere Attribute umfasst, wird Composite Partition Key genannt (engl. für zusammengesetzter Partitionsschlüssel). Wird der primäre Schlüssel neben dem Partitionsschlüssel um weitere Attributnamen ergänzt, sorgen diese als *Clustered Keys* in der Reihenfolge in der sie hinzugefügt wurden, für eine Gruppierung der Datensätze. Dabei kann eine auf- oder absteigende Sortierung der Gruppierung eingestellt werden. Dabei muss berücksichtigt werden, dass die Attribute bei INSERT oder UPDATE Operationen anzugeben sind [75].

Datenbankmodell

Unter Berücksichtigung der Funktionsweise der Spaltenfamilien-Datenbank Cassandra und der Schlüssel-Wert Beziehung aus Abbildung 4.7 ergibt sich das in Abbildung dargestellte 4.9 Modell. Zur Verbesserung der Übersicht in diesem Beispiel wurden für die IDs der Eigentümer und Einträge Integer anstelle der vorgesehenen UUIDs verwendet.

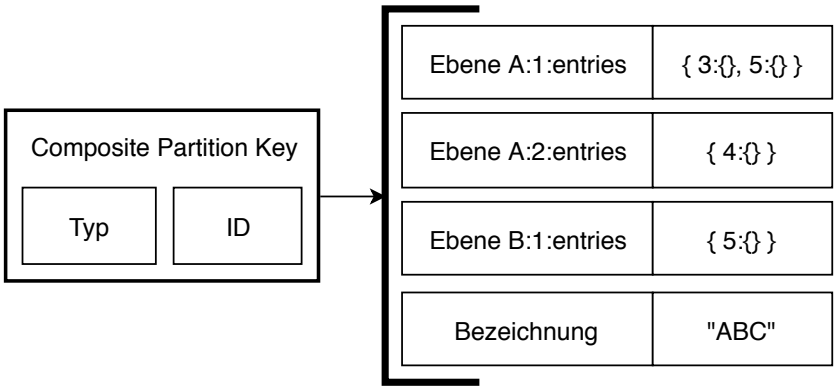

Abbildung 4.9.: Cassandra Datenmodell

In Cassandra kann standardmäßig jeder Datensatz mit einem Zeitstempel (engl.: Timestamp) des letzten Schreibvorgangs versehen werden, der automatisch aktualisiert wird. Zudem kann die Lebenszeit (engl.: Time-to-live) der Datensätze bei jeder INSERT oder UPDATE Operation auf Sekunden begrenzt werden [76]. Die Kombination dieser Funktionen bietet eine praktische Möglichkeit ungültige oder nicht mehr verwendete Daten zu entfernen, um die Datenmenge zu reduzieren und Kosten zu sparen.

Es können zur Steigerung der Verfügbarkeit und Abfrageeffizienz niedrige Konsistenzanforderungen für die Datenbank eingestellt werden. Der Read-Repair-Mechanismus (s. Abschnitt 4.5.3) sorgt in jedem Fall dafür, dass ältere Daten nach dem Lesen aktualisiert werden. Im ungünstigsten Fall kann es passieren, dass ein Client nicht den Wert seines letzten Schreibvorgangs, sondern einen älteren Wert liest. Dazu müssen Daten über mehrere Replikate verteilt sein, der Schreibvorgang vor Abschluss der Synchronisierung aller Replikate bestätigt werden und eine neue Anfrage an denselben Datensatz desselben Clients erfolgen.

Da der Use Case der Anwendung vorsieht, laufend auf neue Signale zu reagieren und Änderungen abzufragen bzw. über einen Server der Anwendung zu abonnieren, kann sich die Informationsübertragung im schlimmsten Fall bis zum nächsten Updatezyklus verzögern. Diese Anforderung richtet sich jedoch an den Entwurf eines anwendungsbezogenen Servers, der die Kontextinformationen verwaltet und das Verzeichnis zur Verknüpfung dieser Informationen mit signalbezogenen IDs nutzt.

4.5.4. Kommunikation

Für die Kommunikation mit anderen Systemen ist eine nachhaltig gültige Schnittstellendefinition besonders wichtig, da nachträgliche Änderungen an dieser Stelle oftmals einen erhöhten Aufwand durch clientseitige Software-Aktualisierungen erforderlich machen.

Die Komplexität der Schnittstellendefinition hängt von der geforderten Flexibilität hinsichtlich der Datenabfrage ab. Die Verzeichnisstruktur wurde auf das Wesentliche reduziert und beschränkt sich auf die Abfragen der Meta- und Kontextinformationen anhand einer eindeutigen ID. Die Aufgabe der Schnittstelle besteht darin, diese Abfragen für externe Anwendungen zugänglich zu machen.

Eigenschaften von flexiblen Schnittstellendefinitionen, in denen der Client die Antwort des Servers durch individuelle Anfragen beeinflussen kann, wie es z. B. mit GraphQL [9] möglich ist, können in diesem Fall nicht sinnvoll eingesetzt werden.

[9] GraphQL ist eine Abfragesprache für APIs, https://graphql.org (aufgerufen am: 04.06.2018)

Eine Schnittstellendefinition nach dem Programmierparadigma des Representational State Transfers (REST) eignet sich, um eine einfache, verständliche und eindeutige Schnittstelle zur Abfrage von Kontextinformationen zu entwickeln [77].

REST-Paradigma

Das REST-Paradigma wird in vielen Anwendungen auf unterschiedliche Art und Weise angewendet, um Ressourcen bereitzustellen und abzufragen. REST wird für Webschnittstellen verwendet und setzt daher in der Regel auf dem Protokoll HTTP [10] auf. Nach dem Richardson Maturity Model [78] lassen sich vier Level von Implementationen des Architekturstils unterscheiden:

Level 0: In diesem Level des Modells wird HTTP als Transportsystem für die Umsetzung von entfernten Prozeduraufrufen (engl.: Remote Procedure Calls) genutzt.

Level 1: In diesem Level werden die Daten in Ressourcen unterteilt. Auf jede Ressource kann über eine eindeutige URL zugegriffen werden.

Level 2: In diesem Level werden definierte Verben verwendet, so dass die Adressierung einer Ressource je nach Verb unterschiedliche Reaktionen auslösen kann. Bei der Verwendung der HTTP Befehlswörter, werden die Verben GET, POST, PUT und DELETE verwendet.

Level 3: In diesem Level werden im Sinne des HATEOAS-Ansatzes [11] Hypermedia controls verwendet.

Entwurf der REST-konformen Schnittstelle

Der Entwurf der Schnittstelle des Verzeichnisdienstes orientiert sich am Level 2 der beschriebenen REST-Implementationsarten. Auch wenn die Nutzung von Hypermedia Controls durch die Nutzung von JSON Hyper-Schema [1] im Wertebereich des Eintrages (s. Abschnitt 4.4.6) möglich ist, ist eine zwingende Verwendung von Hyperlinks für dynamische Navigationselemente nicht vorgesehen.

Jedoch wird empfohlen, dass Anwender die Einträge mit mindestens einem Hyperlink ergänzen, um auf die entsprechende API zu verweisen, über welche die Ressourcen des Eintrages

[10] HTTP: Hypertext Transfer Protocol

[11] Das Akronym HATOEAS steht für: Hypertext As The Engine Of Application State HATEOAS ist eine Ausprägung des REST-Architekturstils, in der der Server durch Hypermedia Links in der Antwort einer Anfrage die Navigationsmöglichkeiten des Clients dynamisch vorgibt.

und des Eigentümers abgefragt werden können. Über diesen Weg kann eine verständliche und anwenderfreundliche API erzeugt werden. Ohne Hyperlinks wird das Wissen über die Adresse zur externen API vorausgesetzt und muss an anderer Stelle dokumentiert werden.

Das folgende Beispiel zeigt einen Eintrag mit Verweis auf eine fiktive externe API im JSON Hyper-Schema:

```
7eedfbecd8e740da92a159868dabf0af : {
  "base": "http://api.someapplicationname.com/",
      "links": [
        {
          "rel": "self",
          "href": "pathToResource/7eedfbecd8e740da92a159868dabf0af",
        }
      ]
}
```

Listing 4.1: REST-API: Beispiel einer API-Referenz mittels JSON Hyper-Schema [1]

Nachfolgend wird die Definition der API-Schnittstelle dargestellt. Die URL der API spiegelt die in Abschnitt 4.4.9 festgelegte Verzeichnisstruktur wider. Weitere Parameter, z. B. zur Kontrolle von Zugangsberechtigungen, sind nicht Bestandteil dieser Arbeit.

Einträge: Der folgende Aufruf fügt einen neuen Eintrag zu einem Eigentümer in einer bestimmten Ebene zu einem bestimmten Signal hinzu:

```
Request:
POST
/types/{type}/ids/{id}/layers/{layer}/owners/{owner}/entries/{entryId}
Body:
{
   "name"    : "",
   "entry"   : "",
}

Response: HTTP-Statuscode: 200 (Ok)

Response: HTTP-Statuscode: 500 (Internal Server Error)
```

Listing 4.2: REST-API: Hinzufügen eines Eintrags eines Eigentümers in einer Ebene

Im Body des Requests werden die aktuelle Bezeichnung zum Informationsschlüssel und der Eintrag übergeben. Der Status *Internal Server Error* wird gesendet, wenn das Hinzufügen oder Überschreiben des Eintrages nicht möglich war. Dieser Fall sollte niemals eintreten.

Der folgende Aufruf fügt einen neuen Eintrag zu einem Eigentümer in einer bestimmten Ebene zu einem bestimmten Signal hinzu:

```
Request:
DELETE
/types/{type}/ids/{id}/layers/{layer}/owners/{owner}/entries/{entryId}

Response: HTTP-Statuscode: 200 (Ok)

Response: HTTP-Statuscode: 204 (No Content)
```

Listing 4.3: REST-API: Entfernen eines Eintrags eines Eigentümers in einer Ebene

Der folgende Aufruf gibt alle Einträge eines Eigentümers in einer bestimmten Ebene zu einem bestimmten Signal zurück:

```
Request:
GET /types/{type}/ids/{id}/layers/{layer}/owners/{owner}

Response: HTTP-Statuscode: 200 (Ok)
{
  entries : []
}
```

Listing 4.4: REST-API: Anfrage der Einträge eines Eigentümers in einer Ebene

Der folgende Aufruf gibt alle Einträge aller Eigentümer in einer bestimmten Ebene zu einem bestimmten Signal zurück:

```
Request:
GET /types/{type}/ids/{id}/layers/{layer}

Response: HTTP-Statuscode: 200 (Ok)
Body:
{
  entries : []
}
```

Listing 4.5: REST-API: Anfrage der Einträge aller Eigentümer in einer Ebene

Der folgende Aufruf gibt alle vorhandenen Eigentümer aus dem in der Anfrage mitgelieferten Array und deren Einträge in einer bestimmten Ebene zu einem bestimmten Signal zurück.

```
Request:
PUT /types/{type}/ids/{id}/layers/{layer}/owners
Body:
{
  owners:[]
}

Response: HTTP-Statuscode: 200 (Ok)
Body:
{
  entries : []
}
```

Listing 4.6: REST-API: Anfrage der Eigentümer in einer Ebene anhand von IDs

Eigentümer: Der folgende Aufruf gibt alle vorhandenen Eigentümer in einer bestimmten Ebene zu einem bestimmten Signal zurück:

```
Request:
GET /types/{type}/ids/{id}/layers/{layer}/owners

Response: HTTP-Statuscode: 200 (Ok)
Body:
{
    owners : []
}
```

Listing 4.7: REST-API: Anfrage aller Eigentümer in einer Ebene

Der folgende Aufruf löscht einen Eigentümer in einer bestimmten Ebene zu einem bestimmten Signal:

```
Request:
DELETE /types/{type}/ids/{id}/layers/{layer}/owners/{owner}

Response: HTTP-Statuscode: 200 (Ok)

Response: HTTP-Statuscode: 204 (No Content)
```

Listing 4.8: REST-API: Löschen eines Eigentümers in einer Ebene

Ebenen: Der folgende Aufruf löscht eine Ebene, alle in dieser Ebene eingetragenen Eigentümer sowie deren Einträge zu einem bestimmten Signal:

```
Request:
DELETE /types/{type}/ids/{id}/layers/{layer}

Response: HTTP-Statuscode: 200 (Ok)

Response: HTTP-Statuscode: 204 (No Content)
```

Listing 4.9: REST-API: Löschen einer Ebene

4.5.5. Bearbeitung von Anfragen

Erstellen oder Aktualisieren eines Eintrages

Die Operationen zum Erstellen oder Aktualisieren des Eintrages durchlaufen die selbe Plausibilitätsprüfung, weshalb sie zu einer Operation zusammengefasst werden können.

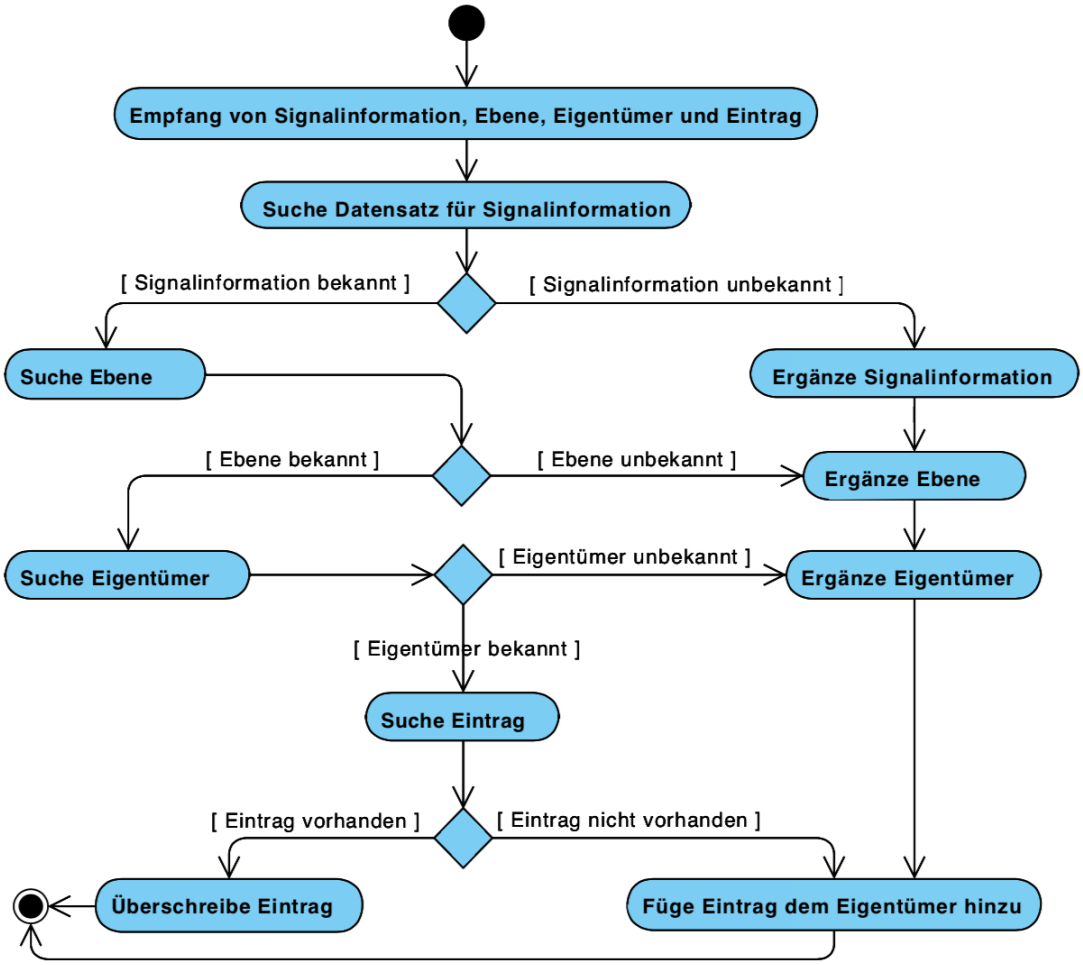

Abbildung 4.10.: Aktivitätsdiagramm zum Erstellen oder Aktualisieren eines Eintrages

Lesen von Einträgen

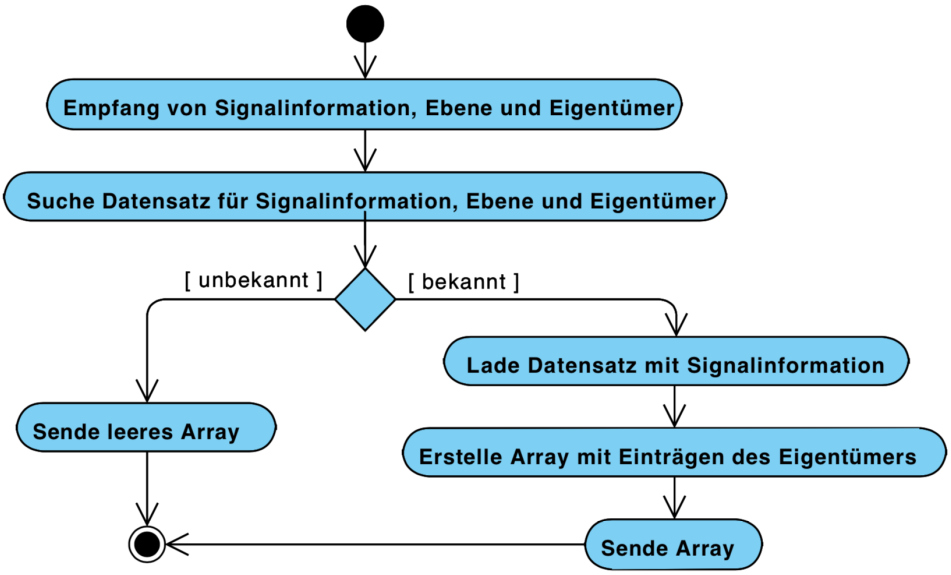

Abbildung 4.11.: Aktivitätsdiagramm zum Lesen von Einträgen

Löschen eines Eintrages

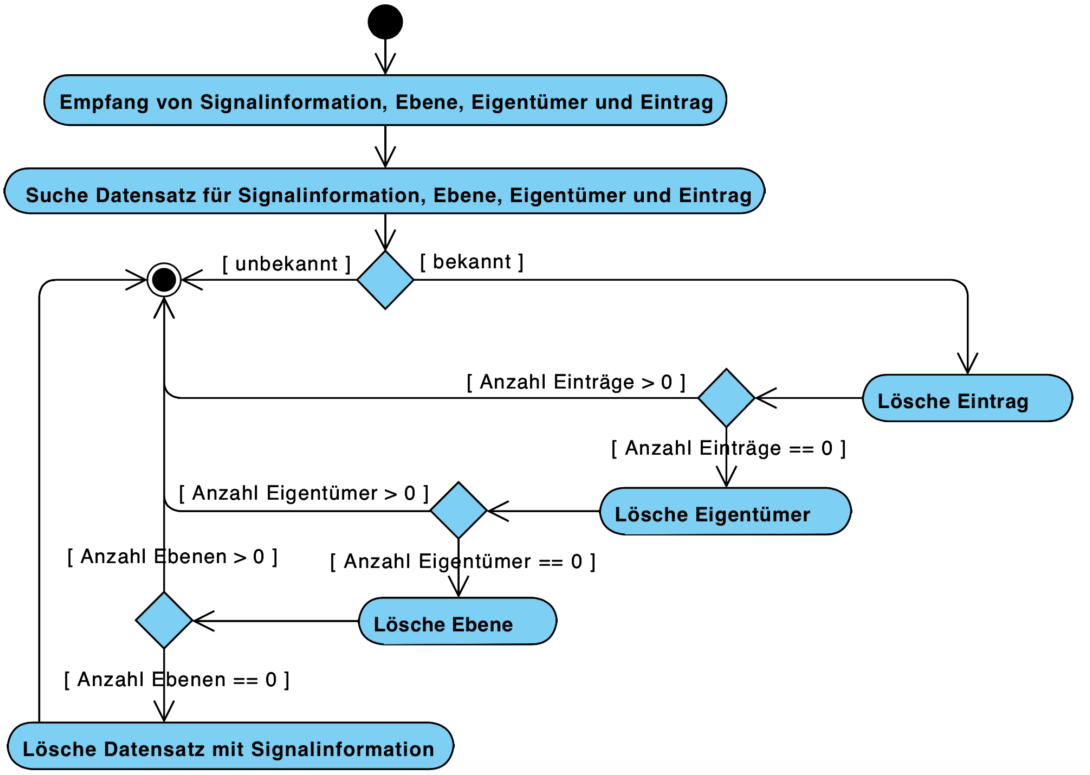

Abbildung 4.12.: Aktivitätsdiagramm zum Löschen eines Eintrages

4.5.6. Skalierbare Serverarchitektur

Der Server nimmt über die I/O-Schnittstelle die HTTP-Requests entgegen. Bei steigenden Nutzerzahlen kann die Anzahl der eintreffenden Anfragen stark ansteigen und die Laufzeitumgebung des Servers zunehmend beanspruchen. Deshalb darf das System an keiner Stelle blockieren und muss in der Lage sein, die Last die sich aus den Anfragen ergibt, effizient zu verteilen.

Die Vorkehrungen, um das System skalierbar zu machen, lassen sich in Stufen unterteilen:

1. Nicht blockierende, eventbasierte Kommunikation
2. Verteilung der Last auf die verfügbaren Kerne der CPU
3. Verteilung der Last auf mehrere Maschinen

Nicht blockierende, eventbasierte Kommunikation

Node.js [12] ist eine asynchrone Laufzeitumgebung basierend auf Chrome's V8 [13] JavaScript Engine.

Node.js wurde entwickelt, um skalierbare Netzwerk Anwendungen zu erstellen. Node.js nutzt ein eventgetriebenes, nicht blockierendes I/O-Modell, das es leichtgewichtig und effizient macht. Viele Verbindungen können nebenläufig über eine Event-Schleife bearbeitet werden. Bei jeder neuen Verbindung wird ein Callback (engl. für Rückruf) ausgeführt. Wurden alle angestoßenen Programmabschnitte durchlaufen, schläft Node.js bis zum nächsten Callback [79].

Die Event-Schleife steht in Kontrast zum thread-basierten Nebenläufigkeitsmodell. Beim thread-basierten Modell können Probleme durch ineffiziente oder falsche Programmierung des wechselseitigen Zugriffs auf Ressourcen auftreten. Im Gegensatz zum eventbasierten Node.js können durch die Nutzung von Zugriffsperren Verklemmungen auftreten, die den Prozess blockieren.

Die Event-Schleife erlaubt Node.js nicht-blockierende I/O-Operationen auszuführen. JavaScript verfügt nur über einen Thread, was thread-basierte Verklemmungen aufgrund von wechselseitigem Zugriff auf Ressourcen ausschließt. Bei Node.js werden zudem Operationen auf den Kernel des Systems ausgelagert.

In Node.js hat kaum eine Funktion direkten Zugriff auf die I/O-Schnittstelle, weshalb es zu keiner Blockierung kommt. Da moderne Kernel mehrere Threads nutzen, können mehrere

[12] Node.js, https://www.nodejs.org (aufgerufen am: 22.06.2018)
[13] Google Chrome V8, https://developers.google.com/v8/?hl=de (aufgerufen am: 18.05.2018)

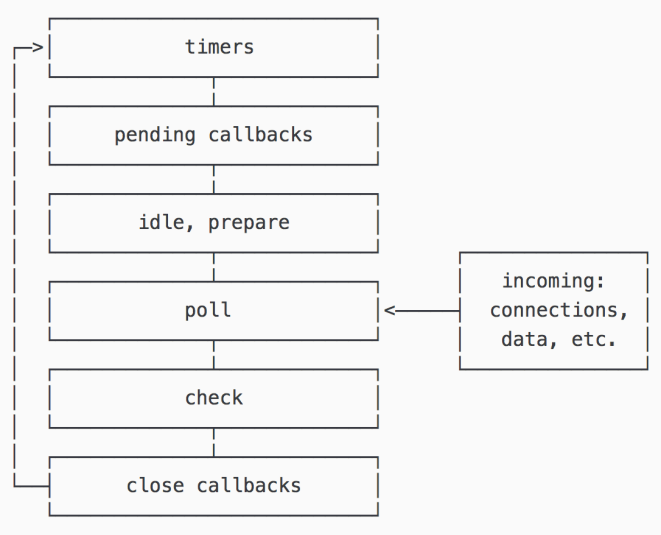

Abbildung 4.13.: Node.js Event Schleife [7]

Operationen im Hintergrund nebenläufig ausgeführt werden. Die Planung der Ausführungsreihenfolge (engl.: Scheduling) wird dabei vom Kernel übernommen. Ist eine Operation abgeschlossen, erhält Node.js vom Kernel eine Meldung und der entsprechende Callback wird zur Poll-Schleife hinzufügt, um zu einem geeigneten Zeitpunkt ausgeführt zu werden (s. Abbildung 4.13) [7].

Die Event-Schleife wird automatisch mit dem Input Skript gestartet und gestoppt wenn es keine Callbacks mehr gibt, die abgearbeitet werden müssen. Damit bleibt die Event-Schleife dem Nutzer vollständig verborgen.

Während sich bei thread-basierten Modellen bei zunehmender Skalierung die Gefahr für Verklemmungen erhöht, wird dieses Risiko bei Node.js vermieden [79].

Verteilung der Last auf die verfügbaren Kerne der CPU

Obwohl in Node.js nur ein Thread existiert, lassen sich die Vorteile von mehreren Kernen auszunutzen. Über die child_process.fork() API [80] können Unterprozesse erstellt werden. Die Schnittstelle bietet die Grundlage für das Node.js Cluster Modul [81] durch das Sockets zwischen Prozessen geteilt werden können und eine Lastverteilung (engl.: Loadbalancing) über die Kerne möglich ist. Der Hauptprozess (engl.: master) hört auf einen Port und verteilt die Verbindungen auf die Unterprozess (engl.: worker). Damit wird die Last gleichmäßig auf die Unterprozesse verteilt.

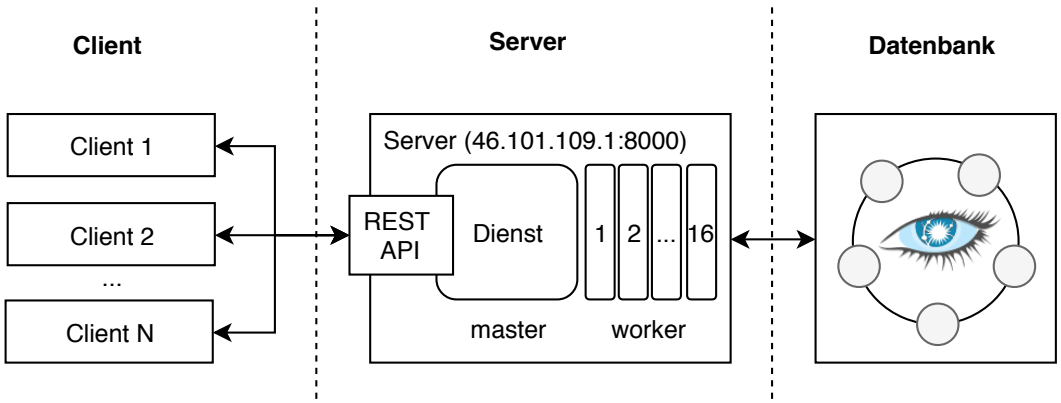

Abbildung 4.14.: Architektur mit Lastverteilung auf verfügbare Kerne der CPU

Durch das Node.js Cluster Modul kann ohne weitere Software eine Lastverteilung implementiert werden. Alternativen sind Nginx [14] oder nftables [15], die die Lastverteilung über die Ansprache der Node.js-Unterprozesse über verschiedene Ports erreichen.

Verteilung der Last auf mehrere Maschinen

Um die Anfälligkeit für Fehler zu reduzieren und unnötige Kosten für die Bereitstellung von Servern zu vermeiden, ist eine Implementierung der Lastverteilung erst bei erhöhter Nachfrage sinnvoll. Das Design erlaubt eine nachträgliche horizontale Skalierung zur Laufzeit. Eine Automatisierung des Auslieferungsprozesses (engl.: Continuous Deployment) kann über die Verwendung von Docker [16] -Containern [17]. erreicht werden. Die Verteilung der Last auf unterschiedliche Maschinen ist über Software wie Nginx [18] möglich. Diese nimmt als Reverse Proxy [82] die HTTP-Aufrufe entgegen und verteilt diese nach Kriterien, wie zum Beispiel der geringsten Anzahl aktiver Verbindungen, der geringsten durchschnittlichen Antwortzeit oder nach dem Rundlauf-Verfahren (engl.: Round-Robin).

[14] Nginx Webserver-Software, https://www.nginx.com (aufgerufen am: 14.05.2018)
[15] Nftables von Netfilter, https://netfilter.org (aufgerufen am: 14.05.2018)
[16] Docker, https://www.docker.com (aufgerufen am: 13.06.2018)
[17] What is a container, https://www.docker.com/what-container (aufgerufen am: 13.06.2018)
[18] Nginx, https://www.nginx.com (aufgerufen am: 13.06.2018)

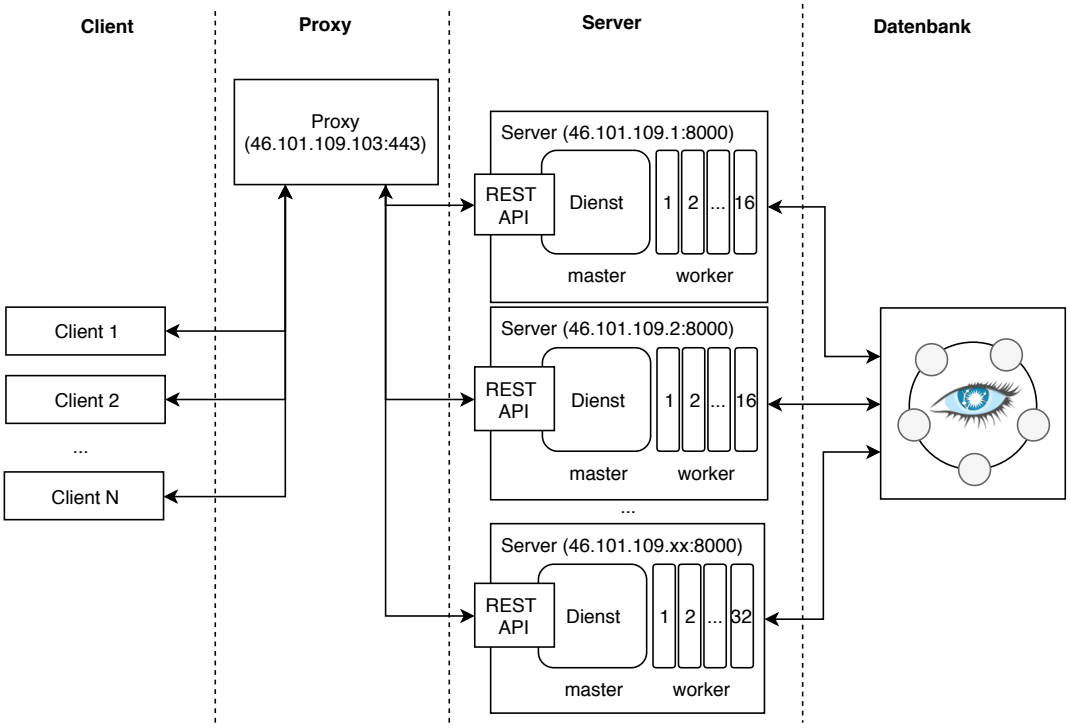

Abbildung 4.15.: Architektur mit Proxy zur Lastverteilung auf mehrere Maschinen

4.6. Zusammenfassung

Um den Datenzugriff hinsichtlich der Effizienz, der Skalierbarkeit und der Datenverfügbarkeit zu optimieren, wird die Datenstruktur in Form einer einfachen Schlüssel-Wert Beziehung aus Signal- und Kontextinformationen abgebildet. Durch den primären Schlüssel aus *Typ* und *ID* wird ein effizienter Zugriff auf die signalbezogenen Kontextinformationen in einem Cluster gewährleistet und die Replikation und Synchronisation der Datensätze erleichtert. Der Schlüssel wird zudem um eine Indexierung der Werte-Attribute *Ebene* und *Eigentümer* ergänzt, so dass eine optimale Sortierung der Datensätze vorliegt.

Da es nur einen einzigen primären Schlüssel in Form der Signalinformation gibt und keine Fremdbeziehungen zu anderen Tabellen existieren, bietet sich für eine effiziente Abfrage die Sortierung der Datensätze mit Hilfe einer Spaltenfamilien-Datenbank an.

Die Spaltenfamilien-Datenbank Cassandra zeichnet sich insbesondere durch die schnelle Ausführung von Schreiboperationen aus. Bei Schreiboperationen werden Daten im Knoten temporär hinterlegt und dem Client eine Rückmeldung gesendet, dass die Operation erfolgreich

war. Erst danach werden die Daten periodisch gespeichert und auf weitere Knoten repliziert. Somit muss der Client nicht den vollständigen Speichervorgang abwarten.

Bei hohen Nutzerzahlen werden Schlüssel häufig und in kurzen Zeitabständen abgefragt und es muss eine große Datenmenge effizient verwaltet werden. Dank der einfachen Datenstruktur, die atomare Zugriffe erlaubt, können zur Steigerung der Verfügbarkeit und Abfrageeffizienz niedrige Konsistenzanforderungen für die Datenbank angenommen werden.

Zur Verbesserung der Ausfallsicherheit und Verfügbarkeit wird bei Cassandra Consistent Hashing mit virtuellen Knoten eingesetzt. Dabei wird die Adressberechnung sowohl für die Knotenadressen als auch für die Speicheradressen der Objekte verwendet.

Um eine einfache, verständliche und eindeutige Schnittstelle zur Abfrage der als Ressourcen angesehenen Kontextinformationen zu entwickeln, ist die Schnittstelle nach dem Programmierparadigma des Representational State Transfers (REST) definiert [77]. Dabei ist die Nutzung von *Hypermedia Controls* im Wertebereich des Eintrages optional.

Die Event-Schleife des Node.js Servers erlaubt es nicht-blockierende I/O Operationen auszuführen. Es werden Verklemmungen ausgeschlossen und wenn möglich, Operationen auf den Kernel des Systems ausgelagert.

Durch das Node.js Cluster Modul wird eine Lastverteilung auf die verfügbaren Kerne des Rechensystems implementiert.

Als Reaktion auf eine erhöhte Nachfrage erlaubt das Design eine horizontale Skalierung über die Verteilung der Last auf mehrere Maschinen über einen Reverse Proxy.

5. Implementierung

In diesem Kapitel werden die wesentlichen Programmabschnitte aufgeführt, die zur Erstellung und Nutzung des Verzeichnisdienstes benötigt werden. Diese umfassen die Erzeugung und Konfiguration der Spaltenfamilien-Datenbank Cassandra sowie die Bearbeitung von API-Anfragen. Darüber hinaus werden Möglichkeiten zur Lastverteilung auf mehrere Kerne und Maschinen aufgezeigt.

5.1. Erzeugung einer verteilten Datenbank mit Cassandra

Im Folgenden wird die Erstellung des Verzeichnisses anhand von CQL [1] dargestellt. Bevor die Tabelle erstellt werden kann, ist die Festlegung des übergeordneten „Keyspace“ (engl. für Schlüsselraum) erforderlich. Jeder Keyspace erfordert die Konfiguration des Distributions- und Replikationsverhaltens. Dabei wird die Platzierungsstrategie für Replikate, der Replikationsfaktor und die Einstellung zum dauerhaften Festschreiben für alle Tabellen vorgenommen. Dauerhaftes Festschreiben ist standardmäßig aktiv [85].

Konfiguration des Keyspace

```
1 CREATE KEYSPACE signalcontext WITH REPLICATION =
2 {
3   'class'   : 'SimpleStrategy',
4   'replication_factor' : 1
5 }
6 AND DURABLE_WRITES = true;
```

Listing 5.1: Cassandra: Konfiguration des Keyspace

Als Alternative zur Klasse *SimpleStrategy* kann über die Klasse *NetworkTopologyStrategy* definiert werden, wie viele Replikate in unterschiedlichen Datenzentren platziert werden

[1] CQL steht für Cassandra Query Language, und bietet ein Modell zur Abfrage von Daten, die stark dem weitverbreiteten Standard SQL [83] ähnelt. Daten werden in Tabellen mit Reihen und Spalten dargestellt. Die Reihen werden mit Daten aus den Spalten gefüllt, so dass die Darstellung der Daten einer SQL-Abfrage gleicht. Neben der gewohnten und anschaulichen Darstellung stehen auch mehrere Befehle zur Verfügung, die aus dem SQL-Gebrauch bekannt sind [84].

sollen. Die Replikationsstrategie vermeidet die Verteilung von Replikaten auf den selben Serverschrank, so dass möglichst keine gleichen Replikate bei einem Ausfall eines Serverschranks betroffen sind und immer mindestens ein Replikat zur Verfügung steht. Im Rahmen dieses Prototyps ist die *SimpleStrategy* ausreichend, da aufgrund der geringen Nutzerzahlen die Verteilung über mehrere Datenzentren nicht notwendig ist.

Erstellung der Verzeichnis-Tabelle

```
CREATE TABLE directory.signalcontext (
  type text,
  id text,
  name text,
  layer text,
  owner text,
  entrymap map<text, text>,
  ts_write timestamp,
  PRIMARY KEY ((type, id), layer, owner)
) WITH CLUSTERING ORDER BY (layer DESC);
```

Listing 5.2: Cassandra: Erstellung der Verzeichnis-Tabelle

5.2. Erstellung von Methoden zur Bearbeitung der API-Anfragen

Die in Abschnitt 4.5.5 entworfene REST-API, wird mit Hilfe des HTTP-Clients des Web-Frameworks Express [2] realisiert. Dieser nimmt API-Anfragen über die HTTP-Schnittstelle des Node.js Servers entgegen und stellt sie der Anwendung zur Verfügung. Anhand einer Callback-Funktion der jeweiligen Anfrage-Methode, kann durch die Servicelogik eine entsprechende HTTP-Antwort vorbereitet und zurückgesendet werden. Die Servicelogik greift dabei mit Hilfe eines Cassandra-Clients [3] auf die Daten der Cassandra-Datenbank zu. Der Vorgang ist für den Client synchron, d. h., dass dieser aktiv auf eine Rückmeldung des Servers wartet.

Die in Abbildung 5.1 dargestellte Beispielsequenz für die Verarbeitung einer Anfrage, zeigt die Interaktion der beteiligten Komponenten.

Die vollständige Implementierung der Servicelogik befindet sich im Anhang dieser Arbeit (s. A.1.2).

[2] Express, http://expressjs.com (aufgerufen am: 03.08.2018)
[3] Cassandra-Driver, https://www.npmjs.com/package/cassandra-driver (aufgerufen am: 03.07.2018)

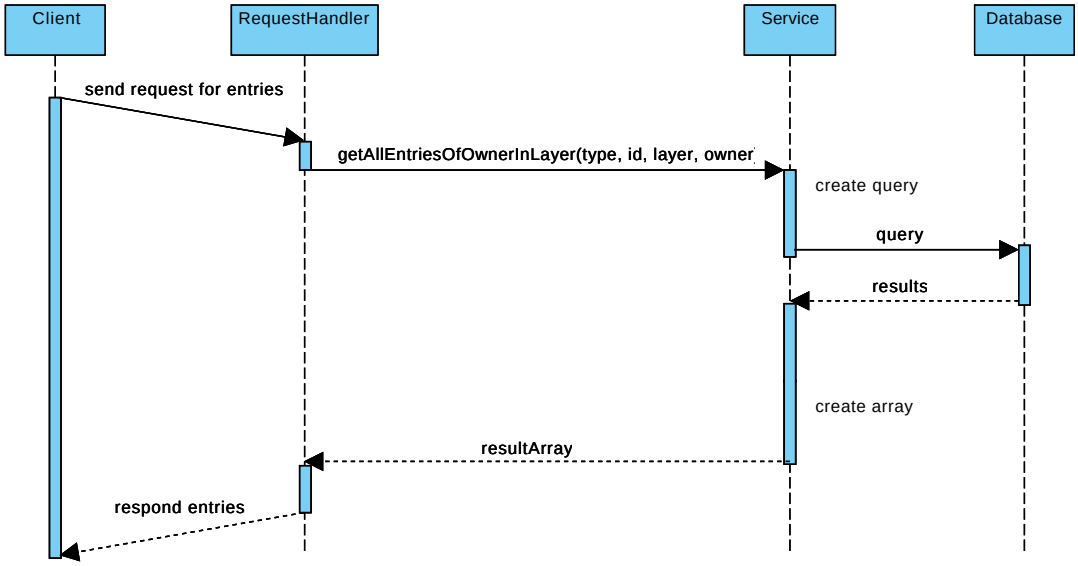

Abbildung 5.1.: Sequenzdiagramm: Anfrage für alle Einträge eines Eigentümers in einer Ebene

5.3. Erstellung eines Node.js Clusters über die verfügbaren Kerne

Die erforderlichen Programmabschnitte zur Erstellung des Clusters und der clusterbezogenen Unterprozesse (engl.: Worker) in Abhängigkeit der Anzahl der Kerne, wird im folgenden Beispiel [2] dargestellt:

Cluster

```
class Cluster {
  constructor () {
    if (cluster.isMaster) {
      this.fork()
    }
    else {
      new Worker()
    }
  }

  fork () {
    let cpus = os.cpus().length
```

```
13
14      for (let i = 0; i < cpus; i++) {
15        cluster.fork({id: i})
16      }
17    }
18  }
19
20  new Cluster()
```

Listing 5.3: Node.js Cluster Module: Definition der Cluster Klasse [2]

Worker

```
1  let id;
2
3  class Worker {
4    constructor () {
5      id = Number(process.env.id)
6      this.webserver()
7    }
8
9    webserver () {
10
11     // [...] REST-API, Cassandra-Client und andere Bibliotheken
12
13     let server = http.createServer((req, res) => {
14       res.writeHead(200)
15       res.end('ok')
16     }).listen(80, () => {
17       console.log('Worker', id, 'listening on port',
18       server.address().port)
19     })
20   }
21 }
22
23 module.exports = Worker
```

Listing 5.4: Node.js Cluster Module: Definition der Worker Klasse [2]

Im nächsten Kapitel wird ein Experiment präsentiert, indem die Bedingungen für die Nutzung von Wi-Fi- und Bluetooth-Beacons an öffentlichen Standorten für den Informationsaustausch mit Hilfe des Verzeichnisdienstes untersucht werden.

6. Diskussion

Der Verzeichnisdienst ermöglicht die Bereitstellung von Informationen zu einer eindeutigen signalbezogenen ID. In diesem Kapitel wird untersucht, unter welchen Bedingungen dem Nutzer an öffentlichen Standorten eine entsprechende ID zur Verfügung steht und ein Informationsaustausch mit anderen Signalempfängern möglich ist.

6.1. Experiment zum Empfang von Beacons an öffentlichen Standorten

6.1.1. Thema

In diesem Experiment soll untersucht werden, ob der entwickelte Verzeichnisdienst den Austausch von Informationen anhand von öffentlich zugänglichen Signalinformationen ermöglicht. Dazu wird die Nutzung von öffentlichen Wi-Fi-Hotspots mit der Verwendung von nachträglich installierten Eddystones bzw. iBeacons verglichen.

6.1.2. Vermutung

Es wird davon ausgegangen, dass die Wi-Fi-Hotspots an den Standorten jeweils über mehrere Accesspoints angeboten werden. Daher ist es möglich, dass Empfänger mit unterschiedlichen Accesspoints verbunden sind. Nur wenn die Nutzer dieselben Signalinformationen zur Abfrage von Informationen verwenden, kann ein Informationsaustausch über das Verzeichnis stattfinden.

6.1.3. Versuchsaufbau

Für die Dokumentation von Wi-Fi-Accesspoints und Bluetooth-Beacons ist eine Anwendung notwendig, die eine Liste der empfangenen, relevanten Signalinformationen anzeigen kann. Relevant sind für Wi-Fi-Accesspoints die SSID und BSSID, für iBeacons die UUID, die Major- und Minor-Werte und für Eddystone-Beacons der Namespace und die Instance ID.

Es wurden zwei Standorte in der Kölner Innenstadt ausgewählt, die sich in der Nähe des Doms befinden. Die Standorte wurden in Bereiche unterteilt. In Abbildung 6.3 und 6.4 sind die Bereiche und Standorte der einzelnen Messungen alphabetisch dargestellt. Die Markierungen „B1“, „B2“ und „B3“ stellen die Installationspunkte der Bluetooth-Beacons dar. Vor den Messungen an den jeweiligen Standorten wurden die iBeacons und Eddystone-Beacons an den gekennzeichneten Positionen platziert. Die genaue Anzahl oder die Position der Wi-Fi-Accesspoints ist nicht bekannt.

In jedem Bereich muss die WLAN-Verbindung erneuert werden. Die empfangenen WLAN- und Bluetooth-Signalinformationen der relevanten Signalquellen werden dokumentiert. Nach einer Sendephase, in der in jedem Bereich einmalig Informationen auf Grundlage der empfangenen Signalinformationen bereitgestellt werden, folgt eine Empfangsphase, in der sich zeigt, ob die gesendeten Informationen empfangen werden können. Der Empfangsvorgang wird fünfmal wiederholt. Um den zeitlichen Einfluss zu berücksichtigen wird der letzte Empfangsvorgang am Folgetag durchgeführt.

Konfiguration von iBeacon und Eddystone-Beacons

1. **B1** (Eddystone-UID)
   Namespace: 1a2b3c4daffeaffeaffe
   Instance ID: cafecafe1001

2. **B2** (Eddystone-EID)
   Namespace: 81e38e47bbfb99869cd8
   Instance ID: 11527dde5458

3. **B3** (iBeacon)
   UUID: 1A2B3C4D-AFFE-AFFE-AFFE-CAFECAFE0000
   Major: 1000
   Minor: 3

Informationen zum Roncalliplatz

Der Roncalliplatz in Köln ist eine hochfrequentierte Fußgängerplattform, die einen Teilbereich des Vorplatzes des Kölner Doms abdeckt. Dort halten sich im Wesentlichen Besucher des Doms, Touristen oder Passanten auf, die den Platz als Fußgängerweg nutzen.

Der Roncalliplatz gehört zu einem der Standorte in Köln, in denen Wi-Fi-Hotspots den kostenfreien Zugang zum Netzwerk „hotspot.koeln“ ermöglichen (s. Abbildung 6.1) [8].

Abbildung 6.1.: Hinweis zum Wi-Fi-Hotspot auf dem Roncalliplatz in Köln [8]

Informationen zum Café Starbucks am Hauptbahnhof

Das Café Starbucks [1] befindet sich im Gebäude des Kölner Hauptbahnhof und grenzt an den Bahnhofsvorplatz, von dem aus der Kölner Dom zu sehen ist. Hauptsächlich Touristen und Reisende überqueren den Bahnhofsvorplatz. Das Café kann als Durchgang zum Bahnhofsgebäude genutzt werden.

Abbildung 6.2.: Ansicht auf den Starbucks vom Vorplatz des Hauptbahnhofs in Köln

[1] Starbucks ist eine Einzelhandelskette, die auf Kaffeeprodukte spezialisiert ist, https://starbucks.de (aufgerufen am: 25.06.2018)

Messpunkte

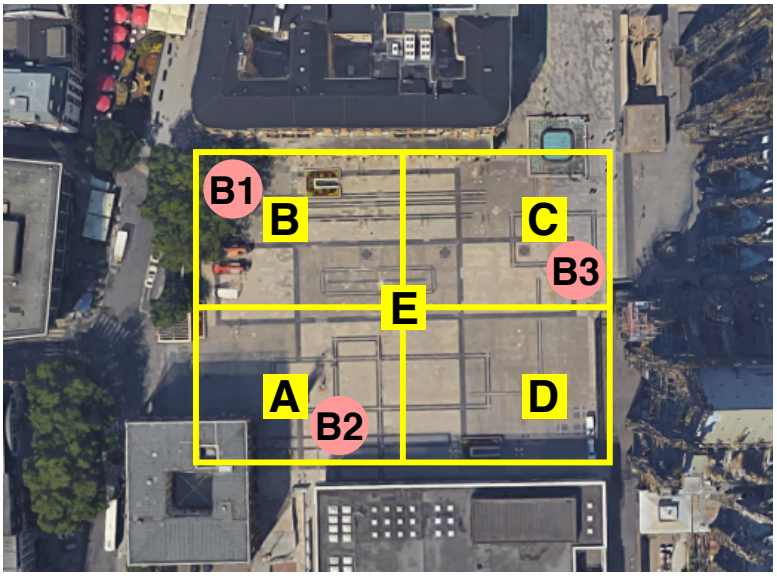

Abbildung 6.3.: Messpunkte auf dem Roncalliplatz in Köln

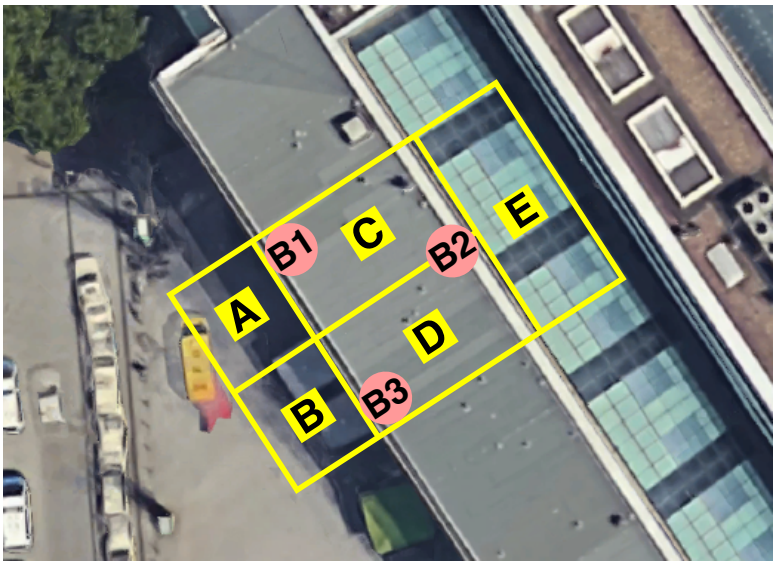

Abbildung 6.4.: Messpunkte im Starbucks am Hauptbahnhof in Köln

6.1.4. Durchführung

Die festgelegten Messpunkte in 6.3 und 6.4 wurden mit einem Smartphone (Hersteller: HTC, Android Version: 5.0.2) abgelaufen. An jedem der Messpunkte wurde die bestehende WLAN-Verbindung beendet und neu aktiviert. Das Smartphone wählte sich daraufhin neu in das bekannte Netzwerk „hotspot.koeln“ ein. Die SSID und BSSID mit der MAC-Adresse des Accesspoints wurden dokumentiert. Gleichzeitig wurden alle empfangenen Bluetooth-Signale der installierten iBeacons und Eddystone-Beacons protokolliert.

6.1.5. Ergebnisse

Im folgenden Abschnitt werden die empfangenen Signale beschrieben und die Messungen tabellarisch aufgeführt. Die fettgedruckten Einträge in den Tabellen zeigen, dass die empfangene ID dieselbe ist, die für das Senden genutzt wurde und damit die Bedingungen für einen erfolgreichen Informationsaustausch bestehen.

Signalinformationen von Wi-Fi-Accesspoints am Roncalliplatz

Erfasste Wi-Fi-Signalquellen von hotspot.koeln

1. SSID: hotspot.koeln, BSSID: [...]:18
2. SSID: hotspot.koeln, BSSID: [...]:1c
3. SSID: hotspot.koeln, BSSID: [...]:c8
4. SSID: hotspot.koeln, BSSID: [...]:cc
5. SSID: hotspot.koeln, BSSID: [...]:38
6. SSID: hotspot.koeln, BSSID: [...]:3c
7. SSID: hotspot.koeln, BSSID: [...]:68
8. SSID: hotspot.koeln, BSSID: [...]:6c
9. SSID: hotspot.koeln, BSSID: [...]:48

Messung von Wi-Fi-Signalquellen am Roncalliplatz

Messpunkt	A	B	C	D	E
15.06.2018, Senden	:38	:1c	:1c	:18	:c8
15.06.2018, 1. Empfang	**:38**	**:1c**	**:1c**	:cc	:1c
15.06.2018, 2. Empfang	:cc	**:1c**	:18	:c8	**:c8**
15.06.2018, 3. Empfang	:cc	**:1c**	**:1c**	:c8	:cc
15.06.2018, 4. Empfang	**:38**	:18	:18	:1c	:1c
16.06.2018, 5. Empfang	:cc	:18	:18	:cc	:1c
Empfangsrate	40 %	60 %	40 %	0 %	20 %

Tabelle 6.1.: Senden und Empfangen von Wi-Fi-Signalinformationen am Roncalliplatz

Mit der Nutzung der Signalinformationen von einzelnen Accesspoints eines Wi-Fi-Hotspots konnte lediglich eine durchschnittliche Empfangsrate von 32 % erreicht werden. In jedem Bereich hat sich der Empfänger in den Messungen mindestens zweimal mit einem anderen Accesspoint verbunden, so dass keine zuverlässige Informationsübertragung stattfinden konnte.

Messung von iBeacons und Eddystone-Beacons am Roncalliplatz

Messpunkt	A	B	C	D	E
15.06.2018, Senden	B1	B2	B3	B1	B3
15.06.2018, 1. Empfang	**B1**, B2, B3	B1, **B2**, B3	B1, B2, **B3**	**B1**, B2, B3	B1, B2, **B3**
15.06.2018, 2. Empfang	**B1**, B2, B3	B1, **B2**, B3	B1, B2, **B3**	**B1**, B2, B3	B1, B2, **B3**
15.06.2018, 3. Empfang	**B1**, B2, B3	B1, **B2**, B3	B1, B2, **B3**	**B1**, B2, B3	B1, B2, **B3**
15.06.2018, 4. Empfang	**B1**, B2, B3	B1, **B2**, B3	B1, B2, **B3**	**B1**, B2, B3	B1, B2, **B3**
16.06.2018, 5. Empfang	**B1**, B?, B3	B1, B?, B3	B1, B?, **B3**	**B1**, B?, B3	B1, B?, **B3**
Empfangsrate	100 %	80 %	100 %	100 %	100 %

Tabelle 6.2.: Senden und Empfangen von iBeacons und Eddystone-Beacons am Roncalliplatz

Am ersten Tag der Messung wurden bei jeder Messung in allen Bereichen die Signalinformationen aller Beacons empfangen. Die Bedingungen für eine Informationsübertragung mit Hilfe des Verzeichnisdienstes waren am 15.06.2018 zu jedem Zeitpunkt erfüllt. Am Folgetag wurde der Eddystone-Beacon B2 weiterhin erfasst, jedoch ging ein anderer Namespace und eine andere Instance-ID von ihm aus, wodurch die mit der vorherigen ID verknüpften Informationen nicht mehr abgerufen werden konnten.

Signalinformationen von Wi-Fi-Accesspoints im Starbucks

Erfasste Wi-Fi-Signalquellen von Starbucks:

1. SSID: MEET ME @ STARBUCKS, BSSID: e2:55:**6d**[...]
2. SSID: MEET ME @ STARBUCKS, BSSID: e2:55:**7d**[...]

Messung von Wi-Fi-Signalquellen im Starbucks

Messpunkt	**A**	**B**	**C**	**D**	**E**
15.06.2018, Senden	:7d	:6d	:6d	:7d	:6d
15.06.2018, 1. Empfang	:6d	**:6d**	**:6d**	:6d	**:6d**
15.06.2018, 2. Empfang	:6d	:7d	**:6d**	:6d	**:6d**
15.06.2018, 3. Empfang	**:7d**	:7d	**:6d**	**:7d**	**:6d**
15.06.2018, 4. Empfang	**:7d**	:7d	**:6d**	:6d	**:6d**
16.06.2018, 5. Empfang	:6d	**:6d**	**:6d**	**:7d**	**:6d**
Emfangsrate	40 %	40 %	100 %	40 %	100 %

Tabelle 6.3.: Senden und Empfangen von Wi-Fi-Signalinformationen im Starbucks

Mit der Nutzung der Signalinformationen von einzelnen Accesspoints eines Wi-Fi-Hotspots konnte lediglich eine durchschnittliche Empfangsrate von 64 % erreicht werden. Nur in den Bereichen C und E wurden in jeder Messung die Bedingungen für eine Informationsübertragung erfüllt.

Messung von iBeacon und Eddystone-Beacons im Starbucks

Messpunkt	**A**	**B**	**C**	**D**	**E**
15.06.2018, Senden	B3	B2	B3	B1	B2
15.06.2018, 1. Empfang	B1, B2, **B3**	B1, **B2**, B3	B1, B2, **B3**	**B1**, B2, B3	B1, **B2**, B3
15.06.2018, 2. Empfang	B1, B2, **B3**	B1, **B2**, B3	B1, B2, **B3**	**B1**, B2, B3	B1, **B2**, B3
15.06.2018, 3. Empfang	B1, B2, **B3**	B1, **B2**, B3	B1, B2, **B3**	**B1**, B2, B3	B1, **B2**, B3
15.06.2018, 4. Empfang	B1, B2, **B3**	B1, **B2**, B3	B1, B2, **B3**	**B1**, B2, B3	B1, **B2**, B3
16.06.2018, 5. Empfang	B1, B?, **B3**	B1, B?, B3	B1, B?, **B3**	**B1**, B?, B3	B1, B?, B3
Empfangsrate	100 %	80 %	100 %	100 %	80 %

Tabelle 6.4.: Senden und Empfangen von iBeacons und Eddystone-Beacons im Starbucks

Die Bedingungen für eine Informationsübertragung mit Hilfe des Verzeichnisdienstes waren am 15.06.2018 zu jedem Zeitpunkt erfüllt. Am 16.06.2018 wurde die am Vortag empfangene Eddystone-EID des Eddystone-Beacon B2 nicht mehr empfangen.

6.1.6. Bewertung

Das Experiment hat gezeigt, dass sich die MAC-Adresse eines Wi-Fi-Accesspoints als Schlüssel zum Informationsaustausch über das Verzeichnis nur dann für eine zuverlässige Informationsübertragung nutzen lässt, wenn nicht mehr als ein Accesspoint für den Wi-Fi-Hotspot zur Verfügung steht. Ein Nutzer müsste darüber informiert werden, welche Wi-Fi-Hotspots in Frage kommen oder sich mit den Empfängern auf einen Accesspoint einigen. Beide Szenarien sind aufgrund des erhöhten Aufwandes nicht praktikabel. Außerdem bietet die von iOS bereitgestellte Schnittstelle keinen Zugriff auf die MAC-Adressen von nicht verbundenen Wi-Fi-Accesspoints oder Bluetooth-Geräten, was den signalbasierten Austausch zusätzlich limitiert. Bei der Verwendung der MAC-Adresse als Informationsschlüssel muss berücksichtigt werden, dass die Information auf den Senderadius des Geräts beschränkt ist.

Auch wenn die Nutzbarkeit von MAC-Adressen grundsätzlich möglich ist, kann sie für öffentliche Standorte aufgrund der ungewissen Anzahl von Accesspoints und dem eingeschränkten Zugriff auf MAC-Adressen durch iOS weitestgehend ausgeschlossen werden.

Die Informationsübertragung über die Nutzung von Bluetooth-Beacons, die im iBeacon oder Eddystone-Format statische IDs aussenden (B1, B3), zeigten im Ergebnis über zwei aufeinanderfolgende Tage eine durchschnittlichen Empfangsrate von 100 % und damit eine hohe Zuverlässigkeit. Die Signalinformationen des Eddystone-Beacons B2, der mit einer Eddystone-EID konfiguriert wurde, war dagegen am zweiten Tag nicht mehr verfügbar.

Bei dem Experiment wurde deutlich, dass der Sender einer Information bei der Wahl eines Eddystone-Beacons nicht wissen kann, ob es sich bei der empfangenen ID um eine statische Eddystone-UID oder Eddystone-EID handelt. Beim Empfang werden beide Varianten als Eddystone-UID interpretiert. Der Zeitraum in der sich eine Eddystone-EID ändert, kann nicht vorausgesagt werden. Damit kann auch die Nutzung von Eddystone-Beacons als Medium zum Austausch von Informationen ausgeschlossen werden.

Da die Informationsübertragung über die statische UUID des iBeacons B3 zu jedem Zeitpunkt erfolgreich war, kann davon ausgegangen werden, dass iBeacons für den Informationsaustausch verwendet werden können.

6.2. Übersicht zur Eignung von Signalinformation zum Informationsaustausch

In der folgenden Tabelle werden die Eignungen der untersuchten Signaltypen zum signalbasierten Informationsaustausch begründet. Die Einschätzung zur Eignung berücksichtigt die Möglichkeiten der Betriebssysteme iOS und Android.

Typ	ID	Eignung	Begründung
WLAN	MAC	nicht geeignet	Über iOS kann im Gegensatz zu Android nur die MAC-Adresse der aktuellen Verbindung ausgelesen werden. Andere iOS-Geräte können im gleichen Netzwerk mit anderen Accesspoints verbunden sein und eine andere MAC-Adresse empfangen.
Bluetooth	MAC	nicht geeignet	Über iOS kann im Gegensatz zu Android nur die MAC-Adresse der aktuellen Verbindung ausgelesen werden.
Eddystone	UID	nicht geeignet	Die statischen Eigenschaften der empfangenen Eddystone-UID können nicht garantiert werden. Ein Empfänger kann nicht zwischen der Eddystone-UID und der regelmäßig angepassten Eddystone-EID unterscheiden.
iBeacon	UUID	geeignet	Die statischen Eigenschaften der empfangenen UUID können garantiert werden, solange der Administrator diese nicht aktiv ändert.

Tabelle 6.5.: Übersicht zur Eignung von Signalinformation zum Informationsaustausch

Die Möglichkeit die Signalquellen aus der vorhandenen Signalinfrastruktur auszunutzen, beschränkt sich auf die Verwendung von UUIDs, die z. B. durch iBeacons ausgesendet werden.

Eine Alternative, die jedoch mindestens eine einmalige Konfiguration des Beacon-Administrators notwendig macht, ist die Versendung einer generierten, kontextgebundenen ID über Attachments mit Hilfe der Google Proximity Beacon API (s. Abschnitt 3.4.3). Diese können sowohl von Android- als auch von iOS-Geräten empfangen werden. Nachteilig ist dabei die Abhängigkeit von der Nearby Messaging API (s. Abschnitt 3.4.2), die für den Abruf der Attachments benötigt wird.

7. Fazit

Die Arbeit hat gezeigt, dass ein Informationsaustausch auf Basis der verfügbaren Signalinfrastruktur aus Wi-Fi-Hotspots und Bluetooth-Signalen über einen Verzeichnisdienst grundsätzlich möglich, jedoch nur über die Verwendung von Signalinformationen von iBeacons oder Attachments praktikabel ist.

Insbesondere die Einschränkungen durch das Betriebssystem iOS auf den Zugang zu den MAC-Adressen von empfangenen Signalquellen, verhindert die Entwicklung einer Anwendung, welche die Nutzung von MAC-Adressen ermöglicht.

Bluetooth-Beacons, die das iBeacon oder Eddystone-Format unterstützen, versenden IDs, die im Gegensatz zu MAC-Adressen, den Einsatzzweck des Beacons repräsentieren. Administratoren von Bluetooth-Beacons können mehrere Signalquellen mit der gleichen eindeutigen ID konfigurieren. Damit kann eine Information über mehrere Signalquellen an unterschiedlichen Standorten bereitgestellt werden. Für die Bereitstellung und den Empfang von Informationen bedeutet das jedoch, dass Nutzer nicht einschätzen können, in welchen Bereichen sich Sender und Empfänger befinden.

Eine Voraussetzung für den Informationsaustausch über Beacons sind statische IDs. Im Eddystone-Format können Beacon-Administratoren statische IDs vermeiden, indem als Alternative zur Eddystone-UID die Eddystone-EID (s. Abschnitt 3.4.1) verwendet wird. Die Eddystone-EID ändert sich in einem konfigurierbaren Zeitraum und lässt sich nur mit entsprechender Berechtigung über einen Dienst wie die Google Beacon Plattform auflösen. Da der Empfänger nicht zwischen Eddystone-UID und Eddystone-EID unterscheiden kann, ist die Verwendung der Eddystone-UID als Informationsschlüssel ebenfalls nicht sinnvoll.

Eine signalbasierte Alternative, die jedoch die Anforderung an die Nutzung vorhandener Infrastruktur nicht erfüllt, ist die Versendung einer generierten, kontextgebundenen ID über Attachments durch den Beacon-Administrator mit Hilfe der Google Proximity Beacon API (s. Abschnitt 3.4.3). Diese ID kann sowohl von Android- als auch von iOS-Geräten mit Hilfe der Google Nearby Messaging API (s. Abschnitt 3.4.2) empfangen werden.

Unabhängig von der Art der Signalinformation ist der Dienst aufgrund der Ungewissheit über die Lebensdauer einer Signalquelle oder ID-Konfiguration nicht für eine andauernd

zuverlässige und kritische Informationsübermittlung geeignet. Die Nutzung des Dienstes setzt voraus, dass die Vergänglichkeit der Information eine akzeptierte Eigenschaft darstellt.

7.0.1. Ausblick

Diese Arbeit zeigt, dass ein signalbasierter Austausch von Informationen seitens der Nutzer möglich ist und schließt die Verwendung von MAC-Adressen von Eddystone-Beacons aus. Die alternative Lösung, Attachments mit entsprechenden IDs zu nutzen, ist abhängig von der Mitarbeit der Administratoren. Jedoch lassen Projekte, wie das *Amsterdam Open Beacon Network* [1] vermuten, dass die Motivation, sich mit der Administration von Beacons zu beschäftigen, aufgrund von verbesserten, benutzerfreundlichen Schnittstellen gestiegen ist oder steigen wird.

Zukünftige Arbeiten könnten daher auf der Grundlage der Erkenntnis über die Nutzbarkeit von UUIDs oder Attachments, erforschen wie der nutzerseitige Austausch verbessert werden kann. Andere, nicht berücksichtigte Signalinformationen könnten auf ihre Eignung zum kontextgebundenen Informationsaustausch untersucht werden.

Eine Ergänzung zu dieser Arbeit würde die Konzeption einer mobilen Anwendung oder eines Anwendungsservers für die Nutzung des Verzeichnisdienstes darstellen. Dabei könnte auch die in dieser Arbeit nicht berücksichtigte Kontrolle von Zugriffsberechtigungen entwickelt werden. Auch der Umgang mit sensiblen Daten ist ein Thema, das in dieser Arbeit nicht bearbeitet werden konnte.

[1] Amsterdam Open Beacon Projekt, https://amsterdamsmartcity.com/projects/amsterdam-open-beacon-network (aufgerufen am: 06.07.2018)

A. Appendix

A.1. Programmcode des Verzeichnisdientes

A.1.1. Programmcode zur Initialisierung

```
const express         = require('express');
const bodyParser      = require('body-parser')
const assert          = require('assert');
const cassandra       = require('cassandra-driver');
```

Listing A.1: Programmcode: Installation verwendeter Bibliotheken

```
var app = express();
app.use(bodyParser.urlencoded({ extended: true }))
app.use(bodyParser.json())

var server = app.listen(8080, function () {
  var port = server.address().port;
  console.log("Service listening at port %s", port);
})
```

Listing A.2: Programmcode: Konfiguration des Servers

```
"CREATE KEYSPACE IF NOT EXISTS directory WITH REPLICATION = { " +
"'class'  : 'SimpleStrategy', " +
"'replication_factor' : 1 }";
```

Listing A.3: Programmcode: Definition des Namensraums

```
const create_table_signalcontext =
  "CREATE TABLE IF NOT EXISTS directory.signalcontext (" +
  "type text, " +
  "id text, " +
  "name text, " +
  "layer text, " +
```

```
  "owner text, " +
  "entrymap map<text, text>, " +
  "ts_write timestamp, " +
  "PRIMARY KEY ((type, id), layer, owner)) " +
  "WITH CLUSTERING ORDER BY (layer DESC)";
```

Listing A.4: Programmcode: Definition der Tabelle

```
var client = new cassandra.Client(
  {contactPoints: ['127.0.0.1:9042']}
);

// connect to database
client.connect(function (err) {
  assert.ifError(err);
});
// create keyspace (if not exists)
client.execute(create_keyspace_directory, function (err){
  assert.ifError(err);
});
// use keyspace
client.execute("USE directory", function (err){
  assert.ifError(err);
});
// create table (if not exists)
client.execute(create_table_signalcontext, function (err, result) {
    assert.ifError(err);
});
```

Listing A.5: Programmcode: Erstellung des Keyspace und der Tabelle

A.1.2. Programmcode zur Bearbeitung von API-Anfragen

```
// get all owners and entries for id in given layer
app.get('/types/:type/ids/:id/layers/:layer',
function (request, response) {
  var type    = request.params.type;
  var id      = request.params.id;
  var layer   = request.params.layer;

  const query =
    "SELECT owner, entrymap "+
```

```
    "FROM directory.signalcontext "+
    "WHERE type = ? AND id = ? AND layer = ?";
  client.execute(query, [type, id, layer], { prepare: true },
    function (err, result) {
    if (err) {
        response.writeHead(204,
          {"Content-Type": "application/json"}
        );
        response.end(JSON.stringify({"error": err}));
    } else {
        var ownerDict = {};
        result.rows.forEach(function(row){
          ownerDict[row.owner]=row.entrymap;
        });
        response.writeHead(200,
          {"Content-Type": "application/json"}
        );
        response.end(JSON.stringify(ownerDict));
    }
  });
});
```

Listing A.6: Programmcode: Bearbeitung der Anfrage nach Einträgen aller Eigentümer in einer Ebene

```
// get entries from multiple owners for id in given layer
app.put('/types/:type/ids/:id/layers/:layer/owners',
function (request, response) {
  var type     = request.params.type;
  var id       = request.params.id;
  var layer    = request.params.layer;
  var ownerSet = request.body.owners;

  var ownerEntryDict = {};
  const query =
  "SELECT owner, entrymap " +
  "FROM directory.signalcontext "+
  "WHERE type = ? AND id = ? AND layer = ? AND owner = ?";

  if(ownerSet.length == 0){
    response.writeHead(200,
```

```
      {"Content-Type": "application/json"}
    );
    response.end(JSON.stringify(ownerEntryDict));
  }else{
    var count = 0;
    ownerSet.forEach(function(owner){
      client.execute(query, [type, id, layer, owner],
        { prepare: true }, function (err, result) {
        if (!err) {
          if(result.rows.length > 0){
            ownerEntryDict[result.rows[0].owner] =
              result.rows[0].entrymap;
          }
        }
        count++;
        if(count == ownerSet.length){
          response.writeHead(200,
            {"Content-Type": "application/json"}
          );
          response.end(JSON.stringify(ownerEntryDict));
        }
      });
    });
  }
});
```

Listing A.7: Programmcode: Bearbeitung der Anfrage nach Einträgen bestimmter Eigentümer in einer Ebene

```
// add entry for owner in certain layer
app.post('/types/:type/ids/:id/layers/:layer/owners/:owner/entries/:entryId',
function (request, response) {
  var type        = request.params.type;
  var id          = request.params.id;
  var layer       = request.params.layer;
  var owner       = request.params.owner;
  var entryId     = request.params.entryId;
  var entry       = request.body.entry;
  var name        = request.body.name;

  var input       = entryId+"':'"+entry;
```

```
  var currentTime = new Date().getTime();
  const query =
    "UPDATE directory.signalcontext " +
    "SET entrymap = entrymap + {'"+input+"'}, " +
    "name = '"+name+"', ts_write = "+currentTime+" " +
    "WHERE type = ? AND id = ? AND layer = ? AND owner = ?";
  client.execute(query, [type, id, layer, owner],
    { prepare: true }, function (err) {
  if (err) {
      response.writeHead(500, {"Content-Type": "application/json"});
      response.end(JSON.stringify({"error": err}));
    } else {
      response.writeHead(200, {"Content-Type": "application/json"});
      response.end();
    }
  });
});
```

Listing A.8: Programmcode: Bearbeitung der Anfrage zum Hinzufügen eines Eintrages von einem Eigentümer in einer Ebene

```
// remove entry from owner in certain layer
app.delete('/types/:type/ids/:id/layers/:layer/owners/:owner/entries/:entryId',
function (request, response) {
  var type        = request.params.type;
  var id          = request.params.id;
  var layer       = request.params.layer;
  var owner       = request.params.owner;
  var entryId     = request.params.entryId;

  const query =
    "UPDATE directory.signalcontext " +
    "SET entrymap = entrymap - {'"+entryId+"'} " +
    "WHERE type = ? AND id = ? AND layer = ? AND owner = ?";
  client.execute(query, [type, id, layer, owner],
    { prepare: true }, function (err) {
    if (err) {
      response.writeHead(204, {"Content-Type": "application/json"});
      response.end(JSON.stringify({"error": err}));
    } else {
      response.writeHead(200, {"Content-Type": "application/json"});
```

```
      response.end();

      // manually remove owner without entries
      // cassandra removes whole database entry
      // automatically if no other owner exists!
      const query =
        "SELECT owner, entrymap " +
        "FROM directory.signalcontext " +
        "WHERE type = ? AND id = ? AND layer = ? AND owner = ?";
      client.execute(query, [type, id, layer, owner],
        { prepare: true }, function (err, result) {
        if (err) {
          assert.ifError(err);
        } else {
          if(result["rows"][0]["entrymap"] == null){
            const query =
              "DELETE FROM directory.signalcontext " +
              "WHERE type = ? AND id = ? AND layer = ? AND owner = ?";
            client.execute(query, [type, id, layer, owner],
              { prepare: true }, function (err) {
              assert.ifError(err);
            });
          }
        }
      });
    }
  });
});
```

Listing A.9: Programmcode: Bearbeitung der Anfrage zum Entfernen eines Eintrages von einem Eigentümer in einer Ebene

```
// delete owner in certain layer
app.delete('/types/:type/ids/:id/layers/:layer/owners/:owner',
function (request, response) {
  var type     = request.params.type;
  var id       = request.params.id;
  var layer    = request.params.layer;
  var owner    = request.params.owner;

  const query =
```

```
    "DELETE FROM directory.signalcontext " +
    "WHERE type = ? AND id = ? AND layer = ? AND owner = ?";
  client.execute(query, [type, id, layer, owner],
    { prepare: true }, function (err) {
    if (err) {
        response.writeHead(204, {"Content-Type": "application/json"});
        response.end(JSON.stringify({"error": err}));
    } else {
        response.writeHead(200, {"Content-Type": "application/json"});
        response.end();
    }
  });
});
```

Listing A.10: Programmcode: Bearbeitung der Anfrage zum Entfernen eines Eigentümer in einer Ebene

```
// delete certain layer
app.delete('/types/:type/ids/:id/layers/:layer',
function (request, response) {
  var type     = request.params.type;
  var id       = request.params.id;
  var layer    = request.params.layer;

  const query =
    "DELETE FROM directory.signalcontext " +
    "WHERE type = ? AND id = ? AND layer = ?";
  client.execute(query, [type, id, layer],
    { prepare: true }, function (err) {
    if (err) {
        response.writeHead(204, {"Content-Type": "application/json" });
        response.end(JSON.stringify({"error": err}));
    } else {
        response.writeHead(200, {"Content-Type": "application/json"});
        response.end();
    }
  });
});
```

Listing A.11: Programmcode: Bearbeitung der Anfrage zum Entfernen einer Ebene

```
// get name of id set by owner in layer
app.get('/types/:type/ids/:id/layers/:layer/owners/:owner/name',
function (request, response) {
  var type     = request.params.type;
  var id       = request.params.id;
  var layer    = request.params.layer;
  var owner    = request.params.owner;

  const query =
    "SELECT name FROM directory.signalcontext " +
    "WHERE type = ? AND id = ? AND layer = ? AND owner = ?";
  client.execute(query, [type, id, layer, owner],
    { prepare: true }, function (err, result) {
    if (!err && result.rows.length > 0) {
      response.writeHead(200, {"Content-Type": "application/json"});
      response.end(JSON.stringify({"name": result.rows[0].name}));
    } else {
      response.writeHead(500, {"Content-Type": "application/json"});
      response.end(JSON.stringify({"error": err}));
    }
  });
});
```

Listing A.12: Programmcode: Bearbeitung der Anfrage zum Abruf einer vom Eigentümer in einer Ebene gesetzten Bezeichnung

```
// update name of id
app.post('/types/:type/ids/:id/layers/:layer/owners/:owner',
function (request, response) {
  var type     = request.params.type;
  var id       = request.params.id;
  var layer    = request.params.layer;
  var owner    = request.params.owner;

  var currentTime = new Date().getTime();
  var query =
    "UPDATE directory.signalcontext " +
    "SET name = '"+name+"', ts_write = "+currentTime+" " +
    "WHERE type = ? AND id = ? AND layer = ? AND owner = ?";
  client.execute(query, [type, id, layer, owner],
    { prepare: true }, function (err) {
```

```
16      if (err) {
17        response.writeHead(500, {"Content-Type": "application/json"});
18        response.end(JSON.stringify({"error": err}));
19      } else {
20        response.writeHead(200, {"Content-Type": "application/json"});
21        response.end();
22      }
23    });
24  });
```

Listing A.13: Programmcode: Bearbeitung der Anfrage zum Aktualisieren einer vom Eigentümer in einer Ebene gesetzten Bezeichnung

A.2. Inhalt der CD

- Dieses Dokument als PDF
- Programmcode des Verzeichnisdientes
- Anleitung zur Installation und Nutzung

Literaturverzeichnis

[1] IETF, H. Andrews, and Ed. JSON Hyper-Schema: A Vocabulary for Hypermedia Annotation of JSON. Internet-Draft. URL: https://tools.ietf.org/html/draft-handrews-json-schema-hyperschema-01 (aufgerufen am: 14.05.2018).

[2] Fernando Miçalli (fermads). Github Projekt: node.js-process-load-balancing. URL: https://github.com/fermads/node.js-process-load-balancing (aufgerufen am: 14.05.2018).

[3] J. Meyer. Lageplan HAW Hamburg (erstellt: 10.03.2011). URL: http://www.mp.haw-hamburg.de/a1-7/Lageplan_BT.pdf (aufgerufen am: 10.04.2018).

[4] Anthony LaMarca, Yatin Chawathe, Sunny Consolvo, Jeffrey Hightower, Ian Smith, James Scott, Tim Sohn, James Howard, Jeff Hughes, Fred Potter, Jason Tabert, Pauline Powledge, Gaetano Borriello, and Bill Schilit. *Place Lab: Device Positioning Using Radio Beacons in the Wild.* Paper, University of California, Intel Research Cambridge, Intel Research Seattle, 2005. URL: https://www.cc.gatech.edu/classes/AY2006/cs6452_spring/readings/week12-placelab.pdf (aufgerufen am: 03.04.2018).

[5] POLS Privacy-Observant Location System. URL: http://pols.sourceforge.net/ (aufgerufen am: 03.04.2018).

[6] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled*, chapter Aggregate Data Models, pages 21–24. Addison-Wesley, Boston, 2013.

[7] Linux Foundation. The Node.js Event Loop, Timers, and process.nextTick(). URL: https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/ (aufgerufen am: 14.05.2018).

[8] NetCologne. HOTSPOT.KOELN Standortkarte. URL: https://www.netcologne.de/ueber-uns/hotspot/standortkarte (aufgerufen am: 02.07.2018).

[9] Bauer, Haber, and Reichardt. *Location Based Services – Große Potenziale für Direktmarketing*, page 37. 2006.

[10] Wolfgang Kocha and Beate Frees. ARD/ZDF-Onlinestudie 2017: Neun von zehn Deutschen online. *Medien und ihr Publikum*, pages 434–446, 2017. URL: http://www.ard-zdf-onlinestudie.de/files/2017/Artikel/917_Koch_Frees.pdf (aufgerufen am: 10.03.2018).

[11] Dan Wang, Zheng Xiang, and Daniel R. Fesenmaier. Smartphone Use in Everyday Life and Travel. *Journal of Travel Research*, 55:52–63, 2016. URL: https://doi.org/10.1177/0047287514535847 (aufgerufen am: 10.03.2018).

[12] Abowd G.D., Dey A.K., Brown P.J., Davies N., Smith M., and Steggles P. Towards a Better Understanding of Context and Context-Awareness. In *Gellersen HW. (eds), Handheld and Ubiquitous Computing HUC 1999, Lecture Notes in Computer Science, vol 1707.* Springer, Berlin, Heidelberg, 1999.

[13] Axel Küpper. *Location-based services: fundamentals and operation*, chapter Satellite positioning, page 162. John Wiley & Sons, Ltd, 2005.

[14] Martin Werner. *Indoor Location-Based Services*, chapter Introduction, pages 1–3. Springer, Cham, 2014. ISBN: 978-3-3191-0699-1.

[15] Martin Sauter. *Wireless Netzwerke für den Nahbereich: Eingebettete Funksysteme: Vergleich von standartisierten und proprietären Verfahren*, chapter Grundlagen, page 58. Springer Vieweg, 2015.

[16] Mozilla Location Service. URL: https://location.services.mozilla.com (aufgerufen am: 21.03.2018).

[17] Georg Schneider, Björn Dreher, and Ole Seidel. *Using GeoFencing as a means to support flexible real time applications for delivery services.* Paper, University of Applied Sciences Trier, 2008. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.582.1744&rep=rep1&type=pdf (aufgerufen am: 23.03.2018)).

[18] T. Kruse Brandao, C. Klug, C. Rabsch, W. Hagenhoff, and F. Gard. *Promitity Solutions.* Bundesverband Digitale Wirtschaft (BVDW) e. V., 2016. URL: https://www.bvdw.org/fileadmin/bvdw/upload/publikationen/mobile/Leitfaden_Proximity_Solutions_2016.pdf (aufgerufen am: 25.03.2018).

[19] Christopher Habel and Christiane von Stutterheim. *Linguistische Arbeiten*, chapter Einleitung: Räumliche Konzepte und sprachliche Strukturen, page 1. Max Niemeyer Verlag, 2000.

[20] Ludwig Wittgenstein. *Tractatus Logico-Philosophicus*, chapter Satz 5.6. Ludwig Wittgenstein, 1921.

[21] Euklid. Die Elemente. In *Buch XI: Raumlehre*. Euklid, 300 v. Chr.

[22] Axel Küpper. *Location-based services: fundamentals and operation*, chapter What is location?, page 18. John Wiley & Sons, Ltd, 2005.

[23] Axel Küpper. *Location-based services: fundamentals and operation*, chapter Satellite positioning, page 19. John Wiley & Sons, Ltd, 2005.

[24] Google Android Developer Plattform. Optimize location for battery. URL: https://developer.android.com/guide/topics/location/battery (aufgerufen am: 04.07.2018).

[25] Stephen Statler. *Beacon Technologies : The Hitchhiker's Guide to the Beacosystem*, chapter GeoFencing: Everything you Need to Know, page 311. Apress, Berkeley, CA, 2016. ISBN: 978-3-3191-0699-1.

[26] Scott Madry. *Global Navigation Satellite Systems and Their Applications. Springer Briefs in Space Development*, chapter Precision and Navigational PNT Systems, pages 27–55. Springer, New York, NY, 2015. ISBN: 978-1-4939-2607-7.

[27] Sean Barbeau. Github Projekt: gpstest. URL: https://github.com/barbeau/gpstest (aufgerufen am: 06.06.2018).

[28] Fabio Dovis. GNSS Interference Threats and Countermeasures. In *Artech House GNSS technology and applications series*, chapter What is interference?, pages 19–21. Artech House, Boston, 2015. ISBN: 978-1-6080-7811-0.

[29] Axel Küpper. *Location-based services: fundamentals and operation*, chapter Fundamentals of positioning, pages 130–143. John Wiley & Sons, Ltd, 2005.

[30] Martin Werner. *Indoor Location-Based Services*, chapter Basic Positioning Techniques, pages 73–89. Springer, Cham, 2014. ISBN: 978-3-3191-0699-1.

[31] Martin Sauter. *Grundkurs Mobile Kommunikationssysteme: LTE-Advanced, UMTS, HSPA, GSM, GPRS, Wireless LAN und Bluetooth*, chapter Die MAC-Schicht, page 325. Springer Vieweg, 2015.

[32] Axel Küpper. *Location-based services: fundamentals and operation*, chapter Indoor positioning, pages 234–239. John Wiley & Sons, Ltd, 2005.

[33] T. Kruse Brandao, C. Klug, C. Rabsch, W. Hagenhoff, and F. Gard. *Promitity Solutions*, page 21. Bundesverband Digitale Wirtschaft (BVDW) e. V., 2016. URL: https://www.bvdw.org/fileadmin/bvdw/upload/publikationen/mobile/Leitfaden_Proximity_Solutions_2016.pdf (aufgerufen am: 25.03.2018).

[34] Google. Android WifiManager. URL: https://developer.android.com/reference/android/net/wifi/WifiManager.html (aufgerufen am: 22.03.2018).

[35] Apple. iOS NEHotspotHelper. URL: https://developer.apple.com/documentation/networkextension/nehotspothelper (aufgerufen am: 23.03.2018).

[36] Apple. iOS Captive Network, CNCopySupportedInterfaces. URL: https://developer.apple.com/documentation/systemconfiguration/1494829-cncopysupportedinterfaces#return-value (aufgerufen am: 22.03.2018).

[37] Google. Proximity Beacon API. URL: https://developers.google.com/beacons/proximity/sharing (aufgerufen am: 18.05.2018).

[38] Google. Nearby Messages API. URL: https://developers.google.com/nearby/messages/overview (aufgerufen am: 18.05.2018).

[39] Google. Nearby Notifications API. URL: https://developers.google.com/nearby/notifications/overview (aufgerufen am: 18.05.2018).

[40] Stefan Brüning, Johannes Zapotoczky, Peter Ibach, and Vladimir Stantchev. *MagicMap*. Paper, University of Applied Sciences Trier, 2007. URL: https://www.researchgate.net/profile/Vladimir_Stantchev/publication/224699304_Cooperative_Positioning_with_MagicMap/links/5673d8b208ae04d9b09be376/

Cooperative-Positioning-with-MagicMap.pdf (aufgerufen am: 03.04.2018).

[41] Stefan Brüning, Johannes Zapotoczky, Peter Ibach, and Vladimir Stantchev. MagicMap. URL: https://sourceforge.net/projects/magicmap/files/magicmap/ (aufgerufen am: 03.04.2018).

[42] Timothy Sohn, William G. Griswold, James Scott, Anthony LaMarca, Yatin Chawathe, and Ian Smith. *Place Lab*. Paper, University of California, Intel Research Cambridge, Intel Research Seattle, 2005. URL: https://www.researchgate.net/publication/228362239_Place_Lab-An_Open_Architecture_for_Location-Based_Computing (aufgerufen am: 03.04.2018).

[43] POLS Privacy-Observant Location System. URL: http://pols.sourceforge.net/software/ (aufgerufen am: 03.04.2018).

[44] Andrew Turner. *Introduction to Neogeography*, page 23. O'Reilly, 2006. ISBN: 978-0-596-52995-6.

[45] Microsoft Research, Jeffrey Hightower, Anthony LaMarca, and Yatin Chawathe. Place Lab: Device Positioning Using Radio Beacons in the Wild, 2005. URL: https://www.youtube.com/watch?reload=9&v=udC6K6mN81M (aufgerufen am: 03.04.2018).

[46] Unwired Labs LocationAPI. URL: https://unwiredlabs.com/docs#geolocation (aufgerufen am: 03.04.2018).

[47] Mozilla Location Service. URL: https://location.services.mozilla.com/downloads (aufgerufen am: 09.04.2018).

[48] Github Projekt: ichnaea (API Dokumentation). URL: https://mozilla.github.io/ichnaea/api/index.html (aufgerufen am: 03.04.2018).

[49] Mozilla Location Service. URL: https://wiki.mozilla.org/CloudServices/Location/Software (aufgerufen am: 09.04.2018).

[50] Google. Google Beacon Platform. URL: https://developers.google.com/beacons/ (aufgerufen am: 18.05.2018).

[51] Google. Github Projekt: eddystone. URL: https://github.com/google/eddystone (aufgerufen am: 18.05.2018).

[52] Google. Nearby API. URL: https://developers.google.com/nearby/ (aufgerufen am: 18.05.2018).

[53] Google. Nearby Conntections API. URL: https://developers.google.com/nearby/connections/overview (aufgerufen am: 18.05.2018).

[54] Google. Github Projekt: physical-web. URL: https://google.github.io/physical-web/ (aufgerufen am: 18.05.2018).

[55] Alexander Schill and Thomas Springer. *Verteilte Systeme*, chapter Systemarchitekturen: Client/Server-Modell, pages 14–15. Springer Vieweg, 2012.

[56] Facebook. Newsroom Stats. URL: https://newsroom.fb.com/company-info/ (aufgerufen am: 08.06.2018).

[57] AllFacebook. Offizielle Facebook Nutzerzahlen für Deutschland (Stand: September 2017). URL: https://allfacebook.de/zahlen_fakten/offiziell-facebook-nutzerzahlen-deutschland (aufgerufen am: 08.06.2018).

[58] Nginx. Choosing a Microservices Deployment Strategy. URL: https://www.nginx.com/blog/deploying-microservices/ (aufgerufen am: 02.07.2018).

[59] Mark Richards. *Microservices vs. Service-Oriented Architecture*, pages 24–28. O'Reilly, 2016. ISBN: 978-1-491-95242-9.

[60] Andreas Meier and Michael Kaufmann. *SQL- NoSQL-Datenbanken*, chapter NoSQL-Datenbanken, pages 224–228. Springer Vieweg, 2016.

[61] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled*, chapter More Details on Data Models, page 28. Addison-Wesley, Boston, 2013.

[62] Eric Evans. *Domain-Driven Design*, chapter Chapter 6: The Life Cycle of a Domain Object, page 125. Addison-Wesley, Boston, 2004. ISBN: 0-321-12521-5.

[63] Andreas Meier and Michael Kaufmann. *SQL- NoSQL-Datenbanken*, chapter Konsistenzsicherung, pages 136–137. Springer Vieweg, 2016.

[64] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled*, chapter Aggregate Data Models, page 19. Addison-Wesley, Boston, 2013.

[65] Andreas Meier and Michael Kaufmann. *SQL- NoSQL-Datenbanken*, chapter NoSQL-Datenbanken, pages 226–228. Springer Vieweg, 2016.

[66] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled*, chapter Column-Family Stores, pages 99–109. Addison-Wesley, Boston, 2013.

[67] Andreas Meier and Michael Kaufmann. *SQL- NoSQL-Datenbanken*, chapter SQL-Datenbanken, pages 9–11. Springer Vieweg, 2016.

[68] The Apache Software Foundation. Apache Cassandra 3.11.2, 2018. URL: http://cassandra.apache.org (aufgerufen am: 11.04.2018).

[69] Dikang Gu. Cassandra at Instagram 2016 (Dikang Gu, Facebook) | Cassandra Summit 2016. Youtube, 2016. URL: https://www.youtube.com/watch?v=_BfMH4GQWnk (aufgerufen am: 14.05.2018).

[70] Reddit. Announcements: Reddit Originals - She who entagles men, 2010. URL: https://redditblog.com/2010/03/12/she-who-entangles-men/ (aufgerufen am: 10.05.2018).

[71] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled*, chapter Distribution Models: Peer-to-Peer Replication, pages 42–43. Addison-Wesley, Boston, 2013.

[72] Andreas Meier and Michael Kaufmann. *SQL- NoSQL-Datenbanken*, chapter Konsistenzsicherung, pages 144–148. Springer Vieweg, 2016.

[73] Andreas Meier and Michael Kaufmann. *SQL- NoSQL-Datenbanken*, chapter Speicher- und Zugriffstrukturen, pages 166–168. Springer Vieweg, 2016.

[74] DataStax. Consistent hashing. URL: https://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archDataDistributeHashing.html (aufgerufen am: 14.05.2018).

[75] Adam Hutson. C* Keys: Partitioning, Clustering, CrossFit (Adam Hutson, DataScale) | Cassandra Summit 2016. Youtube, 2016. URL: https://www.youtube.com/watch?v=L0XqRF8C6D0 (aufgerufen am: 14.05.2018).

[76] DataStax. CQL commands - INSERT. URL: https://docs.datastax.com/en/cql/3.3/cql/cql_reference/cqlInsert.html (aufgerufen am: 14.05.2018).

[77] Alexander Schill and Thomas Springer. *Verteilte Systeme*, chapter Dienstbasierte Architekturen und Technologien, page 266. Springer Vieweg, 2012.

[78] Leonard Richardson and Martin Fowler. Richardson Maturity Model. URL: https://martinfowler.com/articles/richardsonMaturityModel.html#TheMeaningOfTheLevels (aufgerufen am: 14.05.2018).

[79] Linux Foundation. Über Node.js. URL: https://nodejs.org/de/about/ (aufgerufen am: 14.05.2018).

[80] Linux Foundation. Node.js child_process() API. URL: https://nodejs.org/api/child_process.html (aufgerufen am: 14.05.2018).

[81] Linux Foundation. Node.js cluster Modul. URL: https://nodejs.org/api/cluster.html (aufgerufen am: 14.05.2018).

[82] Nginx. Nginx Reverse Proxy Dokumentation. URL: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/ (aufgerufen am: 14.05.2018).

[83] ISO/IEC 9075-1 Framework, 2011. URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/c053681_ISO_IEC_9075-1_2011.zip (aufgerufen am: 14.05.2018).

[84] DataStax. Using CQL. URL: https://docs.datastax.com/en/cql/3.3/cql/cql_using/useAboutCQL.html#useAboutCQL__accessing-cql (aufgerufen am: 14.05.2018).

[85] DataStax. Data replication. URL: https://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archDataDistributeReplication.html (aufgerufen am: 14.05.2018).

Glossar

ACID Atomicity, Consistency, Isolation, Durability. 46, 47

BASE Basically Available, Soft state, Eventual consistency. 46

BSSID Basic Service Set Identifier. 69, 74, 76

CID Cell Identification. 37

CQL Cassandra Query Language. 65

GIS Geographic Information System. 11

GPS Global Positioning System. iii–v, 1, 15, 16, 20

GSM Global System for Mobile Communications. 15, 16, 22, 37

HAL Hardware Abstraction Layer. 23

HATOEAS Hypertext As The Engine Of Application State. 52

IrDA Infrared Data Association. 1

MAC Media Access Control. 19, 23, 25, 43, 44, 49, 74, 77, 79, 80

NFC Near Field Communication. 1

PAN Personal Area Network. 1

REST Restful State Representation. 51, 52

RFID Radio Frequency Identification. 1, 18, 21

SQL Structured Query Language. 46

SSID Service Set Identifier. 69, 74, 76

TLS Transport Layer Security. 28

URL Unique Resource Locator. 28, 52, 53

UUID Universally Unique Identifier. 19, 39, 49, 70, 78

WAN Wide Area Network. 1

WLAN Wireless Local Area Network. iii, 1, 15, 16, 18, 19, 21, 22, 25, 27, 35, 44, 64, 70, 74

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 12. Juli 2018 Sebastian Wilkes