Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Kjell Otto

Aktuelle Entwicklungskonzepte zur Projektintegration in einem Smart Home anhand von Maven, OSGi und Drools Fusion

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Kjell Otto

Aktuelle Entwicklungskonzepte zur Projektintegration in einem Smart Home anhand von Maven, OSGi und Drools Fusion

Masterarbeit eingereicht im Rahmen der Masterprüfung

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

Betreuender Prüfer: Prof. Dr. Gunter Klemke
Zweitgutachter: Prof. Dr. Kai von Luck

Eingereicht am: 24.04.2013

Kjell Otto

Thema der Arbeit
Aktuelle Entwicklungskonzepte zur Projektintegration in einem Smart Home anhand von Maven, OSGi und Drools Fusion

Stichworte
Maven, OSGi, Drools Fusion, Java, Software Design, Softwarearchitektur, Living Place Hamburg, Smart Home, Ambient Assisted Living, Ubiquitous Computing, Context Aware Computing

Kurzzusammenfassung
In einem Labor zur Forschung im Bereich der intelligenten Wohnumgebung können viele Konzepte und Ideen zur Gestaltung zukünftigen Wohnens entstehen. Die Anforderungen an die Infrastruktur dieses Labors sind komplex und verändern sich mit Studenten und Projekten ständig. In diesem Kontext konnte die vorliegende Arbeit auf verschiedenen Ebenen Konzepte und Lösungen vorstellen, die grundlegende Strukturen ergänzt und Projekte in den Bereichen der Verhaltensänderung, Sensorik und Aktorik vereinfacht. Darüber hinaus wurde ein System implementiert, mit dem nun die Möglichkeit besteht die Wohnumgebung mit regelbasierten Verfahren um Verhaltensweisen zu erweitern.

Kjell Otto

Title of the paper
Actual development concepts for project integration in a smart home on the basis of Maven, OSGi and Drools Fusion

Keywords
Maven, OSGi, Drools Fusion, Java, Software Design, Software Architecture, Living Place Hamburg, Smart Home, Ambient Assisted Living, Ubiquitous Computing, Context Aware Computing

Abstract
Many concepts and ideas for future living can come up in a laboratory for smart home research. The requirements for the infrastructure in such a laboratory are complex and change with every student and project. In this context the work in hand has delivered concepts and solutions on different levels, extending the existing structures and easing the development of behavioural changes, sensors and actors. Beyond this, a system was implemented with which it is now possible to exdend the existing living rooms behaviors with rule based extensions.

Inhaltsverzeichnis

1 Einleitung

In der heutigen Informatik zeichnet sich immer deutlicher ab, dass Computer kleiner, leistungsfähiger und kostengünstiger werden. Vom Standrechner über den Laptop bis hin zum Smartphone finden Computer immer häufiger den Weg in das Leben der Menschen und halten allgegenwärtig Einzug in den Alltag. Bereits Anfang der neunziger Jahre hat Mark Weiser in seinem Paper über Probleme mit allgegenwärtigen Computern geschrieben:

> The goal is to achieve the most effective kind of technology, that which is essentially invisible to the user. To bring computers to this point while retaining their power will require radically new kinds of computers of all sizes and shapes to be available to each person. I call this future world „Ubiquitous Computing“ (Ubicomp).
> (Weiser, 1993, S. 75)

Die von Weiser beschriebenen Computer sind mittlerweile vorhanden und Informationstechnologien ermöglichen es heute Computer in allen Bereichen einzusetzen. Die allgegenwärtigen Computer ermöglichen eine voll vernetzte Informationsstruktur in allen Lebenslagen der Menschen. Auch in private Wohnräume werden zukünftig immer häufiger Informationssysteme Einzug finden und dem Menschen Unterstützung bieten. An der Hochschule für Angewandte Wissenschaften Hamburg (HAW) Hamburg existiert seit eineinhalb Jahren ein Labor das sich damit beschäftigt, wie diese Informationssysteme miteinander und mit dem Menschen agieren, der Living Place Hamburg. In diesem Labor wird eine natürliche Wohnumgebung mit Sensorik und Aktorik ausgestattet, um die Anwendbarkeit von Konzepten, Ideen und Technologien in zukünftigen Lebensräumen zu untersuchen.

1.1 Motivation

Der Living Place Hamburg wurde vor eineinhalb Jahren eröffnet[1] und steht Studenten als Forschungslabor zur Verfügung. Das Labor ist in drei Bereiche unterteilt: Entwicklungsräume, Wohnraum und Kontrollraum, vgl. Abb. 1.1. Die Entwicklungsräume(*Development area*) enthalten zum einen Büroflächen in denen die Software entwickelt wird, die im Labor zum Einsatz kommt, zum anderen aber auch eine Werkstatt, in der Hardware und Platinen entwickelt werden können. Der Wohnbereich(*Living area*) enthält die Sensorik und Aktorik des Living Place. Darunter befinden sich unter anderem eine Lichtsteuerung, bewegliche Kameras und ein Indoor Positioning System. Der Kontrollraum(*Control room*) zeichnet sich durch Bildschirme aus, die es ermöglichen, von außen den Wohnraum zu überwachen und Nutzbarkeitsstudien durchzuführen, ohne den Bewohner zu beeinflussen. Im Kontrollraum wird die Infrastruktur des Living Place überwacht und eventuell manuell in das Laborgeschehen eingegriffen.

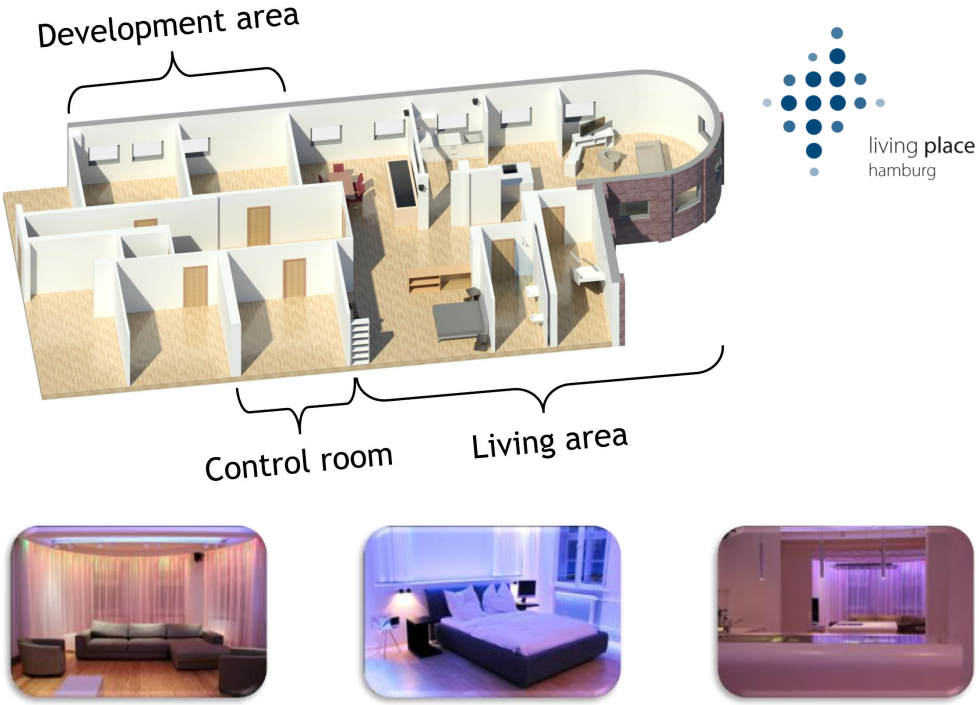

Abbildung 1.1: Living Place Hamburg Laborstruktur, Quelle: (Voskuhl, 2011, S. 53)

[1] Living Place Eröffnung - Hello World! - http://livingplace.informatik.haw-hamburg.de/blog/?p=439

Die Projekte, die im Living Place Hamburg entstehen, werden von Studenten unter Anleitung der Professoren von Luck, Klemke und Wendholt entwickelt und in das Labor integriert. Seit dem Projektstart sind 28 Projekte in die Praxis umgesetzt worden und viele der Projekte sind in das Labor integriert. Die gemeinsame Infrastruktur umfasst bisher nur die Kommunikation der Teilnehmer im Labor über eine Vermittlungsinfrastruktur mittels eines Message Broker Systems.

Die Richtlinien für eine Kommunikation über die Vermittlungsinfrastruktur umfassen grundlegende Elemente wie eine Auskunft über die Anwendungsversion und einen eindeutigen Identifikator. Der Umgang mit der Kommunikationsinfrastruktur wurde zunächst absichtlich einfach gehalten, um einen schnellen Einstieg für neue Studenten im Projekt zu gewährleisten. Studenten, deren Arbeit im Rahmen des Living Place Hamburg beendet ist, haben ihre Arbeit in einem Projektwiki festgehalten und dokumentiert. Trotz der Projektdokumentation sind die Arbeiten der Studenten meist zu komplex, um sich in die Thematik der Bachelor- oder Masterarbeit ohne massiven Zeitaufwand einzuarbeiten. Die Probleme, die entstehen wenn ein Student nach seinem Abschluss die Hochschule verlässt und sich Strukturen im Living Place ändern, sind komplex und der Student konnte diese nicht in seine Planung mit einbeziehen. Der damit einhergehende Einarbeitungsaufwand für Projekte, die eventuell vor Jahren entstanden sind, ist hoch und somit schwer planbar.

Mit vielen Projekten entstehen neue Komponenten in Sensorik oder Aktorik des Living Place. Die Kommunikation zwischen Sensoren und Aktoren im Living Place findet, genau wie die Kommunikation anderer Anwendungen, über den Message Broker statt und ermöglicht somit einen Zugriff auf diese. Momentan gibt es weder eine Möglichkeit herauszufinden, in welchem Zustand sich die Aktorik bzw. Sensorik befindet, noch ob die nötigen Anwendungen gestartet sind, die einen reibungslosen Laborbetrieb gewährleisten.

Um die Sensorik mit der Aktorik zu verbinden und ein reaktives Verhalten abzubilden, müsste man für jede Anwendung, die eine Funktionalität in das Labor integrieren soll, den gleichen Aufwand treiben um die Daten von Sensoren einzulesen, zu interpretieren und zu Aktionen der Aktoren zu führen. Außerdem würden Anwendungen, die sich mit der Steuerung von Aktoren im Living Place auseinandersetzen, um diese konkurrieren. Wenn ein Verhalten erstellt wird, das in Bereiche eingreift, die bereits mit anderen Anwendungen gesteuert werden, müssen diese angepasst werden.

Die Probleme in der aktuellen Infrastruktur befinden sich auf architektonischer und konzeptioneller Ebene. Es gibt keine Möglichkeit, für Studenten die Abhängigkeiten, die sich ergeben, automatisch aufzulösen. Weder Abhängigkeiten von Open Source Projekten,

noch solche, die im Rahmen des Labors entstanden sind, lassen sich auflösen.

Die Komplexität in der Softwareentwicklung ist seit Jahrzehnten ein Problem in der Informatik, vgl. (Diederichs, 2005, S. 3 ff). Viele der Herausforderungen, die in der Laborumgebung entstehen, lassen sich auch in der freien Wirtschaft wiederfinden. Mit Hilfe moderner Konzepte aus verschiedenen Bereichen der Softwareentwicklung lässt sich diese Komplexität im Entwicklungsprozess mindern.

1.2 Ziele dieser Arbeit

Im Rahmen des Living Place Hamburg soll in dieser Arbeit ein System entwickelt werden, das es den Projekten im Labor ermöglicht, zielgerichtet in einem Gesamtkonzept für Infrastruktur und Programmiermodell neue Ansätze zu entwickeln. Das Gesamtkonzept soll eine Lösung für die in der Motivation genannten Problemstellungen bieten und auch im Vergleich zu anderen Lösungsansätzen möglichst flexibel bleiben.

Die Schwierigkeiten in der Wiederverwendbarkeit von Projekten wird durch eine einheitliche Projektdefinition behoben. Durch die Projektdefinitionen lassen sich die Projekte als Abhängigkeiten nutzen. Mit Hilfe einer geeigneten Erweiterung der Infrastruktur wird es in Zukunft die Möglichkeit geben, Abhängigkeiten im Rahmen des Labors automatisch aufzulösen. Es wird ein geeignetes Werkzeug gewählt um diesen Anforderungen gerecht zu werden.

Um den Laborteilnehmern die Möglichkeit zu geben, einen Überblick über die Projekte und ihre Status zu bekommen, wird die Infrastruktur um eine Laufzeitumgebung erweitert, die eine Möglichkeit dazu bietet. Diese Arbeit ermittelt eine geeignete Laufzeitumgebung, die diese Aufgabe erfüllen kann und integriert diese in die Laborumgebung.

Zur Integration von neuer Sensorik und Aktorik wird eine programmatische Abstraktion entwickelt, die eine Laborintegration dieser vereinfacht. Des weiteren werden Softwarekomponenten implementiert, die Sensorik und Aktorik verwalten und an eine regelbasierte Maschine anbinden. Mit dieser Anbindung wird es möglich, die Wohnumgebung des Labors mit regelbasierter Funktionalität zu erweitern. Es wird eine geeignete regelbasierte Maschine gewählt und mit den entwickelten Komponenten auf der Laufzeitumgebung installiert.

Diese Arbeit konzentriert sich auf die infrastrukturelle Umsetzung der Mechanismen, die oben genannte Vorteile im Laborbetrieb realisieren können. Die Überführung der vorhandenen Anwendungen, sowie Aktorik und Sensorik, ist nicht Teil dieser Arbeit.

Die Integration der Mechanismen beschränkt sich auf das Bereitstellen der Komponenten und die Integration von fiktiver Sensorik und Aktorik zur Demonstration der Lauffähigkeit.

1.3 Gliederung der Arbeit

Die vorliegende Arbeit ist in sechs Kapitel aufgeteilt. Das erste Kapitel, die Einleitung, schließt mit diesem Abschnitt, der Gliederung.

Im folgenden Kapitel werden die für diese Arbeit nötigen Grundlagen erarbeitet, vgl. Abschnitt 2. Hier wird das Labor und die untersuchten Forschungsschwerpunkte vorgestellt, vgl. Abschnitt 2.1. Anschließend werden die Grundlagen für die drei Schwerpunkte dieser Arbeit vorgestellt, vgl. Abschnitt 2.2, 2.3 und 2.4.

In der anschließenden Analyse wird anhand von Szenarien die Problemstellung verdeutlicht, vgl. Abschnitt 3 und 3.3. Die Art der vorhandenen Softwareprojekte, das Life Cycle Management und die Integration neuer Aktorik und Sensorik wird im Abschnitt 3.4 untersucht. Weitergehend wird eine Bestandsaufnahme der bereits integrierten Aktorik und Sensorik vorgenommen und die vorhandenen Kommuniktationsmechanismen untersucht, vgl. Abschnitt 3.1. Der folgende Abschnitt untersucht vergleichbare Arbeiten die bereits in diesem Kontext durchgeführt wurden, vgl. Abschnitt 3.5.

Das vierte Kapitel ist das Designkapitel dieser Arbeit, vgl. Abschnitt 4. Es werden Lösungen zu den Problemstellungen auf den verschiedenen Ebenen der Laborinfrastruktur im Bezug auf Dependency Resolution, einer geeigneten Laufzeitumgebung und der Einsatz von Complex Event Processing (CEP) in den Abschnitten 4.1, 4.2 und 4.3 entwickelt. Der Abschnitt 4.4 befasst sich eingehend mit der im Rahmen dieser Arbeit verfassten Implementation zur Integration einer geeigneten CEP-Maschine in die Laborumgebung.

Das vorletzte Kapitel befasst sich mit der Integration der gewählten Komponenten in die Laborumgebung, vgl. Abschnitt 5. Hier wird die Nutzung eines Maven-Proxy zur Dependency Resolution, der Einsatz einer Apache Karaf Open Services Gateway initiative (OSGi) Laufzeitumgebung und die daraus resultierenden Mechanismen erläutert, vgl. Abschnitt 5.1, 5.2 und 5.2.4.

Das letzte Kapitel schließt diese Arbeit mit einer Zusammenfassung und einem Ausblick ab, vgl. Abschnitt 6.1 und 6.2.

2 Grundlagen

Dieser Abschnitt erläutert die Grundlagen als Voraussetzung für die in dieser Arbeit verwendeten Begrifflichkeiten und Konzepte. Die Programmiersprache ist Java, daher beziehen sich die folgenden Erläuterungen konkret auf die Mechanismen, die in dieser Sprache eingesetzt werden. Es wird ein Überblick der Forschungsschwerpunkte im Living Place Hamburg gegeben und anschließend die drei Hauptthemen, mit denen sich diese Arbeit auseinander setzt, erläutert.

2.1 Der Living Place Hamburg

Der Living Place Hamburg ist ein Labor, in dem Forschung zum Thema Context Awareness, Ubiquitous Computing und Smart Homes betrieben wird, vgl. (Voskuhl, 2011, S. 10 ff.). Im Folgenden werden diese Begriffe im Kontext dieser Arbeit beschrieben.

Context Awareness Context Awareness beschäftigt sich mit der Frage, wie man Computer mit einer Umgebungswahrnehmung ausstatten kann, sodass diese in die Lage versetzt werden sich entsprechend der Umgebungsinformationen zu verhalten. Der Computer der diese Umgebungsinformationen durch Sensorik oder Kommunikationsmechnismen erhält, reagiert auf diese und verhält sich der Situation angemessen. Informationen über den Kontext können zum Beispiel Positions-, Temperatur- oder Helligkeitsmesswerte enthalten, aber auch abstraktere Daten wie Umgebungsressourcen wie die Nähe von Bildschirmen oder Geräten auf die der Benutzer zurückgreifen kann, vgl. (Schilit u. a., 1994, S. 85 f.). Um Projekte mit Informationen über die Umgebung zu versorgen damit sich diese entsprechend der Umstände verhalten können, also *context aware*, kann jedes System auf die Sensorik und Aktorik im Living Place Hamburg zurückgreifen.

Ubiquitous Computing Der von Mark Weiser geprägte Begriff des Ubiquitous Computing, beschreibt die heute in vielen Bereichen des Lebens vorzufindende allgegenwärtige Rechenleistung. Bereits Anfang der neunziger Jahre hat er festgestellt, dass Computer heute in unserer Umgebung nicht mehr als solche wahrzunehmen

sind, vgl. Weiser (1991). Die Menschen umgeben sich meist mit weit mehr als einem Computer und haben stetigen Zugriff auf die Informationen mit denen sie versorgt werden wollen. Die meistgenutzten Geräte sind dabei Laptops, Desktop PCs und SmartPhones, vlg. Dearman und Pierce (2008). Im Living Place Hamburg wurde bei der Sensorik besonders darauf geachtet, dass diese nicht hervorsticht und den Bewohner nicht in seinem natürlichen Verhalten stört. Auch Verbindungen von Systemen wurden entweder kabellos realisiert oder die Kabel wurden im Unterboden des Wohnbereichs verlegt.

Smart Homes Der Begriff des Smart Home beschreibt ein Zuhause das Intelligenz auf mehreren Ebenen besitzt. Eine Definition aus Jiang u. a. (2004) setzt für ein Smart Home drei Aspekte voraus:

> *„Accordingly, a home, which is smart, must contain three elements, which are internal network, intelligent control and home automation. Internal network is the basis of a smart home, and it can be wire, cable and wireless. Intelligent control means gateways to manage the systems. Home automation means products within the homes and links to services and systems outside the home.“*

Der Living Place Hamburg bietet auf allen drei Ebenen die genannten Voraussetzungen um Forschung in diesem Bereich zu betreiben, vgl. Abschnitt 3.1. Auch die Interaktion der Benutzer mit dem Smart Home wird im Rahmen des Labors untersucht, vgl. von Luck u. a. (2010). In diesem Zusammenhang wird auch oft von Ambient Assisted Living (AAL) gesprochen, was konkreter davon ausgeht, dass die Wohnung, die den Bewohner umgibt, diesen unterstützt und mittlerweile Aspekte wie soziale Akzeptanz als gegeben angenommen werden, vgl. (Marit Hansen und Meissner, 2008, S. 202 f.).

2.2 Dependency Resolution

Der Begriff der „dependency resolution“ aus der Softwareentwicklung bezieht sich auf das Auflösen von Abhängigkeiten bei der Verwendung von Bibliotheken oder Programmkomponenten, die nicht lokal auffindbar sind. Lokal heißt in diesem Kontext, dass die benötigte Ressource nicht über die dem Compiler bekannten Mechanismen aufzufinden ist.

Eine Abhängigkeit in einem Programm entsteht durch das Einbinden von Entitäten aus Namensräumen die nicht Teil des Programms sind. Die Abhängigkeit ist durch den Namensraum allerdings nicht vollständig definiert, denn Software wird versioniert und kann sich entsprechend von Version zu Version unterscheiden.

Die Abhängigkeiten in einem Programm bilden einen Graph $G = (V, E)$, in dem $V = vertices$ die Knoten und $E = edges$ die Kanten darstellt. Der Graph G muss ein gerichteter azyklischer Graph sein. Existiert ein so gearteter Graph, so kann dieser topologisch sortiert werden. Ein topologisch sortierter Graph G ist eine nach Kanten sortierte Form des Graphs G, bei der sichergestellt ist, dass u vor v in der Ordnung von G kommt, wenn G eine Kante (u, v) enthält. Man kann den topologisch sortierten Graph als eine von links nach rechts geordnete Liste betrachten, in der die weiter links stehenden Elemente als Voraussetzung für Elemente weiter rechts gelten, vgl. (Cormen u. a., 2009, S. 612 f.). Ein in dieser Art sortierter Graph bildet eine Liste von Abhängigkeiten, die von links nach rechts alle Abhängigkeiten eines Programms beschreibt, vgl. Abb. 2.1.

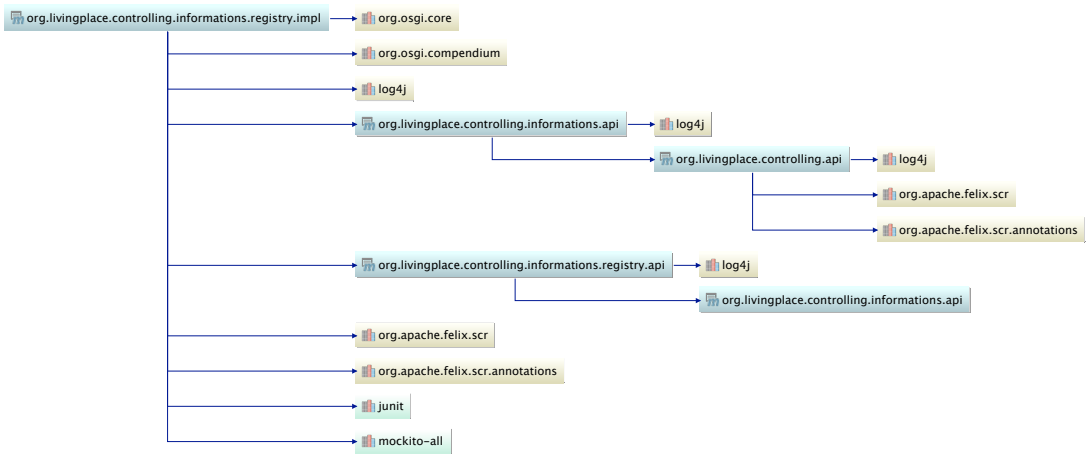

Abbildung 2.1: Beispielhafter Abhängigkeitsbaum eines Maven Projekts

Es gibt zwei Möglichkeiten, Abhängigkeiten aufzulösen, manuell und automatisch.

manuelle Abhängigkeitsauflösung Wenn man eine Abhängigkeit manuell auflöst, sorgt man selbst als Programmierer dafür, dass die Bibliothek oder Komponente, von der man abhängt, zum Zeitpunkt des Übersetzens dem Compiler bekannt ist. In der Praxis sucht man im Internet nach der Funktionalität, die man in seinem Programm wieder verwenden möchte und lädt diese in binärer Form herunter. Dann

erweitert man den Klassenpfad[1] um die Abhängigkeit in binärer Form, kopiert also die Binärdatei in den dem Compiler bekannten Klassenpfad. Der Compiler kann dann beim Übersetzen die Abhängigkeit auflösen, weil diese nun bekannt ist. Viele Abhängigkeiten basieren aber ihrerseits auch auf Bibliotheken die deren Abhängigkeiten darstellen. Dem Compiler ist die direkte Abhängigkeit bekannt, aber nicht die transitiven, Abhängigkeiten der Abhängigkeit. Die transitiven Abhängigkeiten müssen wiederholt manuell aufgelöst werden, bis der Abhängigkeitsbaum manuell aufgelöst ist und sich das Programm vollständig übersetzen lässt.

automatische Abhängigkeitsauflösung Die automatische Abhängigkeitsauflösung übernimmt die Aufgabe des manuellen Kopierens und Erweiterns der Binärdatei und des Klassenpfades. Werkzeuge, die diese Schritte automatisieren, greifen meist auf Verzeichnisse, die eine maschinenlesbare Form der Abhängigkeit anbieten, zurück. Um eine Abhängigkeit eindeutig identifizieren zu können, enthält die maschinenlesbare Form der Abhängigkeit die Versionsnummer und bietet damit auch die Möglichkeit, nach einer bestimmten Version zu suchen und aufzulösen.

Durch die mittlerweile immer größer werdende Verfügbarkeit von OpenSource Bibliotheken steigt auch die Anzahl von frei verfügbaren Bibliotheken und Komponenten, die sich wiederverwenden lassen. In Ossher u. a. (2010) wird darauf eingegangen, dass diese freien Komponenten auch untereinander Abhängigkeiten gebildet haben und sich damit ein komplexer Baum von Abhängigkeiten bildet. Durch die Abstraktion von Komplexität in Programmen werden diese Abhängigkeiten immer größere Äste im Abhängigkeitsbaum eines Programms bilden und damit den Aufwand erhöhen, diese Abhängigkeiten manuell aufzulösen. Der Aufwand, der damit verbunden ist, kann vermieden werden, indem spezielle Programme zur Automatisierung verwendet werden. In Abschnitt 4.1 wird die für diese Arbeit gewählte Software detailliert vorgestellt.

2.3 Application Container

Ein Applikationscontainer bündelt eine Applikation in einem Container und beschreibt diese inklusive aller Abhängigkeiten, Schnittstellen und Metadaten vollständig. Das Konzept einer abschließend beschriebenen Entität dient der Modularisierung von Bibliotheken und ist ein wichtiger Bestandteil des Softwareengineerings. Ein Modul ist dabei eine logisch unabhängige Komponente in einem System, die eine klar definierte

[1] Bekannter unter der englischen Bezeichnung „classpath“

Funktionalität abbildet. Es definiert die logischen Grenzen eindeutig, indem der Quelltext entweder ein Teil des Moduls ist(im Modul enthalten oder eingebettet) oder kein Teil des Moduls(also außerhalb des Moduls). Die internen Details eines Moduls sind nur innerhalb des Moduls bekannt. Alle nach außen bekannten Informationen über das Modul und dessen Schnittstellen werden explizit nach außen veröffentlicht und stellen das Application Programming Interface (API) dar. Ein weiterer Vorteil eines Application Containers ist die Möglichkeit, die Komponenten einzeln zu überwachen, beziehungsweise ihre Status abzufragen. Es gibt also die Möglichkeit das Softwaresystem in Funktionalitäten zu unterteilen und diese getrennt voneinander zu beobachten, vgl. (Hall u. a., 2011, S. 24 ff.).

Die Vorteile von Modularisierung und modularer Programmierung wurden schon in den 70er Jahren von Parnas (1972) beschrieben und sollen hier herangezogen werden, um eine Definition für die Vorteile eines modularen Systems zusammenzufassen:

> *„A well-defined segmentation of the project effort ensures system modularity. Each task forms a separate, distinct program module. At implementation time each module and its inputs and outputs are well-defined, there is no confusion in the intended interface with other system modules. At checkout time the integrity of the module is tested independently; there are few scheduling problems in synchronizing the completion of several tasks before checkout can begin. Finally, the system is maintained in modular fashion; system errors and deficiencies can be traced to specific system modules, thus limiting the scope of detailed error searching.“* - Parnas (1972)

Im Rahmen dieser Arbeit wird ein modulares System vorgestellt, in das die Softwarestruktur im Living Place Hamburg integriert werden kann. Das für diese Arbeit gewählte Modell wird in 4.2 detaillierter erläutert.

2.4 Regelbasiertes System

Ein regelbasiertes System ist ein System, das aus einer Wissensbasis, einer Menge von Regeln und einer Regelmaschine besteht. Die Wissensbasis, auch Faktenbasis genannt, ist dafür zuständig, die Fakten, die das System kennt, in einer geeigneten Form zu halten. Die geeignete Form hängt davon ab, wie dieses Wissen durch Regeln abgefragt werden soll. Die Menge von Regeln, auch Regelbasis genannt, gibt dem System einen flexiblen Mechanismus, Bedingungen zu formulieren. Die Regelmaschine versteht die

Regeln aus der Regelbasis und evaluiert ihr Zutreffen anhand der Faktenbasis. Generell beschreiben Regeln die Umstände, auf deren Basis Aktionen folgen können. Regeln haben die folgende Form:

```
1 WENN
    ... // Praemisse
3 DANN
    ... // Konklusion
```

Wenn die Faktenbasis alle Fakten enthält, um die Prämisse der Regel zu erfüllen, wird die Konklusion ausgeführt. Ob die Fakten vorhanden sind, die die Prämisse erfüllen, stellt die Regelmaschine anhand der Faktenbasis fest. Der Mechanismus mit dem die Regelmaschine diese Prämisse evaluiert ist ein Algorithmus, der erstmals als Rete-Algorithmus präsentiert wurde, vgl. Forgy (1982).

2.4.1 Rete-Algorithmus

Der Rete-Algorithmus wurde entwickelt, um die Regelauswertung in regelbasierten Systemen zu optimieren und effizienter zu gestalten. In der Praxis besteht die Programmlogik aus einem azyklischen, gerichteten Graph, der ein Netz aufspannt, das die Regeln der Regelbasis abbildet und Fakten verarbeitet. Der Graph ist in zwei Teile aufgeteilt, den vorderen Teil des Netzes, das *Alpha*netz, und den hinteren Teil des Netzes, das *Beta*netz. Bevor ein Fakt diesen Graph durchläuft, wird er in ein oder mehrere sog. working memory elements (WMEs) umgewandelt. Ein WME besteht immer aus einem Typ, einem oder mehreren Attributen und dessen Wertigkeiten. WMEs bestehen aus Token die dem System als Abstraktion des Fakts dienen und gemeinsam den working memory (WM) des Systems bilden. Der WM bildet den sich ständig ändernden Gesamtzustand des Systems, vgl. (Brachman und Levesque, 2004, S. 119 ff). Die Eintrittsknoten des Alphanetzes prüfen zunächst den Typ des WME und entscheiden damit welchen Teil des Graphen dieses WME-Token durchläuft. Anschließend folgen Knoten, die einfache Bedingungen wie zum Beispiel `age < 14` abbilden. So läuft jedes WME-Token durch das Alphanetz, indem alle einfachen Bedingungen, die nur auf dem Token selbst basieren, evaluiert werden. Im Betanetz werden die Knoten von einfachen Bedingungen zu Bedingungen erweitert, die mehrere Eigenschaften zusammenfassen. Betaknoten verbinden mehrere Eigenschaften der WMEs durch die Kombination von Bedingungen. Betaknoten haben zwei Eingänge und verbinden damit mehrere Alphaknoten. Wenn ein Token durch alle Alpha- und Betaknoten traversiert ist, kommt es bei einem Endknoten

an. Der Endknoten stellt das Ausführen einer Regel dar, die als Prämisse alle vorher durchlaufenen Knoten hat, vgl. Abb. 2.2[2] und (Brachman und Levesque, 2004, S. 127 ff).

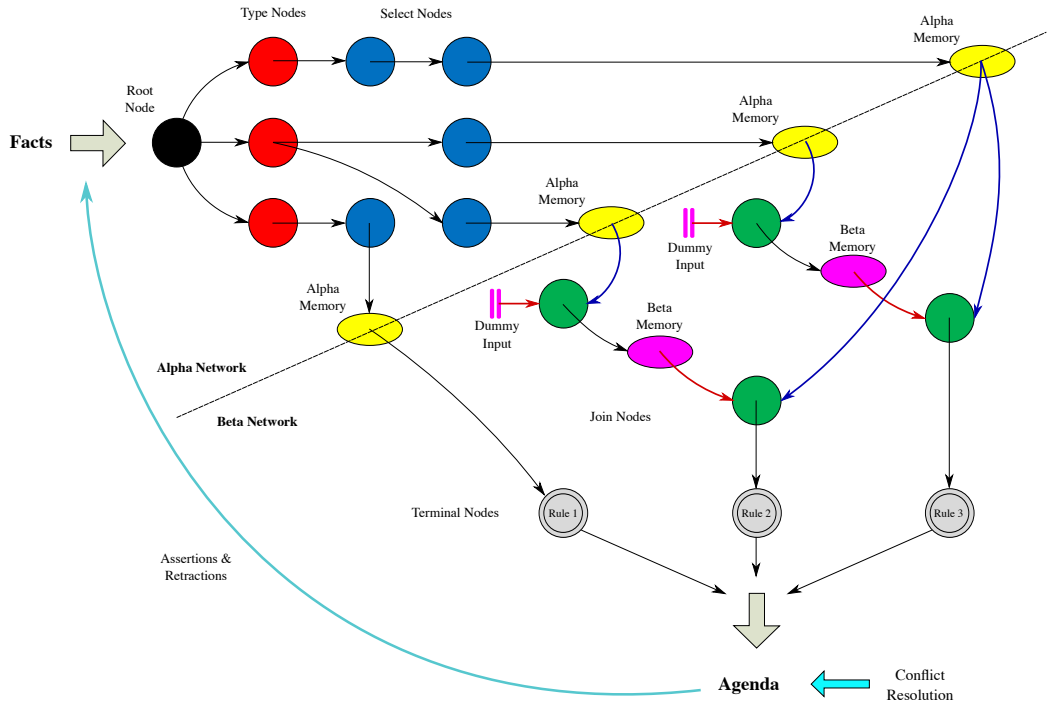

Abbildung 2.2: Schematische Abbildung eines für den Rete-Algorithmus verwendeten Alpha- und Betanetzes

Bis heute wurde der Algorithmus mehrfach verbessert und bis zu seiner aktuellen Version weiterentwickelt. Die aktuelle Version des Rete-Algorithmus ist der Rete-NT Algorithmus der von Dr. Charles Forgy für die Firma Sparkling Logic lizensiert wurde[3].

2.4.2 Complex Event Processing

Event Processing beschreibt die Analyse und Verarbeitung von Events innerhalb eines Datenstroms. CEP ist eine spezielle Form des Event Processing, in dem der Fokus nicht auf einem einzelnen Datenstrom liegt, sondern darauf, komplexe Zusammenhänge in

[2] Quelle: http://commons.wikimedia.org/wiki/File:Rete.svg; abgerufen am: 16.04.2013
[3] http://my.sparklinglogic.com/index.php/about-sparkling-logic sparklinglogic.com - Dr. Charles Forgy, PhD - Strategic Advisor; abgerufen am: 21.11.2012

Events unterschiedlicher Quellen zu identifizieren und darauf zu reagieren. In Walzer u. a. (2007) wird die Aufgabe von CEP wie folgt definiert:

> Complex event processing (CEP) is the technology used to perform the detection of complex events consisting of combinations of single events. Complex event detection is the process of identifying patterns of events with logical, temporal or causal relationships out of the single event occurrences.
>
> The Rete algorithm is commonly used in rule-based systems to trigger certain actions if a corresponding rule holds. It allows for a high number of rules and is therefore ideally suited for event processing systems.

CEP wird zur Identifikation und Auswahl von Ereignissen, die in einem logischen, temporären oder kausalen Zusammenhang stehen, verwendet. Die Zusammenhänge sind dabei unterschiedlicher Natur und können sich von Ereignis zu Ereignis unterscheiden. Eine CEP Maschine ist eine Anwendung, die auf den Einsatz in diesem Umfeld spezialisiert ist und effiziente Algorithmen einsetzt, um die beschriebenen Zusammenhänge zwischen Ereignissen zu erkennen.

Für den Einsatz in einem Smart Home ist eine CEP Maschine interessant, weil die Möglichkeit zum Formulieren zeitlicher Abhängigkeiten eine Vielzahl realistischer Szenarien bietet. Zeitliche Abhängigkeiten können dabei helfen, das Verhalten der intelligenten Wohnung an die Bedürfnisse der Bewohner anzupassen.

3 Analyse

Die Analyse dieser Arbeit befasst sich mit der Untersuchung der vorhandenen Infrastruktur und den Anforderungen die sich durch ihre Schwächen ergeben. Zunächst wird die aktuelle Sensor- und Aktorinfrastruktur analysiert und im Anschluss die Kommunikationsmechanismen zwischen diesen untersucht. Anhand von drei Szenarien werden die Probleme veranschaulicht denen Studenten aktuell begegnen wenn sie neue Projekte in die Laborinfrastruktur integrieren. Mit Hilfe dieser Szenarien werden im Anschluss drei Ebenen identifiziert, auf denen Verbesserungen die größten Auswirkungen hätten. Durch die Analyse der vergleichbaren Arbeiten werden bereits bekannte Konzepte auf deren Anwendbarkeit untersucht, um im Anschluss dieses Kapitel mit einer Anforderungsaufstellung und Abgrenzung zu schließen.

3.1 Laborinfrastruktur

Der Living Place Hamburg ist, wie in den Grundlagen bereits erwähnt, ein Labor zur Forschung auf den Gebieten Context Awareness, Ubiquitous Computing und Smart Homes. An dieser Stelle wird die Infrastruktur erläutert und die Sensorik und Aktorik in diesem Labor vorgestellt. Alle hier aufgeführten Sensoren und Aktoren sind bereits in das Labor integriert und werden auf unterschiedliche Art und Weise genutzt. Anhand eines konkreten Bezugs zum Szenario aus Abschnitt 3.3.2 werden Anforderungen erarbeitet, die den Einstiegsaufwand für neue Projekte gering halten sollen.

3.1.1 Sensoren

Die Laborsensorik reicht von einfachen Temperaturgebern, über Positionssensorik bis zu einer Ausstattung mit der Nutzbarkeitsuntersuchungen möglich sind. Im Folgenden werden die Sensoren des Labors vorgestellt, in Abbildung 3.1.1 sind sie farblich blau hinterlegt.

Kameras Die im Labor befindlichen Kameras sind schwenkbar, vandalisierungsresistent und liefern HD-Farbbilder. Sie verfügen über zwei Anschlüsse, einen analogen

Anschluss, der in Echtzeit Bilder mit niedriger QualitätStandard Definition (SD) liefert und einen Netzwerkanschluss der High Definition (HD) Bilder liefert[1]. Diese Datenströme werden über einen Aufnahmerechner abgegriffen und im Kontrollraum angezeigt. Die insgesamt fünf Kameras sind so installiert, dass sie jeden Teil des Labors einsehen können, vgl. Abb. 3.1.1 [2]. Sie dienen zur Beobachtung von Probanden des Labors.

Mikrofone Auch 5 Richtmikrofone wurden im Labor installiert. Diese hochsensitiven Mikrofone ermöglichen die Aufnahme von Gesprächen und Geräuschen im gesamten Labor. Sie sind so angebracht, dass sie möglichst nicht wahrgenommen werden um den Eindruck der Beobachtung möglichst gering zu halten, vlg. Abb. 3.1.1 [5].

Positionserkennung Das Sensorsystem zur Positionserkennung[2] im Labor ist in der Lage sog. Tags im Labor zu lokalisieren. Diese Tags können Personen gegeben oder an Gegenständen angebracht werden, um Bewegungen oder Positionen nachvollziehen zu können, vgl. (Voskuhl, 2011, S. 82 ff). Auch diese Sensoren sind so angebracht, dass sie alle Teile des Labors einsehen können und damit die Möglichkeit bieten Personen und Gegenstände bis auf 15 *cm* genau im Labor zu lokalisieren, vgl. Otto und Voskuhl (2011) und Abb. 3.1.1 [1].

Intelligentes Bett Das im Labor befindliche Bett ist mit einer Sensorik ausgestattet, die es dem Labor erlaubt eine berührungslose Schlafphasenerkennung durchzuführen, vgl. Quast (2011). Dieser Sensor stellt über Dehnungsmessstreifen die Möglichkeit zur Verfügung Probanden im Bett zu erkennen und deren Schlafverhalten zu analysieren, vgl. Abb. 3.1.1 [7].

Kabelloses Sensornetzwerk Das kabellose stromsparende Sensornetzwerk im Labor umfasst insgesamt 8 Knoten, die in dem gesamten Labor verteilt sind, vgl. Abb. 3.1.1 [3]. Diese Sensoren geben Auskunft über Helligkeit und Temperatur der Wohnung. Sie werden auch zur Auskunft über den Kippstand der Fenster verwendet. Das Sensornetzwerk organisiert sich selbst in einem MESH und kann um weitere Messknoten erweitert werden, vgl. Pautz (2011).

Cubical Der Cubical ist ein sog. Tangible Device der Firma Seamless Interaction. Dieses Tangible Device gibt Auskunft über die Lage der drei Achsen und lässt damit die

[1] http://livingplace.informatik.haw-hamburg.de/wiki/index.php/Hardwareausstattung; abgerufen am 26.11.2012

[2] Ubisense Indoor Positioning System, http://de.ubisense.net/en/; abgerufen am: 04.12.2012

Steuerung von Lautstärke, Programm oder Licht zu. Dieser Sensor ist besonders wichtig für die Steuerung des Labors, denn mit ihm können weitere Aktoren einfach gesteuert werden[3].

Multitouch-Tresen SiBar Ein Multitouch-Tresen, die sog. „*SiBar*“, ist ein weiteres zentrales Element der Erkennung von Verhalten und Position der Probanden. Der Tresen misst 2 mal 1 Meter Touchfläche und beinhaltet Sensorik die eine Positionserkennung am Tisch ermöglicht, vgl. Abb. 3.1.1 [4]. Der Tresen kann außerdem als Steuerelement fungieren. Für das Touchinterface wurden bereits Applikationen zur Steuerung des Labors integriert[4].

Multitouch-Tisch Der Multitouch-Tisch im Loungebereich der Wohnung wird als Eingabegerät zur Steuerung des Fernsehers sowie anderer Geräte eingesetzt. Das Microsoft Surface 2[5] dient dem Proband außerdem als stationärer Zugang zum Internet und bildet den Mittelpunkt im Loungebereich, vgl. Abb. 3.1.1 [9].

Couchsensorik Die Couch im Loungebereich des Labors hat eine eingebaute Sitzdetektion, mit der es möglich ist, die Position und Lage des Probanden zu detektieren, vgl. (Dreschke, 2011, S. 44 ff) und Abb. 3.1.1 [6]. Bei dieser Sensorik kommt es zum Einsatz von Schwingkreisen, die unter anderem verwendet werden, um eine gezielte Ausrichtung von Steuerungselementen auf dem Touchtisch vor dem Sofa vorzunehmen.

Sturzerkennung Die Sturzerkennung der Laborumgebung wurde wie auch das Bett im vorderen Schlafbereich platziert, vgl. Abb. 3.1.1 [8]. Hier kommen kapazitive Sensoren zum Einsatz und liefern präzise Ergebnisse über die Art des Falls einer Person auf den Teppich, vgl. (Teske, 2011, S. 59).

3.1.2 Aktoren

Die Aktorik im Labor umfasst neben Anzeigen und Bildschirmen auch eine Fenster- und Heizungssteuerung. Diese und weitere Aktoren werden hier vorgestellt.

Fenstersteuerung Die Fenstersteuerung ermöglicht es den Programmen im Labor, die Fenster zu einem bestimmten Grad zu öffnen oder zu schließen. Außerdem kann mit

[3] http://seamlessinteraction.com/cubical/Cubical.html; abgerufen am: 26.11.2012
[4] http://seamlessinteraction.com/blog/?p=70; abgerufen am: 21.11.2012
[5] http://www.microsoft.com/en-us/pixelsense/default.aspx; abgerufen am: 25.11.2012

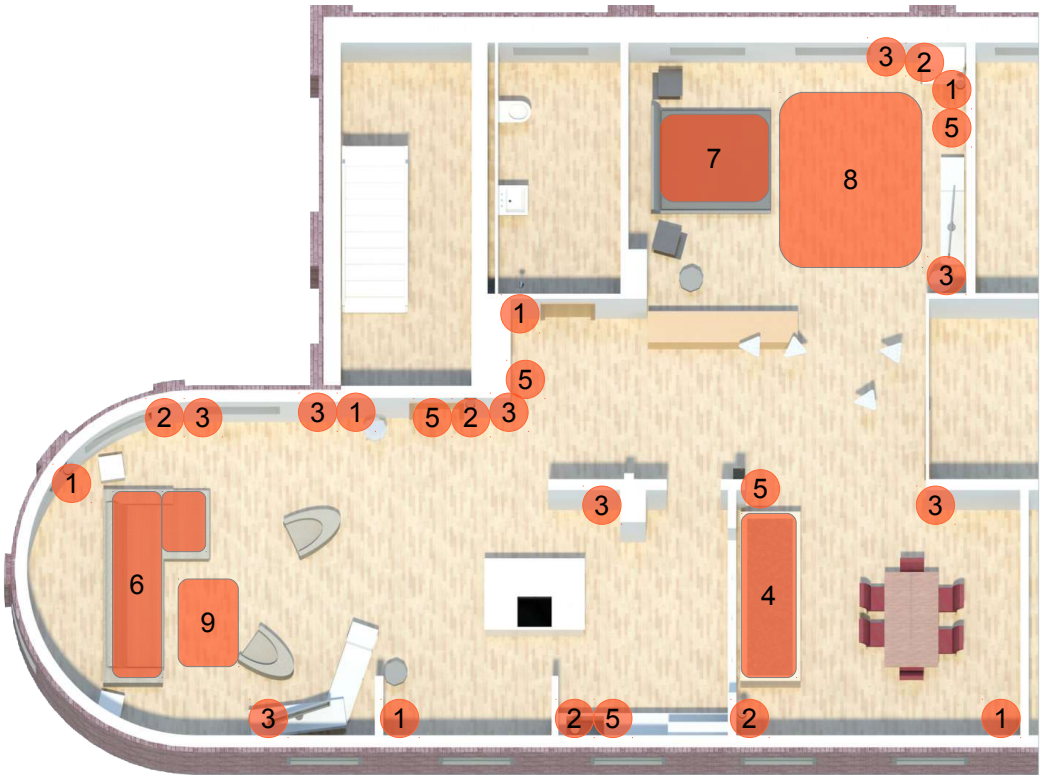

Abbildung 3.1: Laborsensorik im Wohnbereich des Living Place Hamburg: [1] Positionserkennung [2] Kameras [3] Kabelloses Sensornetzwerk [4] Multitouch-Tresen SiBar [5] Mikrofone [6] Couchsensorik [7] Intelligentes Bett [8] Sturzerkennung [9] Multitouch Tisch

ihr die Temperatur und Luftbeschaffenheit reguliert werden. Auch die Rollos im Labor können angesteuert werden. Alle Fenster und Rollos sind einzeln ansteuerbar und es gibt Mechanismen mit denen man alle auf einmal ansteuern kann, vgl. Abb. 3.1.2 [1].

Heizungssteuerung Alle Heizungen im Labor sind mit der Heizungssteuerung ansteuerbar und ermöglichen mit der Fenstersteuerung eine Klimaregelung im Labor, vgl. Abb. 3.1.2 [2]. Die Klimaregelung kann Projekten dabei helfen auf die Gewohnheiten der Probanden einzugehen und dabei ein energiesparendes Heizverhalten zu automatisieren.

Türsteuerung Die Türsteuerung dient der Wohnung als Klingelsystem mit Videoausgabe. Außerdem werden die Anzeigegeräte im Lounge- und Schlafbereich genutzt, um den Probanden über die Person vor der Tür zu informieren, vgl. Bornemann (2011) und Abb. 3.1.2 [3].

Anzeigegeräte Im Labor sind Monitore zum normalen Fernsehen im Lounge- und Schlafbereich. Diese Anzeigen werden nicht zur Eingabe genutzt, sondern nur als darstellende Entitäten verwendet. Auch der Multitouch Tresen „*SiBar*“ sowie der Multitouch-Tisch im Loungebereich bilden Anzeigen ab, eignen sich allerdings auch zur Eingabe von Gesten und Aktionen der Probanden, vgl. Abb. 3.1.2 [4].

Lichtsteuerung Die Lichtsteuerung des Labors ist aufwendig integriert, RGB-LED Leisten und zusätzliche Halogenspots wurden im gesamten Wohnraum verbaut. Das komplette Labor lässt sich durch diese Beleuchtung einfärben und es lassen sich unterschiedliche Szenen abbilden, die zu Anlässen wie einer Morgensituation oder einem Abendessen passen, vgl. Abb. 3.1.2 [5].

3.2 Kommunikationsschnittstelle

Die Kommunikation im Living Place Hamburg wird über ein Message Broker System abgebildet. Ein Message Broker bietet viele Vorteile zur Vereinfachung der Kommunikation, da er als zentraler Knotenpunkt in der Lage ist, Nachrichten zu verteilen, ohne das die Kommunikationsteilnehmer einander kennen. Um eine Kommunikation zu realisieren, bei der sich die Teilnehmer nicht kennen müssen, werden hier zwei unterschiedliche Mechanismen eingesetzt. Der eine Mechanismus ist das Consumer-Producer Pattern und der andere ist das Publisher-Subscriber Pattern. Das Consumer-Producer Pattern

Abbildung 3.2: Laboraktorik im Wohnbereich des Living Place Hamburg: [1] Fenstersteuerung [2] Heizungssteuerung [3] Türsteuerung [4] Anzeigegeräte [5] Lichtsteuerung

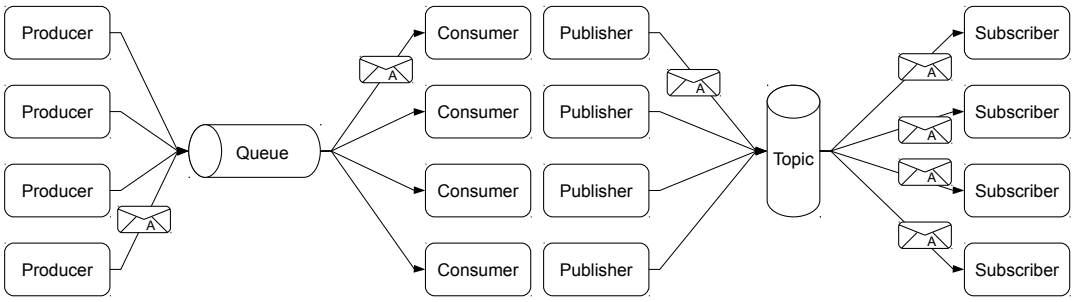

Abbildung 3.3: Producer-Consumer Pattern

Abbildung 3.4: Publisher-Subscriber Pattern

wird über ein *Point-to-Point* Mechanismus mit einer Queue abgebildet, vgl. (Snyder u. a., 2011, S. 32) und Abb. 3.3. Das Publisher-Subscriber Pattern wird über einen *one-to-many* Mechanismus mittels Topics abgebildet, vgl. (Snyder u. a., 2011, S. 33) und Abb. 3.4.

Der ActiveMQ Message Broker Service hält ein Verzeichnis über alle Queues und Topics. In dem Verzeichnis wird verwaltet welche Consumer und Producer an welchen Queues bzw. welche Publisher und Subscriber an welchen Topics interessiert sind. Wenn sich ein Teilnehmer für eine Kommunikation anmeldet, gibt dieser nicht an, welche anderen Teilnehmer er erreichen möchte, sondern welche Queue oder welches Topic die Kommunikation betrifft, vgl. Otto und Voskuhl (2010).

Um im Laborrahmen Kommunikationsabläufe nachvollziehen zu können, wurde ein Mechanismus außerhalb der internen Persistenz des ActiveMQ erstellt. Der ActiveMQ hält die Daten in einer eigenen Datenbank[6] vor, um die Auslieferung zu gewährleisten, auch wenn der Broker einmal neu gestartet werden muss. Das reicht allerdings nicht, um einen Kommunikationsstrang zu verfolgen. Zur Vereinfachung der Kommunikationen im Labor wurde eine Wrapper-Bibliothek für den ActiveMQ erstellt. Diese Bibliothek kann verwendet werden, um die Standardeinstellungen nicht für jedes Projekt vornehmen zu müssen. Um eine Kommunikation zwischen mehreren Applikationen aufnehmen zu können, speichert der ActiveMQ-Wrapper alle Nachrichten in einer MongoDB, vgl. (Otto und Voskuhl, 2011, S. 5 ff).

Das Nachrichtenformat zum Austausch von Informationen ist JavaScript Object Notation (JSON). Die Inhalte der Nachrichten sind also in einem portablen Format verfasst, denn es gibt für nahezu jede Programmiersprache und Plattform verfügbare Bibliotheken,

[6] KahaDB, die ActiveMQ interne, dateibasierte Datenbank: `http://activemq.apache.org/kahadb.html`; abgerufen am: 04.12.2012

um dieses Format zu lesen bzw. zu interpretieren[7]. Dieses Nachrichtenformat ist auch geeignet, um Nachrichten in der MongoDB ab zu legen, da diese Datensätze im JSON Format annimmt[8].

Nahezu alle in Abschnitt 3.1.1 und 3.1.2 vorgestellten Sensoren und Aktoren lassen sich über diesen zentralen Kommunikationsmechanismus steuern beziehungsweise abfragen. Einzige Ausnahme bilden die Kameras und Mikrofone, die den ActiveMQ als Kommunikationsmittelpunkt im Labor umgehen, um diesen nicht mit den immensen Datenmengen von Audio- und Videodatenströmen zu belasten.

Die aktuell vorhandene Kommunikationsinfrastruktur bietet eine Möglichkeit für alle Projekte im Labor, miteinander zu kommunizieren. Was bei der Kommunikation fehlt, ist eine zentrale Stelle, an der die Aktionen und Informationen gebündelt zur Verfügung stehen. Es gibt keine Möglichkeit, ein Gesamtbild der Informationen zu interpretieren, um anhand der dadurch gewonnen Erkenntnisse eine Aktion auszuführen. Gerade bei der Kommunikation auf niedrigen Abstraktionsleveln, zum Beispiel bei Sensordaten oder Befehlen für Aktoren, ist es wichtig zu wissen, ob eine Aktion schon ausgeführt wird oder ob ein anderes Projekt um eine exklusive Ressource konkurriert. Die Rohdaten werden zwar alle über den ActiveMQ vermittelt, aber es gibt keine Möglichkeit diese interpretieren zu lassen, um im Zusammenhang Schlüsse zu ziehen.

Die Kommunikationsinfrastruktur wird in allen Bereichen des Labors eingesetzt und muss erhalten bleiben. Es soll auch weiterhin ein einfacher Einstieg in diese Strukturen gewährleistet werden. Bereits integrierte Projekte sollten nicht ausgeschlossen oder überarbeitet werden müssen. Die sich daraus ergebende Anforderung ist der Erhalt aller Möglichkeiten die bestehen. Des weiteren muss die Kommunikationsinfrastruktur mit einem Mechanismus erweitert werden der es erlaubt einen Überblick über die aktiven Softwarekomponenten abzubilden.

3.3 Szenarien

Im Folgenden werden drei Szenarien vorgestellt, die einen Einblick in die Problemstellungen geben. Die drei fiktiven Studenten *Alice*, *Bob* und *Carol* werden neue Projekte im Labor beginnen. Anhand der praktischen Beispiele wird des weiteren der Zusammen-

[7] http://www.json.org/; abgerufen am: 03.12.2012
[8] http://www.mongodb.org/display/DOCS/Schema+Design; abgerufen am: 03.12.2012

hang der drei Szenarien herausgestellt, um in den folgenden Abschnitten analysieren zu können, welche Probleme sich konkret, auf den unterschiedlichen Ebenen, ergeben.

Das erste Szenario zeigt die Probleme die *Alice* hätte, wenn sie eine Bibliothek die schon vorhanden ist, wieder verwenden möchte. *Bobs* Problemstellung ist auf einer anderen Ebene, er hat sein Softwareprojekt integriert, hat aber keine Möglichkeit den Zustand seiner Software zu erfragen. *Carol* hat das Problem, dass sie ein neues Verhalten in das Labor integrieren will, aber es ohne großen Einarbeitungsaufwand keine Möglichkeit für sie gibt, das zu integrieren.

3.3.1 Szenario: Neue Teilnehmer im Labor

Das Laborteam hat eine neue Mitstreiterin gewonnen, *Alice* interessiert sich für die Infrastruktur der Kommunikation im Living Place Hamburg und möchte sich näher damit beschäftigen. Sie fragt die Verantwortlichen für die Kommunikationsschnittstelle und erfährt, dass es einen so genannten ActiveMQ-Wrapper gibt, um die Kommunikation in Softwareprojekten zu vereinfachen. Die Quellen liegen im Subversion[9] Server des Labors und die Version, die sie benutzen soll, ist die Revision 83. Die aktuellste Revision der Bibliothek ist allerdings schon 112 und kann im Moment nicht erstellt werden, denn die neue Funktion ist leider mit Übersetzungsfehlern eingecheckt worden. Also installiert *Alice* sich einen Subversion-Client und lädt die Quellen der Revision 83 herunter. *Alice* verwendet normalerweise Eclipse[10] als Integrated Development Environment (IDE), aber der ActiveMQ-Wrapper wurde mit der Netbeans[11] IDE erstellt. Da das Importieren der Netbeans Projektdefinition nicht funktioniert hat, muss sie sich Netbeans installieren. Nachdem sie in die aktuelle Netbeans Version das alte Format der Projektdefinition von Revision 83 migriert hat, kann sie die Bibliothek kompilieren. Nun hat *Alice* sich in die Quellen eingearbeitet und den Fehler aus Revision 112 korrigiert und möchte diese Funktion auch anderen Studenten zur Verfügung stellen. Sie dokumentiert im Wiki, dass sie mit der Revision 113 die aktuellste Version erstellen kann und diese die neuen Funktionen enthält.

3.3.2 Szenario: Neue Sensoren/Aktoren im Labor

Bob ist ein Student der Technischen Informatik und möchte gerne am Living Place Hamburg teilnehmen. Er hat sich während eines anderen Projekts mit einem Kontaktsensor

[9] http://subversion.apache.org/; abgerufen am: 05.12.2012
[10] http://www.eclipse.org/; abgerufen am: 05.12.2012
[11] http://netbeans.org/; abgerufen am: 05.12.2012

beschäftigt und schlägt vor, diesen an alle Schubladen und Türen im Labor anzubringen. Nachdem er mit dem Team die Umsetzung beschlossen hat und die Sensoren angebracht wurden, hat er die Kabel verlegt. Nun möchte *Bob* seine Softwareschnittstelle entwickeln. Er möchte wie andere Projekte auch die Kommunikationsschnittstelle nutzen, um seine Sensorik abfragbar zu machen und nutzt dazu die von *Alice* erstellte Revision 113 des ActiveMQ-Wrappers. *Bob* formuliert ein Nachrichtenformat und erstellt sein Programm als Java Anwendung. Er installiert diese auf dem Desktop eines Laborrechners und testet die Sensoren über die Kommunikationsschnittstelle. Eine Woche später wird eine Vorführung der neuen Projekte geplant und *Bob* freut sich schon darauf seine neuen Sensoren vorzustellen. Leider antwortet seine Anwendung nicht auf Anfragen und *Bob* fragt sich, wieso das nun der Fall ist, obwohl er doch alles getestet hat. Seine Software war leider nach einem Neustart für Updates des Laborrechners nicht mit gestartet worden und somit nicht aktiv.

3.3.3 Szenario: Neue Verhalten im Labor

Eine weitere Studentin, *Carol*, möchte sich gerne darum kümmern, dass die neue Sensorik von *Bob* auch zum Einsatz kommt und ein neues Verhalten abbilden. Ihre Idee sieht vor, die Lichter über geöffneten Schubladen einzuschalten, um den Probanden im Labor die Einsicht der Schublade zu erleichtern. Sie erkundigt sich bei *Bob* über die Dokumentation des Projekts und er verweist sie auf das Laborwiki, dort sind alle Nachrichtendetails dokumentiert. *Carol* arbeitet sich in das Nachrichtenformat des Kontaktsensors ein und schafft es schon bald die Nachrichten korrekt zu interpretieren. Nun will sie die richtigen Lichter, passend zu den Schubladen, einschalten. Dazu muss *Carol* sich in die Lichtsteuerung einlesen und das Nachrichtenformat in ihr Programm integrieren. Die erste Version funktioniert schon bald und sie probiert das neue Verhalten in der Praxis aus. Schon bald vergisst ein anderes Labormitglied eine Schublade zu schließen und das Licht bleibt, entsprechend der Logik in *Carols* Programm, eingeschaltet. *Carol* ist allerdings mittlerweile im Auslandssemester und nicht mehr ohne weiteres erreichbar. Ein anderes Labormitglied arbeitet sich in das Programm von *Carol* ein und erweitert es um eine Schnittstelle zur Positionserkennung. Für das Verwenden dieses Aktors muss wieder ein Interpreter für das Nachrichtenformat programmiert werden, denn dafür gibt es keine allgemeine Bibliothek. Die neue Version lässt das Licht nur zwei Minuten eingeschaltet nachdem eine Schublade geöffnet wurde und der Proband die Position verlassen hat.

Bewertung der Szenarien

Die hier vorgestellten Szenarien zeigen die Problemstellung in der aktuellen Infrastruktur und helfen den Anforderungskatalog für diese Arbeit zu entwerfen. Folgende Punkte werden müssen im Anforderungskatalog berücksichtigt werden:

- Im Idealfall sollten neue Teilnehmer im Labor eine Möglichkeit haben, sich die vorangegangenen Arbeiten zu nutze zu machen und auf diese aufzubauen. Damit das möglich wird, müssen einheitliche und wiederverwendbare Projektdefinitionen eingesetzt werden. Auch das Veröffentlichen einer neuen Version von einem Projekt muss möglich sein, ohne das man auf bestimmte Versionierungswerkzeuge oder Revisionen angewiesen ist.
- Wenn ein neuer Sensor oder Aktor in das Labor integriert werden soll, muss es eine Möglichkeit geben, diesen in die Infrastruktur einzugliedern, sodass die Komponenten des Labors wartbar in Betrieb genommen werden können. Eine Möglichkeit für die Einsicht aller Sensor- und Aktorstatus muss gegeben sein.
- Neues Verhalten muss in die Laborumgebung integriert werden können, ohne das die Software zum Kommunizieren mit den beteiligten Sensoren und Aktoren neu implementiert werden muss. Diese Integration soll durch eine geeignete Sprache zum Formulieren von Verhaltensweisen geschehen.

3.4 Softwareprojekte im Living Place Hamburg

An dieser Stelle wird der aktuelle Stand der Software- und Projektentwicklung im Labor untersucht und entsprechende Anforderungen ausgearbeitet. Die in Abschnitt 3.3 vorgestellten Szenarien werden auf Problemstellungen untersucht und dienen als Grundlage für die Formulierung dieser Anforderungen.

3.4.1 Projektdefinitionen

Mit dem Living Place Hamburg ist ein Labor entstanden, in dem Studenten ihre Projekte realisieren und Prototypen entwickeln können. Im Entstehungsprozess des Labors wird besonderer Wert darauf gelegt, dass der Einstieg in die vorhandene Infrastruktur einfach ist. Damit ist es möglich, kleinere Projekte schnell umzusetzen und wenig Zeit für Einarbeitung aufzuwenden. Das resultiert in einer freien Wahl der IDE und damit auch der Projektdefinition. Dadurch, dass die IDE benutzt wird, um eine IDE

spezifische Projektdefinition zu erstellen, ist das Kompilieren des Projekts nur mit der verwendeten IDE möglich. Das hat zur Folge, dass Projekte an IDEs gebunden sind und nicht mehr unabhängig kompiliert werden können. Des weiteren findet keinerlei Abhängigkeitsauflösung statt.

Alle Projekte, die erstellt werden, auch solche die von mehreren anderen verwendet werden, sind in Binärform(als Java ARchive (JAR)-Datei) an andere Studenten ausgeliefert. Wenn eine Bibliothek wie zum Beispiel der ActiveMQ-Wrapper, die Bibliothek zur Kommunikationsabstraktion und Nachrichtenpersistierung, von einem Projekt verwendet wird, ist die gängige Praxis das Herunterladen und anschließende Einbinden der JAR-Datei. Durch diese Praxis ist es zum einen undurchsichtig, welche Version einer Bibliothek ein Projekt verwendet, zum anderen aber auch, ob es möglich ist, diese zu aktualisieren. Eine automatische Abhängigkeitsauflösung kann auch nicht statt finden, denn es gibt keine Möglichkeit, die IDE spezifischen Projektdefinitionen von einem Mechanismus auslesen zu lassen. Damit kann auch kein topologisch sortierter Graph erstellt werden und das verhindert die automatische Abhängigkeitsauflösung, vgl. Abschnitt 2.2.

Eine zentrale Anforderung ist die Einführung einer einheitlichen Projektdefinition, durch die IDE Unabhängigkeit und automatisierte Abhängigkeitsauflösung erreicht werden. Es soll eine Möglichkeit geschaffen werden, Abhängigkeiten an Bibliotheken, die im Rahmen von Projekten im Labor entstanden sind, wie zum Beispiel dem ActiveMQ-Wrapper, aufzulösen.

3.4.2 Life Cycle Management

Im Laborbetrieb werden neue Projekte von Studenten umgesetzt und in die vorhandene Infrastruktur eingegliedert. Bisher gibt es keine Vorgaben zur Integration einer Abfrage des Laufzeitzustands von Softwareprojekten. Das führt dazu, dass wenn eine Komponente fertig gestellt wird und ein Student die Hochschule verlässt, diese Komponente nicht mehr betreut wird. Sollte sich die Laborumgebung ändern, sei es durch die Aktualisierung anderer Komponenten oder Teilen der Infrastruktur, lässt sich die Softwarekomponente meist nicht mehr verwenden. Es kann auch ein Fehler in einer Komponente auftreten, sodass diese Abstürzt oder sich im Fehlerzustand befindet.

Das Problem ist aber nicht der Fehlerzustand selbst, den man eventuell sogar mit einem Neustart der Komponente beheben kann, sondern das dieser niemandem auffällt. Es gibt keinen zentralen Punkt in der Laborinfrastruktur indem die Zustände aller

Komponenten gesammelt zur Verfügung stehen. Teilweise basieren die Komponenten aufeinander, sodass die Fehlersuche eventuell komplex und zeitaufwendig ist.

Für den reibungslosen und zuverlässigen Laborbetrieb, soll eine Möglichkeit geschaffen werden, den Laufzeitzustand über die Softwarekomponenten der Laborinfrastruktur zu gewährleisten. Im Kontext dieser Arbeit soll also eine Möglichkeit entstehen diese Zustände zu erfassen.

3.4.3 Sensor- und Aktorintegration

Das Labor enthält schon heute viele Sensoren und Aktoren, die komplexe Informationen und Steuermöglichkeiten anbieten, vgl. Abschnitt 3.1.1 und 3.1.2. Die bereits vorhandenen Komponenten bieten ihre Informationen und Aktionen über die Kommunikationsschnittstelle an, vgl. 3.2. Über Topics und Queues kann man das Ausführen von Aktionen und Erfragen von Informationen in Auftrag geben. Alle Informationen, die man in seinem Projekt verwenden will, müssen manuell erfragt und in den Programmcode integriert werden. Auch Aktionen, die eventuell in Folge der Informationen ausgeführt werden sollen, müssen manuell integriert werden und stehen an keiner Stelle gesammelt zur Verfügung. Der zentrale Mittelpunkt im System, die Kommunikationsinfrastruktur, kann diese Aufgabe nicht übernehmen, denn es gibt keine Möglichkeit, die Nachrichtenformate einheitlich auszulesen, vgl. Otto und Voskuhl (2010). Wenn neue Sensoren oder Aktoren zum Labor hinzugefügt werden, ist es nicht möglich, diese zu integrieren, ohne den Programmcode in Projekten, die vor der Integration entstanden sind, modifizieren zu müssen. Das bedeutet zum Beispiel, das jedes Programm das die Lichtsteuerung verwendet um Lampen im Labor zu steuern, diese Logik neu implementiert. Es gibt also keine Möglichkeit, eine Funktionalität oder einen reaktiven Mechanismus im Labor um eine Information oder Aktion zu erweitern, ohne das Programm zu ändern, was diese abbildet.

Deshalb soll eine Möglichkeit geschaffen werden, neue Aktoren und Sensoren in den Living Place Hamburg einfach und standardisiert zu integrieren und vor allem diese auch verwendbar zu machen. Diese Anforderung soll die Möglichkeit bieten, neue Funktionen hinzuzufügen, ohne den Quelltext der zuvor erstellten Programme ändern zu müssen. Wenn eine Integration statt gefunden hat, sollte diese wiederverwendbar sein.

3.5 Vergleichbare Arbeiten

Dieser Abschnitt befasst sich mit Arbeiten, die in ähnlichem Kontext oder unter ähnlicher Problemstellung verfasst wurden. Es werden einige Arbeiten genauer untersucht, um eventuell Ansätze weiter zu verfolgen oder mit in das Lösungskonzept dieser Arbeit zu integrieren. Untersucht werden Arbeiten, die im Umfeld von Smart Homes und AAL durchgeführt wurden und eventuelle Lösungsansätze für die Problemstellung im Living Place Hamburg bieten.

Die hier ausgewählten Arbeiten sind nur ein Teil der Forschungsprojekte, die sich mit einer Architektur für das Smart Home beschäftigen. Die Projekte AMbient IntelliGence fOr the networked home environment (AMIGO) und PERceptive Spaces prOmoting iNdependent Aging (PERSONA) sind allerdings interessant, weil sie von Forschungsgruppen aus Wirtschaft und Universitäten kooperativ entwickelt wurden[12] und damit dem Laborumfeld dieser Arbeit am nächsten kommen. Das dritte vorzustellende Projekt, open Home Automation Bus (openHAB), ist ein Open Source Projekt, das aktuell weiter entwickelt wird und einen Ansatz zur Eigenheimautomatisierung verfolgt, der besonderen Wert auf praktische Anwendbarkeit legt.

3.5.1 AMIGO

Das AMIGO Projekt ist ein von der Europäische Union (EU) geleitetes Projekt, welches sich mit dem Entwurf einer Architektur zur Heimautomatisierung auseinander setzt. Ziel bei diesem Projekt ist die Integration der Geräte, die Möglichkeiten zur vernetzten Kommunikation bieten, in ein Netzwerk auf dem die Funktionalitäten zusammengeführt werden können. Die integrierten Geräte werden als Services in eine OSGi basierte Infrastruktur eingebunden und können dort verwendet werden. Das System ist in drei heterogene Schichten, Anwendungs-, Dienst- und Plattformschicht, eingeteilt. Als besondere Merkmale der Architektur werden erweiterte Servicebeschreibungen auf Anwendungsschicht, Servicekommunikation und -auffindung, sowie Komposition und vor allem Sicherheit in der Dienstschicht und Quality of Service (QOS) Unterstützung, sowie Ressourcen- und Fähigkeitsbewusstsein der Geräte in der Plattformschicht aufgeführt.

Die Anwendungsschicht kombiniert verschiedene Informationsquellen und verbindet den Benutzer mit dem System. Sie gibt Kontextinformationen und macht musterbasierte Vorhersagen.

[12] http://www.hitech-projects.com/euprojects/amigo/amigo.htm; abgerufen am: 19.02.2013

AMIGO macht sich in der Dienstschicht semantische[13] Mechanismen zur Servicefindung und -beschreibung zu nutze und verwendet ein flexibles Modell zur Integration von Geräten in das System. Auch vorhandene Protokolle wie UPnP können in die Infrastruktur integriert werden. Ein Teilaspekt in diesem Projekt ist die Programmiersprachen-Unabhängigkeit, denn es werden ausdrücklich verschiedene, Java und C#, als mögliche Varianten genannt[14].

In der Plattformschicht werden Auslieferungsmechanismen und Programmiermodelle als Infrastruktur bereit gestellt, mit der sich neue Versionen der Software automatisiert veröffentlichen lassen, vgl. Georgantas u. a. (2005).

Das AMIGO Projekt wird seit 2008 nicht mehr von der EU gefördert und ist seit dem im Entwicklungsstillstand. Der finale Bericht über den Projektausgang schildert im Ausblick die Zusammenarbeit in einem Team mit verteilten Mitarbeitern. Eine gute Infrastruktur zur gemeinsamen Entwicklung eines über Jahre hinweg geförderten Projekts ist ein Schlüsselpunkt in der Realisierung, vgl. Janse (2008)

Bewertung

Das AMIGO Projekt bildet eine solide Referenzarchitektur für eine Smart Home Plattform auf Basis von OSGi. Die semantischen Ansätze zur Vereinheitlichung der Geräte im vernetzten Heim und die dazugehörigen Sicherheitskonzepte zeichnen dieses Projekt aus. Die Modularisierung der Dienstschicht ist für eine flexible und offene Gestaltung einer Softwarearchitektur relevant und kann verwendet werden. Teilkomponenten wie zum Beispiel das Context Management System oder der Awareness and Notification Service gehen über den Rahmen dieser Arbeit hinaus und sind damit nicht relevant.

3.5.2 PERSONA

Das PERSONA Projekt ist ein von der EU gefördertes Projekt zur Untersuchung von AAL für ältere Menschen. Ziel des Projekts ist eine einheitliche Architektur für den Heimeinsatz zu finden und die Kosten dabei auf ein Minimum zu beschränken, um eine solche Entwicklung serientauglich zu machen. PERSONA bildet einen weiteren Ansatz für eine Architektur im Bereich AAL, geht aber einen Schritt weiter und zielt auf einen Standard ab. Die Architektur bedient sich der Mechanismen von AMIGO und erweitert diese mit R-OSGi[15] um Dezentralisierung, vgl. Wu u. a. (2008). R-OSGi ist eine

[13] http://www.w3.org/2001/sw/; abgerufen am: 12.02.2013
[14] http://www.hitech-projects.com/euprojects/amigo/software.htm; abgerufen am: 13.02.2013
[15] http://r-osgi.sourceforge.net/; abgerufen am: 13.02.2013

Erweiterung des OSGi Frameworks, um ein Bundle das es ermöglicht zwischen Services zu kommunizieren, die nicht in der gleichen Laufzeitumgebung ausgeführt werden[16]. Die Verteilung der Services ist für Bundles selbst nicht unterscheidbar und sie können bis auf die Initialisierung weiterhin nach den OSGi üblichen Modularisierungsprinzipien entworfen werden. Die Vorteile des verteilten Ansatzes sind durch Skalierbarkeit und die Möglichkeit der Auslagerung gegeben. PERSONA unterteilt die Architektur in drei Schichten, den *Abstract Connection Layer (ACL)*, den *SodaPop Layer* und den *PERSONA specific Layer*, vgl. Abb. 3.5.

Die niedrigste Schicht, der ACL, stellt die Verbindung der Middlewareinstanzen über ZigBee, UPnP, R-OSGi oder Bluetooth sicher. Eine einheitliche Schnittstelle bildet die Möglichkeit mit verschiedenen Protokollen über diese Schicht zu kommunizieren. Über Listener kann auf erscheinende Dienste Interesse angemeldet werden, um bei ihrer Aktivierung zu reagieren. Der Sensor Abstraction and Integration Layer (SAIL) ist zur Kommunikation mit der ZigBee Sensorik integriert. Dieser wird auch durch eine Drei-Schichten-Architektur realisiert und bietet Möglichkeiten zu Erweiterung, vgl. Fabbricatore u. a. (2011).

Der SodaPop Layer registriert die Kommunikationspartner aus dem ACL lokal und macht diese damit der obersten Schicht verfügbar. In dieser Schicht wird die Kommunikation zwischen Instanzen der Middleware auf Busse(eventbasiert oder aufrufbasiert) und Busstrategien abstrahiert.

Die PERSONA spezifische Schicht ist für die Ein- und Ausgaben, den Kontext und die Servicebusse zuständig. Das Buskonzept ist aufgeteilt in *Context Bus* und *Service Bus*. Der Context Bus ist für die Kommunikation zuständig und wird genutzt um Nachrichten und Ereignisse zu lokalen oder entfernten Endpunkten zu transportieren. Kontext veröffentlichende Bundles müssen den OSGi Service finden und sich bei ihm anmelden um Kontextinformationen publizieren zu können. Sie registrieren sich mit einem Web Ontology Language (OWL) Profil, auf dem die aufrufenden Services Abfragen machen können. Kontextinteressierte Bundles müssen zusätzlich noch einen Filter mit angeben um sich für bestimmte Kontextevents anzumelden. Der Service Bus findet die entsprechenden Services anhand von seinem Service Profil Repository und leitet die Anfragen an diese weiter. Im Anschluss sammelt er die Antworten und übergibt sie dem Aufrufenden, vgl. (Nakashima u. a., 2010, S. 1187 f).

[16] `http://r-osgi.sourceforge.net/`; abgerufen am: 10.04.2013

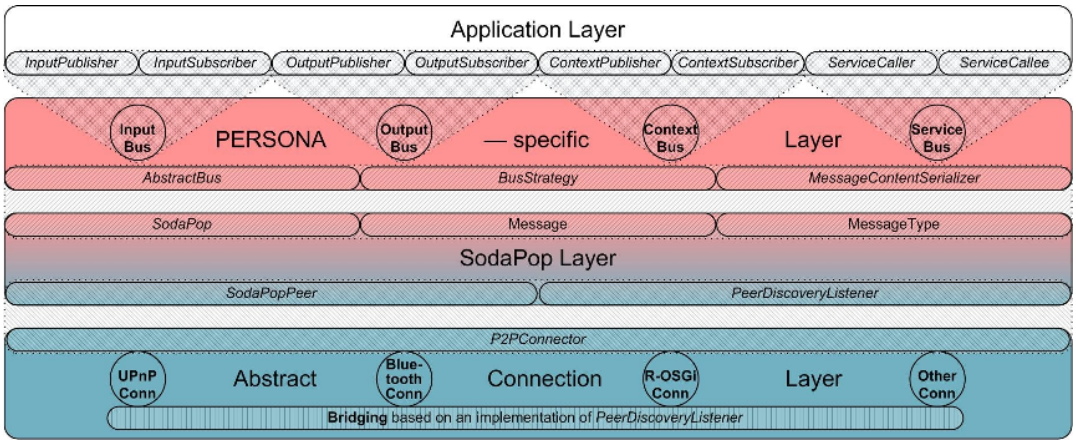

Abbildung 3.5: PERSONA Architektur, Quelle: Nakashima u. a. (2010)

Bewertung

Das PERSONA Projekt erweitert die Ansätze von AMIGO mit einer verteilten Architektur sowie einem `Context-` und `Service Bus`. Über diese Architekturerweiterungen werden verteilte Abfragen möglich, allerdings auch komplexere Kommunikationsstrukturen eingeführt. Ein Schwerpunkt dieser Arbeit ist die einfache Gestaltung einer Lösung, in die auch zukünftig mit möglichst wenig Aufwand eingegriffen werden kann. Ein guter Ansatz ist die Verwendung von R-OSGi. Über den Einsatz von R-OSGi können mit wenig Verteilungsaufwand Bundles in unterschiedlichen Frameworks und Computern ausgeführt werden. Die Dezentralisierung ist für diese Arbeit kein Schwerpunkt und steht daher für zukünftige Arbeiten offen.

3.5.3 openHAB

Das openHAB Projekt ist ein Open Source Projekt, das sich seit 2010 mit Heimautomatisierung auseinandersetzt. Das Projekt ist eine reine Java Implementation und komplett OSGi basiert. openHAB setzt dabei auf die Eclipse Equinox OSGi Runtime und dessen Rich Client Platform (RCP).

Das Software Design ist darauf ausgelegt, von Hersteller-, Hardware- und Protokoll unabhängig zu sein. Umgesetzt wird diese Unabhängigkeit über eine Kernkomponente(openHAB Core), die Basisbibliothek(openHAB Base Library), das Reposito-

ry(openHAB Repository) und den REpresentational State Transfer (REST) Service (openHAB REST Service), vgl. Abb. 3.6. Das Konzept zur Umsetzung einer Anbindung wird in der openHAB Terminologie `Binding` genannt. Zur Integration von Protokollen, spezifischer Aktorik oder Sensorik werden neue Bindings verfasst und in die Registrierung aufgenommen. Ein Binding kann Kommandos entgegen nehmen oder Informationen liefern. Die Kommunikation zwischen der Automatisierungslogik und den Bindings wird über die Registrierung umgesetzt, sie verbindet die Bindings über den Event Bus(openHAB Event Bus).

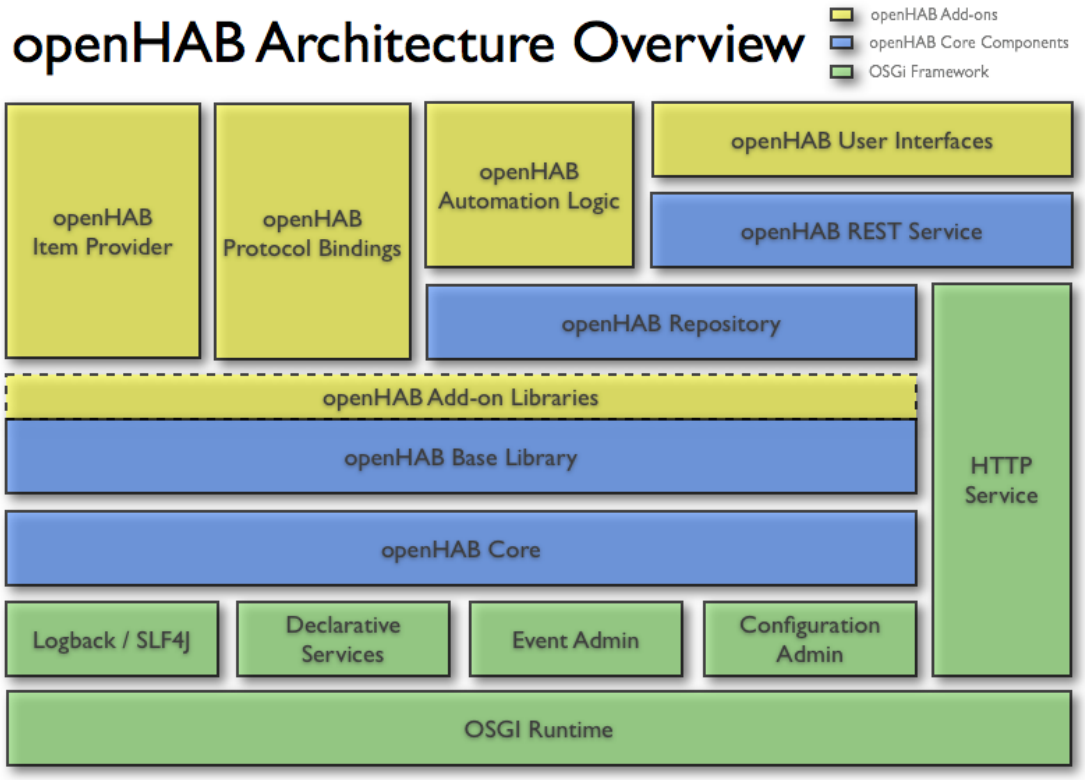

Abbildung 3.6: openHAB Architektur, Quelle: Kreuzer (2012)

Ein Aspekt, der bei openHAB im Vordergrund steht, ist die einfache Bedienung und Nutzerunterstützung. Der openHAB Designer bietet eine Eclipse Umgebung, mit der das Erstellen und Verwalten von Bindings und Items sowie Regeln vereinfacht wird. Des weiteren kann für openHAB mit Hilfe der bereitgestellten Werkzeuge eine grafische Benutzeroberfläche für eine Website, iOS oder Android erstellt werden. openHAB unterstützt unterschiedliche Mechanismen zur Persistierung der Daten. Dabei kann

zwischen verschiedenen Datenbanken gewählt werden die gleichzeitig existieren können. Als regelbasierte Maschine ist bis Version 0.9.0 die JBoss Entwicklung Drools Expert zum Einsatz gekommen. Seitdem ist diese optional als Addon verfügbar und wird durch eine Eigenentwicklung ersetzt, vgl. Kreuzer (2012). Eine mit Xtext[17] entwickelte Regelsprache vereinfacht das Verfassen von Regeln durch diese Standardsprache und bietet trotzdem die Möglichkeit eine erweiterte Lösung mittels Drools Expert zu installieren.

Bewertung

Das openHAB Projekt ist ein offenes Projekt zur Integration von Automatisierungsmechanismen in eine intelligente Wohnumgebung. Es ist offen für die Entwicklung von Bindings verschiedener Protokolle und Schnittstellen. Die Unterstützung der Benutzer mit einer Entwicklungsumgebung zum Verfassen der Regeln ist hilfreich, die Integration in die vorhandene Laborinfrastruktur ist dennoch nicht Bestandteil dieser Arbeit. Des weiteren ist das Projekt auf die Eclipse Equinox Plattform angewiesen.

3.5.4 Abschließende Bewertung der vergleicharen Arbeiten

Die hier untersuchten Arbeiten befassen sich mit unterschiedlichen Aspekten des Smart Home Umfeldes. Keines der untersuchten Projekte bietet eine vollständige Lösung für alle Anforderungen dieser Arbeit. Dennoch könnten Teilkonzepte übernommen und weiter ausgearbeitet werden um den Anforderungen gerecht zu werden. Alle drei Arbeiten werden mit OSGi umgesetzt und nutzen dessen Vorteile. Keine der vorgestellten Arbeiten kann komplett übernommen werden, da die Schwerpunkte dieser Arbeit entweder nicht angesprochen werden, oder anders geartet sind.

Die Vereinheitlichung der Sensorik und Aktorik ist in allen drei vorgestellten Projekten ein Schwerpunkt. Hier sollte der einfachste Ansatz gewählt werden, um die Komplexität zur Umsetzung gering zu halten und die Erweiterbarkeit für eventuelle Nachfolger zu gewährleisten. Besonders der Ansatz des SAIL in PERSONA ist interessant für eine Umsetzung der Sensor- und eventuell Aktorabstraktion in dieser Arbeit.

Ein zentraler Punkt zur Einsicht aller Komponenten im System ist vor allem im Projekt openHAB mit den verschiedenen Benutzerschnittstellen flexibel gelöst und sollte als Vorlage zur Realisierung dienen. Die Verwirklichung einer Abhängigkeitsauflösung

[17] http://www.eclipse.org/Xtext/; abgerufen am: 18.02.2013

wird im openHAB Projekt genutzt, mit Tycho[18] und p2[19] realisiert, ist damit allerdings auch an die Eclipse Equinox Plattform gebunden.

3.6 Ergebnis der Anforderungsanalyse und Abgrenzung

Die Analyse hat einige Anforderungen für diese Arbeit definiert die nicht alle gleich hoch zu priorisieren sind. Anschließend wird ein Anforderungskatalog erstellt in dem die verschiedenen Ebenen der Problemstellung verdeutlicht werden. Den Schluss des Analysekapitels bildet die Abgrenzung.

3.6.1 Anforderungskatalog

Die Anforderungen sind in die Kategorien infrastrukturell, Projektdefinitionen und Implementation unterteilt. Die Priorität ist jeweils in Reihenfolge abnehmend sortiert.

Infrastrukturelle Anforderungen

Die folgenden Anforderungen ergeben sich im Kontext der Umstellungen die vorgenommen werden soll, sie sind nicht explizit zu realisieren, sondern auf sie muss bei der Realisierung Rücksicht genommen werden:

- Die infrastrukturelle Anforderung mit höchster Priorität muss gewährleisten, dass die vorhandene Laborinfrastruktur mit allen Aktoren und Sensoren, die heute in den Living Place Hamburg integriert sind, erhalten und einsatzbereit bleibt. Die Komponenten können dann iterativ übertragen werden ohne den Laborbetrieb zu stören.
- Eine Anforderung an die Kommunikationsinfrastruktur ist, dass sie auch weiterhin erhalten bleibt, um mit dem Ergebnis dieser Arbeit nicht bereits vorhandene Projekte zu stören. Es muss also eine Möglichkeit für *Alice* geben, die vorhandene Kommunikationsinfrastruktur zu erhalten und trotzdem um weitere Funktionen einer einheitlichen Kommunikation zu erweitern. Die Art der Vereinheitlichung sollte dabei möglichst plattformunabhängig sein, sodass auch Programme, die nicht mit Java erstellt werden, von dieser Erweiterung profitieren können.

[18] http://eclipse.org/tycho/; abgerufen am: 19.02.2013
[19] http://www.eclipse.org/equinox/p2/; abgerufen am: 19.02.2013

- Die letzte infrastrukturelle Anforderung ist die Zustandsübersicht. Um den Zustand im Labor einheitlich erfassen zu können, sollte es eine Möglichkeit geben, den aktuellen Status der Projekte zu erfragen oder anzeigen zu lassen. Für einen Überblick über die aktiven Komponenten im Labor soll dazu ein zentraler Punkt entstehen, an dem die Projekte auf Zustand und Funktion untersucht werden können. Es muss für Studenten in der Lage von *Bob* die Möglichkeit geben, den Status von Komponenten in die Gesamtübersicht zu integrieren, sodass ein Überblick erhalten wird, auch wenn ein Projekt neu hinzu kommt.

Anforderungen an die Projektdefinitionen

Die Projektdefinitionen sollen vereinheitlicht werden damit es möglich wird die Softwarekomponenten im Labor wiederverwendbar und testbar zu machen. Folgende Anforderung ist dafür auf zu stellen:

- Eine Anforderung ist die Vereinheitlichung der Projektdefinitionen für Softwarekomponenten. Die Softwareentwicklung im Kontext des Living Place Hamburg sollte in einen klar strukturierten Rahmen gebracht werden. Es soll eine Möglichkeit geschaffen werden, Abhängigkeiten automatisch aufzulösen um damit auch die Zusammenarbeit von Projekten zu vereinfachen. *Bob* muss die Möglichkeit haben, die Bibliothek von *Alice* zu verwenden, ohne eine bestimmte Revision aus der Versionierung zu holen und diese dann mit einer bestimmten IDE zu übersetzen. Die Verteilung von bestimmten Versionen einer Bibliothek soll mit einer einheitlichen Projektdefinition vereinfacht und automatisierbar werden.

Anforderungen an die Implementation dieser Arbeit

Die Implementation der Basisstrukturen, die als praktischer Teil dieser Arbeit realisiert werden, müssen unabhängig von vorhandener Infrastruktur bleiben. Die folgenden Anforderungen sollen realisiert werden:

- Eine Anforderung der Realisierung ist die Vereinfachung der Sensor- und Aktorintegration. Dazu muss es eine Möglichkeit geben die vorhandenen Sensoren und Aktoren zu verwalten und programmatisch verfügbar zu machen. Studenten, die wie *Carol* ein neues Verhalten in das Labor integrieren wollen, sollte es ermöglicht werden dabei auf die Schnittstellen und Nachrichtenformate anderer Sensoren oder Aktoren zuzugreifen, ohne diese immer neu implementieren zu müssen.

- Eine weitere Anforderung ist die zentrale Zusammenführung der Sensordaten. Durch eine zentrale Stelle im Labor, an der alle Informationen für eine fundierte Aussage über den Zustand der Wohnung zur Verfügung stehen, wird die Realisierung von komplexem Verhalten ermöglicht. Mit dem Wissen über alle Sensordaten können Entscheidungen im Zusammenhang getroffen und anschließend Aktion ausgeführt werden.

- Die letzte Anforderung ist die Integration regelbasierter Verfahren zur vereinfachten Verhaltensintegration. Eine CEP-Maschine soll in die Laborumgebung integriert werden, sodass es möglich wird die Sensorik und Aktorik aus Regeln zu nutzen.

Die vergleichbaren Arbeiten, die in diesem Zusammenhang betrachtet wurden enthalten Konzepte um Teile des Anforderungskatalogs umzusetzen. In allen drei Arbeiten wird OSGi als Application Container verwendet, vgl. Abschnitt 2.3 und 3.5.4.

3.6.2 Abgrenzung

In diesem Abschnitt wird zusammen gefasst welche Problemen diese Arbeit offen lässt und nicht behandelt.

- Die in dieser Arbeit verwendete Programmiersprache ist, sowie auch in den meisten bereits vorhandenen Softwarekomponenten, Java. Es wird eine Lösung für diese Programmiersprache angestrebt. Ziel ist jedoch nicht der Ausschluss anderer Sprachen, sodass mit einem Adapter die Möglichkeit zur Teilnahme an der entwickelten Infrastruktur bleibt.

- Die vorhandene Sensorik und Aktorik kann, neben der in dieser Arbeit vorgestellten Lösung, parallel betrieben werden. Die Infrastrukturerweiterungen dieser Arbeit werden von den betroffenen Projekten dann nicht verwendet. Eine Überführung der alten Softwarekomponenten in die neue Infrastruktur ist nicht Teil dieser Arbeit.

- Die bisherige Infrastruktur bietet keine Möglichkeit zum automatisierten Testen der Softwarekomponenten. Die Integration eines Testframeworks zum automatisierten Testen ist nicht vorgesehen.

- Die Integration eines Continuous Integration Services sieht diese Arbeit nicht vor. Dieser könnte in Zukunft der automatischen Auslieferung von Komponenten vorangestellt werden um die Softwarequalität zu heben.

4 Design

In diesem Abschnitt der Arbeit werden Lösungskonzepte für die Anforderungen aus Abschnitt 3.6 in der aktuellen Laborumgebung erarbeitet. Diese Lösungskonzepte werden detailliert erläutert und beschrieben, um im Anschluss die konkrete Implementation darzustellen zu können.

Das Konzept dieser Arbeit umfasst verschiedene Ebenen des Entwicklungsprozesses und wird daher an dieser Stelle erläutert, um den Überblick vor der Behandlung der Einzelthemen zu geben, vgl. 3.6.1.

Zur Verbesserung des Entwicklungsrahmens wird eine einheitliche Projektdefinition vorausgesetzt. Die IDE unabhängige Projektdefinition ist eine Voraussetzung für Plattform- und Projektübergreifende Abhängigkeitsauflösung. Wie die Anforderungen an diese gewährleistet werden, wird im folgenden Abschnitt erläutert, vgl. 4.1.

Unabhängig von der Abhängigkeitsauflösung ist die Anforderungen zur Vereinfachung der Aktor- und Sensorintegration zu betrachten. Diese Anforderung lässt sich mit den Anforderungen zur zentralen Übersicht für Softwarekomponenten und Informationen der Sensoren kombinieren. Durch Einführung eines einheitlichen Laufzeitcontainers wird diese zentrale Komponente entstehen. Sie ermöglicht, wie in Abschnitt 4.2 detailliert erläutert, einen globalen Zustandsüberblick der Softwarekomponenten. In diesem Zuge vereinfacht sich auch die Zusammenführung der Sensorik und Aktorik an einer zentralen Stelle für die Implementation. Durch diese Designentscheidung lassen sich die genannten Anforderungen kombinieren.

Die Integration einer CEP-Maschine steht nicht im Zusammenhang mit den vorangegangenen Anforderungen. Dennoch wird die Implementation von der Einführung einer einheitlichen Projektdefinition und Laufzeitumgebung profitieren, vgl. 4.4. Welche CEP-Maschine zum Einsatz kommt und welche Eigenschaften diese mit sich bringt wir in Abschnitt 4.3 erläutert.

4.1 Apache Maven zur Projektdefinition und Dependency Resolution

Eine einheitliche Projektdefinition setzt die Wahl eines Werkzeugs voraus das mit dieser automatisiert Arbeiten kann. Es gibt verschiedene Alternativen die für Java zum erstellen des Übersetzens genutzt werden können:

Ant Das Apache Ant Projekt ist eine Java-Bibliothek und Kommandozeilen-Werkzeug, das verwendet wird um Programme zu übersetzen. Ant wurde für Java entwickelt, kann aber genutzt werden, um beliebige Quelltexte zu übersetzen, solange sich diese über Ziele und Aufgaben definieren lassen. Wenn man die Übersetzung des Quelltextes mit Abhängigkeitsauflösung kombinieren möchte, kann Ant mit Ivy kombiniert werden[1].

Ivy Das Apache Ivy Projekt ist in Kombination mit Apache Ant zu verwenden und erweitert Ant um Dependency Resolution. Beide Projekte bilden nur einen Teil der Anforderungen ab, könnten aber zusammen als Lösung verwendet werden[2].

Gradle Gradle ist ein Werkzeug zur automatischen Übersetzung von Java Projekten. Mit Groovy als Skriptsprache wird eine Domain Specific Language (DSL) verwendet um Übersetzungsskripte zu entwickeln die Vorteile von Maven und Ant kombinieren[3].

Die vorgestellten Werkzeuge zur Verwaltung von Projekten und deren Abhängigkeiten zeichnen sich durch unterschiedliche Eigenschaften und Stärken aus. Ant und Ivy ließen sich im Labor einsetzen, sind aber dem Autor unbekannt und erfordern Einarbeitungsaufwand. Besonders Gradle ist ein innovatives Werkzeug zur Automatisierung von Übersetzungsprozessen und Aufgaben die in der Softwareentwicklung auftreten. Allerdings müsste Groovy erlernt werden um dieses Werkzeug in vollem Umfang nutzen zu können.

Die Wahl des Werkzeugs für diese Arbeit ist auf Apache Maven gefallen, weil Maven dem Autor bereits bekannt ist und einzelne Projekte, wie zum Beispiel der ActiveMQ-Wrapper, bereits damit umgesetzt wurden. Somit kann die Projektdefinition weiter geführt und der ActiveMQ-Wrapper als erste Bibliothek referenziert werden.

[1] http://ant.apache.org/; abgerufen am: 11.04.2013
[2] http://ant.apache.org/ivy/; abgerufen am: 25.02.2013
[3] http://www.gradle.org/; abgerufen am: 25.02.2013

Maven wird dabei die komplette Abhängigkeitsauflösung zur Verfügung stellen und projektübergreifend Bibliotheken wiederverwendbar machen. Des weiteren wird Maven eingesetzt um eine einheitliche Projektdefinition vorzugeben. Außerdem bietet Maven die Möglichkeit Maven-Plugins zu nutzen, die eine Projektdefinition erleichtern. Diese Plugins sind Java-Bibliotheken die, in der Projektdefinition angegeben, den Übersetzungsprozess erweitern können. Weitere Details zu den verwendeten Plugins und der Laborintegration sowie konkrete Verwendung werden in Abschnitt 5.1 und 5.2 erläutert.

Im folgenden Abschnitt wird Maven und dessen Funktionsweise detailliert erläutert. Von der Maven Projektseite[4] wird das Werkzeug wie folgt beschrieben:

> „Maven is a software project management and comprehension tool. Based on the concept of a Project Object Model (POM), Maven can manage a project's build, reporting and documentation from a central place."

Maven hilft bei der Entwicklung von Java Programmen und ist von IDE und Betriebssystem unabhängig. Das Werkzeug stützt sich auf eine projektspezifische POM-Datei, eine XML-Datei die ein Projekt Maven lesbar beschreibt. Diese Datei bezeichnet eine JAR-Datei so eindeutig, dass diese, trotz binärer Form, eindeutig identifizierbar ausgeliefert werden kann. In der Maven Terminologie wird diese Kombination als Artifakt beschrieben.

Zur Beschreibung eines Projekts gehört vor allem eine Möglichkeit das Projekt zu identifizieren und von anderen Projekten zu differenzieren. Maven identifiziert Artifakte anhand von folgenden drei Attributen:

groupId Die *groupId* wird verwendet um einen Kontext zu kennzeichnen in dem sich alle Module befinden. Im Fall vom Living Place Hamburg wurde hier folgender Namensraum gewählt:

```
<groupId>org.livingplace.unterprojekt</groupId>
```

Listing 4.1: groupId des Living Place Hamburg

Diese Namensgebung ist durch die Domain, auf der die Internetpräsenz des Labors zu finden ist, vorgegeben und wird durch den Kontext des Projekts erweitert.

artifactId Die *artifactId* beschreibt die Bibliothek in dem Kontext der *groupId*. Sie wird genutzt, um die Funktion von anderen innerhalb der Gruppe zu trennen. Hier gibt

[4] http://maven.apache.org/; abgerufen am: 05.12.2012

es zwei Namensschemata die beide genutzt werden. Im Living Place können beide eingesetzt werden. Ein Schema sieht im Artifaktbezeichner die komplette *groupId* als Präfix vor, das andere verbindet Bezeichner mit Bindestrichen und bildet so die *artifactId*.

```
<artifactId>
    org.livingplace.unterprojekt.spezifische.bibliothek
</artifactId>
<artifactId>
    unterprojekt-spezifische-bibliothek
</artifactId>
```

Listing 4.2: Beispiel artifactId im Living Place Hamburg

version Die Version des Projekts. Versionen sind besonders wichtig, denn sie machen eine Aussage über Zustand, Funktionsumfang und Stabilität des Projekts. Maven unterscheidet Versionen über vier Teile der vollständigen Versionsnummer, der Hauptversion (*major*), Unterversion (*minor*) und der inkrementellen Version (*incremental*). Um diese dreistellige Version mit Meilensteinen versehen zu können, wird sie um einen qualifizierenden Anteil (*qualifier*) erweitert, vgl. Listing 4.3.

```
<major>.<minor>.<incremental>-<qualifier>
```

Listing 4.3: Versionsbestandteile einer Maven Artifaktversion

Geht man zum Beispiel von einer Hauptversion 3, Unterversion 1 und einer inkrementellen Version 11 aus, würde sich für einen *Release Candidate* eins die Version aus Listing 4.4 ergeben.

```
<version>3.1.11-RC1</version>
```

Listing 4.4: Beispielversionen für Maven POMs

Versionen sind besonders wichtig, weil Maven es erlaubt, Versionsbereiche in Abhängigkeiten zu verwenden. Typischerweise ändert sich die API eines Programms nicht innerhalb einer *minor* Version, sodass sich bei inkrementell fortsetzender Versionsnummer nur Implementationsdetails ändern, um zum Beispiel Fehler zu korrigieren.

Versionsbereiche in einer Abhängigkeitsbeschreibung werden mit normalen und geschweiften Klammern bezeichnet. Einfache Klammern beschreiben ein offenes

Intervall, also eines, in dem die beschriebene Version nicht vorkommt. Geschweifte Klammern beschreiben ein geschlossenes Intervall. Wird am unteren Ende des Intervalls keine Version spezifiziert, ist damit gemeint, dass alle Versionen bis zum höheren Ende des Intervalls gültig sind. Beispielsweise zeigt Listing 4.5 einen Versionsbereich der alle Versionen von einschließlich 2.3.1 bis exklusiv 5.0 umfasst.

```
1 <version>[2.3.1,5.0)</version>
```

Listing 4.5: Versionsbereich einer Abhängigkeit

Diese Eigenschaften für eine POM-Datei sind minimal, aber sie kann weit mehr Eigenschaften enthalten als die Beschreibung des Projekts selbst, vgl. Anhang 1.1. Unter anderem werden dort Eigenschaften im Bezug auf die Versionsverwaltung und Plugins definiert. Die Maven Lifecycle Phasen werden von Maven von vorne nach hinten durchlaufen, können aber auch einzeln angesprochen werden, vgl. Tabelle 1.

Die POM-Datei beschreibt auch die Abhängigkeiten des Projekts, aber um sie auflösen zu können muss Maven die Artifakte kennen, die es auflösen soll. Verwaltet werden die Artifakte dazu in Archiven(*Repositories*), in denen gesucht werden kann. Repositories bilden eine Baumstruktur, in der die Artifakte und die dazugehörigen POM-Dateien nach *groupId*, *artifactId* und Version gegliedert vorliegen. Maven pflegt ein lokales Repository um nicht immer alle mehrmals verwendeten Artifakte neu herunterladen zu müssen. Gesucht wird immer zuerst im lokalen Repository und dann im Hauptrepository(*Maven Central Repository*)[5]. Das Maven Central Repository ist das öffentliche Verzeichnis der Apache Software Foundation[6] in dem nahezu alle Open Source Java Bibliotheken zu finden sind. Bei einer Standardinstallation von Maven auf einem Entwicklerrechner werden dort die Abhängigkeiten, die man für das Projekt spezifiziert, gesucht. Das Maven Central Repository ist allerdings nicht für alle Projekte geeignet. Zum Beispiel sollte man Bibliotheken, die nur laborintern benötigt werden oder nur im Kontext vom Living Place Hamburg Sinn machen, nicht im Maven Central Repository ausliefern, sondern über einen sog. Maven-Proxy.

Ein Maven-Proxy ist ein Repository das man für Firmen oder hochschulinterne Projekte nutzt, bei denen alle beteiligten Entwickler die Projekte anderer mitnutzen können. Das Repository unterscheidet sich nicht vom zentralen oder lokalen Repository, aber es bietet den Projektteilnehmern die Möglichkeit Softwareprojekte mit allen Vorzügen der Maven Infrastruktur zu pflegen und zu erweitern.

[5] http://search.maven.org/; abgerufen am: 10.12.2012
[6] http://www.apache.org/; abgerufen am: 10.12.2012

Im Rahmen dieser Arbeit wird ein Maven Proxy für die HAW Hamburg installiert und ist auch für die Projekte im Living Place Hamburg benutzbar, vgl. Abschnitt 5.1. In jedem Standard Maven Proxy wird grundsätzlich zwischen einem sog. Release und einem sog. Snapshot Repository unterschieden. Das Release Repository wird verwendet, um fertige Versionen der Software auszuliefern. Das Snapshot Repository wird genutzt um Meilenstein- oder Entwicklungsversionen auszuliefern.

4.2 OSGi als Application Container

Die Implementationsanforderung für einen zentralen Punkt an dem Sensorinformationen zur Entscheidungsfindung zur Verfügung stehen und die Infrastrukturanforderung zur Zustandsübersicht können kombiniert werden.

Für die Zustandsübersicht wurde zunächst ein Konzept erarbeitet, das die Verwendung eines Agent Lifecycles, ähnlich dem der Foundation for Intelligent Physical Agents (FIPA), vorsieht. Als Bibliothek[7] für Projekte sollte diese Implementation zum Einsatz kommen. Dieser Ansatz hat jedoch zwei Nachteile. Ein Nachteil ist, dass diese Bibliothek entgegen der Absprachen, von einem Projektteilnehmer vergessen werden kann. Dann ist der Zustand des Projekts nicht erfragbar und es besteht keine Möglichkeit den Zustand dessen zu erfragen. Der andere Nachteil ist eine möglicherweise falsche Nutzung. Wenn das Konzept hinter dem Mechanismus nicht korrekt eingesetzt wird, kann die Information, die über das Projekt erfragt wird, falsch sein oder der Agent Lifecycle wird auf der Projektseite falsch integriert. Durch die genannten Nachteile wird dieser Lösungsansatz ausgeschlossen.

Damit ein Projektteilnehmer den Einsatz eines Lösungskonzepts nicht vergessen kann, muss die Lösung Teil der Laufzeitumgebung werden. Ein Application Container bietet diese Eigenschaft und ermöglicht den Betrieb der Softwareprojekte in einer Umgebung die programmatisch beobachtet werden kann.

Als Application Container wird die OSGi Service Plattform eingesetzt, sie besteht aus dem OSGi Framework und den OSGi Standard Services. Alternativen zu einer Modularisierung auf Ebene der Java Virtual Machine (JVM) sind nicht zu finden. Die von der OSGi Alliance[8] entwickelte Spezifikation[9] bildet die Basis für die Service Plattform.

[7] http://livingplace.informatik.haw-hamburg.de/wiki/index.php/Agenten_Lifecycle; abgerufen am: 12.04.2013
[8] http://www.osgi.org/; abgerufen am: 11.12.2012
[9] http://www.osgi.org/Specifications/HomePage; abgerufen am: 11.12.2012

Die OSGi Alliance ist eine nonprofit Organisation, die seit ihrer Gründung im März 1999 fortlaufend an der OSGi Standard Spezifikation entwickelt. Das OSGi Framework ist die Laufzeitumgebung für die Standard Services der OSGi Service Plattform. Die Framework Implementationen der Spezifikation werden von der OSGi Alliance als standardkonform zertifiziert. Die aktuell zertifizierten Implementationen für Revision R4 der Spezifikation sind folgende[10]:

- Makewave Knopflerfish Pro 2.0 (www.makewave.com)
- ProSyst Software mBedded Server 6.0 (www.prosyst.com)
- Eclipse Equinox 3.2 (www.eclipse.org/equinox/)
- Samsung OSGi R4 Solution (www.samsung.com)
- KT OSGi Service Platform (KOSP) 1.0 (http://www.kt.co.kr/)
- Hitachi Solutions SuperJ Engine Framework V4 (www.hitachi-solutions.com)
- Apache Felix Framework 3.0.0 (http://felix.apache.org)

Aus dieser Liste wurden die jeweiligen Folgeversionen Makewave Knopflerfish Pro 3, ProSyst Software mBedded Server 7, Hitachi Solutions SuperJ Engine Framework V4 und Apache Felix Framework 3.0.0 schon für die Revision R4 V4.2 zertifiziert. Die Projekte Eclipse Equinox, Apache Felix und Knopflerfish sind Open Source und können unter den angegebenen Uniform Resource Locators (URLs) heruntergeladen werden. Die Eclipse Equinox Plattform ist, wie der Name schon vermuten lässt, die Basis für die bekannte Eclipse IDE. Auch der Application Server GlassFish[11] baut auf dem OSGi Framework auf, als Implementation wurde hier Apache Felix gewählt. Das OSGi Framework wird in vielen Bereichen eingesetzt, unter anderem Mobil-, Desktop-, Automobil- und neuerdings auch Enterpriseanwendungen, vgl. (Hall u. a., 2011, S. 7 ff).

Das OSGi Framework basiert darauf, dass der Entwickler die Funktionalitäten der Applikation in Modulen, in diesem Kontext *Bundles* genannt, aufteilt und getrennt voneinander implementiert. Bundles sind JARs die mit zusätzlichen Metadaten angereichert werden. Das Bundle umfasst alle vom Compiler erzeugten Klassendateien sowie die Ressourcen die zum JAR gehören, zudem auch die Metainformationen in Form einer Manifestdatei, vgl. Abb. 4.1. Die Metadaten in der Manifestdatei werden vom

[10] http://www.osgi.org/Certification/Certified; abgerufen am: 11.12.2012
[11] http://glassfish.java.net/; abgerufen am: 12.12.2012

Framework genutzt, um dieses zu identifizieren und deren Eigenschaften zu analysieren. Bundles können als Komponenten der Unified Modeling Language (UML) verstanden

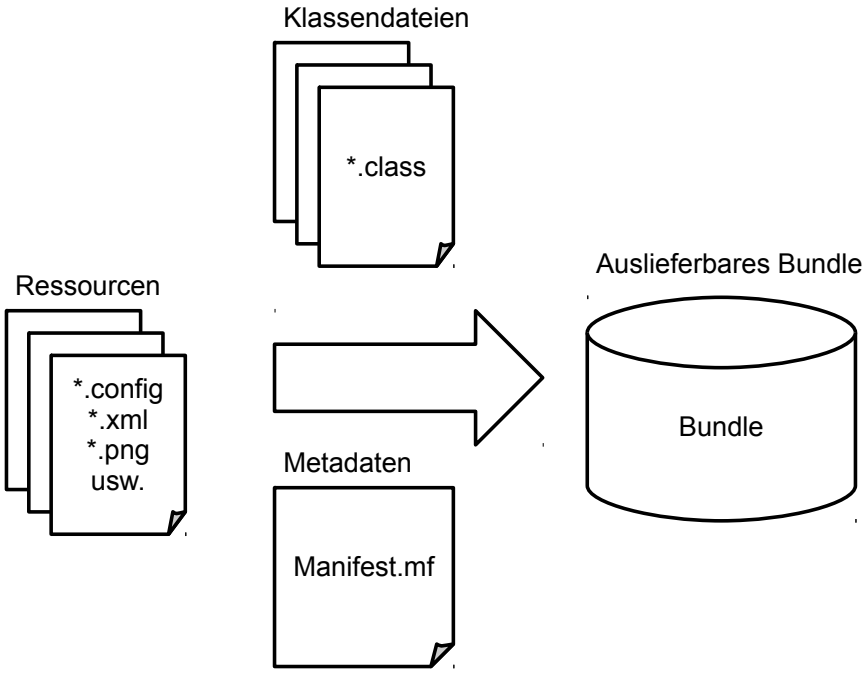

Abbildung 4.1: Inhalt eines OSGi-Bundles

werden. Komponenten sind Strukturelemente, und werden benutzt um sich einen Systemüberblick zu verschaffen. Sie beschreiben ihre Schnittstellen und Ports, sodass die Abhängigkeiten und Beziehungen zwischen ihnen deutlich werden, vgl. (Kecher, 2011, S. 145 ff). Ein Bundle könnte beispielsweise eine Webseite mit einem Zeitstempel darstellen. Die Funktion des Bundles erfordert dazu einen Webserver und die lokale Zeit. Der Webserver gehört allerdings nicht zu der Funktionalität des Bundles, sondern stellt eine Abhängigkeit dessen dar. Die lokale Zeit und dessen Formatierung ist in dem Bundle integriert.

Der Vorteil bei einer Trennung der Funktionen über Bundles ist die Wiederverwendbarkeit der Funktionen. Der Webserver könnte auch von anderen Bundles benutzt werden. Dadurch das die Bundles aber alle einen eigenen ClassLoader durch das Framework gestellt bekommen, können unterschiedliche Bundles problemlos unterschiedliche Versionen der gleichen Bibliothek benutzen nutzen, vgl. Hall u. a. (2011).

Das OSGi Framework ist in Schichten aufgeteilt, um dies zu verstehen, sollte man die unterschiedlichen Ebenen, die das Framework ausmachen, betrachten, vlg. Abb. 4.2. Auf die detaillierte Erläuterung der Spezifikation wird hier verzichtet, um den Rahmen dieser Arbeit nicht zu verlassen, weiterführende Informationen sind in der Spezifikation selbst zu finden, vgl. OSGi Alliance (2009a).

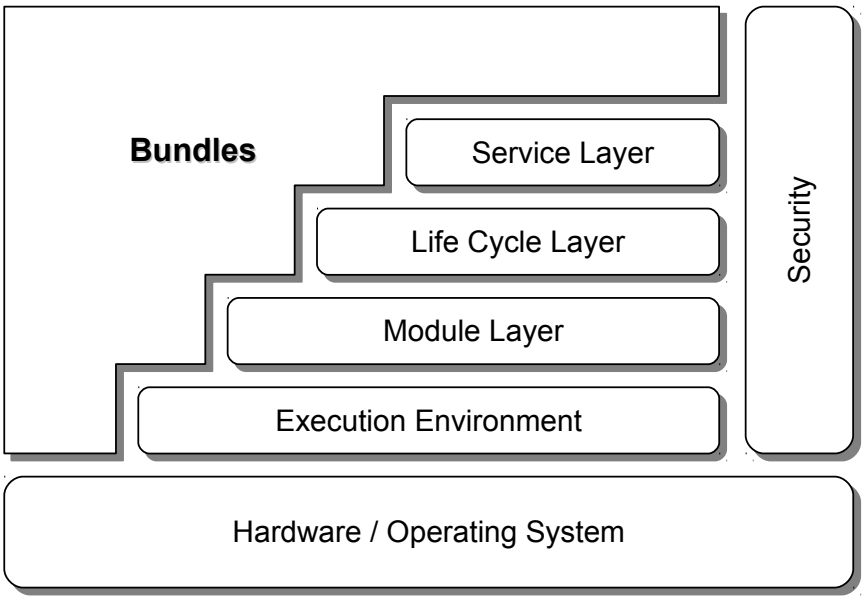

Abbildung 4.2: Aufbau der Schichten im OSGi Framework

Security Layer Der Security Layer ist ein optionaler Teil des Frameworks und ist nicht zwingend erforderlich, um ein vollständiges Framework abzubilden. Die API für den Layer baut auf der Java 2 Security Architecture Version 1.2 auf. Ein Framework muss zwar die Methoden nicht implementieren, aber darf auch keine *SecurityException* werfen. Die Spezifikation macht außerdem Aussagen darüber, wie Metadaten für Signaturen in JARs verwendet werden können und beschreibt eine Rechtestruktur zwischen Bundles, vgl. (OSGi Alliance, 2009a, S. 11 ff).

Execution Environment Das Execution Environment ist die Laufzeitumgebung, in der Bundles und OSGi Services ausgeführt werden. Generell sind Bundles Execution

Environment unabhängig und können damit die Laufzeitumgebung wechseln. Wenn das nicht der Fall ist, müssen sie das Execution Environment, von dem sie abhängen, explizit als Abhängigkeit mit der `Bundle-RequiredExecutionEnvironment` Direktive im Manifest kennzeichnen, vgl. (OSGi Alliance, 2009a, S. 35 ff).

Module Layer Der Module Layer des OSGi Frameworks bietet eine ClassLoader Infrastruktur, die es erlaubt Bundles voneinander isoliert und mit unterschiedlichen Versionen von Abhängigkeiten zu laden. Außerdem werden in dieser Schicht auch Bundleabhängigkeiten und native Bibliotheken geladen und sichergestellt, dass ein Bundle valide ist. Durch die Metadaten, typischerweise `META-INF/MANIFEST.MF` in der JAR-Datei, werden Bundles vollständig beschrieben. Unter anderem werden diese Metadaten mit `Import-Package` und `Export-Package` Direktiven des Bundles erweitert, um die Funktionalitäten zu beschreiben die zur Verfügung gestellt oder benötigt wird. Auch eine explizite Beschreibung der Abhängigkeiten des Bundles und der Services wird dort deklariert. Durch diese ClassLoader Hierarchie, bei der jedes Bundle einen eigenen ClassLoader und damit auch seinen eigenen ClassPath hat, kann das Framework den Scope der Klassen auf Bundles beschränken, vgl. (OSGi Alliance, 2009a, S. 27 ff).

Life Cycle Layer Der Life Cycle Layer baut auf dem Security und Module Layer auf und ermöglicht es dem Framework die Bundles zu installieren, diese zu starten, zu stoppen und zwischen den unterschiedlichen Bundle States zu wechseln, vgl. Abb. 4.3. Bundles werden mit einem Objekt gestartet und gestoppt, welches das `BundleActivator` Interface implementiert. Diese Klasse bietet dem Framework dann mit den Methoden `start()` und `stop()` die Möglichkeit zwischen den Status *RESOLVED*, *STARTING*, *ACTIVE* und *STOPPING* zu wechseln. Mit jedem Bundle pflegt das Framework ein damit assoziiertes `BundleActivator` Objekt, mit dem der Life Cycle des Bundles verwaltet wird. Jedes Bundle wird in einem Kontext, dem `BundleContext`, ausgeführt. Dieser Kontext wird beim Start eines Bundles erstellt und dient als Proxy zwischen dem Bundle und dem Framework. Es bietet dem Bundle die Möglichkeit, im Service Layer Ereignisse des Frameworks zu abonnieren, andere Bundles im Framework zu installieren, sowie das Anmelden der eigenen Services und das Referenzieren anderer Services zur eigenen Nutzung, vgl. (OSGi Alliance, 2009a, S. 85 ff).

Service Layer Der Service Layer ist stark mit dem Life Cycle Layer verwoben, denn diese Schicht führt das Verzeichnis der vorhandenen Services und ermöglicht

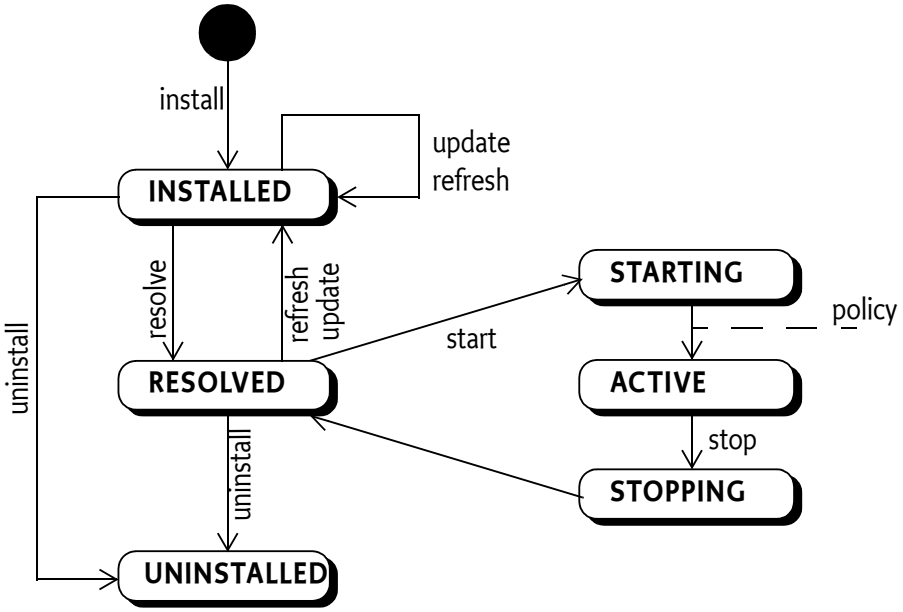

Abbildung 4.3: Zustandsdiagramm der möglichen Bundlezustände, vgl. OSGi Alliance (2009a)

das Suchen und Registrieren darin. Der Service selbst läuft im Bundle, aber das Framework kümmert sich um die Verwaltung, also korrekte Nachrichten für An- und Abmeldung von Services, Auflösen der Laufzeitabhängigkeiten zwischen Bundles, sowie automatische Mechanismen zur Registrierung selbst. Eine der wichtigsten Aufgaben des Service Layers ist die Pflege der Abhängigkeiten zwischen Bundles. Wenn ein Bundle gestoppt wird oder sich abmeldet und in den *RESOLVED* oder *UNINSTALLED* Zustand übergeht, werden alle Referenzen die dieses Bundle zur Verfügung gestellt hat als *stale*, ungültig markiert. Diese *stale* Referenzen müssen von anderen Bundles aufgeräumt werden und dürfen nicht weiter verwendet werden, vgl. (OSGi Alliance, 2009a, S. 123 ff).

Die Schichten der OSGi Plattform können auch getrennt voneinander verwendet oder nur teilweise instrumentalisiert werden um das gewünschte Maß an Kohäsion und Kopplung zu erreichen. Die Spezifikation beschreibt ein Framework nach dem sich die unterschiedlichen Schichten implementieren lassen. Die individuelle Funktionsweise wird allerdings durch Bundles erreicht. Die OSGi-Plattform bringt einige Standardservices als Bundles mit und bietet auch dafür Spezifikationen aus dem OSGi Compendium,

Item	Package	Version
101 Log Service Specification	org.osgi.service.log	Version 1.3
102 Http Service Specification	org.osgi.service.http	Version 1.2
112 Declarative Services Specification	org.osgi.service.component	Version 1.1

Tabelle 4.1: OSGi Compendium Service Specifications, vgl. OSGi Alliance (2009b)

vgl. OSGi Alliance (2009b). Tabelle 4.1 zeigt die hier verwendeten, eine vollständige Auflistung ist unter Anhang 2 zu finden.

In dieser Arbeit wird vor allem der *Log Service(Item 101)* häufig verwendet, um auf die Anforderungen zur Übersicht der Laborumgebung einzugehen, vgl. Tabelle 4.1. Sollte eine Komponente ausfallen, wird es dadurch die Möglichkeit geben, die Gründe in einem zentralen Log zu suchen, indem auch alle anderen Informationen zum Gesamtzustand des Labors einzusehen sind. Des Weiteren wird der *Http Service* über die Weboberfläche benutzt um einzelne Bundles neu zu starten oder zu steuern.

Um die Anforderungen aus den Abschnitten 3.3.2 und 3.3.3 zu bewältigen, wird diese Plattform als Grundlage zur Modularisierung dienen. Es können im Anschluss, aus unterschiedlichen Bibliotheken der Studenten, Bundles entwickelt werden, um diese dann in einer OSGi-Laufzeitumgebung zu integrieren. Ziel dabei wird es sein, die Bundles möglichst flexibel zu gestalten und die Schnittstellen von den Implementationen zu trennen, sodass eine hohe Kohäsion entsteht und die Implementationen trotzdem ausgetauscht werden können.

Weitere Details zur konkreten Implementation und Integration im Living Place Hamburg werden in den Abschnitten 4.4 und 5 erläutert.

4.3 Drools Fusion für Complex Event Processing

Eine der in Abschnitt 3.6 erarbeiteten Anforderungen ist eine möglichst einfache Integration von neuem Verhalten in die Laborumgebung. Es soll eine Möglichkeit geschaffen werden, Verhaltensprototypen testen und integrieren zu können, ohne dabei wiederholt programmieren zu müssen.

Ein geeigneter Mechanismus ist eine regelbasierte Maschine, vgl. Abschnitt 2.4. Wenn es einen Mechanismus gibt, mit dem die Informationen der Sensoren in die Wissensbasis

der regelbasierten Maschine aufgenommen werden können, kann ein reaktives Verhalten durch Regeln produziert werden. Alle Entscheidungen, die getroffen werden und alle Reaktionen, die in dieser Wohnumgebung automatisiert werden, müssen klar als solche zu identifizieren sein. Allerdings fehlt in einer regelbasierten Maschine eine elegante Möglichkeit, Zeiten in die Formulierung von Regeln mit einzubeziehen. Aus diesem Grund wird statt einer regelbasierten Maschine, eine CEP Maschine gewählt.

Die Kombination einer regelbasierten Maschine mit der Fähigkeit, zeitliche Abhängigkeiten in die Regelverarbeitung einzubeziehen bietet das Drools Fusion Projekt, vgl. Walzer u. a. (2007). Das Drools Fusion Projekt[12] von JBoss bietet neben offenen Quellen[13] auch eine umfangreiche Dokumentation und eignet sich daher für den Laboreinsatz, vgl. JBoss Drools Team (2012). Alternativen wie Jess[14] oder IBM ILOG[15] haben entweder den Nachteil, eine neue Sprache lernen zu müssen, um Regeln formulieren zu können, oder sind ein rein kommerzielles Produkt, bei dem die Quellen nicht offen liegen.

Das Drools Fusion Projekt bietet eine Plattform zur dynamischen Verwaltung von Events. Events sind Fakten die folgende Terminologie und Charakteristika aufweisen, vgl. (JBoss Drools Team, 2012, S. 5 ff):

immutability Events sind Zustandsänderungen in einer bestimmten Anwendungsdomäne. Diese Änderung wurde von einem Sensor registriert, der daraufhin diese Änderung als Event publiziert. Die Änderung des Zustandes ist allerdings schon geschehen, wenn der Sensor diese wahrnimmt und liegt damit in der Vergangenheit. Ereignisse in der Vergangenheit können nicht geändert werden und damit sind Events unveränderlich, englisch *immutable*.

strong temporal constraints Events treten typischerweise in einem zeitlichen Zusammenhang auf. Ein Beispiel ist der Zusammenhang zwischen dem Betreten des Wohnraums und der Öffnung der Wohnungstür. Um diese Vorgänge in einen Zusammenhang bringen zu können, müssen die Events in eine zeitliche Abhängigkeit gebracht werden. Welche zeitlichen Zusammenhänge Drools Fusion unterstützt, wird im nächsten Abschnitt erläutert.

managed lifecycle Durch die Eigenschaften der Unveränderlichkeit und der zeitlichen Zusammenhänge können die Lebenszyklen der Events zu einem großen Teil von der

[12] http://www.jboss.org/drools/drools-fusion.html; abgerufen am: 14.01.2013
[13] https://github.com/droolsjbpm/drools; abgerufen am: 14.01.2013
[14] http://www.jessrules.com/; abgerufen am: 26.02.2013
[15] http://www-01.ibm.com/software/websphere/ilog/; abgerufen am: 26.02.2013

CEP Maschine automatisch verwaltet werden. Der Lebenszyklus von einem Event ist begrenzt durch seine Lebensdauer, diese wird mit einer `@expires` Direktive in den Regeln gekennzeichnet. Drools Fusion macht sich dieses Attribut zu Nutze und sorgt dafür, dass Ressourcen, die nicht mehr benötigt werden, weil Events nicht mehr auf Regeln zutreffen können, freigegeben werden.

sliding windows Alle Events werden beim Eintritt in die Faktbasis des Systems mit einem Zeitstempel versehen. Diese Zeitstempel können verwendet werden, um gleitende Zeitfenster für Bedingungen und Aggregationen zu verwenden. Zum Beispiel können mit diesem Verfahren durchschnittliche Temperaturen für Räume ermittelt werden.

Die Entscheidung für eine CEP Maschine und gegen eine reine regelbasierte Maschine ist vor allem durch die Möglichkeit der zeitlichen Abhängigkeiten gefallen. Diese ermöglichen erst den effektiven Einsatz von Regeln zur Steuerung in einer intelligenten Wohnumgebung. Zum Beispiel ist die Information, dass jemand das Sofa verlassen hat, nur für einen relativ kurzen Zeitraum relevant für Systeme, die auf Positionsänderungen reagieren wollen. Die Möglichkeit, Events mit einer Lebensdauer zu koppeln und diese dann von Drools Fusion selbst verwalten zu lassen, garantiert eine Faktbasis, die nur relevante Events enthält und damit auch performant bleibt. Welche Möglichkeiten es mit Drools Fusion zum Schreiben der Regeln in zeitlichem Zusammenhang gibt, zeigt Abbildung 4.4.

Mit diesen Modellierungsmöglichkeiten lassen sich die aktuellen und zukünftigen Szenarien im Living Place Hamburg abbilden. Wie die konkrete Implementation dieser CEP Maschine im Labor aussieht und welche Implementationsdetails zu beachten sind, wird in den Abschnitten 4.4.2 beschrieben.

4.4 Konzeption der OSGi Bundles

Der praktische Teil dieser Arbeit sieht die Implementation von OSGi-Bundles vor. Sie werden die Integration der CEP-Maschine und die Zusammenführung aller Sensorik und Aktorik gewährleisten.

Die Entwicklung der Lösungsansätze zu den Anforderungen wird in den folgenden Abschnitten beschrieben und gibt einen Überblick zu der praktischen Umsetzung der Konzepte dieser Arbeit. Ziel ist, eine zukunftsfähige Infrastruktur für Folgeprojekte und die damit verbundene Integration in das Labor zu entwickeln.

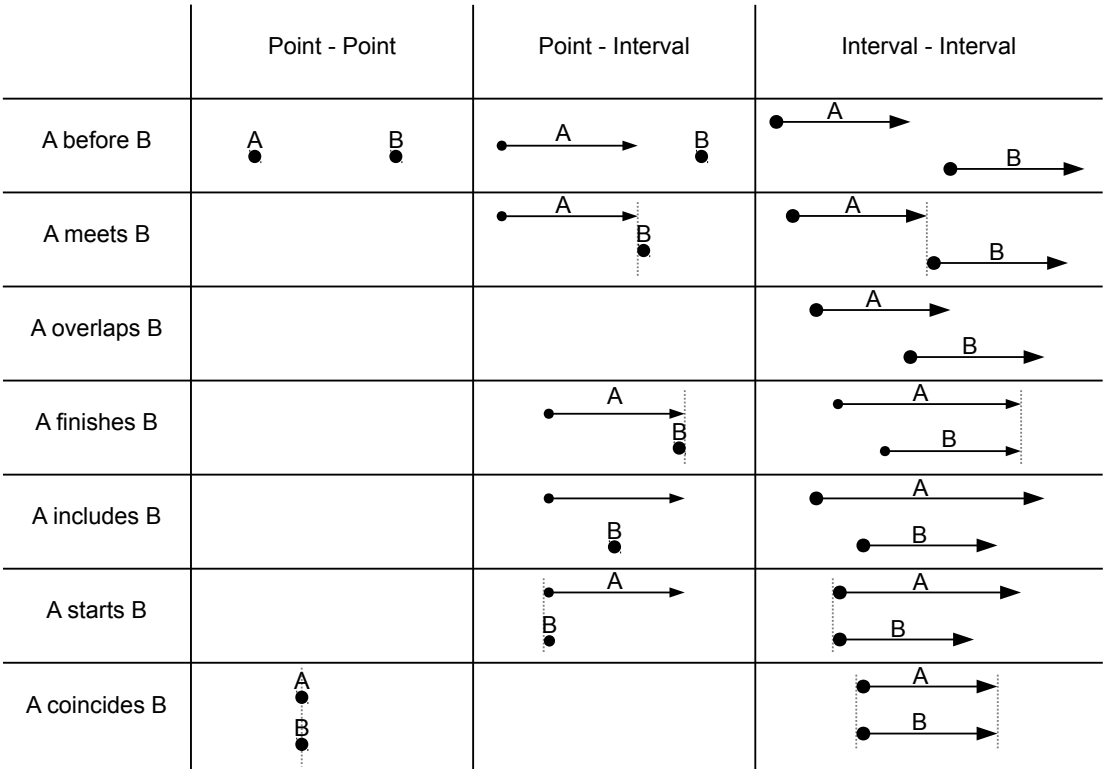

Abbildung 4.4: Modellierungsmöglichkeiten von zeitlichen Abhängigkeiten mit der Drools Fusion CEP Maschine

4.4.1 Applikationsstruktur

Die in Abschnitt 3.6 beschriebenen Anforderungen werden hier aufgegriffen und in die Modellierung der Applikationsstruktur einfließen. Eine Anforderung ist der Erhalt der Kommunikationsinfrastruktur. Nach Einführung der OSGi-Laufzeitumgebung wird der ActiveMQ, genau wie andere Kommunikationsmechanismen auch, bestehen bleiben.

Die Idee hinter diesem Ansatz ist, dass man die Art der Anbindung von Sensoren und Aktoren unabhängig von der Kommunikationsstruktur macht. Sensoren können Informationen liefern. Aktoren können Aktionen zur Ausführung bereitstellen. Um diese Kernkomponenten der Laborumgebung nicht an eine einzige Lösung zu binden, werden diese zwar mit einer abstrakten Implementation und einem Interface versehen, aber sind damit nicht gezwungen nur den Mechanismus des ActiveMQ zur Kommunikation zu nutzen. Das ermöglicht eine flexible Integration von Kommunikationsmechanismen, die völlig unabhängig von bisherigen Lösungen agieren können, diese aber dennoch nicht ausschließen, vgl. Abb. 4.5.

Ursprünglich sollten solche Kommunikationen nicht stattfinden, werden aber in Zukunft von Vorteil sein um die Informationen und Aktionen weiter zu abstrahieren. Man könnten zum Beispiel die Rohdaten eines Positionssensors nutzen und daraus eine abstraktere Information bilden, eine Information die Auskunft darüber gibt, ob sich eine bestimmte Person in einem bestimmten Raum befindet. Wenn diese Information nicht für die Logik außerhalb der Laufzeitumgebung zur Verfügung stehen soll, wird der Aktor oder Sensor diese nicht über den ActiveMQ publizieren.

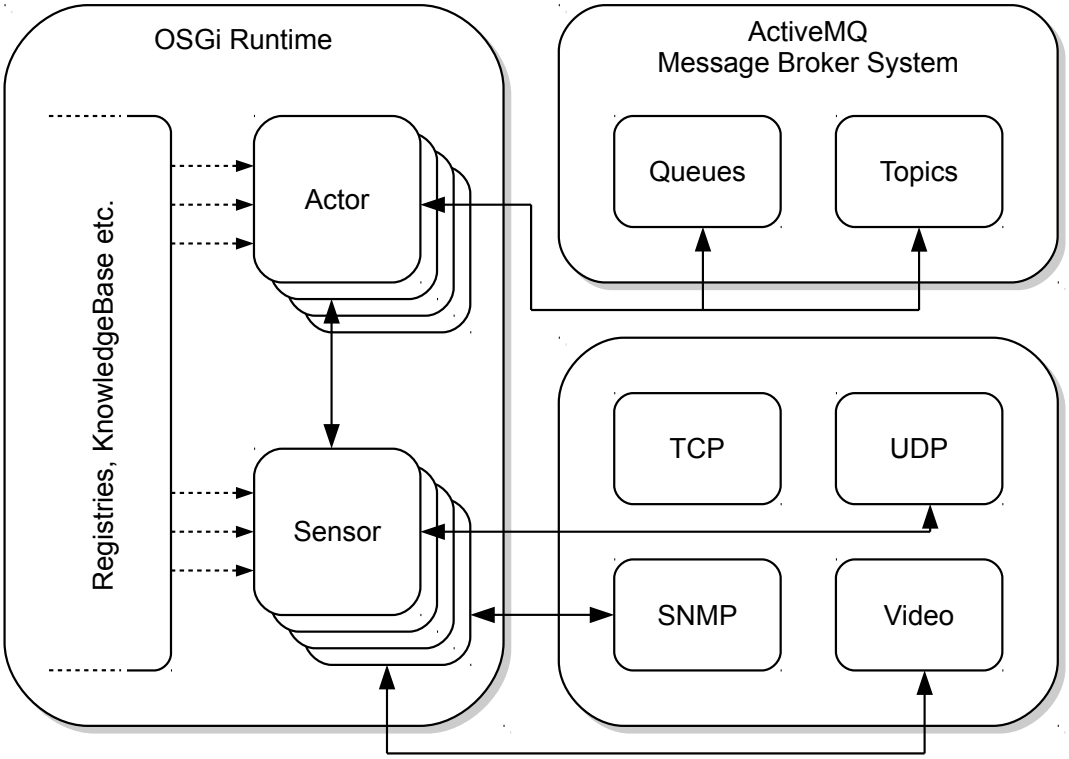

Abbildung 4.5: Erweiterte Kommunikationsstrukturen nach der Einführung der OSGi Laufzeitumgebung

Die Unabhängigkeit von IDEs sowie Wiederverwendbarkeit und Versionierung der einzelnen Projekte sind bereits durch den Einsatz von Maven gelöst. Der Einsatz von Maven ermöglicht auch die Verwendung von PlugIns, die helfen können, eine Projektstruktur zu generieren. Das verwendete Werkzeug zum Erstellen der einheitlichen Projektstruktur ist Pax Construct[16]. Mit diesem Werkzeug werden die POM Dateien für alle Module und Submodule in dieser Arbeit erstellt. Es vereinfacht die Verwaltung

[16] http://team.ops4j.org/browse/PAXCONSTRUCT; abgerufen am: 26.02.2013

und Erweiterung der mavenbasierten Umsetzung und bildet logische Ordnerstruktur ab. Eine ausführliche Dokumentation für die Verwendung im Laborbetrieb würde den Rahmen dieser Arbeit verlassen, ist aber im Wikimedia des Labors zu finden[17].

Einen Überblick über die angestrebte Applikationsstruktur gibt Abbildung 4.6. Die Abbildung veranschaulicht die drei Schichten auf denen Teilkomponenten im System integriert werden. Die oberste Schicht enthält die CEP Drools Fusion mit einem besonderen ClassLoader. Der ClassLoader wird auf die Registrierungen der Schicht darunter zugreifen, über die Informationen und Aktionen in diesem System registriert werden. Die Registrierungen ermöglichen das dynamische Laden von Klassen und erfüllen zwei unterschiedliche Aufgaben. Die Informationsregistrierung verwaltet zu jeder Information Listener, mit denen die Sensoren aus der darunter liegenden, letzten Schicht, diese publizieren. Die Aktionsregistrierung bietet, neben der Verwaltung aller bekannten Aktionen, die Möglichkeit Aktionen in ihrem Executor Framework ausführen zu lassen. Die unterste Schicht bilden die Sensoren und Aktoren, die als Abstraktionsmittel dienen um Informationen und Aktionen in Bundles zusammenzufassen.

Die Infrastruktur, die diese Arbeit bereitstellt, ermöglicht es einem neuen Projekt im Labor, eine vorgefertigte Struktur zu nutzen und gibt vor, welche Teile der Implementation von dem jeweiligen Projekt zu implementieren sind. Durch die Einführung von OSGi-Bundles wurde hier eine Plattform in die Laborstruktur gebracht, die den heutigen Industriestandard in der Modularisierung von Java Funktionalitäten bildet. Mit Hilfe dieser Plattform ist es möglich, den Anforderungen an Statusüberprüfung der einzelnen Programme nachzukommen. Alle Bundles werden in der OSGi Runtime gelistet und können damit auch geprüft werden. Auch zentrale und individualisierte Logs stehen allen Projekten zur Verfügung.

Sobald neue Sensoren oder Aktoren mit in den Living Place integriert werden sollen, werden hier nur minimale Arbeitsschritte notwendig sein. Die Integration ist vollständig von der Kommunikationsinfrastruktur gelöst und kann damit alle denkbaren Mechanismen abbilden. Diese Arbeit stellt dazu Prototypen und konkrete Realisierungen zur Verfügung und ermöglicht damit den schnellen Einstieg in die Strukturen, sowie die einfache Realisierung des neuen Projekts.

Im Folgenden werden die unterschiedlichen Bundles-Typen und weshalb diese Aufteilung gewählt wurde detailliert erläutert. Das Prefix `org.livingplace.controlling`

[17] http://livingplace.informatik.haw-hamburg.de/wiki/index.php/OSGi_Anleitung; abgerufen am: 12.04.2013

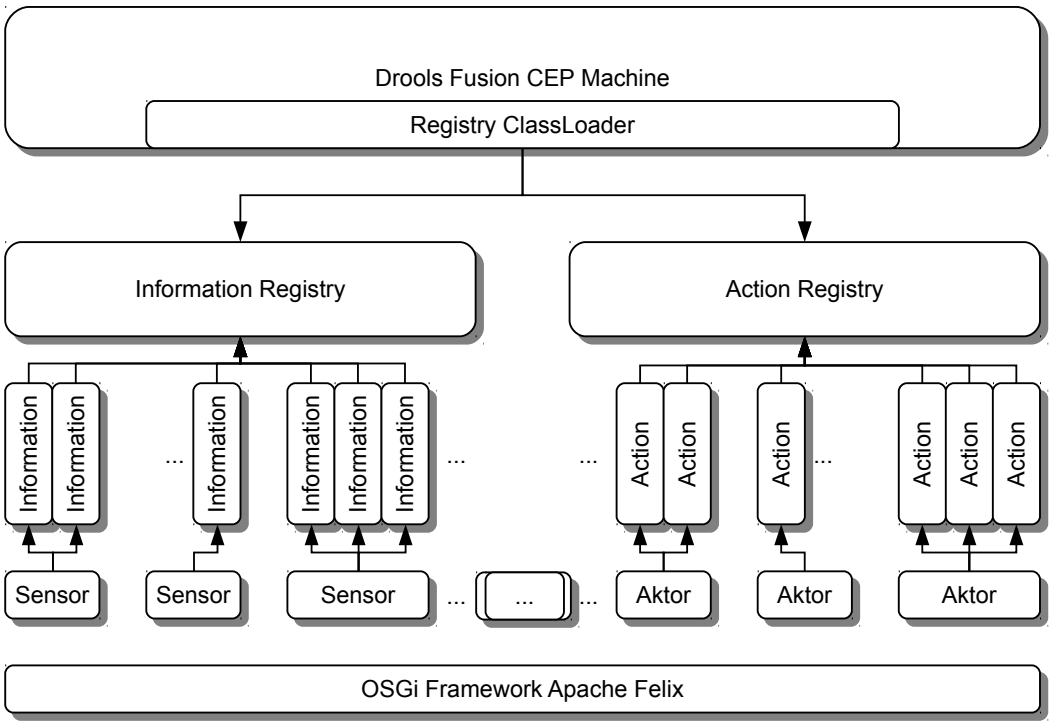

Abbildung 4.6: Struktur und Zusammenhang der modellierten Bundles

des Living Place Hamburg wird nicht mehr zur Identifikation der Namensräume im Fließtext verwendet, sondern als gegeben angesehen und mit einem ∗ substituiert. Die unterschiedlichen Bundletypen und deren Aufteilung werden in diesem Abschnitt erläutert, um im Anschluss die konkrete Implementation zu erläutern.

4.4.2 *.api Bundles

Die OSGi Bundles dieser Arbeit teilen sich in drei unterschiedliche Abschnitte. Dieser Abschnitt umfasst alle Namensräume, die auf API enden und der Definition von Schnittstellen innerhalb der Programmstruktur dieser Arbeit dienen. Alle konkreten Implementationen in diesen Bundles erfüllen die gegebenen Schnittstellen, sodass diese nicht zwangsweise verwendet werden müssen. Man kann diese jederzeit mit eigenen Implementationen ersetzen. Im Folgenden sind die API Bundles beschrieben, die Vorgaben für die in Abbildung 4.6 gezeigte Struktur bilden. Sie enthalten Schnittstellendefinitionen für die Registrierungen, Informationen und Sensoren, sowie Aktionen und Aktoren.

`org.livingplace.controlling.api` Der oberste Namensraum, `*.api`, ist das oberste API Bundle in dem alle Basisschnittstellen beschrieben werden. Die Schnittstellen umfassen Beschreibungen zur Registrierung, der Qualifizierung der Klassen die dynamisch zur Laufzeit zu identifizieren sind, sowie Basisimplementationen, die für alle hierarchisch tiefer liegenden Namensräume genutzt werden können.

`org.livingplace.controlling.actions.api` Der Namensraum, `*.actions.<...>`, wird verwendet um alle Bundles, die mit Aktionen in Verbindung stehen, zusammenzufassen. Dieser Namensraum, `*.actions.api`, bietet der Registrierung für Aktionen die nötigen Schnittstellen zur Identifikation und Verwaltung dieser. Auch die generellen Definitionen für Identifikationsklassen der Aktoren und Aktionen sind hier zu finden. Neben den Schnittstellen sind hier auch abstrakte Klassen vorgegeben, die eine einfache Implementation weiterer Aktoren und Aktionen gewährleisten.

`org.livingplace.controlling.actions.registry.api` Dieser Namensraum enthält die Schnittstellen für die Aktionsregistrierung. Diese ist von dem übergeordneten Namensraum getrennt, sodass man diese wenn nötig austauschen kann. Um zu gewährleisten, dass es immer nur eine Aktionsregistrierung gibt, wird hier auch eine Schnittstelle für eine Fabrik definiert, die eine Singleton Instanz erzeugen kann.

`org.livingplace.controlling.informations.api` Das Bundle `*.informations.<...>` wird für alle Schnittstellen und Implementationen im Kontext von Informationen und Sensoren verwendet. In `*.informations.api` werden die Schnittstellen zu Identifikation von Informationen und Sensoren gehalten. Auch die Klassen von denen als Beispielimplementationen geerbt werden soll, sind hier zusammengefasst.

`org.livingplace.controlling.informations.registry.api` Der Namensraum für die Registrierung von Informationen ist ähnlich dem Namensraum für Aktionsregistrierung gestaltet. Auch hier werden die Schnittstellen zur Definition der Registrierung selbst, sowie die einer Fabrik zu dessen Instantiierung angesiedelt. Die Schnittstelle und dessen Implementation hingegen unterscheiden sich durch Funktionsweise und

`org.livingplace.controlling.knowledge.api` Der `*.knowledge.api` Namensraum enthält die Anbindung an die CEP Maschine. Hier wird der Zugriff auf die Aktions- und Informationsregistrierungen gesteuert.

4.4.3 *.impl Bundles

Die Implementationsbundles sind die Bundles, in denen nicht nur Interfaces enthalten sind, sondern auch konkrete Implementationen für diese zur Verfügung gestellt werden. Die Implementationen der Registrierungen sind dabei getrennt von deren Schnittstellen, um einen möglichst hohen Entkopplungsgrad zu erreichen.

- `org.livingplace.controlling.actions.registry.impl` Das Bundle `*.actions.registry.impl` beherbergt die Implementation zu den Schnittstellen, die aus dem dazu passenden Bundle, `*.actions.registry.api`, stammen. Die Aktionsregistrierung ist für die dynamische Registrierung neuer Aktionen und deren Ausführung zuständig.
- `org.livingplace.controlling.informations.registry.impl` Dieses Bundle enthält die Registrierung für Informationen. Sie bietet die Möglichkeit zur Laufzeit neue Informationen zu registrieren und sich für den Empfang solcher zu melden.

4.4.4 *.commands Bundles

Die Kommandobundles sind Bundles, die vom System nicht zwingend benötigt werden, aber im praktischen Einsatz nützlich sind, um zum Beispiel die Status der verschiedenen Komponenten mit detaillierten Ausgaben abzufragen.

- `org.livingplace.controlling.actions.registry.commands` Dieses Bundle enthält die Kommandos zu der Aktionsregistrierung und ermöglicht das Abfragen bereits registrierter Aktionen. Außerdem ist es möglich, über diese eine parameterfreie Aktion direkt auszuführen.
- `org.livingplace.controlling.informations.registry.commands` Dieses Bundle enthält die Kommandos zu der Informationsregistrierung und ermöglicht das Abfragen bereits registrierter Informationen. Um auch bei dieser Registrierung die Möglichkeit zu haben ein direktes Feedback von den detektierten Informationen beziehungsweise Sensoren zu bekommen, gibt es hier eine Möglichkeit die Informationen gezielt in der Konsole anzeigen zu lassen.

4.4.5 *.informations.sensors- und *.actions.actors- Bundles

Die Aktoren und Sensoren, die über dieses System an die CEP Maschine angebunden werden, stehen allen Verfassern von Regeln zur Verfügung. Um herauszufinden, welche

Aktoren und Sensoren beziehungsweise welche Aktionen und Informationen vorhanden sind, werden diese in jeweils einem eigenen Namensraum gebündelt.

- `org.livingplace.controlling.actions.actors.*`
  `org.livingplace.controlling.informations.sensors.*`
  Diese beiden Namensräume fassen die Aktoren und Sensoren des Systems in eigenen Namensräumen zusammen. Für feinere Unterscheidungsmöglichkeiten werden weitere untergliederte Namensräume wie zum Beispiel `*.actions.actors.light` oder `*.informations.sensors.heating` erstellt. Es sollten für jeden Informations- oder Aktionskontext neue Namensräume gebildet werden, sodass der Kontext derer schon anhand des Namensraums erkenntlich wird. Der Entwurf dieser Bundles ist den Entwicklern neuer Projekte vorgegeben und bietet eine klare Projektstruktur, mit der die Integration einheitlich möglich ist. Die Pflege und Erweiterung wird in Zukunft eigenverantwortlich von neuen Projekten übernommen und ist nicht Teil dieser Arbeit.

4.5 Realisierung

In diesem Abschnitt wird die Realisierung der Software behandelt, welche Konzepte verwendet und auf welchen Mechanismen bei deren Modellierung zurückgegriffen wird. Die im vorangegangenen Abschnitt beschriebenen Bundles werden hier detaillierter erläutert und deren praktische Umsetzung beschrieben.

4.5.1 Declarative Services

Alle Komponenten der Realisierung sind in OSGi Bundles umgesetzt. Die OSGi Spezifikation sieht für Bundles einen extra Start und Stop Mechanismus vor, den sog. `org.osgi.framework.BundleActivator`. Um auf einen Service zugreifen zu können, wird der `start` Methode ein `BundleContext` übergeben, mit dem man Zugriff auf eine `ServiceReferenz` hat. Mit der `ServiceReferenz` kann nun über die `getService` Methode eine Instanz des Service angefordert werden. Wenn man einen Service registriert, verwendet man dazu auch den `BundleContext` und übergibt ihm den Klassennamen und eine Instanz der Klasse selbst. Damit das OSGi Framework weiß, welche Klasse als `BundleActivator` zu verwenden ist, wird auch diese Information in der Datei für OSGi Metadaten deklariert, vgl. (Hall u. a., 2011, S. 13 ff).

Dieser Ansatz bietet keine Möglichkeit, die Zustandswechsel von Services zu beobachten oder darauf zu reagieren. Dazu muss man im `BundleContext` einen `ServiceListener`

registrieren, der auf die Wechsel zwischen Zuständen reagiert. Jeder der ein Bundle entwickelt, müsste diese Funktionalität korrekt integrieren. Es gibt keine Möglichkeit, den Entwickler daran zu hindern, eine Referenz auf einen konkreten Service zu speichern, sodass unbemerkt ungültige Referenzen entstehen können. Sobald der Service gestoppt wird, auf den der Entwickler eine Referenz hält, ist die Servicereferenz ungültig (auch *stale reference* genannt), kann aber trotzdem nicht von der Garbage Collection aufgeräumt werden, vgl. (Hall u. a., 2011, S. 18 ff).

In dieser Arbeit werden, um die Schwierigkeiten der Referenzpflege zu umgehen, Declarative Services genutzt, vgl. Tabelle 2 Item 112. Der Vorteil an diesem Ansatz ist, dass man die Service Referenzen auf die anderen Bundles nicht selbst pflegen muss, sondern der Verwaltungsaufwand von dem Declarative Service Bundle übernommen wird. Im verwendeten OSGi Framework von Apache Felix ist eine Declarative Services Komponente mit umgesetzt. Um den Ansatz weiter zu vereinfachen und auch die eXtended Markup Language (XML)-Konfiguration unnötig zu machen, wird auf die Declarative Services Annotationen zurückgegriffen[18].

4.5.2 Information, Sensor, Aktion und Aktor Abstraktion

Sensoren und Aktoren sind die zentralen Interaktionspunkte in der Infrastruktur, die diese Arbeit bietet. Wie in Abschnitt 4.4.1 erläutert, wird die Implementation für einen Sensor oder Aktor damit auf einen einzigen Punkt beschränkt. Die folgenden Beschreibungen erläutern die Vorgaben die mit dieser Arbeit realisiert wurden.

4.5.2.1 Identifikation

Die Informationen und Aktionen, die von Sensoren und Aktoren zur Verfügung gestellt werden, müssen sich unterscheiden lassen, um eine mehrfache Aufnahme in eine der Registrierungen zu verhindern. Sollte eine Information mehrfach in einer Registrierung auftauchen, würde jede Information mehrfach in die Faktenbasis aufgenommen werden. Bei Aktionen wäre das Problem eine jeweilige Mehrfachausführung von Aktionen.

Um das zu verhindern, werden hier zwei Schnittstellen modelliert, `IQualifier` und `IQualifiable`. Die `IQualifier` Schnittstelle enthält Methoden, um zur Laufzeit den Namensraum, den Namen und die Version von einem Objekt zu erfragen. Diese Informationen reichen aus, um solche Objekte voll zu qualifizieren und Duplikationen

[18] http://felix.apache.org/documentation/subprojects/apache-felix-maven-scr-plugin/scr-annotations.html; abgerufen am: 28.01.2013

auszuschließen. Klassen, die `IQualifiable` implementieren, sind automatisch qualifizierbar, weil sie eine Methode implementieren müssen, die ihren `IQualifier` zurück gibt. Zur Unterscheidung der verschiedenen Qualifizierer für Sensoren, Informationen, Aktoren und Aktionen wird jeweils eine Schnittstelle, `ISensorQualifier` für Sensoren, `IInformationQualifier` für Informationen, `IActorQualifier` für Aktoren und `IActionQualifier` für Aktionen, entwickelt die von `IQualifier` erbt. Die Schnittstelle `IQualifier` wird durch die abstrakte Klasse `Qualifier` realisiert und stellt bis auf `getPrefix()` alle grundlegenden Methoden der Schnittstelle. Die Schnittstellen, die von `IQualifiable` erben, beinhalten das Prefix für die jeweiligen Kategorien und werden von den Klassen `SensorQualifier`, `InformationQualifier`, `ActorQualifier` und `ActionQualifier` implementiert. Diese erben von Qualifier und realisieren jeweils die zur Kategorie passende `getPrefix()` Methode, vgl. Abb. 4.7.

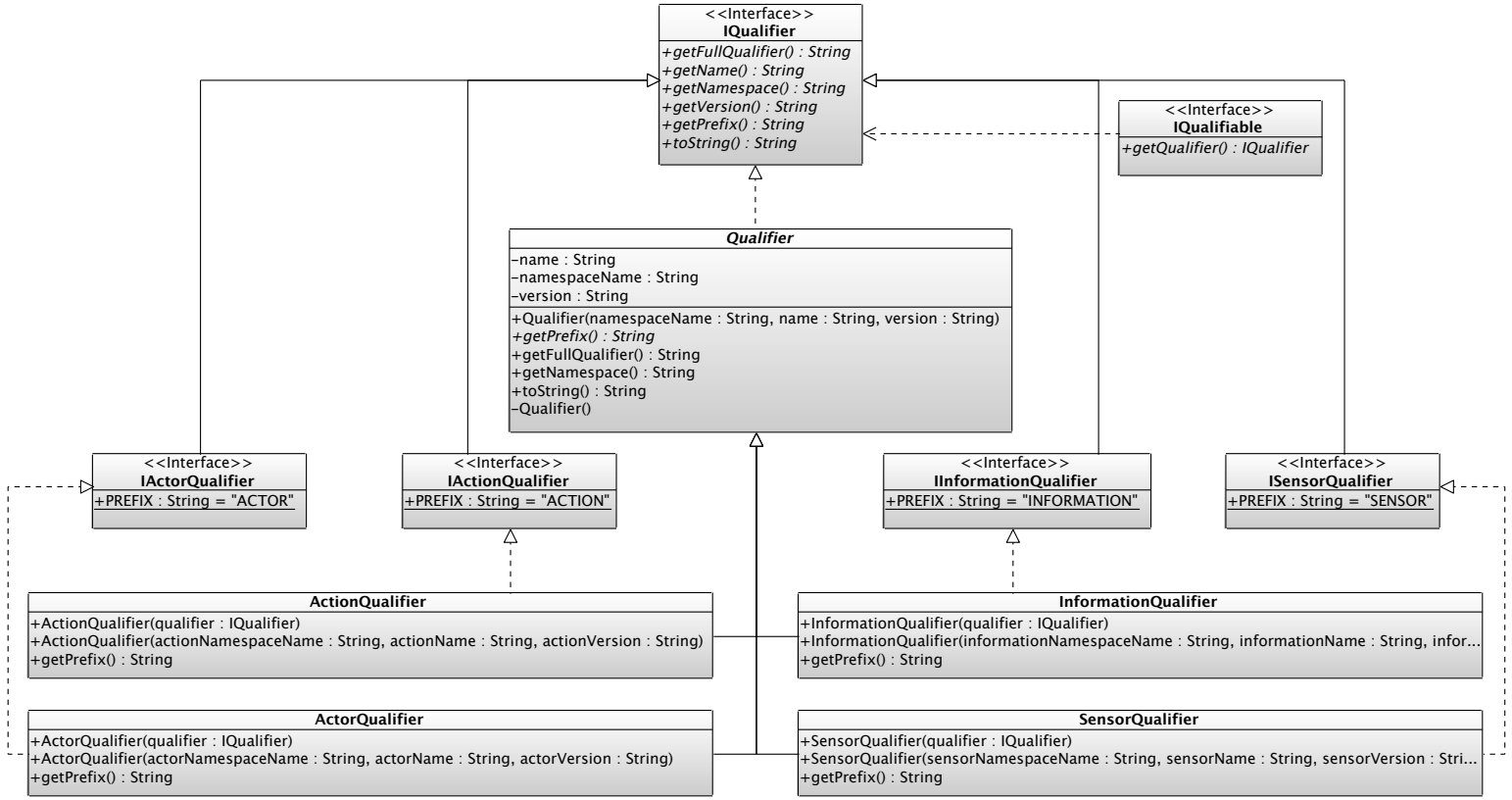

Abbildung 4.7: Struktur und Zusammenhang der modellierten Schnittstellen und Klassen zur Identifikation von Sensoren, Informationen, Aktionen und Aktoren

4.5.2.2 Informationen und Sensoren

Sensoren sind die Abstraktion der Entitäten, die Informationen und die Faktbasis der CEP Maschine liefern. Die Struktur der Klassen und Schnittstellen wird so modelliert, dass man einen möglichst geringen Implementationsaufwand hat. Wenn ein neues Sensorbundle implementiert wird, muss man eine Schnittstelle implementieren und von einer Klasse pro Information erben.

Die Schnittstelle einer Information ist `*.informations.api.IInformation`. Diese Schnittstelle erbt von `IQualifiable` und stellt damit sicher, dass alle Informationen identifizierbar sind. Die Schnittstelle hat neben der überschriebenen Methode aus `IQualifiable`, um einen `IInformationQualifier` zurück zu geben, nur zwei Methoden. Diese Methoden sind `Getter` und `Setter` von dem `Object`, das die eigentliche Information darstellt. Die Klasse, von der eine neue Information erben soll, ist `*.informations.api.providers.Information`. Ausgehend von der vorhandenen Implementation der Klasse muss eine neue Informationsklasse einen Konstruktor implementieren, dem die Quelle der Information, hier der konkrete Sensor, übergeben wird. Der Grund dafür ist, dass die Informationsklasse von dem Java Objekt für Events, `java.util.EventObject`, erbt, vgl. 4.5.3.1. Die neue Informationsklasse muss außerdem einen `IInformationQualifier` bereitstellen, um der Schnittstelle `IQualifiable` zu genügen. Damit ein solcher `IQualifier` nicht vergessen wird, ist dieser als `final` deklariert und muss im Konstruktor beim Aufruf des Superklassenkonstruktors mit übergeben werden, vgl. Abb. 4.8.

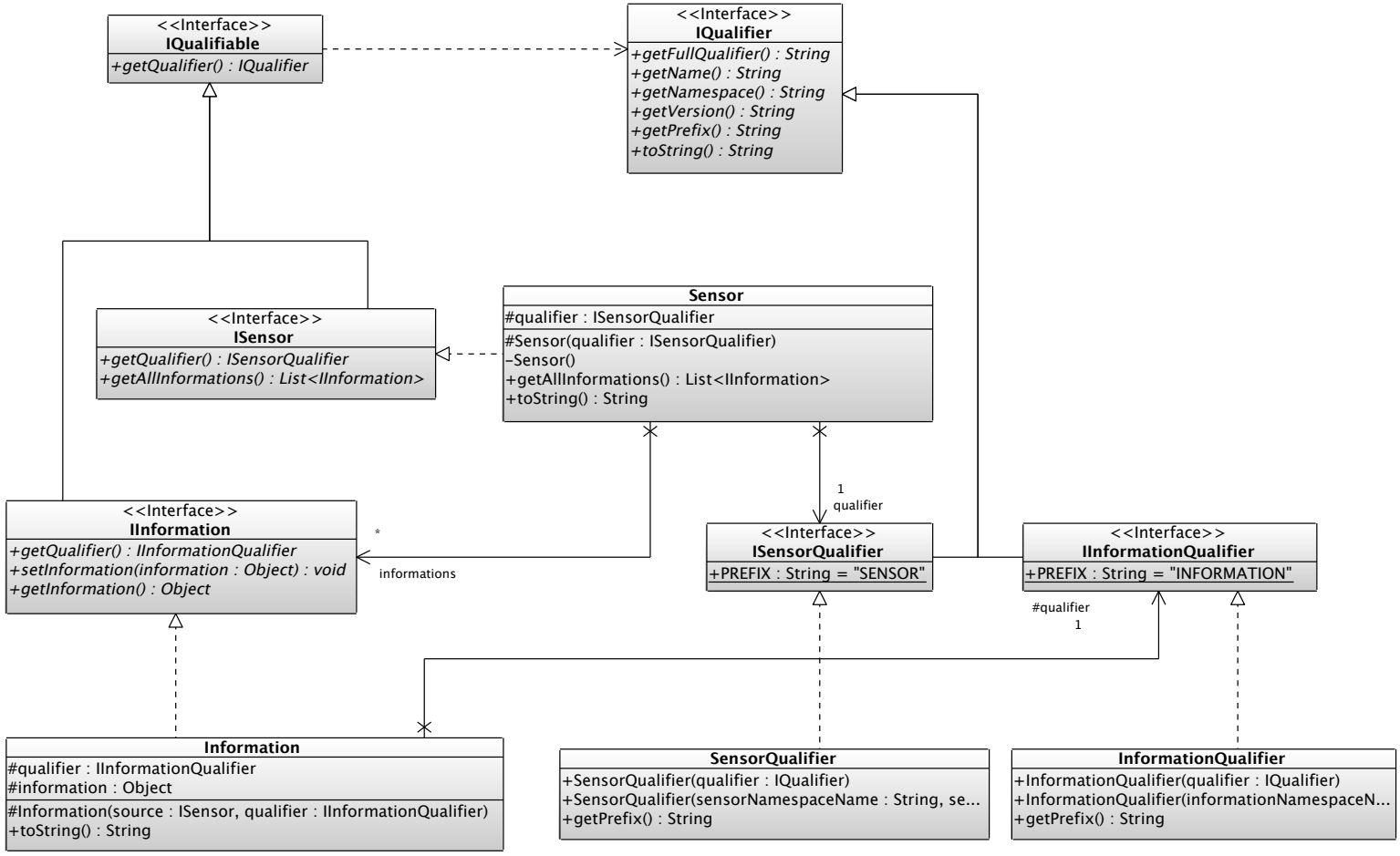

Abbildung 4.8: Struktur und Zusammenhang der modellierten Schnittstellen zur Qualifikation und Verwendung der Informationen und Sensoren

Der Sensor ist für die Funktionsweise des Systems nicht zwingend notwendig, bietet aber die Struktur, um Informationen zusammenzufassen und sie in einem Kontext zu bündeln. Ein Sensor ist immer ein eigenes OSGi Bundle und kann so auch gezielt gestartet und gestoppt werden ohne den Betrieb anderer Sensoren zu stören. Die Schnittstelle `ISensor` und die damit verbundene Klasse `Sensor` bilden analog zur Modellierung von `IInformation` und `AbstractInformation` eine Grundstruktur, die bei der Implementation hilfreich ist. Die `ISensor` Schnittstelle erbt, um auch Sensoren im System qualifizierbar zu halten, von `IQualifiable` und wird mit der Klasse `Sensor` implementiert. Die Sensorimplementation realisiert die Methoden zur Qualifikation und das Abfragen aller Informationen die der Sensor bereitstellt.

Ein Sensor verwaltet die Informationen die er liefern kann mit einer internen Informationsliste. Wird ein neuer Sensor implementiert, fügt dieser jede Information dieser Liste hinzu. Anschließend wird jede Information, die er verwaltet, bei der Registrierung für Informationen registriert. Für jede registrierte Information erhält man einen `IInformationListener`, den man mit dem Aufruf der `sensedInformation` Methode über eine neue Information benachrichtigt. Über diesen Listener wird die Information durch das System propagiert und in die Faktbasis eingestellt, vgl. Abb. 4.11.

4.5.2.3 Aktionen und Aktoren

Aktoren und Aktionen werden verwendet, um von einer Regel aus Aktionen ausführen zu lassen. Diese Kombination erlaubt es den Verfassern von Regeln Einfluss auf andere Komponenten des Systems zu nehmen, ohne den Rahmen der Regeln verlassen zu müssen. Sie sind wie Sensoren und Informationen auch über die Spezialisierungen der `IQualifiable` Schnittstelle, `IActionQualifier` und `IActorQualifier`, qualifizierbar. Auch hier werden wieder Implementationen vorgegeben, um den Entwicklungsaufwand bei der Neuintegration eines Aktors gering zu halten. Die Klassen `Actor` und `AbstractAction` stellen als Superklassen das Grundgerüst zur Verfügung, vgl. Abb. 4.9.

Aktoren sind nicht notwendig, bilden aber als Pendant zum Sensor das logische Gegenstück zum Zusammenfassen von Aktionen. Auch ein Aktor wird immer durch ein eigenes OSGi Bundle repräsentiert und muss seine Aktionen bei der Aktionsregistrierung anmelden. Aktionen werden im Gegensatz zu Informationen ausgeführt. Die Ausführung der Aktionen findet in der Aktionsregistrierung statt, indem diese die Aktionsmethode `execute` aufruft, vgl. Abschnitt 4.5.3.2 und Abb. 4.12.

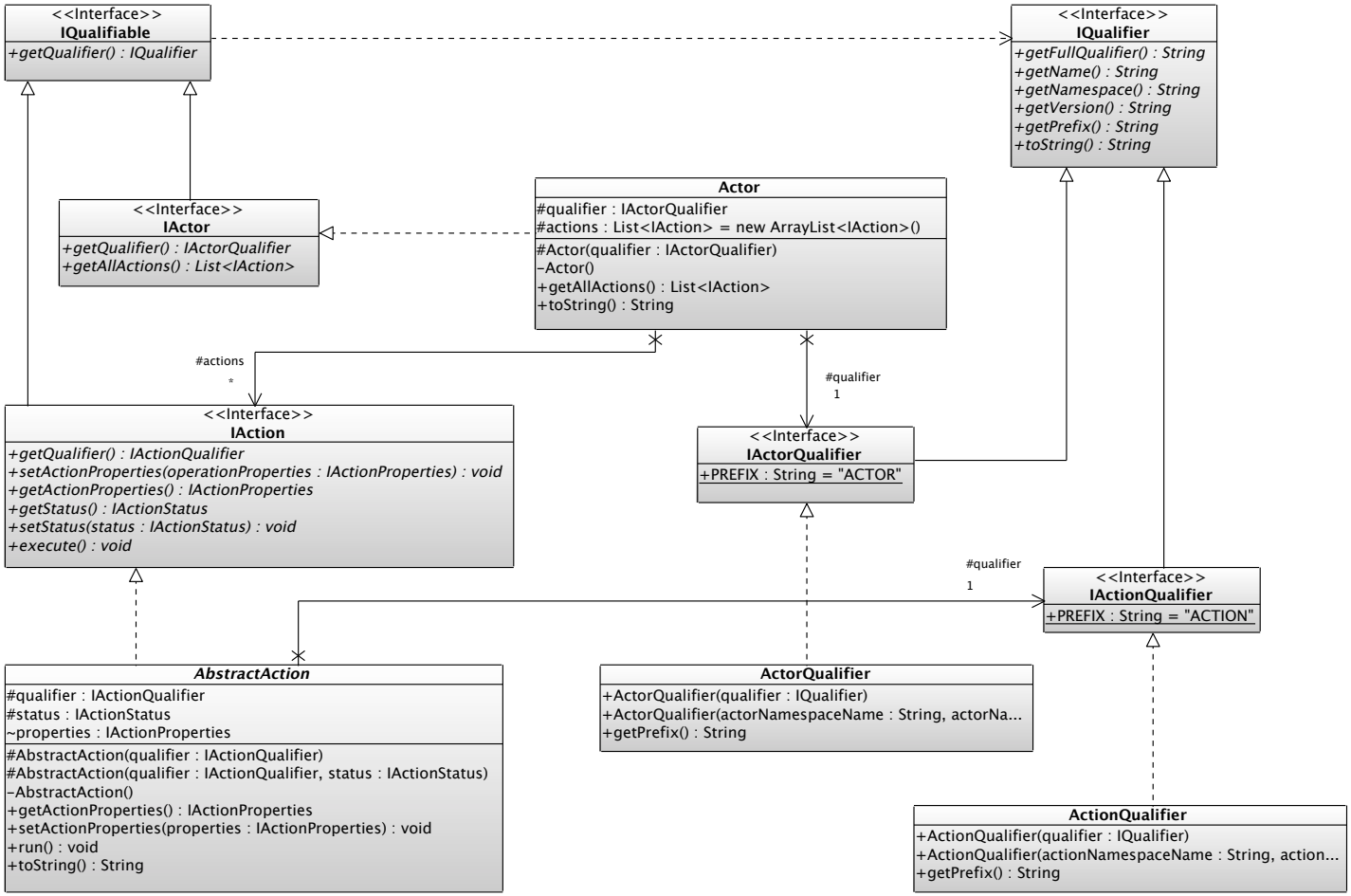

Abbildung 4.9: Struktur und Zusammenhang der modellierten Schnittstellen zur Qualifikation und Verwendung der Aktionen und Aktoren

Zur Ausführung von Aktionen wird die Möglichkeit geschaffen, diese optional auch mit Parametern versehen zu können. Die Parametrisierung der Aktionen wird durch die Übergabe von `IActionProperties` an den `execute` Aufruf in der Aktionsregistrierung vorgenommen. Die Methode `getDefaultProperties` ist zum Zugriff auf die Standardeigenschaften der Aktion erstellt worden und ermöglicht eine Vorgabe von Eigenschaften, die für jede Aktion eines Typs zutreffen. Die Klasse `ActionProperties` wird als abstrakte Klasse bereitgestellt, von der man, um eigene Aktionseigenschaften zu realisieren, erben kann. Diese Vorgabe vereinfacht die Implementation neuer Aktionen.

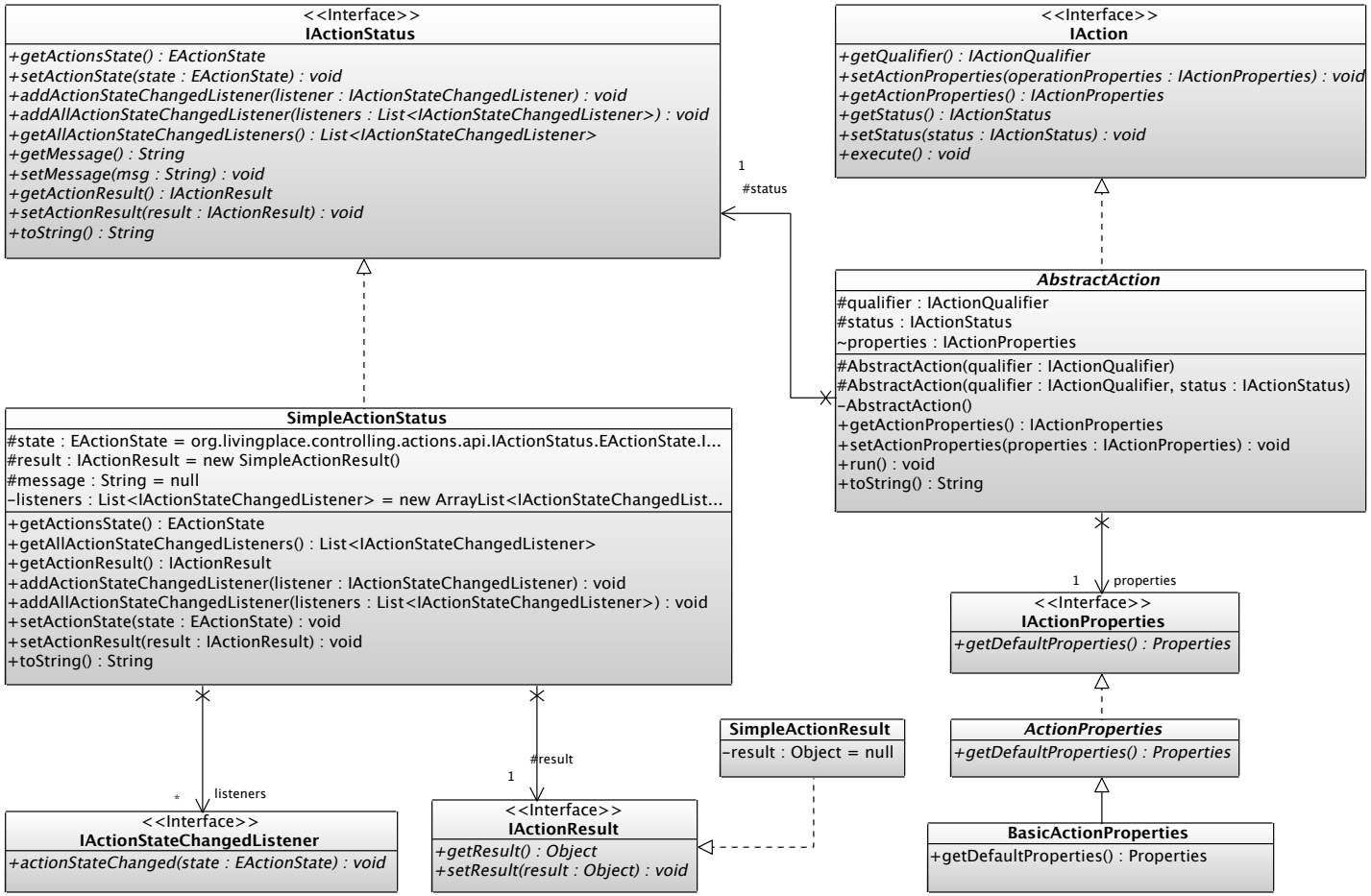

Abbildung 4.10: Klassenzusammenhang zwischen Aktionen, ihren Status und Eigenschaften

Im einfachsten Fall wird eine Aktion ohne Rückmeldung ausgeführt und es ist kein Ergebnis nötig, worauf es sich zu warten lohnt. Aktionen haben aber dennoch immer einen Status, mit dem der Aktionsfortschritt und dessen eventuelles Ergebnis überwacht werden kann. Der Status einer Aktion wird durch die `IActionStatus` Schnittstelle definiert. Eine Beispielimplementation der Schnittstelle ist durch `SimpleActionResult` bereitgestellt und bildet mit dem `SimpleActionResult` eine Definition, mit der eine neue Aktion in Betrieb genommen werden kann, vgl. Abb. 4.10.

4.5.3 Registrierungen für Aktionen und Informationen

Die Registrierungen dienen zwei unterschiedlichen Aspekten. Zum einen ist das der zentrale Punkt, an dem die Informationen und Aktionen, über die das System verfügt, gesammelt werden, zum anderen werden sie benötigt, um die konkreten Klassen für den dynamischen `ClassLoader` bereitzustellen. Die Registrierungen haben unterschiedliche Funktionen und werden im Detail in den folgenden Abschnitten erläutert.

4.5.3.1 Informationsregistrierung

Die Informationsregistrierung, implementiert nach der `IInformationRegistry`, ist ein Verzeichnis für alle Informationen, die dem System bekannt sind. Sobald ein Sensor die Informationen, die er zur Verfügung stellt, bei ihr registriert, werden diese unter ihrem Qualifier hinterlegt. Zur Information wird ein `IInformationListener` gespeichert, der vom Sensor verwendet wird, um eine neue Information zu propagieren. Dieser Listener wird benachrichtigt und verteilt die Information an alle Listener, die sich für diese Information angemeldet haben. Über diesen Mechanismus wird auch der `IInformationListener` der Faktbasis mit Informationsobjekten der Sensoren beliefert. Eine Übersicht der Kommunikation gibt die folgende Aufzählung, vgl. Abb. 4.11:

1. Die Informationsregistrierung erstellt bei der Bundleinitialisierung den Broadcast `IInformationListener`, welcher im weiteren Verlauf genutzt wird um den `DroolsManager` mit Informationsinstanzen zu beliefern.

2. Die `RuleEngine` ist der `BundleActivator` der CEP Maschine. Dieser erstellt zunächst den `DroolsManager`.

3. Ein `FactInsertionListener` wird erstellt und dem `DroolsManager` übergeben, um diesen später über die neuen Fakten zu unterrichten.

4. Der Sensor erstellt das `Information` Objekt mit dem eindeutigen `Information-Identifier`

5. Die `RuleEngine` ist nun vollständig initialisiert und registriert den `FactInsertion-Listener` bei der Informationsregistrierung, um ihn der Liste der Listener bekannt zu machen, die vom Broadcast Listener über neue Nachrichten informiert werden.

6. Ein SensorBundle wird gestartet und registriert die von ihm zur Verfügung gestellten Informationen bei der Informationsregistrierung.
    1. Durch Aufruf der `registerOnListener` Methode erhält der Sensor einen `IInformationListener` zurück, den er, wenn eine solche Information auftritt, benachrichtigt.

7. Ein realer Sensor im Labor, zum Beispiel ein Knopf oder Taster, meldet nun einen neuen Zustand und generiert eine Nachricht die ein Sensorbundle detektiert.
    1. Das Sensorbundle erstellt aus den Informationen des echten Sensors eine `Information` Instanz und ruft damit die Methode `sensedInformation` des Listeners für diese Informationen auf.

    1.1. Der `IInformationListener` der Informationsregistrierung informiert den Broadcast Listener.

8. Der Broadcast Listener informiert alle Listener die sich über die `addAllInformations-Listener` registriert haben, unter anderem den `FactInsertionListener` der `RuleEngine`.
    1. Der `FactInsertionListener` übergibt das Informationsobjekt nun der CEP Maschine und macht so die Information für die Regelevaluierung verfügbar.

9. Die CEP Maschine berücksichtigt die neue Information sobald sie das nächste Mal die `reason` Methode ausführt.

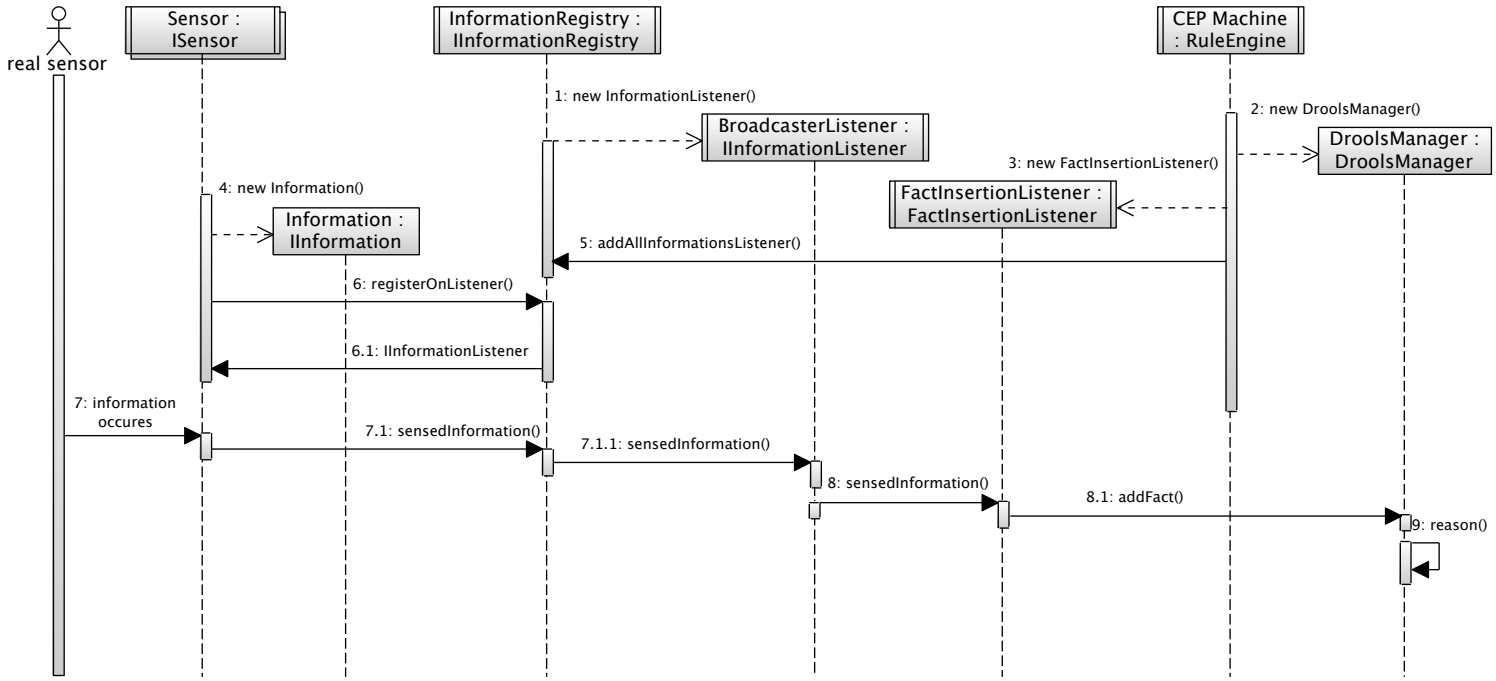

Abbildung 4.11: Verbreitung einer Information vom Sensor über die Informationsregistrierung bis zur Faktbasis

4.5.3.2 Aktionsregistrierung

Die Aktionsregistrierung, implementiert nach der `IActionRegistry` Schnittstelle, verwaltet das Verzeichnis aller Aktionen, die dem System bekannt sind. Wenn ein Aktor seine Aktionen registriert, werden diese von der Aktionsregistrierung ausführbar. Unter Verwendung eines Executor Frameworks werden die Aktionen ausgeführt, die von Regeln verwendet werden, vgl. (Goetz und Peierls, 2006, S. 113 ff). Die Aktionsregistrierung führt die Aktionen aus und befreit den Verfasser einer Regel davon, die Ausführung der Aktion selbst zu realisieren. Stattdessen wird die Aktionsregistrierung in der Faktbasis verfügbar gemacht und erlaubt die direkte Verwendung einer der `executeAction` Methoden in einer Regel, vgl. Abschnitt 4.5.4. Die `IActionRegistry` spezialisiert die `IRegistry<T>`, von der sie erbt, mit einer Erweiterung um die verschiedenen `executeAction` Methoden. Wie eine Aktion vom Aktor bis zur Ausführung gelangt, zeigt die folgende Aufzählung, vgl. Abb. 4.12:

1. Die Aktionsregistrierung erstellt einen `java.util.concurrent.Executor` mit einem fixen Thread Pool.
2. Ein Aktor erstellt die Aktion, die dem System zugänglich gemacht werden soll, mit dem `ActionQualifier`.
3. Der Aktor registriert die Aktion bei der `ActionRegistry`.
4. Der `FactInsertionListener` bekommt eine neue Information von einem der Sensoren und fügt ihn der Faktbasis hinzu, vgl. Abschnitt 4.5.3.1.
5. Die CEP Maschine bezieht den neuen Fakt mit in das Reasoning ein und bei einer Regel ist die Prämisse erfüllt.
6. In der Konklusion einer Regel wird eine Aktion, die ausgeführt werden soll, ermittelt und mit den für diese Aktion nötigen `ActionProperties` vorbereitet. Bei der Aktionsregistrierung wird die Aktion über einen `executeAction` Aufruf ausgeführt.
   6.1. Die `ActionRegistry` übergibt die auszuführende Aktion an das Executor Framework und lässt diese dort ausführen. Dieser Aufruf ist nicht blockierend und die CEP Maschine kann weitere Regeln evaluieren, ohne die Aktion selbst ausführen zu müssen.

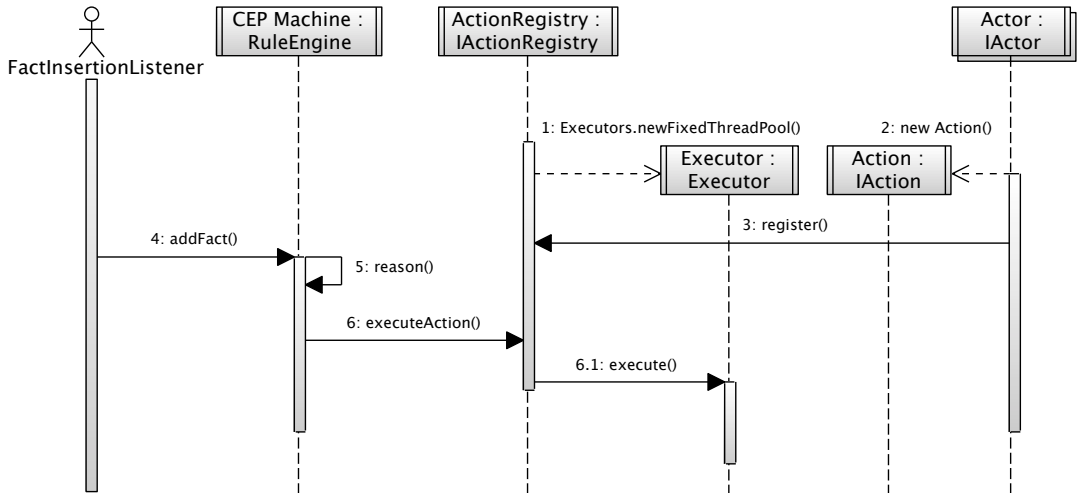

Abbildung 4.12: Auführen einer Aktion auf Basis einer evaluierten Regel

4.5.4 Drools Fusion und Dynamic Classloading

Das Bundle, das die CEP Maschine implementiert, ist unter dem Namensraum `*.knowledge.api` zu finden. Die Komponenten in diesem Bundle sind zum Teil in den Abschnitten 4.5.3.1 und 4.5.3.2 besprochen worden, haben aber dort keinen Einblick in deren Funktion gewährt. Zur Konfiguration der CEP Maschine wird der `DroolsManager` verwendet. Der `RegistryClassLoader` ist der `ClassLoader`, über den die Klassen aus den Registrierungen geladen werden. Der `FactInsertionListener` ist der `IInformationListener`, der von der Informationsregistrierung benachrichtigt wird und die Informationen als Fakten in die Faktbasis einfügt.

Die Faktbasis, `org.drools.KnowledgeBase`, stellt den zentralen Sammelpunkt für alle Informationen dar. Durch die Konfiguration mit der Option `EventProcessingOption.STREAM` wird sie so konfiguriert, dass sie zeitliche Abhängigkeiten in Zusammenhang bringen kann und dadurch als CEP Maschine genutzt werden kann.

Drools Fusion wird mit einem `org.drools.agent.KnowledgeAgent` konfiguriert. Der Agent wird verwendet, um das Verzeichnis für Regeln zu überwachen und neue Regeln mit in die Regelmaschine zu integrieren. Der Agent wird mit einer XML Datei konfiguriert, einem sog. `changeset`. Das `changeset` gibt dem Betreuer des Systems die Möglichkeit, einen beliebigen Ort für Regeldateien anzugeben, welches dann vom Agent überwacht

wird. Dem Agenten wird auch der `RegistryClassLoader` übergeben, um über ihn auch in den Registrierungen nach Klassen suchen zu können.

4.6 Fazit

Das hier präsentierte System stellt eine Lösung für die in den Anforderungen evaluierten Problemstellungen dar. Die drei Hauptbestandteile der Problemstellung, eine einheitliche Verwaltung und Abhängigkeitsauflösung der Softwareprojekte unter der Prämisse der Wiederverwendbarkeit, eine Übersicht zu allen vorhandenen Systemen und deren Status, sowie die vereinfachte Integration neuer Sensoren und Aktoren in das System, ist mit dieser Realisierung gelungen.

Die Wiederverwendbarkeit der Softwareprojekte wird durch den Einsatz eines Maven Proxies realisiert. Dieser bietet allen vorhandenen und zukünftigen Projekten die Möglichkeit, ihre Softwarebibliotheken in verschiedenen Versionen automatisiert auszuliefern. Auch eine gemeinsame Entwicklung und Integration neuer Funktionen wird durch den Einsatz einfacher und bietet Raum für Erweiterungen in Zukunft.

Eine Übersicht der Status aller Sensoren, Aktoren und Programme im System ist nicht nur implizit durch den Einsatz von OSGi entstanden, sondern kann durch die Erweiterung von vorhandenen Bundles, zum Beispiel durch eine Integration auf der Webseite des Apache Felix Frameworks, ausgebaut werden.

Die Integration einer CEP Maschine in das Laborumfeld zur Vereinfachung der Verhaltensintegration ist mit Drools Fusion entstanden. Unter Verwendung der Bundles zur Verwaltung von Sensoren, Informationen, Aktoren und Aktionen ist es gelungen, einen Mechanismus zu schaffen, mit dem ein neues Verhalten integriert werden kann, ohne das System programmatisch erweitern zu müssen, solange nur vorhandene Informationen und Aktionen dafür benötigt werden.

Das Design und die Realisierung des Systems ist hiermit abgeschlossen und das folgende Kapitel schließt mit der Integration in das Laborumfeld des Living Place Hamburg daran an.

5 Integration im Living Place Hamburg

In diesem Abschnitt wird erläutert, wie die im Kapitel 4 erarbeiteten Lösungen in das Laborumfeld integriert werden. Es werden Konzepte zur praktischen Verwendung des Systems vorgestellt, die in Zukunft als Richtlinie verwendet werden können, um einen Systemüberblick zu erhalten und die möglichen Erweiterungen zu realisieren.

5.1 Maven-Proxy Server

Die Anforderung an eine einheitliche Projektstruktur wurde, wie in Abschnitt 4.1 erläutert, mit der Integration eines Maven-Proxies in den Living Place Hamburg gelöst. Die Notwendigkeit für einen Maven-Proxy ist mit der Umsetzung der Abhängigkeitsauflösung gegeben und kann auch für andere abseits des Living Place Hamburg liegende Projekte genutzt werden.

Es gibt verschiedene Möglichkeiten, einen Maven-Proxy in Betrieb zu nehmen, allerdings ist hier darauf zu achten, dass die Anbindung in vorhandene Infrastruktur möglich sein muss. Da im Labor auch Komponenten existieren, die mit C# implementiert wurden, ist eine Möglichkeit, diese über den selben Mechanismus auszuliefern, wünschenswert.

Ein solcher Maven-Proxy wird dem Living Place Hamburg von der Firma Sonatype zur Verfügung gestellt und kann kostenlos genutzt werden. Das Produkt nennt sich Nexus und ist mit Plugins um die nötige Funktionalität zum Ausliefern von C# Paketen erweiterbar[1]. Der eingerichtete Maven Proxy ist unter der folgenden URL zu erreichen:

`http://devsupport.informatik.haw-hamburg.de/nexus`

Die nötigen Schritte, um diesen Maven Proxy zu verwenden, wurden im Wikimedia des Living Place Hamburg dokumentiert[2]. Mit Hilfe dieses Maven-Proxies werden in Zukunft die Softwareprojekte im Laborkontext erstellt und ausgeliefert. Eine Nutzerverwaltung, die mit dem Wikimedia zu vereinbaren ist, findet über den LDAP-Service vom Living Place statt und kann um zusätzliche Komponenten erweitert werden, vgl. Abb.5.1.

[1] `http://www.sonatype.com/Products/Nexus-Professional`; abgerufen am: 25.03.2013
[2] `http://livingplace.informatik.haw-hamburg.de/wiki/index.php/Maven_Anleitung`; abgerufen am: 10.12.2012

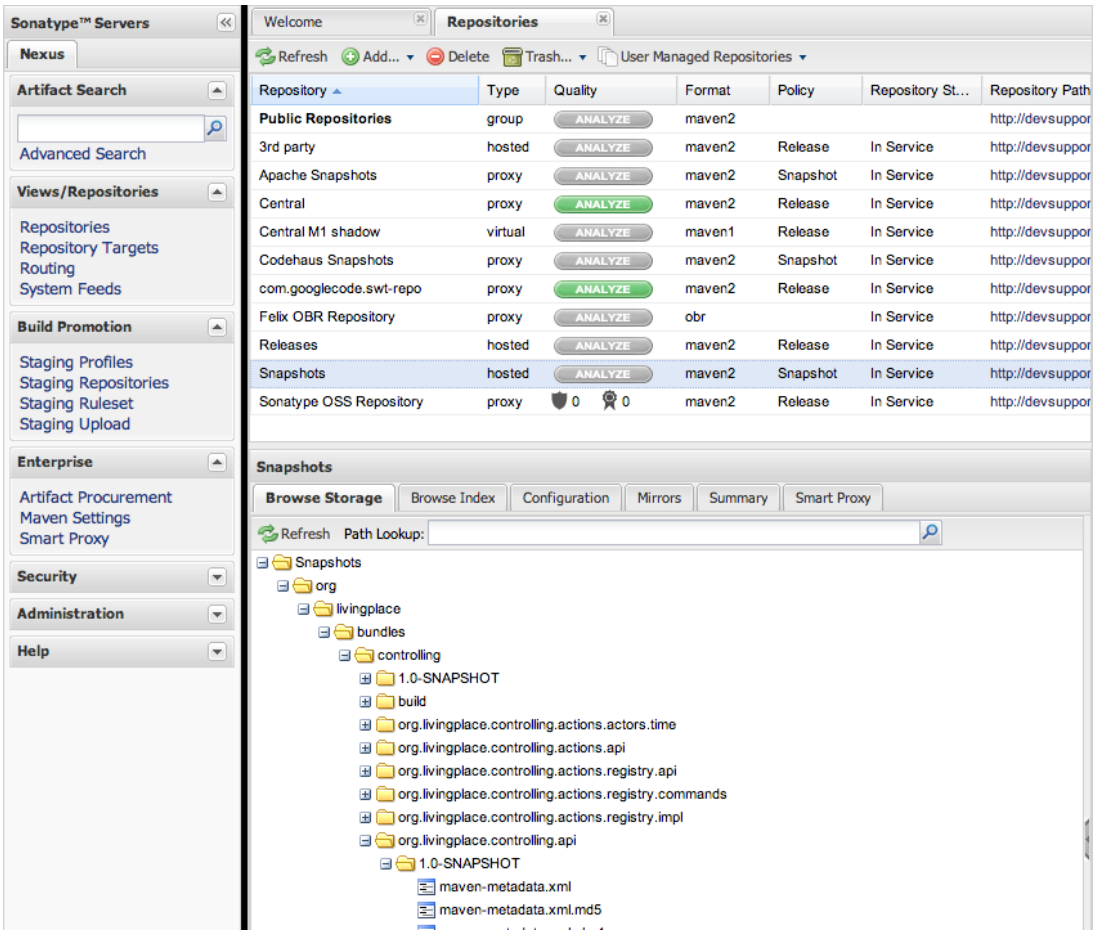

Abbildung 5.1: Sonatype Maven-Proxy zum Ausliefern der Living Place eigenen Bundles und Maven Artifakte

Der Maven-Proxy löst damit das Problem der Verteilung von Projekten innerhalb des Labors und macht Bibliotheken und Applikationen zuverlässig referenzier- und auslieferbar.

5.2 Apache Karaf als Laufzeitumgebung

Im Designabschnitt ist die Entscheidung bezüglich eines Applikationscontainers auf Apache Felix gefallen. Apache Felix bietet der Laborumgebung eine Implementation der OSGi Service Spezifikation. Um die Funktionalität des Applikationscontainers nicht auf die reinen Kernkomponenten zu beschränken, wird dieser in der Laborumgebung jedoch mit Apache Karaf erweitert. Apache Karaf ist eine Erweiterung von Apache Felix und wird von der Apache Software Foundation (ASF) als ein Subprojekt von Felix geführt. Karaf kann mit Eclipse Equinox und Apache Felix betrieben werden und ist damit nicht von der gewählten Laufzeitumgebung abhängig, vgl. Abb. 5.2.

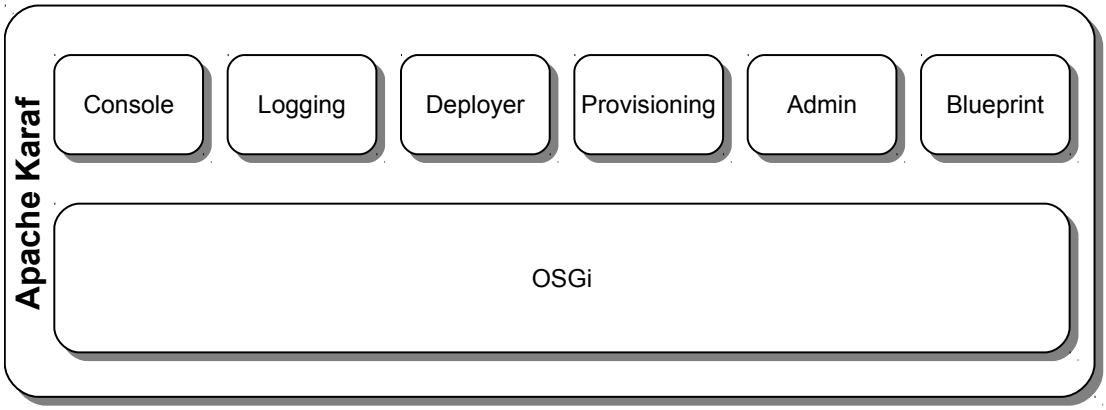

Abbildung 5.2: Apache Karaf Übersicht

5.2.1 Features

Die Vorteile der Apache Karaf Laufzeitumgebung für OSGi-Bundles sind laut der Karaf Website[3] folgende:

Hot deployment Karaf unterstützt das sog. "hot deployment"von OSGi Bundles durch monitoring des Auslieferungsverzeichnisses(`KARAF_HOME/deploy`). Jedesmal wenn

[3] `http://karaf.apache.org`; abgerufen am: 25.03.2013

eine neue JAR-Datei in dem Verzeichnis abgelegt wird, wird diese Datei in der Karaf Laufzeitumgebung installiert. Im Anschluss kann das veränderte Bundle über die Karaf Shell gestartet werden.

Dynamic configuration Servicekonfigurationen werden laut der OSGi Spezifikation durch den `ConfigurationAdmin` Service vorgenommen, vgl. Anhang 2 Item 104. Die Konfiguration mit Karaf kann allerdings auch mittels PropertyFiles geschehen. Unter `KARAF_HOME/etc` können Konfigurationsdateien zur Laufzeit verändert werden. Diese werden dann von Karaf mittels ConfigurationAdmin Service an die Komponenten weiter gegeben und ermöglichen einen möglichst kurzen Testzyklus.

Logging System Karaf setzt ein zentralisiertes Logging Backend ein, mit dem über Log4J ein zentralisiertes Log geführt werden kann.

Provisioning Das Bereitstellen von Bibliotheken oder Applikationen kann mit Karaf über eine Vielzahl von Mechanismen geschehen. Über diese Mechanismen lassen sich Bundles downloaden, installieren und starten. Mit Hilfe eines eigenen Maven-Proxies besteht die Möglichkeit, Bundles direkt aus dem Repository zu referenzieren und zu installieren.

Native OS Integration Karaf kann als Service über das Betriebsystem mitgestartet werden und ermöglicht darüber eine saubere Integration in den Lifecycle des Betriebssystems.

Extensible Shell console Die Karaf Shell ist eine einfach erweiterbare Shell, die es dem Entwickler ermöglicht, seine Applikationen oder Bundles mit Kommandos auszurüsten und den Zugriff während der Laufzeit zu vereinfachen. Um die Kommandos nicht unübersichtlich zu vermischen, ist es möglich, diese zu gruppieren und zusammenzufassen.

Remote access Ein weiteres Feature von Karaf ist die integrierte Secure SHell (SSH). Neben einer installierbaren Web-Konsole ist der Zugriff über einen SSH Daemon sicher und zuverlässig. Man kann einen beliebigen SSH Client verwenden um direkt mit der Karaf Shell verbunden zu werden.

Security framework Karaf realisiert die Sicherheitskonzepte mit dem Java eigenen Java Authentication and Authorization Service (JAAS).

Managing instances Karaf bietet die Möglichkeit mehrere Instanzen zu verwalten. Dieser Mechanismus ermöglicht das Erstellen, Löschen, Starten und Stoppen der

Karaf Instanzen mittels Konsole und ermöglicht eine einfache Erweiterung des Systems.

Durch den Einsatz von Karaf ist das Labor weiterhin flexibel, was den Application Container angeht und kann trotzdem viele Merkmale nutzen, um eine schnelle und einfache Projektintegration zu gewährleisten.

5.2.2 Web-Konsole

Apache Karaf hat eine Web-Konsole, die sich als Standardfeature direkt installieren lässt(erreichbar unter `http://<host>:8080/system/console`). Über diese Webseite lässt sich ein Filter auf alle installierten Bundles anwenden, um einen Überblick über die Status der unterschiedlichen Bundles zu bekommen, vgl. Abb. 5.3. Über diese Auflistung lassen sich Bundles, die Funktionalitäten im Labor abbilden, schnell und einfach finden und inspizieren. Bundleabhängigkeiten können direkt über das Webinterface erkannt werden und auch der Neustart-, Update- oder Löschvorgang kann von hier aus gestartet werden.

Zur Analyse von Fehlern und Vorgängen im Laborbetrieb ist es möglich, das zentralisierte Log einzusehen. Die Web-Konsole hat hierfür ebenfalls einen eigenen Tab, über den die Nachrichten gefiltert werden können und mit dem man auch StackTraces einsehen kann, vgl. Abb. 5.4.

5.2.3 Kommandos

Ein Feature, was Apache Felix sowie auch Apache Karaf mit sich bringen, ist die Integration von Kommandos. Die OSGi Laufzeitumgebungen haben eine Shell, in der man Kommandos nutzen kann, um mit den einzelnen Bundles zu interagieren. Dieses Feature bietet die Möglichkeit, eigene Kommandos zu entwickeln und eine Schnittstelle zur Anwendung zu erstellen, die während der Laufzeit genutzt werden kann, um interne Zustände zu analysieren oder zu ändern.

Kommandos können gruppiert werden und im Kontext des Living Place Hamburg wird der sog. Scope `"lp"` verwendet. Für diese Arbeit wurden Kommandos erstellt, die Auskunft über die Aktor- und Sensorregistrierung geben sowie ein Kommando, das die Registration eines Listeners für Informationen ermöglicht, vgl. Listing 5.1. Die Hilfe kann auch dem `help` Kommando zur Laufzeit abgerufen werden.

```
1 g! help lp:act:reg:show
2     show - shows all registered actions
```

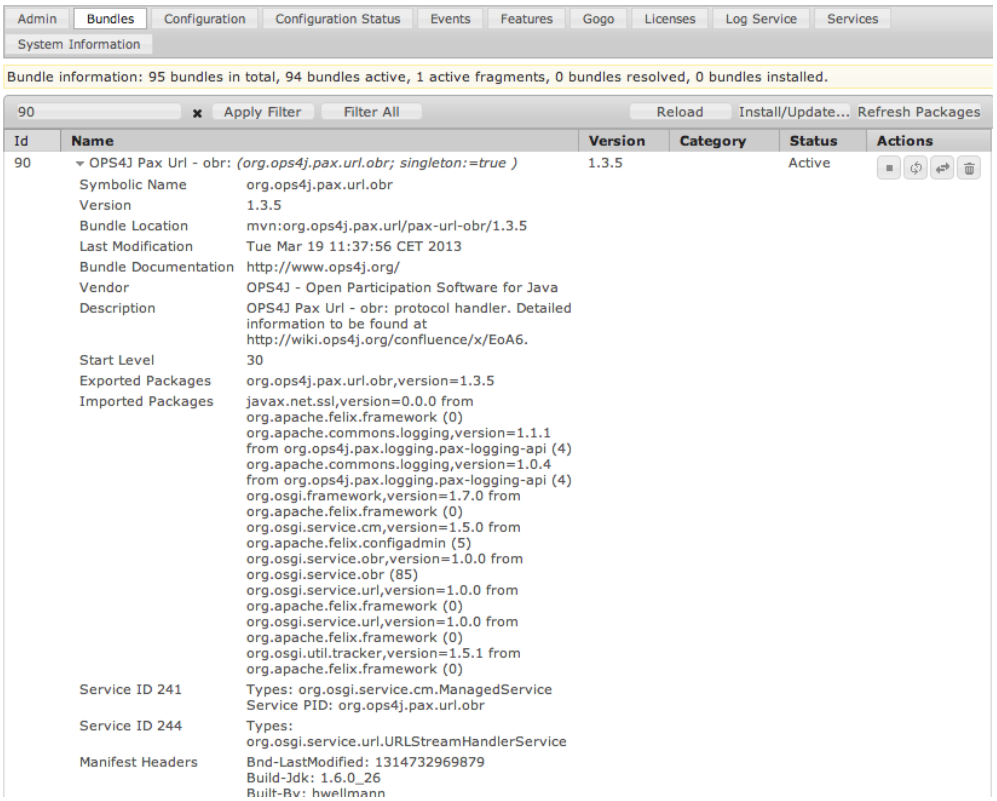

Abbildung 5.3: Apache Karaf Web-Konsole, beispielhafte Detailansicht eines Bundles

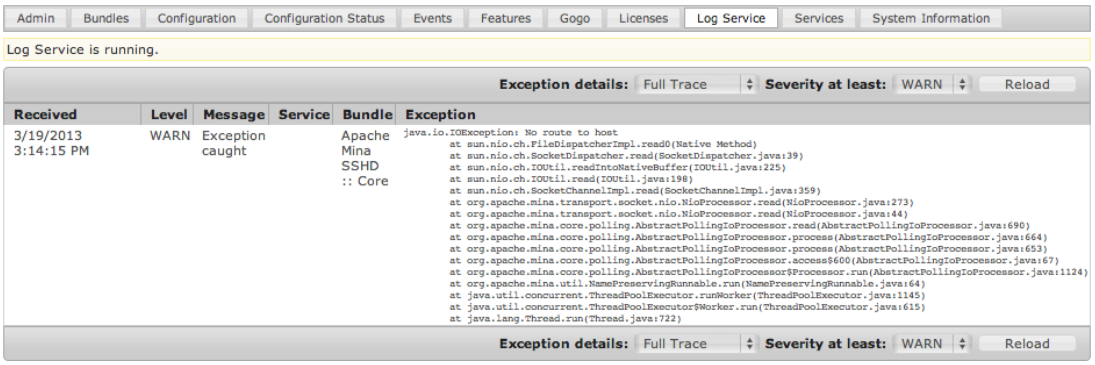

Abbildung 5.4: Apache Karaf Web-Konsole, beispielhafte LogService Ansicht

```
3         scope: lp
4
5 g! help lp:act:reg:execute
6     execute - executes an action qualified by the parameters
7         scope: lp
8         parameters:
9             String    full qualified action in the form: "<
                  namespace>.<name>:<version>"
10
11 g! help lp:inf:reg:show
12     show - shows all registered informations
13         scope: lp
14
15 g! help lp:inf:reg:last
16     last - shows the last seen informations for the qualified
           information
17         scope: lp
18         parameters:
19             String    full qualified information in the form: "<
                  namespace>.<name>:<version>"
20
21 g! help lp:inf:reg:registerInformationPrinter
22     registerInformationPrinter - registeres an Console Printing
           Listener to the given InformationQualifier in order to see
            the sensing of this information
23     scope: lp
24     parameters:
25         String    full qualified information in the form: "<
              namespace>.<name>:<version>"
26
27 g! help lp:inf:reg:unregisterInformationPrinter
28     unregisterInformationPrinter - unregisteres an Console
           Printing Listener to the given InformationQualifier
29     scope: lp
30     parameters:
31         String    full qualified information in the form: "<
              namespace>.<name>:<version>"
```

Listing 5.1: Kommandos der Bundles die in dieser Arbeit entwickelt wurden

Die Möglichkeiten für weitere Kommandos sind offen und die Optionen ermöglichen nahezu grenzenlos komplexe Kommandos, vgl. Listing 5.2. Über Kommandos können auch weitere Mechanismen anderer Bundles, die außerhalb dieser Arbeit entwickelt werden, integriert werden.

```
g! lp:inf:position:stats
    position sensor received 12 positions in the last minute

    overall    last interval     average     tags seen
    12931      12                11,6        013, 005

    configuration:
        average algorithm: median
        window slices:          4
```

Listing 5.2: Beispiel einer Kommandoausgabe von einem fiktiven Positionsensor

5.2.4 Fazit

Die Anforderungen an diese Arbeit sind zum einem im Design und der praktischen Realisierung der Arbeit gelöst, zum anderen aber auch in der Wahl der Komponenten des Gesamtsystems. Eine Anforderung ist es, eine Möglichkeit zu schaffen, den Systemzustand erfassen zu können, vgl. Abschnitt 3.6. Die Möglichkeit, die Status der Bundles bzw. Services auf dem Applikationscontainer zu erlangen, konnte hier durch die Wahl der Komponenten im System realisiert werden.

Zum Ende der Realisierungsphase dieser Arbeit konnten im praktischen Einsatz bereits Erfahrungen mit der Wahl dieser Komponenten gesammelt werden. Die Funktionen des Gesamtsystems wurden in mehreren Vorträgen den Laborteilnehmern vorgestellt. Der erste Eindruck ist sehr positiv ausgefallen und es konnten erste Prototypen mit den gewählten Komponenten erstellt werden.

6 Schluss

In diesem Kapitel erfolgt als erstes eine Zusammenfassung der Arbeit. In einem Ausblick werden Ideen für die Zukunft beschrieben, die im Rahmen dieser Arbeit nicht realisiert werden konnten. Mit einem Fazit schließt die Arbeit ab.

6.1 Zusammenfassung

In dieser Arbeit konnten nach einer Analyse der aktuellen Laborumgebung und der integrierten Projekte Anforderungen formuliert werden, die eine Entwicklung von Projekten im Laborrahmen erheblich vereinfachen können. In den Anforderungen konnten verschiedene Ebenen der Problemstellung identifiziert und damit erste Ansätze zu einer Lösung konzipiert werden. Mit Hilfe von drei Szenarien wurde aus diesen Ansätzen ein Gesamtkonzept formuliert, das in der Realisierung zur praktischen Umsetzung kam, vgl. Abschnitt 3.3.

Die Anforderungen konnten Schwierigkeiten beim Erstellen, Pflegen und Warten von Projekten und vor allem der Integration in die bereits existierende Laborumgebung aufzeigen. Auch die Sensor- und Aktorintegration war völlig individuell und folgte keinem erkennbaren Schema. Damit einhergehend gab es keine Möglichkeit, neues Verhalten ohne größeren Programmieraufwand in die Laborumgebung zu integrieren. Eine Übersicht über die Zustände der Projekte im Living Place Hamburg gab es nicht und auch keine Möglichkeit, diese zu erfragen, vgl. Abschnitt 3.6.

Unter Einsatz von Maven als Werkzeug zum Erstellen und Ausliefern von Projektbibliotheken oder Applikationen konnte die vorliegende Arbeit der Anforderung nach einheitlicher Projektstruktur gerecht werden, vgl. Abschnitt 4.1. Damit ein Überblick über die vorhandenen Komponenten im Labor entstehen kann, wurde ein OSGi-Applikationscontainer gewählt, der Möglichkeiten zur Erweiterung bietet und eine große Anzahl von Grundfunktionen integriert hat, vgl. Abschnitt 4.2. Um neue Sensoren oder Aktoren in die Laborumgebung einzugliedern, wurde in der praktischen Umsetzung der Arbeit eine Reihe von OSGi Bundles erstellt. Diese Software Services ermöglichen es mit Hilfe von Regeln, neues Verhalten zu implementieren, vgl. Abschnitt 4.3. Die

eingesetzte CEP Maschine verfügt über die nötigen Mechanismen, um einen Einsatz über den umgesetzten ClassLoader zu realisieren. Mit Prototypen für Sensoren und Aktoren konnte gezeigt werden, dass die Umsetzung funktioniert und integrierbar ist, vgl. Abschnitt 4.4.

Im Integrationskapitel wurden die Aspekte, die außerhalb der programmatischen Entwicklung dieser Arbeit lagen, wie zum Beispiel der Maven-Proxy und ein Apache Karaf Applikationscontainer erläutert, vgl. Abschnitt 5. Beide Komponenten konnten im Rahmen dieser Arbeit in Betrieb genommen werden und sind zum Zeitpunkt der Fertigstellung dieser Arbeit bereits im praktischen Einsatz. Durch zeitnahe Integration in die Laborumgebung konnten Anregungen für den Ausblick gesammelt werden, vgl. Abschnitt 5.2.4.

Die vorgestellten Konzepte sind auch in einem Umfeld außerhalb des Living Place Hamburg anwendbar. So könnten Alice, Bob und Carol auch Arbeitskollegen statt Studenten sein, denen in einer Arbeitsumgebung ähnliche Problemstellungen begegnen. Die in dieser Arbeit eingesetzten Mittel sind auch über den Rahmen der Laborumgebung hinaus hilfreich.

6.2 Ausblick und Fazit

Die Einführung der Komponenten hat Erkenntnisse gebracht, mit denen im Anschluss an diese Arbeit weitere Funktionen im Labor integriert werden können. An unterschiedlichen Stellen wird die Architektur und auch die praktische Realisierung kontinuierlich weiter entwickelt.

Die Laborarchitektur kann zum Beispiel mit einem System zur Continuous Integration erweitert werden. Ein solches System ermöglicht das Testen neuer Bundles im realistischen Laborumfeld, ohne das reale Labor zu stören. Die Reproduzierbarkeit von Fehlerzuständen kann mit einem solchen Werkzeug gewährleistet werden und vereinfacht die Fehlersuche in Teilsystemen. Die Vorteile eines solchen Systems würden sich positiv auf die Zuverlässigkeit neuer Komponenten im Labor auswirken und ermöglichen es den Entwicklern realitätsnah, neue Komponenten zu entwickeln und von Anfang an integrierbar zu halten.

Eine Komponente, die den Systemüberblick verbessert, könnte einen weiteren Tab auf der Apache Karaf Web-Konsole einrichten. Ein Tab, der alle Living Place Komponenten zusammenfasst und deren interne Zustände visualisiert, würde helfen, den Systemzustand

noch schneller zu erfassen. Auch ein Überblick über alle Fakten in der Faktbasis der CEP Maschine wäre wünschenswert, um Abschätzungen über den Aufwand neuer Regeln anzustellen.

Damit der Einstieg in die Laborumgebung noch einfacher gestaltet wird, ist es denkbar, einen eigenen Architekturtyp für Maven zu entwerfen. Damit ist es möglich, Projektskelette direkt zu erzeugen und damit fertige Verzeichnisstrukturen vorzugeben.

Wenn viele Komponenten auf der Apache Karaf Laufzeitumgebung zum Einsatz kommen, können die Ressourcen der virtuellen Maschine nicht mehr ausreichen. Um dieser Ressourcenknappheit vorzubeugen, kann man verteilte OSGi Frameworks einsetzen. Apache Karaf bietet bereits die Möglichkeit, weitere Instanzen zum verteilten Einsatz zu bringen, aber die praktische Umsetzung wurde bisher nicht getestet.

Der Living Place Hamburg wird ein zukunftsweisendes Labor für Untersuchungen im Kontext von Smart Homes, Context Aware Computing und Ubiquitous Computing bleiben. Diese Arbeit konnte einen Beitrag zu Infrastruktur und Entwicklungsstandards leisten. Neue Projekte können in Zukunft von dieser Infrastruktur profitieren.

Literaturverzeichnis

[Bornemann 2011] Bornemann, Sven: *Android-basierte Smart Home Interaktion am Beispiel einer Gegensprechanlage*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2011

[Brachman und Levesque 2004] Brachman, Ronald ; Levesque, Hector: *Knowledge Representation and Reasoning*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2004. – ISBN 1558609326

[Cormen u. a. 2009] Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L. ; Stein, Clifford: *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. – ISBN 0262033844, 9780262033848

[Dearman und Pierce 2008] Dearman, David ; Pierce, Jeffery S.: It's on my other computer!: computing with multiple devices. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA : ACM, 2008 (CHI '08), S. 767–776. – URL http://doi.acm.org/10.1145/1357054.1357177. – ISBN 978-1-60558-011-1

[Diederichs 2005] Diederichs, H.: *Komplexitätsreduktion in der Softwareentwicklung: Ein systemtheoretischer Ansatz*. Books on Demand, 2005 (DSOR-Beiträge zur Wirtschaftsinformatik). – URL http://books.google.de/books?id=O7ZLCkEaytUC. – ISBN 9783833417900

[Dreschke 2011] Dreschke, Oliver: *Entwicklung kontextsensitiver Möbel für intelligente Wohnumgebungen*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2011

[Fabbricatore u. a. 2011] Fabbricatore, C. ; Zucker, M. ; Ziganki, S. ; Karduck, A.P.: Towards an unified architecture for smart home and Ambient Assisted Living solutions: A focus on elderly people. In: *Digital Ecosystems and Technologies Conference (DEST), 2011 Proceedings of the 5th IEEE International Conference on*, 31 2011-june 3 2011, S. 305 –311

[Forgy 1982] Forgy, Charles: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. In: *Artificial Intelligences* 19 (1982), Nr. 1, S. 17–37. – URL http://dx.doi.org/10.1016/0004-3702(82)90020-0. – ; abgerufen am: 21.11.2012. – ISSN 0004-3702

[Georgantas u. a. 2005] Georgantas, N. ; Mokhtar, S.B. ; Bromberg, Y. ; Issarny, V. ; Kalaoja, J. ; Kantarovitch, J. ; Gerodolle, A. ; Mevissen, R.: The Amigo Service Architecture for the Open Networked Home Environment. In: *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on*, 2005, S. 295 –296

[Goetz und Peierls 2006] Goetz, B. ; Peierls, T.: *Java concurrency in practice*. Addison-Wesley, 2006. – URL http://books.google.de/books?id=6LpQAAAAMAAJ. – ISBN 9780321349606

[Hall u. a. 2011] Hall, R.S. ; Pauls, K. ; McCulloch, S. ; Savage, D.: *OSGi in Action: Creating Modular Applications in Java*. O'Reilly Media, 2011 (Manning Pubs Co Series). – URL http://books.google.de/books?id=y3lPPgAACAAJ. – ISBN 9781933988917

[Janse 2008] Janse, Maddy D.: Amigo Final Report / INFORMATION SOCIETY TECHNOLOGIES (IST). 2008. – Forschungsbericht. http://cordis.europa.eu/documents/documentlibrary/115484471EN6.pdf

[JBoss Drools Team 2012] JBoss Drools Team: *JBoss Drools Fusion Documentation*. 2012. – http://docs.jboss.org/drools/release/5.5.0.Final/drools-fusion-docs/pdf/drools-fusion-docs.pdf; abgerufen am: 13.01.2013

[Jiang u. a. 2004] Jiang, Li ; Liu, Da-You ; Yang, Bo: Smart home research. In: *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on* Bd. 2, aug. 2004, S. 659 – 663 vol.2

[Kecher 2011] Kecher, Christoph: *UML 2 das umfassende Handbuch*. Galileo Press, 2011. – URL https://www.amazon.de/UML-umfassende-Handbuch-Galileo-Computing/dp/383621752X/ref=sr_1_1?s=books&ie=UTF8&qid=1348497866&sr=1-1. – ISBN 9783836217521 383621752X

[Kreuzer 2012] Kreuzer, Kai: *openHAB Architecture*. http://code.google.com/p/openhab/wiki/Architecture?tm=6. August 2012. – abgerufen am: 18.02.2013

[von Luck u. a. 2010] Luck, Prof. Dr. K. von ; Klemke, Prof. Dr. G. ; Gregor, Sebastian ; Rahimi, Mohammad A. ; Vogt, Matthias: A place for concepts of IT based modern living / University of Applied Sciences Hamburg. 2010. – Forschungsbericht. http://livingplace.informatik.haw-hamburg.de/content/LivingPlaceHamburg_en.pdf; abgerufen am: 19.11.2012

[Marit Hansen und Meissner 2008] Marit Hansen, S.M. ; Meissner, S.: *Verkettung digitaler Identitäten*. Lulu Incorporated, 2008. – URL http://books.google.de/books?id=k49bPg1_vYcC. – ISBN 9783000234064

[Nakashima u. a. 2010] Nakashima, Hideyuki ; Aghajan, Hamid ; Augusto, Juan C.: *Handbook of Ambient Intelligence and Smart Environments*. 2nd. Springer Publishing Company, Incorporated, 2010. – 1171–1199 S. – ISBN 0387938079, 9780387938073

[OSGi Alliance 2009a] OSGi Alliance, The: *OSGi Service Platform Core Specification*. June 2009. – URL http://www.osgi.org/download/r4v42/r4.core.pdf. – abgerufen am: 12.12.2012

[OSGi Alliance 2009b] OSGi Alliance, The: *OSGi Service Platform Service Compendium*. June 2009. – URL http://www.osgi.org/download/r4v42/r4.core.pdf. – abgerufen am: 12.12.2012

[Ossher u. a. 2010] Ossher, J. ; Bajracharya, S. ; Lopes, C.: Automated dependency resolution for open source software. In: *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, may 2010, S. 130 –140

[Otto und Voskuhl 2010] Otto, Kjell ; Voskuhl, Sören: Entwicklung einer Architektur für den Living Place Hamburg. In: *Projektbericht* Sommersemester 10 (2010), S. 21. – http://users.informatik. haw-hamburg.de/~ubicomp/projekte/master2010-proj1 /otto_voskuhl.pdf; abgerufen am: 25.11.2012

[Otto und Voskuhl 2011] Otto, Kjell ; Voskuhl, Sören: Weiterentwicklung der Architektur des Living Place Hamburg. In: *Projektbericht* Wintersemester 10/11 (2011), S. 21. – http://users.informatik. haw-hamburg.de/~ubicomp/projekte/master10-11-proj2 /otto-voskuhl.pdf; abgerufen am: 25.11.2012

[Parnas 1972] Parnas, D. L.: On the criteria to be used in decomposing systems into modules. In: *Commun. ACM* 15 (1972), Dezember, Nr. 12, S. 1053–1058. – http://doi.acm.org/10.1145/361598.361623; abgerufen am: 20.11.2012. – ISSN 0001-0782

[Pautz 2011] Pautz, Alexander: *Kabellose, stromsparende Sensornetzwerke im Bereich Ambient Intelligence*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2011

[Quast 2011] Quast, Oliver: *Berührungslose Schlafphasenerkennung zur Integration in ein Smart-Home*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2011

[Schilit u. a. 1994] Schilit, B. ; Adams, N. ; Want, R.: Context-aware computing applications. In: *Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on*, dec 1994, S. 85 –90

[Snyder u. a. 2011] Snyder, B. ; Bosanac, D. ; Davies, R.: *ActiveMQ in Action*. Manning, 2011 (Manning Pubs Co Series). – URL http://books.google.de/books?id=_jjCPwAACAAJ. – ISBN 9781933988948

[Sonatype 2008] Sonatype, Company: *Maven: The Definitive Guide*. O'Reilly Media, 2008. – 470 S. – URL http://www.sonatype.com/books/maven-book/reference/

[Teske 2011] Teske, Philipp: *Ein Multisensor-System zur Sturzerkennung*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2011

[Voskuhl 2011] Voskuhl, Sören: *Modellunabhängige Kontextinterpretation in einer Smart Home Umgebung*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2011

[Walzer u. a. 2007] Walzer, Karen ; Schill, Alexander ; Löser, Alexander: Temporal constraints for rule-based event processing. In: *Proceedings of the ACM first Ph.D. workshop in CIKM*. New York, NY, USA : ACM, 2007 (PIKM '07), S. 93–100. – URL http://doi.acm.org/10.1145/1316874.1316890. – ISBN 978-1-59593-832-9

[Weiser 1991] Weiser, Mark: The Computer for the 21st Century. In: *Scientific American* 265 (1991), September, Nr. 3, S. 94–?? (Intl. ed. 66–75). – ISSN 0036-8733 (print), 1946-7087 (electronic)

[Weiser 1993] Weiser, Mark: Some computer science issues in ubiquitous computing. In: *Commun. ACM* 36 (1993), Juli, Nr. 7, S. 75–84. – URL http://doi.acm.org/10.1145/159544.159617. – ISSN 0001-0782

[Wu u. a. 2008] Wu, Jiankun ; Huang, Linpeng ; Wang, Dejun ; Shen, Fei: R-OSGi-based architecture of distributed smart home system. In: *Consumer Electronics, IEEE Transactions on* 54 (2008), august, Nr. 3, S. 1166 –1172. – ISSN 0098-3063

Abkürzungsverzeichnis

AAL Ambient Assisted Living. 7, 27, 28

ACL Abstract Connection Layer. 29

AMIGO AMbient IntelliGence fOr the networked home environment. 27, 28

API Application Programming Interface. 10, 39, 44, 53, 54

ASF Apache Software Foundation. 74

CEP Complex Event Processing. 5, 12, 13, 35, 36, 48–50, 52, 54, 55, 60, 66, 67, 69–71, 81, 82

DSL Domain Specific Language. 37

EU Europäische Union. 27, 28

FIPA Foundation for Intelligent Physical Agents. 41

HAW Hochschule für Angewandte Wissenschaften Hamburg. 1, 41

HD High Definition. 15

IDE Integrated Development Environment. 22, 24, 25, 34, 36, 38, 42, 51

JAAS Java Authentication and Authorization Service. 75

JAR Java ARchive. 25, 38, 42, 44, 45, 75

JSON JavaScript Object Notation. 20, 21

JVM Java Virtual Machine. 41

openHAB open Home Automation Bus. 27, 30–33

OSGi Open Services Gateway initiative. 5, 27–30, 32, 35, 41–47, 49, 50, 52, 53, 56, 57, 62, 71, 74–76, 80, 82, 94

OWL Web Ontology Language. 29

PERSONA PERceptive Spaces prOmoting iNdependent Aging. 27–30, 32

POM Project Object Model. 38–40, 51, 90, 91

QOS Quality of Service. 27

RCP Rich Client Platform. 30

REST REpresentational State Transfer. 31

SAIL Sensor Abstraction and Integration Layer. 29, 32

SD Standard Definition. 15

SSH Secure SHell. 75

UML Unified Modeling Language. 43

URL Uniform Resource Locator. 42, 72

WM working memory. 11

WME working memory element. 11

XML eXtended Markup Language. 57, 70, 94

Anhang

1 Maven

1.1 Maven Standard POM

Die Standard POM-Datei aus Listing 3, und die Verzeichnisstrukture aus Listing 2 lässt sich mit Befehl aus Listing 1 erstellen.

```
bash> mvn archetype:create \
        -DgroupId=org.livingplace.maven-tutorial \
        -DartifactId=project-creation \
        -DarchetypeArtifactId=maven-archetype-quickstart
```

Listing 1: Befehl zum erstellen einer Beispiel POM-Datei für den Architekturtyp *maven-archetype-quickstart*

```
bash> tree
.
+-- pom.xml
+-- src
    +-- main
    |   +-- java
    |       +-- org
    |           +-- livingplace
    |               +-- maven-tutorial
    |                   +-- App.java
    +-- test
        +-- java
            +-- org
                +-- livingplace
                    +-- maven-tutorial
                        +-- AppTest.java

11 directories, 3 files
```

Listing 2: Verzeichnisstruktur des Beispiel Maven Projekts

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.livingplace.maven-tutorial</groupId>
  <artifactId>project-creation</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>project-creation</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>
        UTF-8
    </project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Listing 3: Standard POM Datei aus der Onlinedokumentation

1.2 Maven Lifecycle Phasen

Die Tabelle 1 zeigt alle von Maven definierten Lifecycle Phasen die während eines Buildprozesses durchlaufen werden können.

Lifecycle Phase	Description
validate	Validate the project is correct and all necessary information is available to complete a build
generate-sources	Generate any source code for inclusion in compilation
process-sources	Process the source code, for example to filter any values
generate-resources	Generate resources for inclusion in the package
process-resources	Copy and process the resources into the destination directory, ready for packaging
compile	Compile the source code of the project
process-classes	Post-process the generated files from compilation, for example to do bytecode enhancement on Java classes
generate-test-sources	Generate any test source code for inclusion in compilation
process-test-sources	Process the test source code, for example to filter any values
generate-test-resources	Create resources for testing
process-test-resources	Copy and process the resources into the test destination directory
test-compile	Compile the test source code into the test destination directory
test	Run tests using a suitable unit testing framework. These tests should not require the code be packaged or deployed
prepare-package	Perform any operations necessary to prepare a package before the actual packaging. This often results in an unpacked, processed version of the package (coming in Maven 2.1+)
package	Take the compiled code and package it in its distributable format, such as a JAR, WAR, or EAR
pre-integration-test	Perform actions required before integration tests are executed. This may involve things such as setting up the required environment
integration-test	Process and deploy the package if necessary into an environment where integration tests can be run
post-integration-test	Perform actions required after integration tests have been executed. This may include cleaning up the environment
verify	Run any checks to verify the package is valid and meets quality criteria
install	Install the package into the local repository, for use as a dependency in other projects locally
deploy	Copies the final package to the remote repository for sharing with other developers and projects (usually only relevant during a formal release)

Tabelle 1: Maven Lifecycle Phasen, vgl. Sonatype (2008, S. 184 f)

2 OSGi

Die folgende Tabelle zeigt alle OSGi Compendium Service Spezifikationen, vgl. OSGi Alliance (2009b).

Item	Package	Version
101 Log Service Specification	org.osgi.service.log	Version 1.3
102 Http Service Specification	org.osgi.service.http	Version 1.2
103 Device Access Specification	org.osgi.service.device	Version 1.1
104 Configuration Admin Service Specification	org.osgi.service.cm	Version 1.3
105 Metatype Service Specification	org.osgi.service.metatype	Version 1.1
106 Preferences Service Specification	org.osgi.service.prefs	Version 1.1
107 User Admin Service Specification	org.osgi.service.useradmin	Version 1.1
108 Wire Admin Service Specification	org.osgi.service.wireadmin	Version 1.0
109 IO Connector Service Specification	org.osgi.service.io	Version 1.0
110 Initial Provisioning	org.osgi.service.provisioning	Version 1.2
111 UPnP™ Device Service Specification	org.osgi.service.upnp	Version 1.1
112 Declarative Services Specification	org.osgi.service.component	Version 1.1
113 Event Admin Service Specification	org.osgi.service.event	Version 1.2
114 Deployment Admin Specification	org.osgi.service.deploymentadmin org.osgi.service.deploymentadmin.spi	Version 1.1
115 Auto Configuration Specification		Version 1.2
116 Application Admin Specification	org.osgi.service.application	Version 1.1

Fortsetzung der Tabelle auf nächster Seite

Tabelle 2 – *Fortsetzung*

Item	**Package**	**Version**
117 DMT Admin Service Specification	info.dmtree info.dmtree.notification info.dmtree.notification.spi info.dmtree.registry info.dmtree.security info.dmtree.spi	Version 1.0
119 Monitor Admin Service Specification	org.osgi.service.monitor	Version 1.0
120 Foreign Application Access Specification	org.osgi.application	Version 1.0
121 Blueprint Container Specification	org.osgi.blueprint.container org.osgi.blueprint.reflect	Version 1.0
701 Tracker Specification	org.osgi.util.tracker	Version 1.4
702 XML Parser Service Specification	org.osgi.util.xml	Version 1.0
703 Position Specification	org.osgi.util.position	Version 1.0
704 Measurement and State Specification	org.osgi.util.measurement	Version 1.0
999 Execution Environment Specification		Version 1.3

Tabelle 2: OSGi Compendium Service Specifications, vgl. OSGi Alliance (2009b)

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

Hamburg, 24.04.2013 Kjell Otto