HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Bachelor-Arbeit im Frühlingssemester 2011

Lokalisierung medizinischer Geräte im Spitalumfeld

Felix, Franziska Ahlenstorf, Andreas

15. Juni 2011

Betreuer:	Dipl. El. Ing. ETH Beat Stettler
Projektpartner:	Universitätsspital Zürich, Zürich
Experte:	Roman Mendelin, Universitätsspital Zürich
Gegenleser:	Dipl. El. Ing. ETH Hans Rudin

Inhaltsverzeichnis

Teil I

Einleitung

1 Eigenständigkeitserklärung

Franziska Felix und Andreas Ahlenstorf (nachfolgend: wir) erklären hiermit, dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in dem vorliegenden Dokument erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde. Wir haben sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben.

Rapperswil-Jona, 15. Juni 2011

Ort, Datum

Franziska Felix

Rapperswil-Jona, 15. Juni 2011

Ort, Datum

Andreas Ahlenstorf

2 Aufgabenstellung

Der in der Studienarbeit [1] erarbeitete Prototyp soll im Rahmen dieser Arbeit ausgebaut werden. Dabei sollen neben der Erweiterung der Gerätelokalisierung auch die Grundsteine für die Lokalisierung von Personen – sei das Krankenhauspersonal oder Patienten – gelegt werden.

Um dies zu erreichen, sollen weitere Lokalisierungswerkzeuge angebunden werden können wie zum Beispiel die CiscoWorks LAN Management Solution (LAN-Lokalisierung) oder Smartphones (Satellitenortung). Da Geräte über mehrere Sensoren (z.B. WLAN und LAN) und Personen über mehrere Geräte (z.B. Smartphone und Laptop) lokalisiert werden können, sollen die gelieferten Positionsmeldungen intelligent gefiltert und kombiniert werden können, um das Risiko für Falschmeldungen zu reduzieren. Zusätzlich sollen anhand der Positionsangaben Ereignisse ausgelöst und an andere Systeme weiter geleitet werden können. Beispiele sind Informationen über Zonenwechsel oder Nachrichten über Statusänderungen, wenn ein Arzt beispielsweise einen Operationssaal betritt und damit in den Status «opertiert» wechselt.

Gesamthaft gesehen soll die Lösung hinsichtlich Anpassungen und Weiterentwicklungen offen sein. Am Ende soll ein erweiterter Prototyp vorliegen, der die Grundlage eines Pilotversuchs bilden kann.

3 Abriss

Um die betrieblichen Abläufe zu unterstützen, ist es für Spitäler und Pflegeheime von grossem Interesse, mobile Medizinalgeräte und Personen lokalisieren zu können. Das Anwendungsgebiet reicht von der Patientenversorgung über die Wartung bis hin zur Qualitätssicherung und Abrechnung. Es existiert zwar eine Reihe von Lokalisierungslösungen. Diese kümmern sich aber meist nur um die Positionsbestimmung. Hinzu kommt, dass sie sich nicht zur Lokalisierung jeder Geräte- beziehungsweise Personenklasse eignen. Deshalb soll ein auf Lokalisierungsinformationen basierendes Informationssystem geschaffen werden, das die Lokalisierungslösungen verschiedener Hersteller integrieren und aufgrund den von ihnen gelieferten Positionen die betrieblichen Abläufe unterstützen kann. In der gleichnamigen Studienarbeit aus dem Jahre 2010 wurden von der Projektgruppe die Anforderungen an ein solches Systems erfasst und ein Prototyp erstellt, der einzelne und die nächstliegenden Geräte eines bestimmten Typs aufspüren konnte. Im Rahmen der Bachelor-Arbeit sollte der Prototyp ausgebaut werden um:

- Integration weiterer Lokalisierungswerkzeuge
- Intelligente Kombination der gelieferten Positionsmeldungen, beispielsweise um Falschmeldungen auszufiltern
- Erzeugung von Ereignissen abhängig von den gelieferten Positionsmeldungen
- Bereitstellung einer robusten und skalierbaren Systemarchitektur
- Erweiterung der durch den Prototypen bereitgestellten Funktonalität

Zu diesem Zweck wurde ein verteiltes System mit einer polyglotten Persistenzarchitektur erstellt, die als Katalog, Geoinformationssystem und Historie für die rund um die Uhr erfassten Positionen dient. Eine Rule Engine ermöglicht eine Kombination der Positonsmeldungen, um die Qualität der Positionsangaben zu verbessern und positionsabhängige Präsenzinformationen wie Ereignisse zu erzeugen. Geometrie und Topologie für das Geoinformationssystem werden mit einem selbstentwickelten Werkzeug aus digitalisierten Grundrissen der Gebäude gewonnen. Zur Abfrage des Informationssystems steht für die Anwender ein Webinterface bereit, das sich bei Bedarf um Smartphone-Apps ergänzen lässt. Für die Analyse der Positionshistorie können Map Reduce Jobs ausgeführt werden.

Der Prototyp aus der Studienarbeit konnte damit soweit ausgebaut werden, dass ein Pilotversuch in einem einzelnen Gebäude durchgeführt werden kann. Dabei können die wesentlichen Funktionen – Positionserfassung mit mehreren Lokalisierungswerkzeugen, Positionsinterpretation, Ereignisauslösung und Auswertung der Positionshistorie – erprobt werden.

4 Überblick

4.1 Ausgangslage

Um die betrieblichen Abläufe zu unterstützen, ist es für Einrichtungen der Gesundheitsversorgung wie Spitäler oder Pflegeheime von grossem Interesse, mobile Medizinalgeräte und Personen lokalisieren zu können, Das Anwendungsgebiet reicht von der Patientenversorgung (Auffinden des benötigten Medizinalgeräts) über die Wartung (Auffinden der zu wartenden Geräte) bis hin zur Qualitätssicherung (z.B. hat der Patient alle Untersuchungsschritte durchlaufen) und Abrechnung (z.B. welche Abteilung welche Geräte wie lange benutzt hat). Insbesondere der letzte Punkt gewinnt durch die Einführung neuer Abrechnungsmodelle wie Fallpauschalen eine immer grössere Bedeutung. Es existiert zwar eine Reihe von Anbietern für Lokalisierungslösungen, die sich aber meist nur um die reine Positionsbestimmung kümmern. Hinzu kommt, dass sie sich nicht zur Lokalisierung jeder Geräte- beziehungsweise Personenklasse gleichermassen eignen. Deshalb soll anhand der Anforderungen des Universitätsspitals Zürich, eines der grössten Spitäler der Schweiz, ein auf Positionsdaten basierendes Informationssystem geschaffen werden, das die Lokalisierungslösungen verschiedener Hersteller integrieren und aufgrund der von ihnen gelieferten Positionsmeldungen die betrieblichen Abläufe unterstützen kann.

In der gleichnamigen Studienarbeit aus dem Jahre 2010 [1] wurden von der Projektgruppe die Anforderungen an ein solches auf Lokalisierungsinformationen basierenden Informationssystems erfasst und ein erster Prototyp erstellt, der einzelne und die nächstliegenden Geräte eines bestimmten Typs aufspüren konnte. In dieser Arbeit soll nun der Prototyp ausgebaut werden, wobei folgende Ziele zu erreichen sind:

- Integration weiterer Lokalisierungswerkzeuge
- Intelligente Kombination der gelieferten Positionsmeldungen, beispielsweise um Falschmeldungen auszufiltern
- Erzeugung von Ereignissen abhängig von den gelieferten Positionsmeldungen
- Bereitstellung einer robusten und skalierbaren Systemarchitektur
- Erweiterung der durch den Prototypen bereitgestellten Funktonalität

4.2 Vorgehen

In einer ersten Phase wurde analysiert, wie sich die geforderte Funktionalität unter verschiedenen Gesichtspunkten wie Leistungsanforderungen, Flexibilität, Wartbarkeit und Datenschutz am besten realisieren lässt und welche architektonischen Herausforderungen sich daraus ergeben. Dabei waren etliche Parameter wie Leistungsmerkmale oder funktionale wie nicht-funktionale Anforderungen weitestgehend aus der Studienarbeit bekannt. Wichtige Punkte der Analyse umfassten Fragen wie wie sich die sehr unterschiedlich funktionierenden Lokalisierungswerkzeuge

integrieren lassen, wie sich unterschiedliche Positionsmeldungen für ein Gerät oder eine Position einfach und flexibel interpretieren und die grossen zu erwartenden Datenmengen speichern und zur effizienten Analyse bereitstellen lassen. Wesentlich war dabei die Berücksichtigung von Auflagen durch die Datenschutzgesetze, die im Medizinalbereich wesentlich strenger sind als im Alltag.

Unter Berücksichtigung der Analyseergebnisse wurde das bestehende Anwendungsdesign überarbeitet, ergänzt und neue Komponenten geplant. Die wichtigsten Resultate dieser Phase sind die Entwicklung einer neuen Systemarchitektur, bei der die Auftrennung nach Aspekten und Verbesserung von Skalierbarkeit und Ausfallsicherheit im Zentrum stand, und die Definition einer modularen Konnektorschnittstelle für Lokalisierungswerkzeuge, die eine unabhängige Entwicklung vom Gesamtsystem und die Verwendung beliebiger Kombinationen von Lokalisierungswerkzeugen unterschiedlichster Hersteller ermöglicht.

Die Umsetzung wurde in einem iterativen Entwicklungsmodell in Angriff genommen, wobei in einer ersten Phase die neue Architektur und die Konnektoren realisiert wurden. Nachher erfolgte die Anbindung der aus der Studienarbeit vorhandenen Komponenten. Zum Abschluss wurde die Funktionalität punktuell ergänzt.

4.3 Ergebnisse

Der Prototyp aus der Studienarbeit konnte soweit ausgebaut werden, dass ein Pilotversuch in einem einzelnen Gebäude durchgeführt werden kann. Dabei können die wesentlichen Funktionen – Positionserfassung mit mehreren Lokalisierungswerkzeugen, Positionsinterpretation, Ereignisauslösung und Auswertung der Positionshistorie – erprobt werden.

Konkret wurde während der Bachelor-Arbeit der Prototyp aus der Studienarbeit in eine skalierbare, modulare Systemarchitektur überführt, bei der die einzelnen Komponenten nur noch für jeweils eine einzige Aufgabe wie Lokalisierung oder Abfrage zuständig sind. Um den Anforderungen an Datenspeicherung und Analysefähigkeit entgegen zu kommen, wurde eine polyglotte Persistenzarchitektur entworfen, wobei ein Relational Database Management System (RDBMS) als Katalog und Geoinformationssystem dient während zur Speicherung und Analyse der Logdaten ein verteilter Document Store zum Einsatz kommt. Die Datenanalyse wird mittels Map Reduce durchgeführt. Messaging und ein verteilter Key Value Store wurden zur Integration der Systeme genutzt.

Hinsichtlich der Anbindung von Lokalisierungswerkzeugen wurde nebst dem bereits aus der Studienarbeit vorhandenen Ekahau-Konnektor ein Konnektor für die Cisco Works LAN Management Solution und für eine vom Institut für Software der HSR entwickelte iPhone-App erstellt. Der Cisco-Works-Konnektor ermöglicht, herauszufinden, an welchem LAN-Port ein Gerät angeschlossen ist, während mit der iPhone-App die Position eines Smartphones aufgrund von Zell-Informationen aus dem Mobilfunk-Netz und Satellitenortung bestimmt werden kann. Ebenso wurde der Ekahau-Konnektor umgestaltet, so dass die Position eines Senders direkt mittels einer Koordinatentransformation gewonnen werden kann, ohne dass das Ekahau-System noch speziell konfiguriert werden müsste, wie das bisher der Fall war. Weitere Lokalisierungswerkzeuge lassen sich über definierte Schnittstellen einfach integrieren.

Zu lokalisierende Geräte und Personen können über eine Webservice-Schnittstelle registriert werden, woraufhin sie in regelmässigen Intervallen rund um die Uhr lokalisiert werden. Benut-

zerdefinierte Regelmengen, die auf die Positionsmeldungen angewendet werden, erlauben, aus widersprüchlichen Meldungen die korrekte Position zu destillieren und Ereignisse auszulösen, beispielsweise den Status eines Arztes auf *operiert* zu setzten, wenn dieser einen Operationssaal betritt.

Die Digitalisierung der Gebäudegeometrie wurde verbessert und kann nun nicht mehr nur mit einzelnen Stockwerken umgehen, sondern Stockwerke über Treppen und Fahrstühle hinweg miteinander verbinden, sodass aus einer Menge von zweidimensionalen Plänen ein echtes zweieinhalbdimensionales Modell eines Gebäudes entsteht. So können beispielsweise Distanzen zwischen Punkten auf verschiedenen Stockwerken berechnet werden.

Die Abfrage der Informationen – wie die Position des nächstliegenden Geräts eines bestimmten Typs – erfolgt wie bis anhin über ein Webinterface, das nun auch die Position des Anfragestellers über die Lokalisierung ermittelt. Ausserdem wurde es um die zusätzlich realisierten Use Cases ergänzt. Gesamthaft wurden bis zur Drucklegung des Berichts 11 weitere Use Cases umgesetzt.

4.4 Ausblick

Nachdem nun ein Prototyp bereitsteht, sollten in einer Erprobungsphase die erarbeiteten Konzepte überprüft und angepasst werden. Ein Hauptaugenmerk dürfte darauf liegen, zu ermitteln, welches Lokalisierungswerkzeug wofür geeignet ist, wie die Genauigkeit der Positionsmeldungen verbessert werden kann und wie gut sich mithilfe der Regeln Falschmeldungen ausfiltern oder Ereignisse erkennen lassen. Ebenso sollten die Use Cases ausprobiert und an die tatsächlichen Bedürfnisse angenähert werden. In Frage kommende Ergänzungen dürften zusätzliche Lokalisierungswerkzeuge (z.B. RFID), Abfrageschnittstellen (z.B. Smartphone-Apps) und weitere Use Cases sein.

Mittelfristig dürfte die Weiterentwicklung zu einem produktionsreifen System anstehen, wobei der Prototyp robuster gemacht und viele bislang vernachlässigte Aspekte wie Sicherheit und Übereinstimmung mit gesetzlichen Anforderungen – beispielsweise hinsichtlich vollständiger Nachvollziehbarkeit – betrachtet werden müssen.

Beim längerfristigen Ausbau dürfte die zuverlässige Behandlung von Alarmen (z.B. Patientennotruf), welche auf Sensoren ausgelöst werden können, und die schnellere Reaktion auf Positionsmeldungen mittels Event-Verarbeitung im Zentrum stehen. Beides ist aber stark von den Fähigkeiten der Lokalisierungswerkzeuge abhängig, die aber bislang hinsichtlich Genauigkeit, Zuverlässigkeit und Programmierschnittstellen noch nicht weit genug sind.

5 Danksagung

Wir möchten an dieser Stelle den Unterstützern danken, ohne deren Hilfe die Realisierung der Bachelor- und Studienarbeit nicht möglich gewesen wäre. In chronologischer Reihenfolge:

- Andreas Maurer und Roman Mendelin vom Universitätsspital Zürich für Ihr grosses Interesse, die vielen hilfreichen Vorschläge und positive Bestärkung
- Viktoria Slukan vom Institut für Raumentwicklung IRAP der HSR und Stefan Keller vom Institut für Software der HSR für ihre Unterstützung bei der Erfassung von Geometrie und Topologie. Ohne sie wüssten wir wohl heute noch nicht, wie wir Stockwerkgrundrisse ins Geoinformationssystem einlesen können.
- Den beiden HSR-Mathematikern Andreas Müller und Olaf Tietje für ihre Unterstützung bei der Realisierung der Transformation der von der Ekahau Positioning Engine gelieferten Koordiaten, wodurch wir ein erheblich genaueres und effizienteres Verfahren erhalten haben als wir uns ursprünglich selber ausgedacht haben.
- Wolfgang Giersche von Zühlke Engineering, der uns mit viel Geduld dabei geholfen hat, die Anwendungskontexte von Spring zu bändigen und den Mechanismus zum Laden der Konnektoren richtig zu implementieren.

Unser Dank gilt auch unserem Betreuer-Team von INS Institute for Networked Solutions der HSR, bestehend aus unserem Betreuer Beat Stettler, Maurin Egler, Marco Facetti, Markus Kolb und Michael Schneider, die uns während des gesamten Projekts zur Seite standen, sowie unserem Gegenleser Hans Rudin vom Institut für Software der HSR.

Teil II

Analyse

6 Geometriedigitalisierung

6.1 Problemstellung

In der Studienarbeit [1] wurde bereits beschreiben, wie die Topologie erfasst werden kann. Dabei wurde die vorgesehene Arbeitsweise beschreiben. Ziel ist nun, anhand der beschriebenen Vorgehensweise die eingeschränkten und nicht vorhandenen Punkte so weiterzuentwickeln, dass folgende Bereiche abgedeckt sind:

- Unterstützung von Polygonen mit inneren Ringen, also die Möglichkeit, Räume mit Aussparungen zu verarbeiten
- Einlesen mehrerer Shapefiles
- Verknüpfung von Stockwerken
- Allgemeines Refactoring

6.2 Lösungsansätze

Die in der Studienarbeit [1] beschriebene Arbeitsweise kann übernommen werden. Wobei die zusätzlichen Punkte wie zum Beispiel die inneren Ringe zusätzlich überdacht werden.

6.3 Umsetzung

Die Geometriedigitalisierung wird gemäss den in der Studienarbeit [1] beschriebenen Ablauf durchgeführt. In einem ersten Schritt werden die Sektoren und die dazugehörigen Metadaten ausgelesen. Anschliessend werden die Laufwege eingelesen und aufbereitet. In einem weiteren Schritt wird ein gewichteter, ungerichteter Graph aufgebaut und mittels eines All-Pairs-Shortest-Path-Algorithmus die kürzesten Pfade zwischen allen Punkten des Graphen vorberechnet. Zum Schluss werden die aus dem Ablauf erhaltenen Daten in Form von Structured Query Language (SQL)-Dateien auf der Festplatte gespeichert.

7 Lokalisierung

7.1 Problemstellung

Die vom Universitätsspital Zürich (USZ) definierten Use Cases (siehe Abschnitt 17, Seite 105) erfordern eine möglichst lückenlose, kontinuierliche Verfolgung von Geräten (und damit auch Personen), die über eine längere Zeit nachvollzogen werden können muss. Vorgesehene und mögliche Anwendungsszenarien:

- Nutzungsprofile
  - Welche Geräte werden wie viel genutzt/überhaupt verwendet?
- Abrechnung nach Nutzung der Geräte
  - Ableitung der Kostenstelle von Position (Gerät in Abteilung A wird Abteilung A zugerechnet) oder Patient
  - Berechnung der Nutzungsdauer anhand räumlicher Nähe von Patient und Gerät
- Qualitätssicherung
  - Hat Patient nach Behandlungsschritt X Behandlungsschritt Y durchlaufen?
- Überwachung
  - Hat Gerät Campus «verlassen»? (Diebstahl, unabsichtliche Mitnahme)
  - Wo ist Patient?
- Statusableitung
  - Arzt in Operationssaal ist *beschäftigt*, Arzt in Mensa ist *frei*

Diese Anforderungen können mit gelegentlicher Abfrage von Positionen nicht erfüllt werden. Statt dessen muss die Positionserfassung möglichst kontinuierlich erfolgen und auf das Eintreffen respektive Ausbleiben bestimmter Ereignisse reagiert werden können. Beispiele:

- Alarm, wenn Patient Zimmer verlässt (statt Sensormatten vor Zimmertüre)
- Alarm, wenn Gerät Campus verlässt
- Personenstatus (Arzt, Krankenpfleger) auf *abwesend* setzen, wenn Person Campus verlässt.

Um eine Verfolgung möglichst verschiedener Geräte und damit indirekt Personen zu ermöglichen, müssen verschiedene Lokalisierungswerkzeuge integriert werden können. Diese unterscheiden sich in:

- Art der Positionsangabe
- Art der Abfrage
  - Polling

  - Events
- Verhalten, wenn Gerät vom «Radar» verschwindet
  - Meldung, dass Gerät nicht mehr aufzufinden
  - Wird nicht bemerkt

Aufgrund der erfassten Daten müssen Reports von mehrmals täglich bis jährlich erstellt werden können. Aktuell verfügt das USZ über etwa 40'000 Geräte. Das bedeutet, dass der Lokalisierungsservice vorerst auf 50'000 Geräte ausgelegt werden sollte mit Potential zum Wachstum auf mindestens 100'000 Geräte, um den Lokalisierungsservice auch andernorts zum Einsatz bringen zu können. Daten müssen bis zu 2 Jahre aufbewahrt werden, um Reports für ganze Kalenderjahre zu ermöglichen.

7.2 Konnektoren

7.2.1 Problemstellung

Je nach Installation des Lokalisierungsservice müssen andere Lokalisierungswerkzeuge unterstützt werden. Dies kommt daher, da je nach Umgebung unterschiedliche Produkte und Kombinationen von Lokalisierungswerkzeugen (z.B. Radio Frequency Identification (RFID)/Wireless LAN (WLAN) oder Smartphone/WLAN) zum Einsatz kommen. Zusätzlich können unterschiedliche Produktgenerationen im Einsatz sein, die sich bezüglich Application Programming Interface (API) oder sogar Funktionsweise unterscheiden.

Bereits bei der Analyse der in Frage kommenden Lokalisierungswerkzeuge [1] (S. 17 ff.) hat sich gezeigt, dass nicht mit jedem Werkzeug auf die gleiche Weise interagiert werden kann. Die grössten Schnittmengen bilden die aktive Frage nach Positionen (Pull) oder die automatische Einlieferung von Positionsmeldungen (Push). Insbesondere bei den Werkzeugen, die nur von sich aus senden, muss bedacht werden, dass der Strom von Positionsmeldungen aus verschiedensten Gründen unterbrochen sein kann. Ein weiteres Problem, das bereits in [1] (S. 15 ff.) diskutiert wurde, ist die Vereinheitlichung der unterschiedlichen verwendeten Bezugssysteme. Um beispielsweise eine Positionsmeldung der LAN-Lokalisierung («Gerät X an Switch-Port Y») umwandeln zu können, muss ein Dictionary hinterlegt werden, mit dem sich der Switch-Port in eine Positionsangabe umwandeln lässt.

Eine weitere Aufgabe ist die Erzeugung von Events. Wer ist dafür zuständig? Wie können Ereignisse zuverlässig erzeugt werden, selbst wenn einmal keine Positionsmeldungen eintreffen (keine Meldung ist auch eine Meldung)?

7.2.2 Anforderungen

Zur Bewältigung der Problemstellung sollte es möglich sein, die Konnektoren für Lokalisierungswerkzeuge unabhängig vom Lokalisierungsservice zu entwickeln und sie bei Bedarf jederzeit austauschen zu können. Dabei gilt es, die unterschiedlichen Betriebsarten (Push, Pull) sowie Bezugssysteme zu berücksichtigen und zu überlegen, welche Aufgaben konnektorspezifisch sind und welche vom Lokalisierungsservice für alle Konnektoren erledigt werden können.

Es muss ausserdem eine Möglichkeit gefunden werden, wie Konnektoren Alarme einliefern können. Ebenso benötigten Konnektoren ein «Gedächtnis», das beispielsweise dazu dienen kann, ein Dictionary zur Übersetzung von Positionsmeldungen zu hinterlegen.

Bei der Gestaltung der Konnektoren muss beachtet werden, dass sich die Abfrage respektive Einlieferung der Meldungen von den Lokalisierungswerkzeugen auf verschiedene Rechner verteilen lassen können muss.

7.2.3 Lösungsansätze

Konnektor-Gestaltung

Um die Menge der Schnittstellen für die Konnektor-Integration in Grenzen zu halten, dürfte es sinnvoll sein, nur Schnittstellen für Push- und Pull-Konnektoren anzubieten, womit sich Kommunikationsmodelle wie Request-Reply und Producer-Consumer abbilden lassen. Damit sollte es möglich sein, jedes Lokalisierungswerkzeug in irgendeiner Art zu integrieren und Alarme entgegenzunehmen. Bei Push-Konnektoren sollte speziell die Einlieferung über Hyper Text Transfer Protocol (HTTP) (Servlet) und Java Message Service (JMS) (in Form eines `MessageListener`) berücksichtigt werden. Dies ermöglicht es auch relativ einfach, die Abfrage respektive die Einlieferung der Meldungen der Lokalisierungswerkzeuge auf verschiedene Rechner zu verteilen. Andere Kommunikationsmuster, beispielsweise Server-Push mittels HTTP, wie sie von Ekahau verwendet werden, können nicht sinnvoll umgesetzt werden, weil sie sich nicht auf mehrere Rechner verteilen und nur beschränkt überwachen lassen.

Bei Pull-Konnektoren sind nur das Abfragen des Lokalisierungswerkzeugs und die Umwandlung der Antwort in eine Position des jeweiligen Bezugssystems konnektorspezifische Aufgaben. Alle anderen Aufgaben wie die Umwandlung der Position des Bezugssystems in eine Sektorangabe oder die Beschaffung der Sensoradresse sind für jeden Konnektor identisch und können damit vom Lokalisierungsservice übernommen werden. Bei Push-Konnektoren kommt im Vergleich zu den Pull-Konnektoren eine Aufgabe dazu: Das Behandeln der HTTP- beziehungsweise JMS-Nachricht, da dies aufgrund der (möglicherweise) unterschiedlichen Nachrichten-Strukturen je Konnektor nicht global erledigt werden kann.

Bezugssystem

Wie bereits in [1] (S. 15 ff.) diskutiert, werden innerhalb des gesamten Lokalisierungssystems die Bezugssysteme der jeweiligen Landesvermessungen verwendet, da diese genauer sind als globale Bezugssysteme wie WGS84 (Längen- und Breitengrad). Dies hat Auswirkungen auf die Gestaltung der Konnektoren: Entweder muss ein Konnektor für jedes Bezugssystem angepasst werden, damit er die regional verwendeten Koordinaten berechnet, oder es muss ein Zwischenschritt eingeführt werden. Dies kann so aussehen, dass die Konnektoren alle WGS84 produzieren und im Lokalisierungsservice dann entsprechende Umrechnungsroutinen hinterlegt werden. Dies macht zwar die Konnektor-Entwicklung einfacher, hat aber Nachteile hinsichtlich Genauigkeit, da die Wechsel zwischen den Bezugssystemen oft nur über Näherungsformeln möglich sind. Im Fall der Schweizer Landeskoordinaten erhält man bei den in [4] (S. 11 ff.) beschriebenen Näherungsformeln eine Abweichung von unter 1 Zentimeter bei der Umrechnung Landeskoordinaten in WGS84, für den umgekehrten Weg dagegen eine Abweichung von bis zu 1 Meter.

Zusätzlicher Aufwand kann ausserdem bei Konnektoren entstehen, die Dictionaries verwenden, um beispielsweise eine Ethernet-Switch-Port-Angabe in Koordinaten umzusetzen. Um das Dictionary zu erstellen, ist es am einfachsten, die Koordinaten ihrer Position auf digitalen Grundrissen abzulesen. Die Koordinaten liegen aber normalerweise in der jeweiligen Landesvermessung vor, was bedeutet, dass sie bei Verwendung eines globalen Bezugssystems wie WGS84 erst einmal manuell umgerechnet werden müssten, bevor sie ins Dictionary eingefügt werden könnten.

Konnektor-Datenbank

Es lässt sich im Voraus schwer abschätzen, welche Informationen die Konnektoren für ihrem Betrieb brauchen, wie sie strukturiert sind und somit wie sie in Form einer Konnektor-Datenbank am besten angeboten werden können. Denn dies ist stark abhängig von den Anforderungen des jeweiligen Lokalisierungswerkzeugs und dessen weiterer Entwicklung. Eine weitere Herausforderung ist die Isolation der Konnektoren von Architekturproblemen wie Skalierung, weil sich die korrekte Implementierung von Caching oder Sharding nur schwer mit Interfaces durchsetzen lässt, wenn Art und Struktur der Daten im Voraus nicht bekannt sind.

Die einfachste Lösung für eine Konnektor-Datenbank ist, dass die Konnektoren diese selber mitbringen. Dies kann in Form einer in den Code eingebetteten Collection oder einer eingebetteten Datenbank wie HSQLDB sein. Die Collection ist für einfache Fälle denkbar, wenn eine Reihe von Konstanten hinterlegt werden müssen, die sich nicht von Installation zu Installation des Konnektors unterscheiden. Für alle andere Anwendungsfälle verbietet sich die Collection wie die eingebettete Datenbank von selbst. Gründe sind unter anderem Speicherverbrauch, mögliche Konflikte mit anderen Konnektoren oder das Überraschungspotential für Administratoren, die nicht wissen, dass ein Konnektor irgendwo Daten speichert und diese gespeicherten Daten daher nicht angemessen in die Datensicherung integrieren können. Eine «Bring-your-own»-Lösung scheidet damit aus.

Will man eine zentrale Lösung zur Verfügung stellen, gibt es einige Möglichkeiten: Ein RDBMS oder eine der nichtrelationalen Datenbanken, die im Abschnitt 7.3 (Seite 23) erwähnt werden.

Nichtrelationale Datenbanken haben allgemein den Vorteil, dass sie einigermassen schnell und einfach zu bedienen sind, weil kein Schema vorgegeben wird. Ausserdem kümmern sie sich meist selber um die Skalierung über mehrere Nodes hinweg. Der Nachteil ist, dass sie in der Regel keine Benutzerverwaltung und Privilegien-Beschränkung kennen und sich Daten nicht ohne weiteres logisch separieren lassen, beispielsweise in Form mehrerer Datenbanken. Dies bedeutet, dass man den Konnektoren nicht direkten Zugriff auf die Datenbank geben kann und statt dessen den Zugriff kapseln müsste, um zu verhindern, dass Konnektoren unkontrolliert auf den Datenspeicher zugreifen können. Dies liesse sich beispielsweise bei einem Key Value Store mit einem Prefixing der Keys durchsetzen. Eine Alternative wäre die Nutzung von MongoDB, das über das Konzept von Datenbanken und eine einfache Rechtevergabe[1] verfügt (Benutzer plus Unterscheidung Lese- und Schreibzugriff oder nur Lesezugriff). Indem pro Konnektor eine separate Datenbank verwendet wird, liesse sich eine ausreichende Isolation erreichen.

Wenn es um Benutzerverwaltung und Privilegien-Beschränkung geht, bieten RDBMS wie das ohnehin bereits verwendete PostgreSQL aufgrund der feingranularen Rechtevergabe klare Vorteile. Hinsichtlich Skalierung müsste man darauf achten, dass nicht geschrieben wird, sodass

[1] `http://www.mongodb.org/display/DOCS/Security+and+Authentication` (abgerufen: 17. März 2011)

man mehrere Datenbank-Server ohne Wissen des Clients hinter einem Proxy verstecken kann, der als Load Balancer agiert. Oder man bietet einen Caching-Service an, der aber hinsichtlich Isolation der Konnektoren ähnliche Probleme bringt wie die Verwendung einer nichtrelationalen Datenbank, die über keine Rechtevergabe verfügt. In dem Moment, in dem man Schreiboperationen erlaubt, wird die Architektur ungleich schwieriger, weil unter Umständen Sharding im Konnektor implementiert werden muss.

7.2.4 Umsetzung

Konnektor-Gestaltung

Hinsichtlich Konnektor-Gestaltung werden Push- und Pull-Konnektoren realisiert, da diese den kleinsten gemeinsamen Nenner bezüglich Unterstützung der Lokalisierungswerkzeuge darstellen und die benötigten Kommunikationsmuster unterstützen, um Positionen zu ermitteln und Ereignisse verarbeiten zu können. Spezielle Kommunikationsmuster wie Server-Push werden nicht realisiert, da sie nicht in die Architektur passen und auch bezüglich Zuverlässigkeit unzureichend sind.

Bezugssystem

Die Konnektoren liefern die Positionen direkt im jeweiligen Bezugssystem, das für die Landesvermessung verwendet wird. Im Falle der Schweiz wären dies die Schweizer Landeskoordinaten (CH1903). So wird keine zusätzliche Ungenauigkeit bei der Positionsberechnung eingeführt. Soll der Lokalisierungsservice in einem anderen Land eingesetzt werden, müssen die Umrechnungsroutinen der Konnektoren entsprechend angepasst werden, was je nach Datenquelle ohnehin nötig wäre.

Konnektor-Datenbank

Aufgrund der Erkenntnisse aus Abschnitt 7.3 (Seite 23), um sich nicht um Leseskalierung kümmern zu müssen und Schreiboperationen erlauben zu können, wird den Konnektoren ein Gedächtnis in Form einer MongoDB-Datenbank «hineingereicht».

7.3 Datenspeicherung

7.3.1 Problemstellung

Etliche der noch zu implementierenden Use Cases (siehe Abschnitt 17, Seite 105) erfordern eine volle Historie über kontinuierlich, in festen Intervallen ermittelte Positionsdaten, um sie bei Bedarf analysieren zu können, was täglich (beispielsweise zur Feststellung der Geräte, die sich seit dem Vortag bewegt haben) bis monatlich geschehen kann (zur Ermittlung der Nutzung). Typischerweise wird dabei der gesamte respektive nahezu gesamte Datenbestand herangezogen. Die Reaktionszeit steht dabei nicht im Vordergrund.

Die restlichen Use Cases hängen dagegen von den aktuellen Positionen, allenfalls den letzten n Positionsbestimmungen ab, wobei typischerweise $n << 10$. Dafür steht die Reaktionszeit im Vordergrund. Eine Antwort sollte in unter 1 Sekunde gefunden werden, was beispielsweise dann

eine Herausforderung darstellt, wenn das nächste Gerät eines Typs gefunden werden soll, von dem 300 Stück existieren.

7.3.2 Anforderungen

Zur Bewältigung der Problemstellung sollte das System zur Datenspeicherung folgende Kriterien erfüllen:

- Eignung für Speicherung vollständiger Historie von Positionsmeldungen
- Unterstützung möglichst vieler Plattformen für Abfrage
- Replikation für bessere Verfügbarkeit
- Unterstützung von Map Reduce für effiziente Analyse des gesamten Datenbestands (siehe Abschnitt 7.7, Seite 37)
- Einfache und schnelle Abfragen von einzelnen Datensätzen (mit bestimmten Schlüssel oder Kriterien) sowie Bereichen (z.B. letzte 5 Datensätze, die Kriterium `X` erfüllen)
- Konsistenzmodell BASE oder besser, allerdings mit möglichst kleinem Risiko für den Verlust einzelner Datensätze
- Einfache Inbetriebnahme und Administrierbarkeit

Dabei soll von folgenden Datenmengen ausgegangen werden:

- 50'000 zu erfassende Sensoren, die je eine Positionsmeldung liefern (ein Gerät respektive eine Person kann über mehrere Sensoren verfügen)
- Positionserfassung alle 5 Minuten rund um die Uhr
- Datensatz-Grösse von 1 kB inklusive Verwaltungs-Overhead

Dies ergibt bei 8640 Messpunkten ($12\ \mathrm{h}^{-1} \cdot 24\ \mathrm{h/d} \cdot 30\ \mathrm{d} = 8640$) pro Monat 432 GB ($50000 \cdot 10^{-6}\ \mathrm{GB} \cdot 8640 = 432\ \mathrm{GB}$) an Positionsdaten. Es soll aber möglich sein, die Lösung auf bis zu 100'000 Sensoren zu erweitern und die Abfragefrequenz auf durchschnittlich 150 Sekunden zu senken, was eine Datenmenge von gegen 2 TB pro Monat bedeuten würde. In Schreiboperationen übersetzt heisst dies, dass zwischen 168 ($\frac{50000}{5 \cdot 60\ \mathrm{s}} = 166.6$) und 667 ($\frac{100000}{150\ \mathrm{s}} = 666.6$) Datensätze pro Sekunde geschrieben werden können müssen.

Ausserdem werden folgende Leistungsdaten erwartet:

- Abruf 500 beliebiger Datensätze in weniger als 1 Sekunde
- Abruf 500 Datensätze mit Kriterium X in weniger als 1 Sekunde

7.3.3 Lösungsansätze

Relationale Datenbank

Da bereits ein RDBMS in Form von PostgreSQL als GIS verwendet wird, wäre es naheliegend, ein RDBMS zur Speicherung der Positionshistorie zu verwenden. Grundsätzlich ist es machbar, die geforderte Schreib- und Leseleistung zu erreichen, wie ein Benchmark zeigt. Da die geforderte Schreibleistung deutlich unter der möglichen Gesamtleistung liegt, liesse sich mit Master-Slave-Replikation nicht nur die Verfügbarkeit verbessern, sondern auch die Lese-Leistung skalieren.

Angesichts der Speicherkapazität aktueller Festplatten wäre es auch problemlos möglich, den gesamten Datenbestand mehrerer Monate auf einem einzelnen Server zu lagern.

Die effiziente Nutzung eines RDBMS würde allerdings ein paar Tricks erfordern: Die Daten müssten auf mehrere Tabellen verteilt werden (z.B. Rotation), damit die Indices nicht zu gross werden und die benötigte Zeit für einen Index-Rebuild nicht stetig ansteigt. Um von Master-Slave-Replikation profitieren zu könnten, müssten Schreib- und Leseoperationen applikationsseitig getrennt und Load Balancing eingesetzt werden. Für PostgreSQL stehen eine Reihe von Optionen[2] zur Verfügung, die das vereinfachen und wie beispielsweise pgpool[3] im beschränkten Mass eine parallele Ausführung von Anfragen ermöglichen. Zusätzlich würde sich die Verwendung eines Cache für Leseanfragen anbieten (z.B. Redis, memcached oder Ehcache). Möchte man Map Reduce verwenden, ist man auf ein separates Framework wie Hadoop angewiesen.

Nichtrelationale Datenbanken

Eine Alternative zu einem RBDMS sind in diesem Fall nichtrelationale Datenbanken, da nur ein kontinuierlicher Strom von Positionsmeldungen gespeichert werden muss, also schwach strukturierte Daten. Da das Feld der nichtrelationalen Datenbanken sehr weit ist (nosql-database.com listet alleine deren 122[4]) und die Analyse einer einzelnen Datenbank abhängig von der Komplexität mehrere Tage oder gar Wochen dauern kann, wurde [8] als Leitfaden zur Vor-Filterung möglicher Optionen und Kandidaten verwendet. Aufgrund der beschriebenen Anforderungen kommen grundsätzlich Cassandra (Wide Column Store), HBase (Wide Column Store), MongoDB (Document Store) und Riak (Mischung aus Key Value Store und Document Store) in Frage.

HBase und Cassandra gehören zu den leistungsfähigsten nichtrelationalen Datenbanken, die problemlos in den Tera- und Petabyte-Bereich skalieren [6]. Diese Skalierbarkeit erkauft man sich mit Komplexität, so benötigen sowohl HBase (das darauf basiert) und Cassandra (für Map Reduce Jobs) Hadoop, das sich aus einem ganzen Zoo von Komponenten zusammensetzt, unter anderem HDFS als verteiltes Dateisystem, ZooKeeper für Koordination des Clusters und Pig zur effektiveren Nutzung von Map Reduce. Dazu kommt, dass beide Systeme erst eine einigermassen vernünftige Leistung erzielen, wenn eine Handvoll Nodes zur Verfügung steht [6]. Da der Lokalisierungsservice kaum eine entsprechende Leistung braucht, würde dies nur unnötig Kosten hinsichtlich Hardware und Personalaufwand verursachen. Entsprechend ist eine «kleinere» Lösung wünschenswert, die auch mit 1 bis 3 Nodes eine ansprechende Leistung bietet und nicht so aufwendig in der Inbetriebnahme und der Administration ist. Diese Anforderungen erfüllen grundsätzlich MongoDB und Riak, weshalb sie genauer unter die Lupe genommen wurden.

MongoDB Interessant an MongoDB ist, dass es eine einfache Abfragesprache mitbringt, die die Filterung der Resultate ohne Map Reduce ermöglicht. Dabei stehen viele aus SQL bekannte Operationen bereit[5] wie Vergleiche (gleich, grösser, kleiner usw.), Cursor-Methoden (`count()`, `limit()` usw.) und Gruppierung. Reichen diese Möglichkeiten nicht aus, kann auf Map Reduce-Jobs zurückgegriffen werden, die in JavaScript formuliert und auf den MongoDB-Nodes ausge-

[2] `http://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling` (abgerufen: 17. März 2011)

[3] `http://pgpool.projects.postgresql.org/` (abgerufen: 17. März 2011)

[4] `http://nosql-database.com/` (abgerufen: 17. März 2011)

[5] `http://www.mongodb.org/display/DOCS/Advanced+Queries`, (abgerufen: 17. März 2011)

führt werden. Problematisch scheint im Moment aber noch die Map-Reduce-Leistung zu sein[6], wobei Abhilfe mit Version 2.0 von MongoDB versprochen wird und im Fall der Fälle noch immer auf Hadoop zurückgegriffen werden kann[7]. Ebenfalls wie bei RDBMS können Indices gesetzt[8] werden, um Abfragen zu beschleunigen. Die Anbindung an verschiedene Programmiersprachen (u.a. Java) erfolgt über spezielle Treiber.

MongoDB ist nach dem Master-N-Slaves-Prinzip konzipiert. Horizontale Skalierung wird wie bei RDBMS über Sharding[9] und Replikation[10] (mit automatischem Failover und Recovery) erreicht – mit dem Unterschied, dass MongoDB von Anfang an darauf ausgelegt ist und die nötigen Werkzeuge mitbringt, um keinen Single Point of Failure im Cluster zu haben. Replikation ist ein Muss, um Datenverlust vorzubeugen, auch wenn MongoDB über ein Journal[11] verfügt.

MongoDB unterstützt atomare Operationen[12] auf einzelnen Dokumenten. Optimistic Concurrency wird angeboten, ebenso die Möglichkeit, Datensätze einzufügen, sofern sie nicht vorhanden sind (benötigt Unique Index). Dies sollte es möglich machen, MongoDB als Job Store für Quartz zu verwenden.

Riak Riak zeichnet sich dadurch aus, dass jeder Knoten im Cluster gleichberechtigt ist. Das macht Betrieb und Skalierung einfacher. Verschiedene Parameter, die beispielsweise regeln, wie viele Nodes eine Schreiboperation bestätigen müssen, erlauben eine Feinabstimmung zwischen Konsistenz, Geschwindigkeit und Datensicherheit[13] (CAP-Parameter [10]). Zusätzlich können wie bei MySQL verschiedene Storage Engines gewählt werden: Bitcask[14] und Innostore[15] (Wrapper um Embedded-Variante von MySQLs InnoDB).

Abfragen jenseits der Suche nach einem Schlüssel werden mit Map Reduce formuliert, wobei die Routinen entweder in JavaScript oder Erlang verfasst[16] werden können. Zusätzlich steht mit Riak Search[17] eine Volltext-Suche zur Verfügung. Eine Spezialität von Riak sind Links[18]. Links ermöglichen, vergleichbar zu einem RDBMS Relationen zu modellieren und denen bei der Abfrage zu folgen (Link Walking[19]).

Die Interaktion mit Riak erfolgt grundsätzlich über HTTP. Spezielle Client-Bibliotheken (u.a. für Java) kapseln den HTTP-Zugriff.

Hinsichtlich Concurrency verwendet Riak Vektoruhren[20]. Dies bedeutet, dass der Client letztlich selber entscheiden muss, welche Version er verwendet.

[6] `https://jira.mongodb.org/browse/SERVER-3055` (abgerufen: 4. Juni 2011)
[7] `https://github.com/mongodb/mongo-hadoop` (abgerufen: 4. Juni 2011)
[8] `http://www.mongodb.org/display/DOCS/Indexes` (abgerufen: 17. März 2011)
[9] `http://www.mongodb.org/display/DOCS/Sharding+Introduction` (abgerufen: 17. März 2011)
[10] `http://www.mongodb.org/display/DOCS/Replica+Sets` (abgerufen: 17. März 2011)
[11] `http://www.mongodb.org/display/DOCS/Durability+and+Repair` (abgerufen: 17. März 2011)
[12] `http://www.mongodb.org/display/DOCS/Atomic+Operations` (abgerufen: 17. März 2011)
[13] `http://wiki.basho.com/Tunable-CAP-Controls-in-Riak.html` (abgerufen: 17. März 2011)
[14] `http://blog.basho.com/2010/04/27/hello-bitcask/` (abgerufen: 17. März 2011)
[15] `http://blog.basho.com/2010/02/22/using-innostore-with-riak/` (abgerufen: 17. März 2011)
[16] `http://wiki.basho.com/MapReduce.html` (abgerufen: 17. März 2011)
[17] `http://wiki.basho.com/Riak-Search.html` (abgerufen: 17. März 2011)
[18] `http://wiki.basho.com/Links.html` (abgerufen: 17. März 2011)
[19] `http://wiki.basho.com/Links-and-Link-Walking.html` (abgerufen: 17. März 2011)
[20] `http://wiki.basho.com/Riak-Glossary.html#Vector-Clock` (abgerufen: 17. März 2011)

Logging-Systeme

Eine weitere Option sind grundsätzlich auch Logging-Systeme, beispielsweise syslog-ng[21], Scribe[22] oder Flume[23]. Bei allen besteht das Problem, dass sie sich nur um Speicherung der Logs kümmern, aber keine effiziente Abfragemöglichkeit bereitstellen. Diese müsste entweder selber entwickelt oder zum Beispiel über eine separate Komponente wie Hadoop realisiert werden.

7.3.4 Umsetzung

Ein RDBMS zu verwenden, würde aufgrund der Struktur der Daten, ihrer Menge und der vorgesehenen Nutzung mehr Aufwand bedeuten als die Verwendung einer nichtrelationalen Datenbank, weshalb eine solche vorgezogen wird. Von den beiden Kandidaten MongoDB und Riak weist Riak zwar die robustere Architektur auf, fällt bezüglich Geschwindigkeit und Funktionalität hinter MongoDB zurück, das quasi die besten Eigenschaften von RDBMS und nichtrelationalen Datenbanken vereint. Deshalb wird vorerst MongoDB verwendet. Sollte sich aufgrund der Menge der Daten und Anfragen zeigen, dass ohnehin grössere Cluster (mehr als 3 bis 5 Nodes) benötigt werden, sollte man zu einem späteren Zeitpunkt immer noch auf HBase und Hadoop wechseln können, ohne allzu viele Teile des Lokalisierungsservice anpassen zu müssen.

7.4 Interpretation Positionsmeldungen

7.4.1 Problemstellung

Jedes zu lokalisierende Gerät verfügt über einen oder mehrere «Positionssensoren». Dies können beispielsweise eine RFID-Etikette, ein Ekahau-Tag (WLAN-Lokalisierung) und eine Verbindung mit einem Ethernet-Switch-Port sein, von dem man die Position kennt. Diese Positionssensoren können gleiche, aber auch unterschiedliche oder zum Teil gar keine Positionen melden. Beispiele:

1. RFID-Etikette löst Meldung bei Betreten von Bereich aus, Ekahau-Tag meldet eine Position in einem anderen Bereich, LAN-Lokalisierung meldet gar kein Signal, weil das Gerät nicht mit dem Local Area Network (LAN) verbunden ist. Statt dessen ist eine Position von vor 5 Minuten aus demselben Bereich vorhanden, in dem das Ekahau-Tag seine Position vermutet.
2. RFID-Etikette löst Meldung bei Betreten von Bereich aus, Ekahau-Tag meldet eine Position in demselben Bereich, aber in einem ganz anderen Sektor. LAN-Lokalisierung meldet Position in demselben Bereich, aber in einem dritten Sektor (separates Zimmer).

Wer hat jeweils Recht? Bei Fall 1 dürfte es die RFID-Etikette sein, weil sich die RFID-Etikette sicher in einen anderen Bereich bewegt hat. Dem Ekahau-Tag ist deshalb nicht zu trauen und es dürfte sich um eine Messungenauigkeit handeln. Das alte Resultat der LAN-Lokalisierung kann nicht herangezogen werden, weil das Gerät ausgesteckt ist und sich wohl bewegt hat, sonst hätte die RFID-Etikette nicht ausgelöst. Bei Fall 2 ist die Bereichsmeldung sicher (alle 3 Sensoren

[21] `http://www.balabit.com/network-security/syslog-ng/opensource-logging-system` (abgerufen: 17. März 2011)
[22] `https://github.com/facebook/scribe` (abgerufen: 17. März 2011)
[23] `https://github.com/cloudera/flume` (abgerufen: 17. März 2011)

stimmen überein), nicht aber die Position. Wenn man davon ausgeht, dass man das LAN-Kabel des Geräts nicht im Nebenzimmer einsteckt (sofern dessen Länge dazu überhaupt ausreicht), dürfte die LAN-Lokalisierung die richtige Position melden.

Komplexer wird das Problem, wenn Personen lokalisiert werden müssen. Dies funktioniert nicht direkt (z.B. Chip unter der Haut), sondern indirekt über Geräte, die die Personen auf sich tragen. Es stellt sich also nicht nur das oben erwähnte Problem, widersprüchliche Positionsmeldungen der einzelnen Sensoren auflösen zu müssen. Sondern es müssen auch Widersprüche zwischen den ermittelten Positionen der einzelnen Geräte aufgelöst werden. Beispiele:

1. Smartphone meldet seit einer Woche, es sei nicht auf dem Campus. WLAN-Handset meldet seit einer Woche, es ist auf dem Campus, immer an der gleichen Stelle.
2. Smartphone sendet kein Signal, WLAN-Handset meldet die letzten 30 Minuten immer wieder eine andere Position.

Wer hat Recht? In Fall 1 wohl das Smartphone. Die Person, der Smartphone und WLAN-Handset zugeordnet sind, dürfte nicht auf dem Campus sein, weil das WLAN-Handset seit einer Woche in der Ladestation steht. Bei Fall 2 dürfte wohl das WLAN-Handset recht haben, weil es herumgetragen wird. Das Smartphone sendet kein Signal, weil es vielleicht auf dem Weg zur Arbeit von der Strassenbahn überrollt wurde.

7.4.2 Anforderungen

Es muss ein flexibler und effizienter Weg gefunden werden, um Regeln zur Interpretation der Positionsmeldungen und Auflösung von Widersprüchen in den Lokalisierungsservice einbinden zu können. Da die Regeln je nach Umgebung und über die Zeit hinweg stark ändern können (z.B. wenn ein neuer Konnektor hinzukommt), sollten sie einfach austauschbar sein und möglichst ohne Programmierkenntnisse verfasst werden können. Wichtig ist, dass die Regeln in hoher Geschwindigkeit abgearbeitet werden.

Ebenfalls muss festgelegt werden, welche Daten gebraucht werden, um entscheiden zu können, welchen Sensoren in welchem Fall vertraut wird.

Eine Anforderung aus dem Bereich Persönlichkeitsschutz ist dagegen, bestimmte Positionsmeldungen unterdrücken respektive künstlich unscharf machen zu können, damit sich beispielsweise bei einer Analyse nicht feststellen lässt, wie viel Arbeitszeit eine Person in einem Montag auf der Toilette oder beim Rauchen verbracht hat.

7.4.3 Lösungsansätze

Regelabbildung

Die Regeln könnten in Java ausprogrammiert (`if-else`, `switch`) werden, was aufgrund der Anforderungen wie einfache Änderbarkeit ohne Programmierkenntnisse keine Option sein dürfte. Gleiches gilt für die Verwendung einer logischen Programmiersprache wie Prolog, die sich ausserdem nicht ohne Weiteres innerhalb der Java-Plattform verwenden lässt. Ebenfalls gegen die Java-Option spricht, dass die Ausführung von Regeln, die als normale Bedingungen formuliert sind, schnell ineffizient wird, weil jede Regel einzeln überprüft wird. Effizienter ist die Verwendung des Pattern-Matching-Algorithmus Rete [9], da er nur einen Teil der Regeln evaluiert. Rete wird von vielen Rule Engines implementiert und dient auch in Prolog der Regelauswertung.

Diese Rule Engines dürften auch in anderen Aspekten [16] eine bessere Wahl sein. Sie ermöglichen nämlich, Regelsets während des Betriebs auszutauschen, wobei die Regelsets separat vom Programmcode definiert werden können. Dies erfolgt oftmals mit Hilfe einer DSL. Regeln können auch mit speziellen Werkzeugen verfasst werden, beispielsweise Eclipse-Plug-ins, wie sie für Jess[24] oder Drools Expert[25] angeboten werden. Eine andere Möglichkeit besteht darin, Entscheidungstabellen aus Spreadsheets zu importieren, wie es von Drools Expert ermöglicht[26] wird.

Benötigte Daten

Gerätelokalisierung

- Position des Sensors (sektorgenau, inklusive Gebäude, Stockwerk, Zone)
- Zeitpunkt der Positionsmeldung
- Genauigkeit der Positionsmeldung (sofern vom Lokalisierungswerkzeug unterstützt), vorzugsweise als Aussage der Form `GUT` ($\pm < 1.0$ m), `MITTEL` ($\pm < 2.5$ m), `SCHLECHT` ($\pm \geq 2.5$ m), `UNBEKANNT`

Benötigt wird jeweils die letzte Positionsmeldung jedes Sensors, egal, ob sie gerade eingetroffen ist oder mehrere Tage alt ist. Denn abhängig vom Lokalisierungswerkzeug und seiner Betriebsart kann auch eine mehrere Tage alte Positionsmeldung aktuell sein (wie bei RFID, das nur auslöst, wenn ein Tag an einem Lesegerät vorbeikommt). Wie diese Meldungen behandelt werden, ist Aufgabe der Positionsinterpretation. Damit ist aber nur eine Interpretation aufgrund der aktuellen Meldung möglich. Es lässt sich damit beispielsweise nicht feststellen, ob sich ein Sensor bewegt hat seit der letzten Positionsmeldung. Diese Information ist aber wichtig, um das oben erwähnte Szenario mit dem Smartphone und dem sich bewegenden WLAN-Handset abbilden zu können. Dazu muss mindestens auch die vorletzte Positionsmeldung herangezogen werden. Unterscheidet sich diese von der aktuellen Meldung, hat sich der Sensor bewegt.

Nun kann dieses Vorgehen zu empfindlich sein: Wird beispielsweise der Kleiderständer mit dem Arztkittel, bei dem das WLAN-Handset in einer Tasche steckt, durch eine Reinigungskraft bewegt, kann dies bereits dazu führen, dass der Sensor eine andere Position meldet. Unter Umständen ist dies bei Verfahren wie der WLAN-Lokalisierung sogar ohne reale Positionsänderung möglich, wenn sich der Sensor in einem Grenzbereich befindet und bereits kleine Änderungen bei der Signalstärke dazu führen können, dass bei der Positionsbestimmung durch den Sensor ein anderes Resultat herauskommt. Um diese Empfindlichkeit zu senken, sollten auch ältere Meldungen herangezogen werden können.

Personenlokalisierung Die Personenlokalisierung verläuft indirekt, also über die Geräte, die eine Person auf sich trägt. Nun wäre es möglich, jeden Sensor nochmals einzeln auszuwerten. Dies hätte den Vorteil, geräteübergreifend Aussagen treffen zu können. Der Nachteil wäre aber, dass Geräteregeln grossmehrheitlich dupliziert würden (wenn sie z.B. nur einen Sensor besitzen) und die Komplexität der Regeln stark ansteigt.

[24] `http://www.jessrules.com/jess/docs/71/eclipse.html` (abgerufen: 17. März 2011)

[25] `http://downloads.jboss.com/drools/docs/5.1.1.34858.FINAL/drools-expert/html/ch07.html` (abgerufen: 17. März 2011)

[26] `http://downloads.jboss.com/drools/docs/5.1.1.34858.FINAL/drools-expert/html/ch05.html` (abgerufen: 17. März 2011)

Die andere Möglichkeit wäre, sich auf die Interpretationen der Gerätelokalisierung zu verlassen nach dem Motto, dass die Gerätelokalisierung wohl wissen wird, wo «ihre» Geräte sind. Für die Personenlokalisierung müssten damit nur noch Regeln der Form «Wenn WLAN-Handset auf dem Campus und Smartphone nicht auf dem Campus, dann Person nicht auf dem Campus» formuliert werden, was deren Erstellung stark vereinfachen würde. Benötigt werden dafür folgende Daten:

- Position des Geräts (sektorgenau, inklusive Gebäude, Stockwerk, Zone)
- Zeitpunkt der Positionsmeldung
- Genauigkeit der Positionsmeldung (`GUT`, `MITTEL` etc.)

Da die Position des Geräts unter Umständen aus mehreren Positionsmeldungen einzelner Sensoren abgeleitet wurde, deren Nachrichten unterschiedlich alt sind, stellt sich die Frage, der Zeitpunkt welcher Positionsmeldung als «Zeitpunkt der Positionsmeldung» herangezogen werden soll. Wie bereits bei der Gerätelokalisierung erläutert, ist eine alte RFID-Meldung quasi von «jetzt», während eine zwei Wochen alte Nachricht eines Ekahau-Tags wertlos ist. Es bestehen also zwei Varianten: Entweder man reicht die Zeit der Originalmeldung durch, was bedeutet, dass die Personenlokalisierung wissen muss, dass eine zwei Wochen alte RFID-Meldung aktuell sein kann. Oder man «übersetzt» bereits bei der Gerätelokalisierung den Zeitstempel der Nachricht. So wäre eine zwei Wochen alte RFID-Meldung von «jetzt» und eine 3 Stunden alte Nachricht eines Ekahau-Tags 3 Stunden alt. Dann könnte die Personenlokalisierung direkt mit diesen Informationen weiterarbeiten.

Die Genauigkeit der Positionsmeldung wird benötigt, um einfache Widersprüche aufzulösen: Sind das Smartphone und das WLAN-Handset in zwei nebeneinanderliegenden Räumen, die Genauigkeit der Meldung des Smartphones `SCHLECHT` und diejenige des WLAN-Handsets `GUT`, kann die Personenlokalisierung das WLAN-Handset stärker gewichten. Diese Aufgabe kann nicht an die Gerätelokalisierung delegiert werden, weil diese nicht den Überblick über mehrere Geräte hat. Damit erhält man auch einen Kompromiss zwischen der nochmaligen Auswertung aller Sensoren und der Weiterverwendung der Resultate der Gerätelokalisierung. Als Genauigkeit der Positionsmeldung wird jeweils die Genauigkeit der Meldung verwendet, die die Grundlage der Entscheidung der Gerätelokalisierung gebildet hat.

Wie bei der Gerätelokalisierung wird die jeweils aktuelle und mindestens die vorletzte Position benötigt, um Aussagen über Bewegung treffen zu können.

Unterdrückung von Positionen

Positionsmeldungen, deren Koordinaten nicht innerhalb der erfassten Campus-Geometrie liegen, können konstruktionsbedingt nicht aufgelöst und in eine Positionsangabe nach der Form Gebäude, Stockwerk, Raum übersetzt werden und werden deshalb automatisch unterdrückt respektive in eine generische Meldung wie «Nicht auf dem Campus» umgewandelt. Bei Positionen, die sich eigentlich auf dem Campus befinden, aber global unterdrückt werden sollen, besteht die Möglichkeit, diese bei der Digitalisierung der Grundrisse wegzulassen oder – flexibler – die Positionen bei der Abarbeitung der Regeln selektiv zu unterdrücken und beispielsweise in eine generische Meldung wie «Auf dem Campus» umzuwandeln.

7.4.4 Umsetzung

Regelabbildung

Da die Regeln, die innerhalb des Lokalisierungsservice benötigt werden, nicht trivial sind und eine grosse Flexibilität gefordert ist, ist die Verwendung einer Rule Engine die sinnvollste Variante. Aufgrund der vielfältigen Funktionen und Optionen zur Regelerfassung wird Drools Expert verwendet.

Benötigte Daten

Geräte- und Personenlokalisierung werden in zwei Schritten ausgeführt. Zuerst die Gerätelokalisierung und dann die Personenlokalisierung, welche auf ersterer basiert. Für die Gerätelokalisierung werden folgende Daten zur Verfügung gestellt:

- Position des Sensors (Gebäude, Stockwerk, Sektor, Zone)
- Zeitpunkt der Positionsmeldung (Empfangszeitpunkt der Nachricht vom Sensor)
- Genauigkeit der Positionsmeldung: `GUT` ($\pm < 1.0$ m), `MITTEL` ($\pm < 2.5$ m), `SCHLECHT` ($\pm \geq 2.5$ m) oder `UNBEKANNT`

Für die Personenlokalisierung stehen dann folgende Daten zur Verfügung

- Position des Geräts (Gebäude, Stockwerk, Sektor, Zone)
- Zeitpunkt der Positionsmeldung (Zeit, zu der die Meldung aktuell war)
- Genauigkeit der Positionsmeldung (`GUT`, `MITTEL` etc.)

7.5 Ereignis-Erzeugung

7.5.1 Problemstellung

Verschiedene Use Cases und der Presence Manager, ein Teil des Gesamtsystems, das zur Ermittlung der Präsenzinformation von Personal dient, sind darauf angewiesen, über das Eintreten verschiedener Ereignisse informiert zu werden. Dies kann das Auslösen von Alarmen sein oder die Benachrichtigung über einen Statuswechsel. Da der Lokalisierungsservice nur die Position von Geräten und damit indirekt Personen kennt, kann er nebst der Position nur drei Aussagen treffen: «Gerät hat sich bewegt», «Gerät hat sich nicht bewegt» und «Gerät kann nicht gefunden werden». Es soll deshalb überlegt werden, wie auf Basis der Position und der drei Aussagen weiterführende Statusableitungen wie *frei*, *besetzt* oder *Person operiert* durchgeführt werden können.

Während bei den oben skizzierten Anwendungen eine gewisse Latenz zwischen Eintreten des Ereignisses (z.B. Arzt betritt Operationssaal) und Auslösung im Lokalisierungsservice (Präsenzinformation wird aktualisiert) problemlos oder sogar wünschenswert sein kann (wenn Person z.B. Raum kurz betritt und gleich wieder verlässt), müssen auch Anwendungsszenarien bedacht werden, bei denen die Latenz möglichst klein sein sollte. Ein Beispiel wäre Patientenüberwachung.

Beachtet werden muss, dass Lokalisierungswerkzeuge irreführende Meldung schicken können. Ein Beispiel illustrieren die Abbildungen 7.1 (Seite 33) und 7.2 (Seite 33). Erstere zeigt die vom Lokalisierungswerkzeug Ekahau gemeldeten Positionen. Auf der zweiten Abbildung ist

eingezeichnet, wo sich die Geräte befinden würden, wenn die Positionsmeldung korrekt wäre: draussen, obwohl sie sich im Gebäude befinden. Auf so eine Positionsmeldung zu reagieren, würde zu fehlerhaften Ereignissen führen.

Nebst positionsbezogenen Ereignissen sollen auch Alarme, die nicht positionsbezogen sind, verarbeitet werden können. So ein Alarm kann beispielsweise durch das Drücken eines Kopfes an einem Ekahau-Tag ausgelöst werden. Bei so einer Alarmmeldung ist nicht die Position von primärem Interesse, sondern dass dieser Alarm ausgelöst wurde.

Hinsichtlich Kommunikation mit anderen Systemen muss überlegt werden, wie Ereignisse publiziert werden und welche Möglichkeiten hinsichtlich «Interessebekundung» und Filterung an und von Ereignissen geboten werden.

7.5.2 Anforderungen

Es muss ein flexibler und effizienter Weg gefunden werden, um Regeln zur Statusableitung, zur Interpretation der Positionsmeldungen und zur Auflösung von Widersprüchen in den Lokalisierungsservice einbinden zu können. Da die Regeln je nach Umgebung und über die Zeit hinweg stark ändern können (z.B. wenn ein neuer Konnektor hinzukommt), sollten sie einfach austauschbar sein und möglichst ohne Programmierkenntnisse verfasst werden können. Wichtig ist ebenfalls, dass die Regeln in hoher Geschwindigkeit abgearbeitet werden.

Bei der Analyse der Anbindung der Konnektoren in Abschnitt 7.2 (Seite 20) hat sich gezeigt, dass verschiedene Konnektor-Arten benötigt werden. Deren Eigenheiten müssen bei der Ereignis-Erzeugung berücksichtigt werden. Konkret bedeutet dies, dass das System sowohl selber nach Ereignissen fragen als auch auf ausbleibende Ereignisse reagieren können muss. Ebenso muss beachtet werden, dass Personen indirekt lokalisiert werden, also über die Geräte, die sie auf sich tragen. Da deren Zusammensetzung über die Zeit hinweg schwanken und sie zum Teil unterschiedliche Ergebnisse liefern können (z.B. aufgrund von Messfehlern), muss die Ereigniserzeugung mit widersprüchlichen Aussagen umgehen können und darf für einen Arzt nicht gleichzeitig melden, dass er im Operationssaal und zu Hause ist.

Ereignisse sollten mit einer Latenz zwischen 60 Sekunden und 10 Minuten ausgelöst werden können, wobei die Latenz je nach Geräteklasse reguliert werden können muss. Die Ereignis-Erzeugung soll sich ausserdem für ganze Geräteklassen deaktivieren lassen. Hinsichtlich Alarmmeldungen muss ein Weg gefunden werden, diese möglichst zuverlässig zu den interessierten Systemen zu transportieren, da sie unter Umständen sogar als Patientennotruf verwendet werden können.

7.5.3 Lösungsansätze

Statusableitung

Wie bereits in der Problemstellung erläutert, kann der Lokalisierungsservice nebst der Position nur die Aussage treffen, ob ein Gerät überhaupt gefunden wurde und ob es sich allenfalls bewegt hat. Insofern besteht die einzige Möglichkeit einer Statusableitung darin, die Position mit ihrer Bedeutung zu kombinieren. Befindet sich eine Person beispielsweise in Sektor 1 und ist der Sektor 1 ein Operationssaal, könnte eine Regel «Ist Person in Sektor 1, setze Status auf operierend» formuliert werden. Dies würde ermöglichen, andere Systeme über das Ereignis *Person operiert* zu informieren. Dieses Verfahren hat allerdings einen gewichtigen Nachteil: Für andere Systeme,

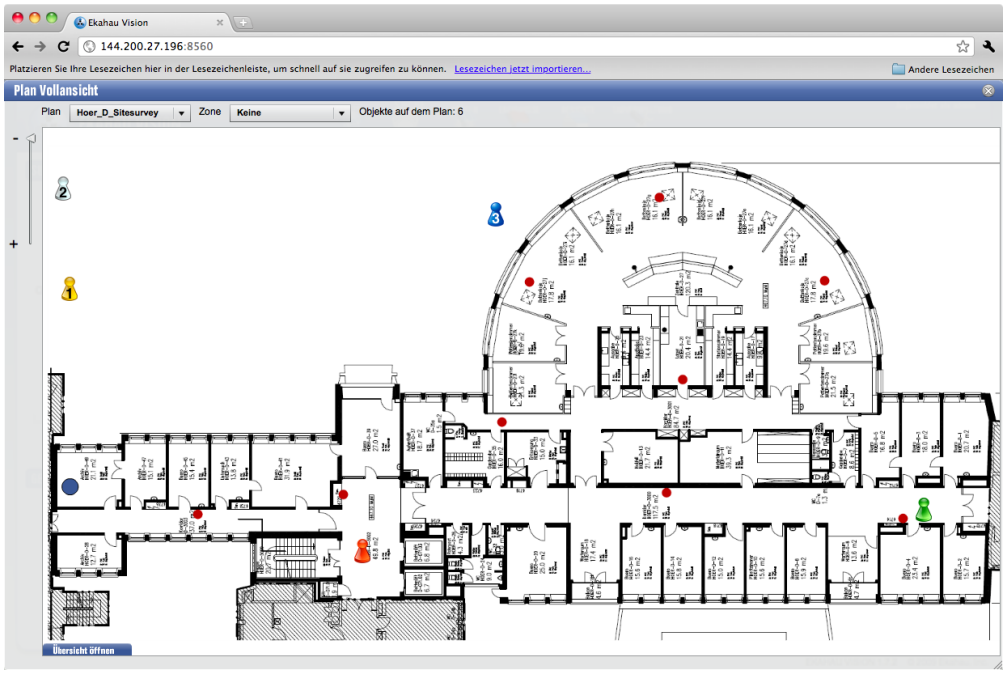

Abbildung 7.1: Der Screenshot aus Ekahau Vision illustriert einen Messfehler, wie er bei der WLAN-Lokalisierung auftreten kann. Die mit 1 bis 3 beschrifteten Tags werden von Ekahau Vision nicht mit ihrer realen Position angezeigt. Abbildung 7.2 auf Seite 33 verdeutlicht den Fehler.

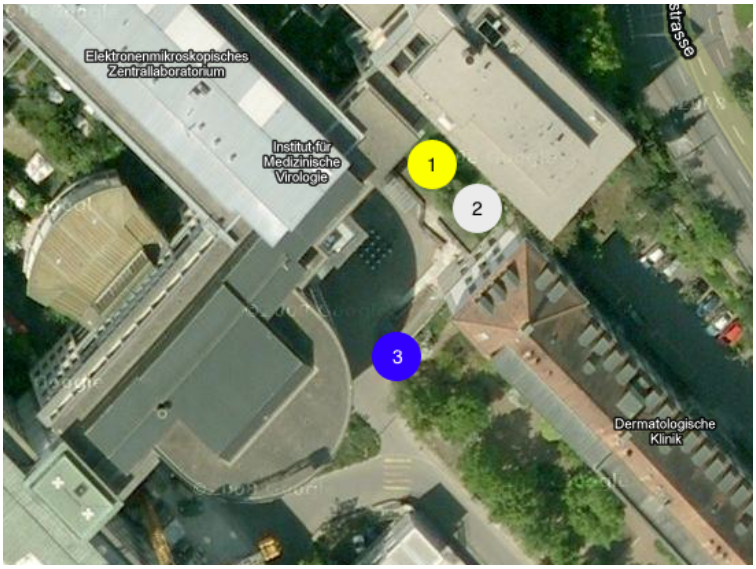

Abbildung 7.2: Das Satellitenfoto aus Google Maps zeigt, wo sich die Tags aus Abbildung 7.1 (Seite 33) befinden würden, wenn die Angabe von Ekahau Vision korrekt wäre. Das Satellitenfoto ist wegen Nord-Süd-Ausrichtung im Vergleich zu Abbildung 7.1 (Seite 33) gedreht.

die das Ereignis konsumieren, wird verwischt, wie dieses zu Stande gekommen ist. Dies kann zu Fehlschlüssen führen, denn nur weil jemand in einem Bereich ist, bedeutet dies nicht immer, dass er auch die Tätigkeit ausführt, die man die meiste Zeit in diesem Bereich tut. Ein Beispiel, das zu so einem Fehlschluss führen könnte, wäre eine Sitzung in der Kantine. Wird statt «Person hat Pause» eine Ortsangabe wie «Person ist im Park» ausgelöst, würde dies nachgelagerten Systemen eher ermöglichen, Fehlschlüsse zu vermeiden, was insbesondere bei Konfliktsituationen wertvoll sein kann. Meldet ein Kalender beispielsweise Sitzung und ist eine Person in der Kantine (Status Pause), kann es sich durchaus um eine Sitzung handeln, vor allem, wenn als Sitzungsort Kantine angegeben wurde. Ist die Person dagegen im Park (Status Pause) und als Sitzungsort ein Sitzungszimmer angegeben, dürfte die Sitzung entweder nicht stattfinden oder bereits vorbei sein. Also dürfte der Status Pause korrekt sein. Dieser Schluss kann aber nicht vollzogen werden, wenn das nachgelagerte System nur den Status Pause sieht. Auch hinsichtlich Alarmen dürfte eine Auslösung im Lokalisierungsservice deshalb keine gute Idee sein. Hinzu kommt, dass für jede temporäre Deaktivierung eines Alarms die Regelsätze angepasst werden müssten, was im Alltagsbetrieb nicht praktikabel ist.

Ereignis-Erzeugung

Der Lokalisierungsservice kann sich aufgrund der verschiedenen Konnektor-Architekturen nicht auf von aussen erzeugte Ereignisse verlassen, um selber Ereignisse zu erzeugen, weil dies die Lokalisierungswerkzeuge nicht her geben. Bei Pull-Konnektoren, die in regelmässigen Abständen nach der Position von Sensoren gefragt werden müssen, ist der Lokalisierungsservice bei der Ereignis-Erzeugung ohnehin vom diesem Lokalisierungsintervall abhängig. Als Beispiel sei die LAN-Lokalisierung auf Basis von Cisco Works erwähnt, die gar keine Events bietet, obwohl es nebst einem regelmässigen Netzwerk-Scan den eigenen Datenbestand mit Hilfe von SNMP-Traps aktuell hält. Bei Push-Konnektoren besteht das Problem, dass diese sich zwar regelmässig melden und damit als Auslöser für Ereignisse dienen können, die Meldungen aber unterbleiben können (z.B. bei Verbindungsproblemen oder Stromausfall) und das ausbleibende Ereignis nicht getriggert wird. Man denke hier an ein Smartphone in einem Funkloch oder an Ekahau, das zwar Events sendet, dies aber über Server Push ohne garantierte Nachrichten-Auslieferung tut.

Was damit bleibt, ist die Zeit in Form eines Scheduler innerhalb des Lokalisierungsservice als Auslöser für Ereignisse zu verwenden und in regelmässigen Intervallen zu überprüfen, ob ein Ereignis stattgefunden hat. Da diese Prüfung direkt von der Frequenz der Positionsbestimmungen abhängig ist, werden beide Aufgaben zusammengelegt und nacheinander ausgeführt: Zuerst die Positionsbestimmung und daraufhin die Auslösung eines Ereignisses, sofern es dafür einen Anlass gibt.

Ereignis-Erkennung

Die Antwort zur Frage, ob ein Ereignis stattgefunden hat, hängt einerseits von der aktuellen Position und andererseits von vorhergehenden Positionen ab. Ist ein Arzt beispielsweise vor 5 Minuten in der Kantine gewesen und jetzt im Operationssaal, muss ein Ereignis ausgelöst werden. War er aber bereits vor 5 Minuten im Operationssaal, ist dies kein Ereignis. Um Fehlmeldungen aufgrund von Messfehlern zu unterdrücken, bleibt einzig die Möglichkeit, sich auf Positionsmeldungen mehrerer Werkzeuge zu verlassen und dann mit geeigneten Regeln – dass

beispielsweise die LAN-Lokalisierung die WLAN-Lokalisierung übersteuert, wenn sie nicht aus demselben Bereich kommt – den Fehlern entgegen zu wirken.

Die Ereignis-Erkennung hängt damit stark mit der Interpretation der Positionsmeldungen (siehe Abschnitt 7.4, Seite 27) zusammen und sollte daher zusammen mit dieser gelöst werden. Gleiches gilt für die Möglichkeit, Ereignisse nach Geräteklassen zu unterdrücken, da diese dort sehr fein reguliert werden kann.

Alarm-Behandlung

Alarme unterscheiden sich im Kontext des Lokalisierungsservice von Ereignissen darin, dass sie von den Lokalisierungswerkzeugen ausgelöst werden und nicht positionsbezogen sind. Ein Beispiel wäre das Drücken eines Knopfs an einem Ekahau-Tag, der als Patientennotruf dient. Da nicht eine akkurate Positionsmessung von grösstem Interesse ist, sondern dass der Alarm ausgelöst wurde, sollte dieser nicht gleich behandelt werden wie ein Ereignis, da sonst beispielsweise durch eine Regel zum Auflösen von Ungenauigkeiten das Risiko besteht, dass er unterdrückt wird. Es sollte deshalb ein zweiter, von der Ereignisbehandlung unabhängiger Pfad vom Lokalisierungsservice für den Transport von Alarmen angeboten werden, der einen sicheren Weitertransport garantiert, wobei nur die vom Lokalisierungswerkzeug gemeldete Position übersetzt und allenfalls anhand eines separaten Regelsets die Meldung transformiert wird.

Ereignis-Publikation

Die sinnvollste Möglichkeit zur Ereignis-Publikation dürfte die Nutzung einer Message Queue sein, da Message Queues Publish Subscribe Channels und die Erstellung eines Message Bus ermöglichen sowie Message Routing und Filtering unterstützten.

7.5.4 Umsetzung

Statusableitung

Als Kompromiss zwischen einfacher Nutzung ohne nachgelagerte Intelligenz, die interpretierte Status wie *Person operiert* oder *Person besetzt* benötigen würde, und Präzision bei Vorhandensein nachgelagerter Intelligenz, die Aussagen wie *Person in Operationssaal* in einen Status ummünzen kann, wird ermöglicht, in den Regeln zur Status-Interpretation hinterlegen zu können, welcher Status gesendet wird.

Ereignis-Erzeugung

Um eine zuverlässige Ereignis-Erzeugung zu garantieren, muss die Zeit als Taktgeber verwendet werden, um regelmässig überprüfen zu können, ob ein Ereignis eingetreten ist oder nicht. Dies hat allerdings negative Auswirkungen auf die Reaktionszeit, die sich mildern lassen können, indem je nach Gerätekategorie das Intervall grösser oder kleiner gewählt werden kann. Die Architektur wird ausserdem so ausgelegt, dass sich bei besserer Unterstützung durch die Lokalisierungswerkzeuge Events zusammen mit der Zeit als Taktgeber verwenden lassen.

Konkret könnte dies so aussehen, dass die Event-Überprüfungsjobs, die in einem regelmässigen Intervall ausgeführt werden, als Timeout-Wächter aufgefasst werden. Sie laufen an, falls kein

Event innerhalb des Intervalls eintrifft. Trifft ein Event ein, wird dieser behandelt und daraufhin die Ausführung des Timeout-Wächters um das Intervall verschoben. So profitiert man einerseits von einer besseren Reaktionszeit durch die Verarbeitung von Events und verfügt trotzdem über ein robustes Modell, das mit unzuverlässigen oder ganz ausbleibenden Meldungen umgehen kann.

Ereignis-Erkennung

Die Ereignis-Erkennung wird analog zur Interpretation der Positionsmeldungen durchgeführt.

Alarm-Behandlung

Um eine zuverlässige Alarm-Behandlung zu ermöglichen, wird für die Alarm-Behandlung ein gesonderter Pfad innerhalb des Lokalisierungsservice geschaffen, der einzig die Positionsmeldungen umwandelt und mit Regeln eine Transformation des Alarms ermöglicht. Die Alarme werden daraufhin gespeichert und analog zu den Ereignissen publiziert. Eine Positionsinterpretation wird nicht durchgeführt, um die Unterdrückung von Alarmen zu verhindern.

Ereignis-Publikation

Zur Ereignis-Publikation wird ein JMS Message Broker verwendet.

7.6 Kontinuierliche Positionserfassung

7.6.1 Problemstellung

Wie in Abschnitt 7.1 (Seite 19) dargelegt, wird eine möglichst lückenlose Positionsverfolgung benötigt. Da das die Konnektoren nicht von sich aus bewerkstelligen können (siehe Abschnitt 7.2, Seite 20) und auch Ereignisse vom Lokalisierungsservice ausgelöst werden müssen (siehe Abschnitt 7.5, Seite 31), muss der Lokalisierungsservice mit der Fähigkeit ausgestattet werden, von sich aus tätig zu werden, also Geräte zu lokalisieren und bei Bedarf Ereignisse auszulösen.

7.6.2 Anforderungen

Aufgrund der Anforderungen aus Abschnitt 7.1 (Seite 19) und 7.5 (Seite 31) muss die kontinuierliche Positionserfassung mit bis zu 100’000 Geräten und einem Intervall zwischen 60 Sekunden und 10 Minuten umgehen können.

7.6.3 Lösungsansätze

Da der Lokalisierungsservice von sich aus tätig werden muss und sich nicht auf externe Ereignisse verlassen kann (siehe Abschnitt 7.2, Seite 20, und 7.5, Seite 31), bleibt nur noch die Zeit als Taktgeber. Dies bedeutet, dass mit Hilfe eines Scheduler regelmässig Jobs zur Positionsbestimmung angestossen werden müssen. Aufgrund der grossen Menge an zu erwartenden Jobs und aus Redundanzgründen muss die Ausführung verteilt erfolgen können.

Java bietet selber mit dem Executor-Framework und `ScheduledThreadPoolExecutor` eine entsprechende Infrastruktur [11] (Seite 123 ff.), die aber nicht für verteilte Ausführung gedacht ist und entsprechend erweitert werden müssen. Gleiches gilt für die verschiedenen Klone der aus UNIX bekannten Crontab.

Verteiltes Scheduling bietet hingegen das Actor-Framework Akka[27] mit dem Scheduler-Modul[28] und der Quartz Enterprise Job Scheduler[29], ein Scheduling-Framework für Java. Die Nutzung von Akka hätte den Vorteil, dass man das robuste Actor-Modell für die Parallelprogrammierung verwenden könnte. Leider ist der Akka-Scheduler etwas gar spartanisch und bietet beispielsweise keine Job-Verwaltung, die hinzuprogrammiert werden müsste. Der Quartz Enterprise Scheduler bietet vielfältige Funktionalität. Sein Nachteil ist allerdings, dass er zur verteilten Ausführung ein RDBMS benötigt, das für ihn als Job Queue dient. Dies kann unter Umständen zu Skalierungsproblemen führen, da Quartz nach jedem Job Daten zurück ins RDBMS schreiben muss. Es ist aber möglich, das `JobStore`-Interface, das Quartz zur Interaktion mit dem RDBMS verwendet, neu zu implementieren, beispielsweise auf Basis von MongoDB. Eine andere Variante besteht darin, den ebenfalls von Terracotta stammenden Ehcache[30] zu verwenden, der Leseoperationen zwischenspeichern und Schreiboperationen über einen grösseren Zeitraum verteilen und zusammenfassen[31] kann.

7.6.4 Umsetzung

Es wird der Quartz Enterprise Job Scheduler als Taktgeber für die Abfrage der Lokalisierungswerkzeuge und die Ereignis-Erkennung verwendet. In einer ersten Phase wird die integrierte Unterstützung für RDBMS als Job Store genutzt. Bei Bedarf kann in einer zweiten Phase ein eigener Job-Store-Konnektor für MongoDB entwickelt werden.

7.7 Datenanalyse

7.7.1 Problemstellung

Wie in Abschnitt 7.3 (Seite 23) dargelegt, fallen pro Monat einige Dutzend bis Hundert Gigabyte an Daten an, die je nach Use Case unterschiedlich tief analysiert werden müssen. Für die Frage, welche Geräte sich in den letzten 24 Stunden bewegt haben, reichen die paar Gigabyte, die während eines Tages angefallen sind. Bei der Anfertigung einer Nutzungsstatistik für den letzten Monat muss der gesamte Datenbestand herangezogen werden. Das Auftreten solcher Analyse-Aufträge schwankt von mehrmals täglich (Änderungen am letzten Tag usw.) bis zu monatlich (Nutzungsstatistik usw.).

Berechnungen in Randzeiten durchzuführen, ist nur beschränkt praktikabel, weil einerseits in Spitälern rund um die Uhr gearbeitet wird und andererseits das Gesamtsystem durch die kontinuierliche Positionserfassung einigermassen gleichmässig ausgelastet ist. Das bedeutet, dass Randzeiten nur beschränkt existieren. Hinzu kommt, dass viele Anfragen wie «Veränderungen

[27] `http://akka.io/` (abgerufen: 18. März 2011)
[28] `http://doc.akka.io/scheduler` (abgerufen: 18. März 2011)
[29] `http://quartz-scheduler.org/` (abgerufen: 18. März 2011)
[30] `http://ehcache.org/` (abgerufen: 18. März 2011)
[31] `http://ehcache.org/documentation/write_through_caching.html` (abgerufen: 18. März 2011)

seit gestern» während der Arbeitszeit gestellt werden und eine zeitnahe Antwort gewünscht wird.

7.7.2 Anforderungen

Anfragen, die sich auf Daten eines oder einiger weniger Tage (bis zu einer Woche) beziehen, sollten in unter 30 Minuten beantwortet werden, alle übrigen Anfragen in unter einer Stunde. Die Beantwortung von Anfragen sollte nach Möglichkeit den Regelbetrieb nicht beeinflussen und zu diesem Zweck die Daten möglichst lokal gehalten werden. Deshalb soll auch festgelegt werden, welche Daten zur Bearbeitung der Use Cases benötigt werden.

7.7.3 Lösungsansätze

Analyse

Die einfachste Möglichkeit ist die Nutzung der von RDBMS und SQL gebotenen Abfragemöglichkeiten wie Aggregatsfunktionen. Das bereits im Rahmen des Projekts verwendete PostgreSQL ist diesbezüglich gut aufgestellt und bietet vielfältige Möglichkeiten zur Erstellung von Stored Procedures, sodass sich die geforderten Analysefunktionen durchaus formulieren lassen sollten. Problematisch ist, dass Abfragen, die die Daten mehrerer Tage umfassen, schnell einen Full Table Scan provozieren und Schreiboperationen aufgrund der Ausführungsdauer aufhalten werden. Es wäre demnach zwingend, die Abfragen auf separaten Datenbank-Instanzen auszuführen. Mit PL/Proxy[32] liessen sich Abfragen auf mehreren Servern parallel ausführen, was aber viel Handarbeit erfordert. Ein weiterer Nachteil wäre, dass ein RDBMS als Datenspeicher für die Historie benötigt würde, was hinsichtlich der Anforderungen nicht optimal ist (siehe Abschnitt 7.3, Seite 23).

Das von Google eingeführte Map Reduce scheint für die Datenanalyse besser geeignet zu sein, da es auf grosse Datenmengen angelegt ist und sich automatisch um Partitionierung und parallele Ausführung der Operationen kümmert [7]. Die in Rahmen der Use Cases benötigten Abfragen sollten sich einigermassen leicht als Map Reduce Jobs ausdrücken lassen, da es sich meist um Aggregatsoperationen handelt. Map Reduce lässt sich zusätzlich zu einem RDBMS verwenden, indem man die Daten zum Beispiel in Hadoop lädt. So liesse sich für einfache Anfragen, die auf wenige Datensätze benötigen, direkt das RDBMS verwenden und bei komplexeren Problemen auf Hadoop zurückgreifen. Wird eine nichtrelationale Datenbank verwendet, kann ebenfalls Hadoop verwendet oder auf die oftmals vorhandene, bereits eingebaute Unterstützung für Map Reduce gesetzt werden.

Benötigte Daten

Damit die Analysen auf lokalen Daten ausgeführt werden können und nicht regelmässig in einer Datenbank nachgesehen werden muss, um beispielsweise Relationen auflösen zu können, sollte jeder Datensatz aus folgenden Daten bestehen:

- Gerät beziehungsweise Person
- Gerätetyp beziehungsweise Rolle

[32] `http://pgfoundry.org/projects/plproxy/` (abgerufen: 18. März 2011)

- Position (Sektor, Stockwerk, Gebäude, Zone)
- Datum

Damit lassen sich Anfangen bezüglich der in der Liste aufgeführten Daten befriedigen, was den in den Use Cases erfassten Anforderungen entspricht.

7.7.4 Umsetzung

Analyse

Wie in Abschnitt 7.3 (Seite 23) festgelegt, wird MongoDB als Datenspeicher verwendet. Dies ermöglicht, SQL-ähnliche Operationen für Abfragen auf kleinen Datenmengen und Map Reduce für aufwendigere Analyse zu verwenden.

Benötigte Daten

Jeder Datensatz besteht aus folgenden Daten:

- Gerät beziehungsweise Person
- Gerätetyp beziehungsweise Rolle
- Position (Sektor, Stockwerk, Gebäude, Zone)
- Datum

7.8 Datenschutz und Vorschriften

7.8.1 Problemstellung

Hinsichtlich Datenschutz und Sicherheit ist der Medizinalbereich stark reguliert. So werden beispielsweise Gesundheitsdaten von Gesetzes wegen als besonders schützenswert eingestuft[33] und sind auch durch das Strafgesetzt über das Berufsgeheimnis geschützt[34]. Ebenso müssen bei der Bearbeitung von Personendaten die einschlägigen Datenschutzgesetze beachtet werden[35]. Auch hinsichtlich der eingesetzten Betriebsmittel werden besondere Anforderungen gestellt. Wird ein Betriebsmittel beispielsweise zur Überwachung von Patienten verwendet – beispielsweise um sicherzustellen, dass sie die nötigen Behandlungsschritte durchlaufen haben –, müssen unter anderem die Vorschriften der Medizinprodukteverordnung beachtet werden[36]. Da mit der im Rahmen des Projekts entwickelten Software alle erwähnten Punkte tangiert werden, muss im besonderen Masse darauf geachtet werden, dass die relevanten gesetzlichen Vorschriften beachtet und umgesetzt werden.

[33] SR 235.1 Bundesgesetz über den Datenschutz, Art. 3, `http://www.admin.ch/ch/d/sr/235_1/a3.html` (abgerufen: 4. Juni 2011)

[34] SR 311.0 Schweizerisches Strafgesetzbuch, Art. 321, `http://www.admin.ch/ch/d/sr/311_0/a321.html` (abgerufen: 4. Juni 2011)

[35] SR 235.1 Bundesgesetz über den Datenschutz, Art. 4, `http://www.admin.ch/ch/d/sr/235_1/a4.html` (abgerufen: 4. Juni 2011)

[36] SR 812.213 Medizinprodukteverordnung, Art. 1, `http://www.admin.ch/ch/d/sr/812_213/a1.html` (abgerufen: 4. Juni 2011)

7.8.2 Anforderungen

In einem ersten Schritt soll überprüft werden, ob sich die geplante Funktionalität überhaupt umsetzen lässt oder von Gesetzes wegen von vornherein unmöglich ist. Die Feststellung der relevanten Vorschriften, die bei der Umsetzung eingehalten werden müssen, und die Sicherstellung deren Einhaltung ist nicht Teil der Arbeit.

7.8.3 Ergebnisse

Unter Mithilfe von Robert Thoma, Leiter Informationssicherheit des Universitätsspitals Zürich, wurde betrachtet, ob Positionen erhoben, gespeichert und weiterverarbeitet werden dürfen, ob eine Analysemöglichket der erhobenen Daten gewährt werden kann und welche Kontroll- beziehungsweise Steuerungsmöglichkeiten den betroffenen Personen gewährt werden muss. Dabei hat sich gezeigt, dass die Umsetzung der skizzierten Funktionalität hinsichtlich Gerätelokalisierung grundsätzlich möglich ist, wobei sich keine Rückschlüsse von der Gerätelokalisierung auf Personeninformationen ziehen lassen dürfen. Hinsichtlich der Personenlokalisierung ist diese ebenfalls grundsätzlich möglich, allerdings unter stark erhöhten Anforderungen. So sind beispielsweise schriftliche Einwilligungen von verschiedenen Stellen notwendig und sehr wahrscheinlich eine Vorprüfung durch den kantonalen Datenschutzbeauftragten[37]. Zusätzlich werden besondere Anforderungen an die Nachvollziehbarkeit gestellt. Für den aktuellen Stand der Arbeit ist zudem zu beachten, dass Personen aktive Lokalisierungsmittel jederzeit ein- respektive abstellen und passive Lokalisierungsmittel ablegen können müssen.

Es sei an dieser Stelle aber nochmals betont, dass diese Betrachtung keinesfalls vollständig oder abschliessend ist und die Thematik von einer etwaigen Folgearbeit gesondert und vertieft betrachtet werden muss.

[37] `http://www.datenschutz.ch/fileadmin/user_upload/datenschutz/04_Publikationen/Merkblatt_Vorabkontrolle.pdf` (abgerufen: 4. Juni 2011)

8 Konnektoren

In Abschnitt 7.2 (Seite 20) wurde das allgemeine Vorgehen zur Anbindung der einzelnen Lokalisierungslösungen an den Lokalisierungsservice analysiert. In den folgenden Abschnitten wird die konkrete Anbindung folgender Lokalisierungswerkzeuge über entsprechende Konnektoren diskutiert:

- Ekahau Positioning Engine (WLAN-Lokalisierung)
- CiscoWorks LAN Management Solution (LAN-Lokalisierung)
- Smartphone (Satellitenortung, Mobilfunk)

8.1 Ekahau Positioning Engine

8.1.1 Ausgangslage

In der dieser Arbeit vorausgegangenen Studienarbeit [1] wurde eine Anbindung für die Ekahau Positioning Engine bereits realisiert. Die Position eines WLAN-fähigen Geräts wurde damals jedoch nicht über die von Ekahau gelieferte Position bestimmt. Statt dessen wurde der von Ekahau gelieferte Sektorname übernommen. Dazu ist es allerdings nötig, vor Inbetriebnahme die Sektoren (in Ekahau Zone genannt) auf der Ekahau Positioning Engine zu hinterlegen. Diese Lösung hat den Nachteil, dass die Sektoren zweimal eingezeichnet werden müssen: Zum einen mit einem Ekahau-Werkzeug für die Ekahau Positioning Engine und zum anderen während der Geometriedigitalisierung für den Lokalisierungsservice.

8.1.2 Problemstellung

Es geht nun darum, eine Lösung zu finden, wie die Positionsangaben der Ekahau Positioning Engine direkt verwendet werden können, ohne dass der Umweg über die Ekahau-Zonen gemacht werden muss. Ausserdem müssen die von der Ekahau Positioning Engine gelieferten Positionsangaben in die Landeskoordinaten umgewandelt werden, um den Schnittstellen-Vertrag mit dem Lokalisierungsservice einzuhalten.

Zudem muss definiert werden, in welcher Form der Konnektor realisiert werden soll: Als Pull-Konnektor, welcher Positionsinformationen auf Anfrage des Lokalisierungsservice hin liefert, oder als Push-Konnektor, welcher die Positionen bei Bedarf selber einliefert.

8.1.3 Lösungsansätze

Umwandlung der Koordinaten

Die von Ekahau gelieferten Positionsinformationen setzen sich aus einer Map (im Normalfall einem Stockwerkplan im Form eines Bildes) und einem Koordinaten-Paar zusammen, das die

```
<TAG>
  <tagid>105463687937</tagid>
  <mac>00:18:8E:20:2B:01</mac>
  <posx>1455</posx>
  <posy>600</posy>
  <posmodelid>1670</posmodelid>
  <posmapid>0</posmapid>
  <poszoneid>54</poszoneid>
  <posmapname>Hoer_D_Sitesurvey</posmapname>
  <poszonename>Hoer-D3003 Korridor</poszonename>
  <posquality>22</posquality>
  <posreason>3</posreason>
  <postime>1296555804024</postime>
  <postimestamp>2011-02-01 11:23:24+0100</postimestamp>
  <poscounter>78703</poscounter>
  <battery>100</battery>
</TAG>
```

Listing 8.1: Beispiel-Antwort der Ekahau Positioning Engine auf eine Positionsanfrage

Distanz in Bildpunkten vom Nullpunkt (linker oberer Rand des Map-Bildes) symbolisiert. Ein Beispiel für eine Positionsangabe zeigt Listing 8.1 (Seite 42).

Um den Anforderungen der Konnektor-Schnittstelle zu entsprechen, muss die Position eines Geräts durch eine dreidimensionale Positionsangabe (x, y, z) in Landeskoordinaten ausgedrückt werden. Da die Ekahau Positioning Engine selber keine Angaben über die Höhe macht, muss die z-Koordinate über die Map, also den Stockwerkplan, ermittelt werden. Da die Höhe der Stockwerke bekannt ist, kann den Maps so eine Höhenangabe zugeordnet werden. Zur Ermittlung der anderen zwei Koordinaten muss mittels einer Koordinaten-Transformation von den gelieferten Positionsinformationen von Ekahau auf die Landeskoordinaten geschlossen werden.

Konnektor-Typ

Die Ekahau Positioning Engine bietet grundsätzlich beide Varianten der Positionsabfrage. Die Push-Variante hat den Vorteil, dass diese Event-basiert ist und somit nur dann Meldung erstattet, wenn sich das Gerät bewegt hat. Der Nachteil dabei ist, dass Ekahau die Benachrichtigung mittels Server-Push realisiert hat. Die Problematik hierbei ist, dass der Server-Push nur von einem Server empfangen werden kann. Will man die Aufgabe auf mehrere Server verteilen, so muss sichergestellt werden, dass jede Nachricht ganz genau nur einmal verarbeitet wird. Das macht die Sache sehr kompliziert, da hierfür ein Konsensprotokoll benötigt wird.

Das nächste Problem ist der Umgang mit verlorenen Nachrichten, die durch Serverabstürze oder Exceptions auftreten können. Zwar wird mit der Ekahau-Antwort ein Nachrichtenzähler (`poscounter`) pro Ekahau-Tag mitgeliefert, womit das Fehlen von Nachrichten festgestellt werden kann. Ein direktes Nachfordern einer fehlenden Nachricht ist jedoch nicht möglich. Will man die fehlende Nachricht finden, so müsste die History durchsucht werden. Dies macht die Angelegenheit unnötig komplex und fehleranfällig.

Das Gegenstück, die Pull-Variante hat den Vorteil, dass die Problematik mit verlorenen Nach-

richten und der Synchronisation entfällt. Man bekommt die Informationen, welche man direkt erfragt oder man merkt, dass man sie nicht bekommt. Der Nachteil ist allerdings, dass die Lösung unter Umständen ressourcenintensiver ist.

8.1.4 Umsetzung

Umwandlung der Koordinaten

Für die Umwandlung der Positionsinformationen wird eine Koordinaten-Transformation eingesetzt, welche die von der Ekahau Positioning Engine gelieferten x- und y-Koordinaten entsprechend transformiert. Die Höhe in Meter über Meer wird über die Bezeichnung der Maps zugeordnet.

Konnektor-Typ

Als Konnektor-Typ wird die Pull-Variante umgesetzt, da dabei die Vorteile klar überwiegen. Zusätzlich ist die Komplexität der Push-Lösung zu hoch im Vergleich zu den Vorteilen, die sie mit sich bringen würde.

8.2 CiscoWorks LAN Management Solution

8.2.1 Ausgangslage

Besonders für die Lokalisierung der Anfragesteller ist es nötig, Geräte wie zum Beispiel Workstations über das LAN zu lokalisieren. Für ein bestimmtes Gerät kann mit Hilfe der CiscoWorks LAN Management Solution herausgefunden werden, an welchem Switch-Port sich dieses befindet (siehe [5]). Die Funktionsweise des Lokalisierungswerkzeugs wurde bereits in [1] (S. 18) beschrieben.

8.2.2 Problemstellung

Es sollen Möglichkeiten aufgezeigt werden, wie die Angabe eines Switch-Ports in eine Position in Landeskoordinaten umgewandelt werden kann. Zusätzlich soll eruiert werden, welcher Konnektor-Typ sich zur Abfrage dieses Lokalisierungswerkzeugs eignet.

8.2.3 Lösungsansätze

Umwandlung der Positionsangabe

Um von der Position eines Gerätes anhand des Switch-Portes zu der Position in Landeskoordinaten zu kommen, ist ein Mapping des Ports auf seine Position nötig.

Im Falle des USZ kann die Positionsangabe aus einer manuell hinterlegten Port-Beschreibung extrahiert werden. Diese Beschreibung enthält die Position als Angabe von Trakt (Gebäude), Stock (Stockwerk) und Raum (siehe Abbildung 8.1, Seite 44). Blau gekennzeichnet sind dabei die für eine Abfrage wichtigen Port-Informationen. Aus den Positionsinformationen kann dann auf die dazu passende Position in Landeskoordinaten geschlossen werden.

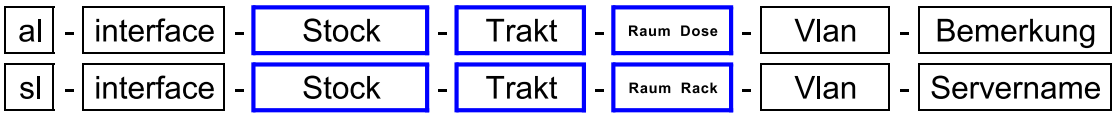

Abbildung 8.1: Die Ports werden im USZ nach diesen Muster beschrieben. Blau gekennzeichnet sind die für die Abfrage der Port-Position relevanten Informationen.

Existiert keine direkte Möglichkeit, von der Port-Beschreibung auf Landeskoordinaten zu schliessen, muss ein anderes Mapping stattfinden. Dies kann so aussehen, dass beispielsweise bei der Erfassung von Geometrie und Topologie die Positionen der Switch-Ports hinterlegt werden.

Konnektor-Typ

Als Konnektor-Typ kommt zur Anbindung der CiscoWorks LAN Management Solution nur die Pull-Variante in Frage. Die Push-Option wird von dem Werkzeug nicht unterstützt, obwohl es selber seine Datenbank über SNMP-Traps aktuell halten kann. Leider ist aber keine Weitergabe dieser vorgesehen.

8.2.4 Umsetzung

Umwandlung der Positionsangabe

Für den Prototyp des USZ kann anhand der aus der Port-Beschreibung gewonnenen Informationen die Umwandlung in Landeskoordinaten stattfinden. Dazu wird aber noch eine Übersetzungstabelle nötig sein, die die Kombination aus Stock-, Trakt- und Raum-Bezeichnung auf Landeskoordinaten abbildet.

Konnektor-Typ

Da CiscoWorks über keine Push-Option verfügt, wird dieses Lokalisierungswerkzeug mit einem Pull-Konnektor angebunden.

8.3 Smartphone

8.3.1 Problemstellung

Smartphones verfügen heutzutage über immer leistungsfähigere Hardware zur Bestimmung ihrer aktuellen Position. Dies macht sie interessant zur Nutzung als Lokalisierungssensor, zumal heute schon sehr viele Personen ein Smartphone besitzen. Hinzu kommt, dass ein Handy-Besitzer in der Regel sein Telefon immer auf sich trägt und somit eine übermittelte Position mit grosser Wahrscheinlichkeit der Position der Person entspricht. Da eine Person das Mobiltelefon auch ausserhalb des Aufenthaltes auf dem Campus bei sich hat, kann das Verlassen und Betreten des Campus darüber sehr gut überwacht werden – oder ist sogar die einzige Möglichkeit. Es soll analysiert werden, welche Voraussetzungen geschaffen werden müssen, damit ein Smartphone als Sensor für den Lokalisierungsservice dienen kann und wie ein passender Konnektor auszusehen hat.

8.3.2 Anforderungen

Die Lokalisierung sollte nach Möglichkeit ohne Zutun des Anwenders arbeiten und diesen auch nicht stören. Da es sich bei Lokalisierungsinformationen um personenbezogene Daten handelt, die einen besonderen Schutz geniessen (siehe Abschnitt 7.8, Seite 39), muss darauf geachtet werden, dass die entsprechenden Vorschriften eingehalten werden.

Als primäre Plattform soll Apples iPhone berücksichtigt werden. Die Ergebnisse sollten sich später aber auch auf andere Plattformen wie Android übertragen lassen.

8.3.3 Lösungsansätze

Smartphone als Lokalisierungssensor

In [1] (S. 18) wurden die Möglichkeiten der Lokalisierung eines Smartphones bereits diskutiert. Dabei hat sich gezeigt, dass das Smartphone nicht von aussen nach seiner Position gefragt werden kann. Statt dessen muss das Smartphone seine Position selbstständig melden. Es ist daher eine App nötig, welche kontinuierlich die aktuelle Position bestimmt und entsprechende Meldungen an den Lokalisierungsservice verschickt. So eine App bringt eine Reihe von Herausforderungen mit sich:

Datenschutz Wie bereits in den Anforderungen erwähnt, handelt es sich bei Lokalisierungsdaten um Personendaten, die nach schweizerischer Gesetzgebung besonders geschützt sind.

Sicherheit Da Smartphones sich vorwiegend in öffentlichen Netzen bewegen, muss die nötige Vertraulichkeit bei der Datenübermittlung sichergestellt werden. Ebenso muss ein praktikabler Weg gefunden werden, wie die Daten an den Lokalisierungsservice übermittelt werden können, ohne die lokale Netzwerksicherheit zu gefährden und sicherzustellen, dass keine Fremden den Lokalisierungsservice stören können.

Energieverbrauch Im Vergleich zu herkömmlichen Mobiltelefonen ist bei Smartphones die Akkulaufzeit deutlich kürzer, oftmals nur ein bis zwei Tage im Stand-by-Modus. Die zu entwickelnde Lösung sollte die Akkulaufzeit daher möglichst wenig beeinträchtigen.

Datenverkehr Da die Übermittlung von Daten übers Mobilfunknetz nach wie vor teuer ist, soll bei der zu gestaltenden Lösung darauf geachtet werden, dass dieser möglichst weitgehend reduziert wird.

Konfigurierbarkeit Um die App universell einsetzen und einfach in Betrieb nehmen zu können, sollte sie einfach über die Device-Management-Lösungen für die entsprechenden Plattformen konfigurierbar sein.

Im Folgenden werden die genannten Herausforderungen untersucht hinsichtlich der Möglichkeiten, wie sie in einer App umgesetzt werden können.

Datenschutz Der Nutzer muss die Datenerfassung jederzeit kontrollieren können (siehe Abschnitt 7.8, Seite 39). Zudem darf der Arbeitgeber seine Angestellten ausserhalb des Arbeitsplatzes nicht überwachen. Um diesen Anforderungen gerecht zu werden, muss die App jederzeit

Abbildung 8.2: Mock-ups für die Smartphone-App, die Möglichkeiten zeigen, wie sich die Datenschutzoptionen steuern und anzeigen lassen. Links: App ist eingeschaltet und die Person befindet sich auf dem Campus. Mitte: App ist eingeschaltet und die Person befindet sich nicht auf dem Campus. Rechts: App ist deaktiviert.

vom Benutzer deaktiviert beziehungsweise aktiviert werden können. Zudem müssen die Dimensionen des Campus hinterlegt werden können, sodass die App den Versand von Positionsmeldungen unterdrücken kann, sobald es sich ausserhalb des Campus befindet und statt dessen nur übermittelt, dass es sich nicht auf dem Campus befindet. Abbildung 8.2 (Seite 46) zeigt eine Möglichkeit, wie der Benutzer die genannten Informationen sehen und steuern kann. Ebenfalls könnte es sinnvoll sein, dem Benutzer eine Möglichkeit zu geben, in Form einer Historie überprüfen zu können, welche Positionen in der Vergangenheit an den Lokalisierungsservice übermittelt wurden.

Sicherheit Da sich Smartphones typischerweise in öffentlichen Netzen befinden, muss einerseits eine vertrauliche Übermittlung der Positionsmeldungen sichergestellt werden. Da es sich beim Lokalisierungsservice um einen internen Service innerhalb des Spitalnetzwerks handelt, muss andererseits gewährleistet sein, dass nur legitimierte Clients Nachrichten einliefern können und dass die Nachrichten unterwegs nicht verändert werden können.

Eine reine Nachrichtenverschlüsselung und Übermittlung im Stile von S/MIME[1] ist unzureichend, weil damit zum Beispiel der Empfänger der Nachricht sichtbar ist oder Nachrichten unterdrückt werden können. Sinnvoller ist daher eine verschlüsselte Verbindung zum Zielserver, womit der Nachrichtenkanal und somit auch die übermittelte Nachricht geschützt ist. Nahe-

[1] `http://tools.ietf.org/html/rfc3851` (abgerufen: 5. Juni 2011)

liegende Lösungen sind damit IPSec[2] und Transport Layer Security (TLS)[3]. Der Nachteil bei IPSec ist, dass IKEv2[4] nach wie vor wenig verbreitet ist und für die Verbindung mit den meisten IPSec-Gateways proprietäre Clients nötig sind. So unterstützt iOS aktuell beispielsweise nur die Implementierungen von Cisco und Juniper[5]. Ein weiteres Problem ist, dass bei Verbindungsunterbrüchen, die beispielsweise bei Zugfahrten häufig auftreten, die IPSec-Verbindung manuell wieder initiiert werden muss. Bei der Lokalisierung, die möglichst autonom arbeiten sollte, empfiehlt sich die Verbindung über IPSec daher nicht. Bleibt also TLS. Es bietet gleich starke Verschlüsselungs- und Authentifizierungsmechanismen wie IPSec, hat aber den Vorteil, dass Verbindungen ohne Benutzerinteraktion aufgebaut werden können. Da nur eine einzelne Verbindung geschützt werden muss, ist es kein Problem, dass im Gegensatz zu IPSec nicht bereits ab Layer 3, sondern erst ab Layer 5 verschlüsselt wird.

Hinsichtlich Authentifizierung bietet es sich an, Client-Zertifikate zu verwenden. Diese können bei iOS beispielsweise mit dem iPhone Configuration Utility hinterlegt werden[6] und liessen sich am Netzwerk-Perimeter validieren. Geht das Smartphone verloren, kann das Zertifikat mittels OCSP[7] gesperrt werden, sofern sich das Smartphone nicht mittels Remote Wipe löschen lässt. Benutzername und Passwört können ebenso verwendet werden, müssen aber in der App hinterlegt werden, wenn sie nicht in den Schlüsselbund des Smartphones eingefügt werden können. Wie die Zertifikate lassen sich Benutzername und Passwort am Netzwerkperimeter überprüfen und können bei Verlust des Geräts gesperrt werden.

Energieverbrauch Die Positionsbestimmung mittels Satellitenortung benötigt viel Energie[8]. Um die Akkulaufzeit zu schonen, sollte die Häufigkeit der Lokalisierung «mit Augenmass» gewählt werden. Smartphones bieten oft die Möglichkeit, einen Schwellenwert für die Verschiebung seit der letzten Positionsbestimmung zu definieren, der überschritten werden muss, bevor eine neuerliche Satellitenortung vorgenommen wird. Es wird empfohlen, diesen ausserhalb des Campus relativ hoch einzustellen, da aus Datenschutzgründen ohnehin nur die Meldung übermittelt wird, dass sich das Smartphone nicht auf dem Campus befindet. Zudem kann das Meldungsintervall hinaufgesetzt werden, weil man nach Verlassen des Campus typischerweise nicht so schnell wieder zurück kommt. Auf dem Campus sollte sich das Smartphone etwas häufiger melden, hier dürfte sich eine genaue Positionsbestimmung aber ebenso wenig lohnen, da Satellitenortung innerhalb von Gebäuden ohnehin praktisch unbrauchbar ist.

Datenverkehr Der Versand von Daten über das Mobilfunknetz ist relativ langsam und ausserdem teuer. Da die Positionsmeldungen nicht unbedingt ein Dienst am Nutzer sind, sollten diese hinsichtlich Datenverkehr möglichst wenig ins Gewicht fallen. Um die Positionsmeldungen zu übermitteln, eignet sich besonders JavaScript Object Notation (JSON), da dieses sehr kompakt und dennoch strukturiert ist. Das Fehlen letzteres ist der Nachteil von herkömmlichen

[2] `http://de.wikipedia.org/w/index.php?title=IPsec&oldid=88980396` (abgerufen: 5. Juni 2011)
[3] `http://tools.ietf.org/html/rfc5246` (abgerufen: 5. Juni 2011)
[4] `http://tools.ietf.org/html/rfc4306` (abgerufen: 5. Juni 2011)
[5] `http://support.apple.com/kb/HT1288?viewlocale=de_DE` (abgerufen: 5. Juni 2011)
[6] `http://images.apple.com/iphone/business/docs/iPhone_Certificates.pdf` (abgerufen: 5. Juni 2011)
[7] `http://tools.ietf.org/html/rfc2560` (abgerufen: 5. Juni 2011)
[8] `http://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/LocationAwarenessPG/LocationAwarenessPG.pdf` (abgerufen: 5. Juni 2011), Seite 12

HTTP-POST-Parametern, weshalb sich eine potentielle Byte-Ersparnis nicht realisieren lässt. Um zusätzlich Datenvolumen einsparen zu können, sollte die JSON-Payload komprimiert werden können (`Content-Encoding: gzip, deflate`).

Konnektor-Gestaltung

Betriebsmodus Da Smartphones nicht von aussen aus lokalisiert werden können, sondern sich nur selber melden können, ist ein Push-Betrieb die einzige Möglichkeit.

Umwandlung der Positionsangabe Satellitenortung verwendet als Bezugssystem World Geodetic System 1984 (WGS84). Für die meisten landesspezifischen Koordinatensysteme existieren passende Umrechnungsformeln, so auch für die Schweizer Landeskoordinaten [4]. Man muss sich allerdings bewusst sein, dass es sich hierbei meist um Näherungsformeln handelt, die eine gewisse Ungenauigkeit mit sich bringen, was aber bei der «Gesamtungenauikgeit» der Smartphone-Lokalisierung mit Abweichungen, die ohne weiteres 40 Meter und mehr betragen können, kaum ins Gewicht fällt. Insofern ist die Umwandlung der Positionsangabe kein Problem.

Damit bleibt die Frage, wo die Umwandlung am besten gemacht wird: Im Rahmen der App oder im Rahmen des Konnektors. Dagegen, dies im Rahmen der App zu erledigen, spricht unter anderem die benötigte Rechenleistung und dass die App jedes Mal angepasst werden muss, wenn das vom Lokalisierungsservice verwendete Bezugssystem gewechselt wird. Dafür spricht nur, dass man sich allenfalls einen Umwandlungsschritt sparen würde, falls das Smartphone die Positionen nicht in WGS84 liefert. Solche Produkte sind uns aber nicht bekannt. Im Gegensatz dazu bestehen beim Konnektor keine Probleme hinsichtlich der benötigten Rechenleistung. Ausserdem ist der Lokalisierungsservice bereits darauf vorbereitet, mit unterschiedlichen Bezugssystemen umzugehen. Sollten weitere Transformationen nötig sein, beispielsweise anhand einer zusätzlichen Umsetzungstabelle, ist ein einmalig änderbarer Konnektor ebenfalls eher dazu geeignet als eine App, die auf mehreren Hundert Smartphones angepasst werden muss.

8.3.4 Umsetzung

Smartphone als Lokalisierungssensor

Datenschutz Die Apps müssen zumindest einen Ein-/Ausschalter und eine Begrenzung der Positionserfassung auf den Campus umfassen. Letztere lässt sich beispielsweise so lösen, dass die Aussengrenze des Campus in Form eines Polygons in Well Known Text (WKT) hinterlegt wird. Dieses lässt sich beispielsweise mit OGR[9] (u.a. C++, Python) oder JTS[10] (Java) einlesen, um zu überprüfen, ob die aktuelle Position innerhalb des Polygons liegt[11][12].

Die Anzeige einer Historie ist optional.

[9] `http://www.gdal.org/ogr/` (abgerufen: 5. Juni 2011)
[10] `http://www.vividsolutions.com/jts/` (abgerufen: 5. Juni 2011)
[11] `http://www.gdal.org/ogr/classOGRGeometry.html#bb4bb4687de9b6f23e61b686177b2856` (abgerufen: 5. Juni 2011)
[12] `http://www.vividsolutions.com/jts/javadoc/com/vividsolutions/jts/geom/Geometry.html#within(com.vividsolutions.jts.geom.Geometry)` (abgerufen: 5. Juni 2011)

Sicherheit Das Mittel der Wahl ist aus Gründen der Einfachheit und Flexibilität ohne Kompromisse bei der Sicherheit TLS. Die Wahl der Authentifizierungsmethode ist freigestellt, wobei aufgrund des einfacheren Device Management zertifikatsbasierte Authentifizierung empfohlen wird.

Grundsätzlich sollte die App so eingestellt werden, dass sie nur mit Endstellen kommuniziert, deren Zertifikate sie mithilfe des lokalen Zertifikatsspeichers validieren kann.

Energieverbrauch Um einen Mittelweg zwischen hohem Energieverbauch und regelmässiger Übermittlung zu finden, wird die Häufigkeit und Grösse der Meldungen entsprechend der Position und der Bewegung angepasst. Vorläufige Empfehlungen:

- Ausserhalb Campus: Meldung alle Stunde, Schwellenwert für aktive Ortung 1 km.
- Innerhalb Campus: Meldung alle 15 Minuten, Schwellenwert für aktive Ortung 10 m.

Aufgrund der aktuell schlechten Genauigkeit innerhalb von Gebäuden sind die Werte vergleichsweise hoch gewählt. Die Werte sollten durch Tests mit aktueller Hardware überprüft und bei Bedarf angepasst werden.

Datenverkehr Die Daten werden über HTTP als JSON (`application/json`) übermittelt. Zusätzlich wird der Server die Möglichkeit bieten, komprimierte Inhalte (`Content-Encoding: gzip, deflate`) zu verarbeiten.

Konfigurierbarkeit Folgende Einstellungen sollten konfigurierbar sein, um eine App an die Gegebenheiten der jeweiligen Installation anpassen zu können:

- Server-Adresse
- Nachrichten-Intervall auf dem Campus
- Nachrichten-Intervall ausserhalb des Campus
- Genauigkeit für die Event-Auslösung auf dem Campus
- Genauigkeit für die Event-Auslösung ausserhalb des Campus
- Campus-Dimension

Konnektor-Gestaltung

Betriebsmodus Der Smartphone-Konnektor wird als Push-Konnektor realisiert. Die Nachrichten können über HTTP und JSON eingeliefert werden.

Umwandlung der Positionsangabe Die Apps senden ihre Positionsmeldungen in WGS84. Die Umwandlung ins jeweilige Bezugssystem erfolgt mittels einer Umrechnungsformel im Konnektor.

Teil III

Umsetzung Geometriedigitalisierung

9 Einführung

Wie in der Analyse (Kapitel 6, Seite 17) beschrieben, wird die Vorgehensweise zur Geometriedigitalisierung aus der Studienarbeit [1] übernommen und entsprechend den hinzugekommenen Anforderungen angepasst. Der Shapeconverter, welcher die Geometriedigitalisierung realisiert, soll am Ende in Form eines Prototyps vorliegen. Dieser soll die nötigen Daten für eine Pilot-Installation in einem einzelnen Gebäude einlesen können. Verbindungen zu anderen Gebäuden sind im Moment nicht nötig, sollen aber bei der Umsetzung berücksichtigt werden.

10 Funktionale Anforderungen

In den nachfolgenden Abschnitten sind die funktionalen Anforderungen an den Shapeconverter beschrieben. Dabei wurde die Methodik nach [14] verwendet. Im Abschnitt 10.1 (Seite 55) sind die User Stories in 1 bis 2 Sätzen beschrieben. Sie beschreiben die Funktionalität des Systems. Die Master Story List in Abschnitt 10.2 (Seite 55) zeigen eine Übersicht über die User Stories, die Aufwandschätzung zur Umsetzung und die Umsetzungspriorität. In Abschnitt 10.3 (Seite 55) werden mittels «Einseiter» die Aufgaben und zu erfüllenden Kriterien zur Umsetzung der User Stories wie die Bezeichnung schon sagt auf etwa einer Seite beschrieben.

10.1 User Stories

Die User Stores zeigen Abbildung 10.1 (Seite 56) und 10.2 (Seite 56).

10.2 Master Story List

Die Tabelle 10.1 (Seite 55) zeigt die User Stories in ihrer Umsetzungsreihenfolge, wobei die User Story, welche am Anfang steht, diejenige ist, welche als erstes umgesetzt werden soll. Die «Wichtigkeit» sinkt, je weiter unten eine User Story steht. Die zusätzlich angegebene Gewichtung beschreibt, wie gross der Umsetzungsaufwand geschätzt wird. 1 steht für einen kleinen, 3 für einen mittleren und 5 für einen grossen Aufwand.

10.3 Einseiter

10.3.1 Story Name: Geometrie und Topologie des Campus einlesen und für Datenbank-Import vorbereiten

Beschreibung

Geometrie und Topologie liegen als Shapefiles vor, die aus ArcGIS exportiert wurden. Bevor sie in die Datenbank importiert werden können, müssen sie aufbereitet, Metadaten wie Raumbezeichnungen extrahiert, ein Graph für die Laufwege aufbereitet und die Distanzen zwischen

User Story	Aufwand	Status
Geometrie und Topologie für Datenbank-Import vorbereiten	5	erledigt
Konnektorspezifische Referenzdaten für Datenbank-Import vorbereiten	3	–

Tabelle 10.1: Master Story List

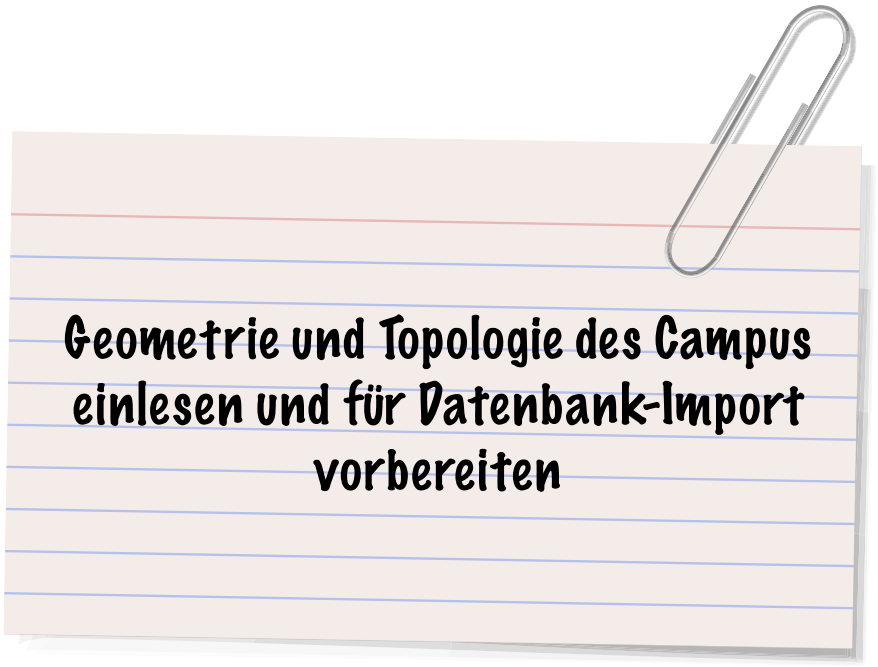

Abbildung 10.1: User Story «Geometrie und Topologie für Datenbank-Import vorbereiten»

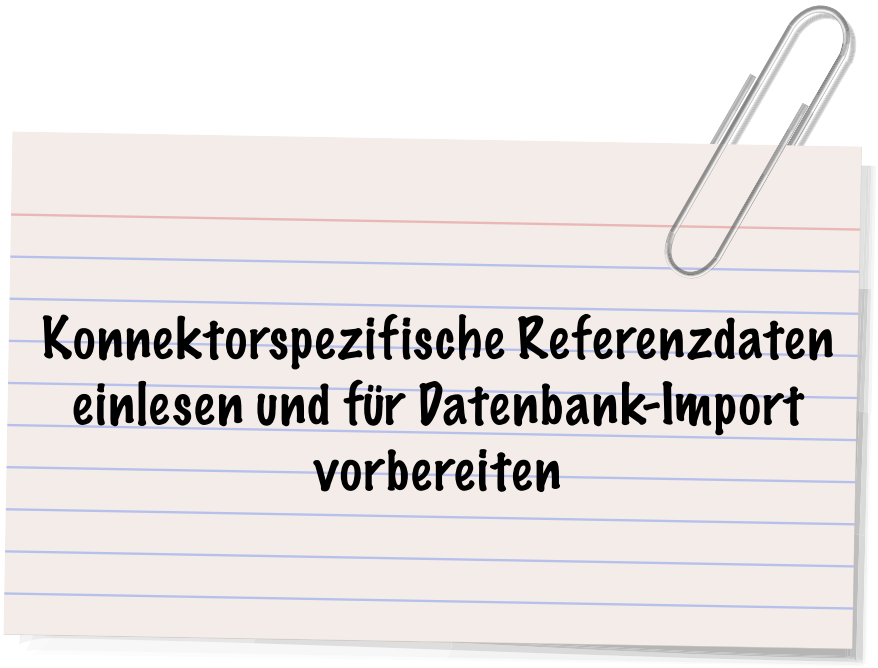

Abbildung 10.2: User Story «Konnektorspezifische Referenzdaten für Datenbank-Import vorbereiten»

allen Wegpunkten berechnet werden. Aus den Informationen werden dann SQL-Dateien für den Import in die Datenbank erzeugt.

Aufgaben

1. Shapefiles mit Geometrie (Sektoren) aus Dateisystem einlesen.
2. Shapefiles mit Topologie (Gehwegen) aus Dateisystem einlesen.
3. Geometrie aus den Shapefiles erzeugen.
4. Topologie aus den Shapefiles erzeugen.
5. Stockwerke innerhalb eines Gebäudes an ihren Übergängen (Lift, Treppe) miteinander verbinden.
6. Für Prototyp noch nicht nötig: Gebäude an den Übergängen miteinander verbinden.
7. Distanzen zwischen allen Positionen berechnen.
8. Aus der Geometrie und Topologie SQL-Dateien erzeugen und im Dateisystem ablegen.

Testkriterien

Die Testkriterien wurden nur für die in den Aufgaben beschriebenen Punkte definiert, welche für den Prototyp nötig sind:

- Korrekt erstellte Shapefiles können eingelesen werden.
- Die aus den Datenbank-Files erzeugten SQL-Dateien (z.B mittels QGIS aus PostGIS ausgelesen) stimmen mit den eingelesenen Shapefiles überein.
- Die Distanzen zwischen den Punkten stimmen mit den in den Stockwerkplänen gemessenen Distanzen überein. Besonderes Augenmerk soll dabei auf die Distanzen zwischen Punkten, welche sich auf verschiedenen Stockwerken befinden, gelegt werden.

11 Nichtfunktionale Anforderungen

11.1 Funktionalität

11.1.1 Stockwerkpläne

Die Stockwerkpläne werden als SHP-Dateien eingelesen. Dazu wird von vorhandenen Computer-Aided Design (CAD)-Dateien als Drawing eXchange Format (DXF) ausgegangen. Diese werden dann entsprechend der Spezifikationen in Abschnitt 12 (Seite 63) bearbeitet und als SHP-Dateien mit den entsprechenden Attributtabellen gespeichert.

11.1.2 Logische Organisation

Die kleinste lokalisierbare Einheit, der Sektor, kann bezüglich der Grösse frei gewählt werden. Es ist zudem auch möglich, Sektoren mit Aussparungen zu realisieren, indem ein Sektor aus einem äusseren Polygon und pro Aussparung einem inneren Ring zusammengesetzt wird. Ein Sektor befindet sich auf einem bestimmten Stockwerk in einem bestimmten Gebäude. Stockwerk oder Gebäude übergreifende Sektoren sind nicht möglich.

Ein Sektor soll einer lokalisierbaren Einheit entsprechen, welche von den Anwendern erkannt wird. Es empfiehlt sich, die Sektoren gemäss den Zimmern zu wählen. Mit dessen Bezeichnung ist sofort klar, in wo sich ein lokalisiertes Objekt befindet. Es aber durchaus auch möglich ein grösserer Raum in mehrere Sektoren einzuteilen. Dabei empfiehlt es sich, die Sektorbezeichnung so anzupassen, dass klar ist, welcher Sektor welcher Teil des Raumes ist (z.B Raum: «Gang 23» in die beiden Sektoren «Gang 23 Ost» und «Gang 23 West» einteilen). Umgekehrt können mehrere Räume in einem Sektor zusammengefasst werden. Auch hier soll die Sektorbezeichnung klar zeigen, welche Räume gemeint sind (z.B Raum «Besprechungszimmer 1a» und Raum «Besprechungszimmer 1b» in den Sektor «Besprechungszimmer 1» zusammenfassen).

Sektoren können einer Zone zugewiesen werden. Welche Sektoren und ob ein Sektor überhaupt zu einer Zone gehört, kann frei gewählt werden. So ist es möglich, Sektoren aus verschiedenen Gebäuden und Stockwerken zusammen als eine Zone zu definieren.

11.1.3 Referenzdaten für Konnektoren

Konnektoren können zusätzliche Referenzdaten benötigen (z.B. Positionen von RFID-Barrieren). Die Referenzdaten sollten, sofern sie einen Bezug zu Geometrie oder Topologie haben, ebenfalls in die Attributtabellen der Stockwerkgrundrisse eingebettet werden und somit als SHP-Datei vorliegen können. Ziel ist, sie später automatisiert einlesen zu können, um Referenzdatenbanken zu erzeugen.

11.2 Usability

Da es sich bei dieser Software um einen Prototyp handelt, wird auf eine graphische Benutzeroberfläche verzichtet. Es ist ausreichend, wenn sich die Funktionalität über eine Konsolen-Eingabe realisieren lässt.

11.3 Zuverlässigkeit

11.3.1 Fehlerbehandlung

Fehler beim Einlesen

Können einzelne Elemente nicht eingelesen werden, so werden diese ignoriert. Die Begründung, weshalb ein Element nicht eingelsen werden konnte, wird in einem Log festgehalten.

11.4 Effizienz

11.4.1 Verarbeitungszeit

Da die Grundrisse selten eingelesen werden müssen und die Daten (dabei fällt vor allem die Berechnung der Distanzen auf dem Graph ins Gewicht) vorberechnet werden, steht die Verarbeitungszeit nicht im Vordergrund. Wichtig ist, dass die erwarteten Datenmengen in nützlicher Frist verarbeitet werden können, wobei eine nützliche Frist weniger als eine Woche ist.

11.4.2 Leistungsvermögen

Der Prototyp muss mit folgenden Datenmengen umgehen können:

Anzahl Sektoren	< 20’000

11.5 Änderbarkeit

11.5.1 Adaptionsfähigkeit

Da für Konnektoren unter Umständen zusätzliche Referenzdaten benötigt werden, sollten sich in den Digitalisierungsprozess Plug-ins einhängen lassen, mit denen sich zusätzliche Referenzdaten aus weiteren Shapefiles einlesen und daraus dann Dateien zur Datenbank-Initialisierung erzeugen lassen. Dabei soll die bereits vollständig eingelesene Geometrie als «Nachschlagewerk» zur Verfügung stehen.

11.6 Implementierung

11.6.1 Plattform

Eine Plattform wird nicht vorgegeben. Die resultierende Software sollte aber auf allen grösseren Plattformen betrieben und um eine grafische Benutzeroberfläche erweitert werden können. Naheliegende Möglichkeiten sind Java 6 mit Swing oder Python mit wxWidgets/PyObjC.

11.6.2 Komponenten

Bezüglich Komponenten-Auswahl bestehen keine spezifischen Vorgaben.

11.6.3 Formate

Als Import-Formate für Geometrie und Topologie wird das DXF von AutoCAD, welches mittlerweile von sehr vielen CAD-Programmen lesbar ist und das Shapefile-Format von Environmental Systems Research Institute Inc. (ESRI) unterstützt.

11.7 Unterstützte Hardware und Software

11.7.1 Hardware

Die Software sollte sich auf einem beliebigen x86-Computer ausführen lassen. Es steht genug Arbeitsspeicher zur Verfügung, um die gesamten Arbeitsdaten im RAM zu halten.

11.7.2 Software

Die Software sollte sich auf einem aktuellen Linux, MacOS X oder Windows ausführen lassen.

11.8 Dokumentation

11.8.1 Code-Dokumentation

Zur Dokumentation des Codes wird eine JavaDoc-Dokumentation erstellt. Zusätzlich wird ein Software-Architektur-Dokument erstellt, das die Grundzüge der Architektur dokumentiert.

11.9 Auslieferung

11.9.1 Packaging

Die Anwendung wird als self-contained Java Archive (JAR) samt aller Abhängigkeiten ausgeliefert.

11.9.2 Deployment

Das Deployment erfolgt manuell.

11.10 Rechtliche Aspekte

Der Shapeconverter ist denselben Regeln unterworfen wie das Gesamtsystem, für die sie in Abschnitt 18.10 (Seite 134) festgehalten sind.

12 Spezifikation zur Erzeugung der Shapefiles

Die nachfolgenden Kapitel beschreiben die Erzeugung der Shapefiles, welche anschliessend mit dem Shapeconverter eingelesen werden können. Es werden nur diejenigen Aspekte beschrieben, welche sich zu den in [1] (Seite 24 ff.) beschriebenen Punkten unterscheiden, genauer spezifiziert wurden oder neu hinzugekommen sind.

12.1 Sektoren und Graph einzeichnen

12.1.1 Sektoren

Die Sektoren, welche als kleinste lokalisierbare Einheiten dienen, müssen für das Einlesen ins System in Form von Shapefiles vorliegen. Als Ausgangslage dienen DXF-Dateien der einzelnen Stockwerke, welche später einfach als Shapefiles exportiert werden können. Das Vorgehen entspricht im Grundsatz dem in [1] (Seite 27 ff.).

Existieren Stockwerkpläne in einem anderen CAD-Format, können diese auch verwendet werden, vorausgesetzt sie lassen sich als DXF-Datei (pro Stockwerk nur eine) exportieren, ohne dass die Layer-Information verloren geht.

Im Vergleich zu dem in [1] (Seite 27 ff.) beschriebenen Vorgehen, dürfen sich Sektoren – auch wenn schlussendlich deren Höhe unterschiedlich sein wird – nicht überschneiden. Für die Umwandlung von 2D in 2.5D wäre sonst eine eindeutige Zuweisung eines Wegpunktes zu einem Sektor nicht möglich. Dies ist insbesondere für die Stockwerkübergänge via Treppenhaus wichtig. Das Vorgehen zum Einzeichnen der Sektoren für die Treppenübergänge, welches sich von dem in [1] beschriebenen Vorgehen unterschiedet, ist in Abschnitt 12.1.3 (Seite 64) beschrieben. Um den Export und anschliessend den Import für das Georeferenzieren zu vereinfachen, müssen zwei verschiedene Layer existieren. Einer für die Polygone (Sektoren) und einer für die Polylines (Gehwege).

12.1.2 Graph

Das Einzeichnen des Graphen erfolgt im Grundsatz gleich wie in [1] (Seite 27 ff.) beschrieben. Damit das Zusammenhängen der Stockwerke sauber möglich ist, unterschiedet sich aber das Vorgehen zum Einzeichnen von Treppen und Liften. Das Vorgehen zum Einzeichnen des Graphen an den Treppenübergängen ist in Abschnitt 12.1.3 (Seite 64) beschrieben. Ein zusätzlicher Punkt, der beachtet werden muss, ist, dass sich kein Start- oder Endpunkt einer Linie genau auf einer Kante eines Polygons befindet. Ansonsten kann dieser dem Sektor nicht zugeordnet werden und wird beim Einlesen ignoriert.

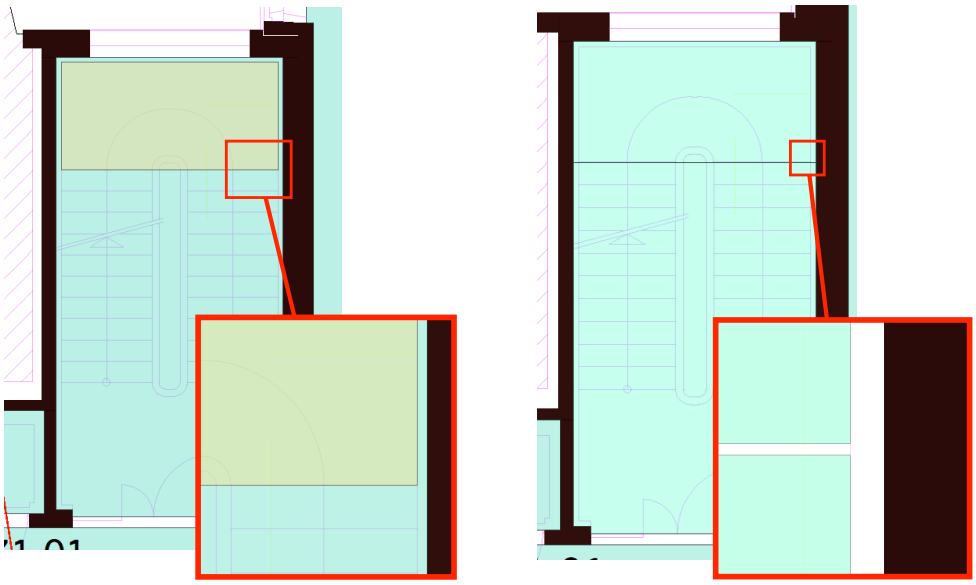

Abbildung 12.1: Realisierung der Sektoren bei einem Stockwerkübergang mit Treppen mit einem Zwischenboden. Links die bisherige Variante aus [1], rechts die neue Variante ohne überschneidende Sektoren

12.1.3 Stockwerkübergänge

Lift

Stockwerkübergänge, die durch einen Lift realisiert werden, werden gemäss dem in [1] beschriebenen Vorgehen eingezeichnet.

Treppen

Für eine Treppe im Sinne eines Treppenhauses wird im jeweils unteren Sektor der Zwischenboden eingezeichnet. Die Sektoren dürfen sich nicht wie in [1] überschneiden. Der Unterschied der Sektoren ist in der Abbildung 12.1 (Seite 64) graphisch dargestellt.

Da der einzelne Stockwerkplan jeweils nur den Zwischenboden kennt, welcher zum nächst höheren Stockwerk führt, kann die Zusammenführung des Graphen nicht auf den Zwischenböden stattfinden. Ansonsten müsste dem höheren Stockwerk das Wissen über den Zwischenboden zum nächst tieferen Stockwerk mitgegeben werden, was dann wieder zu der Problematik mit mehreren übereinander liegenden Sektoren führen würde. Deshalb wurde das Konzept zum Einzeichnen der Gehwege komplett überarbeitet.

Der Weg, welcher jeweils ins nächst höhere Stockwerk führt, wird bis knapp an das Ende des Zwischenbodens im unteren Stockwerk eingezeichnet. Die Verbindung vom oberen Stockwerk ins Untere wird knapp an das Ende es Sektors gezeichnet, bevor der Zwischenboden anfängt. Die Verbindung zwischen den beiden Punkten wird dann durch den Shapeconverter gelöst. Dieser Sachverhalt ist in Abbildung 12.2 (Seite 65) dargestellt, zusammen mit der bisherigen Variante aus [1].

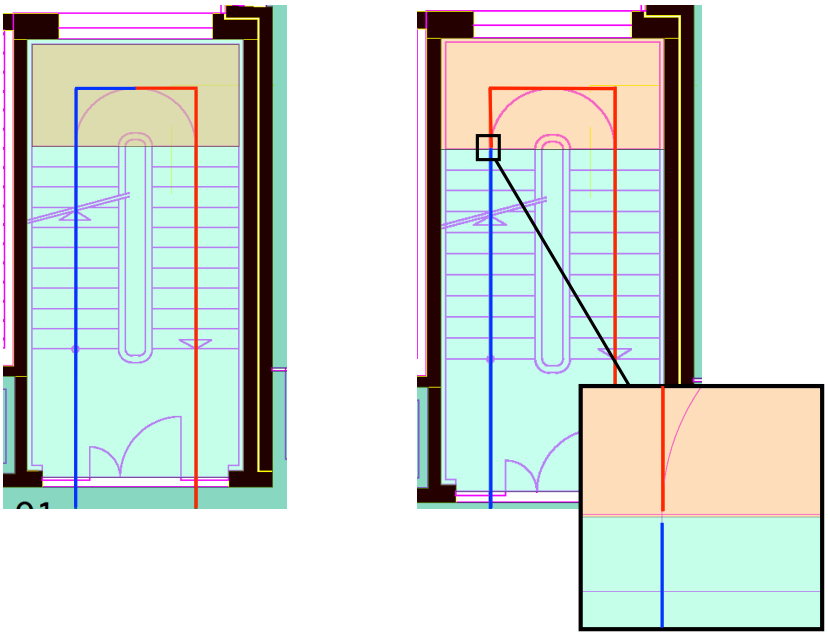

Abbildung 12.2: Realisierung des Graphen bei einem Stockwerkübergang mit Treppen mit einem Zwischenboden. Links die bisherige Variante aus [1], rechts die neue Variante mit nur einer Line auf dem Zwischenboden

12.2 Georeferenzierung und Metadatenerfassung

12.2.1 Georeferenzieren

Das Vorgehen zum Georeferenzieren entspricht den Vorgehen aus [1]. Das selbe gilt für das Vorgehen bezüglich Metadatenerfassung. Bei dieser unterschieden sich allerdings die Metadaten inhaltlich, da mit ihrer Hilfe die Stockwerkübergänge realisiert werden.

12.2.2 Aktualisierung

Um bei Änderungen einer DXF-Datei möglichst schnell wieder auf die georeferenzierten Shapefiles zu kommen, kann für jeden Stockwerkplan beim ersten Georeferenzieren ein World File (`.wld`) angelegt werden. Dieses merkt sich die Koordinaten in der ursprünglichen Position und der georeferenzierten Position. Muss ein DXF-Dokument neu georeferenziert werden, kann das zu dem Stockwerk passende World File geladen werden. So spart man sich das Eintippen der richtigen Positionen.

Voraussetzung, dass diese Technik funktioniert, ist, dass die in der DXF-Datei beschriebene Struktur an sich nicht verschoben wird. Das heisst, es können neue Sektoren gezeichnet oder der Graph angepasst werden, aber die Struktur an sich sollte bezüglich ihrer Position gleich bleiben.

Spaltenbezeichnung	Format	Beispiel
Gebaeude	Text	HOER
Stockwerk	Text	e
Abteilung	Text	IPS
Raumname	Text	27k
Hoehe	Text	470.5

Tabelle 12.1: Format der Polygon-Metadaten

Spaltenbezeichnung	Format	Beispiel
Treppe	<X>;<Y>;<Z>;<Zielhöhe>	684074.446;247924.628;470.5;468.5
Lift	<X>;<Y>;<Z>;<Zielhöhe>	684070.537;247922.753;470.5;473.9
Uebergang	<X>;<Y>;<Z>;<Zielhöhe>	684040.186;247881.665;470.5;470.5

Tabelle 12.2: Format der Polyline-Metadaten

12.2.3 Metadaten

Polygon-Shapefile

Die Metadaten für das Polygon-Shapefile bleiben gleich. Ihre Struktur ist in Tabelle 12.1 (Seite 66) beschrieben.

Polyline Shapefile

Die neue Struktur der Metadaten für das Polyline-Shapefile ist in Tabelle 12.2 (Seite 66) beschrieben.

Für die Beschreibung der Übergänge über eine Treppe oder einen Lift werden jeweils Koordinaten eines Punktes in der Nähe des zu verbindenden Linien-Endpunktes angegeben (<X><Y><Z>). Dabei muss die Höhe (<Z>) genau der Höhe des zu verbindenden Punktes entsprechen, während die anderen Koordinaten frei gewählt werden können, mit der Einschränkung, dass sich dieser im selben Sektor befindet und von anderen Punkten innerhalb des Sektors weiter entfernt sind als von dem Verbindungspunkt. Zusätzlich muss die Höhe des Stockwerkes angegeben werden, auf welchem sich der Punkt befindet mit dem der zu verbindende Linien-Endpunkt verbunden werden soll. Auch hier gilt, es wird immer von unten nach oben verbunden. Konkret heisst das zum Beispiel für eine Lift-Verbindung zwischen dem Stockwerk d (Höhe 467.1), dem Stockwerk e (Höhe 470.5) und den Stockwerk f (Höhe 473.9) folgendes:

Stockwerk d Im Stockwerk d wird zum einen die Höhe 467.1 angegeben für die Höhe, auf der sich der Startpunkt befindet, und die Höhe von Stockwerk e, also 470.5.

Stockwerk e Im Stockwerk e wird zum einen die Höhe 470.5 angegeben für die Höhe, auf der sich der Startpunkt befindet, und die Höhe von Stockwerk f, also 473.9.

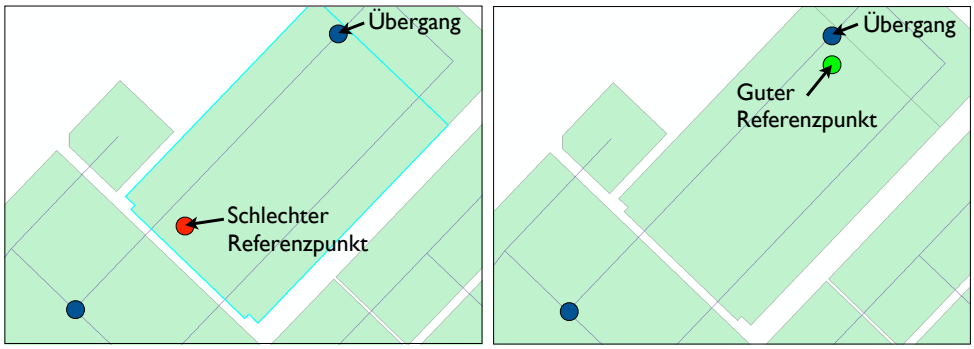

Abbildung 12.3: Beispiel für die Wahl eines Referenzpunktes für die Bestimmung des Übergangspunktes. Links: Schlechte Variante. Rechts: korrekte Variante.

Stockwerk f Im Stockwerk f wird zum einen die Höhe 473.9 angegeben für die Höhe, auf der sich der Startpunkt befindet. Da von dem Stockwerk aus keine Lift-Verbindung zu einem höheren Stockwerk vorhanden ist, wird als Zielhöhe dieselbe Höhe angegeben wie für den Startpunkt.

Die Koordinaten in der Nähe des Punktes werden benötigt, um festzustellen, welches Ende der Linie ein Übergang ist. Ist für eine Linie ein Übergang notiert, wird nachgeschaut, welcher der Endpunkte dieser Linie sich näher bei dem in der Attributtabelle angegebenen Punkt befindet (Abbildung 12.3, Seite 67).

Alle Punkte, die als Übergang gekennzeichnet wurden, werden pro Typ (also Treppe, Lift oder Stockwerkübergang) gesammelt, damit – nachdem alle Stockwerkpläne mit Sektoren und Graph eingelesen wurden – die Liste mit Start- und Zielpunkten durchgegangen werden kann. Für einen Ausgangspunkt wird jeder Punkt, der sich auf der angegebenen Zielhöhe befindet, angeschaut und geprüft, dass er sich in x- und y-Richtung möglichst nahe an dem Ausgangspunkt befindet.

12.3 Endergebnis

Am Ende sollen für jedes Stockwerk je zwei Shapefiles existieren, ein Polygon-Shapefile mit den Sektoren und ein Polyline-Shapefile mit den Gehwegen, also dem Graph.

Zu beachten ist, dass es sich bei einem Shapefile entgegen des Namens nicht um eine einzelne Datei handelt, sondern um ein «Paket» verschiedener Dateien. Entsprechend ist immer darauf zu achten, alle bei der Shapefile-Generierung entstandenen Dateien zusammen im gleiche Pfad abzulegen. Es empfiehlt sich daher, für jedes Stockwerk zwei separate Ordner für die Polygon-Shapefile-Dateien und für die Polyline-Shapefile-Dateien zu erstellen.

13 Domänenanalyse

13.1 Strukturdiagramm

Abbildung 13.1 (Seite 70) zeigt das Domain-Modell für die Geometriedigitalisierung.

13.2 Klassen

13.2.1 Campus

Der `Campus` repräsentiert das gesamte Gebiet in welchem ein Objekt lokalisiert werden kann.

13.2.2 Building

Ein `Building` repräsentiert ein Gebäude. Dieses besteht aus einem oder mehreren Stockwerken.

Attribute

name Bezeichnung des Gebäudes

13.2.3 Floor

Das `Floor`-Objekt repräsentiert ein Stockwerk. Auf einem Stockwerk kann sich eine beliebige Anzahl von Sektoren befinden.

Attribute

name Bezeichnung des Stockwerks

13.2.4 Sector

Ein `Sector` repräsentiert den kleinsten lokalisierbaren Bereich. Dies kann zum Beispiel ein einzelner Raum sein, aber auch ein Teil eines Raums oder eines Korridors. Es können auch mehrere Räume in Form eines einzelnen Sektors zusammengefasst werden. Ein `Sector` wird durch einen einzigen äusseren Ring abgegrenzt und kann durch keinen, einen oder mehrere innere Ringe («Löcher» innerhalb des `Sectors`) aufweisen.

Attribute

name Bezeichnung des Sektors

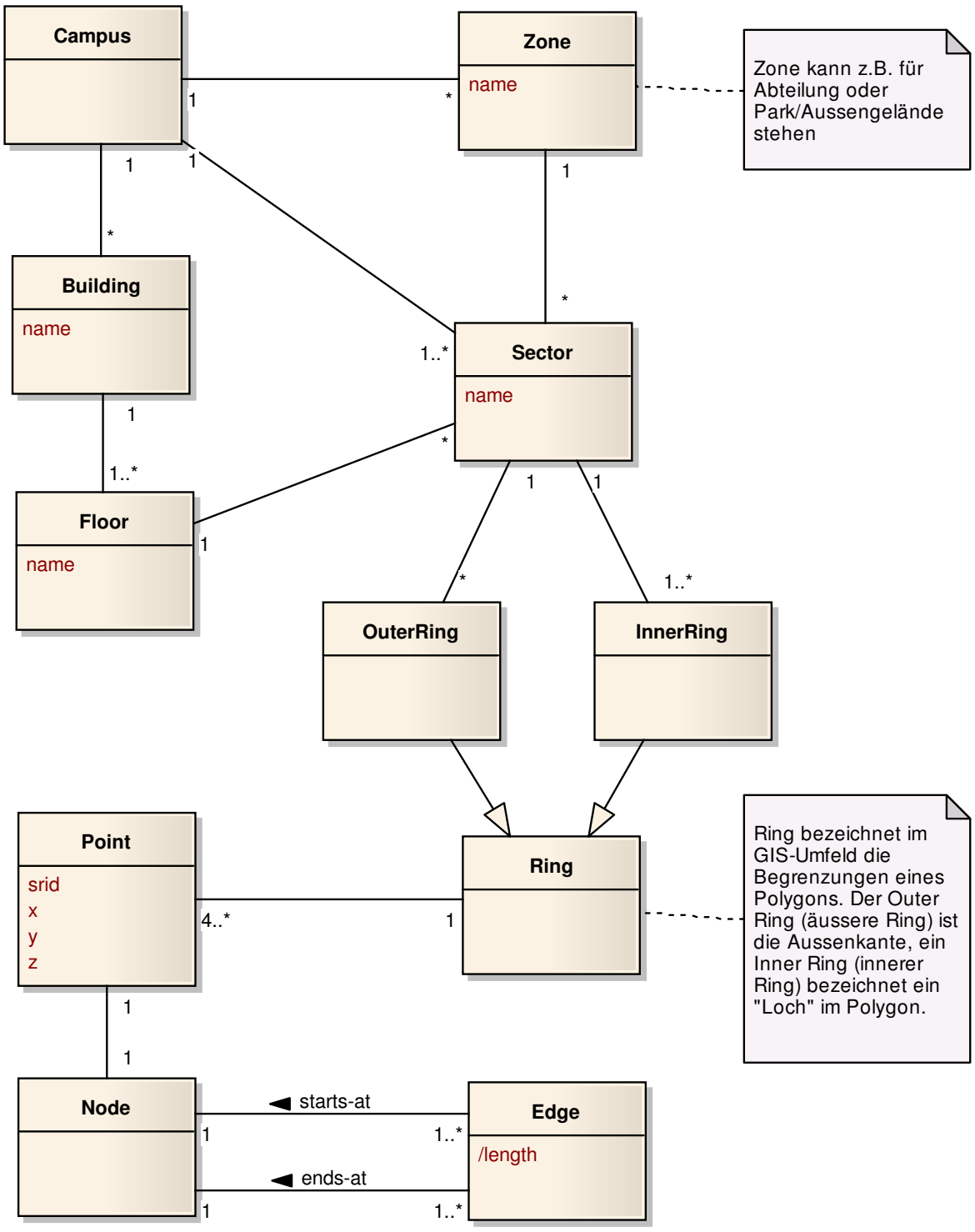

Abbildung 13.1: Domain-Modell Geometriedigitalisierung

13.2.5 Zone

Eine `Zone` beschreibt ein Gebiet, zum Beispiel eine Abteilung oder ein Aussengelände welches sich auf dem Campus befindet. Die `Zone` kann aus mehreren `Sectors` zusammengesetzt sein. Diese `Sectors` können sich in verschiedenen Gebäuden auf verschiedenen Stockwerken befinden.

Attribute

name Bezeichnung der Zone

13.2.6 Ring

Ein `Ring`-Objekt beschreibt im Geoinformationssystem (GIS)-Umfeld die Begrenzungen eines Polygons. Ein `Ring` wird durch mindestens vier `Point`-Objekte abgegrenzt (Startpunkt kommt als Endpunkt ein zweites Mal vor, um den Ring zu schliessen).

13.2.7 OuterRing

Das `OuterRing`-Objekt beschreibt einen Ring, welcher die Aussenkannte des Polygons repräsentiert. Für einen `Sector` kann nur ein `OuterRing` definiert werden.

13.2.8 InnerRing

Das `InnerRing`-Objekt beschreibt einen Ring, welcher innerhalb eines Polygons einzelne Bereiche ausschliesst, also einem «Loch» im Polygon entspricht.

13.2.9 Point

Ein `Point` definiert eine eindeutige Position im dreidimensionalen Raum des verwendeten Bezugssystems.

Attribute

srid Spatial Reference System Identifier

x Erster Koordinatenpunkt

y Zweiter Koordinatenpunkt

z Dritter Koordinatenpunkt

13.2.10 Node

Ein `Node`-Objekt symbolisiert einen Wegpunkt im Topologie-Graphen. Auf einem `Point` kann genau ein `Node` liegen.

13.2.11 Edge

Eine `Edge` verbindet genau zwei `Nodes` (Startpunkt und Endpunkt) miteinander und stellt somit den möglichen Weg zwischen diesen zwei Wegpunkten dar.

Attribute

length Distanz zwischen den zwei `Node`-Objekten, die durch die `Edge` verbunden sind.

13.3 Konzepte

13.3.1 Geometrie-Hierarchie

Die Geometrie-Hierarchie wurde im Vergleich zu [1] (S. 57 ff.) um die Komponente `Campus` erweitert, das als Wurzel der Topologie dient. Auf einem `Campus` befinden sich verschiedene Gebäude, Zonen und Sektoren. Die Gebäudehierarchie besteht wie in [1] (S. 57 ff.) aus Gebäude (engl. Building), Stockwerk (engl. Floor) und der kleinsten lokalisierbaren Einheit, dem Sektor (engl. Sector). Diese Hierarchie ist auf Abbildung 13.2 (Seite 73) rot eingezeichnet. Da sich aber der für die Lokalisierung abzudeckende Bereich nicht nur über Gebäude zieht, sondern zusätzlich aus Aussengelände bestehen kann, wird die Hierarchie durch den Campus und die Zone ergänzt. Die Zone nimmt einerseits die Rolle der in [1] (S. 57 ff.) beschriebenen Abteilungsbegrenzung. Hinzu kommt, dass die Zone auch einen Aussenbereich beschreiben kann. Um die Lösung auch im kleineren Rahmen einsetzen zu können, kann auf Gebäude und Zonen verzichtet werden. Die Minimale Lösung enthält einen Campus der aus einem Sektor besteht. Dies ergibt sich durch die Multiplizität `1..*` zwischen Campus und Sektor. Dieser Sachverhalt ist in Abbildung 13.2 (Seite 73) blau eingezeichnet.

13.3.2 Geometrie

Für die Umsetzung der Geometrie wird dasselbe Konzept wie in [1] (S. 60 ff.) verwendet. Damit die Domäne dem GIS-Umfeld entspricht und um eine möglichst freie Gestaltung der Sektoren zu ermöglichen, wird zusätzlich noch das Objekt `Ring` eingeführt. Ein Sektor wird durch einen äusseren Ring (`OuterRing`) abgegrenzt. Zusätzlich besteht durch die Option, innere Ringe (`InnerRing`) zu definieren, die Möglichkeit, Sektoren mit «Löchern» zu definieren (Abbildung 13.3, Seite 73). Abbildung 13.4 (Seite 74) zeigt diesen Zusammenhang auf.

13.3.3 Topologie

Die Grundidee bezüglich der Topologie bleibt im Verglich zu [1] gleich. Um die Beziehung zwischen `Node` und `Edge` gemäss den im GIS-Umfeld verwendeten Zusammenhängen sauber auszudrücken, wird ein `Edge` durch einen Start-`Node` und einen End-`Node` bestimmt. An einem `Node` können mehrere `Edges` starten, beziehungsweise enden. Dies ist durch die Multiplizität `1..*` zwischen `Node` und `Edge` gewährleistet. Dieser Zusammenhang ist in Abbildung 13.5 (Seite 75) grün dargestellt.

Ein `Node` befindet sich an einem Punkt (`Point`), und erhält so die Position im Raum. Diese Verbindung ist in Abbildung 13.5 (Seite 75) orange dargestellt.

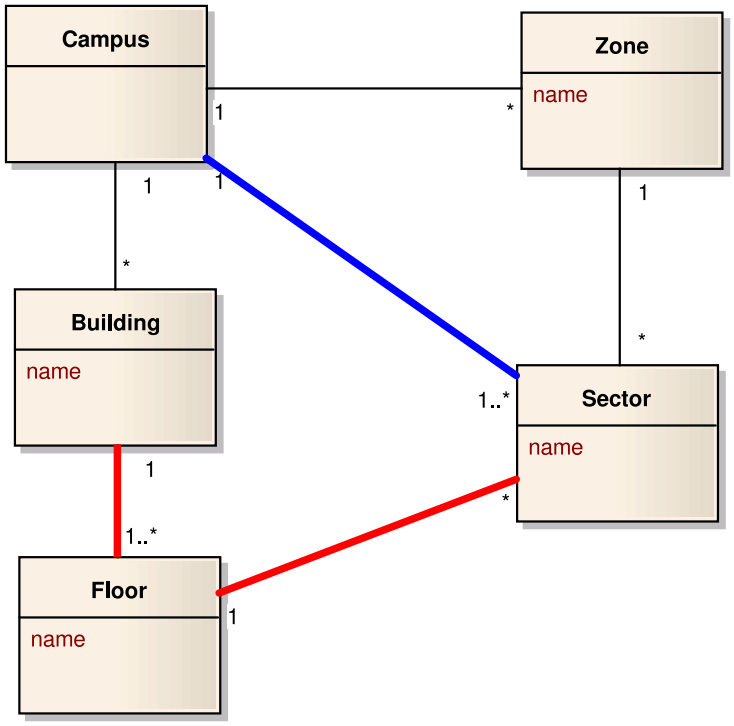

Abbildung 13.2: Abbildung der Gebäudehierarchie im Domain-Modell

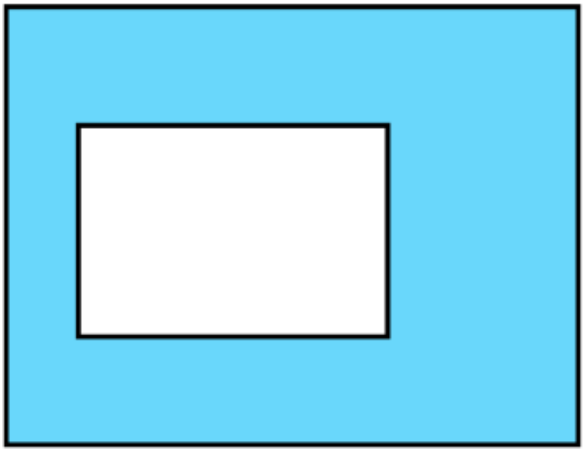

Abbildung 13.3: Der dargestellte Sektor besteht aus dem blau eingefärbten Bereich. Durch den inneren Ring gehört der innere weisse Bereich nicht zum Sektor.

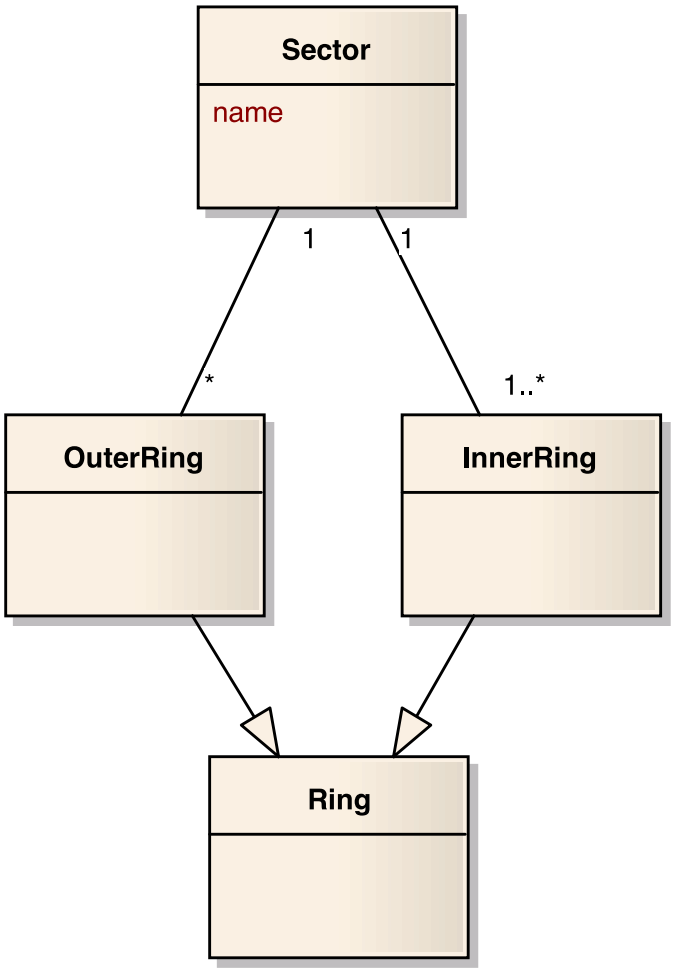

Abbildung 13.4: Abbildung eines Sektors im Domain-Modell

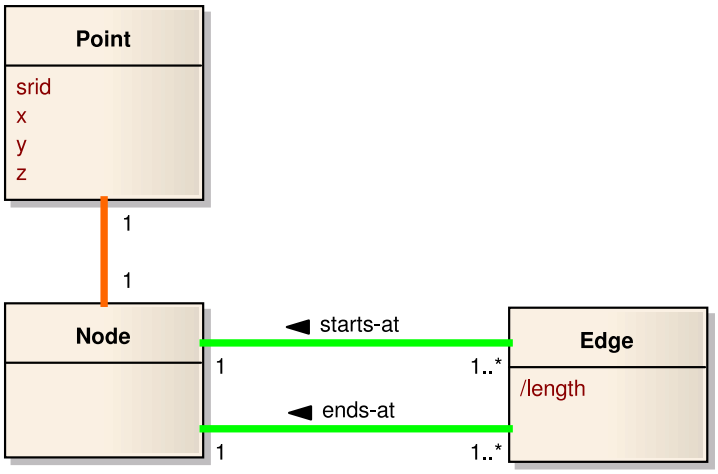

Abbildung 13.5: Abbildung Topologie im Domain-Modell

13.3.4 Verbindung von Geometrie und Topologie

Für die Berechnung der kürzesten Laufwege innerhalb des Campus wird wie in [1] ein Ausgangs- und ein Zielpunkt benötigt. Dabei handelt es sich um einen Sektor, der durch einen äusseren und optionale innere Ringe begrenzt ist. Die Topologie, welche die möglichen Gehwege darstellt, besitzt als Positionspunkt einen `Node`. Das Verbindungsglied zwischen Geometrie und Topologie wird durch die Klasse `Point` realisiert (in Abbildung 13.6, Seite 76, lila dargestellt). Für jeden beliebigen Punkt kann festgestellt werden, in welchem Ring sich dieser befindet (in Abbildung 13.6, Seite 76, blau dargestellt). Da der `Node` durch einen Punkt dargestellt wird, kann so die Verbindung zwischen einem `Node` und einem Sektor geschaffen werden (in Abbildung 13.6, Seite 76, rot dargestellt).

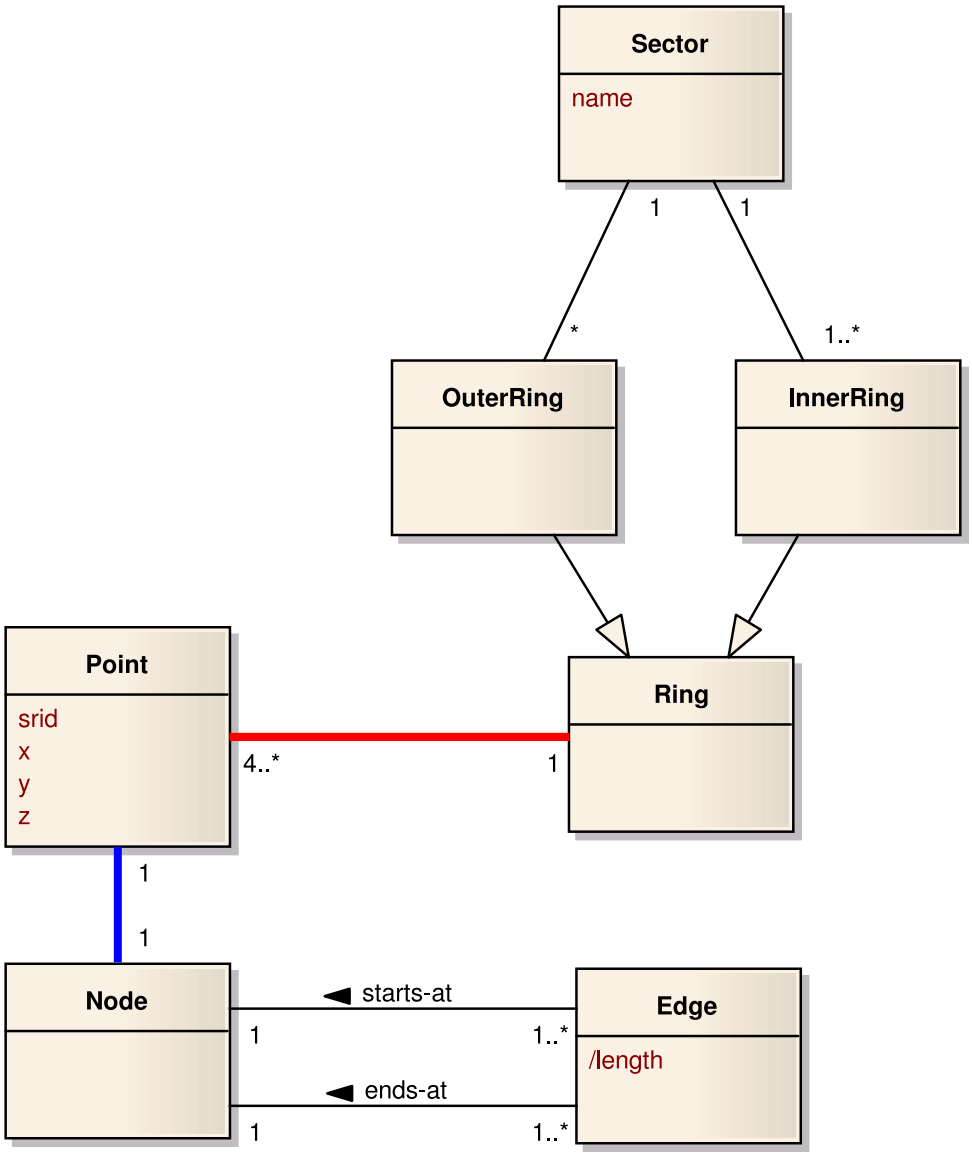

Abbildung 13.6: Abbildung der Verbindung zwischen Geometrie und Topologie im Domain-Modell

14 Software-Architektur

14.1 Architektur

14.1.1 Architekturübersicht

Der Shapeconverter ist eine Softwarekomponente zum Generieren von Geometrie und Topologie-Daten. Ihre Aufgabe besteht darin, bestehende Stockwerkpläne in Form von Shapefiles mit Attributtabellen einzulesen und daraus SQL-Dateien zu erzeugen, welche die Geometrie und Topologie des ganzen Campus widerspiegeln. Ziel ist, daraus die Geo-Datenbank für die Lokalisierung zu erzeugen.

Abbildung 14.1 (Seite 78) zeigt den Ablauf der Geometriedigitalisierung und wie sich der Shapeconverter darin einfügt. Die Stockwerkpläne werden in Form von DXF-Dokumenten im Vektorworks (oder einer anderen CAD-Software) aufbereitet und dann mit ArcGIS georeferenziert. Der als Konsolen-Applikation konzipierte Shapeconverter bereitet die Daten dann so auf, dass sie für den Import in die PostGIS-Datenbank verwendet werden können.

14.1.2 Physische Sicht der Systemarchitektur

Deployment Diagram

Abbildung 14.2 (Seite 79) zeigt das Deployment Diagramm des als Single-Tier-Applikation entwickelten Shapeconverters.

14.1.3 Logische Sicht der Systemarchitektur

Schichtenarchitektur

Der Shapeconverter verwendet eine gelockerte Schichtenarchitektur (siehe Abbildung 14.3, Seite 80). Das heisst, Packages dürfen nicht nur auf die direkt darunterliegenden Packages zugreifen, sondern auch auf Ebenen weiter unten. Dabei gilt, je weniger Schichten übersprungen werden, desto besser. Zugriffe in darüber liegende Schichten sind jedoch nicht erlaubt.

Nachfolgend sind die Verantwortlichkeiten der Schichten von oben nach unten beschrieben:

Service Layer Der Service Layer integriert gemäss Controller-Pattern die Use Case Controller. Zusätzlich beinhaltet er die zur Realisierung der Use Cases benötigten Zusatzklassen.

Domain Layer Der Domain Layer enthält die Domain-Objekte der Geometrie und Topologie.

Foundation Layer Der Foundation Layer enthält Utility-Klassen zur Interaktion mit externen Bibliotheken.

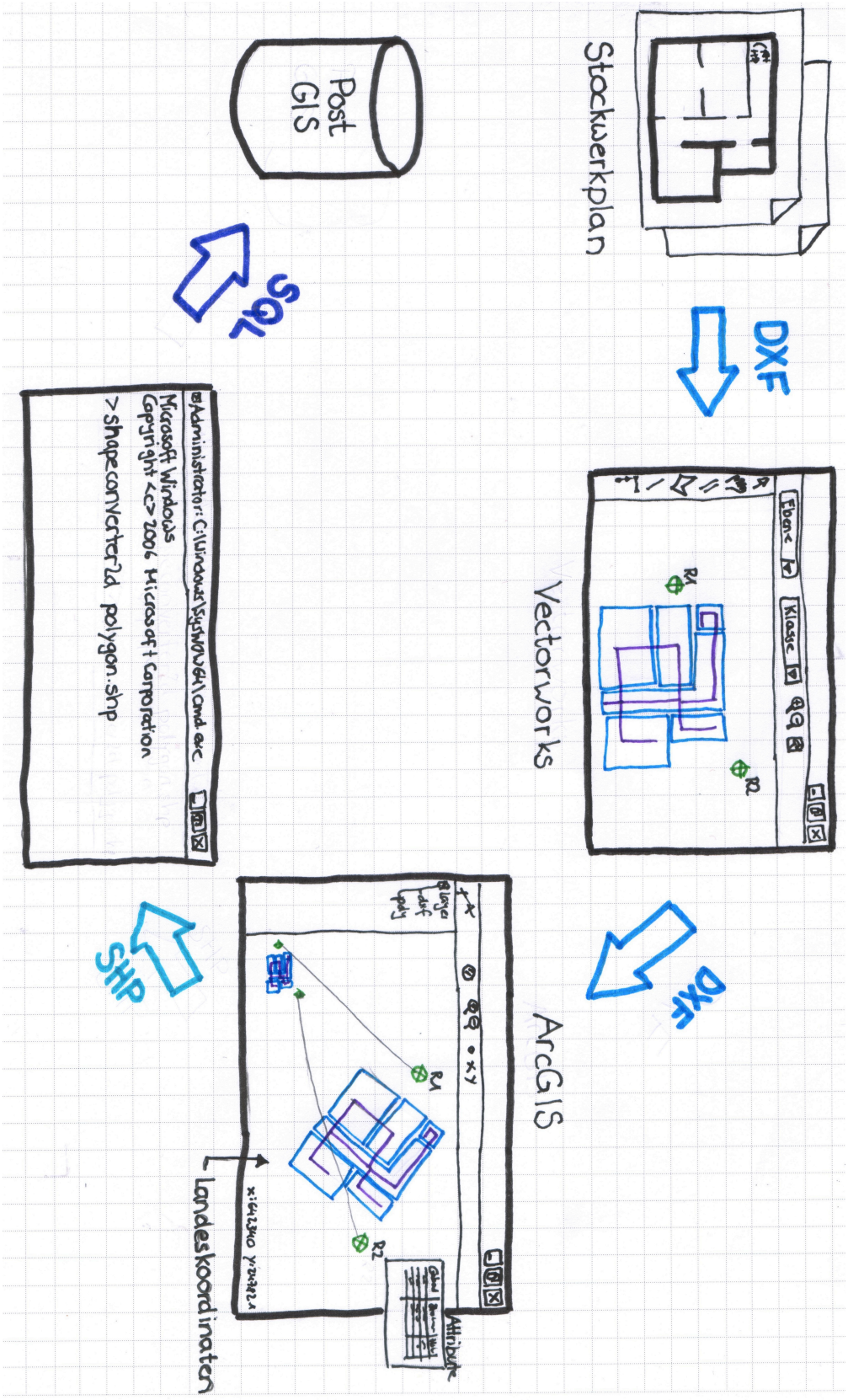

Abbildung 14.1: Architekturübersicht

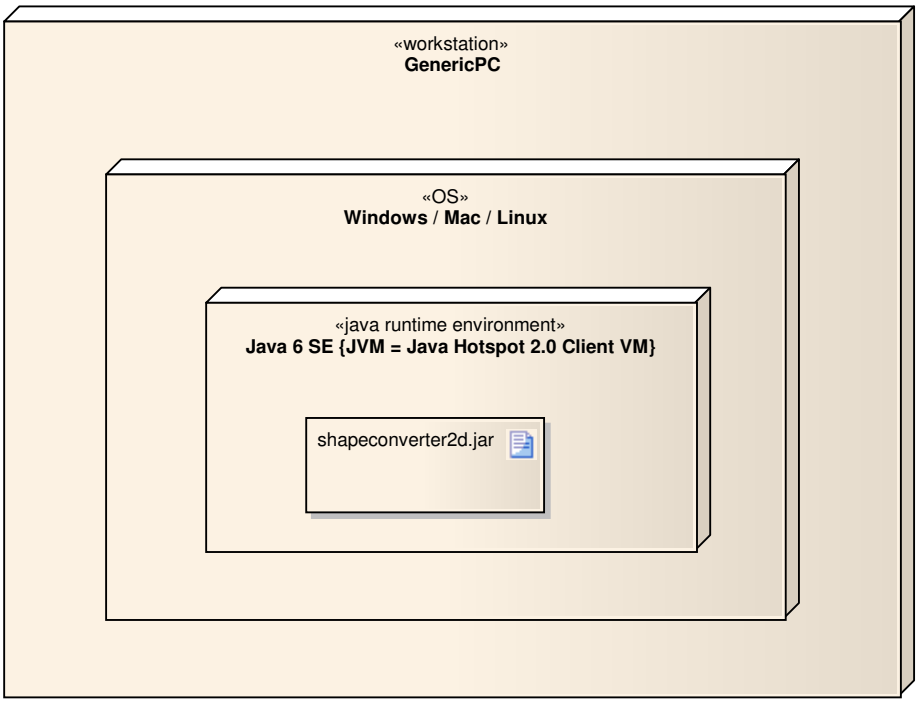

Abbildung 14.2: Deployment Diagram

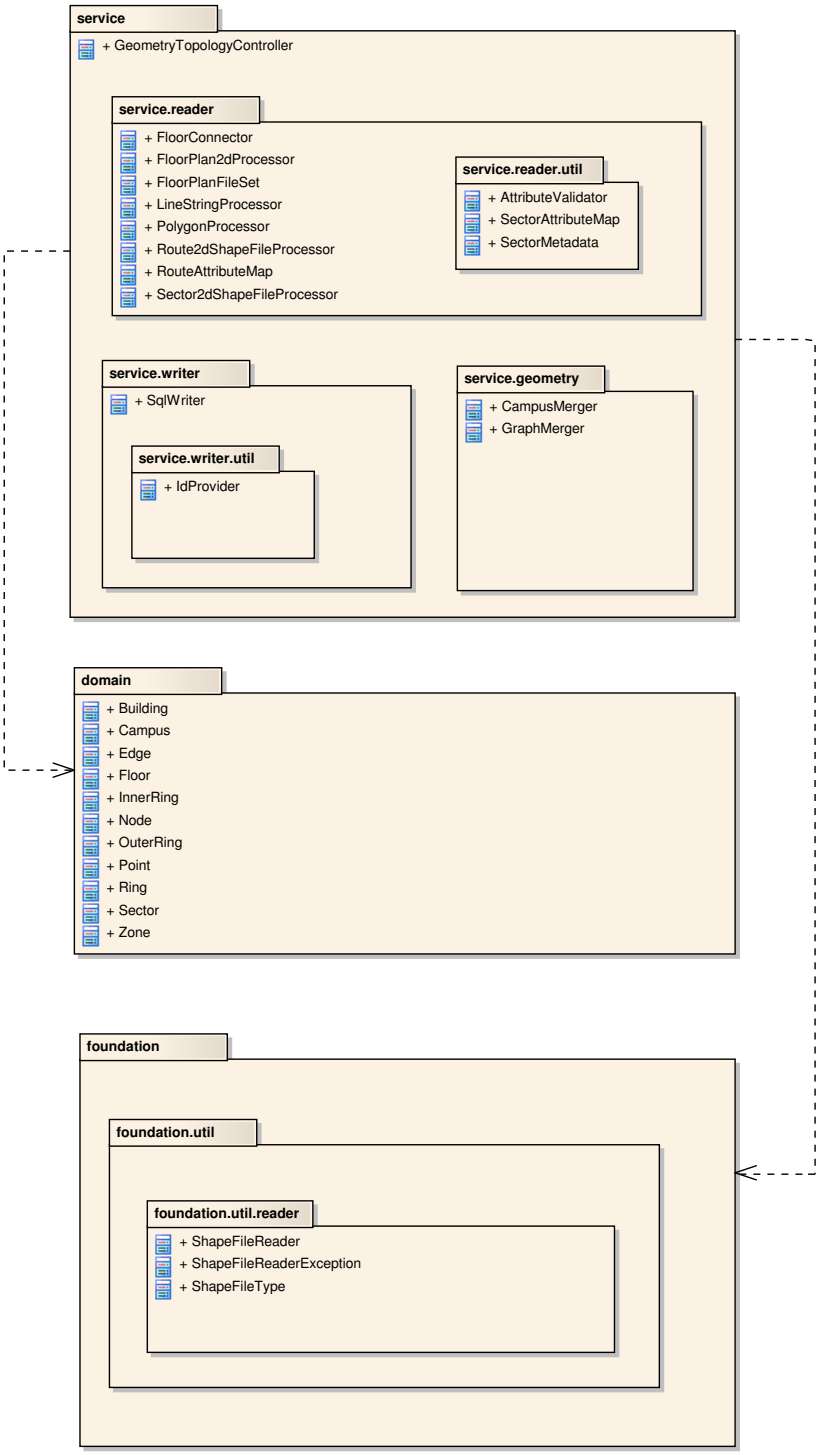

Abbildung 14.3: Das Logical Architecture Diagram zeigt die Schichtenarchitektur des Shapeconverters

Übersicht über Packages

Nachfolgend sind die Packages mit ihren Zuständigkeiten aufgelistet. Eine detaillierte Beschreibung ist in Abschnitt 14.2 (Seite 82 zu finden). Der Präfix `ch.hsr.ins` ist aus Gründen der Lesbarkeit weggelassen.

shapeconverter2d.service Enthält die Klassen, welche mittels Controller Pattern die Ausführung der Use Cases kontrollieren.

shapeconverter2d.service.reader Enthält die Service-Klassen, um aus den eingelesenen Shapefiles die Domain aufzubauen.

shapeconverter2d.service.reader.util Enthält Utility-Klassen, welche zum Einlesen der Shapefiles benötigt werden.

shapeconverter2d.service.geometry Enthält die Service-Klassen, welche die Domain den einzeln eingelesenen Komponenten zusammenbaut.

shapeconverter2d.service.writer Beinhaltet die Service-Klassen zur Generierung der Datenbank-Files.

shapeconverter2d.service.writer.util Enthält die Utility-Klassen, welche zum Schreiben der SQL-Dateien benötigt werden.

shapeconverter2d.domain Umfasst die Domänen-Objekte.

shapeconverter2d.foundation.util.reader Enthält Utility-Klassen zum Zugriff auf externe Bibliotheken im Zusammenhang mit dem Lesen der Shapefiles.

Liste der verwendeten Programmbibliotheken

Google Guava Google-Ergänzungen zu `java.util`; genutzt werden Hilfsmethoden zum Überschreiben von `hashCode()` und `equals()`.

geotools Java GIS Toolkit, genutzt werden Klassen zum Einlesen der Shapefiles.

JGraphT Java Graph Library; wird verwendet, um den gewichteten Graph zu erstellen.

SLF4J Logging-Façade für diverse Logging-Frameworks

JTS Topology Suite Bibliothek zur Durchführung von geometrischen Operationen und Verarbeitung von GIS-Daten

Liste der verwendeten Programmbibliotheken für Testing

JUnit Unit-Testing-Framework zur Erstellung von Unit Tests

Mockito Mocking-Framework, das die Erzeugung von Fakes, Mocks und Stubs vereinfacht

Hamcrest Stellt Matchers zur Verfügung, die fürs Testen verwendet werden können.

14.1.4 Architekturkonzepte

In den nächsten Abschnitten sind die allgememinen, nennenswerten Architekturkonzepte beschreiben, die für die Architektur des Shapeconverters relevant sind. Layerspezifische Architekturkonzepte sind in Abschnitt 14.2 (Seite 82) beschreiben.

Testing

Im Rahmen des Prototyps wurden Microtests mit JUnit erstellt.

14.2 Packages

14.2.1 shapeconverter.2d.service

Klassenstruktur

Abbildung 14.4 (Seite 83) zeigt die Klassenstruktur des Service Layer

Architekturkonzepte

Der Service Layer ist der Eintrittspunkt für Systemereignisse und behandelt diese. Implementiert wurde dieser mittels Controller Pattern. Jeweils ein Controller übernimmt die Ablaufkontrolle einer User Story. Die Benennung erfolgt gemäss der Bezeichnung der User Story. Da nur eine User Story realisiert wurde existiert nur der Controller `GeometryTopologyController` für die Abwicklung des Use Cases «Geometrie und Topologie für Datenbank-Import vorbereiten».

Hilfsklassen, welche die Umsetzung der Use Cases unterstützen, sind ebenfalls im Service Layer zu finden. Dabei handelt es sich um Klassen wie `FloorPlan2dProcessor`, welche die Verarbeitung der Stockwerkpläne übernimmt und entsprechende Aufgaben weiter delegiert.

14.2.2 shapeconverter2d.domain

Klassenstrktur

Die Klassenstruktur des Domain-Layers ist in Abbildung 14.5 (Seite 84) dargestellt.

Architekturkonzepte

Der Domain-Layer enthält die Domain-Objekte. Er separiert die Business Logik von den anderen Packages.

14.2.3 shapeconverter2d.foundation

Klassenstruktur

Abbildung 14.6 (Seite 85) zeigt die Klassenstruktur des Foundation Layers.

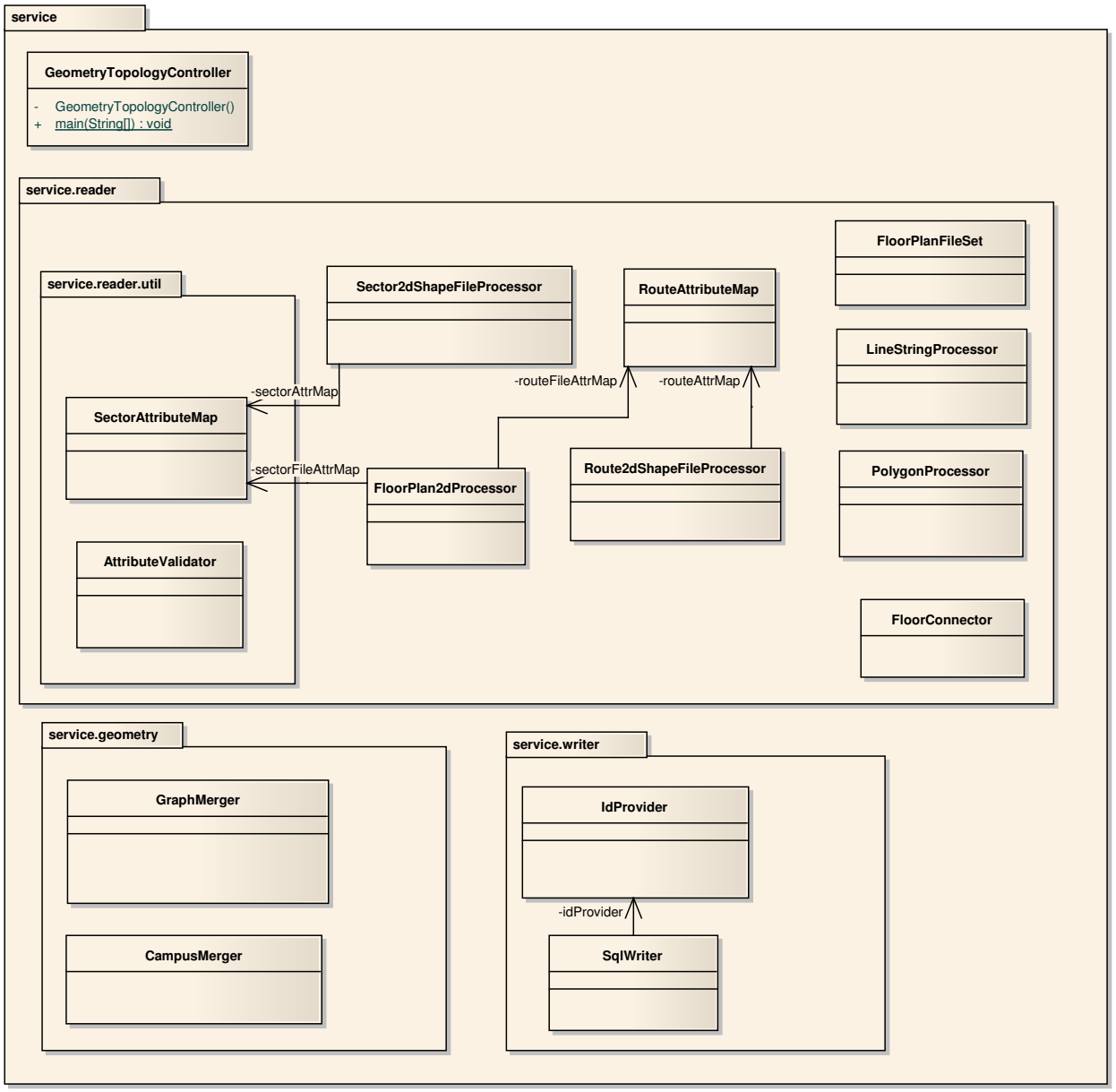

Abbildung 14.4: Klassenstruktur Service Layer

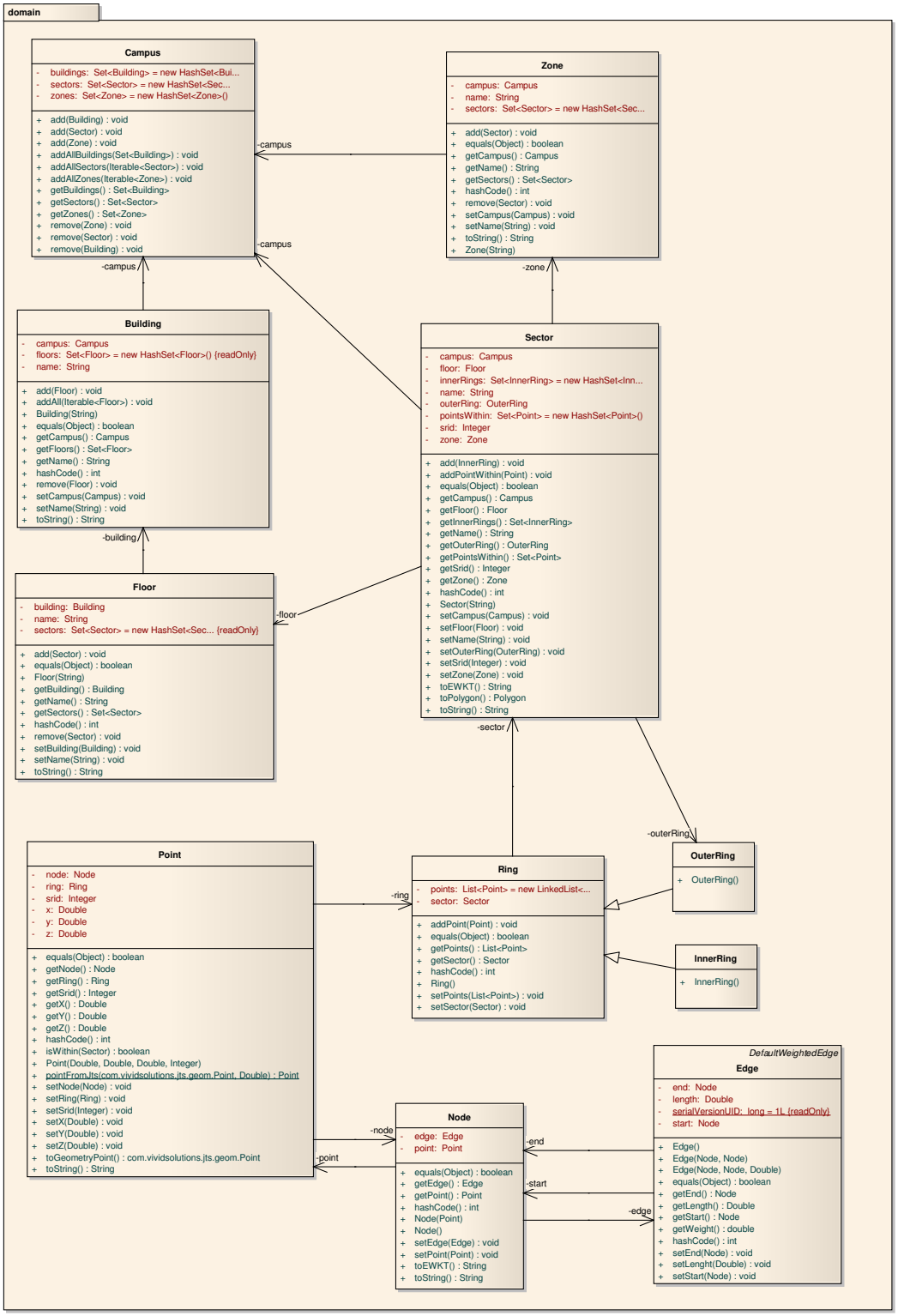

Abbildung 14.5: Klassenstruktur Domain Layer

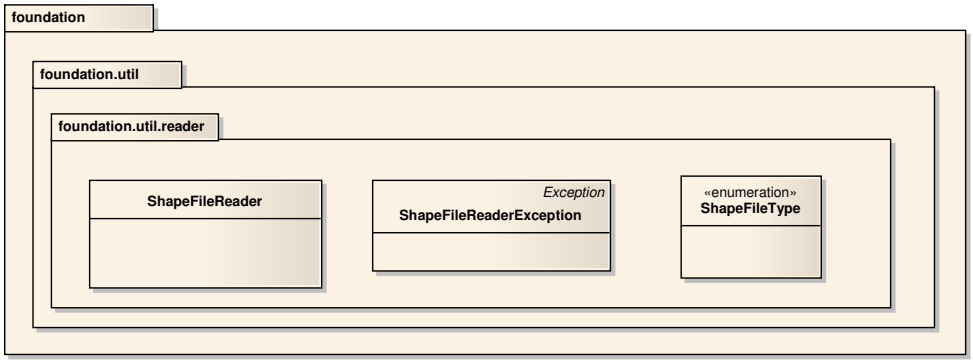

Abbildung 14.6: Klassenstruktur Foundation Layer

Architekturkonzepte

Das Package `shapeconverter.foundation` enthält Schnittstellen und Hilfsklassen zur Interaktion mit externen Komponenten.

14.3 Real Use Cases

Die Real Use Cases zeigen wie das System auf Ereignisse reagiert und wie dabei die Objekte über die verschiedenen Schichten hinweg miteinander interagieren. Der umgesetze Use Case «Geometrie und Topologie für Datenbank-Import vorbereiten» wird in Form eines White-Box-Sequenzdiagramm dargestellt. Aus Gründen der Übersichtlichkeit wurde dieser nicht als gesamter Use Case dargestellt, sondern in mehrere Abschnitte unterteilt:

Stockwerkpläne aus Filesystem einlesen Abbildung 14.7 (Seite 87) beschreibt den Ablauf zum Einlesen eines Stockwerkplans, welcher aus einem Polygon-Shapefile und einem Polyline-Shapefile besteht. Das `FloorPlanFileSet` bestehend aus diesen zwei Files wird dann in eine Liste abgelegt.

Geometrie aus Stockwerkplan erzeugen Für jedes `FloorPlanFileSet` wird nach dem Einlesen über den `FloorPlan2dProcessor` die Geometrie erstellt. Dieser Ablauf ist in Abbildung 14.8 (Seite 88) dargestellt.

Zusätzlich ist die spezifische Umwandlung in Gebäude (`handleBuilding(buildingAttribute)`, Abbildung 14.9, Seite 89), Stockwerk (`handleFloor(building, floorAttribute)`, Abbildung 14.10, Seite 90), Zone (`handleZone(zoneAttribute)`, Abbildung 14.11, Seite 91) und Sektor (`handleSector(floor, zone, sectorAttribute, heightAttribute)`, Abbildung 14.12, Seite 92) in separaten Abbildungen dargestellt.

Topologie aus Stockwerkplänen erzeugen Für jedes `FloorPlanFileSet` wird ebenfalls über den `FloorPlan2dProcessor` die Topologie erstellt. Der entsprechende Ablauf ist in Abbildung 14.13 (Seite 93) dargestellt.

Zusätzlich ist die spezifische Umwandlung in den Graphen (`turn2dGeometryIntoGraph(geometry, campus, stairAttribute, elevatorAttribute)`) in einer separaten Diagramm (Abbildung 14.14, Seite 94) dargestellt.

Das Ermitteln der Verbindungspunkte über Treppen zeigt ein separates Diagramm. Abbildung 14.15 (Seite 95) illustriert den Aufruf `getStaircasePoint(vertices, stairAttribute)`. Da dies vom Prinzip dem Aufruf für `getElevatorPoint(vertices, elevatorAttribute)` entspricht, wurde für diesen Fall kein eigenes Diagramm erstellt.

Stockwerke innerhalb eines Gebäudes in ihren Übergängen miteinander verbinden Um den Ablauf des Verbindens der Stockwerke zu zeigen, wurde der Ablauf wiederum aufgrund der Übersichtlichkeit auf mehrere Diagramme aufgeteilt. Abbildung 14.16 (Seite 96) zeigt den Ablauf in seinen Grundzügen. Das Verbinden über den `FloorConnector` ist in Abbildung 14.17 (Seite 97) dargestellt.

Distanzen zwischen allen Positionen berechnen In Abbildung 14.18 (Seite 97) ist der Ablauf der Berechnung der Distanzen mittels Floyd-Warshall-Shortest-Path-Algorithmus dargestellt.

Aus der Geometrie und Topologie SQL-Files erzeugen und im Filesystem ablegen Zur Beschreibung dieses Ablaufes ist in Abbildung 14.19 (Seite 98) als Beispiel der Ablauf zum Schreiben des Gebäudes (`writeBuilding()`) dargestellt.

14.4 User Interface

Im Rahen des Prototyps wird kein User Interface erstellt, da es für den Moment nur wichtig ist, die Funktionalität verwenden zu können. Bei der Applikation handelt es sich deshalb aktuell noch um eine einfache Konsolen-Applikation.

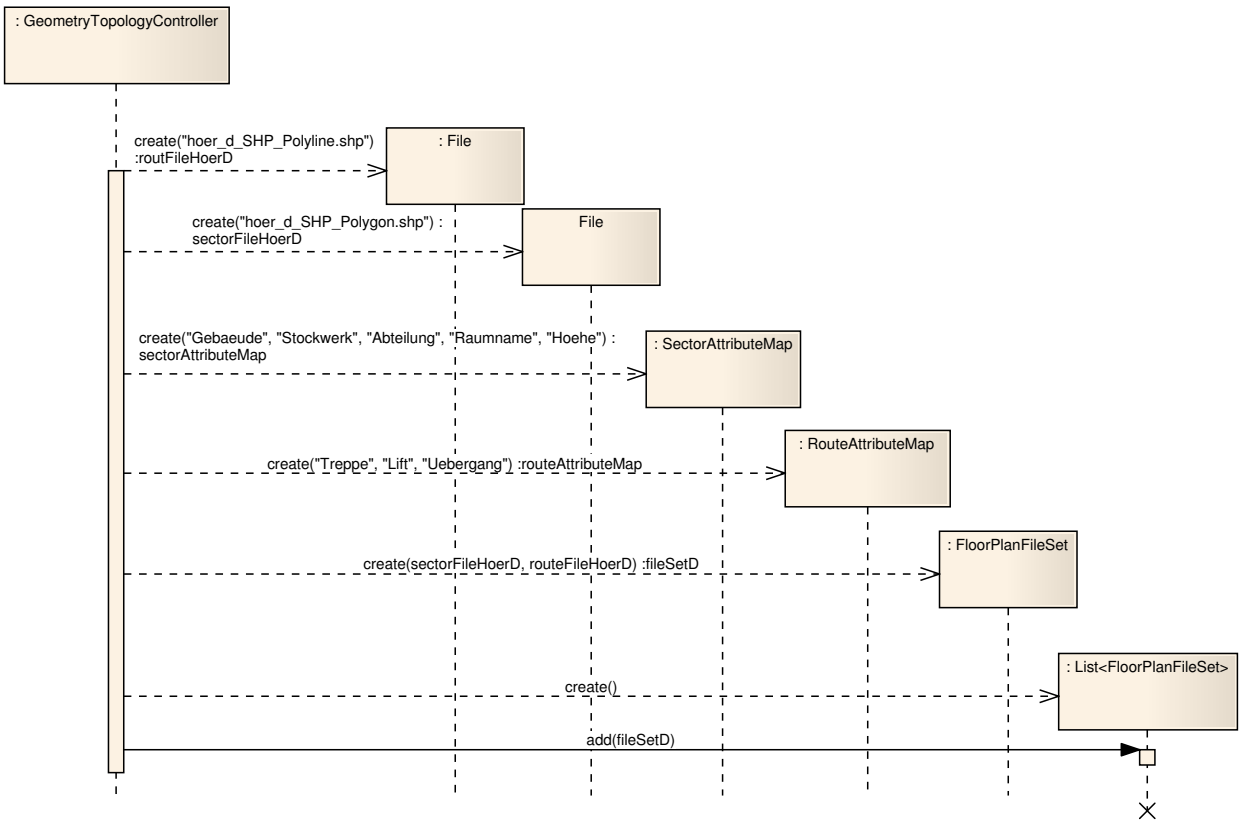

Abbildung 14.7: White-Box-Ansicht vom Einlesen eines Stockwerkplans, was ein Teil der User Story «Geometrie und Topologie für Datenbank-Import vorbereiten» ist.

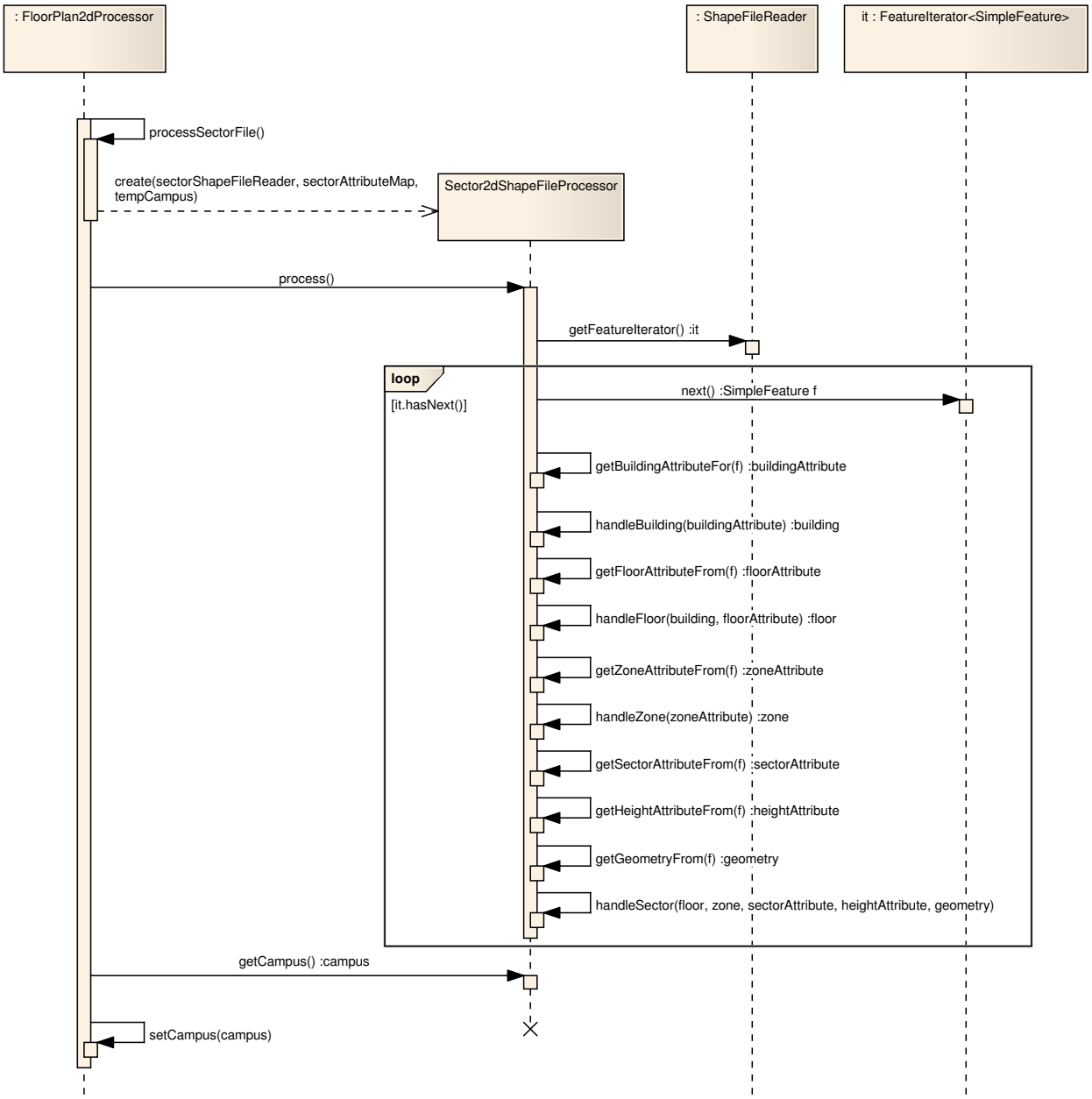

Abbildung 14.8: White-Box-Ansicht vom Erzeugen der Geometrie, was ein Teil der User Story «Geometrie und Topologie für Datenbank-Import vorbereiten»

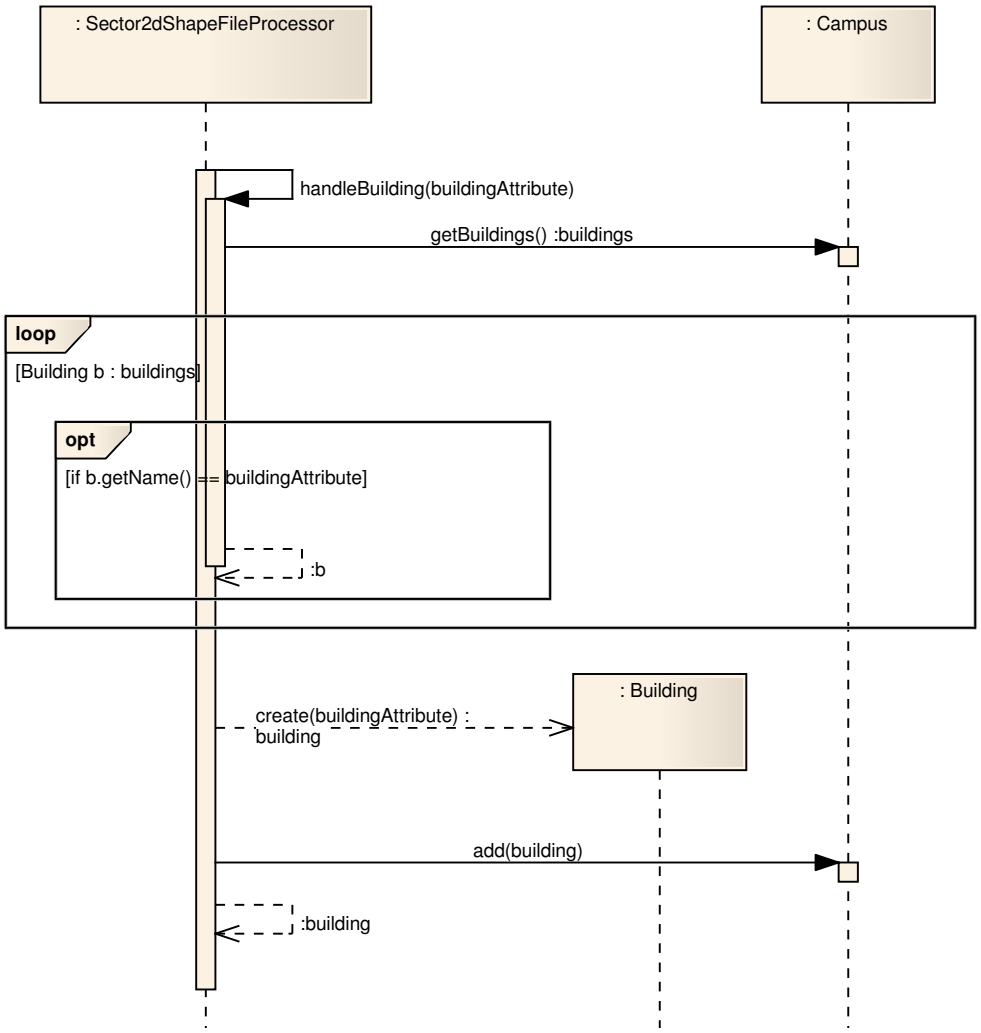

Abbildung 14.9: White-Box-Ansicht des Aufrufs `handleBuilding()`.

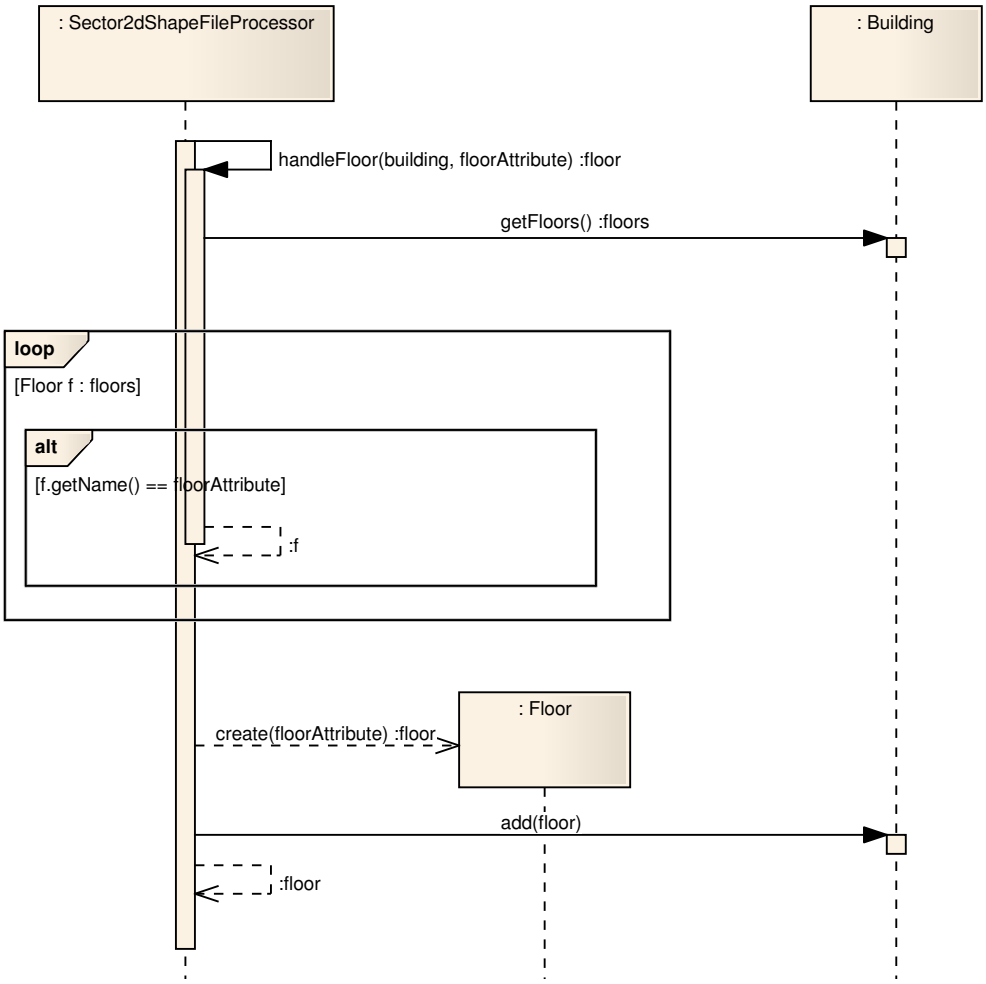

Abbildung 14.10: White-Box-Ansicht des Aufrufs `handleFloor()`.

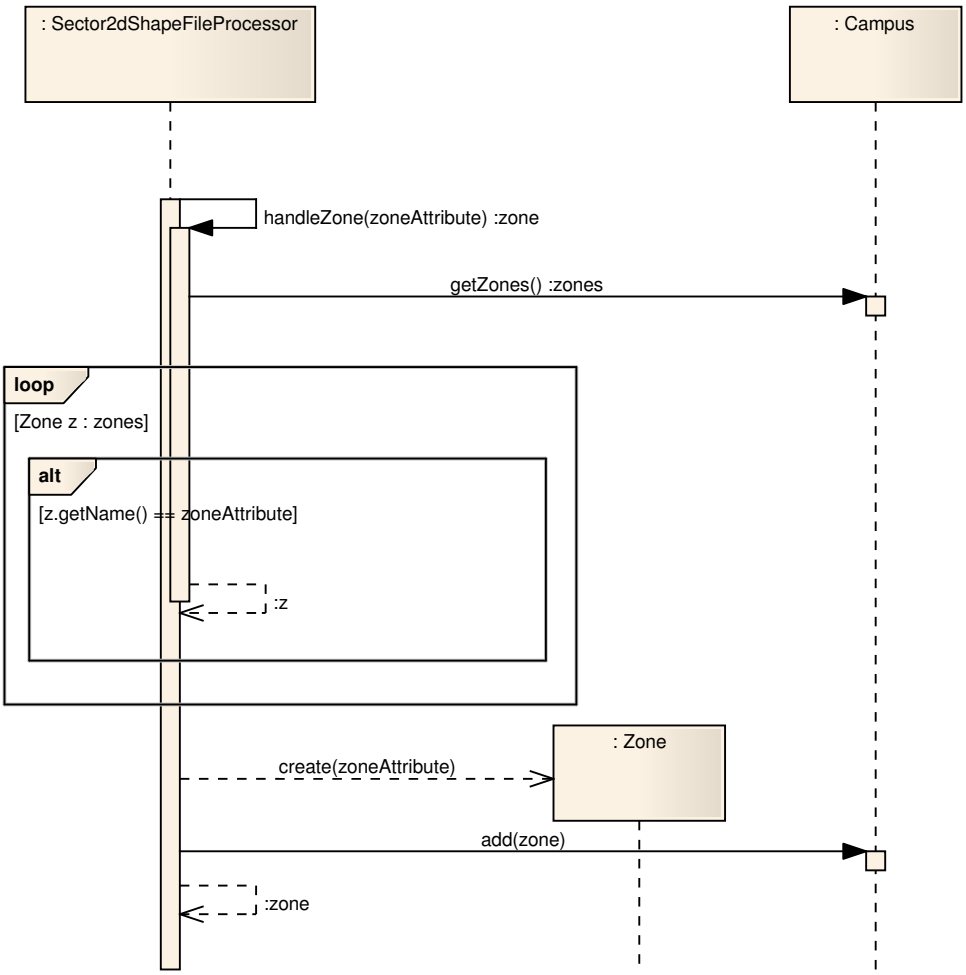

Abbildung 14.11: White-Box-Ansicht des Aufrufs `handleZone()`.

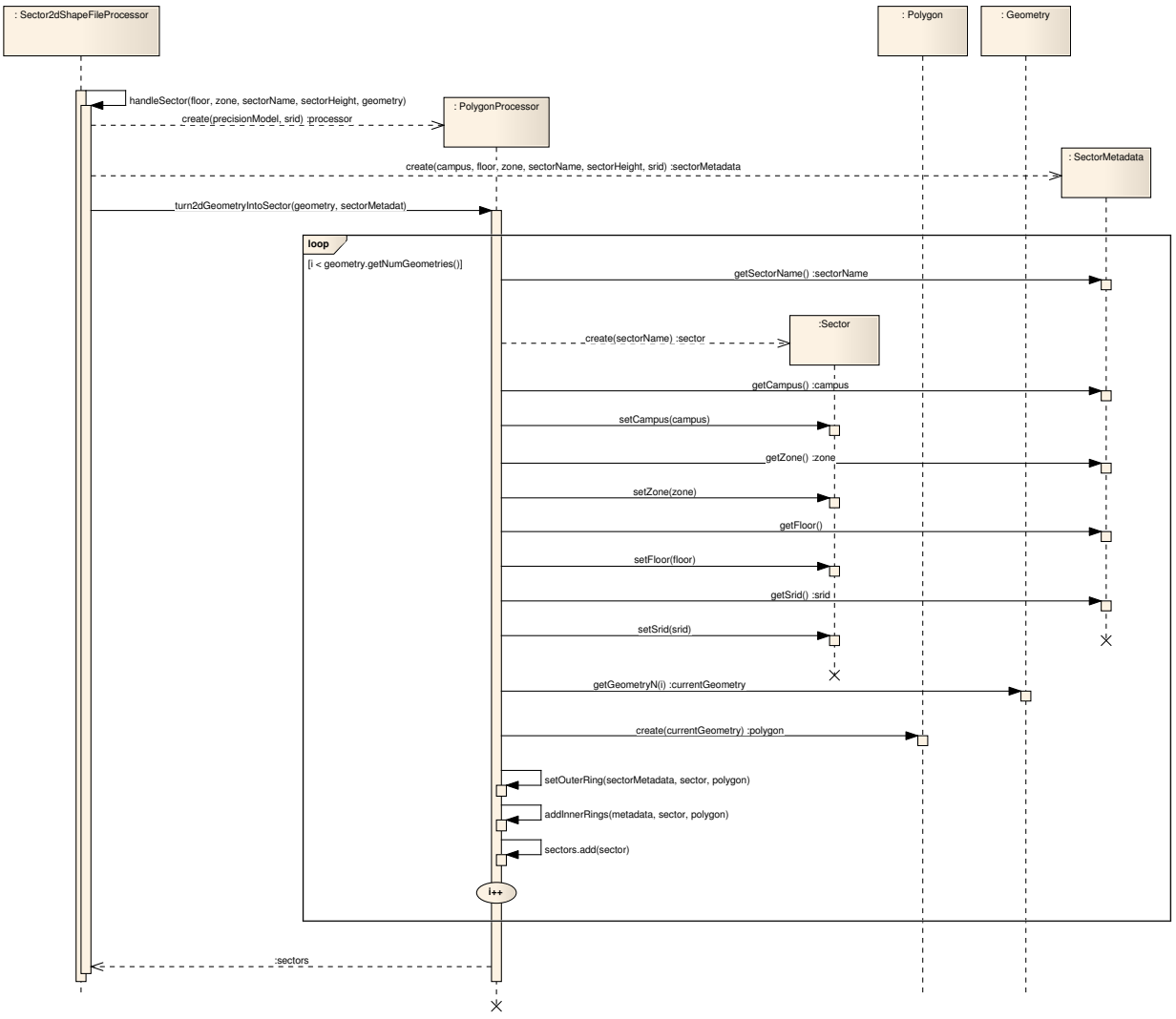

Abbildung 14.12: White-Box-Ansicht des Aufrufs `handleSector()`.

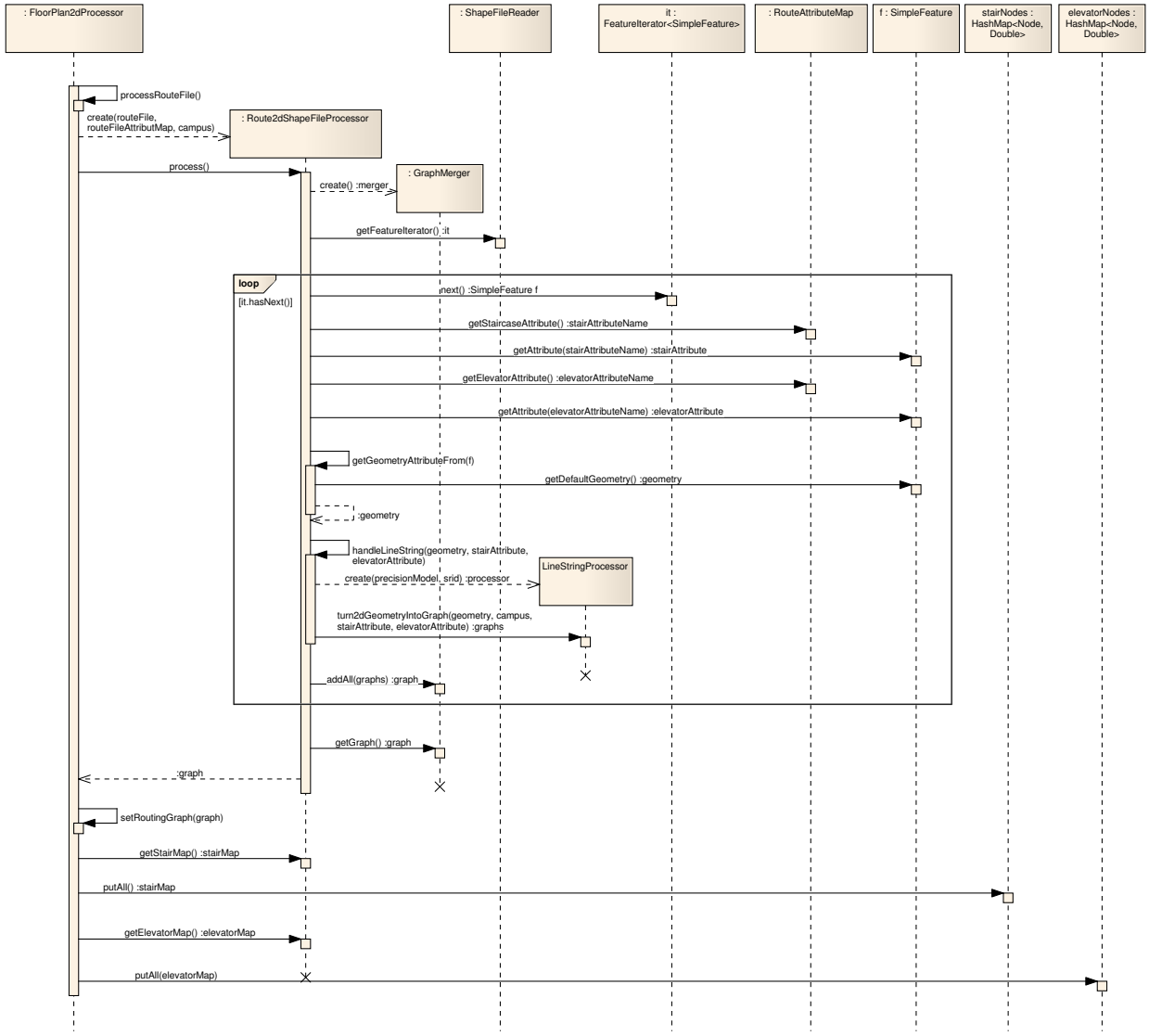

Abbildung 14.13: White-Box-Ansicht vom Erzeugen der Topologie, was ein Teil der User Story «Geometrie und Topologie vom Campus einlesen und Datenbank-Files erzeugen».

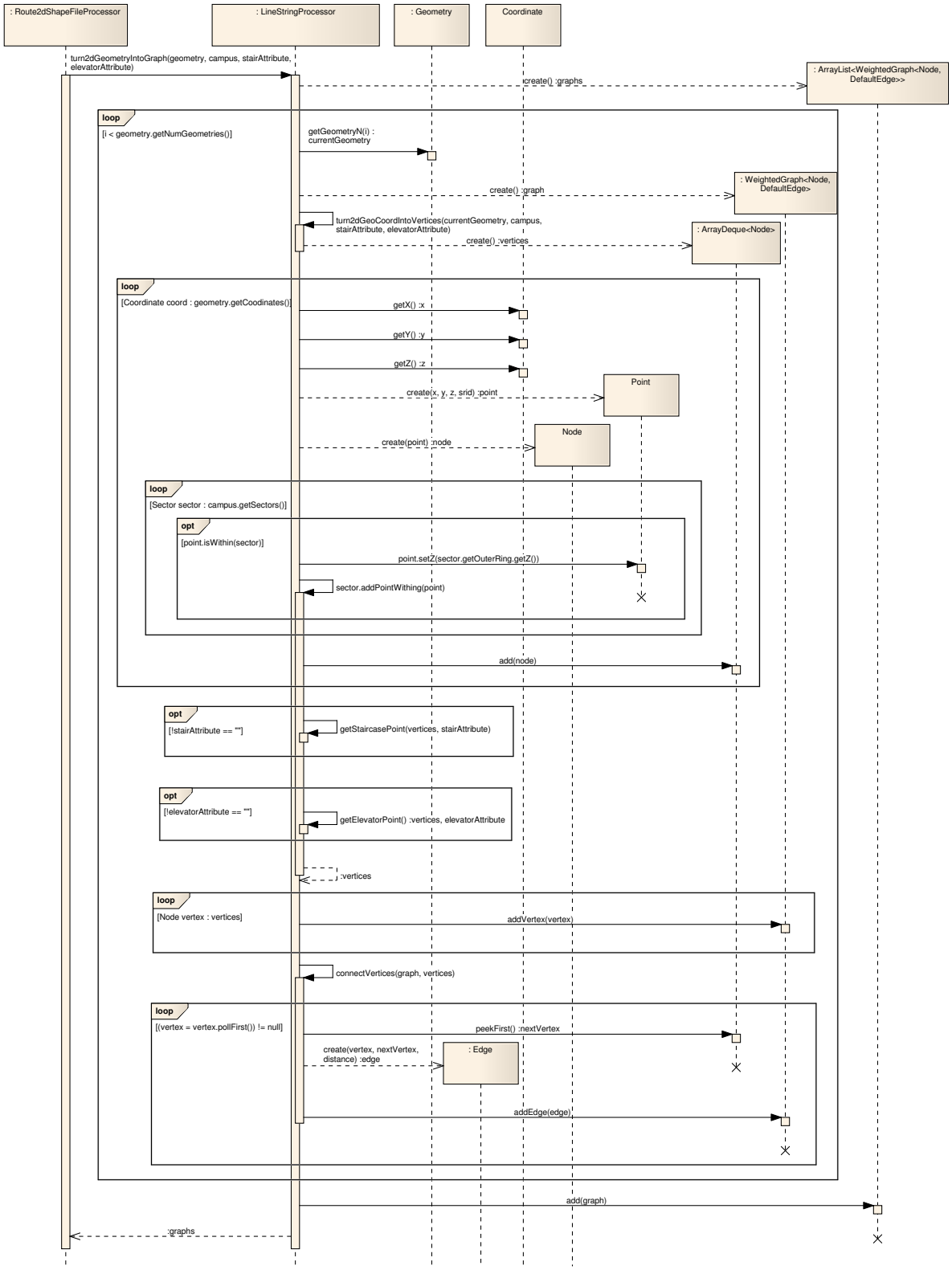

Abbildung 14.14: White-Box Ansicht des Aufrufs `turn2dGeometryIntoGraph()`.

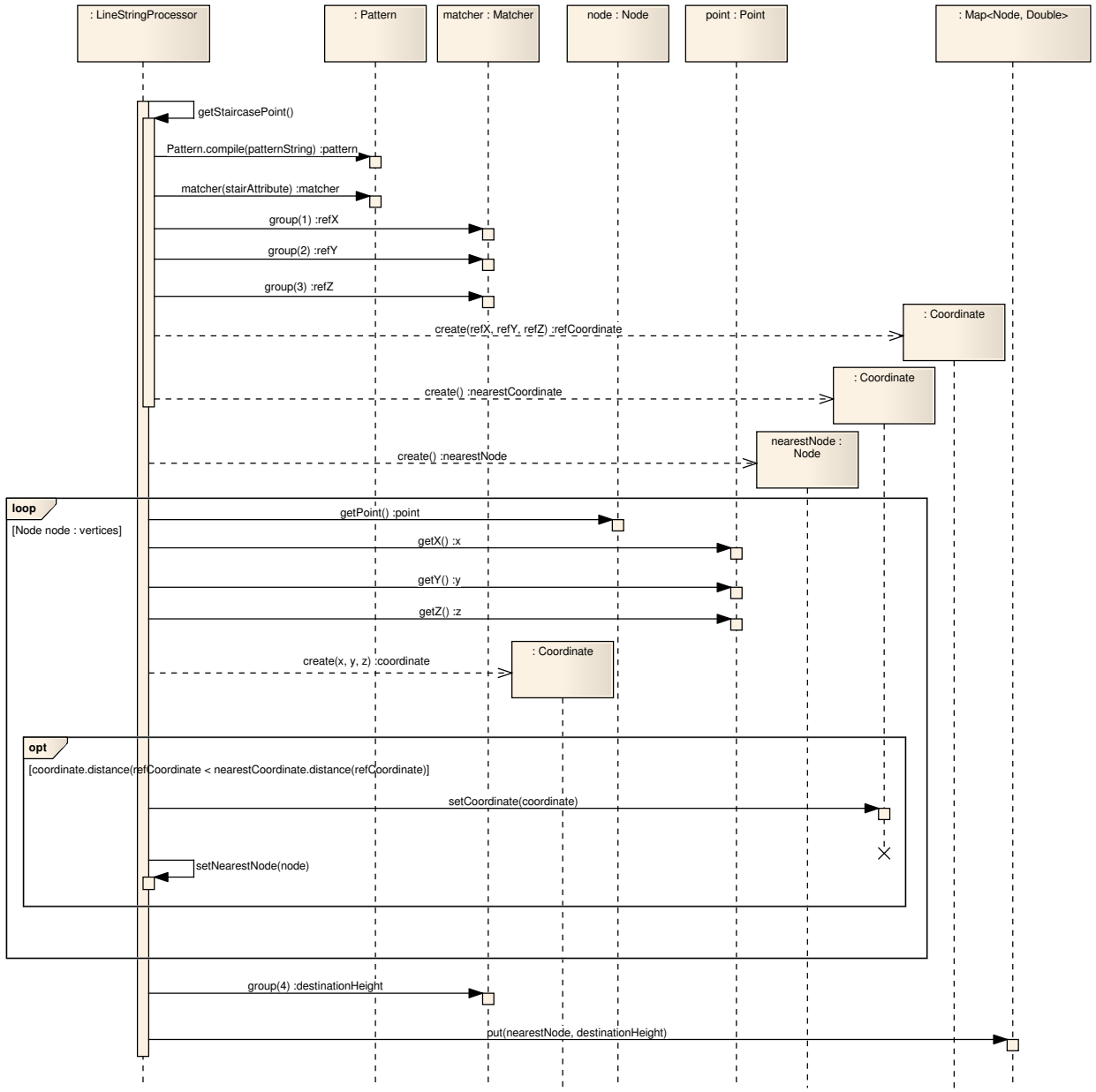

Abbildung 14.15: White-Box Ansicht des Aufrufs `getStaircasePoint(vertices, stairAttribute)`.

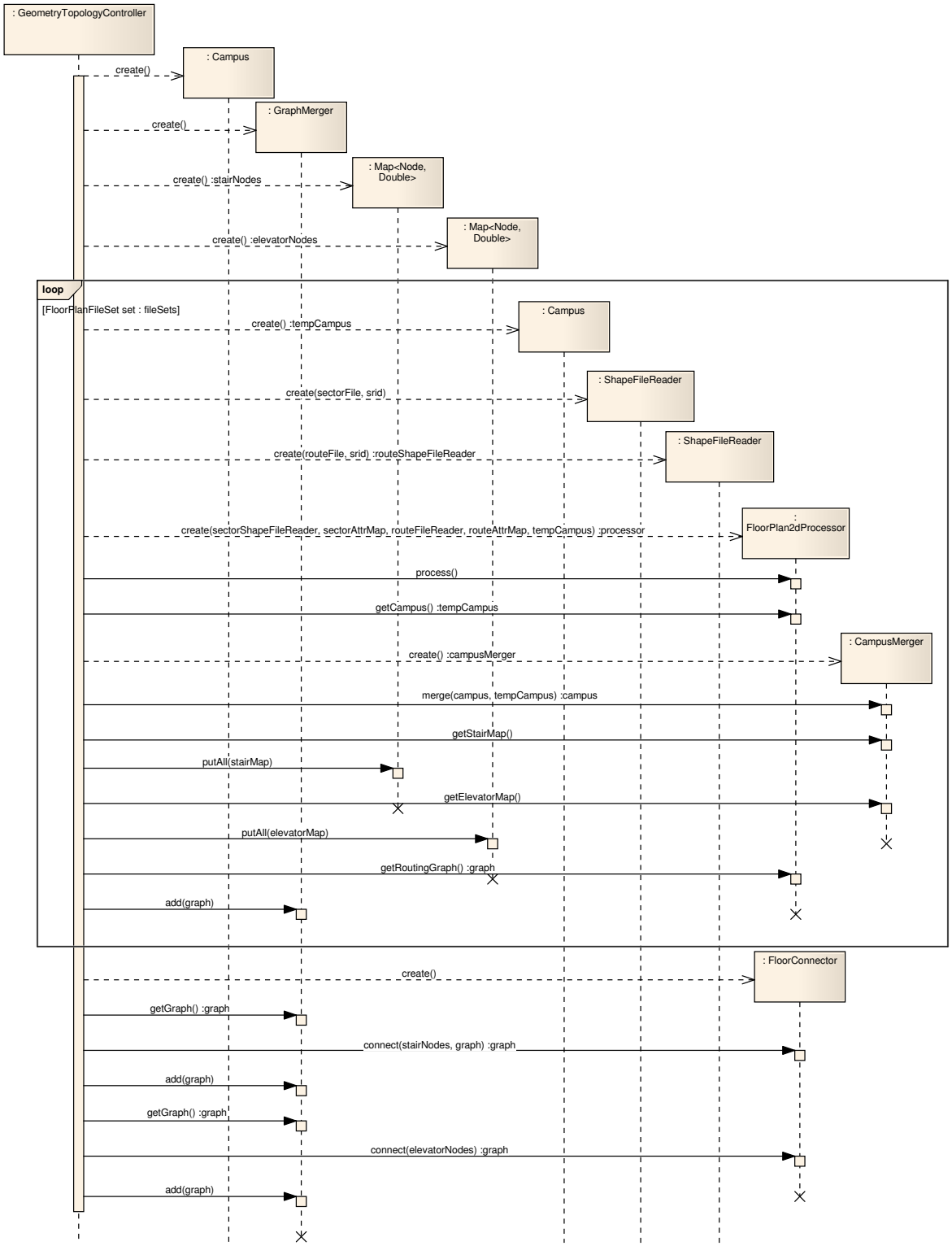

Abbildung 14.16: White-Box Ansicht des Ablaufes beim Zusammenfügen der Stockwerke.

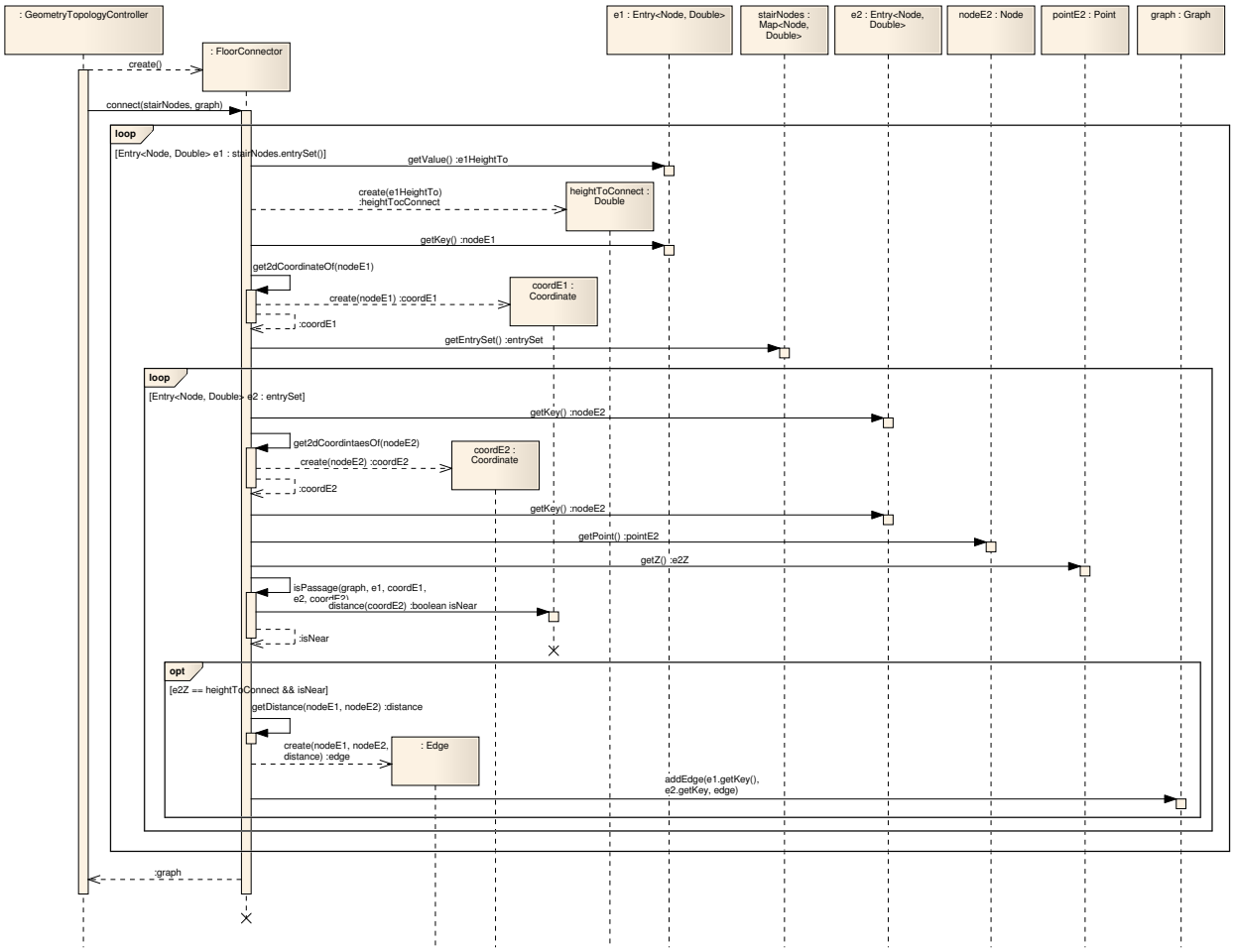

Abbildung 14.17: White-Box Ansicht des Aufrufs `connect(stairNodes, graph)`.

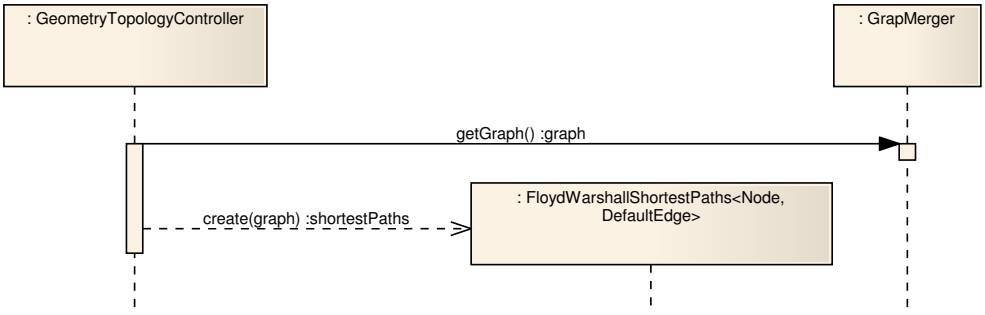

Abbildung 14.18: White-Box Ansicht der Berechnung aller Distanzen.

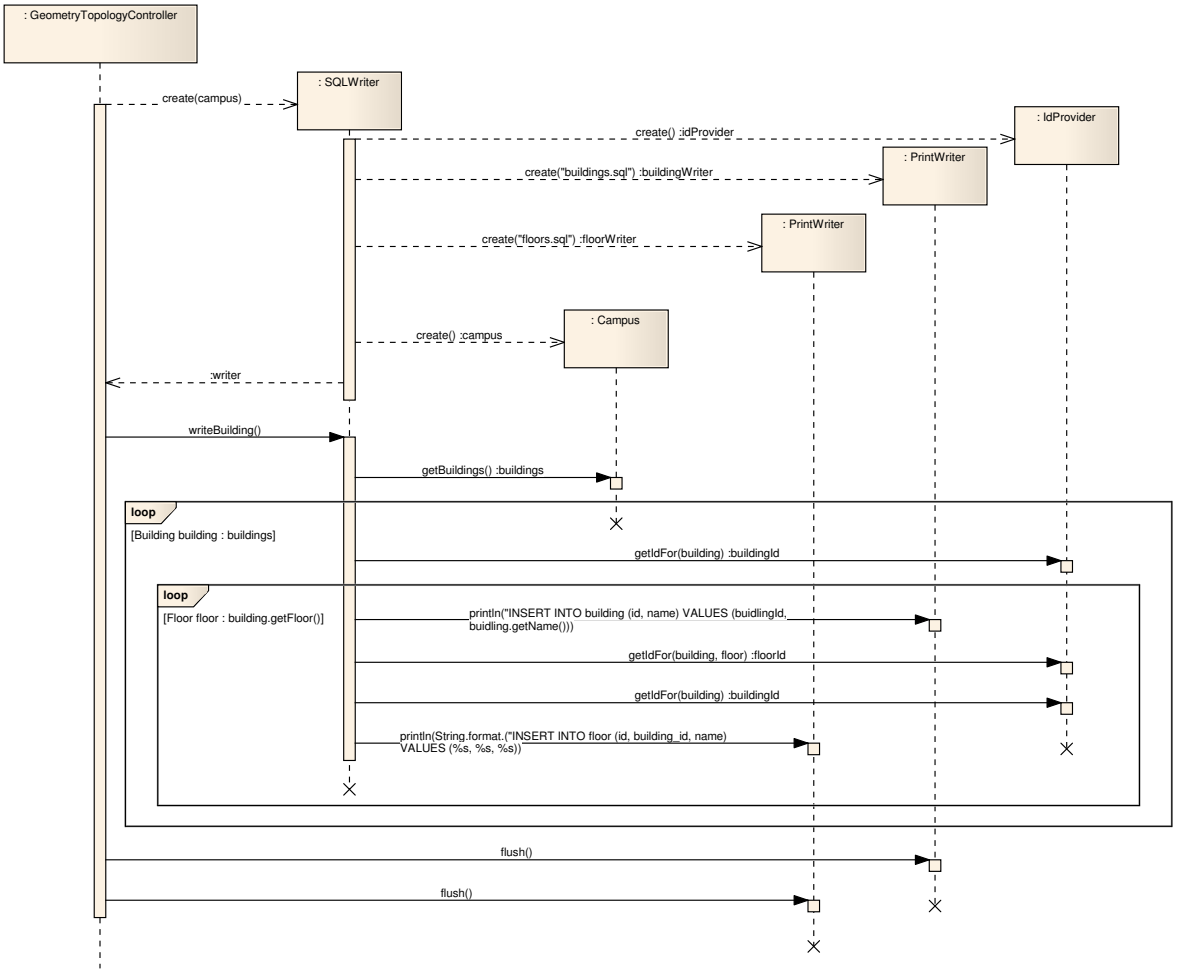

Abbildung 14.19: White-Box Ansicht zum Schreiben der Gebäude mit dem Aufruf `writeBuildings()`

15 Ausbilck

15.1 Aktueller Stand

Der aktuelle Stand des Shapeconverter bietet die Möglichkeit zur Umsetzung des Pilots, also das Vorbereiten der Geometrie und Topologie für das Gebäude HOER mit den drei Stockwerken d, e und f. Aus Zeitgründen konnte jedoch wenig Refactoring durchgeführt werden. Zum Beispiel kann der Pfad zum Einlesen der Shapefiles nicht frei gewählt werden, da dieser hart im Code einprogrammiert sind. Zudem sind im Bereich Testing noch Erweiterungen nötig.

15.2 Anwendung

Mit dem Shapeconverter wurde eine Basis geschaffen, welche sehr breit eingesetzt werden kann. Die eigenständige Software eignet sich nicht nur im Zusammenhang mit der spezifischen Aufgabe, Gebäude und Stockwerke eines Spitals einzulesen. Der Konverter kann überall dort eingesetzt werden, wo aus 2D-Informationen 2.5D-Modelle gewonnen werden müssen. Mögliche Anwendungsfälle sind allgemeine Gebäudeinformationssysteme (man denke an Ausstellungspläne für Messehalten) oder der Einsatz zur Berechnung respektive Visualisierung von Fluchtwegen in Hochhäusern, wie er von [12] demonstriert wird, die ein konzeptionell sehr ähnliches Modell verwenden.

15.3 Weiterentwicklung

Für die Verwendung im Universitätsspital Zürich über die Pilot-Durchführung hinaus muss die bereits vorbereitete Implementierung zur Verbindung von einzelnen Gebäuden noch fertiggestellt werden. Sollen Referenzdaten der Konnektoren ebenfalls über den Shapeconverter realisiert werden, so ist diese User Story ebenfalls noch zu implementieren.

Eine Möglichkeit, welche sich hinsichtlich der längerfristigen Entwicklung anbieten würde, wäre die Umstellung der Software auf eine Script-Sprache wie Python in Kombination mit den Bibliotheken Geospatial Data Abstraction Library (GDAL)[1] und und der damit verbundenen OGR Simple Feature Library (OGR)[2], da sich Java zusammen mit Geotools nicht als ideal erwiesen hat. Die vielen Interaktionen mit dem Betriebsystem, welche beim Shapeconverter gegeben sind, werden durch Python besser unterstützt als durch Java. Auch ist die Arbeit mit vielen verschiedenen Datenstrukturen, die verwendet werden, einfacher, da Python diesbezüglich erheblich ausdrucksstärker ist. Ein plattformübergreifendes Graphical User Interface (GUI) liesse sich beispielsweise mit der Kombination wxPython (Linux, Windows) und PyObjC (MacOS X) umsetzen, wie am Beispiel von Dropbox [15] zu sehen ist.

[1] `http://www.gdal.org/` (abgerufen: 9. Juni 2011)

[2] `http://www.gdal.org/ogr/index.html` (abgerufen : 9. Juni 2011)

Teil IV

Umsetzung Lokalisierung

16 Einführung

In diesem Abschnitt wird die Umsetzung des Lokalisierungssystems beschrieben, also den serverseitigen Software-Komponenten, die die Lokalisierungswerkzeuge abfragen, die Positionen der Sensoren und daraus die Positionen der Geräte und Personen bestimmen. Ebenfalls beschrieben ist die Webbenutzer-Schnittstelle, die zur Abfrage des Lokalisierungssystems dient.

Die Umsetzung der Konnektoren, welche die eigentliche Kommunikation mit den Lokalisierungswerkzeugen erledigen, werden erst im nächsten Teil beschrieben. Bei ihnen handelt es sich nämlich um komplett eigenständig entwickelbare Komponenten, die nur einen mit dem Lokalisierungssystem vereinbarten Vertrag einhalten müssen. Deshalb wurden ihnen ein separater Teil gewidmet.

17 Funktionale Anforderungen

Die nächsten Abschnitte beinhalten die erfassten funktionalen Anforderungen, wobei die Methodik nach [14] verwendet wurde. Der Abschnitt 17.1 (Seite 105) enthält die User Stories, Beschreibungen von 1 bis 2 Sätzen, welche die Funktionalität des Systems beschreiben. Ein Grossteil wurde bereits in [1] erfasst. Die Master Story List in Abschnitt 17.2 zeigt eine Übersicht über die User Stories, die Aufwandsschätzungen zur Umsetzung sowie die Umsetzungspriorität. Die «Einseiter» in Abschnitt 17.3 (Seite 118) beschreiben schlussendlich auf etwa einer Seite (daher der Name) die Aufgaben und zu erfüllenden Kriterien zur Umsetzung der User Stories.

17.1 User Stories

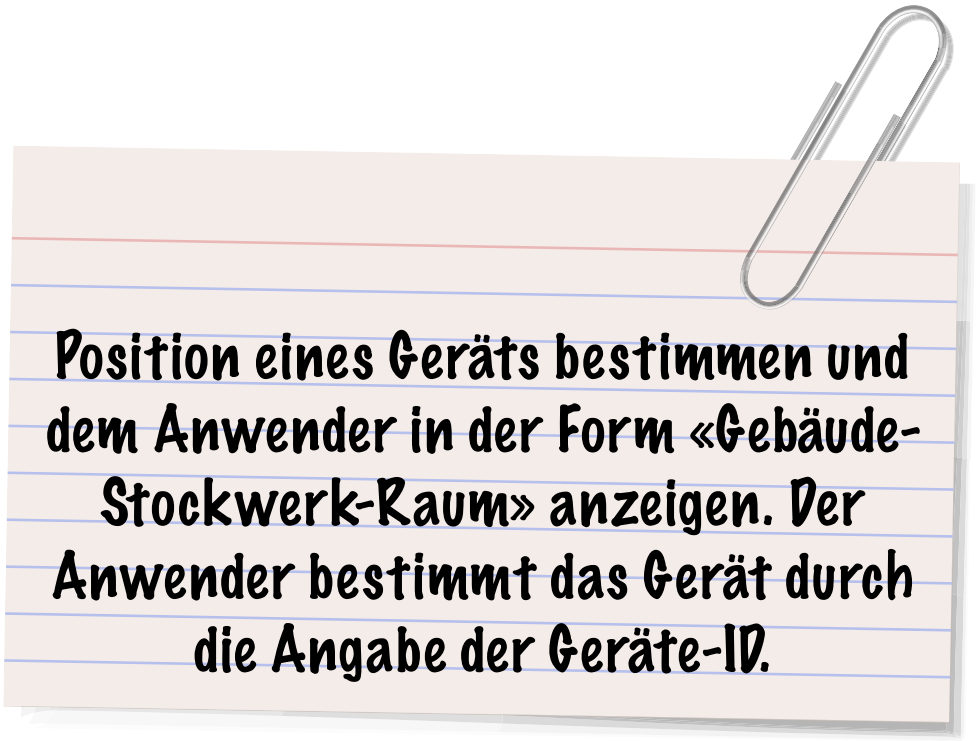

Abbildung 17.1: User Story «Position von einem Gerät bestimmen»

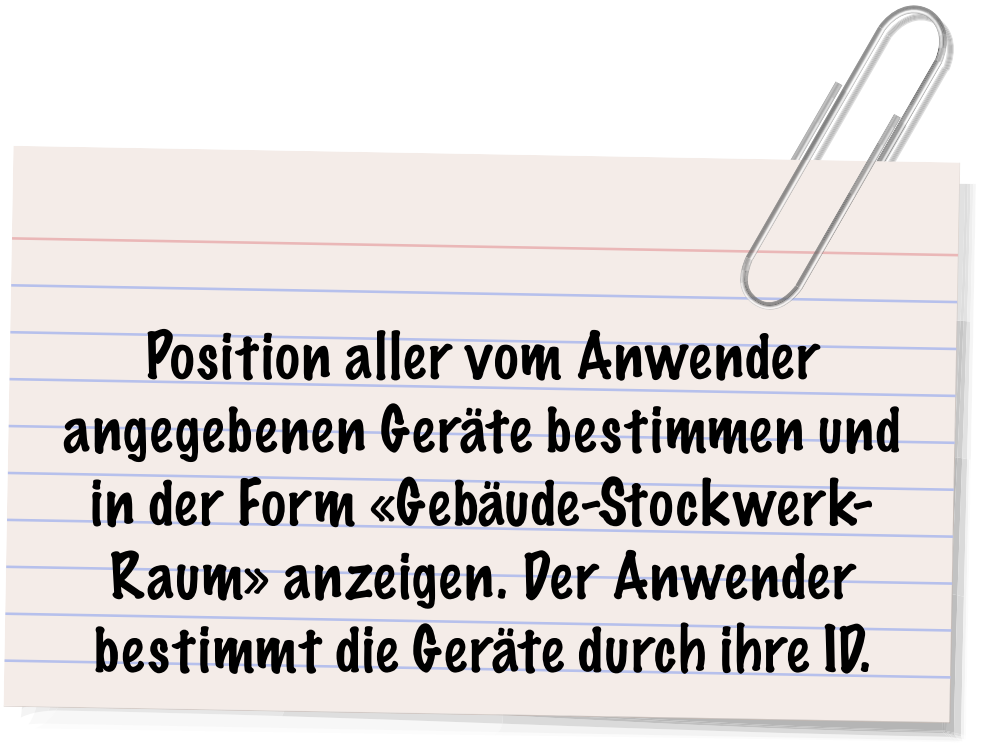

Abbildung 17.2: User Story «Position einer Menge von Geräten bestimmen»

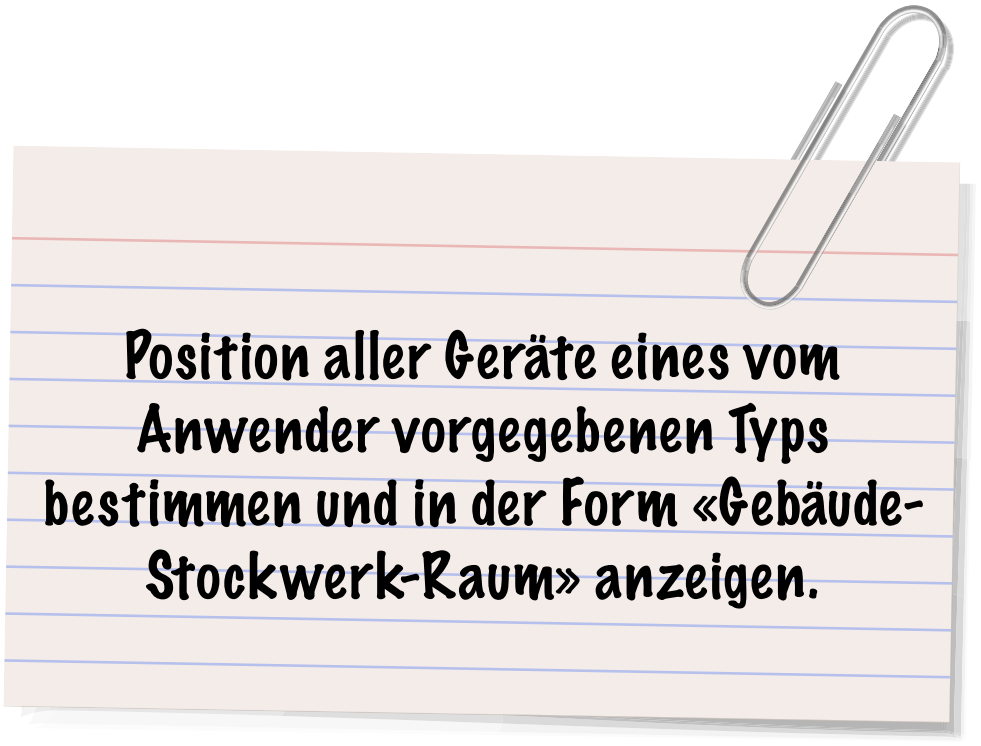

Abbildung 17.3: User Story «Position aller Geräte eines Typs bestimmen»

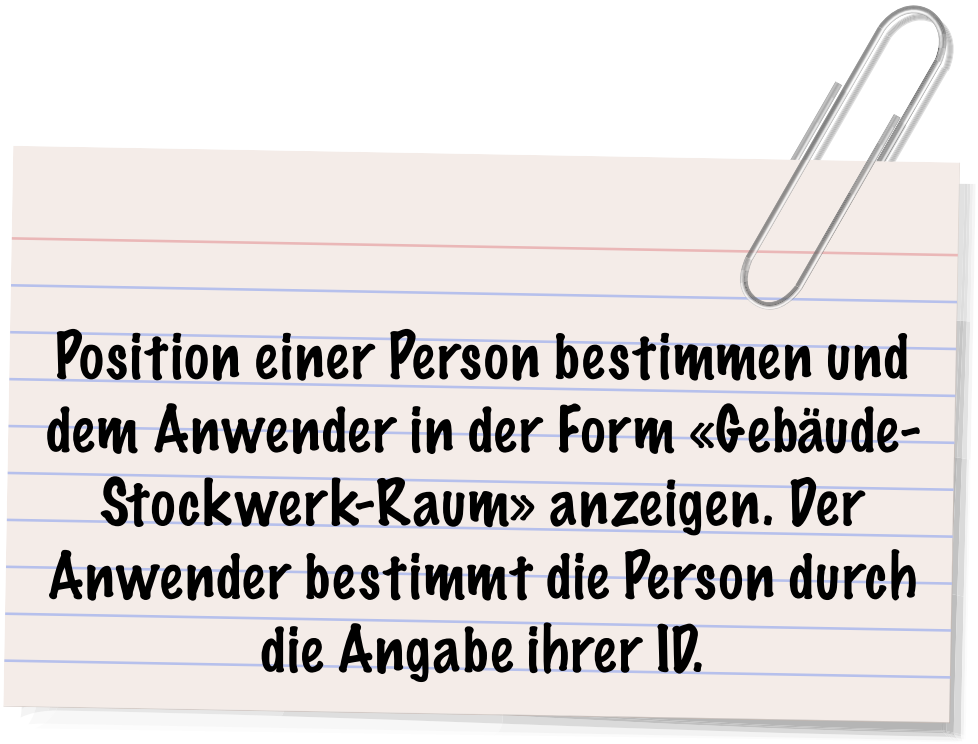

Abbildung 17.4: User Story «Position einer Person bestimmen»

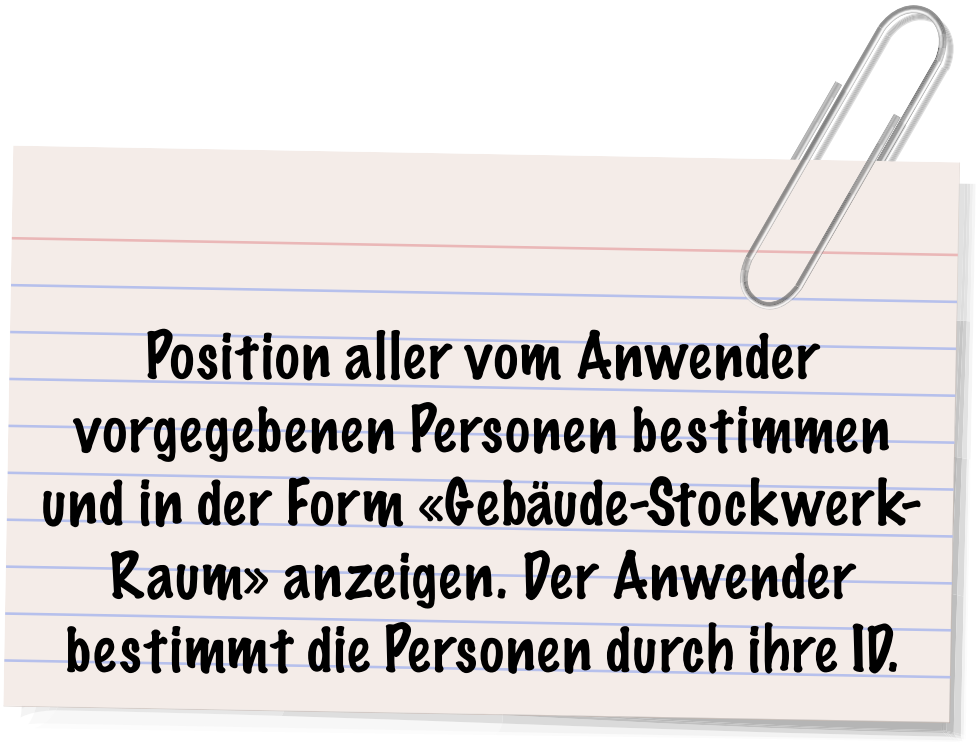

Abbildung 17.5: User Story «Position einer Menge von Personen bestimmen»

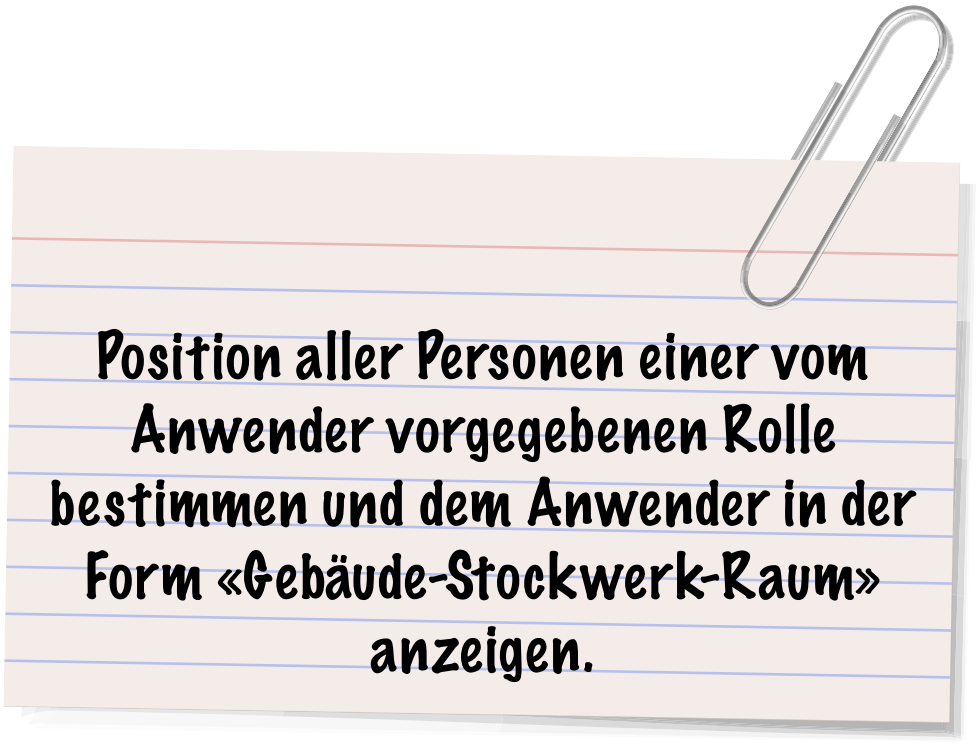

Abbildung 17.6: User Story «Position aller Personen einer Rolle bestimmen»

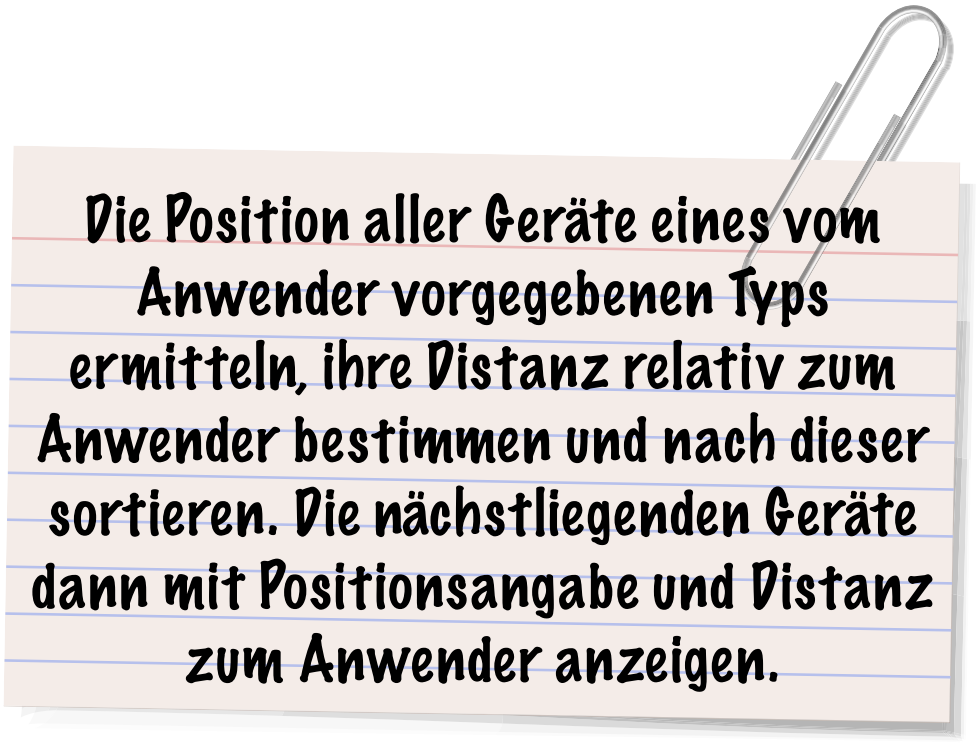

Abbildung 17.7: User Story «Von einer Person aus die nächsten Geräte eines Typs bestimmen»

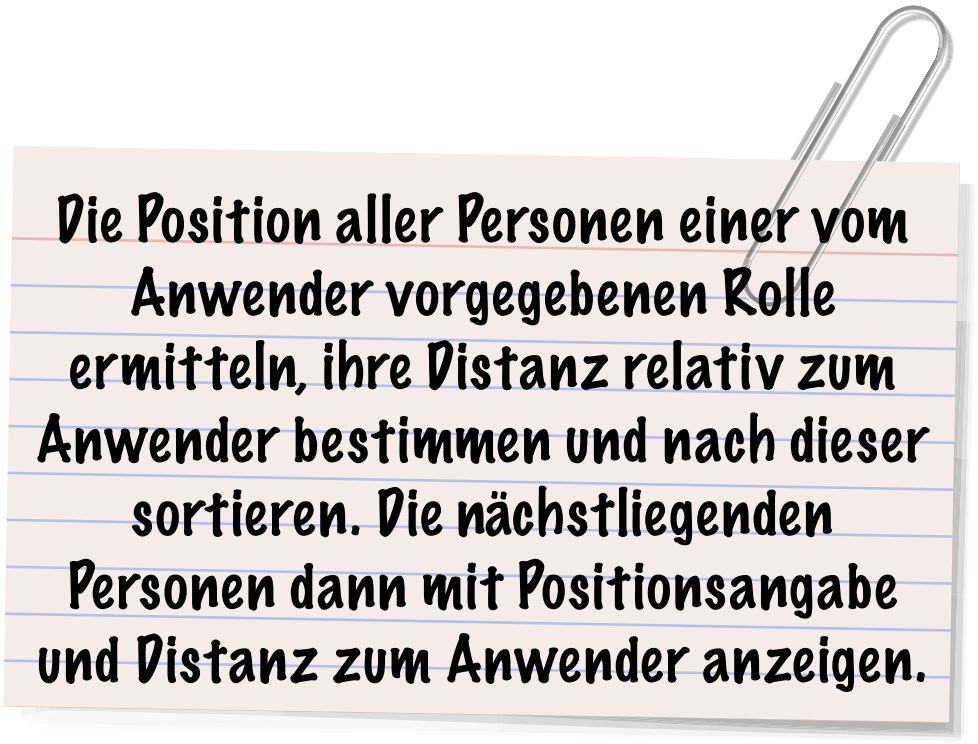

Abbildung 17.8: User Story «Von einer Person aus die nächsten Personen einer Rolle bestimmen»

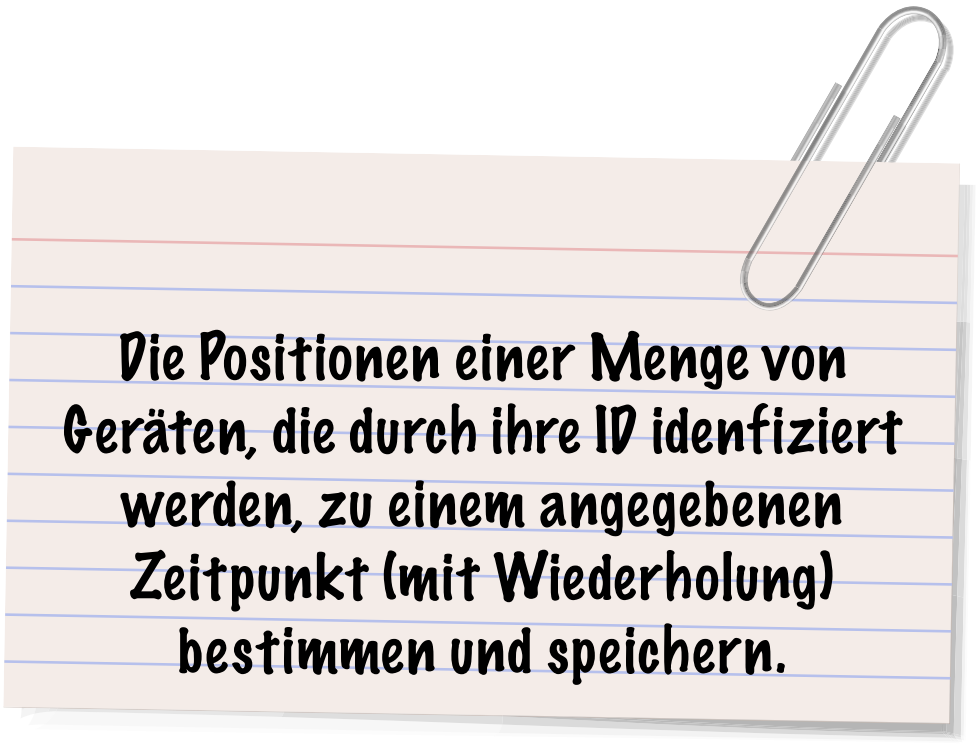

Abbildung 17.9: User Story «Positionen einer Menge von Geräten zu einem bestimmten Zeitpunkt zyklisch speichern»

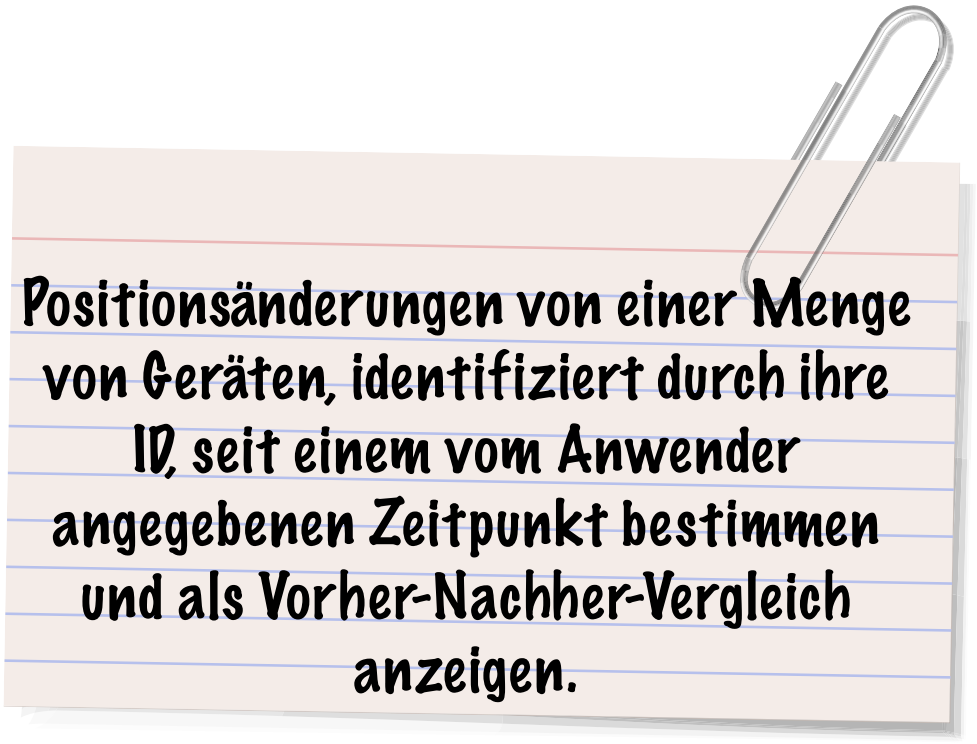

Abbildung 17.10: User Story «Für eine Menge von Geräten die Positionsänderung über einen Zeitraum bestimmen»

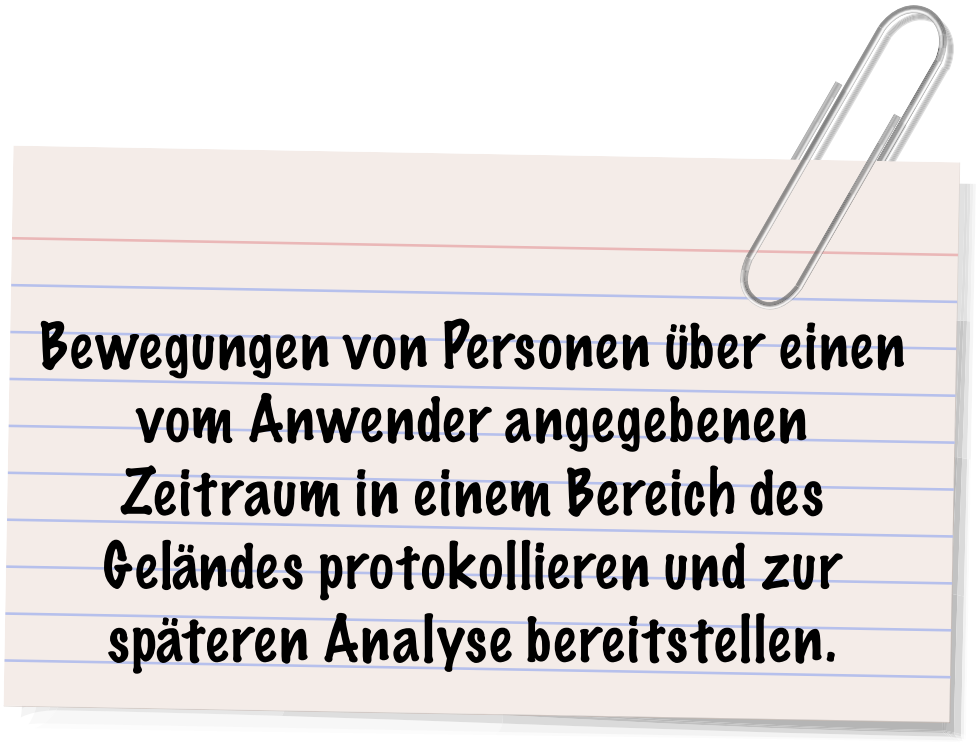

Abbildung 17.11: User Story «Menschenströme aufzeichnen»

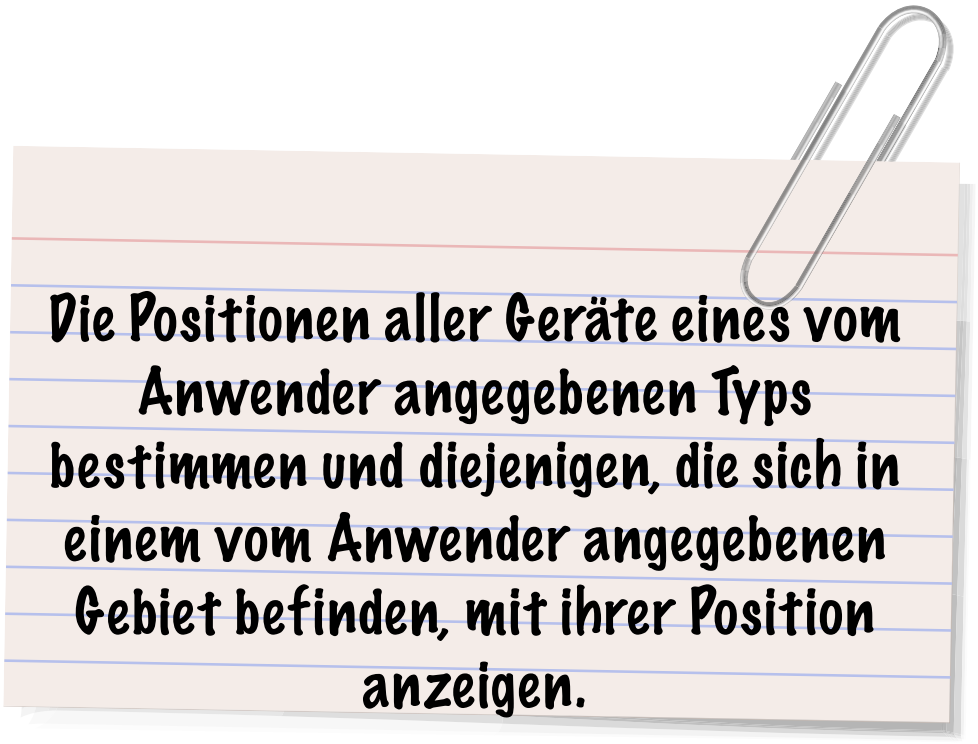

Abbildung 17.12: User Story «Alle Geräte eines Typs in einer Zone bestimmen»

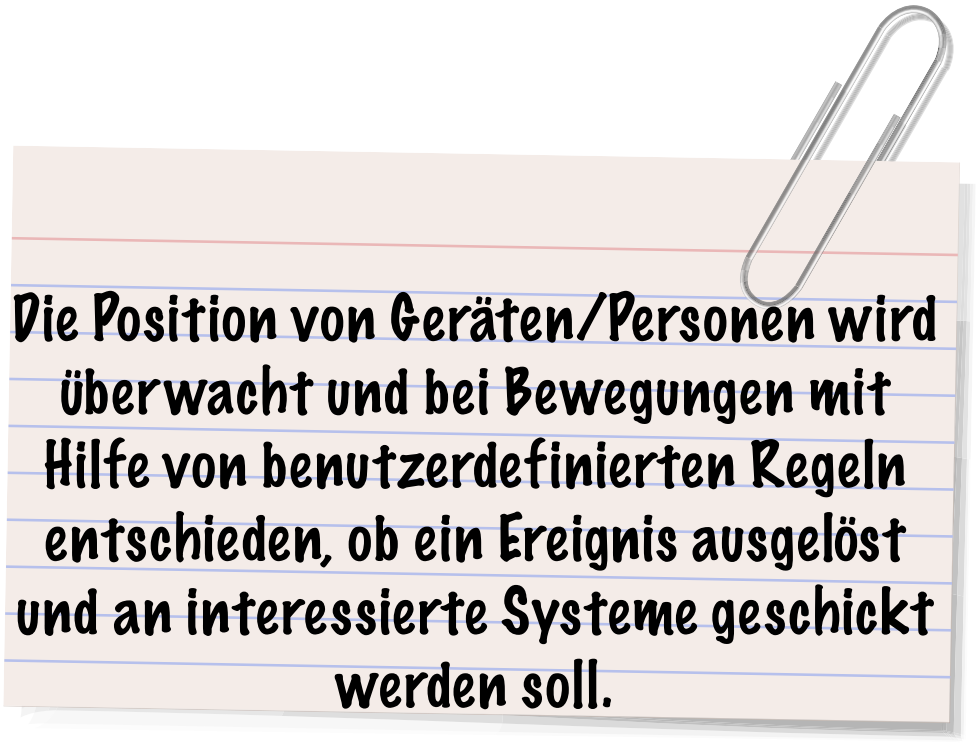

Abbildung 17.13: User Story «Benachrichtigung über Positionswechsel»

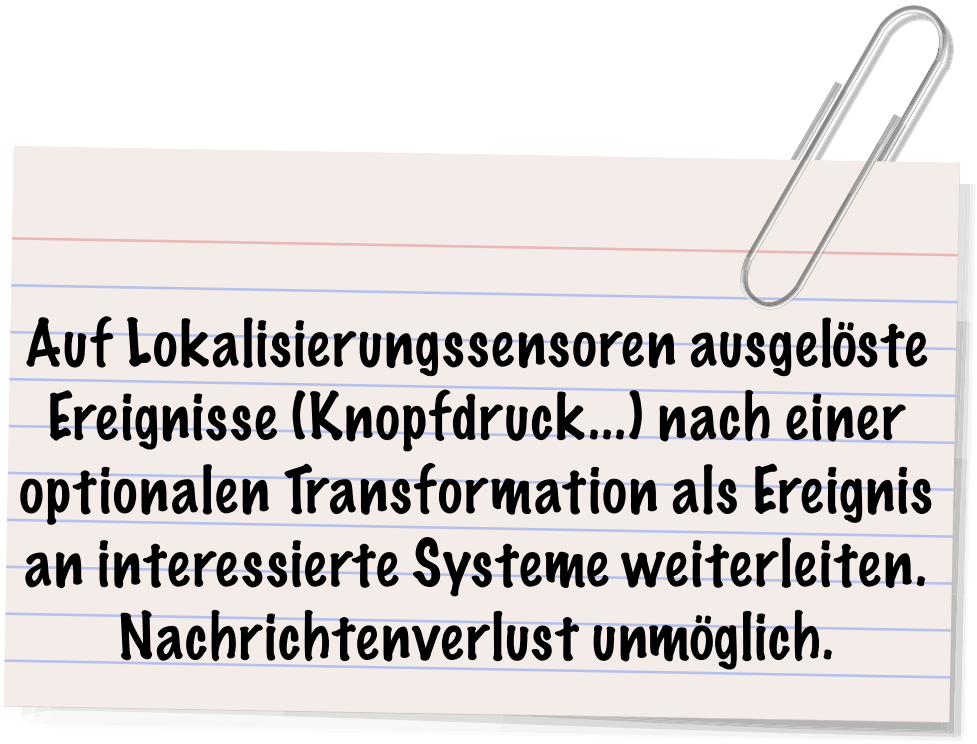

Abbildung 17.14: User Story «Ereignisse weiterleiten»

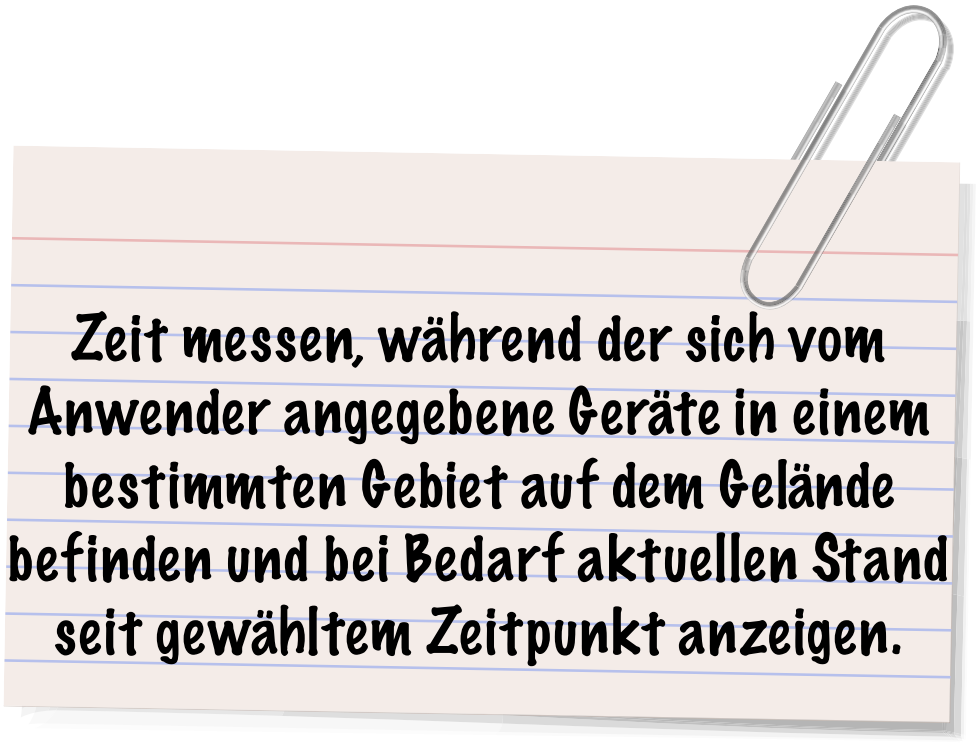

Abbildung 17.15: User Story «Zeit, während der sich Geräte in einer Zone befinden»

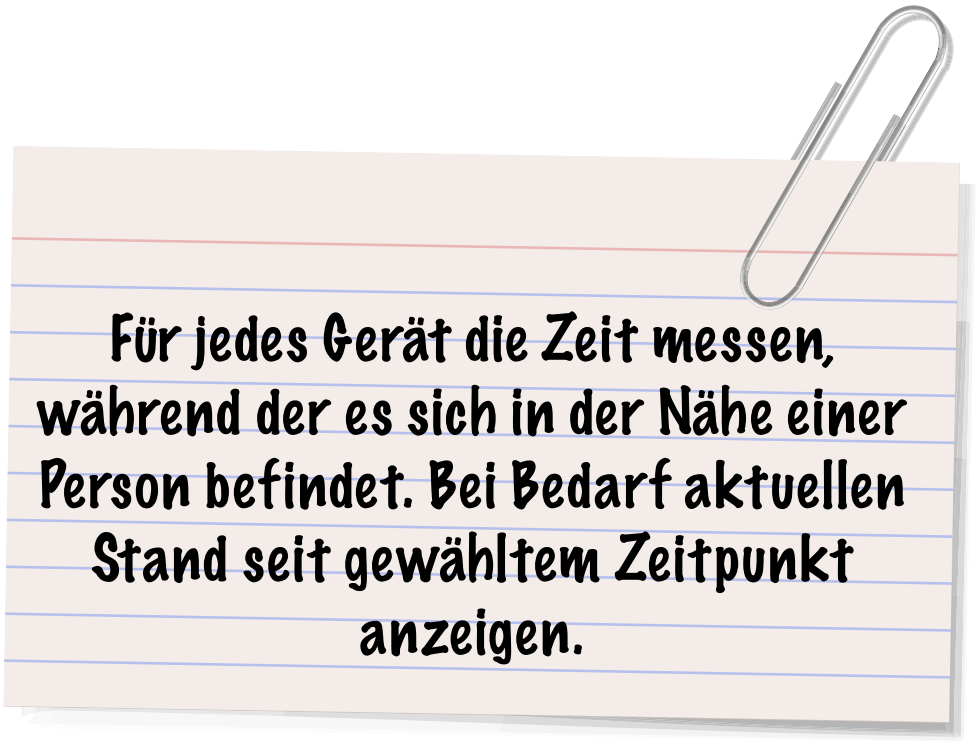

Abbildung 17.16: User Story «Zeit, während der sich Geräte bei einer Person befinden»

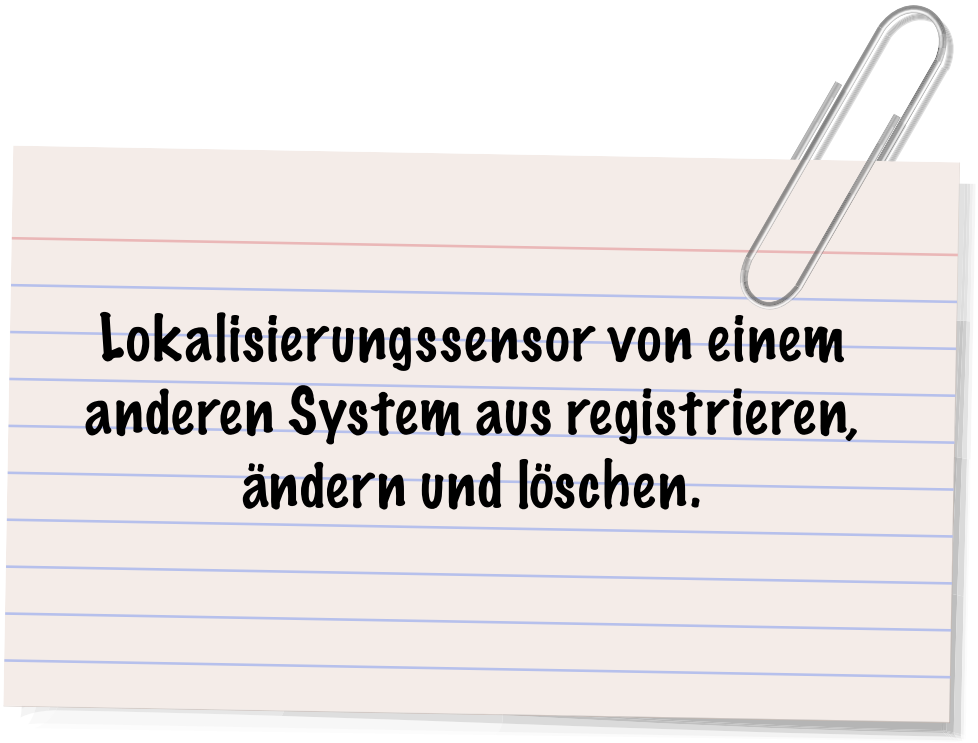

Abbildung 17.17: User Story «Lokalisierungssensoren verwalten»

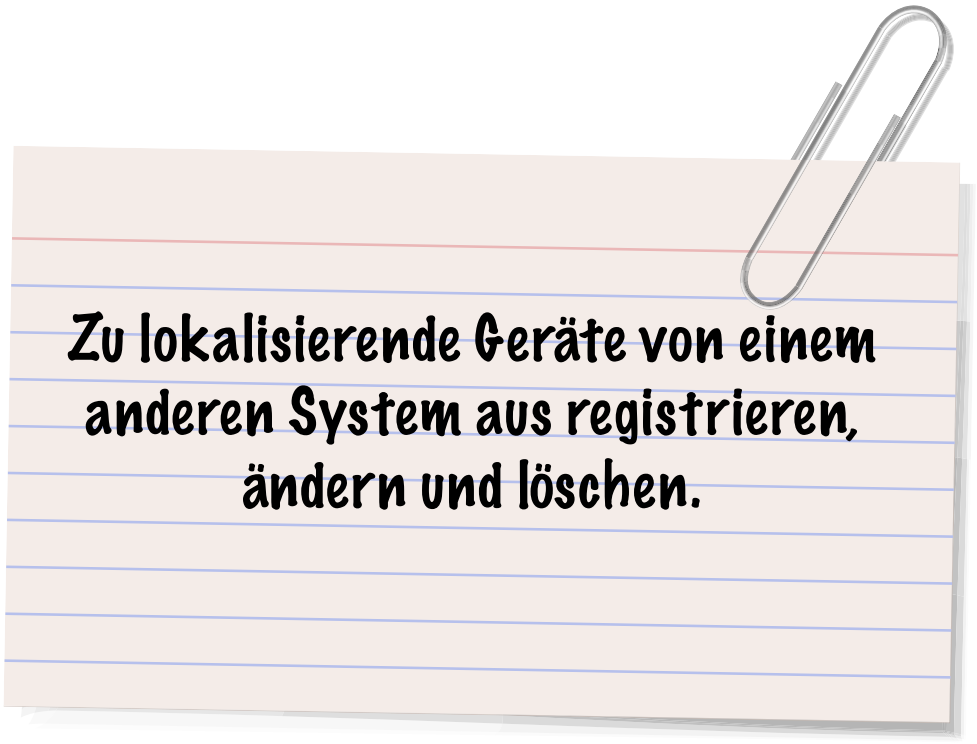

Abbildung 17.18: User Story «Geräte verwalten»

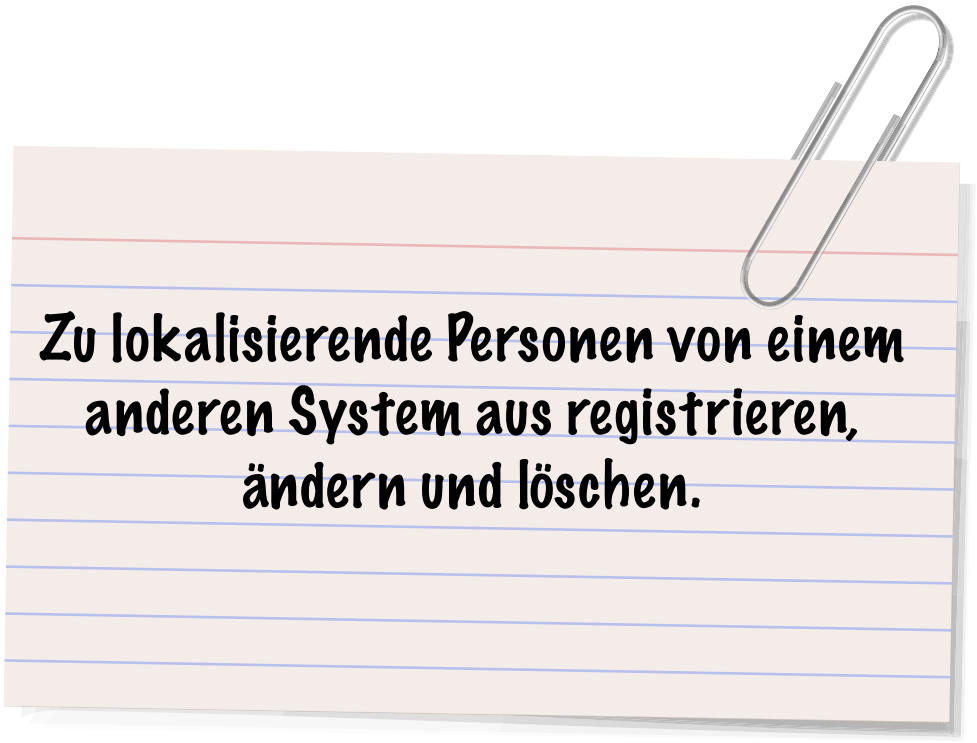

Abbildung 17.19: User Story «Personen verwalten»

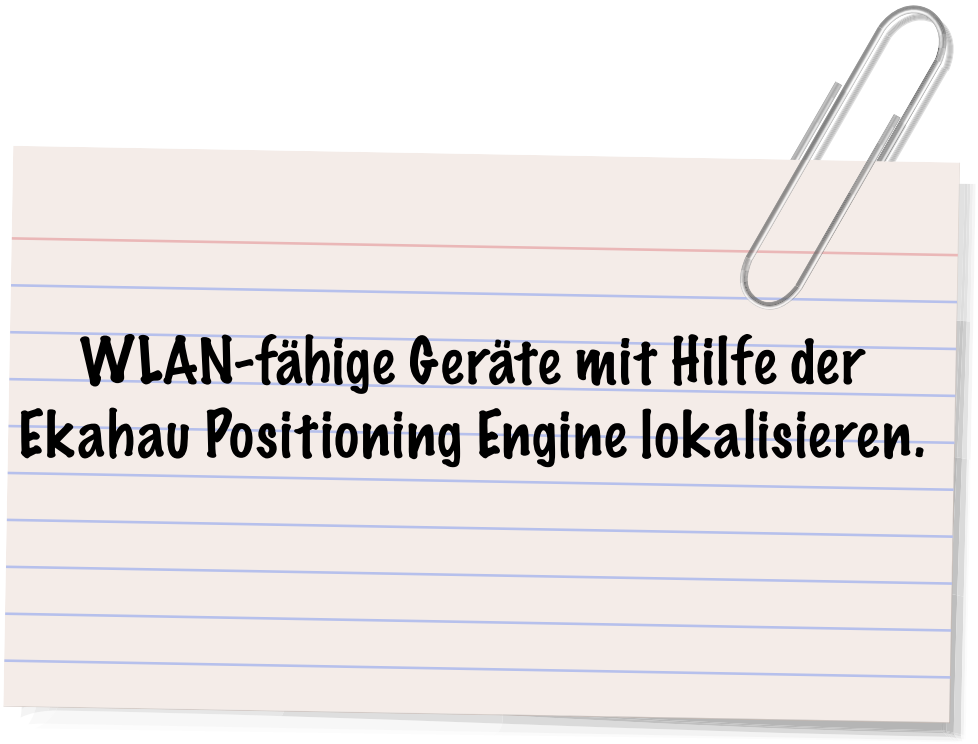

Abbildung 17.20: User Story «Ekahau-Lokalisierung»

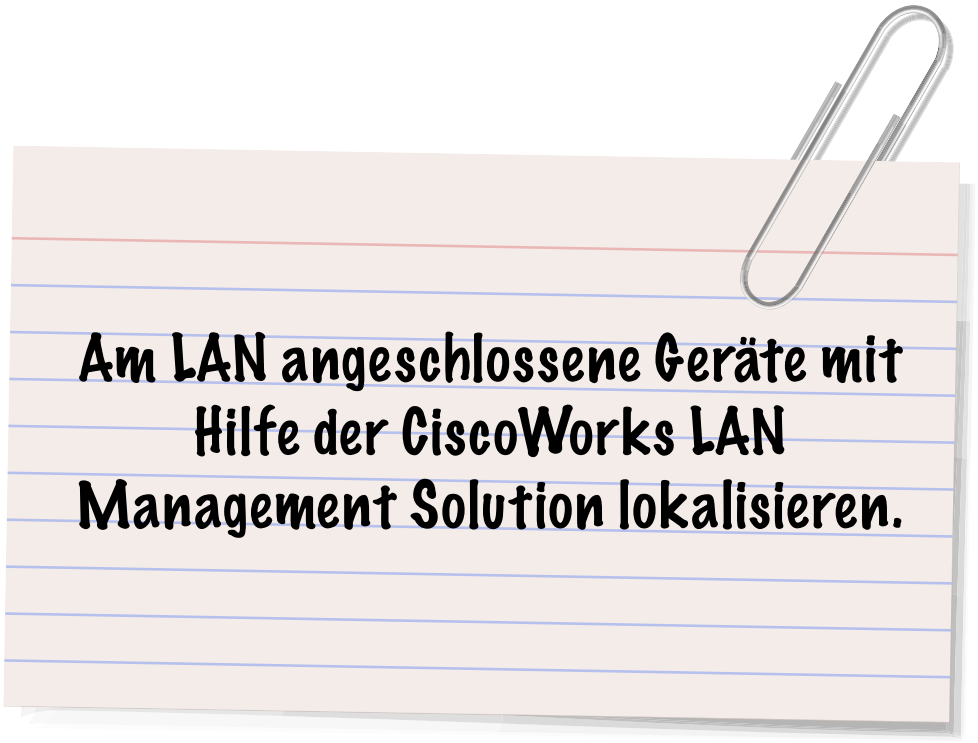

Abbildung 17.21: User Story «CiscoWorks-Lokalisierung»

Abbildung 17.22: User Story «Smartphone-Lokalisierung»

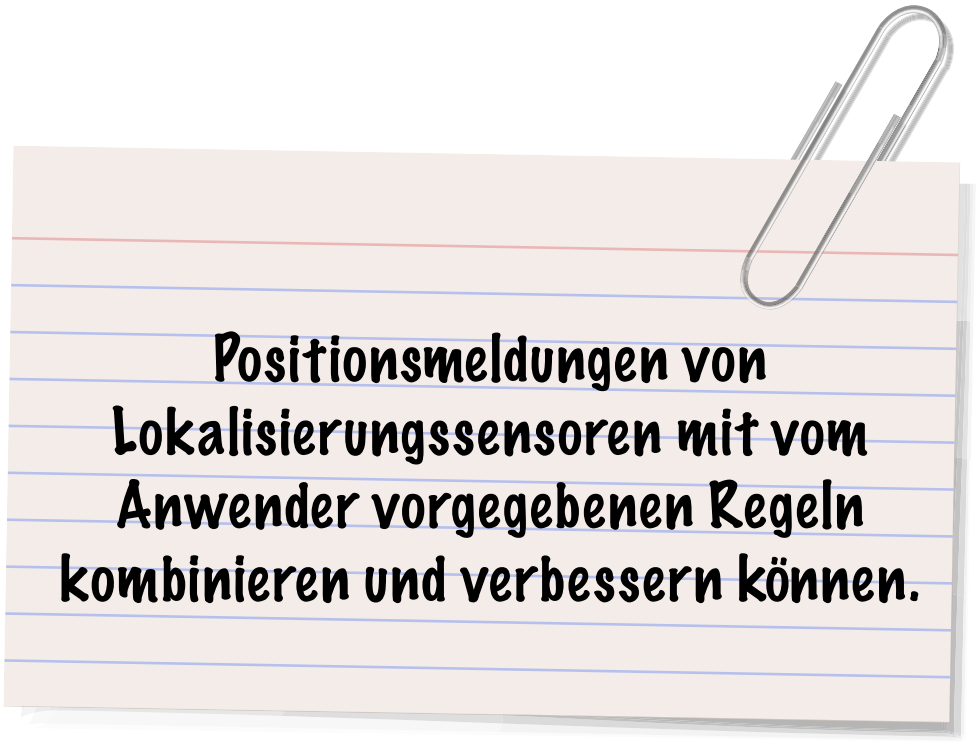

Abbildung 17.23: User Story «Positionen mit Regeln kombinieren und verbessern»

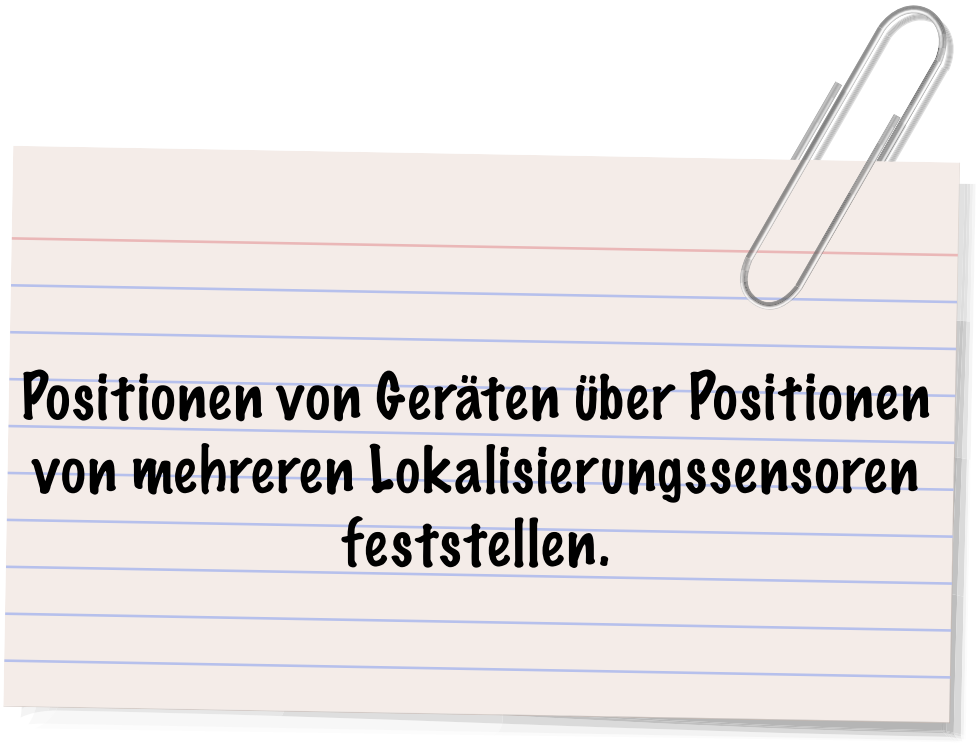

Abbildung 17.24: User Story «Geräte über mehrere Sensoren finden»

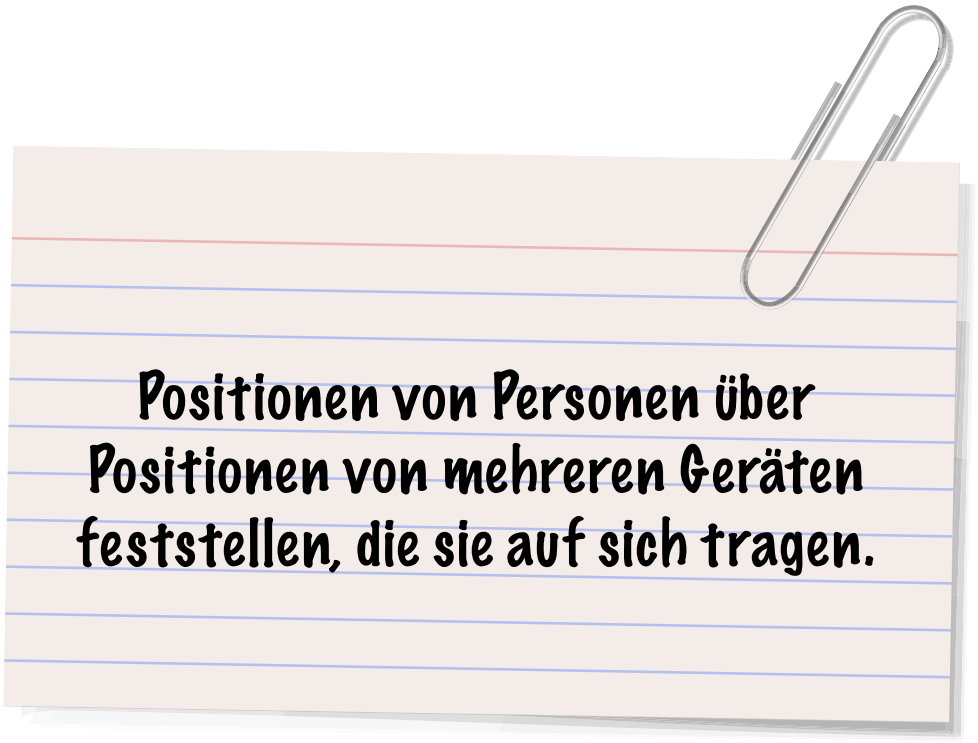

Abbildung 17.25: User Story «Personen über mehrere Geräte finden»

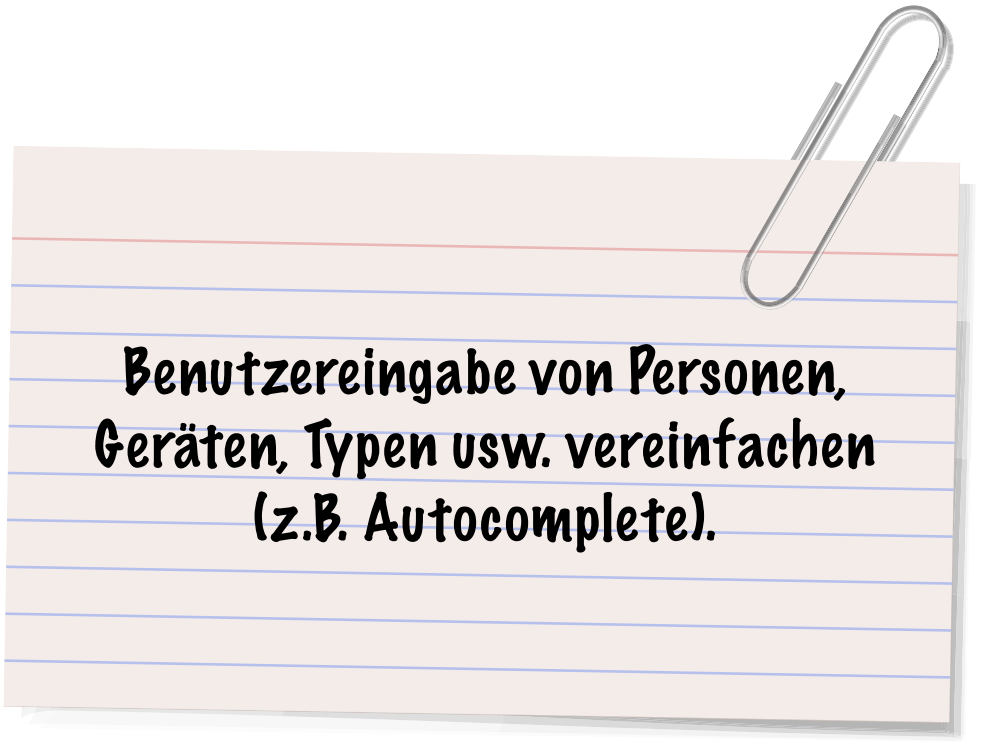

Abbildung 17.26: User Story «Benutzereingabe vereinfachen»

17.2 Master Story List

Die Tabelle 17.1 auf Seite 119 zeigt die Gewichtung der User Stories und ihre Umsetzungsreihenfolge, wobei die zuerst umzusetzenden User Stories am Anfang stehen und die «unwichtigsten» am Ende. Die Gewichtung der User Stories erfolgt relativ, wobei ein Wert von 1 für kleinen, ein Wert von 3 für mittleren und einer von 5 für grossen Umsetzungsaufwand steht.

Die User Stories wurden nicht strikte nach Priorität umgesetzt, da sich einige durch die Umsetzung anderer User Stories (zum Teil in leicht modifizierter Form) von selber ergeben haben oder die Zeit für eine grössere Aufgabe nicht mehr ausgereicht hat und statt dessen mehrere kleine umgesetzt wurden.

17.3 Einseiter

17.3.1 User Story: Ekahau-Lokalisierung

Beschreibung

Das System kann Objekte, welche WLAN-fähig sind oder einen eigenen Ekahau-Tag besitzen, über ihre Ekahau-Adresse orten. Die Positionsangabe, welche Ekahau liefert, wird in Landeskoordinaten (x-, y-, z-Position) an den Lokalisierungsservice geliefert. Zusätzlich wird die Information über die Genauigkeit der Position sowie den Zeitpunkt der Lokalisierung mitgegeben.

User Story	Aufwand	Status
Ekahau-Lokalisierung	5	erledigt
Position von einem Gerät bestimmen	1	erledigt
Von einer Person aus die nächsten Geräte eines Typs bestimmen	5	erledigt
CiscoWorks-Lokalisierung	5	erledigt
Smartphone-Lokalisierung	5	erledigt
Geräte über mehrere Sensoren finden	5	erledigt
Personen über mehrere Geräte finden	5	erledigt
Positionen mit Regeln kombinieren und verbessern	5	erledigt
Position aller Geräte eines Typs bestimmen	1	–
Für eine Menge von Geräten die Positionsänderung über einen Zeitraum bestimmen	5	–
Positionen einer Menge von Geräten zu einem bestimmten Zeitpunkt zyklisch speichern	5	–
Benachrichtigung über Positionswechsel	5	erledigt
Ereignisse weiterleiten	5	–
Zeit, während der sich Geräte in einer Zone befinden	5	–
Alle Geräte eines Typs in einer Zone bestimmen	1	–
Position einer Menge von Geräten bestimmen	3	–
Position aller Personen einer Rolle bestimmen	1	–
Von einer Person aus die nächsten Personen einer Rolle bestimmen	1	–
Zeit, während sich Geräte bei einer Person befinden	5	–
Position einer Person bestimmen	1	–
Position einer Menge von Personen bestimmen	3	–
Menschenströme aufzeichnen	5	erledigt
Lokalisierungssensoren verwalten	3	erledigt
Geräte verwalten	3	erledigt
Personen verwalten	3	erledigt
Benutzereingabe vereinfachen	3	–

Tabelle 17.1: Master Story List zum Zeitpunkt der Drucklegung

Aufgaben

1. Position über Ekahau erfragen.
2. Erhaltene Position in Landeskoordinaten umwandeln.
3. Position an Lokalisierungsservice weiterleiten.

Testkriterien

- Die Position in Landeskoordinaten entspricht der Position welche durch Ekahau gesendet wurde.
- Kann eine Position nicht ermittelt werden, wird ein entsprechender Report generiert.

17.3.2 User Story: Position von einem Gerät bestimmen

Beschreibung

Der Anwender sucht ein Gerät, von dem er die Identifikation kennt, beispielsweise eine Inventarnummer. Um ihm zu helfen, das Gerät zu finden, soll das System eine Positionsangabe der Form «Gebäude-Stockwerk-Raum» liefern.

Aufgaben

1. Eingabeformular erstellen.
2. Gerät lokalisieren.
3. Position in Angabe der Form «Gebäude-Stockwerk-Raum» umwandeln.
4. Resultatanzeige erstellen.
5. Sicherheit (für den Moment) weglassen.

Testkriterien

- Anwender erhält Positionsangabe für auffindbares Gerät, dessen Inventarnummer er kennt.
- Anwender erhält Fehlermeldung für nicht auffindbares Gerät.

17.3.3 User Story: Von einer Person aus die nächsten Geräte eines Typs bestimmen

Beschreibung

Der Anwender sucht ein Gerät eines ihm bekannten Typs (z.B. mobiles Röntgengerät). Dabei will er das von ihm aus nächste Gerät dieses Typs finden. Damit der Benutzer selber entscheiden kann, welches der nächstliegenden Geräte des Typs für ihn relevant ist, werden ihm die nächstliegenden Geräte mit Positionsangabe der Form «Gebäude-Stockwerk-Raum» und der Distanz geliefert. Diese sind bezüglich der Distanz aufsteigend sortiert, beginnend mit den Nächsten.

Aufgaben

1. Eingabeformular erstellen.
2. Anfragesteller lokalisieren.
3. Geräte des gewählten Typs lokalisieren.

4. Distanzen vom Anfragesteller zu den Geräten bestimmen.
5. Position in Angabe der Form «Gebäude-Stockwerk-Raum» umwandeln.
6. Resultatanzeige, sortiert nach Distanzen erstellen.
7. Sicherheit (für den Moment) weglassen.

Testkriterien

- Anwender erhält die Geräte des gewünschten Typs, welche von ihm aus am nächsten sind, zusammen mit relativen Distanzangaben.
- Distanzangaben entsprechen den wirklichen Distanzen.
- Anwender erhält eine Fehlermeldung, kann kein Gerät des gewünschten Typs lokalisiert werden.

17.3.4 User Story: CiscoWorks-Lokalisierung

Beschreibung

Das System kann Objekte über ihre LAN-Adresse orten. Die Positionsangabe welche die CiscoWorks LAN Management Solution liefert, wird in Landeskoordinaten (x-, y-, z-Position) an den Lokalisierungsservice geliefert. Zusätzlich wird die Information über die Genauigkeit der Position sowie den Zeitpunkt der Lokalisierung mitgegeben.

Aufgaben

1. Position über CiscoWorks erfragen.
2. Erhaltene Position in Landeskoordinaten umwandeln.
3. Position an Lokalisierungsservice weiterleiten.

Testkriterien

- Die Position in Landeskoordinaten entspricht der Position welche durch CiscoWorks gesendet wurde.
- Kann eine Position nicht ermittelt werden, wird ein entsprechender Report generiert.

17.3.5 User Story: Smartphone-Lokalisierung

Beschreibung

Es sollen Personen über die Position ihres Smartphones lokalisiert werden. Dazu sendet werden vom Smartphone gesendete Positionsinformationen in Landeskoordinaten umgewandelt (x, y, z-Position) und an den Lokalisierungsservice geliefert. Zusätzlich wird die Information über die Genauigkeit der Position sowie den Zeitpunkt, an dem die Position vom Smartphone gesendet wurde, mitgegeben.

Aufgaben

1. Vom Smartphone gesendete Position in Landeskoordinaten umwandeln.
2. Position an Lokalisierungsservice weiterleiten.

Testkriterien

- Die Position in Landeskoordinaten entspricht der Position, welche durch das Smartphone gesendet wurde.
- Kann die Position nicht ermittelt werden, wird ein entsprechender Report generiert.
- Befindet sich das Smartphone nicht auf dem Campus, wird ein Report generiert, der meldet, dass sich das Smartphone ausserhalb des Campus befindet.

17.3.6 User Story: Positionen mit Regeln kombinieren und verbessern

Beschreibung

Verschiedene Positionen desselben oder mehrerer Sensoren beziehungsweise Geräte von unterschiedlichen Zeitpunkten sollen miteinander verglichen werden können, um «ungenaue» Positionsangaben zu verbessern oder Falschmeldungen ausfiltern zu können. Beispiel: «Wenn WLAN-Handset auf dem Gelände aber Smartphone nicht, dann Position vom Smartphone wählen».

Aufgaben

1. Positionen vor Speicherung durch Rule Engine durchlaufen lassen statt direkt speichern.
2. Positionen aller relevanten Sensoren (bei Geräten) beziehungsweise Geräten (bei Personen) der Rule Engine als Fakten injizieren.
3. Positionshistorie aller relevanten Sensoren (bei Geräten) beziehungsweise Geräte (bei Personen) der Rule Engine als Fakten injizieren.
4. Etwaig in Rule Engine modifizierte Position speichern.

Testkriterien

- Jede für Gerät oder Person zu speichernde Position durchläuft Rule Engine.
- Für jeden Durchlauf produziert die Rule Engine eine Position, die – und nur die – gespeichert wird.
- Jeder Aspekt einer Positionsangabe kann durch die Rule Engine modifiziert werden.

17.3.7 User Story: Geräte über mehrere Sensoren finden

Beschreibung

Einem Gerät können mehrere Lokalisierungssensoren zugewiesen werden, über das es gefunden werden kann. An einem Bett kann beispielsweise ein Ekahau-Tag und ein RFID-Tag befestigt sein. Kann ein Sensor nicht gefunden werden, wäre es damit möglich, auf die Positionsangaben der anderen Sensoren zurückzugreifen.

Aufgaben

1. Ermöglichen, einem Gerät mehrere Lokalisierungssensoren zuzuweisen.
2. Bei jeder Lokalisierung eines Geräts jeden Lokalisierungssensor abfragen.

Testkriterien

- Wenn ein Gerät gesucht wird, dem mehrere Lokalisierungssensoren zugewiesen sind, wird jeder Sensor lokalisiert.

17.3.8 User Story: Personen über mehrere Geräte finden

Beschreibung

Einer Person können mehrere Geräte zugewiesen werden, über die sie gefunden werden kann. Ein Arzt kann beispielsweise über ein WLAN-Handset, ein Smartphone und einen Tablet-PC verfügen. Kann ein Gerät nicht gefunden werden, wäre es damit möglich, auf die Positionsangaben der anderen Geräte zurückzugreifen.

Aufgaben

1. Ermöglichen, einer Person mehrere Geräte zuzuweisen.
2. Bei jeder Lokalisierung einer Person die Position jedes zugewiesenen Geräts bestimmen.

Testkriterien

- Wenn eine Person gesucht wird, der mehrere Geräte zugewiesen sind, wird jedes Gerät lokalisiert.

17.3.9 User Story: Position aller Geräte eines Typs bestimmen

Beschreibung

Der Anwender sucht die Position aller Geräte eines bestimmten Typs (z.B. alle mobilen Röntgengeräte). Das System liefert ihm auf seine Anfrage eine Liste aller Geräte mit ihrer Position in Form von «Gebäude-Stockwerk-Raum».

Aufgaben

1. Eingabeformular erstellen.
2. Geräteidentifikationen aller Geräte des Typs bestimmen.
3. Geräte des gewählten Typs lokalisieren.
4. Positionen in Angabe der Form «Gebäude-Stockwerk-Raum» umwandeln.
5. Resultatanzeige erstellen.
6. Sicherheit (für den Moment) weglassen.

Testkriterien

- Anwender erhält die Positionen aller Geräte des gesuchten Typs mit ihrer Position.
- Kann ein Gerät nicht lokalisiert werden, bekommt er für dieses Gerät eine entsprechende Fehlermeldung.

17.3.10 User Story: Benachrichtigung über Positionswechsel

Beschreibung

Ändert ein Gerät oder eine Person ihre Position, kann mit Hilfe von Regeln entschieden werden, ob ein Ereignis eingetreten ist. Das Ereignis wird dann an interessierte Systeme verschickt.

Aufgaben

1. Die Geräte oder Personen werden regelmässig lokalisiert.
2. Die aktuelle Position wird zusammen mit vorhergehenden Positionen als Fakten in eine Rule Engine injiziert.
3. Die Rule Engine kann aus den injizierten Fakten ein Positionswechsel-Ereignis erzeugen.
4. Wurde ein Positionswechsel-Ereignis erzeugt, wird es an interessierte Systeme verschickt.

Testkriterien

- Für jeden Durchlauf produziert die Rule Engine eine Entscheidung.
- Nachdem ein Positionswechsel-Ereignis erzeugt worden ist, wurde es an alle interessierten Systeme verschickt.
- Jeder Aspekt eines Positionswechsel-Ereignis kann durch die Rule Engine modifiziert werden.

17.3.11 User Story: Alle Geräte eines Typs in einer Zone bestimmen

Der Anwender will zum Beispiel zu Abrechnungszwecken wissen, welche Geräte eines bestimmten Typs sich in einer bestimmten Zone (z.B einer Abteilung) befinden. Dazu gibt er den gewünschten Typ und die Abteilung an. Das System zeigt ihm dann die Geräte mit ihrer Bezeichnung (z.B Inventarnummer) und Position an.

Aufgaben

1. Eingabeformular erstellen.
2. Geräteidentifikationen aller Geräte des Typs bestimmen.
3. Geräte des gewählten Typs lokalisieren.
4. Geräte auswählen, welche sich in der angegebenen Zone befinden.
5. Resultatanzeige erstellen.
6. Sicherheit (für den Moment) weglassen.

Testkriterien

- Anwender erhält die Positionen aller Geräte des gesuchten Typs in der gewünschten Abteilung.

17.3.12 User Story: Position aller Personen einer Rolle bestimmen

Beschreibung

Der Anwender sucht die Position aller Personen einer bestimmten Rolle (z.B. alle Herzchirurgen). Das System liefert ihm auf seine Anfrage eine Liste aller Personen des Typs mit ihrer Position in Form von «Gebäude-Stockwerk-Raum».

Aufgaben

1. Eingabeformular erstellen.
2. Personen-Identifikation aller Personen der Rolle bestimmen.
3. Personen der gewählten Rolle lokalisieren.
4. Position in Angabe der Form «Gebäude-Stockwerk-Raum» umwandeln.
5. Resultatanzeige erstellen.
6. Sicherheit (für den Moment) weglassen.

Testkriterien

- Anwender erhält die Positionen aller Personen der gesuchten Rolle.
- Kann eine Person mit der gesuchten Rolle nicht lokalisiert werden, erhält er eine entsprechende Meldung.
- Befindet sich eine gesuchte Person nicht auf dem Campus wird eine entsprechende Meldung angezeigt, dass sich die Person ausserhalb des Campus befindet.

17.3.13 User Story: Von einer Person aus die nächste Person einer Rolle bestimmen

Beschreibung

Der Anwender sucht eine Person einer bestimmten Rolle (z.B. Herzchirurg). Dabei will er die von ihm aus nächste Person dieser Rolle finden. Damit der Benutzer selber entscheiden kann, welches der nächstliegenden Personen der Rolle für ihn relevant ist, werden ihm die nächstligenden Personen mit Positionsangabe der Form «Gebäude-Stockwerk-Raum» und der relativen Distanz geliefert. Diese sind bezüglich der Distanz aufsteigend sortiert, beginnend mit der Nächsten.

Aufgaben

1. Eingabeformular erstellen.
2. Anfragesteller lokalisieren.
3. Personen der gewählten Rolle lokalisieren.
4. Distanzen vom Anfragesteller zu den Personen bestimmen.
5. Positionen in Angabe der Form «Gebäude-Stockwerk-Raum» umwandeln.
6. Resultatanzeige, sortiert nach Distanzen erstellen.
7. Sicherheit (für den Moment) weglassen.

Testkriterien

- Anwender erhält die Personen der gewünschten Rolle welche von ihm aus am nächsten sind, zusammen mit den von ihm aus relativen Distanzangaben.
- Distanzangaben entsprechen den wirklichen Distanzen.
- Anwender erhält eine Fehlermeldung, kann keine Person der gewünschten Rolle lokalisiert werden.

17.3.14 User Story: Position einer Person bestimmen

Beschreibung

Der Anwender sucht eine Person, von der er die Identifikation kennt, beispielsweise eine Personalnummer. Um ihm zu helfen, die Person zu finden, soll das System eine Positionsangabe der Form «Gebäude-Stockwerk-Raum» liefern.

Aufgaben

1. Eingabeformular erstellen.
2. Person lokalisieren.
3. Position in Angabe der Form «Gebäude-Stockwerk-Raum» umwandeln.
4. Resultatanzeige erstellen.
5. Sicherheit (für den Moment) weglassen.

Testkriterien

- Anwender erhält Positionsangabe für auffindbare Person, deren Personalnummer er kennt.
- Anwender erhält Fehlermeldung für nicht auffindbare Person.
- Anwender erhält entsprechende Meldung, wenn eine Person sich nicht auf dem Campus befindet.

17.3.15 User Story: Menschenströme aufzeichnen

Beschreibung

Es werden in regelmässigen Intervallen die Positionen aller Personen ermittelt und abgespeichert, damit sie für eine spätere Analyse bereitstehen.

Aufgaben

1. Alle Personen werden in einstellbarem Intervall lokalisiert.
2. Die erhaltenen Positionen werden zu Analysezwecken abgespeichert, so das sie nach Gebäude, Stockwerk, Raum oder Zone gefiltert werden können.

Testkriterien

- Personen werden in einstellbarem Intervall lokalisiert.
- Nach jedem Intervalldurchgang ist eine Position für jede Person gespeichert.

17.3.16 User Story: Lokalisierungssensoren verwalten

Lokalisierungssensoren können von einem anderen System aus registriert, aktualisiert, oder gelöscht werden.

Aufgaben

1. Schnittstelle anbieten, um Lokalisierungssensor zu registrieren.
2. Schnittstelle anbieten, um Lokalisierungssensor zu aktualisieren.
3. Schnittstelle anbieten, um Lokalisierungssensor zu löschen.
4. Sicherheit ist (momentan) kein Thema
5. Validierung ist (momentan) kein Thema

Testkriterien

- Lokalisierungssensoren können registriert werden.
- Lokalisierungssensoren können aktualisiert werden.
- Lokalisierungssensoren können gelöscht werden.
- Der aktuelle Stand des Lokalisierungssensors ist in der Datenbank gespeichert.
- Etwaige Scheduler-Jobs werden erzeugt, aktualisiert bzw. gelöscht.

17.3.17 User Story: Geräte verwalten

Beschreibung

Geräte können von einem anderen System aus registriert, aktualisiert oder gelöscht werden.

Aufgaben

1. Schnittstelle anbieten, um Gerät zu registrieren.
2. Schnittstelle anbieten, um Gerät zu ändern.
3. Schnittstelle anbieten, um Gerät zu löschen.
4. Sicherheit ist (momentan) kein Thema.
5. Validierung ist (momentan) kein Thema.

Testkriterien

- Geräte können registriert werden.
- Geräte können aktualisiert werden.
- Geräte können gelöscht werden.
- Der aktuelle Stand des Geräts ist in der Datenbank gespeichert.
- Etwaige Scheduler-Jobs werden erzeugt, aktualisiert bzw. gelöscht.

17.3.18 Personen verwalten

Beschreibung

Personen können von einem anderen System aus registriert, aktualisiert oder gelöscht werden.

Aufgaben

1. Schnittstelle anbieten, um Person zu registrieren.
2. Schnittstelle anbieten, um Person zu ändern.
3. Schnittstelle anbieten, um Person zu löschen.
4. Sicherheit ist (momentan) kein Thema.
5. Validierung ist (momentan) kein Thema.

Testkriterien

- Personen können registriert werden.
- Personen können geändert werden.
- Personen können gelöscht werden.
- Der aktuelle Stand der Person ist in der Datenbank gespeichert.
- Etwaige Scheduler-Jobs werden erzeugt, aktualisiert bzw. gelöscht.

18 Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen wurden bereits in [1] erfasst. Hierbei handelt es sich um eine aktualisierte Version.

18.1 Funktionalität

18.1.1 Logische Organisation

Die kleinste lokalisierbare Position ist ein Sektor (Raum, Bereich in einem Korridor...), der sich auf einem bestimmten Stockwerk in einem bestimmten Gebäude befindet (stockwerk- beziehungsweise gebäudeübergreifende Sektoren sind nicht möglich). Ein Sektor hat die Form eines Polygons. Multiple Ringe (d.h. Polygone mit Aussparungen) werden unterstützt. Die Grösse des Polygons ist frei wählbar (d.h. es gibt kein Minimum und kein Maximum). Eine beliebige Anzahl Sektoren kann zu einer Zone zusammengefasst werden, die sich über mehrere Gebäude erstrecken kann.

18.1.2 Schnittstellen

Lokalisierungsabfrage

Zur Abfrage von aktuellen Positionen (Funktionalität analog Benutzerschnittstelle) soll eine Webservice-Schnittstelle geboten werden, mit der sich beispielsweise native Smartphone-Apps anbinden lassen.

Presence Manager

Von Positionen abgeleitete Status sollen dem Presence Manager als Eingabe dienen können. Die Status-Nachrichten sollen per JMS über einen Message Broker in Form eines Topics bereitgestellt werden. Beim Presence Manager wird bereits Apache ActiveMQ verwendet, das auch von der Lokalisierung verwendet werden soll.

Sensor-, Geräte- und Personenverwaltung

Zur Sensor-, Geräte- und Personenverwaltung wird eine REST-Schnittstelle angeboten. Die Verbindung mit vorhandenen Back-end-Systemen wie Inventar wird über ein separat zu realisierendes Zwischenstück realisiert, damit die Funktionalität in der Lokalisierungskomponente nicht für jede Installation angepasst werden muss.

18.1.3 Fehlerbehandlung, Logging

Alle Arten von Fehlern, die von Interesse sind, werden in einer zentralen Logdatei gespeichert.

18.2 Usability

Das es sich beim Projekt um einen Technologiedemonstrator handelt, wird für die Gestaltung der Benutzerschnittstellen auf einen User-Centered-Design-Prozess verzichtet. Es ist ausreichend, wenn sich die Funktionalität verwenden lässt.

18.3 Zuverlässigkeit

18.3.1 Anforderungen

Programmcode

- Operationen müssen ganz oder gar nicht ausgeführt werden.
- Operationen, die den Datenspeicher modifizieren, müssen diesen in einem konsistenten Zustand hinterlassen (keine Karteileichen...).
- Operationen müssen nach sich selber aufräumen (keine temporären Dateien hinterlassen...)

18.3.2 Fehlerhäufigkeit

Objektlokalisierung Die Objektlokalisierung muss in 95 Prozent der Fälle erfolgreich sein, sofern mindestens ein verwendbares Lokalisierungswerkzeug eine verwertbare Antwort liefert.

Distanzmessung Die Distanzmessung muss in 95 Prozent aller Fälle das nächstliegende Gerät ermitteln, bezogen auf die von den Lokalisierungswerkzeugen gemeldeten Standorte.

Ereignismeldung Eine Ereignismeldung muss in 95 Prozent aller Fälle korrekt erkannt und weiterverteilt werden, sofern die Lokalisierungswerkzeuge korrekte Informationen liefern.

Statuserkennung Eine Statuserkennung muss bezüglich der Kenntnisse der Lokalisierungslogik in 80 Prozent der Fälle korrekt sein.

18.3.3 Fehlerbehandlung

Laufzeitfehler

Bei einem Laufzeitfehler wird die laufende Aktion abgebrochen und die im Rahmen der Operation getätigten Aktionen (z.B. Erzeugen von Dateien, Modifikation des Datenspeichers) rückgängig gemacht. Der Benutzer wird über das Problem und, soweit möglich, dessen Ursache informiert.

Systemabsturz

Der Benutzer wird, sofern noch möglich, darüber informiert, dass das System nicht mehr zur Verfügung steht. Die Wiederinbetriebnahme erfordert eine manuelle Intervention.

18.3.4 Fehlervorbeugung

Benutzereingaben

Benutzereingaben werden, sofern relevant, auf formale und inhaltliche Gültigkeit hin überprüft.

18.4 Effizienz

18.4.1 Antwortverhalten

Aktionen müssen vom Benutzer terminierbar sein.

18.4.2 Antwortzeiten

Die Lokalisierungslogik muss unter Berücksichtigung der Hardware-Anforderungen und der zu erwartenden Datenmengen folgendes Antwortverhalten aufweisen:

Lokalisierung eines Objekts	< 1 s
Lokalisierung des nächsten Objekts	< 2 s

Die Zeiten beziehen sich rein auf die Verarbeitungszeit innerhalb der Lokalisierungslogik. Die Zeit, welche die Befragung der Lokalisierungswerkzeuge in Anspruch nimmt, wird dabei nicht berücksichtigt.

18.4.3 Leistungsvermögen

Die Lokalisierungslogik muss unter Einhaltung der geforderten Antwortzeiten mit folgenden Datenmengen umgehen:

Anzahl lokalisierbare Objekte	< 40’000
Anzahl Sektoren	< 20’000

18.5 Änderbarkeit

18.5.1 Adaptionsfähigkeit

Es wurden folgende Bereiche identifiziert, die wechselnden Anforderungen unterworfen sind:

Bezugssysteme Da die Software potentiell in verschiedenen Ländern zum Einsatz kommen können soll, muss sie sich auf andere geographische Bezugssysteme umstellen lassen.

Anbindung von Lokalisierungswerkzeugen Es muss eine beliebige Anzahl von Lokalisierungswerkzeugen unterstützt werden.

Anbindung von Back-end-Systemen Je nach Umgebung kommen unterschiedliche Back-end-Systeme wie Inventardatenbanken zum Einsatz (SAP, Oracle…), die geeignet angebunden werden müssen.

Um diesen Anforderungen entgegenzukommen und die langfristige Entwicklung zu erleichtern, werden für die Anbindung von Lokaliserungswerkzeugen Plug-in-APIs erstellt, sodass sich variierende Funktionalität verhältnismässig einfach nachrüsten lässt. Die Flexibilität hinsichtlich Anbindung von Back-end-Systemen wird über separat zu realisierende Komponenten bewerkstelligt, für die eine geeignete Schnittstelle bereitgestellt wird.

18.5.2 Konfigurierbarkeit

- Für Installationsabhängige Einstellungen wie Verbindungsparameter werden Konfigurationsdateien bereitgestellt, mit deren Hilfe sich die Einstellungen ohne Modifikation eines JAR oder Web Archive (WAR) ändern lassen.
- Business Rules werden über eine geeignete Rule Engine integriert.

18.6 Implementierung

Funktionalität wird, sofern möglich, mit Hilfe bestehender Open-Source-Komponenten realisiert. Dies, um Vendor Lock-in zu vermeiden und fehlende Funktionalität problemlos selber nachrüsten beziehungsweise die Komponenten notfalls selber warten zu können.

18.6.1 Plattform

Serverseitig

Zwecks Plattformunabhängigkeit, weil es Open Source ist und weil die meisten Software Development Kit (SDK) in Java zur Verfügung stehen, werden serverseitige Komponenten in Java implementiert. Weil keine Rücksicht auf Legacy-Systeme genommen werden muss, wird auf die aktuellen Versionen Java 6 SE und Java 6 EE gesetzt.

Benutzerschnittstelle

Die Benutzerschnittstelle wird zur Abdeckung möglichst vieler Gerätekategorien in Form eines Webinterface realisiert, wobei HTML 5 zum Einsatz kommt, um von Funktionen wie Local Storage profitieren zu können.

18.6.2 Komponenten

Bezüglich Komponenten-Auswahl bestehen keine spezifischen Vorgaben.

18.6.3 Schnittstellen

Für Request-Reply-Szenarien werden Webservice-Schnittstellen, vorzugsweise im REST-Architekturstil, realisiert. Für Producer-Consumer-Szenarien wird JMS verwendet.

18.6.4 Formate

Allgemein

Webservice-Schnittstellen Als Payload wird grundsätzlich sowohl JSON als auch Extended Markup Language (XML) unterstützt. Das präferierte Format ist dabei JSON, weil es kompakter und beispielsweise im Webbrowser direkt auszuführen ist. XML dient als Reserve für Systeme, die mit JSON nicht umgehen können.

JMS Nachrichten werden als `TextMessage` mit einer Payload in Form von JSON oder XML versendet, um sicherzustellen, dass auch Nicht-Java-Plattformen die Nachrichten verarbeiten können. Das präferierte Format ist dabei JSON, weil es kompakter und beispielsweise im Webbrowser direkt auszuführen ist. XML dient als Reserve für Systeme, die mit JSON nicht umgehen können.

Presence Manager

Für die Kommunikation mit dem Presence Manager ist bereits ein Nachrichtenformat definiert. Zu verwenden ist Version 1.0 vom 22. April 2011.

18.7 Unterstützte Hardware und Software

18.7.1 Hardware

Serverseitig

Die Lokalisierungslogik unterstützt grundsätzlich beliebige Systeme, auf denen sich eine Java Virtual Machine (JVM) und alle anderen benötigten Komponenten ausführen lassen.

Benutzerschnittstelle

Als Benutzerschnittstelle kommt ein Webinterface zum Einsatz, das sich auf einer möglichst grossen Auswahl von Geräten angenehm bedienen lassen sollte. Als Referenzplattform wurde ein Apple iPad Wi-Fi aus dem Frühjahr 2010 bestimmt, da es verhältnismässig leistungsschwach ist und mit WebKit über einen Browser verfügt, der auf vielen Mobilgeräten zum Einsatz kommt:

Bildschirmauflösung	1024 x 768
CPU	1 GHz Apple A4 (ARM Cortex A8)
RAM	256 MB

18.7.2 Software

Serverseitig

Betriebssysteme Die Software sollte sich auf jedem Betriebssystem ausführen lassen, auf dem eine JVM vorhanden ist und sich alle anderen benötigten Komponenten ausführen lassen. Als Referenzsystem wurde Ubuntu Linux 10.04 LTS in der Server Edition bestimmt.

Laufzeitumgebung Java Hotspot VM für Java 6.

Benutzerschnittstelle

Browser Es werden folgende Browser unterstützt:

- Internet Explorer 8.0
- Firefox 3.5
- Mobile Safari aus iOS 4.0

18.8 Dokumentation

18.8.1 Code-Dokumentation

Zur Dokumentation der Interfaces der serverseitigen Komponenten wird eine JavaDoc-Dokumentation erstellt. Zusätzlich wird ein Software-Architektur-Dokument erstellt, das die Grundzüge der Architektur dokumentiert.

18.8.2 Schnittstellendokumentation

Es werden geeignete Dokumentationen samt Formatbeschreibungen für die realisierten Schnittstellen erstellt.

18.8.3 Installationsanleitung

Es wird eine Installationsanleitung zur Inbetriebnahme der Software erstellt.

18.9 Auslieferung

18.9.1 Packaging

Die Anwendung wird als self-contained WAR samt aller Abhängigkeiten ausgeliefert. SQL-Scripts zur Initialisierung der Datenbanken werden separat mitgeliefert.

18.9.2 Deployment

Das Deployment erfolgt manuell. Es wird eine Installationsanleitung erstellt, um das Deployment zu unterstützen.

18.10 Rechtliche Aspekte

18.10.1 Software-Lizenz

Die Software ist Teil eines grösseren proprietären Gesamtsystems.

18.10.2 Fremdkomponenten

Bestehende Fremdkomponenten (d.h. Code, Artwork usw.) dürfen nur verwendet werden, sofern die Lizenzen erlauben, das Objekt zu verwenden oder zu modifizieren, ohne dass die Software oder die Modifikationen veröffentlicht werden müssen und keine Beschränkungen bezüglich Vertrieb oder Distribution auferlegt werden.

Open-Source-Lizenzen

Code, Komponenten oder Artwork, die unter einer Open-Source-Lizenz[1] stehen, dürfen nur verwendet werden, sofern sie die oben genannten Bedingungen erfüllen und deren Lizenzen keine Werbeklausel beinhalten. Unbedenklich sind folgende Lizenzen:

- Apache License 2.0
- BSD License (3-clause oder simplified)
- GNU Lesser General Public License
- MIT License

Folgende Lizenzen sind zu vermeiden:

- GNU Free Documentation License
- GNU General Public License (auch Affero-Varianten)
- Open Software License

Creative-Commons-Lizenzen

Code, Komponenten oder Artwork, die unter Creative-Commons-Lizenzen stehen, dürfen nur verwendet werden, sofern sie die oben genannten Bedingungen erfüllen und deren Lizenzen keine Werbeklausel beinhalten. Unbedenklich sind folgende Lizenzen:

- Copyright-Only Dedication or Public Domain Certification
- CC0

Public Domain

Fremdkomponenten, die gemeinfrei sind (Public Domain), können unbeschränkt verwendet werden.

[1] siehe `http://opensource.org/licenses/alphabetical` für Liste der durch Open Source Initiative (OSI) anerkannten Lizenzen

19 Domänenanalyse

Die Domain der Lokalisierung besteht von der Funktionalität her gesehen aus zwei Teilen: Einerseits aus dem Teil, der für Geometrie und Topologie zuständig ist, und andererseits aus dem Teil, der sich um die Lokalisierung kümmert. Der Teil Geometrie und Topologie ist im Abschnitt 13 (Seite 69) ausführlich beschrieben, da er integraler Bestandteil des Shapeconverters ist und im Rahmen der Lokalisierung nur «konsumiert» wird.

Eine erste Version der Domain wurde bereits in [1] entwickelt. Die vorgestellte Variante wurde im Laufe des Projekts anhand der neuen Erfordernisse angepasst und erweitert.

19.1 Strukturdiagramm Lokalisierungsmodell

Abbildung 19.1 (Seite 138) zeigt den Lokalisierungsteil der Domäne.

19.2 Klassen Lokalisierungsmodell

19.2.1 DeviceState

Status, den das `Device` im Moment einnimmt. Spezialisierungen repräsentieren den konkreten, aktuellen Status.

19.2.2 LocatableObject

Generisches zu lokalisierendes Objekt.

Attribute

identifier Identifiziert das Objekt, beispielsweise eine Inventarnummer.

localisationInterval Bestimmt, wie häufig die Position des `LocatableObject` bestimmt werden soll.

Device

Lokalisierbares Gerät. Seine Position wird über * an ihm befestigte `Sensors` bestimmt.

19.2.3 ObjectPosition

Position, die ein `LocatableObject` zum angegebenen Zeitpunkt in der Vergangenheit eingenommen hat. Dient zu Historien-Zwecken.

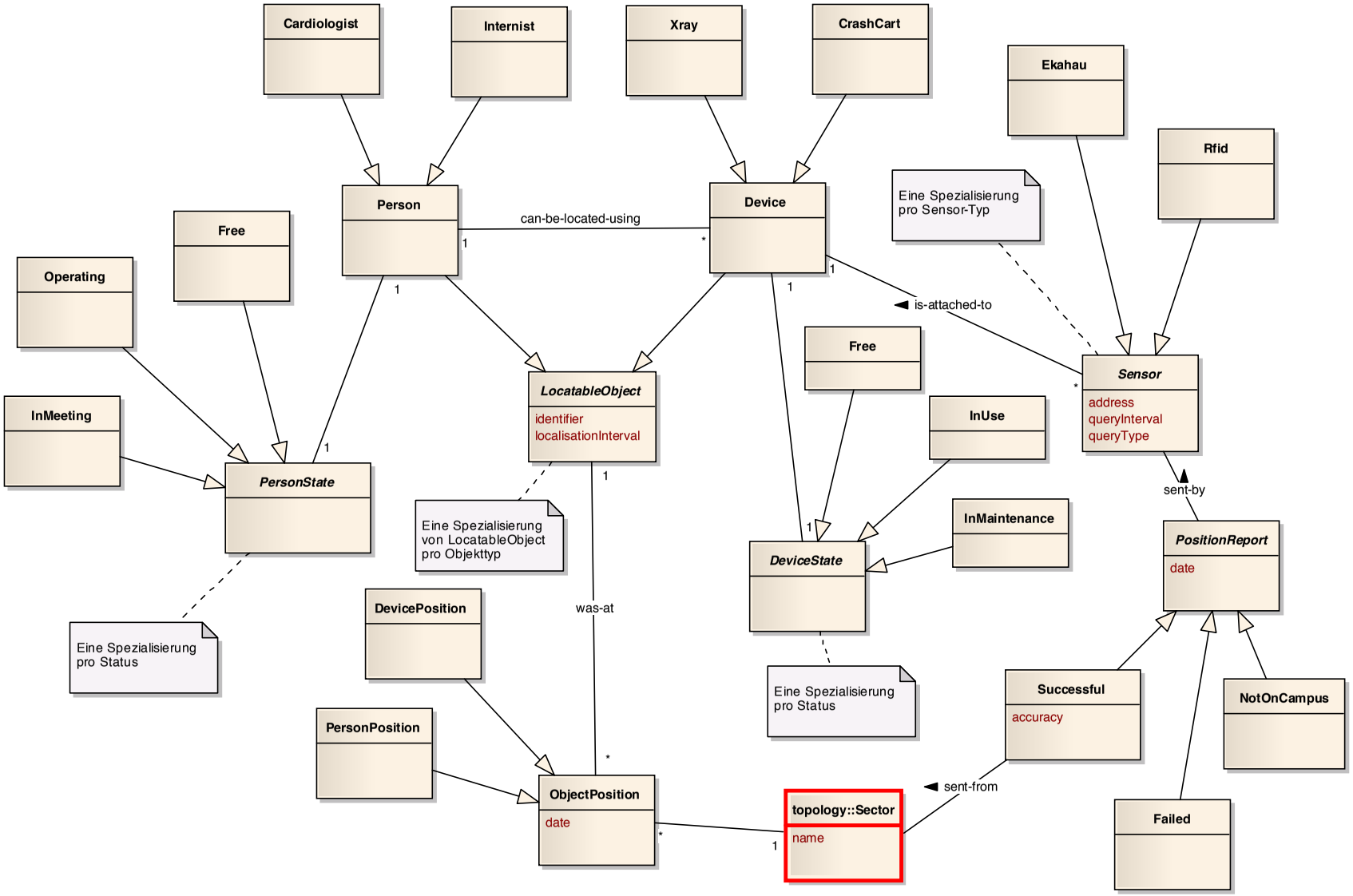

Abbildung 19.1: Domänenmodell der Lokalisierung. Das Objekt `Sector` (rot hervorgehoben) stellt den Übergang zum Teil der Geometrie und Topologie (Abschnitt 13, Seite 69) dar.

Attribute

date Datum, zu dem sich das `LocatableObject` an der Position befunden hat.

Person

Lokalisierbare Person. Ihre Position wird über `* Devices` bestimmt, die sie auf sich trägt.

19.2.4 PersonState

Status, den die `Person` im Moment einnimmt. Spezialisierungen bestimmen den konkreten Status, den die Person aktuell einnimmt.

19.2.5 PositionReport

Meldung über das Resultat eines Lokalisierungsvorgangs, in dessen Rahmen versucht wurde, einen `Sensor` zu lokalisieren.

Attribute

date Datum, zu dem der Lokalisierungsvorgang stattgefunden hat.

Failed

Meldung über einen fehlgeschlagenen Lokalisierungsvorgang. Wird beispielsweise erzeugt, wenn die aktuelle Position nicht festgestellt werden kann, z.B. weil bei Satellitenortung nicht genügend Satelliten in Sichtweite sind.

NotOnCampus

Meldung über einen erfolgreich durchgeführten Lokalisierungsvorgang, bei dem aber festgestellt wurde, dass sich der Sensor ausserhalb des `Campus` befindet. Aus Datenschutzgründen wird die genaue Position dann unterdrückt.

Successful

Meldung über einen erfolgreich durchgeführten Lokalisierungsvorgang. Attribute:

accuracy Genauigkeit der gemeldeten Position

19.2.6 Sensor

Symbolisiert einen Lokalisierungssensor, der an einem Gerät befestigt ist, um es lokalisieren zu können. Dies kann zum Beispiel ein Ekahau-Tag oder eine RFID-Etikette sein. Spezialisierungen bestimmen den Typ des Sensors.

Attribute

address Adresse des Lokalisierungssensors, mit dem er vom zugehörigen Lokalisierungswerkzeug angesprochen wird. Kann zum Beispiel eine MAC-Adresse sein.

queryInterval Abstand in Sekunden, der zwischen zwei Lokalisierungsvorgängen verstreichen soll beziehungsweise Zeit, die verstreichen darf, bis spätestens wieder eine Position eingeliefert werden muss.

queryType Art, wie die Position des Lokalisierungssensors bestimmt werden kann, zum Beispiel «Push» oder «Pull».

19.3 Konzepte

Die nächsten Abschnitte erläutern einige Überlegungen zum Domain-Modell, die nicht auf den ersten Blick oder gar nicht aus dem Modell ersichtlich sind sowie etwaige Probleme und Verbesserungsvorschläge.

19.3.1 Geräte- und Personenspeicherung

`Person` und `Device` verfügen über sehr wenige Attribute und dazu erst noch nicht über die, welche man erwarten würde, beispielsweise den Namen bei einer Person. Grund ist, dass die Lokalisierung nicht als Personen- oder Geräteverzeichnis dient. Statt dessen werden nur die unmittelbar für die Lokalisierung benötigten Daten gespeichert. Werden weitere Informationen benötigt, sollen diese bei den dafür zuständigen Systemen beschafft werden, damit kein unnötiger Synchronisationsaufwand anfällt.

19.3.2 Geräte- und Personenlokalisierung

Für die Geräte und Personenlokalisierung im Spitalumfeld ist es wichtig, die zu lokalisierenden Objekte über mehrere Sensoren lokalisieren zu können, damit im Falle eines Ausfalls oder wenn einer keine Position bestimmen kann, immer noch auf die anderen Sensoren zurückgegriffen werden kann, die hoffentlich funktionieren. Deshalb wird eine «indirekte» Lokalisierung verwendet: Aus den Positionen der verschiedenen Sensoren wird die Position des Geräts bestimmt, mit dem sie verbunden sind. In der Domäne wird diesem Umstand durch die die Multiplizität `*` zwischen `Device` und `Sensor` Rechnung getragen.

Bei den Personen könnte nun analog vorgegangen werden, also aus einer beliebigen Menge von Sensorpositionen die Position der Person bestimmt werden. Diese Lösung haben wir im Rahmen der Studienarbeit realisiert, weshalb die Relation von `Person` und `Device` zu `Sensor` (damals noch `Address`) in [1] zusammengefasst wurde. Das Resultat zeigt Abbildung 19.2 (Seite 141). Das Problem bei dieser Lösung ist, dass Personen eher selten einen RFID- oder Ekahau-Tag auf sich tragen und statt dessen über die Sensoren der Geräte lokalisiert werden, die sie bei sich haben. Dieses Szenario würde bei der damaligen Domain etliche Probleme mit sich bringen:

- Geräte, die die Person auf sich trägt, könnten nicht individuell lokalisiert werden (z.B. wenn sie abgelegt worden sind), weil nicht sie, sondern nur ihre Sensoren bekannt sind.

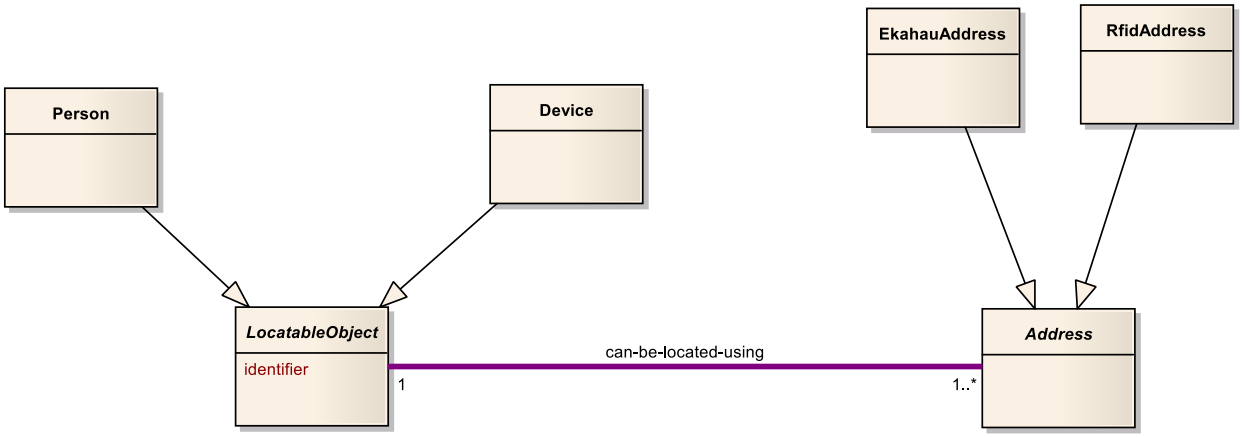

Abbildung 19.2: Das Bild zeigt die während der Studienarbeit bestehende Relation (violett hervorgehoben) zwischen `LocatableObject` und `Address` (heute `Sensor`). In Abbildung 19.3 (Seite 141) ist der aktuelle, verbesserte Stand zu sehen.

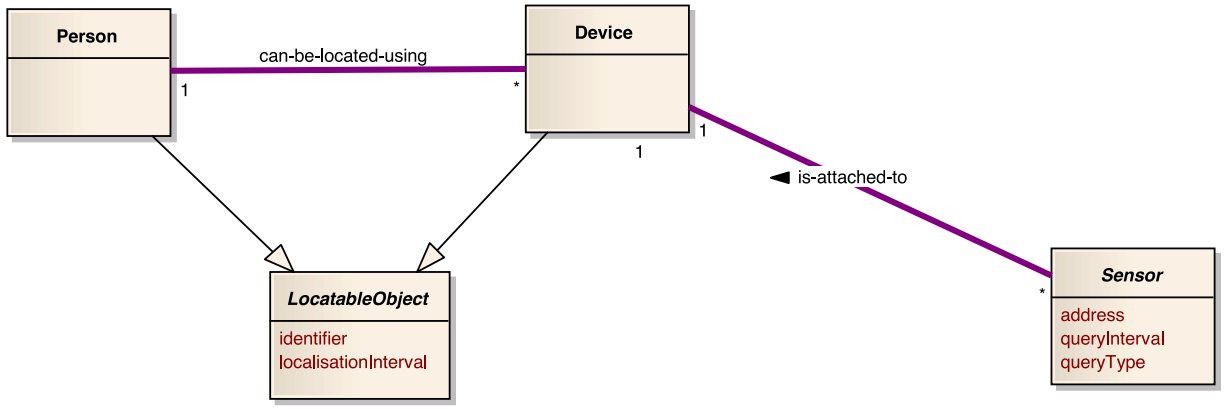

Abbildung 19.3: Das Bild zeigt die aktuell bestehende Relation (violett hervorgehoben) zwischen `LocatableObject` und `Sensor`. Die Personen werden nun über die Geräte lokalisiert, die sie auf sich tragen, und die Geräte über die an ihnen angebrachten Sensoren.

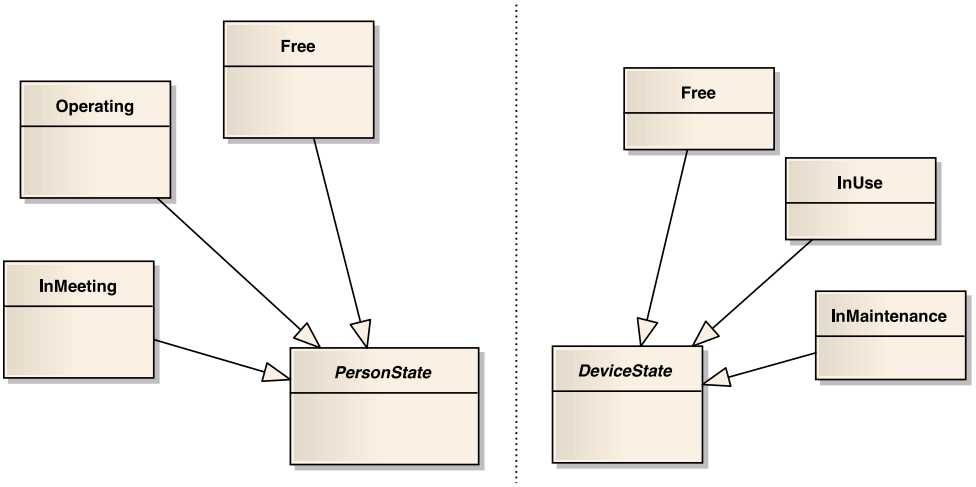

Abbildung 19.4: Zustandsmodellierung für Personen (links) und Geräte (rechts).

- Will man die Geräte doch lokalisieren, müssten sie ein zweites Mal als Gerät registriert werden, was dazu führen würde, dass die Lokalisierung desselben Geräts doppelt ausgeführt würde: Einmal für die Person und einmal für das Gerät.

Um dieses Problem zu umgehen, wurde die Domain so modifiziert, dass nun mehrstufig lokalisiert wird: Zuerst werden die Positionen der Sensoren festgestellt. Daraus wird dann die Position eines Geräts extrahiert und aus den Gerätepositionen dann die Positionen der Personen. Das Resultat zeigt Abbildung 19.3 (Seite 141).

Bleibt nun die Frage, was passiert, wenn man eine Person dennoch über einzelne Sensoren lokalisieren möchte? Eine Variante wäre gewesen, eine zusätzliche Relation zwischen `Person` und `Sensor` einzuführen. Dies hätte den gesamten Lokalisierungsprozess aber unnötig verkompliziert, weshalb darauf verzichtet wurde. Einen brauchbaren Kompromiss aus Flexibilität und einfachem Lokalisierungsprozess stellt die realisierte Variante (Abbildung 19.3, Seite 141) dar, indem man einen einzelnen Sensor einfach als separates Gerät betrachtet, wofür man bei Sendern wie einem Ekahau-WLAN-Tag nicht einmal ein Auge zudrücken muss.

19.3.3 Geräte- und Personenstatus

Von der Position eines Geräts oder einer Person kann grundsätzlich auch ein Status abgeleitet werden. Eine Person in der Kantine ist mit grosser Wahrscheinlichkeit «in der Pause» während ein Gerät im Lager «frei» ist. Dies wird durch die Objekte `PersonState` und `DeviceState` symbolisiert, wobei die konkreten Status durch die Spezialisierungen repräsentiert werden (siehe Abbildung 19.4 auf Seite 142). Durch die Trennung nach «Device» und «Person» wurde eine bereits in [1] (Seite 63) skizzierte Lösung realisiert, da nicht nur Zustände wie «frei» und «besetzt» gefragt sind, die für beide Objektklassen verwendet werden können, sondern auch spezifischere wie «in Wartung», was ein eigenartiger Zustand für einen Arzt wäre.

Nach wie vor bestehen aber Zweifel, ob dies eine gute Lösung ist. So könnte es sinnvoller sein, `DeviceState` respektive `PersonState` mit `ObjectPosition` zu verbinden, um nicht nur den aktu-

ellen Status zu kennen, sondern auch vergangene Zustände. Dies wäre auch logisch sinnvoller, da das Gerät den aktuellen Status nur inne hatte, als es die letzte `ObjectPosition` eingenommen hat, inzwischen bereits aber wieder ganz woanders sein und damit einen anderen Status aufweisen könnte. Schlussendlich dürfte es auch die Speicherung im System vereinfachen, da die `ObjectPositions` ohnehin für häufige Schreibaktivität ausgerichtet sein müssen, die restlichen Objekte abgesehen vom Status aber selten mutiert werden. Es dürfte sich also anbieten, dies bei Gelegenheit zu ändern und `DeviceState` an `DevicePosition`, die `ObjectPosition` spezialisiert, anzuhängen und analog mit `PersonPosition` vorzugehen.

Ebenfalls ist zu hinterfragen, ob die Lokalisierung überhaupt einen Status speichern sollte, da die Lokalisierung nur über Positionswissen verfügt und damit die Statusangaben von keiner guten Qualität sind – man denke nur an eine Sitzung in der Kantine. Was für Geräte noch akzeptabel sein kann, ist für Personen mehr als fraglich, da für sie oftmals weitere Statusindikatoren wie der Status des persönlichen Telefons oder der Kalender vorhanden sind. Wir empfehlen daher, auf die Speicherung des Status komplett zu verzichten und diesen statt dessen von einem besser informierten System zu beziehen, beispielsweise dem Presence Manager.

19.3.4 Vollständige Nachvollziehbarkeit

Das Thema vollständige Nachvollziehbarkeit ist in der Domain noch nicht berücksichtigt, da dies im Projekt nicht gefordert war und eine detaillierte Analyse voraussetzt, welche Anforderungen auch seitens des Gesetzgebers bestehen.

20 Software-Architektur

Dieses Kapitel beschreibt die Architektur der serverseitigen Komponenten. Als Grundlage diente die Architekturbeschreibung aus [1], die aufgrund der geänderten Anforderungen erweitert und angepasst wurde.

20.1 Architektur

20.1.1 Architekturübersicht

Das Lokalisierungssystem ist Teil eines grösseren Spitalinformationssystems. Seine Aufgabe besteht darin, Lokalisierungswerkzeuge unterschiedlicher Hersteller zu integrieren und die von ihnen gelieferten Positionsangaben intelligent zu kombinieren. Mit Hilfe dieses Wissens sollen verschiedenste Aufgaben realisiert werden, beispielsweise:

- Benutzeranfragen nach dem nächstgelegenen Gerät oder Person eines Typs beantworten.
- Status eines Geräts oder einer Person aus ihrer aktuellen Position ableiten und andere Systeme darüber informieren.
- Analysemöglichkeiten bieten, beispielsweise zur Auslastung von Geräten.
- Anhand der Position prüfen, ob ein Patient die notwendigen Handlungsschritte durchlaufen hat.

Abbildung 20.1 (Seite 146) illustriert die Grundzüge des physischen und logischen Aufbaus des Lokalisierungssystems und wie es in Bezug zu anderen Systemen steht.

Das Lokalisierungssystem besteht aus mehreren Einzelkomponenten, die sich je um einen kleinen Teil des Aufgabenbereichs kümmern. Die Hauptkomponenten (im Uhrzeigersinn):

Orakel Stellt die Schnittstellen zur Abfrage von «Datenhalde» und «Bürokrat» zur Verfügung. Es ermöglicht beispielsweise, die Position eines Geräts oder einer Person zu erfragen. Anfragen, die Anwender über die webbasierte Benutzerschnittstelle stellen, werden vom Orakel beantwortet. Bei Bedarf lassen sich weitere Benutzerschnittstellen wie Smartphone-Apps anschliessen. Beim «Orakel» handelt es sich um eine Java-Anwendung, die in einem Servlet-Container ausgeführt wird. Der Name des Java-Package des «Orakels» lautet `ch.hsr.ins.delphi`.

Bürokrat Der «Katalog» des gesamten Lokalisierungssystems und die Autorität über das gespeicherte Wissen. Gespeicherte Informationen sind beispielsweise die digitalisierte Geometrie und Topologie und die Verzeichnisse der Sensoren, Geräte und Personen. Zusätzlich speichert er die Lokalisierungsjobs. Konkret handelt es sich um eine PostgreSQL-Datenbank mit PostGIS-Erweiterung, über die mittels JDBC interagiert wird.

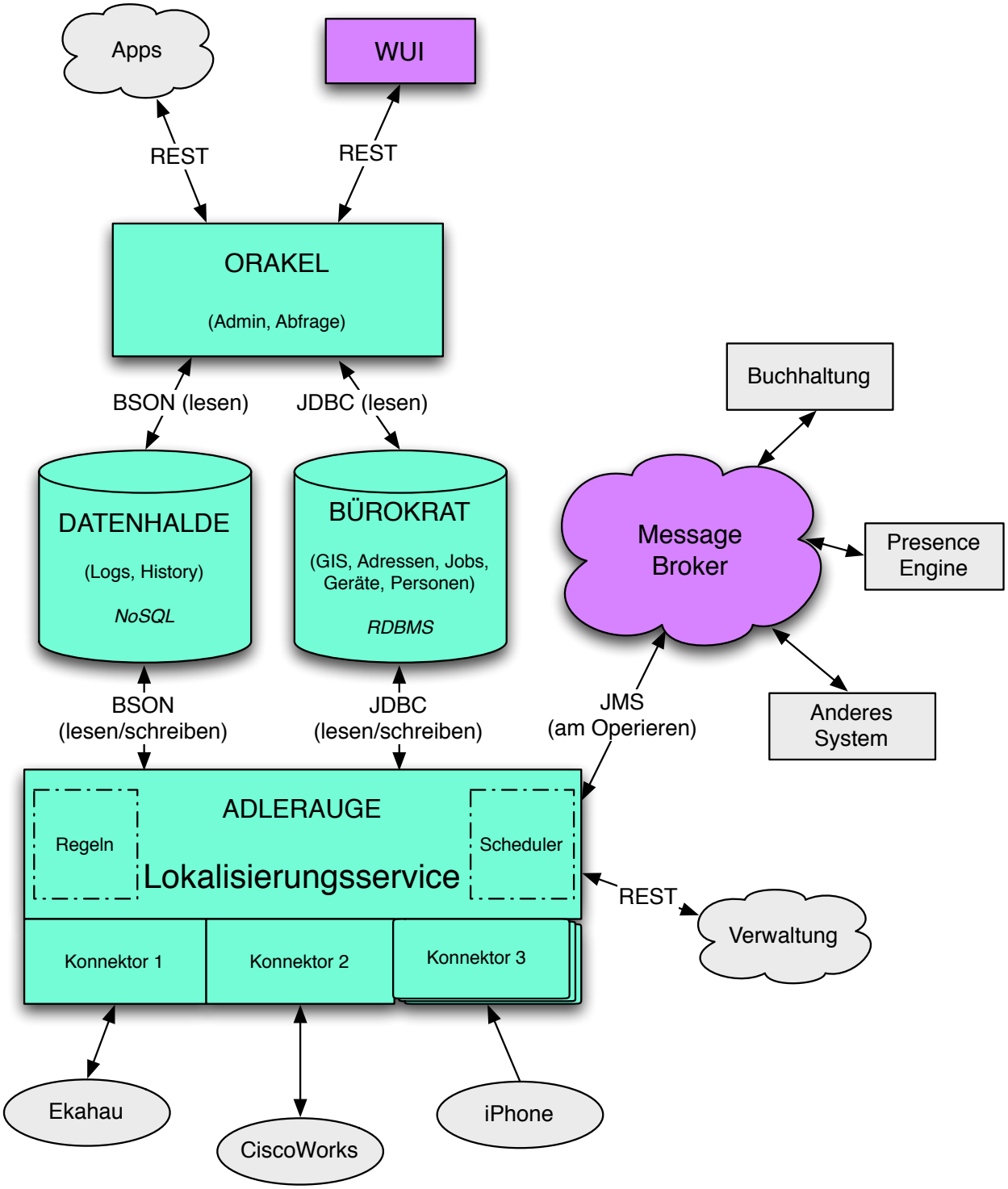

Abbildung 20.1: Übersicht über die serverseitigen Komponenten. Die Hauptkomponenten sind violett hervorgehoben, die Hilfssysteme blau. Hellblau sind externe Systeme.

Adlerauge Der eigentliche Lokalisierungsservice und damit das Herzstück des gesamten Lokalisierungssystems. Seine Aufgabe besteht darin, die Sensoren zu lokalisieren und daraus dann die Positionen von Geräten und Personen zu destillieren sowie Ereignisse auszulösen, wenn bestimmte Umstände eintreten. Welches die Umstände sind, die eintreten müssen und wie die Positionen von Geräten und Personen gewonnen werden, kann mit Hilfe von Business Rules bestimmt werden. Für die Anbindung der unterschiedlichen Lokalisierungswerkzeuge, mit deren Hilfe die Sensoren geortet werden, sind mehrere Konnektoren integriert, die mit den Lokalisierungswerkzeugen kommunizieren und dem Lokalisierungsservice die Positionsinformationen in einer einheitlichen Form liefern. Zusätzlich bietet das «Adlerauge» eine Schnittstelle zur Registrierung respektive Verwaltung von Sensoren, Geräten und Personen an. Beim «Adlerauge» handelt es sich um eine Java-Anwendung, die in einem Servlet-Container ausgeführt wird. Sie enthält zusätzlich noch eine Rule Engine und einen Scheduler. Das Java-Package des «Adlerauges» ist `ch.hsr.ins.eagleeye`.

Datenhalde Die Datenhalde speichert alle vom Lokalisierungsservice gelieferten Positionsdaten und stellt sie dem «Orakel» zur Abfrage und Analyse bereit. Ein weiterer Anwendungszweck ist die Speicherung von Logdaten, um beispielsweise eine lückenlose Nachvollziehbarkeit zu ermöglichen. Konkret handelt es sich um die NoSQL-Datenbank MongoDB, über die mit dem BSON-Protokoll interagiert wird.

Neben den Hauptkomponenten existiert noch eine Reihe von Hilfssystemen (im Uhrzeigersinn):

Message Broker Dient zur Weiterverteilung beziehungsweise Einlieferung von Nachrichten an und von anderen Systemen. Aktuell wird er nur verwendet, um vom «Adlerauge» erzeugte Ereignisse an alle interessierten Systeme weiterzuleiten. Beim Message Broker handelt es sich um einen JMS Message Broker, der vom Lokalisierungssystem gestellt werden kann. Alternativ kann ein bereits vorhandener Message Broker verwendet werden.

WUI Das «WUI» (kurz für Web User Interface) ist die Webbenutzerschnittstelle, über die Anwender ihre Anfragen ans «Orakel» tätigen können.

Diese Verteilung der Zuständigkeiten auf mehrere Anwendungen wurde gewählt, damit diese nicht zu komplex werden, sich einfacher warten und besser skalieren lassen. Ein Beispiel für die Vorteile einer solchen Lösung ist das «Orakel»: Weil es nur Daten liest und zustandslos ist, lässt es sich bei Bedarf beliebig duplizieren, um mehr Benutzeranfragen befriedigen zu können oder im Fall von Updates oder Server-Ausfällen das System weiterbetreiben zu können. Mehr dazu im Abschnitt 20.1.3 (Seite 150).

20.1.2 Änderungen zur Studienarbeit

Im Vergleich zu Abbildung 20.2 (Seite 148), die aus [1] stammt und den Architekturüberblick aus der Studienarbeit zeigt, scheint sich auf den ersten Blick viel geändert zu haben – nur schon aufgrund der zusätzlichen Services. Allerdings wurde die bestehende Architektur nur erweitert, wie Abbildung 20.3 (Seite 149) zeigt.

Die Konnektor-Anbindung, die früher im «Orakel» enthalten war, wurde herausgelöst und durch einen separaten Service ersetzt, der zwischen der Abfrage und den Konnektoren sitzt. Für diesen Schritt gab es gleich mehrere Gründe:

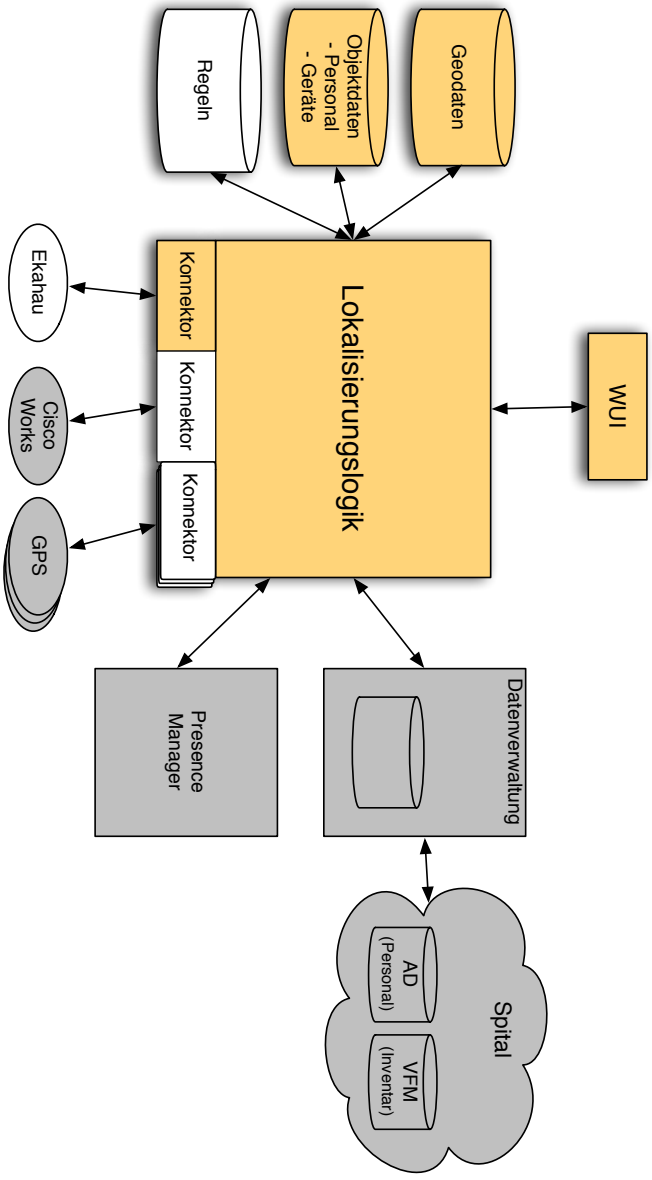

Abbildung 20.2: Übersicht über Systemarchitektur aus der Studienarbeit.

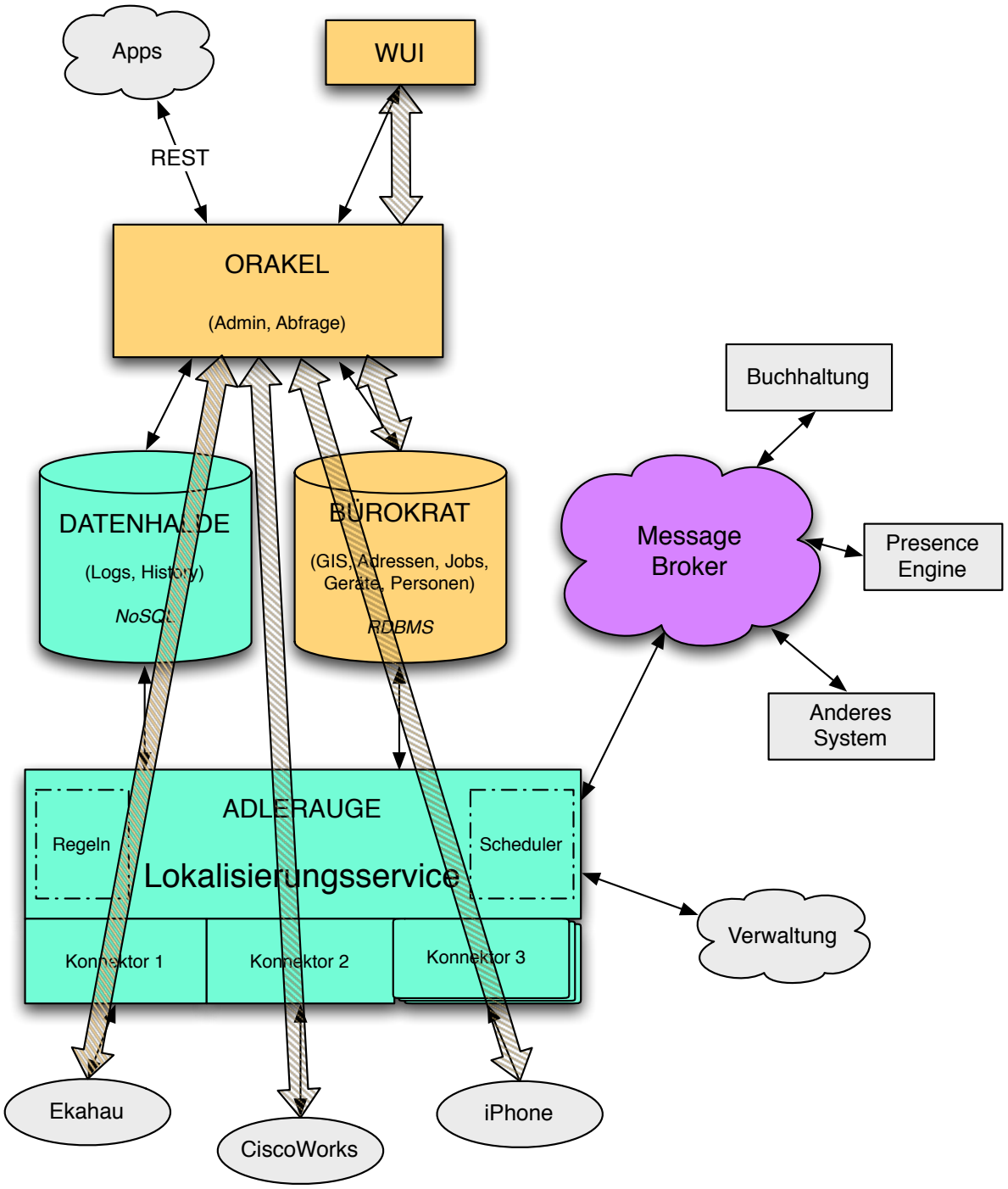

Abbildung 20.3: Die Abbildung zeigt, wie aus der Architektur der Studienarbeit (Abbildung 20.2, Seite 148) die aktuelle Architektur hervorgegangen ist. Der orange Teil entspricht der Studienarbeit. Die Konnektoren zur Interaktion mit den Lokalisierungswerkzeugen wurden dort herausgebrochen und in den neu hinzugekommenen Lokalisierungsservice integriert.

Komplexität Der ganze Lokalisierungsvorgang ist relativ komplex. Zuerst müssen die Sensoren geortet und daraus dann die Positionen der mit ihnen verbundenen Geräte bestimmt werden. Dann werden aus den Positionen der Geräte die Positionen der Personen interpoliert, die sie auf sich tragen. Zusätzlich sollten noch etwaige Ereignisse ausgelöst werden, wobei bei jedem Schritt Business Rules abgearbeitet werden müssen.

Ressourcenverbrauch Es müssen mehrere 10'000 Sensoren mit verschiedenen Lokalisierungswerkzeugen geortet werden können. Dass dies nicht seriell gemacht werden kann, versteht sich von selbst. Die nötige Parallelisierung belegt aber Threads und Arbeitsspeicher, die dann für andere Funktionen nicht zur Verfügung stehen und von einem einzelnen Computer vielleicht gar nicht bereitgestellt werden können.

Inversion of Control Anforderungen wie die Ereignisauslösung oder die Bereitstellung von Daten zur Abrechnung bedingen eine kontinuierliche Ortung der Geräte und Personen. Das bedeutet, dass nicht mehr wie in [1] auf eine Benutzereingabe gewartet werden kann, um die gewünschten Geräte zu lokalisieren. Statt dessen muss dies das Lokalisierungssystem von sich aus tun.

All diese Funktionalität zusammen mit der Abfrage in einem einzelnen Service bereitzustellen, hätte vor allem das Deployment und die Skalierbarkeit unnötig verkompliziert und langfristig wohl auch die Wartung der Software. Das grundsätzliche Konzept der Software und das Vorgehen, beispielsweise hinsichtlich Architektur, sind aber identisch geblieben.

Weitere Änderungen sind eher kosmetischer Natur, beispielsweise, dass zwischen Lokalisierungssystem und Presence Manager statt eines Webservices JMS verwendet wird und hat mit geänderten Anforderungen seitens der angebundenen Systeme zu tun.

20.1.3 Physische Sicht der Systemarchitektur

Die physische Sicht der Systemarchitektur wurde in zwei Teile aufgespaltet, um die Übersichtlichkeit nicht zu gefährden. Teil 1, zu sehen in Abbildung 20.4 (Seite 151), beschäftigt sich vor allem mit dem oberen Teil der Architekturübersicht, also vom «WUI» bis zum «Orakel». Teil 2, zu sehen in Abbildung 20.5 (Seite 152), dreht sich um den unteren Teil, also vom «Orakel» bis zum «Adlerauge».

Beim gesamten Lokalisierungssystem handelt es sich um eine lose gekoppelte n-Tier-Architektur, um sie einfacher skalieren, Komponenten bei Bedarf ersetzen, ergänzen oder in einem anderen Kontext einsetzen können. So lässt sich beispielsweise das Lokalisierungssystem ohne «Orakel» betreiben, wenn keine Abfragemöglichkeit gefragt ist. Weiter ermöglicht die Verteilung auf mehrere Tiers, Caches zwischen die Tiers zu setzen, um Lastspitzen abzufedern, oder einzelne Elemente zu clustern. Das Ziel bei der Entwicklung der Architektur war, dass das Drei-Kamele-Prinzip[1] eingehalten werden kann. Übersetzt auf den Server-Betrieb heisst dass, dass für jeden Service mindestens drei Server verfügbar sein sollten: Stürzt einer ab und verkraftet der zweite die Flut von Anfragen nicht (z.B. weil die Caches nicht vorgewärmt sind), ist noch ein dritter da, der dem zweiten Server hilft.

[1] Wenn man in die Wüste geht, sollte man drei Kamele mitnehmen. Falls eines stirbt und sich eines ein Bein bricht, schafft man es noch immer zur Wüste hinaus (Quelle unbekannt).

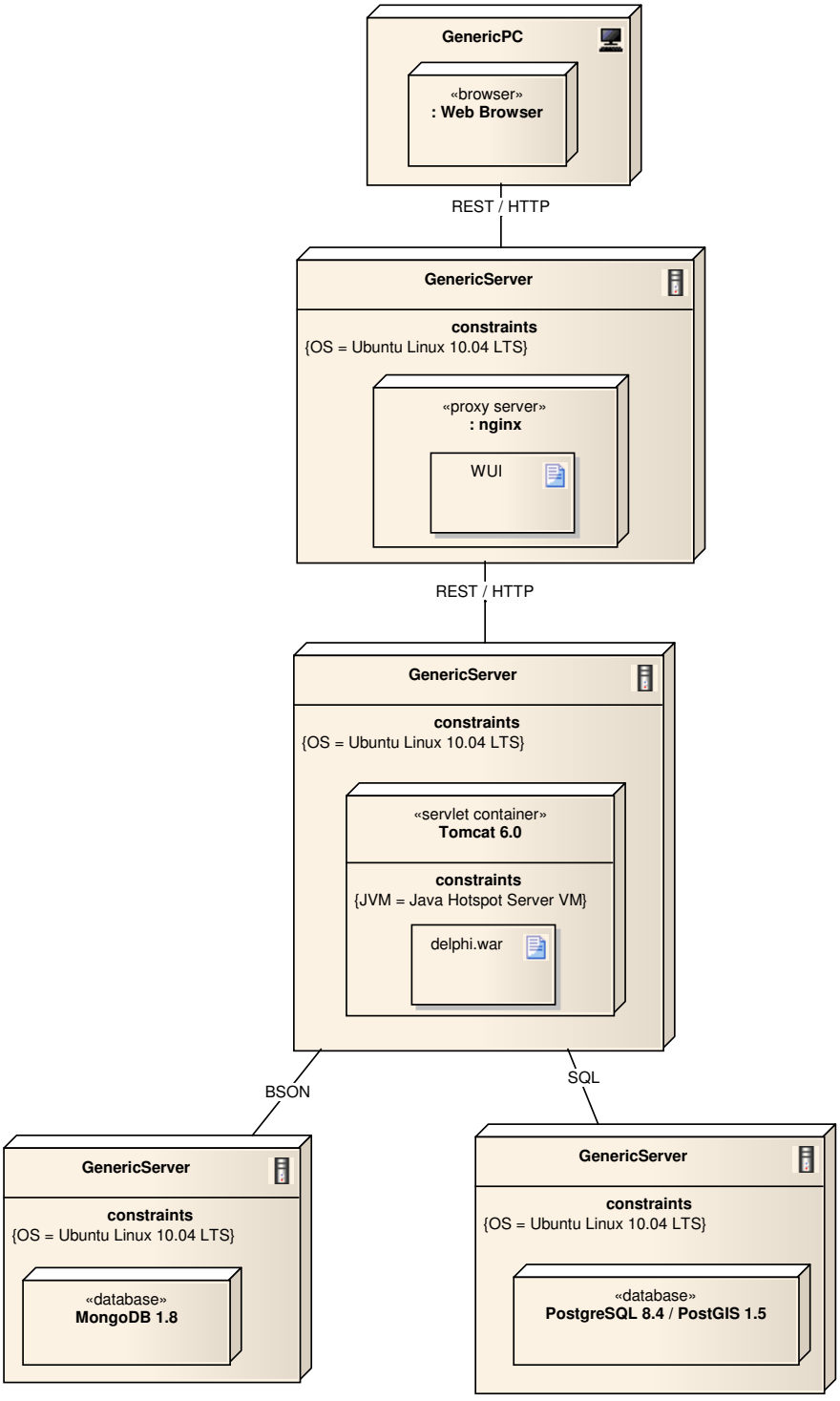

Abbildung 20.4: Deployment-Diagramm, Teil 1: Es zeigt die Minimalversion eines verteilten Deployment von «Orakel» bis Web-UI. Teil 2 (Abbildung 20.5, Seite 152) zeigt das Deployment von «Orakel» bis «Adlerauge».

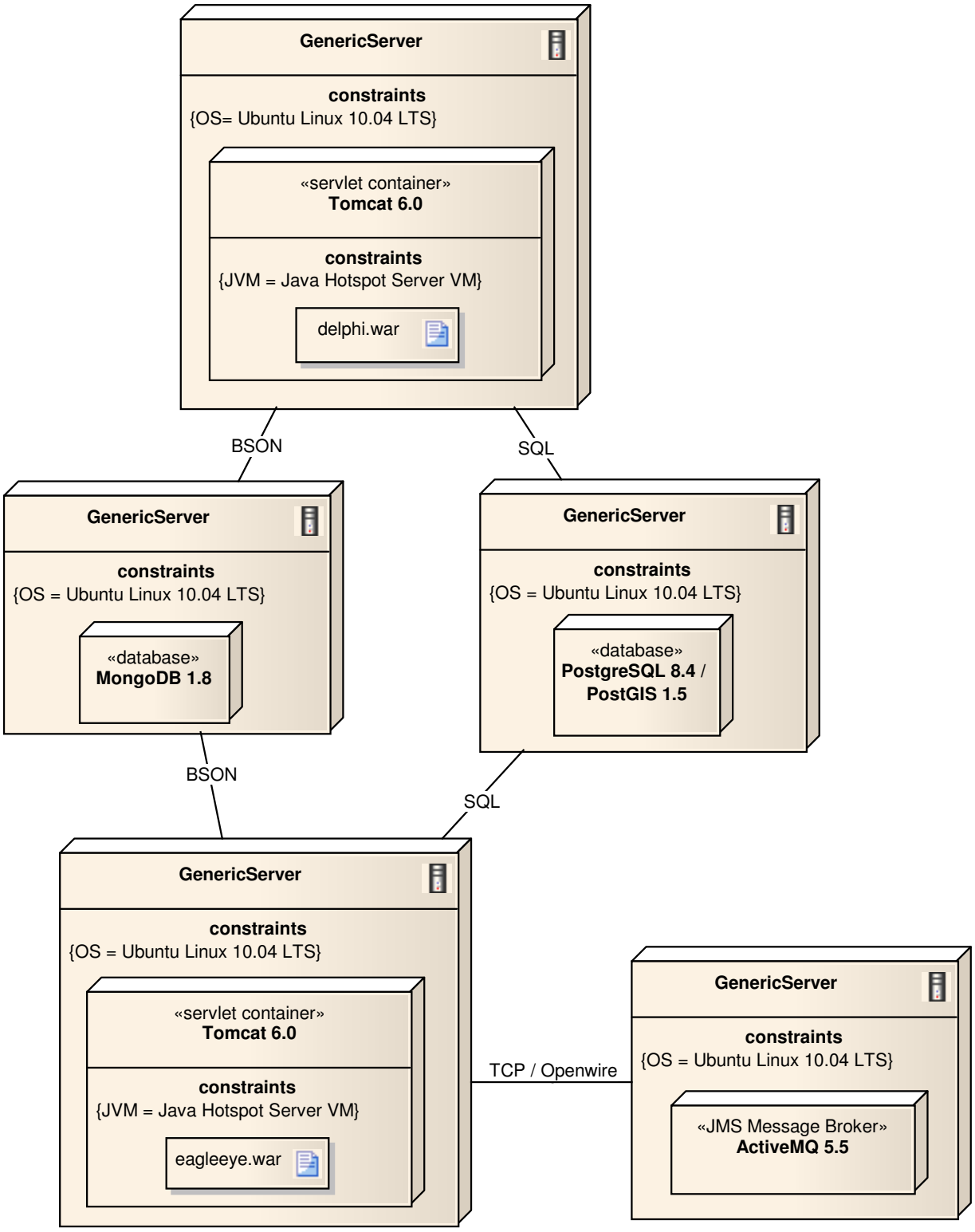

Abbildung 20.5: Deployment-Diagramm, Teil 2: Es zeigt die Minimalversion eines verteilten Deployment von «Orakel» bis «Adlerauge».

Die Deployment-Diagramme zeigen die einfachste Version eines verteilten Deployment, bei dem jeder Service auf einem separaten Server untergebracht wird und individuell nach Bedarf skaliert respektive für Redundanz gesorgt werden kann. Für die Datenbanksysteme (PostgreSQL, MongoDB) und den Message Broker ActiveMQ sei diesbezüglich bereits jetzt an deren Dokumentation verwiesen. Selbstentwickelte Komponenten sind zustandslos und lassen sich einfach duplizieren.

Für einfache Einsatzszenarien, die weder spezielle Anforderungen hinsichtlich Redundanz noch Verarbeitungsleistung haben, ist auch ein deutlich einfacheres Deployment möglich. So lassen sich beispielsweise alle selbst produzierten WAR in denselben Servlet Container werfen und zwei Server sparen. Weitere vier Server lassen sich einsparen, indem Message Broker, Proxy Server, PostgreSQL und MongoDB alle zum Servlet Container auf denselben Server ziehen. Kurzgesagt: Die gesamte Software lässt sich wie im Entwicklungsszenario auf einem einzigen Server ausführen.

Beschreibung der Einzelkomponenten

Client Als Benutzerschnittstelle zur Abfrage der Lokalisierungslogik dient ein Webinterface, das mit einem halbwegs aktuellen, standardkonformen Webbrowser auf einem beliebigen Client ausgeführt werden kann. Einzige Voraussetzung ist, dass die neuen Strukturelemente von HTML5 wie `section` und das Local Storage API unterstützt werden. Das Webinterface erhält der Client von einem Proxy Server. Daten konsumiert er von im REST-Architekturstil realisierten Webservices des «Orakels». Die Logik zur Datenabfrage, Seitenerzeugung und Steuerung des Seitenflusses wurde mittels JavaScript realisiert, sodass sie lokal auf dem Client ausgeführt werden kann. Dies nimmt Rechenlast vom «Orakel», da keine Seiten vorgerendert werden müssen und sorgt dafür, dass dieselbe Schnittstelle für die Anbindung weiterer Benutzerschittstellen wie Smartphone-Apps oder einer WPF-Anwendung komplett wiederverwendet werden kann. Müssen Daten von Drittsystemen abgefragt werden, kann sie der Client direkt dort beziehen, ohne dass sie eine «Rundreise» über das Lokalisierungssystem machen müssen, was die Implementierung einer weiteren Schnittstelle vom Lokalisierungs- zum Drittsystem erfordern würde.

Proxy Server Der Proxy Server kann einerseits als Cache zwischen Client und «Orakel» dienen, um häufig nachgefragte Inhalte, die sich aber selten ändern – wie beispielsweise die Gebäudeliste –, zwischenzuspeichern, damit sie nicht immer neu über die REST-Schnittstelle ausgeliefert werden müssen. Eine weitere Aufgabe besteht in der Auslieferung der statischen Anteile der Webbenutzerschnittstelle (HTML-Dateien, Bilder, JavaScript-Dateien) und in der Einhaltung der Same Origin Policy (SOP), einer Sicherheitsregel von Webbrowsern zum Nachladen von Inhalten, die verlangt, dass diese von demselben Uniform Resource Locator (URL) stammen wie die ursprünglichen Inhalte [17].

Als Proxy-Server wurde der relativ einfach zu konfigurierende und leistungsfähige Open-Source-Proxy-Server nginx[2] von Igor Sysoev gewählt. Es kann aber auch eine beliebige andere Software mit vergleichbarer Funktionalität wie Squid[3] oder sogar Apache [4] gewählt werden. Im

[2] `http://nginx.org/en/` (abgerufen: 10. Juni 2011)
[3] `http://squid-cache.org/` (abgerufen: 10. Juni 2011)
[4] `http://http.apache.org/` (abgerufen: 10. Juni 2011)

Unterschied zur Studienarbeit ist der Proxy Server nicht mehr optional, weil die Webbenutzer-Schnittstelle kein WAR mehr ist, das im Servlet Container ausgebracht werden könnte.

Orakel Es entspricht zu weiten Teilen der Lokalisierungslogik aus [1]. Einzig die Abfrage der Konnektoren wurde herausgelöst und in einen separaten Service (das «Adlerauge») verlagert. Statt dessen fragt es nun die Datenbank mit der Positionshistorie (die «Datenhalde») ab. Geo-, Geräte- oder Personeninformationen werden wie bis anhin aus dem RDBMS geholt. Realisiert ist das «Orakel» in Java. Es exponiert eine Reihe von Webservices in Form von Servlets über einen Servlet Container, der kompatibel sein muss zum Web Tier von Java 6 EE. Als Referenzplattform wurde Apache Tomcat in der Version 6 gewählt. Alternativ kann aber auch eine neuere Version oder Jetty verwendet werden, der ohnehin im Rahmen der Entwicklung verwendet wird. Webservices werden verwendet, da das gesamte Lokalisierungssystem Teil eines grösseren, heterogenen Systems ist, für das Webservices den grössten gemeinsamen Nenner darstellen. Der Presence Manager ist beispielsweise in .Net implementiert, was es schwierig macht, mit ihm über Remote Procedure Call (RPC) oder Remote Method Invocation (RMI) zu kommunizieren. Die Webservices sind im REST-Stil implementiert, da dies nebst Simple Object Access Protocol (SOAP) die am weitesten verbreitete Webservice-Technologie ist, aber deutlich weniger Overhead aufweist und alternative Datenformate wie JSON unterstütz, das sich mit JavaScript besonders einfach weiterverarbeiten lässt.

Bürokrat Als Katalog und Geoinformationssystem wird weiterhin PostgreSQL zusammen mit PostGIS verwendet. Die Gründe sind bereits in [1] (Seite 69) dargelegt worden. Da von der Datenbank weitestgehend gelesen wird, sind auch keine Nachteile hinsichtlich Redundanz und Skalierbarkeit zu erwarten, die es erfordert hätten, über einen Wechsel nachzudenken, zumal brauchbare Alternativen ohnehin mehr als spärlich gesät sind.

Nebst den Funktionen, die der «Bürokrat» bereits in der Studienarbeit erfüllt hat, dient er neu auch als Job Store für den Scheduler, der dazu dient, sicherzustellen, dass die Sensoren regelmässig lokalisiert werden.

Adlerauge Wie bereits erwähnt, übernimmt das «Adlerauge» die eigentliche Lokalisierung und damit die Interaktion mit den verschiedenen Lokalisierungswerkzeugen vom «Orakel». Diese können in Form von Konnektor-Plug-ins, die unabhängig entwickelt werden können, beliebig ausgewechselt werden, ohne Anpassungen am «Adlerauge» vornehmen zu müssen.

Die Architektur wurde vom «Orakel» übernommen, entsprechend gelten die gleichen Hinweise hinsichtlich Implementierung. Hinsichtlich Funktionalität gibt es aber einige Abweichungen. So ist ein Scheduler in Form des Quartz Enterprise Job Scheduler von Terracotta Teil der Software, der als Taktgeber zur regelmässigen Lokalisierung der Sensoren dient. Er ist mit Hilfe von Spring an den Lifecycle des Servlet Container gebunden, wird also automatisch gestartet und gestoppt. Als Job Store dient der «Bürokrat». Ebenso Teil des «Adlerauges» ist eine Rule Engine in Form von JBoss Drools, das zur Ausführung von Business Rules dient. Mit Hilfe der Business Rules wird aus den einzelnen Sensor-Positionen die Position der Geräte und daraus dann die Position der Personen bestimmt. Zudem dient Drools zur Erkennung und Auslösung von Ereignissen, die über JMS in die Welt hinausposaunt werden.

An Schnittstellen ist einerseits eine Verwaltungsschnittstelle für Sensoren, Geräte und Personen vorhanden. Diese wird benötigt, damit das «Adlerauge» weiss, welche Sensoren, Geräte und Personen es lokalisieren soll. Die Verwaltungsschnittstelle ist wie die meisten anderen Webservice-Schnittstellen im REST-Architekturstil gehalten und dürfte primär dazu dienen, Back-end-Systeme anzudocken. Sollten die Back-end-Systeme nicht das nötige Format sprechen, sollte nicht die Schnittstelle angepasst, sondern ein Vermittler dazwischen gesetzt werden – Apache Camel[5] ist bei solchen Problemen eine grosse Hilfe. Ansonsten müsste für jede Installation die Schnittstelle angepasst werden, was bei Systemen, die andere Kommunikationsmuster verwenden, sehr schnell sehr invasiv werden kann.

Datenhalde Die «Datenhalde» basiert auf dem NoSQL Document Store MongoDB und dient als clevere Variante der herkömmlichen Logdatei. Das bedeutet, es können Datenbank-ähnliche Abfragen mit Indices und aufwendige Analyseaufgaben mit Unterstützung von Map Reduce ausgeführt werden. Ohne dass dabei RDBMS-typische Probleme entstehen, wenn man immer nur in die gleiche Tabelle schreibt (kontinuierlich ansteigende Dauer von Index-Rebuilds, Schreibskalierung, mühsame Verteilung von Daten auf mehrere Server). Aktuell werden alle Positionen von Geräten und Personen gespeichert. Sollte gewünscht werden, lückenlose Nachvollziehbarkeit zu implementieren, ist die «Datenhalde» der richtige Ort zur Aufbewahrung der Daten.

Die Kommunikation mit «Orakel» und «Adlerauge» erfolgt über das MongoDB-Protokoll Binary JavaScript Object Notation (BSON), wobei der Zugriff von beiden Seiten mit Spring Data gekapselt ist.

Message Broker Der Message Broker dient primär zur Publikation von Events – also Producer-Consumer-Szenarien. Er kann später auch für asynchrone Maschine-zu-Maschine-Kommunikation genutzt werden. Der Message Broker kann entweder vom Lokalisierungssystem gestellt oder ein bereits Vorhandener genutzt werden.

Verwendet wird Apache ActiveMQ, da dieser bereits beim Presence Manager zum Einsatz kommt. Es kann aber problemlos auf JBoss HornetQ oder eine andere JMS-kompatible Implementierung gewechselt werden, da die ganze JMS-Interaktion über Spring gekapselt ist und so nur die Connection Factory von ActiveMQ in einer XML-Konfigurationsdatei ausgetauscht werden muss.

20.1.4 Logische Schicht der Systemarchitektur

Die Architektur der beiden von uns entwickelten serverseitigen Komponenten «Orakel» und «Adlerauge» ist vom Aufbau her identisch. Sofern nicht explizit erwähnt, beziehen sich alle Ausführungen auf beide Komponenten gleichermassen.

Schichtenarchitektur

Die Architektur der serverseitigen Komponenten verwendet eine gelockerte Schichtenarchitektur. Abbildung 20.6 (Seite 157) zeigt die Schichtenarchitektur des «Orakels», Abbildung 20.7 (Seite 158) diejenige des «Adlerauges». Dies bedeutet, dass Packages auf alle tiefer gelegenen Pakete

[5] `http://camel.apache.org/` (abgerufen: 10. Juni 2011)

zugreifen und damit einzelne Ebenen überspringen dürfen (je weniger, desto besser). Ein Zugriff nach oben ist allerdings verboten. Verantwortlichkeiten der Schichten, von oben nach unten:

Service Layer Der Service Layer implementiert das Controller Pattern nach [13] (Seite 302) und kümmert sich damit um die Ausführung der Use Cases. Er umfasst ausserdem die Webservice- und JMS-Schnittstellen samt der zugehörigen Data Tranfer Objects, in die die Entitäten konvertiert werden, um sie über die Leitung zu schicken. Er kümmert sich ausserdem um die Validierung ankommender Daten.

Business Layer Implementiert Operationen auf den Entity-Objekten und kapselt den Zugriff auf den persistenten Datenspeicher.

Data Access Layer Abstrahiert den Zugriff auf die verschiedenen verwendeten Datenspeicher (RDBMS, Document und Key Value Stores).

Entity Layer Umfasst Java Beans – nur bestehend aus Getters und Setters – zum Mapping der Daten auf Datenbank-Tabellen (RDBMS), Dokumente (Document Stores) respektive Schlüssel-Wert-Paare (Key Value Stores).

Foundation Layer Enthält Utility-Klassen zur Anbindung und zum Zugriff auf externe Bibliotheken.

Auf einen echten Domain Layer mit Logik zwischen Service und Data Access Layer wurde verzichtet. Die Gründe dafür sind:

- Viele Operationen (Datenanalyse, Geo-Operationen) lassen sich deutlich effizienter direkt in den Datenbanken ausführen, da sie über die nötige Infrastruktur zur automatischen Parallelisierung der Aufgaben verfügen oder beispielsweise dank spezieller Indices deutlich schneller die benötigten Daten finden.
- Der ganze Objektgraph kann nicht im Arbeitsspeicher gehalten werden, da dazu die Datenmengen viel zu gross sind.
- Die Nutzung eines Object-Relational Mapping (ORM) wie Hibernate, das dank Lazy Loading nur die Daten laden würde, die für Operationen gebraucht werden, und sich automatisch ums Mapping kümmern würde, ist nicht möglich, da einerseits aufwendige Anpassungen für geometrische Datentypen und Operationen vorgenommen werden müssten (noch machbar) und das Java Persistence API (JPA) auf RDBMS ausgerichtet ist und mit den ebenfalls verwendeten Document und Key Value Stores nicht zurecht kommt. In einer Webservice-Umgebung, bei der Objekte «auf die Reise gehen», müsste ausserdem peinlich genau darauf geachtet werden, dass der `EntityManager` von JPA nicht die Synchronizität verliert (Stichwort Detached Entity Syndrome) und das Lazy Loading von Referenzen bei der Konvertierung von Entities in Data Transfer Objects und umgekehrt nicht die Transaktionsgrenzen verletzt – ein Aufwand, der angesichts der anderen Herausforderungen, ungerechtfertigt ist.

Statt dessen wurden die Verantwortlichkeiten eines echten Domain Layer aufgeteilt. Um die Interaktion mit den Data Stores kümmert sich der Data Access Layer und die Manipulation der Entities übernimmt der Business Layer.

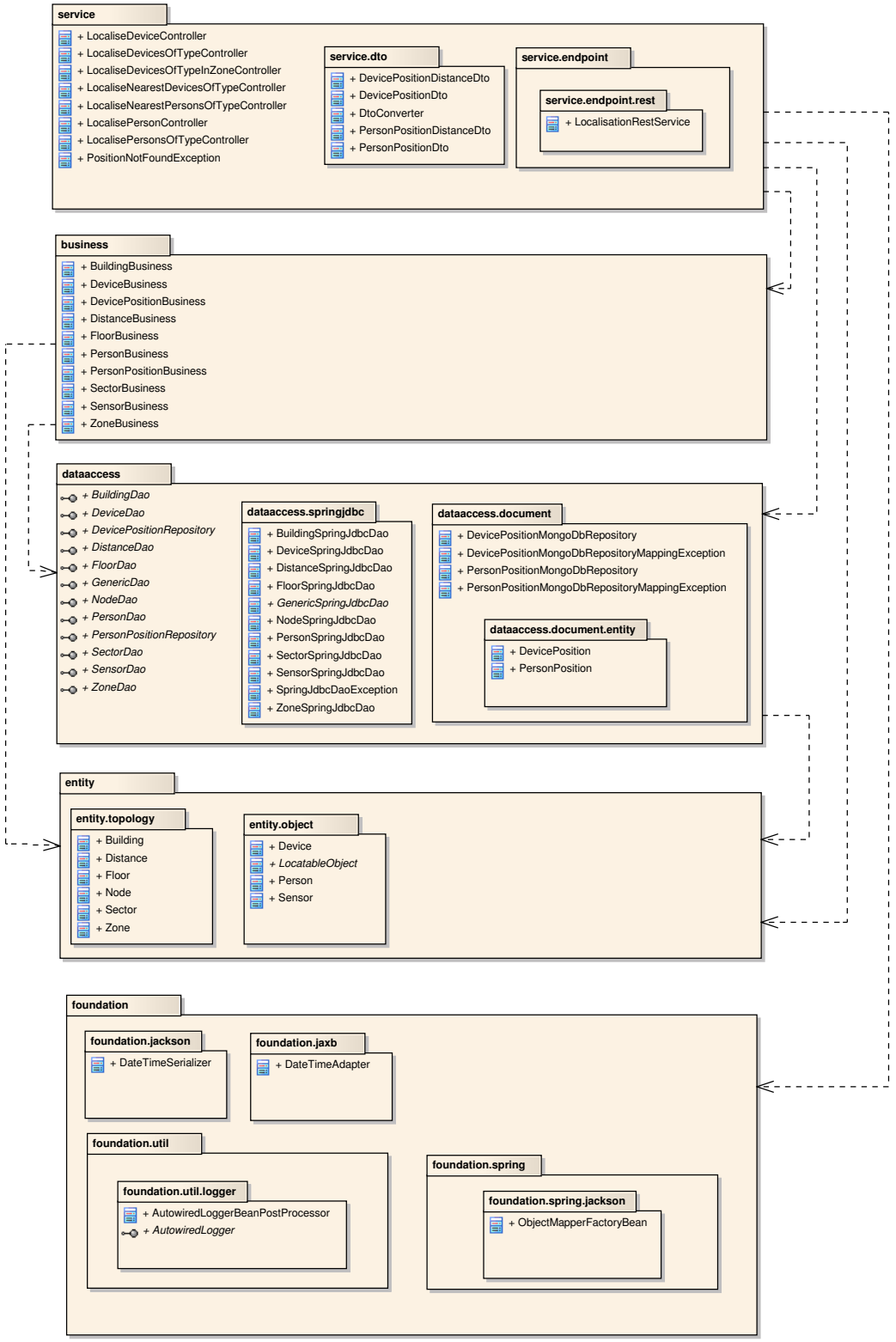

Abbildung 20.6: Schichtenarchitektur «Orakel»

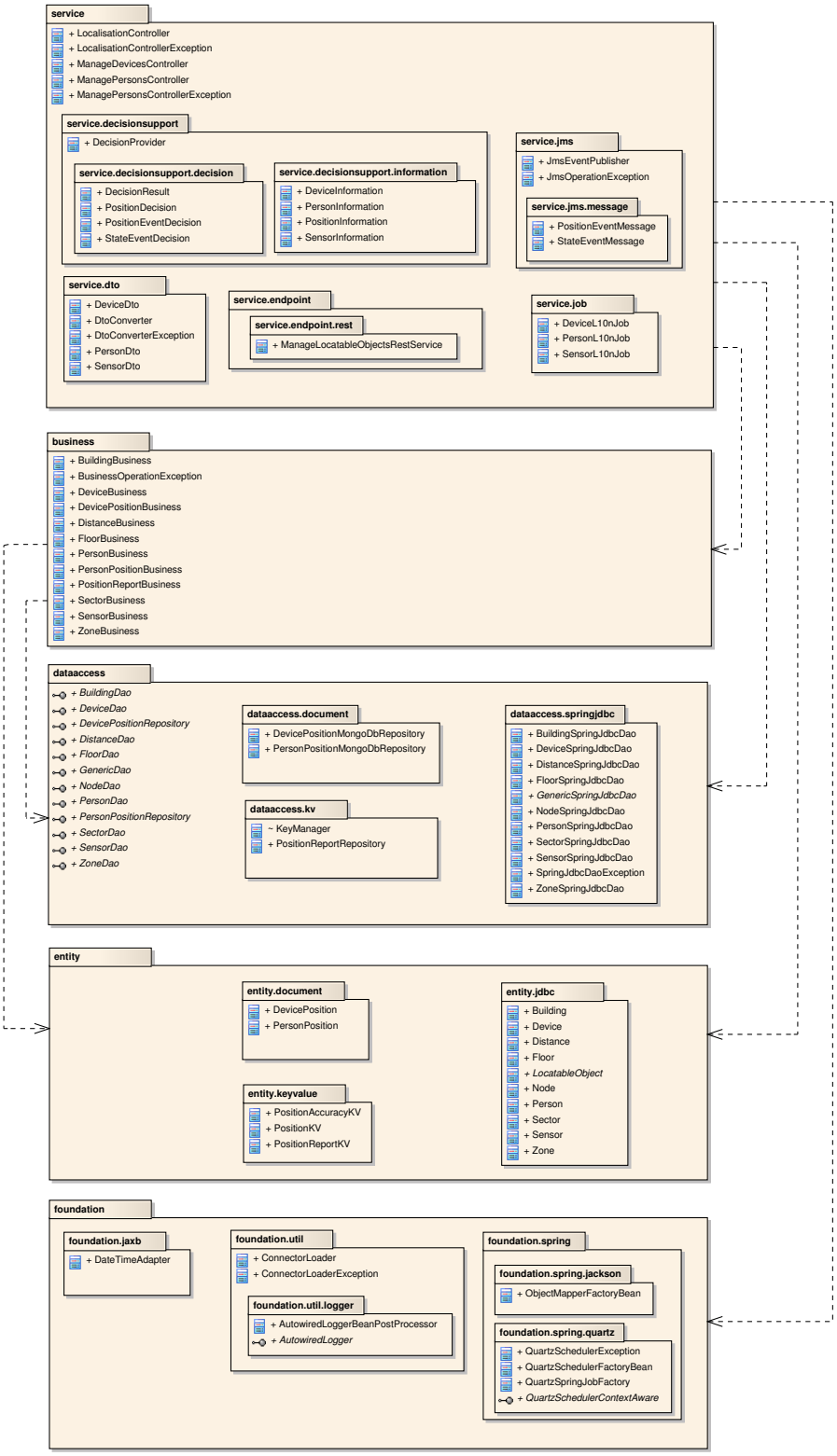

Abbildung 20.7: Schichtenarchitektur «Adlerauge»

Übersicht über die Packages

Die Liste gibt eine Übersicht über die Zuständigkeiten der verschiedenen Packages. Die detaillierten Beschreibungen sind in Kapitel 20.2 (Seite 164) zu finden. Bei sämtlichen Packages wurde das Präfix `ch.hsr.ins.delphi` beziehungsweise `ch.hsr.ins.eagleeye` aus Gründen der Lesbarkeit weggelassen.

business Enthält Klassen, die Operationen auf den Entities ausführen, beispielsweise das Erstellen und Aktualisieren von Objekten.

dataaccess Enthält Interfaces für Data Access Objects und Document Repositories, die von den konkreten Implementierungen zum Zugriff auf die verschiedenen Data Stores verwendet werden.

dataccess.document Klassen für den Zugriff auf Document Repositories, die von Document Stores bereitgestellt werden. Die vorhandenen Implementierungen sind gegen MongoDB programmiert.

dataaccess.kv Nur beim «Adlerauge». Klassen für den Zugriff auf Key Value Stores. Die vorhandenen Implementierungen sind gegen Redis programmiert.

dataccess.springjdbc Implementierung der Data Access Objects mit Hilfe von Spring JDBC für den Zugriff auf PostgreSQL.

entity Enthält die Entitäten für alle Data Stores.

entity.document Enthält die Entitäten für Document Stores.

entity.keyvalue Nur beim «Adlerauge». Enthält die Entitäten für Key Value Stores.

entity.jdbc Enthält die Entitäten für über JDBC-abfragbare RDBMS.

foundation Sammlung von Façaden und Utility-Klassen zur Integration von und Interaktion mit externen Bibliotheken.

foundation.jaxb Utility-Klassen zum Mapping von Value Objects zwischen Java und XML.

foundation.spring Utility-Klassen zur Integration mit Spring. Es handelt sich vorwiegend um `FactoryBeans`, um externe Bibliotheken laden zu können.

foundation.util Verschiedene Hilfsklassen und Glue Code.

service Umfasst den Service Layer mit den verschiedenen Controllers zur Ablaufsteuerung der Use Cases.

service.decisionsupport Nur beim «Adlerauge». Drools-Integration, welche die Anwendung von Business Rules erlaubt, zur Manipulation des Kontrollflusses oder von Daten. So lässt sich mit Business Rules beispielsweise entscheiden, ob Ereignisse gefeuert werden sollen oder nicht.

service.dto Entities werden in Data Transfer Objects, die sich in diesem Package befinden, konvertiert, bevor sie das System verlassen und umgekehrt. Bei ankommenden Daten findet hier ausserdem die Validierung statt.

service.endpoint Enthält die Endpunkte der unterstützten Webservice-Protokolle. Sie bilden die Brücke zwischen Webservices und den Controllers.

service.jms Nur beim«Adlerauge». Enthält die Klassen zur Interaktion mit JMS Message Brokers.

service.job Nur beim «Adlerauge». Klassen, die vom Scheduler ausführbare Jobs repräsentieren. Die Jobs werden vom Scheduler angestossen, worauf die Ausführung an die Controller delegiert wird.

Technologiekomponenten

Die Liste zeigt einer Aufstellung der wichtigsten verwendeten Programmbibliotheken:

Drools Business Rule Engine von JBoss zur Ausführung von Business Rules. Nur beim «Adlerauge».

Guava Google-Ergänzung zu `java.util`. Wird unter anderem verwendet zur Concurrency-Unterstützung, zur Collection-Manipulation und zum Überschreiben von `hashCode()` und `equals()`.

Jackson Dient zur Umwandlung von Java Beans in JSON und umgekehrt.

JTS Topology Suite Bibliothek zur Durchführung von geometrischen Operationen und Verarbeitung von GIS-Daten.

Joda Time Bibliothek mit einfacheren und leistungsfähigeren Datumsoperationen als sie Java von Haus aus bereitstellt.

Quartz Enterprise Job Scheduler Scheduler von Terracotta. Dient zur regelmässigen Ausführung von Jobs, beispielsweise der Lokalisierung von Sensoren, Geräten und Personen. Nur beim «Adlerauge».

RESTeasy JAX-RS Provider von JBoss zur programmatischen Erstellung der REST Endpoints. Wurde gewählt, da es sehr gute Testing- und SpringMVC-Unterstützung bietet.

SLF4J Logging-Façade für diverse Logging Frameworks. Verwendet wird die Log4j-Weiterentwicklung Logback.

Spring Data Abstrahiert und vereinfacht den Umgang mit verschiedenartigen Datenquellen. Wird benutzt zum Zugriff auf MongoDB und Redis (nur beim «Adlerauge»).

Spring Framework Meta-Framework, auf dem die Anwendung zu weiten Teilen basiert. Verwendet wird es unter anderem für Dependency Injection, aspektorientierte Programmierung, vereinfachten JDBC-Zugriff, vereinfachte JMS-Nutzung, Transaktionen, Sicherheit und Servlet-Unterstützung. Verwendet werden die Komponenten AOP, Beans, Context, Core, Expression, JDBC, JMS (nur «Adlerauge»), TX, Web und WebMVC.

Stringtemplate Template Engine zur Auslagerung von Templates aus dem Programmcode. Beispiele sind Map-Reduce-Funktionen für MongoDB. Nur beim «Orakel».

Die folgende Liste zeigt eine Aufstellung der wichtigsten, fürs Testing verwendeten Programmbibliothken:

JUnit Unit-Testing-Framework zur Erstellung und Ausführung von Unit Tests.

Mockito Mocking-Framework, das die Erzeugung von Mocks, Fakes und Stubs vereinfacht.

Spring Test Teil des Spring Frameworks, der das Testing von Spring-Anwendungen vereinfacht.

Eine komplette Liste der verwendeten Programmbibliotheken sowie der transitiven Dependencies kann mit Hilfe des Build-Werkzeugs Maven abgefragt werden.

20.1.5 Architekturkonzepte

Die nächsten Abschnitte beschreiben einige allgemeine, nennenswerte Architekturkonzepte, die für die gesamte Architektur relevant sind. Spezifische Architekturkonzepte, die nur innerhalb eines Layers von Interesse sind, sind im Abschnitt 20.2 (Seite 164 ff.) beschrieben.

Aspektorientierte Programmierung

Wo sinnvoll, werden die AOP-Fähigkeiten von Spring verwendet, um allgemeines Verhalten von der Business-Logik zu trennen und einfach hinzufügen beziehungsweise entfernen zu können. Typische Anwendungsfälle sind Auditing, Security oder Transaktionen. Aspektorientierte Programmierung (AOP) wird aktuell für die Transaktionsdemarkation verwendet.

Concurrency

Sowohl das «Orakel» als auch das «Adlerauge» machen intensiv Verwendung von Multithreading, wenn auch nur indirekt über Servlets, JMS `MessageListener` (vorgesehen für Konnektoren) und die Scheduler Jobs. Dies bedeutet, dass sowohl das «Orakel» als auch das «Adlerauge» grundsätzlich Thread safe sein müssen. Am einfachsten geht dies, indem man auf Conversational State verzichtet und die Anwendung zustandslos macht. Dies hat einen weiteren Vorteil: Die Installationen der Anwendung können beliebig dupliziert werden, weil es dann für den Client egal ist, mit welcher Instanz er spricht.

Beim «Orakel» und dem «Adlerauge» wird das mit Hilfe von Spring erreicht, das ein zustandsloses Design von Collaborators fördert (sie müssen ja mit XML konfigurierbar sein) und das für die Konfiguration und Ausstattung der Klassen mit Collaborators zuständig ist. Eine Re-Konfiguration von Klassen zur Laufzeit ist per Konvention verboten. Ein zweiter Schritt ist, dass auf Shared Mutability verzichtet und versucht wird, soweit möglich mit nicht modifizierbaren Objekten zu arbeiten. Dies ist aber nicht ganz einfach, da die Java-Beans-Spezifikation Setters erfordert und viele Frameworks mit Java Beans arbeiten. Solange Java Beans aber nur zwischen Methoden hin und hergeschoben werden, ist es kein Problem, dass die Java Beans mutierbar sind.

Dependency Injection

Um die Kopplung von Architekturkomponenten niedrig zu halten und das Einführen von alternativen Implementierungen oder Fakes bei Unit Tests zu erleichtern, wird Dependency Injection verwendet. Dabei kommt der Inversion of Control Container von Spring zum Einsatz. Collaborators werden typischerweise über Annotationen nach Java Specification Request (JSR) 250[6] (v.a. `@Ressource`) injiziert. Ist Rekonfigurierbarkeit zur Laufzeit gefragt, wird Constructor Injection respektive bevorzugt normale Setter Injection verwendet, wobei mittels XML-Konfiguration festgelegt wird, was wo injiziert werden soll.

Die Dependency Injection spielt eine wichtige Rolle dabei, die Gesamtarchitektur zustandlos zu halten (siehe Abschnitt 20.1.5 auf Seite 161 für Details). Collaborators über `new` respektive eine Factory regelmässig neu zu erzeugen, ist deshalb ebenso verpönt, wie Collaborators zur Laufzeit selber zu rekonfigurieren. Diese Aufgabe ist alleine von Spring zu erledigen, da dieses darauf achtet, dass die Konstruktion und Konfiguration von Objekten Thread safe ist. Anders sieht es bei Datenobjekten aus (Java Beans), die nur zwischen Methoden hin und her durch die Ebenen der Architektur «geschoben» werden. Diese können problemlos mit `new` o.ä. erzeugt und mutiert werden.

Geodatenverarbeitung

Wie bereits in [1] (Seite 73 ff.) ausführlich dargelegt, werden Geometrieoperationen nach Möglichkeit in PostGIS ausgeführt, da dieses einen grossen Funktionsumfang mitbringt und dank spatialer Indices Abfragen deutlich schneller ausführen kann als die zum Beispiel in Java möglich wäre. Die Geodaten werden innerhalb der Software in Form von Strings in WKT transportiert, da sie nicht mutiert werden müssen, von Drittsoftware direkt verarbeitet werden können und ausserdem ein aufwendiges Mapping entfällt.

Konfiguration

Das Wiring von Spring Beans erfolgt grundsätzlich zentralisiert per XML Bean Definitions. Auf Classpath Scanning (mit `@Component` u.a.) wird nach Möglichkeit verzichtet, da Änderungen an der Konfiguration schnell Modifikationen am Code erfordern und die Konfiguration über die einzelnen Klassen verteilt ist. Zudem machen sie es einfacher, externe Konfigurationseinstellung zu laden und einzusetzen. Voraussetzung ist, dass Konfigurationsinformationen wie Verbindungsinformationen zu Datenbanken grundsätzlich mittels Setter based Injection[7] konfigurierbar sind. Sie können dann über einen `PropertyPlaceholderConfigurer` von Spring aus Properties-Dateien von ausserhalb des JAR beziehungsweise WAR geladen und eingesetzt werden.

Lifecylce Management

Müssen irgendwelche interne Services wie der Scheduler des «Adlerauges» gestartet werden, damit die Anwendung ihre Arbeit aufnehmen kann, sollten sie an den Lifecylce des `ApplicationContext` von Spring (und damit an den Lifecylce des Servlet Containers beziehungsweise

[6] `http://jcp.org/en/jsr/detail?id=250` (abgerufen: 11. Juni 2011)

[7] `http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/beans.html#beans-setter-injection` (abgerufen: 11. Juni 2011)

der JVM) gehängt werden, damit sie automatisch gestartet und gestoppt werden, ohne dass sich der Anwender darum kümmern muss. Spring bietet in Form von `SmartLifecycle` die nötige Infrastruktur.

Logging

Um sich nicht auf einen spezifischen Logging-Mechanismus festlegen zu müssen, ihn überall verwenden und die Kopplung reduzieren zu können, wurde entschieden, eine Façade um den gewählten Logging-Mechanismus zu legen. Mit der Simple Logging Façade For Java (SLF4J) existiert bereits so eine Façade, die anstatt einer Eigenentwicklung verwendet wird. Als Backend kommt im Moment die Log4J-Weiterentwicklung Logback zum Einsatz.

Mapping und Serialisierung

Insbesondere bei den Webservice- und JMS-Schnittstellen, aber beispielsweise auch bei der Speicherung von Objekten im Key Value Store, müssen Java Beans auf andere Datenformate abgebildet werden. Aktuell kommen XML und bevorzugt JSON zum Einsatz.

Für die Übersetzung der Java Beans in XML wird Java Architecture for XML Binding (JAXB) verwendet, da dies weitgehend automatisch arbeitet und von etlichen Frameworks wie dem ebenfalls verwendeten RESTeasy unterstützt wird. Ein weiterer Vorteil ist, dass es vom JSON-Serialisierer Jackson unterstützt wird, womit sich mithilfe weniger JAXB-Annotationen automatisch XML und JSON erzeugen lässt.

Für Serialisierung von Java-Objekten wird, soweit möglich, vorzugsweise JSON als Serialisierungsformat verwendet, um sich nicht mit dem `Serializable`-Interface und der Versionsproblematik der Java-Klassen herumschlagen zu müssen.

Spring Framework

Die gesamte Software greift, wo möglich und sinnvoll, auf die Fähigkeiten des Spring Frameworks zurück. Es vereinfacht viele Aspekte der Anwendungsentwicklung wie die Arbeit mit Java Database Connectivity (JDBC) oder JMS und bietet viele Funktionen eines Enterprise Java Beans (EJB) Containers, ohne dass ein solcher benötigt würde. Beispiele sind deklarative Transaktionen, Security oder aspektorientierte Programmierung. Es bringt zwar ein paar Nachteile mit sich, beispielsweise, dass für Komponenten ohne Spring-Unterstützung eventuell ein Adapter, meist in Form einer `FactoryBean`, entwickelt werden muss. Sie sind aber im Vergleich zum Nutzen von Spring vernachlässigbar.

Testing

Die Liste beschreibt nach Architekturebene, was wie getestet wird:

Service Layer Microtests mit JUnit, Integrationstests über alle Layers mit JUnit und Jetty (noch nicht realisiert)

Business Layer Microtests mit JUnit

Data Access Layer Integrationstests innerhalb des Layers mit JUnit.

Foundation Layer Microtests mit JUnit (soweit möglich und sinnvoll)

Validierung

Validierung erfolgt beim Eintritt von nicht vertrauenswürdigen Daten ins System, typischerweise direkt im Konverter für Data Transfer Objects, die empfangen wurden und im Service Layer zur Weiterverarbeitung in Entites umgewandelt werden sollen.

Zeitdarstellung

Zur Zeitdarstellung wird ISO 8601[8] verwendet. Es bietet eine einfache Darstellung von Zeitpunkten, Zeitdauern und Intervallen und berücksichtigt Zeitzoneninformationen, sodass beispielsweise klar zwischen Normal- und Sommerzeit unterschieden werden kann. Zum Umgang mit ISO 8601 wird die Java-Implementierung Joda Time verwendet.

equals(), hashCode() und toString()

Bei Java Beans werden die Methoden `equals()` und `hashCode()` überschrieben, um sie einfacher vergleichen oder sortieren zu können – praktische Hilfsklassen bringt die Bibliothek Guava mit. Ausnahme sind die Mitglieder von Klassenhierarchien, weil `equals()` (und damit auch `hashCode()`) in Klassenhierarchien nicht überschrieben werden können, wenn in einer abgeleiteten Klasse ein Datenfeld dazu kommt, ohne den Contract bezüglich Symmetrie von `java.lang.Object` zu verletzten [3] (Seite 35 ff.). `toString()` sollte immer überschrieben werden.

20.2 Packages

In diesem Abschnitt werden die einzelnen Packages des «Orakels» und des «Adlerauges» detailliert beschrieben. Sofern nicht explizit erwähnt, treffen alle Ausführungen auf beide Applikationen zu. Aus Gründen der Lesbarkeit wird das Prefix `ch.hsr.ins.eagleeye` respektive `ch.hsr.ins.delphi` jeweils weggelassen.

20.2.1 business

Klassenstruktur

Abbildung 20.8 (Seite 165) zeigt die Klassenstruktur des Business Layer.

Architekturkonzepte

Der Business Layer liegt zwischen Data Access Layer und Service Layer. Er fragt die Data Stores ab, erzeugt Objekte und führt Operationen auf den Entities aus. Er entspricht damit dem funktionalen Teil der Domain, wobei die Struktur in Form der Entites auf dem Entity Layer liegt.

Ein Design-Ziel beim Business Layer ist, so viele Operationen wie möglich an die dahinter liegenden Data Stores zu delegieren, da diese Aufgaben wie Selektion oder die Berechnung der

[8] `http://www.iso.org/iso/date_and_time_format` (abgerufen: 11. Juni 2011)

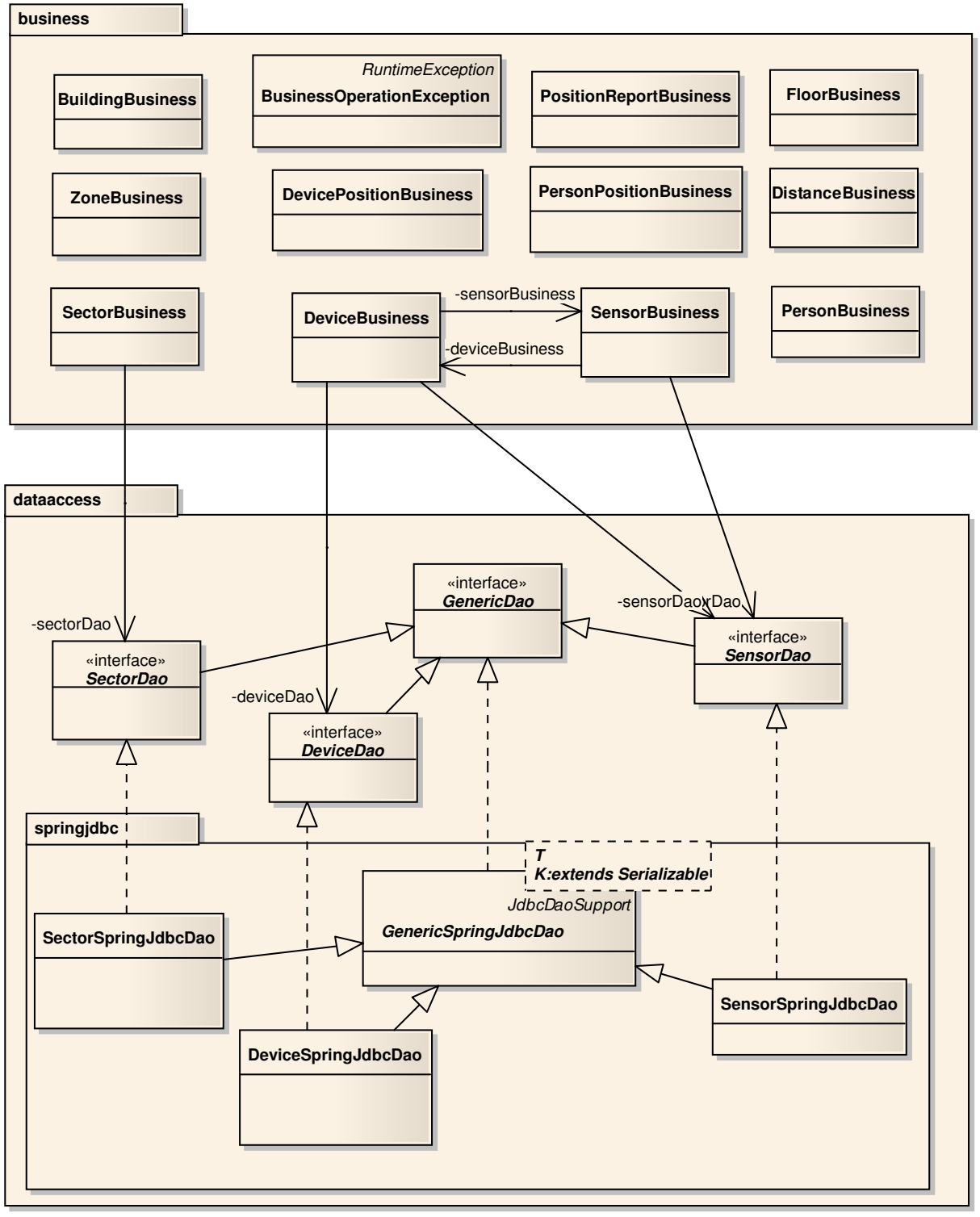

Abbildung 20.8: Klassenstruktur Business Layer

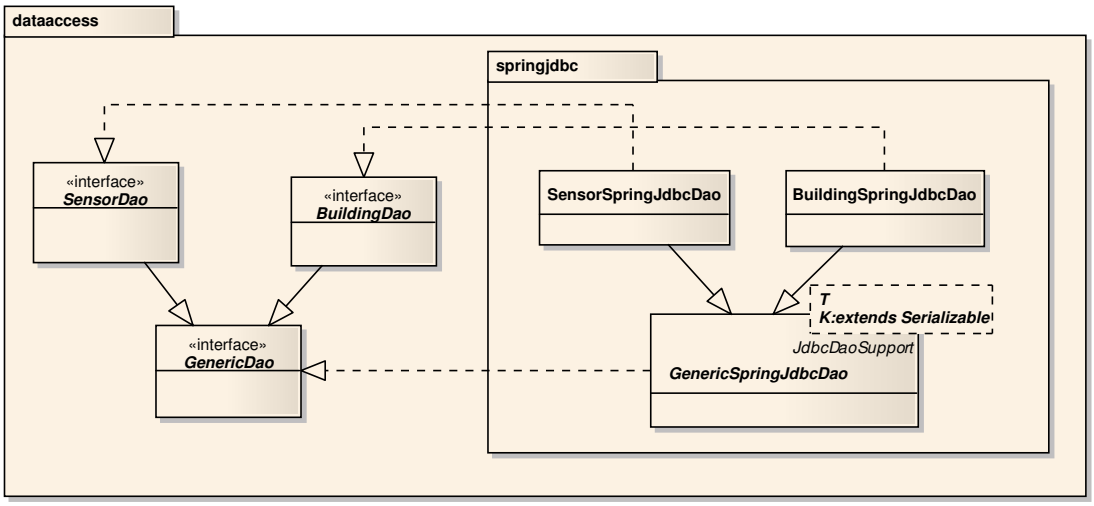

Abbildung 20.9: Klassenstruktur der JDBC-Anbindung im Data Access Layer

Distanz zwischen zwei Punkten auf der Topologie effizienter erledigen können. Muss beispielsweise ein Gerät anhand seiner Inventarnummer gefunden werden, teilt dies der Business Layer dem Data Access Layer mit, der daraufhin die passende Abfrage, beispielsweise in SQL, auslöst. Er holt nicht selber alle Geräte aus dem Speicher und iteriert über jedes einzelne Gerät, um das passende herauszufischen. Der Business Layer ist damit mehr ein «Durchlauferhitzer», der der Code-Deduplizierung dient und nur noch darauf achten muss, die richtigen Befehle zu erteilen und die unterschiedlichen Data Stores konsistent zu halten.

Die Klassen sind nach der Entität benannt und damit der Datenbank-Tabelle beziehungsweise dem Document Repository, auf dem sie Operationen anbieten. Eine Klasse wie `SectorBusiness` dient damit zur Verwaltung und Abfrage von `Sector`-Entitäten respektive der Datenbanktabelle `sector`.

Klassenspezifikation

Die Klassenspezifikation kann der separaten JavaDoc-Dokumentation entnommen werden.

20.2.2 dataaccess

Klassenstruktur

Abbildung 20.9 (Seite 166) zeigt einen Auszug aus der Klassenstruktur des Data Access Layer, der zur PostgreSQL-Anbindung über JDBC dient. Abbildung 20.10 (Seite 167) zeigt einen Auszug aus der Klassenstruktur des Data Access Layer, der die MongoDB-Anbindung illustriert.

Architekturkonzepte

Im Vergleich zu [1] hat sich der Data Access Layer stark verändert, da eine polyglotte Persistenzarchitektur eingeführt wurde. Kurz gesagt wird für jede Aufgabe jetzt die geeignete Datenbank-

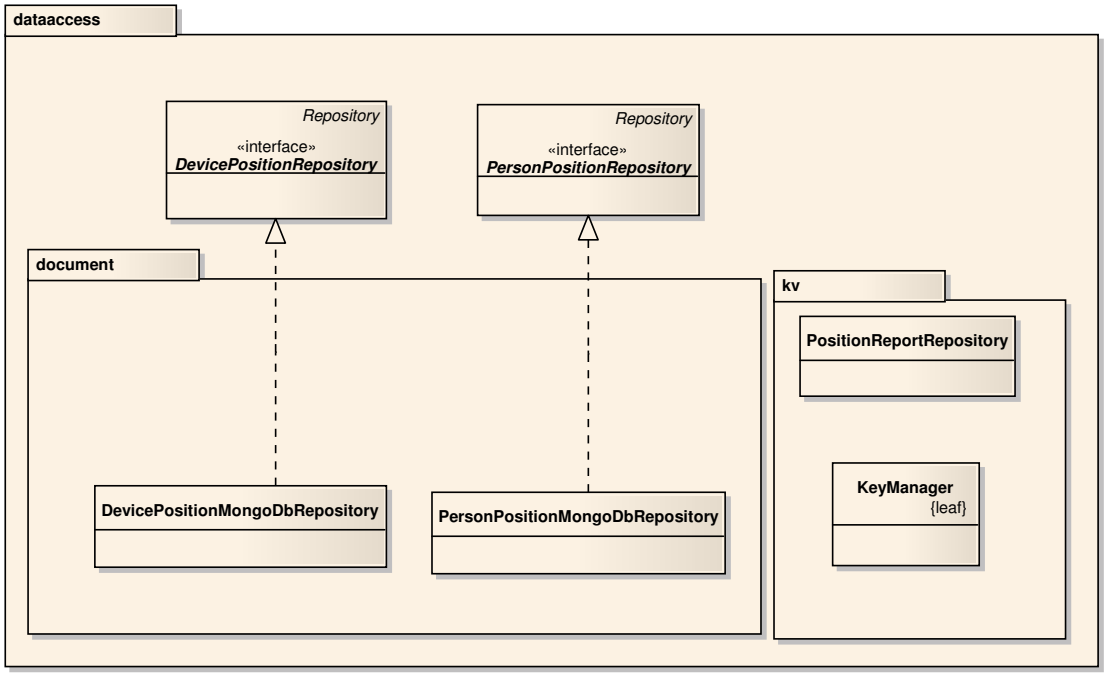

Abbildung 20.10: Klassenstruktur der MongoDB-Anbindung im Data Access Layer

Technologie verwendet und nicht mehr für alle Aufgaben dieselbe, die den grössten gemeinsamen Nenner darstellt. Zum Einsatz kommen jetzt drei Varianten von Data Stores: ein RDBMS (via JDBC), ein Document Store (MongoDB) und ein Key Value Store (Redis), wobei letzterer nur beim «Adlerauge» verwendet wird. Entsprechend wurde das Paket in drei Unterpakete aufgeteilt, die je für einen Data Store zuständig sind. Auf der obersten Ebene befinden sich nach wie vor die Interfaces, welche dazu dienen, den Datenzugriff von der gewählten Technologie zu entkoppeln. Das endgültige Ziel wäre, dass sich jedes Objekt an dem Ort speichern liesse, der für seine Bedürfnisse am besten geeignet ist, beispielsweise im Key Value Store bei häufigem Zugriff oder im RDBMS, wenn es selten mutiert wird und dafür die Konsistenz wichtig ist.

Grundsätzlich muss festgehalten werden, dass wir das Design des Data Access Layer noch nicht für besonders gut erachten, was einerseits der geringen Erfahrung im Umgang mit mehreren Speichertechnologien geschuldet ist und andererseits des noch sehr frühen Stadiums des verwendeten Spring Data (Details weiter unten), was dazu führt, dass etliche Workarounds um noch nicht funktionierende Funktionalität nötig waren und Best Practices zur Umsetzung der Anbindungen an die verschiedenen Datenbanken noch unzureichend dokumentiert oder gar nicht vorhanden sind. Nichts desto trotz ist Spring Data «The Way to Go» zur Anbindung der unterschiedlichen Datenspeicher.

RDBMS Um den Datenzugriff von der gewählten Technologie zu entkoppeln, wurde beim RDBMS das Data Access Object Pattern nach [2] (Seite 708 ff.) implementiert. Allerdings wurde auf die beschriebenen Factories verzichtet, da die mit Spring nicht nötig sind – jeder Spring `ApplicationContext` ist ja nichts anderes als eine intelligente Abstract Factory. Die zur Verfü-

```
1  public interface GenericDao <T, K extends Serializable> {
2      T findById(K id);
3      List<T> findAll();
4      T makePersistent(T entity);
5      void makeTransient(T entity);
6  }
7
8  public interface BuildingDao extends GenericDao <Building, Long> {
9      Building findByName(String name);
10 }
```

Listing 20.1: Auszug aus den Interfaces für die Data Access Objects

gung stehenden Methoden werden über eine Reihe von Interfaces definiert, wobei pro Entität ein DAO-Interface zur Verfügung gestellt wird, das DAO-spezifische Methoden deklariert. Methoden, die allen Data Access Objects (DAOs) gemein sind, sind in `GenericDao` definiert, das alle spezifischen DAOs erweitern. Listing 20.1 (Seite 168) zeigt als Beispiel `GenericDao` inklusive einer Erweiterung für den Zugriff auf Gebäude in Form des `BuildingDao`. Eine Implementierung für Spring JDBC befindet sich im Subpackage `dataaccess.springjdbc`. `GenericSpringJdbcDao` implementiert wiederum die allen DAOs gemeinsamen Methoden. Der Vorteil dieser Konstruktion ist, dass je nach Bedarf alternative Implementierungen ergänzt und je nach Entität eine andere verwendet werden kann, indem einfach die Bean Definitions von Spring neu verdrahtet werden. Ein nahe liegender Anwendungsfall wäre, die DAOs um einen Cache wie das bereits verwendete Redis zu ergänzen.

Wie erwähnt, wurde der RDBMS-Zugriff mittels JDBC realisiert, unter anderem weil viele Geo-Operationen ausgeführt werden müssen, worauf JPA QL nicht vorbereitet ist, weshalb der Einsatz eines ORM schwierig ist. Spring JDBC macht ausserdem die JDBC-Nutzung einfacher, da es viel vom ansonsten nötigen Boilerplate Code verschwinden lässt.

Document Stores Vom Prinzip her wurde die Anbindung von Document Stores identisch realisiert wie diejenige der RDBMS. Es konnte aber in Form von Spring Data Document bereits auf vorgefertigte Infastruktur und Interfaces zurückgegriffen werden. Für eine detaillierte Erklärung sei deshalb an dieser Stelle an die Dokumentation von Spring Data Document[9] verwiesen.

Implementiert wurde eine Anbindung an das im Projekt verwendete MongoDB, wobei die an JPA erinnernde automatische Mapping-Unterstützung verwendet wurde.

Key Value Stores Aufgrund der stark abweichenden Struktur des Key Value Stores im Vergleich zu RDBMS und den Document Stores hat die Anbindung der Key Value Stores wenig Ähnlichkeit mit den anderen Datenbanken. Wie die Doument Stores wurde die Anbindung mit Spring Data realisiert, das aber im Fall der Key Value Stores noch keine speziellen Vorgaben hinsichtlich Applikationsstruktur macht. Da bislang nur eine Entität angebunden wurde, wurde auch auf die Erstellung von Interfaces verzichtet, da bislang noch nicht abzusehen ist, welche Anforderungen an diese gestellt werden.

[9] `http://www.springsource.org/spring-data/` (abgerufen: 12. Juni 2011)

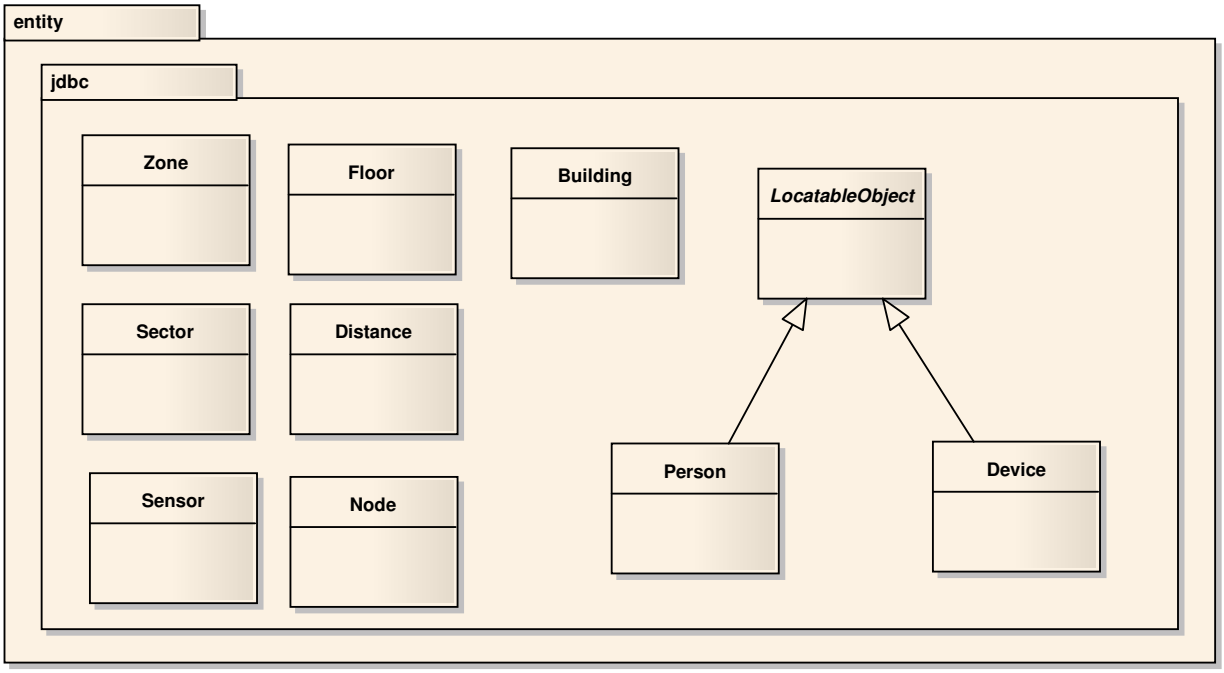

Abbildung 20.11: Entities für über JDBC angebundene Datenbanken im Entity Layer

Implementiert wurde eine Anbindung an das im Projekt verwendete Redis. Gespeichert werden serialisierte Java Beans, das heisst, es wird kein Mapping benötigt. Entgegen der Konventionen wird noch die JDK-Serialisierung verwendet, da die JSON-Serialisierung von Spring Data Key Value zum Implementierungszeitpunkt noch nicht sauber gearbeitet hat.

Klassenspezifikation

Die Klassenspezifikation kann der separaten JavaDoc-Dokumentation entnommen werden.

20.2.3 entity

Klassenstruktur

Abbildung 20.11 (Seite 169) zeigt die Struktur der Entities für über JDBC angebundene Datenbanken und Abbildung 20.12 (Seite 170) diejenigen für Document und Key Value Stores.

Architekturkonzepte

Der Entity Layer umfasst die Entitäten, die in den verschiedenen Data Stores gespeichert werden. Eine Entität repräsentiert jeweils eine zu speichernde Einheit, also eine Datenbank-Zeile, ein einzelnes Dokument oder ein Schlüssel-Wert-Paar. Bei den Entitäten handelt es sich um Java Beans, die nur aus Feldern, Getters und Setters bestehen. Wie Abbildung 20.11 (Seite 169) zeigt, bestehen zwischen den Entitäten keine sichtbaren Relationen. Dies rührt daher, dass sie

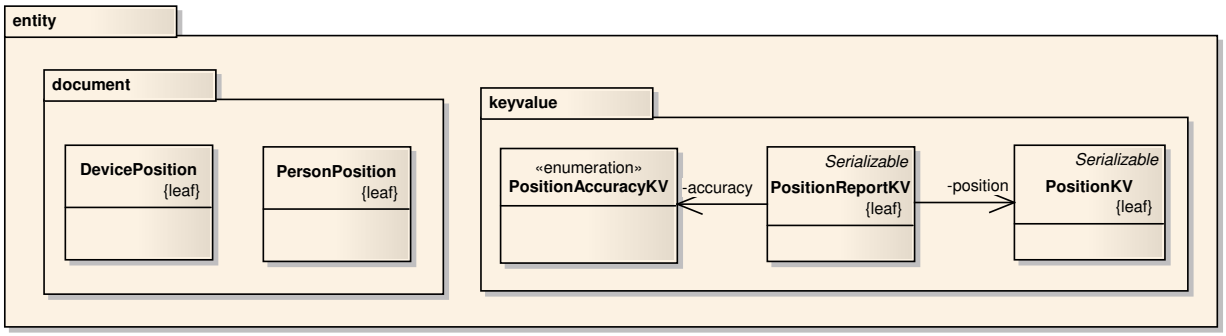

Abbildung 20.12: Entities für Document und Key Value Stores im Entity Layer

```
1  public class Sector {
2      private Long id;
3      private Long floorId;
4      private Long zoneId;
5      private String name;
6      private BigDecimal height;
7      private String centroid;
8      private String polygon;
9      private String srid;
10 }
```

Listing 20.2: Auszug aus der Entitätsklasse Sector

sich untereinander nicht über den Typ referenzieren, sondern über die Objekt-Identität, die vom jeweiligen Datenspeicher verwendet wird. Beim RDBMS handelt es sich beispielsweise um den numerischen Primärschlüssel. Wie das im Quellcode aussieht, zeigt Listing 20.2 (Seite 170) für eine Entität zur Speicherung eines `Sectors` in einer Datenbank-Tabelle und Listing 20.3 (Seite 171) für eine Entität zur Speicherung einer `DevicePosition` in einem Document Store.

Für jeden Data Store werden separate Entitäten verwendet, da die unterstützten Primärschlüssel eine unterschiedliche Struktur aufweisen oder gar keine benötigt werden. Sie sind je in einem Subpackage untergebracht, aufgeteilt nach Back-end-Speicher. Details zum Mapping und allgemein zur Datenspeicherung hält Abschnitt 20.4 (Seite 180) bereit.

Klassenspezifikation

Die Klassenspezifikation kann der separaten JavaDoc-Dokumentation entnommen werden.

20.2.4 foundation

Klassenstruktur

Abbildung 20.13 (Seite 171) zeigt einen repräsentativen Auszug aus der Klassenstruktur des Foundation Layer.

```
@Document
public class DevicePosition {
    @Id private String id;
    @Indexed private String objectIdentifier;
    @Indexed private String type;
    @Indexed private String building;
    @Indexed private String floor;
    @Indexed private String sector;
    private String zone;
    private String accuracy;
    @Indexed private Date date;
}
```

Listing 20.3: Auszug aus der Entitätsklasse DevicePosition

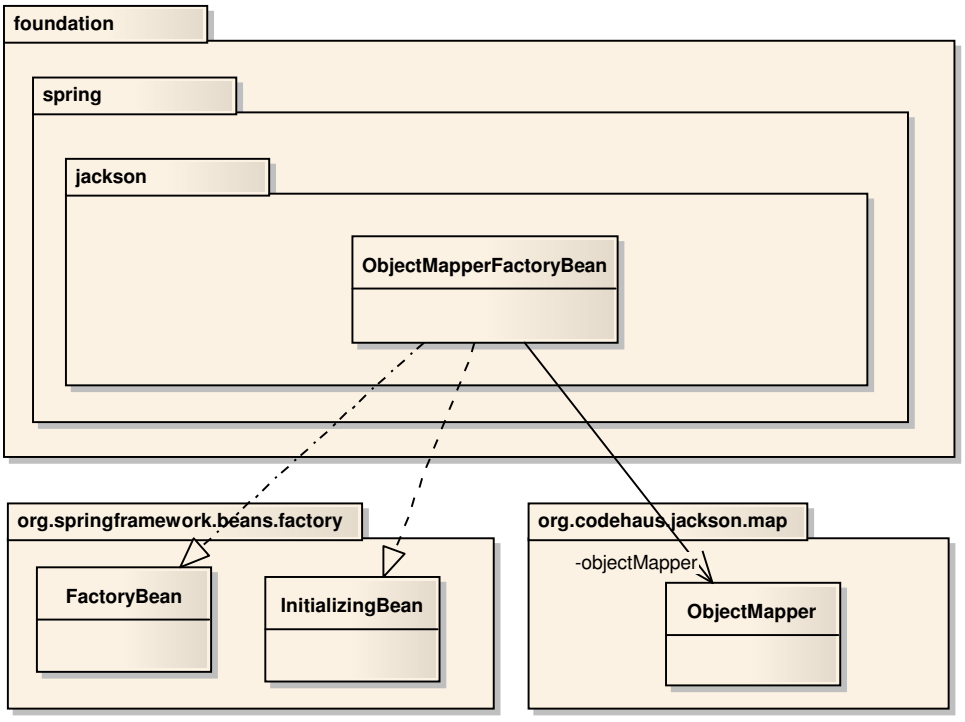

Abbildung 20.13: Auszug aus der Klassenstruktur des Foundation Layer

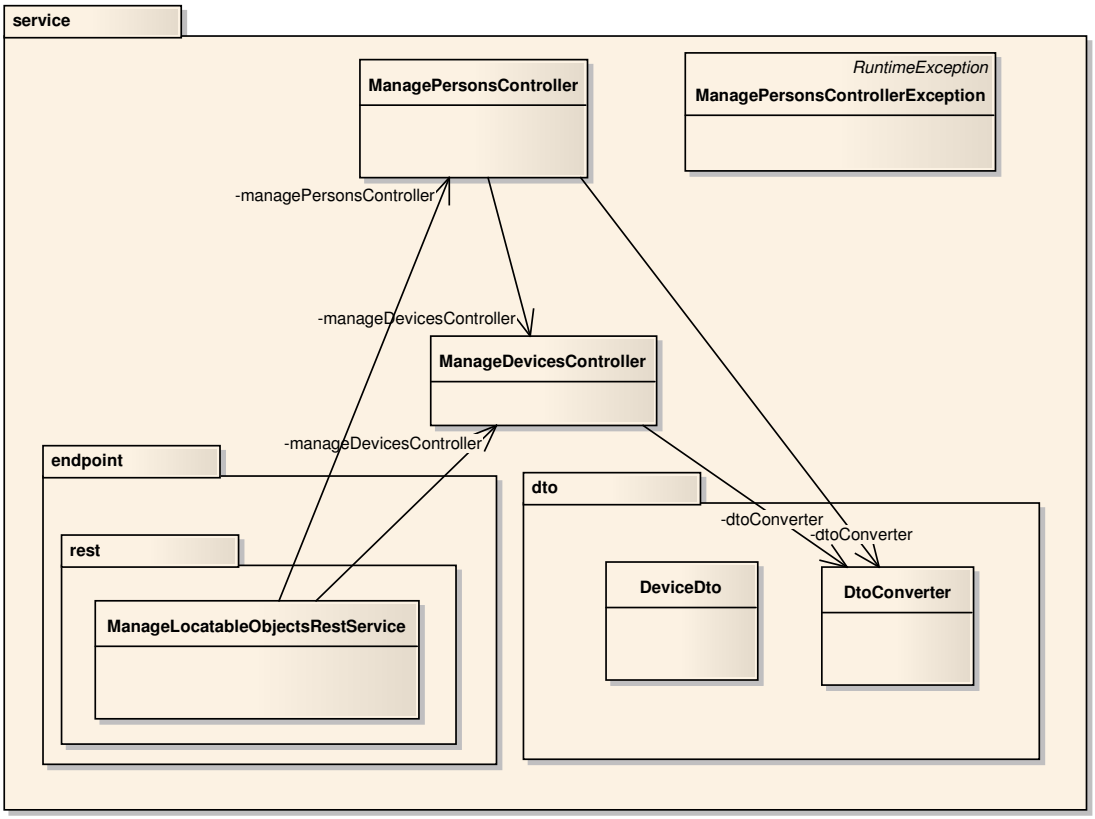

Abbildung 20.14: Auszug aus der Klassenstruktur des Service Layer

Architekturkonzepte

Das Paket `foundation` enthält Schnittstellen und Hilfsklassen zur Integration von und Interaktion mit externen Komponenten, auf die von der gesamten Anwendung zurückgegriffen werden kann. Ein typisches Beispiel zeigt Abbildung 20.13 (Seite 171), das eine `FactoryBean` zur Instanzierung des `ObjectMapper` aus der JSON-Bibliothek Jackson zeigt. Die `ObjectMapperFactoryBean` wird benötigt, um den `ObjectMapper` als Spring Bean laden und konfigurieren zu können. Weitere typische Vertreter aus dieser Schicht sind Konverter oder Adapter zum (Un-)Marshalling von Objekten.

Klassenspezifikation

Die Klassenspezifikation kann der separaten JavaDoc-Dokumentation entnommen werden.

20.2.5 service

Klassenstruktur

Abbildung 20.14 (Seite 172) zeigt einen charakteristischen Auszug aus der Klassenstruktur des Service Layer.

Architekturkonzepte

Der Service Layer dient als Eintrittspunkt für die System-Ereignisse und behandelt diese. Das dominierende Architekturmuster ist das Controller Pattern, das zur Ablaufkontrolle der Use Cases dient. Jeder Controller ist nach dem Use Case benannt, um den er sich kümmert. So heisst beispielsweise der Controller für den Use Case «Gerät lokalisieren», der Teil des «Orakel» ist, `LocaliseDeviceController`. Die Ausführung der eigentlichen Aufgaben wird an den Business Layer delegiert. Für jede Art, wie ein System-Ereignis eintreffen kann, existieren separate «Annahmestellen». Beispiele sind das Subpackage `endpoint`, da Endpunkte für Webservice-Protokolle anbietet, oder das Subpackage `jobs`, das vom Scheduler auslösbare Jobs enthält. Die in den Subpackages enthaltenen Klassen nehmen die Daten von aussen entgegen, konvertieren und validieren sie und reichen dann die komplette Ablaufausführung an die Controller weiter, sodass alle Controller grundsätzlich für die Reaktion auf jede Art von System-Ereignis verwendet werden können. Die Real Use Cases in Abschnitt 20.3 (Seite 173) zeigen die Funktionsweise noch detailliert.

Die Methoden der Controller retournieren im Gegensatz zu den tieferen Layers keine Entitätsobjekte, sondern Data Transfer Objects (DTOs), die im Subpackage `dto` angesiedelt sind. Die DTOs dienen dazu, die Daten aus den Entitäten so zusammenzufassen, dass beispielsweise weniger Webservice Requests zur Beschaffung der nötigen Daten zur Erledigung eines Use Case nötig sind. Ein gutes Beispiel ist das `PersonPositionDistanceDto` aus dem «Orakel». Es fasst eine Entität `PersonPosition` mit einem `BigDecimal` zusammen, um die Distanz zu einer `Person` gleich zusammen mit der `Person` zu übermitteln statt nur mit einer Referenz in Form eines Primärschlüssels. Damit spart man sich einen Webservice Request und das nochmalige Holen der Daten aus einer der Datenbanken. Weniger clever war die Entscheidung, dass die DTOs direkt von den Controllern retourniert werden, da so dieselben DTOs für alle Transportmedien verwendet werden müssten, die aber nicht zwingend dieselben Anforderungen haben, beispielsweise hinsichtlich Datenformaten. Dies bedeutet, dass ein weiterer Konvertierungsschritt notwendig wäre. Sauberer wäre es, dass die Controller weiterhin Entitäten retournieren und die Konvertierung dann direkt auf das richtige Format in den Schnittstellen zu den verschiedenen Transportmedien gemacht wird. Ein ähnliches Vorgehen ist auch zur Behandlung von Exceptions nötig, deren korrekte Behandlung bislang, insbesondere bei den JAX-RS Endpoints, vernachlässigt wurde.

Klassenspezifikation

Die Klassenspezifikation kann der separaten JavaDoc-Dokumentation entnommen werden. Die Spezifikation für die REST-Schnittstellen lässt sich mit der JavaDoc-Erweiterung jax-doclets[10] ebenfalls direkt aus dem Quellcode erzeugen.

20.3 Real Use Cases

Die Real Use Cases zeigen in White-Box-Manier, wie das System auf verschiedene Ereignisse reagiert und Objekte über die verschiedenen Schichten hinweg interagieren. Es wurden für diese Zwecke eine Handvoll repräsentativer Use Cases gewählt, die das Zusammenspiel der Architekturkomponenten zeigen oder wichtige Abläufe erklären.

[10] `http://www.lunatech-labs.com/open-source/jax-doclets` (abgerufen: 11. Juni 2011)

20.3.1 Adlerauge

In diesem Abschnitt soll illustriert werden, wie die Lokalisierung von Sensoren, Geräten und Personen abläuft und wo welcher Einfluss mit Business Rules genommen werden kann. Als Hilfestellung, um das Vorgehen nachzuvollziehen, dient Abbildung 20.15 (Seite 175). Sie zeigt in schematischer Art und Weise, wie die gesamte Lokalisierung abläuft. Sie wird von unten nach oben gelesen. Der gesamte Vorgang besteht aus drei Schritten:

1. Lokalisierung der Sensoren
2. Positionsbestimmung der Geräte und Auslösen etwaiger gerätebezogener Ereignisse
3. Positionsbestimmung der Personen und Auslösen etwaiger personenbezogener Ereignisse

Der Vorgang ist im Moment so ausgelegt, dass er mit jedem Lokalisierungswerkzeug funktioniert und möglichst robust ist. Eine mögliche Optimierung wird in Abschnitt 21.2.1 (Seite 202) beschrieben.

Lokalisierung der Sensoren

Der erste Schritt zur Lokalisierung von Geräten und Personen ist die Lokalisierung der einzelnen Sensoren (Ekahau-Tags etc.). Sie setzt sich aus zwei Etappen zusammen, die in Abbildung 20.15 (Seite 175) in Bereich 1 dargestellt sind:

1. Es erfolgt die Lokalisierung der Sensoren. Dabei werden zwei Varianten unterschieden. Einerseits sind da Sensoren, die auf Initiative des «Adlerauges» lokalisiert werden müssen, weil sie ihre aktuelle Position nicht von sich aus Melden. Zum Einsatz kommen dabei die «Pull-Konnektoren», die vom Scheduler des «Adlerauges» angeregt werden. Ein typisches Beispiel ist die LAN-Lokalisierung mithilfe von Cisco Works. Andere Sensoren melden ihre Position von sich aus an «Push-Konnektoren».
2. Die von den Sensoren gemeldeten Positionen werden von Landeskoordinaten, die die Konnektoren liefern, auf eine Position auf dem Campus (Gebäude, Stockwerk, Sektor) abgebildet und in einer verteilten Hash Table (Redis) gespeichert, wobei die Sensor-Adresse als Schlüssel verwendet wird. Zu jedem Zeitpunkt kann also die zuletzt gemeldete Position eines jeden Sensors der verteilten Hash Table entnommen werden.

Den Ablauf in der Software zeigt Abbildung 20.16 (Seite 176).

Man mag sich nun Fragen, weshalb die Sensor-Positionen in einer verteilten Hash Table gespeichert werden. Dies ist nötig, um die Positionsbestimmung der Geräte und Personen auf mehreren Servern gleichzeitig ausführen zu können, ohne sich überlegen zu müssen, welcher Server über das nötige Wissen verfügt. Mit Hilfe der verteilten Hash Table verfügt jeder Server über einen kompletten Überblick über alle aktuellen Sensor-Positionen und kann damit bei der Positionsbestimmung eines Geräts die Positionsmeldungen aller Sensoren, die mit ihm verbunden sind, aus der Hash Table holen. Speichert man für jeden Sensor statt eines einzelnen Wertes ausserdem eine Liste vergangener Positionsmeldungen, erhält man eine kleine Historie, die zur Positionsbestimmung herangezogen werden kann. Diese Möglichkeit spielt bei der Anwendung der Business Rules eine grosse Rolle, wie die nächsten beiden Abschnitte zeigen.

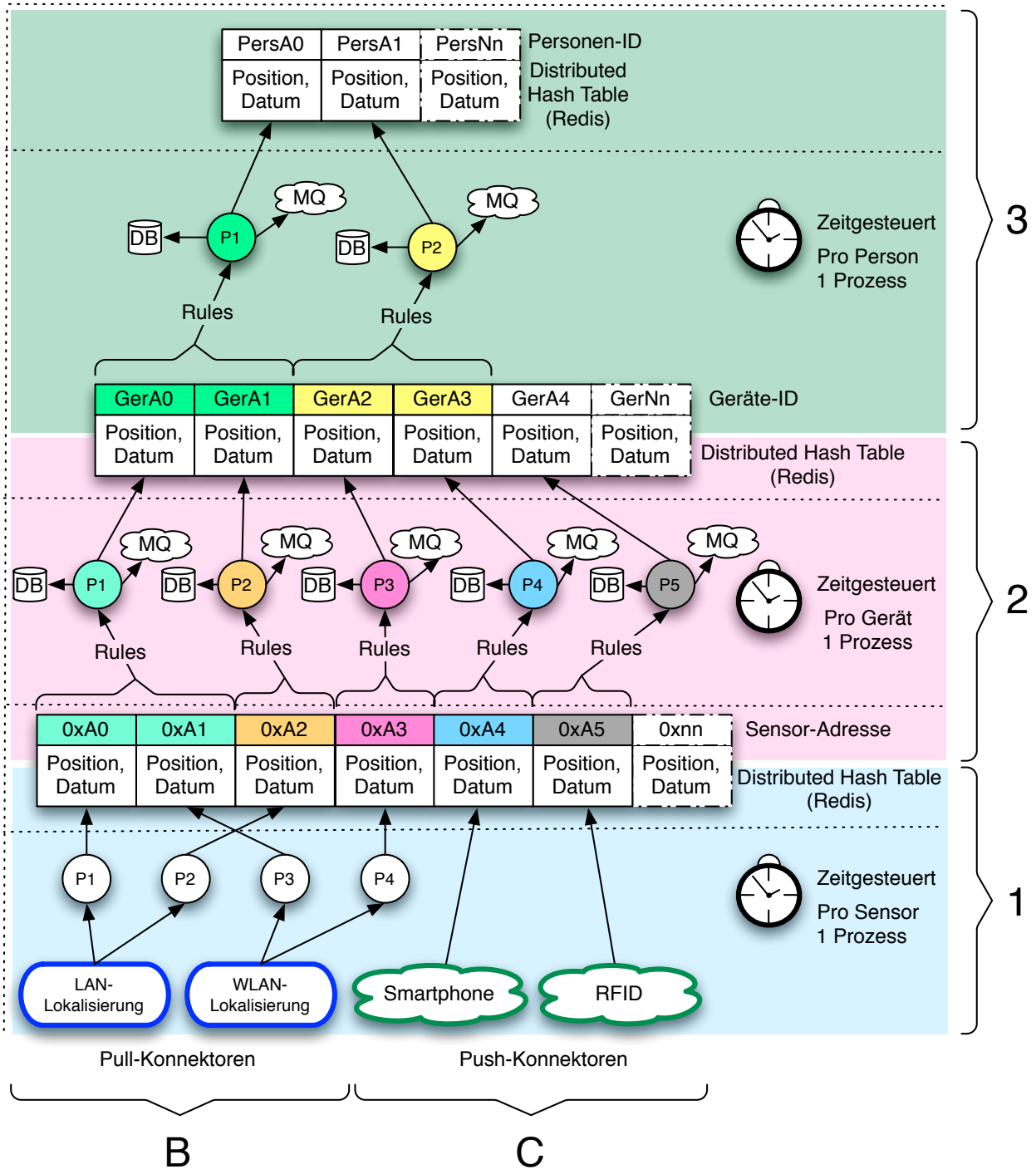

Abbildung 20.15: Die Abbildung zeigt schematisch den Ablauf der Lokalisierung. Von unten nach oben: Sensoren (1), Geräte (2) und Personen (3).

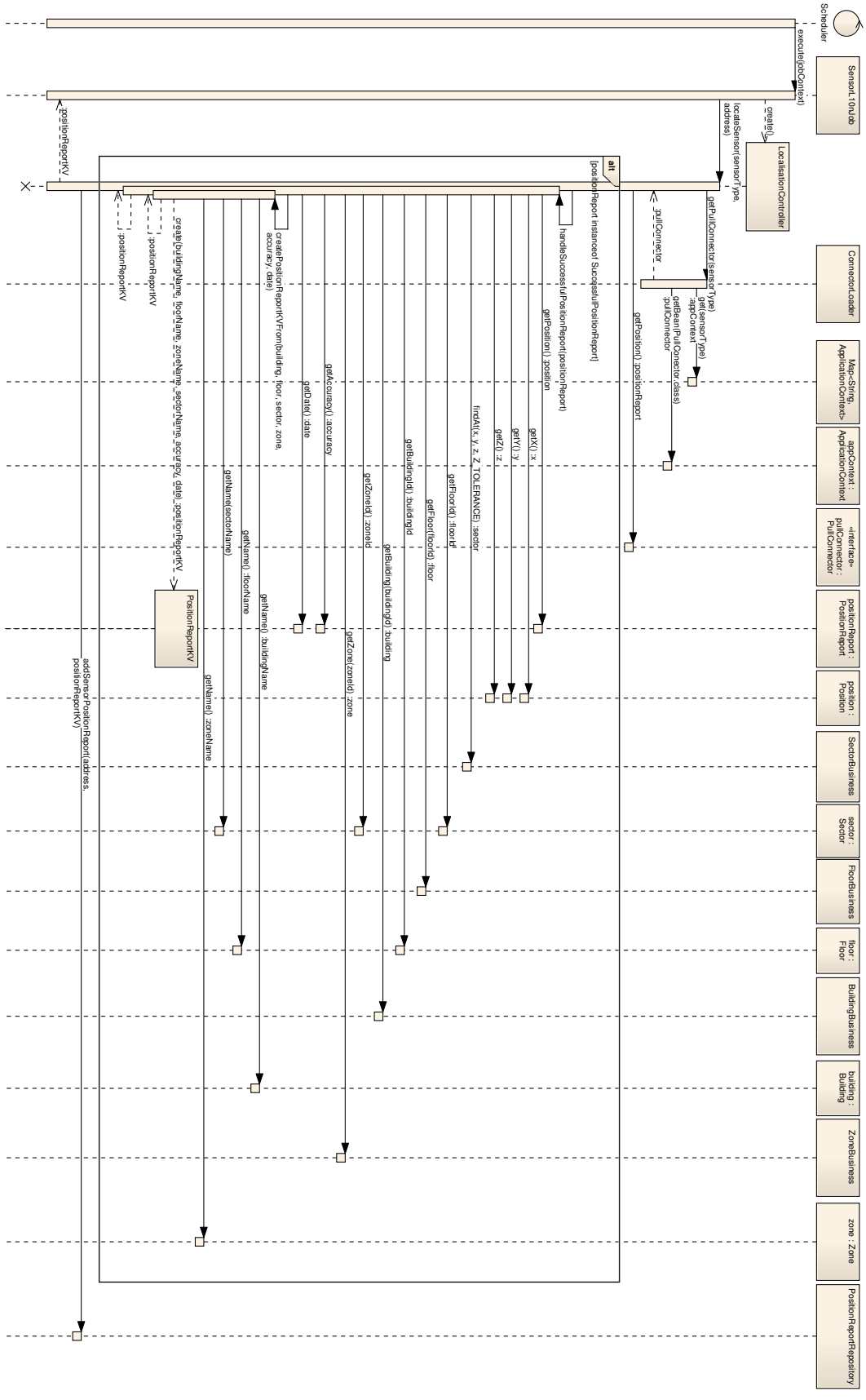

Abbildung 20.16: White-Box-Ansicht des Ablaufs der Sensorlokalisierung

Positionsbestimmung der Geräte

Aus den Positionsmeldungen der einzelnen Sensoren können nun die Positionen der mit ihnen verbundenen Geräte interpoliert werden und abhängig von den ermittelten Positionen Ereignisse ausgelöst werden. Doch wie wird nun aus *n* Sensor-Positionen die *richtige* Position des Geräts ermittelt? Dies ist in Abbildung 20.15 (Seite 175) in Bereich 2 zu sehen.

1. In regelmässigen Abständen (ausgelöst vom Scheduler) werden die Positionen aller Sensoren, die mit einem Gerät verbunden sind, aus der verteilten Hash Table geholt.
2. Die Positionen werden in die Rule Engine (Drools) als Fakten injiziert.
3. Die Rule Engine entscheidet nun mit Hilfe hinterlegter Regeln aufgrund der aktuellen Faktenlage, welches die *richtige* Position des Geräts ist. Zusätzlich entscheidet die Rule Engine, ob Ereignisse gefeuert werden sollen.
4. Die von der Rule Engine getroffenen Entscheidungen werden entnommen. Die Geräteposition wird in der Positionshistorie (MongoDB) gespeichert und etwaige Events gefeuert. Zusätzlich wird die Geräteposition wieder in der verteilten Hash Table abgelegt, wobei die Geräte-ID als Schlüssel verwendet wird. Dies wird gemacht, weil die Gerätepositionen benötigt werden, um die Positionen der Personen zu bestimmen. Ein weiterer Grund ist, einen schnellen Zugriff auf die letzten ermittelten Positionen eines Geräts zu haben («Kurzzeitgedächtnis» der Positionshistorie).

Den Ablauf in der Software zeigt Abbildung 20.17 (Seite 178).

Positionsbestimmung der Personen

Analog zu der Position der Geräte kann nun die Position der Personen bestimmt werden. Nur dass dieses Mal nicht die Positionen der Sensoren als Grundlage dienen, sondern diejenigen der Geräte, die die Personen auf sich tragen. Wie das funktioniert, ist in Abbildung 20.15 (Seite 175) in Bereich 3 zu sehen.

1. In regelmässigen Abständen (ausgelöst vom Scheduler) werden die Positionen aller Geräte, die eine Person auf sich tragen *kann*, aus der verteilten Hash Table geholt.
2. Die Positionen werden in die Rule Engine (Drools) als Fakten injiziert.
3. Die Rule Engine entscheidet nun mit Hilfe hinterlegter Regeln aufgrund der aktuellen Faktenlage, welches die *richtige* Position der Person ist. Zusätzlich entscheidet die Rule Engine, ob Ereignisse gefeuert werden sollen.
4. Die von der Rule Engine getroffenen Entscheidungen werden entnommen. Die Personenposition wird in der Positionshistorie (MongoDB) gespeichert und etwaige Events gefeuert. Zusätzlich wird die Personenposition wieder in der verteilten Hash Table abgelegt, wobei die Personen-ID als Schlüssel verwendet wird. Dies wird gemacht, um einen schnellen Zugriff auf die letzten ermittelten Positionen einer Person zu haben («Kurzzeitgedächtnis» der Positionshistorie).

Den Ablauf in der Software zeigt Abbildung 20.18 (Seite 179).

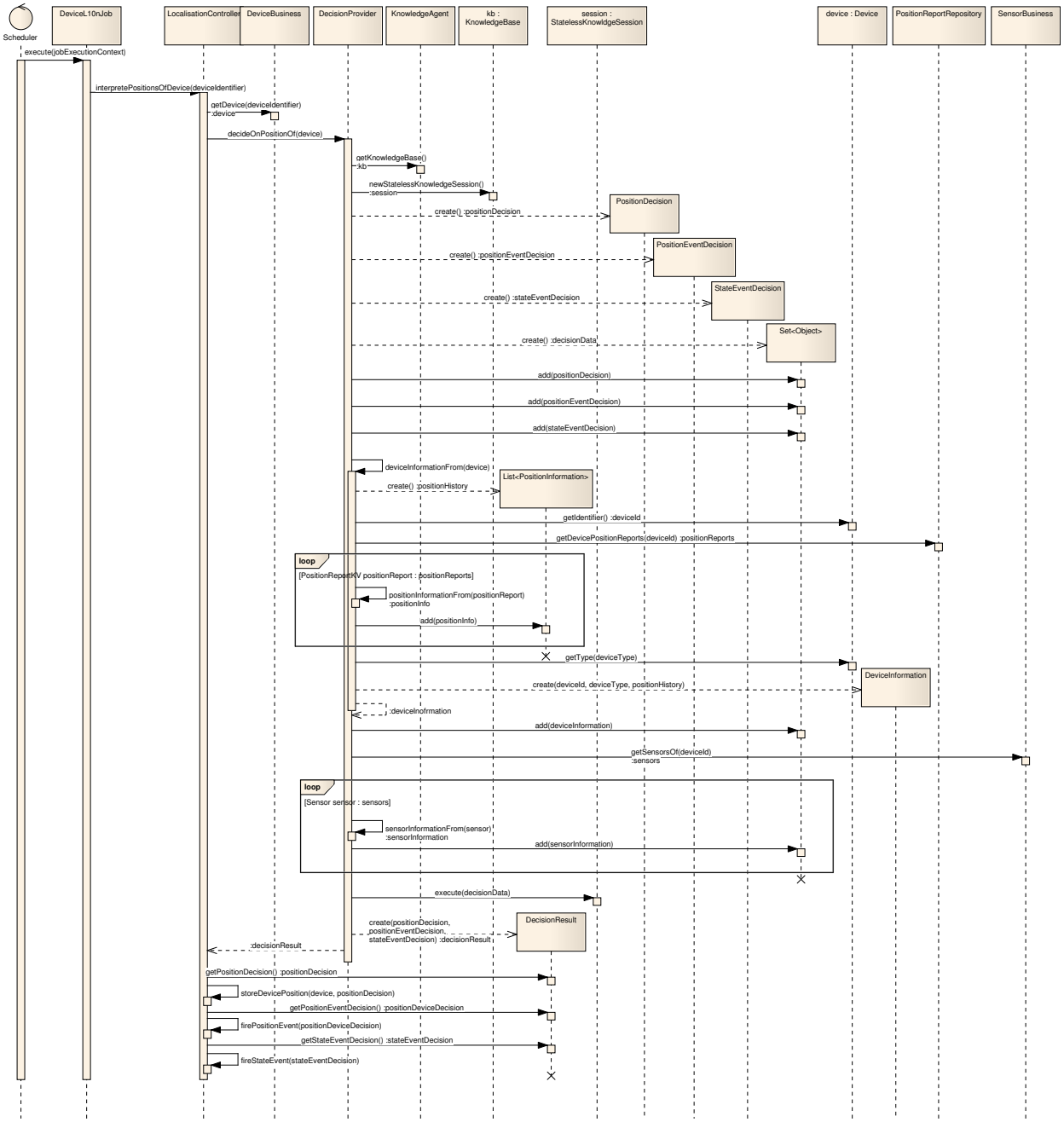

Abbildung 20.17: White-Box-Ansicht des Ablaufs der Gerätelokalisierung

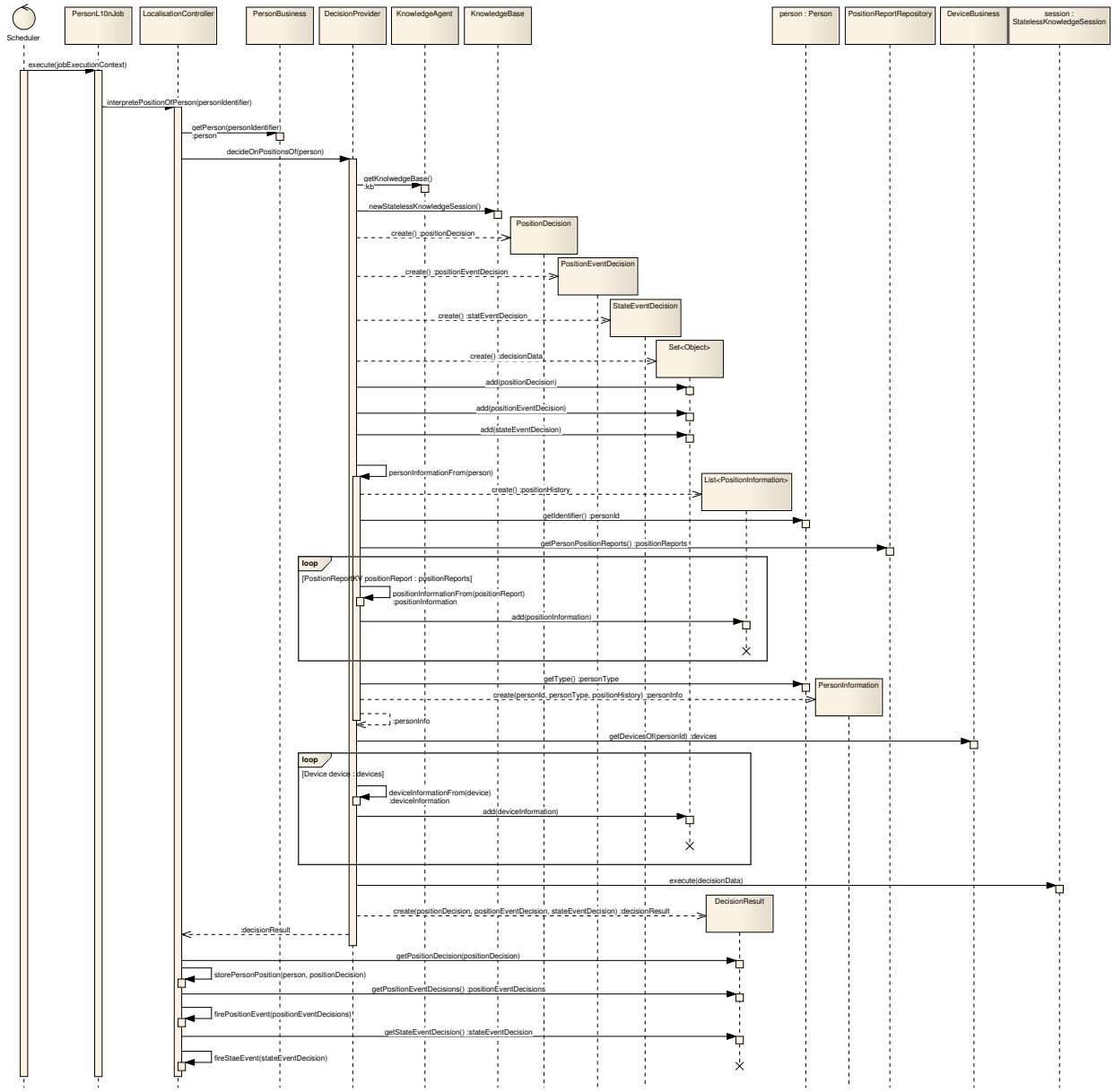

Abbildung 20.18: White-Box-Ansicht des Ablaufs der Personenlokalisierung

20.3.2 Orakel

Nächste Geräte eines bestimmten Typs

Dieser Use Case repräsentiert einen typischen Use Case für eine Interaktion über eine Webservice-Schnittstelle, wie er nicht nur im «Orakel», sondern auch im «Adlerauge» vorkommt und analog funktioniert. Zu sehen ist der erste Schritt in Abbildung 20.19 (Seite 181), bei dem die Position des Fragestellers bestimmt wird. Das Resultat bildet dann den Ausgangspunkt für Schritt 2, bei dem, wie in Abbildung 20.20 (Seite 182) zu sehen ist, die nächsten Geräte eines bestimmten Typs relativ zu einer vorgegebenen Position gefunden werden.

20.4 Daten

20.4.1 Datenbanken

Vorgehen

Für die Datenspeicherung wird im Gegensatz zu [1] nicht mehr nur ein Datenspeicher verwendet, sondern mehrere Arten. Das RDBMS (PostgreSQL, PostGIS) wurde durch einen Document Store (MongoDB) und einen Key Value Store (Redis) ergänzt. Hauptmotivation war, dass mit dem RDBMS durch die neu hinzugekommenen Aufgaben nicht mehr alle Anforderungen ausreichend gut abgedeckt werden konnten. Statt dessen werden nun mehrere spezialisierte Lösungen eingesetzt. MongoDB dient grob gesagt als intelligtenes Logfile (Speicherung der Positionshistorie, vollständige Nachvollziehbarkeit) und Redis zur Entkopplung von Architektur-Komponenten (Speicherung der aktuellen Positionsmeldungen zwischen den einzelnen Lokalisierungsschritten).

Schemata

Die Datenbank-Schemata wurden vom Domain Model (siehe Abschnitt 19, Seite 137, und 13, Seite 69) abgeleitet. Grössere Änderungen haben sich durch die Behandlung von Geodaten, der starken Nutzung vorberechneter Werte und der gewählten Abbildungsstrategie für Klassenhierarchien ergeben.

Geodaten Die Geodaten werden im Domain Model einzig durch das Objekt `Point` ausgedrückt und entsprechenden Relationen. So wird beispielsweise ein Ring eines Sektors durch eine Menge von Objekten des Typs `Point` beschrieben. Statt dieses Modell in der Datenbank nachzubauen, wurden die von PostGIS zur Verfügung gestellten geometrischen Datentypen wie `polygon` oder `linestring` verwendet, da auf ihnen Geometrie-Operationen (wie «Liegt X innerhalb von Y?») und spatiale Indices definiert sind. Zudem wurden zu vielen Objekten nicht direkt benötigte Geodaten gespeichert, die beispielsweise direkt für die Erzeugung von Visualisierungen verwendet werden können. Ein Beispiel sind die Tabellen `node` und `edge` in PostgreSQL, die um die Spalten `point` respektive `line` ergänzt wurden, die den bei der Geometrieerfassung gezeichneten Laufwegen entsprechen.

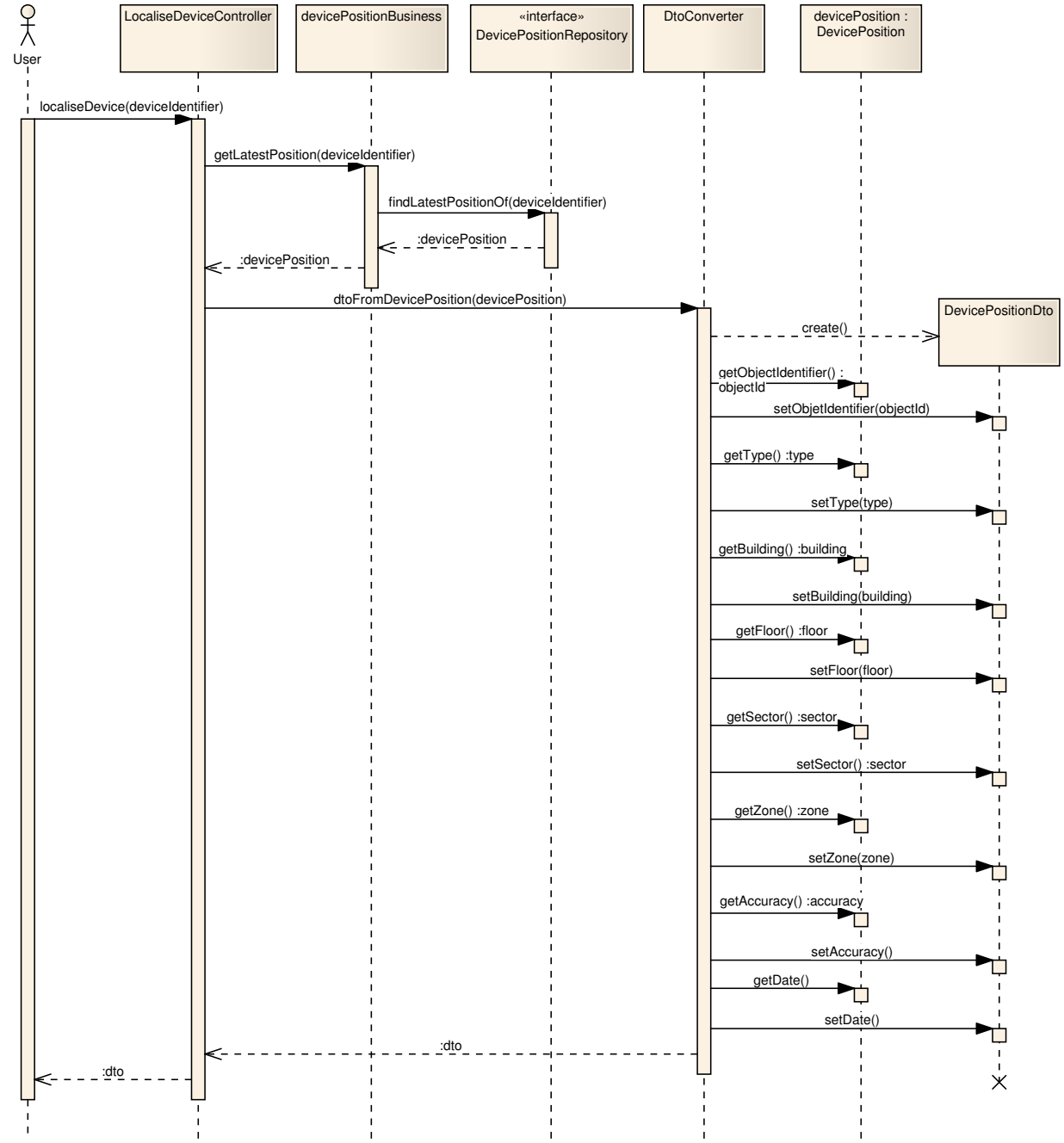

Abbildung 20.19: White-Box-Ansicht des Ablaufs der Lokalisierung der nächsten Geräte eines Typs. Die Abbildung zeigt den ersten Schritt, die Lokalisierung der Position des Anfragestellers.

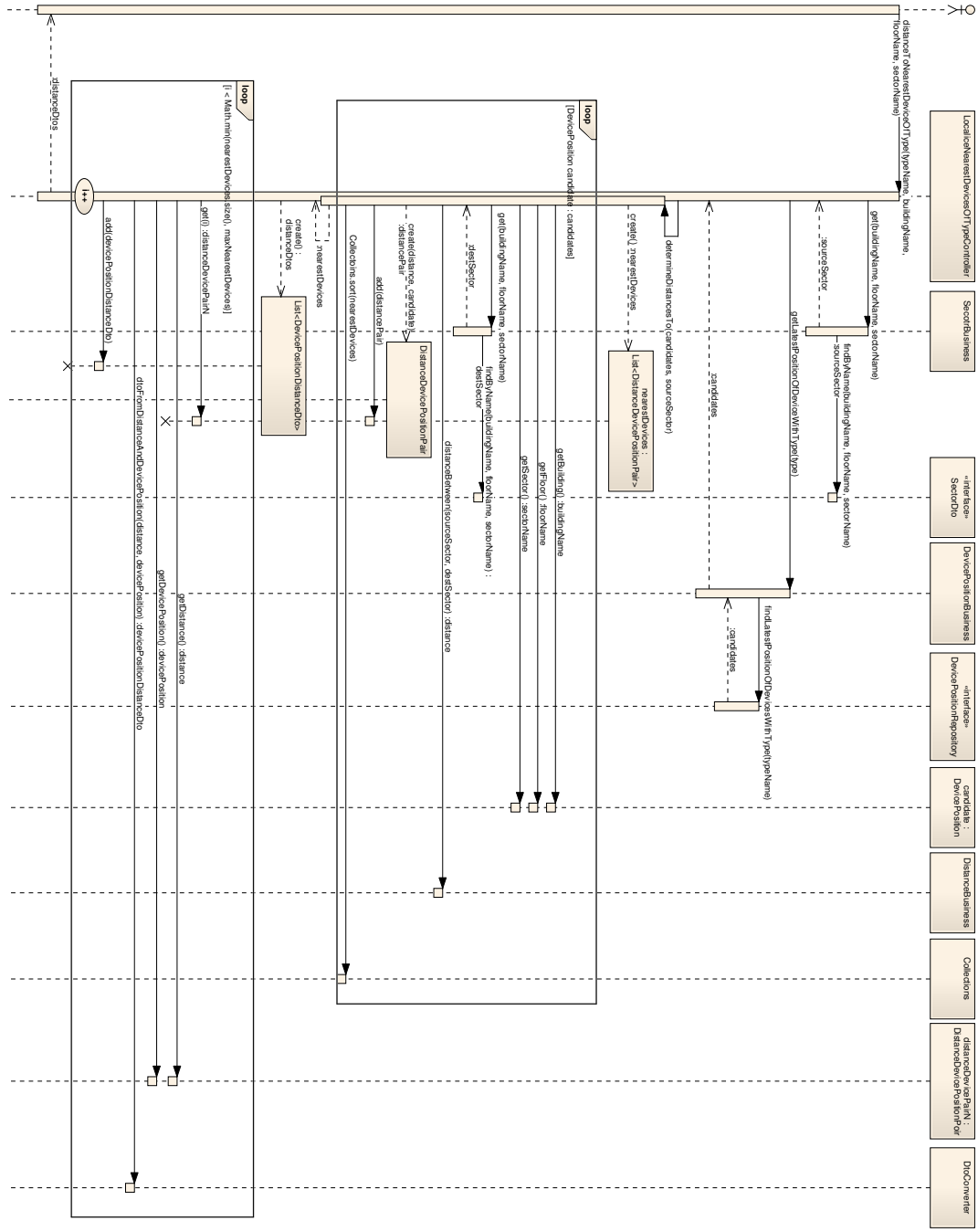

Abbildung 20.20: White-Box-Ansicht des Ablaufs der Lokalisierung der nächsten Geräte eines Typs. Die Abbildung zeigt den zweiten Schritt, die Ermittlung der nächsten Geräte eines Typs.

Vorberechnete Werte Das Datenmodell wurde um viele vorberechnete Werte ergänzt, um Abfragen zu vereinfachen oder die Abfragegeschwindigkeit zu erhöhen. Im Fall der in MongoDB gespeicherten Positionshistorie für Geräte (`DevicePosition`) und Personen (`PersonPosition`) wurden gleich sämtliche benötigten Daten zu jedem Datensatz aus PostgreSQL herauskopiert, da die Daten auch nach 2 Jahren noch analysierbar und auf dem gleichen Stand wie zum Speicherungszeitpunkt sein müssen. Ein anderer nennenswerter Fall ist die Tabelle `distance` in PostgreSQL, welche die kürzesten Wege zwischen allen Vertizes im Routing-Graph enthält. Die Berechnung des kürzesten Weges zwischen zwei Punkten ist damit nur noch ein Primary Key Lookup statt die zeit- und ressourcenaufwendige Ausführung des Dijkstra-Algorithmus.

Polymorphismus Auf Polymorphismus im eigentlichen Sinne wurde nach Möglichkeit verzichtet. Das bedeutet beispielsweise, dass nicht für jeden Gerätetyp (Röntgengerät, Perfusor, Blutdruckmessgerät etc.) eine separate Klasse existiert. Statt dessen ist eine generische Klasse `Device` vorhanden, die über ein Attribut `type` verfügt, über das der Typ des Objekts gewechselt werden kann. Analog wird unter anderem auch beim Geräte- beziehungsweise Personenstatus vorgegangen. Dieses Vorgehen hat mehrerer Gründe: So müssen bei der Nutzung von herkömmlichem JDBC ohne ORM die Objekte beim Laden aus der Datenbank verhältnismässig aufwendig über Reflection erzeugt werden. Ein weiterer Grund ist, dass beispielsweise die Menge vorhandener Geräte nicht konstant oder vorgegeben ist. Statt dessen sollen beispielsweise neue Gerätetypen zur Laufzeit hinzugefügt werden können.

Mapping

PostgreSQL Das Mapping von Entitäten auf Tabellenzeilen wird mit einem `RowMapper` aus Spring JDBC gemacht. Listing 20.4 (Seite 184) zeigt, wie ein Objekt aus einer Tabellen-Zeile wieder hergestellt wird. Sämtliche Objekthierarchien sind mittels Single Table Inheritance umgesetzt. Die Tabellen enthalten Spalten für alle Klassen der Hierarchie. Um die verschiedenen Klassen zu unterscheiden, existiert eine separate Typenspalte, mit deren Hilfe die Objekte unterschieden werden. Als Unterscheidungswert wird der Typenname der Klasse verwendet – zum Beispiel `Device` bei der Klasse `Device`. Zudem wurden Datenbank-Views angelegt, mit deren Hilfe nur die Objekte eines bestimmten Typs angezeigt werden können und man sich, solange man liest, nicht um die Objektunterscheidung kümmern muss.

MongoDB Das Mapping von Entitäten auf Dokumente wird mit Hilfe des `MappingMongoConverter` aus Spring Data erledigt. Dieser funktioniert ein wenig wie JPA und bildet die Entitäten automatisch auf Dokumente ab. Objekthierarchien werden über sogenannte Embedded Documents realisiert, wobei das Embedded Document die Spezialisierung repräsentiert.

Redis Bei Redis werden die Entitäten so wie sie sind, also direkt als Java Bean, serialisiert und als Wert abgelegt. Dies ist kein Nachteil, weil über Werte in Redis ohnehin nicht gut gesucht werden kann. Gesucht wird statt dessen über die Schlüssel. Das sind zwar nur normale Strings, lassen sich über einen cleveren Aufbau aber zum Suchen nach Untermengen verwenden oder um – zusammen mit Redis-Datenstrukturen wie Listen – mehrdimensionale Datenstrukturen aufzubauen. Listing 20.5 (Seite 184) zeigt beispielsweise einen Auszug aus dem `KeyManager`,

```
private static final class BuildingMapper
        implements RowMapper <Building> {

    @Override
    public Building mapRow(ResultSet rs, int rowNum) throws SQLException {
        Building building = new Building();
        building.setId(rs.getLong("id"));
        building.setName(rs.getString("name"));
        return building;
    }
}
```

Listing 20.4: `RowMapper`-Implementierung, die eine Zeile aus der Tabelle `building` auf die zugehörige Java Bean umsetzt

```
final class KeyManager {
    public static String sensorPositionReport(String sensorAddress) {
        return String.format("sensorPosReport:%s", sensorAddress);
    }

    public static String devicePositionReport(String deviceIdentifier) {
        return String.format("devicePosReport:%s", deviceIdentifier);
    }

    public static String personPositionReport(String personIdentifier) {
        return String.format("personPosReport:%s", personIdentifier);
    }
}
```

Listing 20.5: Die Klasse `KeyManager` verwaltet die Muster für Redis-Schlüssel, mit denen die Positionen von Sensoren, Geräten und Personen auseinander gehalten werden

der im «Adlerauge» verwendet wird, um die Positionen von Sensoren, Geräten und Personen auseinander zu halten.

Transaktionen

Transaktionsunterstützung bietet einzig PostgreSQL und wird dort auch genutzt. Bei den anderen Datenspeichern muss die Konsistenz manuell eingehalten werden.

Die Transaktionsunterstützung in PostgreSQL wird über das Spring-Modul TX gesteuert. Dies erfolgt über die Annotation `@Transactional` auf Controller- und Endpunkt-Methoden. Dies bedeutet, dass beispielsweise für jeden Webservice Request eine separate Transaktion verwendet wird. Dies ist ausreichend, da die Anwendung zustandslos ist und eine Transaktion deshalb ohnehin nicht mehrere Webservice Requests umspannen braucht.

Das dagegen noch fehlt, ist die Unterstützung von Optimistic Concurrency, um zu verhindern, dass ein Client unwissentlich die Änderungen eines anderen Clients überschreibt.

```
function() {
    emit(
        this.objectIdentifier,
        {
            _id: this._id,
            objectIdentifier: this.objectIdentifier,
            type: this.type,
            building: this.building,
            floor: this.floor,
            sector: this.sector,
            zone: this.zone,
            accuracy: this.accuracy,
            date: this.date
        }
    );
}
```

Listing 20.6: Die Map-Funktion gruppiert die Positionen aller Geräte nach Geräte-IDs.

Datenbankseitige Funktionalität

PostgreSQL Triggers, Stored Procedures und User Functions werden für administrative Aufgaben wie Sicherstellung der Datenintegrität oder Aktualisierung von vorberechneten Werten verwendet.

MongoDB MongoDB bietet die Möglichkeit, für Aggregationsoperationen und Datenanalyse serverseitig Map Reduce Jobs auszuführen, die in JavaScript definiert werden. Map Reduce Jobs werden im Rahmen des «Orakels» verwendet, beispielsweise um die aktuelle Position aller Geräte zu finden. Die zugehörige Map-Funktion zeigt Listing 20.6 (Seite 185), die Reduce-Funktion Listing 20.7 (Seite 186). Zu beachten ist, dass die Map-Reduce-Implementierung von MongoDB derzeit Single Threaded[11] ist. Dies hat negative Auswirkungen auf die Performance. Deshalb ist es wichtig, bis sich Verbesserungen diesbezüglich einstellen, die Map Reduce Jobs nur auf bereits mit einer herkömmlichen Query vorgefilterten Untermengen auszuführen. Ganz auf Map Reduce zu verzichten, ist aktuell nicht möglich, da MongoDBs Aggregationsoperationen hinsichtlich Resultatgrösse beschränkt sind und bei horizontaler Skalierung nicht verwendet werden können[12]. Da MongoDB recht schnell Fortschritte macht, kann es durchaus sein, dass die Limitierungen aufgehoben sind, bevor sie zu einem Problem werden. Andernfalls müsste überlegt werden, ob HBase trotz seiner Komplexität nicht vielleicht doch eine bessere Lösung wäre.

Redis Redis bietet zwar nicht die Möglichkeit, Benutzerfunktionen zu hinterlegen, hilft aber bei einigen administrativen Aufgaben mit nützlichen Kommandos mit. Im Rahmen der Anwendung werden sogenannte Capped Collections im Rahmen der Positionshistorie verwendet, also Collections, die über eine gewisse Grösse nicht hinauswachsen. Realisiert wird das über Listen, die

[11] `http://www.mongodb.org/display/DOCS/MapReduce#MapReduce-Parallelism` (abgerufen: 13. Juni 2011)
[12] `http://www.mongodb.org/display/DOCS/Aggregation#Aggregation-Group` (abgerufen: 13. Juni 2011)

```
function(key, values) {
    var result = {
        _id: undefined,
        objectIdentifier: undefined,
        type: undefined,
        building: undefined,
        floor: undefined,
        sector: undefined,
        zone: undefined,
        accuracy: undefined,
        date: 0
    };

    values.forEach(function(value) {
        if (result.date < value.date) {
            result._id = value._id,
            result.objectIdentifier = value.objectIdentifier;
            result.type = value.type;
            result.building = value.building;
            result.floor = value.floor;
            result.sector = value.sector;
            result.zone = value.zone;
            result.accuracy = value.accuracy;
            result.date = value.date;
        }
    });

    return result;
}
```

Listing 20.7: Die Reduce-Funktion geht alle von der Map-Funktion gelieferten Positionen durch und sucht für jede Geräte-ID nach der neusten.

```
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
@Path("/objects")
public class ManageLocatableObjectsRestService {

    @Resource
    private ManageDevicesController manageDevicesController;

    @GET
    @Path("/device/{identifier}")
    public DeviceDto getDevice(@PathParam("identifier") String id) {
        return manageDevicesController.getDevice(id);
    }
}
```

Listing 20.8: Auszug aus einem mit JAX-RS erstellten Webservice Endpoint

von links gefüllt werden (`LPUSH`) und dann mit `LTRIM` von links her gesehen auf eine bestimmte Grösse getrimmt werden.

20.4.2 Schnittstellen

Webservices

Die Webservice-Schnittstellen werden nach Möglichkeit im REST-Architekturstil erstellt, da dieser weniger schwergewichtig ist als SOAP. Sie werden mit Hilfe von JAX-RS realisiert, das ermöglicht, Ressourcen (URLs) mittels Annotationen auf Java-Klassen abzubilden. Listing 20.8 (Seite 187) zeigt ein Beispiel für den Endpunkt zur Verwaltung von lokalisierbaren Objekten, der Teil des «Adlerauges» ist. Die Methode `getDevices()` wird durch einen URL der Form `/objects/device/123` aktiviert, wobei `123` als Argument eingesetzt wird.

Als Datenformate werden grundsätzlich XML und bevorzugt JSON unterstützt, wobei über die HTTP-Header der Anfrage gesteuert wird, welches Format geliefert wird respektive werden soll.

Das (Un-)Marshalling erfolgt wie überall in der Software mittels JAXB und Jackson. Allerdings sind im Moment parallel noch Jackson-Annotationen nötig, da der Jackson-Provider der JAX-RS-Implementierung RESTeasy noch keine JAXB-Annotationen unterstützt.

JMS

Die JMS-Interaktion wird über das Spring-Modul JMS realisiert, da dieses einen Grossteil des Boilerplate Code versteckt, der bei der direkten Nutzung des JMS API anfällt. Versendet werden ausschliesslich Textnachrichten mit XML oder bevorzugt JSON Payload, damit diese auch von anderen Plattformen verstanden werden können – der Presence Manager basiert beispielsweise auf .Net. Das (Un-)Marshalling erfolgt wie überall in der Software mittels JAXB und Jackson.

20.5 Sicherheit

Die Software enthält zum aktuellen Stand keine Sicherheitsfunktionen, da diese bislang noch nicht gefordert wurden. Sie können aber analog zu den Transaktionen mit Hilfe von Spring Security problemlos über XML-Konfiguration nachgerüstet werden.

20.6 Benutzerschnittstelle

20.6.1 Realisierung

Die Benutzerschnittstelle zur Abfrage des «Orakels» wurde in HTML 5 und JavaScript realisiert. Zur Vereinfachung der JavaScript-Entwicklung wurden die JavaScript-Bibliotheken jQuery[13], jStorage[14] und Modernizr[15] verwendet. jQuery wurde gewählt, da sich damit der überschaubare Funktionsumfang schnell und einfach realisieren liess. Es handelt sich aber nicht mehr als um eine hastig zusammengesteckte Demo, um etwas zu haben, das man zeigen kann.

jQuery hat sich bereits während der Studienarbeit als alles andere als ideal erwiesen, da seine Stärken beim Anreichern bestehender Webseiten um Effekte, DOM-Manipulation und das Nachladen von Inhalten mittels Asynchronous JavaScript and XML (AJAX) liegen, nicht bei der Entwicklung von Rich Internet Applications (RIAs). Um Aufgaben wie Anwendungsstrukturierung, Unterstützung von Objektorinetierte Programmierung (OOP), Templating oder Internationalisierung kümmert es sich verständlicherweise nicht, was im aktuellen Fall ein Problem ist, wenn die gesamte Benutzeroberfläche rein clientseitig betrieben werden soll. Denn so erhält man schnell ein unwartbares Code-Geflecht oder muss viel Funktionalität hinzu programmieren. Zur Beginn der Bachelor-Arbeit haben wir deshalb die vielversprechendsten RIA-Frameworks (Google Web Toolkit, Sproutcore, Dojo Toolkit) angeschaut, sind aber leider nicht fündig geworden. Einer der Hauptkritikpunkte ist, dass alle kein deklaratives Templating der Benutzeroberfläche ermöglichen. Deshalb wurde sogar kurzfristig erwogen, die Benutzerschnittstelle mit Silverlight umzusetzen. Damit hätten wir uns aber das Problem eingehandelt, dass die Benutzeroberfläche nicht mehr auf möglichst unterschiedlichen Geräten zu benutzen ist.

Hinsichtlich der weiteren Entwicklung empfiehlt es sich daher, die Anforderungen an die Benutzerschnittstelle zu erfassen und die möglichen Technologien zur Umsetzung der Oberfläche mit HTML 5 und JavaScript gründlich zu analysieren und zu vergleichen, wozu ein paar wenige Tage pro Framework nicht ausreichen. Zusätzlich wurden mögliche Kandidaten wie Cappuchino[16] oder Ext JS[17] noch gar nicht angeschaut.

20.6.2 Lokalisierung des Fragestellers

Ein Problem, das während [1] noch aufgekommen ist, ist die Lokalisierung des Fragestellers, also der Person, die Anfragen über die Benutzeroberfläche stellt. Deren Position muss nämlich bekannt sein, um beispielsweise die nächsten Geräte eines bestimmten Typs zu bestimmen. Während [1] haben wir uns damit beholfen, dass man seine aktuelle Position manuell eingibt. Dies ist

[13] `http://jquery.org/` (abgerufen: 13. Juni 2011)
[14] `http://www.jstorage.info/` (abgerufen: 13. Juni 2011)
[15] `http://www.modernizr.com/` (abgerufen: 13. Juni 2011)
[16] `http://cappuccino.org/` (abgerufen: 13. Juni 2011)
[17] `http://www.sencha.com/products/extjs/` (abgerufen: 13. Juni 2011)

aber sehr unhandlich, wenn man sich ständig bewegt (beispielsweise mit einem Tablet PC) und sollte deshalb nur verwendet werden, wenn die Position nicht automatisch bestimmt werden kann.

In [1] wurden verschiedene Ansätze gewälzt, um die Position des Fragestellers zu bestimmen, die aber komplizierter sind als nötig. Die nun umgesetzte Lösung ist erheblich einfacher: Die Position des Anfragestellers wird ebenfalls über das Lokalisierungssystem ermittelt. Zu diesem Zweck muss einzig bei jeder Anfrage die ID des Geräts mitgeschickt werden, von dem aus die Anfrage gesendet wird. Dank Local Storage ist es ganz einfach, die Geräte-ID permanent im Browser zu hinterlegen. Es muss einfach einmal ein bestimmter URL aufgerufen werden, dem die Geräte-ID beispielsweise als Query-Parameter angehängt wird. Dieser bleibt dann permanent über Tage und Wochen hinweg im Browser gespeichert.

Das Aufrufen des URL kann automatisiert werden, indem beispielsweise ein Icon auf dem Desktop für den Browser abgelegt wird, das den aufzurufenden URL als Kommandozeilenargument mitgibt. Ein Beispiel für den Internet Explorer zeigt Abbildung 20.21 (Seite 190) – bei anderen Webbrowsern funktioniert dies analog. Das Icon kann beispielsweise beim Device Provisioning erzeugt werden. Ist die Geräte-ID (typischerweise eine Inventarnummer) zu diesem Zeitpunkt nicht bekannt, kann noch immer ein Script hinterlegt werden, das beim Gerätestart das Icon erstellt. Dieses kann die Geräte-ID via «Orakel» über die im «Bürokraten» erfassten Geräteinformationen ermitteln, indem das «Orakel» nach dem Gerät gefragt wird, an dem ein Sensor mit der MAC-Adresse der Station hängt.

20.7 Konnektoren

Die Konnektoren zu den Lokalisierungswerkzeugen sind ein Teil der Software, der grossen Änderungen unterworfen ist, weil unterschiedliche Bezugssysteme und eine Vielzahl von Lokalisierungswerkzeugen unterschiedlicher Hersteller unterstützt werden müssen. Deshalb wurde entschieden, die Konnektoren als vom Lokalisierungssystem unabhängig entwickelbare Komponenten zu realisieren. Der allgemeine Kontrakt zwischen Lokalisierungssystem und den einzelnen Konnektoren ist deshalb auch in Abschnitt 22 (Seite 209) beschrieben. An dieser Stelle wird nur dokumentiert, wie das Laden der Konnektoren im «Adlerauge» funktioniert.

20.7.1 Ausgangslage

Innerhalb des «Adlerauges» wird der IoC Container von Spring für Dependency Injection und Konfiguration von Collaborators verwendet, da er unter anderem das Testing sehr erleichtert. Da die einzelnen Konnektor-Plug-ins in der Regel ebenfalls Collaborators und Konfiguration benötigen, wurde beschlossen, die Nutzung des IoC Container von Spring auch auf die Konnektor-Plug-ins zu erweitern.

Damit sind nun aus Sicht des «Adlerauges» und des zu ladenden Konnektor-Plug-ins zwei IoC Container vorhanden: der des «Adlerauges» und der des Konnektors. Da sowohl das «Adlerauge» Services vom Konnektor bezieht (`PullConnector`) und der Konnektor vom «Adlerauge» («PushResultHandler»), müssen beide Kontexte miteinander verbunden und die Abhängigkeiten wechselseitig befriedigt werden. Eine Herausforderung stellt hierbei die Anforderung dar, dass das «Adlerauge» erst beim Start die vorhandenen Konnektoren dynamisch einlesen, laden

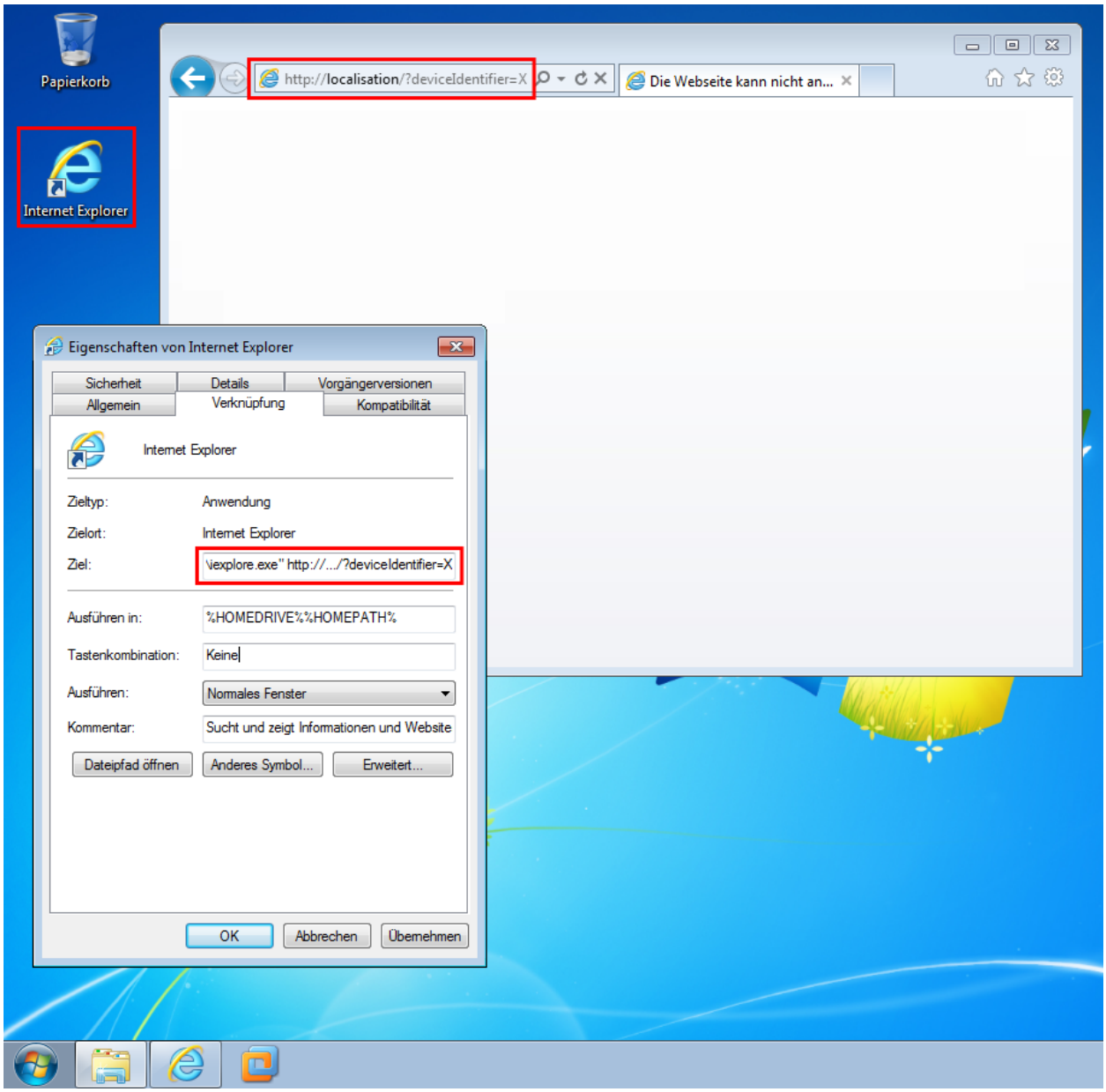

Abbildung 20.21: Durch Einbettung der Geräte-ID in den URL zum Aufruf der Benutzerschnittstelle kann die Geräte-ID im Browser automatisch registriert und in Zukunft zur Lokalisierung des Fragestellers verwendet werden.

und konfigurieren soll. Dies bedeutet, dass das «Adlerauge» kein Wissen über die Konnektoren besitzt ausser den gemeinsamen Contract und wo es die Konnektoren zu laden hat.

20.7.2 Ladestrategien

Da es ausreicht, die Konnektoren wechseln zu können, wenn das Lokalisierungssystem heruntergefahren ist, wurde von Anfang auf die Nutzung des OSGi-Programmiermodells verzichtet, auch wenn Spring OSGi-fähig ist, um nicht noch mehr Komplexität einzuführen. Blieben damit noch die in den nächsten Abschnitten beschriebenen Ladestrategien.

Globaler ApplicationContext

Dabei werden die `ApplicationContexts` der Konnektoren in den `ApplicationContext` des Lokalisierungsservice importiert (mittels `<import/>`), sodass ein einziger, globaler `ApplicationContext` entsteht. Dieses Vorgehen hätte den Vorteil, dass die wechselseitigen Abhängigkeiten automatisch durch Spring hätten befriedigt werden können. Der grosse Nachteil ist, dass es dann zu Kollisionen zwischen den Konnektoren kommen kann, wenn Beans mit identischem Namen definiert werden, die sich dann unter Umständen auch nicht nach Typ auseinander halten lassen. Da jeder Konnektor beispielsweise eine Referenz zu einer MongoDB-Datenbank besitzt, ist diese Gefahr durchaus real und hätte die Einhaltung vieler Konventionen erfordert, weshalb auf diesen Ansatz verzichtet worden ist.

Kontext-Hierarchie

Eine Variante, die die Möglichkeit der Konflikte zwischen Konnektoren eliminiert, ist die Einführung einer Kontext-Hierarchie, wie sie schematisch von Abbildung 20.22 (Seite 192) illustriert wird. Bei der Kontext-Hierarchie hat jeder Konnektor seinen eigenen `ApplicationContext`, wobei der Kontext des «Adlerauges» als übergeordneter (Parent) Kontext definiert ist. So wäre es für die Konnektoren (Childs) möglich, Services vom «Adlerauge» zu beziehen. Da die Verbindung unidirektional ist (vom Child zum Parent), ist es für das «Adlerauge» nicht möglich, Services von den Konnektoren zu beziehen. Dafür kommt nun der `ConnectorLoader` ins Spiel. Dieser ist Teil des «Adlerauges» und lädt die Kontexte der Konnektoren, sodass er Beans aus ihnen herausholen und dem «Adlerauge» zur Verfügung stellen kann. Dies war lange Zeit die präferierte Variante. Zuerst wurde versucht, dies mit einem `ContextSingletonBeanFactoryLocator` aus Spring zu lösen, was aber fehlgeschlagen ist, da sich mit diesem nicht die Verbindung zum Parent-Kontext herstellen lassen konnte, da der Kontext eines SpringMVC Servlet nicht benannt ist.

sDer nächste Versuch war, die Funktionalität des `ContextSingletonBeanFactoryLocator` manuell nachzubauen, indem die Kontexte einer nach dem anderen programmatisch eingelesen und dann die Verbindung zum Parent-Kontext etabliert wurde. Dies hat zwar einwandfrei funktioniert, leider wurde aber ein wichtiger Punkt übersehen: Die Push-Konnektoren können bei diesem Vorgehen keine Servlets exponieren, weil das `DispatcherServlet` von Spring, das im Kontext des «Adlerauges» lebt, in der Hierarchie nicht nach unten schauen und sie deshalb nicht sehen kann. Und ein neues Servlet zu erzeugen, hätte einen zweiten `ApplicationContext` im Konnektor erstellt, der aber nicht mit der Hierarchie verbunden und auch nicht zu verbinden gewesen wäre. Die Idee der Kontext-Hierarchie war damit gescheitert.

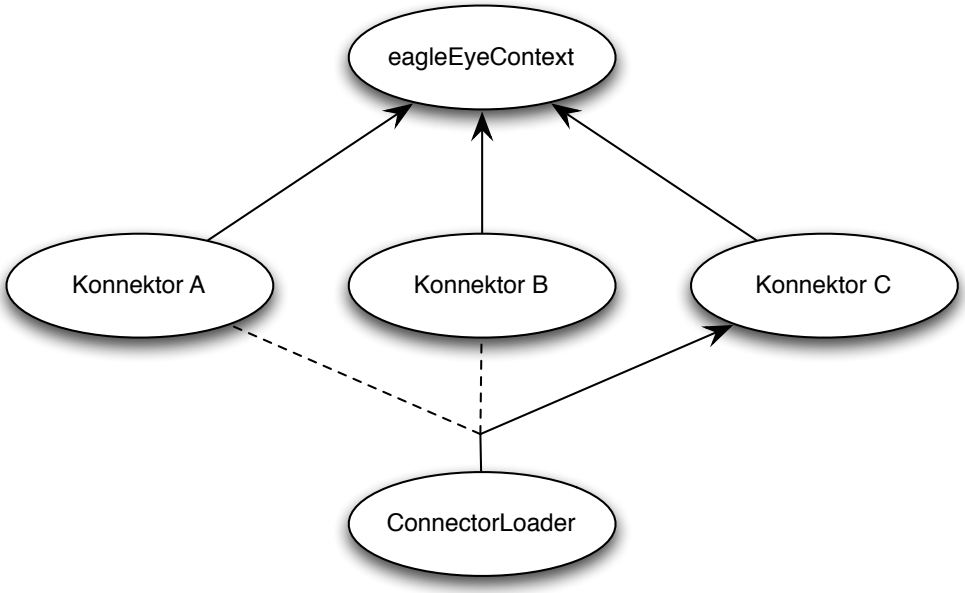

Abbildung 20.22: Bei einer Kontext-Hierarchie wären die Konnektoren untereinander isoliert. Vom Eltern-Kontext könnten sie aber Services beziehen. Die Verbindung zum Eltern-Kontext würde über den `ConnectorLoader` hergestellt.

Unabhängige Kontexte

Was blieb, war die Verwendung mehrerer unabhängiger Kontexte mit manueller Verarztung der Abhängigkeiten, wie sie uns von Wolfgang Giersche von Zühlke Engineering vorgeschlagen wurde. Statt die Abhängigkeiten automatisch von Spring mit Hilfe der Kontext-Hierarchie befriedigen zu lassen, wird dies manuell erledigt, indem jeder Kontext nach Klassen abgesucht wird, die bestimmte Marker-Interfaces tragen. Mit Hilfe dieser Marker-Interfaces können sie signalisieren, welche Art von Service sie benötigen, der ihnen dann über einen gewöhnlichen Setter injiziert wird. Den Ablauf illustriert Abbildung 20.23 (Seite 193).

Die Frage der zu exportierenden Servlets ist damit noch nicht gelöst, was einen weiteren Schritt erfordert. Damit das im Kontext des «Adlerauges» lebende `DispatcherServlet` die Servlets vom Konnektor sieht, müssen diese in den Kontext importiert werden. Dies bedeutet zwar, dass sich die Konnektor-Entwickler an ein festes Namensschema halten müssen, ist für eine einzelne Klasse aber noch vertretbar. Die zu importierenden Servlets werden an einem weiteren Marker-Interface (z.B. `JaxRsPushConnector` für JAX-RS Endpoints) erkannt, wobei Classpath Scanning verwendet wird, um sie aufzuspüren. Nun müssen die Servlets aber noch eine Verbindung zu ihrem Konnektor-Kontext erhalten, um die empfangenen Daten zur Verarbeitung weiterzuleiten. Dazu dient das Marker-Interface `RequiresDelegate<T>`, das analog `RequiresService<T>` arbeitet, nur in der umgekehrten Richtung. Abbildung 20.24 (Seite 193) zeigt den prinzipiellen Ablauf. Zu beachten ist, dass die im Kontext des «Adlerauges» lebenden Servlets keine Services von ihm beziehen können und auch nicht vom Konnektor-Kontext. Ebenso ist ihr Verdrahten im Konntektor-Kontext *verboten*. Die einzige Aufgabe der Servlets ist, die Daten entgegen zu nehmen und dem Delegate zur Verarbeitung zu übergeben.

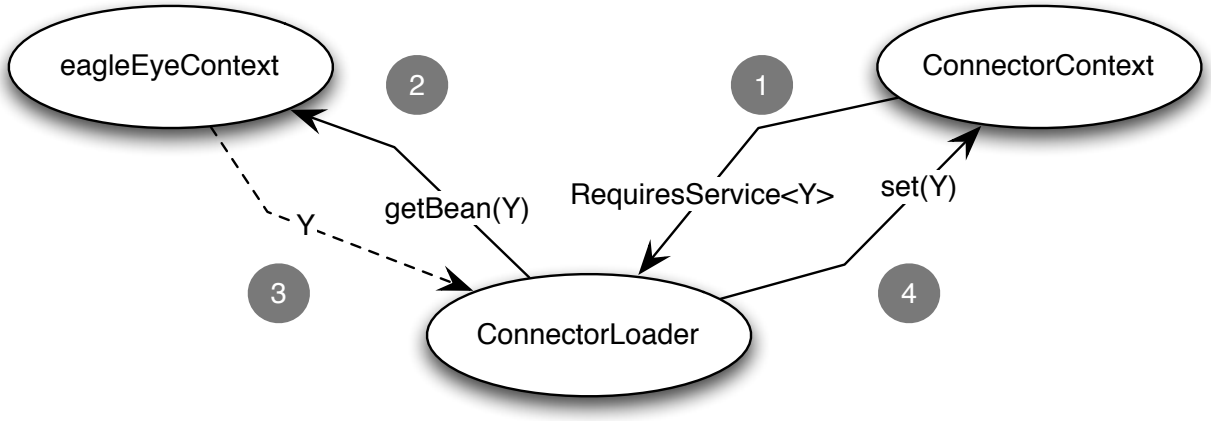

Abbildung 20.23: Die Abbildung zeigt, wie Konnektor-Klassen, die das Marker-Interface `RequiresService<T>` tragen, mit Services aus dem Kontext des «Adlerauges» versorgt werden.

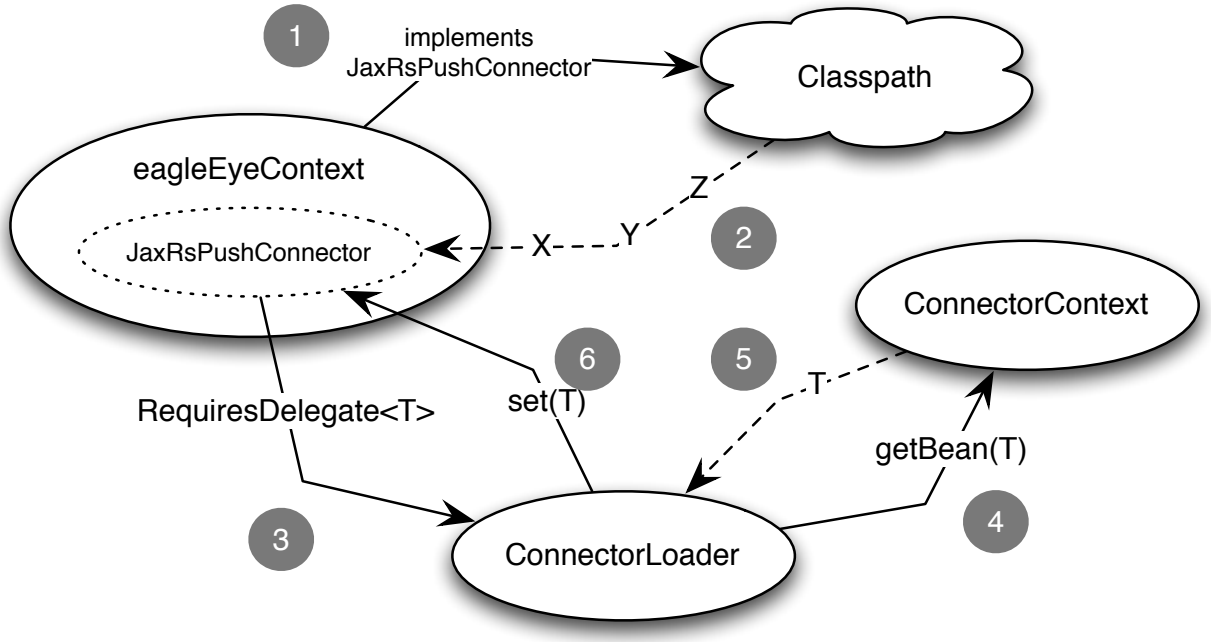

Abbildung 20.24: Die Abbildung zeigt, wie exportierte Servlets, die das Marker-Interface `RequiresDelegate<T>` tragen, mit Delegates aus dem Kontext des «Konnektors» versorgt werden.

Zu beachten ist, dass das komplizierte Vorgehen mit `RequiresDelegate<T>` nur notwendig ist, wenn eine Komponente im Kontext des «Adlerauges» leben muss. Für die meisten anderen Arten von Push-Konnektoren, beispielsweise JMS `MessageListener`, ist das nicht nötig, weil es ausreicht, sie ausschliesslich im Kontext des Konnektors zu registrieren.

Ohne Spring

Natürlich wäre es auch denkbar gewesen, die Konnektoren nicht mit Spring zu realisieren und statt dessen ein paar Abstract Factories zu verwenden. Dies hätte die Aufgabe aber nicht einfacher gemacht, weil die Servlets weiterhin hätten in den Kontext des «Adlerauges» importiert und die Abhängigkeiten der Klassen hätten manuell verarztet werden müssen. Spring hat diese Aufgabe in Realität erheblich einfacher gemacht, weil die nötige Infrastruktur wie Classpath Scanning bereits vorhanden ist. Der ganze Mechanik im `ConnectorLoader` ist lediglich ein paar Dutzend Zeilen lang, wobei ein grosser Anteil noch auf Logging entfällt.

20.7.3 Ladevorgang

Sämtliche Konnektoren werden beim Start des Servlet Container – respektive des Kontexts des «Adlerauges» – automatisch geladen und im `ConnectorLoader` registriert. Pull-Konnektoren – das sind solche, die das Interface `PullConnector` implementieren und vom «Adlerauge» aktiviert werden – können über `ConnectorLoader` bezogen werden. Sie werden dabei nach Typ aus dem Kontext des Konnektors herausgefischt. Die Push-Konnektoren – das sind solche, die das Interface `PushConnector` implementieren – werden ohnehin «von aussen» (Servlet, `MessageListener` etc.) aktiviert, sodass man sich nicht weiter um sie zu kümmern braucht.

20.8 Scheduling

Zur regelmässigen Ausführung oder zur Ausführung von Aufgaben zu bestimmten Zeitpunkten, was bei der Lokalisierung der Sensoren, Geräte und Personen der Fall ist, wurde ein Scheduler in das «Adlerauge» integriert. Zum Einsatz kommt der Quartz Enterprise Job Scheduler von Terracotta in der Open Source Edition.

20.8.1 Integration

Der Scheduler wird innerhalb des «Adlerauges» ausgeführt, nicht separat. Dies bedeutet, dass jede Instanz des «Adlerauges» über einen lokalen Scheduler verfügt. Sie synchronisieren sich selber über einen zentralen Job Store.

20.8.2 Lifecycle Management

Scheduler

Da der Scheduler Teil des «Adlerauges» ist, ist er auch an den Lifecylce des «Adlerauges» gebunden. Das heisst, wenn das «Adlerauge» ausgebracht wird, wird der Scheduler automatisch gestartet. Analog wird er gestoppt, wenn das «Adlerauge» wieder eingeholt wird. Realisiert wird dies mit der `SmartLifecycle`-Infrastruktur von Spring.

Jobs

Trifft der Zeitpunkt zur Ausführung einer Aufgabe an, wird ein Job vom Scheduler erzeugt und ausgeführt. Bei den Jobs handelt es sich um Java-Klassen, welche das Interface `Job` implementieren. Der Lifecylce der Job-Objekte wird deshalb von Quartz verwaltet, nicht von Spring. Den Jobs wird aber ein `ApplicationContext` über einen Setter injiziert, damit sie die Ausführung von Aufgaben an den von Spring verwalteten Teil der Anwendung übergeben können.

20.8.3 Job Store

Als Job Store kommt im Moment PostgreSQL zum Einsatz, da der geclusterte Einsatz von Quartz im Moment nur mit einem RDBMS möglich ist. Problematisch hierbei ist, dass nach jeder Job-Ausführung Daten zurück ins RDBMS geschrieben werden müssen. Je nach Menge von zu lokalisierenden Objekten und Kürze des Lokalisierungsintervalls kann das RDBMS schnell zum Flaschenhals werden. Durch Implementierung eines vorgegebenen Interfaces ist es aber auch möglich, andere Datenbanken als Job Store zu verwenden – beispielsweise MongoDB.

Zu beachten ist, dass Quartz aktuell nicht dieselbe JDBC Data Source verwendet wie das «Adlerauge» selber. Das heisst, das Transaktionsmanagement erstreckt sich nicht auf Quartz.

20.9 Business Rules

Die Anwendung von Business Rules im «Adlerauge» wurde in eine Rule Engine ausgelagert, damit diese von Fachexperten und einfach geändert werden können, ohne dass der Code dazu angefasst werden muss. Zum Einsatz kommt dabei Drools. Dieses entscheidet mit Hilfe hinterlegter Regeln bei der Lokalisierung von Geräten und Personen, wo sich die Geräte beziehungsweise Personen befinden und ob Ereignisse eingetreten sind, über die interessierte Systeme informiert werden müssen.

20.9.1 Integration

Drools wurde über die vom Projekt angebotene Spring-Unterstützung integriert. Jedes «Adlerauge» verfügt so über eine lokale Drools-Instanz. Da das ganze «Adlerauge» zustandslos ist, wurde Drools ebenfalls zustandslos integriert. Dies bedeutet, dass jede Entscheidung unabhängig von vorhergehenden Entscheidungen getroffen wird.

20.9.2 Rule Authoring

Regeln können von Drools her als Regeldatei, Entscheidungstabelle oder mittels einer Domain Specific Language (DSL) beschrieben werden. Aktuell werden Regeldateien verwendet, da diese während der Entwicklung am einfachsten anzupassen sind. Es kann aber jederzeit auf eine andere Methode gewechselt werden, indem ein paar wenige Zeilen Code angepasst werden. Dafür sei an die Dokumentation von Drools verwiesen.

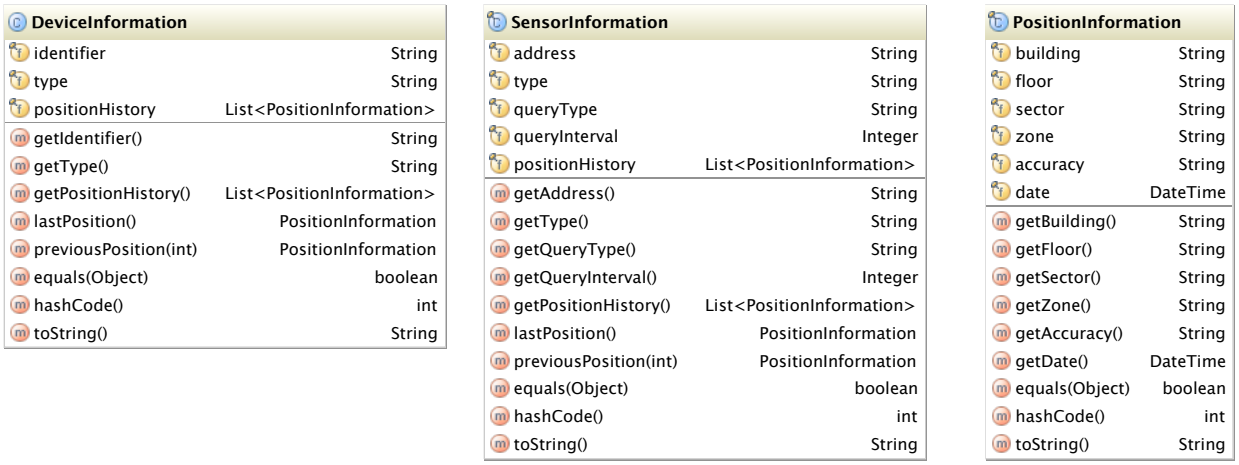

Abbildung 20.25: Die Abbildung gibt einen Überblick über die Objekte, die als Fakten im Entscheidungsprozess über Gerätepositionen und Ereignisse mit Gerätebezug zur Verfügung stehen.

20.9.3 Fakten und Entscheidungen

Es werden zwei verschiedene Regelsätze erwartet: Einer für alle Geräte und und einer für alle Personen. Für jeden Entscheidungsdurchgang werden Drools eine Reihe von Fakten injiziert. Abbildung 20.25 (Seite 196) zeigt die Fakten, die für die Entscheidungen über Gerätepositionen und -Ereignisse zur Verfügung stehen, Abbildung 20.26 (Seite 197) die Fakten für Entscheidungen über Personenpositionen und -Ereignisse.

Die getroffenen Entscheidungen werden daraufhin in Drools in Entscheidungsobjekte abgepackt. Beispiel für so eine Regel zeigt Listing 20.9 (Seite 197). Die Entscheidungsobjekte, in die die Entscheidungen abgepackt werden, zeigt Abbildung 20.27 (Seite 198). Die Entscheidungsobjekte werden nachher aus der Rule Engine entnommen und bestimmen das weitere Vorgehen.

20.10 Konfigurierbarkeit

Das «Adlerauge» und das «Orakel» werden über Spring Bean Definitions (XML-Dateien mit Wurzelelement `<beans>` in `src/main/resources`) für den Spring-IoC-Container konfiguriert. Diese Konfiguration wird ins WAR der Anwendung eingebettet und ist damit nicht ohne Weiteres änderbar, da entweder das expandierte WAR modifiziert oder das WAR entpackt, modifiziert und neu verpackt werden müsste. Dies ist insbesondere unpraktisch, da diese Prozedur bei jedem Deployment (neuer Build, andere Maschine) wiederholt werden müsste.

Um die Konfiguration von Parametern zu vereinfachen, die häufig geändert werden müssen beziehungsweise sich von Maschine zu Maschine unterscheiden (z.B. Benutzername/Passwort für Datenbank), wurde die Möglichkeit geschaffen, diese auszulagern, sodass sie von Dateien im Property-Format von ausserhalb des Servlet Containers geladen werden können. Dies ermöglicht, die Konfiguration wiederzuverwenden sowie einfach zu ändern.

Der Einstiegspunkt zu den Spring Bean Definitions (`springmvc-servlet.xml`) enthält einen

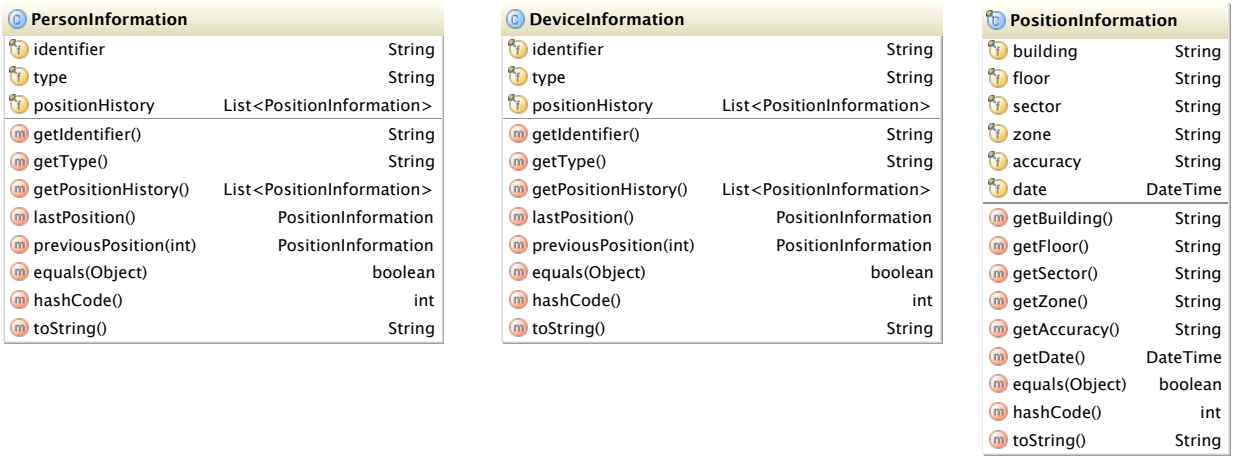

Abbildung 20.26: Die Abbildung gibt einen Überblick über die Objekte, die als Fakten im Entscheidungsprozess über Personenpositionen und Ereignisse mit Personenbezug zur Verfügung stehen.

```
rule "Device␣State␣Event␣Decision"
    when
        $deviceInfo : DeviceInformation()
        $sensorInfo : SensorInformation(type == "CiscoWorks",
                $lastPos : lastPosition)
        $stateEventDecision : StateEventDecision()
    then
        $stateEventDecision.setFire(true);
        $stateEventDecision.setName("IN_MAINTENANCE");
        $stateEventDecision.setObjectIdentifier($deviceInfo.
                getIdentifier());
        $stateEventDecision.setPriority("URGENT");
        $stateEventDecision.setSource("LOCALISATION_SERVICE");
        $stateEventDecision.setDate($lastPos.getDate());
end
```

Listing 20.9: Das Listing zeigt, wie zuerst eine Entscheidung getroffen wird (`when`), bevor diese Entscheidung dann in das betreffende Entscheidungsobjekt verpackt wird (`then`).

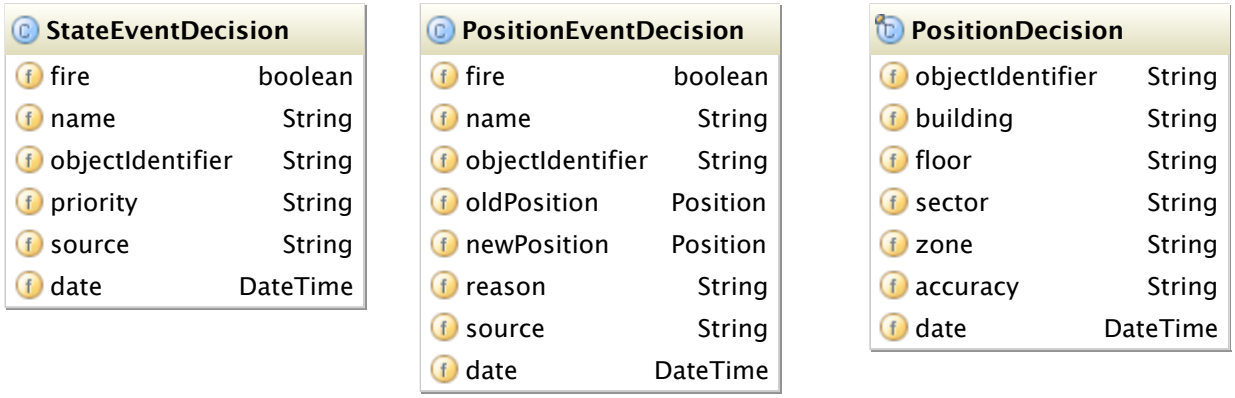

Abbildung 20.27: Die Abbildung gibt einen Überblick über die Objekte, die die in der Rule Engine getroffenen Entscheidungen repräsentieren.

```
<bean
        class="org.springframework.beans...PropertyPlaceholderConfigurer">
    <property name="locations">
        <list>
            <value>classpath:eagleeye.properties</value>
            <value>
              #{systemEnvironment.EAGLEEYE_HOME}/conf/eagleeye.properties}
            </value>
        </list>
    </property>
    <property name="ignoreResourceNotFound" value="true"/>
</bean>
```

Listing 20.10: Der `PropertyPlaceholderConfigurer` lädt die Properties-Dateien von den angegeben Quellen. Später definierte Dateien überschreiben vorher definierte. Kann eine Datei nicht gefunden werden, wird sie ignoriert.

`PropertyPlaceholderConfigurer`, der beliebige Property-Platzhalter im JSP-EL-Stil `${...}` durch die entsprechenden Properties ersetzt, die in einer der Dateien angegeben sein müssen, die im Element `<property name="locations">...</property>` definiert sind. Im Beispiel wären dies die Dateien `eagleeye.properties` auf dem Classpath und `/conf/eagleeye.properties` relativ zur Pfadangabe, die aus der Umgebungsvariable `EAGLEEYE_HOME` extrahiert wird.

Jeder Anwendung liegt eine Datei mit Standardwerten für alle Properties bei. Die Standardwerte werden verwendet:

- Falls der Administrator der Anwendung keine andere Konfiguration angegeben hat.
- Falls die vom Administrator angegebene Konfiguration nicht gefunden werden konnte.
- Falls die vom Administrator angegebene Konfiguration unvollständig ist respektive nur einen Teil der Properties überschreibt. Dann werden die Standardwerte und die überschriebenen Werte gemischt verwendet (Vorrang haben vom Administrator spezifizierte Werte).

```
# database configuration
delphi.db.bureaucrat.driver = org.postgresql.Driver
delphi.db.bureaucrat.url = jdbc:postgresql://localhost:5432/bureaucrat
delphi.db.bureaucrat.username = postgres
delphi.db.bureaucrat.password = postgres

delphi.db.dataheap.host = localhost
delphi.db.dataheap.port = 27017
delphi.db.dataheap.database = dataheap
```

Listing 20.11: Ausgelagerte Konfigurationseinstellungen für das «Orakel» im Properties-Format

```
<bean id="bureaucrat" class="org.apache.commons.dbcp.BasicDataSource"
      destroy-method="close">
    <property name="driverClassName"
              value="${delphi.db.bureaucrat.driver}"/>
    <property name="url" value="${delphi.db.bureaucrat.url}"/>
    <property name="username" value="${delphi.db.bureaucrat.username}"/>
    <property name="password" value="${delphi.db.bureaucrat.password}"/>
</bean>
```

Listing 20.12: Im WAR enthaltene Bean Definitions (Auszug), die Platzhalter für die Properties enthalten.

Listing 20.11 (Seite 199) zeigt einen Auszug aus den Standard-Properties des «Orakels» und Listing 20.12 (Seite 199) die im WAR enthaltene XML-Konfigurationsdatei mit den Properties. Nachdem der `PropertyPlaceholderConfigurer` seine Arbeit erledigt hat, sind die Werte aus der Properties-Datei ins XML eingesetzt, wie in Listing 20.13 (Seite 199) zu sehen.

Die Umgebungsvariablen `EAGLEEYE_HOME` (für das «Adlerauge») und `DELPHI_HOME` (für das «Orakel») müssen vor dem Starten der Anwendung definiert werden. Auf einem unixoiden System mit Bash-Shell kann das zum Beispiel mittels `$ export EAGLEEYE_HOME=/path/to/home` erledigt werden. Erlaubt ist die Angabe aller von Spring verstandenen Ressourcen-Angaben.

```
<bean id="bureaucrat" class="org.apache.commons.dbcp.BasicDataSource"
      destroy-method="close">
    <property name="driverClassName" value="org.postgresql.Driver"/>
    <property name="url"
              value="jdbc:postgresql://localhost:5432/bureaucrat"/>
    <property name="username" value="postgres"/>
    <property name="password" value="postgres"/>
</bean>
```

Listing 20.13: Nach der Bearbeitung durch den `PropertyPlaceholderConfigurer` sind die Properties aus der externen Konfigurationsdatei in die Bean Definitions eingesetzt.

20.11 Implementierung

20.11.1 Organisation

Die Entwicklung der Anwendungen ist auf mehrere Projekte aufgeteilt:

delphi «Orakel» zur Abfrage von Positionen

eagleeye-connectors Konnektoren für das «Adlerauge»

eagleeye «Adlerauge», das sich um die Lokalisierung von Sensoren, Geräten und Personen kümmert

wui Web-Benutzerschnittstelle

Alle Java-Projekte verwenden Apache Maven als Buildwerkzeug. Die Projekte sind entsprechend der Maven-Konventionen organisiert. Bibliotheken werden in seinem separaten Maven Repository verwaltet und sind nicht Teil der Projekte. Die Web-Benutzerschnittstelle kann einfach in den öffentlichen Bereich eines Webservers kopiert werden.

Zusätzlich existieren noch einige Beispiel-Projekte im Projekt `examples` und Hilfswerkzeuge im Projekt `tools`.

20.11.2 Deliverables

Adlerauge

Ein WAR namens `eagleeye.war`, das sämtliche Bibliotheken enthält und direkt im Servlet Container ausgebracht werden kann.

Konnektoren

Jeder Konnektor produziert ein separates JAR mit allen benötigten Bibliotheken, die auf den Classpath des «Adlerauges» ausgebracht werden müssen.

Orakel

Ein WAR namens `delphi.war`, das sämtliche Bibliotheken enthält und direkt im Servlet Container ausgebracht werden kann.

Webbenutzerschnittstelle

Die Dateien im Repository sind einsatzbereit, wie sie sind.

21 Ausblick

21.1 Aktueller Stand

Der Prototyp des Lokalisierungssystems wurde weiter ausgebaut und ist nun soweit, dass er sich für einen Pilotversuch in einem einzelnen Gebäude eignet. Dabei können die wesentlichen Funktionen getestet werden:

- Anbindung mehrerer Lokalisierungswerkzeuge
- Lokalisierung von Sensoren, Geräten und Personen
- Ereignis-Auslösung (Position, Status) abhängig von Position für Geräte und Personen
- Positionshistorie für Geräte und Personen

Von den vorgesehenen Konnektorarten zur Anbindung von Lokalisierungswerkzeugen sind aktuell deren zwei vorhanden: Die Pull- und Push-Konnektoren, die zur Lokalisierung verwendet werden. Das heisst, es können sowohl Lokalisierungswerkzeuge vom Lokalisierungssystem her abgefragt werden als auch Lokalisierungswerkzeuge Änderungen autonom einliefern. Nicht realisiert wurde bislang die sogenannte «Alarm»-Variante, also die Weiterleitung von Ereignissen wie Knopfdrucken auf Lokalisierungssensoren wie einem Ekahau-Tag, da diese einen separaten Verarbeitungspfad benötigt. Die Interpretation der gemeldeten Positionen und damit die eigentliche Lokalisierung von Geräten und Personen erfolgt im Moment rein zeitgesteuert. Dies bedeutet, dass selbst wenn ein Lokalisierungswerkzeug autonom Positionen einliefert, diese erst nach Ablauf eines Timers weiterverarbeitet werden. Mögliche Verbesserungsmassnahmen werden zusammen mit den Alarmen im nächsten Abschnitt diskutiert.

Um den Prototyp in Betrieb zu nehmen, werden in einem ersten Schritt die nötigen Referenzdaten für die Lokalisierungswerkzeuge hinterlegt werden müssen. Im Rahmen der bisherigen Arbeiten wurde aus Zeigründen nur so viel wie nötig gemacht, um die Funktionalität ordentlich testen zu können. Ebenso muss mit der Entwicklung erster Regelsätze begonnen werden, da die Vorhandenen nur dazu dienen, die ermittelten Positionen weiterzuleiten. Etwaige Anpassungen an der Drools-Integration können dabei auch noch nötig sein.

Wie bereits in den vorhergehenden Kapiteln erwähnt, ist die Webbenutzerschnittstelle nicht mehr als ein hastig zusammengesteckter Demonstrator, bei dem nicht einmal alle implementierten Use Cases nach aussen geführt sind. Dies liesse sich ohne grossen Aufwand nachholen. Nachher empfiehlt es sich aber, die Webbenutzerschnittstelle auf eine saubere Basis zu stellen – mehr dazu im nächsten Abschnitt.

Nicht berücksichtigt wurden bei den nennenswerten Aspekten bislang Sicherheit und vollständige Nachvollziehbarkeit. Diese dürften für einen Prototyp aber noch nicht von allzu grosser Relevanz sein. Kleinere Unschönheiten, Design-Probleme oder Aspekte, denen zu wenig Aufmerksamkeit gewidmet wurde, sind direkt in der Dokumentation an der betreffenden Stelle erwähnt.

21.2 Weiterentwicklung

Die nachfolgenden Abschnitte sollen als Ideen verstanden werden, in welche Richtung die weitere Entwicklung gehen könnte und welchen Punkten unserer Meinung nach dabei besondere Aufmerksamkeit geschenkt werden müsste.

21.2.1 Event-basierte Lokalisierung

Wie bereits erwähnt, ist die Lokalisierung rein zeitbasiert. Dies bedeutet, dass jeweils ein festgelegtes Intervall abgewartet werden muss, bis ein Sensor, ein Gerät beziehungsweise eine Person lokalisiert wird. Dieses Vorgehen zuerst zu realisieren, war sinnvoll, da damit alle Lokalisierungswerkzeuge abgedeckt werden können. Gleichzeitig kann es als Grundlage für einen Ausbau auf Event-basierte Lokalisierung dienen.

Event-basierte Lokalisierung bedeutet, dass nicht mehr ein Intervall abgewartet wird, bis ein Objekt lokalisiert wird. Statt dessen wird die Position jedes Mal bestimmt, nachdem eine neue Positionsmeldung für einen Sensor eines Geräts respektive einer Person eingegangen ist. Abbildung 21.1 (Seite 203) zeigt schematisch eine Umsetzungsmöglichkeit. Jedes Mal, wenn eine Nachricht eingeht, wird in einer Message Queue die Nachricht eingereiht, dass für den Sensor beziehungsweise für das Gerät eine neue Position eingegangen ist. Sobald ein Prozess der Geräte- beziehungsweise Personenlokalisierung frei wird, nimmt er die Nachricht aus der Message Queue, bestimmt die Positionen des betroffenen Geräts beziehungsweise der betroffenen Person, gibt sie weiter und teilt dem Timeout Watchdog mit, dass das betroffene Gerät beziehungsweise Person lokalisiert wurde. Der Timeout Watchdog stellt das Timeout daraufhin für das betreffende Gerät respektive Person zurück. Der Timeout Watchdog ist der Scheduler, der bereits jetzt verwendet wird. Seine Aufgabe ist, sicherzustellen, dass für alle Geräte beziehungsweise Personen in regelmässigen Abständen eine Position bestimmt wird. Würde er fehlen, könnte nicht bemerkt werden, dass beispielsweise ein Smartphone keine Positionen mehr sendet und es würde eine falsche Position für das Smartphone gespeichert. Das bedeutet, der Scheduler arbeitet so weiter, wie er es jetzt bei der intervallbasierten Lokalisierung tut. In Zukunft kann mit Hilfe der Events einfach eine Ausführung der intervallbasierte Lokalisierung übersprungen werden.

21.2.2 Alarm-Transport

Nicht realisiert wurde bislang der «Alarm»-Transport, also die Weiterleitung von irgendwelchen Ereignissen, die auf einem Sensor ausgelöst werden. Ein Beispiel wäre ein Knopfdruck auf einem Ekahau-Tag. Die Weiterleitung solcher Ereignisse ist nicht ganz unproblematisch, da es in diesem Bereich unter Umständen um Menschenleben gehen kann und die Anforderungen an die Zuverlässigkeit von Seiten des Gesetzgebers verständlicherweise sehr hoch sind. So muss beispielsweise sichergestellt werden, dass die Nachrichten transportiert werden. Hier beginnen bereits die Probleme, da beispielsweise die Abfrageschnittstelle der Ekahau Positioning Engine – die als eines der Lokalisierungswerkzeuge für solche Aufgaben geeignet wäre – nicht die nötigen Überwachungsmöglichkeiten bietet. Auch beim Lokalisierungssystem können die Nachrichten nicht über den üblichen Pfad transportiert werden, da erstens das Nachrichtenformat für die Alarme anders aussieht als das für die Positionsmeldungen und andererseits beim herkömmlichen Lokalisierungspfad (siehe Bereiche B und C in Abbildung 21.2 auf Seite 205) der Transport nicht garantiert ist. So kann es passieren, dass eine Regel die Nachricht unterdrückt,

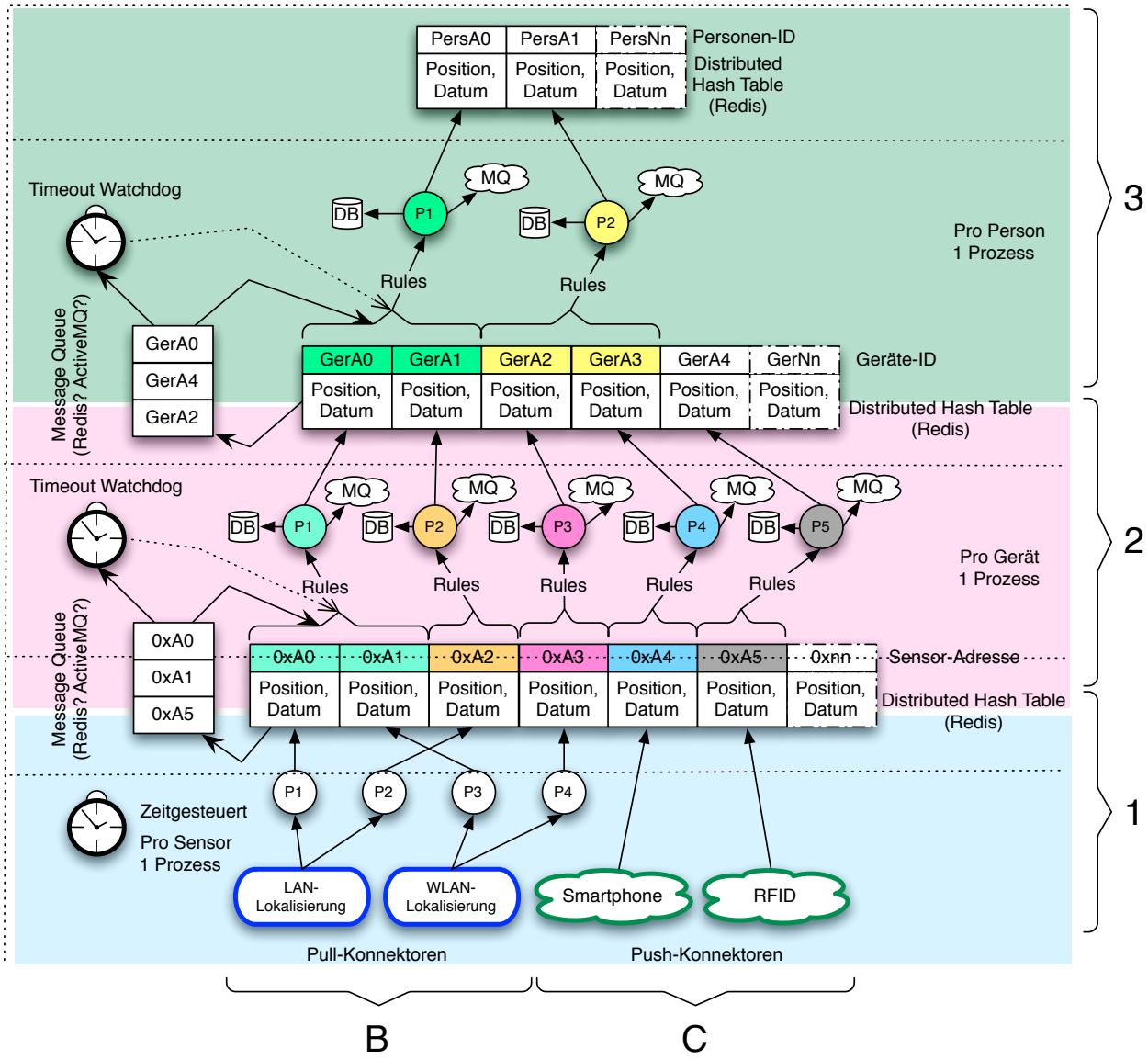

Abbildung 21.1: Die Abbildung zeigt eine mögliche Erweiterung auf Event-basierte Lokalisierung mit Timeout Watchdog. Dabei wird die Position eines Geräts respektive einer Person jedes Mal sofort neu bestimmt, wenn ein Ereignis von einem Lokalisierungswerkzeug eingeliefert wird. Hört das System innerhalb einer Zeitspanne nicht mehr vom Lokalisierungswerkzeug (z.B. Smartphone in See gefallen), wird der Timeout Watchdog ausgelöst und eine entsprechende Aktion getriggert (z.B. «Position unbekannt»). Aus Gründen der Übersichtlichkeit sind alle Verbindungslinien nur für die Objekte ganz links eingezeichnet.

weil beispielsweise die Genauigkeit der Positionsangabe ungenügend ist. Deshalb muss ein separater Pfad geschaffen werden, wie er in Abbildung 21.2 (Seite 205) in Bereich A zu sehen ist. Dieser leitet nach einer Übersetzung der vom Lokalisierungswerkzeug gelieferten Koordinaten die Nachrichten direkt an die Datenbank und die Message Queue weiter. Allenfalls ist es möglich, noch Regeln zur Transformation der Nachricht zu verwenden, wobei aber sichergestellt sein muss, dass die Nachricht nicht zerstört wird.

Auf der anderen Seite muss man sich fragen, ob es nicht klüger wäre, aus Komplexitätsgründen den «Alarm»-Transport nicht im «Adlerauge» anzusiedeln, sondern statt dessen eine separate Software-Komponente zu entwickeln. Diese kann dieselben Konnektoren wie das «Adlerauge» verwenden und auf denselben Datenbestand zugreifen, sodass sich der Zusatzaufwand in Grenzen halten würde.

21.2.3 Sicherheit

Sicherheit wurde bislang noch gar nicht betrachtet und ist auch hinsichtlich Datenschutz ein wichtiger Punkt. Entsprechend wird eine vertiefte Analyse unumgänglich sein. Mit Spring Security ist zwar geeignete Infrastruktur zum Nachrüsten vorhanden. Ob diese aber ausreicht, kann nicht beurteilt werden.

Ein wichtiger Sub-Aspekt der Sicherheit ist Validierung, vor allem der Daten, die über die Schnittstellen hineinkommen. Diese werden aktuell rudimentär in den DTO-Konvertern überprüft. Dies ist aber nicht annähernd ausreichend und sollte deshalb ausgebaut werden. Besonders hervorzuheben ist dabei die Speicherung der Typen (Geräte, Personen, Status) in der Datenbank. Aktuell kann von aussen einfach etwas vorgegeben werden. Da auf diesen Typen beispielsweise aggregiert wird, können Tippfehler bei der Typvorgabe zu falschen Ergebnissen führen. Hier würde es sich anbieten, eine Typenliste in der Datenbank verwalten zu können, gegen die dann die Typen validiert werden.

21.2.4 Vollständige Nachvollziehbarkeit

Da die vollständige Nachvollziehbarkeit im Medizinalbereich durch Vorschriften geregelt ist, wird wie bei der Sicherheit eine vertiefte Analyse der Anforderungen unumgänglich sein.

Hinsichtlich Realisierung bietet es sich an, die Daten wie die Positionshistorien in einer NoSQL-Datenbank abzulegen. Dies kann das bereits verwendete MongoDB sein. Aufgrund der grossen zu erwartenden Datenmenge – für jede gespeicherte Positionsmeldung muss ein Vielfaches an Ausgangsinformationen abgelegt werden, die bei der Entscheidungsfindung vernichtet werden – dürfte schnell die Nutzung des Hadoop-Stacks in den Fokus rücken. Die Analyse erwähnt eine Reihe von möglicherweise geeigneten Werkzeugen.

21.2.5 Webbenutzerschnittstelle

Wie bereits dargelegt, ist die Webbenutzerschnittstelle vom technischen Standpunkt her stark verbesserungswürdig. Wichtigster Schritt dürfte die Wahl beziehungsweise Erstellung einer geeigneten Anwendungsplattform sein – dies kann ein vorgefertigtes Framework oder eine Eigenlösung sein –, mit deren Hilfe die Webbenutzerschnittstelle auf eine saubere und zukunftsträchtige Basis gelegt wird. Dann kann daran gegangen werden, sich um Punkte wie Validierung oder Vereinfachung der Benutzereingabe (beispielsweise mittels Autocomplete) zu kümmern. Besonderes

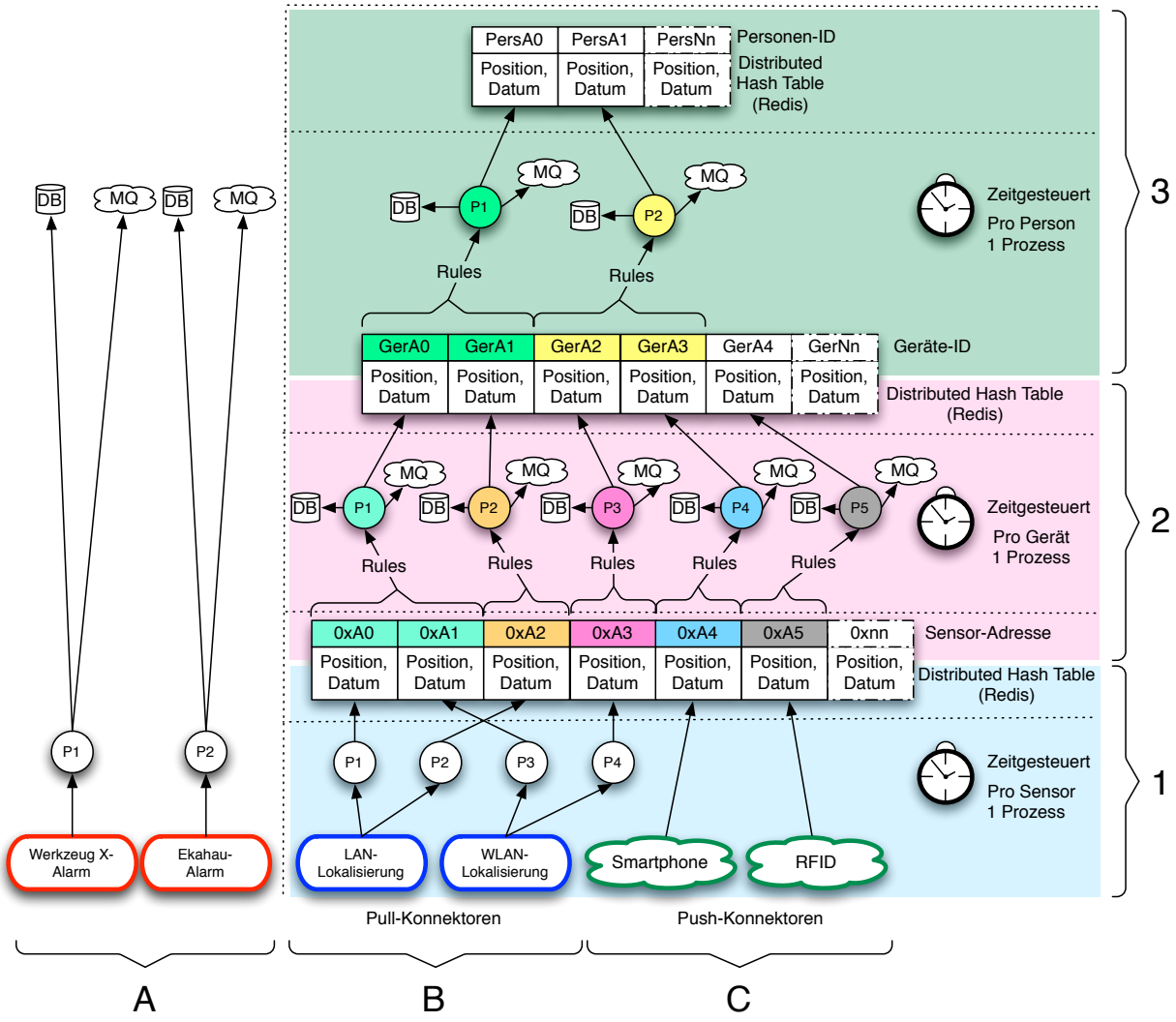

Abbildung 21.2: Mögliche Erweiterung des «Adlerauges» um einen separaten Pfad zum Transport von «Alarm»-Nachrichten. Dieser ist nötig, um sicherzustellen, dass die Nachrichten garantiert transportiert werden.

Augenmerk, auch in Bezug auf das serverseitige REST-Interface, muss auf die Fehlerbehandlung gelegt werden, da beispielsweise auf eine nicht gefundene Ressource im Moment nicht, wie es von HTTP vorgesehen ist, mit dem Status-Code 404 geantwortet wird. Ebenso wurde Optimistic Concurrency bislang nicht implementiert.

21.2.6 Testing und Qualitätssicherung

Die Funktionalität der Anwendungen ist im Moment hauptsächlich durch Microtests abgesichert. Ein wichtiger Punkt, der aus Zeitgründen nicht realisiert werden konnte, sind automatische Integrationstests, da diese eine relativ aufwendige Konfiguration und eine ausgebaute Test-Umgebung benötigen. Eine Herausforderung sind in dieser Beziehung insbesondere geeignete Simulatoren für die Lokalisierungswerkzeuge.

Teil V

Umsetzung Konnektoren

22 Allgemeiner Contract

22.1 Einführung

Dieses Kapitel beschreibt den allgemeinen Contract (Vertrag) zwischen Konnektoren und Lokalisierungssystem. Zusätzlich gibt es einen kurzen Überblick über die wichtigsten Architekturkonzepte. Für Implementierungsbeispiele sei bereits hier an die bereits realisierten Konnektoren verwiesen, da die Dokumentation sonst nur unnötig aufgeblasen würde.

22.2 Contract

Abbildung 22.1 (Seite 210) zeigt das Design-Diagramm des allgemeinen Contracts. Er umfasst einerseits die Interfaces, die von den verschiedenen Konnektorarten zu implementieren sind, und andererseits die Java Beans, die zum Austausch von Daten genutzt werden. Die nächsten Abschnitte geben einen Überblick über den Zweck der Klassen. Details können der JavaDoc-Dokumentation entnommen werden.

22.2.1 ConnectorInformation

Jeder Konnektor muss ein Objekt des Typs `ConnectorInformation` als Spring Bean exportieren, mit dessen Hilfe der Konnektor dem Lokalisierungssystem beschrieben wird. Das Attribut `forSensorType` definiert nämlich, für welchen Sensor-Typ (Ekahau, CiscoWorks etc.) der Konnektor geeignet ist. Zu beachten ist, dass zu jedem Zeitpunkt jeweils nur ein einziger Konnektor für einen bestimmten Sensor-Typ geladen sein kann.

22.2.2 PullConnector

Pull-Konnektoren, also solche, die vom Lokalisierungssystem aus aktiviert werden, müssen das Interface `PullConnector` implementieren und als Spring Bean exportieren, damit sie vom Lokalisierungssystem geladen werden können. Die Methode `getPosition(String)` wird vom Lokalisierungssystem aufgerufen, um die Lokalisierung des Sensors mit der angegebenen Adresse zu veranlassen.

22.2.3 PushConnector

Push-Konnektoren, also solche, die von «aussen» aus aktiviert werden (HTTP, JMS etc.), müssen das Interface `PushConnector` implementieren, damit das Lokalisierungssystem sie finden kann. Sie dürfen aber nicht als Spring Beans verdrahtet werden.

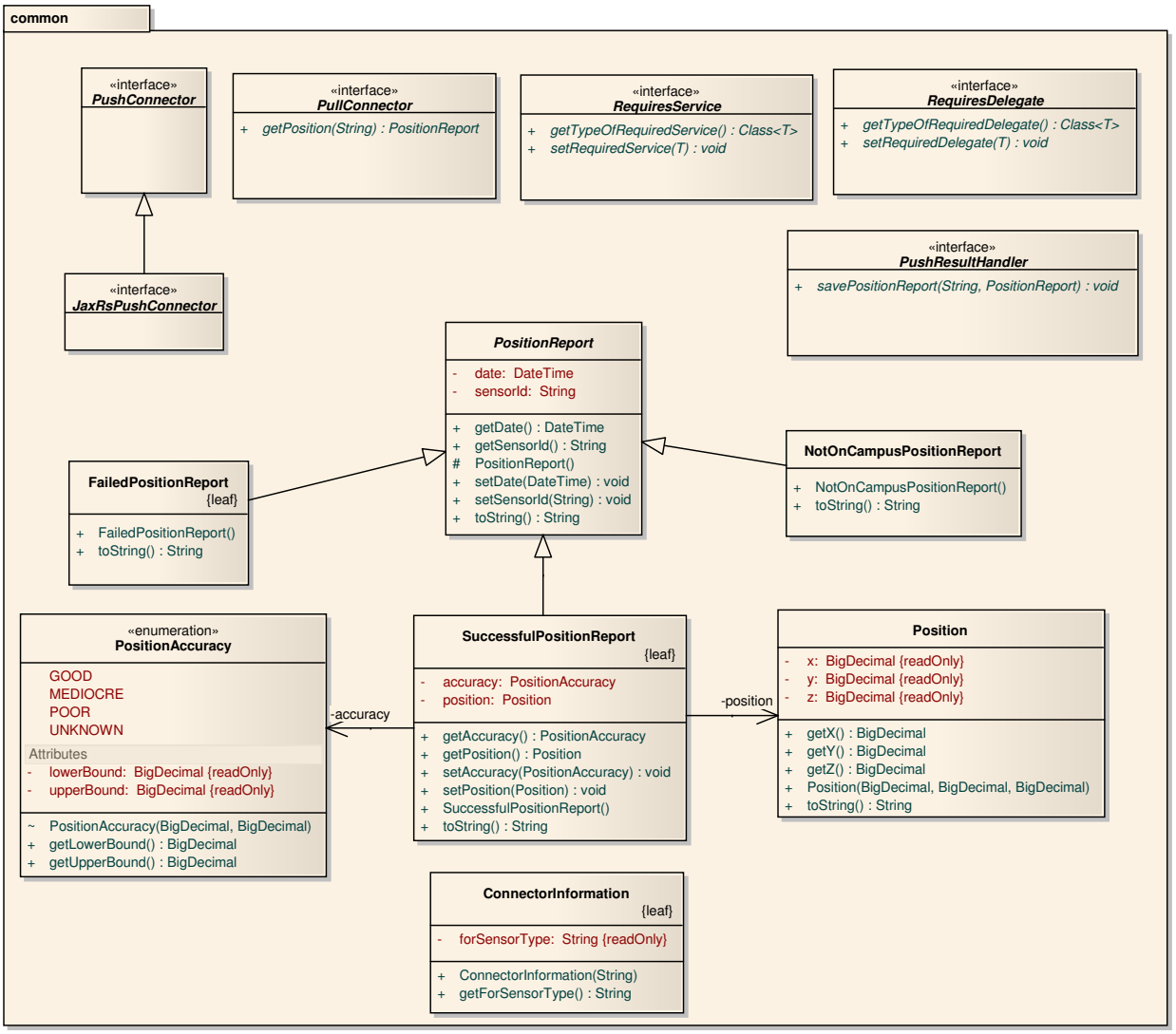

Abbildung 22.1: Design-Diagramm des allgemeinen Konnector-Contract

Eine besondere Rolle spielt die Spezialisierung `JaxRsPushConnector`. Diese markiert einen `PushConnector` als JAX-RS Endpunkt. Dies ist wichtig, damit das Lokalisierungssystem das zugehörige Servlet exportieren kann. Abschnitt 20.7.2 (Seite 192) erklärt die Funktionsweise im Detail.

22.2.4 PushResultHandler

Der `PushResultHandler` wird vom Lokalisierungssystem angeboten und ermöglicht den Push-Konnektoren, die eingelieferte Positionsmeldung an das Lokalisierungssystem zu übergeben. Der `PushResultHandler` kann mit Hilfe des Interface `RequiresService<T>` vom Lokalisierungssystem angefordert werden.

22.2.5 RequiresDelegate<T>

Anteile von Push-Konnektoren, die im Lokalisierungssystem selber leben wie Implementierungen von `JaxRsPushConnector`, können über `RequiresDelegate<T>` Delegates aus dem eigenen Konnektor-Kontext beziehen. Wie das genau funktioniert, illustriert Abschnitt 20.7.2 (Seite 192). Dass nur ein einzelnes Delegate pro Klasse vom Konnektor bezogen werden kann, ist Absicht.

22.2.6 RequiresService<T>

Benötigen Pull-Konnektoren Services vom Lokalisierungssystem, können sie diese über das Interface `RequiresService<T>` anfordern. Die genaue Funktionsweise von `RequiresService<T>` erklärt Abschnitt 20.7.2 (Seite 192).

Der typische Anwendungsfall für `RequiresService<T>` ist in einem Push-Konnektor das Beziehen des `PushResultHandler`. Andere Services sollten nicht benötigt werden, weil dies die Kopplung erhöht. Normalerweise ist die Verwendung von `RequiresService<T>` deshalb ein starkes Indiz dafür, dass der Konnektor den Service selber anbieten sollte. Entsprechend ist es Absicht, dass nur ein einzelner Service pro Klasse vom Lokalisierungssystem bezogen werden kann.

22.2.7 PositionReport

Der Lokalisierungsservice erwartet die gemeldete Position in Form eines `PositionReport`.

FailedPositionReport Wird übermittelt, wenn die Lokalisierung fehlgeschlagen ist, z.B. bei Auftreten einer Exception.

NotOnCampusPositionReport Wird übermittelt, wenn sich der Sensor nicht auf dem Campus befindet. Die eigentliche Position ist aus Datenschutzgründen nicht Teil dieser Nachricht. Wenn der Sensor nicht bereits eine solche Nachricht schickt, muss im Konnektor sichergestellt werden, dass die genauen Positionsdaten unterdrückt werden und das Lokalisierungssystem sie nicht erfährt.

SuccessfulPositionReport Wird übermittelt, wenn die Lokalisierung erfolgreich war. Umfasst eine relative Angabe der Genauigkeit der Position und die Position in Landeskoordinaten. Für die Abbildung der Genauigkeit sind jeweils eine untere und obere Grenze der maximal

möglichen Abweichung in Metern hinterlegt. Die Position muss in Landeskoordinaten angegeben werden, weil dies eine gültige Positionsangabe ermöglicht, ohne dass die gesamten Services zur Abbildung auf Sektoren in die Konnektoren hineingereicht werden müssen, was die Kopplung erhöhen und negative Auswirkungen hinsichtlich Skalierbarkeit hätte. Die Umrechnung in Landeskoordinaten ist Aufgabe der Konnektoren.

22.3 Architektur

22.3.1 Technologiekomponeten

Die einzelnen Konnektoren steht es grundsätzlich frei, welche Technologiekomponenten sie für interne Funktionalität verwenden. Es wird aber dringend davon abgeraten, an Integrationspunkten für bestimmte Standards wie Java API for RESTful Web Services (JAX-RS) oder JAXB andere Technologien als der Lokalisierungsservice zu verwenden – beispielsweise Jersey statt RESTeasy für JAX-RS –, da dies zu Problemen beim Laden und Aktivieren der Konnektoren führen kann.

Übersicht über die vorgegebenen Technologie-Komponenten:

Jackson Dient zur Umwandlung von Java Beans in JSON und umgekehrt.

Joda Time Bibliothek mit einfacheren und leistungsfähigeren Datumsoperationen als sie Java von Haus aus bereitstellt.

RESTeasy JAX-RS Provider von JBoss zur programmatischen Erstellung der REST Endpoints. Wurde gewählt, da es sehr gute Testing- und SpringMVC-Unterstützung bietet.

SLF4J Logging-Façade für diverse Logging Frameworks. Verwendet wird die Log4j-Weiterentwicklung Logback.

Spring Data Abstrahiert und vereinfacht den Umgang mit verschiedenartigen Datenquellen. Wird vor allem für den Zugriff auf MongoDB verwendet.

Spring Framework Meta-Framework, auf dem die Anwendung zu weiten Teilen basiert. Verwendet wird es unter anderem für Dependency Injection, aspektorientierte Programmierung, vereinfachten JDBC-Zugriff, vereinfachte JMS-Nutzung, Transaktionen, Sicherheit und Servlet-Unterstützung. Verwendet werden die Komponenten AOP, Beans, Context, Core, Expression, JDBC, JMS, TX, Web und WebMVC.

Übersicht über die für Testzwecke bereitgestellten Technologie-Komponenten:

JUnit Unit-Testing-Framework zur Erstellung und Ausführung von Unit Tests.

Mockito Mocking-Framework, das die Erzeugung von Mocks, Fakes und Stubs vereinfacht.

Spring Test Teil des Spring Frameworks, die das Testing von Spring-Anwendungen vereinfacht.

22.3.2 Architekturkonzepte

Die Konzepte des Lokalisierungssystem sind auch für die Konnektoren gültig. In den nächsten Abschnitten werden noch einige spezielle Punkte beschrieben.

```
<bean class="org.springframework.beans...PropertyPlaceholderConfigurer">
    <property name="locations">
        <list>
            <value>classpath:ciscoworks21781.properties</value>
            <value>#{systemEnvironment.EAGLEEYE_HOME}/conf/
                connector/ciscoworks21781.properties</value>
        </list>
    </property>
    <property name="ignoreResourceNotFound" value="true"/>
</bean>
```

Listing 22.1: Beispiel für ein `PropertyPlaceholderConfigurer` eines Konnektors

Spring Framework

Jeder Konnektor muss über einen eigenen `ApplicationContext` mit XML Bean Definitions verfügen. Ansonsten ist es für das Lokalisierungssystem nicht möglich, den Konnektor zu laden. Konnektoren zu unterstützen, die keinen `ApplicationContext` verwenden, ist nicht vorgesehen.

22.3.3 Referenzdaten

Die Konnektoren müssen sich selber um die Beschaffung und Verwaltung von Referenz- oder Geodaten kümmern. Ein Zugriff auf das Wissen des Lokalisierungssystems ist nicht möglich. Sie erhalten lediglich Zugriff auf den MongoDB-Cluster des Lokalisierungssystems, dürfen darauf aber lediglich eine einzelne, vom Administrator zugewiesene Datenbank verwenden, die über eine Konfigurationsdatei inklusive Benutzername und Passwort definierbar sein muss.

22.4 Konfigurierbarkeit

Konnektoren müssen wie das Lokalisierungssystem über externe Properties-Dateien konfigurierbar sein. Es bietet sich an, wie das Lokalisierungssystem einen `PropertyPlaceholderConfigurer` zu verwenden; die Umgebungvariable `EAGLEEYE_HOME` steht ebenfalls zur Verfügung. Konnektoren können ihre Konfiguration relativ zu dieser im Pfad `/conf/connnector` ablegen. Ein Beispiel zeigt Listing 22.1 (Seite 213).

22.5 Implementierung

Alle Konnektoren bilden zusammen ein Maven-Multimodul-Projekt, wobei jeder Konnektor sein eigenes JAR produziert. Eine Hilfestellung zur Entwicklung eines neuen Konnektors bietet Anhang C (Seite 277).

23 Ekahau Pull-Konnektor

23.1 Einführung

23.1.1 Überblick

Die Anbindung der Positionsbestimmung über Ekahau an den Lokalisierungssenvice erfolgt über einen Pull-Konnektor. Dieser muss gemäss dem Contract im Abschnitt 22.2 (Seite 209) umgesetzt werden.

Kann die Ekahau Positioning Engine das Gerät mit der gesuchten Adresse lokalisieren, wird ein `SuccessfulPositionReport` erzeugt. Dieser besteht aus der umgerechneten Position sowie der Zeitangabe der Anfrage. Zusätzlich wird die Genauigkeit anhand der von Ekahau gelieferten Genauigkeit in Prozent umgewandelt auf die Genauigkeitstypen `GOOD`, `MEDICORE` und `BAD`.

23.2 Umwandlung der Koordinaten

Wie in der Analyse (Abschnitt 8.1, Seite 41) beschreiben, erfolgt die Umrechnung der von Ekahau gelieferten Koordinaten in die von dem Lokalisierungsservice geforderten Landeskoordinaten über eine Transformationsmatix. Die Umwandlung der Höhe erfolgt über die Zuweisung der Höhe an die Ekahau-Map.

Für die Transformationsmatrix, welche die von der Ekahau Positioning Engine gelieferten Positionsangaben in Landeskoordinaten umwandelt, sollen so wenig Informationen wie möglich benötigt werden. Da Ekahau die Positionskoordinaten jeweils bezogen auf eine Map liefert, braucht es für jede Map eine eigene Matrix. Dabei wird es nötig sein, verschiedene Referenzpunkte für jede Map und jeden Stockwerkplan zu hinterlegen.

In diesem Kapitel sind die Vorgehensweisen für die Umwandlung mittels einer Transformationsmatrix beschrieben.

23.2.1 Referenzpunkte

Ganz allgemein gesagt existieren zwei verschiedene Nullpunkte, die in keiner Beziehung zueinander stehen. Ausserdem liegen zwei völlig verschiedene Koordinatensysteme vor: das Bild-Koordinatensystem auf den Ekahau-Maps und das Landeskoordinatensystem auf den georeferenzierten Stockwerkplänen. Um vom Bild-System auf das Landeskoordinaten-System umzurechnen, müssen diese beiden Systeme in Beziehung zueinander gesetzt werden. Um von einem System ins andere umrechnen zu können, sind im zweidimensionalen Raum drei Referenzpunkte nötig. Von diesen müssen jeweils die Position im Bild-System und im Landeskoordinaten-System bekannt sein. Um ein möglichst genaues Ergebnis zu erhalten, müssen diese so gewählt werden, dass beim Verbinden der drei Punkte ein möglichst grosses Dreieck entsteht. Ein Beispiel ist in Abbildung 23.1 (Seite 216) gezeigt.

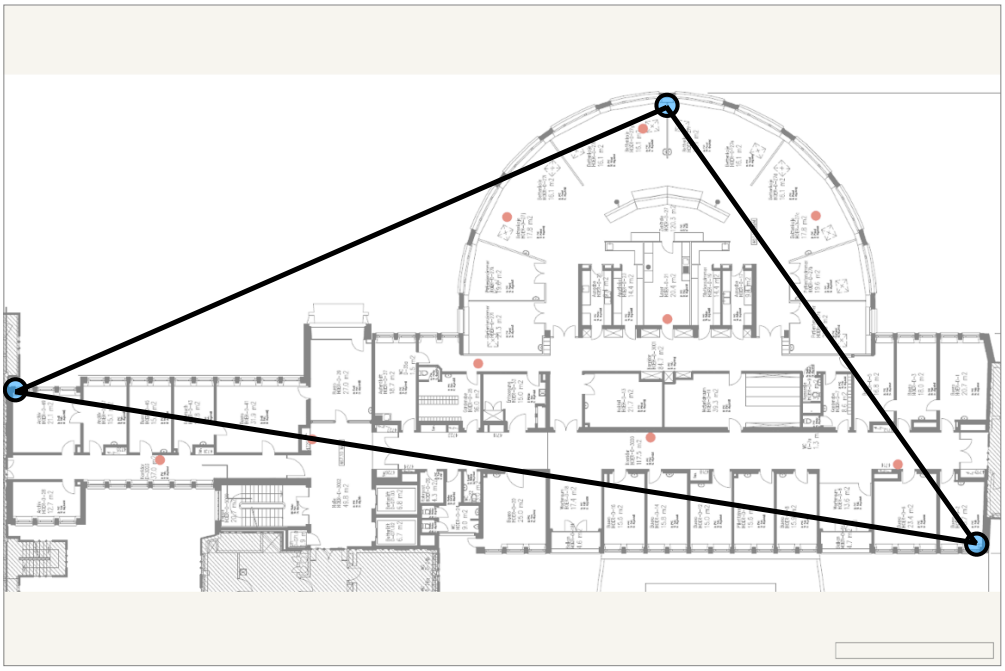

Abbildung 23.1: Beispiel für die Wahl der Bezugspunkte

Einer der drei Referenzpunkte stellt den Basispunkt dar. Dieser ersetzt sozusagen den Nullpunkt. Das heisst, für die Berechnungen werden die anderen zwei Punkte immer in Bezug auf den Basispunkt gesetzt. Befindet sich zum Bespiel der Basispunkt an der absoluten Position $(10, 250)$ und ein Referenzpunkt hat die absolute Position $(375, 44)$, so wird die Position des Referenzpunktes relativ zum Basispunkt ausgedrückt. Der Referenzpunkt hat also in x-Richtung relativ eine Distanz von 365 und in y-Richtung eine Distanz von -208 (Siehe Abbildung 23.2, Seite 217).

Sind die Referenzpunkte gewählt, existieren zwei Systeme mit jeweils drei Punkten, deren Position sowohl in Bild-Koordinaten als auch in Landeskoordinaten bekannt ist (Abbildung 23.3, Seite 218).

23.2.2 Umrechnung über Rotation, Verschiebung und Streckung

Die naheliegende Lösung zur Umrechnung ist eine Transformation des Bilds in das System der Landeskoordinaten in mehreren Schritten. Dabei setzt sich diese Transformation aus einer Rotation , einer Verschiebung und einer Streckung zusammen. Daraus folgt für die Umrechnung die Gleichung in Formel 23.1 (Seite 216).

$$\begin{pmatrix} X \\ Y \end{pmatrix} = s \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} a \\ b \end{pmatrix} \tag{23.1}$$

Ausgehend vom Basispunkt können mit den zwei Referenzpunkten die Verschiebung $\binom{a}{b}$, der Winkel α und die Streckung s bestimmt werden. Soll nun ein neuer Punkt bestimmt werden, so entstehen zwei Gleichungen mit zwei Unbekannten, wobei $\binom{x}{y}$ die Bildposition und $\binom{X}{Y}$ die Position im Landeskoordinatensystem darstellen (Formel 23.2, Seite 217, und Formel 23.3, Seite

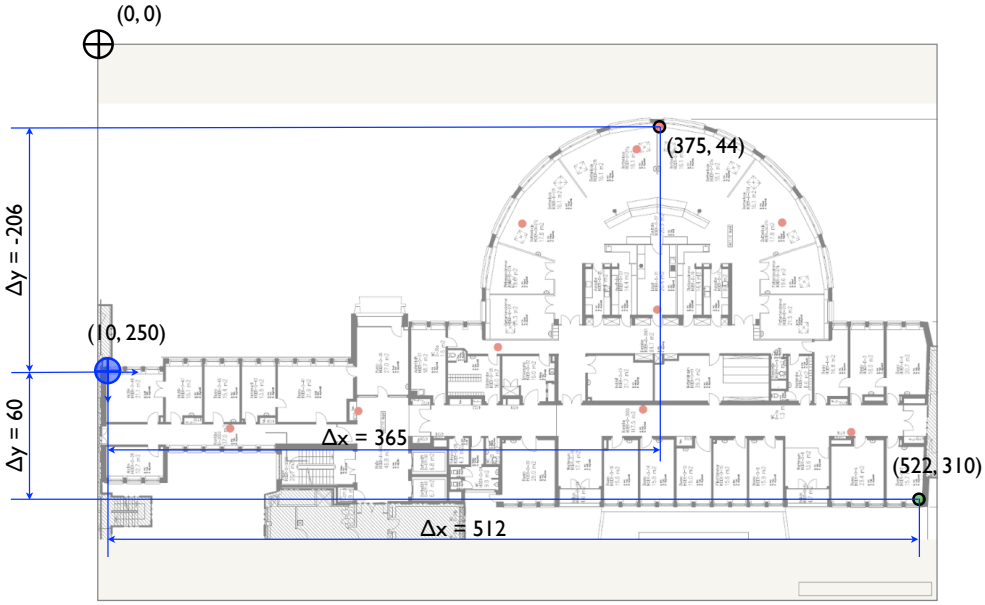

Abbildung 23.2: Beispiel des Basispunkts

217):

$$X = s \cdot \cos(\alpha) \cdot x + s \cdot \sin(\alpha) \cdot y + a \tag{23.2}$$

$$X = s \cdot -\sin(\alpha) \cdot x + s \cdot \cos(\alpha) \cdot y + a \tag{23.3}$$

Die Problematik bei dieser Lösung besteht darin, dass für die Programmierung des Ekahau-Konnektors mit Winkelfunktionen gearbeitet werden muss. Das macht es unnötig kompliziert und hat zusätzlich Ungenauigkeiten zur Folge. Deshalb wurde nach einer besseren Lösung gesucht, welche im folgenden Kapitel beschrieben ist.

23.2.3 Direkte Umrechnung über eine Transformationsmatrix

Um die Rechnerei mit den Winkelfunktionen zu umgehen, kommt nur eine Lösung in Frage, die auf einer einzigen Transformationsmatrix basiert. Es ergibt sich am Ende eine Formel, die wie folgt aussieht, wobei die Punkte P_1, P_2, P_3 die Referenzpunkte im Landeskoordinatensystem sind:

$$A' = \varphi \cdot \overrightarrow{P_{LK1}P_{LK3}} + \mu \cdot \overrightarrow{P_{LK1}P_{LK2}} + P_{LK1} \tag{23.4}$$

Multipliziert man den Koordinatenpunkt P_{W3} des Bilds mit einer Matrix M_e, so entsteht der Einheitsvektor im Bild-Koordinatensystem:

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} = M_e \cdot P_{W3} \tag{23.5}$$

Multipliziert man die gleiche Matrix M_e mit dem Bild-Koordinatenpunkt P_{W2} so erhält man den anderen Einheitsvektor:

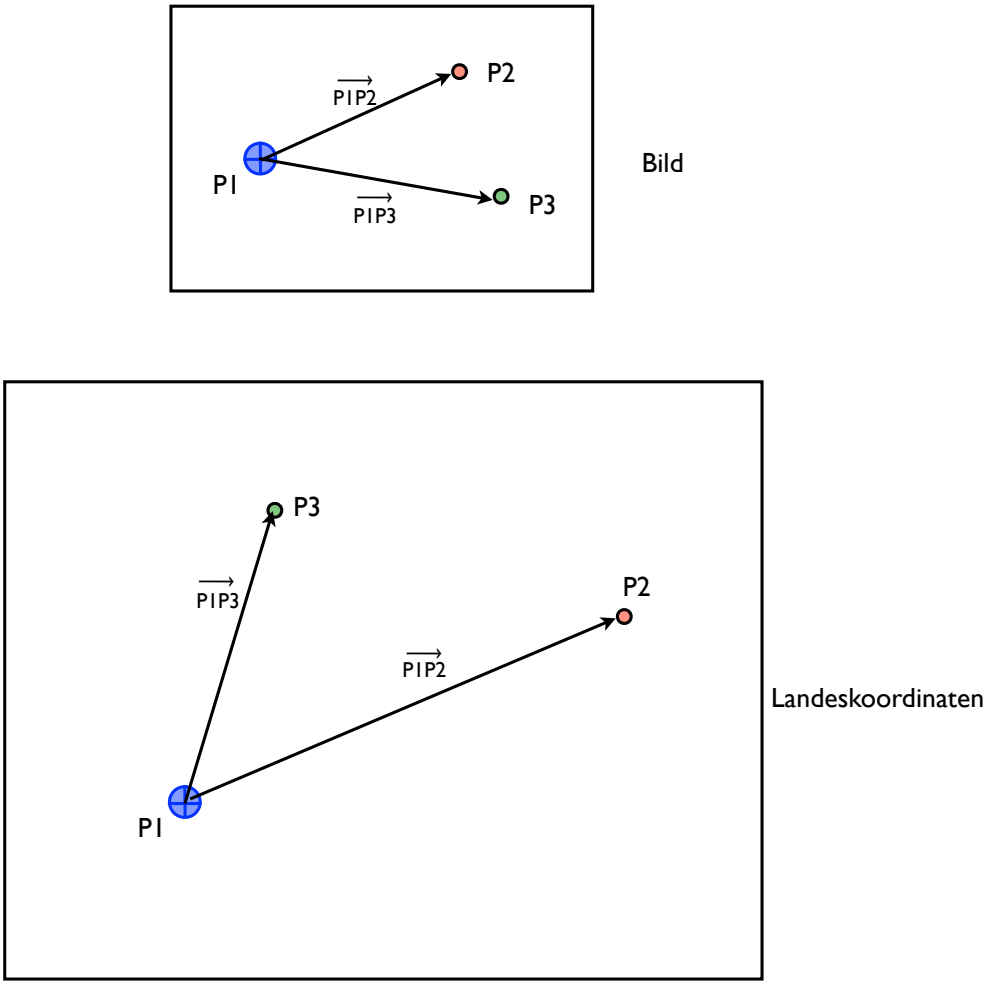

Abbildung 23.3: Beispiel der beiden Systeme der Bild-Koordinaten und der Landeskoordinaten

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix} = M_e \cdot P_{W2} \tag{23.6}$$

Wird ein beliebiger Punkt A aus dem Bild-Koordinatensystem mit der Matrix M_e multipliziert, so erhält man einen Punkt A_V, dessen Koordinaten genau φ und μ sind, die für die Berechnung der Formel 23.4 (Seite 217) verwendet werden:

$$\begin{pmatrix} \varphi \\ \mu \end{pmatrix} = A_V = M_e \cdot A \tag{23.7}$$

Die Gleichung 23.5 (Seite 217) kann dann nach P_W3 umgeformt werden:

$$P_{W3} = M_e^{-1} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{23.8}$$

Sei $M_e^{-1} = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ so ergibt sich für die P_{W3}:

$$P_{W3} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} a \\ c \end{pmatrix} \tag{23.9}$$

Die Gleichung 23.6 (Seite 219) kann ebenfalls umgeformt werden und zwar nach P_{W2}:

$$P_{W2} = M_e^{-1} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{23.10}$$

$$P_{W2} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} b \\ d \end{pmatrix} \tag{23.11}$$

Daraus ist ersichtlich, dass sich die Matrix M_e^{-1} durch die Koordinaten der Punkte P_{W3} und P_{W2} ausdrücken lässt:

$$M_e^{-1} = \begin{pmatrix} P_{W3x} & P_{W2x} \\ P_{W3y} & P_{W2y} \end{pmatrix} \tag{23.12}$$

Die Matrix M_e ist nun die Inverse-Matrix von M_e^{-1}:

$$M_e = M_e^{-1^{-1}} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} = \frac{1}{P_{W3x}P_{W2y} - P_{W2x}P_{W3y}} \begin{pmatrix} P_{W2y} & -P_{W2x} \\ -P_{W3y} & P_{W3x} \end{pmatrix} \tag{23.13}$$

Mit der Transformationsmatrix kann nun für jeden beliebigen Punkt, der von der Ekahau Positioning Engine geliefert wird, die passenden Landeskoordinaten berechnet werden. Was es dabei noch zu beachten gilt, ist, dass der bei einer Positionsabfrage erhaltene Punkt nicht direkt verwendet werden kann, da sich dieser auf den absoluten Nullpunkt im Bild-System bezieht. Die Koordinaten müssen, wie in Abschnitt 23.2 (Seite 217) dargelegt, umgerechnet werden. Daraus ergibt sich zusätzlich noch folgende Formel zur Umrechnung der Koordinaten bezogen auf den absoluten Nullpunkt bezogen auf den Basispunkt, wobei P_1 der Basispunkt und P_2 und P_3 die Referenzpunkte sind:

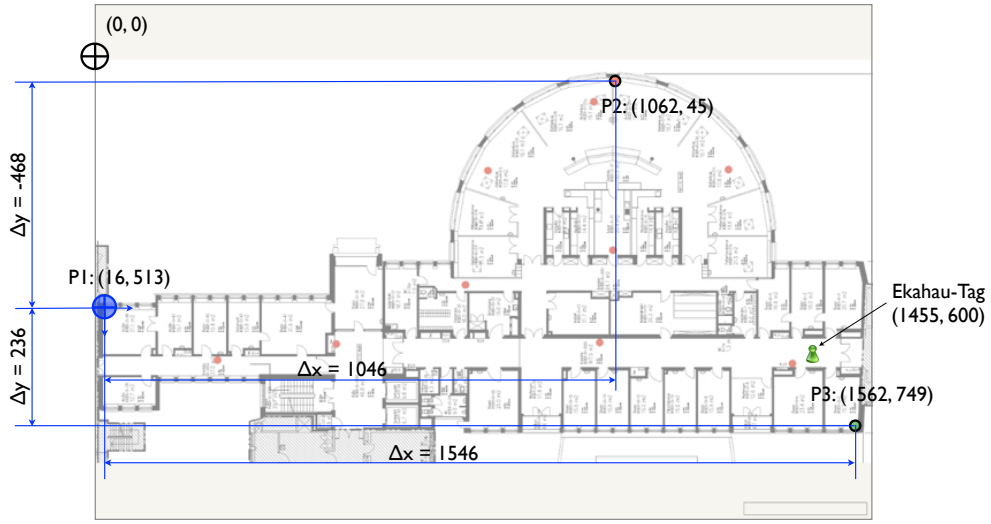

Abbildung 23.4: Bilddatei eines Stockwerks, bei dem drei Referenzpunkte eingezeichnet sind

$$\begin{pmatrix} P_{2x} \\ P_{2y} \end{pmatrix} = \begin{pmatrix} P_{2x0} - P_{1x} \\ P_{2y0} - P_{1y} \end{pmatrix} \tag{23.14}$$

$$\begin{pmatrix} P_{3x} \\ P_{3y} \end{pmatrix} = \begin{pmatrix} P_{3x0} - P_{1x} \\ P_{3y0} - P_{1y} \end{pmatrix} \tag{23.15}$$

Zur anschliessenden Umrechnung eines Punktes im Bild-System in das Landeskoordinaten-System wird der Bildpunkt mit der Matrix multipliziert. Daraus ergeben sich die für die Formel 23.4 (Seite 217) notwendigen Werte für φ und μ. Die beiden Werte werden dann in der Formel 23.4 (Seite 217) zusammen mit den beiden Vektoren zwischen Basispunkt und jeweiligem Referenzpunkt aus dem Landeskoordinatensystem eingesetzt. Zusätzlich wird noch der Basispunkt P_{LK1} addiert, um so die absolute Position zu bekommen.

23.2.4 Beispiel

Um für die Programmierung des Konnektors gute Testdaten zu haben, werden hier an einem Beispiel die Berechnungen durchgeführt. Somit können nachher die einzelnen Teilresultate für das Schreiben von sinnvollen Tests verwendet werden.

Bild

Für ein Stockwerkplan wurden in der Bilddatei drei Referenzpunkte gewählt. In Abbildung 23.4 (Seite 220) sind diese zusammen mit den jeweiligen absoluten Koordinaten und den Distanzen zum Basispunkt (P_1) eingezeichnet.

Gemäss Formel 23.14 (Seite 220) und 23.15 (Seite 220) ergeben sich folgende Koordinaten für die Punkte P_2 und P_3:

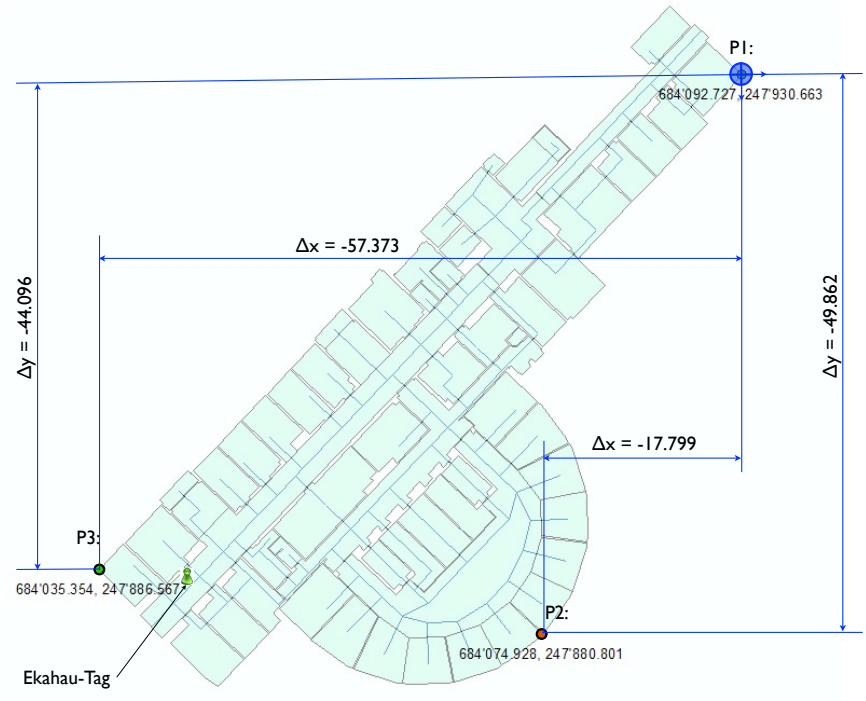

Abbildung 23.5: Georeferenziertes Stockwerk, auf dem die drei selbst gewählten Referenzpunkte eingezeichnet sind.

$$P_{W2} = \begin{pmatrix} P_{W2x} \\ P_{W2y} \end{pmatrix} = \begin{pmatrix} 1046 \\ -468 \end{pmatrix} \tag{23.16}$$

$$P_{W3} = \begin{pmatrix} P_{W3x} \\ P_{W3y} \end{pmatrix} = \begin{pmatrix} 1546 \\ 236 \end{pmatrix} \tag{23.17}$$

Landeskoordinaten

Für den gleichen Stockwerkplan wie in Abschnitt 23.2.4 (Seite 220) wurden im georeferenzierten Dokument die drei Punkte gewählt. Die Punkte entsprechen den in Abschnitt 23.2.4 (Seite 220) gewählten Punkten. In Abbildung 23.5 (Seite 221) sind diese zusammen mit den jeweiligen absoluten Landeskoordinaten und den Distanzen zum Basispunkt (P_1) eingezeichnet.

Gemäss Formel 23.14 (Seite 220) und 23.15 (Seite 220) ergeben sich folgende Koordinaten für die Punkte P_2 und P_3:

$$P_{LK2} = \begin{pmatrix} P_{LK2x} \\ P_{LK2y} \end{pmatrix} = \begin{pmatrix} -17.799 \\ -49.862 \end{pmatrix} \tag{23.18}$$

$$P_{LK3} = \begin{pmatrix} P_{LK3x} \\ P_{LK3y} \end{pmatrix} = \begin{pmatrix} -57.373 \\ -44.096 \end{pmatrix} \tag{23.19}$$

Berechnung der Werte für φ und μ

Gemäss den Herleitungen in Abschnitt 23.2.3 (Seite 217 ff.) können die Koordinaten der Bild-Punkte für die Berechnung der Matrix M_e eingesetzt werden. Nach Formel 23.13 (Seite 219) ergibt sich dann folgende Matrix:

$$M_e = \frac{1}{P_{W3x}P_{W2y} - P_{W2x}P_{W3y}} \begin{pmatrix} P_{W2y} & -P_{W2x} \\ -P_{W3y} & P_{W3x} \end{pmatrix} \tag{23.20}$$

$$= \frac{1}{(1546 \cdot -468) - (1046 \cdot 236)} \begin{pmatrix} -468 & -1 \cdot 1046 \\ -1 \cdot 236 & 1546 \end{pmatrix} \tag{23.21}$$

$$= \begin{pmatrix} \frac{117}{242596} & \frac{523}{485192} \\ \frac{59}{242596} & \frac{-773}{485192} \end{pmatrix} \tag{23.22}$$

Als Beispiel für die Ekahau-Antwort wurde die in Abschnitt 8.1 (Seite 42) beschriebene verwendet. In den Abbildungen 23.4 (Seite 220) und 23.5 (Seite 221) ist die Position des Tags ebenfalls gekennzeichnet.

Mit der Formel 23.7 (Seite 219) können nun aus der Matrix M_e und dem Punkt aus der Ekahau-Antwort die Variablen φ und μ berechnet werden. Dabei ist noch zu beachten, dass die Pixel-Position relativ zum Basispunkt und nicht absolut verwendet werden sollte. Das ergibt nun für den Punkt A die Koordinaten $A_x = 1439$ und $A_y = 87$. Entsprechend ergibt sich für A_V:

$$\begin{pmatrix} \varphi \\ \mu \end{pmatrix} = A_V = \begin{pmatrix} \frac{117}{242596} & \frac{523}{485192} \\ \frac{59}{242596} & \frac{-773}{485192} \end{pmatrix} \cdot \begin{pmatrix} 1439 \\ 87 \end{pmatrix} = \begin{pmatrix} \frac{382227}{485192} \\ \frac{102551}{485192} \end{pmatrix} \tag{23.23}$$

Nach der Formel23.4 (Seite 217) kann nun der Punkt A' berechnet werden:

$$A' = \frac{382227}{485192} \begin{pmatrix} -57.373 \\ -44.096 \end{pmatrix} + \frac{102551}{485192} \begin{pmatrix} -17.799 \\ -49.862 \end{pmatrix} + \begin{pmatrix} 684092.727 \\ 247930.663 \end{pmatrix} \tag{23.24}$$

$$\begin{pmatrix} A'_x \\ A'_y \end{pmatrix} = \begin{pmatrix} 684043.767 \\ 247885.386 \end{pmatrix} \tag{23.25}$$

Dass die Position des Tags stimmt, sieht man, wenn man die Koordinaten für A' auf der georeferenzierten Karte einsetzt (Abbildung 23.6, Seite 223).

23.2.5 Benötigte Daten zur Umrechnung

Für die Umrechnung vom Bild-System in das Landeskoordinaten-System werden folgende Daten benötigt:

- Bezeichnung der Ekahau-Map
- Höhe in Meter über Meer
- Drei Referenzpunkte auf der Ekahau-Map (Angabe der x- und y-Koordinaten in Pixel)
- Drei entsprechende Referenzpunkte auf der georeferenzierten Karte (Angabe der x- und y-Koordinaten in Landeskoordinaten)

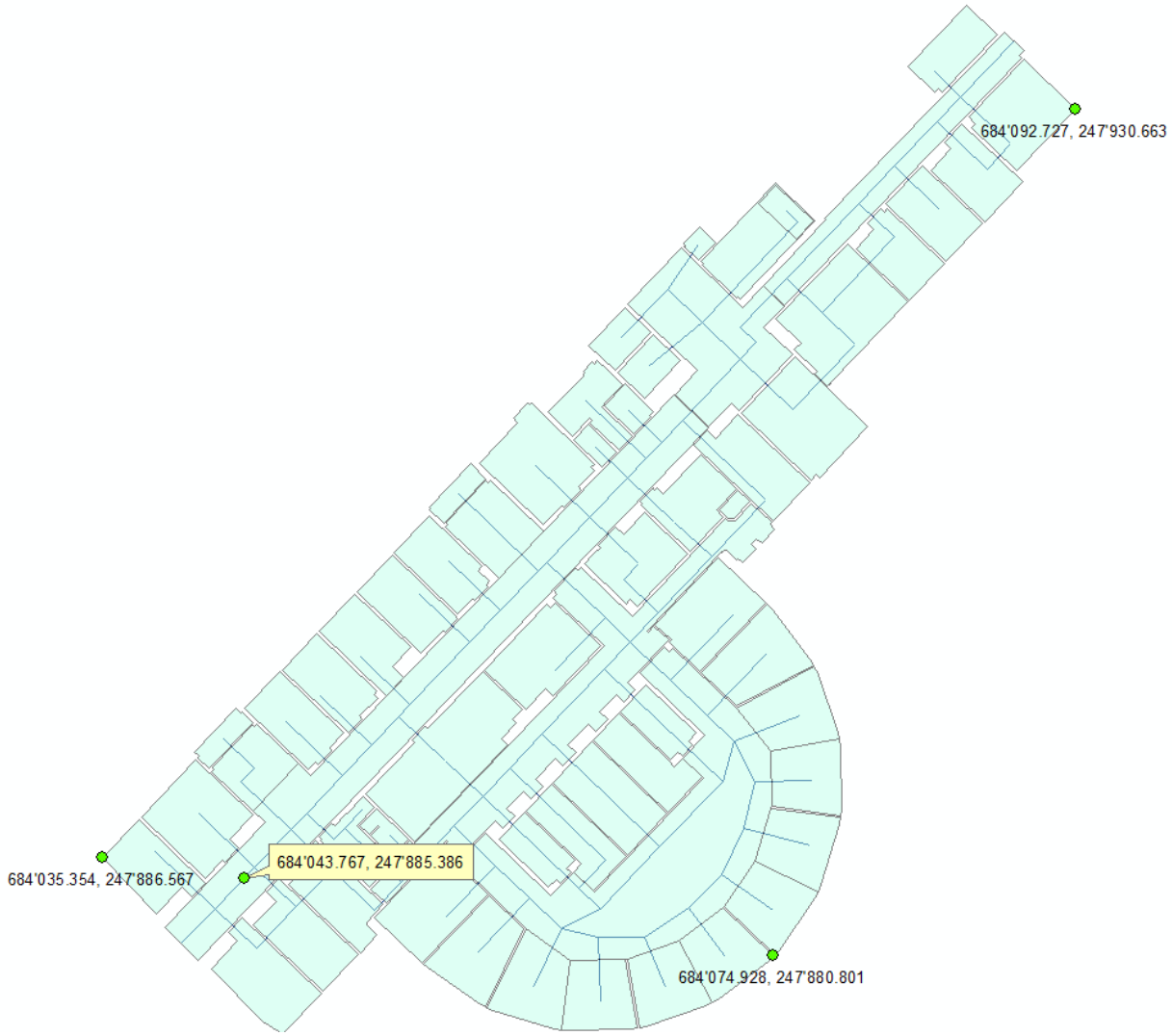

Abbildung 23.6: Die errechnete Position des Ekahau-Tags stimmt mit dem erwarteten Resultat in Abbildung 23.5 (Seite 221) überein

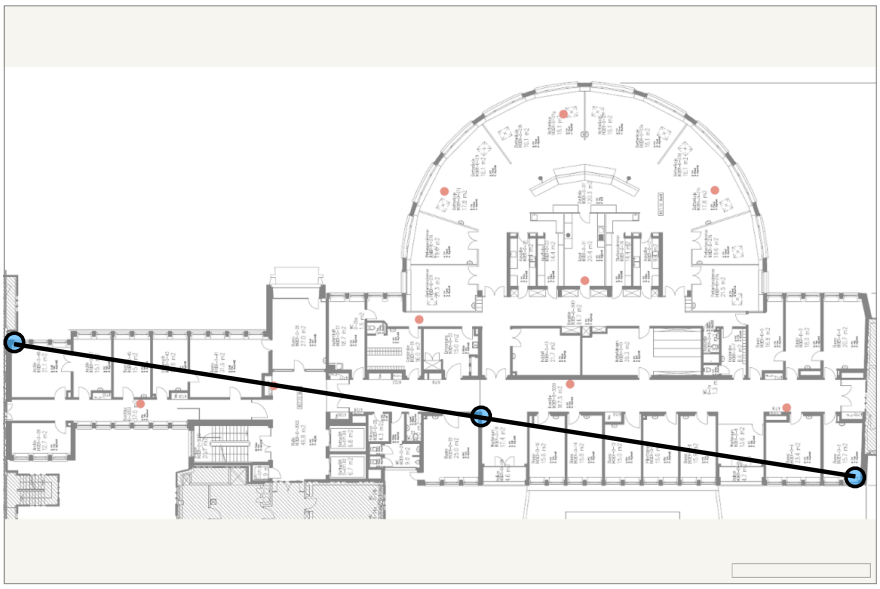

Abbildung 23.7: Beispiel für eine schlechte Wahl der Bezugspunkte

23.2.6 Anforderungen und Empfehlungen

Anforderungen an das Bild der Ekahau-Map

Damit insgesamt ein ausreichend gutes Ergebnis der Umrechnung erreicht werden kann, müssen die Bilder, die als Ekahau-Map für das Site Survey verwendet werden, eine gewisse Genauigkeit vorweisen. Wie bereits in [1] beschrieben, muss ein Pixel etwa 0.05 Meter entsprechen. Das heisst für die Bilder, welche für das Site Survey verwendet werden, dass ein Gebäude von 72 Metern Breite eine Breite in Pixel von mindestens 2000 aufweisen muss.

Referenzpunkte

Grundsätzlich ist die Wahl der Referenzpunkte frei. Dabei muss einzig darauf geachtet werden, dass die Punkte sich nicht in einer Linie befinden (siehe Abbildung 23.7, Seite 224). Zusätzlich empfiehlt es sich, da die Position der Referenzpunkte manuell gemessen werden muss, markante Punkte auf der Karte zu wählen. Dies kann zum Beispiel eine Ecke eines Raumes sein.

Falls in Ekahau die gleichen Stockwerkpläne verwendet werden, die auch als CAD-Pläne für die Georeferenzierung verwendet werden, können (falls vorhanden) auch die Referenzpunkte, die auf den Plänen eingezeichnet sind, weiter verwendet werden. Dabei besteht zusätzlich die Möglichkeit, die Referenzpunkte beim Einzeichnen der Sektoren in den CAD-Plänen einzuzeichnen und die Referenz-Datenbank direkt über den Shapeconverter zu generieren.

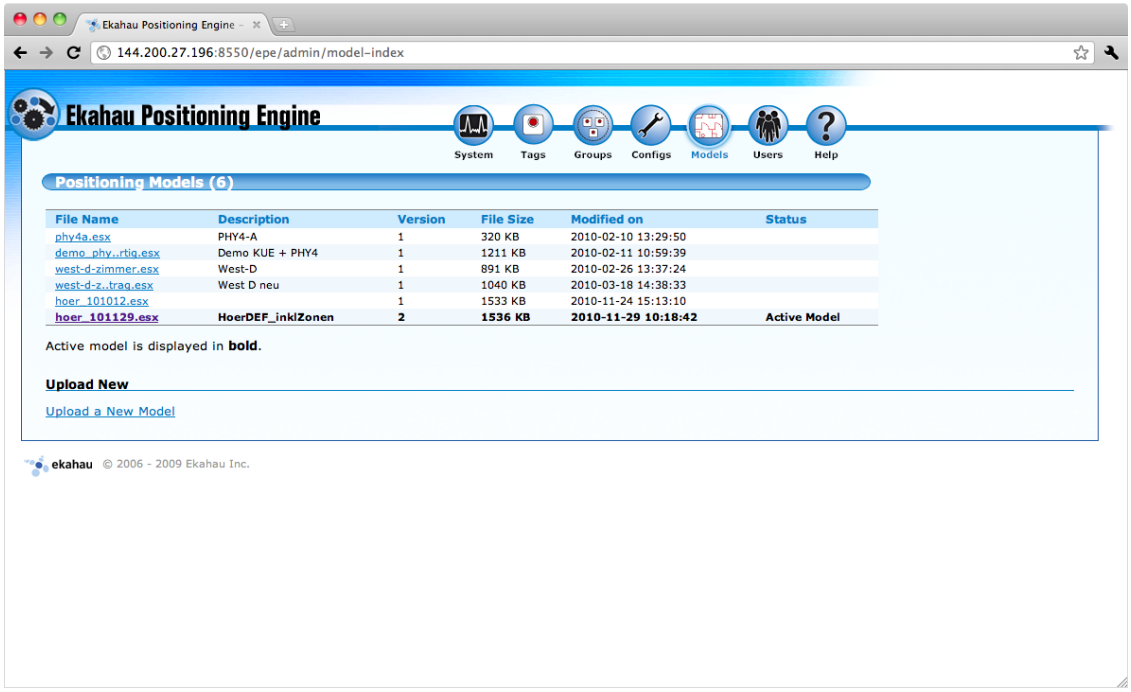

Abbildung 23.8: In der Ekahau Positioning Engine sind die Models aufgelistet

Ekahau Map aus Ekahau Positioning Engine abrufen

Für die Bestimmung der Position der Referenzpunkte auf der Ekahau-Map muss die Bilddatei vorhanden sein, die für das Site Survey eingelesen wurde. Hierfür kann die Bilddatei nicht einfach aus dem Site Survey heraus kopiert werden, da die Datei unter Umständen durch die Ekahau-Software skaliert wurde.

Die Originalbilder können, sofern nicht mehr vorhanden, über die Ekahau Positioning Engine unter dem Menüpunkt «Models» abgerufen werden. Dort sind alle Modelle verzeichnet. Das im Moment aktive Modell wird fett dargestellt (Abbildung 23.8, Seite 225). Klickt man darauf, kann man auf der Detail-Ansicht das Modell herunterladen (Abbildung 23.9, Seite 226). Als Resultat erhält man eine Datei mit der Endung `.esx`, wobei es sich aber um eine ganz gewöhnliche ZIP-Datei handelt. Dort drin sind unter anderem die Bilder im Original enthalten, welche für die Bestimmung der Referenzpunkte verwendet werden können.

23.3 Umsetzung der Genauigkeit

Die Genauigkeit der Positionsangabe wird von Ekahau in Prozent angegeben: 100 Prozent heisst absolut genau, 0 Prozent heisst absolut ungenau. Diese Werte müssen auf die geforderten Werte `GOOD`, `MEDICORE`, `POOR` abgebildet werden. Für eine erste Testphase wurden die Werte wie folgt eingeteilt:

- `GOOD`: 69% - 100%
- `MEDICORE`: 36% - 68%
- `POOR`: 0% - 35%

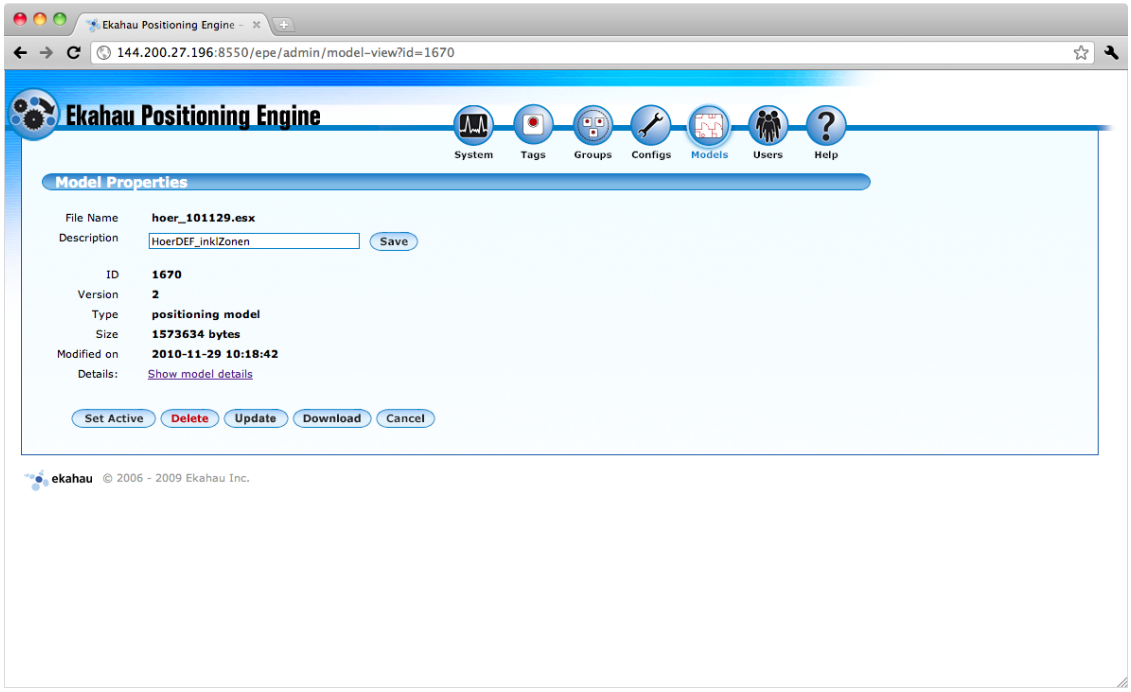

Abbildung 23.9: Das aktive Modell mit der Endung .esx kann heruntergeladen werden

23.4 Architektur

23.4.1 Packages

Abbildung 23.10 (Seite 227) zeigt die Package-Gestaltung des Ekahau Pull-Konnektors.

ch.hsr.ins.eagleeye.connector.ekahau21781

Die Klasse `Ekahau21781PullConnector` implementiert das `PullConnector`-Interface. Um die Zuständigkeiten sauber zu trennen, wird die Generierung und Umrechnung mittels Delegate Pattern an die Klasse `Ekahau21781PullConnectorDelegate` weitergegeben. Diese erzeugt dann den `PositionReport`.

ch.hsr.ins.eagleeye.connector.ekahau21781.calculator

Das Package `ch.hsr.ins.eagleeye.connector.ekahau21781.calculator` beinhaltet die Klassen, die für die Umrechnung der Ekahau Position in die Position in Landeskoordinaten benötigt werden.

ch.hsr.ins.eagleeye.connector.ekahau21781.springdata

Das Package `ch.hsr.ins.eagleeye.connector.ekahau21781.springdata` beinhaltet die Service-Klassen für den Zugriff auf die MongoDB. Mit der Unterstützung von Spring Data[1] können die Abfragen einfach gehandhabt werden.

[1] `http://www.springsource.org/spring-data` (abgerufen: 4. Juni 2011)

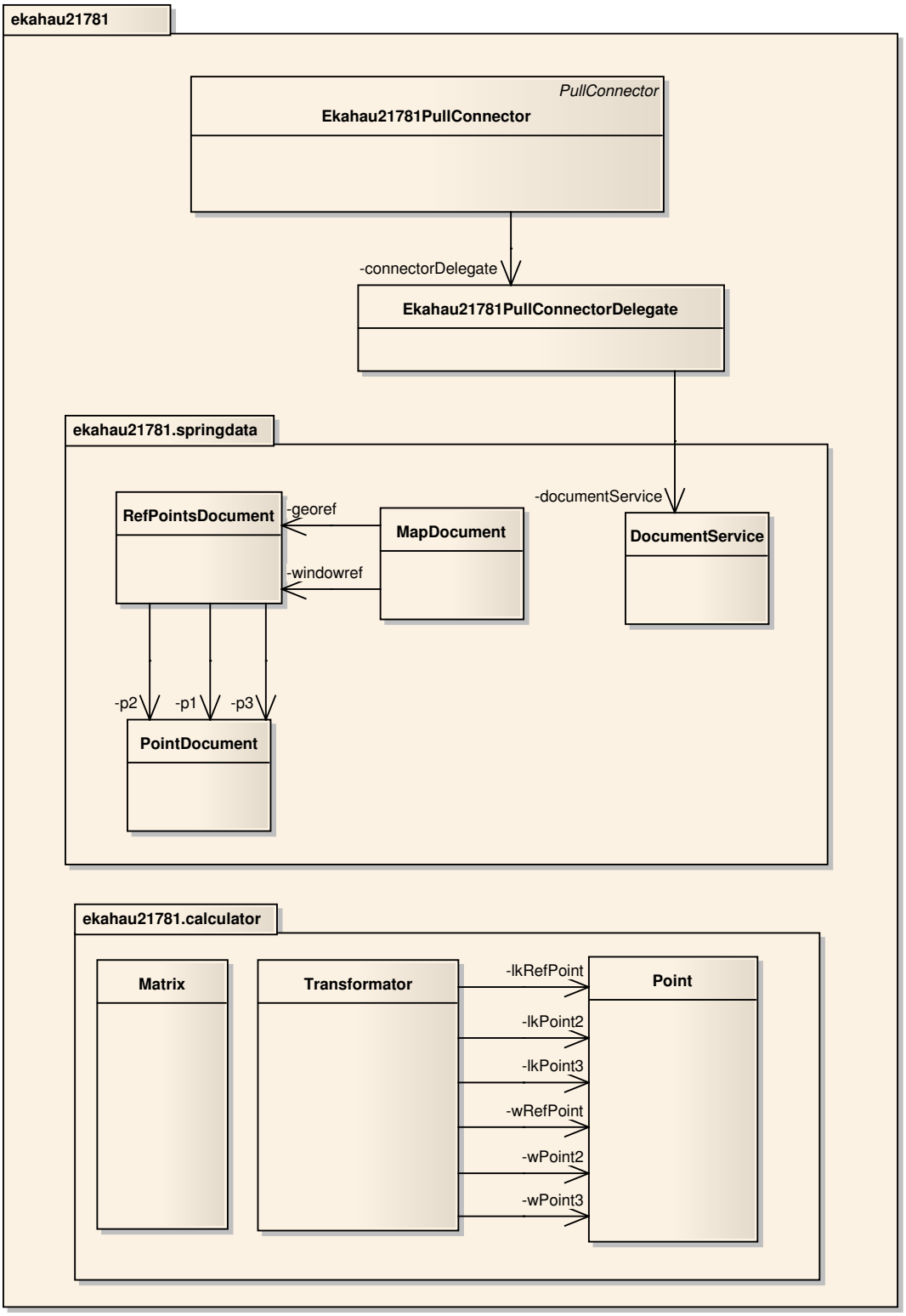

Abbildung 23.10: Ekahau Packages

23.4.2 Technologiekomponenten

Liste der zusätzlichen verwendeten Programmbibliotheken

Ekahau SDK SDK zur Interaktion mit der Ekhau Positioning Engine

Apache Commons Math Mathematik-Bibliothek, verwendet für die Umrechnung der Koordinaten, um dank `BigFraction` genauere Ergebnisse erzielen zu können.

Spring Data Framework zur einfacheren Kommunikation zu MongoDB

Spring Allgemeines Anwendungs-Framework, verwendet für Dipendency Injection und aspektorientierte Programmierung

Liste der verwendeten Programmbibliotheken für Testing

JUnit Unit-Testing-Framework zur Erstellung von Unit Tests

Mockito Mocking-Framework, das die Erzeugung von Fakes, Mocks und Stubs vereinfacht

Hamcast Stellt Matcher zur Verfügung, welche für das Testen verwendet wurden

23.4.3 Real Use Cases

In den Folgenden Abschnitten ist der Ablauf der Positionsabfrage genauer beschrieben. Aufgrund der Übersichtlichkeit wurden die Abfrage in erfolgreiche und nicht erfolgreiche Positionsabfrage unterteilt.

Erfolgreiche Positionsabfrage

Kann eine Positionsabfrage erfolgreich durchgeführt werden, so wird ein `SuccessfulPositionReport` erzeugt und dem Lokalisierungsservice zurückgeliefert. Abbildung 23.11 (Seite 229) zeigt eine solche Abfrage mittels einem White-Box-Systemsequenz-Diagramm.

Nicht erfolgreiche Positionsabfrage

Kann die Positionsabfrage nicht erfolgreich durchgeführt werden, so wird ein `FailedPositionReport` erzeugt und dem Lokalisierungsservice zurückgeliefert. Dies ist der Fall, wenn die Abfrage der Ekahau Positioning Engine nicht erfolgreich war oder die Umwandlung von den Bild-Koordinaten in Landeskoordinaten fehlschlägt. Abbildung 23.12 (Seite 231) zeigt eine solche Abfrage mittels einem White-Box-Systemsequenz-Diagramm. Es handelt sich dabei um die fehlschlagende Variante, bei der die Ekahau Positioning Engine die gesuchte Adresse nicht lokalisieren kann und daher `null` zurück liefert.

23.4.4 Daten

Listing 23.1 (Seite 230) zeigt an einem Beispiel, wie die in Abschnitt 23.2.5 (Seite 222) definierten Referenzdaten in der MongoDB gespeichert werden.

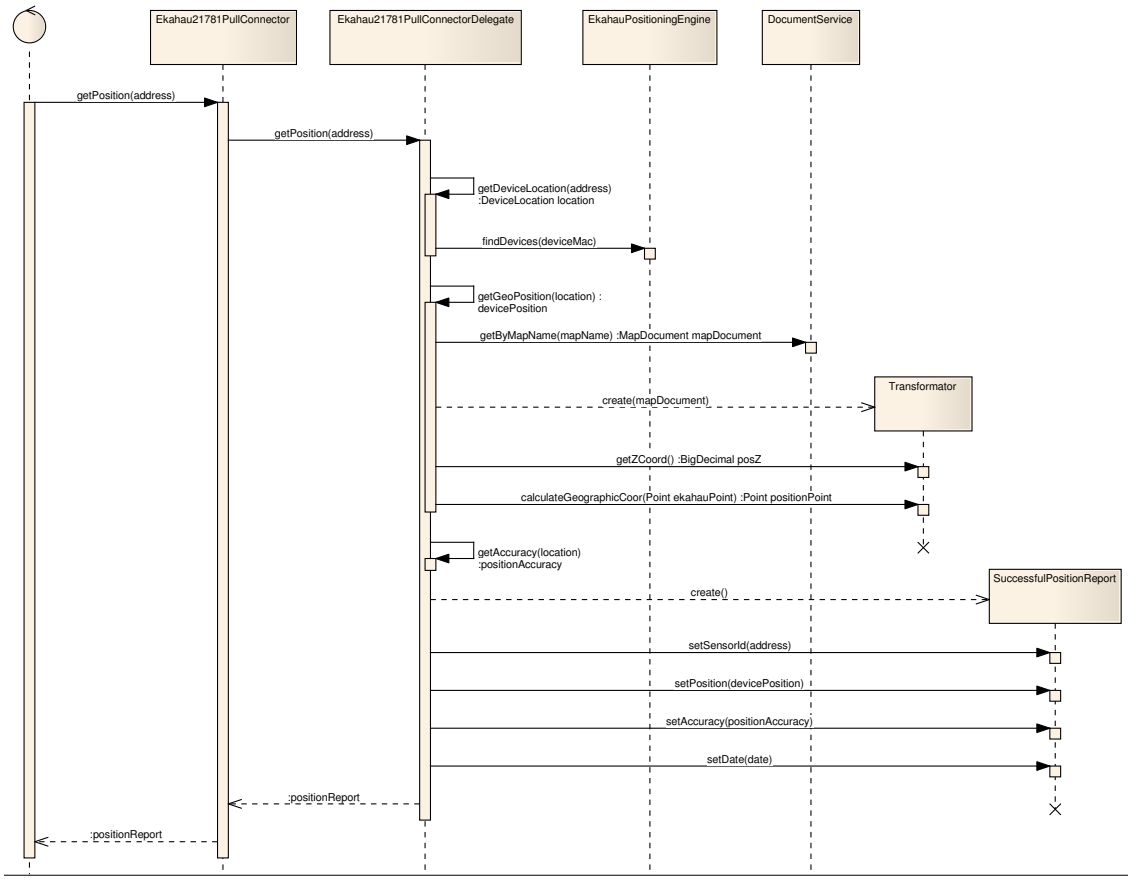

Abbildung 23.11: Das Systemsequenz-Diagramm zeigt den Ablauf bei der Abfrage nach der Position einer gegebenen Adresse eines WLAN-fähigen Gerätes. Es stellt nur den Fall dar, bei dem die gesamte Lokalisierung erfolgreich ausgeführt werden konnte.

```
{
    "mapname" : "hoer_d",
    "height" : 467.1,
    "windowref" : {
         "p1" : {
            "x" : 16,
            "y" : 513
        },
        "p2" : {
            "x" : 1062,
            "y" : 45
        },
        "p3" : {
             "x" : 1562,
            "y" : 749
        }
    },
    "georef" : {
        "p1" : {
            "x" : 684092.727,
            "y" : 247930.663
        },
        "p2" : {
            "x" : 684074.928,
            "y" : 247880.801
        },
        "p3" : {
            "x" : 684035.354,
            "y" : 247886.567
        }
    }
}
```

Listing 23.1: Beispiel der Referenzdaten, welche in der MongoDB gespeichert werden.

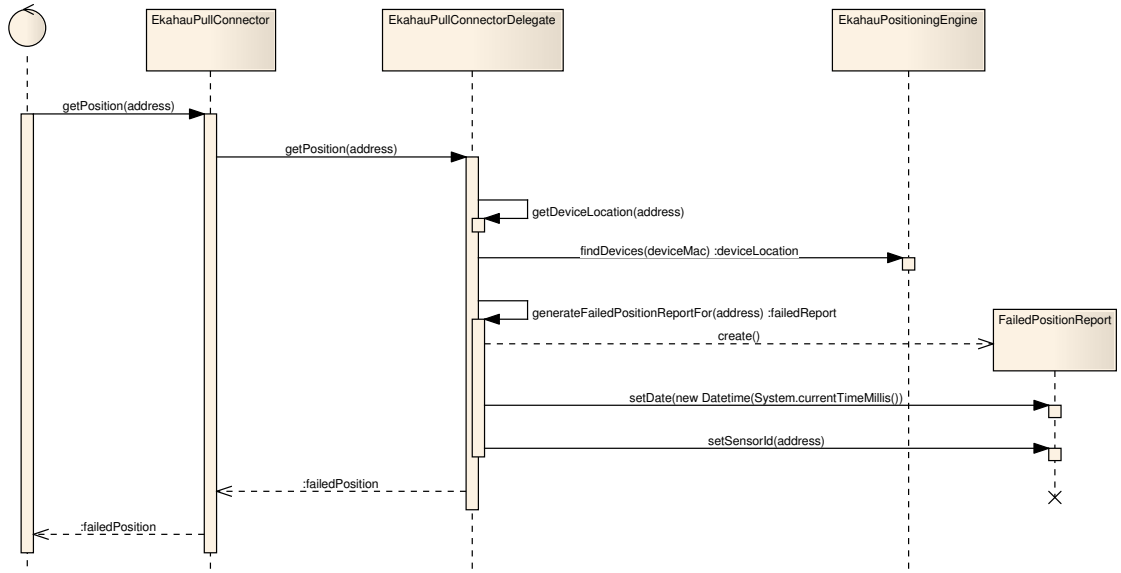

Abbildung 23.12: Das Systemsequenz-Diagramm zeigt den Ablauf bei der Abfrage nach der Position einer gegebenen Adresse eines WLAN-fähigen Gerätes. Da die Ortung auf der Stufe der Ekahau Positioning Engine fehlschlägt, wird ein `FailedPositionReport` erzeugt.

23.4.5 Konfigurationen

Gemäss der Beschreibung in Abschnitt 22.4 (Seite 213) sollen konnektorspezifische Einstellungen konfigurierbar sein.

Datenbank der Ekahau Positioning Engine

Listing 23.2 (Seite 232) zeigt die Konfigurationen der Ekahau Positioning Engine. Nachfolgend eine Beschreibung der einzelnen Konfigurationsoptionen:

connector.ekahau21781.epe.host IP-Adresse des Servers, auf der die Ekahau Positioning Engine läuft

connector.ekahau21781.epe.port Port für den Zugriff auf die Ekahau Positioning Engine

connector.ekahau21781.epe.username Username für den Zugriff auf die Ekahau Positioning Engine

connector.ekahau21781.epe.passwort Passwort für den Zugriff auf die Ekahau Positioning Engine

Übersetzungsdatenbank (MongoDB)

Listing 23.3 (Seite 232) zeigt die Konfigurationsoptionen der MongoDB. Nachfolgend eine Beschreibung der einzelnen Konfigurationsoptionen:

```
<bean id="ekahauPositioningEngine"
    class="com.ekahau.engine.sdk.PositioningEngine">
        <constructor-arg index="0"
            value="${connector.ekahau21781.epe.host}"/>
        <constructor-arg index="1"
            value="${connector.ekahau21781.epe.port}"/>
        <constructor-arg index="2"
            value="${connector.ekahau21781.epe.username}"/>
        <constructor-arg index="3"
            value="${connector.ekahau21781.epe.password}"/>
</bean>
```

Listing 23.2: Konfiguration des Zugriffes auf die Ekahau Positioning Engine

```
<mongo:mongo id="mongo" host="${connector.ekahau21781.mongodb.host}"
             port="27017"/>

<bean id="mongoTemplate"
      class="org.springframework.data.document.mongodb.MongoTemplate">
    <constructor-arg ref="mongo"/>
    <constructor-arg
        value="${connector.ekahau21781.mongodb.db}"/>
    <constructor-arg
        value="${connector.ekahau21781.mongodb.collection}"/>
</bean>
```

Listing 23.3: Konfiguration des Zugriffes auf die Referenzdatenbank

connector.ekahau21781.mongodb.host IP des Server, auf welchem MongoDB läuft. Aufgrund eines Bugs in `spring-mongo-1.0.xsd` ist es nicht möglich, den Parameter für den Port zu ersetzen. Daher ist dieser Wert nicht momentan konfigurierbar.

connecotr.ekahau21781.mongodb.db Datenbankname der Referenzdatenbank

connector.ekahau21781.mongodb.collection Bezeichnung der Collection in der Referenzdatenbank, in welcher die Referenzdaten abgelegt sind.

24 CiscoWorks Pull-Konnektor

24.1 Einführung

Wie in der Analyse in Abschnitt 8.2 (Seite 43) beschrieben, besteht für die CiscoWorks-Anbindung nur die Möglichkeit, diese in Form eines Pull-Konnektors zu realisieren. Die allgemeine Umsetzung eines Pull-Konnektors ist in Abschnitt 20.7 (Seite 189) beschrieben. CiscoWorks wird von diesem nach der Port Description gefragt, für welche die MAC-Adresse des zu lokalisierenden Geräts gemeldet ist. Aus der erhaltenen Beschreibung wird die Position ausgelesen und in Landeskoordinaten umgewandelt.

Kann für die Adresse eine zugewiesene Port-Beschreibung gefunden werden und lässt sich daraus die Position in Landeskoordinaten ableiten, so wird ein `SuccessfulPositionReport` generiert. Dazu wird der Zeitpunkt der Abfrage sowie die Genauigkeit der Position angegeben. Kann zu der gesuchten Adresse keine Port Description gefunden werden oder kann eine gefundene Beschreibung nicht einem Raum und den dazugehörigen Landeskoordinaten zugewiesen werden, so wird ein `FailedPositionReport` generiert.

24.2 Umwandlung der Port-Beschreibung in Landeskoordinaten

Im Falle des Universitätsspital Zürich lässt sich aus der Port-Beschreibung direkt ableiten, in welchem Raum sich ein Gerät befindet. Wie bereits in der Analyse in Abschnitt 8.2 (Seite 43) beschrieben, beinhaltet die Beschreibung die Bezeichnung des Stockwerkes, des Trakts und des Raumes. Obwohl sich auf den ersten Blick anbieten würde, anstelle der geforderten Landeskoordinaten gleich die Position in Form von Raum, Stockwerk und Gebäude weiterzuleiten, wird darauf verzichtet. Grund dafür ist, dass keine Abhängigkeit geschaffen werden soll, wie die Port-Beschreibung auszusehen hat, und zusätzlich die Schnittstelle der Pull-Konnektoren einheitlich bleiben soll.

24.2.1 Umwandulung der Port-Beschreibung in Raum, Stockwerk, Gebäude

Listing 24.1 (Seite 233) beschreibt die Regular-Expression, mit welcher aus der Port-Beschreibung die Position in Form von Raum, Stockwerk und Gebäude ausgelesen werden kann.

```
(al|sl)\-([^\-]+)\-([^\-]+)\-([^\-]+)\-([^\-]+)\-([^\-]+)\-([^\-]*)
```

Listing 24.1: Regular Expression zum Auslesen der Position aus der Port-Beschreibung

24.2.2 Umwandlung von Raum, Stockwerk, Gebäude in Landeskoordinaten

Für die Generierung des `PositionReports` ist es notwendig, die aus der Port Description erhaltene Position in Form von Raum, Stockwerk und Gebäude in Landeskoordinaten umzuwandeln. Dies geschieht über die Zuordnung des Raumes zu einer Koordinate in der Mitte des Raumes. Die Zuordnung muss für jeden Raum definiert werden. Um dies einfach zu gestallten, ist es möglich, die Punkte direkt im georeferenzierten Stockwerkplan einzuzeichnen und der Raumbezeichnung zuzuordnen. Mit der Unterstützung des Shapeconverters könnte dann daraus die Datenbank für die Zuordnung abgefüllt werden.

Eine andere Möglichkeit, welche sich gerade im Falle des Universitätsspital Zürich anbieten würde, ist ein Skript, welches die Port-Beschreibungen ausliesst. Daraus werden die Bezeichnungen für den Raum, das Stockwerk und das Gebäude ausgelesen und anhand dieser Informationen in der Geometrie- und Topologiedatenbank der Mittelpunkt des gewünschten Sektors herausgelesen. Am Ende erstellt dieses Skript das Datenbank-Dokument für die Speicherung in der MongoDB. Diese Option funktioniert aber nur, wenn von der Port-Beschreibung auf den Sektor geschlossen werden kann und alle Port-Beschreibungen bekannt sind. Für fehlende Beschreibungen existiert sonst keine Abbildung.

24.2.3 Beispiel

Als Beispiel wird die folgende Port-Beschreibung verwendet:

```
al-fa0/8-D-Hoer-12-VLan 102-
```

Mit der Regular Expression ergibt sich für das Stockwerk «D», für das Gebäude «Hoer» und für den Raum «12». Auf Abbildung 24.1 (Seite 235) ist das entsprechende Zimmer gekennzeichnet. Die Mitte des Raumes befindet sich etwa am Punkt $(684051.107, 247899.31, 467.1)$. Die Beschreibung wird also auf diesen Punkt abgebildet.

24.2.4 Benötigte Daten

Für den Zuordnungsvorgang von der aus der Port-Beschreibung gewonnen Position in die geforderten Landeskoordinaten sind für jede mögliche Raumbezeichnung folgende Daten nötig:

- Bezeichnung des Stockwerkes
- Bezeichnung des Gebäudes
- Bezeichnung des Raumes
- Position in Landeskoordinaten, der die Raumbezeichnung zugeordnet werden soll

24.3 Umsetzung der Genauigkeit

Kann eine Port-Beschreibung für die gesuchte MAC-Adresse gefunden und erfolgreich in Landeskoordinaten umgewandelt werden, so kann davon ausgegangen werden, dass die Positionsangabe sehr genau ist. Im Normalfall befindet sich das Gerät auch in dem Raum, wo sich auch der Port befindet. Deshalb wird als Genauigkeit jeweils der Wert `GOOD` mitgegeben.

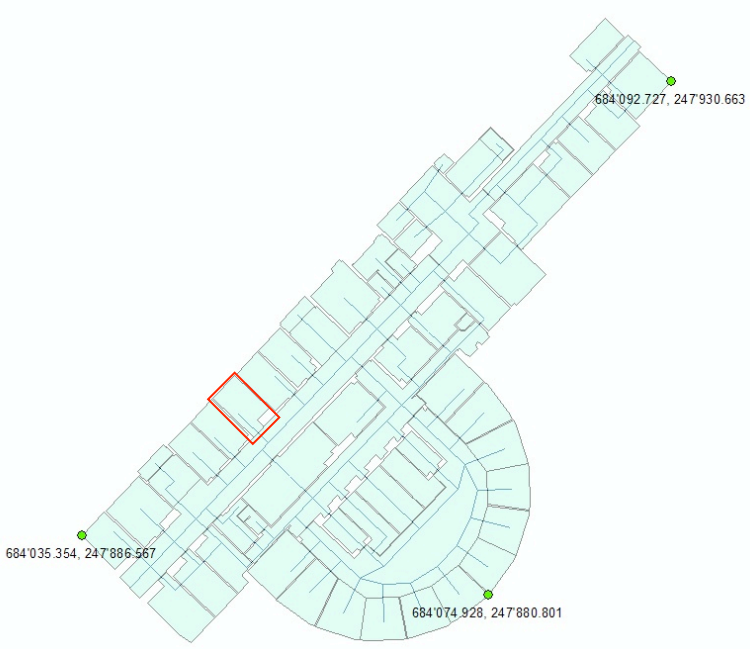

Abbildung 24.1: Der Raum mit der Bezeichnung «Hoer D Raum 12» befindet sich an der rot markierten Stelle.

24.4 Architektur

24.4.1 Packages

Abbildung 24.2 (Seite 236) zeigt die Package-Gestaltung des CiscoWorks-Konnektors.

ch.hsr.ins.eagleeye.connector.ciscoworks21781 Die Klasse `CiscoWorks21781PullConnector` implementiert das `PullConnector`-Interface. Um die Zuständigkeit sauber zu Trennen, wird die Abfrage der Port Description und das Zuweisen der entsprechenden Landeskoordinaten mittels Delegate Pattern über die Klasse `CiscoWorks21781PullConnecotrDelegate` ausgeführt. Diese erzeugt dann auch den `PositionReport`.

ch.hsr.ins.eagleeye.connector.ciscoworks21781.dataaccess Das Package `dataaccess` beinhaltet die Klassen, welche für den Zugriff auf die CiscoWorks-Datenbank benötigt werden.

ch.hsr.ins.eagleeye.connector.ciscoworks21781.springdata Das Package `springdata` beinhaltet die Service Klassen für den Zugriff auf die MongoDB. Mit der Unterstützung von Spring Data[1] können die Abfragen einfach gehandhabt werden.

24.4.2 Technologiekomponenten

Liste der zusätzlichen verwendeten Programmbibliotheken

Sybase jConnect JDBC-Treiber für die Sybase-Datenbank von CiscoWorks

[1] `http://www.springsource.org/spring-data` (abgerufen: 4. Juni 2011)

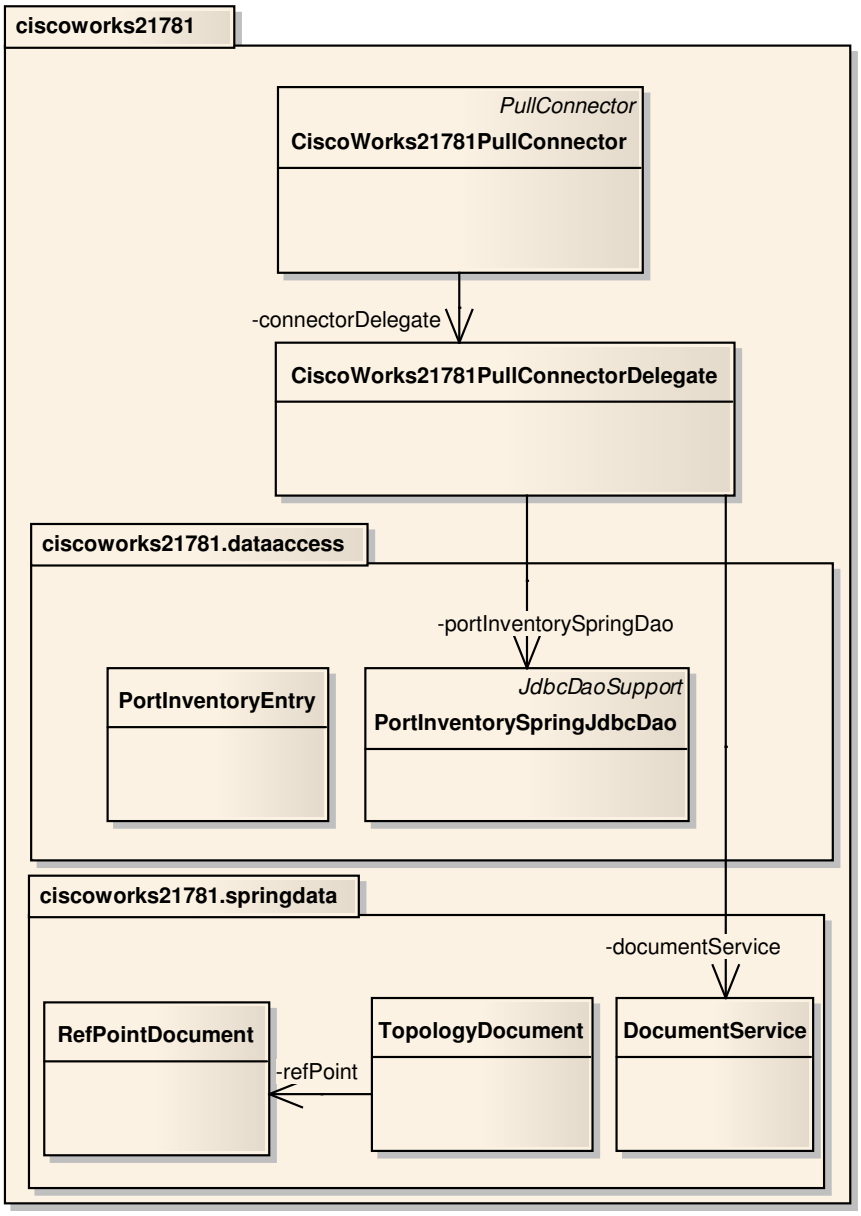

Abbildung 24.2: Packages des CiscoWorks-Konnektors

Liste der verwendeten Programmbibliotheken für Testing

JUnit Unit-Testing-Framework zur Erstellung von Unit Tests

Mockito Mocking-Framework, das die Erzeugung von Fakes, Mocks und Stubs vereinfacht

Hamcast Stellt Matcher zur Verfügung, welche für das Testen verwendet wurden

24.4.3 Zugriff auf die CiscoWorks-Datenbank

Die Port-Beschreibung zu einer MAC-Adresse wird über die CiscoWorks-Datenbank abgefragt[2]. Da CiscoWorks die Daten in einer Sybase-Datenbank hält, kann diese über einen passenden JDBC-Treiber abgefragt werden.

24.4.4 Real Use Cases

Im folgenden Abschnitt ist der Ablauf der Positionsabfrage genauer beschrieben. Aufgrund der Übersichtlichkeit wurde die Abfrage in erfolgreiche und nicht erfolgreiche Positionsbestimmung unterteilt.

Erfolgreiche Positionsbestimmung

Kann die Port-Beschreibung erfolgreich abgefragt werden und kann die daraus erhaltenen Position in Form von Raum, Stockwerk und Gebäude einer Position in Landeskoordinaten zugewiesen werden, so wird ein `SuccessfulPositionReport` erzeugt und an den Lokalisierungsservice zurückgeliefert. Abbildung 24.3 (Seite 238) illustriert eine solche Abfrage durch ein White-Box-Sequenzdiagramm.

Nicht erfolgreiche Positionsabfrage

Kann die Port-Beschreibung nicht erfolgreich abgefragt werden oder kann aus der erhaltenen Position die Umsetzung in Landeskoordinaten nicht gemacht werden, so wird ein `FailedPositionReport` erzeugt und an den Lokalisierungsservice zurückgeliefert. Abbildung 24.4 (Seite 239) zeigt die Abfrage, bei der keine Port-Beschreibung für die gesuchte Adresse gefunden wird mittels einem White-Box-Sequenzdiagramm.

24.4.5 Daten

Listing 24.2 (Seite 238) zeigt an einem Beispiel, wie die in Abschnitt 24.2.4 (Seite 234) definierten Referenzdaten in der MongoDB gespeichert werden.

24.4.6 Konfiguration

Gemäss der Beschreibung in Abschnitt 22.4 (Seite 213) sollen konnektorspezifische Einstellungen konfigurierbar sein.

[2] `http://www.cisco.com/en/US/docs/net_mgmt/ciscoworks_lan_management_solution/4.0/database_schema4.0/guide/dbviews.html` (12. Mai 2011)

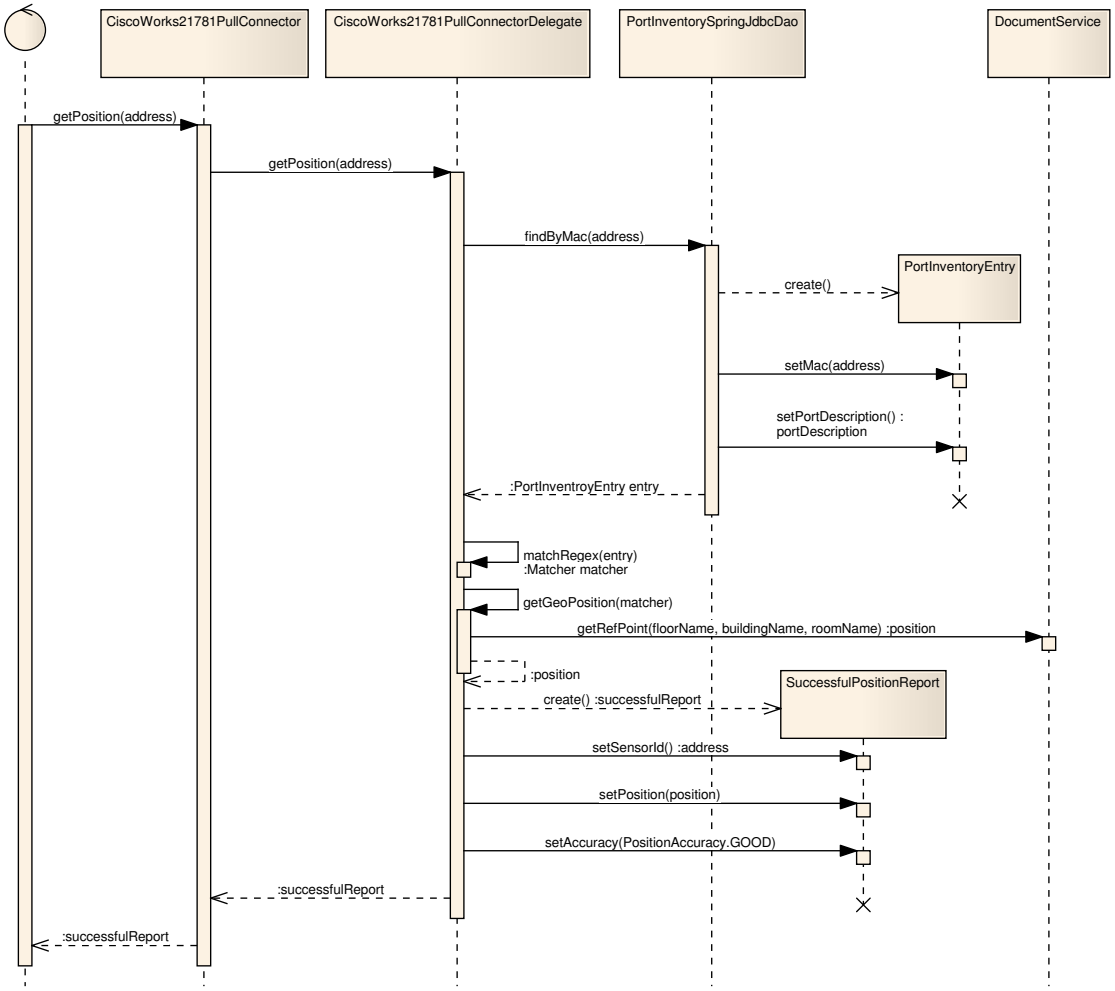

Abbildung 24.3: Das Sequenz-Diagramm zeigt den Ablauf bei einer erfolgreichen Abfrage nach der Position zu der gegebenen Adresse.

```
{
    "floor" : "D",
    "building" : "Hoer",
    "room" : "12␣links",
    "refPoint" : {
        "x" : 684051.107
        "y" : 247899.31
        "z" : 467.1
    }
}
```

Listing 24.2: Beispiel der Referenzdaten, welche in der MongoDB gespeichert werden

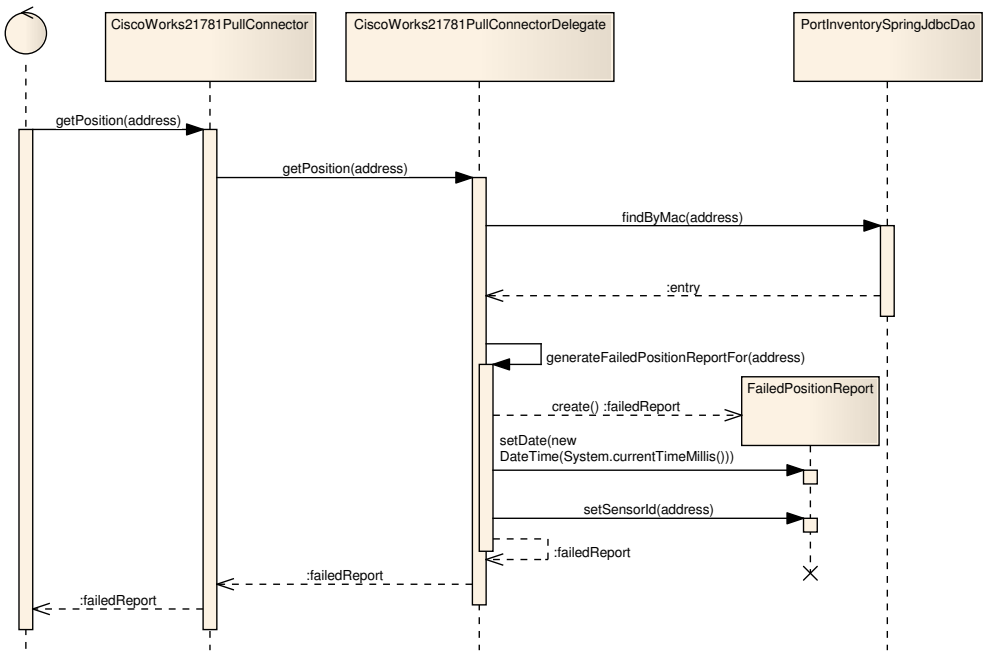

Abbildung 24.4: Das Sequenz-Diagramm zeigt den Ablauf einer nicht erfolgreichen Abfrage nach der Position zu der gegebenen Adresse.

CiscoWorks-Datenbank

Listing 24.3 (Seite 240) zeigt die Konfigurationsoptionen der CiscoWorks-Datenbank. Nachfolgend eine Beschreibung der einzelnen Konfigurationsoptionen:

connector.ciscoworks21781.ciscoworks.url URL der Datenbank, in der die Port-Beschreibungen gespeichert sind

connector.ciscoworks21781.ciscoworks.username Datenbank-Benutzer

connector.ciscoworks21781.ciscoworks.password Passwort des Datenbank-Benutzers

Übersetzungsdatenbank (MongoDB)

Listing 24.4 (Seite 240) zeigt die Konfigurationsoptionen der MongoDB. Nachfolgend eine Beschreibung der einzelnen Konfigurationsoptionen:

connector.ciscoworks21781.mongodb.host IP des Servers, auf welchem MongoDB läuft. Aufgrund eines Bugs in `spring-mongo-1.0.xsd` ist es nicht möglich, den Parameter für den Port zu ersetzen. Daher ist dieser Wert nicht aktuell konfigurierbar.

connector.ciscoworks21781.mongodb.db Datenbankname der Referenzdatenbank

connector.ciscoworks21781.mongodb.collection Bezeichnung der Collection auf der Referenzdatenbank, in welcher die Referenzdaten abgelegt sind

```
<bean id="dataSource" class=
    "org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName"
        value="com.sybase.jdbc3.jdbc.SybDriver"/>
    <property name="url"
        value="${connector.ciscoworks21781.ciscoworks.url}"/>
    <property name="username"
        value="${connector.ciscoworks21781.ciscoworks.username}"/>
    <property name="password"
        value="${connector.ciscoworks21781.ciscoworks.password}"/>
</bean>
```

Listing 24.3: Konfiguration des Zugriffs auf die CiscoWorks-Datenbank

```
<mongo:mongo id="mongo" host="${connector.ciscoworks21781.mongodb.host}"
             port="27017"/>

<bean id="mongoTemplate"
    class="org.springframework.data.document.mongodb.MongoTemplate">
    <constructor-arg
        ref="mongo"/>
    <constructor-arg
        value="${connector.ciscoworks21781.mongodb.db}"/>
    <constructor-arg
        value="${connector.ciscoworks21781.mongodb.collection}"/>
</bean>
```

Listing 24.4: Konfiguration des Zugriffes auf die Referenzdatenbank

25 Smarphone Push-Konnektor

25.1 Einführung

Bei dem Smartphone-Konnektor handelt es sich wie in der Analyse in Abschnitt 8.3 (Seite 44) beschrieben, um einen Push-Konnektor. Das Smartphone sendet also selbstständig Reports, welche dann in Landeskoordinaten umgewandelt und an den Lokalisierungsservice weitergeleitet werden. Neben der Realisierung der Umwandlung und Weiterleitung der Position wurde eine Formatspezifikation für den Datenaustausch mit Smartphone-Apps erstellt.

25.2 Umwandlung der Koordinaten

Für die Umwandlung von den vom Smartphone gesendeten Positionsangaben existiert eine Näherungsformel. Die Formel ist beim Bundesamt für Landestopografie swisstopo [4] dokumentiert. Ebenfalls von swisstopo existiert eine Umrechnungs-Software in C#[1], welche als Vorbild für die Umrechnungsfunktion verwendet wurde.

25.2.1 Benötigte Daten

Für die Umrechnung von WGS84-Koordinaten in Landeskoordinaten sind keine zusätzlichen Daten nötig.

25.3 Architektur

25.3.1 Packages

Abbildung 25.1 (Seite 242) zeigt die Package-Gestaltung des Smartphone-Konnektors. Nachfolgend sind die einzelnen Packages bezüglich ihrer Funktion beschrieben.

ch.hsr.ins.eagleeye.connector.smartphone21781 Beinhaltet den `RestService` welcher das Interface `JaxRsPushConnector` implementiert. Die Weiterverarbeitung wird gemäss Delegate Pattern an das `SmartphoneRestServiceDelegate` delegiert, welches die Umrechnung in die geforderten Landeskoordinaten vornimmt und den `PositionReport` an den Lokalisierungsservice weiterleitet. Zusätzlich beinhaltet es Adapter-Klassen zum (Un-)Marshalling von Value-Objekten.

ch.hsr.ins.eagleeye.connector.smartphone21781.calculator Das Package `calculator` beinhaltet die Klassen, welche zur Umrechnung von den WGS84-Koordinaten in Landeskoordinaten benötigt werden.

[1] `http://www.swisstopo.admin.ch/internet/swisstopo/de/home/products/software/software.html` (abgerufen: 15. Juni 2011)

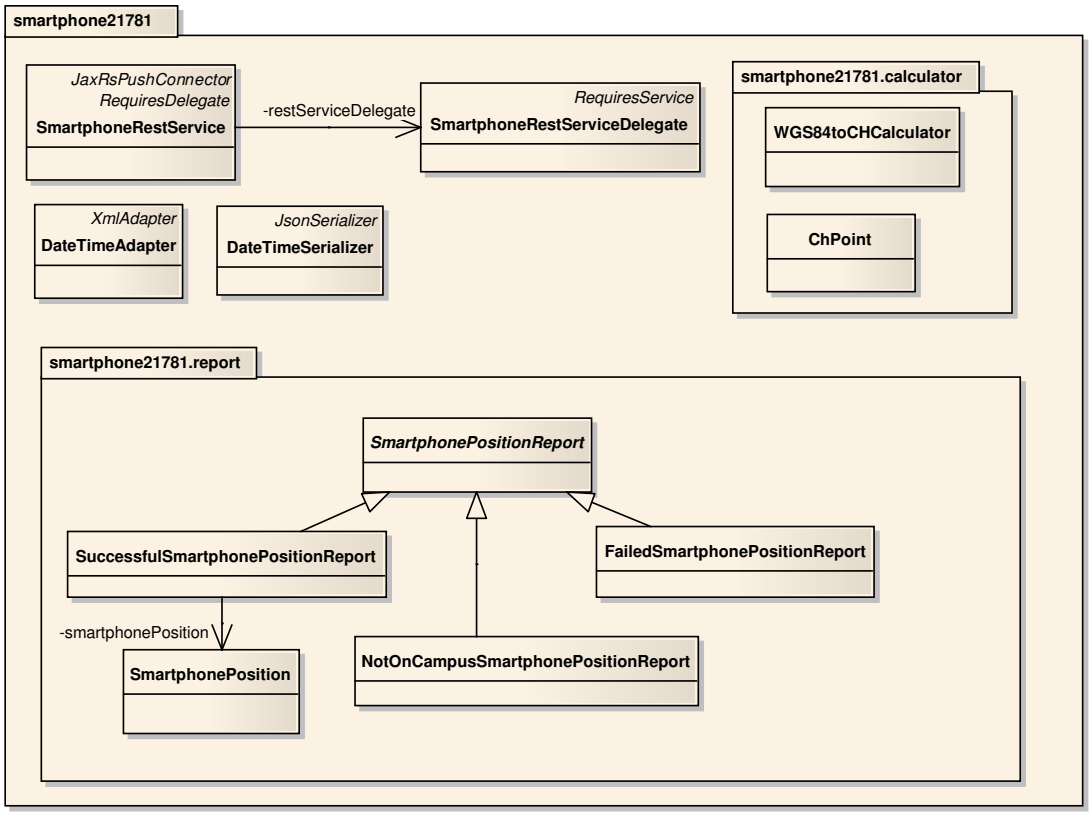

Abbildung 25.1: Smartphone Packages

ch.hsr.ins.eagleeye.connector.smartphone21781.report Das Package `report` beinhaltet die spezifischen Smartphone Reports, mit deren Hilfe die Positionsmeldungen von den Smartphones übermittelt werden.

25.3.2 Technologiekomponenten

Liste der zusätzlichen verwendeten Programmbibliotheken

Es wurden keine zusätzlichen Programmbibliotheken benötigt.

Liste der verwendeten Programmbibliotheken für Testing

JUnit Unit-Testing-Framework zur Erstellung von Unit Tests

Mockito Mocking-Framework, dass die Erzeugung von Fakes, Mocks und Stubs vereinfacht

Spring Test Spring-Komponente, die Testing von Spring-Anwendungen und Testing allgemein vereinfacht

25.3.3 Real Use Cases

In den nachfolgenden Abschnitten ist der Ablauf bei Erhalt einer Positionsmeldung genauer beschrieben. Aufgrund der Übersichtlichkeit wurde die Positionsbestimmung in die drei Varianten unterteilt: erfolgreiche Positionsbestimmung, nicht erfolgreiche Positionsbestimmung und Positionsbestimmung mit einer Position, die ausserhalb der Campus-Grenzen liegt.

Erfolgreiche Positionsbestimmung

Befindet sich das Smartphone auf dem Campus und konnte es die aktuelle Position erfolgreich bestimmen, so sendet das Smartphone einen `SuccessfulSmartphonePositionReport`. Dieser Report wird dann in einen `SuccessfulPositionReport` umgewandelt. Der genaue Ablauf dieses Szenarios ist in Abbildung 25.2 (Seite 244) in Form eines White-Box-Sequenzdiagramms gezeigt.

Nicht erfolgreiche Positionsbestimmung

Kann das Smartphone seine aktuelle Position nicht erfolgreich bestimmen, so sendet es einen `FailedSmartphonePositionReport`. Dieser Report wird dann in einen `FailedPositionReport` umgewandelt. Der genaue Ablauf dieses Szenarios ist in Abbildung 25.3 (Seite 245) in Form eines White-Box-Sequenzdiagramms gezeigt.

Positionsbestimmung «Not On Campus»

Befindet sich das Smartphone nicht auf dem Campus, sendet es einen `NotOnCampusSmartphone-PositionReport`. Dieser Report wird dann in einen `NotOnCampusPositionReport` umgewandelt. Der genaue Ablauf dieses Szenarios ist in Abbildung 25.4 (Seite 245) in Form eines White-Box-Sequenzdiagramms gezeigt.

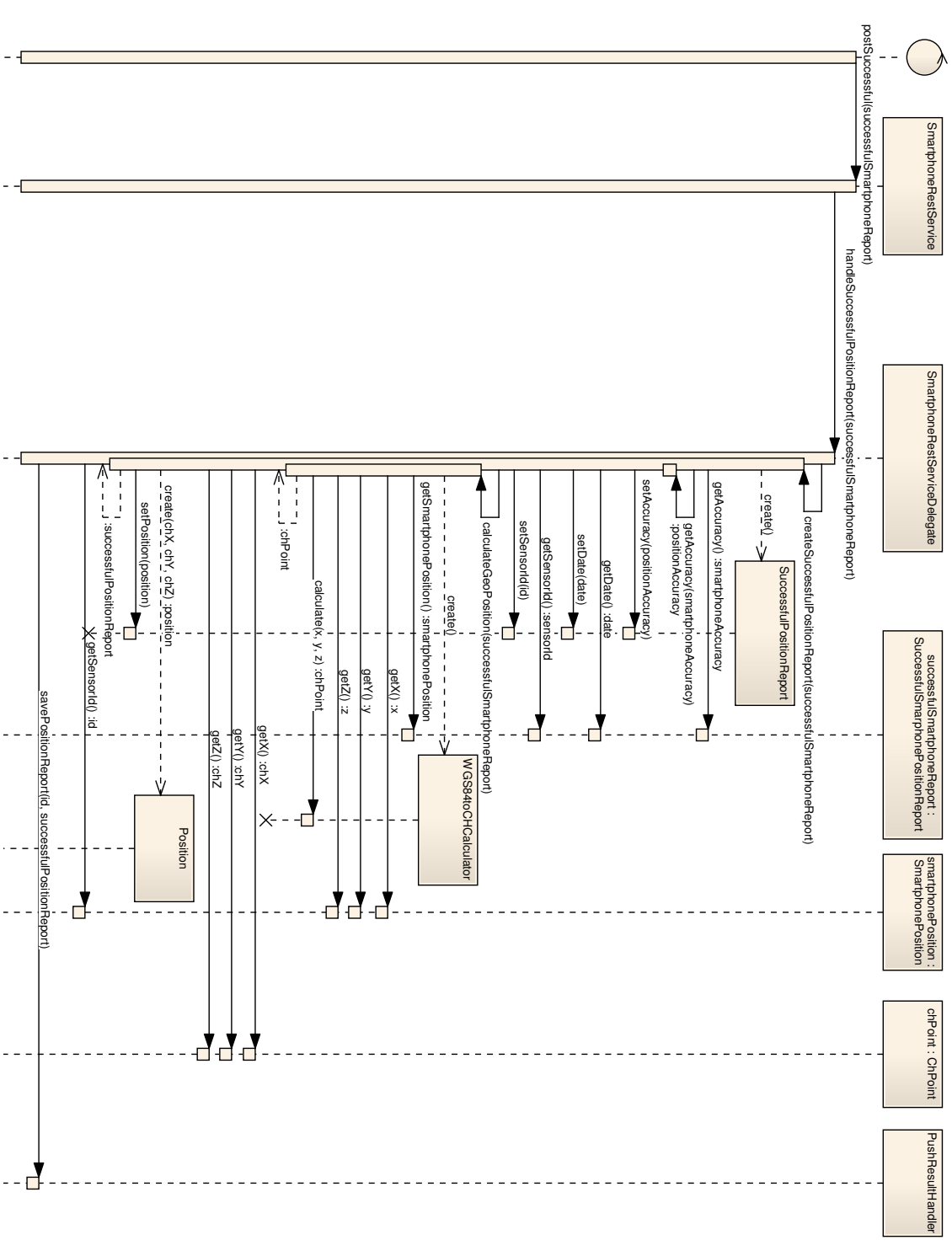

Abbildung 25.2: Das Systemsequenzdiagramm zeigt den Ablauf nach Erhalt einer erfolgreich ermittelten Position.

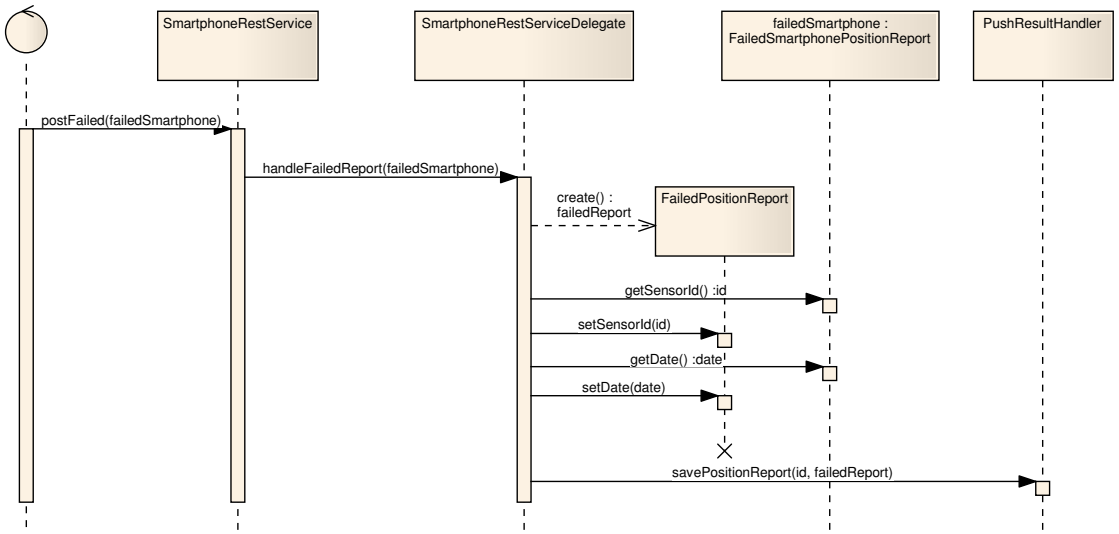

Abbildung 25.3: Das Sequenzdiagramm zeigt den Ablauf nach Erhalt einer nicht erfolgreich ermittelten Position.

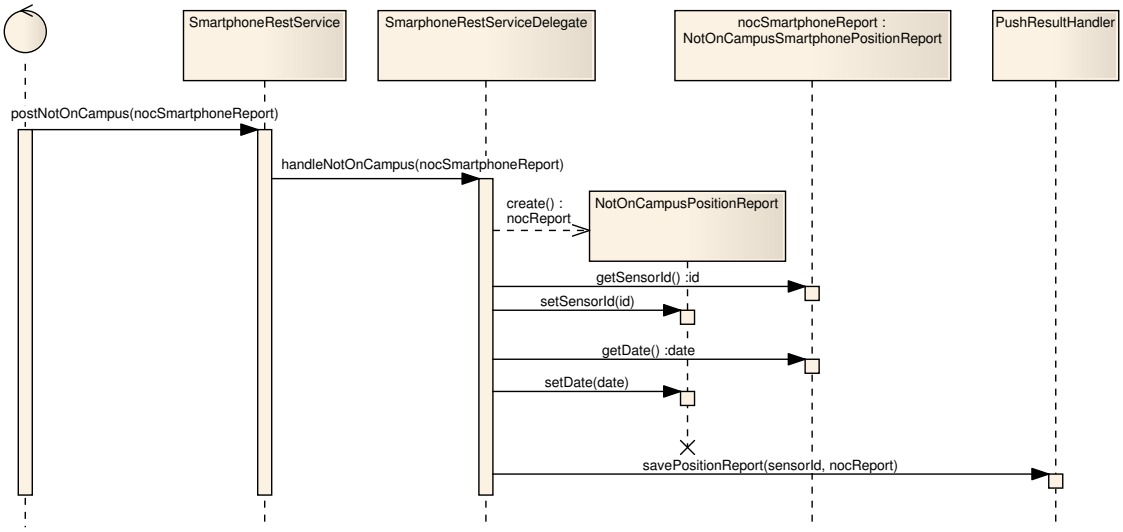

Abbildung 25.4: Das Systemsequenzdiagramm zeigt den Ablauf nach Erhalt einer Position, die ausserhalb der Campus-Grenzen liegt.

Teil VI

Projektmanagement

26 Projektorganisation

26.1 Methodik

Das Projekt wird nach den agilen Prinzipien aus [14] (hauptsächlich Projektmanagement) und einer abgespeckten Variante des Unified Process wie von [13] beschreiben (hauptsächlich Dokumentation) durchgeführt. Dies entspricht dem bewährten Vorgehen aus der dieser Arbeit vorausgegangenen Studienarbeit [1].

26.2 Schnittstellen

Name	Funktion
Beat Stettler	Betreuer der Arbeit
Michael Schneider	Co-Betreuer der Arbeit
Marco Facetti	Co-Betreuer der Arbeit
Andreas Maurer	Ansprechpartner Universitätsspital Zürich
Roman Mendelin	Ansprechpartner Universitätsspital Zürich
Robert Thoma	Ansprechpartner Datenschutz im Universitätsspital Zürich
Markus Kolb	Ansprechpartner Ekahau Site Survey im Universitätsspital Zürich
Thorsten Biskup	Ansprechpartner Ekahau
Andreas Müller	Ansprechpartner Vektorgeometrie zur Umrechnung verschiedener Koordinatensysteme
Olaf Tietje	Ansprechpartner Vektorgeometrie zur Umrechnung verschiedener Koordinatensysteme
Wolfgang Giersche	Ansprechpartner Spring-Framework

Tabelle 26.1: Liste der beteiligten Personen

27 Planung

27.1 Zeitplan

Der grobe Zeitplan, welcher die Arbeiten gemäss Arbeitspaketen pro Tag zeigt, ist in einem separaten Dokument zu finden. Ebenfalls in einem separaten Dokument ist eine genaue Auflistung der erledigten Arbeiten mit den genauen Arbeitsaufwand in Stunden zu finden.

27.2 Meilensteine

27.2.1 Meilenstein 1 (MS1): Analyse

Ergebnisse

- Analyse der Architektur allgemein
- Analyse der einzelnen Teilelemente der Architektur

27.2.2 Meilenstein 2 (MS2): Lokalisierungsservice und Ekahau-Konnektor

Ergebnisse

- Lokalisierungsservice bis und mit Distributed Hash Table
- Konnektorschnittstelle für Pull-Konnektoren
- Ekahau-Konnektor

27.2.3 Meilenstein 3 (MS3): Positionshistorie

Ergebnisse

- Lokalisierungsservice aus MS2 mit Positionshistorie

27.2.4 Meilenstein 4 (MS4): Regeln, CiscoWorks-Konnektor und Smartphone-Konnektor

Ergebnisse

- Kompletter Lokalisierungsservice mit Regeln für Events
- CiscoWorks-Konnektor
- Smartphone-Konnektor

27.2.5 Meilenstein 5 (MS5): Lokalisierungslogik

Ergebnisse

- An Architektur angepasste Lokalisierungslogik (Lösung aus Studienarbeit)
- An Architektur angepasstes ehemaliges shp2sgll

27.2.6 Meilenstein 6 (MS6): Dokumentation und Refactoring

Ergebnisse

- Restliche Elemente
- Refactoring
- Erster Entwurf der Dokumentation

27.2.7 Meilenstein 7 (MS7): Abgabe Bachelor-Arbeit

Ergebnisse

- Vollständige Dokumentation
- Plakat
- CD

27.2.8 Meilenstein 8 (MS8): Präsentation Bachelor-Arbeit

Ergebnisse

- Präsentation der Arbeit

27.3 Reviews

Im Rahmen einer wöchentlichen Sitzung zusammen mit den Betreuern wurden die zum entsprechenden Zeitpunkt wichtigen Themen besprochen (siehe Traktandenlisten und Sitzungsprotokolle in separaten Dokumenten).

Zusammen mit den Ansprechpartnern von Universitätsspital Zürich wurde so weit wie möglich nach den jeweiligen Meilensteinen sowie nach Bedarf ein Review durchgeführt.

27.3.1 Review 1 (RV1): Allgemeines

Themen

- Ziele Bachelor-Arbeit
- Resultate der Analyse

27.3.2 Review 2 (RV2): Architektur

Themen

- Lokalisierungsservice bis und mit Distributed Hash Table

- Konnektoren-Schnittstelle Pull-Konnektoren
- Ekahau-Konnektor
- Projektplan

27.3.3 Review 3 (RV3): Lokalisierungsservice

Themen

- Kompletter Lokalisierungsservice
- CiscoWorks-Konnektor
- Smartphone-Konnektor

27.3.4 Review 4 (RV4): Abgabe und Präsentation Bachelor-Arbeit

Themen

- Abgabe der Dokumentation
- Präsentation Bachelor-Arbeit

27.3.5 Review a (RVa): Präsentation aktueller Stand mit H. Rudin

Themen

- Rückblick Studienarbeit
- Bachelor-Arbeit: Was wurde bereits gemacht
- Bachelor-Arbeit: Was ist noch zu tun

27.4 Burn Down Chart

Abbildung 27.1 (Seite 254) zeigt den Burn Down Chart nach [14], welcher die Team-Geschwindigkeit in der Umsetzung der User Stories repräsentiert. Auf der Ordinatenachse ist der relativ gewichtete Aufwand für die Umsetzung aller User Stories (siehe Master Story List in Abschnitt 10.1, Seite 55, und 17.1, Seite 119) aufgelistet. Die Höhe der einzelnen Balken zeigt die verbleibenden Arbeiten nach Abschluss eines Milestones. Der erste Balken zeigt den Stand beim Abschluss der dieser Arbeit vorausgegangenen Studienarbeit (SA).

Da sich während der Arbeit die Anforderungen angepasst haben, ist am Anfang der Aufwand gestiegen, obwohl einzelne User Stories abgeschlossen werden konnten.

27.5 Auswertung

Abbildung 27.2 (Seite 255) zeigt die Auswertung der mittels mite[1] dokumentierten Aufwände zum Stand vom 12. Juni 2011. Ein aktueller Stand kann der gesamten Auswertung in einem separaten Dokument entnommen werden.

[1] `http://mite.yo.lk/` (abgerufen 13. Juni 2011)

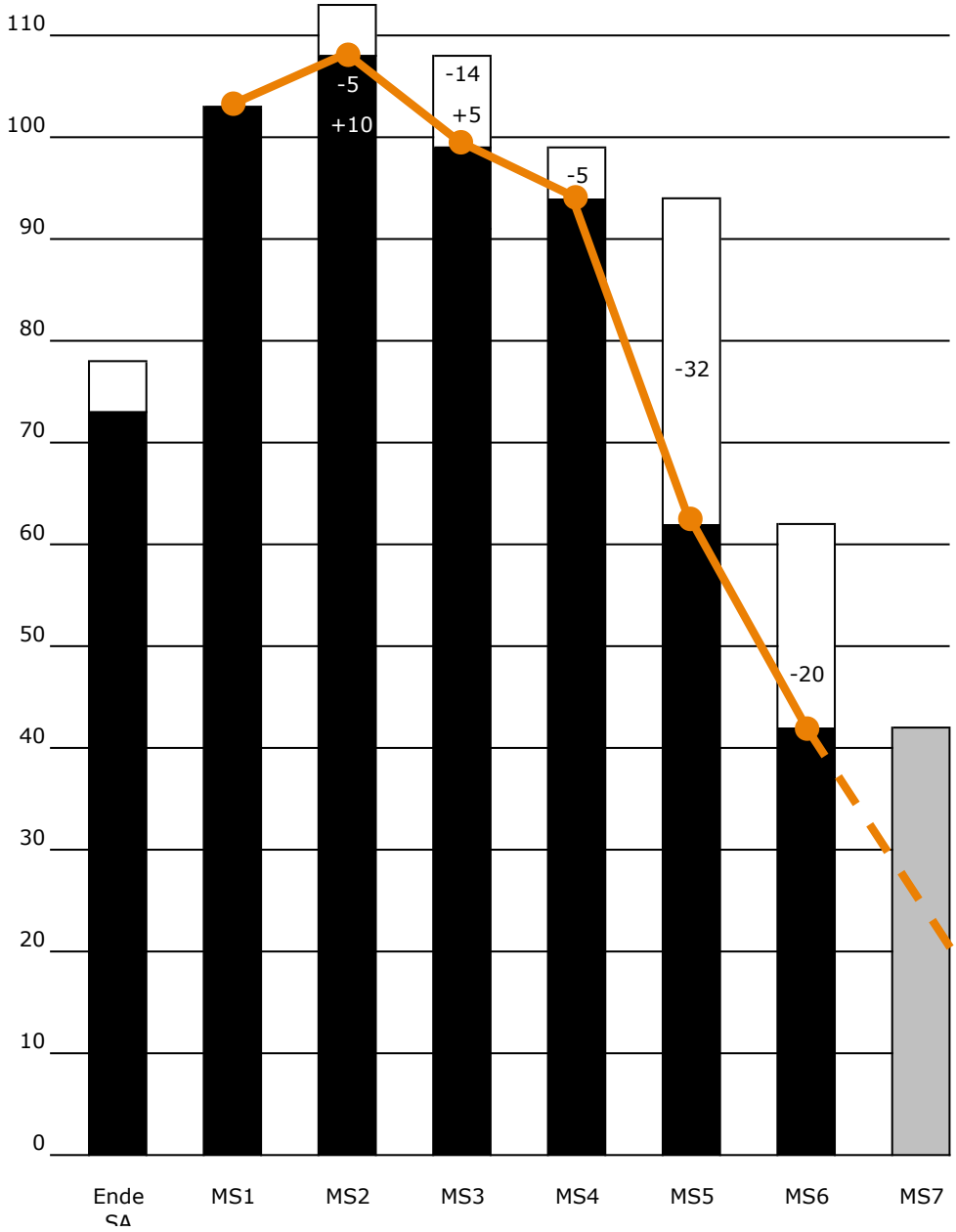

Abbildung 27.1: Der Burn Down Chart zeigt die Team-Geschwindigkeit bei der Umsetzung der User Stories, zum Stand Ende Meilenstein 6.

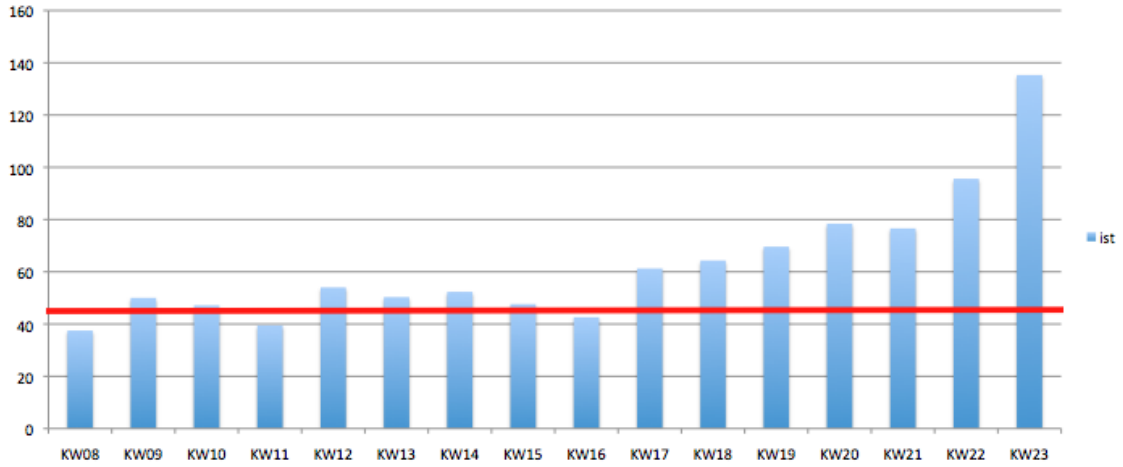

Abbildung 27.2: Auswertung des Aufwandes in Stunden, rot das wöchentliche Soll, blau das Ist.

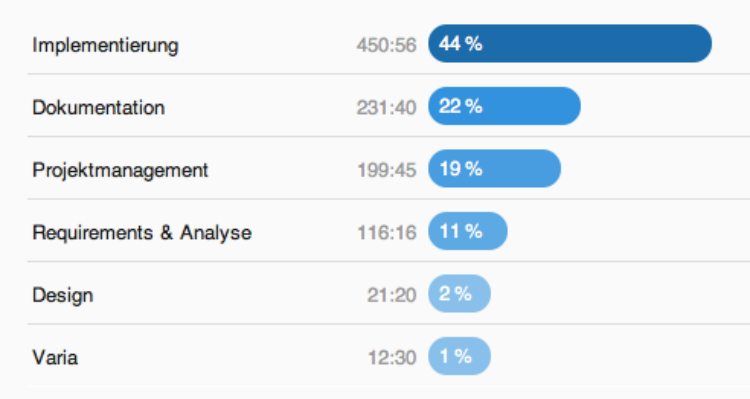

Abbildung 27.3: Auswertung des Aufwandes in den einzelnen Bereichen

Das Soll der Arbeitsstunden sollte sich zum gezeigten Stand auf insgesamt knapp 700 belaufen. Die tatsächliche Stundenanzahl beträgt jedoch 315 Stunden mehr, nämlich 1015 Stunden. Die konstant grosse Erwartungshaltung hat dazu geführt, dass wir uns ohnehin mehr vorgenommen haben als eigentlich vorgesehen war. Technische Probleme haben schlussendlich zu einer weiteren Verschärfung bei der Überzeit beigetragen.

Abbildung 27.3 (Seite 255) zeigt eine Auswertung der Aufwände gemäss Themengebiete. Auffallend ist, dass für das Design verhältnismässig wenig Zeit aufgewendet wurde. Grund dafür ist, dass Design und Implementierung vermischt wurde und die Zeit nur für die Implementierung eingetragen wurde. Ebenfalls fällt auf, dass in das Projektmanagement relativ viel Zeit hineingesteckt wurde. Dies hat wohl damit zu tun, dass neben den Sitzungen mit Betreuer und Co-Betreuer noch Besprechungen mit dem Kunden hinzugekommen sind.

28 Infrastruktur

28.1 Hardware

- Jedes Projektmitglied arbeitet mit dem privaten Computer. Bei Ausfall eines Rechners oder wenn zusätzliche Computer benötigt werden, stehen Ersatzcomputer oder die HSR-Arbeitsplatzrechner im Bachelor-Arbeitszimmer zur Verfügung.
- Ein virtueller Server, der von Hochschule für Technik Rapperswil (HSR) gestellt wird, steht als Build-Server bereit.

28.2 Software

28.2.1 Dokumentation

- Wiki auf GitHub
- Enterprise Architect
- LaTeX
- OmniGraffle

28.2.2 Entwicklungsumgebung

- beliebige Java-IDE (Eclipse, IntelliJ, Netbeans...)
- Firefox
- Maven

28.2.3 Geo-Werkzeuge

- ESRI ArcGIS
- Quantum GIS
- Nemetschek VectorWorks
- OpenOffice.org

28.2.4 Hilfswerkzeuge

- Python
- Ruby

28.2.5 Infrastruktur

- Hudson

- Sonarsource Sonar
- Sonatype Nexus

28.2.6 Laufzeitumgebung

- Apache Tomcat
- JVM
- MongoDB
- PostGIS
- PostgreSQL
- Redis
- nginx

28.2.7 Versionsverwaltung

- Git mit Hosting auf GitHub

28.3 Backup

Regelmässige Spiegelung des VPS mittels rsync.

28.4 Kommunikation

28.4.1 Kommunikationsmittel

- Besprechungen
- E-Mail
- Skype
- Telefon
- Wiki
- Windows Messenger

28.4.2 Datenaustausch

- Dropbox

29 Qualitätssicherung

29.1 Trade-off Sliders

Abbildung 29.1 (Seite 260) zeigt die Trade-off Sliders nach [14]. Der Schalter liegt für jeden Bereich an einem anderen Ort und zeigt so die Flexibilität. Demnach bedeutet ein Schalter, der nahe beim «ON» ist, dass im dazugehörige Bereich keine Flexibilität möglich ist. Ein Schalter auf der anderen Seite beim «OFF» bedeutet, dass eine hohe Flexibilität vorhanden ist in dem Bereich.

Scope Im Projektscope besteht wenig Flexibilität, da die Realisierung von möglichst viel Funktionalität als besonders wichtig taxiert wurde.

Budget Für diese Arbeit ist kein Budget vorhanden.

Time Da der Zeitpunkt der Abgabe fix definiert ist, besteht bezüglich Zeit keinerlei Flexibilität.

Qualität Bezüglich Qualität können unter gewissen Umständen Abstriche gemacht werden.

GUI Optik Da es sich um einen Prototypen handelt, der für einen Pilotversuch verwendet wird, ist die Optik des User Interfaces nebensächlich.

Usability Bei dem Prototyp geht es primär um ein lauffähiges System. Usability steht dabei eher im Hintergrund

Verarbeitungsgeschwindigkeit Die relativ grossen Datenmengen machen eine relativ hohe Verarbeitungsgeschwindigkeit nötig.

Erweiterbarkeit Da bei den verwendeten Technologien noch viel Erweiterungspotenzial liegt (z.B im Bereich Genauigkeit der Lokalisierungstechniken), soll das System einfach an veränderte Umgebungen angepasst oder erweitert werden können.

Dokumentation Da der Prototyp ein Grundgerüst darstellt, worauf weiter aufgebaut wird, soll eine gut nachvollziehbare Dokumentation vorliegen.

29.2 Risiken

Die Risiken des Projekts sind gemäss [14] evaluiert.

- Technologie-Problem
  - «Neue» Technologien, die wenig dokumentiert sind
  - Zusammenspiel der einzelnen Komponenten
  - Modulare Anbindung der Konnektoren
- Projektmanagement

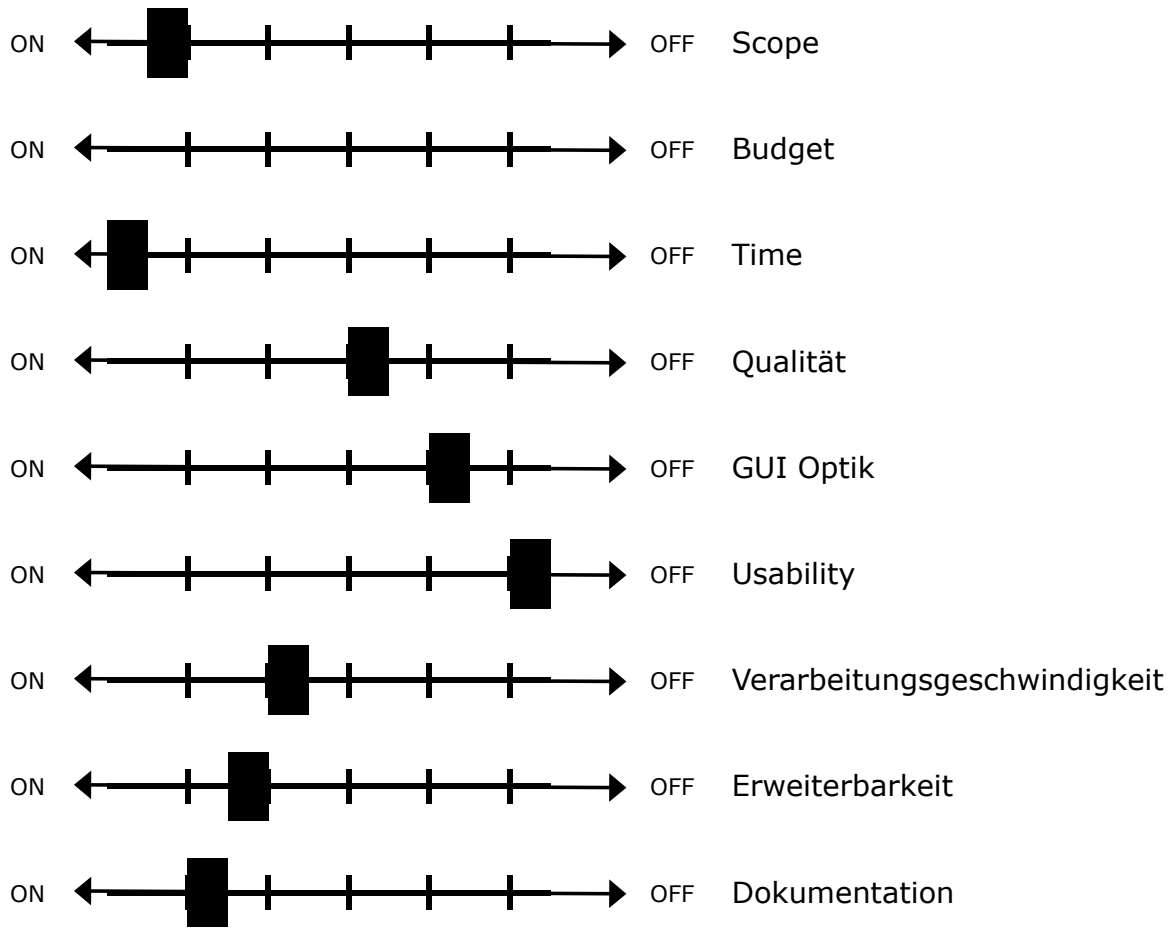

Abbildung 29.1: Die Trade-off Sliders zeigen die Flexibilität des Projektes

  - Backup
  - Verfügbarkeit von Ansprechpersonen
  - Zeitmanagement
- Varia
  - Laufend neue Requirements

29.3 Qualitätsziele

29.3.1 Dokumentation

Dokumente, die zu einem Arbeitspaket gehören, werden laufend nachgeführt und erweitert. Während der Entwicklung wird das Wiki als Plattform für Gedankenstützen genutzt. Die daraus entstandenen Inhalte werden laufend in die Dokumente eingefügt.

Zur Sitzungsvorbereitung werden Traktandenlisten erstellt, welche die für die Sitzung relevanten Themen enthalten. Sie werden jeweils spätestens am Vorabend des Sitzungstermins an die beteiligten Personen versendet. Wichtige Entscheide und Inhalte jeder Sitzung werden in Form von Sitzungsprotokollen festgehalten.

Der grobe Zeitplan, der die Arbeiten pro Tag aufzeigt, wird laufend nachgeführt und gemäss dem iterativen Vorgehen angepasst und erweitert. Die detaillierte Zeiterfassung wird gemäss den verrichteten Arbeiten laufend ausgefüllt.

29.3.2 Unit-Testing

Grundsätzlich wird mittels TDD gearbeitet. Nur das Web User Interface ist davon ausgenommen. Vor einem Check-in der Änderungen wird ein Build-Test durchgeführt, welcher automatisch alle Unit-Tests ausführt.

29.3.3 Statische Analyse

Zur Prüfung der Codequalität wird Sonar[1]. Es werden laufend statische Analysen durchgeführt und falls nötig Massnahmen zur Verbesserung der Qualität des Codes eingeleitet.

Code Coverage

Die Code Coverage soll über das gesamte Projekt mindestens 90 Prozent betragen. Dabei werden die Integrationstests sowie die Microtests berücksichtigt. Aufgrund von Spezialfällen wie zum Beispiel der `EngineException` von Ekahau, die nicht gemockt werden kann, ist das Erreichen einer Code Coverage von 100 Prozent nicht möglich.

Sonar «Rules of compilance»

Für die «Rules of compilance» von Sonar wird eine Übereinstimmung von 90 Prozent angestrebt. Dabei wird das Profil «Sonar with Findbugs» ohne die Regel «Design for Extension»[2] verwendet.

[1] `http://www.sonarsource.org` (abgerufen: 3. Juni 2011)

[2] Weist auf fragile Basis-Klassen hin. Diese können im Projekt aber teilweise nicht stabilisiert werden, ohne ihre Testbarkeit einzuschränken.

29.4 Qualitätssicherung

29.4.1 Dokumentation

Qualität wurde gemäss den Qualitätszielen (Abschnitt 29.3, Seite 261) eingehalten.

29.4.2 Unit-Testing

Der Build Server führte nach jedem Commit die Unit Tests aus und erstellte einen neuen Build. Rund um die Uhr wurden alle 3 Stunden zudem die Integrationstests vom Build Server ausgeführt.

29.4.3 Statische Analyse

Die Codequalität wurde mit Sonar bei jedem Build durch den Build Server überprüft. Durch die laufende Prüfung konnten allfällige Probleme sofort behoben werden. Die geplanten Werte wurden eingehalten.

Teil VII

Anhang

A Glossar

A.1 Begriffe

Tabelle A.1 (Seite 267) listet wichtige domänenrelevante Begriffe und ihre Bedeutung im Rahmen des Projekts. In der Spalte Kontext ist aufgeführt, in welchem Zusammenhang welche Definition des Begriffs verwendet wird.

Begriff	**Kontext**	**Definition und Information**
Ausgangspunkt	–	Position, an der sich die suchende Person befindet
Bezugssystem	–	Eindeutiges Koordinatensystem, auf das für die Angabe der Positionen Bezug genommen wird
Centroid	Lokalisierungslogik	siehe «Schwerpunkt»
Distanz	–	Wegdistanz in Meter, welche zurückgelegt werden muss, um von einem Punkt zum anderen zu kommen
Edge	–	siehe «Kante»
Endpunkt	GIS	Endpunkt eines Linestrings, welcher keine Verbindung zu einem anderen Linestring hat
Gehweg	–	siehe «Weg»
Gerät	–	Geräte, die im Spitalumfeld verwendet werden
Grundriss	–	siehe «Stockwerkplan»
Höhe	–	Angabe der Höhe in Meter über Meer
Karte	Ekahau	Bild eines Stockwerkplans. Besteht aus einer eindeutigen ID und einem Namen. Ist genau einem Modell zugewiesen
Klasse	Gerätelokalisierung	siehe «Objekttyp»
Knoten	Domain	Position in einem Graphen, enthält zusätzlich eindeutige Positionsinformationen
Kreuzungspunkt	GIS	Endpunkt eines Linestrings, welcher weitere Verbindungen zu einem oder mehreren anderen Linestrings hat

Begriff	Kontext	Definition und Information
Laufweg	–	siehe «Weg»
Linie	–	Verbindung zwischen zwei Knoten in einem Graphen, entspricht dem Weg zwischen zwei Punkten
Linienzug	–	siehe «Linie»
Lokalisierungswerkzeug	–	Werkzeug, das auf irgendwelche Art Objekte lokalisieren kann
Map	Ekahau	siehe «Karte»
Metadaten	Lokalisierungslogik	Zur Topologie gehörende Daten wie Raumbezeichnung oder Stockwerkbezeichnung
Model	Ekahau	siehe «Modell»
Modell	Ekahau	Grösster Bereich der Ekahau Lokalisierung. Dieser stellt zum Beispiel ein Gebäude dar. Besteht aus einer eindeutigen ID und einem Namen
Node	Domain	siehe «Knoten»
Objekte	Lokalisierungslogik	Geräte, Personen oder Rollen, die lokalisiert werden können
Objektstatus	–	siehe «Status»
Objekttyp	–	Art des Objekts, beispielsweise «Blutdruckmessgerät» oder «Spritzenpumpe»
Point	Domain	siehe «Punkt»
Polygon	GIS	Geschlossene Linienzüge mit Flächenbezug
Punkt	–	Eindeutig im dreidimensionalen Raum lokalisierbare Position
Raum	–	Einzelnes Zimmer; besteht aus einer eindeutigen Bezeichnung, die sich aus Gebäudebezeichnung, Stockwerk und Nummer zusammensetzt
Referenzsystem	–	siehe «Bezugssystem»
Rolle	Lokalisierungslogik	Person, die einen bestimmten Status hat, beispielsweise Herzchirurg mit Bereitschaftsdienst
Referenzsystem	–	siehe «Bezugssystem»

Begriff	Kontext	Definition und Information
Schwerpunkt	Lokalisierungslogik	Schwerpunkt («Mittelpunkt») einer Geometrie, beispielsweise eines Polygons
Sector	Domain	siehe «Sektor»
Sektor	–	Kleinster unterscheidbarer Bereich in einer Zone, besteht aus einer Fläche, die durch Punkte begrenzt wird, und ist mit einem Knoten assoziiert. Beispiel: Raum 308.
Status	Lokalisierungslogik	Status eines Objekts, beispielsweise «besetzt»
Stockwerkplan	–	Plan, der jeweils ein Stockwerk eines Gebäudes darstellt
Stockwerkübergang	–	Verbindung in Form eines Weges zwischen verschiedenen Stockwerken
Typ	Gerätelokalisierung	siehe «Objekttyp»
Verbindungspunkte	Lokalisierungslogik	Punkte, die als Verbindung zu anderen Stockwerken oder Gebäuden markiert sind
Weg	–	Anhand der Topologie möglicher Weg zwischen Ausgangs- und Zielpunkt
Zielpunkt	–	Position an dem sich das gesuchte Objekt befindet
Zone	Ekahau	Kleinster unterscheidbarer Bereich in einer Map. Besteht aus einer eindeutigen ID und einem Namen
Zone	Lokalisierungslogik	Logischer Bereich innerhalb eines Gebäudes, der aus einem oder mehr Sektoren besteht (stockwerkübergreifend). Beispiel: Notfallaufnahme.

Tabelle A.1: Liste der domänenrelevanten Begriffe

A.2 Formate und Regeln

Tabelle A.2 (Seite 268) listet die domänenrelevanten Formate und Validierungsregeln auf, die im Rahmen des Projekts verwendet werden. Sie ergänzt die Tabelle A.1 (Seite 267), die domänenrelavante Begriffe beziehungsweise ihre projektspezifische Verwendung beschreibt. Die Formate beziehungsweise Validierungsregeln können entweder als Text oder regulären Ausdruck in Perl-Syntax[1] angegeben werden.

Begriff	Kontext	Format	Validierung
Stockwerkübergang Treppe	Topologie	[x-Kooridnate];[y-Koordinate];[z-Koordinate];[z-Koordinate des zu verbindenden Stockwerks]	–
Stockwerkübergang Lift	Topologie	[x-Kooridnate];[y-Koordinate];[z-Koordinate];[z-Koordinate des zu verbindenden Stockwerks]	–
Gebäudeübergang	Topologie	[x-Kooridnate];[y-Koordinate];[z-Koordinate];[z-Koordinate des zu verbindenden Stockwerks]	–
Datum für History	Topologie	YYYY-MM-DDThh:mm:ssTZD nach ISO 8601, z.B. 1997-07-16T19:20:33+01:00	–

Tabelle A.2: Liste der domänenrelevanten Formate und Regeln

A.3 Format-Spezifikation Events

A.3.1 Anwendungsbereich

Dieses Dokument erläutert das Nachrichtenformat, in dem die Presence Engine von anderen Systemen über Ereignisse notifiziert wird. Datei wird als Transport-Medium ein JMS Message Broker verwendet. Die Spezifikationen wurden zusammen mit der Projektgruppe, welche die Bachelor-Arbeit zum «Presence Manager» (Ricardo Alvarez und Reto Brühwiler) realisiert hat, erarbeitet.

[1] `http://perldoc.perl.org/perlre.html` (abgerufen: 15. Juni 2011)

A.3.2 Formatbeschreibung

Übermittlungsformat

Um die Interoperabilität zwischen verschiedenen Plattformen zu gewährleisten, werden Ereignisse als Textnachrichten übermittelt. Als Nutzlast wird JSON (`application/json`) verwendet. Das Dokumentformat wird durch das beiliegende JSON Schema spezifiziert. Einen grober Überblick über die Dokumentstruktur gibt folgendes Listing:

```
{
    "name":"String",
    "priority":"String",
    "timestamp":"String",
    "objectIdentifier":"String",
    "source":"String"
}
```

Feld-Spezifikation

Feld	Typ	Standard	Pflichtfeld	Format, Wertebereich
`name`	String		ja	UTF-8
`objectIdentifier`	String		ja	UTF-8
`priority`	String	`NORMAL`	ja	`{NORMAL, URGENT}`
`source`	String		ja	UTF-8
`timestamp`	String		nein	`[YYYY]-[MM]-[DD]T[hh]:[mm].[fff]Z`

Tabelle A.3: Feld-Spezifikation

Feld-Beschreibung

name Name des Ereignisses, das ausgelöst wurde. Beispiel: `BESETZT`.

objectIdentifier Bezeichner für das Objekt (typischerweise Person), für welches das Ereignis ausgelöst wurde, beispielsweise `hmuster`.

priority Verarbeitungspriorität der Ereignismeldung. Kann zwei Werte annehmen: `NORMAL` und `URGENT` (dringend).

source Name der Ereignisquelle, also des Systems, das das Ereignis generiert hat.

timestamp Datum, zu der das Ereignis erzeugt wurde (nicht zwingend identisch mit Erzeugungszeit der Nachricht), nach ISO 8601 in UTC. Beispiel: `2011-04-21T19:45:00.000Z`. Der Verhalten bei Angabe der Lokalzeit (insbesondere im Bereich der Zeitumstellung von Normalzeit zu Sommerzeit und zurück) ist undefiniert.

```
public class Event {
    private String name;
    private String objectIdentifier;
    private Priority priority;
    private String source;
    private DateTime timestamp;
    public static enum Priority {
        NORMAL, URGENT
    }
}
```

Listing A.1: Klassen-Definition in Java für Event

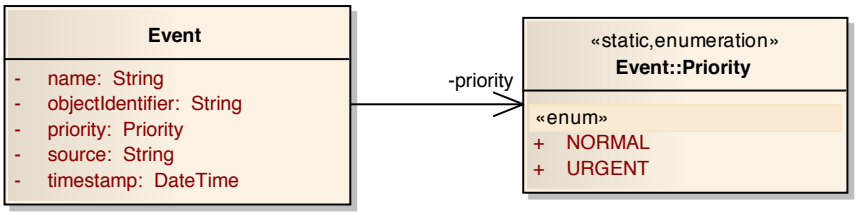

Abbildung A.1: UML-Diagramm einer Event-Nachricht

A.3.3 Kommunikation, Fehlerverhalten

Der Erhalt eines Ereignis-Nachricht wird nicht bestätigt. Tritt bei der Behandlung des Ereignisses ein Fehler auf, wird die Quelle nicht benachrichtigt.

A.3.4 Implementierungshinweise

Abbildung A.1 (Seite 270) zeigt die Objekt-Repräsentation der Ereignis-Nachricht mit Java-Datentypen. Die Priorität wird als Enumeration abgebildet, die statisch in das `Event`-Objekt eingebettet wurde. Die korrespondierende Klassen-Definition in Java ist in Listing A.1 (Seite 270) dargestellt.

A.4 Format-Spezifikation Smartphone-App

A.4.1 Anwendungsbereich

Dieses Kapitel erläutert das Nachrichtenformat, in dem Smartphone-Apps Positionsmeldungen an den Smartphone-Konnektor (Server-Komponente) übermitteln.

A.4.2 Transport

Protokoll

Als Transportprotokoll wird HTTP 1.1 verwendet, wobei für jede Meldung eine neue Verbindung aufgebaut wird. Nachrichten werden mittels `POST` ausgetauscht.

```
{
    "sensorId":"String",
    "position":{
        "x":"Double",
        "y":"Double",
        "z":"Double",
    },
    "accuracty":"Double",
    "date":"String"
}
```

Listing A.2: JSON-Schema für erfolgreiche Positionsbestimmung

Der Smartphone-Konnektor ist ein HTTP-1.1-konformer Webserver, d.h. den Smartphone-Apps steht es frei, welche Funktionen von HTTP 1.1 sie nutzen oder nicht.

Empfehlungen

- Verwendung von HTTP-Kompression (z.B. `Content-Encoding: gzip, deflate`)

Sicherheit

Die Kommunikation mit dem Smartphone-Konnektor darf ausschliesslich TLS-gesichert erfolgen. Kann das Zertifikat des Smartphone-Konnektors nicht über die im Gerät hinterlegten Zertifikate validiert werden, darf keine Verbindung aufgebaut werden. D.h. der Benutzer darf auch nicht gefragt werden, ob er ein nicht validierbares Zertifikat manuell akzeptieren möchte.

Optional wird zertifikatsbasierte Authentifizierung über ein im Zertifikatsspeicher des Geräts hinterlegtes Zertifikat unterstützt. HTTP-Authentifizierung (`BASIC`, `DIGEST`) wird nicht unterstützt.

A.4.3 Formatbeschreibung

Übermittlungsformat

Um die Interoperabilität zwischen verschiedenen Plattformen zu gewährleisten, werden Ereignisse als Textnachrichten übermittelt. Als Nutzlast wird JSON (`application/json`) verwendet. Die Dokumentenformate werden durch die beiliegenden JSON-Schemata spezifiziert. Die nachfolgenden Abschnitte geben einen groben Überblick über die Datenstrukturen.

Erfolgreiche Positionsbestimmung Wird an `/report/successful` übermittelt, wenn die Position des Smartphones erfolgreich bestimmt werden konnte (Listing A.2, Seite 271).

Position ausserhalb des Erfassungsbereichs Wird an `/report/not-on-campus` übermittelt, wenn die Position sich ausserhalb des Erfassungsbereichs befindet (Listing A.3, Seite 272).

```
{
    "sensorId":"String",
    "date":"String"
}
```

Listing A.3: JSON-Schema für Position ausserhalb des Erfassungsbereichs

```
{
    "sensorId":"String",
    "date":"String"
}
```

Listing A.4: JSON-Schema für fehlgeschlagene Positionsbestimmung

Fehlgeschlagene Positionsbestimmung Wird an `/report/failed` übermittelt, wenn die Position nicht erfolgreich bestimmt werden konnte.

Feld-Spezifikation

Feld	Typ	Standard-Wert	Pflichtfeld	Format, Wertebereich
`sensorId`	String		ja	UTF-8
`position.X`	Double		nein	WGS84
`position.Y`	Double		nein	WGS84
`position.Z`	Double		nein	WGS84
`accuracy`	Double		nein	in Metern
`date`	String		nein	`[YYYY]-[MM]-[DD]T[hh]:[mm].[fff]Z`

Tabelle A.4: Feld-Spezifikation

Feld-Beschreibung

sensorId ID des Sensors, also des Smartphones.

position.X Breitengrad nach WGS84, z.B. `47.226665`

position.Y Längengrad nach WGS84, z.B. `8.818334`

position.Z Ellipsoidhöhe, z.B. `5.68`

accuracy Ungenaugikeit der Positionsbestimmung mit der möglichen Abweichung in Metern (positiv und negativ aufgefasst), z.B. `1.0` für eine Abweichung von ± 1.0 m.

date Datum, an dem die Position erfasst wurde, nach ISO 8601 in UTC. Beispiel: `2011-04-21T19:45:00.000Z`. Der Verhalten bei Angabe der Lokalzeit (insbesondere im Bereich der Zeitumstellung von Normalzeit zu Sommerzeit und zurück) ist undefiniert.

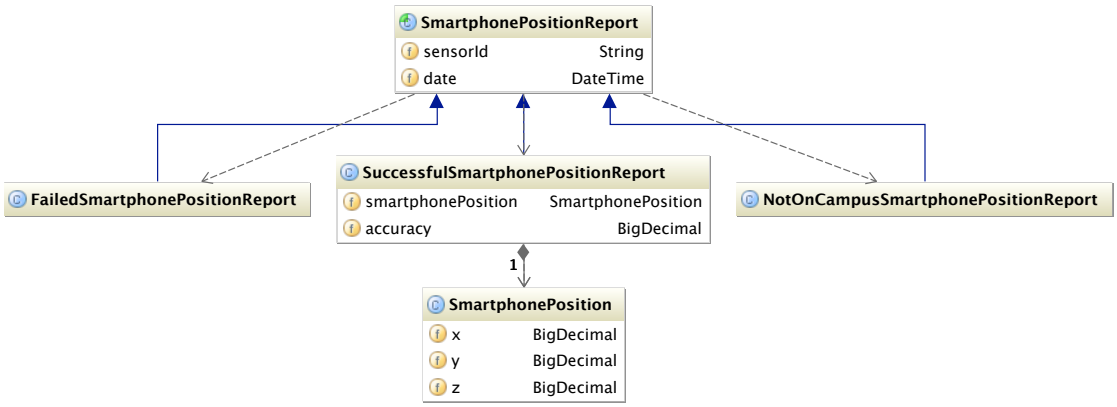

Abbildung A.2: UML-Diagramm der Report-Hierarchie

A.4.4 Kommunikation, Fehlerverhalten

Der Erhalt eines Positionsnachricht wird bestätigt (HTTP-Status-Code `201 Created`). Tritt bei der Behandlung ein Fehler auf, wird die Quelle mit dem entsprechenden Status-Code benachrichtigt (HTTP-Status-Code `40x` oder `50x`). Die Übermittlung der Positionsnachricht soll nicht wiederholt werden.

A.4.5 Implementierungshinweise

Abbildung A.2 (Seite 273) zeigt die Objekt-Repräsentation der Positionsmeldungsnachricht mit Java-Datentypen.

B Beiträge von Dritten

B.1 Transformation von Koordinaten

Im Zusammenhang mit der Umwandlung der von der Ekahau Positioning Engine gelieferten Positionskoordinaten in die verwendeten Koordinaten wurde Unterstützung von Drittpersonen eingeholt.

B.1.1 Umwandlung mit Rotation, Verschiebung und Skalierung

Um von der erarbeiteten Grundidee auf eine konkrete Lösung zu kommen, wurde Hilfe vom HSR-Dozenten Andreas Müller eingeholt. Er konnte auf den grundlegenden Gedankengängen aufbauende Inputs geben bezüglich der Umsetzung mittels Rotation, Verschiebung und Skalierung.

B.1.2 Umwandlung mit einer Transformationsmatirx

Aufgrund der in der Umsetzung komplexen Variante mittels Rotation, Verschibung und Skalierung wurde Unterstützung vom HSR-Dozenten Olaf Tietje eingeholt. Zusammen konnte einen Lösungsansatz zur Transformation erarbeitet werden, welcher nicht nur einfacher, sondern programmiertechnisch mit verhältnismässig kleinem Aufwand umgesetzt werden konnte.

B.2 Dynamisches Laden der Konnektoren

Zur Umsetzung des Lademechanismus der Konnektoren, wurde Unterstützung bezüglich des Spring Frameworks geholt. Der ehemalige HSR-Dozent Wolfgang Giersche hat die erarbeiteten Lösungsansätze in Frage gestellt und Inputs geliefert, welche Ansätze dem Problem entsprechend besser umsetzbar sind.

C Anleitungen

C.1 Installationsanleitung

Die Installationsanleitung für das Lokalisierungssystem ist zusammen mit einer Demo-Installation (Virtuelle Maschine im VMware-Format) auf der beiliegenden DVD zu finden.

C.2 Konnektor-Entwicklung

Dieser Abschnitt beschreibt die ersten Schritte zur Erstellung eines neuen Konnektors. Sie bestehen aus folgenden Punkten:

1. Anlegen eines neuen Konnektor-Moduls
2. Contract einbinden
3. Eigentlichen Konnektor erstellen
4. Konnektor konfigurierbar machen

C.2.1 Modul erzeugen

Um einen neuen Konnektor zu erstellen, muss zuerst ein Modul für den Konnektor erzeugt werden. Dazu muss man auf die Kommando-Zeile und ins Verzeichnis `eagleeye-connectors` (relativ zum Wurzelverzeichnis des Projekts) wechseln:

```
$ ls
connector-common     pom.xml
```

Nun kann mit Hilfe von Maven ein neues Modul erzeugt werden:

```
$ mvn archetype:create -DgroupId=ch.hsr.ins.eagleeye.connector \
    -DartifactId=konnektor-name
```

Für `konnektor-name` muss der Name des neu hinzuzufügenden Moduls eingesetzt werden. Soll zum Beispiel ein Ekahau-Konnektor für die in der Schweiz aktuell verwendeten Landeskoordinaten entwickelt werden, der `ekahau-21781` genannt werden soll, würde der Befehl folgendermassen aussehen:

```
$ mvn archetype:create -DgroupId=ch.hsr.ins.eagleeye.connector \
    -DartifactId=ekahau-21781
```

Für das neue Modul wird dann ein Verzeichnis namens `ekahau-21781` angelegt:

```
$ ls
connector-common    ekahau-21781        pom.xml
```

Zusätzlich wird das Modul in die `pom.xml` vom Eltern-Projekt eingetragen:

```
$ grep -b2 -a2 ekahau-21781 pom.xml
600-  <modules>
612-    <module>connector-common</module>
650:    <module>ekahau-21781</module>
685-  </modules>
698-</project>
```

Analog muss das Eltern-Projekt in der `pom.xml` des Moduls eingetragen werden:

```
$ grep -a3 -b3 connector ekahau-21781/pom.xml
172-    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
231-  <modelVersion>4.0.0</modelVersion>
268-  <parent>
279:    <artifactId>connector</artifactId>
318-    <groupId>ch.hsr.ins.eagleeye</groupId>
361-    <version>1.0-SNAPSHOT</version>
397-  </parent>
```

Die Konvention für die Benennung der Konnektoren lautet `<werkzeug>-<epsg-code>`. Der EPSG-Code[1] bezeichnet das verwendete Koordinatensystem. Ein Ekahau-Konnektor für die Schweizer Landeskoordinaten (CH1903 LV03) würde demnach `ekahau-21781` heissen. Ein Cisco-Works-Konnektor für West-Deutschland mit Gauss-Krüger-Koordinaten `ciscoworks-31467`.

C.2.2 Contract einbinden

Jeder Konnektor hängt vom bereits vorhandenen Modul `connector-common` ab. Es enthält unter anderem die zu implementierenden Interfaces. Um `connector-common` in das Modul einzubinden, muss es als Dependency in die `pom.xml` (z.B. `ekahau-21781/pom.xml`) des Moduls eingetragen werden, wie Listing C.2.2 (Seite 279) zeigt. Eventuell muss dabei noch die Versionsangabe angepasst werden.

[1] `http://www.epsg-registry.org/` (abgerufen: 14. Juni 2011)

```
<project>
<!-- ... -->
    <dependencies>
        <!-- ... -->
        <dependency>
            <groupId>ch.hsr.ins.eagleeye</groupId>
            <artifactId>connector-common</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <!-- ... -->
    </dependencies>
    <!-- ... -->
</project>
```

C.2.3 Konnektor erstellen

Ein Konnektor ist grundsätzlich spezifisch für ein bestimmtes Bezugssystem, weil je nach Bezugssystem andere Referenzdaten und Formeln zur Umrechnung benötigt werden. Das heisst, dass für ein Lokalisierungswerkzeug pro Bezugssystem ein separater Konnektor erstellt werden sollte. Sollte ein Lokalisierungswerkzeug sowohl über einen Push- als auch einen Pull-Konnektor angebunden werden, kann dies dagegen problemlos über ein einziges Modul realisiert werden.

Konnektor-Beschreibung

Jeder Konnektor muss in seinen XML Bean Definitions ein Objekt des Typs `ConnectorInformation` hinterlegen, das den Konnektor beschreibt. Es dient vor allem dazu, dem Lokalisierungsservice mitzuteilen, für die Lokalisierung welcher Art von Sensor der Konnektor geeignet ist. Ein Beispiel aus dem Smartphone-Konnektor:

```
<bean id="connectorInformation"
      class="ch.hsr.ins.eagleeye.connector.common.ConnectorInformation">
    <constructor-arg name="forSensorType" value="Smartphone"/>
</bean>
```

Beim CiscoWorks-Konnektor müsste statt `Smartphone` der Wert `CiscoWorks` eingesetzt werden.

Pull-Konnektor

Ein Pull-Konnektor zeichnet sich dadurch aus, dass er auf Anregung des Lokalisierungssystems hin einen bestimmten Sensor lokalisiert. Dazu nimmt er Kontakt mit dem Lokalisierungswerkzeug auf, fragt es ab, verarbeitet das Resultat und gibt es an das Lokalisierungssystem zurück. Ein Beispiel für einen Pull-Konnektor ist der realisierte Konnektor für die CiscoWorks LAN Management Solution. Die Erstellung eines Pull-Konnektors besteht aus folgenden Schritten:

1. Definition und Erstellung einer Klasse, die vom Lokalisierungsservice aufgerufen wird und die Positionsbestimmung iniitiert.

2. Erstellung allfällig weiterer nötiger Klassen zur Positionsbestimmung der Sensors.

Für die Klasse, die vom Lokalisierungsservice aufgerufen wird, besteht keine Namenskonvention. Sie muss einzig das Interface `PullConnector` implementieren.

Push-Konnektor

Ein Push-Konnektor zeichnet sich dadurch aus, dass er eine Positionsnachricht von einem Lokalisierungswerkzeug entgegen nimmt, verarbeitet und das Resultat dieser Verarbeitung dann an das System weitergibt, womit seine Aufgabe erledigt wäre. Ein Beispiel für einen Push-Konnektor ist der realisierte Smartphone-Konnektor. Die Erstellung eines Push-Konnektors besteht aus folgenden Schritten:

1. Definition und Erstellung einer Klasse zum Entgegennehmen der Positionsnachricht (typischerweise ein Servlet oder JMS Message Listener)

2. Definition und Erstellung eines Delegates, das die Daten vom Servlet oder JMS Message Listener entgegen nimmt und weiterverarbeitet.

3. Erstellung allfällig weiterer nötiger Klassen zur Verarbeitung der Positionsnachricht.

Für die Klasse zum Entgegennehmen der Positionsnachricht besteht keine Namenskonvention. Sie muss einzig das Marker-Interface `PushConnector` implementieren beziehungsweise `JaxRsPushConnector`, wenn es sich um einen JAX-RS Endpunkt handelt.

Die Gestaltung der Klassen zur Verarbeitung der Positionsnachricht ist völlig frei. Die nötige Referenz auf das System zur Weitergabe der Positionsnachricht an das System kann über das Marker-Interface `RequiresService<T>` bezogen werden, wobei für `T` das Interface `PushResultHandler.class` angegeben werden muss.

Handelt es sich um einen Push-Konnektor mit JAX-RS Endpunkt, der ausserhalb des Konnektor-Kontexts lebt, muss auf den `JaxRsPushConnector` zusätzlich noch das Interface `RequiresDelegate<T>` angebracht werden, wobei für `T` die Klasse des Delegates im Konnektor eingesetzt werden muss.

Konnektor-Verdrahtung

Jeder Konnektor muss per Konvention über einen `ApplicationContext` verfügen, der mittels XML Bean Definitions in einer Datei `connector-context.xml` beschrieben wird (abzulegen in `src/main/resources`). Ansonsten kann der Konnektor nicht geladen und gefunden werden.

Grundsätzlich müssen alle Beans, die vom Lokalisierungsservice bezogen werden können, in der `connector-context.xml` beschrieben werden. Dies gilt insbesondere für:

- Implementierungen von `PullConnector`

- das `T` aus `RequiresDelegate<T>`

Explizit *nicht definiert* werden dürfen dagegen Implementierungen von `PushConnector`. Der Name der Beans spielt dabei allerdings keine Rolle, da sie nach Typ aus dem Kontext herausgefischt werden.

C.2.4 Konnektor konfigurierbar machen

Jeder Konnektor sollte installationsspezifische Einstellungen wie Datenbankpasswörter oder Verbindungs-URLs über Properties-Dateien konfigurierbar machen, um eine einfache Inbetriebnahme zu gewährleisten. Das empfohlene Vorgehen ist die Verwendung eines `PropertyPlaceholderConfigurer`, mit dem die in den Properties-Dateien definierten Werte in die XML Bean Definitions eingesetzt werden können. Einen Konnektor konfigurierbar zu machen, besteht aus drei Schritten:

1. Zu konfigurierende Einstellungen auswählen und durch Platzhalter ersetzen.
2. Für jeden Platzhalter einen sinnvollen Default-Wert in einer in `src/main/resources` abgelegten Datei angeben.
3. `PropertyPlaceholderConfigurer` vorbereiten, der zuerst die Default-Werte einliest und, sofern vorhanden, die Werte mit einer alternativen, externen Konfiguration überschreibt.

Beispiel für die Definition eines `PropertyPlaceholderConfigurer`, wie er im Rahmen des CiscoWorks-Konnektors verwendet wird:

```
1  <bean class="org.springframework...PropertyPlaceholderConfigurer">
2      <property name="locations">
3          <list>
4              <value>classpath:ciscoworks21781.properties</value>
5              <value>#{systemEnvironment.EAGLEEYE_HOME}/conf/
6                  connector/ciscoworks21781.properties</value>
7          </list>
8      </property>
9      <property name="ignoreResourceNotFound" value="true"/>
10 </bean>
```

Bei dieser Konfiguration werden zuerst die Default-Werte aus der Datei `ciscoworks21781.properties` augelesen. Ist eine alternative Konfiguration relativ zur Umgebungsvariable `EAGLEEYE_HOME` in `conf/connector/ciscoworks21781.properties` vorhanden, werden die Default-Werte mit dieser Konfiguration überschrieben.

D Persönliche Berichte

D.1 Franziska Felix

Da es sich um eine Folgearbeit handelte, dachte ich, der Einstig würde sich einfacher gestalten, als es in der Studienarbeit der Fall war. Da jedoch viele neue Anforderungen an uns und das Projekt gestellt wurden, war dies nur begrenzt der Fall.

Es stellte sich schnell heraus, dass die während der Studienarbeit noch relativ überschaubare Architektur für die neuen Anforderungen nicht geschaffen ist und deshalb nochmals überdacht werden musste. Mit der neue Architektur, durch welche sehr interessante Aspekte behandelt wurden, konnte ich viele Kenntnisse aufbauen und erweitern. Zusätzlich motivierend war dabei, dass es sich dabei nicht um ein «Schubladen-Projekt» handelt, welches nach Abschluss der Arbeit nicht mehr angerührt wird.

Aufgrund der hohen Erwartungshaltung war leider nicht viel Spielraum für Probeläufe oder auch Fehlschläge, welcher mit den vielen unbekannten Technologien und Themengebieten manchmal wünschenswert gewesen wäre. Ein Projekt von dieser Komplexität und Umfang wäre wohl zum Scheitern verurteilt gewesen ohne die grossartige Teamarbeit mit Andreas Ahlenstorf.

D.2 Andreas Ahlenstorf

Da es sich bei der Bachelor-Arbeit um eine Fortsetzung unserer Studienarbeit handelt, dachte ich vor Beginn noch, dass ich wüsste, was auf uns zukommt. Sie war vom Charakter her aber eine komplett andere Aufgabe. Nachdem bei der Studienarbeit noch Geometrie und Topologie im Fokus standen, ging es nun primär um Software-Architektur und die Realisierung zusätzlicher Use Cases. Besonders die Arbeit an der Software-Architektur hat mir grossen Spass gemacht, da sie sehr abwechslungsreich war und es viele neue Werkzeuge kennenzulernen gab. Für das nötige Experimentieren, Fehlschläge und Neuversuche gab es aber kaum Zeit, was zu entsprechend viel Überzeit geführt hat. Der Projektverlauf erinnerte manchmal an die Fahrt in einer Achterbahn und verlangte sehr viel von mir ab.

Nach zwei Semestern Beschäftigung mit der Lokalisierung von medizinischen Geräten im Spitalumfeld kann ich rückblickend sagen, dass es eine sehr intensive, lehr- und abwechslungsreiche, aber auch anstrengende Reise war. Sie war geprägt von sehr hoch gestellten Erwartungen und Zielen, technischen Misserfolgen und Durchbrüchen, Entmutigung und Ermutigung und ganz besonders von der besten Teamarbeit, die ich je erlebt habe. Danke Franziska! Das Resultat kann sich meiner Meinung nach aber mehr als sehen lassen und ich habe persönlich das Gefühl, dass wir unseren Nachfolgern ein Projekt hinterlassen, auf dem man gut aufbauen kann.

Abkürzungsverzeichnis

AJAX Asynchronous JavaScript and XML
AOP Aspektorientierte Programmierung
API Application Programming Interface
BASE Basically Available, Soft state, Eventual consistency
BSON Binary JavaScript Object Notation
CAD Computer-Aided Design
CAP Consistency, Availability and Partition Tolerance
DAO Data Access Object
DOM Document Object Model
DSL Domain Specific Language
DTO Data Transfer Object
DXF Drawing eXchange Format
EJB Enterprise Java Beans
ESRI Environmental Systems Research Institute Inc.
GDAL Geospatial Data Abstraction Library
GIS Geoinformationssystem
GUI Graphical User Interface
HSR Hochschule für Technik Rapperswil
HTTP Hyper Text Transfer Protocol
JAR Java Archive
JAX-RS Java API for RESTful Web Services
JAXB Java Architecture for XML Binding
JDBC Java Database Connectivity
JDK Java Development Kit
JMS Java Message Service
JPA Java Persistence API
JSON JavaScript Object Notation
JSR Java Specification Request
JVM Java Virtual Machine
LAN Local Area Network
MAC Media Access Control
OGR OGR Simple Feature Library
OOP Objektorinetierte Programmierung
ORM Object-Relational Mapping
OSGi Open Services Gateway initiative
OSI Open Source Initiative
RDBMS Relational Database Management System
REST Representational State Transfer

RFID Radio Frequency Identification
RIA Rich Internet Application
RMI Remote Method Invocation
RPC Remote Procedure Call
SDK Software Development Kit
SHP Shapefile
SLF4J Simple Logging Façade For Java
SOAP Simple Object Access Protocol
SOP Same Origin Policy
SQL Structured Query Language
TLS Transport Layer Security
URL Uniform Resource Locator
USZ Universitätsspital Zürich
WAR Web Archive
WGS84 World Geodetic System 1984
WKT Well Known Text
WLAN Wireless LAN
WPF Windows Presentation Foundation
XML Extended Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Ahlenstorf, Andreas ; Felix, Franziska: *Lokalisierung medizinischer Geräte im Spitalumfeld*. 2010

[2] Bauer, Christian ; King, Gavin: *Java Persistence with Hibernate*. Manning Publications, 2006. – ISBN 978–1–93239488–5

[3] Bloch, Joshua: *Effective Java, Second Edition*. Prentice Hall, 2008. – ISBN 978–0–321–35668–0

[4] Bundesamt für Landestopografie swisstopo: *Formeln und Konstanten für die Berechnung der Schweizerischen schiefachsigen Zylinderprojektion und der Transformation zwischen Koordinatensystemen*. `http://www.swisstopo.admin.ch/internet/swisstopo/de/home/topics/survey/sys/refsys/switzerland.parsysrelated1.24280.downloadList.32633.DownloadFile.tmp/refsysd.pdf` (abgerufen: 17. März 2011), 2008

[5] Cisco Systems, Inc.: *Open Database Schema Support in CiscoWorks LAN Management Solution 4.0*. `http://www.cisco.com/en/US/docs/net_mgmt/ciscoworks_lan_management_solution/4.0/database_schema4.0/guide/dbviews.html` (abgerufen: 5. Juni 2011), 2010

[6] Cooper, Brian F. ; Silberstein, Adam ; Tam, Erwin ; Ramakrishnan, Raghu ; Sears, Russell: Benchmarking cloud serving systems with YCSB. In: *Proceedings of the 1st ACM symposium on Cloud computing*, ACM, 2010. – ISBN 978–1–4503–0036–0, S. 143–154

[7] Dean, Jeffrey ; Ghemawat, Sanjay: MapReduce: simplified data processing on large clusters. In: *Commun. ACM* 51 (2008), January, S. 107–113

[8] Edlich, Stefan ; Friedland, Achim ; Hampe, Jens ; Brauer, Benjamin: *NoSQL*. Hanser Fachbuchverlag, 2010. – ISBN 978–3–446–42355–8

[9] Forgy, Charles L.: Rete: A fast algorithm for the many pattern/many object pattern match problem. In: *Artificial Intelligence* 19 (1982), Nr. 1, S. 17 – 37

[10] Gilbert, Seth ; Lynch, Nancy: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. In: *SIGACT News* 33 (2002), June, S. 51–59

[11] Goetz, Brian: *Java Concurrency in Practice*. Addison-Wesley, 2006. – ISBN 978–0–321–34960–1

[12] Kim, Hyeyoung ; Jun, Chulmin ; Yi, Hyunjin: A SDBMS-Based 2D-3D Hybrid Model for Indoor Routing. In: *MDM '09: Proceedings of the 2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*, IEEE Computer Society, 2009. – ISBN 978–0–7695–3650–7, S. 726–730

[13] Larman, Craig: *Applying UML and Patterns*. Prentice Hall, 2004. – ISBN 978–0–131–48906–6

[14] Rasmusson, Jonathan: *The Agile Samurai*. The Pragmatic Programmers, 2010. – ISBN 978–1–93435–658–6

[15] Rossum, Guido van: *The depth and breadth of Python*. `http://neopythonic.blogspot.com/2011/06/depth-and-breadth-of-python.html` (abgerufen: 9. Juni 2011), 2011

[16] Rudolph, George: *Some Guidelines For Deciding Whether To Use A Rules Engine*. `http://www.jessrules.com/jess/guidelines.shtml` (abgerufen: 17. März 2011), 2008

[17] W3C: *Same Origin Policy*. `http://www.w3.org/Security/wiki/index.php?title=Same_Origin_Policy&oldid=265` (abgerufen: 15. Juni 2011), 2010