HAW HAMBURG

Bachelorarbeit

Christoph Wessarges

Erstellung einer Middleware zur Abbildung von unterschiedlichen Lokalisierungsdatenquellen auf ein einheitliches Koordinatensystem

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Christoph Wessarges

Erstellung einer Middleware zur Abbildung von unterschiedlichen Lokalisierungsdatenquellen auf ein einheitliches Koordinatensystem

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Wirtschaftsinformatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Ulrike Steffens
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 14.08.2019

Christoph Wessarges

Thema der Arbeit

Erstellung einer Middleware zur Abbildung von unterschiedlichen Lokalisierungsdatenquellen auf ein einheitliches Koordinatensystem

Stichworte

Lokalisierung, Echtzeitsystem, Middleware, Software-Engineering, Java Spring

Kurzzusammenfassung

Lokalisierungstechnologien können als hilfreiche Werkzeuge von Unternehmen eingesetzt werden. Anwendungsbereiche sind dabei beispielsweise Prozess-, Kontroll- und Transportaufgaben. Es existieren jedoch eine Vielzahl an unterschiedlichen Lokalisierungssystemen auf dem Markt, die jeweils anwendungsspezifische Vorteile und Nachteile besitzen. Durch die vielseitige Einsetzbarkeit von Lokalisierungslösungen kann es häufig vorkommen, dass Unternehmen sich dazu entscheiden, Lokalisierung in verschiedene Unternehmensbereiche zu implementieren. Oft gibt es jedoch nicht ein Lokalisierungssystem, welches für divergente Bereiche einheitlich praktikabel ist. Aus diesem Grund wurde im Zuge dieser Arbeit eine Software geschrieben, dessen Ziel die Zusammenführung und Vereinheitlichung der Daten aus diversen Lokalisierungssystemen ist. Der gesamte Software-Engineering-Prozess wurde dargelegt und begründet.

Christoph Wessarges

Title of Thesis

Implementation of a middleware for mapping different localization data sources to a uniform coordinate system

Keywords

Localization, Real-time system, Middleware, Software-Engineering, Java Spring

Abstract

Localization technologies can be used as helpful tools by companies. Applications include process, control and transport tasks. However, there are a number of different localization systems on the market, each with application-specific advantages and disadvantages. Due to the versatility of localization solutions, it can often happen that companies decide to implement localization in different business areas. Often, however, there is not one localization system that is uniformly practicable for different areas. For this reason, a software was written in the course of this work, the aim of which is the consolidation and standardization of data from various localization systems. The entire software engineering process was explained and justified.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Diese Arbeit beschreibt die Entwurfsphase und frühe Umsetzungsphase einer Middleware zur Verarbeitung und Vereinheitlichung von Lokalisierungsdaten unterschiedlicher Systeme in nahezu Echtzeit. Der genaue Umfang der im Zuge dieser Arbeit programmierten Software wird mit all seinen Komponenten abgegrenzt und begründet. Insbesondere werden im Folgenden die Entwurfs- und Architekturentscheidungen erläutert und mithilfe von Diagrammen veranschaulicht. Ein besonderer Fokus bezüglich der ausgewählten Anforderungen lag auf der Erweiterbarkeit, Modularität und Leistungsfähigkeit der Software.

1.1 Problemzusammenhang und Motivation

Echtzeitlokalisierung kann Anwendung in vielen unterschiedlichen Bereichen, verbunden mit verschiedensten Anforderungen, finden. Wichtig ist dabei zu beachten, die Genauigkeit der zeitlichen und räumlichen Lokalisierung des Lokalisierungssystems auf das Anwendungsfeld des Kunden anzupassen, um ein stabiles Kosten-Nutzen-Verhältniss zu gewährleisten. Aufgrund der unterschiedlichen Stärken und Schwächen verschiedener Lokalisierungssysteme und Lokalisierungstechnologien kann es sich für ein Unternehmen häufig lohnen mehrere Lokalisierungssysteme in den jeweiligen Unternehmensbereichen zu installieren. Eine gemeinsame Aggregation aller Systeme wäre, explizit im Kontext von Lokalisierung, ausschließlich von Vorteil.
Moderne, elektronische Kartendienste, wie z.B. Google Maps, erlauben einen sofortigen Wechsel zwischen globalen und lokalen Ansichten durch Zoomen. Daher wäre es wünschenswert, alle lokalisierten Objekte im Unternehmen auf einer solchen gemeinsamen Karte anzeigen zu lassen. Um eine Gesamtansicht und Auswertung auf alle georteten Objekte zu erhalten, müssten die Ergebnisse aller einzelnen Systeme ständig aufwendig zusammengefügt werden.

Dieser iterative Prozess erfordert:

1) Die Beschaffung aller Ortsdaten zu einem gewünschten Zeitpunkt oder -intervall.

2) Die Umwandlung der zumeist unterschiedlichen Ortsdaten auf ein einheitliches Datenformat.

3) Die Auswertung der Daten, ggf. mit vorangehender Persistierung der Daten.

Da einige Lokalisierungssysteme ihre Daten in Echtzeit zur Verfügung stellen, würde durch einen aufwendigen manuellen Aggregationsprozess entweder nur ein Schnappschuss auf alle Ortsdaten verfügbar sein oder eine nachträgliche Rekonstruktion der Ortungen mithilfe einer aufwendigen Historisierung der Daten wäre nötig. Außerdem könnten Auswertungen immer nur langsam und retrospektiv geschehen. Es wäre jedoch von Vorteil, wenn bestimmte Auswertungen in Echtzeit vorlägen.
Aus diesen Problemen entstammt der naheliegende Vorschlag, den genannten iterativen Prozess zu automatisieren und die Echtzeitverarbeitung gleichzeitig zu unterstützen. Der Kunde bekäme dadurch einen einzigen Dienst, an dem er Zustände und Veränderungen seiner Lokalisierungsdaten abrufen und verfolgen kann. Da die Anwendungsmöglichkeiten von Lokalisierung äußerst vielfältig sind, werden anwendungsnahe Nutzerapplikationen benötigt, die die Echtwelt-Abstraktionen und Lösungen umsetzen, die auf die spezifischen Probleme der jeweiligen Unternehmen zugeschnitten sind. Diese Endanwendungen können allesamt von dem Ergebnis dieser Arbeit profitieren, weshalb es sich anbietet diese Software als Middleware zu implementieren.

1.2 Themenabgrenzung

Diese Arbeit erläutert am Beispiel des Entwicklungsprozesses einer speziellen Middleware eine mögliche Herangehensweise ein Softwareprojekt zu planen und durchzuführen. Dabei wird bei der Modellierung und Umsetzung besonders auf bekannte „Best Practices“ eingegangen, die im Zuge der Arbeit verwendet wurden. Der Funktionsumfang wurde iterativ in Zusammenarbeit mit Lufthansa Industry Solutions besprochen und ausgearbeitet.
Die Middleware ist solchermaßen aufgebaut, dass sie mit einer unbestimmten Anzahl an externen Schnittstellen kommuniziert und selbst ein umfangreiches Application Programming Interface bereitstellt. Die Middleware ist folglich ein eigenständiger Service, der keine Integration von oder zu anderen Softwaresystemen voraussetzt. Das bedeutet,

dass die Kopplung zu den Lokalisierungsdatenquellen minimal gehalten wird, um die Modularität bestmöglich zu gewährleisten.
Außerdem soll die Middleware ausschließlich als Hilfsebene für alle erdenklichen Endanwendungen dienen, die auf Lokalisierungslösungen aufbauen. Daher wird eine inhaltliche Kopplung der Middleware mit Funktionalitäten jeglicher Endanwendung ebenfalls vermieden. Die Kernidee war demnach, allen Endanwendungen eine einheitliche Schnittstelle zugreifbar zu machen.

1.3 Zielsetzung

Die Arbeit beschreibt Überlegungen und Ergebnisse des Konzeptions- und Entwicklungsprozesses der Middleware zur Konsolidierung von Lokalisierungsdatenquellen sowie der einheitlichen Bereitstellung ihrer Daten. Die Arbeit verdeutlicht zudem, wie ausgearbeitete Softwareanforderungen in klare Spezifikationen übersetzt, iterativ verfeinert und ergänzt werden können. Außerdem zeigt sie auf, welche Vorgehensweisen und Muster sich in bestimmten Fällen bei der Anfertigung des Entwurfs eignen können. Zusätzlich wird erläutert welche Entwurfsentscheidungen explizit für die Echtzeitsystemunterstützung getroffen werden können, wobei eine geeignete Architekturauswahl begründet und erklärt wird.
Aufgrund der im Folgenden beschriebenen Anforderungen liegt der wesentliche Fokus des Entwurfs und der Implementierung auf drei Faktoren:

1) Die einfachen Erweiterbarkeit der Software um weitere Lokalisierungs-Quellsysteme.
2) Der modulare Aufbau der Komponenten innerhalb der Middleware und deren Schnittstellen nach außen.
3) Die Leistungs- und Reaktionsfähigkeit der Software.

1.4 Struktur der Arbeit

Zunächst gibt Kapitel 2 eine Einleitung in den Bereich der Lokalisierungstechnologien. Zudem werden ausgewählte populäre Konzepte der Intersystemkommunikation vorgestellt, die im Laufe dieser Arbeit verwendet werden.

Anschließend befasst sich Kapitel 3 mit der Analyse der Anforderungen an die zu entwickelnde Software. Sie beschreibt den Mindestfunktionsumfang der fertigen Applikation. Außerdem werden die Anforderungen mittels fachlicher Konzepte wie Geschäftsprozessen, der Einbindung von Nachbarsystemen und einem Datenmodell spezifiziert.
In Kapitel 4 werden die Anforderungen in einen Softwareentwurf gegossen, deren Ziel es ist, einen vollständigen Bauplan der zu realisierenden Applikation zu entwerfen. Im Grobentwurf werden die Architekturentscheidungen erklärt und begründet sowie Lösungen für weitere, allgemeine Aspekte und Probleme auf dem Weg zur Zielerreichung erarbeitet. Im anschließenden Feinentwurf werden die genauen Bausteine und Schnittstellen der Software ausgefeilt.
Das Kapitel 5 beinhaltet die Beschreibung der systematischen Umsetzung des Entwurfs. Das Kapitel fasst nennenswerte Entscheidungen und Besonderheiten der Implementationsphase hinsichtlich der verschiedenen Teilabschnitte der Applikation zusammen.
Schlussendlich werden in Kapitel 6 die Erkenntnisse dieser Arbeit zusammengefasst. Danach wird ein Fazit über die verwendeten Methoden und die gesamte Vorgehensweise gezogen und abschließend ein Ausblick auf die Zukunft der entwickelten Software gegeben.

2 Grundlagen

In diesem Kapitel wird eine Wissensgrundlage für den Themenbereich dieser Arbeit aufgebaut. Außerdem werden Konzepte für das Verständnis der Architektur und Funktionsweise der Software vermittelt.

2.1 Lokalisierungstechnologien

Ein Lokalisierungssystem besteht aus einer Menge von Elementen, die den Zweck der Lokalisierung eines Objektes verfolgen. Lokalisiert wird ein Gegenstand, indem dessen Position im Raum sowie gegebenenfalls dessen Orientierung bestimmt wird. Typischerweise erfolgen die Angaben in drei Dimensionen mit Bezug auf ein kartesisches Koordinatensystem. Ein Echtzeit-Lokalisierungssystem (Real-Time Locating System, Abk. RTLS) ist eine spezifische Ausprägung eines Lokalisierungssystems. RTLS sind Systeme, die auf der Basis von Funkverbindungen kurzer Reichweite Positionsdaten innerhalb einer kurzen Latenzzeit übermitteln.

Die technische Vorrichtung eines Lokalisierungssystems besteht allgemein aus vier funktionalen Modulen [Bansky, 2008]. Der elektronische „Tag“ kennzeichnet das zu lokalisierende Objekt, deren gesendete Signale am „Location Sensor“ erfasst und gemessen werden. Der Location Sensor übermittelt das Signal an die „Location Engine“, die letztendlich die Messgrößen des gemessenen Signals zu einer Position auswertet. Die „Location Applications“ nutzen die ermittelten Positionsinformationen, um diese an übergeordnete Unternehmenssysteme (z.B. ERP) zu übermitteln oder sie verfügbar bzw. abfragbar zu machen.

Für die Kommunikation der Signale zwischen den Tags und den Location Sensors existieren unterschiedliche Ortungstechniken, wie die Satelliten-, Mobilfunk- oder WLAN-Technik, die wiederum in GPS (global positioning system) und LPS (local positioning

system) klassifiziert werden können [ITWissen, 2017]. LPS ist eine Technologie zum Erhalten der Standortinformationen von Objekten in Bezug auf ein lokales Gebiet. GPS basiert auf Satellitentechnik und erlaubt eine präzise Ortung auf dem gesamten Globus. Anhand der globalen Ortung eines ganzen lokalen Gebiets mittels GPS können die Standortinformationen aus dem LPS allerdings ebenfalls in ein globales Referenzsystem übertragen werden. Z.B. erlaubt die Kombination aus GPS und Mobilfunktechnik (GSM) ebenfalls eine präzise globale Lokalisierung des Mobilfunkgerätes.

Bei der Indoor-Ortung mit RTLS wird überwiegend Radiofrequenztechnologie, aber zunehmend auch optische oder akustische Technologie für die Kommunikation verwendet. Die RTLS-Systeme mit Radiofrequenztechnologie richten sich nach bekannten Standards wie z.B. Radio-Frequency Identification (RFID), Ultra Wide Band (UWB) oder Bluetooth Low Energy (BLE), die sich in ihrer Signalreichweite und Lokalisierungsgenauigkeit unterscheiden. BLE ist aufgrund der Verwendung von Bluetooth kompatibel mit Mobiltelefonen und die Tags besitzen, wegen des geringen Stromverbrauchs, eine lange Batterielebenszeit.

Die Auswertung der Signale in der Location Engine kann durch eine Vielzahl an Positionsbestimmungsverfahren durchgeführt werden [Malik, 2009]. Beispielsweise kann bei der Methode „Angle of Arrival“ (AoA) jeweils aus den eintreffenden Winkeln eines Tag-Signals an mehreren Location Sensors die Position des Tags zentimetergenau ermittelt werden.

Der Schwerpunkt der zu erstellende Software liegt auf der Verarbeitung von Echtzeitlokalisierungsdaten. Dennoch wird versucht jegliche Art von Lokalisierungssystem potenziell integrierbar zu machen.

2.2 Middleware

Analyst und Systemtheoretiker Nick Gall sagte, „Middleware ist Software für Software.“ Middleware ist Software zwischen einem Betriebssystem und den Anwendungen, die darauf ausgeführt werden. Im Prinzip fungiert Middleware als versteckte Übersetzungsebene. Es ist ein Oberbegriff für Software, die dazu dient, die Kommunikation und Datenverwaltung verteilter Anwendungen zu ermöglichen [RedHat, 2019]. Die Kernaufgabe von Middleware besteht daraus einzelne, oft komplexe und bereits existierende Programme zusammenzufügen. Der Name Middleware ergibt sich daher aus der Tatsache, dass die

Software zwischen der Seite des Klienten im Frontend und der angeforderten Ressource im Backend sitzt.

Es existieren viele unterschiedliche Architekturmuster und Typen von Middleware. *Anwendungsintegration* beschreibt die allgemeine Kopplung von mehreren beliebigen Anwendungen in einer Middleware. Eine Middleware zur Anwendungsintegration beschäftigt sich mit der Funktions- und Datenintegration unterschiedlicher Systeme eines Unternehmens und typischerweise auch der Bereitstellung dieser Funktionen und Daten. Die Enterprise Application Integration (EAI) ist eine bekannte Form der Anwendungsintegration von IT-Systemen betriebswirtschaftlicher Art [Keller, 2002].

2.3 Representational State Transfer (REST)

Ein Webservice ist ein Dienst, der eine Schnittstelle für einen „maschinellen“ Nutzer zur Verfügung stellt [Booth and Haas, 2004]. Das heißt, dass eine Maschine-zu-Maschine-Kommunikation über diese Schnittstelle stattfindet. Diese Kommunikation wird auf Basis von HTTP oder HTTPS über Rechnernetze abgewickelt. Ein Webservice besitzt zudem einen Uniform Resource Identifier (URI), über den er eindeutig identifizierbar ist.

REST ist genau genommen ein Architekturstil und ist unabhängig von der tatsächlichen Implementierung und der Verwendung bestimmter Protokolle. REST ist heutzutage jedoch ein etablierter Standard für die Implementierung von Webservices und verwendet in der Regel das HTTP-Protokoll [Helmich, 2013].
Ein URI ist in mehrere Teile unterteilt, wobei der vordere Teil die eindeutigen Netzwerkinformationen des Webservices enthält, um ihn zu identifizieren. Dieser Teil bleibt bei Anfragen an einen REST-Webservice daher gleich. Der hintere Teil der URI fragt die gewünschte Ressource dieses Webservices an. In REST dreht sich alles um solche Ressourcen. Laut Roy Fielding, dem Erfinder von REST, soll eine Ressource folgende Anforderungen erfüllen:

1. **Adressierbarkeit:** Eine Ressource muss über genau ein URI identifiziert werden können. Eine abfragbare Ressource könnte z.B. ein Kunde mit der Kundennummer 1234 sein.

2. **Zustandslosigkeit:** Die Kommunikation der Teilnehmer ist zustandslos. Es sollen keine Benutzersitzungen angelegt werden. Stattdessen müssen bei jeder Anfrage wieder alle notwendigen Informationen mitgeschickt werden.

3. **Einheitliche Schnittstelle:** Auf jede Ressource kann über einen einheitlichen Satz von Standardmethoden zugegriffen werden. Es müssen allerdings für eine Ressource nicht zwingend alle Methoden von der Anwendung unterstützt werden. Bei der Nutzung von HTTP existieren die Standard-HTTP-Methoden wie GET, POST, PUT, DELETE, und mehr.

4. **Entkopplung von Ressourcen und Repräsentation:** Es können verschiedene Repräsentationen einer Ressource existieren, die vom Client explizit angefordert werden kann. Beispielsweise kann eine Ressource in den Formaten XML oder JSON abfragbar sein. Die Formate müssen jedoch von der Anwendung unterstützt werden.

Ein Webservice, der auf HTTP basiert und mittels REST entworfen wurde, wird als RESTful HTTP-Service bezeichnet. In HTTP wird die Kennzeichnung der HTTP-Methode zu Beginn einer HTTP-Anfrage aufgeführt, um dem Server mitzuteilen, wie er mit der jeweiligen Anfrage umgehen soll. Ein kleiner Überblick soll Aufschluss über die wichtigsten HTTP-Methoden, in Verbindung mit REST, geben:

- Die **GET**-Methode greift lesend auf eine Ressource zu. Sie darf nicht dazu führen, dass Daten auf dem Server verändert werden.
- Mit der **POST**-Methode werden neue Ressourcen erstellt, die im Vorfeld nicht bereits eine URI besaßen.
- Die **PUT**-Methode wird verwendet, um Ressourcen, deren URI bereits bekannt ist, zu erstellen oder zu verändern.
- Mit der **DELETE**-Methode werden Ressourcen gelöscht.

2.4 Message Broker

Ein Message Broker (auch Integration Broker genannt) agiert als ein Vermittler von Nachrichten zwischen Systemen [Clark et al., 2013]. Ein Message Broker ist eine Nachrichtenorientierte Middleware, die die Aufgabe besitzt Nachrichten zu empfangen und an einen oder mehrere Empfänger weiterzuleiten. Zusätzlich besteht die Möglichkeit das Nachrichtenformat für die jeweiligen Empfängersysteme anzupassen. Message Broker realisieren üblicherweise das Publish-Subscribe-Muster. Das bedeutet, dass ein System eine Nachricht „veröffentlichen“ kann, ohne direkt mit dessen Empfängern in Verbindung zu stehen. Systeme, die den Message Broker „abonniert“ haben, erhalten daraufhin diese Nachricht.

Weil jedoch nicht immer jede Nachricht mittels Broadcast an alle Systeme gesendet werden muss, gibt es Filterstrukturen, die eine ausgewählte Weiterleitung der Nachrichten an die Empfänger erlaubt.

Im Bereich des Internet-of-Things (IOT) werden häufig Message Broker verwendet, um den intensiven Nachrichtenaustausch zwischen vielen Geräten zu erlauben. Für die Message Broker des IOT ist „MQTT“ ein beliebtes Protokoll. In MQTT funktioniert die Filterstruktur mithilfe von virtuellen Adressen namens „Topic“. Die Topics können aus mehreren Pfadebenen (Topic levels) bestehen. Ein Topic könnte beispielsweise wie folgt aussehen:
myhome/groundfloor/livingroom/temperature

Die Schrägstriche separieren die Topic levels. Darüber hinaus kann man Wildcards beim Abonnieren benutzen, mit denen man z.B. ganze Topic levels abonnieren kann.

3 Anforderungsanalyse

> „Anforderungen sind Aussagen über zu erfüllende Eigenschaften oder zu erbringende ‚Leistungen' eines Systems (bzw. Produkts), eines Prozesses oder der am Prozess beteiligten Menschen. Typischerweise umfassen sie Informationen darüber, warum ein System entworfen wird, was dieses System leisten soll und welche Einschränkungen dabei einzuhalten sind." [Partsch, 2010]

Eine knappere, allgemeinere Definition von „Anforderungen" wird in [Weißbach et al., 2013] formuliert:

> „Anforderungen sind die Planungsgrundlage für Projekte, da sie das beschreiben, was getan werden muss, um das geplante Ergebnis zu erstellen."

Die Anforderungsanalyse ist der wichtigste Schritt, um die Wünsche des Kunden geradewegs in eindeutige und zielführende Arbeitsschritte zu übersetzen. Während des Abarbeitungsprozesses der definierten Anforderungen können die Bearbeitungszustände als Maß der Fortschrittskontrolle untersucht werden, wie beschrieben in [Weißbach et al., 2013].
Es gibt [Partsch, 2010] zufolge einige Tätigkeiten, die zu Beginn der Anforderungsanalyse gemacht werden, um Ziele zu finden, Ziele zu klassifizieren, sogenannte Stakeholder zu finden und den Systemumfang festzulegen. Diese Tätigkeiten werden in den Sektionen der Anforderungsspezifikation erklärt. Des Weiteren gibt es drei weitere Schritte, die im Rahmen der Anforderungsanalyse mindestens einmal durchlaufen werden sollten:

1. Das Sammeln und die Ermittlung der Anforderungen.
2. Die Beschreibung und Strukturierung der Anforderungen.
3. Die Überprüfung und Bewertung der Anforderungen.

Spezifikationen beschreiben, im Gegensatz zu den Anforderungen, eine abstrakte, modellierte Sicht, wie etwas getan werden muss, um das geplante Ergebnis zu erstellen. Sie resultieren direkt aus den Anforderungen.

3.1 Vorgehen

Aufgrund der Kooperation mit Lufthansa Industry Solutions entfällt die erste Analyse der Anforderungen und Ziele aus der Zuständigkeit dieser Arbeit. Vor Beginn der Spezifikationsanalyse wurde vom Kunden bereits eine klar definierte Liste mit dem Großteil der Anforderungen ausgearbeitet.

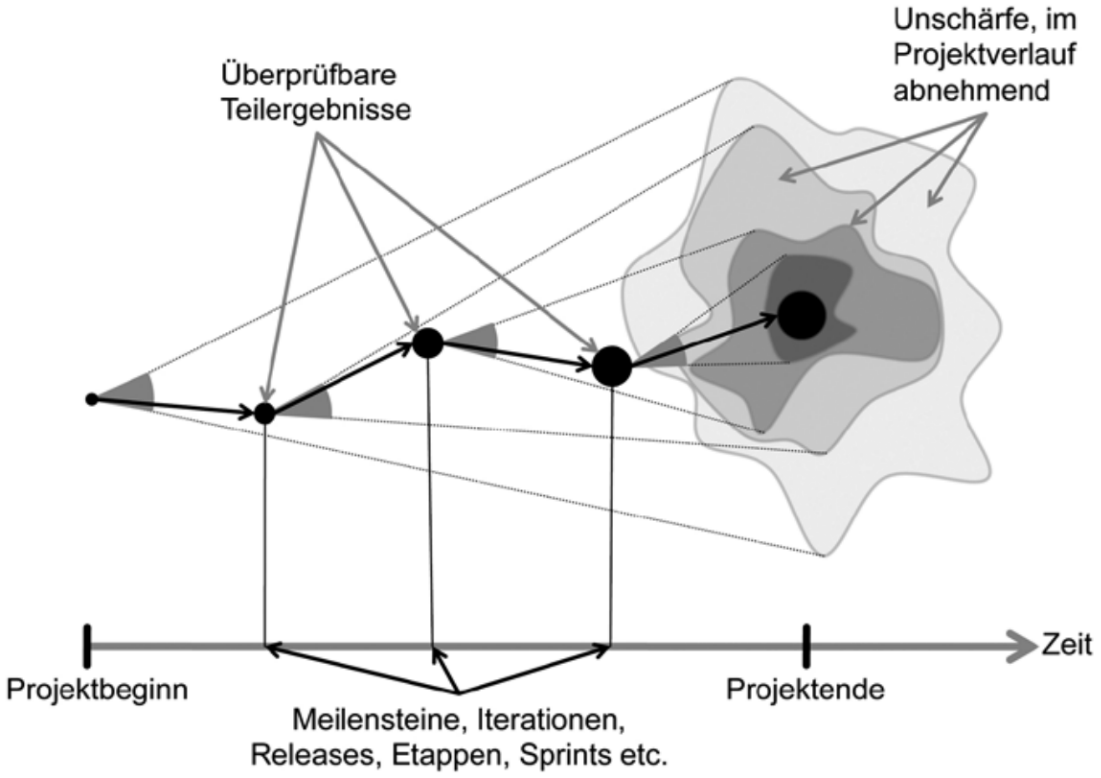

Abbildung 3.1: Vorteil des iterativen Vorgehensmodells[1]

Die Idee des iterativen Vorgehensmodells ist die schrittweise Abklärung ausgewählter, geeinigter Anforderungen.

[1] Quelle: [Weißbach et al., 2013]

Da die Anforderungen und Spezifikationen allerdings während der Entwicklungsphase iterativ auf Vollständigkeit und Machbarkeit geprüft und revidiert wurden, folgt eine Darstellung der aktuellen Anforderungen. Mithilfe von Software-Engineering-Modellen wie der Stakeholderanalyse, den Use Cases, den fachlichen Datenmodellen, Geschäftsprozessen und Schnittstellendefinition wurden die Anforderungen schriftlich und bildlich spezifiziert, um anstehende Aufgaben zur Erreichung des Ziels verständlicher und zugänglicher zu machen.

3.2 Zielsetzung

Die zu erstellende Middleware (im Folgenden auch RTLS-Middleware genannt) soll Softwareentwicklern für Benutzerapplikationen, die mit den Daten mehrerer Lokalisierungssysteme arbeiten wollen, eine zentralisierte und einheitliche Schnittstelle mit Lokalisierungsdiensten bieten. Die RTLS-Middleware erreicht dies, indem sie als ein intermediärer Knoten zwischen Anwendungen und Lokalisierungssystemen fungiert, der als eine zentrale Verteilerstelle für alle unternehmenseigenen Lokalisierungsdaten agiert. Die Aufgabe der RTLS-Middleware ist demnach die Anwendungsintegration (siehe Abschnitt 2.2) der Lokalisierungssysteme eines beliebigen Unternehmens. Die bereitgestellten Schnittstellen sollen möglichst wenig Einbuße in zeitlicher und räumlicher Genauigkeit im Vergleich zu den Ausgangssystemen erleiden. Das bedeutet, dass der gesamte Verarbeitungspro-

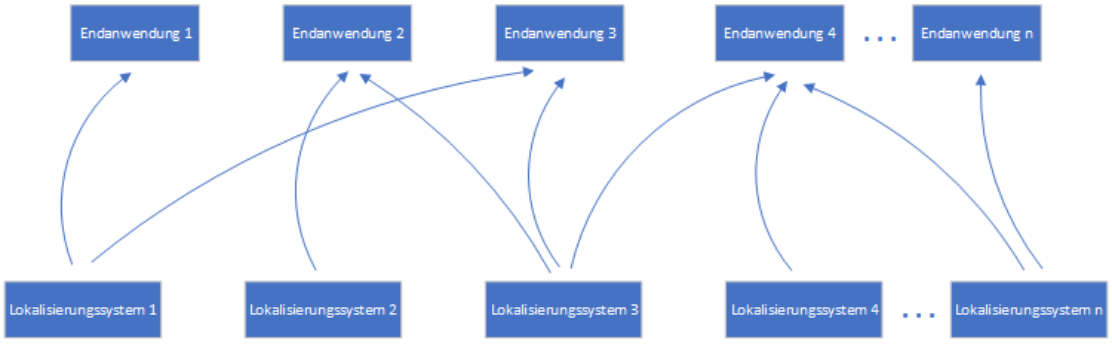

Abbildung 3.2: Beispielhafte Lokalisierungsserver-Infrastruktur ohne RTLS-Middleware

zess innerhalb der RTLS-Middleware - von der Datenabfrage bis zur Datenbereitstellung - ausreichend leistungsfähig sein muss, sodass die Abweichung zwischen dem Zeitpunkt der originalen Ortsmessung und dem Zeitpunkt der Zurverfügungstellung minimal ist. Die Echtzeitunterstützung von Ausgangssystemen soll also weitestgehend konserviert blei-

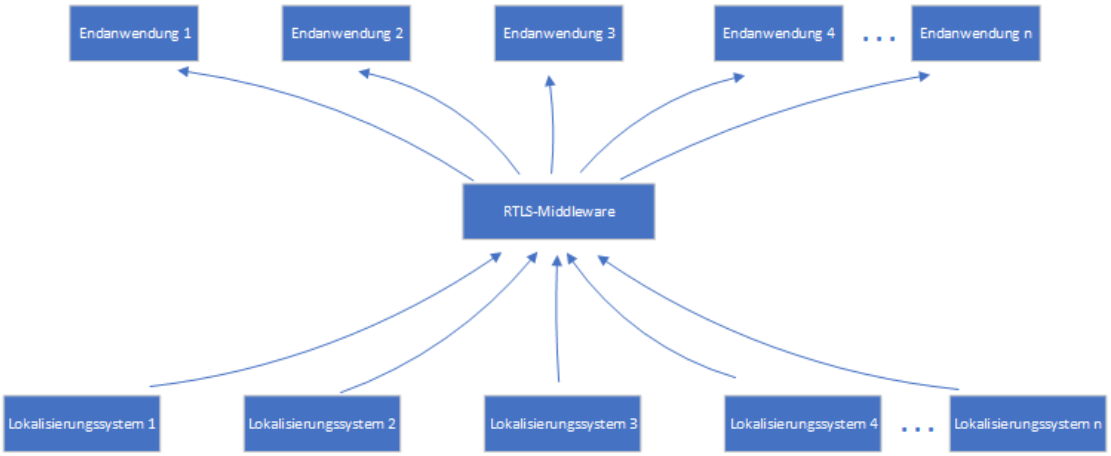

Abbildung 3.3: Beispielhafte Lokalisierungsserver-Infrastruktur mit RTLS-Middleware

ben.

Die abgerufenen Lokalisierungsdaten sollen außerdem global eindeutige Informationen besitzen, sodass sie leicht darstellbar und vergleichbar sind. Die Lokalisierungswerte sollen dabei auf der globalen Karte, selbst im kleinen Maßstab, eine präzise Ortung gewährleisten. Die RTLS-Middleware soll zudem ohne Probleme eine realistische Anzahl von mindestens mehreren hundert Lokalisierungsobjekten gleichzeitig verarbeiten können.

Die Abbildungen 3.2 und 3.3 sollen den Unterschied zwischen der Lokalisierungsinfrastruktur eines beispielhaften Unternehmens mit und ohne der Integration der RTLS-Middleware verdeutlichen. Sei n die Anzahl an Lokalisierungssystemen und k die Anzahl an Endanwendungen, die die Lokalisierungsdaten benötigen. Ohne die Verwendung der Middleware, gibt es bei einer Vollvernetzung der Endanwendungen mit den Lokalisierungssystemen, $n \cdot k$ Verbindungen. Die Schnittstellen zu allen verbundenen Lokalisierungssystemen müssen in jede Endanwendungen einzeln integriert werden. Mithilfe der Verwendung der Middleware gibt es im Falle der Vollvernetzung nur $n+k$ Verbindungen. Die Endanwendungen müssen außerdem lediglich eine zusätzliche Schnittstelle integrieren. Darüber hinaus entsteht insgesamt weniger Kommunikationsverkehr an den Lokalisierungssystemen, da die Lokalisierungssysteme ausschließlich direkt mit der Middleware kommunizieren müssen.

Die Installation, Administration und Konfiguration soll betriebssystemunabhängig und unkompliziert sein.

Die RTLS-Middleware soll nur auf dem aktuellen Stand der Lokalisierungsdaten arbeiten. Dieser aktuelle Datenstand sowie applikationseigene Daten sollen in einer Datenbank gespeichert werden. Beim Aktualisieren der Daten soll die Möglichkeit einer simplen, bei-

läufigen Auswertung existieren, die den vorherigen mit dem aktuellen Stand vergleicht und daraufhin spezifizierte Events auslösen kann. Daten sollen zusätzlich zur historischen Auswertung periodisch an einen Service für Datenanalysen geschickt werden können.

3.3 Stakeholder

Als Stakeholder werden diejenigen Personen bezeichnet, die Interesse an dem Softwareprojekt und dessen Ergebnis haben. Die Analyse dieser Stakeholder ist ein wichtiger Planungsschritt, um einen besseren Überblick beim Ermitteln möglicher zukünftiger Anforderungen zu bekommen, da verschiedene Adressaten einer Software nicht die gleichen Wünsche und Anforderungen teilen. Die Klassifizierung von Nutzern hilft daher der besseren Abdeckung beim Überlegen von potenziellen Anforderungen. Die Tabellen 3.1 und 3.2 zeigen die ermittelten fiktiven Stakeholder, die mit der Middleware interagieren.

Rolle	Administrator der Middleware
Beschreibung	Der Administrator der RTLS-Middleware ist für die Installation und Konfiguration vor Inbetriebnahme verantwortlich. Außerdem muss er Kontrollaufgaben und ggf. darüber hinausgehende Datenverwaltungsaufgaben übernehmen.
Ziel	Flexible und schnelle Einrichtung und Installation. Intuitive Konfigurationsoptionen und zentraler Ort für Konfigurationen. Aussagekrätige Log-Nachrichten. Einfache Schnittstelle für die Verwaltung der Daten.

Tabelle 3.1: Stakeholder: Administrator

Rolle	Endanwendungs-Stakeholder
Beschreibung	Ein Endanwendungs-Stakeholder ist z.B. ein Nutzer oder Administrator einer Applikation, die mit der RTLS-Middleware interagiert. Diese Applikationen benötigen Lokalisierungsdaten von einer beliebigen, unbestimmten Anzahl an Lokalisierungssystemen. Die Applikationen werden unabhängig von der RTLS-Middleware entwickelt und werden in beliebigen, unvorhersehbaren Bereichen eingesetzt.
Ziel	Eine zentrale und vollständige Schnittstelle. Intuitive Schnittstellenoperationen. Einheitliche, eindeutig identifizierende Datenformate.

Tabelle 3.2: Stakeholder: Endanwendungs-Stakeholder

3.4 Anforderungsspezifikation

Spezifikationen sind nach [Radatz, 1990] im Kontext der Softwareentwicklung formalisierte, vollständige, präzise und überprüfbare Beschreibungen von Merkmalen eines Systems oder einer Komponente. Die Anforderungsspezifikation ist folglich ein Dokument, das die Anforderungen an ein System oder eine Komponente festlegt. Typischerweise sind funktionale Anforderungen (Abschnitte 3.4.1, 3.4.2 und 3.4.4), Designanforderungen (Abschnitt 3.4.3), Schnittstellenanforderungen (Abschnitt 3.4.5), Leistungsanforderungen und Entwicklungsstandards (Abschnitt 3.4.6) enthalten.

3.4.1 Funktionale Anforderungen

In den funktionalen Anforderungen werden Kernaufgaben der Software, hinsichtlich der Nutzung durch die verschiedenen Stakeholder, gesammelt. Die funktionalen Anforderungen beschreiben Ansprüche der Stakeholder an Funktionalitäten der zu erstellenden Software. Die gelisteten funktionalen Anforderungen werden im Abschnitt Use Cases erneut aufgegriffen.

Anforderungen - Administrator

FA 1. Die Konfigurationsoptionen sollen zentral, an möglichst wenigen verschiedenen Stellen, anpassbar sein.

FA 2. Die Middleware muss eine einfache Schnittstelle bieten, mit der persistierte Stammdaten hinzugefügt und verändert werden können.

FA 3. Die Middleware muss Funktionalitäten zur Verfügung stellen, mit der neue Verbindungen zu Lokalisierungssystemen hinzugefügt werden können.

FA 4. Die Middleware muss Funktionalitäten zur Verfügung stellen, mit der Verbindungen aktiviert und deaktiviert werden können.

FA 5. Die Middleware muss unabhängig von den unterliegenden Lokalisierungssystemen lauffähig und verwaltbar sein.

Anforderungen - Endanwendungs-Stakeholder

FA 6. Die Middleware soll, je nach Bedarf, jedes beliebige Lokalisierungssystem unterstützen können.

FA 7. Die Middleware muss ein einziges zentrales System bieten, welches mit allen Lokalisierungssystemen verbunden werden kann.

FA 8. Die Middleware muss alle relevanten Daten mittels entsprechender Operationen an einer zentralen Schnittstelle abrufbar machen.

FA 9. Die verschiedenen Schnittstellen der Middleware müssen die Daten in einheitlichen, definierten Formaten ausgeben.

FA 10. Die Lokalisierungsdaten müssen , falls nötig, umgewandelt werden, sodass ihre Ortsrepräsentation global nachvollziehbar ist.

FA 11. Die Lokaliserungsdaten von aktivierten Verbindungen sollen periodisch aktualisiert werden.

FA 12. Die Middleware muss eine eigene Ebene der Abstraktion auf seine Daten bieten, die die originalen Lokalisierungsdaten, um weitere Informationen erweitert.

FA 13. Die Middleware muss Funktionen zum Filtern von Datenabfragen zur Verfügung stellen.

FA 14. Die aktuellen Daten der Middleware müssen persistiert sein.

FA 15. Falls gewünscht, sollen Lokalisierungsdaten bei jeder Aktualisierung an einen Dienst für Datenanalyse gesendet werden.

FA 16. Benutzerdefinierte Events sollen initialisierbar und parametrisierbar sein.

FA 17. Events müssen ausgelöst werden, wenn spezifische Zustandsänderungen in den Daten auftreten.

FA 18. Die Auslösung eines Events muss sofort an die Endanwendungen weitergeleitet werden.

3.4.2 Anwendungsfälle (Use Cases)

Anwendungsfälle dienen der grafischen oder textuellen Beschreibung von fachlichen Anforderungen eines Systems. In diesem Abschnitt werden die Anwendungsfälle in textueller Repräsentation erklärt. Die Diagramme im Abschnitt der Geschäftsprozesse 3.4.4 orientieren sich teilweise an den hier beschriebenen Anwendungsfällen.

> „Ein Anwendungsfall beschreibt die Art und Weise, wie ein Akteur mit dem zu erstellenden System interagieren kann und dient daher als eine Beschreibung für das äußerlich sichtbare Systemverhalten.“ [Weißbach et al., 2013]

Für Anwendungsfälle, und allgemein für fachliche Anforderungen, gilt, dass ihre Beschreibungen sich darauf beziehen, welche Leistungen erbracht werden sollen und nicht wie diese Leistungen erbracht werden sollen. Der Business-Use-Case in Abbildung 3.4 zeigt ein Gesamtüberblick der Funktionalitäten der Applikation. Im Abschnitt Geschäftsprozesse sieht man die Interaktion der verschiedenen Funktionalitäten, wobei diese hier, für ein besseres Verständnis, klar voneinander abgegrenzt werden.
Folgende System-Use-Cases ergeben sich aus den funktionalen Anforderungen:

1. Installation der Middleware.
2. Hinzufügen einer Verbindung.
3. Bearbeiten der systemeigenen Stammdaten.
4. Abfrage aktueller Lokalisierungsdaten.
5. Auswertungen und Ereignisverwertung.

3.4.2.1 Geschäftsanwendungsfälle

Ein Geschäftsanwendungsfall ist eine Aktivitätsfolge, die von einem Ereignis ausgelöst wird und zu einem Teilergebnis führt, das einen geschäftlichen Wert darstellt. In diesem Abschnitt sehen wir eine Auflistung der verschiedenen Geschäftsanwendungsfälle in Bezug auf die Stakeholder. Die Geschäftsanwendungsfälle werden allerdings in Bezug auf die Geschäftsprozesse (Abschnitt 3.4.4) wieder aufgegriffen und anschaulich modelliert.

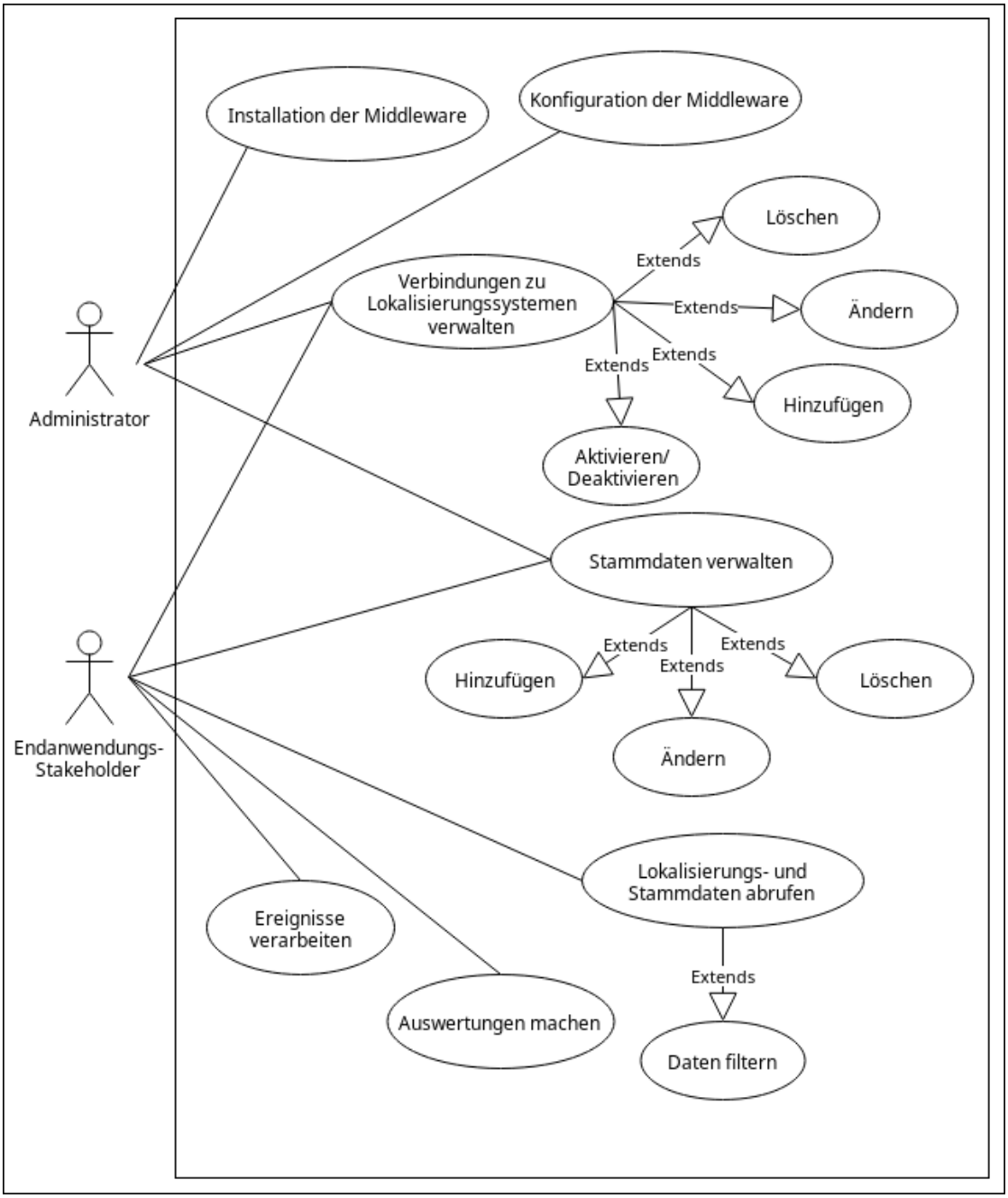

Abbildung 3.4: Geschäftsanwendungsfälle Übersicht

3.4.2.2 System-Anwendungsfälle

Anwendungsfall 1 - Installieren der Middleware

Bezeichnung	Installieren der Middleware
Ziel im Kontext	Inbetriebnahme der Middleware
Akteur	Administrator
Auslöser	Der Kunde besitzt mehrere Lokalisierungssysteme und möchte die RTLS-Middleware nutzen, um deren Daten zu vereinen.
Vorbedingung	Die RTLS-Middleware wurde korrekt erworben. Die Lokalisierungssysteme sind konfiguriert, sodass sie ihre Daten an externe Systeme übermitteln können.
Nachbedingung	Eine saubere Installation der RTLS-Middleware läuft auf einem Server, die nach den Wünschen des Kunden operiert.
Anforderungen	FA 1, FA 5, FA 12

Erfolgsszenario

1. Der Administrator spezifiziert und erzeugt ggf. ein Verzeichnis, in dem Applikationsdatein wie beispielsweise Logs und Datenbankeinträge, gespeichert werden.
2. Der Administrator konfiguriert weitere Start- und Laufzeitparameter in der RTLS-Middleware.
3. Der Administrator installiert einen (Web-)Server oder Container.
4. Der Administrator konfiguriert Laufzeitparameter des (Web-)Servers.
5. Der Administrator stellt die Middleware auf dem (Web-)Server bereit.
6. Der Administrator startet die RTLS-Middleware.

Anwendungsfall 2 - Verbindung hinzufügen

Bezeichnung	Verbindung hinzufügen
Ziel im Kontext	Das Hinzufügen einer funktionierende Verbindung
Akteur	Administrator
Auslöser	Ein neues Lokalisierungssystem soll in die RTLS-Middleware eingebunden werden und der Administrator soll dies direkt an der Middleware vornehmen.
Vorbedingung	Die RTLS-Middleware ist installiert und läuft. Die Verbindung zu dem Lokalisierungssystem wurden noch nicht hinzugefügt.
Nachbedingung	Aktuelle Lokalisierungsdaten der neuen Verbindung sind an der RTLS-Middleware abrufbar.
Anforderungen	FA 2, FA 3, FA 4, FA 6, FA 7

Erfolgsszenario

1. Der Administrator muss die Verbindungs- und Authentifizierungsinformationen des Lokalisierungssystems identifizieren.
2. Der Administrator muss bzw. kann an dem Lokalisierungssystem ggf. zusätzliche Konfigurationsoption bezüglich der Konnektivität auswählen.
3. Mittels Aufruf der Funktion mit Angabe der gewünschten Parameter wird eine Verbindung erzeugt und automatisch aktiviert.
4. Die Funktion gibt eine Antwort, ob der Aufruf erfolgreich war.
5. Beim Auftreten eines Fehlers, aufgrund fehlerhafter Verbindungsparameter, bleibt die Verbindung gespeichert, wird jedoch deaktiviert.
6. Der Administrator kann durch den Aufruf einer weiteren Funktion, die Verbindungsparameter der fehlerhaften Verbindung korrigieren.
7. Beim Auftreten eines Verbindungs- bzw. Netzwerkfehlers versucht die Middleware in periodischen Abständen die Verbindung wieder herzustellen.

Anwendungsfall 3 - Stammdaten bearbeiten

Bezeichnung	Stammdaten bearbeiten
Ziel im Kontext	Stammdatensatzveränderung
Akteur	Endanwendungs-Stakeholder oder Administrator
Auslöser	Der Administrator wird beauftragt oder ein Endanwender will selbstständig den Datenbestand der RTLS-Middleware verändern, da der Endanwender einen Nutzen davon hat.
Vorbedingung	Die RTLS-Middleware läuft.
Nachbedingung	Der Stammdatensatz wurde erweitert, reduziert oder bearbeitet.
Anforderungen	FA 2, FA 5, FA 14

Erfolgsszenario

1. Ein Nutzer füllt eine entsprechende Eingabemaske einer Endanwendung aus und bestätigt.
2. Die Applikation übermittelt die eingegebenen Daten an die entsprechende Funktion der RTLS-Middleware.
3. Die Funktion antwortet der Applikation, ob der Aufruf erfolgreich war.
4. Im Falle eines Fehlers dürfen unter keinen Umständen Änderungen aus diesem Aufruf entspringen.

Anwendungsfall 4 - Lokalisierungsdaten abfragen

Bezeichnung	Lokalisierungsdaten abfragen
Ziel im Kontext	Gewinnung der gewünschten Lokalisierungsinformationen.
Akteur	Endanwendungs-Stakeholder
Auslöser	Die Endanwendung benötigt Lokalisierungsdaten.
Vorbedingung	Die RTLS-Middleware läuft. Die nötigen Verbindungen in der RTLS-Middleware sind aktiviert.
Nachbedingung	Ein Schnappschuss der aktuellen Lokalisierungsdaten wurde übermittelt.
Anforderungen	FA 8, FA 9, FA 10, FA 11, FA 12, FA 13

Erfolgsszenario

1. Ein Nutzer öffnet den integrierten Kartendienst einer Endanwendung.
2. Der Nutzer wählt anhand eines Filtermenüs aus, welche Lokalisierungsdaten in die Karte geladen werden sollen.
3. Nach dem Bestätigen ruft die Applikation die entsprechenden Funktionen zum Abrufen der Lokalisierungsdaten an der RTLS-Middleware auf.
4. Die gewünschten Lokalisierungsdaten werden der Applikation übermittelt und in die Karte geladen.

Anwendungsfall 5 - Ereignisse verarbeiten

Bezeichnung	Ereignisse verarbeiten
Ziel im Kontext	Ereignisse werden in der Endanwendung berücksichtigt.
Akteur	Endanwendungs-Stakeholder
Auslöser	Ein Endnutzer möchte sofort über eine bestimmte Zustandsänderung informiert werden.
Vorbedingung	Die RTLS-Middleware läuft.
Nachbedingung	Neue Ereignisse werden ausgelöst, sobald eine gegebene Zustandsänderung auftritt.
Anforderungen	FA 16, FA 17, FA 18

Erfolgsszenario

1. Ein Nutzer einer Endanwendung wählt über ein Menü die Events aus, über die er benachrichtigt werden möchte.
2. Der Nutzer wechselt zu einer Übersichtsseite, auf der ihm ausgelöste Events angezeigt werden.
3. Sobald die Middleware ein Ereignis erkennt und ein Event auslöst, wird die Applikation umgehend benachrichtigt.
4. Die Übersicht zeigt dem Nutzer das Event an, sodass er daraufhin handeln kann.

Anwendungsfall 6 - Historische Auswertung

Bezeichnung	Historische Auswertung
Ziel im Kontext	Datenanalyse zur Erkenntnisgewinnung.
Akteur	Endanwendungs-Stakeholder
Auslöser	Ein Endnutzer möchte Aufschluss über den Zustand oder Veränderungen der vergangene Lokalisierungsdaten erhalten.
Vorbedingung	Die RTLS-Middleware läuft. Die Datenanalyse ist eingeschaltet und eine Verbindung zu einem Datenanalysedienst ist konfiguriert.
Nachbedingung	Übermittelte Daten müssen selbstständig gepflegt und ausgewertet werden.
Anforderungen	FA 1, FA 9, FA 11, FA 15

Erfolgsszenario

1. Die Middleware sendet bei jedem Aktualisieren der Lokalisierungsdaten, die neuen Daten an den Datenanalysedienst.
2. Ein Datenanalyst baut in dem Datenanalysedienst eine Datenbank mit historischen Lokalisierungsdaten auf.
3. Der Datenanalyst wertet die Daten aus, um beispielsweise Wege oder Prozesse zu optimieren.

3.4.3 Fachliches Datenmodell

Das fachliche Datenmodell beschreibt die im Gesamtkontext relevanten Entitäten und Datentypen der Software in einer konzeptionellen Sichtweise. Es werden keine technischen Informationen über das Datenbankmanagementsystem (logische Sicht) oder die Datenspeicherung (physische Sicht) dargestellt. Das fachliche Datenmodell schränkt den Entwickler daher nicht in der technischen Umsetzung ein, sondern erleichtert lediglich die Entscheidungsfindung im datentechnischen Entwurf und Implementierung. Oft ist das fachliche Datenmodell bereits eine vollständige konzeptionelle Darstellung der Datenbank.

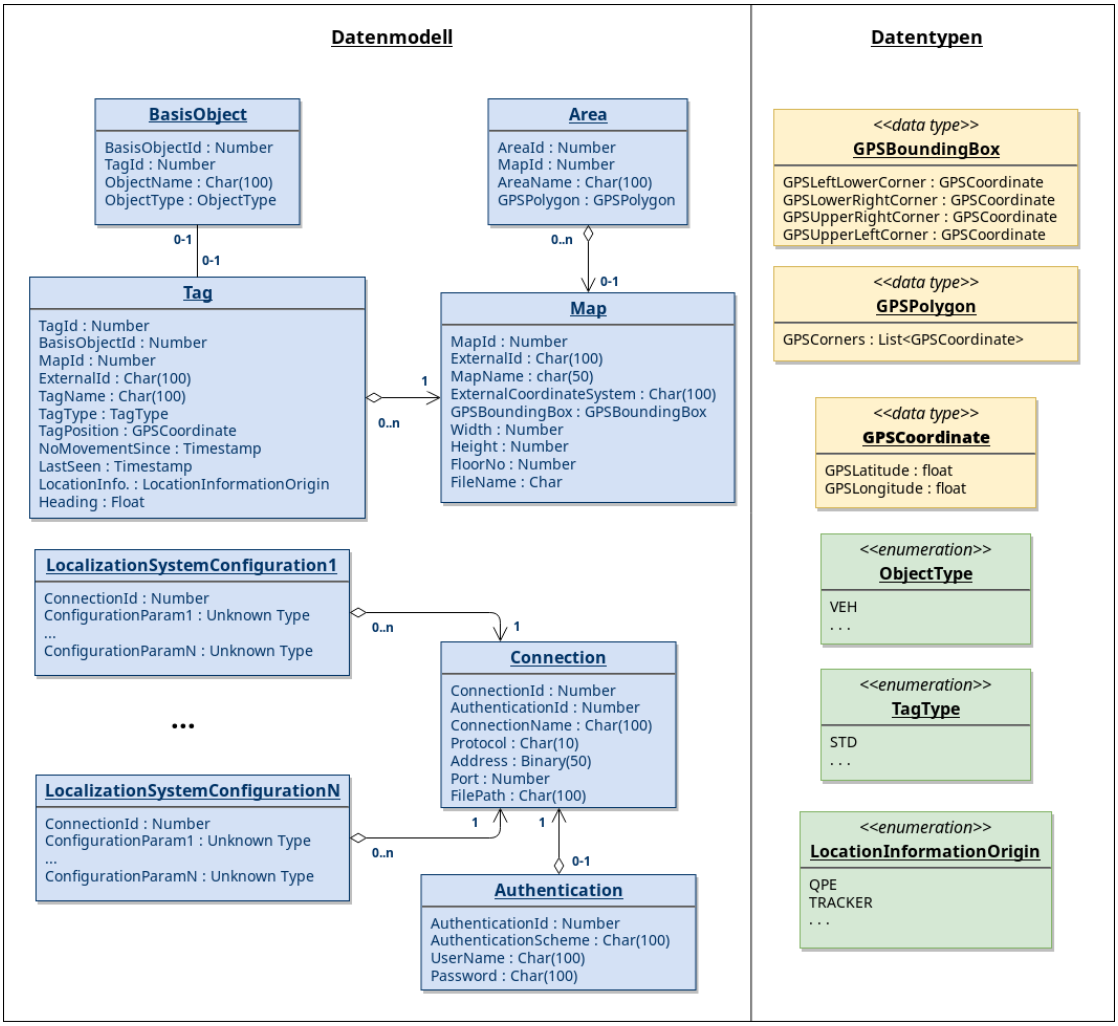

Abbildung 3.5: Fachliches Datenmodell als UML-Klassendiagramm

Abbildung 3.5 zeigt ein fachliches Datenmodell im Stil eines UML-Klassendiagramms (Unified Modeling Language). Die Entität *Tag* ist die zentrale Komponente der RTLS-Middleware. Sie beschreibt die aktuellen Informationen eines Ortungsgerätes und bildet die Lokalisierungsdaten eines Herkunftslokalisierungssystems ab. Der Name *Tag* entstammt dem Fakt, dass die Ortungsgeräte meist kleine Anhänger sind, die an beliebigen Objekten befestigt werden können, um diese zu lokalisieren. Die *ExternalId* eines *Tags* ist die identifizierende Bezeichnung im Herkunftssystem. *TagType* und *LocationInformationOrigin* sind als „Enumeration"-Datentyp definiert, welche jeweils einen festen Wertebereich vorgeben. Der *TagType* ist die Art des *Tags* und findet Verwendung, falls mehrere verschiedene Typen von Ortungsgeräten in einem Herkunftssystems existieren. Die *LocationInformationOrigin* enthält die Kennung des Herkunftssystem, um identifizieren zu können, aus welchem Lokalisierungsprodukt die Daten stammen. Das Feld *Heading* zeigt an in welche Himmelsrichtung sich der *Tag* bewegt. Die Himmelsrichtung wird als Kreiswinkel [0°,360°) angegeben (0° ist Norden, 90° ist Osten etc.). Eines der wichtigsten Felder der Middleware ist die Ortskoordinate *GPSCoordinate*. Für eine global eindeutige Lokalisierung bot sich diesbezüglich das WGS84-Format, mit der Darstellung der Koordinate als Breitengrad (Latitude) und Längengrad (Longitude), an.
Ein *Tag* kann optional eine Repräsentation des Objekts, an dem der *Tag* befestigt ist, referenzieren. Das Objekt wird durch die Entität *BasisObject* dargestellt, welches einen bezeichnenden Namen des Objekts und den Typ des Objekts (*ObjectType*) enthält.

Eine *Map* repräsentiert die Karte eines abgegrenzten Gebiets, in dem sich eine feste Teilmenge der Tags aufhalten. Viele lokal installierte Lokalisierungssysteme benutzen eine solche, klar abgegrenzte Karte und orten ausschließlich Ortungsgeräte innerhalb des abgegrenzten Gebiets. Die Id der Karte des Herkunftssystems ist in der Entität wieder als *ExternalId* gespeichert. Häufig spannt das Herkunftssystem zusätzlich ein eigenes, lokales Koordinatensystem auf. Die Kennung des lokalen Koordinatensystems wird in *ExternalCoordinateSystem* gespeichert. Da die Karten üblicherweise rechteckig sind, kann das Gebiet der Karte mit einem rechteckigen Begrenzungsrahmen dargestellt werden, wobei dafür die vier Eckpunkte in der *GPSBoundingBox* gespeichert werden müssen. Sollte keine rechteckige Fläche als *Map* vorliegen, muss ein Begrenzungsrahmen gefunden werden, der das Gebiet der Karte vollständig, aber mit möglichst wenig zusätzlicher Fläche absteckt.

Eine *Area* ist eine polygonale Fläche, die entweder innerhalb oder unabhängig und außerhalb einer *Map* definierbar ist. Eine *Area* erlaubt die Abfrage des Enthaltenseins eines *Tags* und dadurch bestimmte Events wie z.B., wenn ein *Tag* die *Area* betreten oder verlas-

sen hat. Außerdem kann die polygonale Fläche *GPSPolygon* ungefähr bemessen werden. *GPSPolygon* speichert die Eckpunkte des Polygons in einer Liste mit einer beliebigen, aber festen Anzahl an Eckpunkten.

Die Entität *Connection* repräsentiert eine allgemeine Serververbindung. Falls eine Authentifizierung verlangt wird, kann ein *Authentication*-Objekt referenziert werden, wobei das *AuthenticationScheme* die Art der Authentifizierung ist, anhand dessen beim Verbinden, aus dem *UserName* und *Password* ein Authentifizierungsschlüssel berechnet wird. Da der Authentifizierungsschlüssel auf unserer Clientseite oft dynamisch generiert werden muss, muss das Passwort in Klartext gespeichert werden. Aus Sicherheitsgründen sollte die RTLS-Middleare daher nicht direkt aus dem öffentlichen Netz erreichbar sein.

Die *LocalizationSystemConfiguration*-Entitäten repräsentieren jeweils ein Lokalisierungssystem. Jedes System besitzt eigene, individuelle Konfigurationsoptionen, die sich auf unterschiedliche Weise auf die Abfrage der *Tag*-Informationen auswirken können. Aufgrund der Unterschiedlichkeit der Felder und im Hinblick auf die Erweiterbarkeit der RTLS-Middleware gibt es für jedes Lokalisierungssystem eine eigene *LocalizationSystemConfiguration*-Entität.

3.4.4 Geschäftsprozesse

> „Aus formaler Sicht ist ein Geschäftsprozess charakterisiert durch die sachliche und zeitliche Abfolge von Geschäftsanwendungsfällen, die zu einem wohldefinierten und messbaren Ergebnis führen. Dabei wird ein Geschäftsprozess durch ein oder mehrere Ereignisse ausgelöst und endet mit einem oder mehreren Ergebnisereignissen.“ [Rau, 2016]

Abbildung Abb. 3.6 bildet die Geschäftsprozesse in Bezug auf deren Datenflüsse als ein konzeptionelles Datenflussmodell ab. Das Modell beschreibt die logische Unterteilung der Datenbestände (Kästen mit schwarzen Rändern oben und unten), der Aktionen (Blaue Kreise) sowie der externen Systeme (Blaue Kästen). Das Modell nimmt keinen Bezug auf die tatsächliche interne Umsetzung der Datenspeicherung und Kommunikationsverbindungen.

Ein Geschäftsprozess umfasst mehrere Geschäftsanwendungsfälle (Abschnitt 3.4.2). Die wichtigsten Geschäftsprozesse werden im Folgenden in Form von Aktivitätsdiagrammen in UML Notation und in ihrer Ablaufstruktur dargestellt. Sie sollen einen detaillierteren

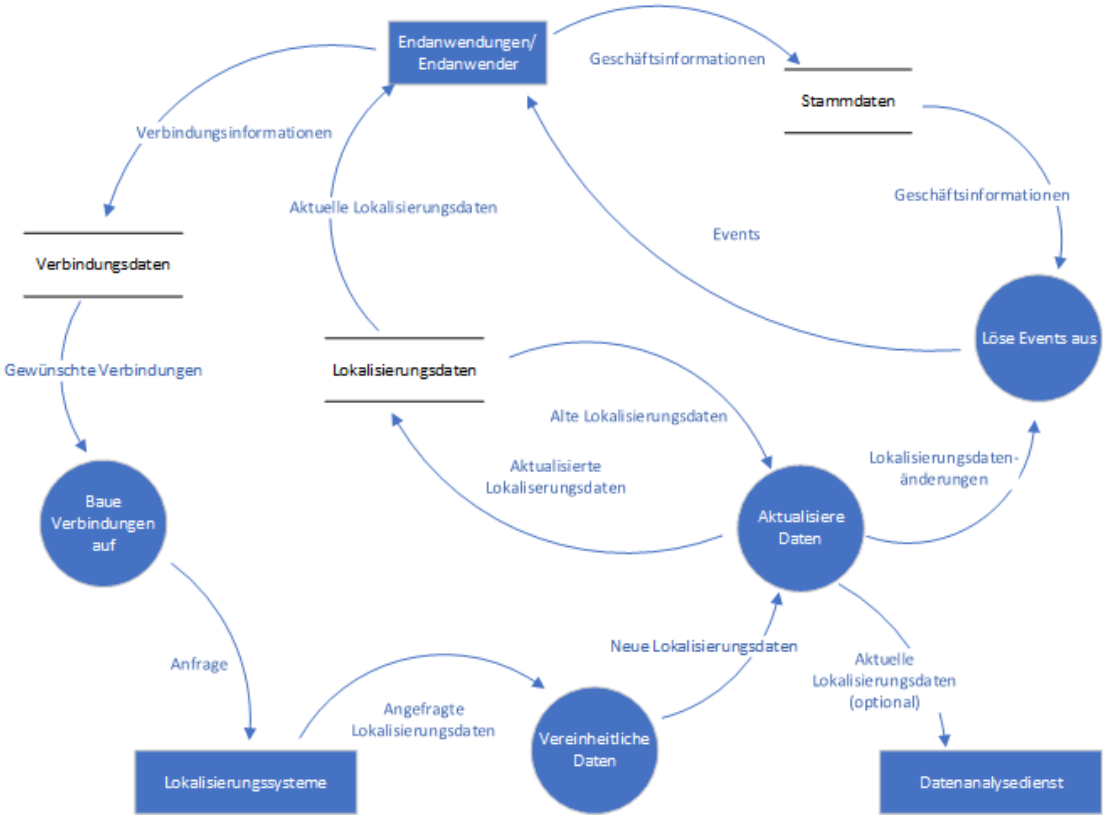

Abbildung 3.6: Darstellung des Datenflusses der RTLS-Middleware

Einblick in die Abläufe der Funktionen der Middleware geben. Normalerweise wird die Ergebniserbringung der Geschäftsprozesse in Hinblick auf die Benutzerinteraktion sachlich dargestellt. Da die RTLS-Middleware nach der Startkonfiguration jedoch komplett automatisiert laufen kann und darüber hinaus lediglich einfache Datenabfragen an den Schnittstellen abgerufen werden sollen, wird mithilfe der Abbildung 3.7 bereits innerhalb dieses Abschnittes eine erste leicht technischere Sicht dargestellt, die einen Grundbaustein für den späteren Softwareentwurf legt.

Abbildung 3.7 zeigt den Prozess der Abfrage der Lokalisierungsdaten an der RTLS-Middleware. In diesem Diagramm wird insbesondere angestrebt, die automatisierte, interne Ablaufstruktur der RTLS-Middleware abzubilden. Die Lokalisierungsdaten werden periodisch aktualisiert. Je geringer in diesem Prozess die Zeit der Intervalle ist, desto näher sind die Daten der Middleware an den tatsächlichen, gegenwärtigen Ortsdaten der Herkunftssysteme und desto akkurater sind die Zustandsänderungen.

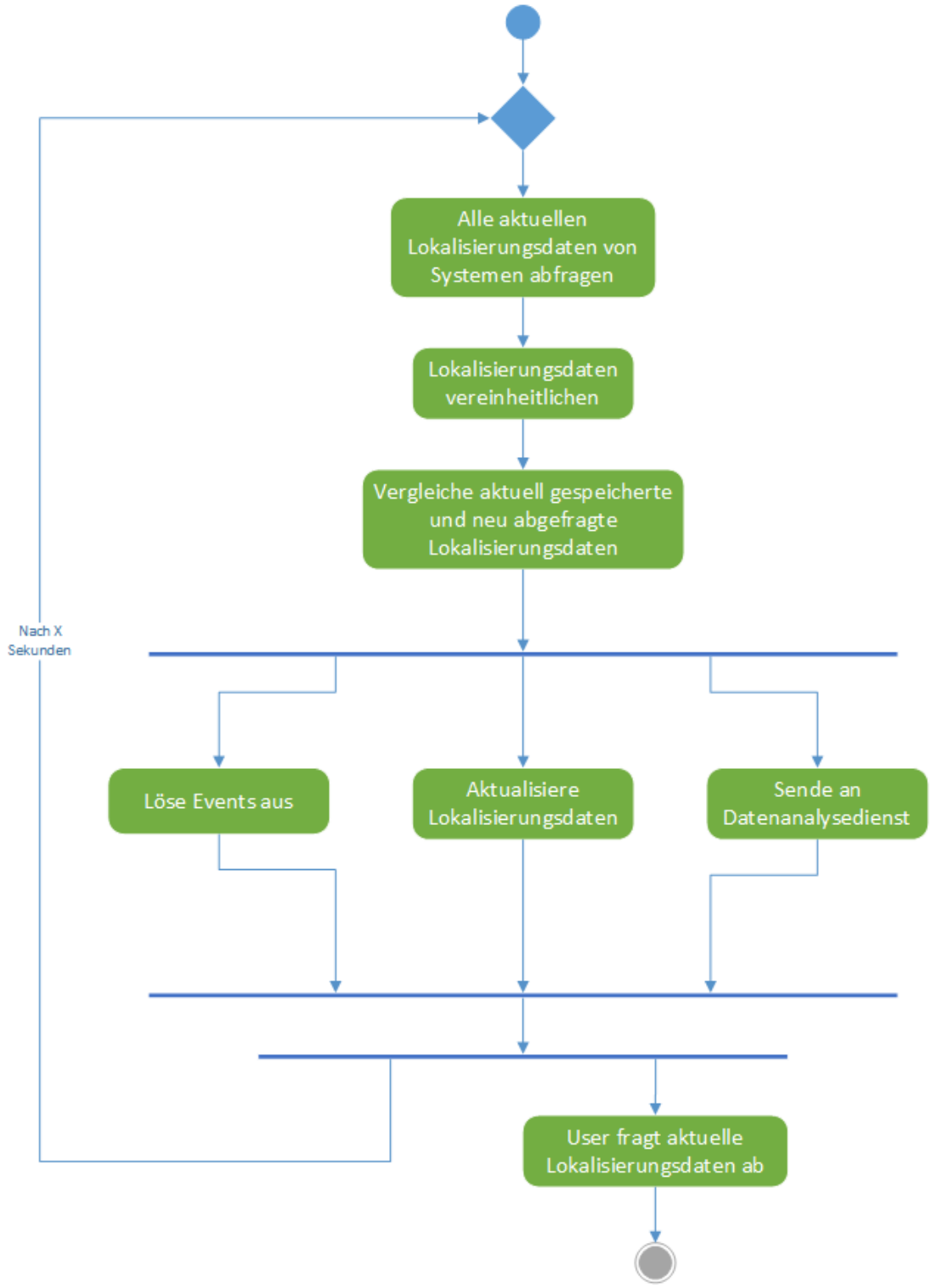

Abbildung 3.7: Prozess der Zurverfügungstellung aktueller Daten

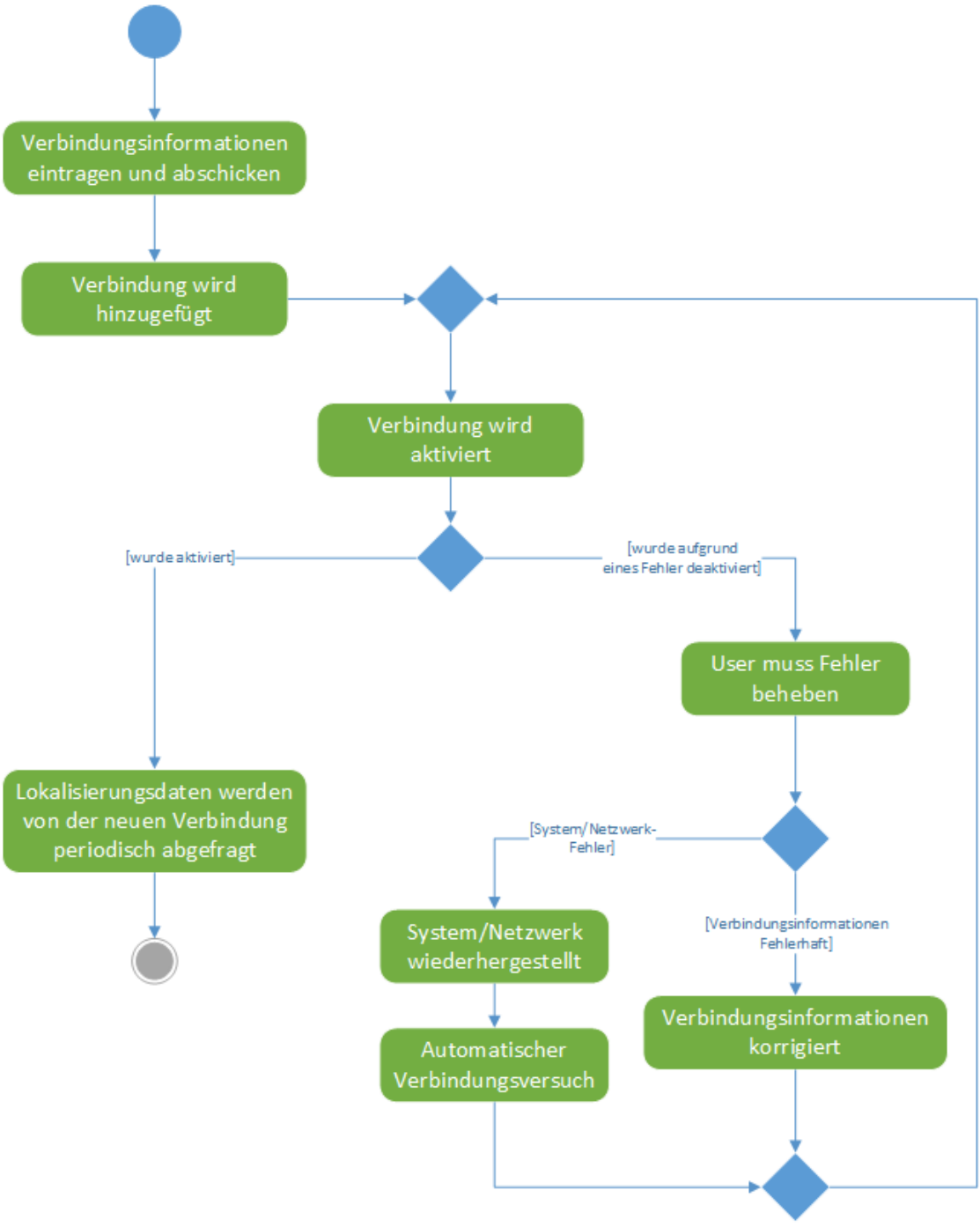

Abbildung 3.8: Prozess des Hinzufügen einer Verbindung

Abbildung 3.8 zeigt den Prozess des Hinzufügens einer neuen, aktivierten Verbindung von der Middleware zu einem Lokalisierungssystem. Nach dem Hinzufügen und Ändern einer Verbindung wird diese sofort beiläufig versucht zu aktivieren. Falls die Verbindung nicht aktiviert werden kann, wird sie deaktiviert. Nach behobenen oder temporären Netzwerk- oder Systemfehlern, kann die hinzugefügte Verbindung unverändert bleiben, weil ein periodischer Batch Job versucht die deaktivierten Verbindungen automatisch wiederzuverbinden. Falls die Verbindungsinformationen hingegen fehlerhaft sind, muss die Verbindung vom User korrigiert werden. Nach der Korrektur der Verbindungsdaten wird die Verbindung ebenfalls direkt aktiviert.

3.4.5 Schnittstellen zu Nachbarsystemen

Die Schnittstellen eines Systems besitzen eine besondere Bedeutung, da über die Schnittstellen die gesamte Kommunikation zwischen den verschiedenen Systemen abläuft. Über sie läuft der Transfer von Daten und Steuerungsinformationen, wodurch die Systeme zusammenarbeiten und letztlich den Mehrwert des Gesamtsystems erzeugen.

Ein wichtiger Aspekt, vor dem Entwurf der Schnittstellen des eigenen Systems, ist die Frage nach der Regulierung von Berechtigungen: Sowohl Applikationen als auch Administratoren der Middleware sollen alle Berechtigungen erhalten und auf dieselbe Schnittstelle zugreifen können. Die Schnittstelle sollte aufgrund dessen mithilfe eines bekannten Standards implementiert werden, sodass die Applikationen ihrerseits mit geringem Aufwand über existierende Bibliotheken auf die Schnittstelle zugreifen können. Die Administratoren sollen andererseits entweder über ein integriertes oder ein externes, bereits existierendes Tool direkt mit der Middleware kommunizieren können.
Da die Middleware an unternehmensinterne Software gerichtet ist, sollen die Applikationen aus einem lokalen, gesicherten Netz auf die Middleware zugreifen. Falls das Unternehmen beispielsweise einen öffentlichen Service anbieten will, müssen die Sicherheits- und Rollenthematiken auf dieser Ebene umgesetzt werden. Einerseits wird dadurch der Overhead der Middleware verringert und andererseits werden in den Bereichen der Sicherheit und Berechtigungen keine eigenen Implementierung und Konzepte vorgegeben. Die Option des nachträglichen Hinzufügens eines einfachen Sicherheitskonzeptes in die RTLS-Middleware soll jedoch offen gelassen werden.

Die Schnittstellen der Lokalisierungssysteme, die von der RTLS-Middleware unterstützt werden, sind vorerst unbekannt. Weder die Implementierung bzw. Benutzung der Schnitt-

stelle noch die Syntax und Semantik sind von den unterschiedlichen Lokalisierungssystemen einheitlich. Deswegen muss die RTLS-Middleware für jedes Lokalisierungssystem eine eigene Komponente zur Verfügung stellen, die die Kommunikation mit der jeweiligen individuellen Schnittstelle ermöglicht.

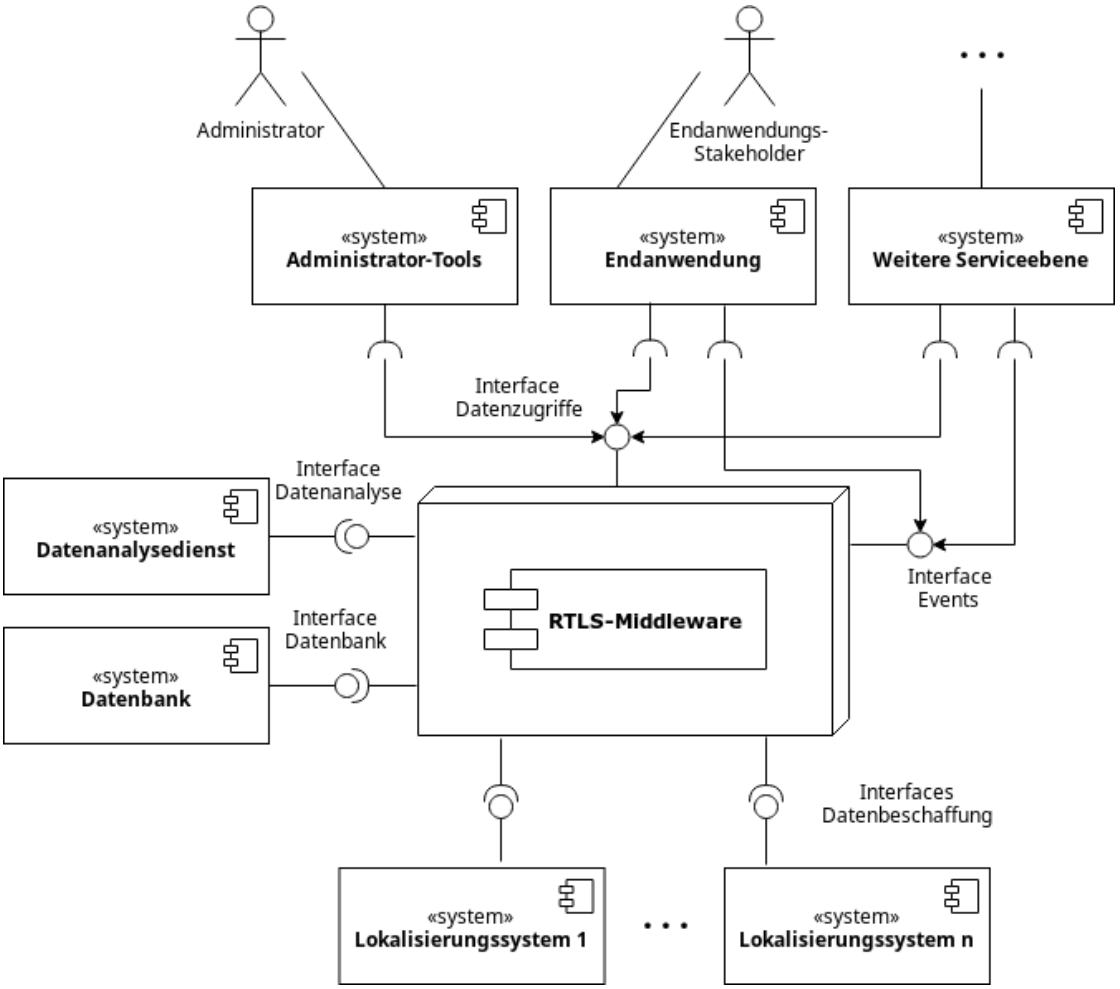

Abbildung 3.9: Komponentendiagramm der externen Systemschnittstellen

Die RTLS-Middleware bietet nach außen insgesamt drei verschiedene Schnittstellen an. Die herkömmlichen Datenabfrage- und Datenmanipulationsoperationen sollen über die oben beschriebene, standardisierte Schnittstelle bei Bedarf selbstständig aufgerufen werden. Die Schnittstelle zu dem Datenanalysedienst und die Schnittstelle für Events sollen eigenständig neue Daten zu dem Zeitpunkt der Entstehung an die jeweiligen Empfänger übermitteln.

Die Semantik aller systemeigenen Schnittstellen soll entwicklerfreundlich dokumentiert und möglichst intuitiv gestaltet werden. Das bereitgestellte Übertragungsformat soll ausschließlich auf JSON (JavaScript Object Notation) basieren. Ein in JSON entwickelter Standard namens Geo-JSON bietet ein spezielles Datenformat für Lokalisierungsdaten, das mit dem geodätischen Referenzsystem „WGS84“ kompatibel ist. Weitere Vorteile von JSON sind die einfache Lesbarkeit, die sehr weit verbreitete Unterstützung in den gängigen Programmiersprachen und dessen kompakte Darstellung.

Abbildung 3.9 zeigt noch mal eine Zusammenfassung der eben genannten Systemschnittstellen als Komponentendiagramm mit der RTLS-Middleware als Black Box (Vergl. [Zörner, 2012]). Das heißt es werden keine internen RTLS-Middleware-Komponenten definiert, da dies bereits Teil des Entwurfs in Kapitel 4 ist.

3.4.6 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen beziehen sich häufig auf mehrere oder alle funktionalen Anforderungen und ergeben sich häufig als notwendiges Beiprodukt der Anforderungsanalyse aus den Kundenwünschen, den technologischen Grenzen oder dem Ziel der kurz- oder langfristigen Arbeitsersparnis. [Rau, 2016] erklärt die im Folgenden aufgelisteten Kategorien für nicht-funktionale Anforderungen.

3.4.6.1 Qualitätsanforderungen

Obwohl es möglich wäre eine Benutzeroberfläche für die visuelle Verfolgung und Kontrolle der Ortspositionen an die Middleware anzuschließen, ist diese Aufgabe kein Teil dieser Arbeit. Die RTLS-Middleware bietet einen eigenständigen, unabhängigen Dienst, weshalb in diesem Projekt kein Fokus auf der Mensch-Maschine-Interaktion liegt. Ein großer Fokus liegt hingegen insgesamt auf der Intersystem-Kommunikation.
Der Standard ISO/IEC 9126 beschreibt verschiedene Qualitätsmerkmale sowie erklärende Teilmerkmale, die in der folgenden Liste berücksichtigt wurden:

Funktionalität

1. Neu eingehende Lokalisierungsdaten sollen schnellstmöglich an der Middleware abrufbar sein.
2. Es soll kein Verlust in der zeitlichen Genauigkeit der Herkunftsdaten entstehen.

3. Die Daten sollen den Ort im globalen Kontext ebenso genau bestimmen können, wie die Herkunftssysteme.
4. Die versprochene maximale Genauigkeit der Herkunftssysteme soll in der RTLS-Middleware berücksichtigt werden.
5. Die Middleware soll auf eine modulare Art und Weise programmiert werden, mit dem Ziel neue Lokalisierungssysteme mit wenig Implementierungsaufwand zukünftig unterstützbar zu machen.
6. Anfangs soll die Middleware kein eigenes Sicherheitskonzept bereitstellen.
7. Es soll die Möglichkeit offen gelassen werden ein Sicherheitskonzept in der Middleware zu implementieren.
8. Es sollen möglichst viele Entwurfsmuster verwendet werden, um die Software verständlich und leicht erweiterbar zu gestalten.
9. Die angebotenen Schnittstellen sollen Standards folgen, um eine unkomplizierte und schnelle Einbindung an andere Systeme zu erlauben.

Zuverlässigkeit

10. Die Ausfallzeit bei Softwareupdates soll gering sein.
11. Auf unerwartete Fehler muss die RTLS-Middleware tolerant reagieren. Sie dürfen keinen Systemabsturz herbeiführen.
12. Ausgegeben und geloggte Fehlermeldungen müssen intuitiv und verständlich sein.
13. Der letzte aktuelle Zustand beim Abschalten des Systems soll beim Starten automatisch wiederhergestellt werden.

Benutzbarkeit

14. Die Semantik der Schnittstellen sollte intuitiv sein, sodass die Middleware einfach zu benutzen und bedienen ist.

Effizienz

15. Die Middleware sollte im eingeschalteten Zustand zu jeder Zeit ansprechbar sein und auf jegliche Anfragen unmittelbar reagieren.

16. Die Middleware sollte ohne Probleme 500 Ortungsgeräte von verschiedenen Lokalisierungssystemen verwalten können.

Wartbarkeit

17. Die Weiterentwicklung und Wartung ist vorgesehen und muss daher besonders zugänglich und verständlich gemacht werden.

3.4.6.2 Technologische Anforderungen

Ein besonderer, maßgeblicher Fokus in Echtzeitsystemen liegt auf der Leistungsfähigkeit, die einen Einfluss auf die Latenz hat, mit der neue Daten übermittelt werden können.

Für eine „echte“ Unterstützung der Echtzeitsysteme in der RTLS-Middleware müssten besondere Protokolle, Systeme und Geräte verwendet werden und die Anwendung müsste auf speziellen Betriebssystemen laufen, wie beschrieben in [Halang and Unger, 2014]. Da bei der Verarbeitung der Daten allerdings immer Ungenauigkeiten und Latenzen auftreten können, ist die Echtzeitdarstellung nie perfekt. In unserem Fall soll der Nutzer konfigurieren können wie präzise die Echtzeitunterstützung durchgeführt werden soll. Eine exakte Abbildung der Echtzeitdaten ist keine Anforderung an die Middleware. Die Lokalisierungsdaten werden periodisch abgefragt. Je geringer die Zeit der Intervalle ist, desto geringer ist die maximale zeitliche Diskrepanz der aktuellen Daten der Middleware zu den Echtzeit-Daten und die Wegverfolgung wird dementsprechend flüssiger. Gleichzeitig wird bei einer höheren Rate an Anfragen ebenfalls der Datendurchsatz erhöht, weswegen der Benutzer der Middleware gegebenenfalls infrastrukturelle Anpassungen machen muss, um dem Durchsatz von Daten gerecht zu werden.

Es folgt eine Zusammenfassung der zuvor beschriebenen und ergänzenden technologischen Anforderungen:

1. Ein Middleware-Administrator soll entscheiden können, welche Abfragerate die Middleware haben soll.

2. Die ausreichenden Betriebssystemeinstellungen und die Infrastruktur muss anhand der individuellen Ansprüche an die Middleware vom Kunden selbst entschieden werden.

3. Die Middleware muss auf eine möglichst effektive Weise hinsichtlich der Verarbeitungsgeschwindigkeit programmiert werden, sodass die Middleware auch auf schwacher Hardware einen hohen Datendurchsatz bewältigen kann.

4. Eine integrierte Datenbank ist für die Middleware ausreichend. Sie muss nicht mit einer externe Datenbank zur Persistierung verbunden werden.

5. Die Middleware muss Verschlüsselungen oder Authentifizierungen von Lokalisierungssystemen unterstützen können.

3.4.6.3 Rechtliche und vertragliche Anforderungen

Das Erheben von personenbezogenen Standortdaten ist nach dem Datenschutzgesetz ohne Kenntnis und Einwilligung verboten. Falls automatisch erzeugte Daten erfasst werden, die Informationen über Personen enthalten, müssen diese Daten anonymisiert werden. Der Zweck der RTLS-Middleware ist nicht die Erhebung und Verfolgung von personenbezogenen Daten und soll daher keine explizite Unterstützung zum Orten von Personen besitzen. Die Verantwortung der rechtmäßigen Nutzung der RTLS-Middleware obliegt dem Benutzer.

Des Weiteren existieren keine vertraglichen Verpflichtungen bei der Erstellung der Middleware.

3.5 Randbedingungen

Randbedingungen im Softwareprojekt dienen der Eingrenzung der Entscheidungsalternativen. Im Softwareprojekt müssen insbesondere in der Planung viele Entscheidungen bezüglich der Architektur getroffen werden. Zum Beispiel ist zu klären welche Programmiersprache, welche Frameworks und welche Hilfssoftware verwendet werden soll. Eine Entscheidung wirkt sich meistens eingrenzend auf die Menge an folgenden Alternativen aus. Es ist daher hilfreich die bedeutsamsten Randbedingungen zu erfassen, um einen besseren Überblick für Folgeentscheidungen zu bekommen. Somit wird ein Rahmen für die Architektur geschaffen. Die Tabellen 3.3 und 3.4 zeigen die Randbedingungen, dokumentiert nach dem Schema von [Zörner, 2012]. Dabei wird eine Unterscheidung der Randbedingungen in technische und organisatorische Randbedingungen vorgenommen.

Randbedingung	Erläuterung
Betriebssystem-unabhängige Anwendung	Die RTLS-Middleware muss eine leicht portierbare Applikation sein, die auf allen gängigen Betriebssystemen läuft.
Mengengerüst Ortungsgeräte	Es wird davon ausgegangen, dass ein Unternehmen selten mehr als ein paar Hundert Ortungsgeräte gleichzeitig überwachen wird.
Server Hardware	Die RTLS-Middleware muss auch auf leichtgewichtiger Hardware eines Servers laufen. Die RTLS-Middleware soll dementsprechend konfigurierbar sein.
Java-Architektur	Die RTLS-Middleware muss in der Standard-Architektur von Lufthansa Industry Solutions implementiert werden, die auf dem Spring-Boot Framework in Java aufbaut.
SQL-Datenbank	Die Standard-Architektur enthält eine relationale Datenbank.

Tabelle 3.3: Technische Randbedingungen

Randbedingung	Erläuterung
Zeitrahmen 3 Monate	Ein lauffähiger Prototyp der RTLS-Middleware soll nach 3 Monaten auf einer Frankfurter Messe vorgestellt werden, der mindestens das Lokalisierungssystem Quuppa Intelligent Locating System™ unterstützt.
Einzelner Entwickler	Vorerst bin ich der einzige Entwickler der Middleware. Als bisher einziger Entwickler war die Verwendung der Architektur von Vorteil, da meine Kenntnisse in Java und SQL gut entwickelt sind.
Java-Kenntnisse	Java-Kenntnisse sind im Unternehmen Lufthansa Industry Solutions weit verbreitet. Die Weiterentwicklung kann daher unproblematisch weitergeführt werden.

Tabelle 3.4: Organisatorische Randbedingungen

Die Kunden von Lokalisierungssystemen wollen grundsätzlich ihre eigenen Use-Cases umsetzen und wollen die Ortsdaten schnell und einfach integrieren können. Für den konkreten Auftrag eines Traktorenherstellers kam die Idee der RTLS-Middleware zustande. Der Auftrag beinhaltete lediglich die Anfertigung eines „Proof of Concept“ ohne vertraglich erwartete Ergebniserbringung. Aufgrund einer vorhergehenden Zusammenarbeit von Lufthansa Industry Solutions mit der Firma Quuppa, die der Anbieter des Lokalisierungssystems Quuppa Intelligent Locating System™ [Quuppa, 2019] ist, war es naheliegend dessen Lokalisierungssystem, als erstes unterstütztes System in die RTLS-Middleware zu integrieren. Das Quuppa Intelligent Locating System™ basiert auf den Technologien BLE und AoA (siehe Abschnitt 2.1).

4 Softwareentwurf

In [Balzert, 2011] wird der Softwareentwurf definiert, als das Entwickeln einer softwaretechnischen Lösung aus den gegebenen Anforderungen an das Softwaresystem, im Sinne einer Softwarearchitektur. Beim Entwerfen müssen eine Vielzahl an Einflussfaktoren, die sich oft gegenseitig beeinflussen, berücksichtigt werden. Außerdem muss bereits der gesamte Lebenszyklus des zu entwickelnden Softwaresystems betrachtet werden. Die Herausforderung beim Erstellen eines Softwareentwurfs besteht in der Gestaltung einer Softwarearchitektur im Hinblick auf die möglichst umfassende Erfüllung der vorher spezifizierten Anforderungen aus Abschnitt 3.4, insbesondere der funktionalen (3.4.1) und nicht-funktionalen Anforderungen (3.4.6) an das Softwaresystem, sowie die Berücksichtigung der Randbedingungen (3.5). Das Ziel des Entwurfs läuft letztlich auf die Lösungsfindung einer multikriteriellen Optimierungsaufgabe hinaus.
Der Entwurf lässt sich in einen Grob- und einen Feinentwurf unterteilen. Der Grobentwurf beschäftigt sich mit der Festlegung einer globalen Systemarchitektur (Abschnitt 4.1). Der Feinentwurf hingegen beschäftigt sich damit wie einzelne Subsysteme und Komponenten im Detail, z.B. mithilfe von Entwurfsmustern, entworfen werden (Abschnitt 4.2).

4.1 Architekturentscheidungen des Grobentwurfs

In [Rau, 2016] sowie in [Balzert, 2011] wird die Bedeutung eines ersten grob durchdachten Entwurfs der Software-Architektur in der Vorbereitungsphase nahegelegt, der Klarheit über die auszuarbeitende Architektur bringen soll. In diesem Schritt werden grundlegende Architekturentscheidungen diskutiert und evaluiert, die großen Einfluss auf den gesamten fortlaufenden Softwareentwicklungsprozess haben. Eine weitreichende Recherche an Frameworks, Bibliotheken, Architekturkonzepten und Technologien, sowie die Überprüfung der eigenen verfügbaren Mittel müssen hierbei einen unabdingbaren, intensiven Arbeitsschritt darstellen und sollte unter keinen Umständen vernachlässigt werden.

4.1.1 Softwaretechnische Infrastruktur

Zuallererst werden in dieser Sektion die grundlegendsten Architekturentscheidungen erklärt, die sich direkt aus den vorher ermittelten Anforderungen ableiten lassen und die den Grundstein für den anschließenden Entwurf legen werden.

Die RTLS-Middleware muss allgemein für mehrere Anwendungen und Services zugänglich sein. Bei der RTLS-Middleware wird es sich, bezüglich der generellen Anwendungsart, daher natürlicherweise um eine Mehrplatzanwendung handeln. Da Mehrplatzanwendungen auf mehreren Computersystemen verteilt implementiert werden, wird eine Entscheidung über die Art der Verteilung erfordert. Die Anforderungen tendieren bezüglich der Verteilungsart stark zu einer service-orientierten Architektur.

Die service-orientierte Architektur (SOA) und Entwicklung ist ein Paradigma, bei dem Softwarekomponenten mit kompakten Schnittstellen erstellt werden und jede Komponente einen diskreten Satz von zusammenhängenden Funktionen ausführt. Mit seiner klar definierten Schnittstelle und seinem Nutzungsvertrag stellt jede Komponente einen Service für andere Softwarekomponenten bereit. Webservices orientieren sich an dieser Architektur. Abbildung 4.1 zeigt ein beispielhaftes Diagramm einer solchen Architektur. Die Schichten innerhalb des Anwendungs-Servers sind nach [Rau, 2016] definiert. In die-

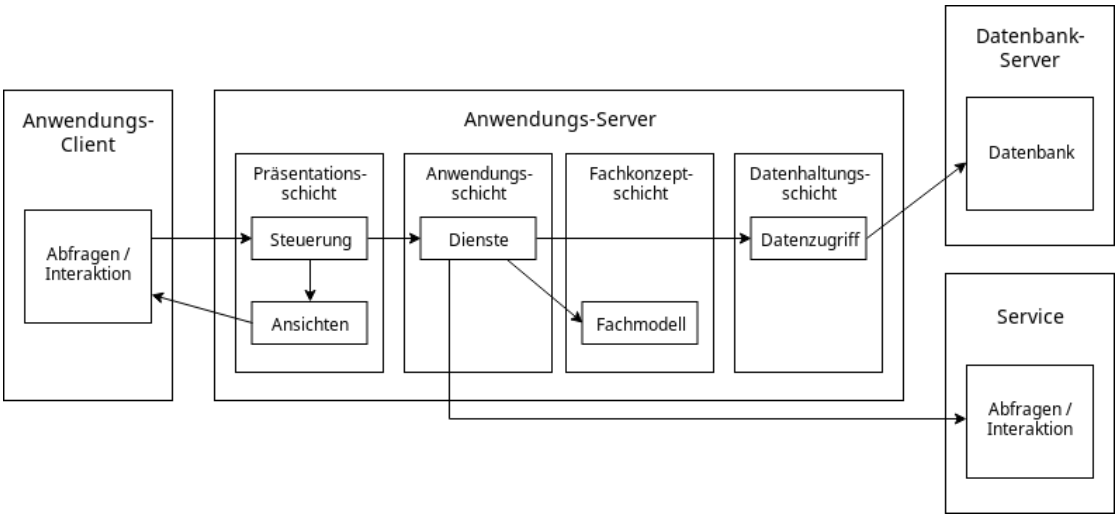

Abbildung 4.1: Beispielhafte Architektur einer Service-orientierten Anwendung

ser Architektur erkennt man die Zentrierung um die Dienste der Anwendungsschicht. Die Dienste interagieren mit allen anderen Schichten des Anwendungs-Servers und kommunizieren ebenfalls mit den externen Diensten (Services). Diese Verknüpfung der Services

untereinander ist der Grundgedanke der service-orientierten Architektur, die auf den Prinzipien der Erweiterbarkeit und der Trennung von Abhängigkeiten basiert.
Die Präsentationsschicht bereiten die Abfrage- und Steuerungselemente der Dienste lediglich als Schnittstelle auf. Die Präsentationsschicht in der RTLS-Middleware soll mithilfe von Remote Application Programming Interfaces (Remote-APIs) realisiert werden.

> „API spezifiziert die Operationen sowie die Ein- und Ausgaben einer Softwarekomponente. Ihr Hauptzweck besteht darin, eine Menge an Funktionen von ihrer Implementierung zu definieren, sodass die Implementierung variieren kann, ohne die Benutzer der Software zu beeinträchtigen.“ [Bloch, 2014]

Die API soll die Daten in einem verständlichen und effizient übermittelbaren Format präsentieren, sodass dessen Funktionen unaufwändig an Softwaresysteme angebunden werden können. Zu einer API gehört deswegen auch eine detaillierte Dokumentation der Schnittstellenfunktionen. Wegen der Trennung zwischen API und Implementierung, ist die Implementierung änderbar und austauschbar, ohne Einfluss auf die Schnittstelle zu haben. Diese Änderbarkeit und Austauschbarkeit der Implementierung ist [Spichale, 2019] zufolge ein wichtiger Vorteil von APIs. Weitere Vorteile sind die Modularisierung und eine lose Kopplung, welche Unabhängigkeit und Stabilität in der Applikation und in dessen Entwicklung zur Folge haben. Die erstellte, modulare Software ist zudem für weitere Projekte wiederverwendbar.

4.1.2 Fragestellungen

In diesem Abschnitt werden Fragestellungen behandelt, die eine besonders wichtige Rolle in den bevorstehenden Entwurfsentscheidungen spielen. Die Fragestellungen und ihre Diskussionen spiegeln die Gedankengänge und Argumentionen einiger Kernproblematiken wider, die im Vorfeld und im Laufe des Projektes eine leitende Rolle spielten. Diese Problematiken werden in diesem Abschnitt weitestgehend versucht zu klären. Einige Entscheidungen sind in dieser Phase des Projektes jedoch noch nicht abzusehen, weshalb in diesen Fällen zumindest der Lösungsraum weiter eingegrenzt werden soll.

1. Auf welcher Ablaufumgebung basiert der Server?

Die RTLS-Middleware ist ein alleinstehender, zentraler und klientenunabhängiger Service, dessen Kommunikation über einen in Java zu realisierenden Server erfolgt. Für

diesen gibt es Anforderungen, unter Anderem bezüglich der Zuverlässigkeit und Portierbarkeit. Aufgrund der Entscheidung, Java als Programmiersprache zu nutzen, profitiert die Software beiläufig von einer plattformübergreifenden Entwicklungs- und Laufzeitumgebung. Bezüglich des Applikationsservers ist eine konkrete Lösung während der Installationsphase auszuwählen. Bestenfalls soll die ausführbare Datei der RTLS-Middleware als eigenständiger Dienst auf Server-Instanzen bereitgestellt werden können.

2. Welches Framework wird für den Server verwendet?

Die Wahl des Frameworks muss vor Beginn der Implementierungsphase getroffen werden, da das Framework offensichtlich direkten Einfluss auf die Implementation hat. Spring, Java EE, und OSGi sind Beispiele für mögliche Alternativen in der Wahl des Frameworks für kommerzielle Serveranwendungen. Als Grundlage der RTLS-Middleware wird das praktische Framework Spring Boot verwendet, mit dessen Hilfe viele große Programmierschemata für kommerzielle Software vereinfacht nutzbar sind. [Walls, 2015] gibt eine tiefgehende Einleitung in das Spring Boot Framework.
Spring Boot simplifiziert und verbessert die Nutzung des bekannten Spring Frameworks, indem es beispielsweise wenig Konfiguration benötigt und insgesamt die Menge an zusätzlich erforderten, verstreuten Ressourcen verringert. Die Entwicklungszeit für herkömmliche Konzepte wird verringert und die Produktivität erhöht sich infolgedessen. Es stellt häufig benötigte Codefragmente (Boilerplate Code) zur Verfügung und bietet einen unterstützenden Rahmen, um die Serverapplikation mit bewährten Erfolgsrezepten sauber und zielführend zu realisieren. Ein HTTP-Server ist außerdem bereits in dem Framework eingebettet. Spring Boot bietet daher ein bereits vollständiges Softwarepaket, welches hinsichtlich einer expliziten Problemlösung lediglich mit anwendungsspezifischen Quellcode gefüllt werden muss.

3. Wo werden die aktuellen Zustände der Lokalisierungsdaten gehalten?

Diese bedeutsame Überlegung entstammt der Kernaufgabe der RTLS-Middleware. Die Zustände der Lokalisierungsdaten sollen frequent abgefragt und unmittelbar übermittelt werden. Wo sollen welche Daten gehalten werden? Allgemein kommen Datenbanken und der virtuelle Prozessspeicher infrage. Persistente relationale oder NoSQL Datenbanken können für die laufende Datenhaltung bzw. transiente Datenverwaltung ausgeschlossen werden, da die Leistungsfähigkeit des Systems, aufgrund der ständigen Schreibe- und Leseoperationen auf den persistenten Speichermedien, leidet.
In-Memory Datenbanken wären folglich die einzig überbleibende Option im Bereich der Datenbanken. Da allerdings keine Ansprüche an die bereitgestellten Dienstleistungen

eines Datenbankmanagementsystem gestellt werden, würde die Verwendung die Architektur unnötig komplex machen. Die Leistungsfähigkeit würde zudem wiederum, durch die zusätzlichen Datenbankfunktionen und aufgrund des notwendigen „Mappings“ von Daten auf die internen Objekte der Anwendung, leiden.
Daher ist es sinnvoll, direkt auf der internen Datenrepräsentation mit Java-Objekten zu arbeiten. Eine Art Cache soll, insbesondere für die Lokalisierungsdaten, angelegt werden, der die Daten im Hauptspeicher verwaltet. Da die RTLS-Middleware historische Daten nicht selbstständig speichert, wird die Datenmenge im Cache nicht stetig und unkontrolliert anwachsen. Welche Entitäten von Daten in diesem Cache gehalten werden sollen, muss im Folgenden noch entschieden werden. Allerdings gilt zu beachten, dass in jedem Fall genügend Arbeitsspeicherressourcen bereitgestellt werden müssen.

4. Wo werden Daten persistiert?

Der aktuelle Datenbestand und insbesondere die Stammdaten dürfen bei wiederholtem Starten der RTLS-Middleware nicht verloren gehen. Die Daten müssen demzufolge in einer Datenbank abgespeichert werden. Eine Datenbank übernimmt jegliche Persistierungsaufgaben. Einerseits muss die Wahl des Datenbanksystems getroffen werden und andererseits muss dessen Benutzung in der Software implementiert werden. Aus Gründen der Leistungsfähigkeit, Kompaktheit und Datensicherung ist eine eigene Serverumgebung für die Datenbank weder gewünscht noch notwendig. Die Ansprüche sprechen stattdessen für eine eingebettete Datenbank. Die Installation und Konfiguration der RTLS-Middleware beim Kunden profitieren ebenfalls davon, indem sie schnell und einfach durchgeführt werden können. Beliebte eingebettete Datenbanken für Java sind H2, HSQLDB, Derby, ObjectDB, Neo4j und OrientDB.

5. Wie werden Daten persistiert?

Da die Daten der RTLS-Middleware strukturiert sind und wir keine enormen Datenmengen abspeichern brauchen, eignet sich eine relationale Datenbank außerordentlich gut. Das Framework „Hibernate“ zum objektrelationalen Abbilden in Java ist zusätzlich in Spring Boot enthalten und wird vollständig für die Implementation der Datenbankzugriffe verwendet. Die möglichen Alternativen für die eingebettete Datenbank grenzen sich, durch die Wahl einer relationalen Datenbank, auf H2, HSQLDB und Derby ein. Das Austauschen zwischen diesen DBMS ist unproblematisch.
Die Datenbank und der Cache stehen in enger Verbindung. Der wesentliche Zweck der Datenbank ist die Sicherung der Daten, während die RTLS-Middleware abgeschaltet

ist. Ihre Hauptaufgabe besteht darin, beim Hochfahren der RTLS-Middleware den Programmzustand vor dem letzten Herunterfahren wiederherzustellen. Der Cache ist ein Puffer-Speicher, der Abbilder von ausgewählten vollständigen Datenbanktabellen enthält. Der Cache soll alle Daten enthalten, die regelmäßig zur Laufzeit abgerufen und verändert werden. Datenanfragen und -manipulationen sollen des Weiteren direkt auf dem Cache durchgeführt werden, falls die Tabellen der betroffenen Daten ebenfalls im Cache angesiedelt sind.
Die Aktualität des Caches muss immer gewährleistet sein. Je mehr Datenmanipulationen auf dem Cache ausgeführt werden, desto größer ist die Differenz zwischen den zwischengespeicherten und den persistierten Daten. Weil die Datenbank jedoch hauptsächlich der Datensicherung dient, muss die Datenbank nicht jederzeit auf dem aktuellen Stand bleiben, sondern muss lediglich beim Herunterfahren garantiert aktualisiert werden.
Um den Datensatz teilweise vor einem Systemabsturz zu schützen, sollen die aktuellen Daten von dem Cache regelmäßig in die Datenbank geschrieben werden. Die Datenbankoperationen können praktischerweise dadurch, genau dann durchgeführt werden, wenn die Systemlast gerade gering ist. Das Operieren auf dem Hauptspeicher mithilfe des Caches und dem damit verbundenen Verzicht auf ständige Lese- und Schreiboperationen vom Speichermedium bringt eine erhebliche Leistungssteigerung mit sich, die zwar abhängig von der installierten Hardware ist, aber zu diesem Zeitpunkt auf mindestens 4 mal schneller geschätzt werden kann. Abbildung 4.2 zeigt die soeben beschriebenen Interaktionen der Datenhaltungskomponenten innerhalb der Programmarchitektur.

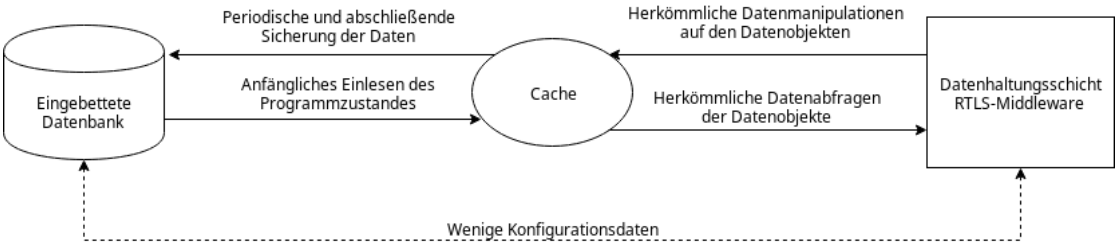

Abbildung 4.2: Architektur der Datenhaltungsebene

5. Wie interagieren die Anwendersysteme mit der RTLS-Middleware?

Die Anwendersysteme werden als Klienten behandelt. Das bedeutet, dass die Systeme selbstständig, je nach Bedarf, mit den Funktionen der RTLS-Middleware interagieren. Wie in Abbildung 3.9 zu erkennen, existieren zwei verschiedene Schnittstellen für die Anwendersysteme. Die Schnittstelle für Datenzugriffe und die Steuerung der RTLS-

Middleware enthält eine Reihe an Funktionen, über die der Klient mit der Middleware kommuniziert. Ein Webservice ist für diese Art der Kommunikation konzipiert. Es gibt die drei typischen Varianten von Webservices namens RPC, SOAP und REST.
Mithilfe der anderen Schnittstelle für Anwendersysteme sollen auftretende Events an die Klienten kommuniziert werden. Die Events werden allerdings zu ungewissen Zeitpunkten ausgelöst, sind jedoch für die Klienten am relevantesten, sobald sie auftreten. Die Events über einen Funktionsaufruf an jeweils einen Webservice der Anwendersysteme zu schicken, würde eine enge Kopplung und Abhängigkeit zwischen Anwendersystemen und der RTLS-Middleware zur Folge haben. Eine bessere Lösung für diese Schnittstelle ist das Konzept des Message Brokers (Abschnitt 2.4). Die RTLS-Middleware schickt in dieser Variante alle Events an einen einzigen bekannten Message Broker. Somit bleibt der Klient auch weiterhin ein Klient und abonniert ausschließlich diejenigen Event-Themen, die für die jeweilige Anwendung interessant sind.
Es gibt verschiedene Produkte und Protokolle für Message Broker. Die gängigsten Protokolle sind generell JMS, STOMP, OpenWire, MQTT und AMQP. Einige Produkte verwenden stattdessen auch ihre eigens entwickelten Protokolle. Da der Message Broker jedoch kundenseitig installiert werden muss, sollen möglichst wenig Einschränkungen bezüglich der Produktwahl gegeben sein. Die RTLS-Middleware soll deswegen auf dem gängigen Protokoll „MQTT“ arbeiten. MQTT ist ein robustes Protokoll, das für Geräte mit wenig Rechenleistung in unzuverlässigen Netzwerken entwickelt wurde. Insbesondere erlaubt MQTT eine 1:n-Kommunikation über sogenannte Topics, die äußerst angemessen für die Umsetzung der Event-Schnittstelle ist. Der Kunde kann sich demnach selbst für eine Brokerlösung entscheiden, insofern dieser MQTT unterstützt.

6. Wie werden die verschiedenen APIs realisiert?

Die zwei Schnittstellen für Anwendersysteme definieren jeweils eine eigene API. Die Schnittstelle für die Datenzugriffe und Steuerung soll als RESTful HTTP-Service implementiert werden. Genauere Informationen zu dem Softwarearchitekturstil REST sind in Abschnitt 2.3 zu finden. Spring enthält Bibliotheken, um REST-APIs innerhalb des Java Quellcodes zu definieren. Die bereitgestellten Daten sollen möglichst vollständig mit der Datenstruktur JSON übermittelt werden. Der sogenannte MIME-Type spiegelt die Art der Daten der jeweiligen REST-API-Funktionen mittels HTTP-Header wider. Das genaue Datenschema einer Operation wird von dem Anbieter der API (in diesem Fall unsere RTLS-Middleware) vorgegeben und ist für den Klienten anhand der API-Dokumentation ersichtlich.
Der MQTT Broker stellt die zweite API dar. Die zusätzlich einzubindende Java-Bibliothek

„Paho“ erlaubt die Kommunikation über das MQTT-Protokoll mit einem Message Broker. Der MQTT Broker funktioniert nach dem Publisher-Subscriber-Konzept. Die Klienten können die Topics abonnieren, auf denen sie Nachrichten erwarten, die für sie relevant sind. Die RTLS-Middleware veröffentlicht die entsprechenden Event-Nachrichten auf diesen Topics, die somit von dem Broker an alle Abonnenten des Topics verteilt werden. Die einzig gültige Operation auf dieser Event-API ist das Abonnieren der spezifizierten Topics. Die Nachrichteninhalte der Events sollen ebenfalls mithilfe von JSON übermittelt werden.

7. Wie werden die APIs dokumentiert?

Die Dokumentation der APIs ist unverzichtbar und sollte ausführlich genug sein, um die Schnittstellenfunktionen intuitiv und benutzerfreundlich zu beschreiben. Für Java existieren einige Hilfsmittel, die die Anfertigung einer Dokumentation erleichtern und beschleunigen können [Spichale, 2019]. Beispielsweise kann eine API mithilfe des Frameworks „apiDoc“, analog zu Javadoc, direkt im Quellcode anhand von Kommentaren dokumentiert werden. Mithilfe von apiDoc kann ein HTML-Dokument aus den Kommentaren erzeugt werden, welches die Dokumentation enthält.
Die Dokumentation der REST-API soll mit der bekannteren Alternative zu apiDoc namens „Swagger“, umgesetzt werden. Swagger erlaubt die API-Spezifikation direkt aus dem Quellcode mittels Java-Annotationen zu generieren. Die Eigenschaften der Schnittstellenfunktionen können automatisch aus dem annotierten Quelltext ausgelesen und in eine vollständige API-Dokumentation übersetzt werden. Swagger erzeugt JSON-Dokumente, die mithilfe von Swagger UI dynamisch gerendert werden können. Die gerenderte Dokumentation kann anschließend über eine URL an der Applikation mittels eines Browsers aufgerufen werden.
Die Dokumentation der Event-API soll eine manuelle Auflistung der unterschiedlichen Topics mit den dazugehörigen Beschreibungen enthalten. Zusätzlich wird jeweils das vorgegebene JSON-Schema mit Beispielen in der Dokumentation protokolliert. Auf welchem Weg die gesamte Dokumentation dem Kunden ausgeliefert werden soll, ist noch nicht entschieden.

8. Wie kommuniziert die RTLS-Middleware mit einem Datenanalysedienst?

Aufgrund der Entscheidung, die RTLS-Middleware als Echtzeitsystem zu implementieren, soll die Middleware zu keiner Zeit mehr als die gegenwärtige Abbildung der Lokalisierungsobjekte enthalten. Für eine Kontrolle der aktuellen Positionen und weiteren

derzeitigen Eigenschaften ist die RTLS-Middleware, ohne der Verwendung des Datenanalysedienstes, ausreichend. Die Schnittstelle zu dem Datenanalysedienst soll daher optional sein und soll bei Bedarf über die Startkonfigurationen geöffnet werden können. Der Datenanalysedienst muss vom Kunden selbstständig entwickelt werden und kann folglich den Anforderungen des Kunden zugeschnitten werden. Der Datenanalysedienst muss lediglich eine Webservice-Funktion zur Verfügung stellen, um Daten der RTLS-Middleware zu empfangen. Die Middleware soll voraussichtlich die neuen Daten bei jeder Iteration der Datenaktualisierung mittels HTTP-POST, fortlaufend über die Webservice-Funktion, an den Datenanalysedienst übermitteln.

4.2 Sichten

[Starke, 2017] empfiehlt die Erarbeitung und Dokumentation der Architektur mithilfe der Darstellung von verschiedenen Sichten. Die Sichten sollen dabei helfen, unterschiedliche Aspekte der Architektur zu beleuchten, weil die Architektur aufgrund ihrer Vielschichtigkeit und Komplexität nicht effektiv mit einer einzigen Darstellung umfassend beschrieben werden kann. Durch die Sichten bekommen die verschiedenen Projektbeteiligten meh-

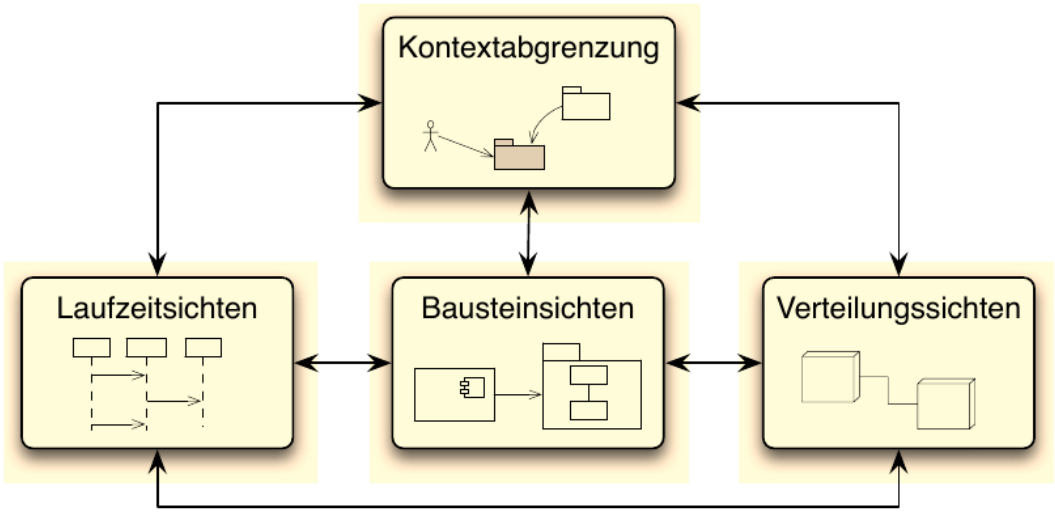

Abbildung 4.3: Vier Arten von Sichten[1]

rere Perspektiven auf das Gesamtsystem und die Darstellungskomplexität wird in Zuge

[1] Quelle: [Starke, 2017]

dessen reduziert. Nach [Starke, 2017] sind die wichtigsten Arten von Sichten die Kontextabgrenzung, Bausteinsichten, Laufzeitsichten und Verteilungssichten. Abbildung 4.3 zeigt die genannten Arten von Sichten und deren Abhängigkeiten und Wechselwirkungen untereinander, die durch die Pfeile zwischen den Sichten symbolisiert sind.

4.2.1 Kontextabgrenzung

Die Kontextabgrenzung beschreibt, wie das eigene System in seine Umgebung eingebettet ist. Die Interaktion mit den Stakeholdern sowie wesentliche Teile der umgebenden Infrastruktur werden klar definiert. In der Kontextabgrenzung werden insbesondere die Schnittstellen zu Nachbarsystemen ausführlich erklärt. Das eigene System wird dabei als Blackbox betrachtet.

Im Abschnitt 3.4.5 wurden im Zuge der Anforderungsanalyse bereits Anforderungen an die Systemschnittstellen definiert. Mithilfe der Anforderungen wurde außerdem ein erstes Modell der Infrastruktur entwickelt, welches Nachbarsysteme und deren Schnittstellen benennt (Abbildung 3.9). In diesem Abschnitt wird dieses Modell verfeinert, an die tatsächlichen Entwurfsentscheidungen angepasst. Außerdem werden die expliziten Lösungen für die Nachbarsysteme benannt.
Wie bereits im Abschnitt 3.4.6.3 der rechtliche und vertragliche Anforderungen erwähnt wurde, besteht eine Zusammenarbeit mit Lufthansa Industry Solutions und Quuppa. Das Quuppa Intelligent Locating System beinhaltet die Location Engine namens „Quuppa Positioning Engine“ (QPE), an dem die Lokalisierungsdaten über eine REST-API angefragt werden können.
Die Abschnitte 4.2.1.1 und 4.2.1.2 beschreiben die Komponenten und die Datenflüsse der Schnittstellen aus Abbildung 4.4. Im Vergleich zur Abbildung 3.9 des ersten Grundrisses der externen Schnittstellen erkennt man, dass für die Datenbank kein externes System notwendig ist. Wie bereits in den Fragestellungen diskutiert, wird stattdessen eine eingebettete Datenbank verwendet. An der Abbildung 4.4 ist zu beachten, dass die mit «systems» markierte Komponenten ein oder mehrere gleichartige Systeme darstellen können.

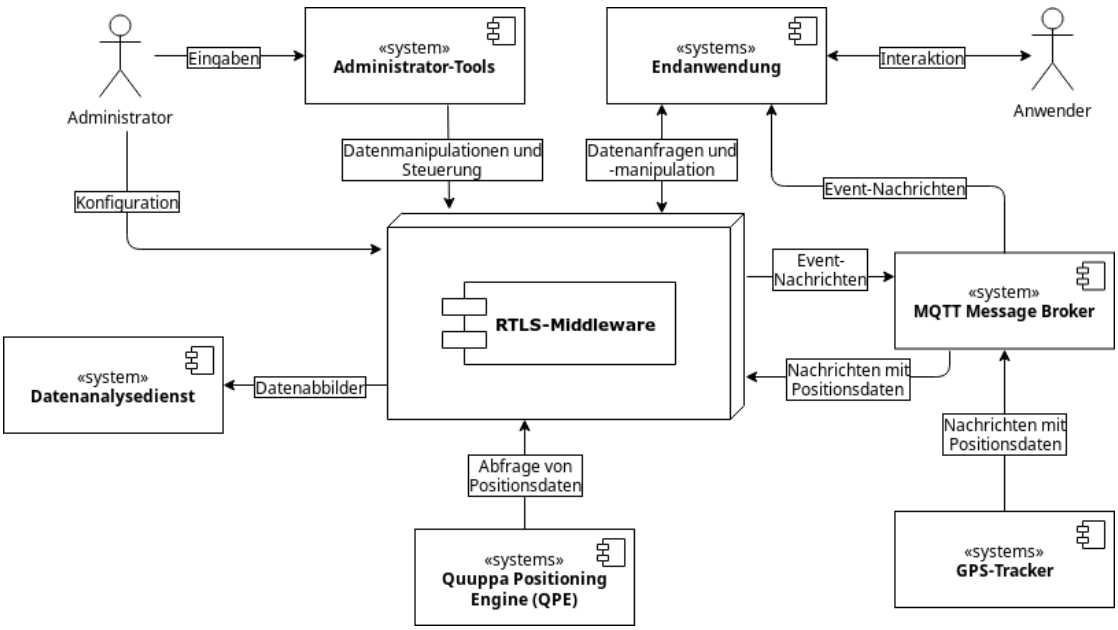

Abbildung 4.4: Kontextabgrenzung

4.2.1.1 Kurzbeschreibung der externen Systeme

Administrator-Tool

Das Administrator-Tool muss lediglich auf dieselbe API zugreifen wie die Endanwendungen. Das Tool muss daher, dafür geeignet sein, REST-Anfragen auszuführen. Mit Swagger UI kann diese Aufgabe über einen Browser erledigt werden. Alternativ kann beispielsweise ein schlichter, benutzerfreundlicher HTTP-Client wie „Postman“ benutzt werden.

Datenanalysedienst

Der Datenanalysedienst ist ein optionales Nachbarsystem, das vom Kunden selbstständig zur Verfügung gestellt werden muss. Die Schnittstelle wurde eingeplant, um die Optionen der Historisierung der Daten und der erweiterten Auswertungen zu ermöglichen. Der Datenanalysedienst stellt womöglich eine HTTP-POST-Funktion bereit, an die neue Daten gesendet werden können.

GPS-Tracker Die GPS-Tracker sind neben den QPEs eine weitere Art der einzubindenden Lokalisierungssysteme. Ein GPS-Tracker ist ein Ortungsgerät, welches zu benutzerdefinierten Zeitpunkten eine Nachricht mit Positionsdaten an einen Server schickt. Die GPS-Tracker können sehr unterschiedlich sein und benötigen daher eine Zwischeninstanz, um Daten über MQTT verschicken zu können. Ein Beispiel für einen solchen Aufbau wäre unsere Testumgebung. Dort wurden Tracker verwendet, die über LTE (Long Term Evolution) ihre Daten an einen Cloud-Server schicken. Daraufhin werden die Daten mithilfe von Node-RED an einer API angefragt, formatiert und an den Message Broker gesendet.

Endanwendung Eine Endanwendung ist eine beliebige Anwendung oder ein Service, der die herkömmlichen Nutzerdienstleistungen der RTLS-Middleware in Anspruch nimmt.

MQTT Message Broker Der Message Broker kann ein beliebiges Produkt sein, welches das MQTT-Protokoll unterstützt. Der Message Broker wird 1) zum Bereitstellen von Lokalisierungsdaten an die RTLS-Middleware und 2) dem Weiterleiten von Events an die Endanwendungen gebraucht. Es kann theoretisch pro Aufgabe auch jeweils ein separater Message Broker verwendet werden.

Quuppa Positioning Engine Eine QPE kann auf einem normalen Webserver eingerichtet werden. Sie stellt eine REST-API zur Verfügung, anhand dessen alle erdenklichen Informationen des Quuppa Intelligent Locating Systems angefragt werden können. In unserem Kontext sind nur ausgewählte Funktionen und Informationen des Systems interessant.

4.2.1.2 Kurzbeschreibung der externen Schnittstellen

Konfiguration	Der Administrator bearbeitet die Konfigurationsdateien der RTLS-Middleware, um ihr Verhalten den Wünschen der anderen Stakeholder anzupassen.
Datenmanipulation/ Steuerung	Der Administrator bearbeitet die Daten und das Verhalten der RTLS-Middleware mithilfe von Befehlseingaben an dem Administrator-Tool. Beispielsweise kann der Administrator über das Tool neue Verbindungen anlegen oder auch andere Stammdaten pflegen.
Datenanfragen/ -manipulation	Eine Endanwendung benötigt typischerweise bestimmte Informationen der RTLS-Middleware, die sie über die Remote-API bei Bedarf anfragen kann. Über die gleiche Schnittstelle soll der Anwendung ebenfalls gestattet sein, Datenmanipulationen vorzunehmen.
Event-Nachrichten	Die RTLS-Middleware löst unter Umständen bestimmte Events aus, die in Nachrichten verpackt an ein Topic des Message Brokers gesendet werden. Der Message Broker verteilt die Event-Nachricht an die Endanwendungen, die dieses Topic abonniert haben.
Nachrichten mit Positionsdaten	Ein GPS-Tracker schickt seine Positionsdaten in frei wählbaren Zeitintervallen an den Message Broker, welche direkt an die RTLS-Middleware geleitet werden. Die Nachricht muss in einem vordefinierten Schema des JSON-Formats angelegt werden, ansonsten sie wird verworfen.
Abfrage von Positionsdaten	Die Daten der QPE müssen selbstständig von der RTLS-Middleware an der REST-API angefragt werden. Die für dieses Projekt relevanten Daten sind insbesondere die Position, Richtung und Geschwindigkeit der Geräte, welche regelmäßig, in kleinen periodischen Abständen, aktualisiert werden sollen.

Datenabbilder Ein gesamtes Abbild aller Lokalisierungsdaten, die in der RTLS-Middleware vereint werden, sollen dem Datenanalysedienst in periodischen Abständen mitgeteilt werden. Dieser kann daraufhin künftig den zeitlichen Verlauf und Bewegungen der georteten Objekte analysieren.

Eine Verbindung zu einem GPS-Tracker wird indirekt über die Netzwerkadresse des entsprechenden Message Brokers hergestellt, indem das passende Topic abonniert wird. Eine Verbindung zur QPE wird hingegen direkt über dessen URL im Netzwerk hergestellt. Weitere Query-Parameter können in jeder Anfrage an die API übergeben werden, um nach Kriterien zu filtern oder das Schema der Antwort anzupassen. Die RTLS-Middleware interagiert vollständig automatisiert mit den Verbindungen der Klientenschnittstellen, nachdem deren Verbindungsinformationen der RTLS-Middleware hinzugefügt und aktiviert wurden. Wie aus den Anforderungen hervorgeht, sollen die QPEs, GPS-Tracker und zukünftig einzubindende Lokalisierungssysteme trotz der unterschiedlichen Art ihrer Datenbereitstellung, möglichst analog in den Datenverarbeitungszyklus der RTLS-Middleware integriert werden.

4.2.2 Verteilungssichten

Zur Abbildung der Verteilung der verschiedenen Hardwarekomponenten auf die Ablaufumgebung wird die Verteilungssicht als beschreibendes Medium herangezogen. Zusätzlich wird in dieser Sicht die Verteilung der Softwarebausteine auf die Hardware und die technische Kommunikation zwischen den Komponenten dargestellt.

Die Verteilungssicht unterstützt die Ausprägung der entworfenen Kontextabgrenzung 4.2.1. Abhängig von ihrer Relevanz können Leistungsdaten und Parameter der beteiligten Elemente, wie etwa Speichergrößen, Kapazitäten oder Mengengerüste, dargestellt werden und externe System, z.B. Betriebssysteme, mit in die Verteilungssicht aufgenommen werden.

Die RTLS-Middleware wurde auf einem Rechner mit Microsoft Windows 10 entwickelt. Aufgrund der Nutzung der Java Virtual Machine ist die RTLS-Middleware auf alle Betriebssysteme portierbar. Die Anforderung der Portierbarkeit wurde ausgewählt, damit

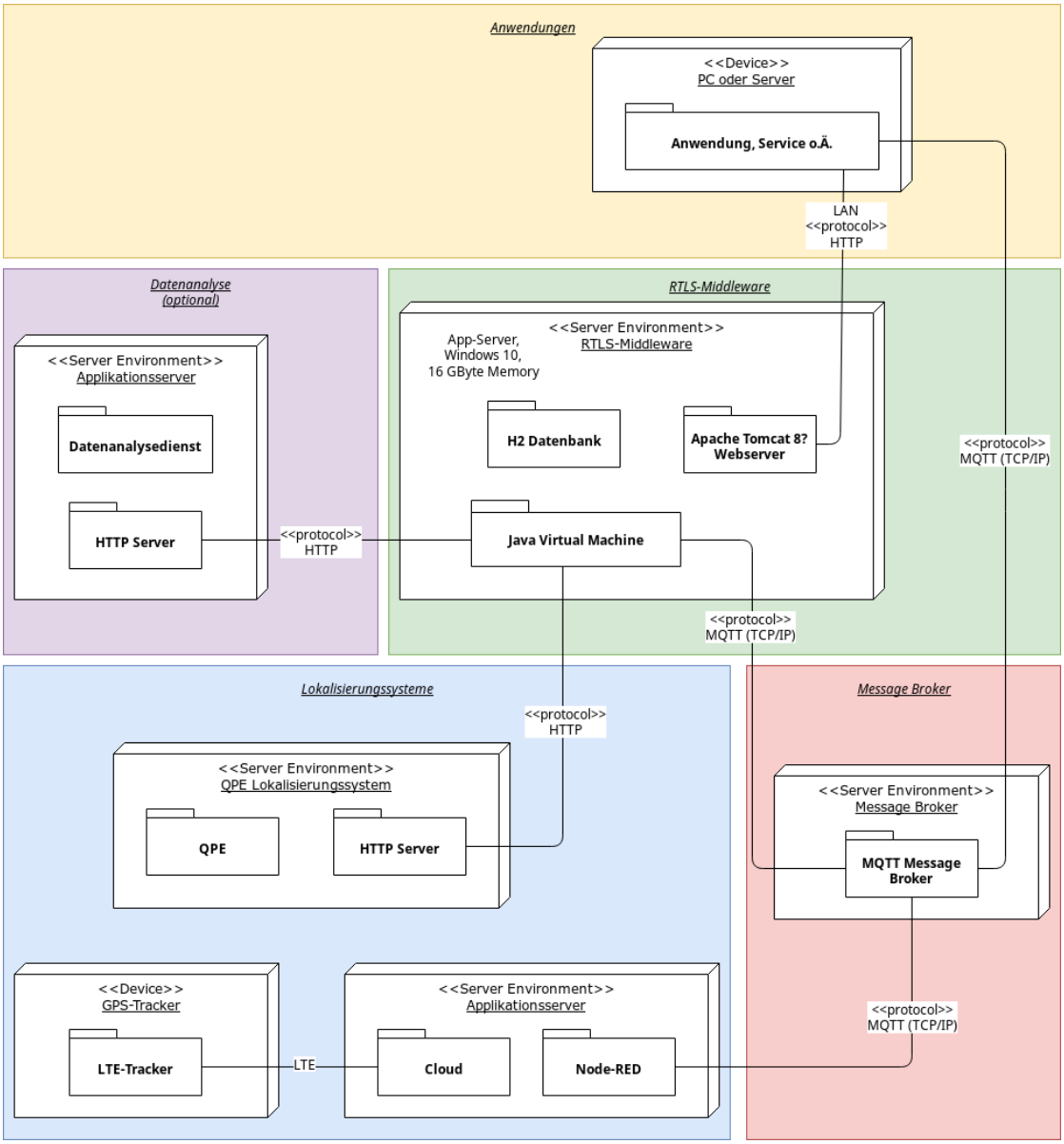

Abbildung 4.5: Verteilungssicht im Stil des UML Einsatzdiagramms

der Kunde die Software besonders einfach in seine bestehende Infrastruktur integrieren kann, ohne selbst irgendwelche Voraussetzungen erfüllen oder große Anpassungen machen zu müssen. Die Portierung könnte dennoch zu unerwarteten und ungewollten Nebenwirkungen führen, weshalb die Software vor der Veröffentlichung auf allen gängigen Betriebssystemen getestet werden muss.
Aufgrund der voraussichtlich großen Menge an Objekten im Cache der RTLS-Middleware,

sollte die Größe des Hauptspeichers nicht zu knapp gewählt werden. Schätzungsweise sollten 8 GByte freier RAM in jedem Fall für den Prozess ausreichend sein. Daher wird empfohlen, dass das Gerät mindestens über 16 GByte RAM verfügt.
Wie an Abbildung 4.5 zu erkennen, sollen alle direkten Verbindungen zur RTLS-Middleware aus Sicherheitsgründen innerhalb des lokalen Firmennetzes verlaufen, müssen aber nicht zwangsweise auf unterschiedliche Geräte verteilt werden. Die Serverumgebungen (Server Environment) können somit entweder beliebig im Netzwerk verteilt oder auch auf einem Gerät installiert werden. Das bedeutet, dass die QPE Lokalisierungssysteme, der Message Broker, der Datenanalysedienst und die RTLS-Middleware auf derselben Serverumgebung arbeiten könnten. Serveranwendungen, die auf die RTLS-Middleware zugreifen, können im Grunde genommen ebenfalls auf demselben Gerät wie die Middleware installiert werden. Allerdings ist vorgesehen, dass diese Anwendungen auf jeweils auf einem eigenen Gerät installiert werden, da sich dadurch die Ausfallsicherheit erhöht und die Last verteilt wird.

Die Verteilung der Anwendungen auf mehrere Geräte kann ein paar weitere Vorteile mit sich bringen, die bei der Integration der RTLS-Middleware in die Infrastruktur berücksichtigt werden sollten. Beispielsweise ist ein verteiltes System besser skalierbar, kann echte Nebenläufigkeit realisieren und erlaubt eine einfache Verteilung der Last.

4.2.3 Bausteinsichten

In der Bausteinsicht werden Aufgaben auf Komponenten und Softwarebausteine des Systems verteilt. Die Bausteine trennen das System anhand von Abhängigkeiten und Anforderungen in modulare Einheiten. Das Ziel ist die Reduzierung der Komplexität des Gesamtsystems, indem es stufenweise in mehreren Ebenen in kleine, leicht verständliche Einheiten heruntergebrochen wird. Die Bausteinsicht verfolgt demzufolge die Idee des Teile-und-herrsche-Prinzips, die besagt, dass man ein Problem beherrschbar machen und lösen kann, wenn man es in kleinere Teilprobleme zerlegt, bis diese gelöst sind und sie anschließend wieder zusammenfügt [Goll, 2018]. Die Bausteine werden in den einzelnen Ebenen fortlaufend verfeinert. Dabei ist eine sinnvolle Trennung und Definition der Bausteine essenziell, sodass sich ein Baustein bestenfalls nur mit einem einzigen Teilproblem beschäftigt. Die Zerlegung in Teilprobleme ist effektiv, wenn die Interaktion der Bausteine untereinander gering und der innere Zusammenhalt der Bausteine stark ist. Anders formuliert, versuchen wir die Kopplung zu minimieren und die Kohäsion zu maximieren.

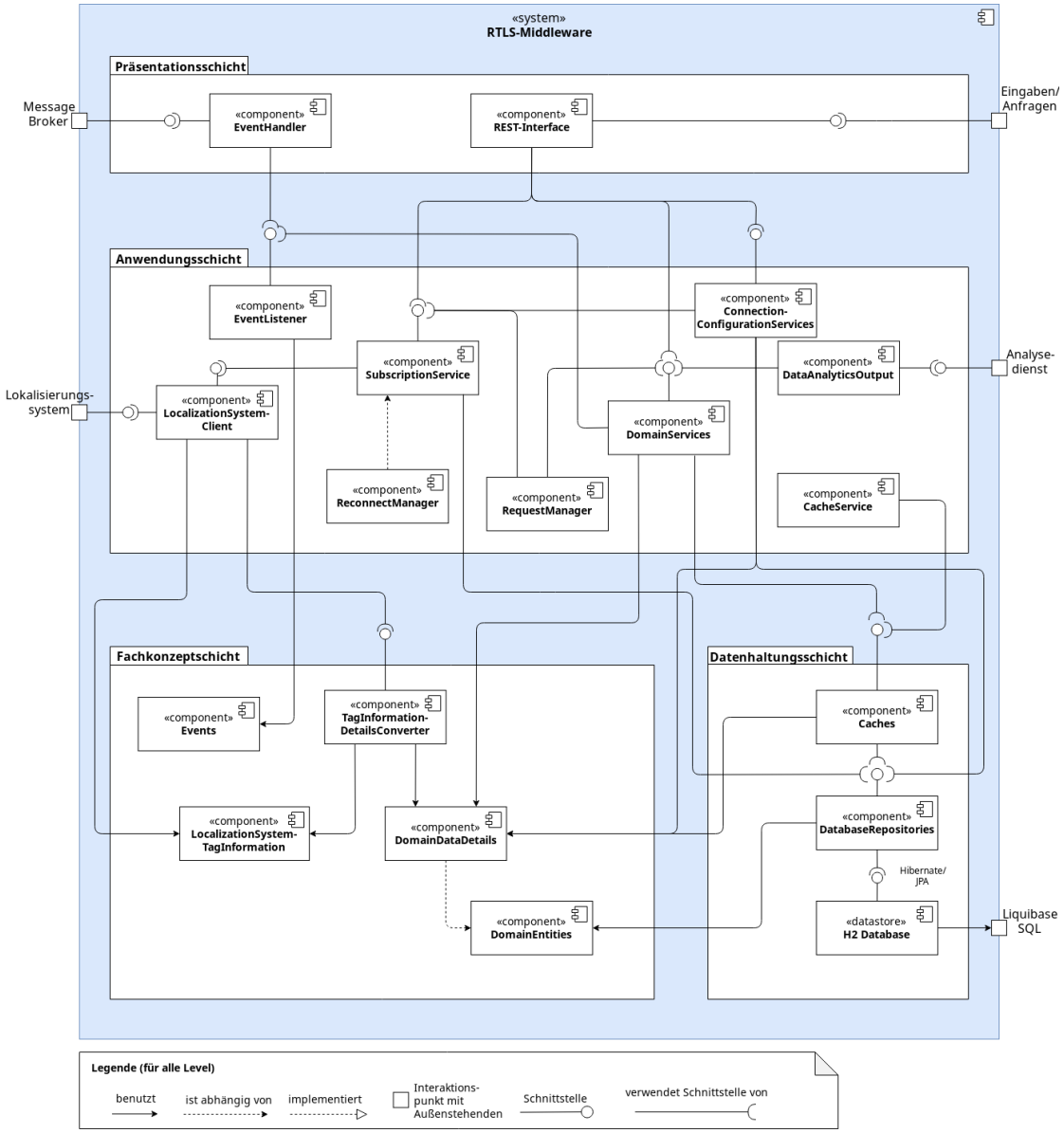

Abbildung 4.6: Bausteinsicht als UML-Komponentendiagramm

Aus den fein ausgearbeiteten Bausteinen lassen sich Arbeitspakete mühelos ableiten.

Nach [Starke, 2017] soll die Bausteinsicht folgende Fragen beantworten:

- Aus welchen Komponenten, Paketen, Klassen, Subsystemen oder Partitionen (Bausteinen) besteht das System?
- Welche Abhängigkeiten bestehen zwischen diesen Bausteinen? Welcher Baustein muss welche Schnittstelle(n) implementieren?
- Welche Bausteine müssen Sie implementieren, konfigurieren oder kaufen, um die gewünschten Anforderungen zu erfüllen?

Die Bausteinsicht beschreibt den internen Aufbau der RTLS-Middleware, vergleichbar mit dem Anwendungs-Server in Abbildung 4.1, und dessen Schnittstellen zwischen Komponenten sowie von Komponenten zu externen Systemen. Abbildung 4.6 zeigt das Komponentendiagramm der gesamten Bausteinsicht. Die Software wird in dieser ersten Ebene in seine Hauptbestandteile zerlegt. Die Interaktionen der Komponenten verbinden die Einzelteile wieder zu dem Gesamtsystem.

Ein komplexes Softwaresystem kann viele Komponenten und Verbindungen aufweisen. Der Übersichtlichkeit halber kann es sich daher anbieten die Komponenten mithilfe einer Klassifizierung logisch zu trennen. Für das Projekt der RTLS-Middleware bot sich die Trennung in die 4 Infrastrukturschichten an. Durch die Darstellung mehrerer Ebenen kann die Komplexität der Gesamtansicht ebenso reduziert werden. Einige große, aber nicht alle Komponenten, werden im Folgenden durch Subkomponenten weiter verfeinert. Das Verfeinern ist hilfreich, wenn die Komponente mehrere unterschiedliche Aufgaben verfolgt oder verschiedene Inhalte besitzt.
Da Komponenten nicht selbsterklärend sind, ist eine Beschreibung je Komponente absolut unverzichtbar. Beginnend mit den Komponenten der Datenhaltung werden die Komponenten nacheinander, in Richtung des Datenstroms zum Anwender d.h. zur Präsentationsschicht, erklärt.

4.2.3.1 Datenhaltungskomponenten

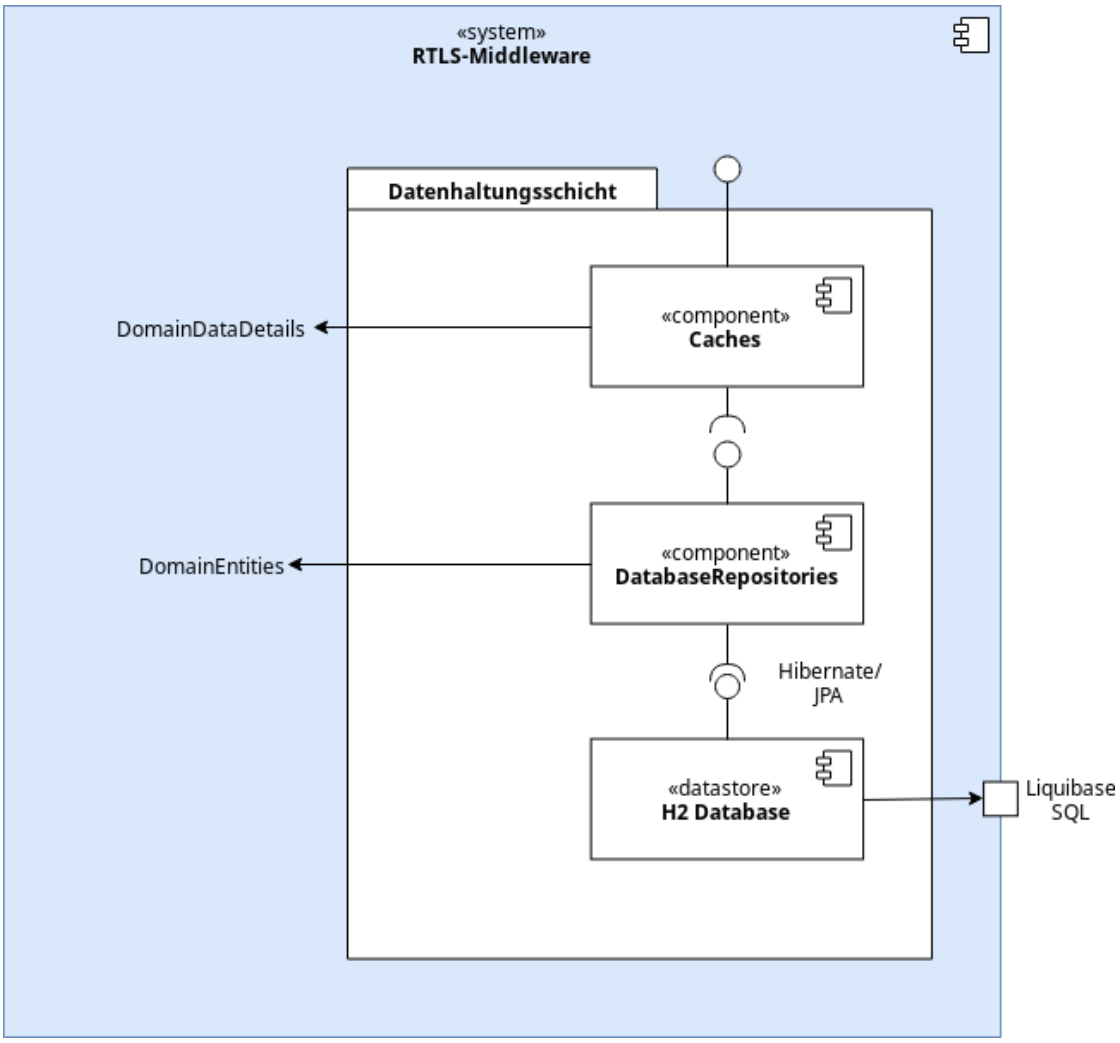

Abbildung 4.7: Datenhaltungsschicht

Die Komponenten dieser Ebene sind für die Speicherung und Bereitstellung der Daten zuständig. Die Daten werden in der Ebene gehalten und von den oberen Schichten aufgerufen. Die Datenhaltungsschicht ist daher in der Darstellung neben der Fachkonzeptschicht die unterste Schicht. Die oberen Schichten fragen die Daten je nach Bedarf selbstständig an dieser Schicht ab, wodurch die Daten nach oben fließen. Dadurch entsteht eine Art Pipeline aus Schnittstellen. Eine Schicht interagiert immer nur mit einer oberen Schicht, indem sie Schnittstellen zur Verfügung stellt und selbst keine Schnittstellen der oberen Schicht aufrufen sollte.

H2 Database

Die H2 Datenbank ist ein relationales SQL-Datenbankmanagementsystem (DBMS), welche direkt in eine Java-Anwendung eingebettet werden kann. Datenbankoperationen können mittels Java über *JDBC* aufgerufen werden. Das *Hibernate* Framework, das auf *JPA* aufbaut, erlaubt die objektrelationale Abbildung, mit dessen Hilfe die Datensätze als Java-Objekte abgebildet werden können. Der Programmierer muss somit die Umgebung der objektorientierten Programmierung nicht verlassen.
Das Datenbankschema wird in einer SQL-Datei definiert, die von *Liquibase* verwendet wird. Liquibase ist eine datenbankunabhängige Bibliothek zur Verfolgung, Verwaltung und Anwendung von Datenbankschemaänderungen und ist besonders nützlich in agilen Softwareprojekten, in denen sich das Datenbankschema möglicherweise häufig ändern kann. Liquibase besitzt einen eigenen SQL-Dialekt, der in den Dialekt des tatsächlichen unterliegenden DBMS übersetzt wird. Liquibase bietet eine leichte Austauschbarkeit des Datenbanksystems sowie eine einfache Nachverfolgung etwaiger Änderungen des Datenbankschemas und ist damit während der Entwicklungsphase eine wertvolle Unterstützung.

Database Repositories

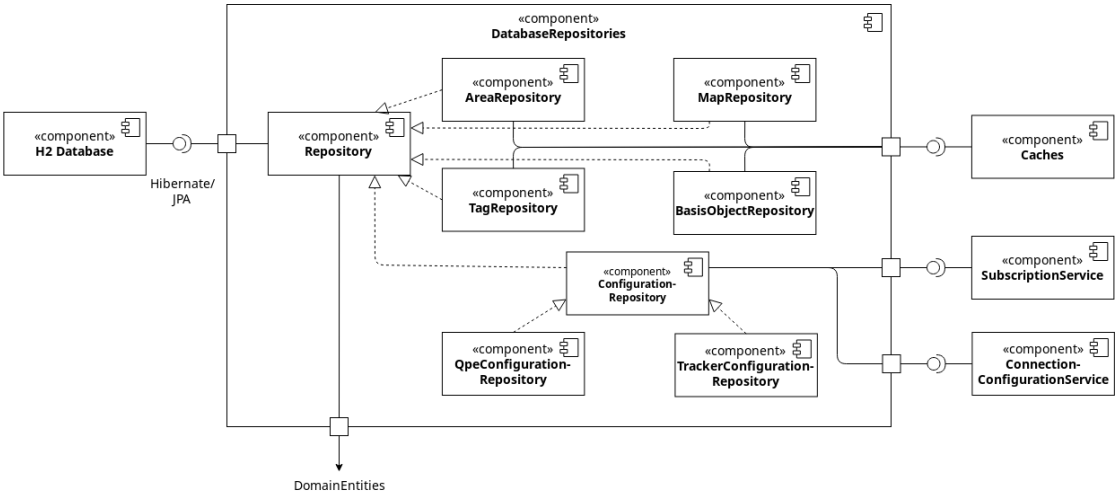

Abbildung 4.8: Datenlager-Komponente

Ein *Repository* bildet in JPA und Hibernate ein Datendepot der Datenbank ab. Im Kontext der relationalen Datenbanken repräsentiert ein Repository schlichtweg eine Datenbanktabelle. Je Tabelle existiert ein Repository, das eine entsprechende Entity-Klasse

(aus *DomainEntities*) verwendet, die die Attributwerte der Tabelle als Java-Variablen abbildet. Die Objekte dieser Klassen repräsentieren einen Datensatz. Ein Repository definiert Java-Methoden, die beim Aufruf in SQL-Operationen umgewandelt werden. Dadurch kann mit Hibernate und den Repositories in einer vollkommen objektorientierten Weise auf die Datenbank zugegriffen werden.
Die ortungsbezogenen Daten in den Repositories *Area-*, *Map-*, *Tag-* und *BasisObject-Repository* werden (wie in Abschnitt 4.1.2 beschrieben) gänzlich in die Caches geladen. Lediglich die Caches interagieren mit diesen Repositories.
Die *ConfigurationRepositories* interagieren hingegen direkt mit Komponenten aus der Anwendungsschicht. Die Verbindungskonfigurationen müssen einmal bei Systemstart geladen und aktiviert werden. Danach treten selten Veränderungen daran auf. Die voraussichtlich häufigste Veränderung auf diesem Datenbestand wird das Hinzufügen einer neuen Verbindungseinstellung sein. Die neue Konfiguration kann sofort aktiviert werden und in die Datenbank gespeichert werden. Ansonsten werden die Daten zur Laufzeit nicht für funktionelle Aufgaben der Middleware benötigt, weshalb ein Cache nicht notwendig ist. Da die Konfigurationen der Verbindungseinstellungen sich zwischen den Lokalisierungssystemen im gesamten Aufbau stark unterscheiden, benötigt jedes System eine eigene Datenbanktabelle, um die Konfigurationen zu speichern.

Caches

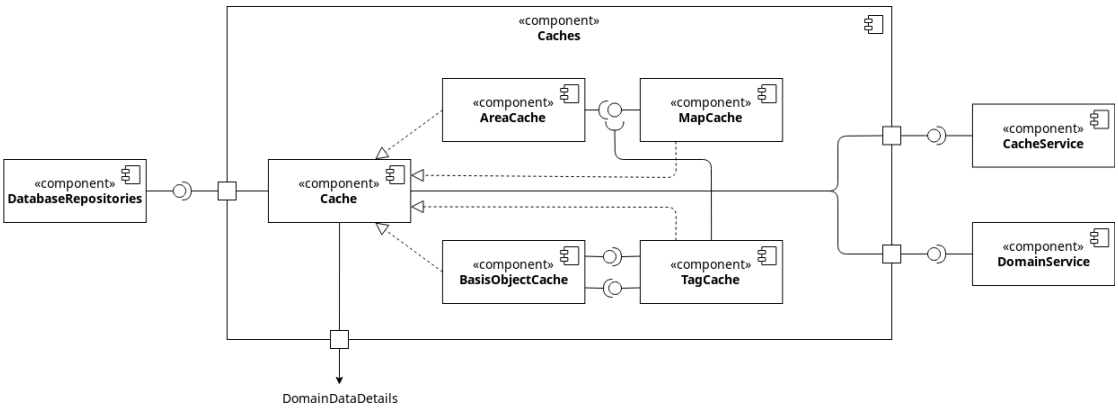

Abbildung 4.9: Cache-Komponente

Jeder *Cache* hält den Datenbestand des ihn betreffenden Datentyps. Jede Entität des fachlichen Datenmodells (Abschnitt 3.4.3) besitzt ein Attribut mit einer eindeutigen Nummer zur Identifikation. Dieses Attribut ist der Primärschlüssel und wird zum Refe-

renzieren zwischen den Tabellen verwendet.
Die Identifikationsnummer spielt zudem eine Rolle für die Datenübertragung der Präsentationsschicht nach außen. Anstatt die referenzierten Objekt mittels JSON in dem angefragten Objekt zu schachteln, wird die entsprechende ID für das referenzierte Objekt eingetragen. Falls die Informationen zu dem Objekt mit der ID gewünscht werden, kann das Objekt mittels ebendieser ID anhand einer separaten Anfrage an die entsprechende Ressource abgerufen werden. Diese Lösung beschränkt den Datendurchsatz auf das Nötigste, wodurch das Netzwerk entlastet wird.
Die eben genannte Entwurfsentscheidung ist für das Design der Chaches äußerst relevant, denn ein Objekt wird nahezu immer über dessen ID angefragt. Das Objekt soll im Datenbestand daher mit so wenig Aufwand wie möglich anhand seiner ID aufzufinden sein. Die Caches für die Maps und die Tags müssen allerdings leicht anders aufgebaut sein, weil diese Daten aus den Lokalisierungssystemen entspringen und dort bereits eine ID haben, die unserem System vorerst fremd ist. Um neue Daten mit systemfremder ID den systemeigenen Daten zuordnen zu können, müssen die eigenen Daten die Ursprungs-ID beibehalten und ebenfalls über diese ID effizient abrufbar sein.
Die Schnittstellen zwischen den implementierten Caches spiegeln die Referenzen der enthaltenen Daten wider. Sie werden alleinig für die Zuordnung der Referenzen von neu hinzugefügten Objekten benötigt. Sobald ein Objekt mit Referenzen zu IDs dem Cache hinzugefügt wird, werden die ID-Referenzen in die Java-Objektreferenzen übersetzt, indem die Objekte in den anderen Caches anhand ihrer ID angefragt und anschließend referenziert werden.

4.2.3.2 Fachkonzeptkomponenten

Die Fachkonzeptkomponenten bilden den fachlichen Rahmen der Software. Sie definieren hauptsächlich Container, die die Daten in strukturierter Form beinhalten und ein Medium bieten, um die Daten innerhalb der Anwendung zu transportieren und kommunizieren.

Domain Entites

Die *DomainEntites* orientieren sich an dem fachlichen Datenmodell. Sie stellen die Datenbanktabellen als Java-Klassen dar, indem beispielsweise die Datentypen automatisch konvertiert werden, die Fremdschlüssel als Java-Referenzen abgebildet und Constraint-Bedingungen durch Hibernate geprüft werden. Die objektrelationale Abbildung wird mit

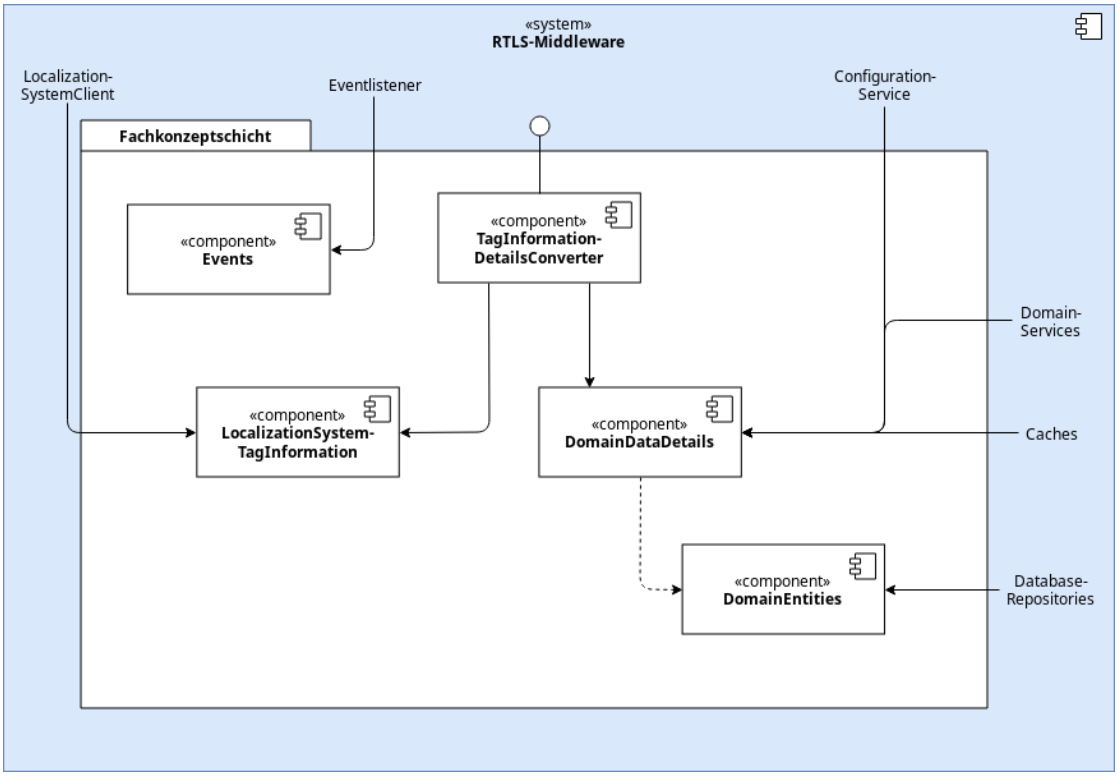

Abbildung 4.10: Fachkonzeptschicht

Java-Annotationen realisiert. Der Entwickler muss allerdings sicherstellen, dass das Datenbankschema und die Entitäten kompatibel sind. Ansonsten wird die Software nicht ordnungsgemäß funktionieren. Die Gültigkeit der Entitäten kann von Hibernate überprüft werden.
Änderungen an einem Entitäts-Objekt, das mit der Datenbank verbunden ist, resultieren neben der Änderung des Java-Objektes zusätzlich automatisch in einer Änderung des Datenbanktupels. Deswegen müssen Entitäten mit Vorsicht behandelt werden. Wenn die Daten verarbeitet werden sollen, ist es ratsam die Daten in ein Objekt zu kopieren, das unabhängig von der Datenbank ist (siehe *Domain Data Details*).

Domain Data Details

Die *Data Details* sind die Datencontainer, auf denen die Anwendung üblicherweise operiert. Ein *Details*-Objekt soll alle notwendigen Daten des entsprechenden Entitäts-Objektes beinhalten und kann außerdem dazu benutzt werden weitere relevante Daten aus Quellen mannigfaltiger Art zu pflegen, wenn sie logisch zu der jeweiligen Klasse gehören. Das

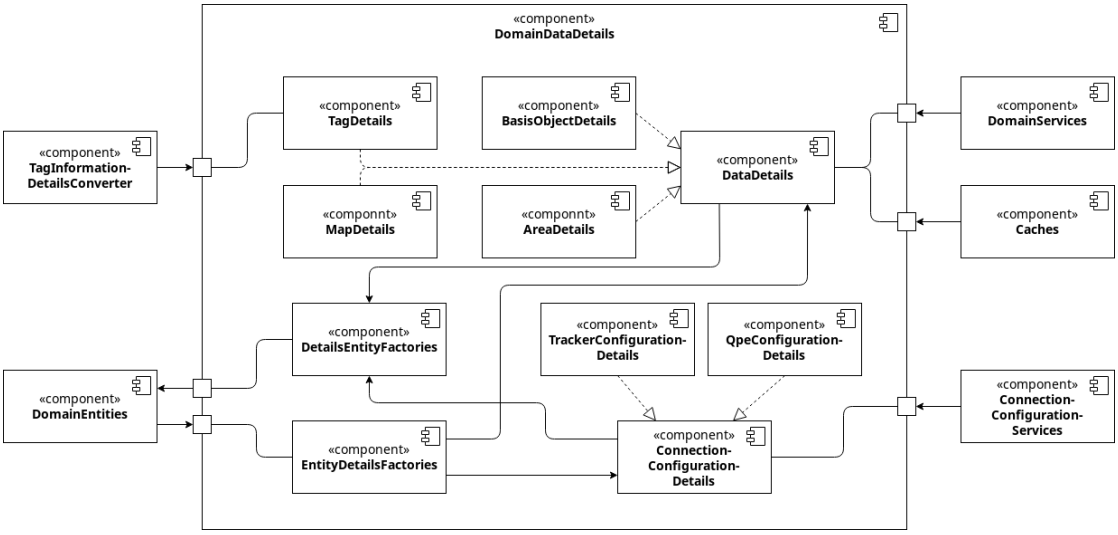

Abbildung 4.11: Java-Datencontainer

Entwurfsmuster der Fabrikmethode [Eilebracht and Starke, 2013] wird genutzt, um in der entsprechenden *EntityDetailsFactory* aus einem Entität-Objekt ein Details-Objekt zu generieren. Dergleichen kann ein Details-Objekt persistiert werden, indem im Vorfeld in der entsprechenden *DetailsEntityFactory* ein Entität-Objekt aus ihm erzeugt wird.
Die Ausprägungen der Data Details werden in den jeweiligen Caches angesammelt. Über die *DomainServices* können an den Caches einzeln Details hinzugefügt, gelöscht und Details bearbeitet werden.
Im Übrigen gibt es für jede Verbindungskonfiguration eines Lokalisierungssystem eine eigene Details-Klasse. Je Klasse gibt es in den *ConnectionConfigurationServices* einen Dienst, der die Verbindungseinstellungen verwaltet. Änderungen an den Verbindungseinstellungen werden nicht zwischengespeichert, sondern wirken sich direkt auf die laufenden Verbindungen aus. Die Änderungen werden danach direkt in eine Entität konvertiert und in die Datenbank gespeichert.

Localization System Tag Information

Jedes Lokalisierungssystem besitzt ein eigenes Datenschema, um die Lokalisierungsdaten zu übermitteln. Damit die RTLS-Middleware die Semantik der Daten verstehen kann, wird für jedes Schema eine Hilfskomponente angelegt, die die Daten in ein Java-Objekt gießen. Diese *TagInformations* werden in den *LocalizationSystemClients* benutzt, da sie an den Klientenschnittstellen zu den Lokalisierungssystemen anfallen.

Tag Information Details Converter

Diese Komponente ist für die Konvertierung der Repräsentationen der Tag-Informationen eines Lokalisierungssystems (*LocalizationSystemTagInformation*) in das einheitliche Schema der *TagDetails* zuständig. Da nicht jedes Lokalisierungssystem den gleichen Informationsgehalt besitzt, dürfen *TagDetails*, bis auf eine kleine Menge an Pflichtfeldern, lückenhaft sein. Dadurch kann jede Ortungsinformation, unabhängig von dessen Ursprung, gleich behandelt werden.

Events

Events sind im Kontext der Anwendung komplett generisch. Es sollen benutzerdefinierte *Events* erzeugt werden können, die potenziell in jeglichem Prozessschritt der Anwendungsschicht auslösbar sein sollen. Sobald ein *Event* ausgelöst wird, wird der entsprechende Datencontainer bzw. das Java-Objekt mit Informationen über das aufgetretene Ereignis befüllt und dem *Eventlistener* mitgeteilt.

4.2.3.3 Anwendungskomponenten

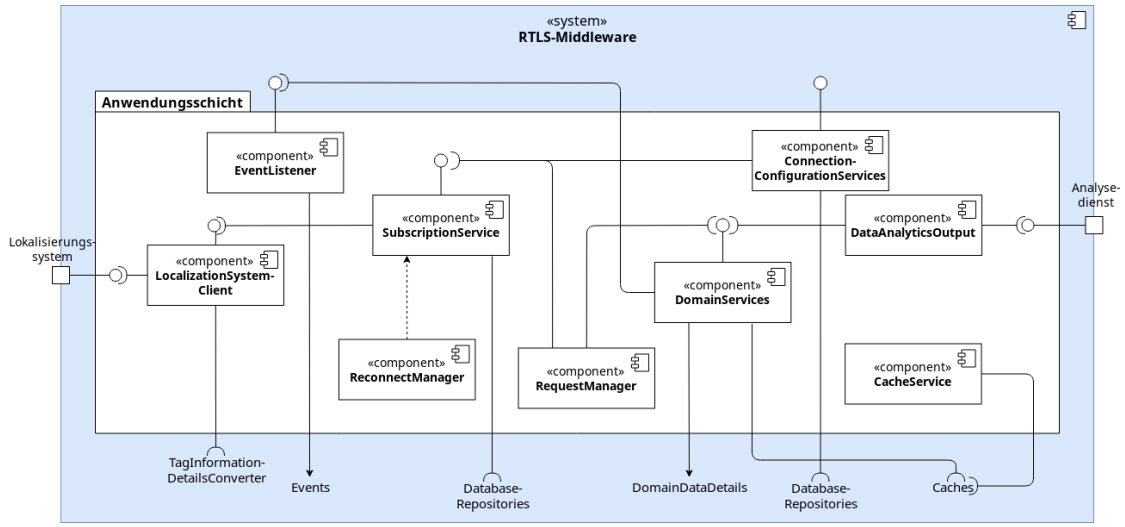

Abbildung 4.12: Anwendungsschicht

Die Anwendungskomponenten stellen Dienstleistungen bereit und knüpfen gegebenenfalls an andere externe Dienste an, um Daten zu verarbeiten. Die Daten werden entweder aufbereitet, um echte oder virtuelle Umgebungen darzustellen, oder kombiniert, um neue

Informationen oder Erkenntnisse zu gewinnen. In diesem expliziten Fall müssen die Anwendungskomponenten einige der expliziten Kernaufgaben bewältigen:

- Sie müssen an die Lokalisierungssysteme anknüpfen.
- Sie müssen die Lokalisierungsdaten mit Hinblick auf die einfache Verarbeitung und Darstellung aufbereiten.
- Sie müssen dem Analysedienst und der Präsentationsschicht die Daten zugänglich machen.
- Sie müssen das Auftreten von Events erkennen und die Events an die Präsentationsschicht weiterleiten.

Localization System Client

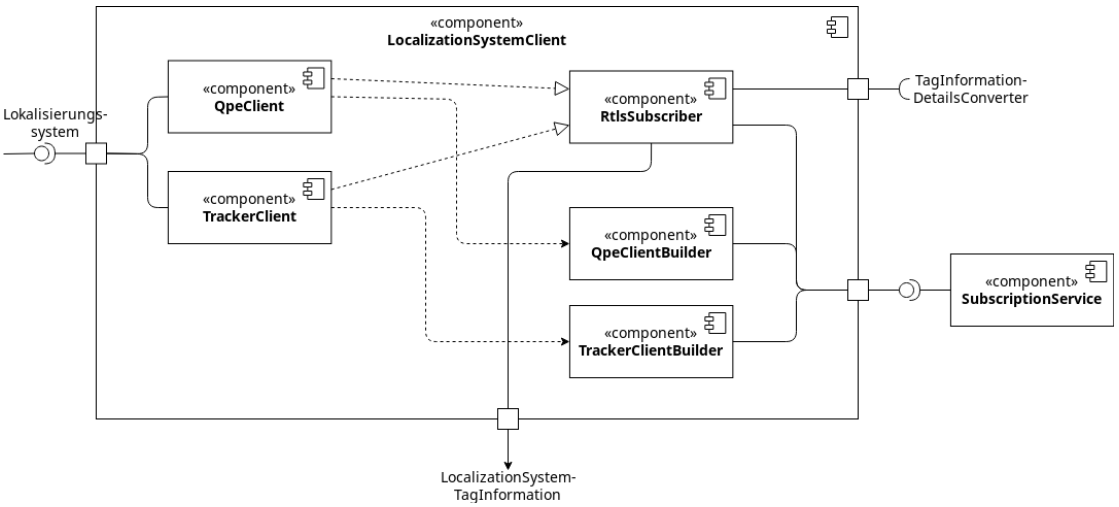

Abbildung 4.13: Klienten der Lokalisierungssysteme

Für jede Art von Lokalisierungssystem wird eine *Client*-Komponente benötigt. Anfangs werden lediglich die Tracker und QPE-Systeme unterstützt. Nichtsdestotrotz können natürlich mehrere Tracker und QPEs mit der Middleware verbunden werden. Je Verbindung muss ein eigener *Client* erzeugt werden, der für die Datenanfragen an dem jeweiligen Lokalisierungssystem zuständig ist. Da die Clients alle den gleichen strukturellen Aufbau haben und jeweils eine feste Verbindung abonnieren, werden diese als *RtlsSubscriber* zusammengefasst. Ein *RtlsSubscriber* stellt eine Funktion bereit, die die Daten an der jeweiligen Verbindung, gemäß seiner konfigurierten Vorgehensweise, abfragt und in das

einheitliche Datenformat konvertiert. Der *SubscriptionService* ruft diese Funktion ab, um die Daten zu verarbeiten. Die Konfiguration der *RtlsSubscriber*-Verbindung wird bei der Erzeugung des Clients über ihre entsprechende *Builder*-Komponente parametrisiert. Die *Builder*-Komponente erzeugt einen *Client* mithilfe des Erbauer-Entwurfmusters [Eilebracht and Starke, 2013]. Ein Builder kann einen Client erzeugen, indem die Informationen zur Erstellung fest im Code verankert sind oder indem sie als *ConfigurationDetails*-Objekt an den Builder übergeben werden. Zweiteres ermöglicht das Erstellen der Clients aus der Datenbank, indem die ConfigurationEntitities in die ConfigurationDetails konvertiert werden.

Eventlistener

Ein *Event* tritt während der Zustandsänderung eines Objektes auf, z.B. wenn ein Ortungsgerät ein Gebiet betritt oder verlässt. Aus diesem Grund befindet sich der Ursprung zumeist in den *DomainServices*. Ein Event wird ausgelöst, indem das Event erzeugt wird, mit den relevanten Informationen über das aufgetretene Event gefüllt wird und dem *Eventlistener* mitgeteilt wird. Der *Eventlistener* funktioniert gemäß dem Erzeuger-Verbraucher-Muster (producer-consumer). Das bedeutet, der Eventlistener wartet auf Events, nimmt die erzeugten Events an und reiht sie in einer Warteschlange ein, damit die Events sequentiell von dem *Eventhandler* entnommen und verarbeitet werden können.

Subscription Service

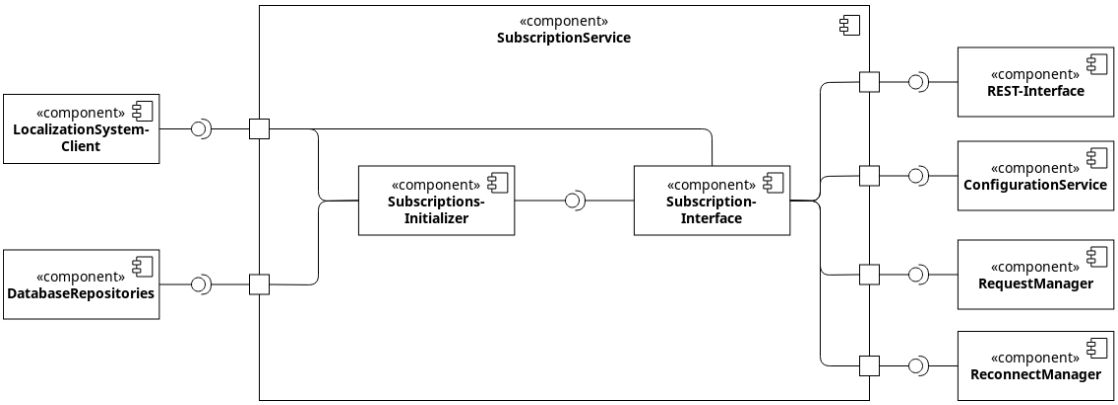

Abbildung 4.14: Dienst für Verbindungen zu Lokalisierungssystemen

Der *SubscriptionService* besteht aus zwei Subkomponenten, dem *SubscriptionInitializer* und dem *SubscriptionInterface*. Der SubscriptionInitializer wird lediglich beim Systemstart tätig und legt dabei alle Verbindungen zu Lokalisierungssystemen an, die in der Datenbank gespeichert sind und aktiviert diejenigen Verbindungen, die aktiviert werden sollen. Der SubscriptionService verwaltet alle Verbindungen (Abonnements bzw. *Rtls-Subscriber*) zu den Lokalisierungssystemen. Es gibt eine Vielzahl an Funktionen, die an den Verbindungen aufgerufen werden können, wie z.B. das Aktivieren oder Deaktivieren einer Verbindung oder das Abfragen von Verbindungsinformationen. Diese Funktionen sind an dem *SubscriptionInterface* aufrufbar, von denen einige Funktionen ebenfalls über das *REST-Interface* von außen aufrufbar sind.

Reconnect Manager

Der *ReconnectManager* hat eine einzige Aufgabe, die direkt auf dem *SubscriptionService* aufbaut. Der ReconnectManager versucht gescheiterte Verbindungen in regelmäßigen Abständen automatisch wiederzuverbinden. Eine gescheiterte Verbindung kann nicht aktiviert werden und ist demzufolge deaktiviert. Vorsätzlich deaktivierte Verbindungen sollen allerdings dauerhaft deaktiviert bleiben, bis sie manuell wieder aktiviert werden. Das bedeutet, dass sie von dem ReconnectManager ignoriert werden sollen. Gescheiterte Verbindungen müssen dementsprechend markiert werden, damit ausschließlich diese von dem ReconnectManager behandelt werden.

Request Manager

Der *RequestManager* ist eine zeitgesteuerte Komponente mit zentraler Bedeutung. In kurzen Abständen werden alle aktiven Verbindungen des SubscriptionServices nach neuen Lokalisierungsdaten angefragt. Nachdem die vereinheitlichten Lokalisierungsdaten von allen aktiven Verbindungen gesammelt wurden, wird der vorherige Datenbestand des Caches mithilfe des *TagService* in der *DomainServices*-Komponente aktualisiert. Der RequestManager stellt den Kern der angestrebten Echtzeitanwendung dar, indem er den Prozess der Datenbeschaffung und Datenaktualisierung iterativ anstößt. Das periodische Zeitintervall des Aktualisierungsbefehls kann in der Anwendung beliebig konfiguriert werden. Je geringer das Intervall ist, desto präziser wird die Ortungsgenauigkeit.

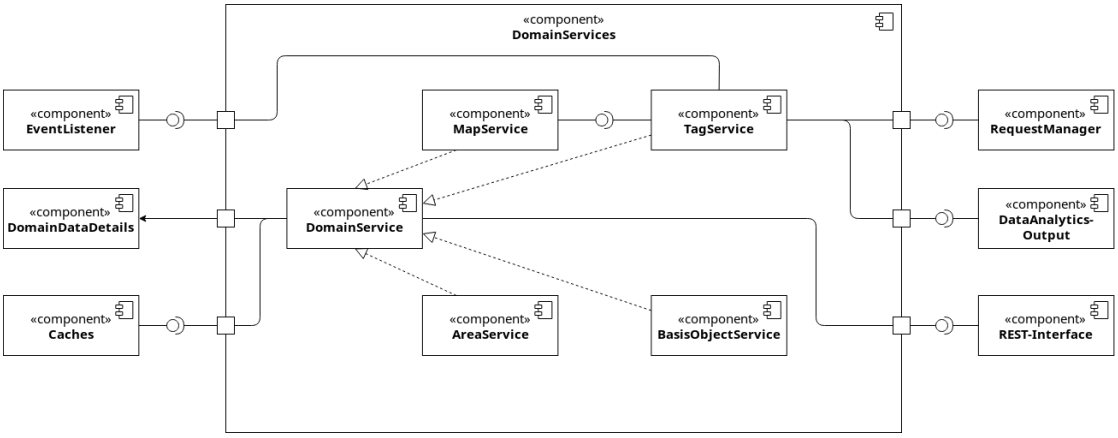

Abbildung 4.15: Dienste für Datenabfragen und -operationen

Domain Services

Die *DomainServices* bestehen aus je einem Subservice pro ortungsbezogener Datenklasse. Ein Subservice enthält alle Funktionen, die für die jeweilige *Details*-Klasse notwendig sind oder verlangt wurden. Der *TagService* stellt beispielsweise die Funktionen zum Einfügen, Ändern, Löschen, aber auch Abfragen von ausschließlich Tags über deren Details-Objekte zur Verfügung. Hierbei muss der TagService die Tags des *TagCaches* abfragen und bearbeiten und benötigt zudem unter Umständen für einzelne Funktionen auch die Caches der anderen Datenklassen. Für die Verbindungskonfigurationen gibt es eine eigene Service-Komponente (*ConnectionConfigurationService*), da die Schnittstellen und Aufgaben der Dienste sehr unterschiedlich sind.
Bei dem TagService werden vermutlich die meisten *Events* auftreten, weshalb der TagService eine Verbindung zu dem *Eventlistener* aufweist, um diesem die Events mitzuteilen. Die Funktionen aller DomainServices sind über das *REST-Interface* nach außen zugänglich. Auf der Ebene des REST-Interfaces müssen jedoch noch Plausibilitätsprüfungen sowie Antwortschemata hinsichtlich der expliziten REST-Anfragen implementiert werden.
Die Daten der Tags spielen offensichtlich eine besondere Rolle, da sie die tatsächlichen Ortsdaten enthalten. Der TagService wird zusätzlich speziell von dem Datenanalysedienst mittels der *DataAnalyticsOutput*-Komponente sowie von dem *RequestManager* angefragt.

Data Analytics Output

Diese Komponente ist optional und soll über die Konfigurationsdateien aus- oder angeschaltet werden können. Die Anforderungen für diese Komponente sind zu dem Zeitpunkt der Verfassung dieser Arbeit noch nicht vollständig ausgearbeitet worden. Der vorläufige Entwurf der Funktionsweise ist wie folgt definiert: Wenn die Komponente aktiv ist, fragt sie in gewünschten Zeitabständen ein Abbild des gesamten Datensatzes an dem *TagService* ab und schickt sie an den externen Datenanalysedienst über dessen Webservice. Dadurch sind die Zeitpläne der Komponenten allesamt in den Konfigurationsdateien definiert und können zentral miteinander abgestimmt werden. Alternative Realisierungen sind jedoch denkbar und in Zukunft zu klären.

Connection Configuration Service

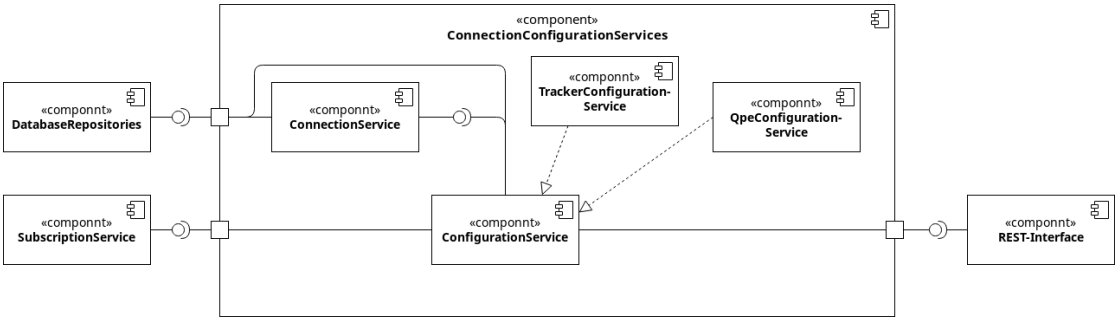

Abbildung 4.16: Dienst für Verbindungseinstellungen

Diese Komponente verwaltet alle Verbindungseinstellungen von der RTLS-Middleware zu Lokalisierungssystemen. Weil zum Erstellen einer Verbindung mehrere Arten von Verbindungsinformationen benötigt werden, wurden mehrere Subkomponenten entworfen. Das fachliche Datenmodell 3.5, das mithilfe der Normalformen für relationale Datenschemata entworfen wurde, trennen die *Configurations*, *Connections* und *Authentications* in unterschiedliche Entitäten. Da zwischen Connections und Authentications eine 1-zu-1 Beziehung herrscht und Authentications ausschließlich einen Nutzen für Connections bieten, werden die Authentications mit den Connections gemeinsam im *ConnectionService* verwaltet und stellen gemeinsam die technischen Verbindungsinformationen dar.
Pro Lokalisierungssystem existiert ein Schema für Configurations und soll infolgedessen auch jeweils ein *ConfigurationService* besitzen. Beim Hinzufügen einer Verbindung werden an dem *REST-Interface* alle nötigen Konfigurations- und Netzwerkadressinformationen eingegeben. Der zuständige ConfigurationService persistiert die Konfigurati-

onsinformationen mithilfe des entsprechenden *Database Repositorys* und delegiert die Netzwerkadressinformationen an die Operationen des ConnectionServices. Eine gültige Verbindung wurde demnach erfolgreich in der Datenbank gespeichert. Nach dem Anlegen der Verbindung aktiviert der ConfigurationService die Verbindung mithilfe des *SubscriptionServices*.

Cache Service

Der *CacheService* hält eine Liste mit allen *Cache*-Komponenten der Middleware und ermöglicht Operationen auf allen oder einer Submenge der Caches. Die vorläufig einzig geplante Aufgabe für diese Komponente ist die Persistierung aller Caches nach konfigurierbaren, periodischen Zeitintervallen.

4.2.3.4 Präsentationskomponenten

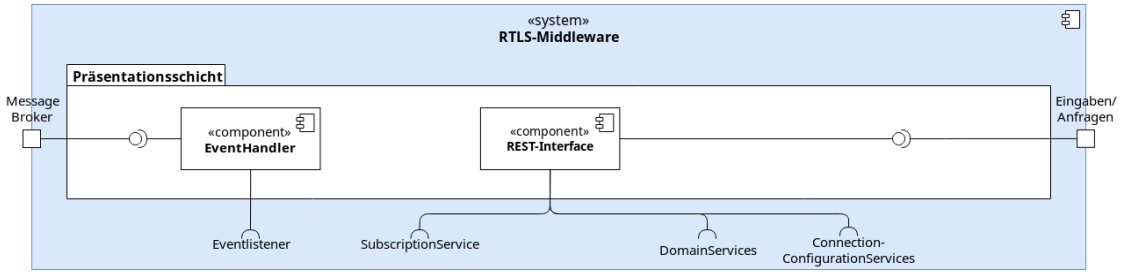

Abbildung 4.17: Präsentationsschicht

Die Präsentationskomponenten bilden die reguläre Benutzerschnittstelle. Sie sind für die Kommunikation der eigenen Dienstleistung zuständig und handhaben die Benutzerinteraktion sowie die Darstellung der gewünschten Inhalte.

REST-Interface

Die Kerneinheit des *REST-Interfaces* ist der *Controller*. Der Controller nimmt die Benutzeranfragen an, ruft die passende Operation der angefragten REST-Ressource auf und sendet die Antwort auf die jeweilige Anfrage zurück. Die definierten Operationen des Controllers greifen je nach Anfrage auf die Dienste des *SubscriptionServices*, der *DomainServices* und der *ConnectionConfigurationServices* zu, die die Informationen der ortungsbezogenen oder verbindungsbezogenen Datensätze abfragen oder manipulieren.

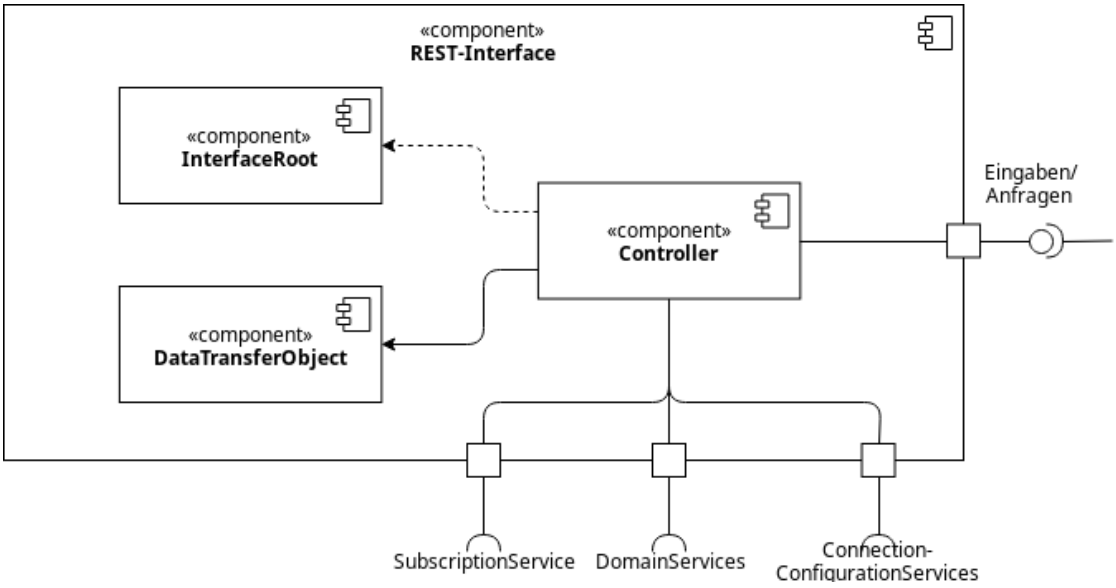

Abbildung 4.18: REST-Schnittstellenkomponente

Die *DataTransferObjects* definieren je nach Operation unterschiedliche Schemata des Antwortinhalts. Sie dienen dazu, die Antworten gestalten zu können, ohne alleinig auf das Schema der Details-Objekte angewiesen zu sein. Beispielsweise kann mit den DataTransferObjects die Umwandlung der Daten ins GeoJSON-Format realisiert werden.
Die *InterfaceRoot*-Komponente definiert die URIs der Applikation, die jeweils eine Operation repräsentiert.

Event Handler

Der *EventHandler* agiert als Verbraucher des Erzeuger-Verbraucher-Musters und nimmt sich demzufolge die *Events* aus der Warteschlange der *EventListener*-Komponente. Die Aufgabe des EventHandlers ist die Verteilung der Events auf die richtigen Topics des Message Brokers. Jedes Event enthält Informationen darüber, in welches Topic des Message Brokers es geschickt werden soll.

4.2.4 Laufzeitsichten

> „Die Laufzeitsicht beschreibt, welche Bestandteile des Systems zur Laufzeit existieren und wie sie zusammenwirken. Dabei kommen wichtige Aspekte des Systembetriebs ins Spiel, die beispielsweise den Systemstart, die Laufzeitkonfiguration oder die Administration des Systems betreffen.“ [Starke, 2017]

Die Laufzeitsicht verwendet Instanzen der Bausteine aus der Bausteinsicht. In der Laufzeitsicht sollen die statischen Bausteine in ihrer Dynamik bezüglich der Geschäftsprozesse (Abschnitt 3.4.4) und der dazugehörigen Use-Cases (Abschnitt 3.4.2) dargestellt werden. Der Entwurf der Laufzeitsichten ist im Vergleich zu den Geschäftsprozessen und Use-Cases konkreter und implementationsnäher.

Die Laufzeitsichten bilden die interne Laufzeitstruktur der RTLS-Middleware ab. Es ist möglich die gesamte Laufzeit in einem großen Sequenzdiagramm abzubilden, allerdings wurde die Laufzeitsicht für eine bessere Übersichtlichkeit auf mehrere Diagramme aufgeteilt. Zusätzlich wird dadurch die Aufteilung der Komponenten auf die verschiedenen internen Laufzeitprozesse veranschaulicht. Die Laufzeitsichten zeigen die 5 Hauptprozesse der RTLS-Middleware:

1. REST-Anfragen (Abbildung 4.19).
2. Verwaltung der Verbindungskonfigurationen über REST (Abbildung 4.20)
3. Aktualisierung mit gleichzeitiger Untersuchung der Zustandsänderung von Ortungsdaten (Abbildung 4.21).
4. Datenübermittlung für den Datenanalysedienst (Abbildung 4.22 oben).
5. Die automatische Persistierung der Daten (Abbildung 4.22 unten).

Die in den Abbildungen erkennbaren Stakeholder „User“ und „Analyst“ stehen symbolisch für alle möglichen Instanzen zwischen der Middleware und dem tatsächlichen Endnutzer. Die Middleware stellt immerhin einen Dienst für weitere Instanzen zur Verfügung, von dem der Endnutzer eigentlich indirekt profitiert. Beispielsweise werden die REST-Anfragen häufig nicht direkt durch den Befehl eines Nutzers angefragt, sondern im Zuge der Prozesse der Klientenanwendungen, die die Daten weiterverarbeiten.

Die Datenanfragen und Datenmanipulationen werden über die REST-Schnittstelle abgewickelt. Die Daten werden mittels HTTP-GET-Anfragen am RTLS-Middleware-Server

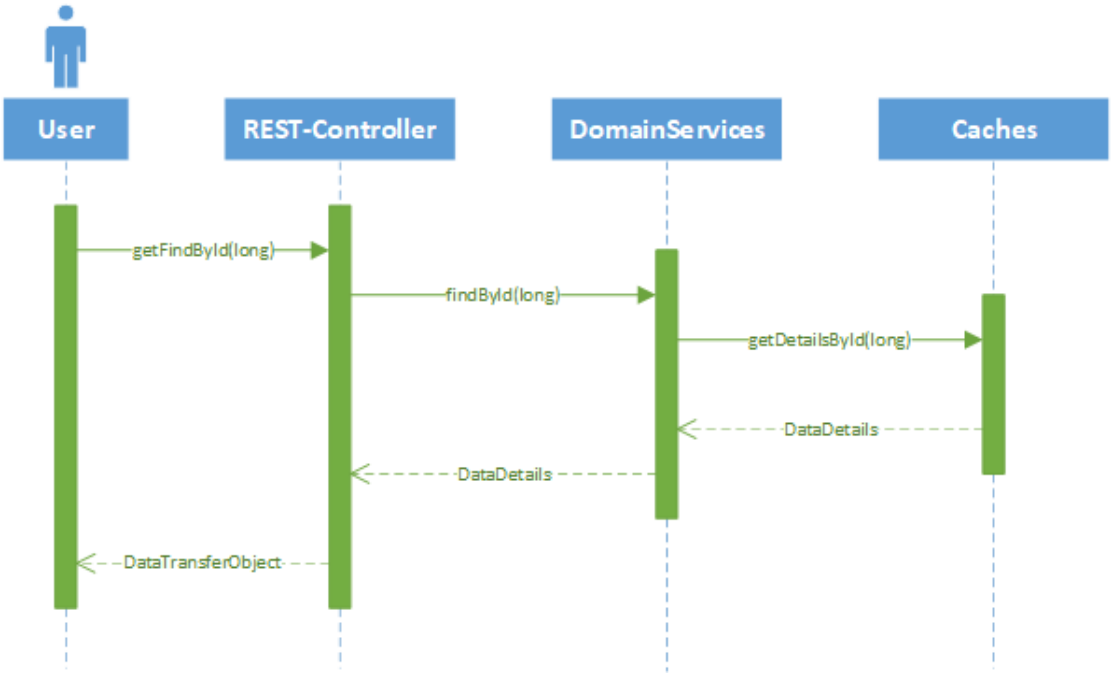

Abbildung 4.19: Sequenz einer allgemeinen REST-Anfrage

angefragt. Für die Datenmanipulationen werden die POST, PUT und DELETE-Operationen von HTTP verwendet. In Abbildung 4.19 ist eine einfache Datenanfrage dargestellt. Der Controller übersetzt die HTTP-Anfrage in die dazugehörige Java-Funktion, die wiederum eine Funktion in den DomainServices aufruft, um die gewünschten Daten vom Cache auszuwählen und ggf. zu filtern. Die Daten werden, wenn nötig, von dem Controller noch für die richtige Darstellung umgewandelt und dem Klienten als HTTP-Antwort übermittelt. Die Durchführung von Datenmanipulationen funktioniert analog zu dieser einfachen, sequentiellen Struktur von Befehlsfolgen.

Für das Hinzufügen, Verändern und Löschen einer gespeicherten Verbindung und dessen Konfiguration ist der ConfigurationService zuständig. Für die Operationen auf den geladenen Verbindungen der RtlsSubscriber ist der SubscriptionService zuständig. Die für die äußere Schnittstelle relevanten Operationen der beiden Services sind über die REST-Schnittstelle verfügbar. Abbildung 4.20 stellt einen generischen Prozess des Hinzufügens einer Verbindungskonfiguration eines beliebigen Lokalisierungssystems mit anschließendem Aktivieren der Verbindung dar. Dort übersetzt der REST-Controller die HTTP-POST-Anfrage in die dazugehörige Java-Funktion zum Anlegen der Verbindungskonfiguration des jeweiligen Lokalisierungssystems. Dem ConfigurationService wird das

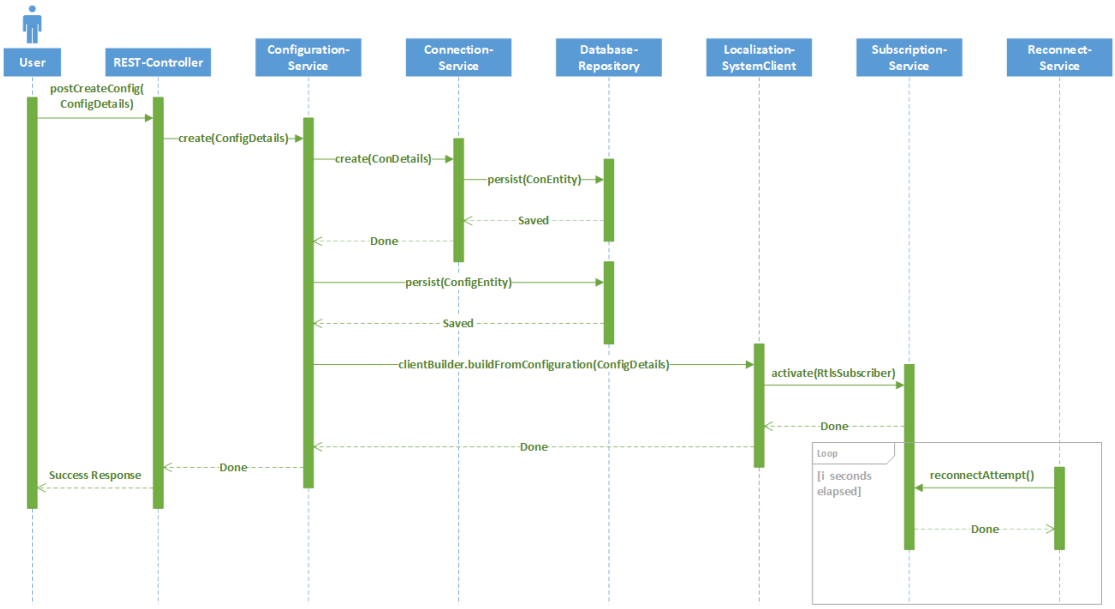

Abbildung 4.20: Sequenz des Hinzufügens einer Verbindungskonfiguration

Details-Objekt mit den anzulegenden Daten an die Hinzufügeoperation überreicht. Die geschachtelten Netzwerkadressinformationen werden separat über den ConnnectionService mithilfe des Repositorys persistiert. Erst daraufhin werden die systemspezifischen Konfigurationsinformationen vom ConfigurationService persistiert. Wenn die Verbindung erfolgreich gespeichert wurde, wird sie anschließend noch versucht zu aktivieren. Eine fehlgeschlagene Verbindung bleibt trotz des Abbruchs in dem SubscriptionService gespeichert und wird regelmäßig durch den ReconnectService versucht zu aktivieren. Dem Administrator wird dadurch die lästige Aufgabe abgenommen, die Verbindungen nach einer kurzen Netzwerkstörung wieder manuell aktivieren zu müssen.

Die automatisierten Prozesse der RTLS-Middleware bestehen aus Endlosschleifen, die parallelisiert beim Systemstart betreten werden und die jeweiligen Funktionen nach benutzerdefinierten Zeitintervallen anstoßen. Die Intervalle (x, y, z seconds) der Prozesse sind jeweils in den Konfigurationsdateien definiert und vor Systemstart anpassbar.
Abbildung 4.21 zeigt den automatischen Aktualisierungsprozesses, bei dem der aktuelle Datenbestand des TagCaches mit neuen Ortungsdaten der Lokalisierungsdaten aktualisiert wird. Dabei werden die neuen Datenobjekte mit den aktuellen Datenobjekten verglichen. Zuerst wird die ID überprüft. Sollte die ID noch nicht bereits bekannt sein, wird das Objekt mit seinen Daten dem TagCache als neues Ortungsgerät hinzugefügt. Falls die ID bekannt ist, werden die Daten nur aktualisiert, wenn das Ortungsgerät mindestens

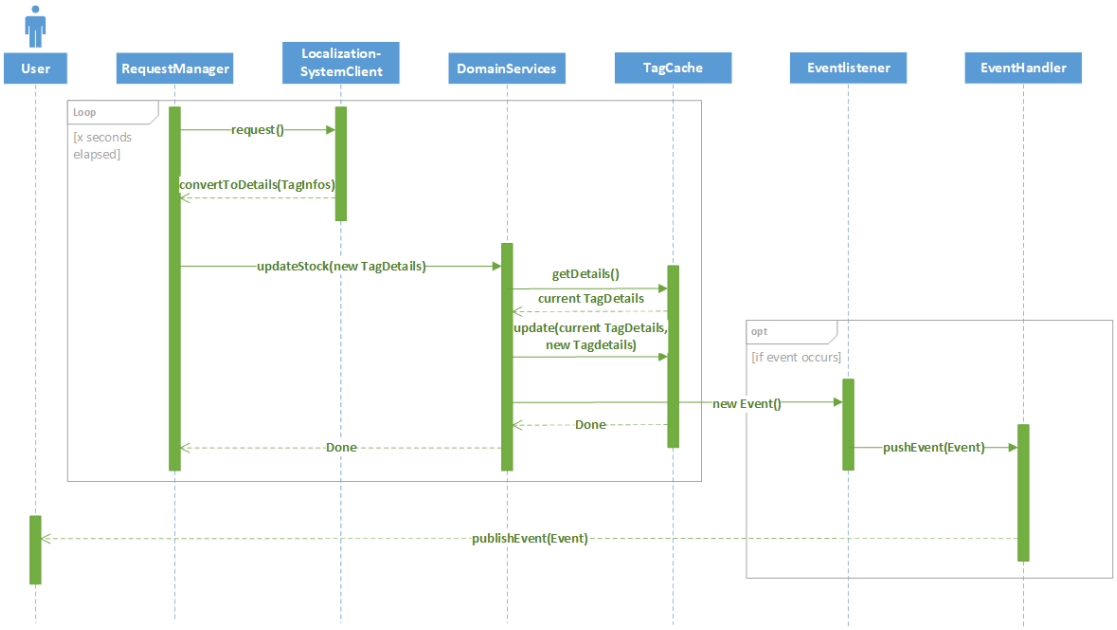

Abbildung 4.21: Sequenz des Aktualisierungsprozesses der Ortungsdaten

eine gewisse Distanz von dem letzten Standort entfernt ist. Die Notwendigkeit der Distanzschwelle rührt von dem Phänomen her, dass ein Ortungsgerät, aufgrund von Signal- und Berechnungsungenauigkeiten, Ruckler in seinen Ortsdaten aufweisen kann, obwohl es in Wahrheit stationär ist. Falls ein Zustand verändert wurde und sich die Position des Ortungsgerätes verschoben hat, kann es sich beispielsweise von außerhalb einer Area in eine Area bewegt haben. Die Zustandsänderung kann zu einem entsprechenden Event führen, das über den EventHandler und den Message Broker an die Rezipienten verteilt wird. Beliebige Events sind denkbar, die bei Bedarf unkompliziert in der Implementation berücksichtigt werden können.

Die Datenbereitstellung für den Datenanalysedienst und die Persistierung der zwischengespeicherten Daten der Caches sind in Abbildung 4.22 kombiniert. Dennoch sind die Prozesse unabhängig voneinander.
Im Datenanalyseprozess (oben) wird über die DomainServices ein gesamtes Datenabbild der Caches abgefragt und das Abbild anschließend über einen Webservice-Klienten an den Datenanalysedienst gesendet. Die iterative Ausgabe für die Datenanalyse kann eine eigene Periodizität verfolgen oder an die Ausführung die Aktualisierungsiterationen gekoppelt werden und im sofortigen Anschluss operieren.
In der Cache-Persistierung (unten) werden alle Daten der Caches nach regelmäßigen Zei-

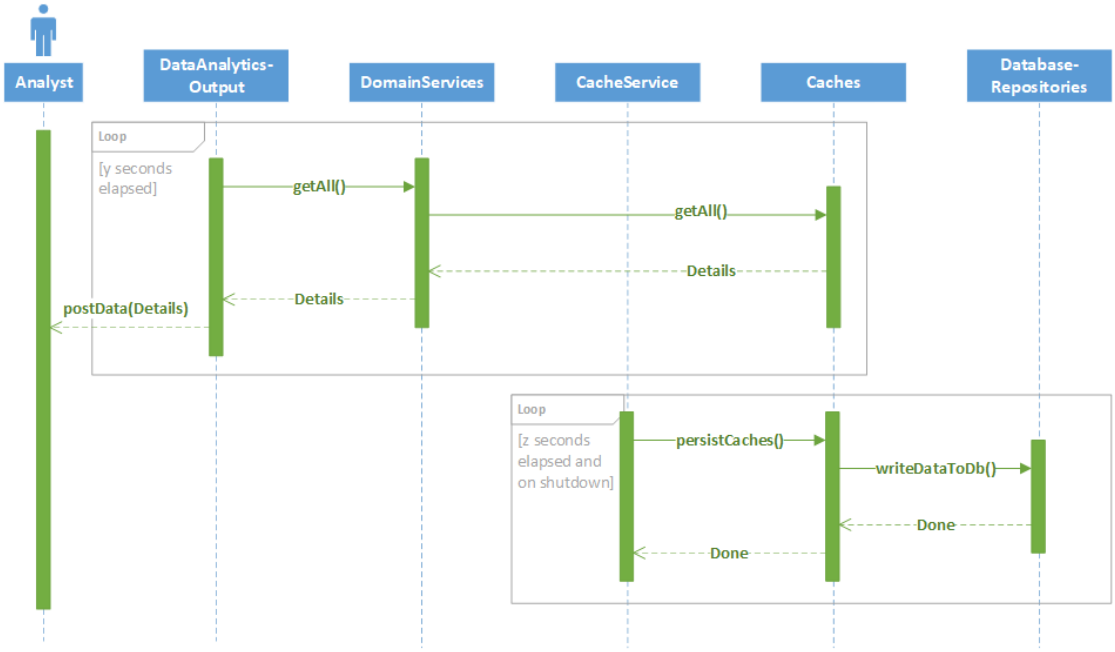

Abbildung 4.22: Sequenzen des Datenanalyseprozesses und der Cache-Persistierung

tintervallen in die Datenbank gespeichert, um die Daten im Falle eines Systemabsturzes abzusichern. Beim ordnungsgemäßen Herunterfahren werden die Daten nach demselben Prinzip gespeichert, um beim nächsten Systemstart wieder in die Caches geladen werden zu können. Jede Instanz der Caches besitzt eine Operation, um dessen gesamten Daten zu persistieren. Der CacheService ruft diese Operation nacheinander an jeder Instanz auf.

4.3 APIs (Schnittstellen)

Die Relevanz von Schnittstellen und die fachlichen Anforderungen an die Schnittstellen der RTLS-Middleware wurden bereits im Abschnitt 3.4.5 erklärt. In den Beantwortungen der Fragestellungen 4.1.2 wurden zudem klare Architekturentscheidungen der Schnittstellen bezüglich der Produkte und Technologien getroffen. Darüber hinaus vertieft die Kontextabgrenzung 4.2.1 den Entwurf der gesamten Umgebung um die RTLS-Middleware und dessen externe Schnittstellen.

In diesem Abschnitt sollen nun die Schnittstellen des gesamtheitlichen Systems näher betrachtet werden, die von der RTLS-Middleware angeboten werden. Die RTLS-Middleware

bietet selbst die zwei Schnittstellen der REST-API und des Message Brokers für Event-Nachrichten an, die in diesem Abschnitt genauer erklärt und teilweise dokumentiert werden. Die gewünschten Schnittstellenfunktionen der REST-API wurden vor dem Beginn der Implementation als eine Liste mit verbalen Beschreibungen der Funktionen niedergeschrieben und kommuniziert. Im Laufe der Realisierung wurde die API, um weitere nützliche Funktionen erweitert. Da die API von der Implementation unabhängig ist, wird die API oftmals erst definiert und die Implementation auf die API zugeschnitten. Dieser Ansatz nennt sich API-First.
Die Entscheidung, ob der Datenanalysedienst die für ihn vorgesehene Schnittstelle selbst bereitstellen soll, bleibt weiterhin unsicher. Da die Komponente allerdings eine optionale Nebenkomponente des Systems ist, wird diese Schnittstelle zu diesem Zeitpunkt erst einmal undefiniert bleiben.
Zu Beginn der beiden folgenden Untersektionen wird jeweils erklärt, inwiefern die RTLS-Middleware die beiden Schnittstellen anbietet, wonach die jeweilige Beschreibung und Dokumentation der Schnittstelle folgt.

4.3.1 REST-API

Viele externe Schnittstellenarchitekturstile, wie auch REST, verfolgen das Client-Server-Modell, bei dem die Schnittstelle die Dienste des Anbieters nach außen zur Verfügung stellt. Klienten müssen die Dienste selbstständig anfragen, worauf der Anbieter der Schnittstelle eine Antwort sendet. Die Schnittstelle ist demnach in dem System des Anbieters integriert und definiert. Die gesamte reguläre Interaktion der Klienten mit der RTLS-Middleware laufen über diese Schnittstelle.

Die REST-API wird automatisch von Swagger dokumentiert, sodass sich jede Änderung der REST-API in der Dokumentation der API widerspiegelt. Neben anderen bekannten Hilfsmitteln zur Dokumentation empfiehlt [Spichale, 2019] Swagger für die automatische Generierung der REST-API-Dokumentation. Falls der API-First-Ansatz bevorzugt wird, kann dieser ebenfalls mittels Swagger erfüllt werden, indem man das Swagger-JSON-Dokument über den Swagger-Editor bearbeitet. Die Swagger-API-Dokumentation der laufenden Anwendung ist unter einer benutzerdefinierten URL zu finden. Dort kann eine Beschreibung der Schemata und Semantiken der einzelnen Funktionen aufgeklappt werden, indem auf die gewünschte Funktion geklickt wird.

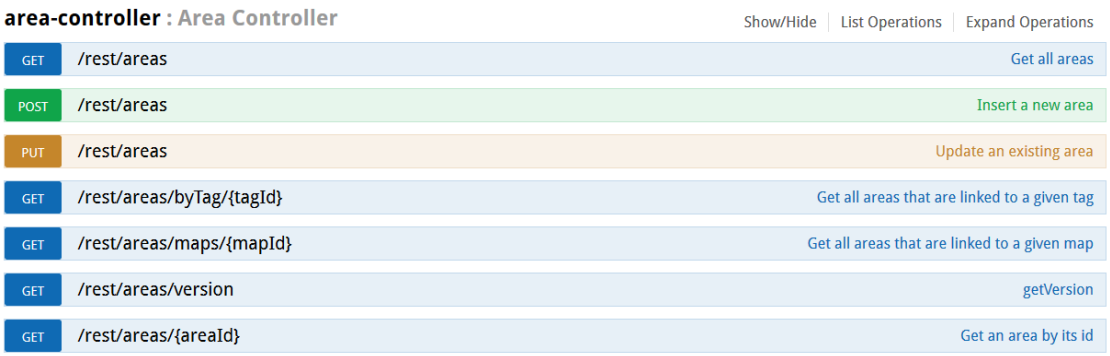

Abbildung 4.23: REST-Funktionen der Areas

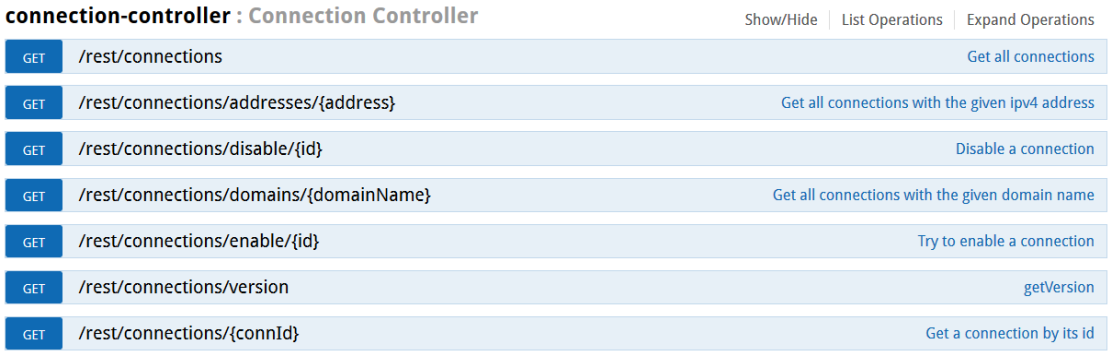

Abbildung 4.24: REST-Funktionen der Connections

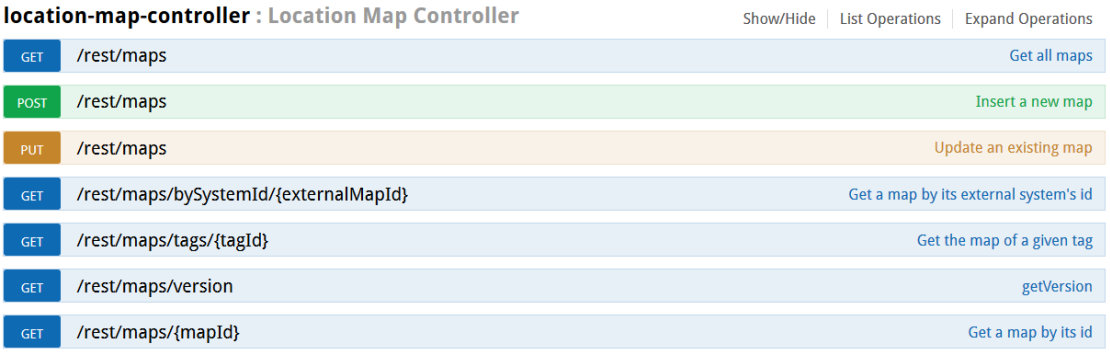

Abbildung 4.25: REST-Funktionen der Maps

Die Abbildungen 4.23 bis 4.29 zeigen alle Funktionen der API bis auf die der Tracker-Configurations, ObjectTypes und LocalizationInformationOrigin. Die Schnittstelle der Tracker-Configurations ist analog zu der Schnittstelle der Qpe-Configurations. Gleichermaßen sind die Schnittstellen der ObjectTypes und LocalizationInformationOrigin analog zu der Schnittstelle der TagTypes. Die DELETE-Operationen fehlen zu diesem Zeitpunkt

basis-object-controller : Basis Object Controller — Show/Hide | List Operations | Expand Operations

GET	/rest/objects	Get all objects
POST	/rest/objects	Insert a new object
PUT	/rest/objects	Update an existing object
GET	/rest/objects/coordinates/{objectId}	Get the coordinate of the object's tag
GET	/rest/objects/tags/{tagId}	Get the object that is linked to a given tag
GET	/rest/objects/version	getVersion
GET	/rest/objects/{objectId}	Get an object by its id

Abbildung 4.26: REST-Funktionen der Basis Objects

qpe-controller : Qpe Controller — Show/Hide | List Operations | Expand Operations

GET	/rest/connections/qpes	Get all qpe configurations
POST	/rest/connections/qpes	Insert and activate a new qpe configuration
PUT	/rest/connections/qpes	Update and reactivate an existing qpe configuration
GET	/rest/connections/qpes/addresses/{address}	Get qpe configurations with a given ipv4 address
GET	/rest/connections/qpes/domains/{domainName}	Get qpe configurations with a given domain name
GET	/rest/connections/qpes/version	getVersion
GET	/rest/connections/qpes/{connId}	Get a qpe configuration by its id

Abbildung 4.27: REST-Funktionen der QPE Configurations

tag-controller : Tag Controller — Show/Hide | List Operations | Expand Operations

GET	/rest/tags	Get all tags
PUT	/rest/tags	Update the updateable fields of a tag
GET	/rest/tags/areas/{areaId}	Get the tags in a given area
GET	/rest/tags/bySystemId/{externalTagId}	Get a tag by its external system's id
GET	/rest/tags/coordinates	Get the coordinates of all tags in a single GeoJSON feature collection
GET	/rest/tags/maps/{mapId}	Get all tags linked to a given map
GET	/rest/tags/simple	Get all tags in a simple format
GET	/rest/tags/version	getVersion
GET	/rest/tags/withoutObject	Get all tags without a linked object
GET	/rest/tags/withoutType	Get all tags without a tag type
GET	/rest/tags/{tagId}	Get a tag by its id

Abbildung 4.28: REST-Funktionen der Tags

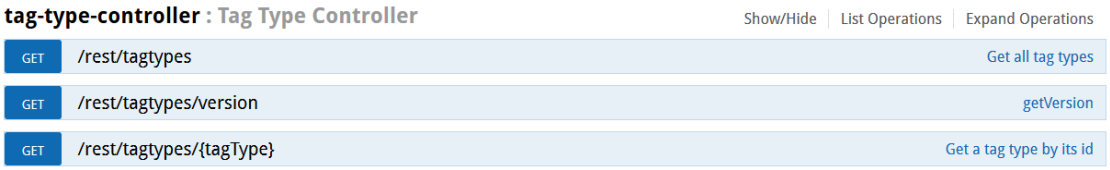

Abbildung 4.29: REST-Funktionen der Tag Types

noch in der gesamten API.

4.3.2 Event-API (Message Broker)

In der Architektur der Event-API sendet die RTLS-Middleware die Event-Nachrichten selbstständig an den Message Broker, der die Nachrichten an die Abonnenten der Topics verteilt. Aus systemtechnischer Sicht ist der Message Broker der Server der Schnittstelle. Die RTLS-Middleware beansprucht den Dienst und sendet Daten selbständig an die Topics des Message Brokers. Die RTLS-Middleware ist dadurch eigentlich ein Klient. Die Rezipienten der Nachrichten müssen die Topics selbständig abonnieren, wodurch sie ebenfalls den Dienst des Message Brokers anfragen. Sie sind also ebenfalls Klienten.
Aus Sicht der Nachrichtenübertragung ist die stattfindende Kommunikation ein unidirektionaler Multicast. Der Message Broker ist dabei nur das Medium der Übertragung. Die RTLS-Middleware ist als Absender demnach prinzipiell der Anbieter der Schnittstelle. Die Formate, die Schemata und die Inhalte der Nachrichten werden ausschließlich von der RTLS-Middleware spezifiziert.

Im Folgenden ist das Schema einer Event-Nachricht definiert, das ausgelöst wird, wenn ein Tag eine Area betritt:

```
{
  "timestamp":"2019-07-31T18:35:54.610000000Z",
  "eventMessage":"Tag Tag0001 has entered area Halle Ost"
}
```

Das Event wird dann auf dem Topic mit einem vorher definierten Ressourcen-Schema veröffentlicht. Das Topic könnte z.B. das Schema **/entered/{areaId}/{tagId}** haben, wobei areaId die ID der Area Halle Ost ist und tagId die Id des Tag0001. Damit ein Subscriber über alle Events dieser Art informiert wird, kann er die Ressource **/entered**/+ abonnieren. Das „+“ ist eine sogenannte Wildcard.

5 Realisierung

Die Realisierung des Softwareprojektes ist die Umsetzung der iterativ ausgearbeiteten Anforderungen und dem daraus resultierenden Entwurf. Die Anforderungen und der Entwurf bilden den Bauplan der Software, dem in der Realisierung äußerst strikt gefolgt werden soll. Da die Planung jedoch oft nicht jeden Aspekt beleuchten und Probleme der praktischen Umsetzung voraussehen kann, müssen während der Realisierung alternative Lösungen im laufenden Betrieb gefunden werden, um das gewünschte Ziel dennoch bestmöglich zu erfüllen. In diesem Kapitel werden ausschließlich besondere oder von der Planung abweichende Implementationsentscheidungen und -ergebnisse, während der Realisierung des Softwareprojektes, zusammengefasst. Die besonderen Implementierungseinzelheiten der Komponenten werden jeweils im Kontext der 5 Hauptprozesse erklärt. Die getroffenen Entscheidungen in den implementierten Komponenten werden angesichts deren Einflusses auf die Prozesse besser ersichtlich.
Die Entwicklungsphase ist zu dem Zeitpunkt des Verfassens der Arbeit noch nicht vollständig abgeschlossen. Alle im Entwurf beschriebenen Bausteine und Schnittstellen wurden in der Software jedoch bereits umgesetzt und funktionieren prototypisch. Die Einbindung der QPE und der Tracker-Systeme ist ebenfalls implementiert. Bis zu dem im Folgenden beschriebenen Entwicklungsstand wurden innerhalb 3,5 Monaten insgesamt ca. 400 Arbeitsstunden für die Programmierung der RTLS-Middleware aufgebracht. Entwickelt wurde die RTLS-Middleware auf der integrierten Entwicklungsumgebung (IDE) „Eclipse“ mit Java 8 und der Spring-Boot-Version 1.5.2. Der in Spring Boot enthaltenen Tomcat Webserver erlaubt einen schnellen Programmstart über die IDE für Usability-Testing von frisch implementierten Code.

Das Java-Projekt wurde in 5 Pakete unterteilt. 4 Pakete entsprechen dem Schichtenmodell der Architektur aus Abschnitt 4.1. Die Präsentationsschicht besitzt den Namen „app“, die Anwendungsschicht „service“, die Fachkonzeptschicht „domain“ und die Datenhaltungsschicht „persistence“. Das letzte Paket „base“ enthält allgemeine Hilfsklassen und -funktionen für fachbereichunspezifische Aufgaben sowie Konfigurationsdateien und -code für verwendete Bibliotheken und Frameworks.

Die Funktionalität der eingebundenen Lokalisierungssysteme konnte mit von Lufthansa Industry Solutions bereitgestellten Testsystemen kontrolliert werden. Während der gesamten Programmierphase wurde eine Simulationssoftware für das Quuppa Intelligent Locating System genutzt, um die korrekte Anbindung und qualitative Verarbeitung der Ortsdaten zu gewährleisten. Für die Integration der GPS-Tracker wurden mehrere verschiedene Ortungsgeräte mit einem Internet-of-Things-Netz (IoT-Netz) verbunden, das die Daten letztendlich über einen MQTT-Broker versendet, welcher von der RTLS-Middleware abonniert wurde.

5.1 REST-Anfragen

Zugunsten der Übersichtlichkeit wurden der REST-Controller, das InterfaceRoot und die DataTransferObjects den Entitäten der RTLS-Middleware zufolge logisch in Pakete aufgeteilt und folgt somit der Pfadstruktur der REST-API-Ressourcen. Beispielsweise steuert die REST-Ressource /**api**/**areas**, mit dem Pfadsegment „areas“, den Controller für Anfragen zu Areas an. Das gleiche Muster der Auftrennung nach Entitäten wiederholt sich in der Anwendungsschicht. Dementsprechend sind die Subkomponenten der Domain-Services und des ConnectionConfigurationService aus der Bausteinsicht (4.2.3) in Pakete aufgetrennt, wodurch die Navigation im Java-Projekt eingängig und einfach ist.

Die äußere Darstellung der Daten mittels JSON wird durch die Programmierbibliothek „Jackson“ automatisiert erledigt. Jackson erlaubt das allgemeine Mapping zwischen Java-Objekten und JSON-Daten. Damit das Mapping fehlerfrei funktioniert, müssen die Klassendefinitionen der Objekte stets genau dem Schema der JSON-Daten entsprechen.
Ein REST-Controller muss jeweils eine Funktion pro REST-Anfrage definieren, die die tatsächliche Ausführung der entsprechenden Befehle im Server anstößt. Jackson sorgt dafür, dass die Funktionen der Controller als normale Java-Methoden implementiert werden können, indem deren Parameter und Rückgabewerte an der Schnittstelle automatisch in das JSON-Format umgewandelt werden. Eine Anfrage eines Klienten, bei der Daten übermittelt werden (wie z.B. ein HTTP-POST), muss die Daten entsprechend dem korrekten JSON-Schema gemäß der Schnittstellendokumentation enthalten. Gleichzeitig müssen die Datentypen der Parameter im Java-Projekt das gleiche Schema der Schnittstellendokumentation abbilden, um aus den JSON-Daten in ein Objekt konvertiert werden zu können. Der Datentyp des Rückgabewertes muss ebenfalls korrekt gewählt werden, um dem dokumentierten JSON-Schema der Antwort zu genügen. Der korrekte Aufbau der

Anfrage liegt demnach in der Verantwortung des Klienten. Bei der Übermittlung der Antwort liegt die Verantwortung hingegen beim Server.
Eine Instanz der Details-Klassen bildet die Gesamtheit der Daten eines Objektes in seiner eigenen Semantik ab, die durch die Java-Klasse spezifiziert ist. Mithilfe der Umwandlung des Dateninhalts in eine beliebige andere Klasse (*DataTransferObject*), können die Daten in ein anderes Schema für das Mapping gebracht werden. Im Kontext der Tag-Daten wird dadurch eine minimalisierte Darstellung der wichtigsten Ortsdaten sowie eine Darstellung in GeoJSON realisiert.

Eine REST-Anfrage wird immer durch den REST-Controller zu einem Service geleitet, der die Befehle, Berechnungen, Datenbereitstellung und eventuelles Filtern vornimmt, um die Applikation zu steuern, zu verändern oder eine gewünschte Antwort zu generieren. Die herkömmlichen REST-Anfragen beziehen sich auf die Abfrage und Manipulation der Anwendungsdaten. Das bedeutet in dem Aufbau der RTLS-Middleware schlichtweg, dass der Datenbestand der Caches verändert wird oder Objekte der Caches abgerufen oder verändert werden.
Wie die Caches implementiert werden, ist eine programmiertechnische Entscheidung und wurde im Entwurf daher nicht vorgegeben. Angesichts der ermittelten Anforderungen konnte eine leistungsstarke Datenstruktur für die Caches gefunden werden. Die Caches wurden jeweils mithilfe einer Hashtabelle implementiert, die die IDs auf deren entsprechenden Details-Objekte abbildet. Diese direkte Abbildung von IDs zu Objekten wurde allerdings nur für den AreaCache und den BasisObjectCache implementiert. Für den TagCache und MapCache mussten zwei Hashtabellen miteinander verknüpft werden. Die Notwendigkeit der Maßnahme zwei verschiedene Arten von Caches zu implementieren, wird sich im Abschnitt der Realisierung des Aktualisierungsprozesses 5.3 erschließen.
Eine Hashtabelle eignet sich hervorragend, wenn die Zieldaten durch einen Schlüssel abgebildet werden können, der aus den Zieldaten einfach ermittelt werden kann und auf die Position der Zieldaten in der Datenstruktur zeigt. Dadurch, dass die Position für ein neues oder ein bestehendes Objekt direkt ermittelt werden kann, besitzen die Einfüge-, Lösch- und Abfrageoperationen einen konstanten Zeitaufwand. Aus der ID der Objekte wird der Schlüssel erzeugt. Da die Objekte fast ausschließlich über ihre ID repräsentiert und referenziert werden und alle Objekte eine ID besitzen, eignet sich die Datenstruktur optimal für die Caches, um die Datenzugriffe performant durchzuführen.

5.2 Verbindungskonfigurationen

Die Details-Klassen der Verbindungskonfigurationen werden auf die gleiche Weise geschachtelt wie die entsprechenden Entitäten. Die ConfigurationEntity enthält bekanntermaßen eine ConnectionEntity mit den Netzwerkadressinformationen. Gleichermaßen gibt es für die Details-Klassen die ConnectionDetails, die in den ConfigurationDetails referenziert werden. Beim Erzeugen einer Verbindung bzw. einem RtlsSubscriber werden die gesamten Verbindungsinformationen verwendet, um den implementierenden Lokalisierungssystem-Client zu erzeugen. Nach der Erzeugung des Objektes werden die lokalisierungssystemspezifischen Konfigurationsdaten nicht mehr benötigt. Damit der RtlsSubscriber allerdings wiederholt Anfragen an einen Server schicken oder sich im Falle eines Verbindungsabbruchs wieder verbinden kann, müssen die Netzwerkadressinformationen (d.h. die ConnectionDetails) in dem RtlsSubscriber referenziert und damit konserviert bleiben.
Weil die Konfigurationsinformationen nach der Erzeugung des RtlsSubscribers aus dem Hauptspeicher aufgeräumt werden, müssen die Konfigurationsdaten aus der Datenbank geladen werden, wenn sie am REST-Interface für Konfigurationen abgefragt werden. Die ConnectionDetails sind hingegen über die RtlsSubscriber zugänglich und können daher am REST-Interface für Verbindungen angefragt werden, welches die Operationen des SubscriptionServices zur Verfügung stellt.

Die unterschiedlichen Lokalisierungssysteme benutzen teilweise unterschiedliche Schnittstellentechnologien. Die benötigten Programmkomponenten, die nötig sind, um auf eine Schnittstellentechnologie zuzugreifen, sind typischerweise verallgemeinerbar. Deswegen wurden abstrakte Klassen für die Clients und ClientBuilder geschrieben, die einen Rahmen für die Clients und ClientBuilder der implementierten Systeme schaffen. An dieser Stelle wurden die Klassen hinsichtlich der Schnittstellentechnologien der Lokalisierungssysteme abstrahiert. Bisher wurde die Unterteilung in REST-basierte und MQTT-basierte Systeme vorgenommen. Dadurch können neu zu integrierende System-Clients die entsprechenden abstrakten Klassen der Technologie implementieren, wodurch weniger Programmieraufwand für das Hinzufügen eines neuen System-Clients resultiert. Vorerst gibt es je Technologie nur ein implementiertes System. Die Klienten für QPEs benötigen Unterstützung für REST-APIs und die Klienten für GPS-Tracker benötigen Unterstützung für MQTT. Die Einbindung der QPE mit den Simulationsdaten war der erste Meilenstein in der Programmierphase. Erst danach wurde die Einbindung der GPS-Tracker erfolgreich umgesetzt, woraus die Idee der Abstraktion der Schnittstellentechnologien

entsprang.

5.3 Aktualisierungsprozess

Die Daten, die im Laufe des Aktualisierungsprozesses aus den Lokalisierungssystemen anfallen, besitzen einen eigene externe ID des Ursprungssystems. Diese ID ist normalerweise eine Nummer oder eine Zeichenkette. Damit die RTLS-Middleware die externen IDs gleich behandeln kann, werden sie zu Zeichenketten konvertiert. Die externe ID muss zwingend in einem Datensatz eines Tags vorhanden sein und ist aufgrund dessen ein Pflichtfeld beim Erzeugen der TagDetails. Neu abgefragte Tag-Informationen haben zur Identifikation lediglich die externe ID. Um direkt überprüfen zu können, ob ein Tag mit dieser externen ID bereits im Cache enthalten ist, enthält der TagCache eine Hashtabelle, die die externen IDs den entsprechenden TagDetails zuordnet. Falls die externe ID bekannt ist, wird das Objekt mit den neuen Daten einfach aktualisiert. Falls die externe ID nicht bekannt ist, wird das Objekt hinzugefügt und braucht allerdings eine zusätzliche interne ID für die Identifikation innerhalb der Applikation und der Datenbank. Eine eindeutige interne ID wird beim Hinzufügen des Objekts generiert. Eine Abfrage mithilfe dieser ID soll jedoch ebenfalls performant sein, weshalb eine weitere Hashtabelle die internen IDs auf die externen IDs abbildet. Dadurch entsteht eine Kette aus Hashtabellen, in der eine interne Id auf eine externe ID verweist und die externe ID auf das tatsächliche TagDetails-Objekt. Maps besitzen häufig ebenfalls eine externe ID, weswegen der MapCache analog aufgebaut ist.

Die anwendungsspezifischen Datentypen des fachlichen Datenmodells (Abbildung 3.5) besitzen, aufgrund der Repräsentation von geografischen Koordinaten und Flächen- und Entfernungsberechnungen, eine zentrale Bedeutung in der Anwendung.
Die enumeration-Datentypen stellen allerdings einen Spezialfall innerhalb des Programms dar. Sie stehen bereits vor der Laufzeit des Systems fest, sind als Java-Enums realisiert und im Programmcode vordefiniert. Der TagType, ObjectType und LocalizationInformationOrigin sollen, der Vollständigkeit der Datenbank halber, jedoch zusätzlich in der Datenbank gespeichert sein. Für die Versicherung der Datenaktualität sollen die Enums mit der Datenbank beim Systemstart synchronisiert werden.
Die 3 anwendungsspezifischen Datentypen werden intern aus dem Datenmodell in Objekte einer von 3 verschiedenen Klassen übersetzt. Diese Übersetzung findet bei der Umwandlung der Java-Entitätsobjekte in die Details-Objekte mithilfe der EntityDetailsFac-

tories statt. Es folgen die Definitionen und Erklärungen der Relevanz der 3 anwendungsspezifischen Datentyp-Klassen GPSCoordinate, GPSPolygon und GPSBoundingBox.

5.3.1 GPSCoordinate

Eine GPS-Koordinate wird typischerweise durch den Breitengrad (latitude) und Längengrad (longitude) des geodätischen Referenzsystems WGS-84 dargestellt. Die Koordinate wird durch die Klasse WgsCoordinate repräsentiert und besitzt mit dem Breitengrad und Längengrad genau zwei Instanzvariablen, die Fließkommazahlen (double) speichern. Die Klasse besitzt eine Methode, um die direkte Strecke zwischen der Koordinate zu einer anderen Koordinate in Metern zu berechnen:

```
public double orthodromicDistance(WgsCoordinate coordinate) {
        double radialLat = Math.toRadians(latitude);
        double endRadialLat = Math.toRadians(
                coordinate.getLatitude());
        double deltaLon = Math.toRadians(
                coordinate.getLongitude() - longitude);

        double distance = Math.acos(
                        Math.sin(radialLat) *
                        Math.sin(endRadialLat) +
                        Math.cos(radialLat) *
                        Math.cos(endRadialLat) *
                        Math.cos(deltaLon) ) *
                        (avgEarthRadius(this, coordinate));

        return distance;
}
```

Die Methode arbeitet mit dem Kosinussatz der sphärischen Geometrie und geht dabei von einer perfekten Kugel aus. Um die Streckenlänge möglichst präzise zu berechnen, wird mit der Methode **avgEarthRadius()** der Durchschnitt der Erdradien beider Koordinaten berechnet. Die Berechnung der Strecke wird im Aktualisierungsprozess verwendet, um zu prüfen, ob sich ein Ortungsgerät eine signifikante Distanz bewegt hat und demnach

aktualisiert werden kann. Außerdem können spezielle Events mithilfe dieser Methode definiert werden.

5.3.2 GPSPolygon

Ein Polygon ist ein zweidimensionales Vieleck. Das bedeutet, dass es eine geometrische Figur mit beliebig vielen Eckpunkten aufspannt. Die entsprechende Klasse heißt BoundingPolygon und enthält eine Instanzvariable einer Liste mit allen Eckpunkten als Wgs-Coordinates. Eine BoundingPolygon kann genutzt werden, um das geografische Gebiet einer Area zu repräsentieren. Beim Erzeugen eines BoundingPolygons wird der Flächeninhalt in Quadratmetern ermittelt, da diese Abmessung allgemein interessant für manche Nutzer sein mag. Die wichtigste Operation dieser Klasse ist allerdings die Prüfung, ob sich eine Koordinate innerhalb des Polygons befindet. Die verwendete Berechnung für GPS-Koordinaten ist zu komplex, um sie hier mithilfe des Codes darzustellen. Die Idee der Methodenimplementation ist jedoch recht einfach zu erklären. Man versucht einen Strahl von außerhalb des Polygons zu zeichnen, der auf dem betrachteten Punkt P endet. Falls der Strahl eine ungerade Anzahl an Kanten des Polygon geschnitten hat, liegt der Punkt innerhalb des Polygons, sonst liegt er außerhalb. Das Verfahren ist in Abbildung 5.1 veranschaulicht.

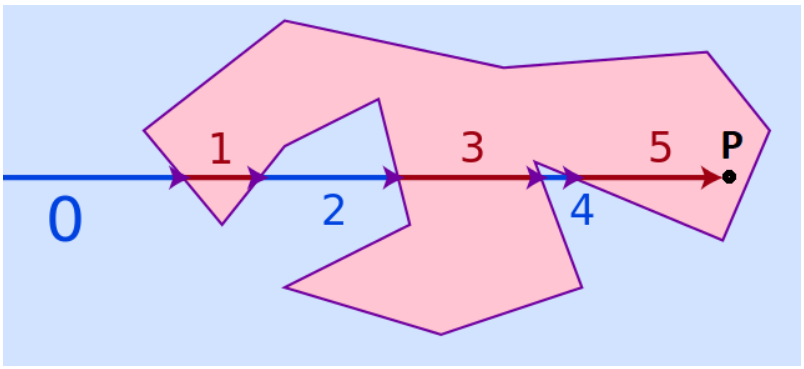

Abbildung 5.1: Strahl-Methode des Punkt-in-Polygon-Tests[1]

Da bei den zu erwarteten Flächen von Polygonen der Areas, viel mehr Fläche außerhalb der Polygone liegen wird, wird vor der Ausführung der Strahl-Methode eine Bedingung abgefragt, die wesentlich schneller erkennen kann, falls eine Koordinate weit von dem Polygon entfernt liegt. Dafür wird ein minimal umschließendes Rechteck als GPSBounding-

[1] Quelle: [Melchoir, 2007]

Box um das Polygon gelegt und geprüft, ob die Koordinate außerhalb dieses Rechtecks liegt. Wenn die Koordinate außerhalb des Rechtecks liegt, kann die Koordinate unmöglich innerhalb des Polygons liegen.

5.3.3 GPSBoundingBox

Eine BoundingBox spannt ein rechtwinkliges Rechteck mit 4 Koordinaten als dessen Eckpunkte auf. Die Klasse der BoundingBox repräsentiert das geografische Gebiet von den Maps. Rechteckige Gebiete von Areas werden ebenfalls durch BoundingBoxes dargestellt, statt durch BoundingPolygons.
Ein Rechteck ist eine Unterklasse der Polygone. Eine BoundingBox ist somit ein Spezialfall des BoundingPolygons. Sie verhält sich entsprechend analog zum BoundingPolygon und erbt von der Klasse. Die Funktion der Enthaltenseinsprüfung einer Koordinate wird in der BoundingBox allerdings überschrieben.

Für das spezielle Rechteck, bei dem alle Seiten entweder parallel zu den Breiten- oder den Längengraden verlaufen (Rotation = 0), reichen ein paar einfache Vergleiche der Breiten- und Längengrade mit der Koordinate des Punktes aus, um zu ermitteln, ob der Punkt innerhalb der BoundingBox liegt. Ein solch unrotiertes Rechteck wird, wie im vorherigen Abschnitt erwähnt, bei der Enthaltenseinsprüfung des Polygons verwendet, um früh und effizient zu erkennen, falls der Punkt weit von dem Polygon entfernt liegt.
Für ein rechtwinkliges Rechteck mit beliebiger Ausrichtung bzw. Rotation, muss noch eine weitere Prüfung durchgeführt werden, die mithilfe des Skalarproduktes berechnet, ob der Punkt sich in dem Rechteck befindet. Der Quelltext zu der soeben beschriebenen Funktion befindet sich im Anhang).

5.3.4 Eventhandling

Die Kommunikation des EventListeners mit dem EventHandler läuft über eine blockierende Queue, die typischerweise dazu verwendet wird das Erzeuger-Verbraucher-Muster zu implementieren. Der EventListener legt die erzeugten Events in die Queue, wobei diese eine unbegrenzte Kapazität besitzt, sodass die Threads, die die Events in die Queue ablegen, niemals blockiert werden. Der EventHandler läuft auf einem eigenen Thread und entnimmt die Events zum Veröffentlichen auf dem Message Broker. Sobald alle Events

der Queue verarbeitet wurden, wird der EventHandler an der Queue blockiert und wartet auf neue Events zum Verarbeiten.

Mithilfe der geografischen Formen von Polygonen und Rechtecken können verschiedene Events definiert werden. Zu diesem Zeitpunkt wurden insgesamt nur zwei ähnliche Events definiert. Das TagEnteredEvent wird ausgelöst, wenn ein Tag eine Area betritt und das TagExitEvent wird ausgelöst, wenn ein Tag eine Area verlässt. Mehrere Tests dieser Events haben die Funktionsfähigkeit bewiesen.

5.4 Ausgabedaten für die Datenanalyse

Im Laufe dieser Arbeit wurde parallel ein schmales Programm von der Data-Analytics-Abteilung von Lufthansa Industry Solutions programmiert. Dieses Programm stellt einen einfachen HTTP-Webservice mit einer Funktion bereit, an den die Daten über einen HTTP-POST an das Programm geschickt werden können. Die DataAnalyticsOutput-Komponente ist noch nicht vielseitig einsetzbar und soweit lediglich auf exakt diese Aufgabe und Schnittstelle zugeschnitten. Wenn die Komponente aktiviert wird, sendet sie die gesamten TagDetails-Daten des TagCaches nach festen periodischen Intervallen an das Programm.

5.5 Cache-Persistierung

Die Cache-Persistierung konnte genau nach Plan umgesetzt werden und funktioniert einwandfrei. Für das Laden bei Systemstart und dem Persistieren beim Herunterfahren wurden Java-Annotationen von Spring verwendet, die es ermöglichen eine Funktion augenblicklich nach dem Initialisieren und unmittelbar vor dem Zerstören einer Komponente aufzurufen. Diese Aufgaben werden daher automatisch über Spring erledigt.

6 Fazit

Das Fazit fasst die Kapitel dieser Arbeit zusammen und beschreibt die gewonnenen Erfolge des Softwareprojektes in Retrospektive. Der abschließende Ausblick gibt einen Überblick über voraussichtliche Entwicklungen und interessante Anwendungsbereiche der RTLS-Middleware.

Das Ziel dieser Arbeit war die Ausarbeitung und Realisierung einer Middleware, die einen einheitlichen Dienst für die Verwaltung und die Bereitstellung von mannigfaltigen Lokalisierungsdaten für ein Unternehmen darstellt. Die Grundidee der Middleware ist, eine intermediäre Verteilerstelle für Lokalisierungsdaten zwischen Anwendungen und Lokalisierungssystemen zu schaffen. In der Anforderungsanalyse (Kapitel 3) wurden während der gesamten Projektdauer die Ziele und Anforderungen mithilfe eines iterativen Vorgehens ausgearbeitet. Die Anforderungsspezifikation des Projektziels enthält eine vollständige und ausführliche Darstellung der ermittelten Anforderungen aus unterschiedlichen Blickwinkeln. Aus den Systemanforderungen konnte eine geeignete Softwarearchitektur (Abschnitt 4.1) entwickelt werden. Ausgehend von der groben Architektur wurden Problematiken und notwendige Überlegungen in Hinsicht auf wichtige Aspekte des Softwareentwurfs mittels Fragestellungen (Abschnitt 4.1.2) formuliert. Die Fragen wurden teilweise direkt beantwortet oder eingegrenzt und im weiteren Verlauf des Softwareentwurfs beantwortet. Der Entwurf wurde zum größten Teil konkretisiert, indem die zu erstellende Software aus vier verschiedenen Sichten betrachtet wurde und resultierende Entwurfsergebnisse dieser Sichtweisen schriftlich und visuell festgehalten wurden (Abschnitt 4.2).
Das Ziel der Realisierung (Kapitel 5) ist die Implementierung eines Softwaresystems, das die definierten Anforderung möglichst vollständig erfüllt, indem es sich eng an dem Bauplan des Softwareentwurfs orientiert. In dem Kapitel der Realisierung wurden lediglich vom Entwurf abweichende Implementationsdetails genannt und besondere Programmierentscheidungen der Komponenten erläutert. Das Kapitel wurde anhand der Implementierung der Hauptprozesse, namentlich REST-Anfragen (5.1), Datenaktualisierung (5.2), Verbindungsverwaltung (5.3), Datenausgabe für die Datenanalyse (5.4) und Cache-Persistierung (5.5), strukturiert.

Mittels wiederholter Usability-Tests konnten einige fehlerhafte Verhaltensweisen korrigiert und eine zufriedenstellende Funktionsfähigkeit der RTLS-Middleware sichergestellt werden. Die verwendeten Vorgehensweisen und Software-Engineering-Methoden haben dem Projekt eine klare Struktur und einen fein ausgearbeiteten Rahmen geschaffen, weshalb große Umplanungen oder spontane Fehlentscheidungen vermieden werden konnten. Die klar kommunizierten Anforderungen sowie die vorangehende Entwurfsplanung ermöglichten eine effiziente und unkomplizierte Programmierung.
Die nicht-funktionale Anforderung mehrere hundert Tags in der RTLS-Middleware verwalten zu können, konnte mithilfe eines Lasttests bewiesen werden, bei dem 1000 bewegliche Tags im Quuppa-Simulator simuliert wurden. Die Rate, mit der die Daten aktualisiert wurden, betrug in diesem Versuch 2 Sekunden. Anhand einer Veränderung der Rate und der Anzahl an Tags, könnte man die Limitierung des Systems im Betrieb feststellen, die allerdings abhängig von der Hardware und Auslastung der Laufzeitumgebung ist. Die Erfüllung der Anforderung mit einer Aktualisierungsrate von 2 Sekunden wurde jedoch bereits als zufriedenstellender Erfolg wahrgenommen.
Durch die gewählte Aufteilung und Auswahl von Komponenten, die konsequente Beachtung von objektorientierter Abstraktion und durch die Verwendung von Entwurfsmustern, sollte zudem keine lange Einarbeitungszeit für andere Entwickler nötig und eine Weiterentwicklung unkompliziert sein.

Bevor die RTLS-Middleware in tatsächlichen Betrieb gehen kann, sind noch einige Maßnahmen notwendig. Nichtsdestoweniger konnte diese Arbeit das Ziel, einen Beitrag für die Weiterentwicklung der Middleware zu leisten, wahrscheinlich erreichen. Aufgrund der entwickelten Struktur der RTLS-Middleware ist die Erweiterbarkeit um weitere Lokalisierungssysteme effizient implementierbar. Genauer gesagt müssen lediglich die Komponenten der Verbindungsverwaltung und des Aktualisierungsprozesses erweitert werden. Welche Lokalisierungssysteme zusätzlich benötigt werden, ist abhängig von den Anfroderungsansprüchen potenzieller Kunden der RTLS-Middleware. Ein hilfreicher Zusatz, um die Effektivität der RTLS-Middleware zu maximieren, wäre eine dauerhafte Abschätzung der benötigten und verfügbaren Ressourcen, damit die Aktualisierungsrate der Daten dynamisch minimiert werden kann.

Die RTLS-Middleware kann insbesondere für Applikationen des Logistiksektors von Interesse sein, obgleich die Anwendbarkeit von Lokalisierungslösungen fast grenzenlos ist. Beispielsweise machen sich Fast-Food-Restaurants bereits Lokalisierung zunutze, um die Bestellungen ihrer Kunden an die richtigen Tische zu bringen. Andere Anwendungsbereiche, wie eine zentralisierte Lokalisierung von Polizeiwagen oder von Sicherheitskräften

für eine Optimierung der Gebietsabdeckung oder eine Routenfindung für Patienten im Krankenhaus, sind vorstellbar. Ein Frontend wurde parallel für die RTLS-Middleware entwickelt, welches zukünftig eine Benutzerschnittstelle für die REST-API bieten und die Lokalisierungsdaten in Echtzeit visuell auf einer Weltkarte darstellen soll. Über dieses Frontend könnten administrative Aufgaben erledigt und Positionen und Wege von georteten Objekten vom Kunden kontrolliert werden.

Literaturverzeichnis

Helmut Balzert. *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. Spektrum Akademischer Verlag, 2011. ISBN 9783827417060. URL https://www.amazon.de/Lehrbuch-Softwaretechnik-Entwurf-Implementierung-Installation/dp/3827417066.

Alan Bansky. *Wireless Positioning Technologies and Applications*. Artech House Publishers, 2008. ISBN 9781596931305. URL https://www.amazon.de/Wireless-Positioning-Technologies-Applications-Technology/dp/1596931302.

Joshua Bloch. Qcon 2018. In *A Brief, Opinionated History of the API*, 2014. URL https://www.infoq.com/presentations/history-api/.

David Booth and Hugo Haas. Web services architecture. *W3C Working Group Note 11 February 2004*, 2004. URL https://www.w3.org/TR/ws-arch/.

Mike Clark, Peter Fletcher, and Jeffrey J. Hanson. *Web Services Business Strategies and Architectures*. Apress, 2013. ISBN 9781430253563. URL https://books.google.de/books?id=2EonCgAAQBAJ.

Karl Eilebracht and Gernot Starke. *Patterns kompakt: Entwurfsmuster für effektive Software-Entwicklung*. Springer, 2013. ISBN 9783642347184. URL https://www.springer.com/de/book/9783642347184.

Joachim Goll. *Entwurfsprinzipien und Konstruktionskonzepte der Softwaretechnik*. Springer, 2018. ISBN 9783658200558. URL https://www.springer.com/de/book/9783658200558.

Wolfgang A. Halang and Herwig Unger. *Industrie 4.0 und Echtzeit*. Springer, 2014. ISBN 9783662451090. URL https://www.springer.com/de/book/9783662451083.

Martin Helmich. Restful webservices: Was ist das überhaupt?, 2013. URL https://www.mittwald.de/blog/webentwicklung-design/webentwicklung/restful-webservices-1-was-ist-das-uberhaupt.

ITWissen. Rtls (realtime location system), 2017. URL https://www.itwissen.info/RTLS-realtime-location-system-Echtzeit-Lokalisierungssystem.html.

Wolfgang Keller. *Enterprise Application Integration: Erfahrungen aus der Praxis*. dpunkt.verlag, 2002. ISBN 9783898641869. URL https://www.amazon.de/exec/obidos/ASIN/3898641864.

Ajay Malik. *RTLS For Dummies*. For Dummies, 2009. ISBN 9780470398685. URL https://www.amazon.de/RTLS-Dummies-Ajay-Malik/dp/047039868X.

Melchoir, 2007. URL https://commons.wikimedia.org/wiki/File:RecursiveEvenPolygon.svg. Attribution: Melchoir [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)].

Helmuth Partsch. *Requirements Engineering systematisch*. Springer, 2010. ISBN 9783642053580. URL https://www.springer.com/de/book/9783642053573.

Quuppa. The quuppa intelligent locating system overview, 2019. URL https://quuppa.com/technology/overview/.

Jane Radatz. Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, 1990.

Karl Heinz Rau. *Agile objektorientierte Software-Entwicklung*. Springer, 2016. ISBN 9783658007768. URL https://www.springer.com/de/book/9783658007751.

RedHat. Was ist middleware?, 2019. URL https://www.redhat.com/de/topics/middleware/what-is-middleware.

Kai Spichale. *API-Design: Praxishandbuch für Java- und Webservice-Entwickler*. dpunkt.verlag, 2019. ISBN 9783864906114. URL https://www.amazon.de/API-Design-Praxishandbuch-f%C3%BCr-Java-Webservice-Entwickler/dp/3864903874.

Gernot Starke. *Effektive Software-Architekturen*. Carl Hanser Verlag, 2017. ISBN 9783446452077. URL https://www.amazon.de/Effektive-Softwarearchitekturen-Ein-praktischer-Leitfaden/dp/3446452079.

Craig Walls. *Spring Boot in Action*. Manning Publications Co., 2015. ISBN 9781617292545. URL https://www.amazon.de/Spring-Boot-Action-Craig-Walls/dp/1617292540.

Rüdiger Weißbach, Andrea Herrmann, and Eric Knauss. *Requirements Engineering und Projektmanagement*. Springer, 2013. ISBN 9783642294327. URL https://www.springer.com/de/book/9783642294310.

Stefan Zörner. *Softwarearchitekturen dokumentieren und kommunizieren*. Carl Hanser Verlag, 2012. ISBN 9783446429246. URL https://www.amazon.de/Softwarearchitekturen-dokumentieren-kommunizieren-Entscheidungen-nachvollziehbar/dp/3446429247.

A Anhang

Es folgt der Java-Quelltext der Enthaltenseinsprüfung einer WgsCoordinate in Bezug auf eine BoundingBox:

```
public boolean isInside(WgsCoordinate coordinate) {
        double latitude = coordinate.getLatitude();
        double longitude = coordinate.getLongitude();
        // Vergleiche Punktkoordinaten mit Eckpunkten
        if(latitude >= this.corners[1].getLatitude() &&
                latitude <= this.corners[3].getLatitude() &&
                longitude >= this.corners[0].getLongitude() &&
                longitude <= this.corners[2].getLongitude()) {
                if(rotation == 0.0) {
                        return true;
                } else {
                        if(isInsideAny(latitude, longitude)) {
                                return true;
                        }
                }
        }
        return false;
}
```

```
// Fuer ein Rechteck mit beliebiger Ausrichtung
private boolean isInsideAny(double lat, double lon) {
        double[] vectorAB = new double[] {
                this.corners[0].getLatitude() -
                this.corners[3].getLatitude(),
                this.corners[0].getLongitude() -
                this.corners[3].getLongitude()};
        double[] vectorBC = new double[] {
                this.corners[1].getLatitude() -
                this.corners[0].getLatitude(),
                this.corners[1].getLongitude() -
                this.corners[0].getLongitude()};
        double[] vectorARef = new double[] {
                lat - this.corners[3].getLatitude(),
                lon - this.corners[3].getLongitude()};
        double[] vectorBRef = new double[] {
                lat - this.corners[0].getLatitude(),
                lon - this.corners[0].getLongitude()};

        double dotABsquare = vectorAB[0] * vectorAB[0] +
                vectorAB[1] * vectorAB[1];
        double dotBCsquare = vectorBC[0] * vectorBC[0] +
                vectorBC[1] * vectorBC[1];
        double dotABxARef = vectorAB[0] * vectorARef[0] +
                vectorAB[1] * vectorARef[1];
        double dotBCxBRef = vectorBC[0] * vectorBRef[0] +
                vectorBC[1] * vectorBRef[1];

        if((0 <= dotABxARef && dotABxARef <= dotABsquare) &&
          (0 <= dotBCxBRef && dotBCxBRef <= dotBCsquare)) {
                return true;
        }
        return false;
}
```

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Erstellung einer Middleware zur Abbildung von unterschiedlichen Lokalisierungsdatenquellen auf ein einheitliches Koordinatensystem

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original