UNIVERSITÄT
DES
SAARLANDES

ASaP - Integrationsplattform für Smart Services in Intelligenten Umgebungen

Dipl.-Inform. Jochen Frey

Dissertation zur Erlangung des Grades
Doktor der Ingenieurswissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Saarbrücken, 22. Juli 2015

Tag des Kolloquiums:
21. Juli 2015

Dekan:
Univ.-Prof. Dr. Markus Bläser

Vorsitzender des Prüfungsausschusses:
Prof. Dr. Antonio Krüger

Berichterstatter:
Prof. Dr. Dr. h.c. mult. Wolfgang Wahlster
Prof. Dr. Paul Lukowicz

Akademischer Mitarbeiter:
Dr. Jan Alexandersson

Danksagung

Diese Arbeit entstand im Rahmen des von der Europäischen Kommission geförderten Projekts i2home (FP6-033502) und der vom Bundesministerium für Bildung und Forschung (BMBF) geförderten Projekte SmartSenior (16KT0902) und im Rahmen des Software Campus durchgeführten AdAPT (01IS12050). Teile der Arbeit wurden innerhalb des vom Bundesministerium für Wirtschaft und Energie (BMWi) geförderten Projekts Peer Energy Cloud (01MD11002) entwickelt.

Zu der Entstehung der Arbeit haben viele Personen sowohl durch fachliche als auch durch moralische Unterstützung beigetragen. Ihnen allen möchte ich an dieser Stelle meinen ausdrücklichen Dank aussprechen.

An erster Stelle möchte ich meinem Doktorvater Prof. Wolfgang Wahlster dafür danken, in seiner Forschungsgruppe am Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI) an interessanten Projekten arbeiten und im Rahmen dieser Tätigkeit meine Doktorarbeit anfertigen zu dürfen. Besonders in der Schlussphase meiner Arbeit unterstützte er mich mit konstruktiven Diskussionen und wertvollen Denkanstößen in einer stets sehr angenehmen Arbeitsatmosphäre. Prof. Paul Lukowicz möchte ich für seine kurzfristige Bereitschaft danken, das Zweitgutachten für diese Arbeit zu verfassen.

Ein großer Dank geht an meine Arbeitskollegen am DFKI für das tolle Arbeitsumfeld, für interessante Gespräche und Diskussionen sowie für die gute Zusammenarbeit auch in stressigen Phasen. Darüber hinaus danke ich meinen Kollegen für das mir entgegengebrachte Verständnis und die Rücksichtnahme in der Schlussphase meiner Arbeit. Ohne die geschaffenen Freiräume wäre das Niederschreiben dieser Arbeit nur schwer möglich gewesen.

Ein spezieller Dank geht an Jan Alexandersson für das *An Bord holen*, an Boris Brandherm für das *Rücken freihalten* und an Gerd Herzog für den unerschöpflichen Quell an bibliographischem Wissen.

Ganz besonders bedanke ich mich bei den ehemaligen und aktuellen Doktoranden, die mich auch in den Frustphasen der Arbeit immer unterstützt und gemeinsam mit mir mein Leid geteilt haben.

Von ganzem Herzen danke ich meiner Familie und meinen Freunden. Ohne ihre allgegenwärtige moralische und praktische Unterstützung hätte diese Arbeit nicht entstehen können.

> Je kaputter die Welt draußen, desto heiler muss sie zu Hause sein.
> (Reinhard Mey)

Die letzten Worte gehören Lena, die mir das größte Geschenk gemacht hat.

Zusammenfassung

Die Entwicklung von mit vernetzten Sensoren und Aktuatoren ausgestatteten Wohnungen, sogenannten *Smart Homes*, ist ein wichtiger Zukunftstrend in Deutschland. Laut aktuellen Studien wird die Zahl der *Smart Homes* in Deutschland bis spätestens 2020 die Millionenmarke erreicht haben. Das übergeordnete Forschungsgebiet von *Smart Homes* sind *Intelligente Umgebungen*, welche im Allgemeinen alle räumlichen Umgebungen umfassen, die dem Benutzer eine Form von intelligenter Assistenz und Unterstützung bereitstellen.

Die vorliegende Arbeit beschäftigt sich mit der Fragestellung, wie neuartige digitale Mehrwertdienste in Form von sogenannten *Smart Services* innerhalb einer *Intelligenten Umgebungen* geschaffen und nahtlos darin integriert werden können. Die Aufgabenstellung beinhaltet zwei grundlegende Herausforderungen:

- Der Kunde als Bewohner eines zukünftigen *Smart Homes* soll durch ein durchgängiges Marktplatzkonzept dazu befähigt werden, auf seine Situation passende *Smart Services* zu finden und auf einer zentralen Plattform zu installieren. Bisherige Ansätze fokussieren oftmals auf einen einzigen expliziten Anwendungsfall. Eine allgemeine und dienstübergreifende Interoperabilität der Services ist somit nicht gewährleistet.
- Den Dienstanbietern und Entwicklern müssen Werkzeuge und Entwicklungsumgebungen an die Hand gegeben werden, mit denen *Smart Services* schnell und effizient konzipiert, umgesetzt und auf einem zentralen Marktplatz verteilt werden können.

In der vorliegenden Arbeit wird ein allgemeines Integrationskonzept vorgestellt, welches sowohl die Ansätze und Konzepte des bestehenden Industriestandards *Universal Remote Console* als auch die zugrunde liegende Referenzarchitektur der *Smart Service Welt* berücksichtigt.

Aufbauend auf einer einheitlichen semantischen Modellierung einer *Intelligenten Umgebung* wird neben einem Konzept zur Realisierung von personalisierten Benutzerschnittstellen auch ein auf einem Multiagentensystem aufbauende Plattform zur dienstübergreifenden Integration von *Smart Services* vorgestellt.

Die Schwerpunkte der Arbeit liegen auf der Konzeption, dem Entwurf und der Realisierung einer offenen, hochflexiblen, erweiterbaren und auf Standards basierenden Softwarearchitektur für *Smart Services* im Kontext von *Smart Home* Technologien.

Abstract

Equipping homes with networked sensors and actuators represents an important future trend in Germany. According to recent studies, the total number of *Smart Homes* in Germany will exceed the one-million mark in 2020. The primary research field of *Smart Homes* are *Intelligent Environments*, which focus on all spatial environments providing the user some sort of ambient intelligence and assistance.

This thesis addresses the research question, how novel value-added *Smart Services* can be realised and seamlessly integrated within these *Intelligent Environments*. Achieving this objective involves two fundamental challenges:

- Customers, as residents of a future *Smart Home*, should be supported by means of a consistent marketplace. This marketplace enables the user to find appropriate *Smart Services* that fit his current situational and environmental context and to install these services on a central platform. Existing approaches often focus on one dedicated use case and are lacking in a universal and comprehensive interoperability.
- The provisioning of tools and development environments is an essential requirement for service providers and developers to achieve an efficient and comprehensive development and deployment workflow for *Smart Services*.

First, this thesis provides an overall concept for integrating *Smart Services* in *Intelligent Environments*. The concept comprises existing approaches of the industry standard *Universal Remote Console* as well as of the reference architecture described in the *Smart Service World*.

Second, a consistent semantic model of the underlying *Intelligent Environment* is introduced, which forms the basis for the realisation of a concept for personalised user interfaces and of a multi-agent based software platform for integrating interoperable *Smart Services*.

The main focus of the thesis concentrates on the conception, the design and the realisation of an open, highly flexible, expandable and standardised software architecture for *Smart Services* within the scope of *Smart Home* technologies.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Glossar

Agent Ein *Agent* ist ein Computersystem, welches innerhalb einer bestimmten Umgebung dazu in der Lage ist, autonome Handlungen und Aktionen auszuführen mit dem Zweck, die zuvor definierten und geplanten Ziele zu erreichen.

Agentenplattform *Agentenplattformen* bieten die grundlegende Funktionalitäten und Dienste zur Realisierung eines *Multiagentensystems*. Kernbestandteil einer *Agentenplattform* ist die Bereitstellung von Möglichkeiten zur Erstellung, Registrierung, Lokalisierung und Zerstörung einzelner *Agenten* sowie Kommunikationsmechanismen zum Informationsaustausch zwischen den *Agenten*.

Aktuator *Aktuatoren* übermitteln Informationen an den Benutzer und sind dazu in der Lage, Zustandsveränderungen und Ereignisse innerhalb der Umgebung auszulösen und somit den *Kontext* der Umgebung zu beeinflussen.

App Store Ein *App Store* ist ein digitaler Onlinemarktplatz innerhalb eines Softwareökosystems, welcher es Entwicklern und Anbietern ermöglicht, ihre Softwareprodukte innerhalb einer definierte Plattforminfrastruktur an die entsprechenden Kunden zu verkaufen (Jansen und Bloemendal, 2013).

Ambient Assisted Living *Ambient Assisted Living* (AAL) betrachtet einen Teilaspekt von *Ambienter Intelligenz* und untersucht, wie entsprechende Methoden und Verfahren eingesetzt werden können, um älteren oder unterstützungsbedürftigen Menschen so lange wie möglich ein selbstbestimmtes Leben in ihrer häuslichen Umgebung zu ermöglichen.

Ambiente Intelligenz *Ambiente Intelligenz* (AmI) ist die proaktive und allgegenwärtige Unterstützung von Personen in ihrem privaten und beruflichen Leben.

Big Data *Big Data* beschreibt Datenbestände, die aufgrund ihres Umfangs, Vielfalt oder ihrer Schnelllebigkeit nur begrenzt durch aktuelle Datenbanken und Daten-Management-Tools in Echtzeit analysiert und verarbeitet werden können.

Controller Ein *Controller* ist ein Endgerät, auf dem eine Benutzerschnittstelle zur Überwachung und Steuerung eines oder mehrerer *Targets* realisiert und visualisiert wird.

Cyber-Physisches System Ein *Cyber-Physisches System* (CPS) stellt eine Hardware-Software-Kombination dar, durch welche die reale und die virtuelle Welt zusammengeführt werden, um eine vernetzte Welt zu erzeugen, in der intelligente Objekte miteinander kommunizieren und interagieren können.

Instrumentierte Umgebung Eine *Instrumentierte Umgebung* beschreibt einen Ausschnitt der realen Welt, welche mit einer technischen Infrastruktur ausgestattet wurde und dazu in der Lage ist, den aktuellen *Kontext* der Umgebung zu erfassen und eine Veränderung und Manipulation der Umgebung zu ermöglichen.

Intelligente Umgebung Der Begriff der *Intelligenten Umgebung* bezeichnet eine Umgebung, welche dazu in der Lage ist, Wissen über die Umgebung und die darin befindlichen Benutzer zu akquirieren und anzuwenden mit dem Ziel, die Benutzer innerhalb der Umgebung optimal zu unterstützen.

Internet der Dinge Das *Internet der Dinge* bezeichnet die Verknüpfung eindeutig identifizierbarer physischer Objekte mit einer virtuellen Repräsentation und digitalen Modellierung im Internet. Innerhalb des *Internet der Dinge* werden passive Objekte zu intelligenten Objekten transformiert, indem diese dazu in die Lage versetzt werden, Daten zu speichern, auf verwandte Daten zu verweisen und Kommunikationsmöglichkeiten für Mensch und Maschine bereitzustellen.

Internet der Dienste Das *Internet der Dienste* beschreibt die Realisierung von Entwicklungs- und Dienstplattformen, um neuartige, webfähige Dienste einfacher und schneller erstellen und im Internet anbieten zu können[3].

Kontext Als *Kontext* wird die Menge der verfügbaren Informationen angesehen, welche eine räumliche Umgebung, entsprechende Umgebungseigenschaften, wie zum Beispiel Lautstärke, Helligkeit und Temperatur, sowie die darin befindlichen Personen, Objekte und Agenten charakterisiert.

Middleware Eine *Middleware* ist ein Teil der *Software-definierten Plattform* (siehe Kapitel 3.3), der Kommunikationsdienste für verteilte Anwendungen über Standardschnittstellen bereitstellt und somit eine Integration von unterschiedlichen Anwendungen und deren Daten ermöglicht.

Multiagentensystem Ein *Multiagentensystem* (MAS) besteht aus einer Menge von *Agenten*, welche untereinander Informationen austauschen können und somit dazu in der Lage sind, durch Kooperation, Koordination und Kollaboration ein übergeordnetes Gesamtproblem zu lösen.

Ontologie Eine *Ontologie* definiert das verfügbare Vokabular innerhalb eines Themenbereichs, indem die grundlegenden Begrifflichkeiten und Beziehungen sowie die Regeln zur Kombination und Erweiterung des Vokabulars spezifiziert werden (Gomez-Pérez et al., 2007).

Pervasive Computing *Pervasive Computing* beschreibt die Möglichkeit des Benutzers, jederzeit innerhalb einer Umgebung Zugriff auf relevante Informationen zu haben. Dies beinhaltet alle verfügbaren Geräte und Anwendungen, die dem Benutzer diese Funktionalität bereitstellen.

[3] http://www.bmwi.de/DE/Themen/Digitale-Welt/Internet-der-Zukunft/internet-der-dienste.html [Letzter Zugriff: 25.01.2015]

Resource Server Ein *Resource Server* stellt innerhalb der URC-Infrastruktur alle benötigten Ressourcen zur Realisierung von personalisierten Benutzerschnittstellen zur Verfügung.

Resource Sheet Ein *Resource Sheet* organisiert alle für die Realisierung einer konkreten Benutzerschnittstelle notwendige Information zu einem bestimmten *Target*.

Sensor *Sensoren* sind dazu in der Lage, Zustandsinformationen und Zustandsveränderungen der Umgebung und der darin befindlichen Personen, Objekte und mobilen Agenten zu erfassen und somit den *Kontext* einer Umgebung zu ermitteln.

Serviceplattform Eine *Serviceplattform* stellt eine unternehmens- und branchenübergreifende Kollaborationsplattform dar, auf der Wertschöpfungsketten modular konfiguriert und zu neuen, internetbasierten Geschäftsmodellen zusammengeführt werden können.

Smart Data Als *Smart Data* werden Datenmengen bezeichnet, die eine sinnvolle semantische Beschreibung beinhalten. Solche Datenmengen entstehen durch die Verarbeitung von großen Datenmengen (*Big Data*), aus denen mithilfe von Informationsextraktionsverfahren und maschinellem Lernen kontextbezogene und personalisierte Daten abgeleitet, analysiert, interpretiert und schließlich zu neuen Informationen veredelt werden (Kagermann et al., 2015).

Smart Product Das *Smart Product* beschreibt die Verknüpfung von realen, physischen Produkten mit einer Kombination aus intelligenten Hard- und Softwaresystemen. Durch die Einbettung von Kleinstcomputern können *Smart Products* als *Sensoren* und *Aktuatoren* agieren und somit Daten aus ihrer direkten Umgebung erfassen sowie Informationen in die Umgebung zurückfließen lassen.

Smart Service Als *Smart Service* wird die Verschmelzung von digitalen und physischen Dienstleistungen zu einer auf den einzelnen Konsumenten bedarfsgerecht zugeschnittenen Gesamtdienstleistung bezeichnet.

Smart Space Als *Smart Space* wird eine *Intelligente Umgebung* bezeichnet, welche eine Kommunikation und Kooperation von *Smart Products* realisiert, sowie dem Benutzer unterschiedliche Möglichkeiten zur Interaktion anbietet.

Software-definierte Plattform *Software-definierte Plattformen* sind die technologische Basis für neuartige, internetbasierte Dienstleistungen. Sie können durch Virtualisierung die Zusammenarbeit unterschiedlicher *Vernetzter physischer Plattformen* sicherstellen und die Konnektivität einzelner *Smart Products* bereitstellen.

Target Als *Targets* werden alle vernetzten Geräte, Objekte, *Sensoren* und *Aktuatoren* sowie webbasierte Dienste innerhalb einer definierten *Instrumentierten Umgebung* bezeichnet, welche durch die UCH-Architektur integriert und durch einen entsprechenden *Controller* angesteuert werden können.

Target Adapter Ein *Target Adapter* ist eine modulare Softwarekomponente innerhalb der UCH-Infrastruktur und verwaltet die benötigte Kommunikationsinfrastruktur für die Anbindung eines spezifischen oder eine Menge von gleichartigen *Targets*.

Technische Infrastruktur Die *Technische Infrastruktur* beschreibt die technisch-infrastrukturelle Voraussetzung zur Realisierung von *Smart Spaces* und aller darauf aufsetzenden *Smart Services*.

Ubiquitous Computing *Ubiquitous Computing* beschreibt die Allgegenwärtigkeit kleinster, drahtlos miteinander vernetzter Computer, die in beliebige Alltagsgegenstände eingebaut werden können. Mit dem Begriff des *Ubiquitous Computing* geht ein Paradigmenwechsel weg vom Computer als Werkzeug hin zu einer impliziten Informationsverarbeitung einher.

UCH-Bundle Ein *UCH-Bundle* ist ein spezielles OSGi-*Bundle*, welches die konkrete Funktionalität einer Komponente der allgemeinen UCH-Architektur implementiert und dynamisch zur Laufzeit hinzugefügt und wieder entfernt werden kann.

User Interface Protocol Module Ein *User Interface Protocol Module* (UIPM) ist eine dedizierte Softwarekomponente innerhalb der UCH-Infrastruktur, welche eine Kommunikations- oder Netzwerkschnittstelle zur Verfügung stellt, um über die UIS-Beschreibung Zugriff auf die Zustände und Funktionalitäten einzelner *Targets* zu haben.

User Interface Socket Ein *User Interface Socket* (UIS) definiert eine abstrakte Schnittstellenbeschreibung zu den jeweiligen Funktionalitäten und Zuständen eines *Targets*.

Vernetzte physische Plattform Eine *Vernetzte physische Plattform* bezeichnet einzelne sowie den Zusammenschluss mehrerer *Smart Products*, welche über die *Technische Infrastruktur* vernetzt sind.

Der Anfang ist die Hälfte des Ganzen.

Aristoteles

1 Einleitung

1.1 Motivation

Der Begriff *Smart Home* beschreibt allgemein Wohnungen und Häuser, deren technologische Ausstattungen, wie Haustechnik, Haushaltsgeräte und Unterhaltungselektronik, untereinander vernetzt sind. Die Weiterentwicklung von *Smart Home* Technologien stellt einen wichtigen Zukunftstrend in Deutschland dar. Laut einer Studie von BITKOM und Deloitte[1] wird die Zahl der *Smart Homes* bis spätestens 2020 die Millionenmarke erreicht haben. Diese Trendentwicklung lässt das zukünftige Potential von *Smart Home* Anwendungen nicht nur in Deutschland erahnen. Abbildung 1.1 skizziert anhand einer konservativen und einer progressiven Rechnung die Ergebnisse der Studie.

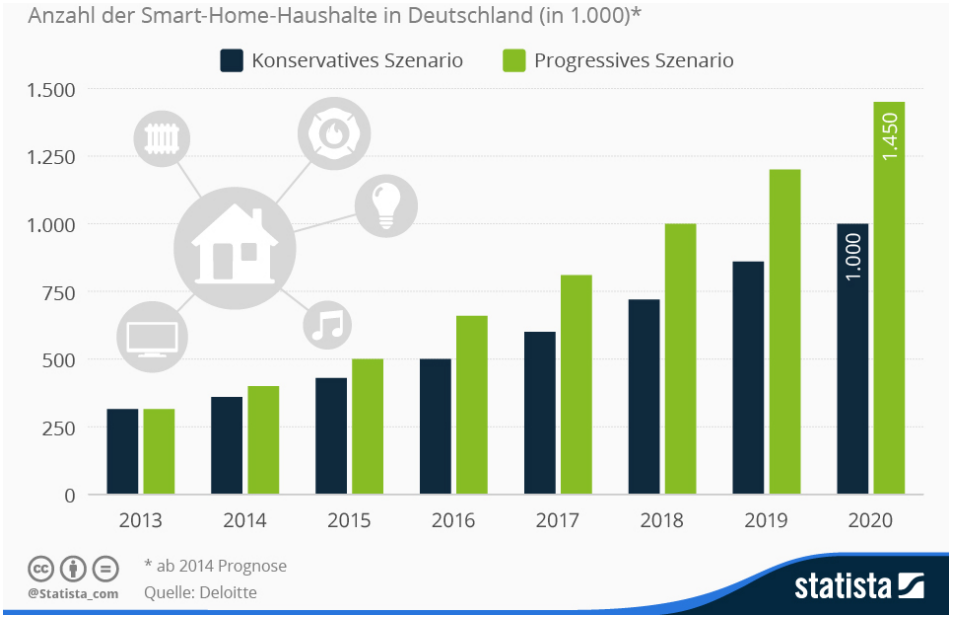


Abbildung 1.1: Übersicht über die prognostizierte Anzahl von *Smart Homes* in Deutschland[2]

[1] http://www2.deloitte.com/content/dam/Deloitte/de/Documents/technology-media-telecommunications/TMT-Deloitte-Bitkom-Marktaussichten-SmartHome.pdf [Letzter Zugriff: 07.01.2015]

Der Oberbegriff eines *Smart Homes* ist die sogenannte *Intelligente Umgebung*, welche allgemein alle räumlichen Umgebungen umfasst, die dem Benutzer eine Form von intelligenter Assistenz bereitstellen. Intelligenz ist jedoch ein sehr breit und oftmals unscharf definierter Begriff. Grundsätzlich lassen sich aber drei notwendige Voraussetzungen für eine *Intelligente Umgebung* ableiten.

Eine erste Voraussetzung ist, die notwendigen Kontextinformationen über den Benutzer und seine Umgebung zu erfassen. Hierzu benötigt man eine Vielzahl von *Sensoren* und *Aktuatoren*, welche kontinuierlich Daten über ihren eigenen Zustand und den ihrer Umgebung sammeln. Schließt man viele dieser *Sensoren* und *Aktuatoren* zu einer einheitlichen Plattform zusammen, spricht man von einer *Instrumentierten Umgebung*. Aktuell ist zu beobachten, dass die Entwicklung von solchen Plattformen kontinuierlich von Industrie und Forschung vorangetrieben wird. Man kann also behaupten, dass die ursprüngliche Vision des unsichtbaren und allgegenwärtigen Computers (Weiser, 1991) allmählich realisiert wird. Groß angelegte Forschungsprojekte wie SemProM (Wahlster, 2013), RES-COM[3] und SmartF-IT[4] zeigen, dass man bereits in der Lage ist, eine Vielzahl alltäglicher Produkte mit *Sensoren*, *Aktuatoren* oder *Digitalen Produktgedächtnissen* (Haupert, 2013) zu versehen und somit digitale Abbildungen von realen physischen Objekten zu erzeugen. Ein weiterer Forschungsschwerpunkt sind, besonders im medizinischen Bereich, sogenannte *Wearable Systems* (Lukowicz et al., 2004; Amft und Lukowicz, 2009; Lukowicz et al., 2010), also tragbare Mikrocomputer, welche kontinuierlich Informationen über den physischen Zustand des Benutzers erfassen können. Das Speichern von großen Datenmengen über den Benutzer und dessen Umgebung reicht allerdings noch nicht aus, um von einer intelligenten Assistenz für den Benutzer sprechen zu können. Eine zweite Voraussetzung ist daher die Analyse und Interpretation der Daten. Begriffe wie *Big Data* und *Smart Data* haben hier die Forschungslandschaft geprägt und bieten Möglichkeiten, wie man aus großen Datenmengen die für den jeweiligen Anwendungsfall relevanten und nützlichen Informationen extrahieren kann. Als dritte und letzte Voraussetzung für eine *Intelligente Umgebung* ist schließlich die Nutzung der gewonnenen Erkenntnisse über den Benutzer und dessen Umgebung zu nennen, um persönliche und auf seine Bedürfnisse angepasste Assistenzsysteme zu realisieren.

Jede der Voraussetzungen zeigt die Notwendigkeit aber auch das Potential für die Schaffung neuartiger digitaler Mehrwertdienste (*Smart Services*) im Kontext von *Intelligenten Umgebungen*. Ein wichtiger Bestandteil ist zudem das Bereitstellen von persönlichen und barrierefreien Benutzerschnittstellen. Das Forschungsgebiet *Ambient Assisted Living* (AAL) zeigt hierbei die Notwendigkeit von spezifischen und individualisierten Interaktionsmechanismen für den Endbenutzer. Die vorgestellten Voraussetzungen setzen sich wiederum aus vielen weiteren Anforderungen zusammen und umfassen mehrere interdisziplinäre Forschungsthemen, zu denen zwar jeweils einzelne Lösungen und

[2]http://de.statista.com/infografik/3105/anzahl-der-smart-home-haushalte-in-deutschland [Letzter Zugriff: 31.01.2015]

[3]http://www.res-com-projekt.de [Letzter Zugriff: 31.01.2015]

[4]http://www.smartf-it-projekt.de [Letzter Zugriff: 31.01.2015]

Komponenten bereits vorhanden sind, deren Zusammenspiel untereinander allerdings oftmals vernachlässigt wird.

1.2 Ziele dieser Arbeit

Das Ziel der vorliegenden Arbeit ist die Realisierung eines einheitlichen Architekturkonzepts zur Integration von *Smart Services* innerhalb *Intelligenter Umgebungen* als Grundvoraussetzung zur Implementierung intelligenter Assistenzsysteme. Darüber hinaus sollen Entwicklungswerkzeuge in Form einer durchgängigen Werkzeugkette bereitgestellt und ein zentraler Marktplatz für *Smart Services* umgesetzt werden.

1.2.1 Einordnung in Forschungsthemen

Das Thema der vorliegenden Arbeit liegt in der Schnittmenge aus unterschiedlichen interdisziplinären Forschungsgebieten. Abbildung 1.2 stellt eine Übersicht über das Zusammenspiel der einzelnen Bereiche dar.

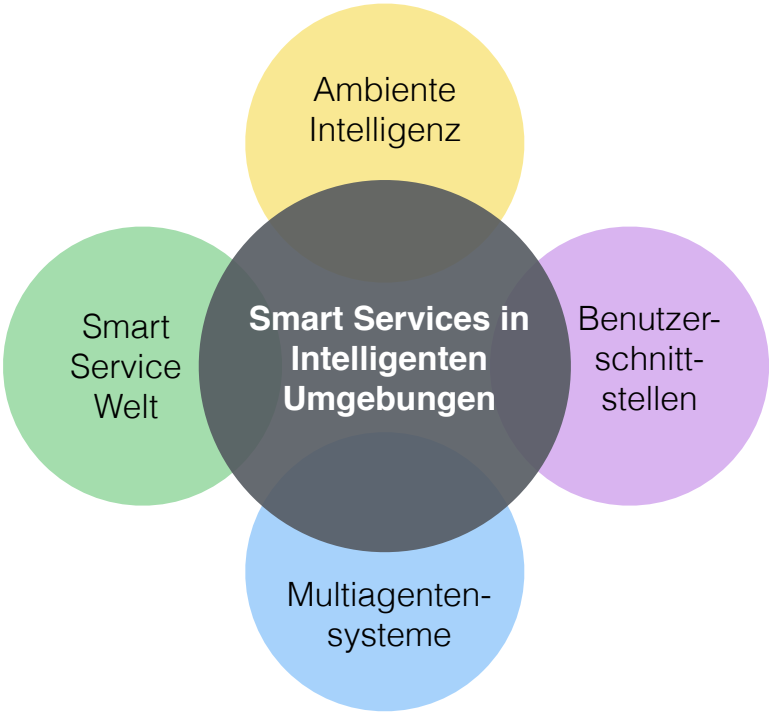


Abbildung 1.2: Übersicht über die thematischen Schwerpunkte der vorliegenden Arbeit.

Die folgende Aufzählung enthält eine Kurzbeschreibung zu jedem der genannten Forschungsgebiete:

- **Ambiente Intelligenz:** Das Forschungsgebiet der *Ambienten Intelligenz* (AmI) beschäftigt sich mit der Frage, wie alltägliche Gegenstände sowohl um Kommunikationsfähigkeit als auch um autonome, intelligente

Verhaltensweisen erweitert werden können (Nakashima et al., 2009). Das Ziel von AmI ist, dass eine *Intelligente Umgebung* dazu in der Lage ist, den Benutzer auf natürliche Art und Weise bei alltäglichen Aktivitäten zu unterstützen. Eine Einführung in das Forschungsgebiet der *Ambienten Intelligenz* wird in Kapitel 2 gegeben.

- **Smart Service Welt:** Das Forschungsgebiet der *Smart Service Welt* setzt den Fokus auf die Fragestellung, wie man intelligente Produkte mit physischen und digitalen Dienstleistungen zu sogenannten *Smart Services* kombinieren und somit neue digitale Wertschöpfungsketten generieren kann (Kagermann et al., 2014, 2015). Im Vordergrund steht hierbei eine unternehmens- und branchenübergreifende Kooperation, wobei nicht das Produkt oder die Dienstleistung, sondern vielmehr der Konsument in seiner Rolle als Servicenutzer im Mittelpunkt steht. Eine Einführung in das Forschungsgebiet der *Smart Services* wird in Kapitel 3 gegeben.

- **Multiagentensysteme:** Das Forschungsgebiet der *Multiagentensysteme* (MAS) beschäftigt sich mit gleichartigen oder unterschiedlich spezialisierten handelnden Einheiten, die kollektiv ein Problem lösen (Weiss, 2000). Ein MAS besteht aus einer Menge von individuellen *Softwareagenten* mit einem gewissen Grad an Autonomie in Bezug auf ihre Wahrnehmung und ihren Handlungen innerhalb einer konkreten Anwendungsdomäne. Die Berechnung der Gesamtlösung basiert auf individuellen autonomen Berechnungen jedes *Agenten* und einer definierten Kommunikationstruktur der *Agenten* untereinander. Eine Einführung in das Forschungsgebiet der *Multiagentensysteme* wird in Kapitel 4 gegeben.

- **Intelligente Benutzerschnittstellen:** Mobile multimodale Benutzerschnittstellen zu kontextsensitiven Diensten, insbesondere für die Verwendung in Fahrzeugen, in *Smart Homes* oder in Einkaufsszenarien, bilden einen Schwerpunkt des Forschungsgebietes der *Intelligenten Benutzerschnittstellen* (Maybury und Wahlster, 1998). Neben dem intelligenten Zugang zum *Internet der Dienste* und dem *Internet der Dinge* im Rahmen des *Semantischen Webs*, wird auch der barrierefreie Zugang zu *Instrumentierten Umgebungen* und zu vernetzen Welten für Senioren und Behinderte untersucht. Kapitel 6 beschreibt ein im Rahmen der Arbeit entwickeltes, standardisiertes Konzept zur Realisierung von personalisierten Benutzerschnittstellen.

1.2.2 Wissenschaftliche Fragestellungen

Das primäre Ziel dieser Arbeit ist Entwicklung eines Plattformkonzeptes zur Integration von *Smart Services* in *Intelligenten Umgebungen* und zur Realisierung von intelligenten Assistenzsystemen. Hierzu wird ein einheitlicher Architekturansatz vorgeschlagen, welcher *Sensoren*, *Aktuatoren*, *Smart Services* und *Intelligente Benutzerschnittstellen* von verschiedensten Herstellern in ein einheitliches Plattformkonzept überführt. Grundvoraussetzung für die nahtlose

Interaktion aller beteiligten Soft- und Hardwarekomponenten ist eine formale, semantische Beschreibung der zu Grunde liegenden *Intelligenten Umgebung* und aller darin enthaltenen Objekte und Beziehungen. Zur Vorbereitung einer späteren kommerziellen Nutzung wird im letzten Schritt ein *Smart Service Marktplatz* entwickelt mit dem Ziel, Dienstanbieter, Entwickler und Endkunden auf einer zentralen Plattform zusammenzubringen. In der vorliegenden Arbeit sollen die folgenden sechs Forschungsfragen beantwortet werden.

Die ersten drei Fragestellungen beziehen sich auf die Konzeption, den Entwurf und die Realisierung einer Integrationsplattform für:

1. ***Sensoren* und *Aktuatoren*:** *Wie können unterschiedlichste Sensoren und Aktuatoren in eine Intelligente Umgebung integriert werden?*

   Grundbestandteil einer *Intelligenten Umgebung* ist eine Vielzahl von *Sensoren* und *Aktuatoren*, um den Zustand der Umgebung sowie deren Benutzer möglichst genau zu erfassen und gegebenenfalls beeinflussen zu können. Die Herausforderung dabei ist, dass die zum Einsatz kommenden *Sensoren* und *Aktuatoren* von Umgebung zu Umgebung variieren und darüber hinaus in der Regel von unterschiedlichen Herstellern stammen. Die daraus resultierende Schnittstellenproblematik muss sowohl durch geeignete semantische Beschreibungssprachen, als auch durch den Einsatz von *Middlewareplattformen* gelöst werden.

2. ***Smart Services*:** *Wie können Smart Services in eine Intelligente Umgebung integriert und eine dienstübergreifende Interoperabilität gewährleistet werden?*

   Aufbauend auf der Integration von *Sensoren* und *Aktuatoren* sollen im nächsten Schritt digitale Mehrwertdienste in Form von *Smart Services* in die Plattform eingebunden werden. Existierende Plattformen fokussieren oftmals auf eine einzige spezialisierte Problemstellung, wie beispielsweise eine automatische Licht- oder Heizungssteuerung, oder weisen ein hohes Maß an Komplexität auf. Ein Ziel der vorliegenden Arbeit ist es, ein Plattformkonzept für *Smart Services* zu entwickeln, welches auf der einen Seite allgemein genug ist, um viele Problemstellungen innerhalb *Intelligenter Umgebungen* zu adressieren, und auf der anderen Seite einfach genug ist, um Entwicklern und Anwendern die Benutzung der Plattform so weit wie möglich zu erleichtern.

3. **Personalisierte Benutzerschnittstellen:** *Wie kann eine Vielzahl von Benutzerschnittstellen mit minimalem Aufwand in eine Intelligente Umgebung integriert werden?*

   Die Realisierung von intelligenten Assistenzsystemen innerhalb einer *Intelligenten Umgebung* erfordert eine Möglichkeit mit dem Benutzer zu interagieren. Zum einen soll dem Benutzer eine personalisierte, transparente und allgegenwärtige Sicht auf seine Umgebung angeboten werden.

Er soll zum Beispiel in die Lage versetzt werden, den Zustand einzelner Geräte überwachen zu können oder bei Fehlfunktionen und Gefahren rechtzeitig informiert zu werden. Zum anderen dient die Interaktion des Benutzers mit seiner Umgebung den *Smart Services* dazu, weitere Informationen über das Verhalten des Benutzers abzuleiten. So können zum Beispiel typische Verhaltensweisen oder sich wiederholende Interaktionsmuster erkannt werden. Der Schwerpunkt liegt auf dem Finden persönlicher Interaktionskonzepte, welche genügend Informationen für Assistenzsysteme liefern, hierbei aber keinen zu großen Eingriff in den Alltag des Benutzers darstellen.

Die letzten drei Fragestellungen betrachten die Konzeption, den Entwurf und die Realisierung einer semantischen Modellierung, eines *Smart Service Marktplatzes* und einer durchgängigen Werkzeugunterstützung innerhalb einer *Intelligenten Umgebung*.

4. **Semantische Modellierung:** *Wie kann eine Intelligente Umgebung semantisch repräsentiert werden, insbesondere das Zusammenspiel von Sensoren, Aktuatoren, Smart Services und Benutzerschnittstellen?*

   Um einen einheitlichen Informations- und Wissensaustausch untereinander zu garantieren, ist es notwendig, die Funktionsweisen und Eigenschaften aller Komponenten einer *Intelligenten Umgebung* semantisch zu modellieren und abzubilden. Insbesondere die Kommunikation zwischen *Sensoren*, *Aktuatoren*, *Smart Services* und dem Benutzer spielen hier eine wichtige Rolle. Nur wenn alle beteiligten Komponenten sich gegenseitig verstehen können, kann das übergeordnete Ziel einer *Intelligenten Umgebung* gelöst werden. Die semantische Modellierung dient darüber hinaus als Grundlage für die Entwicklung eines *Smart Service Marktplatzes* sowie einer durchgängigen Werkzeugunterstützung.

5. ***Smart Service Marktplatz*:** *Wie können neuartige digitale Geschäftsmodelle in Form von Smart Services auf Basis eines einheitlichen Marktplatzkonzeptes realisiert und vertrieben werden?*

   Wie bereits erwähnt sind an der Realisierung und zum Betrieb einer *Intelligenten Umgebung* unterschiedliche Interessengruppen notwendig. Hersteller von *Sensoren* und *Aktuatoren* wollen primär Hardwarelösungen an den Endkunden verkaufen, aber auch offen für zukünftige Entwicklungen im Bereich von *Smart Home* Technologien sein. Folglich haben sie ein Interesse daran, die Schnittstellen ihrer Produkte teilweise offen zu legen und zu vermarkten. Diese Schnittstellen können wiederum von *Smart Service* Anbietern und Entwicklern genutzt werden, um digitale Mehrwertdienste anzubieten, zum Beispiel Prognoseverfahren zum Optimieren des privaten Energieverbrauchs oder Verfahren zur intelligenten Heimautomation. Durch einen elektronischen Marktplatz sollen alle

zur Realisierung einer *Intelligenten Umgebung* erforderlichen Teilnehmer miteinander interagieren und Geschäfte tätigen können. Insbesondere soll es dem Endnutzer ermöglicht werden, in Abhängigkeit von seiner individuellen häuslichen Infrastruktur, verfügbare *Smart Services* zu finden und zu installieren.

6. **Durchgängige Werkzeugunterstützung:** *Wie kann eine durchgängige Werkzeugkette konzipiert werden, um die Entwicklung von Smart Services in Intelligenten Umgebungen zu unterstützen?*

   Das Fehlen von geeigneten Werkzeugen oder Lücken innerhalb von existierenden Werkzeugketten zieht in der Regel in allen softwareorientierten Branchen große Produktivitätseinbußen nach sich. Besonders auf dem *Smart Home* Markt herrscht ein großer Mangel an geeigneten Werkzeugen zur Unterstützung der Entwickler. Aktuell ist aber eine Trendwende zu erkennen, da große Firmen wie Apple und Google nun mit Produkten wie HomeKit[5] oder Nest[6] in den *Smart Home* Sektor eindringen. Aber auch Open Source Projekte wie Eclipse SmartHome[7] bieten eine gute Ausgangsbasis für die zukünftige Entwicklung von *Intelligenten Umgebungen.*

1.2.3 Anforderungen

Der Schwerpunkt der Arbeit liegt auf der Konzeption, dem Entwurf und der Realisierung einer offenen, hochflexiblen, erweiterbaren und auf Standards basierenden Softwarearchitektur für *Smart Services* im Kontext von *Intelligenten Umgebungen.* Im Vordergrund steht neben der dynamischen Integration von unterschiedlichsten *Sensoren* und *Aktuatoren* die Schaffung einer leicht verständlichen, dienstübergreifenden Interoperabilität einzelner *Smart Services.* Die resultierende Plattform sowie die dazugehörigen Entwicklungswerkzeuge sollen als Open Source Software zur Verfügung gestellt werden. Ein detaillierter Überblick über die der Arbeit zugrunde liegenden Anforderungskriterien ist in Kapitel 5.2 zu finden.

1.3 Aufbau der Arbeit

Im Anschluss an das vorliegende Einleitungskapitel gliedert sich die vorliegende Arbeit in vier Hauptteile, welche wiederum in insgesamt elf Kapitel aufgeteilt sind. Die einzelnen Kapitel werden im Folgenden kurz beschrieben. Abbildung 1.3 gibt einen Gesamtüberblick über die Struktur der Arbeit.

[5] https://developer.apple.com/homekit [Letzter Zugriff: 31.01.2015]
[6] https://nest.com [Letzter Zugriff: 31.01.2015]
[7] http://www.eclipse.org/smarthome [Letzter Zugriff: 31.01.2015]

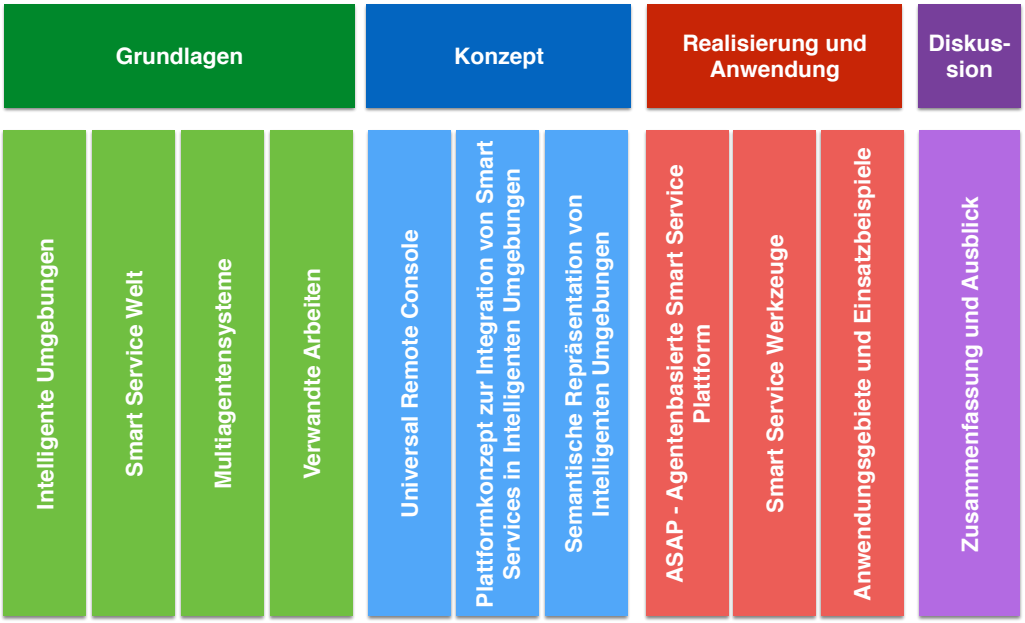


Abbildung 1.3: Übersicht über die Struktur der vorliegenden Arbeit.

Grundlagen

Nach einer einleitenden Motivation und der Definition der relevanten Forschungsziele und -fragen in Kapitel 1 besteht der Grundlagenteil aus vier weiteren Kapiteln. In Kapitel 2 werden die Forschungsgebiete *Intelligente Umgebungen*, *Ambient Intelligence* und *Ambient Assisted Living* eingeführt und gegeneinander abgegrenzt. Kapitel 3 führt das Forschungsgebiet der *Smart Service Welt* ein und gibt einen Überblick über wichtige Begriffserklärungen sowie die vorgeschlagene Referenzarchitektur. Besonders hervorzuheben ist hier die Abgrenzung der Begriffe *Software-definierte Plattform* und *Serviceplattform*. Kapitel 4 beschreibt einen Exkurs in das Themengebiet der *Multiagentensysteme* und legt somit den Grundstein zum Verständnis der späteren Implementierungsarbeiten. Kapitel 5 schließt den Grundlagenteil mit einer Übersicht über verwandte Arbeiten in den relevanten Forschungsgebieten ab.

Konzept

Im zweiten Teil der Arbeit werden die erarbeiteten Konzepte anhand von drei Inhaltskapiteln vorgestellt. Kapitel 6 führt das Konzept einer sicheren und auf Standards basierenden *Smart Home* Plattform am Beispiel der *Universal Remote Console* (URC) ein. Nach der Definition einer Standardarchitektur und einer Beschreibungssprache für personalisierte Benutzerschnittstellen werden Besonderheiten sowie Anwendungs- und Erweiterungsmöglichkeiten der Plattform detailliert beschrieben. Kapitel 7 stellt ein allgemeines Konzept zur Integration von *Smart Services* in *Intelligente Umgebungen* vor. Neben der Vorstellung der grundlegenden Anforderungen werden eine Referenzarchitektur sowie ein durchgängiges Marktplatzkonzept vorgeschlagen. Ein Schwachpunkt des

bisherigen URC Ansatzes ist das Fehlen einer einheitlichen, integrierten und erweiterbaren semantischen Struktur. In Kapitel 8 werden daher semantische Repräsentationen der räumlichen Umgebung, der darin enthaltenen Objekte, der zu integrierenden *Smart Services* sowie deren Beziehungen zueinander modelliert.

Realisierung und Anwendung

Im dritten Teil der Arbeit werden die bisher beschriebenen theoretischen Konzepte realisiert und angewendet. Kapitel 9 beschreibt mit *ASaP* eine Implementierung der *Agentenbasierten Middleware Plattform* für *Intelligente Umgebungen*. Neben den wesentlichen Komponenten der Softwareplattform wird ebenfalls die prototypische Realisierung eines *Smart Service Marktplatzes* ausführlich beschrieben. Kapitel 10 beschreibt mit der *ASaP-Workbench* eine auf Eclipse basierende, durchgängige Werkzeugkette zur Entwicklung und zum Wirkbetrieb von *Smart Services* in *Intelligenten Umgebungen*. Kapitel 11 zeigt schließlich die Anwendbarkeit der vorgestellten Ansätze anhand mehrerer Einsatzgebiete und Anwendungsbeispiele aus unterschiedlichen Forschungsprojekten.

Diskussion

Zum Abschluss der hier vorliegenden Arbeit werden in Kapitel 12 die erzielten Ergebnisse vor dem Hintergrund der wissenschaftlichen Fragestellungen diskutiert. Insbesondere werden die Beiträge der Arbeit aus wissenschaftlicher sowie aus praktischer Sicht näher beleuchtet. Neben einer Aufzählung aller im Rahmen der Arbeit erzielten wissenschaftlichen Veröffentlichungen auf nationalen und internationalen Konferenzen sowie Veröffentlichungen in Fachzeitschriften wird zudem ein Ausblick auf mögliche zukünftige Erweiterungen der Arbeit gegeben.

Teil I

Grundlagen

Der Nachteil der Intelligenz besteht darin, dass man ununterbrochen gezwungen ist, dazuzulernen.

George Bernard Shaw

2 Intelligente Umgebungen

2.1 Einleitung

Das Forschungsgebiet der *Intelligenten Umgebung* spielt eine immer größere Rolle in der Forschungslandschaft und umfasst eine Vielzahl von unterschiedlichen Arbeiten, Zielsetzungen und Forschungsschwerpunkten. In der Literatur werden die Begriffe *Instrumentierte Umgebung*, *Intelligente Umgebung*, *Ambiente Intelligenz*, *Ubiquitous Computing*, *Pervasive Computing* und *Cyber-Physische Umgebungen* oftmals synonym betrachtet, und die Grenzen der einzelnen Bereiche sind über die Jahre immer unschärfer geworden. In diesem Kapitel wird anhand existierender Ansätze und Definitionen eine für die vorliegende Arbeit gültige Definition des Begriffes *Intelligente Umgebung* hergeleitet (Abschnitt 2.2). Nachfolgend werden die grundlegenden Begriffe und wichtigen Komponenten von *Intelligenten Umgebungen* eingeführt. Im Fokus der Betrachtung steht die Abgrenzung der Begriffe *Instrumentierte Umgebung* (Abschnitt 2.3) und *Ambiente Intelligenz* (Abschnitt 2.4) als Grundbestandteil einer *Intelligenten Umgebung* (siehe Abbildung 2.1). In Abschnitt 2.5 werden verwandte Konzepte und Begriffe vorgestellt, die eine komplette oder teilweise Überlappung mit dem Forschungsgebiet der *Intelligenten Umgebungen* aufweisen. Abschließend werden in Abschnitt 2.6 relevante Anforderungen an eine *Intelligente Umgebung* gemäß der eingeführten Definitionen abgeleitet und beschrieben.

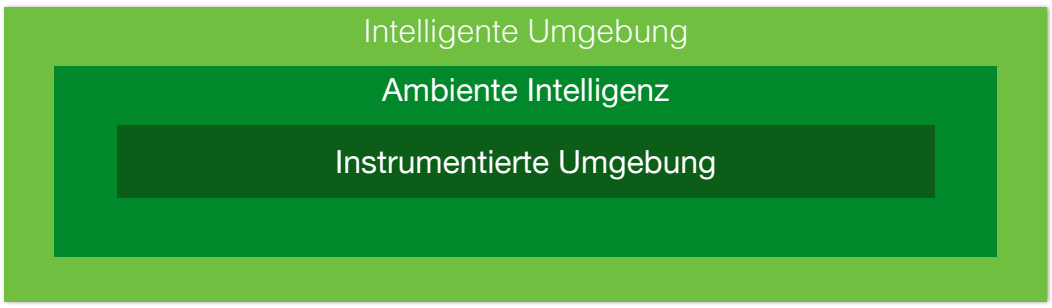


Abbildung 2.1: Zusammenhang zwischen einer *Instrumentierter Umgebung* und *Ambienter Intelligenz* als Grundbestandteil von *Intelligenten Umgebungen*.

2.2 Begriffserklärung

Augusto et al. (2013) definieren eine *Intelligente Umgebung* als den Zusammenschluss mehrerer vernetzter Steuergeräte, welche einerseits verschiedenste Aspekte der Umgebung kontrollieren können und andererseits durch autonome, präemptive Prozesse derart eingesetzt werden, dass ein interaktives und ganzheitliches Umgebungsverhalten erzielt wird, welches den Benutzer in seinem Alltag unterstützt. Cook und Das (2007) geben eine etwas allgemeinere Definition für den Begriff der *Intelligenten Umgebung*:

Definition 1 (*Intelligente Umgebung*)

> *Der Begriff der Intelligenten Umgebung bezeichnet eine Umgebung, welche dazu in der Lage ist, Wissen über die Umgebung und die darin befindlichen Benutzer zu akquirieren und anzuwenden mit dem Ziel, die Benutzer innerhalb der Umgebung optimal zu unterstützen.*

Der Ursprung der *Intelligenten Umgebungen* geht auf Mark Weisers Vision des allgegenwärtigen und in die Umgebung integrierten Computers zurück (Weiser, 1991). Die grundlegende Idee ist hierbei, dass Rechen- und Verarbeitungsleistung nicht mehr nur in zentral ausgerichteten Großrechnern oder *Personal Computern* (PC) realisiert wird, sondern zunehmend in verteilten intelligenten Objekten innerhalb der alltäglichen Umgebung integriert sein wird.

Krumm (2010) unterteilt die Entwicklung moderner Computersysteme in drei unterschiedliche Innovationswellen. Die erste Welle ist durch den Einsatz von Zentralrechnern geprägt, die in der Regel durch Firmen und Organisationen betrieben und von mehreren Personen gleichzeitig genutzt werden. Die zweite Welle ist die Epoche des *Personal Computers* (PC), welcher nach und nach seinen Einzug in Privathaushalte vollzieht. Wie der Name schon besagt, ist die Nutzung des PC durch eine 1:1 Beziehung charakterisiert, das bedeutet eine Person benutzt in der Regel genau einen Computer. Während der aktuell laufenden dritten Entwicklungswelle ist eine regelrechte Explosion der für eine einzelne Person zur Verfügung stehenden Computer zu beobachten. Durch die große Verbreitung von mobilen und vernetzten Geräten wie Smartphone, Tablet oder Smartwatch, hat der Benutzer jederzeit Zugriff auf internetbasierte Dienste und Wissensquellen. Die immer weiter fortschreitende Miniaturisierung der Prozessoren führt zudem zu immer kleineren und leistungsstärkeren Mikrocomputern, welche mittlerweile in nahezu jedes Alltagsobjekt eingebettet werden können.

Die gesamte Entwicklung hat dazu beigetragen, dass heutzutage eine Vielzahl unterschiedlicher Computersysteme innerhalb des Alltags einer einzelnen Person eine wichtige Rolle spielen. Abbildung 2.2 gibt einen Überblick über die beschriebene Trendentwicklung und lässt hierbei auch eine Prognose der fortschreitenden Verbreitung von unterschiedlichen Computersystemen in naher Zukunft erkennen (Krumm, 2010).

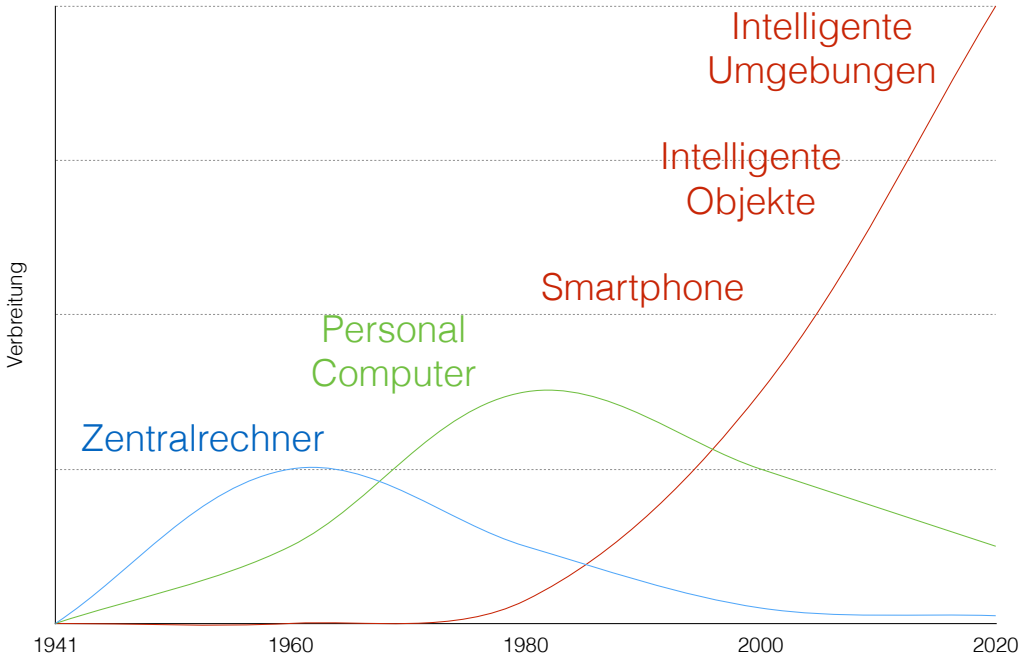


Abbildung 2.2: Trendentwicklung der Verteilung von unterschiedlichen Computersystemen.

In Anlehnung an Mattern (2007) werden drei Aspekte bezüglich des informations- und kommunikationstechnologischen Fortschritts hervorgehoben, welche die Entwicklung von intelligenten Umgebungen maßgeblich vorantreiben:

- **Intelligente Objekte:** Durch die Einbettung von Kleinstcomputern in alltägliche Gegenstände können die digitale und die physische Welt immer besser miteinander verbunden werden. Haupert (2013) stellt beispielsweise ein Architekturkonzept zur Integration intelligenter Objekte (*Smart Products*) anhand einer Infrastruktur für digitale Objektgedächtnisse vor.

- **Kommunikationsinfrastruktur:** Durch den Einsatz von drahtlosen Kommunikationsmechanismen ist es möglich, sogar kleinste physische Objekte an die umgebende Kommunikationsinfrastruktur, beziehungsweise an das Internet, anzuschließen. Somit kann der Zustand dieser Objekte besser erfasst und folglich der Gesamtzustand der Umgebung abgeleitet werden.

- **Datenauswertung und -verarbeitung:** Die Ausstattung von alltäglichen Objekten mit Möglichkeiten zur Datenerfassung und die Anbindung an die umgebenden Kommunikationsinfrastrukturen führt zu enormen Mengen an anfallenden Daten (*Big Data*) (Plattner, 2013; Laney, 2001), die aus der physischen Welt gesammelt werden können. Durch die Kombination mit Informationen aus der digitalen Welt können durch den Einsatz von Informationsextraktionsverfahren hieraus kontextbezogene und personalisierte Daten abgeleitet, analysiert, interpretiert und schließlich

zu neuen Informationen (*Smart Data*) (Kagermann et al., 2015) veredelt werden. Eine ausführliche Definition der Begriffe *Big Data* und *Smart Data* ist in Kapitel 3 zu finden.

Wie zuvor erwähnt, baut das Forschungsgebiet der *Intelligenten Umgebung* auf den Erkenntnissen und Fortschritten in anderen Teilgebieten auf. Abbildung 2.3 gibt einen Überblick über die beteiligten Gebiete.

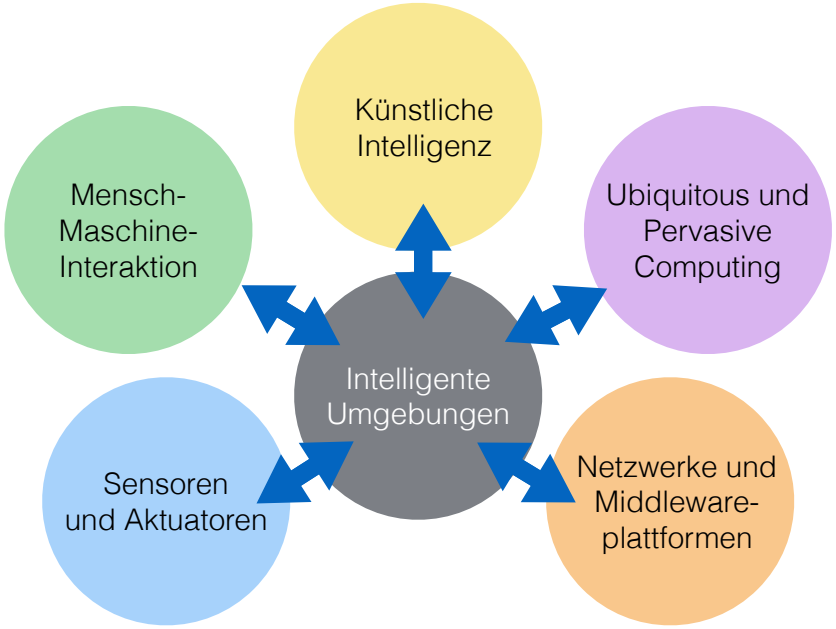


Abbildung 2.3: Übersicht über den Zusammenhang zwischen *Intelligenten Umgebungen* und verwandten Teilgebieten (nach Augusto (2010)).

Augusto (2010) gibt hierzu einen Überblick über die verwandten und relevanten Forschungsfelder und erklärt, inwieweit diese die Weiterentwicklung von *Intelligenten Umgebungen* unterstützt haben. Ein wichtiger Aspekt bei der Realisierung von *Intelligenten Umgebungen* ist die Berücksichtigung des *Kontextes* eines Benutzers und dessen Umgebung. In Anlehnung an Dey (2001) und Chen (2004) wird der Begriff des *Kontext* im Rahmen der Arbeit wie folgt definiert:

Definition 2 (*Kontext*)

Als Kontext wird die Menge der verfügbaren Informationen angesehen, welche eine räumliche Umgebung, entsprechende Umgebungseigenschaften, wie zum Beispiel Lautstärke, Helligkeit und Temperatur, sowie die darin befindlichen Personen, Objekte und Agenten charakterisiert.

Der *Kontext* einer Umgebung ist in der Regel mit einer zeitlichen Abhängigkeit verbunden. Ereignisse innerhalb der Umgebung oder Handlungen von Personen und Agenten stellen Zustandsübergänge dar und führen zu einer Veränderung des aktuellen *Kontext*. Ist ein System dazu in der Lage, seine

Funktionsweise auf den veränderten Kontext zu adaptieren, so spricht man von sogenannten kontextsensitiven Systemen (Lukowicz et al., 2012).

2.3 Instrumentierte Umgebungen

Eine *Instrumentierte Umgebung* stellt einen Kernbestandteil einer *Intelligenten Umgebung* dar. Aufbauend auf Schneider (2010b) wird der Begriff der *Instrumentierten Umgebung* wie folgt definiert:

Definition 3 (*Instrumentierte Umgebung*)

Eine Instrumentierte Umgebung beschreibt einen Ausschnitt der realen Welt, welche mit einer technischen Infrastruktur ausgestattet wurde und dazu in der Lage ist, den aktuellen Kontext der Umgebung zu erfassen und eine Veränderung und Manipulation der Umgebung zu ermöglichen.

Typischerweise besteht eine solche Umgebung aus einer Menge an *Sensoren* und *Aktuatoren*. *Sensoren* reagieren auf physische Stimuli beziehungsweise Zustandsveränderungen, wie zum Beispiel Helligkeits- oder Temperaturveränderungen innerhalb der Umgebung. Allgemein kann der Begriff eines *Sensors* wie folgt definiert werden:

Definition 4 (*Sensor*)

Sensoren sind dazu in der Lage, Zustandsinformationen und Zustandsveränderungen der Umgebung und der darin befindlichen Personen, Objekte und mobilen Agenten zu erfassen und somit den Kontext einer Umgebung zu ermitteln.

Der Begriff des *Aktuators* stammt ursprünglich aus der Mess-, Steuerungs- und Regelungstechnik und bezeichnet im Allgemeinen einen Signalwandler, der eine Eingangsgröße in Form eines elektrischen Steuersignals in eine physikalische Ausgangsgröße umwandelt. In der Robotik werden *Aktuatoren* auch als Effektoren bezeichnet, da beispielsweise das Ergreifen und Bearbeiten eines Gegenstandes einen Effekt innerhalb der Umgebung auslöst. In dieser Arbeit wird der Begriff des *Aktuators* wie folgt definiert:

Definition 5 (*Aktuator*)

Aktuatoren übermitteln Informationen an den Benutzer und sind dazu in der Lage, Zustandsveränderungen und Ereignisse innerhalb der Umgebung auszulösen und somit den Kontext der Umgebung zu beeinflussen.

Um die Vielzahl von *Sensoren* und *Aktuatoren* sinnvoll nutzen zu können, bedarf es in der Regel einer virtuellen Zwischenschicht, welche die *Sensoren*

und *Aktuatoren* miteinander vernetzt und die Verarbeitung der Daten übernimmt. Eine solche Zwischenschicht wird als (*Middleware*) bezeichnet. Lackes und Siepermann (2014a) definieren eine *Middleware* in folgender Weise:

Definition 6 (*Middleware*)

Eine Middleware ist ein Teil der Software-definierten Plattform (siehe Kapitel 3.3), der Kommunikationsdienste für verteilte Anwendungen über Standardschnittstellen bereitstellt und somit eine Integration von unterschiedlichen Anwendungen und deren Daten ermöglicht.

Der Begriff der *Instrumentierten Umgebung* betrachtet Umgebungen also aus einer technisch orientierten Perspektive als eine Kombination aus *Sensoren*, *Aktuatoren* und einer verbindenden *Middleware*. Abbildung 2.4 gibt einen Überblick über die grundlegende hierarchische Struktur.

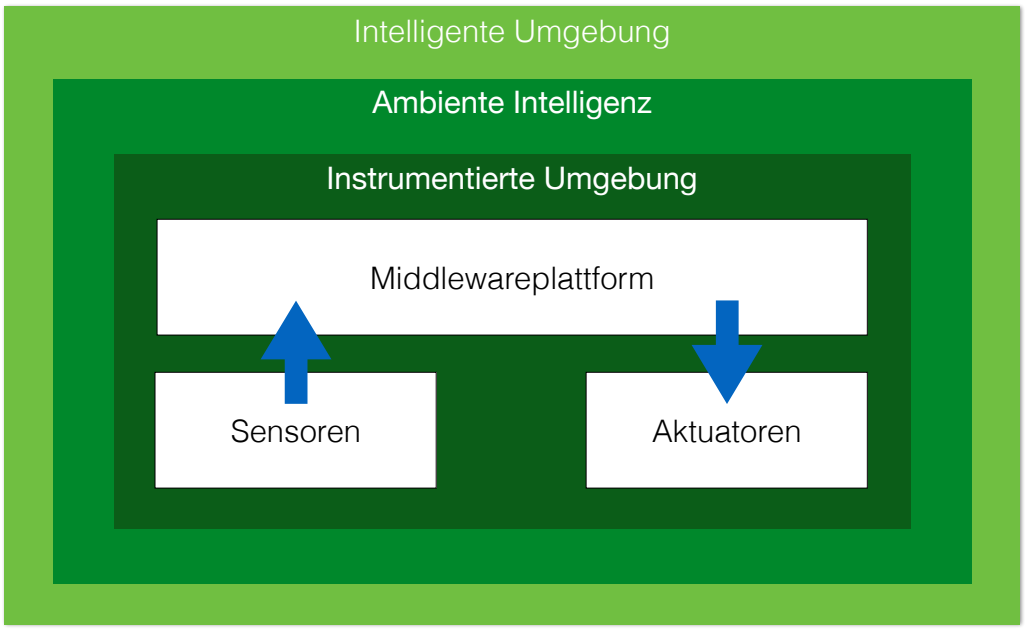


Abbildung 2.4: Überblick über die hierarchische Struktur einer *Instrumentierten Umgebung*.

Zusammengefasst liegt der Fokus einer *Instrumentierten Umgebung* auf der Integration der verfügbaren *Sensorik* und *Aktuatorik* sowie der Bereitstellung der jeweiligen Schnittstellenbeschreibungen.

2.4 Ambiente Intelligenz

Im nächsten Schritt werden, aufbauend auf der *Instrumentierten Umgebung*, intelligente Mehrwertdienste realisiert, um den Benutzer in seinem privaten und beruflichen Leben zu unterstützen. Das Forschungsgebiet der *Ambienten*

Intelligenz (AmI) beschäftigt sich hierzu mit der Idee, alltägliche Objekte sowie gesamte Umgebungen mit Methoden und Berechnungsverfahren der *Künstlichen Intelligenz* anzureichern, um somit dem Benutzer kontextbezogene Assistenzsysteme zur Verfügung zu stellen. Augusto (2007) definiert den Begriff der *Ambienten Intelligenz* wie folgt:

Definition 7 (*Ambiente Intelligenz*)

> *Ambiente Intelligenz (AmI) ist die proaktive und allgegenwärtige Unterstützung von Personen in ihrem privaten und beruflichen Leben.*

Ambiente Intelligenz beschreibt somit eine Vision der Zukunft, in der die Benutzer von intuitiven und intelligenten Schnittstellen umgeben sind, die in ganz unterschiedlichen Alltagsobjekten integriert sind, und wodurch die Umgebung dazu befähigt wird, den Benutzer und seinen Kontext zu erkennen und optimal darauf zu reagieren. Der Fokus liegt weniger auf der technischen Realisierung, sondern vielmehr auf der Art und Weise, wie der Benutzer unterstützt werden kann (Stahl, 2009). Die entsprechenden Assistenzanwendungen setzen hierzu auf der technischen Infrastruktur der unterliegenden *Instrumentierten Umgebungen* auf (siehe Abbildung 2.5).

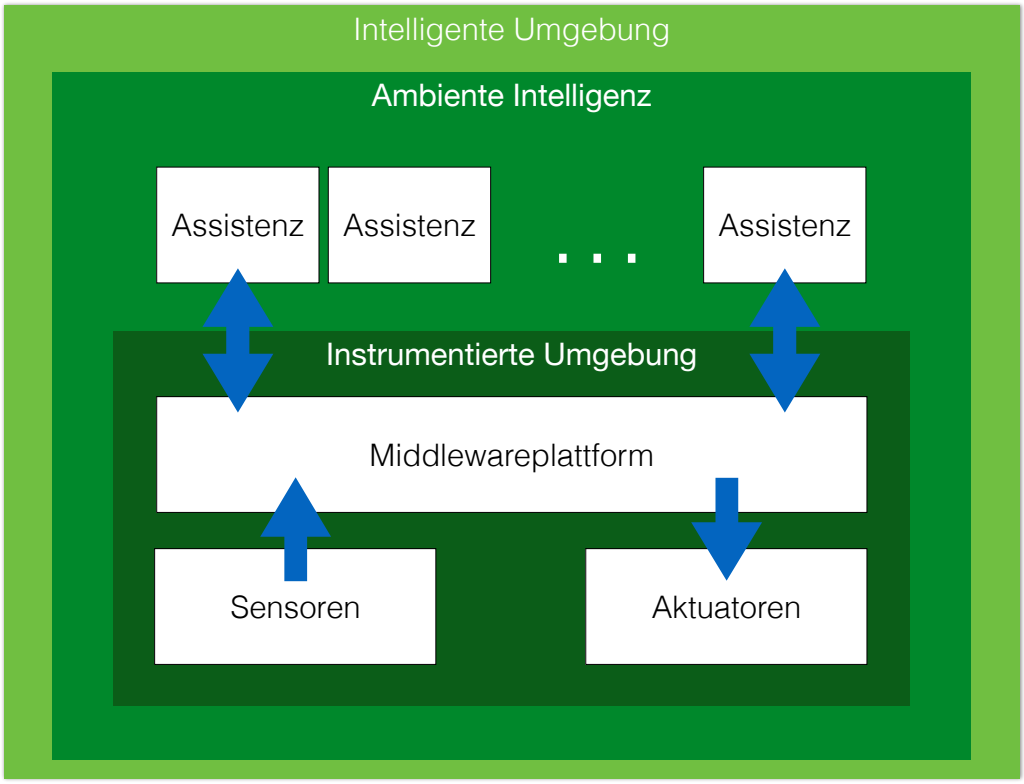


Abbildung 2.5: Übersicht über das Zusammenspiel zwischen *Ambienter Intelligenz* und *Instrumentierten Umgebungen*.

Als relevante Eigenschaften einer Anwendung mit *Ambienter Intelligenz* sind die folgenden fünf Charakteristika zu nennen:

- **Eingebettet:** Intelligente Objekte werden nahtlos in die Umgebung des Benutzers integriert.
- **Situationsbezogen:** Objekte können einzeln oder in Kombination mit anderen den Zustand der Umgebung und somit den Kontext des Benutzers erkennen und ableiten.
- **Personalisiert:** Die gesamte Benutzerinteraktion ist so ausgerichtet, dass sie die persönlichen Vorlieben, Bedürfnisse und das Vorwissen des Benutzers berücksichtigt. Eine wichtige Voraussetzung hierzu ist die Erstellung einer entsprechenden Benutzermodellierung (Wahlster und Kobsa, 1986).
- **Adaptiv:** Die Interaktionsmechanismen und die zur Verfügung gestellten Assistenzsysteme können sich dynamisch auf sich ändernde Gegebenheiten innerhalb der Umgebung oder bezüglich der Ziele eines Benutzers anpassen.
- **Proaktiv:** Um eine bedarfsgerechte Unterstützung anbieten zu können, versuchen die zugrunde liegenden Assistenzanwendungen die individuellen Ziele und folglich die möglichen Handlungen eines Benutzers zu antizipieren.

Das Konzept der *Ambienten Intelligenz* erfordert einen Paradigmenwechsel in Bezug auf die Interaktion mit Computern innerhalb einer definierten Umgebung. Wie bereits in Abbildung 2.2 gezeigt, vollzieht sich hierbei eine Trendwende von der Nutzung eines einzelnen Rechners hin zu der Nutzung von vielen verteilten und nahtlos in die Umgebung integrierten Kleinstcomputern. Diese Trendwende erfordert folglich auch ein Umdenken bezüglich der Mensch-Maschine-Interaktion. Klassische Interaktionskonzepte wie Tastatur und Maus treten dabei mehr und mehr in den Hintergrund, wohingegen neuartige, multimodale Benutzerinteraktionen wie Sprachdialogsysteme, Gesten- und Mikrogestenerkennung sowie beispielsweise *Brain-Computer-Interaction* (BCI) immer mehr in den Fokus der Betrachtungen geraten. Neben der Art und Weise, wie Benutzer zukünftig mit der Umgebung interagieren, ist zudem eine steigende Erwartungshaltung der Benutzer bezüglich einer intelligenten Verhaltensweise der Umgebung zu beobachten. Um proaktive und adaptive Verhaltensmuster aufweisen zu können, muss die Umgebung dazu in der Lage sein, typische Verhaltensweisen der Benutzer zu erlernen und zukünftige Handlungen vorherzusagen. Cook und Das (2007) und Augusto (2010) geben hierzu detaillierte Überblicke über den Stand der Forschung und skizzieren notwendige Herausforderungen zur Realisierung von intelligenten Assistenzsystemen innerhalb *Intelligenter Umgebungen*. Augusto und O'Donoghue (2009) definieren mit der 6W-Architektur die notwendigen Fähigkeiten innerhalb des Lernprozesses von AmI Systemen.

- **Wer:** Identifikation des Benutzers und seiner Rolle im Zusammenspiel mit der *Intelligenten Umgebung*.

- **Wo:** Positionierung des Benutzers und der Objekte innerhalb einer Umgebung.

- **Wann:** Verknüpfung von Aktivitäten und Ereignissen mit zeitlichen Abhängigkeiten.

- **Was:** Erkennen der grundlegenden Aktivitäten des Benutzers.

- **Warum:** Verstehen der Intention des Benutzers während der Ausführung einer Aktivität.

- **Wie:** Finden alternativer Möglichkeiten zur Erreichung der Ziele innerhalb einer *Intelligenten Umgebung*.

Zusammenfassend identifizieren Aztiria et al. (2010) drei relevante Hauptphasen beim Erlernen von individuellen Nutzeraktivitäten:

1. Eine Phase der **Datensammlung**, wobei das System trotz unvollständiger Datenmodelle schon so intelligent wie möglich agieren soll.

2. Eine Phase des **Lernens**, in der sich wiederholende Muster in den gesammelten Daten extrahiert und erkannt werden.

3. Eine Phase der kontinuierlichen **Adaption** der Aktivitätsmuster.

Des Weiteren geben Friedewald und Da Costa (2003) einen Überblick über die notwendigen, technologischen Konzepte zur Realisierung von *Ambienter Intelligenz*.

Bezüglich des zugrunde liegenden Lebenszyklus einer AmI Umgebung lassen sich die wichtigsten Technologien in die folgenden drei Kategorien einordnen (siehe Abbildung 2.6):

1. **Erfassen:** Die Datenerfassung erlaubt dem System Daten über den Zustand der Umgebung und des Benutzers zu ermitteln.

2. **Analyse und Planung:** Die Analyse- und Verarbeitungsverfahren analysieren die erfassten Daten, um zu entscheiden, welche Aktionen, sowohl von der Umgebung als auch von dem Benutzer, notwendig sind, um die zugrunde liegenden Ziele zu erreichen.

3. **Handeln:** Die Handlungskomponente setzt die definierten Handlungspläne innerhalb der Umgebung in die Realität um.

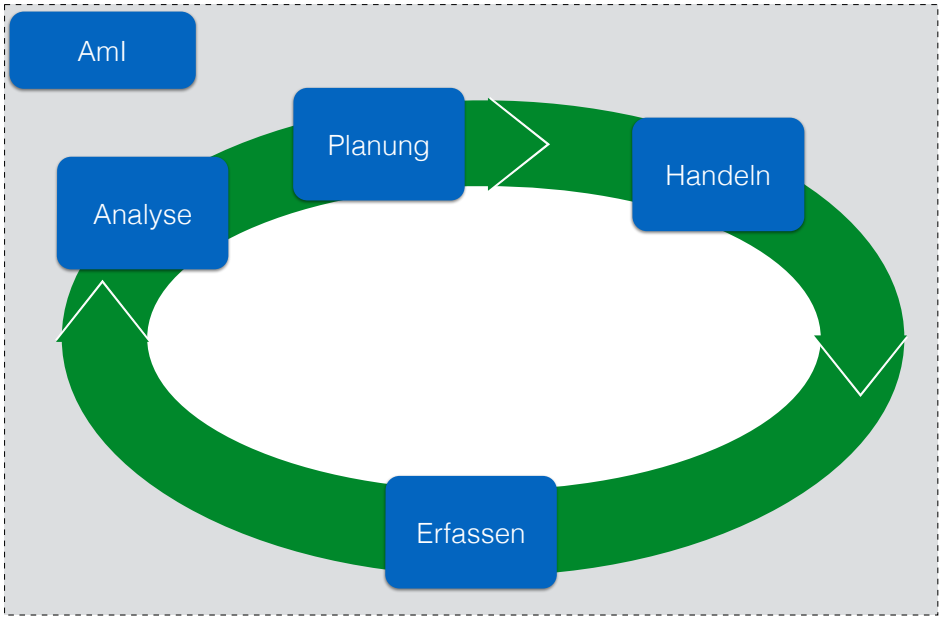


Abbildung 2.6: Übersicht über den Verarbeitungszyklus innerhalb einer AmI Umgebung.

Ambient Assisted Living

Ambient Assisted Living (AAL) ist ein Teilgebiet des Forschungsgebietes *Ambienter Intelligenz* und untersucht, wie entsprechende Methoden und Verfahren eingesetzt werden können, um älteren und unterstützungsbedürftigen Menschen möglichst lange ein selbstbestimmtes Leben innerhalb ihrer häuslichen Umgebung zu ermöglichen. Ein wichtiger Bestandteil ist hierbei das Erkennen von Aktivitäten des täglichen Lebens (ATL) (Reisberg et al., 2001). Cook und Krishnan (2014) geben einen Überblick über den Stand der Forschung auf dem Gebiet der Datenverarbeitung und der Erkennung von Benutzeraktivitäten in häuslichen Umgebungen.

Definition 8 (*Ambient Assisted Living*)

Ambient Assisted Living (AAL) betrachtet einen Teilaspekt von Ambienter Intelligenz und untersucht, wie entsprechende Methoden und Verfahren eingesetzt werden können, um älteren oder unterstützungsbedürftigen Menschen so lange wie möglich ein selbstbestimmtes Leben in ihrer häuslichen Umgebung zu ermöglichen.

Ein detaillierter Überblick über das Forschungsgebiet *Ambient Assisted Living* ist in Augusto et al. (2012) zu finden.

2.5 Verwandte Konzepte und Begriffe

In diesem Abschnitt wird anhand einer für die vorliegende Arbeit gültigen Begriffsbestimmung eine Abgrenzung der bisher eingeführten Begriffe zu verwandten Konzepten und Prinzipien angestrebt und es werden sowohl die Gemeinsamkeiten als auch die Unterschiede herausgearbeitet.

2.5.1 Ubiquitous Computing

Der Begriff des *Ubiquitous Computing* wurde ursprünglich von Weiser (1991) geprägt. Weiser betont hierbei einen Paradigmenwechsel in der Betrachtungs- und Benutzungsweise von Computern im alltäglichen Leben. Seine Vision ist, dass Computer derart in die Umgebung des Benutzers integriert werden können, dass sie für diesen *unsichtbar* werden. Der Begriff *unsichtbar* ist durchaus wörtlich zu verstehen. Durch die fortlaufende Miniaturisierung und Leistungssteigerung der Prozessoren (Moore, 1965) können Alltagsgegenstände zunehmend mit Kleinstcomputern ausgestattet werden, die auf den ersten Blick nicht als solche erkannt werden. Weiterhin interpretiert Weiser *unsichtbar* aber auch metaphorisch im Sinne der Art und Weise, wie der Benutzer die Interaktion mit Computern in seiner Umgebung als solche wahrnimmt. Kernbestandteil des *Ubiquitous Computing* ist die Transformation der Benutzerinteraktion von einer bewussten und als solche direkt wahrnehmbaren, expliziten Benutzung eines Computers hin zu einer unbewussten, impliziten und natürlichen Nutzung der in der Umgebung eingebetteten Rechen- und Verarbeitungsleistung. Diese Transformation führt letztlich zu einer computergestützten Erweiterung der menschlichen Fähigkeiten (Preece et al., 2002). Lackes und Siepermann (2014b) definiert *Ubiquitous Computing* wie folgt:

Definition 9 (*Ubiquitous Computing*)

> *Ubiquitous Computing beschreibt die Allgegenwärtigkeit kleinster, drahtlos miteinander vernetzter Computer, die in beliebige Alltagsgegenstände eingebaut werden können. Mit dem Begriff des Ubiquitous Computing geht ein Paradigmenwechsel, weg vom Computer als Werkzeug hin zu einer impliziten Informationsverarbeitung, einher.*

2.5.2 Pervasive Computing

Der Begriff des *Pervasive Computing* wird oftmals synonym zu dem Begriff *Ubiquitous Computing* verwendet. Preece et al. (2002) beschreiben die grundlegende Idee hinter *Pervasive Computing* als die Möglichkeit, jederzeit und an jedem Ort Zugriff auf Umgebungsinformationen zu haben. *Pervasive Computing* umfasst alle physischen Objekte innerhalb der Umgebung, mit deren Hilfe es dem Benutzer ermöglicht wird, auf die für ihn relevanten Informationen zuzugreifen (Dillon, 2006). Der Begriff des *Pervasive Computing* kann in

folgender Weise definiert werden:

Definition 10 (*Pervasive Computing*)

> *Pervasive Computing beschreibt die Möglichkeit des Benutzers, jederzeit innerhalb einer Umgebung Zugriff auf relevante Informationen zu haben. Dies beinhaltet alle verfügbaren Geräte und Anwendungen, die dem Benutzer diese Funktionalität bereitstellen.*

2.5.3 Cyber-Physische Systeme

Parallel zu den bereits erwähnten Begriffen *Instrumentierte Umgebungen* und *Ambiente Intelligenz* hat sich gerade im Industrie- und Produktionssektor der Begriff der *Cyber-Physischen Systeme* (CPS) etabliert. MacDougall (2013) definiert ein *Cyber-Physisches System* wie folgt:

Definition 11 (*Cyber-Physisches System*)

> *Ein Cyber-Physisches System (CPS) stellt eine Hardware-Software-Kombination dar, durch welche die reale und die virtuelle Welt zusammengeführt werden um eine vernetzte Welt zu erzeugen, in der intelligente Objekte miteinander kommunizieren und interagieren können.*

Kahl (2014) gibt darüber hinaus einen detaillierten Überblick über den Zusammenhang von CPS und deren Komponenten sowie den Zusammenschluss von mehreren CPS zu einer *Cyber-Physischen Umgebung* (CPE). Eine CPE setzt sich aus allen in der Umgebung befindlichen CPS zusammen und kann demnach als *System-of-Systems* (Jamshidi, 2011) bezeichnet werden (siehe Abbildung 2.7).

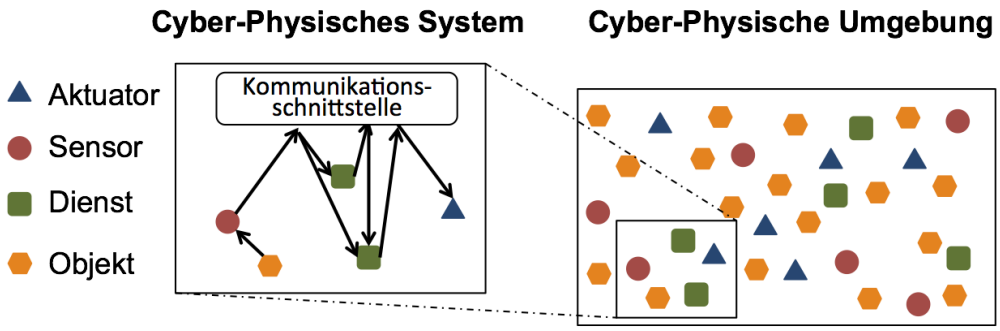


Abbildung 2.7: *Cyber-Physische Umgebungen* als Zusammenschluss mehrerer *Cyber-Physischer Systeme* (aus Kahl (2014).

Ein einzelnes CPS besteht wiederum aus einer Vernetzung von Sensoren, Aktuatoren, Objekten und Diensten. Ein wichtiger Bestandteil von CPS ist

die Schaffung geeigneter Kommunikationsinfrastrukturen, welche einen Informationsaustausch zwischen den einzelnen Komponenten innerhalb eines CPE ermöglichen und somit die Grundlage für die Komposition neuartiger Mehrwertdienste zur Nutzung der Sensorinformationen darstellen. *Cyber-Physische Systeme* und *Cyber-Physische Umgebungen* erfordern folglich eine Anbindung an das Internet. Sind stellen eine wichtige Voraussetzung für die Realisierung des *Internet der Dinge* (Fleisch und Mattern, 2005) dar. Durch die Verschmelzung mit dem *Internet der Dienste* (Heuser und Wahlster, 2011) bilden CPS schließlich eine entscheidende Schlüsseltechnologie für das Zukunftsprojekt *Industrie 4.0* (Kagermann et al., 2013).

2.6 Anforderungen an eine Intelligente Umgebung

Zwei zentrale Eigenschaften, die von *Intelligenten Umgebungen* erwartet werden, sind die Fähigkeit zum proaktiven Handeln sowie die Fähigkeit zur Adaption auf den Benutzer. Um dem Benutzer jederzeit eine auf die Situation passende Assistenz anbieten zu können, muss die *Intelligente Umgebung* kontinuierlich ihren eigenen Zustand sowie den Kontext des Benutzers bestimmen und jederzeit neu entscheiden, wann und wie sie den Benutzer in seinem Handeln unterstützen kann. Dieser Entscheidungsprozess gestaltet sich in vielen Situationen als anspruchsvoll und hängt in hohem Maße von den zur Verfügung stehenden Informationen und dem daraus abgeleiteten Wissen über die Umgebung und den Benutzer ab. Augusto et al. (2013) identifizieren die besondere Schwierigkeit dieses Entscheidungsprozesses darin, dass das System auf der einen Seite antizipieren muss, wann der Benutzer eine Hilfestellung erwartet, und auf der anderen Seite erkennen muss, wann eine Hilfestellung vom Benutzer nicht gewünscht ist. Das Finden eines solchen Gleichgewichts bedarf eines Anpassungsprozesses an die persönlichen Vorlieben des Benutzers. Die *Intelligente Umgebung* muss dazu in der Lage sein, die Benutzer über die Zeit besser kennen zu lernen und somit die persönlichen Fähigkeiten und Präferenzen mit den aktuellen Zielsetzungen und dem Zustand der Umgebung in Bezug zu setzen. Abschließend lassen sich die bisherigen Erkenntnisse anhand der folgenden Anforderungen an eine *Intelligente Umgebung* zusammenfassen (Augusto et al., 2013). Eine *Intelligente Umgebung* muss demnach:

- selbständig Situationen erkennen, in denen eine Benutzerassistenz wünschenswert und sinnvoll ist,
- sich darüber bewusst sein, wann eine Benutzerassistenz erlaubt ist,
- kontextbezogene Hilfestellungen anbieten, die auf die Bedürfnisse und die Wünsche des Benutzers angepasst sind,
- dazu in der Lage sein, die eigenen Ziele zu verfolgen, ohne dabei vom Benutzer technisches Vorwissen vorauszusetzen,

- die Privatsphäre des Benutzers jederzeit schützen,
- die Sicherheit des Benutzers jederzeit gewährleisten,
- autonome Verhaltensweisen aufweisen,
- die eigene Funktionalität sicherstellen, ohne jedoch dabei große Eingriffe und Veränderungen in der Umgebung oder dem normalen Tagesablauf des Benutzers zu fordern,
- sicherstellen, dass der Benutzer jederzeit die volle Kontrolle über die Umgebung hat,
- interoperablel in Bezug auf verfügbare Benutzerschnittstellen und installierte Mehrwertdienste sein, sowie
- resilient gegenüber Störungen und teilweisen Ausfällen sein.

In der vorliegenden Arbeit werden die einzelnen Anforderungen zwar berücksichtigt, es wird hierbei aber keine gezielte Lösung für eine spezifische Anforderung angestrebt. Vielmehr soll ein allgemeines Plattform- und Entwicklungskonzept erarbeitet werden, auf dessen Basis es möglich sein soll, eine Vielzahl von unterschiedlichen Lösungsansätzen zu realisieren.

2.7 Zusammenfassung

In diesem Kapitel wurde ein allgemeiner Überblick über das Forschungsgebiet der *Intelligenten Umgebungen* gegeben. Neben einer Begriffserklärung und einer Abgrenzung zu verwandten Forschungsgebieten wurde eine für die vorliegende Arbeit gültige Definition einer *Intelligenten Umgebung* gegeben. Darüber hinaus wurden die Begriffe *Instrumentierte Umgebung* und *Ambiente Intelligenz* eingeführt und im Kontext der Arbeit definiert. Abschließend wurden allgemeine Anforderungen an eine *Intelligente Umgebung* charakterisiert, welche als Motivation und als Leitfaden für das in der vorliegenden Arbeit entwickelte Plattformkonzept dienen sollen.

Mehr als die Vergangenheit interessiert mich die Zukunft, denn in ihr gedenke ich zu leben.

Albert Einstein

3 Smart Service Welt

3.1 Einleitung

Im Zuge der steigenden Digitalisierung der Produkte und Prozesse in der Wirtschaft, im Gesundheitswesen, in der Bildung aber auch im Privatleben kommt es zu tiefgreifenden Veränderungen in den Geschäftsmodellen und damit verbundenen Organisationssystemen, Netzwerken, Konsum- und Arbeitsformen (Kagermann et al., 2014, 2015). Vernetzte Dienstleistungen und Produkte stellen die Grundlage für neue Geschäftsmodelle auf den unterschiedlichsten Anwendungsebenen dar. Groß angelegte Forschungsprojekte wie SemProM[1], RES-COM[2] und SmartF-IT[3] zeigen, dass es bereits möglich ist, eine Vielzahl alltäglicher Produkte mit Sensoren, Aktuatoren oder sogenannten digitalen Produktgedächtnissen (Haupert, 2013) zu versehen und somit digitale und semantisch modellierte Abbildungen von realen physischen Objekten zu erzeugen. Man spricht in diesem Zusammenhang auch von dem *Internet der Dinge* (Wahlster, 2013).

Definition 12 (*Internet der Dinge*)

Das Internet der Dinge bezeichnet die Verknüpfung eindeutig identifizierbarer physischer Objekte mit einer virtuellen Repräsentation und digitalen Modellierung im Internet. Innerhalb des Internet der Dinge werden passive Objekte zu intelligenten Objekten transformiert, indem diese dazu in die Lage versetzt werden, Daten zu speichern, auf verwandte Daten zu verweisen und Kommunikationsmöglichkeiten für Mensch und Maschine bereitzustellen.

Projekte wie THESEUS[4] haben darüber hinaus die Relevanz von neuartigen, internetbasierten Dienstleistungen für Wirtschaft und Industrie aufgezeigt und wichtige Fortschritte, insbesondere im Hinblick auf semantische Verfahren, für das *Internet der Dienste* (Wahlster, 2014) erzielt.

[1] http://www.semprom.de [Letzter Zugriff: 25.01.2015]
[2] http://www.res-com-projekt.de [Letzter Zugriff: 25.01.2015]
[3] http://www.smartf-it-projekt.de [Letzter Zugriff: 25.01.2015]
[4] http://theseus.pt-dlr.de [Letzter Zugriff: 31.01.2015]

Definition 13 (*Internet der Dienste*)

Das Internet der Dienste beschreibt die Realisierung von Entwicklungs- und Dienstplattformen, um neuartige, webfähige Dienste einfacher und schneller erstellen und im Internet anbieten zu können.[5]

Das Ziel von THESEUS war die Vereinfachung des Zugangs zu Informationen und die Zusammenführung relevanter Daten, um diese neu zu vernetzen und somit die Grundlage für die Entwicklung neuer Dienstleistungen im Internet zu schaffen. Aufbauend auf dem ersten Zukunftsprojekt *Industrie 4.0* legt die Forschungsunion[6] mit dem zweiten Zukunftsprojekt *Smart Service Welt* nun den Fokus auf die Fragestellung, wie man intelligente Produkte mit physischen und digitalen Dienstleistungen zu sogenannten *Smart Services* kombinieren und somit neue Wertschöpfungsketten für die Industrie, für die Wirtschaft aber auch für Endkunden generieren kann. Das übergeordnete Ziel ist also die Verschmelzung des *Internet der Dinge* mit dem *Internet der Dienste* zu einem *Internet der Dinge, Daten und Dienste*. Das Förderprogramm *Smart Service Welt* ist Teil der *Digitalen Agenda* und der *Hightech-Strategie für Deutschland*. Im Vordergrund steht hierbei eine unternehmens- und branchenübergreifende Kooperation, wobei nicht das Produkt oder die Dienstleistung, sondern vielmehr der Konsument in seiner Rolle als Servicenutzer im Mittelpunkt steht. In diesem Kapitel wird nun ein allgemeiner Überblick über das neue Forschungsgebiet der *Smart Services* gegeben. Die hier vorgestellten Ansätze und Konzepte der *Smart Service Welt* basieren in großen Teilen auf den Studien der *Deutschen Akademie der Technikwissenschaften* (Kagermann et al., 2014, 2015).

In Abschnitt 3.2 wird zunächst anhand relevanter Begriffe und Konzepte der Informationsfluss innerhalb einer *Smart Service* Architektur beschrieben. Darauf aufbauend stellt Abschnitt 3.3 die allgemeine Referenzarchitektur einer *Smart Service Welt* vor. Die Architektur ist hierbei hierarchisch strukturiert und besteht aus vier Soft- und Hardwareschichten mit definierten Verantwortlichkeiten.

3.2 Informationsfluss

Der Informationsfluss innerhalb einer *Smart Service* Architektur besteht im Wesentlichen aus vier modular aufgebauten Komponenten. Die Grundlage der gesamten *Smart Service* Architektur bilden zunächst die sogenannten *Smart Spaces*.

[5] http://www.bmwi.de/DE/Themen/Digitale-Welt/Internet-der-Zukunft/internet-der-dienste.html [Letzter Zugriff: 25.01.2015]

[6] http://www.forschungsunion.de [Letzter Zugriff: 06.03.2015]

Definition 14 (*Smart Space*)

Als Smart Space wird eine Intelligente Umgebung bezeichnet, welche eine Kommunikation und Kooperation von Smart Products realisiert, sowie dem Benutzer unterschiedliche Möglichkeiten zur Interaktion anbietet.

Im Wesentlichen setzt sich ein *Smart Space* aus einer Vielzahl von unterschiedlichen *Smart Products* zusammen.

Definition 15 (*Smart Product*)

Das Smart Product beschreibt die Verknüpfung von realen, physischen Produkten mit einer Kombination aus intelligenten Hard- und Softwaresystemen. Durch die Einbettung von Kleinstcomputern können Smart Products als Sensoren und Aktuatoren agieren und somit Daten aus ihrer direkten Umgebung erfassen sowie Informationen in die Umgebung zurückfließen lassen.

Die Vielzahl der möglichen *Smart Products* und die Anbindung an die umgebenden Kommunikationsinfrastrukturen führt zu enormen Mengen an anfallenden Daten (*Big Data*), die mit Hilfe von Datenbanken und Daten-Management-Tools nicht oder nur unzureichend in Echtzeit verarbeitet werden können (Plattner, 2013).

Definition 16 (*Big Data*)

Big Data beschreibt Datenbestände, die aufgrund ihres Umfangs, Vielfalt oder ihrer Schnelllebigkeit nur begrenzt durch aktuelle Datenbanken und Daten-Management-Tools in Echtzeit analysiert und verarbeitet werden können.

Laney (2001) charakterisiert *Big Data* durch das von ihm vorgestellte 3-V-Modell, welches das Datenwachstum anhand von drei Dimensionen beschreibt:

- **Data Volume:** Die *Volume* Dimension bezieht sich auf einen ständig größer werdenden Umfang der erzeugten Daten.
- **Data Variety:** Die *Variety* Dimension bezieht sich auf eine erhöhte Vielfalt und Unterschiedlichkeit der Daten.
- **Data Velocity:** Die *Velocity* Dimension bezieht sich auf eine ansteigende Geschwindigkeit, mit der Daten erzeugt und verarbeitet werden.

Abbildung 3.1 gibt eine Übersicht über das vorgestellte 3-V-Modell.

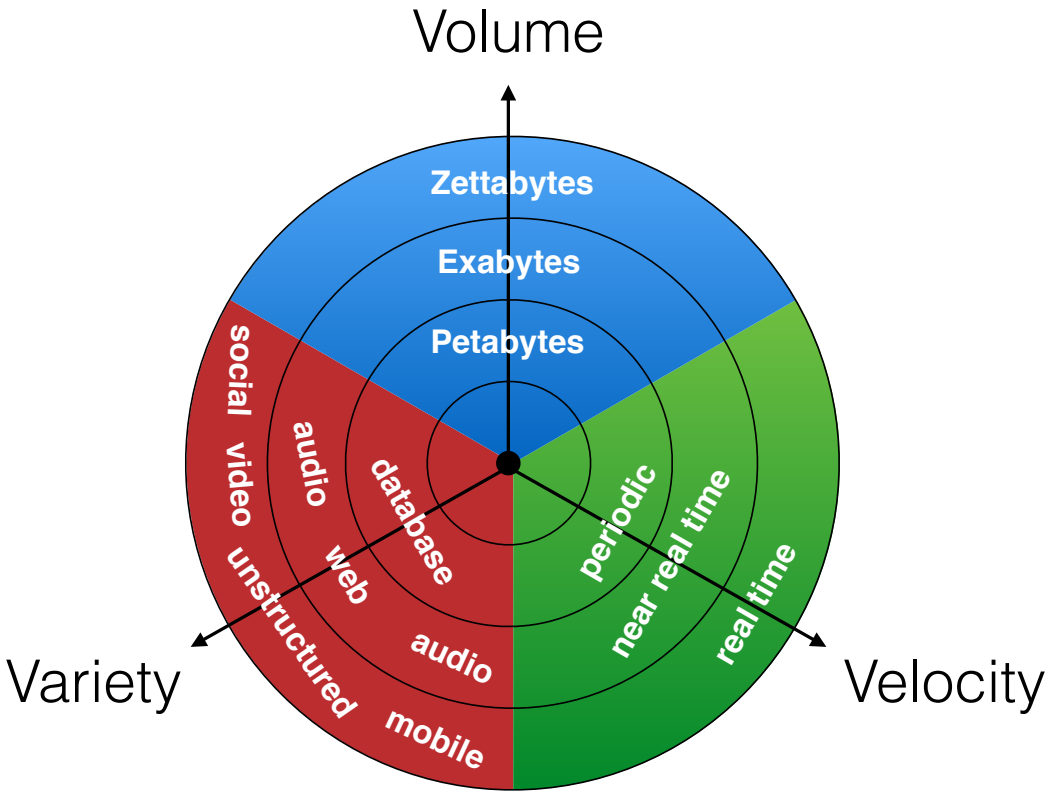


Abbildung 3.1: Übersicht über die drei Dimensionen des 3-V-Modells (Laney, 2001) (Abbildung nach GI-Informatiklexikon[7]).

Durch die Kombination mit Informationen aus der digitalen Welt können durch den Einsatz von geeigneten Verarbeitungsverfahren auf den jeweiligen Anwendungsfall passende neue Informationen abgeleitet werden.

Definition 17 (*Smart Data*)

Als Smart Data werden Datenmengen bezeichnet, die eine sinnvolle semantische Beschreibung beinhalten. Solche Datenmengen entstehen durch die Verarbeitung von großen Datenmengen (Big Data), aus denen mit Hilfe von Informationsextraktionsverfahren und maschinellem Lernen kontextbezogene und personalisierte Daten abgeleitet, analysiert, interpretiert und schließlich zu neuen Informationen veredelt werden (Kagermann et al., 2015).

[7] http://www.gi.de/service/informatiklexikon/detailansicht/article/big-data.html [Letzter Zugriff: 07.03.2015]

Aus einem *Big Data* Bestand können somit je nach zugrunde liegendem Anwendungsfall mehrere unterschiedliche *Smart Data* Informationen abgeleitet werden. Abbildung 3.2 zeigt den beschriebenen Zusammenhang zwischen *Big Data* und *Smart Data*.

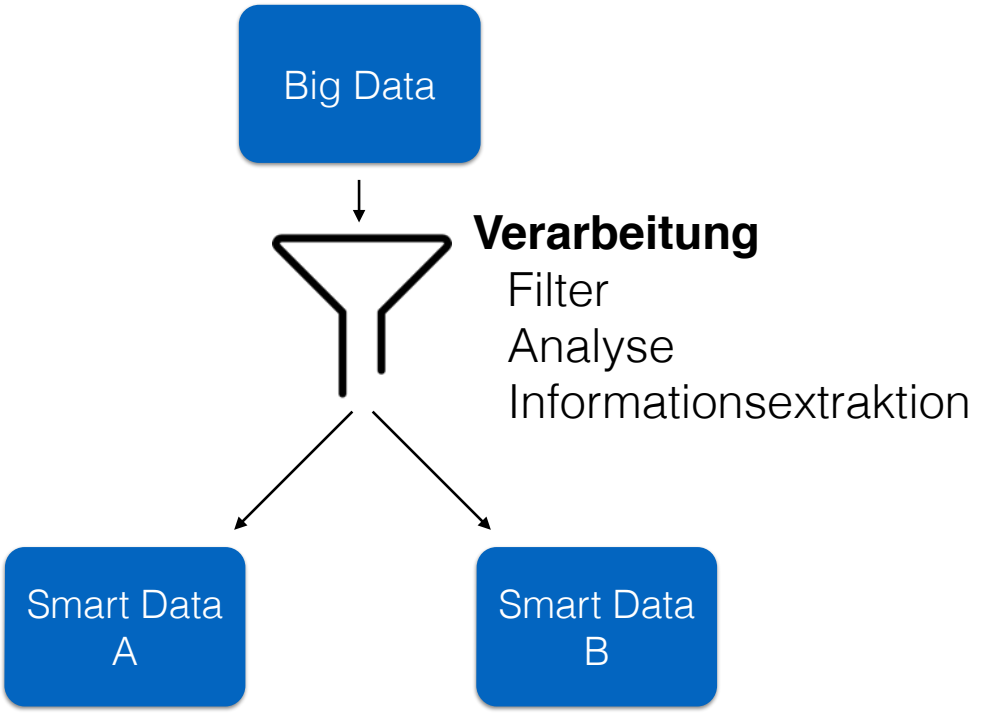


Abbildung 3.2: Übersicht über den Zusammenhang zwischen *Big Data* und *Smart Data*.

Auf Basis der generierten *Smart Data* Informationen können schließlich neuartige *Smart Services* entwickelt werden, welche einen direkten Mehrwert für den jeweiligen Benutzer darstellen sollen.

Definition 18 (*Smart Service*)

Als Smart Service wird die Verschmelzung von digitalen und physischen Dienstleistungen zu einer auf den einzelnen Konsumenten bedarfsgerecht zugeschnittenen Gesamtdienstleistung bezeichnet.

Zusammengefasst liefern *Smart Products* ihre gesammelten Sensorinformationen über den Benutzer und die Umgebung an dedizierte Cloud-Zentren. Diese sind dazu in der Lage solche großen Datenmengen effizient zu verarbeiten (*Big Data*) und im nächsten Schritt relevante Informationen daraus zu extrahieren (*Smart Data*). Mit Hilfe dieser gewonnenen Informationen können *Smart Services* intelligente Mehrwertdienste anbieten, welche einen direkten Einfluss auf die Umgebung des Benutzers haben können. Abbildung 3.3 gibt einen Überblick über die Beziehungen und den Informationsfluss zwischen den vorgestellten Komponenten.

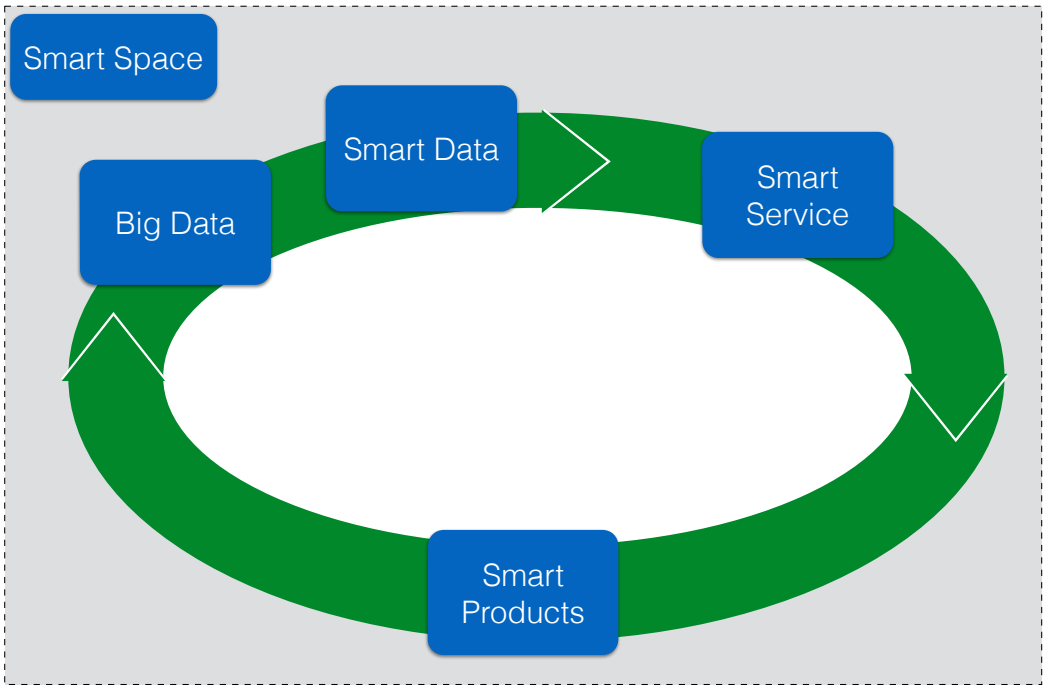


Abbildung 3.3: Überblick über das Zusammenspiel der einzelnen Komponenten und den Informationsfluss innerhalb der *Smart Service Welt*.

Weiterhin sind *Smart Services* beispielsweise dafür verantwortlich, Umgebungsdaten innerhalb des *Smart Space* zu erheben, entsprechend bereitzustellen, die relevanten Daten gegebenenfalls zu filtern und zu aggregieren sowie die Informationen im letzten Schritt nutzer- und bedarfsgerecht zu präsentieren. Der folgende Abschnitt beschreibt, wie die eingeführten Begriffe, Konzepte und der Informationsfluss der *Smart Service Welt* in ein allgemeines Architekturmodell überführt werden können. Hierzu werden die einzelnen logischen Komponenten auf dedizierte funktionale Schichten mit klar aufgeteilten Verantwortungsbereichen verteilt.

3.3 Referenzarchitektur

In Kagermann et al. (2014, 2015) wird eine Referenzarchitektur für *Smart Service* Anwendungen basierend auf vier voneinander getrennten Schichten vorgeschlagen. Die Architektur spiegelt wiederum den zuvor beschriebenen Informationsfluss innerhalb der *Smart Service Welt* wider. Auf der Basis dieser Referenzarchitektur können verschiedenste Hersteller ihre digitalen Geschäftsmodelle auf eigens dafür entwickelten Ökosystemen realisieren. Abbildung 3.4 gibt eine Übersicht über die Struktur und den internen Aufbau des zugrunde liegenden Schichtenmodells.

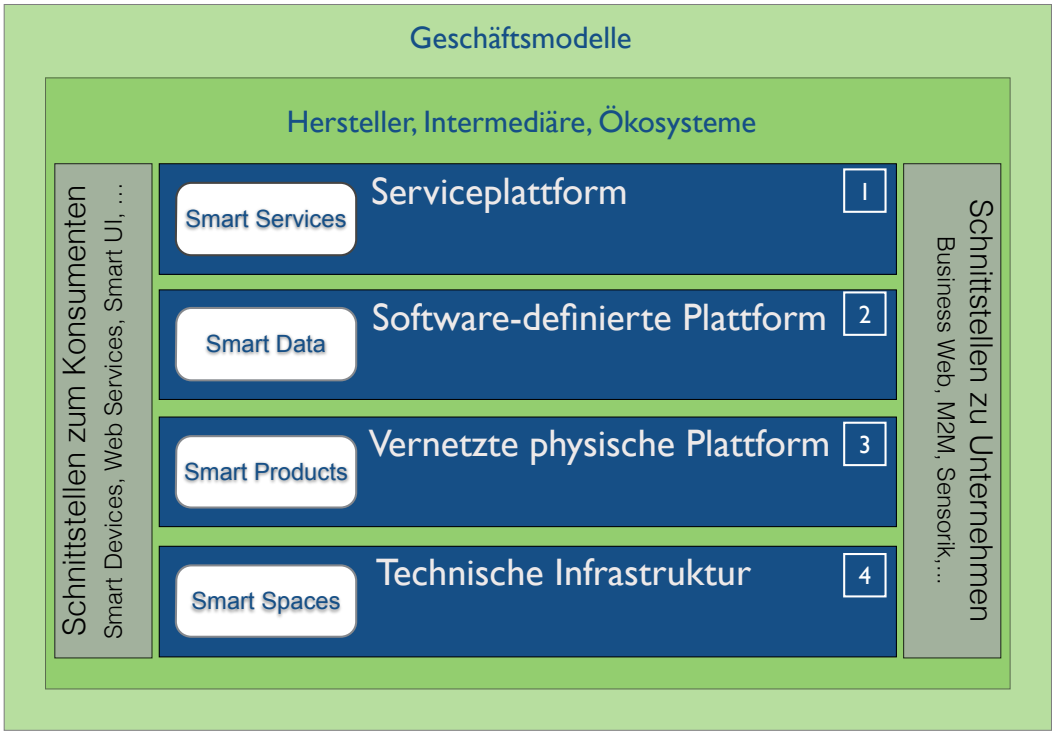


Abbildung 3.4: Übersicht über das Schichtenmodell der *Smart Service Welt* (nach Kagermann et al. (2014, 2015)).

Als Basistechnologien werden auf der untersten Schicht neben einem großflächigen Ausbau der verfügbaren Breitbandnetze ein ubiquitärer Internetzugang, Sensornetze sowie Lokalisierungssysteme als wesentlicher Bestandteil der *Technischen Infrastruktur* angesehen. Die *Technische Infrastruktur* stellt somit die technologische Grundvoraussetzung für die Realisierung eines *Smart Space* und den darauf aufbauenden Anwendungen und Lösungen in Form von *Smart Services* dar.

Definition 19 (*Technische Infrastruktur*)

Die Technische Infrastruktur beschreibt die technisch-infrastrukturelle Voraussetzung zur Realisierung von Smart Spaces und aller darauf aufsetzenden Smart Services.

Die nächste Schicht wird durch sogenannte *Vernetzte physische Plattformen* gebildet. Sie besteht aus einzelnen oder mehreren untereinander vernetzten *Smart Products*.

Definition 20 (*Vernetzte physische Plattform*)

Eine Vernetzte physische Plattform bezeichnet einzelne sowie den Zusammenschluss mehrerer Smart Products, welche über die Technische Infrastruktur vernetzt sind.

Zur Generierung neuer digitaler Mehrwertdienste ist eine Zusammenarbeit einzelner *Vernetzter physischer Plattformen* erforderlich. Auf der nächsten Schicht sind sogenannte *Software-definierte Plattformen* dafür zuständig, die verfügbaren und relevanten *Vernetzten physischen Plattformen* zusammenzuschließen und die entsprechende Konnektivität für die enthaltenen *Smart Products* bereitzustellen.

Definition 21 (*Software-definierte Plattform*)

Software-definierte Plattformen sind die technologische Basis für neuartige, internetbasierte Dienstleistungen. Sie können durch Virtualisierung die Zusammenarbeit unterschiedlicher Vernetzter physischer Plattformen sicherstellen und die Konnektivität einzelner Smart Products bereitstellen.

Auf der Ebene der *Software-definierten Plattformen* werden die gesammelten Daten der einzelnen *Smart Products* zum Beispiel in hochperformanten Cloud-Zentren analysiert sowie die für den jeweiligen Anwendungsfall relevanten Informationen extrahiert und zu neuem Wissen in Form von *Smart Data* aggregiert. Es ist davon auszugehen, dass es, je nach Anwendungsfall, eine Vielzahl solcher *Software-definierten Plattformen* geben wird. Die daraus resultierenden intelligenten Mehrwertdienste und Anwendungen stellen die Grundlage für digitale Geschäftsmodelle und Wertschöpfungsketten dar. Zur Realisierung dieser internetbasierten Geschäftsmodelle, in denen digitale und physische Dienstleistungen dynamisch zu *Smart Services* kombiniert werden sollen, kommen auf der obersten Schicht sogenannte *Serviceplattformen* zum Einsatz.

Definition 22 (*Serviceplattform*)

Eine Serviceplattform stellt eine unternehmens- und branchenübergreifende Kollaborationsplattform dar, auf der Wertschöpfungsketten modular konfiguriert und zu neuen, internetbasierten Geschäftsmodellen zusammengeführt werden können.

Auf einer *Serviceplattform* stellen verschiedene Produkt- und Dienstanbieter Mehrwertdienste in Form von *Smart Services* für den Konsumenten zur Verfügung. *Serviceplattformen* ermöglichen hierbei die Verknüpfung einzelner *Smart Services* zu kombinierten Mehrwertdiensten.

Auf der Basis der durch die *Software-definierten Plattformen* bereitgestellten Schnittstellen haben *Smart Services* Zugriff auf die erfassten *Smart Data* Informationen. Die wesentliche Funktion einer *Serviceplattform* ist das Ermöglichen einer Kollaboration aller relevanten Marktteilnehmer. *Serviceplattformen* definieren hierzu die zur Zusammenarbeit erforderliche Kollaborationsumgebung. Die jeweiligen Prozesse, Schnittstellen, Werkzeuge, Standards und Spielregeln für die Interaktion und Kollaboration der Teilnehmer sind ebenfalls auf den *Serviceplattformen* enthalten.

Die in der realen Welt vorhandenen digital-anschlussfähigen Objekte und

Produkte verfügen über eine digitale Repräsentanz sowohl auf den *Serviceplattformen* als auch auf den *Software-definierten Plattformen*. Im Folgenden wird eine kurze Abgrenzung beider Begriffe gegeben.

Wie zuvor erwähnt, stellt die *Serviceplattform* eine Kollaborationsumgebung zur Verfügung, auf deren Basis sowohl der Endnutzer die Möglichkeit hat, neuartige *Smart Services* zu finden und zu benutzen, als auch der Dienstanbieter dazu befähigt wird, auf Basis bestehender Dienste neue *Smart Services* zu schaffen. *Serviceplattformen* dienen somit als Integrationspunkt für neue digitale Wertschöpfungsketten und Geschäftsmodelle. Durch Prozessdefinitionen, Schnittstellenbeschreibungen, Standards und Werkzeuge geben *Serviceplattformen* einen formalen Rahmen für die (betriebswirtschaftlich orientierte) Entwicklung von *Smart Services* auf der Basis der unterliegenden *Software-definierten Plattformen*.

Die *Software-definierte Plattform* stellt die virtuelle Laufzeitumgebung für *Smart Services* zur Verfügung. Sie beinhaltet die grundlegenden Basistechnologien und Bestandteile, die für den Betrieb von *Smart Services* und die Verknüpfung mit der realen Welt, den *Smart Products* und deren Anwendern, notwendig sind. Insbesondere spielen eine semantische Dienstintegration sowie effiziente Mechanismen zum Auffinden und zur Koordination von *Smart Services* eine wichtige Rolle.

Software-definierte Plattformen und *Serviceplattformen* stehen in einer n:m-Beziehung. Das bedeutet einerseits, dass eine *Software-definierte Plattform* als Grundlage für mehrere *Serviceplattformen* dienen kann, und andererseits, dass eine *Serviceplattform* Dienste nutzt, die von unterschiedlichen (gegebenenfalls hersteller- und anwendungsfallspezifischen) *Software-definierten Plattformen* bereitgestellt werden.

Tabelle 3.1 fasst die funktionalen Eigenschaften einer *Software-definierten Plattform* und einer *Serviceplattform* zusammen und ermöglicht somit eine bessere Abgrenzung beider Begriffe.

Kriterien	***Software-definierte Plattform***	***Serviceplattform***
Abstraktionsebene	• Technisch orientiert • Verarbeitungsorientiert • Domänenneutral	• Betriebswirtschaftlich orientiert • Dienstleistungsorientiert • Domänenspezifisch
Abhängigkeiten	• Abhängig von der unterliegenden *Vernetzten physischen Plattform*	• Unabhängig von den *Vernetzten physischen Plattformen* • Abhängig von der *Software-definierten Plattformen*
Zielgruppe	• Serviceplattform- und Hardwareanbieter, Anwendungsentwickler	• Diensteanbieter, Intermediär und Endnutzer
Schnittstellen	• Über Programmierschnittstellen zur *Vernetzten physischen Plattform* • Über semantische Dienstbeschreibungen zu übergeordneten *Smart Services* • Über multimodale Schnittstellen zum Diensteanbieter	• Über Vererbung von der *Software-definierten Plattform* zum Endnutzer • Über *Software Development Kits* (SDK) zum Dienstanbieter • Über multiadaptive Mensch-Maschine Interaktion zum Endnutzer
Anforderungen	• Durchgängige Virtualisierung • Konkrete Service-Orchestrierung • Dienstesemantik • Mobilitätsunterstützung • Sicherheit, Robustheit, Zuverlässigkeit, Resilienz • Skalierbarkeit und Performanz (auch für *Big Data* mit Echtzeitverarbeitung)	• Kontextualisiertes, adaptives Service-Engineering • Abstrakte Service-Orchestrierung und -Choreographie (Mashup) • Service-Automatisierung • Effizientes Service-Monitoring • Web- und Cloud-Fähigkeit
Notwendiges Fachwissen (Designzeit)	• Technologisches Fachwissen • Integrationswissen	• Prozesswissen, Ablauforganisation • Geschäftsmodelle, Aufbauorganisation

Tabelle 3.1: Überblick über die funktionalen Eigenschaften einer *Software-definierten Plattform* und einer *Serviceplattform* (Kagermann et al., 2015).

3.4 Zusammenfassung

In diesem Kapitel wurde das Forschungsgebiet der *Smart Service Welt* eingeführt und die relevanten Begriffe und Konzepte beschrieben. Neben dem Informationsfluss der einzelnen Komponenten wurde eine abstrakte Referenzarchitektur bestehend aus vier unterschiedlichen Schichten vorgestellt. Jede Schicht deckt hierbei bestimmte Aufgabenbereiche auf jeweils unterschiedlichen Abstraktionsebenen ab. Das Zusammenspiel aller Komponenten ermöglicht schließlich die Schaffung neuartiger, digitaler Mehrwertdienste in Form von *Smart Services*. Damit diese innovativen Dienstleistungen für die Wirtschaft entstehen können, sind Lösungen für eine zielorientierte Kombination von *Smart Products*, *Smart Data* und *Smart Services* erforderlich. Dazu gehört insbesondere auch der Aufbau und Betrieb von Plattformen, die als Schnittstelle zwischen Entwicklern, Plattformbetreibern, Dienstanbietern und Konsumenten in den jeweiligen Anwendungsbereichen dienen. Ein wichtiger Bestandteil zur Erreichung eines hohen Grades an Interoperabilität ist hierbei die semantische Beschreibung aller relevanten Informationen. Die Beschreibung umfasst die semantische Modellierung der *Smart Spaces*, der darin enthaltenen *Smart Products*, des Benutzers und der *Smart Services*.

In der vorliegenden Arbeit wird ein in die *Smart Service Welt* eingebettetes Plattformkonzept erarbeitet (Kapitel 7), mit dessen Hilfe *Smart Services* im Kontext von *Smart Home* Technologien realisiert werden sollen. Hierzu wird eine *Software-definierte Plattform* implementiert, auf deren Basis unterschiedliche *Smart Prodcuts*, wie *Sensoren* und *Aktuatoren*, in einer *Intelligenten Umgebung* integriert werden können (Kapitel 9). Darauf aufbauend können unterschiedliche Arten von Mehrwertdiensten entwickelt und anschließend auf einer *Serviceplattform* verteilt und vertrieben werden. Die *Serviceplattform* dient hierbei unter anderem als *Smart Service* Marktplatz, auf dem Dienstanbieter, Dienstentwickler und Endkunden zusammengebracht werden. Weiterhin wird eine semantische Modellierung (Kapitel 8) der entwickelten *Smart Services* vorgeschlagen, welche zusammen mit einer integrierten Entwicklungsumgebung (Kapitel 10) den Ansatz abrunden.

Die Lösung ist immer einfach, man muss sie nur finden.

Alexander Solschenizyn

4 Multiagentensysteme

4.1 Einleitung

Aufbauend auf dem Forschungsgebiet der verteilten künstlichen Intelligenz haben agentenbasierte Technologien zum Ziel, komplexe Probleme durch Koordination und Kooperation von einzelnen autonomen Softwareagenten zu lösen. Der breite Anwendungsbereich von agentenbasierten Technologien lässt ein hochgradig interdisziplinäres Forschungsgebiet erkennen, was zudem die exakte Definition des Agentenbegriffs erschwert. Klusch (2001) betont hierbei jedoch den Begriff der Verhaltensautonomie, welcher durch die Aspekte Proaktivität, Reaktivität und soziales Verhalten charakterisiert ist. Große Bedeutung kommt den Zielen, Plänen und Aktionen der einzelnen *Agenten* zu. Auch die Fähigkeit, mit anderen *Agenten* und dem Benutzer zu kommunizieren, spielt eine wichtige Rolle. Der Begriff der *Agentenorientierten Softwareentwicklung* (AOSE) (Jakob und Weiß, 2004) beschreibt die Verwendung von Methoden der klassischen Softwaretechnik zur Realisierung von Softwareagenten und *Multiagentensystemen*.

Dieses Kapitel gibt eine allgemeine Einführung in das Themenfeld der *Multiagentensysteme* und dient als Grundlage für das in Kapitel 7 vorgestellte Konzept zur einheitlichen Integration von *Smart Services* in *Intelligente Umgebungen* sowie für die in Kapitel 9 beschriebene Realisierung einer multiagentenbasierten *Middlewareplattform* für *Intelligente Umgebungen*. Abschnitt 4.2 beschreibt den Begriff des *Agenten* und führt die relevanten Begriffe innerhalb des Themengebietes ein. Die drei folgenden Abschnitte geben einen Überblick über die grundlegende Plattformarchitektur (siehe Abschnitt 4.3), Kommunikationsmechanismen (siehe Abschnitt 4.4) sowie Kooperations- und Koordinationsmechanismen von *Multiagentensystemen* (siehe Abschnitt 4.5). Kapitel 4.6 beschreibt abschließend verfügbare Implementierungen von Multiagentenplattformen.

4.2 Agentenbegriff

Wooldridge und Jennings (1995) betonen die Schwierigkeit einer umfassenden Definition des Agentenbegriffs und unterscheiden zwischen einer allgemein gültigen und akzeptierten Ansicht und einer etwas genaueren, aber dafür auch kontrovers diskutierten Sichtweise. Allgemein betrachtet definieren sie einen

Agenten als ein Soft- oder Hardwaresystem, welches durch die folgenden Merkmale gekennzeichnet ist:

- **Autonomität:** *Agenten* sind dazu in der Lage autonom, das heißt ohne direkte Intervention eines menschlichen Benutzers zu agieren und hierbei jederzeit die Kontrolle und den Überblick über die eigenen Handlungen und den eigenen internen Zustand zu haben.
- **Sozialfähigkeit:** *Agenten* interagieren mit anderen *Agenten*, beziehungsweise mit menschlichen Benutzern mittels einer definierten Kommunikationssprache.
- **Reaktivität:** *Agenten* können ihre Umgebung wahrnehmen und in angemessener Art und Weise auf Veränderungen innerhalb der Umgebung reagieren.
- **Proaktivität:** *Agenten* besitzen ein zielorientiertes Verhalten und versuchen, die eigenen Ziele sowohl eigenmächtig als auch in Kooperation mit anderen *Agenten* zu erreichen.

Abbildung 4.1 zeigt eine abstrakte Sicht auf einen einzelnen *Agenten* und dessen Einbettung innerhalb einer Umgebung. Die Abbildung veranschaulicht, wie der *Agent* die Inputsignale aus der Umgebung aufnimmt, diese eventuell weiterverarbeitet und schließlich durch seine zielgerichteten Aktionen und Handlungen die Umgebung beeinflusst.

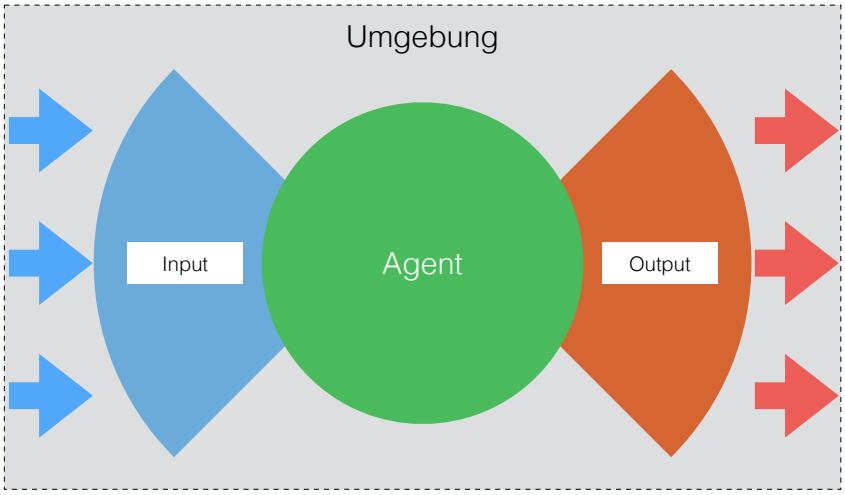


Abbildung 4.1: Übersicht über die vereinfachte Struktur eines *Agenten*.

Besonders im Forschungsgebiet der *Künstlichen Intelligenz* werden die bisher angegebenen Aspekte noch um einige Charakteristika ergänzt, was letztlich zu einer etwas engeren Betrachtungsweise des Agentenbegriffs führt. Hierbei wird oftmals versucht, *Agenten* mit typischen menschlichen Kriterien zu beschreiben. So spielen zum Beispiel der mentale Geisteszustand, Wissen, Glaube, Absicht und Verpflichtung aber auch der emotionale Zustand des *Agenten*

eine Rolle. Aufbauend auf diesen Betrachtungen leitet Wooldridge (2009) eine Definition für den Begriff des *Agenten* ab, welche als Grundlage für die weiteren Betrachtungen innerhalb dieser Arbeit dienen soll:

Definition 23 (*Agent*)

> *Ein Agent ist ein Computersystem, welches innerhalb einer bestimmten Umgebung dazu in der Lage ist, autonome Handlungen und Aktionen auszuführen, mit dem Zweck, die zuvor definierten und geplanten Ziele zu erreichen.*

Agenten unterscheiden sich in der Regel bezüglich ihrer internen Agentenarchitektur. Unter der Agentenarchitektur versteht man grundsätzlich die Art und Weise, wie das eigene Wissen und das daraus resultierende Verhalten eines Agenten bestimmt wird. Wooldridge (2002) unterscheidet zwischen zwei grundlegenden Agentenarchitekturen:

- **Reaktive Agentenarchitektur:** *Reaktive Agenten* umfassen keine Repräsentation des eigenen Wissens, das heißt ihre Berechnungen und ihre Aktionen basieren ausschließlich auf der direkten Wahrnehmung ihrer Umgebung.

- **Kognitive / Deliberative Agentenarchitektur:** *Kognitive Agenten* besitzen eine Abbildung ihrer Umwelt, wodurch sie in der Lage sind, ihre Aktionen und Handlungen zielgerichteter zu planen.

Eine besondere Klasse der *Kognitiven Agenten* stellen *Agenten* basierend auf den drei mentalen Attributen *Belief*, *Desire* und *Intention* (BDI) dar. Der BDI Ansatz geht zurück auf die Arbeiten von Bratman (1987) und ist angelehnt an die tatsächlichen kognitiven Verhaltensmuster eines Menschen. Braubach et al. (2005) geben eine allgemeine Beschreibung der einzelnen Attribute:

- **Belief:** *Beliefs* repräsentieren den aktuellen und subjektiven Informations- und Wissensstand eines *Agenten*, sowohl über sich selbst als auch über den Zustand seiner Umgebung. Wichtig hervorzuheben ist hierbei, dass dieses Wissen nicht zwangsläufig der Wahrheit entsprechen muss. Vielmehr resultiert es aus der Art und Weise, wie der *Agent* seine Umwelt wahrnimmt und diese interpretiert. Typischerweise werden *Beliefs* in einer dedizierten Datenbank (*belief base* oder *belief set*) gespeichert.

- **Desire:** *Desires* beschreiben die Absicht und die grundlegende Motivation hinter den Handlungen eines *Agenten*. *Desires* müssen hierbei keine konsistente Menge bilden sondern können auch gegensätzliche oder sich gegenseitig ausschließende Elemente enthalten. Als Menge von Zielen wird hingegen eine konsistente Menge von *Desires* bezeichnet, von welcher der *Agent* überzeugt ist, dass diese aktuell erreichbar sind.

- **Intention:** *Intentions* umfassen schließlich die eigentlichen Handlungen und Aktionen eines *Agenten* um bestimmte Ziele zu erreichen. *Intentions* basieren auf allgemeinen Handlungsplänen. Pläne beschreiben die Art und Weise, wie ein *Agent* eine Menge von Zielen erreichen kann. Durch ein planvolles und zielorientiertes Vorgehen wird schließlich ein proaktives Agentenverhalten ermöglicht.

Rao und Georgeff (1995) beschreiben eine abstrakte BDI Architektur sowie einen entsprechenden allgemeinen Interpretationsalgorithmus für *BDI Agenten*:

```
initialize-state();
repeat
    options := option-generator(event-queue);
    selected-options := deliberate(options);
    update-intentions(selected-options);
    execute();
    get-new-external-events();
    drop-successful-attitudes();
    drop-impossible-attitudes();
end repeat
```

Zu Beginn eines jeden Zyklus wird zunächst, ausgehend von dem aktuellen Zustand, die Menge der ausführbaren Pläne bestimmt. Durch einen Auswahlprozess werden im nächsten Schritt diejenigen Pläne ausgewählt, welche vom *Agenten* tatsächlich ausgeführt werden sollen, und zu den *Intentions* hinzugefügt. Zum Abschluss werden neue Umgebungsereignisse aufgenommen und bereits erfüllte oder unmöglich zu erreichende Ziele entfernt.

Bezüglich der Umgebung, in der sich die *Agenten* befinden und in der sie handeln und Pläne ausführen können, schlagen Russell und Norvig (2009) eine Klassifizierung entsprechend der folgenden fünf Kategorien vor:

- **Zugänglich vs. unzugänglich:** Eine Umgebung wird als *zugänglich* klassifiziert, wenn ein *Agent* jederzeit einen vollständigen Zugriff auf alle Zustandsinformationen der Umgebung hat. Eine Umgebung ist *zugänglich*, wenn alle für den Entscheidungsprozess des *Agenten* relevanten Informationen durch *Sensoren* erfasst werden können. Eine *zugängliche* Umgebung hat für den *Agenten* den Vorteil, dass eine Zwischenspeicherung der Zustandsinformationen nicht notwendig ist.

- **Deterministisch vs. nichtdeterministisch:** Wird der Zustand einer Umgebung vollständig und eindeutig durch den aktuellen Zustand und die Handlungen des *Agenten* bestimmt, wird die Umgebung als *deterministisch* bezeichnet. Geht man folglich von einer *zugänglichen* und *deterministischen* Umgebung aus, ist es für den *Agenten* nicht notwendig, Unsicherheit und mit Wahrscheinlichkeiten behaftete Werte in seiner Berechnung zu berücksichtigen.

- **Kurzfristig vs. langfristig:** In einer *kurzfristigen* Umgebung basiert die Erfahrung eines einzelnen *Agenten* auf kurzen und in sich geschlossenen Episoden. Jede Episode besteht aus der Wahrnehmung der Umgebung und einer oder mehreren daran anschließenden Handlungen. Die Bewertung der Handlung basiert jeweils nur auf der entsprechenden Episode. Ein vorausschauendes Verhalten ist in *kurzfristigen* Umgebungen nicht vorgesehen.

- **Statisch vs. dynamisch:** Verändert sich die Umgebung während den Berechnungen eines *Agenten*, wird diese als *dynamisch* klassifiziert. *Statische* Umgebungen verändern sich dementsprechend nicht und sind für den *Agenten* leichter zu behandeln, da dieser während seinen Berechnungen nicht ständig die Umgebung überwachen muss.

- **Diskret vs. kontinuierlich:** Eine Umgebung mit einer definierten und begrenzten Anzahl von möglichen Zuständen und Aktionen wird als *diskret*, andernfalls als *kontinuierlich* bezeichnet.

Ein Zusammenschluss von mehreren *Agenten* mit dem Ziel, durch Informationsaustausch ein übergeordnetes Ziel zu erreichen, wird schließlich als *Multiagentensystem* bezeichnet. Wooldridge (2009) betont, dass einzelne *Agenten* unterschiedliche Ziele und Beweggründe haben können, und dass daher Kooperations-, Koordinations- und Verhandlungsbereitschaft von zentraler Bedeutung bei der Kommunikation der *Agenten* untereinander sind.

Definition 24 (*Multiagentensystem*)

Ein Multiagentensystem (MAS) besteht aus einer Menge von Agenten, welche untereinander Informationen austauschen und durch Kooperation, Koordination und Kollaboration ein übergeordnetes Gesamtproblem lösen.

Weiss (2000) betont die Tatsache, dass sich *Multiagentensystem* in erster Linie darin unterscheiden, aus welchen *Agenten* und Agentenarchitekturen sie sich zusammensetzen, wie sie untereinander interagieren und in welcher Umgebung sie sich befinden. Weiss (2000) leitet weiterhin vier Hauptmerkmale von *Multiagentensystemen* ab:

- Jeder einzelne *Agent* innerhalb eines *Multiagentensystems* hat unvollständige Informationen über die Umwelt und ist bezüglich seiner Fähigkeiten beschränkt.

- Die Systemsteuerung und Kontrolle innerhalb eines *Multiagentensystems* ist verteilt.

- Die Datenhaltung eines *Multiagentensystems* ist dezentral.

- Die Berechnungen innerhalb eines *Multiagentensystems* erfolgen asynchron.

4.3 FIPA Plattformarchitektur

Aufgrund der wachsenden Verbreitung von Agentensystemen wurde die Notwendigkeit für standardisierte Infrastrukturen sowie Kommunikationsmechanismen erkannt. Die *Foundation for Intelligent Physical Agents* (FIPA)[1] gehört seit 2005 als Standardisierungsgremium für *Agenten* und *Multiagentensysteme* der IEEE Computer Society an und hat sich zum Ziel gesetzt, den Einsatz von agentenbasierten Technologien voranzutreiben und durch gezielte Standardisierungsprozesse die Interoperabilität einzelner *Agenten* und *Agentenplattformen* zu stärken. Die folgenden Abschnitte beziehen sich in großen Teilen auf die von der FIPA beschriebenen und etablierten Architektur- und Technologiekonzepte. Ein fundamentaler Bestandteil von Agentensystemen ist die Verwaltung der einzelnen *Agenten*. Durch den Einsatz von *Agentenplattformen* werden grundlegende Funktionalitäten angeboten, wie beispielsweise Registrierungs-, Auffindungs- und Kommunikationsmechanismen für einzelne *Agenten* (FIPA, 2002a).

Definition 25 (*Agentenplattform*)

Agentenplattformen bieten die grundlegenden Funktionalitäten und Dienste zur Realisierung eines Multiagentensystems. Kernbestandteil einer Agentenplattform ist die Bereitstellung von Möglichkeiten zur Erstellung, Registrierung, Lokalisierung und Zerstörung einzelner Agenten sowie Kommunikationsmechanismen zum Informationsaustausch zwischen den Agenten.

Die von der FIPA vorgeschlagene Referenzarchitektur für Agentenplattformen umfasst neben den einzelnen *Agenten* die folgenden Basiskomponenten:

- **Agentenmanagement:** Das *Agent Management System* (AMS) ist ein notwendiger Bestandteil einer jeden *Agentenplattform*. Das *Agent Management System* ist für die grundlegende Funktionsweise einer *Agentenplattform* verantwortlich und kümmert sich hierbei um die Erstellung, Registrierung und die Verbindung mit anderen *Agentenplattformen*. Das *Agent Management System* ordnet jedem angemeldeten *Agenten* eine eindeutige Identifikation (AID) zu und verwaltet den Lebenszyklus jedes *Agenten*.

- **Agentenkommunikation** Das *Message Transport System* (MTS) ist ein zentraler Kommunikationsdienst jeder *Agentenplattform* und ermöglicht sowohl den Austausch von Nachrichten zwischen *Agenten* auf der selben *Agentenplattform* als auch zwischen *Agenten* auf unterschiedlichen Plattformen. Das *Message Transport System* ist somit die Grundlage für die im nächsten Abschnitt beschriebene Agentenkommunikationssprache.

- **Agentenregistratur:** Der *Directory Facilitator* (DF) ist eine optionale Agentenregistratur (*yellow pages*) innerhalb einer *Agentenplattform*.

[1] www.fipa.org [Letzter Zugriff: 31.01.2015]

Er verwaltet eine aktuelle und vollständige Liste aller in der Plattform angemeldeten *Agenten* und deren bereitgestellten Dienstbeschreibungen. Grundsätzlich sind unterschiedliche Implementierungen eines *Directory Facilitators* möglich (Lee et al., 2008), wodurch die Funktionalität auf den jeweiligen Anwendungsfall angepasst werden kann.

Abbildung 4.2 gibt einen Überblick über die zuvor beschriebenen Komponenten der FIPA-konformen Referenzarchitektur für Agentenplattformen.

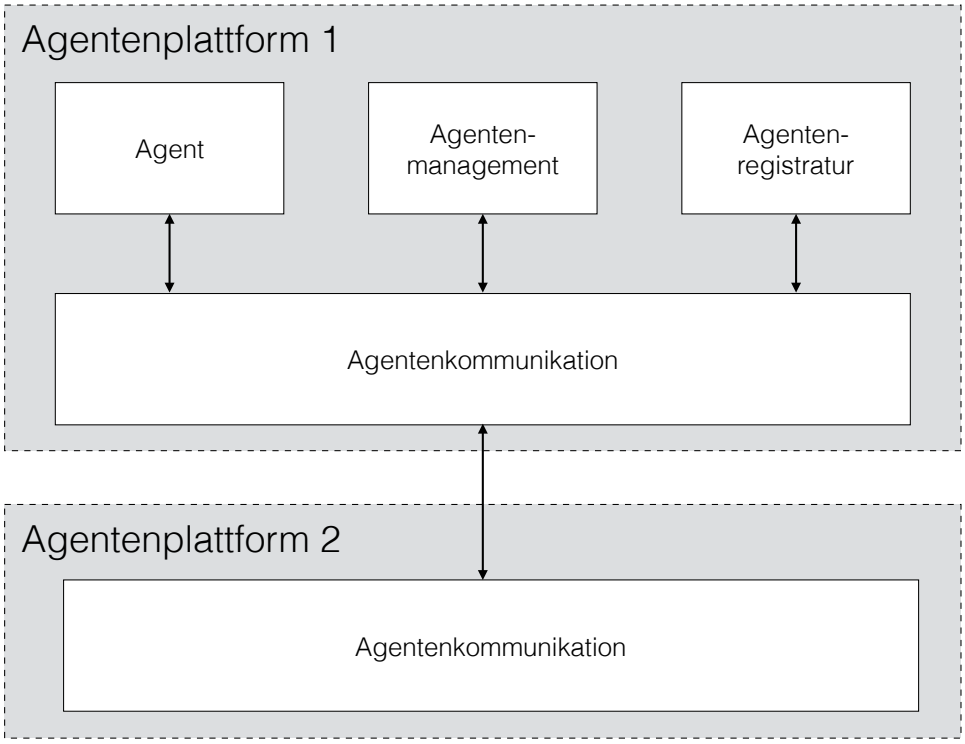


Abbildung 4.2: Überblick über die Komponentenstruktur der FIPA Referenzarchitektur.

4.4 Kommunikation

Die Kommunikation innerhalb eines *Multiagentensystems* spielt eine entscheidende Rolle zur Realisierung von dezentralen Lösungsstrategien. Im Gegensatz zu beispielsweise der objektorientierten Softwarentwicklung, bei welcher der objektübergreifende Informationsaustausch auf Methodenaufrufen beruht, basiert die *Agentenorientierte Softwareentwicklung* auf einem definierten Nachrichtenaustausch der *Agenten* untereinander. Die sogenannte *Agent Communication Language* (ACL) stellt eine Agentenkommunikationssprache dar und definiert hierbei sowohl Syntax, Semantik als auch Pragmatik des Inhalts (Lyre, 2002). Agentenkommunikationssprachen basieren im Allgemeinen auf der Sprechakttheorie, welche auf Austin (1962) und Searle (1969) zurückzuführen

ist und die Pragmatik des Nachrichteninhalts beschreibt. Beide untergliedern hierbei einen Sprech- oder Kommunikationsakt in mehrere Teilakte. In Bezug auf den Einsatz innerhalb von Agentenkommunikationssprachen spielen die folgenden drei Akte eine besondere Rolle:

- **Illokutionärer Akt:** Der *Illokutionäre Akt* ist ein zentraler Aspekt eines Sprechaktes und beschreibt den Zweck beziehungsweise die übergeordnete Absicht des Senders, wie zum Beispiel die Formulierung einer Frage, Bitte, Warnung oder Empfehlung.

- **Perlokutionärer Akt:** Der *Perlokutionäre Akt* beschreibt den Effekt, der beim Empfänger über den *Illokutionären Akt* hinaus ausgelöst werden soll. Im Vordergrund steht also das Erzielen einer Wirkung beim Empfänger, wie zum Beispiel überzeugen, umstimmen oder in dem eigenen Wissen bestärken.

- **Propositionaler Akt:** Der *Propositionale Akt* repräsentiert den eigentlichen Inhalt der Aussage und setzt sich nach Searle (1969) wiederum aus zwei Teilakten zusammen. Der *Referenzakt* stellt beim Sender einen Bezug zu Objekten innerhalb der Umgebung dar. Mittels des *Prädikatakts* werden diesem Objekt eine oder mehrere Eigenschaften zugeordnet.

Darauf aufbauend ordnet Searle (1969) die Sprechakte allgemein in fünf übergeordnete Klassen ein:

- **Repräsentiva:** Der Sender informiert den Empfänger über den Wahrheitsgehalt einer Proposition.

- **Direktiva:** Der Sender versucht, den Empfänger zu einer Aktion oder Handlung zu veranlassen.

- **Kommissiva:** Der Sender verpflichtet sich gegenüber dem Empfänger zu einer Aktion oder Handlung.

- **Expressiva:** Der Sender teilt dem Empfänger den eigenen emotionalen oder psychologischen Zustand mit.

- **Deklarativa:** Der Sender bewirkt durch die Äußerung gegenüber dem Empfänger eine institutionelle Zustandsveränderung.

Als eine der ersten Agentenkommunikationssprachen wurde Mitte der 90er Jahre die *Knowledge Query and Manipulation Language* (KQML) (Finin et al., 1994) entwickelt. KQML stellt sowohl ein Nachrichtenformat als auch eine Protokollspezifikation zum Wissensaustausch dar und erlaubt die Definition von Sprechakten sowie zum Beispiel die Formalisierung des Nachrichteninhalts in Prädikatenlogik erster Stufe. Die KQLM Spezifikation umfasst insgesamt 36 verschiedene Sprechakte. Der nächste Abschnitt betrachtet die weit verbreitete Kommunikationssprache FIPA-ACL als Weiterentwicklung von KQML.

4.4.1 FIPA Agent Communication Language (ACL)

Poslad (2007) beschreibt mit FIPA-ACL ein Modell für die plattformübergreifende Agentenkommunikation. FIPA-ACL setzt sich aus mehreren Protokollen zusammen: Interaktionsprozesse, kommunikative Akte, syntaktische und semantische Inhaltsbeschreibungen. Die FIPA Spezifikation (FIPA, 2002b) definiert die Menge an möglichen Parametern für eine FIPA-ACL-Nachricht. Der einzige für jede Nachricht verpflichtende Parameter ist die Beschreibung des zugrunde liegenden kommunikativen Akts (*performative*). Alle anderen Parameter werden als optional angesehen, wobei in der Regel davon ausgegangen wird, dass zusätzlich Sender, Empfänger und Inhalt angegeben werden (Bellifemine et al., 2007). Eine vollständige Übersicht über die möglichen Parameter einer FIPA-ACL-Nachricht und deren Bedeutung ist im Anhang A durch die Tabelle A.1 gegeben. Eine FIPA-ACL-Nachricht zur Anfrage einer Hotelreservierung könnte zum Beispiel wie folgt aussehen (Bellifemine et al., 2007):

```
initialize-state();
(request
    :sender (agent-identifier :name alice@mydomain.com)
    :receiver (agent-identifier :name bob@yourdomain.com)
    :ontology travel-assistant
    :language FIPA-SL
    :protocol fipa-request
    :content
        ""((action
            (agent-identifier :name bob@yourdomain.com)
            (book-hotel :arrival 15/10/2006 :departure 05/07/2002
    ... )
        ))""
)
```

Listing 4.1: Beispiel FIPA-ACL-Nachricht zur Hotelreservierung (nach Bellifemine et al. (2007)).

Ein wesentlicher Unterschied der FIPA-ACL im Vergleich zu KQML ist ihre eindeutigere Semantik, welche durch eine geringere Anzahl und eine strengere Definition der zugrunde liegenden kommunikativen Akte erreicht wird (Shehory und Sturm, 2014). Aufbauend auf der im Abschnitt 4.4 vorgestellten Sprechakttheorie wird in FIPA (2002c) die Menge der in FIPA-ACL relevanten kommunikativen Akte definiert und in fünf unterschiedliche Kategorien untergliedert: (1) Übermittlung von Informationen, (2) Anforderung von Informationen, (3) Verhandlungen, (4) Ausführung von Aktionen und (5) Fehlerbehandlung. Ein vollständiger Überblick über die verfügbaren kommunikativen Akte innerhalb einer FIPA-ACL-Nachricht ist im Anhang A durch die Tabelle A.2 gegeben.

4.4.2 Ontologien

Das Ziel von FIPA-ACL ist die Realisierung einer einheitlichen Kommunikationsinfrastruktur zwischen mindestens zwei beteiligten *Agenten*, welche miteinander interagieren und Wissen austauschen wollen. Neben den zuvor erwähnten kommunikativen Akten, welche die Pragmatik der Konversation definieren, bietet FIPA-ACL die Möglichkeit den Nachrichteninhalt semantisch zu modellieren. Durch die Verwendung von Ontologien innerhalb einer FIPA-ACL-Nachricht wird somit sichergestellt, dass die an der Konversation beteiligten *Agenten* die ausgetauschten Nachrichten auf die gleiche Art und Weise interpretieren und verstehen.

Definition 26 (*Ontologie*)

Eine Ontologie definiert das verfügbare Vokabular innerhalb eines Themenbereichs, indem die grundlegenden Begrifflichkeiten und Beziehungen sowie die Regeln zur Kombination und Erweiterung des Vokabulars spezifiziert werden (Gomez-Pérez et al., 2007).

Die FIPA Spezifikation erlaubt grundsätzlich das Einbinden von unterschiedlichen Ontologien, die sowohl implizit im Quellcode als auch zum Beispiel deklarativ in der Cloud definiert werden können. Die in FIPA (2002d) beschriebene *fipa-device* Ontologie kann zum Beispiel von *Agenten* genutzt werden, um Informationen über hardware- oder softwarespezifische Geräteprofile auszutauschen. Innerhalb einer FIPA-ACL-Nachricht kann eine vorgegebene *Ontologie* durch den Protokollparameter *:ontology* eingebunden werden (siehe Listing 4.1). Kapitel 8 gibt einen detaillierten Überblick über das in der vorliegenden Arbeit entwickelte semantische Modell einer *Intelligenten Umgebung*, was als Grundlage für die Kommunikation einzelner *Smart Service Agenten* dient. Ein Überblick über verwandte Arbeiten zur semantischen Modellierung von *Intelligenten Umgebungen* ist in Kapitel 5.5 zu finden.

4.5 Kooperation und Koordination

Neben der Kommunikation und dem Informationsaustausch stehen bei *Multiagentensystemen* besonders die Aspekte Kooperation und Koordination mehrerer *Agenten* im Vordergrund. Die Besonderheit ist hierbei, dass jeder einzelne *Agent* eigene Ziele verfolgen kann. Innerhalb eines *Multiagentensystems* müssen die *Agenten* jedoch effektiv zusammenarbeiten um ein übergeordnetes Problem gemeinsam lösen zu können.

Der Begriff des *Cooperative Distributed Problem Solving* (CDPS) wurde durch die Arbeiten von Durfee et al. (1989) eingeführt. CDPS untersucht, inwieweit verteilte und lose gekoppelte *Agenten* zusammenarbeiten können um Problemstellungen zu lösen, die außerhalb der Möglichkeiten eines einzelnen *Agenten* liegen. Kooperation ist notwendig, da ein einzelner *Agent* nicht über das vollständige Wissen verfügt um alle Ziele zu erfüllen. In Bezug auf *Multi-*

agentensysteme können grundsätzlich die vier folgenden Ergebnistypen einer Kooperation unterschieden werden:

- **Symmetrische Kooperation:** Beide *Agenten* erreichen durch die Kooperation Vorteile beziehungsweise eine Verbesserung ihrer Verarbeitung.
- **Symmetrischer Kompromiss:** Für beide *Agenten* ergibt sich eine Verschlechterung, wobei aus Sicht des Gesamtsystems eine Verbesserung eingetreten ist.
- **Nicht-symmetrische Kooperation bzw. Kompromiss:** Einer der beiden *Agenten* erreicht einen Vorteil, der andere einen Nachteil.
- **Konflikt:** Es ist keine Einigung zwischen den *Agenten* möglich, was zu einem erfolglosen Abbruch oder einem Neustart der Verhandlungen führt.

Wooldridge (2009) charakterisiert zwei Kriterien zur Bewertung des *Cooperative Distributed Problem Solving* und somit des Kooperationserfolgs von *Multiagentensystemen*. Die *Kohärenz* bewertet allgemein, wie gut das gesamte *Multiagentensystem* als Einheit operiert, um eine übergeordnete Problemstellung zu lösen. Hierbei spielen Faktoren wie Qualität der Lösung, Effizienz der Ressourcennutzung oder Resilienz in Bezug auf fehlerhaften oder unsicheren Daten eine Rolle. Das Kriterium *Koordination* bewertet die Effizienz der zur Problemlösung aufgebrachten Koordinationsmechanismen. Stehen einzelne *Agenten* häufig in Konflikt zueinander, kann dies ein Indikator für schlechte Koordinationsmechanismen sein, da sehr viel Aufwand zur Synchronisation und zur Konfliktlösung notwendig ist. Wooldridge (2009) hebt weiterhin vier wesentliche Fragestellungen hervor, welche bei der Entwicklung von kooperativen Agentensystemen berücksichtigt werden müssen:

- Wie kann ein übergeordnetes Problem in Teilprobleme zergliedert werden und somit auf mehrere *Agenten* verteilt werden?
- Wie kann die Lösung des Gesamtproblems effektiv aus den Lösungen der Teilprobleme synthetisiert werden?
- Wie kann der Problemlösungsprozess insgesamt optimiert werden, sodass die Kohärenz des Gesamtsystems maximiert wird?
- Welche Koordinationsmechanismen können eingesetzt werden, um unnötige Interaktionen zu vermeiden und die Effizienz des Gesamtsystems zu maximieren?

4.6 Agentenorientierte Softwareentwicklungsplattformen

Sturm und Shehory (2014) geben anhand mehrerer Beispiele einen Überblick über existierende Implementierungen und Entwicklungswerkzeuge auf dem Gebiet der *Agentenorientierten Softwareentwicklung*. Grundsätzlich unterscheiden

sie hierbei zwischen Programmiersprachen, Entwicklungsumgebungen (IDEs) und *Agentenplattformen*. Die Aufgabe von agentenorientierten Programmiersprachen ist es, die Hauptmerkmale von *Multiagentensystemen* realisieren zu können (Bellifemine et al., 2007). Als Mindestanforderung müssen die Sprachen das theoretische Konzept des *Softwareagenten* praktisch umsetzen können. Darüber hinaus können oftmals auch spezifische Eigenschaften, wie die Modellierung von kognitiven BDI Strukturen, in der Sprache integriert sein.

Als Beispiele für agentenorientierte Programmiersprachen sind, ohne Anspruch auf Vollständigkeit zu erheben, FLUX (Thielscher, 2005), JACK™ Agent Language (JAL) (Winikoff, 2005), 3APL (Dastani et al., 2005) oder AgentSpeak und der dazugehörige Interpreter Jason (Bordini et al., 2007) zu nennen. Einige dieser Sprachen, wie zum Beispiel JAL und Jason, verfügen über proprietäre Entwicklungsumgebungen oder bauen auf dedizierten *Agentenplattformen* auf. Wie bereits in Abschnitt 4.3 beschrieben, stellen *Agentenplattformen* sowohl grundlegende Entwicklungsfunktionalitäten als auch wichtige Laufzeitoperationen zur Verfügung. Durch die Berücksichtigung der FIPA Spezifikationen wird die Interoperabilität von *Agentenplattformen* sowie die Austauschbarkeit von *Agenten* gefördert.

Sturm und Shehory (2014) definieren weiterhin eine Menge von Anforderungskriterien, welche zur Realisierung einer agentenorientierten Softwareentwicklungsplattform berücksichtigt werden und anhand derer unterschiedliche Implementierungen verglichen werden können:

- **Laufzeitumgebung:** Die Laufzeitumgebung stellt einen zentralen Mechanismus zur Ausführung und zur Kontrolle von Operationen innerhalb eines *Multiagentensystems* dar.
- **Kommunikationssprachen:** Kommunikationssprachen, wie beispielsweise FIPA-ACL, stellen einen grundlegenden Kommunikationsmechanismus zum Austausch von Informationen und Wissen zwischen einzelnen Agenten dar.
- **Mobilität:** Agentenorientierte Softwareentwicklungsplattformen sollen den mobilen Einsatz von *Multiagentensystemen* in Bezug auf das zugrunde liegende Architekturdesign und deren Ausführung innerhalb unterschiedlicher Umgebungen erleichtern.
- **Sicherheit:** Die Sicherheit und der Datenschutz innerhalb eines *Multiagentensystems* sollten jederzeit berücksichtigt und gewährleistet werden.
- **Ressourcenbeschreibung und -auffindbarkeit:** Innerhalb eines *Multiagentensystems* sollte die Möglichkeit zur Beschreibung und Auffindung von Services und verwendeten Ressourcen gegeben sein.
- **Simulations- und Testumgebung:** Durch die Bereitstellung von Simulations- und Testwerkzeugen können agentenbasierte Anwendungen und die Verhaltensweisen einzelner Agenten vor dem tatsächlichen Einsatz im Wirkbetrieb ausführlich getestet und simuliert werden.

- **Kontrolle und Überwachung:** Durch die Sammlung von Laufzeitinformationen können beispielsweise Performanzanalysen oder Fehlerauswertungen bereitgestellt werden und somit die Weiterentwicklung des Gesamtsystems unterstützt werden.

- **Offenheit:** Ein *Multiagentensystem* sollte dazu in der Lage sein, unterschiedliche *Agenten* in Bezug auf die zugrunde liegende Architektur oder die verwendeten Ressourcen zu integrieren.

- **Skalierbarkeit:** Ein *Multiagentensystem* sollte hinsichtlich der Ausführung und des Softwaredesigns die Skalierbarkeit des Gesamtsystems unterstützen.

- **Benutzerfreundlichkeit:** Agentenorientierte Softwareentwicklungsplattformen sollten eine leichte Anwendbarkeit unterstützen und somit den Benutzer sowohl während der Entwicklung als auch während der Laufzeit eines *Multiagentensystems* optimal unterstützen.

- **Weiterentwicklung:** Eine *Agentenplattform* sollte kontinuierlich weiterentwickelt und verbessert werden.

- **Verwendung:** Eine hohe Verbreitung und Verwendung einzelner *Agentenplattformen* lässt Rückschlüsse auf einen hohen Reifegrad und auf die möglichen Anwendungsfelder der Plattform zu.

- **Standardisierung:** Durch den Einsatz von Standards und standardisierten Technologien kann die Interoperabilität von *Multiagentensystemen* untereinander gesteigert werden.

- **Entwicklungsmethodologien:** Eine agentenorientierte Softwareentwicklungsplattformen sollte über eine klare Entwicklungsmethodologie bezüglich des Entwurfs und der Entwicklung von *Multiagentensystemen* verfügen.

- **Integrierte Entwicklungswerkzeuge:** Durch die Bereitstellung einer spezialisierten, integrierten Werkzeugunterstützung können die Entwickler während des gesamten Entwicklungszyklus von *Multiagentensystemen* optimal unterstützt werden.

- **Templates:** Durch den Einsatz von vorgefertigten Code-Templates und Entwurfsmustern soll die Entwicklung spezifischer agentenbasierter Anwendungen erleichtert werden.

In den folgenden Abschnitten werden drei unterschiedliche agentenorientierte Softwareentwicklungsplattformen vorgestellt und im Anschluss anhand der eingeführten Kriterien miteinander verglichen.

4.6.1 Java Agent Development Framework (JADE)

Das *Java Agent Development Framework* (JADE) (Bellifemine et al., 2007) ist aktuell eins der am weitesten verbreiteten Plattformkonzepte zur Realisierung von agentenbasierten Anwendungen. JADE ist eine in Java realisierte *Middlewareplattform* und stellt eine Menge von Diensten und Werkzeugen zur Entwicklung von agentenbasierten Anwendungen bereit:

- **Monitoring Console:** Mit Hilfe der *Monitoring Console* kann die Ausführung einzelner Agenten kontrolliert und beeinflusst werden. Zudem können gezielte Nachrichten an Agenten generiert und versendet werden.
- **Sniffer:** Der *Sniffer* ermöglicht die Erfassung der plattforminternen Agentenkommunikation. Die gesammelten Nachrichten können entsprechend gefiltert und bei Bedarf gespeichert werden.
- **Debugger:** Durch den *Debugger* kann die Informationsverarbeitung innerhalb des Agenten überwacht und analysiert werden.

Das Verhalten von *Agenten* wird durch sogenannte *Behaviours* definiert. Eine *Behaviour* beschreibt eine eigenständige Aufgabenstellung eines einzelnen *Agenten* und kann jederzeit hinzugefügt werden. Die Ausführung mehrerer *Behaviours* kann grundsätzlich parallel zueinander ablaufen, wobei die Ausführung nicht präemptiv ist. In JADE werden drei Grundtypen von *Behaviours* unterschieden:

- **One-shot Behaviours:** *One-shot Behaviours* sind dazu ausgelegt, genau einmal abzulaufen um eine gesetzte Aufgabenstellung zu erfüllen.
- **Cyclic Behaviours:** *Cyclic Behaviours* können sich wiederholen und bei jedem Durchlauf die gleiche Operation ausführen, solange bis eine bestimmte Abbruchbedingung erfüllt ist.
- **Generic Behaviours:** *Generic Behaviours* erlauben eine flexible Ausführung von unterschiedlichen Operationen in Abhängigkeit von definierten Zuständen.

Ein wichtiger Bestandteil von JADE ist zudem die strenge Einhaltung der FIPA Standardisierung, besonders im Hinblick auf das unterliegende Architekturkonzept und die verwendeten Kommunikationsmechanismen. JADE ist als Open Source verfügbar und wird von der *Telecom Italia Laboratoires* (TILAB) unter der *GNU Lesser General Public License* (LGPL) bereitgestellt[2].

JADE wird darüber hinaus kontinuierlich weiterentwickelt. Eine wichtige Erweiterung stellt zum Beispiel die *Workflows and Agents Development Environment* (WADE) dar (Caire et al., 2008). Durch den Einsatz von WADE können Geschäftsprozesse und deren Prozessabläufe innerhalb von JADE abgebildet und entsprechend verarbeitet werden.

[2] http://jade.tilab.com

Ein weiterer Erweiterungsansatz von JADE basiert auf der Integration von *Kognitiven Agenten*. Braubach et al. (2005) unterteilen hierzu das Gebiet der *Multiagentensysteme* grob in zwei Kategorien. Einerseits betrachten sie *Multiagentensysteme* als *Middleware* mit dem Fokus auf infrastrukturelle Aspekte wie Verwaltungs- und Kommunikationsmechanismen. Hier stellt JADE eine robuste und weit verbreitete Plattform dar. Andererseits betrachten sie *Multiagentensysteme* aus Sicht der zentralen Problemlösungskomponente und stellen mit JADEx eine Erweiterung von JADE zur Verfügung mit dem Ziel, BDI basierte *Kognitive Agenten* zur realisieren.

4.6.2 JACK™

JACK™ ist eine vollentwickelte und kommerziell verfügbare agentenorientierte Softwareentwicklungsplattform. JACK™ basiert im Wesentlichen auf dem BDI Ansatz und stellt, ebenso wie JADE, entsprechende Laufzeitumgebungen und Entwicklungswerkzeuge zur Verfügung. Sturm und Shehory (2014) geben eine allgemeine Übersicht über das zugrunde liegende Metamodell der JACK™ Plattform. Hierbei werden einem *Agent* mehrere *Capabilities* zugewiesen, welche wiederum geeignete Umsetzungs- und Handlungspläne besitzen. Darüber hinaus können *Agenten* zu Teams zusammengeschlossen werden und somit übergeordnete Gruppenziele verfolgen. Winikoff (2005) gibt eine ausführliche Einführung in die JACK™ Plattform.

4.6.3 Java-based Intelligent Agent Componentware (JIAC)

Die *Java-based Intelligent Agent Componentware* (JIAC) ist eine aktuelle und frei verfügbare, agentenbasierte Softwareentwicklungsplattform[3], welche zum Ziel hat, die Entwicklung von komplexen und verteilten Anwendungen im industriellen Kontext zu erleichtern. JIAC unterstützt hierbei den Entwickler entlang des gesamten Softwareentwicklungsprozesses, von der Konzeption bis zum Einsatz im Wirkbetrieb, durch die Bereitstellung von spezialisierten Entwicklungs- und Laufzeitwerkzeugen. Ein Kernaspekt von JIAC ist die Integration von *Agenten* entsprechend einem serviceorientierten Ansatz (Hirsch et al., 2010). Eine *Agentenplattform* besteht in JIAC aus einer oder mehreren verteilten *AgentNodes* (Lutzenberger et al., 2013). Jede *AgentNode* stellt die notwendige Laufzeitumgebung für die einzelnen *Agenten* zur Verfügung. Ein *Agent* setzt sich wiederum aus mehreren Komponenten zusammen, wie zum Beispiel ein interner Ausführungszyklus, ein lokaler Speicher sowie ein Kommunikationsadapter. Das Verhalten und die Funktionsweise des *Agenten* wird durch sogenannte *AgentBeans* definiert, welche durch die folgenden Eigenschaften charakterisiert sind:

- Bereitstellung von Methoden zur Erfassung des Lebenszyklus des zugehörigen *Agenten*, zum Beispiel *initialized* oder *started*.

[3] http://www.jiac.de

- Bereitstellung einer sich in regelmäßigen Abständen wiederholenden *execute* Methode zur Ausführung von Aktionen des *Agenten*.

- Möglichkeit zur An- und Abmeldung von Beobachtern an den internen Speicher des *Agenten*, zum Beispiel zum Überwachen von eingehenden Nachrichten oder Zustandsveränderungen.

- Zentrale Bereitstellung von Servicemethoden, auf die entweder innerhalb des eigenen *Agenten* oder von anderen *Agenten* zugegriffen werden kann.

JIAC basiert darüber hinaus auf aktuellen Technologien, wie zum Beispiel Spring[4], OSGi[5] und Maven[6].

4.6.4 Fazit

Sturm und Shehory (2014)) haben unterschiedliche agentenorientierte Softwareentwicklungsplattformen untersucht und entsprechend der vorgestellten Bewertungskriterien miteinander verglichen. Tabelle 4.1 gibt eine Übersicht über die Auswertung des funktionalen Vergleichs der im Rahmen der vorliegenden Arbeit vorgestellten Ansätze.

	JADE	JACK	JIAC
Laufzeitumgebung	✓	✓	✓
Kommunikationssprache	✓	-	-
Mobilität	✓	-	✓
Sicherheit	✓	-	✓
Ressourcenbeschreibung und -auffindbarkeit	✓	-	✓
Simulations- und Testumgebung	-	-	-
Kontrolle und Überwachung	(✓)	-	✓
Offenheit	-	-	-
Skalierbarkeit	(✓)	✓	✓
Benutzerfreundlichkeit	✓	✓	✓
Weiterentwicklung	✓	✓	✓
Verwendung	✓	✓	-
Standardisierung	✓	-	-
Entwicklungsmethodologien	(✓)	✓	✓
Integrierte Entwicklungswerkzeuge	(✓)	✓	✓
Templates	✓	✓	✓

✓: Kriterium erfüllt (✓): Kriterium teilweise erfüllt -: Kriterium begrenzt erfüllt

Tabelle 4.1: Bewertung und funktionaler Vergleich ausgewählter *Agentenplattformen* (nach Sturm und Shehory (2014)).

[4]http://spring.io [Letzter Zugriff: 12.03.2015]
[5]http://www.osgi.org [Letzter Zugriff: 12.03.2015]
[6]http://maven.apache.org [Letzter Zugriff: 12.03.2015]

Darüber hinaus haben Müller und Fischer (2014) im Rahmen einer umfassenden Studie insgesamt 152 Anwendungen im Bereich von *Multiagentensystemen* analysiert und somit die praktische Anwendbarkeit und die Verbreitung der verfügbaren *Agentenplattformen* untersucht. Abbildung 4.3 gibt eine Übersicht über die Nutzung der angegebenen *Agentenplattformen* innerhalb ausgereifter und im operativen Geschäft bereits eingesetzter Softwareanwendungen.

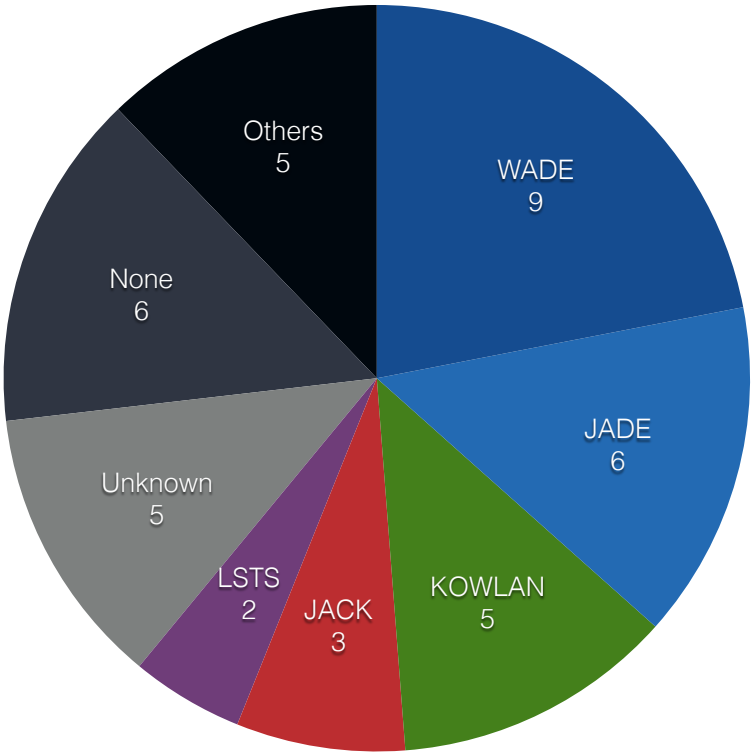


Abbildung 4.3: Übersicht über die Verwendung und die absolute Nutzungszahl von *Agentenplattformen* in existierenden und im Produktivbetrieb eingesetzten Softwareanwendungen (nach Müller und Fischer (2014)).

Zusammenfassend kann festgehalten werden, dass JADE und dessen Erweiterung WADE die am häufigsten eingesetzten *Agentenplattformen* darstellen. Angesichts des funktionalen Vergleichs ist zu erkennen, dass JADE zudem auch die meisten der gestellten Anforderungskriterien an eine agentenorientierte Softwareentwicklungsplattform erfüllt.

4.7 Zusammenfassung

In diesem Kapitel wurden grundlegende Begriffe und Konzepte in Bezug auf *Multiagentensysteme* eingeführt. Dazu wurde ein allgemeiner Agentenbegriff eingeführt und durch eine formale Definition untermauert. Aufbauend auf der durch die FIPA standardisierten Spezifikationen wurden die Plattformarchitektur sowie Kommunikations-, Kooperations- und Koordinationsmechanismen

erläutert. Im Anschluss wurden existierende Implementierungen und Plattformen aufgezeigt und insbesondere mit JADE die Grundlage für die in der vorliegenden Arbeit realisierten *Agentenbasierten Smart Service Plattform* (ASaP) gelegt. Das Einsatzgebiet von *Multiagentensystemen* ist sehr breit gefächert und umfasst sowohl vergleichsweise kleine Anwendungen, wie zum Beispiel Emailfilter, als auch große, komplexe und fehlerkritische Systeme, wie beispielsweise Flugverkehrskontrolle (Jennings und Wooldridge, 1998). Neben industriellen Applikationen, wie Prozesskontrolle, Systemdiagnose, Logistik oder Netzwerkverwaltung, stellen Bereiche wie intelligentes Informationsmanagement (Decker und Sycara, 1997; Klusch, 2001) und semantische Service Koordination (Klusch, 2008; Grütter, 2006) wichtige Anwendungsfälle von agentenbasierten Technologien dar. Auch die Themenfelder Telekommunikation, Robotik oder Gesundheitswesen (Lanzola und Boley, 2002; Hudson und Cohen, 2002) profitieren von der Vielseitigkeit von *Multiagentensystemen*. Murukannaiah und Singh (2014) stellen darüber hinaus mit Xipho eine Entwicklungsmethodologie zur Realisierung von kontextsensitiven persönlichen Agenten vor. Aufgrund der Relevanz für die vorliegende Arbeit werden in Kapitel 5.4 verwandte Arbeiten bezüglich des Einsatzes von *Multiagentensystemen* im Kontext von *Intelligenten Umgebungen* ausführlich behandelt.

Das Geheimnis des Erfolges ist, den Standpunkt des anderen zu verstehen.

Henry Ford

5 Verwandte Arbeiten

5.1 Einleitung

Das in der vorliegenden Arbeit adressierte Forschungsthema befindet sich, wie in Kapitel 1 gezeigt, in der Schnittmenge von unterschiedlichen interdisziplinären Forschungsgebieten. Jeder dieser Bereiche weist unterschiedliche Schwerpunkte und Herangehensweisen zur Realisierung von *Intelligenten Umgebungen* auf. Ein Überblick im Rahmen dieser Arbeit kann daher nicht erschöpfend und vollständig sein. In diesem Kapitel werden aus diesem Grund ausgewählte verwandte Arbeiten und Ergebnisse anhand der zugrunde liegenden Forschungsgebiete vorgestellt. Der Schwerpunkt liegt hierbei auf der Präsentation ausgewählter Forschungsansätze und der Analyse der jeweiligen Stärken und Schwächen in Bezug auf die Zielsetzungen dieser Arbeit. In Abschnitt 5.2 werden zunächst Kriterien eingeführt, auf deren Basis die verwandten Arbeiten analysiert und miteinander verglichen werden. Abschnitt 5.3 gibt einen Überblick über existierende *Middlewareplattformen*, die im Bereich von *Smart Home* und AAL Technologien bereits zum Einsatz kommen. In Abschnitt 5.4 wird der Einsatz von Agentensystemen zur Realisierung von *Smart Services* im Kontext von *Intelligenten Umgebungen* näher beleuchtet. In Abschnitt 5.5 werden, unabhängig von den beiden Abschnitten zuvor, unterschiedliche Arbeiten aus dem Gebiet der semantischen Modellierung von *Intelligenten Umgebungen* und der darin enthaltenen Objekte vorgestellt. Abschnitt 5.6 beinhaltet eine abschließende Diskussion der beschriebenen Ansätze und begründet die weitere Vorgehensweise innerhalb der vorliegenden Arbeit.

5.2 Anforderungen und Bewertungskriterien

Um die in Kapitel 1 definierten wissenschaftlichen und praktischen Zielstellungen dieser Arbeit zu erreichen, werden im Folgenden die wesentlichen Anforderungen an das zu entwickelnde Plattformkonzept beschrieben. Die Anforderungen sind in die Bereiche Plattformarchitektur, Benutzerinteraktion, Werkzeugunterstützung und Semantik unterteilt und dienen als Leitfaden für die Konzeption und die Entwicklung der in dieser Arbeit vorgeschlagenen Softwarearchitektur. Zudem lassen sich anhand der Anforderungen und Bewertungskriterien die verwandten Arbeiten in den jeweiligen Forschungsgebieten je nach Schwerpunkt und Anwendungsgebiet miteinander vergleichen.

5.2.1 Plattformarchitektur

- **Dynamische Dienstintegration (A_P1):** Die Möglichkeit der Integration von Mehrwertdiensten in Form von *Smart Services* stellt eine Grundvoraussetzung zur Realisierung von flexiblen und dynamischen Plattformkonzepten für *Intelligente Umgebungen* dar. Durch die Integration neuer Dienste kann die Plattform somit zur Laufzeit um weitere Funktionalitäten erweitert werden. Hierzu gehören unter anderem die Möglichkeiten, die *Smart Services* in einzelne Module zu bündeln, über eine definierte Infrastruktur zu verteilen und den Lebenszyklus der Module zu kontrollieren, aktualisieren und manipulieren.

- **Semantische Dienstinteroperabilität (A_P2):** Die Interoperabilität zwischen *Smart Services*, der zugrunde liegenden Umgebung und dem einzelnen Benutzer erfordert eine einheitliche semantische Modellierung und eine klar definierte plattforminterne Kommunikationsstruktur. Insbesondere die Modellierung und semantische Repräsentation aller relevanten Objekte in der Umgebung, der räumlichen und physischen Kontextinformation sowie der entsprechenden Aktivitäten und Interaktionen des Benutzers spielen hierbei ein zentrale Rolle. Zur Erzielung einer dienstübergreifenden Interoperabilität von *Smart Services* müssen die Beziehungen einzelner *Smart Services* zueinander durch einen klaren Kommunikationsmechanismus und durch die Modellierung ihrer semantischen Abhängigkeiten abgebildet werden.

- **Skalierbarkeit (A_P3):** *Smart Services* können unter Umständen einen hohen Ressourcenverbrauch aufweisen. So erfordern zum Beispiel Algorithmen zum Prognostizieren des Energieverbrauch eines Privathaushalts sowohl hohe Datenhaltungs- als auch Rechenkapazitäten. Durch den Einsatz von dezentralen Verarbeitungsmethodiken, wie zum Beispiel die Auslagerung von rechenintensiven Prozessen in cloudbasierte Infrastrukturen, soll diesen Anforderungen Rechnung getragen werden.

- **Standards (A_P4):** Der *Smart Home* Markt ist bereits jetzt ein heterogener und stark fragmentierter Markt mit einer Vielzahl von unterschiedlichen und oftmals inkompatiblen Lösungen. Zur Vermeidung weiterer Insellösungen ist es unbedingt notwendig, einen hohen Grad an Wiederverwendbarkeit von bereits verfügbaren Technologien anzustreben. Der Einsatz von Standards kann hier dazu beitragen, die Integration dieser Technologien zu erleichtern und somit eine erhöhte Wiederverwendbarkeit zu ermöglichen.

- **Open Source (A_P5):** Durch den Einsatz von Open Source Software werden unter anderem die relevanten Kommunikations- und Programmierschnittstellen der Plattform offengelegt, was die Verwendung und den initialen Einstieg für Entwickler und Endkunden erleichtert. Die Offenlegung des Quellcodes erhöht zudem das Vertrauen in die Software und die darin verwendeten Algorithmen. Außerdem ermöglichen Open

Source Systeme die innovativsten Softwarekonzepte und stellen somit einen wichtigen Innovationstreiber für Industrie und Wissenschaft dar.

5.2.2 Benutzerinteraktion

- **Marktplatzportal (A_B1):** Der Einsatz von digitalen Marktplätzen, wie beispielsweise *Apple App Store* oder *Google Play*, hat die allgemeine Art und Weise wie Kunden mit Softwareanwendungen umgehen, wie sie diese erwerben und im Alltag einsetzen können, entscheidend verändert(Jansen und Bloemendal, 2013). Digitale Marktplätze sind hierbei ein entscheidender Erfolgsfaktor beim Aufbau eines jeden softwarebasierten Ökosystems. Durch den Einsatz eines einheitlichen Marktplatzportals können Entwickler und Dienstanbieter ihre *Smart Services* anbieten und vertreiben sowie Endnutzer durch geeignete Empfehlungssysteme (Böhmer et al., 2013) dazu befähigt werden, gezielt kompatible und auf ihr *Smart Home* passende *Smart Services* zu finden und zu installieren.

- **Personalisierte Benutzerschnittstellen (A_B2):** In einer *Intelligenten Umgebung* spielt die Interaktion des Benutzers mit der Umgebung eine wichtige Rolle. Entscheidende Aspekte sind beispielsweise, dass der Benutzer jederzeit die volle Kontrolle über die ihn umgebende Technologie haben muss und der Umgebung direkte oder indirekte Rückmeldung geben kann, um somit die Qualität der zugrunde liegenden *Smart Services* kontinuierlich zu verbessern. Im Mittelpunkt steht hierbei eine intuitive und auf die persönlichen Bedürfnisse angepasste Interaktion für alle Benutzergruppen. Auch Menschen mit besonderen Bedürfnissen, wie ältere Menschen oder Menschen mit Einschränkungen, müssen hierbei Berücksichtigung finden. Eine Voraussetzung zur Realisierung stellen standardisierte Beschreibungs- und Modellierungssprachen für Benutzerschnittstellen dar.

5.2.3 Werkzeugunterstützung

- **Entwicklungswerkzeuge (A_W1):** Ein wichtiger Aspekt zur Erzielung einer Marktdurchdringung einer Softwareplattform ist das Bereitstellen einer großen Anzahl von Anwendungen und Erweiterungsmöglichkeiten. Um den Einstieg und die Verbreitung für Entwickler und Unternehmen zu erleichtern, müssen integrierte Entwicklungsumgebungen bereitgestellt werden, die den gesamten Entwicklungsprozesses einer Plattformkomponente unterstützen.

- **Management, Betrieb und Wartung (A_W2):** Durch die technologische Komplexität von *Smart Home* Lösungen können Fehler und Störungen nicht vollständig ausgeschlossen werden. Es ist daher notwendig, geeignete Management- und Wartungswerkzeuge zur Verfügung zu stellen, mit denen *Intelligente Umgebungen* durch den Endnutzer oder die entsprechende Dienstleister auf Fehlfunktionen überwacht werden können.

Darüber hinaus spielt die Bereitstellung von Unterstützungssytemen zur Inbetriebnahme und zur Installation neuer *Smart Services* eine wichtige Rolle.

5.2.4 Semantik

- **Modellierung von *Smart Spaces* (A_S1):** Die meisten Objekte innerhalb einer *Intelligenten Umgebung* besitzen feste räumliche Positionen, während sich Benutzer und mobile Objekte in der Regel frei im Raum bewegen können. Um kontextsensitive Anwendungen und Dienste zu realisieren, muss folglich auch das Wissen über diese räumlichen Beziehungen und Abhängigkeiten von Objekten und Benutzern innerhalb der Umgebungen entsprechend modelliert und repräsentiert werden.

- **Modellierung von *Smart Products* (A_S2):** Neben den räumlichen Beziehungen ist es notwendig auch die physischen Eigenschaften der vernetzten Objekte innerhalb einer *Intelligenten Umgebung* zu repräsentieren. Damit einzelne Produkte voneinander unterschieden werden können, ist besonders die Geräteklassifizierung in einer Vererbungshierarchie von Bedeutung.

- **Modellierung von *Smart Services* (A_S3):** Um kontextsensitive Assistenzsysteme zu realisieren, müssen die zugrunde liegenden digitalen Mehrwertdienste semantisch repräsentiert und ihre Abhängigkeiten zu der Umgebung definiert werden.

- **Modellierung von Zustandsveränderungen (A_S4):** Damit *Smart Services* auf Ereignisse innerhalb der Umgebung reagieren können, müssen die entsprechenden Zustände und Zustandsveränderungen innerhalb einer *Intelligenten Umgebung* modelliert werden.

- **Modellierung von Benutzerassistenz (A_S5):** Um dem Benutzer eine auf die jeweilige Situation passende Hilfestellung und interaktive Assistenz anbieten zu können, muss diese entsprechend semantisch repräsentiert werden. Hierzu gehören insbesondere auch die Modellierung von Management- und Installationsfunktionalitäten.

- **Modellierung von Benutzeraktivitäten (A_S6):** Die Grundlage vieler kontextbezogener Assistenzsysteme ist das Erkennen von menschlichen Handlungen innerhalb einer *Intelligenten Umgebung*. Zur Klassifizierung der menschlichen Verhaltensweisen, beispielsweise in Aktivitäten des täglichen Lebens (Reisberg et al., 2001), muss eine grundlegende semantische Repräsentation der Benutzeraktivitäten vorhanden sein.

- **Integration in Entwicklungsumgebung (A_S7):** Zur Erzielung des maximalen Nutzens einer semantischen Repräsentation müssen sich die entwickelten Modelle nahtlos in die verwendeten Entwicklungswerkzeuge integrieren lassen. Zum einen beinhaltet die Entwicklungsumgebung Methoden zum Erstellen und zum Bearbeiten der jeweiligen Modelle selbst.

Zum anderen sind hiermit aber auch Werkzeuge gemeint, die auf der Grundlage der semantischen Modelle eingesetzt werden, wie zum Beispiel Codegeneratoren zur Erstellung neuer *Smart Services*.

5.3 Middlewareplattformen für Intelligente Umgebungen

Wie bereits in Kapitel 2.3 beschrieben, ist eine *Middleware* ein Teil der *Software-definierten Plattform* (siehe Kapitel 3.3). Lackes und Siepermann (2014a) definieren den Begriff der *Middleware* hierbei als eine Softwareschicht, die Kommunikationsdienste für verteilte Anwendungen über Standardschnittstellen bereitstellt und damit eine Integration der Anwendungen und ihrer entsprechenden Daten ermöglicht. Die *Middleware* ermöglicht innerhalb der *Software-definierten Plattform* somit eine Virtualisierung der an die *Vernetzten physischen Plattformen* angeschlossenen *Sensoren* und *Aktuatoren*. Basierend auf dieser allgemeinen Definition gibt es eine Vielzahl von unterschiedlichen Middlewareansätzen und -lösungen (Fagerberg et al., 2010). Neben groß angelegten Forschungsprojekten wie UniversAAL (Hanke et al., 2011; Ram et al., 2013) existieren auch kleinere, kommerzielle Lösungen. Ein vollständiger Überblick über das Themenfeld kann im Rahmen dieser Arbeit nicht gegeben werden. Im Folgenden wird die Auswahl daher auf Forschungsansätze und kommerzielle Lösungen begrenzt, welche einen besonderen Bezug zu den Schwerpunkten dieser Arbeit aufweisen. Mit Apple HomeKit und Google Nest werden abschließend zwei Beispiele vorgestellt, wie Firmen mit großer Marktmacht versuchen, den *Smart Home* Markt entscheidend zu beeinflussen.

5.3.1 DOG - Domotic OSGi Gateway

Mit dem *Domotic OSGi Gateway* (DOG) stellen Bonino et al. (2008) eine Middlewareplattform im Kontext von Heimautomationsszenarien vor. Der Schwerpunkt des Projekts liegt auf der Erweiterung existierender Heimautomationssysteme durch das Hinzufügen geeigneter Hardware- und Softwarekomponenten mit dem Ziel ein hohes Maß an Interoperabilität und intelligenter Unterstützung zu erreichen. Die zentrale Komponente des Ansatzes ist das sogenannte DOG. Es handelt sich um einen eingebetteten Computer, welcher über geeignete Rechenkapazitäten sowie Konnektivitätsmechanismen verfügt, um unterschiedliche Heimnetze zu integrieren und zu koordinieren. Die zugrunde liegende Softwarearchitektur basiert auf einem modular aufgebauten Schichtenmodell. Ein einzelne Softwareschicht wird als sogenannter Ring bezeichnet. Jeder Ring umfasst mehrere Komponenten mit klar definierten Aufgaben- und Verantwortlichkeitsbereichen (siehe Abbildung 5.1). Technologisch betrachtet basiert DOG auf OSGi und erlaubt somit das dynamische Einbinden neuer Komponenten zur Laufzeit des Systems.

- **Ring 0:** Die unterste Ebene enthält die notwendigen Basismodule zur Verknüpfung der OSGi Infrastruktur mit den übergeordneten DOG Funk-

tionalitäten. Die *DOG Library* dient als Bibliotheksverzeichnis für andere DOG Komponenten. Der *Platform Manager* koordiniert und verwaltet einzelne Komponenten und deren Lebenszyklus. Weiterhin steuert er den Startmechanismus des Gesamtsystems und ist für die korrekte Ausnahmebehandlung bei Systemfehlern zuständig. Durch die *Configuration Registry* können einzelne Module sowie das Gesamtsystem schließlich konfiguriert werden.

- **Ring 1:** Diese Ebene ist für die Bereitstellung und die Verwaltung der Schnittstellen zu den verfügbaren Heimnetzwerken innerhalb der Umgebung verantwortlich. Jeder Netzwerktreiber agiert als eigenständiger Adapter und ist für die Übersetzung der abstrakten Funktionsbeschreibungen der verfügbaren Geräte in das konkrete Netzwerkprotokoll der zugrunde liegenden Hardware zuständig. Nach Angaben der Entwickler[1] sind aktuell die folgenden Netzwerktreiber integriert: KNX[2], ZigBee[3], Z-Wave[4], My Home[5], Modbus[6], ECHELON[7] und Philips hue[8]. Darüber hinaus wird ein generisches Simulationsmodul bereitgestellt, durch das Ereignisse und Zustände innerhalb der Umgebung simuliert und nachgestellt werden können.

- **Ring 2:** Auf dieser Ebene sind die höherwertigen Dienste der DOG Architektur angesiedelt. Der *Message Dispatcher* handelt als zentraler Router für eingehende und ausgehende Nachrichten. Er ist für die Kommunikation mit den jeweiligen Netzwerktreibern und für den Austausch mit anderen Komponenten innerhalb des Systems verantwortlich. Der *Message Dispatcher* verfügt zu diesem Zweck über eine interne Routing Tabelle, in der alle verfügbaren Komponenten mit einem eindeutigen Bezeichner versehen werden. Bevor die Nachrichten den *Message Dispatcher* erreichen, werden diese durch den *Executor* auf syntaktische und semantische Korrektheit überprüft. Die *Status* Komponente speichert alle Zustandsveränderungen der angeschlossenen Geräte und verfügt somit jederzeit über ein aktuelles Abbild des Umgebungskontexts. Grundvoraussetzung für die Realisierung von Anwendungen auf der Basis von DOG ist die semantische Modellierung der Umgebung aller darin enthaltenen Geräte. Das *House Model* enthält eine konkrete Ausprägung der DogOnt Ontologie, welche in Abschnitt 5.5.4 ausführlich beschrieben wird.

- **Ring 3:** Auf der obersten Ebene werden schließlich Schnittstellenfunktionalitäten angeboten, mit deren Hilfe externe Anwendungen Zugriff auf

[1] http://dog-gateway.github.io [Letzter Zugriff: 26.01.2015]
[2] http://www.knx.de [Letzter Zugriff: 26.01.2015]
[3] http://zigbee.org [Letzter Zugriff: 26.01.2015]
[4] http://www.z-wave.com [Letzter Zugriff: 26.01.2015]
[5] http://www.bticino.it/domotica-myhomeweb [Letzter Zugriff: 26.01.2015]
[6] http://www.modbus.org [Letzter Zugriff: 26.01.2015]
[7] http://www.echelon.com/applications [Letzter Zugriff: 26.01.2015]
[8] http://www.hue.philips.de [Letzter Zugriff: 26.01.2015]

die durch DOG bereitgestellten Funktionalitäten erhalten. Das *API* Modul stellt anderen OSGi Bundles eine Programmierschnittstelle zur Verfügung, mit der diese den Zustand der zugrunde liegenden Umgebung ermitteln oder beeinflussen können. Durch die *XML-RPC* Schnittstellen wird ein Service Endpoint realisiert, durch den auch externe Anwendungen Zugriff auf das DOG System und dessen Funktionalitäten erhalten.

Abbildung 5.1 gibt eine abschließende Übersicht über die vorgestellten Komponenten der DOG Architektur.

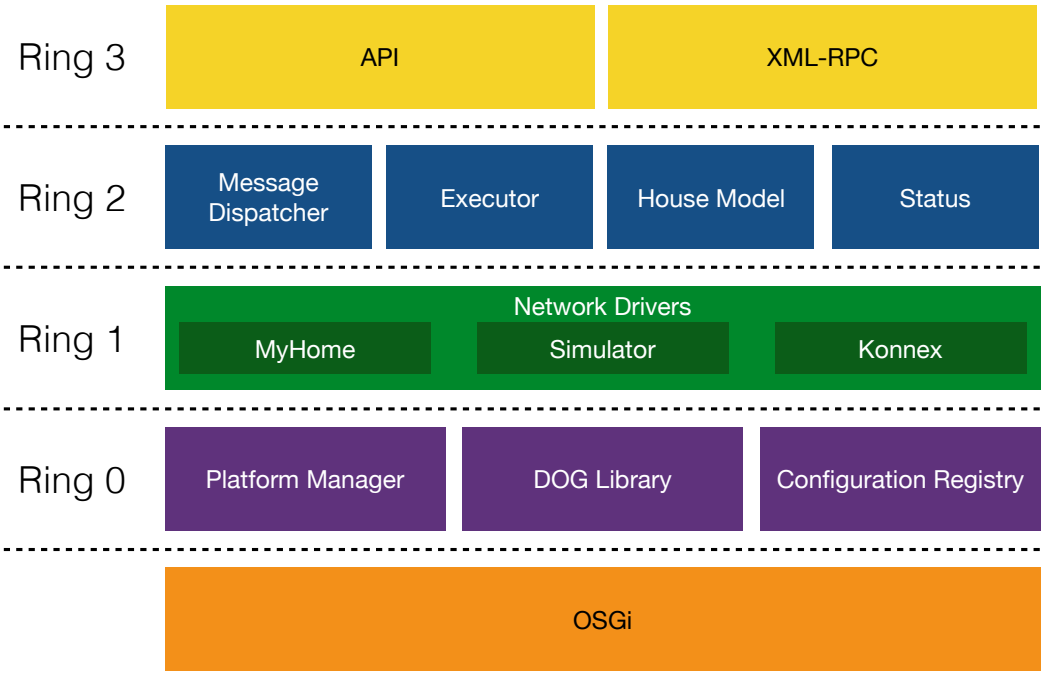


Abbildung 5.1: Übersicht über die Schichtenarchitektur des *Domotic OSGi Gateways* (nach Bonino et al. (2008)).

DOG ist komplett als Quellcode unter der Apache v2.0 License[9] veröffentlicht und steht zusammen mit entsprechenden Dokumentationen und API Beschreibungen zum Herunterladen bereit.

5.3.2 EBS – Event Broadcasting Service

Kahl und Bürckert (2012) stellen mit dem *Event Broadcasting Service* (EBS) eine Kommunikationsinfrastruktur für den Einsatz gemäß eines erweiterten Dual Reality (DR++) Konzepts (Kahl, 2014) vor. Die EBS Architektur ist hierbei modular aufgebaut und ermöglicht eine Konfiguration und Anpassung des Gesamtsystems auf das jeweilige Anwendungsgebiet. EBS erlaubt eine

[9] http://www.apache.org/licenses/LICENSE-2.0 [Letzter Zugriff: 26.01.2015]

flexible Verarbeitungslogik und stellt unterschiedliche Architekturmuster zur Verfügung, zum Beispiel Blackboard, Publish/Subscribe oder *Complex Event Processing* (CPE). Eine Übersicht über die Gesamtarchitektur von EBS ist in Abbildung 5.2 zu finden.

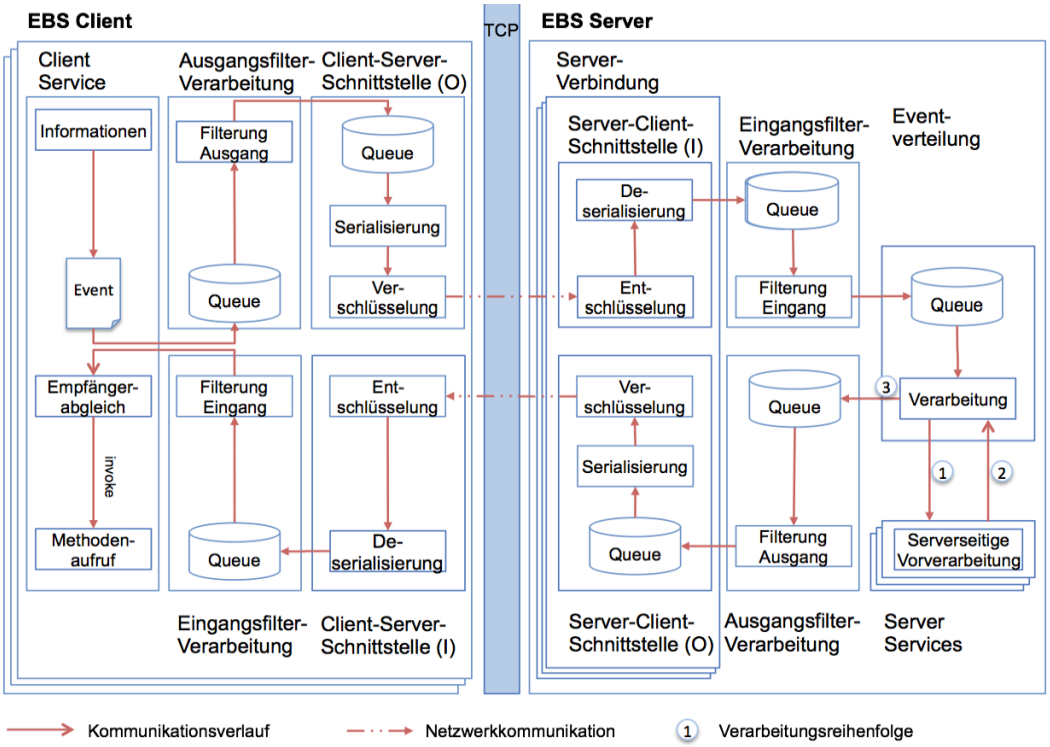


Abbildung 5.2: Architektur und Informationsfluss des *Event Broadcasting Servic* (EBS) (aus Kahl (2014)).

Die Informationsverarbeitung innerhalb von EBS basiert auf der Übermittlung von Ereignissen, die Zustandsänderungen in der Umgebung darstellen. Kahl (2014) führt die grundlegenden Attribute für die in EBS zum Einsatz kommenden Eventstrukturen ein. In Tabelle 5.1 wird diese Grundstruktur kurz dargestellt.

Parameter	**Wert**	**Beschreibung**
Typ	*String*	Spezifiziert den Typ des Events
Zeitpunkt	*Zeitstempel*	Spezifiziert den Zeitpunkt des Auftretens eines Events
Gültigkeit	*Zeitstempel/ms*	Spezifiziert die Gültigkeitsdauer eines Events
Quelle	*String*	Spezifiziert den Sender
Ziel	*String[]*	Spezifiziert die Empfänger
Privat	*true/false*	Spezifiziert, ob das Event als *privat* markiert wird
Verschlüsselung	*true/false*	Spezifiziert, ob das Event verschlüsselt werden soll

Session	*String*	Spezifiziert eine Session-ID zur Erkennung von Systemneustarts
Priorität	*hoch/normal/gering*	Spezifiziert die Priorität eines Events
Konfidenz	*[0..1]*	Spezifiziert die Konfidenz eines Events
Simulation	*true/false*	Spezifiziert, ob das Event auf realen oder simulierten Daten basiert

Tabelle 5.1: Auflistung und Kurzbeschreibung der Attribute der in EBS verwendeten Eventstruktur (nach Kahl (2014)).

Aufbauend auf der einheitlichen Eventstruktur können in EBS verschiedene Filter- und Vorverarbeitungsmechanismen definiert werden. Der Einsatz von Eventfiltern dient primär dazu, die Performanz des Gesamtsystems durch das Aussortieren bestimmter, nicht mehr gültiger oder nicht relevanter Events zu verbessern. Im Gegensatz zu dem Filtermechanismus werden durch die Vorverarbeitungsmodule nicht zwingend Events aussortiert oder blockiert. Vielmehr können Events verändert, kombiniert oder in Teilevents aufgeteilt werden. Tabelle 5.2 fasst die Filter und Vorverarbeitungsmodule zusammen.

Name	**Typ**	**Kurzbeschreibung**
Typfilter	Filter	Filterung nach dem Eventtyp
Inhaltsfilter	Filter	Filterung in Abhängigkeit von dem Inhalt des Events
Fensterfilter	Filter	Filterung nach dem Zeitpunkt oder der Anzahl der Events
Akkumulation	Vorverarbeitung	Analyse mehrerer Events des gleichen Typs
Transformation	Vorverarbeitung	Veränderung von Inhalten des Events
Kombination	Vorverarbeitung	Verschmelzung mehrerer Events
Aufsplittung	Vorverarbeitung	Aufteilung eines übergeordneten Events in mehrere Teilevents
Monitoring	Vorverarbeitung	Überwachung von Clientaktivitäten und Benachrichtigung bei Problemen

Tabelle 5.2: Kurzdarstellung der im EBS integrierten Filter und Vorverarbeitungsmodule (nach Kahl (2014)).

Die Filter und Vorverarbeitungsmodule können darüber hinaus auch zu komplexeren Filterketten zusammengeschlossen werden. Kahl (2014) beschreibt Filterketten als gerichtete azyklische Graphen mit jeweils genau einem Startknoten und genau einem Endknoten. Jeder Knoten muss erreichbar sein und

einen Pfad zum Zielknoten aufweisen. Zur Entwicklung der beschriebenen Filterketten wird im Rahmen von EBS ein graphisches Modellierungswerkzeug bereitgestellt. Der resultierende Graph wird in XML gespeichert und kann daraufhin in der EBS Architektur verwendet werden. Aufbauend auf einer Client-Server-Struktur ermöglicht EBS zudem eine Reduzierung der Komplexität des Systems, so dass damit sowohl eine einfache Einrichtung als auch eine Fehlererkennung realisiert werden kann. Der Kommunikationsfluss zwischen den einzelnen Systemen kann zudem durch eine zentrale Stelle überwacht und gegebenenfalls reguliert werden.

5.3.3 OpenHAB

Der *Open Home Automation Bus* (openHAB) ist eine vollständig in Java und auf Basis von OSGi realisierte Softwareplattform zur Integration unterschiedlicher Systeme zur Heimautomatisierung. OpenHAB unterstützt die Erstellung von Automatisierungsregeln und einheitlichen Benutzerschnittstellen. OpenHAB versucht nicht, bereits existierende *Smart Home* Lösungen zu ersetzen, sondern versteht sich vielmehr als ein *System von Systemen*, welches bestehende Lösungen integriert und von der jeweiligen spezifischen Technologie weitestgehend abstrahiert. Der Fokus von openHAB liegt hierbei auf der Realisierung einer einheitlichen Interaktionsmöglichkeit des Benutzers mit den ihn umgebenden *Smart Home* Technologien. Ein zentraler Aspekt von openHAB ist der Begriff des *Items*. Ein *Item* abstrahiert von konkreten Gerätefunktionalitäten und stellt somit eine einheitliche, plattforminterne Beschreibung für *Sensoren*, *Aktuatoren* und den entsprechenden Daten bereit. Gerätespezifische Informationen werden in der *Item* Definition somit nicht berücksichtigt. Das openHAB Projekt untergliedert sich in zwei Teilbereiche: *openHAB Runtime* und *openHAB Designer*.

OpenHAB Runtime

Die *openHAB Runtime* besteht aus einer Menge von *OSGi Bundles* auf der Basis des Equinox OSGI Frameworks. Die zugrunde liegende Architektur ist somit modular aufgebaut und erlaubt das Hinzufügen und Entfernen von Funktionalitäten zur Laufzeit des Systems. So können beispielsweise neue Technolgien zur Heimautomatisierung über sogenannte *Protocol Bindings* integriert werden. Die Kommunikationsinfrastruktur von openHAB basiert auf zwei verschiedenen Ansätzen. Durch den Einsatz eines *Event Bus* können Informationen an zustandslose Services übermittelt werden.

OpenHAB unterscheidet zwischen Kommandos zum Auslösen einer Aktion und Statusinformationen, beispielsweise zur Abfrage des aktuellen Zustand eines angeschlossenen Gerätes. Der *Event Bus* setzt technologisch auf dem *OSGi EventAdmin Service* auf. Der *Event Bus* ermöglicht zudem die Verknüpfung mit anderen laufenden OpenHAB Instanzen. Das *Item Repository* bietet eine statische Kommunikationsinfrastruktur für Services, die einen internen Zustand haben. Es ist mit dem *Event Bus* verbunden und speichert und verwaltet alle Zustandsinformationen der entsprechenden *Items*. Die Verwendung

des *Item Repositories* befreit einzelne *Bundles* von der Aufgabe, selbst ihren Zustand verwalten zu müssen und bündelt somit die relevanten Informationen an einer zentralen Stelle innerhalb der Plattform. Die zugrunde liegende Architektur der *openHAB Runtime* ist in Abbildung 5.3 dargestellt[10].

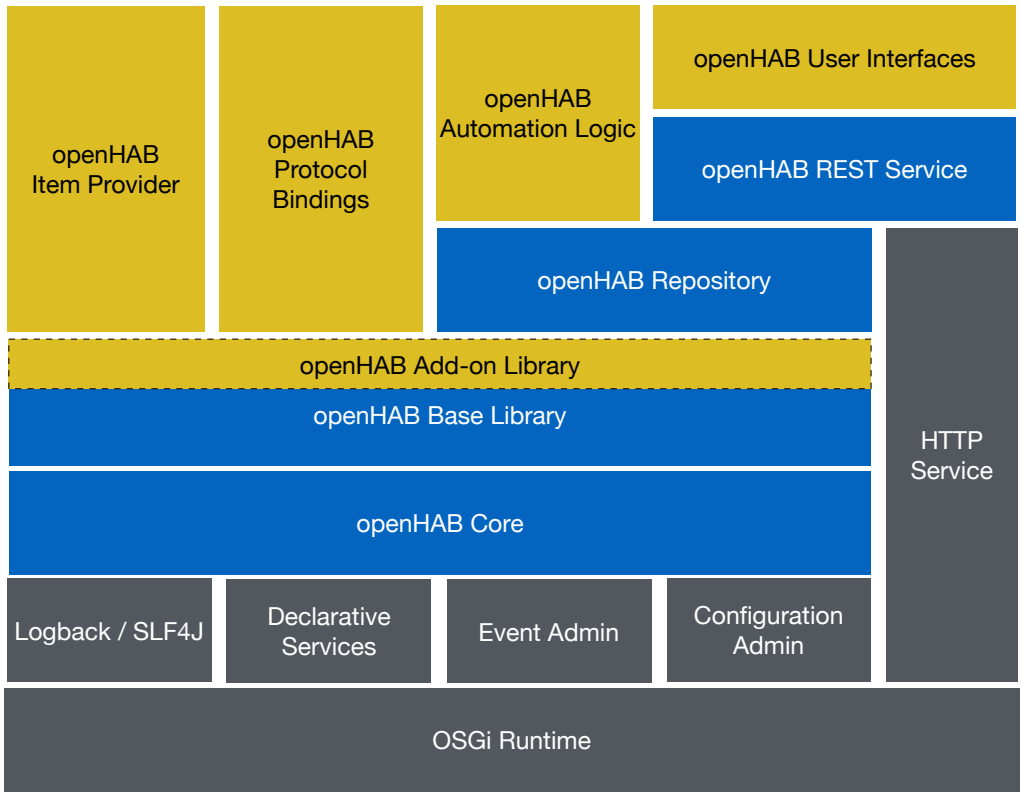


Abbildung 5.3: Überblick über die Architektur der *openHAB Runtime*.

Zur Realisierung von Szenarien zur Heimautomation verfügt openHAB über einen eingebauten und leichtgewichtigen Produktionsregelinterpreter. Darüber hinaus bietet openHAB mit *Sitemaps* eine generische Möglichkeit, Benutzerschnittstellen textuell zu beschreiben. Die *Sitemap* stellt hierbei eine baumartige Repräsentation einzelner Seiten oder Sichten einer Benutzerschnittstelle dar. Die auf diese Art und Weise definierten Schnittstellen können schließlich mit konkreten *Items* innerhalb der Plattform verknüpft werden.

OpenHAB Designer

Mit dem *openHAB Designer* wird eine auf der Eclipse Rich Client Plattform (RCP) basierende Anwendung zur Verfügung gestellt, mit der die openHAB Plattform konfiguriert werden kann. Abbildung 5.4 zeigt die Bedienoberfläche am Beispiel eines Editors zum Bearbeiten von Regeln.

OpenHAB und alle relevanten Komponenten sind komplett als Open Source unter der Eclipse Public License (EPL) veröffentlicht und stehen zusammen

[10] nach http://www.openhab.org [Letzter Zugriff: 31.01.2015]

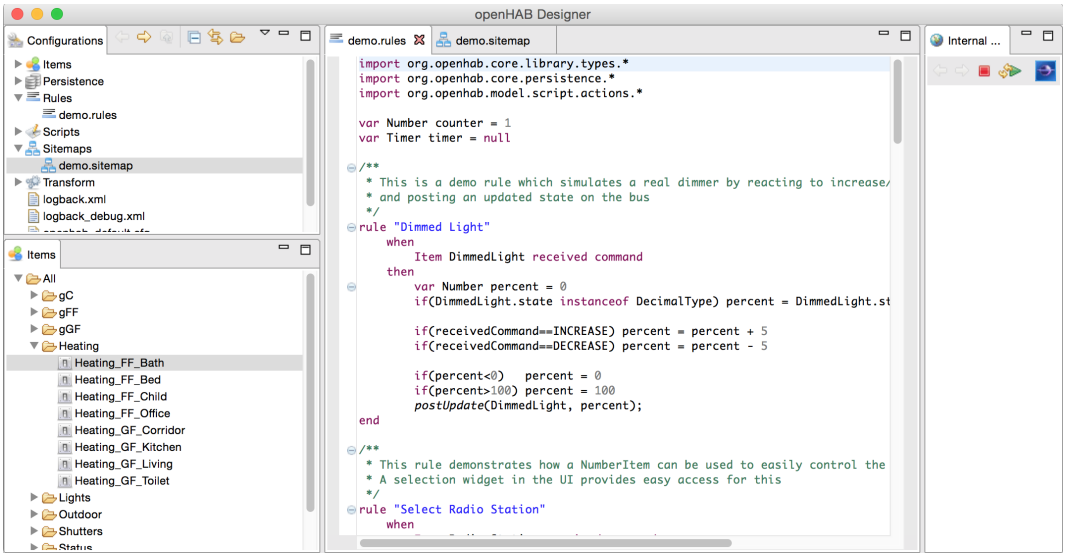


Abbildung 5.4: Überblick über die Bedienoberfläche des openHAB Designers.

mit ausführlichen Dokumentationen, Beispielen und API Beschreibungen auf der entsprechenden Projekthomepage[11] zum Herunterladen bereit.

5.3.4 Eclipse SmartHome

Die Eclipse SmartHome Plattform[12] basiert im Kern auf der zuvor beschriebenen openHAB Architektur. Die Kernkomponenten von openHAB wurden 2013 mit Eclipse SmartHome in die Eclipse Foundation integriert. Der Hauptgrund für die Integration in Eclipse war die Schaffung einer soliden und vertrauenswürdigen Ausgangsbasis für die Weiterentwicklung von *Smart Home* Plattformen. Eclipse SmartHome soll hierbei als Grundlage zur Zusammenarbeit von Industriepartnern aus dem Heimautomatisierungsbereich dienen[13]. Die Eclipse SmartHome Plattform ist als vollständiges Entwicklungsframework zu verstehen, auf dessen Basis konkrete *Smart Home Middlewares* realisiert werden können. Mit der Weiterentwicklung von openHAB wird ein Beispiel für eine solche *Middleware* vorgestellt. Ein wichtiger Aspekt von Eclipse SmartHome ist zudem die Bereitstellung einer vollständig auf Eclipse basierenden integrierten Entwicklungsumgebung sowie die Zusammenarbeit mit anderen *Eclipse Internet of Things* (IoT) Projekten[14].

5.3.5 Google Nest

Google ist mit Android@Home bereits 2011 in den *Smart Home* Markt eingetreten. Mit dem Kauf von Nest Labs, einem US-amerikanischen Unternehmen

[11] http://www.openhab.org [Letzter Zugriff: 31.01.2015]
[12] https://eclipse.org/smarthome [Letzter Zugriff: 31.01.2015]
[13] http://www.heise.de/developer/meldung/Eclipse-SmartHome-soll-Fragmentierung-im-Smart-Home-Bereich-verhindern-2225118.html [Letzter Zugriff: 28.01.2015]
[14] http://iot.eclipse.org [Letzter Zugriff: 31.01.2015]

mit Schwerpunkt auf der Entwicklung von *Smart Home* Technologien, hat Google Anfang 2014 seine Aktivitäten auf dem *Smart Home* Markt wiederum intensiviert. Google bietet derzeit mit einem selbstlernenden Thermostat sowie einem vernetzten Rauchmelder zwei Nest Produkte[15] für den Consumer Markt an. Die Kernkomponente der Nest Architektur ist ein cloudbasierter *Nest Service*. Dieser Service stellt ein globales Datenmodell für eine *Smart Home* Umgebung bereit. Die angeschlossenen Nest Geräte sowie Softwareanwendungen können das Datenmodell auslesen und entsprechend verändern. Änderungen des Umgebungszustandes werden direkt an alle Anwendungen verteilt. Aus Entwicklersicht kann der *Nest Service* über eine REST Schnittstelle adressiert werden, welche den Zustand der Umgebung oder einzelner Nest-Produkte im JSON Format zurückgibt. Darüber hinaus wird mit dem Firebase Protokoll[16] eine weitere effiziente Schnittstelle zu den verfügbaren Nest-Funktionaliäten angeboten. Abbildung 5.5 gibt eine Übersicht über die beschriebene cloudbasierte Nest Architektur.

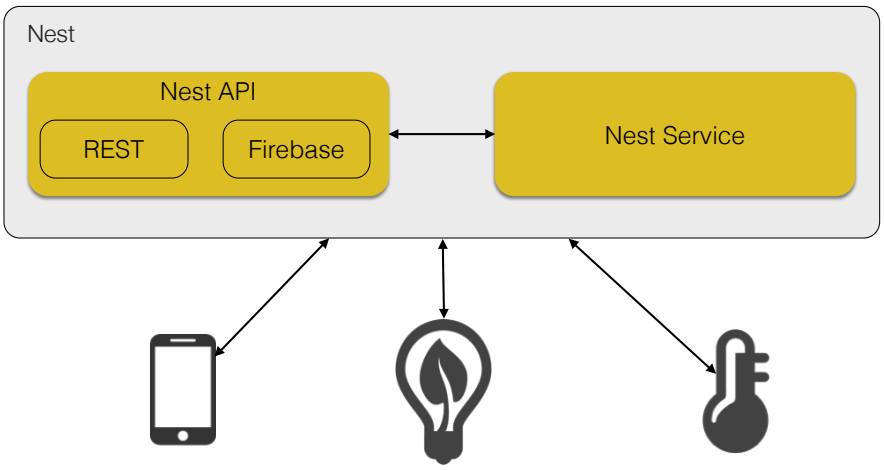


Abbildung 5.5: Überblick über die cloudbasierte Google Nest Architektur (nach http://developer.nest.com).

Dem Benutzer wird standardmäßig eine webbasierte Schnittstelle zu seinem persönlichen Nest Konto angeboten, welche darüber hinaus auch für Android und iOS-Geräte optimiert wurde. Nest verfügt über unterschiedliche Sicherheitsmechanismen sowie über ein auf OAuth 2.0 basierendes Autorisierungsprotokoll. Neben einer ausführlichen Dokumentation werden zudem auch webbasierte Entwicklungswerkzeuge bereitgestellt.

Abbildung 5.6 zeigt ein Beispiel für eine Bedienoberfläche der webbasierten Google Nest Schnittstelle.

[15]https://nest.com [Letzter Zugriff: 28.01.2015]
[16]https://www.firebase.com [Letzter Zugriff: 31.01.2015]

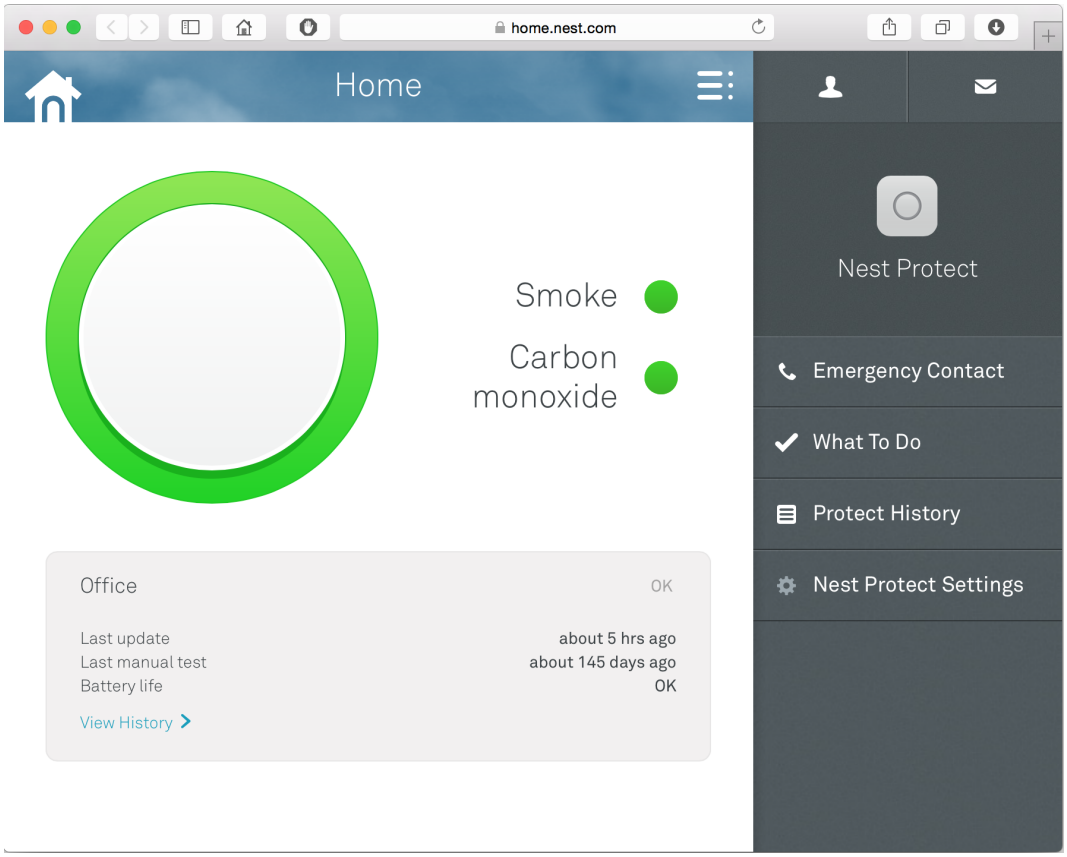


Abbildung 5.6: Beispiel einer Bedienoberfläche der webbasierten Google Nest Schnittstelle.

5.3.6 Apple HomeKit

Mit der neuesten Version seines mobilen Betriebssystems iOS 8 hat Apple mit HomeKit[17] die Möglichkeit geschaffen, unterschiedliche *Smart Home* Produkte zu integrieren und über Smartphone und Tablet zu überwachen und zu steuern. Durch den Einsatz von HomeKit soll eine einheitliche Steuerung für Geräte von verschiedenen Herstellern und mit unterschiedlichen Anwendungsschwerpunkten realisiert werden. Der Fokus des Ansatzes liegt hierbei nicht auf der Bereitstellung eines zentralen Servers oder Gateways innerhalb eines *Smart Homes*. Vielmehr umfasst das Framework Protokolle und Programmierschnittstellen, auf deren Basis einerseits Drittanbieter ihre Hardware in das Apple-Ökosystem integrieren können und andererseits aus Anwendersicht mobile iOS-Anwendungen implementiert werden können, die dann Zugriff auf die angeschlossene Hardware haben. Durch diese Vorgehensweise ermöglicht Apple den Aufbau eines iOS-spezifischen Marktplatzes für eine Fülle von unterschiedlichen *Smart Home* Applikationen. Abbildung 5.7 gibt eine Übersicht über die zugrunde liegende HomeKit Architektur.

HomeKit stellt Hardwareanbietern und App-Entwicklern die folgenden drei Kernfunktionalitäten zur Verfügung:

[17] https://developer.apple.com/homekit [Letzter Zugriff: 31.01.2015]

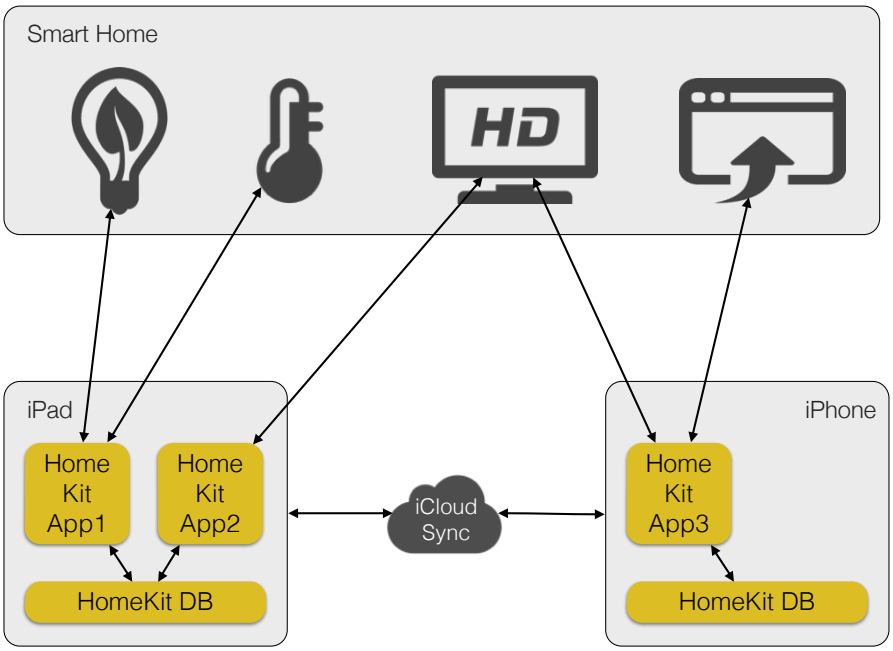


Abbildung 5.7: Überblick über die Apple HomeKit Architektur (nach Ahlers et al. (2014)).

- Automatische Erkennung von *Smart Home* Anwendungen und die Speicherung der entsprechenden Daten in einer zentralen Konfigurationsdatenbank.
- Visualisierung, Bearbeitung und Analyse der in der Datenbank gespeicherten Informationen.
- Kommunikation mit den *Smart Home* Anwendungen und somit Manipulation des Umgebungszustandes.

Durch die zentrale Datenbank innerhalb von iOS kann darüber hinaus auch das hausinterne *Speech Interpretation and Recognition Interface* (Siri) integriert werden und somit eine natürlichsprachliche Interaktion mit der *Smart Home* Umgebung ermöglicht werden[18]. HomeKit betrachtet eine *Smart Home* Umgebung als eine Sammlung von Anwendungen zur Heimautomation. Mit Hilfe der Konfigurationsdatenbank kann der Endnutzer diesen Anwendungen konstante Namen und Beschreibungen zuordnen. Die vorgegebene hierarchische Modellstruktur setzt sich aus den folgenden Elementen zusammen:

- **Homes:** *Homes* stellen die oberste Hierarchieebene dar und entsprechen einer konkreten *Smart Home* Umgebung. Jede Konfiguration muss mindestens ein *Home* Element besitzen. Möchte man verschiedene Umgebungen, wie zum Beispiel die eigene Wohnung und das Büro voneinander unterscheiden, so können mehrere *Home* Elemente erzeugt werden.

[18]Ein vergleichbarer, aber in der Komplexität der sprachlichen Kommandos fortschrittlicherer Ansatz wird beispielsweise durch die von der SemVox GmbH entwickelte ODP S3-Plattform verfolgt (www.semvox.de).

- **Rooms:** *Rooms* sind optionale Bestandteile eines *Home* Elements und symbolisieren die einzelnen Räume innerhalb einer Umgebung. Die *Room* Elemente besitzen hierbei keinerlei physische Eigenschaften, sondern werden lediglich durch einen vom Benutzer zugewiesenen Bezeichner, wie zum Beispiel *Küche*, definiert.

- **Accessories:** *Accessories* repräsentieren die in einem *Home* installierten und einem *Room* Element zugewiesenen *Smart Products*. Apple arbeitet hierzu laut Ahlers et al. (2014) bereits mit mehreren Industriepartnern und Technologieanbietern auf dem deutschen Markt zusammen. Zu ihnen gehören zum Beispiel iHome, Osram, Cree, Chamberlain, Marvell, Skybell, Honeywell, Haier, Schlage, Philips, Kwikset, Tado, Broadcom, Netamo und Withings.

- **Services:** Die *Services* stellen die Funktionalitäten von *Smart Products* dar. Diese umfassen auf der einen Seite kontrollierbare Eigenschaften wie Lichtsteuerung, aber auch interne Managementfunktionen wie das Aufspielen einer neuen Firmware.

- **Zones:** *Zones* stellen optionale räumliche Gruppierungen von *Room* Elementen dar, zum Beispiel *Keller* oder *Dachgeschoss*. Hierdurch können beispielsweise mit Hilfe von Siri Kommandos wie *"Siri, mach das Licht im Keller aus!"* realisiert werden.

5.4 Agentensysteme in Intelligenten Umgebungen

Cook (2009) skizziert den Forschungsbereich der Agentensystems im Hinblick auf ihren Einsatz in *Intelligenten Umgebungen*. Grundsätzlich unterscheiden sie zwischen zwei gegensätzlichen Sichtweisen. Einerseits betrachten sie *Agenten* als die grundlegenden Komponenten einer Softwarearchitektur zur Kontrolle einer *Intelligenten Umgebung*. Andererseits kann jedem Bewohner einer *Intelligenten Umgebung* ein *Softwareagent* innerhalb eines Gesamtsystems zugeordnet werden. Ausgehend von diesen unterschiedlichen Sichtweisen hebt Cook (2009) vier Forschungsrichtungen zum Einsatz von *Multiagentensystemen* in *Intelligenten Umgebungen* hervor. Abbildung 5.8 gibt einen Überblick über die jeweiligen Forschungsgebiete.

Abbildung 5.8: Zusammenfassung der Forschungsrichtungen auf dem Gebiet von *Agenten* in *Intelligenten Umgebungen*.

Der Forschungsbereich 1 beschäftigt sich mit dem Einsatz von *Multiagentensystemen* als Softwarearchitektur innerhalb *Intelligenter Umgebungen*. Forschungsbereich 2 und 3 betrachten jeweils Mehrbenutzerszenarien. Während bei Forschungsbereich 2 die Identifikation und die Lokalisierung des Benutzers im Vordergrund steht, wird bei Forschungsbereich 3 untersucht, inwieweit die Aktivitäten und Verhaltensweisen mehrerer Benutzer erkannt werden können. Der Forschungsbereich 4 untersucht schließlich mögliche Verhandlungsstrategien innerhalb eines *Multiagentensystems* zur kooperativen Problemlösung und zur Konfliktauflösung von gegensätzlichen Zielen der *Agenten*. Die folgenden Abschnitte geben einen Überblick über verwandte Arbeiten, die den Einsatz von *Multiagentensystemen* in *Intelligenten Umgebungen* betrachtet haben.

5.4.1 ISES

Davidsson und Boman (2000) beschreiben ein *Multiagentensystem* zur Überwachung und zur Steuerung von Bürogebäuden, welches im Rahmen des ISES Projekts (Ottosson et al., 1998) entwickelt und eingesetzt wurde. Das Ziel des vorgestellten Ansatzes ist es einerseits den gesamten Energieverbrauch des Bürogebäudes zu minimieren und andererseits die Zufriedenheit der einzelnen Benutzer durch das Bereitstellen von Mehrwertdiensten zu maximieren. Zu den üblichen Maßnahmen zur Einsparung von Energie gehören zum Beispiel das automatische Ausschalten von Licht sowie das Reduzieren der Raumtemperatur in nicht genutzten Räumen. Typische Maßnahmen zur Steigerung der Benutzerzufriedenheit sind beispielsweise die automatische Licht- und Temperaturregelung in Abhängigkeit der jeweiligen persönlichen Präferenzen. Die Grundidee des in ISES entwickelten *Multiagentensystems* besteht darin, dass jeder *Softwareagent* einen bestimmten Teil des Gebäudes, beziehungsweise einen bestimmten Aspekt der Umgebung überwachen und steuern kann. Es wird grundsätzlich zwischen vier Kategorien von *Agenten* unterschieden:

- **Personal Comfort Agent (PCA):** Der PCA stellt die Präferenzen eines konkreten Benutzers innerhalb des Systems dar und versucht durch seine Handlungen dessen Zufriedenheit zu maximieren.

- **Room Agent (RA):** Ein RA kontrolliert einen dedizierten Raum innerhalb der Büroumgebung und versucht, dessen Energieverbrauch unter Berücksichtigung des PCA des darin befindlichen Benutzers zu optimieren.

- **Environmental Parameter Agent (EPA):** Der EPA kontrolliert jeweils einen bestimmten Umgebungsparameter innerhalb eines Raums. So kann beispielsweise ein *Temperatur Agent* die Sensordaten des Temperaturfühlers auslesen und gleichzeitig die Heizkörper im Raum ansteuern.

- **Badge System Agent (BSA):** Der BSA stellt mit Hilfe eines tragbaren Ausweises eine Möglichkeit zur Lokalisierung einzelner Benutzer innerhalb des Bürogebäudes zur Verfügung.

Das Verhalten eines *Agenten* wird durch eine Menge von Regeln definiert, welche das beabsichtigte Kontrollschema der Büroumgebung widerspiegeln. Zustandsänderungen innerhalb der Umgebung führen zu Benachrichtigungen der relevanten Agenten, was wiederum das Ausführen der entsprechenden Regeln nach sich zieht. Abbildung 5.9 zeigt den Informationsfluss des ISES *Multiagentensystems*, wenn ein Benutzer einen neuen Raum innerhalb der Büroumgebung betritt. Daneben beinhaltet das System erste Ansätze zur Auflösung von zueinander in Konflikt stehenden Zielen einzelner *Agenten*, wie zum Beispiel den Einsatz von *Real-time Decision Support Systems* (Boman et al., 2013).

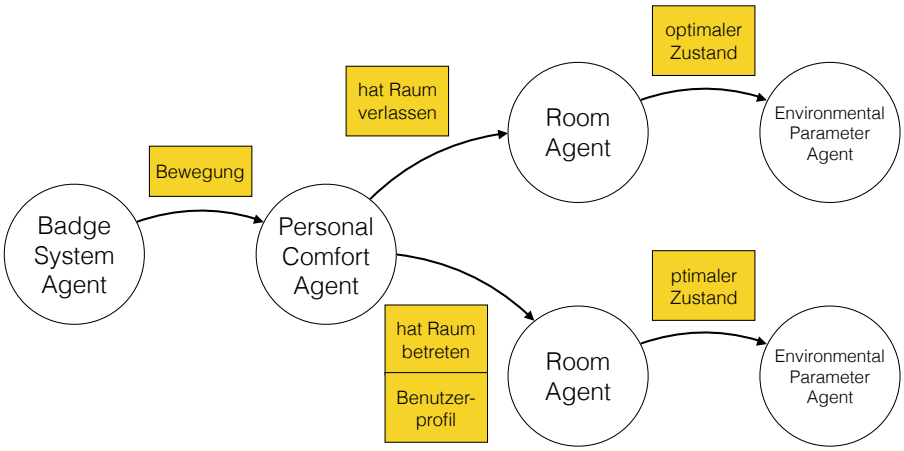


Abbildung 5.9: Überblick über den Informationsfluss innerhalb des ISES *Multiagentensystems*.

5.4.2 iDorm Intelligent Embedded Agents

Hagras et al. (2004) verfolgen innerhalb des iDorm Projektes einen Ansatz zur Realisierung von *Intelligenten Umgebungen* mit Hilfe sogenannter eingebetteter Intelligenz (Callaghan et al., 2001). Aufbauend auf dieser Grundidee wird ein Architekturmodell zur Integration von verteilten, intelligenten und eingebetteten *Agenten* vorgestellt. Im Gegensatz zu reinen softwarebasierten Ansätzen basiert die iDorm Architektur auf einer Kombination aus *Softwareagenten* und *Hardwareagenten*. Die folgende Übersicht skizziert die zugrunde liegenden Agententypen:

- **Embedded Agent:** Der *Embedded Agent* überwacht die Werte des zugrunde liegenden Sensornetzwerkes, enthält ein persönliches Benutzerprofil und ist dazu in der Lage geeignete Steuerbefehle für die Umgebung zu berechnen.
- **Robotic Agent:** Der *Robotic Agent* ist ein mobiler Serviceroboter, der sich innerhalb der Umgebung autonom zurechtfinden und seine Verhaltensweisen über adaptive Lernmechanismen der Umgebung und dem Benutzer anpassen kann.

Callaghan et al. (2002) unterscheiden zwischen vier möglichen, übergeordneten Verhaltensmustern eingebetteter *Agenten*. Die *Safety Behaviour* soll hierbei jederzeit einen sicheren und stabilen Zustand der Umgebung ermöglichen. Die *Emergency Beaviour* beinhaltet geeignete Verhaltensmuster in einer Notfallsituation, wie zum Beispiel Feuer oder Einbruch. Durch die *Economic Behaviour* soll sichergestellt werden, dass durch die Umgebung keine Energie unnötig verschwendet wird. Zuletzt ermöglicht die *Comfort Behaviour* eine dynamische Anpassung der Umgebung an die Präferenzen und persönlichen Vorlieben eines einzelnen Benutzers. Darüber hinaus beinhaltet die Architektur

ein mobiles Endgerät zur Überwachung und Kontrolle der iDorm Umgebung. Der Fokus des iDorm Projektes liegt auf der Realisierung eines inkrementellen Lernalgorithmus basierend auf den Interaktionen zwischen den jeweiligen *Agenten* und der Umgebung.

5.4.3 MavHome

Cook et al. (2003) stellen mit MavHome eine auf *Agenten* basierende *Intelligente Umgebung* vor. Ihr Fokus liegt auf der Annahme, dass eine Umgebung wie ein rational denkender eigenständiger *Agent* handeln sollte, indem er seinen zukünftigen Zustand und den seiner Bewohner ermitteln kann. Durch diese Form der Adaption ist es der Umgebung möglich, den Komfort seiner Bewohner zu maximieren und gleichzeitig die Betriebskosten zu minimieren. Die hierzu verwendete Architektur besteht aus einer hierarchischen Struktur von *Softwareagenten*. Abbildung 5.10 zeigt den Informationsfluss innerhalb einer Agentenarchitektur anhand einer auf Bewegungsmeldern basierenden automatischen Lichtsteuerung.

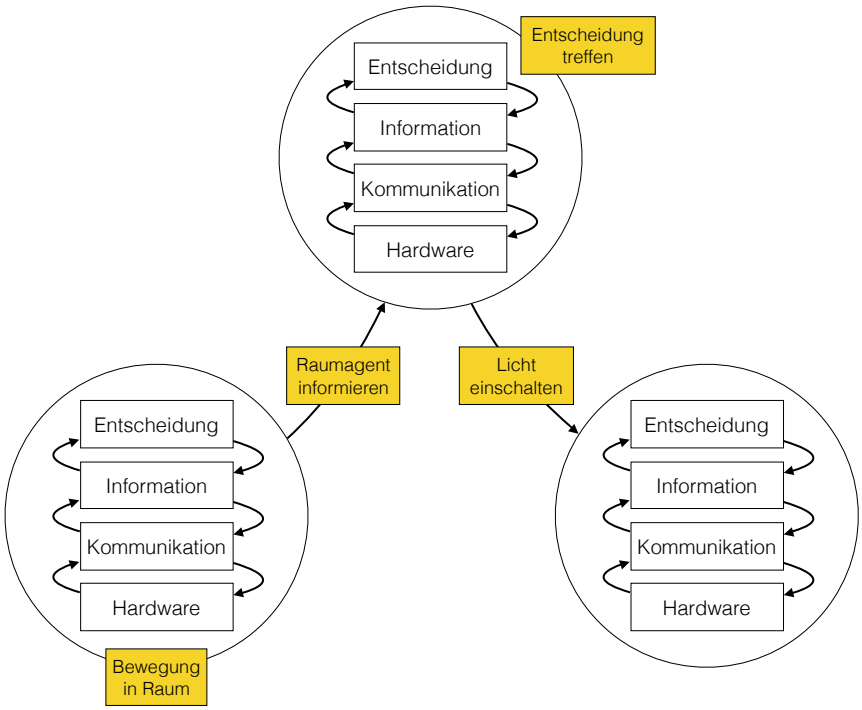


Abbildung 5.10: Übersicht über den Informationsfluss des MavHome Agentensystems am Beispiel einer automatischen Lichtsteuerung.

Jeder *Agent* setzt sich aus vier unterschiedlichen Schichten zusammen:

- **Entscheidungsschicht:** Die Entscheidungsschicht stellt die oberste Ebene dar. Sie ist dafür verantwortlich, basierend auf den verfügbaren Informationen, die für die aktuelle Situation geeigneten Aktionen des *Agenten* auszuwählen beziehungsweise zu erstellen.

- **Informationsschicht:** Die Informationsschicht sammelt alle auflaufenden Informationen über die zugrunde liegende Umgebung und führt die notwendigen Berechnungen durch, wie zum Beispiel das Erkennen von Aktivitäten oder das Vorhersagen des Energieverbrauchs. Sie bildet somit die Grundlage für den Entscheidungsprozess innerhalb der Entscheidungsschicht.

- **Kommunikationsschicht:** Die Kommunikationsschicht unterstützt die Kommunikation zwischen den einzelnen *Agenten* und ermöglicht somit den gegenseitigen Austausch von Informationen, Anforderungen und Anfragen.

- **Hardwareschicht:** Die Hardwareschicht stellt die verfügbaren Gerätefunktionalitäten zur Verfügung.

5.4.4 CoBrA

Chen (2004) verfolgt mit der *Context Broker Architecture* (CoBrA) ein zentralisierten Ansatz zur Verwaltung von Kontextwissen innerhalb einer *Intelligenten Umgebung*. Kernbestandteil der Architektur ist der sogenannte *Context Broker*, ein intelligenter *Softwareagent*, der auf einem leistungsstarken Desktop Computer läuft. Die Idee dieses Ansatzes basiert auf der Feststellung, dass viele *Smart Products* innerhalb einer *Intelligenten Umgebung* nicht über ausreichend Rechenleistung für kontextsensitive Berechnungen verfügen. Durch den Einsatz einer leistungsstarken und zentral ausgerichteten *Context Broker* Komponente können die *Smart Products* dessen verfügbare Rechenleistung nutzen, um ressourcenintensive Berechnungen durchzuführen. Der *Context Broker Agent* setzt sich aus vier Komponenten zusammen:

- **CoBra Ontology (COBRA-ONT):** COBRA-ONT stellt eine Sammlung von ontologischen Konzepten zur Repräsentation von kontextuellem Wissen zur Verfügung. COBRA-ONT ist eine Erweiterung der SOUPA Ontologie (Chen et al., 2005), die in Abschnitt 5.5 ausführlich beschrieben wird.

- **Context Knowledge Base:** Die *Context Knowledge Base* ist dafür verantwortlich, die relevanten Kontextinformationen, welche sich aus Sensordaten der physischen Umgebung und aus dem abgeleiteten Wissen der *Context Reasoning Enginge* zusammensetzt, zu speichern und zu verwalten. Das Kontextwissen kann hierzu in einer entsprechenden Datenbank abgelegt werden.

- **Context Reasoning Engine:** Die *Context Reasoning Engine* ist ein Regelsystem, das es dem *Context Broker* durch den Einsatz von Inferenzregeln erlaubt, das vorhandene Kontextwissen zu interpretieren sowie Inkonsistenten darin zu erkennen. Zum einen werden hierfür die internen ontologischen Konzepte genutzt, zum anderen können externe domänenspezifische Heuristiken zur Wissensableitung verwendet werden.

- **Privacy Protection Module:** Innerhalb des *Privacy Protection Modules* wird sichergestellt, dass Datenschutz und Privatsphäre des Benutzers jederzeit gewährleistet werden. Hierzu verwaltet das Modul die entsprechenden Einstellungen und Präferenzen des Benutzers. Bevor der *Context Broker* Wissen über den Benutzer mit anderen Komponenten und *Agenten* teilt, wird zuvor das *Privacy Protection Module* um Erlaubnis gefragt.

Der *Context Broker* kann dank seiner zentralen Position innerhalb der Architektur mit unterschiedlichen heterogenen Komponenten kommunizieren. Abbildung 5.11 gibt einen Überblick über CoBrA und die darin integrierten Teilsysteme.

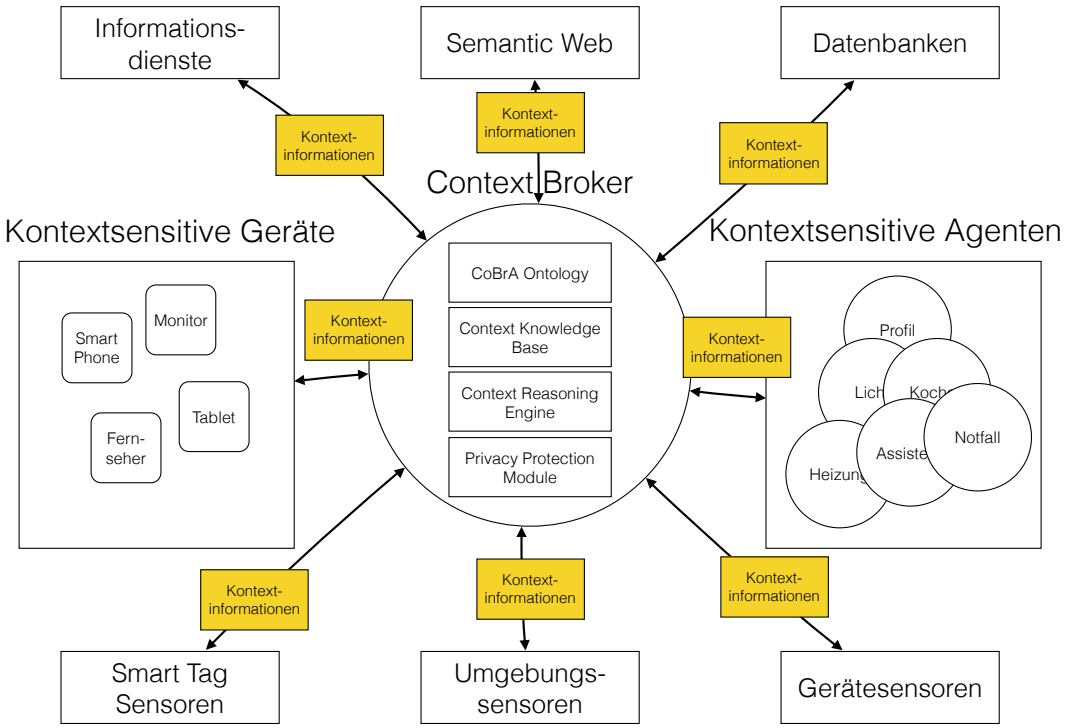


Abbildung 5.11: Überblick über den Informationsfluss innerhalb der zentral ausgerichteten *Context Broker* Architektur (nach Chen (2004)).

5.4.5 Agent-based Intelligent Home Health Care System

Cervantes et al. (2007) beschreiben den Einsatz eines *Multiagentensystems* zur Realisierung eines Entscheidungsunterstützungssystems zur Verbesserung der medizinischen Betreuung von chronisch kranken Patienten. Hierzu werden mittels medizinischer Sensorik die physiologischen Daten der Patienten kontinuierlich gesammelt. Durch die Integration von künstlichen neuronalen Netzen werden die Daten weiterverarbeitet und schließlich ein *Decision Support System* realisiert, um die Kooperation mehrerer behandelnder Ärzte optimal zu

koordinieren. Jeder Teilnehmer innerhalb des Systems wird durch einen dedizierten *Agenten* repräsentiert. Abbildung 5.12 zeigt die Architektur und den Informationsfluss innerhalb des realisierten *Multiagentensystems*.

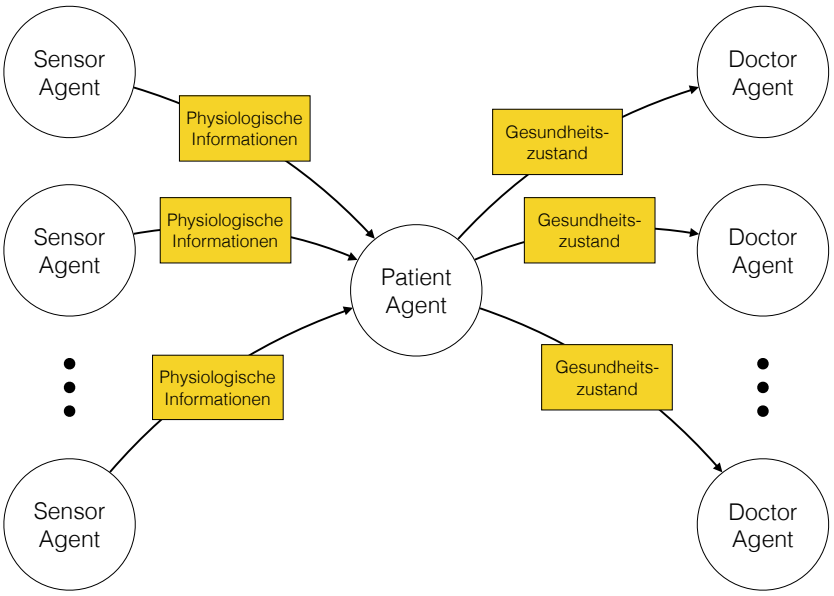


Abbildung 5.12: Übersicht über den Informationsfluss innerhalb des multiagentenbasierten medizinischen Entscheidungsunterstützungssystems (nach Cervantes et al. (2007)).

Innerhalb der Architektur wird zwischen drei Typen von *Agenten* unterschieden:

- **Sensor Agent:** Der *Sensor Agent* ist für das Erfassen und die Analyse der Daten eines medizinischen Messgeräts verantwortlich. Das System kann die einzelnen Sensoren entsprechend der zugrunde liegenden Symptomatik der Krankheit dynamisch konfigurieren.
- **Doctor Agent:** Der *Doctor Agent* verbindet die spezifische Arbeitsumgebung eines behandelnden Arztes mit der lokalen Sensorumgebung des Patienten und sorgt dafür, dass die Patientendaten für den Arzt zugänglich sind.
- **Patient Agent:** Der *Patient Agent* koordiniert die Interaktion zwischen dem *Sensor Agent* und dem *Doctor Agent*. Er überprüft kontinuierlich den Gesundheitszustand des Patienten auf signifikante Veränderungen. Innerhalb des patientenspezifischen *Patient Agent* befindet sich das künstliche neuronale Netz, mit dessen Hilfe der Gesundheitszustand des Patienten interpretiert werden kann, um gegebenenfalls eine Benachrichtigung an einen oder mehrere Ärzte zu generieren.

Die technologische Grundlage wird durch die JADE Plattform gebildet, wobei die plattforminternen Nachrichten auf der *Agent Communication Language* (ACL) basieren (siehe Kapitel 4).

5.4.6 NOCTURNAL

Das Ziel des NOCTURNAL[19] Projekts (Augusto et al., 2011) ist es, die nächtlichen Aktivitäten älterer Menschen mit beginnender Demenz in ihren häuslichen Umgebungen zu überwachen. Falls es die Situation erfordert, sollen entsprechende Hilfestellungen zum Erreichen einer geregelten Nachtruhe gegeben werden. Typische nächtliche Verhaltensmuster dieser Zielgruppe sind zum Beispiel ein unruhiger Schlaf oder das Umherwandern (Carswell et al., 2011).

Durch den Einsatz geeigneter *Sensoren* und die Integration in ein *Multiagentensystem* ist es möglich, solche Aktivitäten gezielt zu erkennen und geeignete therapeutische Interventionsmaßnahmen zu treffen, um dem Benutzer zu einer besseren und geregelteren Nachtruhe zu verhelfen. Innerhalb des Projekts wurde unter anderem eine graphische Benutzerschnittstelle erstellt, die durch den Einsatz von Bildern und Musik versucht, den schlaflosen Benutzer zu entspannen und zu beruhigen.

Die grundlegende Struktur der in NOCTURNAL vorgeschlagenen Architektur besteht aus fünf unterschiedlichen Typen von *Agenten* (McNaull et al., 2011):

- **Sensor Agent:** Der *Sensor Agent* ist für die Weiterverarbeitung der Sensordaten verantwortlich, die zuvor von den einzelnen *Sensoren* in dem *Sensor Data Storage* abgelegt wurden. Hierzu wird eine allgemeine *Sensor Event* Nachricht generiert und an den *Context Agent* übermittelt.

- **Context Agent:** Der *Context Agent* ermittelt anhand der eingehenden *Sensor Events* einen neuen aktuellen Umgebungskontext, welcher in der *Context Data Store* hinterlegt und an den *Intervention Agent* geschickt wird.

- **Intervention Agent:** Der *Intervention Agent* berechnet auf Basis des veränderten Kontexts und des verfügbaren *Intervention Data Stores* die notwendigen Maßnahmen und informiert anschließend den *Interface Agent*.

- **Interface Agent:** Der *Interface Agent* ist für die Präsentation der notwendigen Interventionsmaßnahmen verantwortlich. Im Zusammenspiel mit dem *Profile Agent* wird die vom Benutzer bevorzugte Interaktionsmodalität ausgewählt und eine geeignete Benutzerschnittstelle generiert.

- **Profile Agent:** Der *Profile Agent* erlaubt es dem Endnutzer ein persönliches Benutzerprofil anzulegen, welches die intendierte Benutzerinteraktion spezifiziert.

Die Implementierung des in NOCTURNAL verwendeten *Multiagentensystems* basiert auf der JADE Plattform.

Abbildung 5.13 zeigt die entsprechende Systemarchitektur und den Datenfluss innerhalb des Systems.

[19] *Night Optimised Care Technology for UseRs Needing Assisted Lifestyles*

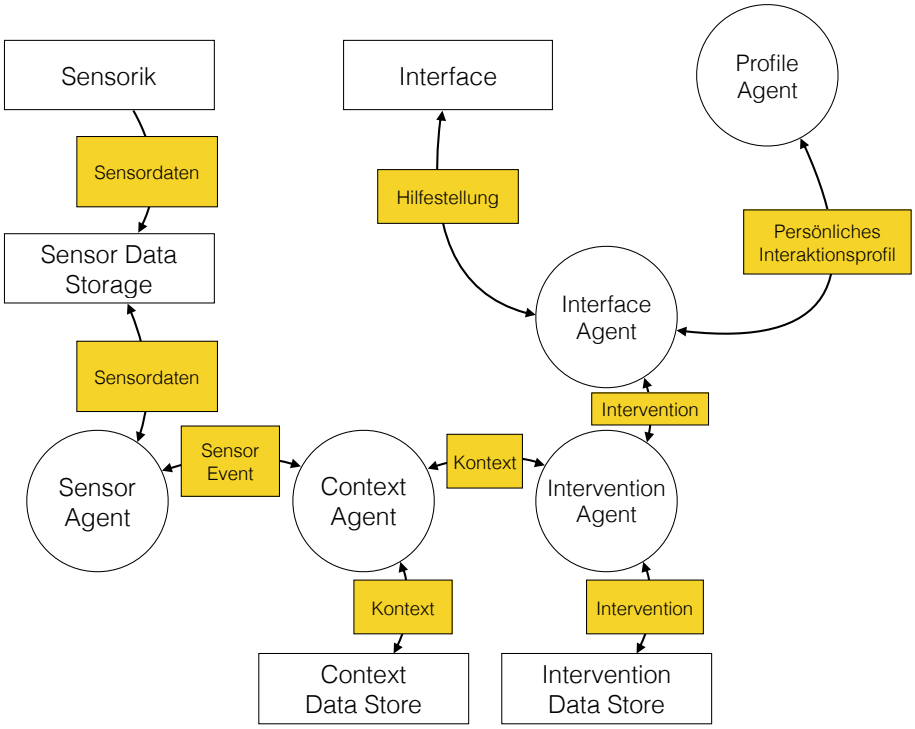


Abbildung 5.13: Übersicht über den Informationsfluss innerhalb des multiagentenbasierten therapeutischen Unterstützungssystems (nach McNaull et al. (2011)).

Ausgehend von den Sensordaten wird der aktuelle Zustand des Benutzers ermittelt, was wiederum zu einer entsprechenden therapeutischen Interventionsmaßnahme führt, die schließlich über eine geeignete Benutzerinteraktion präsentiert wird.

5.4.7 Coalaa

Andriatrimoson et al. (2012) beschreiben mit Coalaa[20] ein adaptives Framework für *Ambient Assisted Living*. Durch den Einsatz eines *Multiagentensystems* und die darauf aufbauende Kooperation von einzelnen *Agenten* wird ein adaptiver Ansatz verfolgt. Als Anwendungsszenario wird der Einsatz eines mobilen Serviceroboters spezifiziert, welcher mit der Umgebung und den darin befindlichen Objekten kommunizieren und interagieren kann. Das Ziel besteht darin, ältere Benutzer innerhalb ihrer gewohnten häuslichen Umgebung durch Bereitstellen von Dienstleistungen zu unterstützen.

Der Einsatz eines mobilen Serviceroboters wird als eine Erweiterung des Forschungsgebiets der *Ambienten Intelligenz* definiert. Das zugrunde liegende *Multiagentensystem* stellt eine virtuelle Repräsentation der Umgebungssensoren sowie des Serviceroboters dar und ermöglicht eine Kooperation der einzel-

[20] *Coalitions for Ambient Assisted Living Applications*

nen Teilkomponenten miteinander (Andriatrimoson et al., 2012).

Abbildung 5.14 gibt einen Überblick über die Coalaa Systemarchitektur.

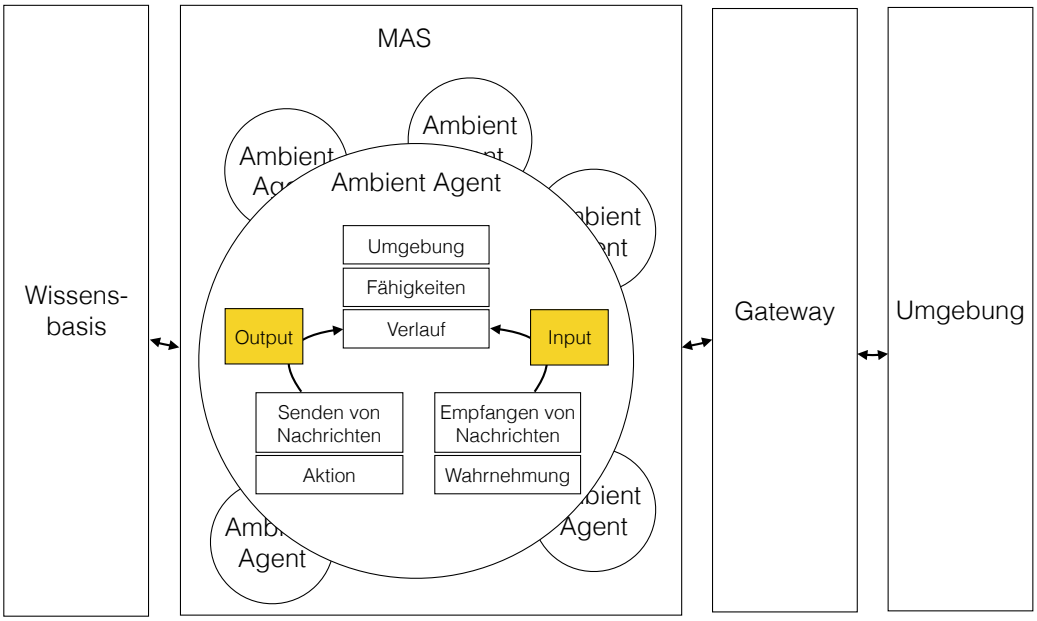


Abbildung 5.14: Übersicht über den Informationsfluss innerhalb der Coalaa Systemarchitektur und den internen Aufbau eines *Ambient Agents*.

Grundsätzlich werden drei notwendige Softwarekomponenten identifiziert:

- **Wissensbasis:** Innerhalb des Systems wird Wissen aus zwei Perspektiven betrachtet. Das dynamische Wissen repräsentiert den Zustand der Umgebung und der darin befindlichen *Sensoren* und *Aktuatoren*. Dieses Wissen wird durch das *Gateway* ständig aktualisiert und dargestellt. Das statische Wissen hingegen umfasst Informationen über die Zusammensetzung und den Aufbau der Umgebung sowie die Eigenschaften und Funktionalitäten der einzelnen *Sensoren*. Dies beinhaltet neben den räumlichen Abhängigkeiten auch eine Modellierung des Benutzers.

- **Multiagentensystem:** Das *Multiagentensystem* besteht aus mehreren *Ambient Agents*. Jeder *Ambient Agent* repräsentiert ein *Smart Product* innerhalb der Umgebung und stellt dem Benutzer die entsprechenden Funktionalitäten als Service zur Verfügung. Jeder *Agent* entscheidet lokal, inwiefern er den Benutzer optimal unterstützen kann. Innerhalb des dafür notwendigen Entscheidungsprozesses kommen drei Hauptparameter zum Tragen. Die *Neighbourhood* definiert die Umgebung des jeweiligen *Agenten*, das heißt alle benachbarten *Agenten*, zu denen eine direkte oder indirekte Relation besteht. Die *History* speichert und verwaltet historische Sensorinformationen der Umgebung. Die *Ability* beschreibt schließlich die Funktionalitäten und Services in Bezug auf das durch den *Agenten* repräsentierte *Smart Product*.

- **Gateway:** Das *Gateway* stellt die Schnittstelle zwischen der *Intelligenten Umgebung* und dem *Multiagentensystem* dar. Aufgrund der Heterogenität der verfügbaren Geräte innerhalb der Umgebung ist es notwendig, eine Standardisierung des Informationsaustauschs vorzunehmen, so dass auch *Sensoren* und *Aktuatoren* von verschiedenen Herstellern einheitlich in das *Multiagentensystem* eingebunden werden können. Das *Gateway* stellt somit den aktuellen Status der Umgebungssensorik für die Weiterverarbeitung im *Multiagentensystem* bereit.

Das Coalaa System wurde auf Basis der JADE *Agentenplattform* und der *Jess Rule Engine*[21] realisiert.

5.4.8 Sensor- und Aktuatornetzwerk

Sun et al. (2013) haben sich bei der von ihnen vorgeschlagenen Multiagentenarchitektur zum Ziel gesetzt, eine intelligente Automation der häuslichen Umgebung zu realisieren. Hierzu verfolgen sie drei unterschiedliche Entwicklungspfade. Zunächst setzen sie die interne Logik der einzelnen *Agenten* auf der BDI Architektur auf (siehe Kapitel 4). Danach wird ein Regelsystem zur Koordination der Agentenkollaboration eingeführt. Zuletzt wird eine Sammlung von Metriken zur Performanzmessung des vorgeschlagenen *Multiagentensystems* vorgestellt.

Das komplette System wurde auf Basis der *JADE Agentenlattform* realisiert. Das System besteht aus vier verschiedenen Typen von *Agenten*:

- **Sensing Agent:** *Sensor Agents* können ihre Umgebung erfassen und zu einem aktuellen Kontextmodell zusammenführen. Die gesammelten Informationen werden an die *Decision Agents* weitergeleitet.

- **Decision Agent:** Die *Decision Agents* empfangen die Sensorinformationen und interagieren mit den *Database Agents*, um Zugriff auf das dort gespeicherte Kontextwissen und die Inferenzregeln zu bekommen. Die getroffene Entscheidung wird im nächsten Schritt an die *Action Agents* übermittelt.

- **Database Agent:** Die *Database Agents* sind für das Speichern des Kontextwissens sowie der entsprechenden Ableitungsregeln verantwortlich.

- **Action Agent:** *Action Agents* führen auf Basis der getroffenen Entscheidungen entsprechende Aktionen innerhalb der Umgebung aus.

Jeder *Agent* wird als unabhängige Kombination von Hard- und Software angesehen. Er kann aus der Umgebung seinen eigenen Kontext ableiten und sich hiermit durch Selbstkonfiguration an sich ändernde Gegebenheiten anpassen.

[21] http://www.jessrules.com [Letzter Zugriff: 10.01.2015]

Des Weiteren verfügt jeder *Agent* über eine eigene Zustandsmaschine und eine Bibliothek an Handlungsmöglichkeiten, auf deren Basis er in Abhängigkeit von seinen Zielen und den Zielen von kooperierenden Agenten die beste Handlungsstrategie auswählen kann. Abbildung 5.15 veranschaulicht den Informationsaustausch zwischen den *Agenten* anhand der jeweiligen Zustände, die jeder einzelne *Agent* einnehmen kann.

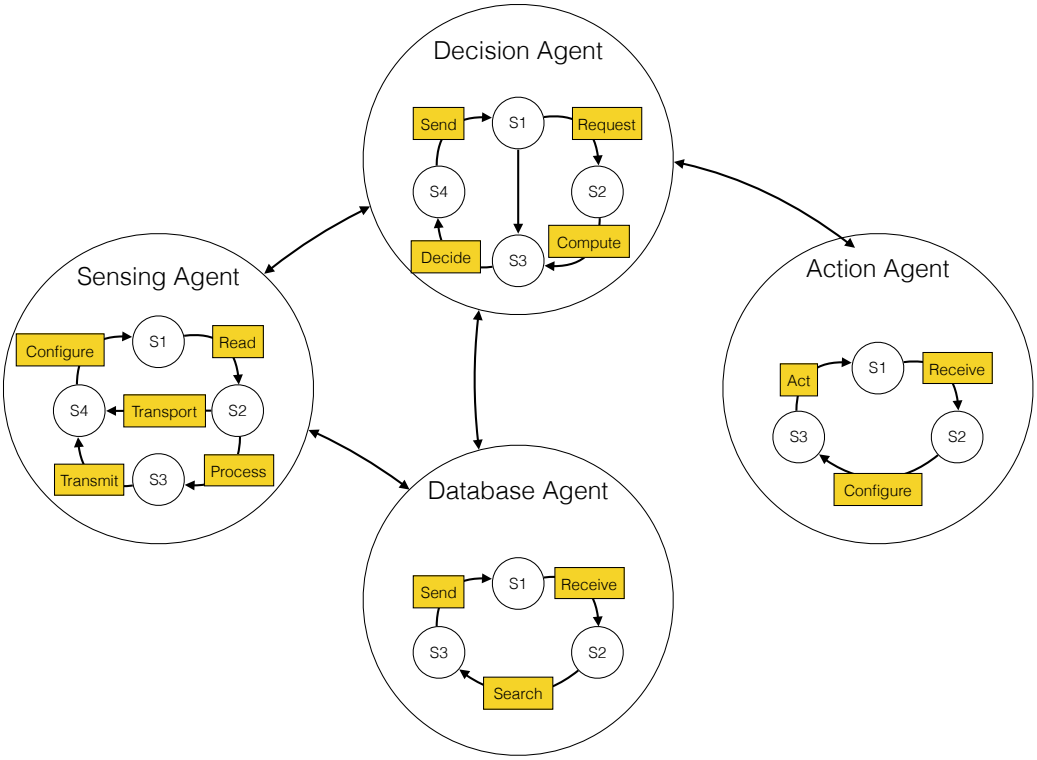


Abbildung 5.15: Übersicht über das Zustandsdiagramm und den Informationsfluss der einzelnen *Agenten* im Sensor- und Aktuatornetzwerk.

Darüber hinaus stellen Sun et al. (2013) eine Methode vor, die durch den Einsatz von Petri-Netzen die Kollaboration einzelner *Agenten* formal beschreiben und auf Schwachstellen überprüfen kann. Hierdurch kann letztendlich das gesamte *Agentensystem* hinsichtlich der Umsetzbarkeit und der Performanz evaluiert werden.

5.4.9 Virtual Carer

Mit dem Virtual Carer Projekt (Sernani et al., 2013b,a) wurde das Ziel verfolgt, innerhalb einer *Intelligenten Umgebung* mit älteren Bewohnern interagieren zu können. Hierzu soll ihr Gesundheitszustand überwacht und situationsgerecht die Umgebung des Benutzers beeinflusst werden.

Abbildung 5.16 zeigt die Architektur und den Informationsfluss des Virtual Carer Projekts.

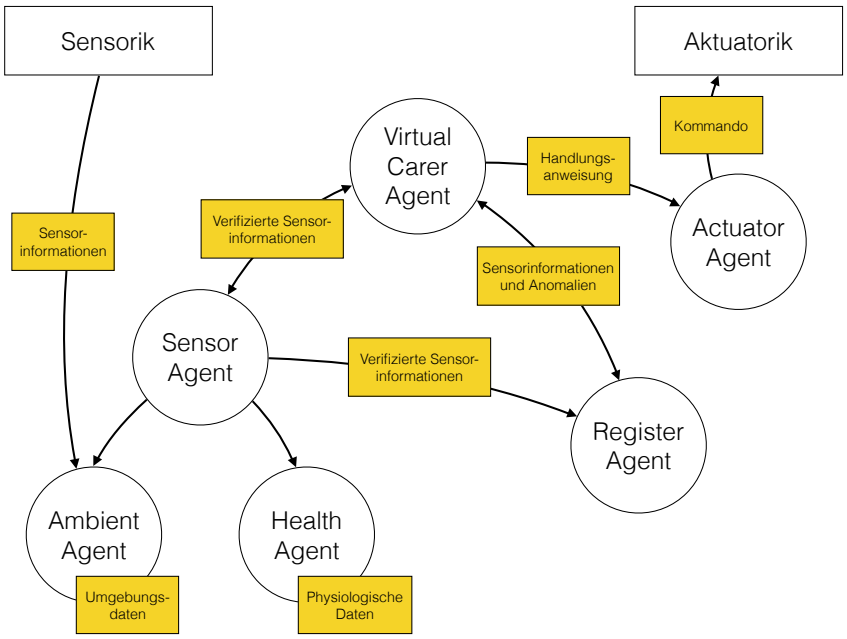


Abbildung 5.16: Überblick über den Informationsfluss innerhalb der Virtual Carer Architektur.

Das zum Einsatz kommende *Multiagentensystem* setzt sich aus den folgenden Komponenten zusammen:

- **Virtual Carer Agent:** Der *Virtual Carer Agent* ist die zentrale Komponente des Systems und wurde als *BDI Agent* (siehe Kapitel 4) implementiert. Die Informationen innerhalb des *Agenten* werden als logische Prädikate modelliert und können als dessen interne Wissensbasis angesehen werden. Die Aufgabe des *Virtual Carer Agents* ist es, die zur Benutzersituation passenden Aktionen innerhalb der Umgebung auszuführen. Seine Vorgehensweise kann als vierstufiger Prozess beschrieben werden: (1) Analyse der Sensorinformationen; (2) Ableitung von neuen Kontextinformationen und Aktualisierung der internen Wissensbasis; (3) Auswahl der zum Kontext passenden Handlungsanweisungen; (4) Information der *Actuator Agents* zur tatsächlichen Planausführung.

- **Sensor Agent:** Der *Sensor Agent* empfängt Umgebungs- und Benutzerdaten und überwacht diese im Hinblick auf signifikante Veränderungen. Im Falle einer relevanten Abweichung wird der *Virtual Carer Agent* über die Veränderung informiert. Es werden zwei Typen von *Sensor Agents* unterschieden. Der *Ambient Agent* ist für die Anbindung aller in der Umgebung verteilten Sensoren verantwortlich, wohingegen der *Health Agent* auf benutzerbezogene, physiologische Informationen beschränkt ist.

- **Register Agent:** Der *Register Agent* ist für das Speichern der Sensorinformationen sowie der vom *Virtual Carer* erkannten Anomalien ver-

antwortlich und stellt den historischen Verlauf der Ereignisse innerhalb der Umgebung zur Verfügung.

- **Actuator Agent:** Die Aufgabe des *Actuator Agent* ist es, die vom *Virtual Carer* gesendeten Handlungsanweisungen weiterzuverarbeiten und die *Aktuatoren* innerhalb der Umgebung entsprechend zu steuern.

Zur Implementierung der beschriebenen Architektur wurden zwei unterschiedliche *Agentenplattformen* verwendet. Für die Realisierung der *Sensor Agents* und *Actuator Agents* kommt JADE zum Einsat. Der *Register Agent* und der *Virtual Carer Agent* basieren auf JASON und AgentSpeak (siehe Kapitel 4).

5.5 Semantische Repräsentation von Intelligenten Umgebungen

Wahlster (2006, 2008) hat die hohe Relevanz und den Mehrwert einer Mensch-Maschine- und Maschine-zu-Maschine-Kommunikation hervorgehoben.

Die grundlegende Idee des semantischen Webs (Fensel et al., 2003) ist die inhaltliche Beschreibung und Auszeichnung digitaler Dokumente mit einem standardisierten Vokabular. Haupert (2013) definiert Auszeichnungssprachen, die eine formale Syntax und Semantik besitzen und in Form einer Ontologie eine standardisierte Begrifflichkeit zur Beschreibung digitaler Inhalte bereitstellen, als den Kern von semantischen Technologien. Eine Ontologie wird allgemein als eine Sammlung von Konzepten und deren Abhängigkeiten innerhalb einer bestimmten Wissensdomäne definiert (Gruber, 1995, 2009). In den folgenden Abschnitten wird eine Auswahl an verwandten Arbeiten auf dem Gebiet der semantischen Repräsentation von *Intelligenten Umgebungen* vorgestellt.

5.5.1 EHS Taxonomie

Als ein Vorläufer einer semantischen Repräsentation ohne explizite Nutzung einer Ontologiebeschreibungssprache ist die *European Home System* (EHS) Taxonomie[22] zu nennen. EHS unterscheidet grundsätzlich zwischen *Weißer Ware* (zum Beispiel Geräte zum Kochen und Waschen) und *Brauner Ware* (Unterhaltungselektronik). Die daraus resultierende Hierarchie setzt sich aus vier Hauptklassen zusammen:

- **Meter Reading:** Die *Meter Reading* Klasse umfasst alle Messwerkzeuge und Applikationen.

- **House Keeping:** Die *House Keeping* Klasse umfasst alle Haushaltsanwendungen.

[22] Die EHS Taxonomie wurde ursprünglich von der European Home Systems Association (EHSA) entwickelt, welche mittlerweile aber der KNX Association (www.knx.org) angehört.

- **Audio and Video:** Die *Audio and Video* Klasse umfasst die gesamte Unterhaltungselektronik und Multimediaanwendungen.
- **Telecommunication:** Die *Telecommunication* Klasse umfasst alle Kommunikationswerkzeuge.

Die EHS Taxonomie stellt ein einfach strukturierte Klassenhierarchie von häuslichen Anwendungen bereit. Sie umfasst hingegen keine Modelle zur Repräsentation der Zustände oder der Funktionalitäten einer spezifischen Geräteklasse.

5.5.2 SOUPA

Chen et al. (2005) beschreiben die sogenannte *Standard Ontology for Ubiquitous and Pervasive Computing* (SOUPA). SOUPA ist vollständig in der *Web Ontology Language* (OWL)[23] repräsentiert und besteht aus zwei unterschiedlichen aber verwandten Sammlungen von Ontologien:

- **SOUPA Core Ontologies:** Die *Core Ontologies* definieren ein Standardsatz von Konzepten und Vokabeln zur allgemeinen Beschreibung von *Pervasive Computing* Anwendungen. Hierzu gehörten zum Beispiel Ontologien zur Modellierung von persönlichen Kontaktinformationen eines Benutzers, der *Beliefs*, *Desires* und *Intentions* eines individuellen Agenten, der ausführbaren Aktionen, den zugrunde liegenden Regeln, der zeitlichen und räumlichen Beziehungen sowie von Ereignissen innerhalb der Umgebung.
- **SOUPA Extension Ontologies:** Die *Extension Ontologies* definieren zusätzliche Konzepte und ein Vokabular zum Erstellen von spezifischen Typen von Applikationen und geben Beispiele für Erweiterungspunkte bestehender Ontologien. Ein Beispiel für eine Erweiterung der *Core Ontology* ist die Bereitstellung von Konzepten zur Modellierung von Meetings, den damit verbundenen Zeitplänen sowie der eingeladenen Besprechungsteilnehmer.

SOUPA bedient sich vieler sprachlicher Konstrukte aus anderen, in der jeweiligen Domäne bereits etablierten Ontologien, welche mittels standardisierten OWL Abbildungen in die SOUPA Ontologie integriert werden. Zu ihnen gehören zum Beispiel Firend-Of-A-Friend (FOAF), DAML-Time, OpenCyc (Matuszek et al., 2006), Regional Connection Calculus (RCC), COBRA-ONT, MoGATU BDI und die *Rei policy* Ontologie.

5.5.3 DomoML

DomoML ist eine auf OWL basierende Auszeichnungssprache zur Definition von häuslichen Umgebungen (Furfari et al., 2004). Sie wurde erstmals im Rahmen des NICHE Projekts[24] entwickelt und eingesetzt. Das Ziel von DomoML

[23] http://www.w3.org/2001/sw/wiki/OWL [Letzter Zugriff: 09.04.2015]
[24] *Natural Interaction in Computerised Home Environment*

ist es, eine semantische Abstraktionsebene zu schaffen, um besonders die natürlichsprachliche Interaktion eines Benutzers mit seiner Umgebung zu ermöglichen (Sommaruga et al., 2005), aber auch die Kommunikation der Geräte untereinander sowie den Einsatz von neuartigen Mehrwertdiensten zu unterstützen. Sommaruga et al. (2011) stellen ein auf DomoML basierendes Framework zur Realisierung von intelligenten Anwendungen im *Smart Home* Kontext vor.

DomoML basiert auf drei unterschiedlichen Kernkomponenten:

- **DomoML-env (environment):** *DomoML-env* beschreibt alle vorkommenden Ressourcen innerhalb einer häuslichen Umgebung. Es wird zwischen Geräten und Objekten unterschieden. Die Geräteklasse beschreibt alle steuerbaren, aktiven *Sensoren* und *Aktuatoren*, wohingegen die Objektklasse alle inaktiven Objekte, wie zum Beispiel Möbel, zusammenfasst. *DomoML-env* umfasst neben Geräten und Objekten auch deren Beziehungen und Abhängigkeiten zueinander. So können zum Beispiel auch räumliche Beziehungen von Geräten und Objekten spezifiziert werden.

- **DomoML-fun (functional):** *DomoML-fun* beschreibt die Funktionalitäten und Verhaltensweisen der jeweiligen Ressourcen und bietet somit die Möglichkeit zur Komposition und Aggregation von Ressourcen. Auf diese Art und Weise erlaubt es *DomoML-fun*, die interne Logik des Gesamtsystems beziehungsweise der häuslichen Umgebung zu erfassen, zu beschreiben und schließlich zu verstehen.

- **DomoML-com (communication):** *DomoML-com* beinhaltet eine plattformunabhängige Beschreibung der Interaktionsmöglichkeiten mit den in *DomoML-env* definierten Ressourcen und stellt somit eine direkte Verknüpfung zwischen den Benutzern und den installierten *Smart Products* innerhalb der Umgebung her.

Die DomoML Ontologie wurde unter anderem im Forschungsgebiet *Ambient Assisted Living* (siehe Kapitel 2.4) erfolgreich eingesetzt, mit dem Ziel ältere Menschen in ihrer häuslichen Umgebung zu unterstützen (Sommaruga et al., 2011).

5.5.4 DogOnt

Bonino und Corno (2008) führen DogOnt als Modellierungssprache für sogenannte *Intelligent Domotic Environments* (IDE) ein. DogOnt bildet die semantische Grundlage des in Abschnitt 5.3.1 vorgestellten *Domotic OSGi Gateways*. Die Ontologie ist vollständig in OWL modelliert und setzt sich aus fünf Hauptbestandteilen zusammen:

- **Umgebungsmodellierung:** Zur Modellierung der Umgebung werden in DogOnt zwei grundlegende Konzepte unterschieden. *Building Environment* unterstützt die Modellierung einer allgemeinen Architekturbe-

schreibung der häuslichen Umgebung. *Building Thing* beschreibt hingegen alle Objekte, die sich innerhalb einer solchen Architektur befinden können. Es wird zwischen steuerbaren Objekten (*Controllable*) und nicht steuerbaren Objekten (*Uncontrollable)* unterschieden.

- **Gerätemodellierung:** Als Gerät werden alle steuerbaren Objekte (*Controllable*) bezeichnet. Sie unterscheiden sich von den nicht steuerbaren Objekten (*Uncontrollable*) dadurch, dass sie über definierte Funktionalitäten sowie über mögliche interne Zustände verfügen.

- **Funktionsmodellierung:** In DogOnt werden die Funktionalitäten unabhängig von der zugrunde liegenden Geräteklasse modelliert und erst im zweiten Schritt einem spezifischen Gerät zugeordnet. Die Funktionalitäten untergliedern sich wiederum in *Control Functionalities*, *Notification Functionalities* und *Query Functionalities*.

- **Zustandsmodellierung:** Die Zustände eines Gerätes werden ebenfalls separat modelliert und erst durch das jeweilige Gerät instanziiert. Zustände werden anhand ihrer zugehörigen Werte weiter untergliedert, wobei grundsätzlich zwischen *diskreten* und *kontinuierlichen* Zustandsveränderungen unterschieden wird.

- **Netzwerkmodellierung:** Die bisher vorgestellten Bestandteile sind unabhängig von der zugrunde liegenden technologischen Infrastruktur. Für den Fall, dass jedoch auch technologieabhängiges Wissen modelliert werden muss, bietet DogOnt einen eigenen Ontolgiezweig zur Modellierung des zugrunde liegenden Kommunikationsnetzwerks an.

Mit Hilfe von DogOnt kann also beschrieben werden, wo sich ein bestimmtes Gerät innerhalb einer *Intelligenten Umgebung* befindet, was seine spezifischen Funktionalitäten und Konfigurationsmöglichkeiten sind, was die technischen Voraussetzungen zur Interaktion sind und ob es räumliche Abhängigkeiten zueinander gibt.

5.5.5 SESAME

Das SESAME Projekt (Fensel et al., 2013) hat sich zum Ziel gesetzt, die Endnutzer eines *Smart Homes* bezüglich ihres Energieverbrauchs optimal zu unterstützen. SESAME-S kombiniert hierfür spezifische Smart Meter Daten mit Funktionalitäten zur Heimautomation. Zur Beschreibung eines energieoptimierten *Smart Homes* sowie der daran beteiligten *Smart Products* und Benutzer nutzt SESAME einen ontologiebasierten Modellierungsansatz. Die daraus resultierende SESAME Ontologie ist in OWL beschrieben und besteht aus drei Hauptkomponenten:

- **SESAME Automation Ontology:** Die *Automation Ontology* beinhaltet allgemeine Konzepte zur Modellierung des Bewohners und dessen

Aufenthaltsort sowie Konzepte zur Modellierung der Anwendungsdomänen Energiemanagement und Heimautomation. Hierzu werden Konzepte für verschiedene Geräte und deren Konfigurationen zur Verfügung gestellt.

- **SESAME Meter Data Ontology:** Die *Meter Data Ontology* bildet das Datenmodell eines Smart Meters sowie die entsprechenden Kommunikationsprotokolle auf ein semantisches Modell ab. Die *Meter Data Ontology* basiert in erster Linie auf dem DLMS Protokoll und der COSEM Spezifikation (IEC, 2014). Innerhalb der Ontologie werden drei Typen von Daten unterschieden: Energieinformationen, Konfigurations- und Zustandsdaten des zugrunde liegenden Smart Meters sowie Informationen über zeitliche Zusammenhänge.

- **SESAME Pricing Ontology:** Die *Pricing Ontology* modelliert alle energiewirtschaftlichen Informationen, die für die optimale Entscheidungsfindung beim Energiemanagement notwendig sind, wie zum Beispiel Tarifpläne, Erzeugerdaten und die Stromart. Die *Pricing Ontology* bildet folglich die Grundlage für das Erstellen von Plänen zur optimalen und energieeffizienten Heimautomation.

5.5.6 UbisWorld

Heckmann (2006) beschreibt UbisWorld als eine Erweiterung der Grundideen der Arbeiten *Blocks World* (Slaney und Thiébaux, 2001) und *Context Toolkit* (Salber et al., 1999). Der Kerngedanke hinter UbisWorld ist eine einheitliche Benutzer- und Interaktionsmodellierung in *Instrumentierten Umgebungen*. Mit Hilfe von *Situational Statements* kann der aktuelle Zustand der Umgebung in Form von auf Subjekt, Prädikat und Objekt basierten Aussagen definiert werden. Stahl (2009) stellt darauf aufbauend eine Designmethode für proaktive Benutzerassistenzsysteme in *Instrumentierten Umgebungen* vor. Sein Ansatz nutzt UbisWorld und beschreibt eine Erweiterung zur Modellierung von Benutzeraktivitäten.

UbisWorld setzt sich aus den folgenden sechs Teilontologien zusammen:

- **Physical Ontology:** Unter der *Physical Ontology* werden alle physischen Objekte, wie zum Beispiel Personen, Geräte, Möbel und Haushaltswaren, zusammengefasst. Innerhalb der *Physical Ontology* wird zwischen den Basiskonzepten *Being*, *Thing* und *System* sowie zwischen Gruppierungen dieser Basiselemente unterschieden.

- **Spatial Ontology:** Alle Elemente der *Physical Ontology* verfügen über eine bestimmte Position innerhalb ihrer Umgebung, was eine relevante Kontextinformation für intelligente Dienste und Interaktionsmechanismen darstellt. Die *Spatial Ontology* unterscheidet zwischen zwei Basiskonzepten. Das erste Konzept *Location* definiert eine Position innerhalb

einer Hierarchieebene oder in Relation zu einem anderen physischen Objekt. Das zweite Konzept *SpatialConstraint* setzt zwei *Location* Konzepte zueinander in Beziehung.

- **Temporal Ontology:** Zur Modellierung von zeitlichen Zusammenhängen untergliedert sich die *Temporal Ontology* in die beiden Konzepte *Time* und *TemporalConstraint&Chronology*. Ersteres beschreibt entweder einen expliziten Zeitpunkt, ein Zeitintervall oder eine Zeitspanne in Bezug zu einem bestimmten *Event*. Das *TemporalConstraint&Chronology* Konzept setzt wiederum *Time* Konzepte zueinander in Beziehung.

- **Activity Ontology:** Die *Activity Ontology* beschreibt dynamische Aspekte und Veränderungen der bisher eingeführten Konzepte. So stellt zum Beispiel die Veränderung der Position eines Objektes eine wichtige Veränderung der Umgebung dar. Auf diese Weise können in der *Activity Ontology* grundlegende Konzepte wie *Nehmen*, *Geben*, *Hinstellen* und *Verändern* modelliert werden.

- **Situation Ontology:** Die *Situation Ontology* umfasst alle Eigenschaften, Attribute und Parameter von physischen, räumlichen und zeitlichen Konzepten. Für eine räumliche Position in einem *Smart Home* spielen zum Beispiel Aspekte wie Umgebungslautstärke oder Beleuchtung eine wichtige Rolle, wohingegen für einen Autofahrer seine kognitive Last oder seine Erschöpfung entscheidende Einflussfaktoren darstellen.

- **Inference Ontology:** Das Ziel der *Inference Ontology* ist es, intelligente Regeln oder proaktive Prozesse darzustellen, um letztendlich Interaktions- und Assistenzkonzepte innerhalb der Umgebung zu realisieren.

UbisWorld verfügt neben dem räumlichen Editor YAMAMOTO (Stahl, 2009; Heckmann, 2006) über zahlreiche Werkzeuge zum Durchsuchen, Auffinden, Bearbeiten und Visualisieren der ontologischen Konzepte. Loskyll et al. (2009) stellen zum Beispiel mit dem UbisEditor ein Konzept zur kollaborativen Entwicklung der ontologischen Konzepte zur Verfügung. Es werden darüber hinaus Möglichkeiten zum Import und Export von OWL-basierten Ontologien angeboten.

5.5.7 COSE

Das CASAS[25] *Smart Home* Projekt ist ein multidisziplinäres Forschungsprojekt der Washington State University mit Fokus auf der Gestaltung von *Intelligenten Umgebungen*. In CASAS wird das gesamte *Smart Home* als ein intelligenter *Agent* betrachtet, der durch *Sensoren* seine Umwelt wahrnehmen und diese durch *Aktuatoren* wiederum beeinflussen kann. Die Arbeiten konzentrieren sich auf die Erkennung von Benutzeraktivitäten und gliedern sich primär in die drei Bereiche Aktivitätsdetektion, Aktivitätserkennung und Aktivitätsvorhersage. Als semantische Grundlage des Ansatzes stellen Wemlinger und

[25] *Center for Advanced Studies in Adaptive Systems*

Holder (2011) die *Casas Ontology for Smart Environments* (COSE) vor. COSE verfolgt den Ansatz, ausgehend von einer sogenannten *upper-level* Ontologie eine eigenständige und domänenspezifische Ontologie für *Intelligente Umgebungen* zu definieren. Die grundlegenden Konzepte innerhalb von COSE sind Gebäude, Bewohner, Sensoren und Benutzeraktivitäten. Über eine Mapping Relation können die spezielleren Konzepte mit Konzepten aus der *upper-level* Ontologie OpenCyc (Matuszek et al., 2006) verknüpft und ineinander überführt werden. COSE ist vollständig in OWL modelliert. Eine weiterführende Anwendung von COSE ist zum Beispiel die Beschreibung von Aktivitäten des täglichen Lebens (Reisberg et al., 2001).

5.6 Bewertung und Schlussfolgerungen

In diesem Abschnitt werden die verwandten Arbeiten aus den vorgestellten Bereichen miteinander verglichen und die Vor- und Nachteile gegenüber der vorliegenden Arbeit dargestellt. Die Schlussfolgerungen orientieren sich an den in Abschnitt 5.2 eingeführten Anforderungen und Bewertungskriterien.

5.6.1 Fazit: Middlewareplattformen für Intelligente Umgebungen

In Abschnitt 5.3 wurden verschiedene Middlewareplattformen für *Intelligente Umgebungen* vorgestellt. Die beschriebenen Ansätze werden nun anhand den Anforderungen bezüglich der zugrunde liegenden Plattformarchitektur (A_P), der Benutzerinteraktion (A_B) und der Werkzeugunterstützung (A_W) bewertet. Mit DOG und EBS wurden zwei *Middlewareplattformen* aus dem Forschungsumfeld beschrieben. Die Stärken beider Ansätze liegen zum einen auf der modularen und flexiblen Architektur und zum anderen auf der Verwendung einer semantischen Beschreibung der internen Datenmodelle. Der Schwachpunkt beider Plattformen ist das Fehlen einer vollständig in den Entwicklungsprozess integrierten Werkzeugsammlung. Diese Lücke wird durch openHAB und dessen Weiterentwicklung Eclipse SmartHome geschlossen. Beide Projekte enthalten eine umfassende auf Eclipse basierende Werkzeugkette zur Weiterentwicklung, zum Betrieb und zur Wartung der jeweiligen Anwendungen. OpenHAB ist zudem zu einer Vielzahl von *Smart Home* Technologien kompatibel und stellt die entwickelten *Bindings* auf der Projekthomepage zum Herunterladen bereit[26]. Mit Nest und HomeKit wurden zudem zwei Beispiele vorgestellt, wie Google und Apple aktuell versuchen in den *Smart Home* Markt einzutreten. Beide Projekte verfolgen zwar einen produktreifen aber doch sehr einfachen und eingeschränkten Ansatz. So liegt der Fokus bei beiden Plattformen auf der Steuerung und Verwaltung der im *Smart Home* installierten Geräte. Wohingegen Google mit Nest eine webbasierte Lösung anbietet, setzt Apple mit HomeKit auf die Bereitstellung einer einheitlichen Programmierschnittstelle für mobile Anwendungen. Keine der vorgestellten Arbeiten bietet jedoch eine

[26] http://www.openhab.org/features-tech.html [Letzter Zugriff: 30.01.2015]

direkte Unterstützung von Mehrwertdiensten in Form von *Smart Services* sowie ein standardisiertes Konzept zur Modellierung und Beschreibung von personalisierten Benutzerschnittstellen. Mit Ausnahme von HomeKit wird auch die Möglichkeit zur Etablierung eines globalen Marktplatzes für *Smart Home* Anwendungen von keinem der Ansätze berücksichtigt. Alle drei Defizite sollen im Rahmen der vorliegenden Arbeit adressiert und überwunden werden. In Tabelle 5.3 wird eine abschließende Übersicht über die Ergebnisse des funktionalen Vergleichs der einzelnen Ansätze gegeben.

Arbeiten	Dynamische Dienstintegration	Semantische Dienstinteroperabilität	Skalierbarkeit	Standards	Open Source	Marktplatzportal	Personalisierte Benutzerschnittstellen	Entwicklungswerkzeuge	Management, Betrieb und Wartung
DOG	(✓)	✓	-	(✓)	✓	-	-	-	✓
EBS	(✓)	✓	✓	(✓)	-	-	-	(✓)	✓
openHab	(✓)	(✓)	-	-	(✓)	-	(✓)	(✓)	(✓)
SmartHome	(✓)	(✓)	(✓)	(✓)	✓	-	(✓)	✓	✓
Nest	-	-	✓	-	-	-	(✓)	✓	✓
HomeKit	-	-	-	-	-	✓	(✓)	✓	✓
UCH	(✓)	(✓)	-	✓	-	(✓)	✓	(✓)	(✓)
ASaP	✓	✓	✓	✓	✓	✓	✓	✓	✓

✓: Kriterium erfüllt (✓): Kriterium teilweise erfüllt -: Kriterium nicht erfüllt

Tabelle 5.3: Bewertung und Vergleich mit verwandten Arbeiten im Kontext von Middlewareplattformen für *Intelligente Umgebungen*.

5.6.2 Fazit: Multiagentensysteme in Intelligenten Umgebungen

In Abschnitt 5.4 wurden verschiedene Beispiele für den Einsatz von *Multiagentensystemen* zur Realisierung von *Intelligenten Umgebungen* vorgestellt. Die beschriebenen Arbeiten umfassen hierbei unterschiedliche Anwendungsszenarien, vom Energiesparen in Bürogebäuden (ISES) bis zur Verbesserung der medizinischen Betreuung von Patienten (Home Health Care System). Das Mav-

Home System betrachtet die Umgebung als einen rational handelnden *Agenten*, der die Umgebung wahrnimmt und basierend auf den Beobachtungen Vorhersagen über zukünftige Zustände macht. Das Ziel ist hierbei, den Komfort der Benutzer zu maximieren und gleichzeitig die Betriebskosten zu minimieren. Mit iDorm und Coalaa wurde eine Kombination aus eingebetteten *Softwareagenten* und mobilen Servicerobotern realisiert, um den Benutzer innerhalb seiner *Smart Home* Umgebung zu unterstützen. CoBrA verfolgt einen zentralen Ansatz, um Performanz und Skalierbarkeit des Systems zu gewährleisten. Durch den Einsatz eines Zentralrechners können *Agenten* ihre eigenen rechenintensiven Prozesse auslagern und somit die verfügbaren Ressourcen optimal ausnutzen. Ein zentraler Aspekt aller Ansätze ist die Aufteilung des zugrunde liegenden *Multiagentensystems* in *Agenten* mit klar definierten Verantwortungsbereichen. So werden beispielsweise *Agenten* zur Anbindung von *Sensoren* und *Aktuatoren* oder zur Realisierung intelligenter Analyse- und Assistenzsysteme spezifiziert (Nocturnal, Virtual Carer). Ein wesentlicher Schwachpunkt der verwandten Arbeiten in Bezug auf die in Kapitel 5.2 eingeführten Anforderungen ist die starke Fokussierung auf ein bestimmtes Anwendungsszenario. Die Ansätze stellen für den jeweiligen Anwendungsfall entsprechende intelligente und lernende Systeme zur Verfügung, bieten allerdings keine oder nur eingeschränkte Möglichkeit, das zugrunde liegenden System um beliebige weitere digitale Mehrwertdienste (*Smart Services*) zu erweitern. Wie auch bei den *Middlewareplattformen* fehlt zudem die Möglichkeit, personalisierte Benutzerschnittstellen bereitzustellen. Auch das Fehlen eines einheitlichen Marktplatzkonzeptes sowie die mangelnde Werkzeugunterstützung motivieren die Entwicklung der in den folgenden Kapiteln beschriebenen Integrationsplattform für *Smart Services* in *Intelligenten Umgebungen*. Die realisierte *Agentenbasierte Smart Service Plattform* (ASaP) abstrahiert hierbei von konkreten intelligenten Diensten und stellt somit ein allgemeines Framework zur Verfügung, auf dessen Basis unterschiedliche Anwendungsfälle umgesetzt werden können. Durch die Einbindung einer integrierten Entwicklungsumgebung und eines beispielhaften *Smart Service Marktplatzes* wird der komplette Entwicklungsprozess von *Smart Services* unterstützt.

5.6.3 Fazit: Semantische Repräsentation von Intelligenten Umgebungen

In Abschnitt 5.5 wurden verschiedene Ansätze zur semantischen Modellierung von *Intelligenten Umgebungen* vorgestellt. Die beschriebenen Ansätze werden nun anhand den Anforderungen bezüglich ihrer semantischen Struktur und ihrer Beschreibungsmächtigkeit (A_S) bewertet. Alle beschriebenen Ansätze unterstützen eine Repräsentation der in einer Umgebung installierten *Smart Products* sowie der damit verbunden Zustandsveränderungen. Des Weiteren ermöglichen, mit Ausnahme der EHS Taxonomie, alle Ansätze die Modellierung der räumlichen Umgebung und der darin enthaltenen Beziehungen. SOUPA, DomoML, UbisWorld und COSE beinhalten zudem Modelle zur Repräsentation der Aktivitäten des Benutzers. DomoML verfügt als einzige Arbeit zusätz-

lich über allgemeine Modelle zur Beschreibung der zur Verfügung stehenden Benutzerassistenz. Die Schwachpunkte aller vorgestellten Arbeiten ist das Fehlen einer einheitlichen Abbildung von beliebigen Mehrwertdiensten innerhalb einer *Intelligenten Umgebung*. Außer UbisWorld bietet keine der beschriebenen semantischen Konzepte eine durchgängige und in den Entwicklungsprozess von *Intelligenten Umgebungen* integrierte Werkzeugunterstützung. In Tabelle 5.4

Arbeiten	Modellierung von Smart Spaces	Modellierung von Smart Products	Modellierung von Smart Services	Modellierung von Zustandsveränderungen	Modellierung von Benutzerassistenz	Modellierung von Benutzeraktivitäten	Integration in Entwicklungsumgebung
EHS	-	✓	-	-	-	-	-
SOUPA	✓	✓	-	✓	-	✓	-
DomoML	✓	✓	(✓)	✓	✓	✓	-
DogOnt	✓	✓	-	✓	-	-	-
SESAME	✓	(✓)	(✓)	✓	-	-	(✓)
UbisWorld	✓	✓	-	✓	-	✓	✓
COSE	✓	✓	-	✓	-	✓	-
ASaP-Ontology	✓	✓	✓	✓	✓	✓	✓

✓: Kriterium erfüllt (✓): Kriterium teilweise erfüllt -: Kriterium nicht erfüllt

Tabelle 5.4: Bewertung und Vergleich mit verwandten Arbeiten im Kontext von semantischer Repräsentation von *Intelligenten Umgebungen*.

wird eine abschließende Übersicht über die Ergebnisse des funktionalen Vergleichs der einzelnen Ansätze gegeben. Zusammenfassend kann festgehalten werden, dass keine der vorgestellten Ansätze alle in Abschnitt 5.2 eingeführten Anforderungen erfüllt. Durch die im Rahmen der vorliegenden Arbeit entwickelten Plattformkonzepte wird im Folgenden eine Lösung angestrebt, welche die Bewertungskriterien vollständig erfüllt. Im folgenden Kapitel wird zunächst eine auf dem *Universal Remote Console* Ansatz basierende Implementierung des *Universal Control Hubs* (UCH) vorgestellt. Der Fokus liegt dabei auf der Realisierung von personalisierten Benutzerschnittstellen. In Kapitel 7 und 9 wird mit der *Agentenbasierten Smart Service Plattform* (ASaP) ein einheit-

liches Plattformkonzept zur Integration von *Smart Services* innerhalb einer *Intelligenten Umgebung* eingeführt. Kapitel 8 stellt mit der *ASaP-Ontology* eine allgemeine semantische Repräsentation zur Verfügung, welche die zuvor definierten Anforderungen an die semantische Struktur erfüllt. Tabelle 5.5 fasst die offenen Punkte zusammen und zeigt, wie die vorliegende Arbeit die einzelnen wissenschaftlichen Fragestellungen adressiert.

Kapitel	**Fragestellung**
Kapitel 6	Wie können personalisierte Benutzerschnittstellen einheitlich in eine *Intelligente Umgebung* integriert werden?
Kapitel 7	Wie können *Smart Services* anhand eines einheitlichen Plattformkonzeptes innerhalb *Intelligenter Umgebungen* integriert werden?
Kapitel 8	Wie können eine *Intelligente Umgebung* und die relevanten Objekte, Dienste und Beziehungen untereinander modelliert und in eine durchgängige Werkzeugkette integriert werden?
Kapitel 9	Wie sieht eine praktische Realisierung des in Kapitel 7 eingeführten theoretischen Plattformkonzeptes aus?
Kapitel 10	Wie kann eine durchgängige Werkzeugkette realisiert werden, die sowohl Softwareentwickler, Dienstanbieter als auch Endnutzer optimal unterstützt, aber auch Möglichkeiten zu späteren Erweiterungen bereitstellt?

Tabelle 5.5: Übersicht über die wissenschaftlichen Fragestellungen und die Strukturierung der vorliegenden Arbeit.

Teil II

Konzeption

Jede Lösung eines Problems ist ein neues Problem.

Johann Wolfgang von Goethe

6 Universal Remote Console

6.1 Einleitung

Die Realisierung von intelligenten Assistenzsystemen innerhalb einer *Intelligenten Umgebung* erfordert eine Möglichkeit, mit dem Benutzer zu interagieren. Hierbei soll dem Benutzer eine personalisierte, transparente und allgegenwärtige Sicht auf seine Umgebung geboten werden. Er soll somit zum Beispiel dazu befähigt werden, den Zustand einzelner Geräte zu überwachen und zu steuern, oder über Fehlfunktionen und Gefahren informiert zu werden. Zimmermann und Vanderheiden (2010) betonen hierbei die Notwendigkeit von individuellen und auf den konkreten Benutzer angepassten Benutzerschnittstellen und skizzieren ein Szenario für ältere Menschen, die sich beispielsweise beim Austausch des Fernsehers oder bei der Fernsehnutzung in fremden Umgebungen nur schwer auf die jeweilige Bedienung anpassen können. Durch die innerhalb der *Universal Remote Console* (URC) definierten abstrakten Schnittstellenbeschreibungen wird es möglich, die bekannten und auf die persönlichen Bedürfnisse angepassten Bedienkonzepte auch in solchen Anwendungsfällen universell zu nutzen. Dieser Ansatz folgt dem in Zimmermann et al. (2010a) vorgestellten Konzept der *Pluggable User Interfaces* und unterstützt die dynamische Nachrüstbarkeit und Austauschbarkeit von Softwareressourcen.

In Kapitel 5 wurden bereits verschiedene existierende Ansätze zur Realisierung von Middlewareplattformen in *Instrumentierten Umgebungen* vorgestellt. Dieser Abschnitt gibt einen detaillierteren Einblick in eine im Rahmen der vorliegenden Arbeit realisierten Implementierung der in OpenURC (siehe Abschnitt 6.3) vorgeschlagenen Plattforminfrastruktur. Kernkomponente der URC Technologie ist der sogenannte *Universal Control Hub* (UCH) (Zimmermann und Vanderheiden, 2007), eine *Middleware*, die beliebige, vernetzte Geräte und Dienste mit personalisierten Benutzeroberflächen verbindet und somit dem Endbenutzer ermöglicht, alle ihn umgebenden *Smart Products* durch eine auf ihn abgestimmte Schnittstelle zu überwachen und zu steuern. Ein *Resource Server* dient als globaler Marktplatz für den Vertrieb von Softwaremodulen und Benutzeroberflächen.

In dem vorliegenden Kapitel wird eine allgemeine Einführung in die URC Welt gegeben sowie die Entwicklungsgeschichte von URC und des zugrunde liegenden ISO Standards skizziert. Im Abschnitt 6.5 wird die im Rahmen dieser Arbeit entwickelte Version der *UCH-Middleware* detailliert vorgestellt, welche

auf die Realisierung personalisierter und barrierefreier Benutzerschnittstellen sowie auf die Integration der umgebenden technischen Infrastruktur ausgerichtet ist. Abschließend werden in Abschnitt 6.6 einige im Rahmen dieser Arbeit realisierten Anwendungsszenarios vorgestellt.

6.2 Entwicklungsgeschichte von URC

2005 formierte sich das internationale URC Konsortium aus Mitgliedern von Industrie und Forschung[1] mit dem Ziel, erstmalig den URC Standard zu entwickeln und voranzutreiben. Kurz nach der Gründung erfolgte die erste Veröffentlichung des URC Standards (ANSI/INCITS 389-2005 bis 392-2005). Die erste Referenzimplementierung des Universal Control Hub wurde 2006 von dem Trace R&D Center an der Universität Wisconsin fertiggestellt. Mit dem von der Europäischen Union geförderten Projekt i2home[2] wurde 2006 die URC Technologie erstmals im europäischen Forschungsraum eingeführt. Insgesamt waren neun Projektpartner aus fünf Ländern an i2home beteiligt, mit dem übergeordneten Ziel, Menschen mit leichten kognitiven Behinderungen sowie älteren Menschen einen intuitiven Zugang zur Haushaltselektronik zu ermöglichen (Frey et al., 2010c). Während der Projektlaufzeit wurde eine ISO Zertifizierung des URC Ansatzes angestrebt, welche 2008 mit der Veröffentlichung des ISO/IEC 24752 Standards erfolgreich abgeschlossen wurde.

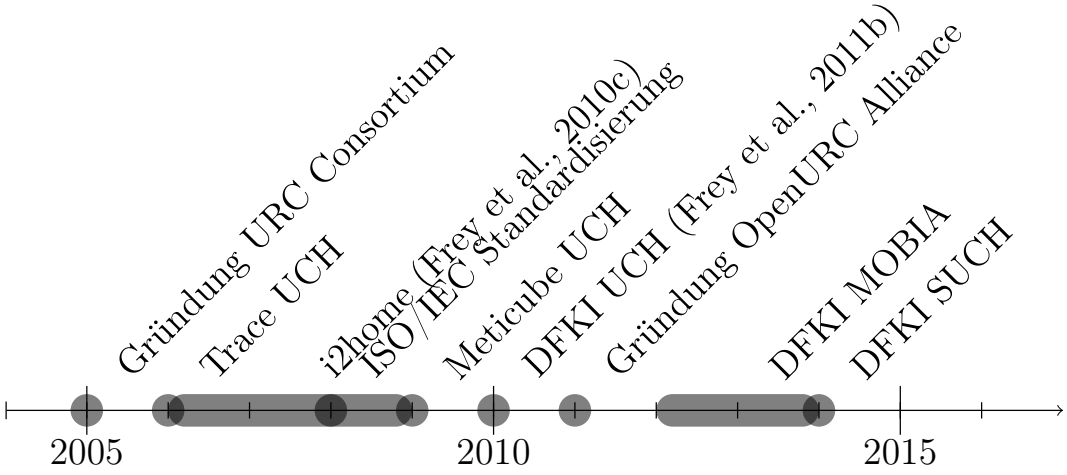


Abbildung 6.1: Überblick über die Entwicklungsgeschichte der *Universal Remote Console*.

Anfang 2009 veröffentlichte der i2home Partner Meticube eine erste kommerzielle Implementierung des UCH für den Einsatz in industriellen Umgebungen. Durch Erweiterungspakete der i2home Partner konnte die Funktionalität

[1] Gründungsmitglieder: Access Technologies Group, Deutschland; DFKI (Deutsches Forschungszentrum für Künstliche Intelligenz GmbH), Deutschland; dot UI, USA; Meticube, Portugal; Plenar Technologies, Indien; Technosite, Spanien und Belgien; University of Wisconsin-Madison, USA; Visual Communication Technologies (VICOMTech), Spanien

[2] www.i2home.org [Letzter Zugriff: 10.01.2015]

des UCH kontinuierlich erweitert werden. 2010 wurde im Rahmen der vorliegenden Arbeit die vorgestellte UCH-Implementierung (Frey et al., 2011b), auf deren Basis zahlreiche Forschungs- und Industrieprojekte realisiert wurden (siehe Kapitel 11). Auf Einzelheiten der Implementierung wird detailliert in Abschnitt 6.5 eingegangen. Die beschriebenen Entwicklungen mündeten 2011 in der Gründung der OpenURC Alliance (Alexandersson et al., 2011) als eingetragener Verein. Abbildung 6.1 fasst die Entwicklungsgeschichte von URC anhand der für diese Arbeit relevanten Ereignisse nochmals zusammen. Zum Ende des dritten Quartals 2011 wuchs die Gesamtsumme der Forschungs- und Entwicklungsprojekte sowie der Produktentwicklungen unter Verwendung des URC Standards auf mehr als 110 Mio. Euro (siehe Abbildung 6.2). Insgesamt waren zu diesem Zeitpunkt mehr als 200 Forschungs- und Entwicklungsorganisationen beteiligt, die URC Konzepte und Technologien in Bereichen wie digitale Barrierefreiheit, *Ambient Assisted Living*, e-Health, Heimautomation, Telekommunikation, Energiemanagement, Mobilität, Autoindustrie, Soziale Netzwerke, Content- und Medienmanagement und öffentlicher Verkehr anwenden [3].

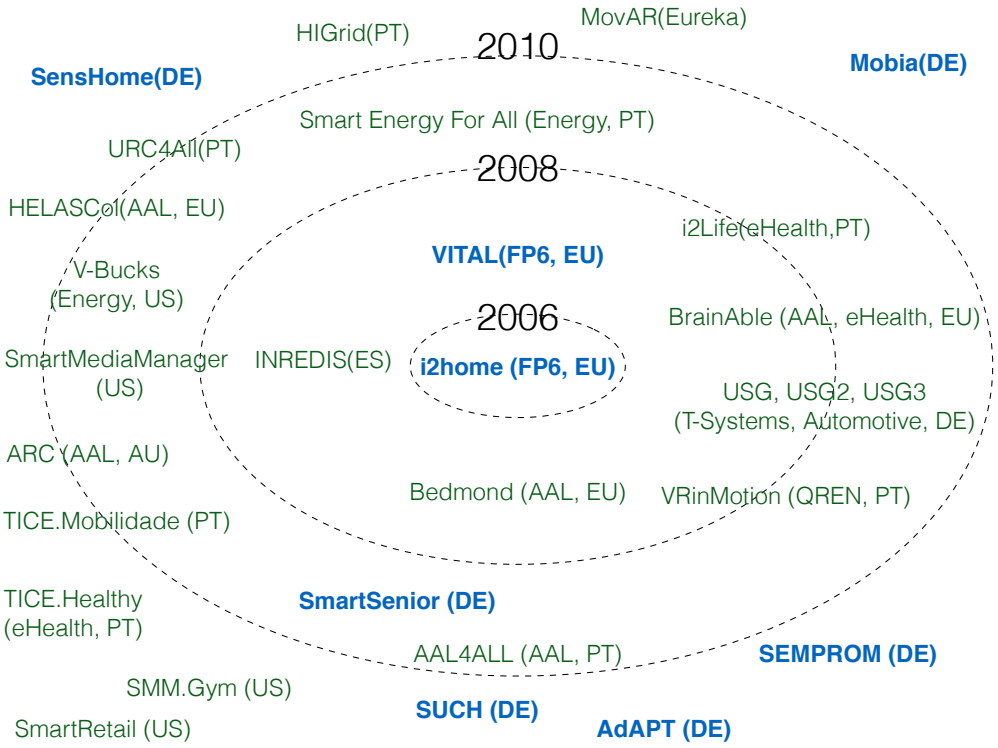


Abbildung 6.2: Übersicht über die Verbreitung von URC Technologie in internationalen Forschungs- und Industrieprojekten (die blau markierten Projekte sind hierbei Projekte, welche unter Beteiligung des DFKI durchgeführt wurden und teilweise auf der in dieser Arbeit vorgestellten UCH-Implementierung basieren).

[3] http://www.openurc.org [Letzter Zugriff: 31.01.2015]

6.3 URC Standard & OpenURC Alliance

Der ISO/IEC 24752 Standard *Universal Remote Console* (URC) stellt eine offene und standardisierte Plattform für personalisierte und austauschbare Mensch-Maschine-Benutzerschnittstellen dar, die sowohl Interoperabilität als auch benutzerzentriertes Design unterstützt. Der 2008 durch ISO[4] und IEC[5] veröffentlichte Standard umfasst insgesamt fünf Teile:

- Teil 1: Framework (ISO, 2008a)
- Teil 2: User Interface Socket Description (ISO, 2008b)iec
- Teil 3: Presentation Templates (ISO, 2008c)
- Teil 4: Target Description (ISO, 2008d)
- Teil 5: Resource Description (ISO, 2008e)

Um den internationalen Interessen, den technologischen Weiterentwicklungen und sich ändernden Anforderungen besser Rechnung tragen zu können, wurde 2011 die OpenURC Alliance (Alexandersson et al., 2011) ins Leben gerufen. Das OpenURC-Konsortium gliedert sich in vier Arbeitsgruppen[6]:

Arbeitsgruppe 1 (Technical): In der technischen Arbeitsgruppe wird die zugrunde liegende URC Technologie kontinuierlich weiterentwickelt und auf aktuelle Gegebenheiten angepasst. Hierzu werden notwendige Dokumente erstellt oder zusätzliche Softwarekomponenten entwickelt.

Arbeitsgruppe 2 (User): Diese Gruppe fokussiert auf Anforderungen und Bedürfnisse der Endnutzer. Hierzu werden anhand sogenannter Persona entsprechende Szenarien entwickelt, welche die Grundlagen für einen benutzerzentrierten Entwicklungsprozess darstellen.

Arbeitsgruppe 3 (Business): Diese Gruppe verantwortet die Außendarstellung der OpenURC Alliance und die Vermarktung der Technologie.

Arbeitsgruppe 4 (Governance): Das *Governance Board* ist das Verwaltungsorgan der OpenURC Alliance und steuert somit die Gesamtentwicklung.

Die Weiterentwicklung des URC Standards sowie die Bereitstellung von Implementierungen, Werkzeugen und Anleitungen wird von der OpenURC Alliance koordiniert. Die OpenURC Alliance ist ein in Deutschland eingetragener Verein und ist offen für weitere Mitglieder. Die Mitgliedschaft ist hierarchisch in verschiedene Kategorien strukturiert. Jede Kategorie umfasst hierbei

[4] *International Organization for Standardization*
[5] *International Electrotechnical Commission*
[6] http://www.aal-kompetenz.de/cms/index.php/plattformen/urc [Letzter Zugriff: 31.01.2015]

unterschiedliche Rechte und Pflichten, zum Beispiel *Core*, *Charter*, *Associate* und *Basic*. Ziel von OpenURC ist es, Forschungs- und Entwicklungsarbeiten über personalisierte und barrierefreie Benutzerschnittstellen mit anderen Aktivitäten im Forschungsbereich *Ambient Assisted Living* zu koordinieren. Die OpenURC Alliance ist ein Forum, in dem verschiedene Interessengruppen zusammenkommen und Ideen und Ergebnisse im Bereich der Forschung, Entwicklung, Bildung, Zertifizierung und Normierung austauschen können.

6.4 Universal Control Hub (UCH)

Die zugrunde liegende Vision von OpenURC ist die Realisierung einer personalisierten und barrierefreien Bedienung von Geräten und Diensten (*Targets*) innerhalb *Instrumentierter Umgebungen*. Es existieren bereits mehrere Produkte, welche durch neuartige Netzwerktechnologien in bestehende Umgebungen integriert werden können. In vielen Fällen existieren zu den Produkten proprietäre Mechanismen zur dynamischen Erkennung und Steuerung. Themen wie Interoperabilität von Produkten unterschiedlicher Hersteller sowie die Erstellung personalisierter, barrierefreier und intuitiver Benutzerschnittstellen werden oftmals nur wenig berücksichtigt. Zur Unterstützung der Integration unterschiedlicher *Targets* verfolgt die OpenURC Alliance mit dem *Universal Control Hub* (UCH) einen zentralen Middlewareansatz (siehe Kapitel 5.3). Aufgrund einer fehlenden einheitlichen Schnittstellenbeschreibung ist die Entwicklung von personalisierten Benutzerschnittstellen oftmals mit erheblichem Aufwand verbunden. URC beinhaltet eine standardisierte und flexible Schnittstellenbeschreibungssprache, den sogenannten *User Interface Socket* (UIS). Der UIS bietet eine abstrakte Beschreibung aller verfügbaren Funktionalitäten der *Targets* und stellt somit eine verbindlichen Interaktionsabsprache zwischen der konkreten Geräte- und Dienstebene und der Benutzerschnittstellenebene dar. Die UIS-Spezifikation unterstützt eine unbegrenzte Zahl von Eingabemodalitäten, von touchbasierten Smartphones über Sprachsteuerung bis hin zu *Brain Computer Interaction* (BCI). Ein wesentlicher Vorteil dieses Ansatzes ist, dass sich Entwickler von Benutzerschnittstellen voll und ganz auf das Design und die Bedienungslogik ihrer Schnittstelle konzentrieren können.

6.4.1 Architektur

URC kann grundsätzlich als eine Kombination von Hard- und Software betrachtet werden. Diese erlaubt es dem Endbenutzer über beliebige personalisierte Benutzerschnittstellen (*Controller*) eine Vielzahl von herstellerübergreifenden Geräten und Diensten (*Targets*) einheitlich anzusteuern, zu überwachen und zu kontrollieren. Zentraler Bestandteil der Architektur ist UCH. Im Zusammenspiel mit einem oder mehreren *Resource Servern* können die benötigten Ressourcen gesammelt und durch einen einheitlichen Marktplatz von den jeweiligen Endkunden bezogen werden. Smirek et al. (2014) geben einen kompakten Überblick über die technologischen Voraussetzungen zur Realisierung eines URC Ökosystems.

Abbildung 6.3 fasst hierzu die grundlegenden Komponenten der vorgeschlagenen URC Architektur zusammen.

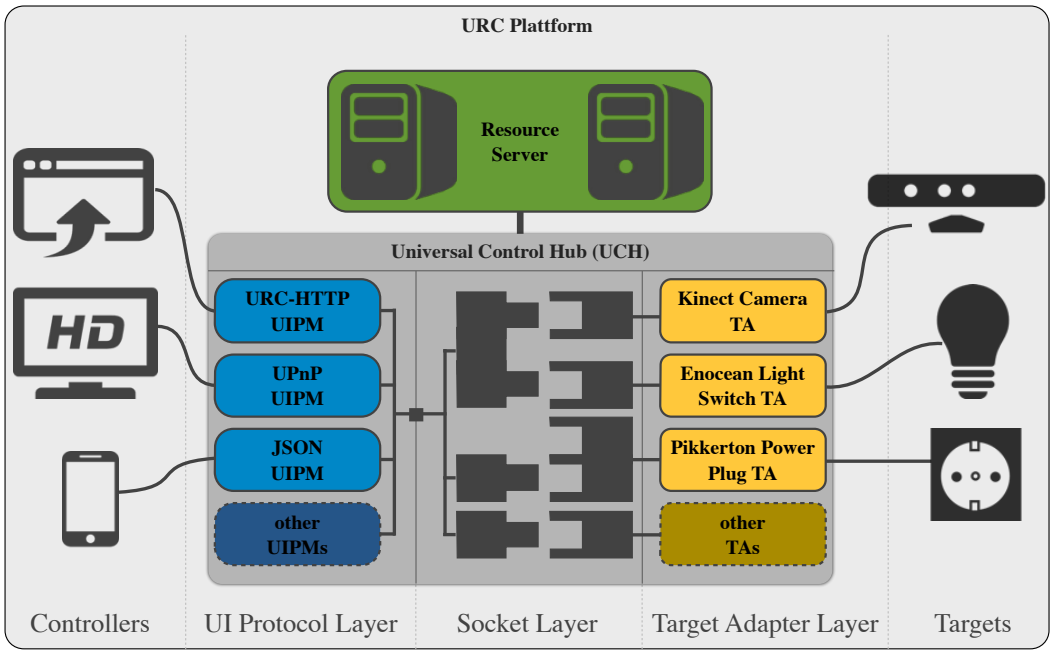


Abbildung 6.3: Übersicht über die URC Architektur bestehend aus einem zentralen UCH, welcher sich wiederum aus drei dedizierten Schichten zusammensetzt: *Target Adapter Layer*, *Socket Layer* und *User Interface Protocol Layer*.

Der *Universal Control Hub* ist als *Middleware* Komponente zentraler Bestandteil einer URC Architektur und setzt sich aus drei dedizierten und modular aufgebauten Schichten zusammen:

- **Target Adapter Layer:** Der *Target Adapter Layer* ist verantwortlich für die Anbindung der zu steuernden *Targets*, also aller Geräte, Objekte und Services innerhalb der Umgebung.
- **Socket Layer:** Der *Socket Layer* besteht aus einer Menge verschiedener *User Interface Sockets* (UIS) und definiert somit eine abstrakte Beschreibungssprache der einzelnen *Target* Funktionalitäten.
- **User Interface Protocol Layer:** Der *User Interface Protocol Layer* bietet letztlich durch den Einsatz von *User Interface Protocol Modules* (UIPM) die Möglichkeit, beliebige Kommunikationsschnittstellen zu integrieren und somit verschiedene konkrete Benutzerschnittstellen anzubinden.

Ein maßgeblicher Vorteil der UCH-Architektur ist der modulare Aufbau und folglich die leichte Erweiterbarkeit durch neue Softwarekomponenten auf jedem der einzelnen Schichten (Smirek et al., 2014). Die folgenden Abschnitte geben einen Überblick über die einzelnen Schichten und ihre zugehörigen Komponenten.

6.4.2 Target Adapter Layer

Der *Target Adapter Layer* bildet die kommunikationstechnische Grundlage für die Integration der *Sensoren* und *Aktuatoren* innerhalb einer *Instrumentierten Umgebung*. Dies beinhaltet einfache *Sensoren*, wie Bewegungsmelder, Türkontakte und Rauchmelder, komplexere Haushaltsgeräte, wie Fernseher, Waschmaschine und Heizungssteuerung, aber auch webbasierte Dienste, wie Wetterprognose oder einen digitalen Terminkalender. Die steigende Vielfalt der verfügbaren *Targets* bringt auch eine Vielzahl unterschiedlicher Kommunikationstechnologien und Netzwerkprotokolle mit sich.

Definition 27 (*Target*)

Als Targets werden alle vernetzten Geräte, Objekte, Sensoren und Aktuatoren sowie webbasierte Dienste innerhalb einer definierten Instrumentierten Umgebung bezeichnet, welche durch die UCH-Architektur integriert und durch einen entsprechenden Controller angesteuert werden können.

Um der Vielfalt der verfügbaren *Targets* Rechnung zu tragen bietet der *Target Adapter Layer* die Möglichkeit, neue Technologien in Form eines modularen *Target Adapters* in die UCH-Plattform zu integrieren. Ein *Target Adapter* ist eine modulare Softwarekomponente innerhalb der UCH-Infrastruktur und verwaltet die benötigte Kommunikationsinfrastruktur für die Anbindung eines spezifischen oder einer Menge von gleichartigen *Targets*. Der *Target Adapter* kann hierzu unterschiedliche Netzwerkprotokolle unterstützen. Seine Aufgabe ist es, alle von der *Target* Applikation bereitgestellten Zustände auszulesen und die verfügbaren Funktionalitäten anzusprechen.

Definition 28 (*Target Adapter*)

Ein Target Adapter ist eine modulare Softwarekomponente innerhalb der UCH-Infrastruktur und verwaltet die benötigte Kommunikationsinfrastruktur für die Anbindung eines spezifischen oder eine Menge von gleichartigen Targets.

Durch einen *Target Adapter* wird also eine konkrete, hardware- und netzwerkfähige Schnittstelle zu der verfügbaren Instrumentierung innerhalb der Umgebung geschaffen. Zur Realisierung einer universellen sowie vor allem persönlichen und barrierefreien Schnittstelle wird im nächsten Abschnitt eine allgemeine Abstraktionsebene eingeführt.

6.4.3 Socket Layer

Der *Socket Layer* führt das Konzept des *User Interface Sockets* (UIS) in die UCH-Architektur ein. Ein UIS definiert hierbei auf einer abstrakten Ebene die Schnittstellen zu den jeweiligen Funktionalitäten und Zuständen eines *Targets*.

Definition 29 (*User Interface Socket*)

Ein User Interface Socket (UIS) definiert eine abstrakte Schnittstellenbeschreibung zu den jeweiligen Funktionalitäten und Zuständen eines Targets.

Eine allgemeine UIS-Beschreibung besteht in der Regel aus einem XML Dokument mit den folgenden Elementen:

- **Variables:** Variablen repräsentieren den dynamischen Zustand eines *Targets* und können sowohl durch das *Target* selbst als auch durch einen *Controller* beziehungsweise durch den Benutzer verändert werden.
- **Commands:** Kommandos bieten dem Benutzer Zugriff auf komplexere Funktionalitäten des *Targets*, welche nicht durch einfache Variablen abgebildet werden können.
- **Notifications:** Notifikationen dienen dazu, dem Benutzer direkt Nachrichten senden zu können, um ihn beispielsweise auf wichtige Situationen oder Ereignisse aufmerksam zu machen.

Am Beispiel einer universellen und barrierefreien Fahrstuhlsteuerung (Gauterin et al., 2012) zeigt Listing 6.1 exemplarisch eine UIS-Beschreibung anhand der relevanten Funktionalitäten.

```
<?xml version="1.0" encoding="UTF-8"?>

<uiSocket>
  <variable id="statusDoorOpen" type="xsd:boolean">
    <dc:description>
      If the door is open
    </dc:description>
    <dependency>
      <relevant>true()</relevant>
      <write>false()</write>
    </dependency>
  </variable>

  <variable id="statusMovementUp" type="xsd:boolean">
    <dc:description>
      If the elevator is moving up
    </dc:description>
    <dependency>
      <relevant>true()</relevant>
      <write>false()</write>
    </dependency>
  </variable>

  <variable id="statusMovementDown" type="xsd:boolean">
    <dc:description>
      If the elevator is moving down
    </dc:description>
    <dependency>
```

```
29        <relevant>true()</relevant>
          <write>false()</write>
31      </dependency>
    </variable>
33
    <variable id="statusCurrentPosition" type="xsd:integer">
35      <dc:description>
          The elevator's floor position
37      </dc:description>
        <dependency>
39        <relevant>true()</relevant>
          <write>false()</write>
41      </dependency>
    </variable>
43

45  <command id="commandOpenDoor" type="basicCommand">
    </command>
47
    <command id="commandCloseDoor" type="basicCommand">
49  </command>

51  <command id="commandFloorRequest" type="basicCommand">
      <param id="floor" dir="in" type="xsd:string"/>
53  </command>

55  <command id="commandRideRequest" type="basicCommand">
      <param id="origin" dir="in" type="xsd:string"/>
57    <param id="destination" dir="in" type="xsd:string"/>
    </command>
59

61  <notify id="elevatorIsMoving" category="info">
        <dc:description xml:lang="en">
63        Raised when the elevator is moving to your destination
      floor.
        </dc:description>
65      <dependency>
          <relevant>true()</relevant>
67        <explicitAck>false()</explicitAck>
        </dependency>
69  </notify>

71 </uiSocket>
```

Listing 6.1: Beispiel einer UIS-Beschreibung anhand einer universalen Fahrstuhlsteuerung

Jedes *Target* innerhalb des UCH wird durch eine abstrakte Schnittstellenbeschreibung in Form eines UIS repräsentiert. Um die abstrakte Schnittstelle in eine konkrete Benutzerschnittstelle zu überführen, sind allerdings noch weitere Informationen, beispielsweise in Form von sogenannten *Labels*, erforderlich (Smirek et al., 2014). Der URC Standard sieht hierzu *Resource Sheets* vor (ISO, 2008e). Ein *Resource Sheet* besteht aus einem XML-Dokument und

organisiert alle zusätzlichen Informationen zu einem bestimmten *Target*. Hierzu gehören Ressourcenbeschreibungen, Gruppierungen von Socket-Elementen sowie textbasierte atomare Ressourcen.

Definition 30 (*Resource Sheet*)

Ein Resource Sheet organisiert alle für die Realisierung einer konkreten Benutzerschnittstelle notwendige Information zu einem bestimmten Target.

Listing 6.2 zeigt exemplarisch ein Auszug aus einem *Resource Sheet* zur Beschreibung der zuvor definierten Fahrstuhlsteuerung.

```
<AResDesc rdf:about="http://res.dfki.de/elevator/elevator.rsheet#
    commandOpenDoor">
  <content rdf:parseType="Literal" xml:lang="en">Open the door</
    content>
  <useFor rdf:parseType="Collection">
  <Context>
    <eltRef rdf:resource="http://res.dfki.de/elevator/command/
    socket#commandOpenDoor"/>
    <role rdf:resource="http://myurc.org/ns/res#label"/>
  </Context>
  </useFor>
</AResDesc>

<AResDesc rdf:about="http://res.dfki.de/elevator/elevator.rsheet#
    commandCloseDoor">
  <content rdf:parseType="Literal" xml:lang="en">Close the door</
    content>
  <useFor rdf:parseType="Collection">
  <Context>
    <eltRef rdf:resource="http://res.dfki.de/elevator/command/
    socket#commandCloseDoor"/>
    <role rdf:resource="http://myurc.org/ns/res#label"/>
  </Context>
  </useFor>
</AResDesc>
```

Listing 6.2: Beispiel eines *Resource Sheets* anhand einer abstrakten Fahrstuhlsteuerung

6.4.4 User Interface Protocol Layer

Der *User Interface Protocol Layer* ist schließlich für die Anbindung der *Controller* an die UCH-Architektur verantwortlich. Unter einem *Controller* versteht man alle Endgeräte, auf denen eine konkrete Benutzerschnittstelle zur Kontrolle eines *Targets* realisiert und visualisiert wird, zum Beispiel Smartphones, Tablets, Webseiten oder Fernsehgeräte.

Definition 31 (*Controller*)

Ein Controller ist ein Endgerät, auf dem eine Benutzerschnittstelle zur Überwachung und Steuerung eines oder mehrerer Targets realisiert und visualisiert wird.

Wie *Targets* können auch die einzelnen *Controller* unterschiedliche Kommunikationsprotokolle und -schnittstellen erfordern. Für die Kommunikation zwischen *Controller* und UCH sind sogenannte *User Interface Protocol Modules* (UIPMs) verantwortlich. Ein UIPM ist hierbei eine dedizierte Softwarekomponente, welche in der Regel genau ein Kommunikations- oder Netzwerkprotokoll realisiert und bereitstellt. Die Aufgabe eines UIPM ist es, die im *Socket* beschriebenen Zustände und Funktionalitäten auf das jeweilige Protokoll abzubilden und den *Controllern* einen Zugriff darauf zu ermöglichen.

Definition 32 (*User Interface Protocol Module*)

Ein User Interface Protocol Module (UIPM) ist eine dedizierte Softwarekomponente innerhalb der UCH-Infrastruktur, welche eine Kommunikations- oder Netzwerkschnittstelle zur Verfügung stellt, um über die UIS-Beschreibung Zugriff auf die Zustände und Funktionalitäten einzelner Targets zu haben.

Im Allgemeinen sollte jede UCH-Implementierung mindestens über ein UIPM verfügen. Die OpenURC Alliance schlägt hierbei ein auf dem HTTP-Protokoll aufbauendes UIPM (URC-HTTP UIPM) vor. Das URC-HTTP UIPM bietet eine REST-Schnittstelle an und erlaubt zudem, dass einzelne *Controller* eine Verbindung in Form einer *Session* zu einem bestimmten *Target* aufbauen können und somit Zugriff auf dessen Funktionalitäten haben. Detaillierte Informationen zu der technischen Spezifikation sind auf der entsprechenden Website[7] der OpenURC Alliance zu finden.

6.4.5 Resource Server

Eine wichtige Voraussetzung zur Realisierung von personalisierten Benutzerschnittstellen in *Intelligenten Umgebungen* ist die Offenheit gegenüber Drittanbietern (Zimmermann und Wassermann, 2009). Ein *Resource Server* stellt innerhalb des URC Ökosystems hierbei alle benötigten Ressourcen zur Realisierung von Benutzerschnittstellen zur Verfügung, wie beispielsweise *Resource Sheets*, die zugehörigen Ressourcen oder *User Interface Protocol Modules*. Durch den Einsatz einer Abstraktionsschicht in Form der *User Interface Sockets* können die Benutzerschnittstellen auf den jeweiligen *Controllern* von der tatsächlichen Kommunikation mit den *Targets* entkoppelt werden, wodurch die Austauschbarkeit der Komponenten ermöglicht wird.

[7] http://www.openurc.org/TR/urc-http-protocol2.0-20131217 [Letzter Zugriff: 3.11.2014]

Definition 33 (*Resource Server*)

Ein Resource Server stellt innerhalb der URC-Infrastruktur alle benötigten Ressourcen zur Realisierung von personalisierten Benutzerschnittstellen zur Verfügung.

Zimmermann und Wassermann (2009) beschreiben den Einsatz eines *Resource Servers* als eine wichtige Schlüsseltechnologie zur Generierung eines Marktmodells für personalisierte Benutzerschnittstellen, was einerseits zu einer erhöhten Anzahl von verfügbaren Benutzerschnittstellen führt, andererseits durch den entstehenden Wettbewerb innerhalb des Marktplatzes eine Steigerung der Produktqualität nach sich zieht. Detaillierte Informationen zu der technischen Spezifikation eines *Resource Servers* sind auf der entsprechenden Website[8] der OpenURC Alliance zu finden.

Im Rahmen der OpenURC Alliance wurden bereits mehrere *Resource Server* als auch UCH-Referenzimplementierungen in unterschiedlichen Programmiersprachen realisiert. Der nachfolgende Abschnitt stellt eine im Rahmen dieser Arbeit entstandene UCH-Implementierung vor, welche bereits erfolgreich in mehreren Projekten eingesetzt wird, was wiederum durch die Vielzahl der Anwendungsszenarien (siehe Abschnitt 6.6) und der vorhandenen Demonstratoren (siehe Kapitel 11) belegt werden kann.

6.5 OSGi-basierte UCH-Implementierung

Auf Basis der in den vorigen Abschnitten vorgestellten URC-Spezifikation wurde im Rahmen dieser Arbeit eine Java-basierte Implementierung der *UCH-Middleware* umgesetzt. Der Fokus der Entwicklung lag auf der Realisierung eines modularen Architekturansatzes. Walls (2009) zeigt die Vorteile einer solchen modularen Softwarearchitektur auf:

- **Veränderlichkeit:** Propagieren die einzelnen Softwaremodule lediglich ihre öffentliche Schnittstellenbeschreibung an andere Systemkomponenten und nicht ihre internen Implementierungsdetails, so wird die Veränderbarkeit und die Austauschbarkeit einzelner Module wesentlich erleichtert.

- **Verständlichkeit:** In der Regel kapseln einzelne Module klar definierte und logisch zusammenhängende Funktionalitäten. Diese funktionale Abgrenzung gegenüber anderen Modulen erhöht die Verständlichkeit sowohl des einzelnen Moduls als auch des Gesamtsystems.

- **Parallele Entwicklung:** Eine modulare Systemarchitektur ermöglicht das parallele Bearbeiten und somit das Aufteilen von Entwicklungsaufgaben anhand der einzelnen Softwaremodule.

[8] http://www.openurc.org/TR/res-serv-http1.0-20140304[Letzter Zugriff: 3.11.2014]

- **Verbesserte Testbarkeit:** Die Modularisierung ermöglicht einen klar definierte Testprozess. In der ersten Phase können durch den Einsatz von Modultests die Module einzeln auf korrekte Funktionalität geprüft werden. Haben alle Module diesen Test bestanden, kann im nächsten Schritt das Zusammenspiel der Module anhand von Integrationstests überprüft werden.

- **Wiederverwendbarkeit und Flexibilität:** Bedingt durch die Aufteilung der Softwarekomponenten in logisch und funktional zusammenhängende Module wird es möglich, einzelne Module oder eine Sammlung bestimmter Module in anderen Anwendungen wiederzuverwenden.

Als technologische Grundlage der in dieser Arbeit vorgestellten modularen UCH-Implementierung kommt die Equinox Plattform zum Einsatz.

6.5.1 OSGi Framework

Die OSGi-Spezifikation (*Open Service Gateway Initiative*) wird von der OSGi-Alliance[9] gepflegt wie auch weiterentwickelt und beschreibt ein modulares und dynamisches Komponenten- und Servicemodell aufbauend auf der Programmiersprache Java. Abbildung 6.4 zeigt die mehrschichtige Architektur der zugrunde liegenden OSGi-Service-Plattform.

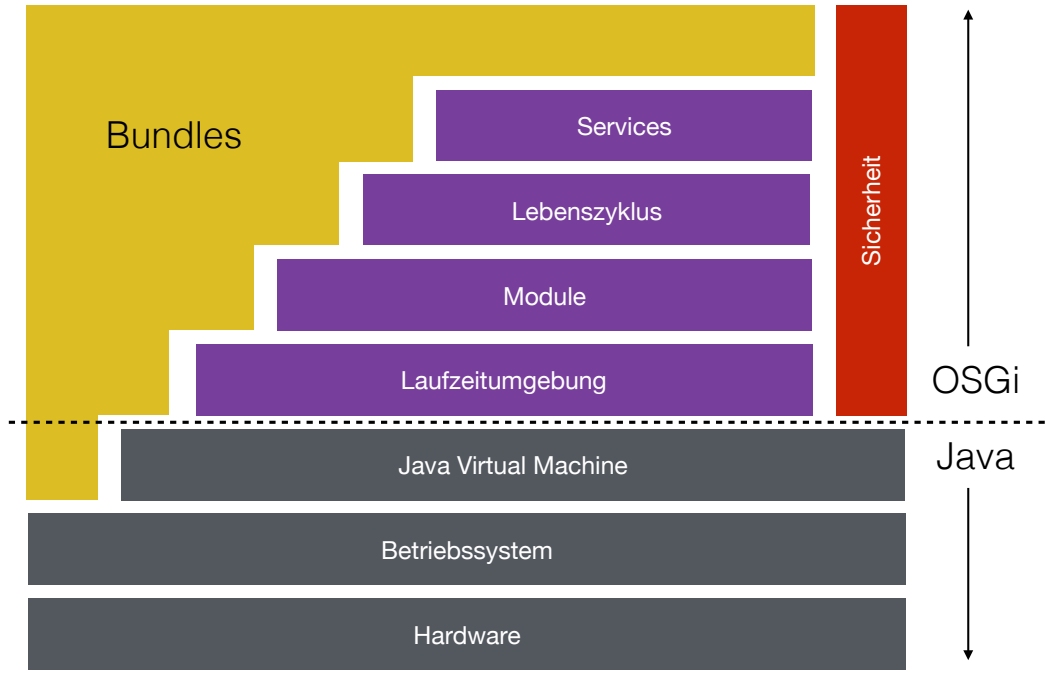


Abbildung 6.4: Überblick über die technologischen Komponenten einer OSGi-Service-Plattform (nach Walls (2009)).

[9] http://www.osgi.org

OSGi setzt grundsätzlich auf der Java Virtual Machine auf, welche wiederum als Schnittstelle zum unterliegenden Betriebssystem und somit zur Rechnerhardware fungiert. Darüber hinaus besteht das Framework aus den folgenden Komponenten:

- **Bundles:** Ein OSGi-*Bundle* ist die kleinste, modulare Einheit innerhalb einer OSGi-Plattform und stellt eine dynamische Serviceanwendung dar, welche zur Laufzeit installiert und manipuliert werden kann.

- **Lebenszyklus:** Jedes *Bundle* verfügt über einen eigenen Lebenszyklus, welcher durch eine Programmierschnittstelle innerhalb der Plattform verfügbar gemacht wird. Ein *Bundle* kann hierbei die folgenden Zustände innerhalb des Lebenszyklus einnehmen: *INSTALLED*, *RESOLVED*, *STARTING*, *ACTIVE*, *STOPPING* und *UNINSTALLED*.

- **Services:** Auf der *Service* Ebene können *Bundles* dynamisch miteinander verbunden werden, in dem sie über eine *Service Registry* selbst Dienste zur Verfügung stellen oder Dienste anderer *Bundles* konsumieren.

- **Module:** Die *Module* beschreiben die Abhängigkeiten einzelner *Bundles*, das heißt sie definieren, wie ein *Bundle* einerseits Quellcode anderer *Bundles* importieren, aber andererseits auch den eigenen Quellcode anderen *Bundles* zur Verfügung stellen kann.

- **Sicherheit:** Die OSGi Spezifikation sieht eine optionale Sicherheitsebene vor, welche unter anderem sicherstellt, dass *Bundles* durch digitale Signaturen authentifiziert werden können, oder dass *Bundles* ihre Updates nur von zuvor definierten Adressen beziehen dürfen.

Die Struktur eines OSGi-*Bundles* basiert auf der eines *Java Archives* (JAR), mit der Ergänzung, dass die darin enthaltende *META-INF/MANIFEST.MF* Datei zusätzliche und OSGi spezifische Metainformationen, wie beispielsweise ein eindeutiger *Bundle* Name, eine Versionsnummer sowie die Abhängigkeiten zu anderen *Bundles*, beschreibt. Neben kommerziellen Implementierungen von OSGi Frameworks existieren auch eine Vielzahl von quelloffenen Implementierungen, wie zum Beispiel Concierge[10], Apache Felix[11] oder Knopplerfish[12]. Die im Folgenden beschriebene Realisierung des *Universal Control Hubs* basiert auf dem durch die Eclipse Foundation vorangetriebenen Equniox[13] OSGi Framework.

[10] http://concierge.sourceforge.net[Letzter Zugriff: 4.11.2014]
[11] http://felix.apache.org[Letzter Zugriff: 4.11.2014]
[12] http://www.knopflerfish.org[Letzter Zugriff: 4.11.2014]
[13] http://eclipse.org/equinox[Letzter Zugriff: 4.11.2014]

6.5.2 UCH-Middleware

Aufbauend auf der zuvor beschriebenen URC-Spezifikation und der in Abbildung 6.3 skizzierten allgemeinen UCH-Architektur setzt sich die hier vorgestellte UCH-Implementierung aus einer Sammlung von *UCH-Bundles* zusammen. Ein *UCH-Bundle* kapselt hierbei jeweils die Funktionalitäten einer spezifischen UCH-Komponente (siehe Abschnitt 6.4).

Definition 34 (*UCH-Bundle*)

> *Ein UCH-Bundle ist ein spezielles OSGi-Bundle, welches die konkrete Funktionalität einer Komponente der allgemeinen UCH-Architektur implementiert und dynamisch zur Laufzeit hinzugefügt und wieder entfernt werden kann.*

Insgesamt werden drei Typen von *UCH-Bundles* innerhalb der hier beschriebenen UCH-Implementierung unterschieden:

- **UCH-Core-Bundle:** Jede UCH-Plattform enthält genau ein *UCH-Core-Bundle*, welches die Basisfunktionalitäten der UCH-Infrastruktur und somit die eigentliche UCH-Middleware Komponente realisiert. Eine der Kernfunktionalitäten ist die Interpretation und Verarbeitung der UIS-Beschreibung sowie die Überführung in ein internes Datenmodell. Eine weitere, essentielle Aufgabe ist die dynamische Anbindung, die funktionale Integration der *Target Adapter* wie auch der *User Interface Protocol Modules*, welche wiederum als einzelne *Bundles* realisiert sind. Hierzu stellt das *UCH-Core-Bundle* zwei zentrale *OSGi-Services* zur Verfügung. Der `ITargetManager` Service bietet eine Managementschnittstelle zur Verwaltung der erkannten *Targets* und der zugehörigen *Target Adapters*. Der `IProtocolModuleManager` Service bietet analog dazu die Möglichkeit zur Verwaltung der entsprechenden *User Interface Protocol Modules*.

- **UCH-TA-Bundle:** Eine UCH-Plattform verfügt in der Regel über mehrere *UCH-TA-Bundles*. Abbildung 6.5 zeigt die interne Struktur eines *UCH-TA-Bundles* am Beispiel der bereits zuvor erwähnten Fahrstuhlsteuerung.

  Ein *UCH-TA-Bundle* implementiert einen konkreten *Target Adapter* innerhalb der UCH-Infrastruktur und ist somit für die Integration der *Targets* verantwortlich. Das *Bundle* enthält alle hierfür notwendigen Ressourcen, wie UIS-Beschreibungen und *Resource Sheets*. Darüber hinaus muss jedes *UCH-TA-Bundle* die drei folgenden Klassen implementieren: Der `Activator` kontrolliert den Lebenszyklus des *Bundles* und startet beziehungsweise stoppt den internen *Target Discovery* Mechanismus. Dieser wird durch die Schnittstelle `ITargetDiscoveryModule` realisiert, welche dazu in der Lage ist, zur Laufzeit die über den jeweiligen *Target Adapter* auffindbaren *Targets* zu entdecken und mit Hilfe des `ITargetManager` Services in die UCH-Plattform zu integrieren. Der `ITargetAdapter` stellt

```
/
├── lib
├── META-INF
│   └── MANIFEST.MF
├── src
│   ├── Activator.java
│   ├── ElevatorTargetAdapter.java
│   └── ElevatorTargetDiscoveryModule.java
└── targets
    ├── elevator.uis
    ├── elevator.rsheet
    └── elevator.td
```

Abbildung 6.5: Übersicht über die Dateistruktur eines *UCH-TA-Bundles*.

schließlich eine Schnittstelle zur Anbindung der konkreten Gerätefunktionalitäten zur Verfügung und gibt die entsprechenden Methoden zum Auslesen und zur Manipulation des konkreten *Targets* vor.

- **UCH-UIPM-Bundle:** Eine UCH-Plattform verfügt ebenfalls über mehrere *UCH-UIPM-Bundles*, welche wiederum jeweils ein bestimmtes UIPM realisieren. Wie beim *UCH-TA-Bundle* ist der `Activator` für den Lebenszyklus des *Bundles*, als auch für die Registrierung des UIPM über den entsprechenden `IProtocolModuleManager` Service des *UCH-Core-Bundles* zuständig. Des weiteren muss jedes *UCH-UIPM-Bundle* die Schnittstelle `IProtocolModule` implementieren. Diese stellt die grundlegenden Kommunikationsmethoden zur Verfügung, mit deren Hilfe *Controller* beispielsweise eine *Session* zu einem *Target* öffnen können und dazu in der Lage sind, dessen Zustand auszulesen oder zu verändern.

Abbildung 6.6 gibt einen abschließenden Überblick über das beschriebene Zusammenspiel der einzelnen *UCH-Bundles* innerhalb der UCH-Gesamtarchitektur.

6.5.3 Secure UCH (SUCH)

Motiviert durch erhöhte Sicherheits- und Datenschutzanforderungen wurde 2013 das Projekt *Secure UCH* (SUCH) gestartet, mit dem Ziel, die hier vorgestellte OSGi-basierte Implementierung des UCH-Middlewareplattform gemäß den Sicherheitsrichtlinien der *Common-Criteria-Spezifikation* (ISO, 2009) neu zu implementieren (Britz et al., 2015). Dabei wird eine Zertifizierung gemäß des sogenannten *Evaluation Assurance Level 4* (EAL 4) angestrebt, was bedeutet, dass die Implementierung während des gesamten Entwicklungsprozesses methodisch konzipiert, getestet und durch eine entsprechende Institution evaluiert wird. Durch *Security by Design* (Bodden et al., 2013) wird hierbei sichergestellt, dass der Begriff der Sicherheit also bereits in der Entwurfsphase der Softwarearchitektur berücksichtigt wird. *Security by Design* bildet folglich

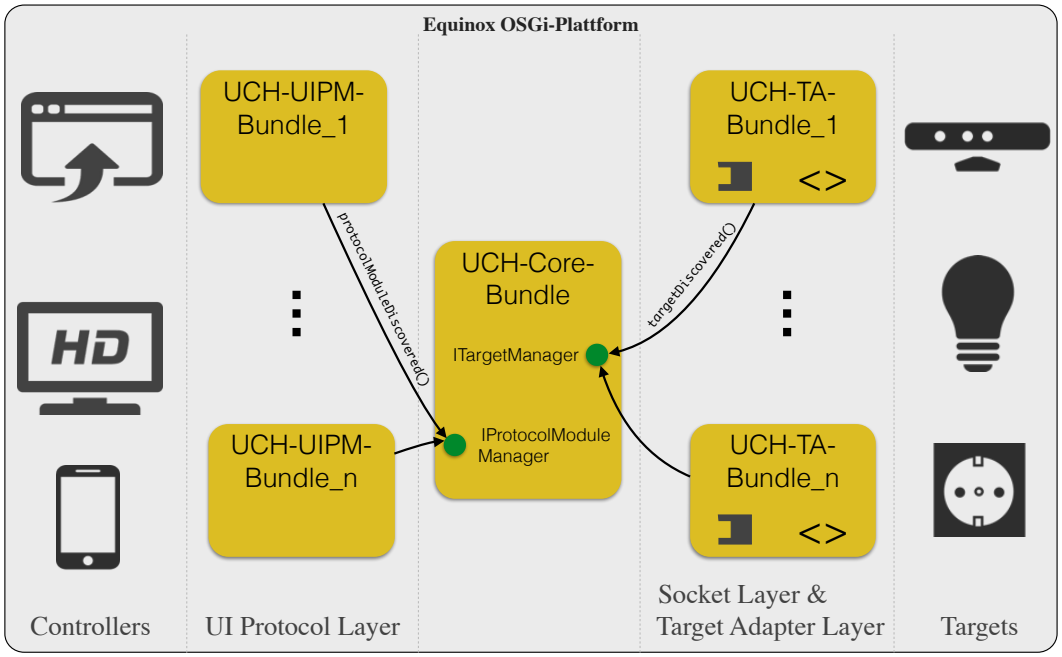


Abbildung 6.6: Überblick über die dynamische Erkennung und Einbindung von *UCH-Bundles* innerhalb der vorgestellten UCH-Architektur.

den methodischen Rahmen zur Entwicklung einer sicheren UCH-Plattform, bei der ein Nutzer jederzeit die volle Kontrolle über seine persönlichen Daten behält und das Gesamtsystem vor potentiellem Missbrauch geschützt wird. Das Hauptaugenmerk wird hierzu auf die Entwicklung einer Zugriffskontrolle in Kombination mit der Realisierung eines durchgängigen Rollenkonzeptes sowie der fehlerfreien und sicheren Kommunikation sensibler Nutzerdaten gelegt.

6.6 Anwendungsszenarien

Die bisher vorgestellten theoretischen Ansätze sowie praktischen Implementierungen wurden im Rahmen des DFKI-Kompetenzzentrums für Ambient Assisted Living (CCAAL) (Frey et al., 2010d) anhand mehrerer Anwendungsszenarien implementiert und getestet. In den folgenden Abschnitten werden ausgewählte Beispielszenarien zur Nutzung der beschriebenen UCH-Implementierung kurz vorgestellt.

6.6.1 DFKI Smart Kitchen

Die *Smart Kitchen* ist ein instrumentierter und täglich genutzter Aufenthaltsraum innerhalb des DFKI-Gebäudes in Saarbrücken. Sie besteht grundsätzlich aus zwei verschiedenen Bereichen. Die *Smart Kitchen* verfügt über einen voll ausgestatteten und funktionstüchtigen Küchenbereich mit Spülmaschine, Kühlschrank, Ofen, Herd und Dunstabzugshaube basierend auf der Siemens Serve@Home-Technologie. Sie beinhaltet zudem RFID-Sensoren, wodurch sensitive Oberflächen realisiert werden können, um speziell gekennzeichnete Kü-

chenutensilien oder Lebensmittelpackungen zu erkennen. Darüber hinaus realisiert die *Smart Kitchen* aber auch eine komplette Wohnzimmerkonfiguration mit Sitzecke und Multimediageräten. So können durch die Integration des Windows Media Centers in die UCH-Plattform neuartige Interaktionsmechanismen mit einem Fernsehgerät realisiert und getestet werden (Epelde et al., 2013; Epelde, 2014). Des Weiteren verfügt die *Smart Kitchen* über drahtlose Funktechnologien wie beispielsweise Enocean und ZigBee. Hierbei handelt es sich primär um schaltbare Steckdosenleisten zur Realisierung von Lichtautomationsszenarien sowie Bewegungs- und Schließkontakte. Auf Basis der URC-Infrastruktur wurden bisher mehrere Küchenszenarien umgesetzt und mit Anwendern unter realistischen Bedingungen getestet. So wurde zunächst ein Konzept realisiert, um den Benutzer durch die Bereitstellung von Kochrezepten und durch die sprachgesteuerte Handlungsunterstützung beim Ausführen der Rezepte zu unterstützen (Schäfer et al., 2013). Weiterhin wurden mehrere multimodale Interaktionskonzepte implementiert, wie zum Beispiel eine Gestensteuerung zur Steuerung der Küchengeräte (Neßelrath et al., 2011b) oder eine Kombination aus Zeigegesten und Sprachkommandos zur Erstellung eines räumlichen Modells der Umgebung (Gard, 2012). Abbildung 6.7 zeigt die Infrastruktur der *Smart Kitchen* sowie eine Auswahl der darauf aufbauenden Demonstratoren.

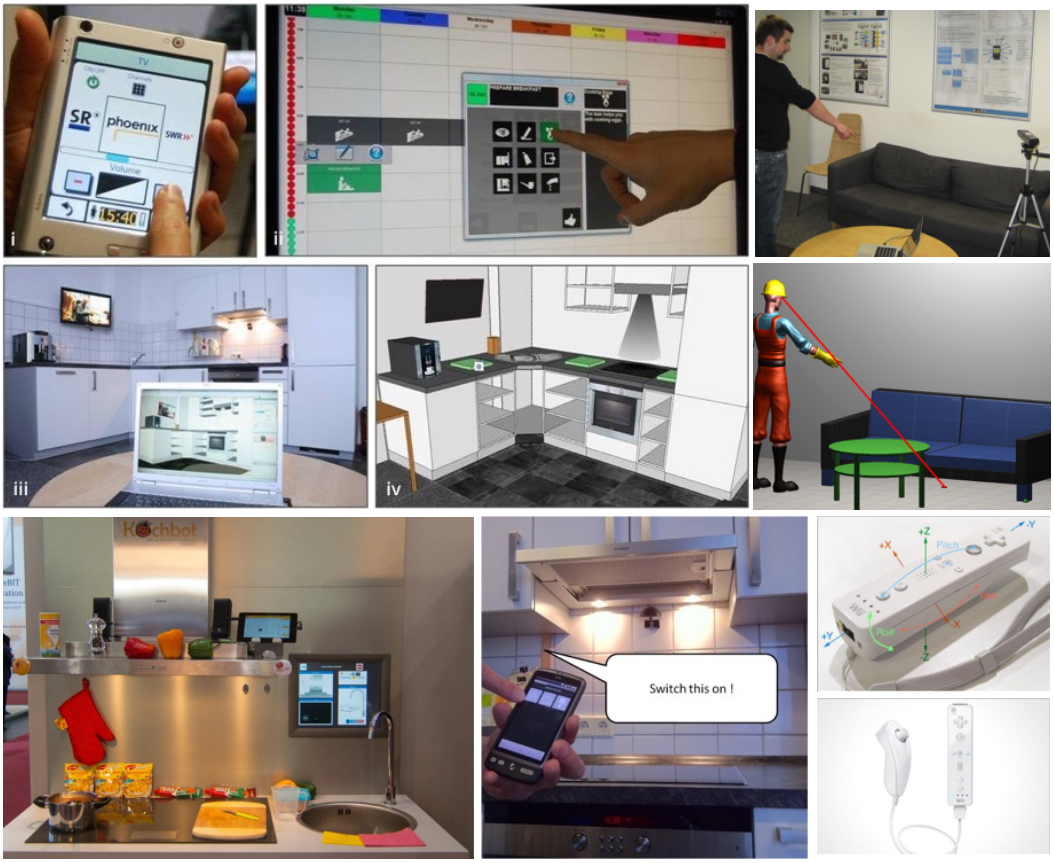


Abbildung 6.7: Übersicht über die Infrastruktur der *Smart Kitchen* und eine Auswahl der darauf aufbauenden Demonstratoren.

6.6.2 DFKI Bremen Ambient Assisted Living Lab

Das *Bremen Ambient Assisted Living Lab* (BAALL) ist eine sechzig Quadratmeter große, voll instrumentierte alters- und behindertengerechte Wohnung in den Projekträumen des Forschungsbereichs *Cyber-Physical Systems* des Deutschen Forschungszentrums für Künstliche Intelligenz am Standort Bremen. Neben einer Küche, einem Wohnzimmer, einer Arbeitsecke und einem Schlafzimmer verfügt das BAALL zudem über ein voll funktionstüchtiges und barrierefreies Badezimmer. Eines der Hauptziele des BAALL ist es, die Alltagstauglichkeit von Assistenzsystemen in realistischen Konfigurationen überprüfen zu können. Hierzu ist es möglich, durch Probewohnen innerhalb der Räumlichkeiten des BAALL realistische Erfahrungen mit der eingesetzten Technologie zu sammeln. Neben Interoperabilitäts- und Sicherheitsfragen bei der Benutzung der umgebenden Technologie spielen vor allem Mobilitätsassistenzsysteme eine wichtige Rolle. Hierzu wurden im BAALL bereits ein intelligenter Rollstuhl (Röfer et al., 2009b) sowie ein intelligenter Rollator (Röfer et al., 2009a) entwickelt. Das BAALL wurde mit einer Vielzahl an *Sensorik* und *Aktuatorik* ausgestattet. So verfügt es beispielsweise über höhenverstellbare Toilettensitze und Küchenschränke, eine vollautomatische Bett- und Türsteuerung sowie ein flächendeckendes Positionierungs- und Ortungssystem. Abbildung 6.8 zeigt eine Auswahl der im BAALL realisierten Demonstratoren.

Abbildung 6.8: Übersicht über die Infrastruktur des *Bremen Ambient Assisted Living Labs* und eine Auswahl der darauf aufbauenden Demonstratoren (Frey et al., 2010d).

Im Rahmen von Stahl et al. (2011) wurde weiterhin eine bidirektionale Synchronisierung der *Smart Kitchen* und der BAALL Infrastruktur realisiert. Hierzu wurde die gesamte *Sensorik* und *Aktuatorik* innerhalb des BAALLs in eine dedizierte *UCH-Middlewareplattform* überführt. Im nächsten Schritt wurden beide UCH-Plattformen durch ein entwickeltes Pluginkonzept synchronisiert.

Detaillierte Informationen zu dem zugrunde liegenden Synchronisierungskonzept sowie der technischen Realisierung sind in Kapitel 11.3.3 zu finden.

6.6.3 DFKI Accessible Elevator

Die Zielsetzung des *Accessible Elevators* (Gauterin et al., 2012) ist es, die Interaktion mit Aufzügen durch alternative und barrierefreien Benutzerschnittstellen zu realisieren. Die Entwicklung der Fahrstuhltechnik führte zu einer Weiterentwicklung der Steuerungskonzepte. Ausgehend von menschlichen Aufzugführern hat sich mittlerweile die Sammelsteuerung durchgesetzt, bei der der Benutzer auf jedem Flur signalisieren kann, ob er nach oben oder unten fahren möchte. Die Kommandos werden in der Regel in Fahrtrichtung der Kabine abgearbeitet. Ein intelligenteres Steuerungskonzept stellt auch die Zielauswahlsteuerung dar, bei der der Benutzer beim Rufen des Aufzugs bereits das Zielstockwerk angeben muss, wodurch eine effizientere Routenplanung ermöglicht wird (Koehler und Ottiger, 2002). Ein oftmals vernachlässigter Aspekt ist jedoch die Verfügbarkeit von alternativen und auf die speziellen Bedürfnisse der Benutzer angepasste Benutzerschnittstellen, zum Beispiel die Nutzung von unterschiedlichen Modalitäten wie Sprache und Gestik oder die automatische Miteinbeziehung des aktuellen Kontextwissen über den Benutzer. Abbildung 6.9 zeigt eine Auswahl an Benutzerschnittstellen zu Steuerung des *Accessible Elevators* im DFKI-Gebäude in Saarbrücken.

Abbildung 6.9: Übersicht über eine Auswahl an Benutzerschnittstellen zur Steuerung des *Accessible Elevators* im DFKI-Gebäude in Saarbrücken.

Durch die Implementierung eines *Target Adapters* und einer entsprechen-

den UIS-Beschreibung konnte die Fahrstuhlsteuerung innerhalb einer am DFKI installierten UCH-Plattform integriert und dadurch unterschiedliche Anwendungsszenarien realisiert werden. Neben mehreren mobilen Schnittstellen, bei denen der Fahrstuhl bereits beim Betreten des Gebäudes gerufen werden kann, wurde auch ein Empfangsszenario realisiert. Hierbei kann der Empfang für ankommende Gäste eine Routenplanung zu den entsprechenden Mitarbeitern erstellen. Daraufhin ist der Fahrstuhl autonom und per Sprachsteuerung dazu in der Lage, den Gast zu seinem Ziel zu führen.

6.7 Zusammenfassung und Fazit

In diesem Kapitel wurde ein allgemeiner Überblick über den Industriestandard URC sowie über die im Rahmen dieser Arbeit realisierte UCH-Implementierung gegeben, mit dessen Hilfe sich ein personalisierte Zugriff auf die den Benutzer umgebende Sensorik und Aktuatorik realisieren lässt. Als wichtiger Bestandteil der Architektur wurde die Notwendigkeit einer Abstraktionsebene für Benutzerschnittstellen eingeführt. Durch den Einsatz einer *User Interface Socket* Beschreibung können die Funktionalitäten von *Target* Applikationen abstrakt beschrieben werden, wodurch eine Entkopplung von tatsächlichen Benutzerschnittstellen (*Controller*) und den spezifischen Netzwerkprotokollen zur Anbindung der individuellen *Sensorik* und *Aktuatorik* erzielt werden kann. Der UCH erfüllt somit die Definition der in Kapitel 2.3 beschriebenen *Middleware* als Teil einer *Software-definierten Plattform* (siehe Kapitel 3.3) und ermöglicht somit eine Virtualisierung der angeschlossenen *Smart Products*.

URC-konforme Architekturen erlauben eine hohe Austauschbarkeit, Flexibilität und Wiederverwendbarkeit bei der Entwicklung von personalisierten und barrierefreien Benutzerschnittstellen. Durch die abstrakte UIS-Beschreibung können sich die Entwickler von *Controller* Anwendungen ausschließlich auf die Gestaltung und die individuellen Voraussetzungen der jeweiligen Benutzerschnittstelle konzentrieren, ohne sich um die konkrete Anbindung der hersteller- und netzwerkspezifischen *Sensorik* und *Aktuatorik* kümmern zu müssen. Mit dem OSGi-basierten UCH wurde eine modular aufgebaute Implementierung dieses theoretischen Ansatzes vorgestellt und in diesem Kontext die Vorteile der zugrunde liegenden dynamischen OSGi-Architektur kurz beleuchtet. Hierzu wurden drei Typen von *UCH-Bundles* anhand ihrer Aufgabe und Funktionsweise beschrieben, welche die Grundlage der vorgestellten Softwarearchitektur bilden. Im Rahmen der Arbeit wurden neben der Kernplattform eine Vielzahl von *Target Adaptern* sowie mehrere *User Interface Protocol Modules* implementiert und über einen *Resource Server* zur Verfügung gestellt. Ein breites Spektrum an ausgewählten Anwendungsszenarien, Demonstratoren und Benutzerschnittstellen aus dem Forschungsbereich *Ambient Assisted Living* runden die Darstellung ab und belegen die Relevanz sowie die praktische Anwendbarkeit des Ansatzes an konkreten Umsetzungsbeispielen.

Schwachpunkte des vorgestellten Ansatzes sind das Fehlen eines konzeptionellen Ansatzes zur Integration von intelligenten Mehrwertdiensten in Form

von *Smart Services*. Die Stärke der URC-Technologie liegt in der Bereitstellung von alternativen und persönlichen Benutzerschnittstellen und nicht in der Integration von *Target* übergreifenden intelligenten Diensten. Weiterhin beinhaltet die URC-Architektur derzeit keine semantische Beschreibung der Umgebung und der darin enthaltenen Objekte. Zwar können mit Hilfe der UIS-Beschreibungen die Funktionalitäten der Sensorik und Aktuatorik durch eine standardisierte und abstrakte Beschreibungssprache syntaktisch formuliert werden, die eigentliche Bedeutung der Umgebung, wie sie beispielsweise durch die in Kapitel 5.5 vorgestellten semantischen Repräsentationen von *Intelligenten Umgebungen* dargestellt werden, wird jedoch nicht modelliert. Darüber hinaus fehlt eine durchgängige Werkzeugkette basierend auf etablierten Technologien, welche sowohl Softwareentwickler, Dienstanbieter als auch Endnutzer während des gesamten Lebenszyklus von *Intelligenten Umgebungen* optimal unterstützt. Neben der Erweiterung des *Resource Server* Konzeptes zu einem vollständigen Marktplatz für *Smart Services* stellt die Bereitstellung von modernen Entwicklungs- und Verwaltungswerkzeugen ein entscheidender Erfolgsfaktor zur Erzielung einer verbreiteten Nutzung der zugrunde liegenden Technologien.

There's an App for That.
Apple

7

Plattformkonzept zur Integration von Smart Services in Intelligenten Umgebungen

7.1 Übersicht

Dieses Kapitel widmet sich der Fragestellung, wie Geräte- und Anwendungsübergreifende Mehrwertdienste innerhalb einer *Intelligenten Umgebung* integriert werden können. Hierzu wird ein einheitliches Plattformkonzept zur Integration von Mehrwertdiensten in Form von *Smart Services* vorgestellt. Abschnitt 7.2 skizziert die Grundidee des hier vorgeschlagenen Ansatzes. In Abschnitt 7.3 wird die Struktur und der Aufbau von *Smart Services* im Kontext von *Intelligenten Umgebungen* definiert. Abschnitt 7.4 beschreibt die zugrunde liegenden Kommunikationsmechanismen der *Smart Services* untereinander. In Abschnitt 7.5 werden fünf unterschiedliche Typen von *Smart Services* eingeführt. Jeder Servicetyp hat hierbei eine definierte Aufgabe und Verantwortlichkeit innerhalb einer Umgebung. Abschnitt 7.6 veranschaulicht das Zusammenspiel und die Kooperation von *Smart Services* mit dem Ziel, dem Benutzer einen durchgängigen Mehrwert anzubieten. Abschnitt 7.7 zeigt, wie *Multiagentensysteme* eingesetzt werden können, um die zuvor beschriebenen *Smart Service* Funktionalitäten zu realisieren. Die Abschnitte 7.8 und 7.9 zeigen, wie das vorgestellte Plattformkonzept sich in die *Smart Service Welt* (siehe Kapitel 3) sowie in die bestehende OpenURC-Architektur (siehe Kapitel 6) einordnen lässt. Abschnitt 7.10 gibt schließlich zwei Beispiele für Anwendungsszenarien des beschriebenen Plattformkonzepts, einerseits aus Sicht des Endanwenders beziehungsweise des Bewohners der Umgebung, und andererseits aus der Sicht eines *Smart Service*-Entwicklers.

7.2 Motivation und Grundidee

Das Ziel dieses Kapitels ist es, das in Kapitel 3 vorgestellte Forschungsgebiet der *Smart Services* durch eine Kombination mit den Konzepten des in Kapitel 6 detailliert beschriebenen URC-Ansatzes in ein einheitliches Plattformkonzept zur dynamischen Integration von Mehrwertdiensten innerhalb einer *Intelligenten Umgebung* zu überführen (siehe Abbildung 7.1).

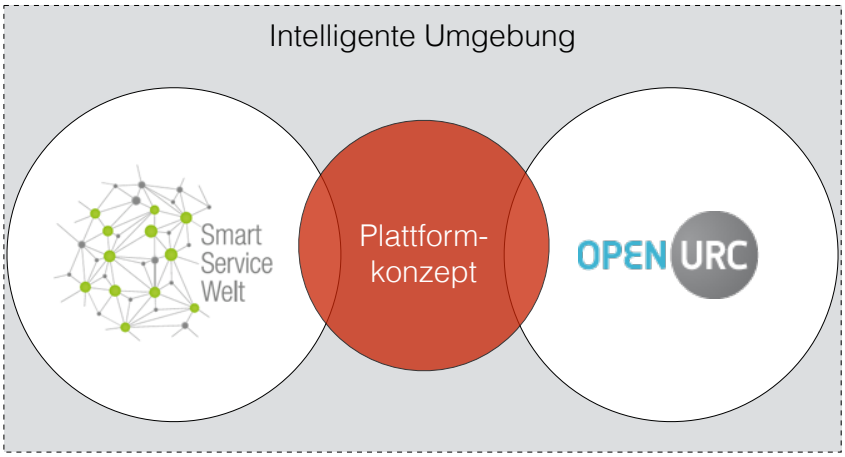


Abbildung 7.1: Verknüpfung der *Smart Service Welt* und der OpenURC-Technologie durch ein einheitliches Plattformkonzept für *Intelligente Umgebungen*.

In Kapitel 3 wurde hierzu das Forschungsgebiet der *Smart Service Welt* eingeführt und die verschiedenen Architekturschichten *Vernetzte Physische Plattform*, *Software-definierte Plattform* und *Smart Service Plattform* gegeneinander abgegrenzt. Jede dieser Schichten verfolgt einen bestimmten Zweck und arbeitet mit unterschiedlichen Daten. Zusammengefasst bilden *Vernetzte Physische Plattform* die unterste der drei Schichten und ermöglichen die Integration von *Smart Products* als Kombination von intelligenten Hard- und Softwaresystemen. Im Kontext einer *Intelligenten Umgebung* umfassen *Smart Products* sowohl *Sensoren* als auch *Aktuatoren*. Die Vielzahl der potentiell verfügbaren Geräte erzeugt dementsprechend auch eine große Menge an anfallenden Daten (*Big Data*), welche durch intelligente Analyseverfahren in aufbereitete *Smart Data* überführt werden können. Sowohl *Big Data* als auch *Smart Data* dienen wiederum als Grundlage für die eigentlichen intelligenten Mehrwertdienste in Form von *Smart Services*. Die in der *Smart Service Welt* vorgeschlagene Architektur stellt die Basis für die Entwicklung von entsprechenden Ökosystemen für *Smart Services* in unterschiedlichen Anwendungsbereichen dar und skizziert die notwendigen Anforderungen auf jeder Abstraktionsebene, sowie den zugrunde liegenden Informations- und Datenfluss. In Kapitel 6 wurde eine auf Standards basierende *Middleware* zur personalisierten und barrierefreien Interaktion mit *Instrumentierten Umgebungen* und eine im Rahmen dieser Arbeit realisierte Implementierung des Konzepts detailliert vorgestellt. Kernbestandteil des Konzepts ist der Einsatz einer einheitlichen Beschreibung der Instrumentierung in Form einer *User Interface Socket* Beschreibung, auf deren Basis mehrere unterschiedlicher Benutzerschnittstellen realisiert wurden (siehe Kapitel 11). Die Verfügbarkeit eines oder mehrerer zentralen *Resource Server* stellt den Zugriff auf alle benötigten Ressourcen sicher und ermöglicht zudem eine Austauschbarkeit und Wiederverwendbarkeit dieser Ressourcen. Beide Konzepte haben jeweils Stärken und Schwächen. So verfügt die *Smart Service Welt* unter anderem über eine klare Strukturierung der unterschied-

lichen Abstraktionsebenen und des Informationsflusses zur Realisierung eines Gesamtökosystems für intelligente Mehrwertdienste. Ein einheitliches Konzept zur Entwicklung und zum Austausch personalisierter Benutzerschnittstellen, wie es in OpenURC vorgeschlagen wurde, wird allerdings direkt nicht berücksichtigt. OpenURC hingegen bietet wiederum kein einheitliches Konzept zur Integration intelligenter Mehrwertdienste innerhalb der *Middleware*, sowohl aus Endnutzer- als auch aus Entwicklersicht. Die nachfolgenden Abschnitte zeigen, wie der *Smart Service* Begriff im Kontext einer *Intelligeten Umgebung* angewendet werden kann und die zuvor beschriebenen Konzepte der *Smart Service Welt* und der OpenURC-Architektur vereint werden können. Die Kombination einzelner *Smart Services* zu einem einheitlichen Mehrwert für den Nutzer folgt hierbei der These von (Brooks, 1991), in der zwei grundlegende Bestandteile einer intelligenten Anwendung voraussetzt werden:

- Die Fähigkeiten eines intelligenten Systems müssen inkrementell aufgebaut werden, wobei zu jedem Zeitpunkt eine Verfügbarkeit der Funktionalität gewährleistet werden muss.
- Jeder Zustand eines inkrementell erweiterbaren Systems sollte in der Lage sein, in der realen Welt mit tatsächlich auflaufenden Sensorinformationen eingesetzt zu werden und darin bestehen zu können.

Diese Betrachtungsweise dient als gedanklicher Leitfaden für das nachfolgende Plattformkonzept zur Integration von *Smart Services*.

7.3 Struktur und Aufbau von Smart Services in Intelligenten Umgebungen

Wie zuvor erwähnt, stellt ein *Smart Service* einen expliziten Mehrwert für den Benutzer, den Bewohner der Umgebung, dar und verfügt über eine semantische Repräsentation seiner Funktionalitäten. Hierzu werden sowohl *Input* als auch *Output* eines *Smart Services* explizit definiert und repräsentiert:

- **Input:** Der *Input* eines *Smart Services* umfasst alle eingehenden Signale, welche einerseits Informationen sind, die durch den Service weiterverarbeitet werden können, oder andererseits konkrete Anweisungen darstellen, um den Service zu einer bestimmten Aktion oder Handlung zu veranlassen.
- **Output:** Der *Output* stellt dementsprechend alle ausgehenden Signale eines *Smart Services* dar und repräsentiert seine Kommunikationsmöglichkeiten mit der Umwelt. Dies geschieht wiederum über einen Informationsaustausch mit anderen Services oder über das Ausführen konkreter Aktionen und Handlungen.

Abbildung 7.2 zeigt die allgemeine interne Struktur eines *Smart Services*, wie sie in der vorliegenden Arbeit im Kontext von *Intelligenten Umgebungen* zum Einsatz kommt.

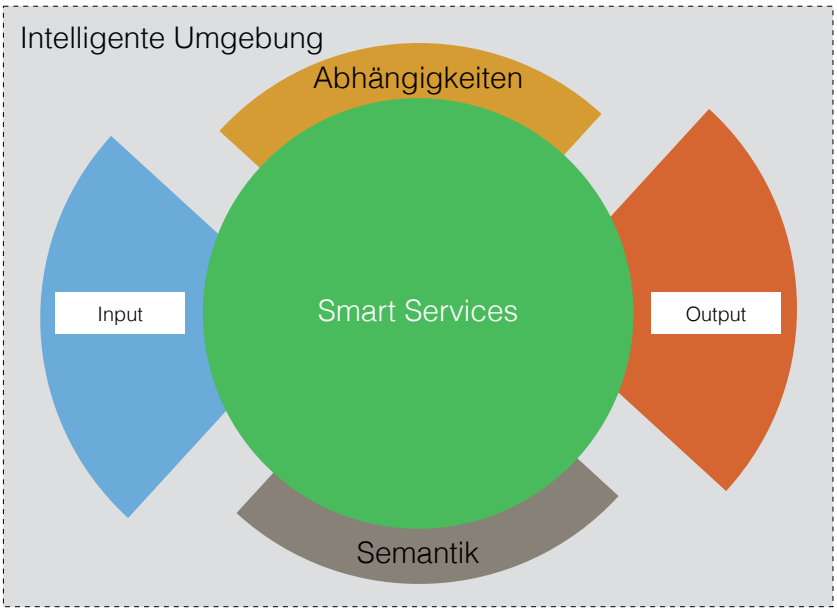


Abbildung 7.2: Übersicht über den strukturellen Aufbau eines *Smart Services* innerhalb einer *Intelligenten Umgebung*.

Man betrachte beispielsweise einen *Smart Service* mit dem Ziel, die Lichtsteuerung eines Raumes so zu automatisieren, dass sich, wenn sich keine Person darin aufhält, das Licht nach einer bestimmten Zeit von selbst ausschaltet. Als *Input* würde der Service die Information erwarten, ob sich aktuell eine Person in dem Raum aufhält. Als *Output* würde der Service entweder das Licht ausschalten, falls sich über einen definierten Zeitraum keine Person im Raum aufhält, oder das Licht anschalten, sobald eine Person den Raum betritt. *Input* und *Output* stellen somit auch indirekt die Abhängigkeiten zu anderen Services dar. Um den zuvor beschriebenen *Smart Service* realisieren zu können, muss also sowohl ein Service existieren, welcher erkennen kann, ob sich eine Person innerhalb des entsprechenden Raumes befindet (Krüger et al., 2014), als auch ein Service vorhanden sein, welcher die Lichtsteuerung innerhalb des Raumes steuern kann.

7.4 Kommunikation von Smart Services

Das zuvor betrachtete Beispiel der automatischen Lichtsteuerung zeigt die Notwendigkeit unterschiedlicher Kommunikationsmöglichkeiten der *Smart Services* untereinander. So muss im einfachsten Fall die Übermittlung einer reinen Sachinformation von einer Aufforderung zu einer bestimmten Aktion voneinander unterschieden werden. Der im Folgenden vorgeschlagene Kommunikationsmechanisums geht im Wesentlichen auf die in Kapitel 4.4 vorgestellte Sprechakttheorie nach Searle (1969) zurück. Die folgende Teilmenge der kommunikativen Akte spielt im Kontext von *Intelligenten Umgebungen* eine besondere Rolle:

- **<inform>:** Der *<inform>* Akt dient der reinen Übermittlung von Sachinformationen, wie beispielsweise der Übermittlung des aktuellen Stromverbrauchs eines angeschlossenen Smart Meters. Ein Service kann diese Informationen einem anderen Service oder dem Benutzer zur Verfügung stellen und ihn über seinen aktuellen Kontext informieren. Der *<inform>* Akt kann wiederum einen *<inform>*, einen *<propose>* oder einen *<request>* Akt bei dem adressierten Service auslösen.

- **<request>:** Der *<request>* Akt fordert eine Zustandsinformation an und löst bei dem Empfänger einen korrespondierenden *<inform>* Akt aus.

- **<propose>:** Der *<propose>* Akt fordert einen Service auf, eine bestimmte Handlung oder Aktion auszuführen. Akzeptiert der adressierte Service diese Anforderung, so wird ein *<accept>* Akt als Antwort geschickt. Bevor die Aktion tatsächlich ausgeführt wird, kann der adressierte Service wiederum ein *<request>* oder einen *<propose>* Akt aussenden, um beispielsweise bestimmte für die auszuführende Aktion benötigte Kontextinformationen zu erhalten.

- **<accept>:** Der *<accept>* Akt informiert den anfragenden Service darüber, dass sein Auftrag akzeptiert wurde und ausgeführt wird.

- **<reject>:** Der *<reject>* Akt informiert den anfragenden Service darüber, dass sein Auftrag abgelehnt wurde und die Aktion nicht ausgeführt wird.

Abbildung 7.3 fasst die vorgestellten kommunikativen Akte zusammen, und ordnet diese den entsprechenden *Input* und *Output* Kategorien eines Smart Services zu.

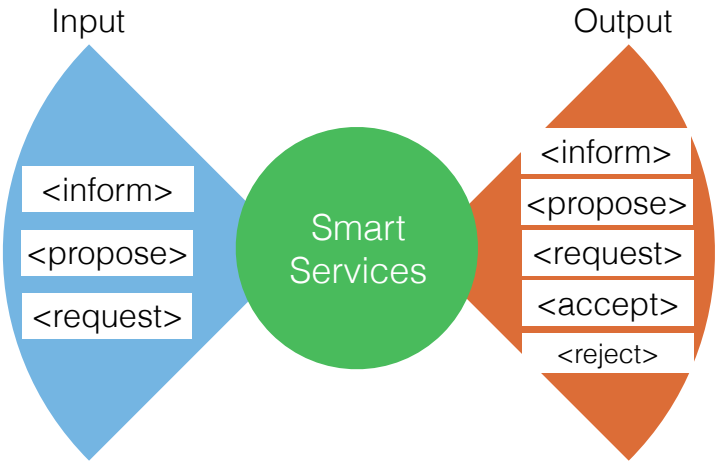


Abbildung 7.3: Übersicht über die kommunikativen Akte von *Smart Services* und ihre Einordnung in die entsprechende *Input* und *Output* Kategorie.

7.5 Typen von Smart Services in Intelligenten Umgebungen

In Kapitel 3 wurde bereits eine Definition für den Begriff des *Smart Service* eingeführt, in der dieser als Verschmelzung von digitalen und physischen Dienstleistungen zu einer auf den einzelnen Konsumenten bedarfsgerecht zugeschnittenen Gesamtdienstleistung bezeichnet wird.

In den folgenden Abschnitten wird nun versucht die technischen, konzeptionellen und funktionalen Anforderungen an einen *Smart Service* im Kontext von *Intelligenten Umgebungen* zusammenzufassen. Das Ziel ist es, eine minimale Menge dedizierter *Smart Service Typen* herauszuarbeiten, die jedoch mächtig genug ist, um alle Anforderungen an eine *Intelligente Umgebung* (vergleiche Kapitel 2.6) erfüllen zu können.

Abbildung 7.4 gibt einen Überblick über die hierarchische Struktur der in der vorliegenden Arbeit erarbeiteten Typen von *Smart Services*. Hierbei werden grundsätzlich fünf Arten von *Smart Services* vorgeschlagen: ein *Interaction Service* zur Interaktion mit dem Benutzer, ein *Assistance Service* zur Bereitstellung von Benutzerassistenz, ein *Persistence Service* zum Speichern von Nutzer- und Umgebungsdaten, ein *Sensor Service* zur Integration von *Sensoren* und schließlich ein *Actuator Service* zur Integration von *Aktuatoren* innerhalb der Umgebung.

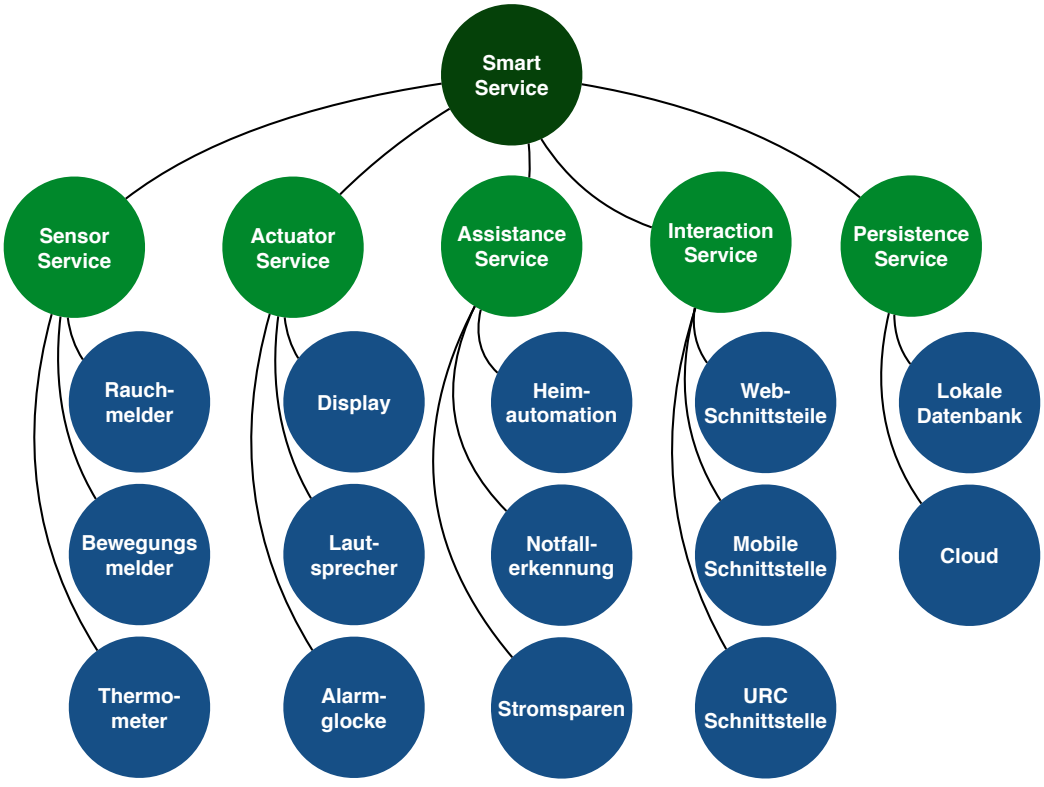


Abbildung 7.4: Übersicht über die Struktur und entsprechende Beispielausprägungen von *Smart Services* in *Intelligenten Umgebungen*.

7.5.1 Sensor Service

Wie in Kapitel 2.3 definiert wurde, dienen *Sensoren* dazu, den Kontext der Umgebung zu bestimmen, indem sie den Zustand der Umgebung und das Verhalten des Benutzers erfassen. Ein *Sensor Service* hat die Aufgabe, einen solchen *Sensor* zu überwachen und dessen Zustandsveränderungen entsprechend weiterzuleiten. Funktional betrachtet, stellt der *Sensor Service* die einfachste Form eines *Smart Services* dar. Er verfügt insgesamt über zwei kommunikative Akte. Als *Input* ist er in der Lage *<request>* Akte zu verarbeiten, welche er mittels eines *<inform>* Akts beantwortet. Ein typisches Beispiel für einen *Sensor Service* ist die Anbindung eines Thermometers. Über einen *<request>* Akt kann die aktuelle Temperatur der Umgebung angefragt werden, welche als *<inform>* Akt übermittelt wird. Abbildung 7.5 zeigt die Abläufe innerhalb eines *Sensor Services* am Beispiel eines Thermometers.

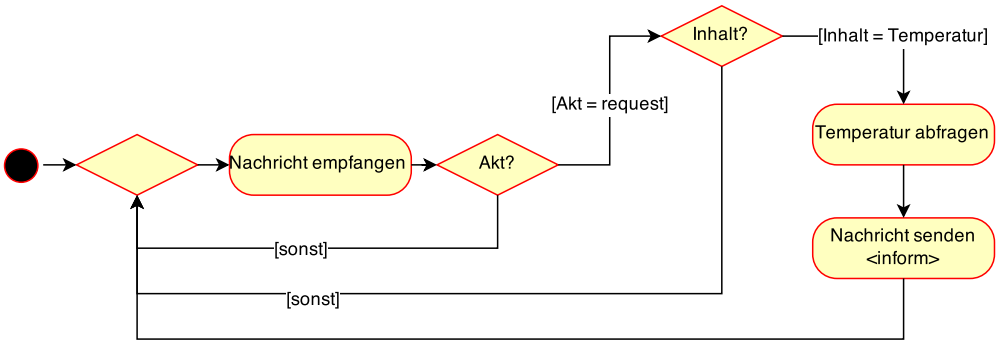


Abbildung 7.5: Übersicht über die Abläufe innerhalb eines *Sensor Services* am Beispiel eines Thermometers.

7.5.2 Actuator Service

Verglichen mit *Sensoren* sind *Aktuatoren* dazu in der Lage, Informationen an den Benutzer zu übermitteln oder den Zustand der Umgebung zu verändern. Ein *Actuator Service* ist also direkt oder indirekt für die Realisierung von Zustandsveränderungen innerhalb der Umgebung verantwortlich. Der *Actuator Service* ist ebenfalls ein sehr einfach strukturierter Service und verfügt wie der *Sensor Service* über zwei kommunikative Akte. Als *Input* verarbeitet er *<propose>* Akte, welche entweder mit einem *<accept>* Akt oder einem *<reject>* Akt beantwortet werden. Ein Beispiel für einen *Actuator Service* ist die Anbindung einer Alarmglocke. Über einen *<propose>* Akt wird ein Läuten der Glocke angefordert, was je nach Kontext entsprechend beantwortet wird. *Acuator Services* treten innerhalb einer *Intelligenten Umgebung* oftmals nicht isoliert auf, sondern sind meistens mit einem *Sensor Service* kombiniert, wie beispielsweise eine kontrollierbare Lampe, welche einerseits ein- und ausgeschaltet werden und zudem auch den eigenen Zustand übermitteln kann.

Abbildung 7.6 zeigt die Abläufe innerhalb eines *Actuator Services* am Beispiel einer Alarmglocke.

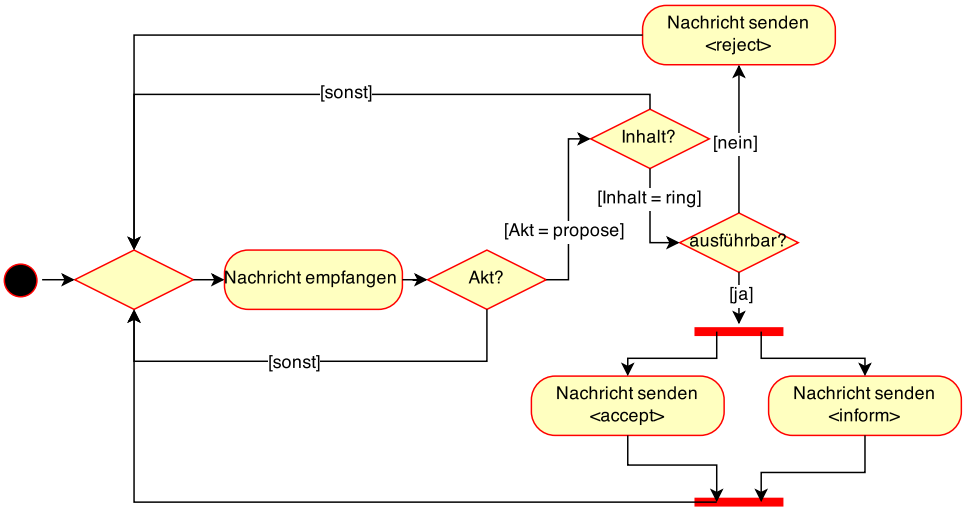


Abbildung 7.6: Übersicht über die Abläufe innerhalb eines *Actuator Services* am Beispiel einer Alarmglocke.

7.5.3 Assistance Service

Ein *Assistance Service* stellt eine direkte Anwendung eines intelligenten Mehrwertdienstes dar, welcher letztendlich immer eine Form von Benutzerassistenz zur Verfügung stellen soll. Abbildung 7.7 zeigt die Abläufe innerhalb eines *Assistance Services* am Beispiel einer einfachen automatischen Lichtsteuerung.

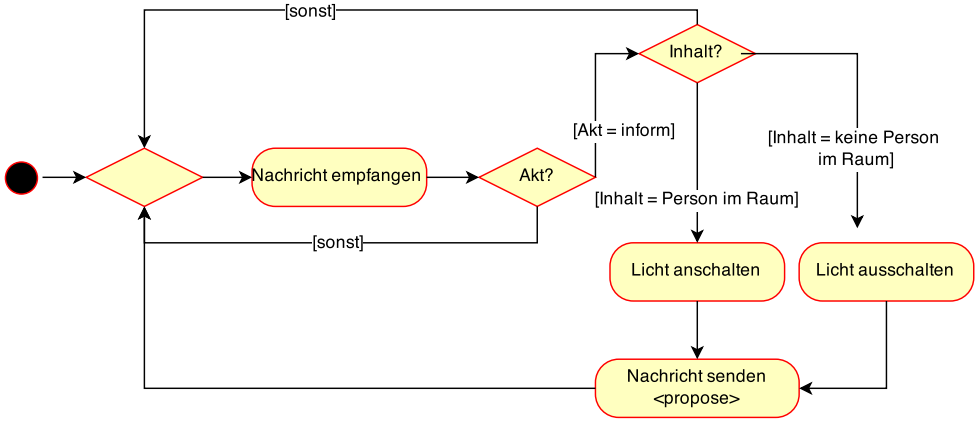


Abbildung 7.7: Übersicht über die Abläufe innerhalb eines *Assistance Services* am Beispiel einer einfachen automatischen Lichtsteuerung.

Der *Assistance Service* kommuniziert mit allen Typen von *Smart Services*, um die notwendigen Informationen über die Umgebung zu bekommen und die

berechneten Assistenzinformationen wieder in die Plattform zurückzuleiten. Ein typischer Anwendungsfall eines *Assistance Service* kann zum Beispiel die klassische Heimautomation sein, wie automatische Lichtsteuerung beim Betreten eines Raumes. Hierzu wird der *Assistance Service* über einen *<inform>* Akt darüber informiert, ob sich eine Person im entsprechenden Raum aufhält. Der *Assistance Service* prüft die Situation anhand seines Kontextwissens über die Umgebung und über die Präferenzen des Benutzers und sendet ein *<propose>* Akt mit dem jeweiligen Schaltbefehl für die entsprechende Lichtsteuerung.

7.5.4 Interaction Service

Interaction Services sind für jegliche Art der Kommunikation mit dem Bewohner der Umgebung verantwortlich, zum Beispiel für das Bereitstellen einer privaten *Smart Home* Steuerung über ein mobiles Endgerät. Ein *Interaction Service* übersetzt die semantisch beschriebenen Ereignisse, Zustände und Zustandsänderungen innerhalb der *Intelligenten Umgebung* in geeignete Transportprotokolle zur Kommunikation mit den jeweiligen Benutzerschnittstellen. Kernbestandteil eines *Interaction Services* ist somit eine Komponente zur Interpretation der in Kapitel 8 beschriebenen semantischen Konzepte und zur Transformation dieser Konzepte in ein geeignetes Kommunikationsprotokoll. In der Regel sind *Interaction Services* dazu in der Lage, alle kommunikativen Akte als *Input* zu empfangen und gegebenenfalls an den Benutzer weiterzuleiten. Hierdurch wird sichergestellt, dass der Benutzer eine transparente Sicht auf die Abläufe innerhalb der Umgebung erhält. Als *Output* übermitteln *Interaction Services* sowohl *<request>* als auch *<propose>* Akte. Abbildung 7.8 zeigt die Abläufe innerhalb eines *Interaction Services*.

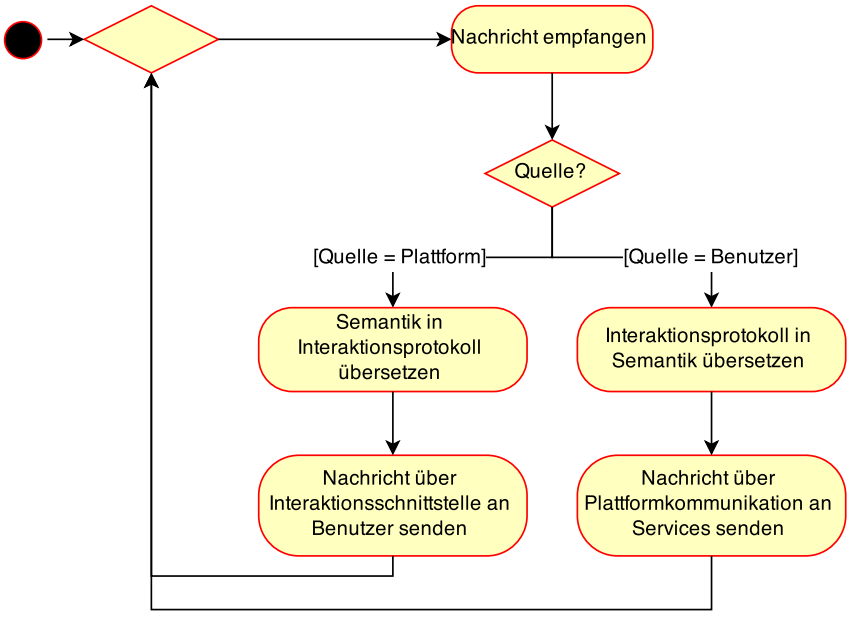


Abbildung 7.8: Übersicht über die Abläufe innerhalb eines *Interaction Services*.

Ein zentraler *Interaction Service* ist der *URC Interaction Service*, welcher als integraler Bestandteil des vorgeschlagenen Plattformkonzepts betrachtet wird. Der *URC Interaction Service* stellt die Integration der standardisierten *User Interface Socket* (UIS) Beschreibungen dar. Hierzu werden die in diesem Kapitel beschriebenen kommunikativen Akte sowie die im nächsten Kapitel vorgestellte semantische Modellierung auf die entsprechenden UIS-Elemente abgebildet, wodurch eine Kompatibilität mit den bereits verfügbaren auf URC aufbauenden Benutzerschnittstellen sichergestellt werden kann. Abschnitt 9.5.7 gibt einen detailierten Einblick in den *URC Interaction Service*.

7.5.5 Persistence Service

Der *Persistence Service* stellt eine weitere Unterklasse der *Smart Services* dar und ist für die Speicherung kontextbezogener Daten sowie, je nach Anwendungsfall, für die Speicherung der gesamten Kommunikation innerhalb der Plattform verantwortlich. Ein *Persistance Service* dient somit in der Regel den *Assistance Services* als *Backend Service* für anfallende *Big Data* und *Smart Data*. Er verfügt über zwei kommunikative Akte. Einerseits können durch *<inform>* Akte Zustands- und Kontextinformationen zur Speicherung übermittelt werden, welche andererseits über *<requeset>* Akte wiederum angefordert werden können. Abbildung 7.9 zeigt die Abläufe innerhalb eines *Persistance Services*.

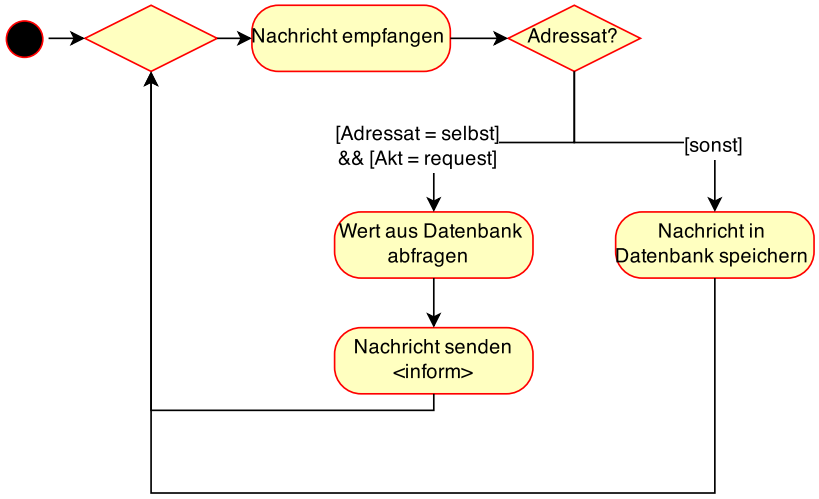


Abbildung 7.9: Übersicht über die Abläufe innerhalb eines *Persistance Services*.

7.6 Kooperation von Smart Services

Abchiche-Mimouni et al. (2013) beschreiben das Bilden von kooperativen Koalitionen von autonomen Softwarekomponenten als notwendige Voraussetzung

für die Realisation von *Intelligenten Umgebungen*. Wie eingangs erwähnt, soll hierbei ein inkrementeller Ansatz verfolgt werden. Durch das Hinzufügen neuer Services werden für den Benutzer neue Mehrwertdienste geschaffen, welche auf den bereits vorhandenen Services aufsetzen oder diese wiederum beeinflussen. In der vorliegenden Arbeit wird eine dynamische Verkettung von *Smart Services* entsprechend ihrer *Input* und *Output* Kategorien vorgestellt. Betrachtet man wiederum das bereits eingeführte Beispiel der automatischen Lichtsteuerung eines Raumes in Abhängigkeit zu den sich darin aufhaltenden Personen, so ist, wie bereits erwähnt, die Voraussetzung für den Einsatz dieses *Assistance Services* das Vorhandensein von mindestens zwei weiteren *Smart Services*. Neben einem *Assistance Service*, der feststellt, ob sich Personen innerhalb eines Raumes befinden (Krüger et al., 2014), muss zudem ein *Actuator Service* zur Lichtsteuerung für den entsprechenden Raum vorhanden sein. Der *Assistance Service* zur Überwachung der Personen innerhalb eines Raumes hängt seinerseits von einem *Sensor Service* ab, zum Beispiel ein Bewegungsmelder am Eingang des jeweiligen Raumes oder ein System zum Verfolgen und zur Aufenthaltsbestimmung einzelner Personen. Durch die Verkettung der verschiedenen Services, wobei jeder einzelne bereits voll funktional ist, kann in mittels der personenabhängigen Lichtsteuerung ein neuer Mehrwertdienst geschaffen werden. Abbildung 7.10 verdeutlicht das Prinzip der dynamischen Verkettung von *Smart Services* anhand des beschriebenen Beispiels.

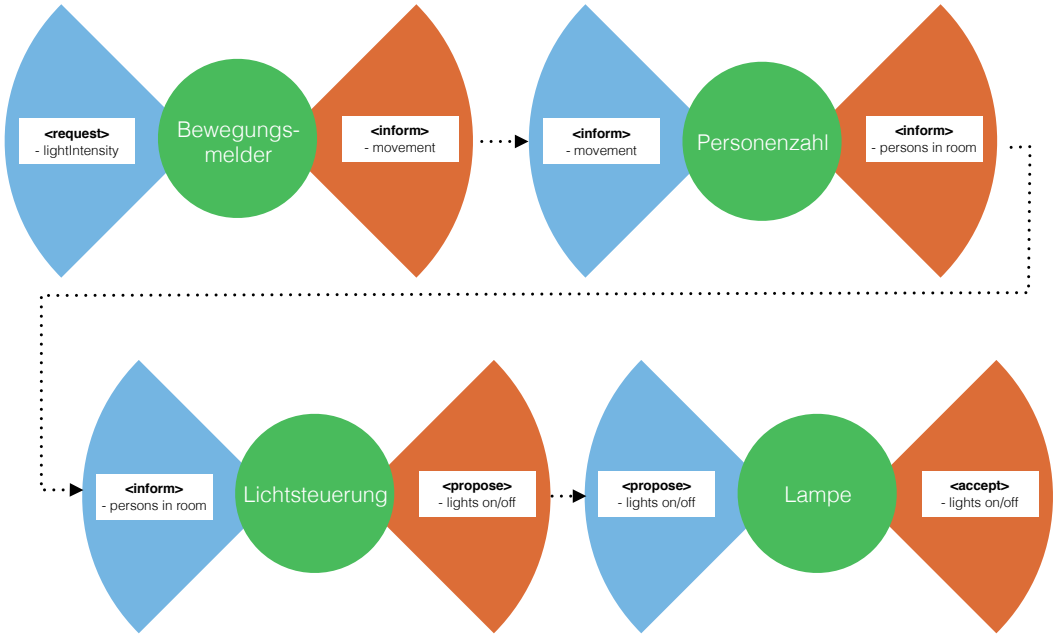


Abbildung 7.10: Überblick über die Kombination neuer Mehrwertdienste durch Kooperation und Verkettung von *Smart Services*.

7.7 Multiagentensystem als Grundlage für Smart Services in Intelligenten Umgebungen

Die Entwicklung von *Intelligenten Umgebungen* gliedert sich grundsätzlich in zwei unterschiedliche Entwicklungsprozesse (Britz et al., 2014). Auf der einen Seite beschäftigen sich *Middlewareplattformen* mit der Integration und Anwendbarkeit von unterschiedlichen Applikationen innerhalb der Umgebung. Eine wichtige Aufgabe ist hierbei das Bereitstellen von Schnittstellen zur Interaktion mit der Umgebung. Auf der anderen Seite stehen intelligente Dienste und Anwendungen, welche primär das Ziel haben, den Benutzer zu unterstützen oder ihm geeignete Mehrwerte anzubieten. In Kapitel 5.3 wurden bereits Beispiele für *Middlewareplattformen* im Kontext von *Intelligenten Umgebungen* aufgeführt und gegeneinander abgegrenzt, was letztlich in der in Kapitel 6 beschriebenen Implementierung des *Universal Control Hubs* mündete. In Kapitel 5.4 wurden hingegen Beispiele für die Anwendung von *Multiagentensysteme* innerhalb *Intelligenter Umgebungen* beschrieben. Die Beispiele haben gezeigt, dass sich *Multiagentensysteme* (siehe Kapitel 4) grundsätzlich zur Realisierung von intelligenten Anwendungsszenarien eignen. Hierbei ist allerdings auch festzustellen, dass viele der skizzierten Systeme auf konkrete Anwendungsfälle spezialisiert sind und oftmals nur eine bestimmte Problemstellung adressieren. Ein Ziel der vorliegenden Arbeit ist es nun, das erarbeitete allgemeine Plattformkonzept zur Integration von intelligenten Diensten in der zugrunde liegenden Umgebungen mit Hilfe eines *Multiagentensystems* zu realisieren und mit Konzepten der in Kapitel 6 eingeführten URC-Infrastruktur zu vereinen. In Britz et al. (2014) wurde hierzu mit der *UCH-Bridge* ein erster prototypischer Ansatz zur Überbrückung der Kluft zwischen *Middlewareplattformen* und intelligenten Diensten vorgestellt und die Integration eines auf JADE basierenden *Multiagentensystems* in die UCH-Implementierung beschrieben. Die Integration des *Multiagentensystems* funktioniert hierbei grundsätzlich in zwei Richtungen. Erstens wird den intelligenten *Agenten* der Zugriff auf die an das UCH angeschlossene *Targets* ermöglicht. Hierzu agiert der *Agent* als *Controller* und ist dazu in der Lage, die angeschlossene *Sensoren* zu überwachen und die *Aktuatoren* zu kontrollieren. Zweitens können *Agenten* als *Targets* agieren und ermöglichen somit die Bereitstellung einer abstrakten Benutzerschnittstelle (UIS) zur Kontrolle und Manipulation des *Agenten*. Für beide Richtungen ist jeweils ein dedizierter *Agent* verantwortlich, der wiederum in einem eigenen Agentencontainer angesiedelt ist. Beide Agentencontainer sind mit dem Hauptcontainer des eigentlichen Agentensystems verbunden und erlauben einen Informationsaustausch über ACL-Nachrichten. Der *UCH-Agent* erlaubt anderen *Agenten* den Zugriff auf Funktionen des UCH und der angeschlossenen Geräte, indem er die entsprechenden Services innerhalb des *Multiagentensystems* zur Verfügung stellt. Die Kommunikation basiert auf einer *URC-Ontology*, welche in erster Linie die Funktionalitäten des URC-HTTP-Protokolls abbildet. Der *UIS-Agent* erlaubt es beliebigen *Agenten*, basierend auf dieser Ontologie, eine abstrakte UIS-Beschreibung ihrer Funktionalitäten zu erzeugen und innerhalb des UCH verfügbar zu machen. Eine detaillierte Beschreibung der Abläufe in-

nerhalb der Architektur ist in Britz et al. (2014) zu finden. Der gezeigte Ansatz hat eine in erster Linie technische Verschmelzung von *Middlewares* und intelligenten Dienstleistungen innerhalb eines *Smart Homes* betrachtet. Ein wichtiger Baustein ist die Realisierung eines *Smart Services* als dedizierter *Softwareagent* und die anschließende Integration in eine *Smart Home* Infrastruktur. Abbildung 7.11 gibt einen Überblick über die Architektur der vorgestellten *UCH-Bridge*.

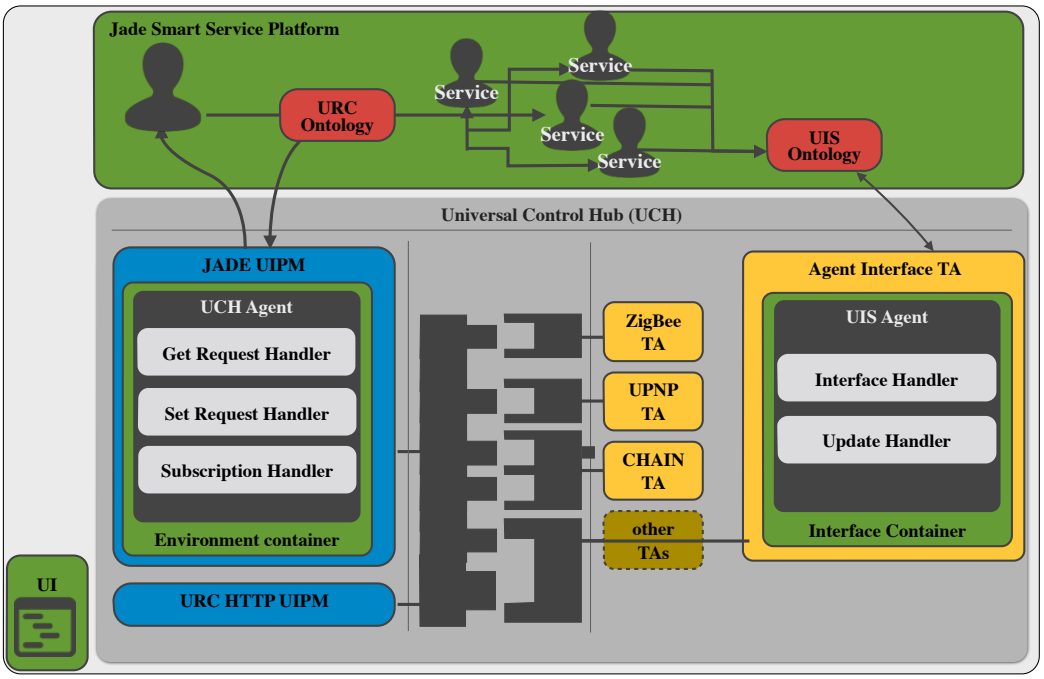


Abbildung 7.11: Überblick über die *UCH-Bridge* zur Integration eines *Multiagentensystems* in die UCH-Infrastruktur.

Der beschriebene Ansatz entspricht noch nicht dem zu Beginn dieses Kapitels vorgestellten einheitlichen Plattformkonzepts zur Integration von *Smart Services*, welches zudem auch die Kommunikation unterschiedlicher *Smart Services* und deren Verkettung betrachtet. Es lassen sich jedoch die folgenden Eigenschaften eines *Multiagentensystem* zur Integration von *Smart Services* in eine *Smart Home* Umgebung ableiten:

- Ein *Softwareagent* repräsentiert eine Dienstleistung innerhalb der *Intelligenten Umgebung*. Diese kann atomar sein oder erst durch die Kombination mit anderen Diensten realisiert werden.
- Ein *Softwareagent* muss dabei in der Lage sein, mit seiner Umgebung zu kommunizieren, das heißt deren Zustand zu erkennen und gegebenenfalls zu manipulieren.
- Dem Benutzer müssen Möglichkeiten geboten werden, mit dem Gesamtsystem sowie mit einzelnen *Agenten* auf unterschiedliche Art und Weise zu interagieren.

- Die zugrunde liegende Plattform sollte eine einheitliche Komponentenstruktur mit klar definierten Verantwortlichkeiten besitzen, was sowohl die Weiterentwicklung des Konzeptes als auch die Anwendbarkeit durch den Endnutzer erleichtert (siehe Szenarienbeschreibugn in Abschnitt 7.10).

Abbildung 7.12 gibt einen Überblick über die Zielarchitektur der Plattform. Die Plattform stellt eine Weiterentwicklung der zuvor beschriebenen *UCH-Bridge* dar und basiert auf den in Abschnitt 7.5 definierten Typen von *Smart Services* und ist als verteiltes *Multiagentensystem* konzipiert. Jeder *Agent* entspricht einem bestimmten *Smart Service*. Die Kommunikation ist nicht zentral gesteuert, sondern als Peer-to-Peer Verbindung entsprechend den zuvor definierten Kommunikations- und Kooperationsmechanismen realisiert.

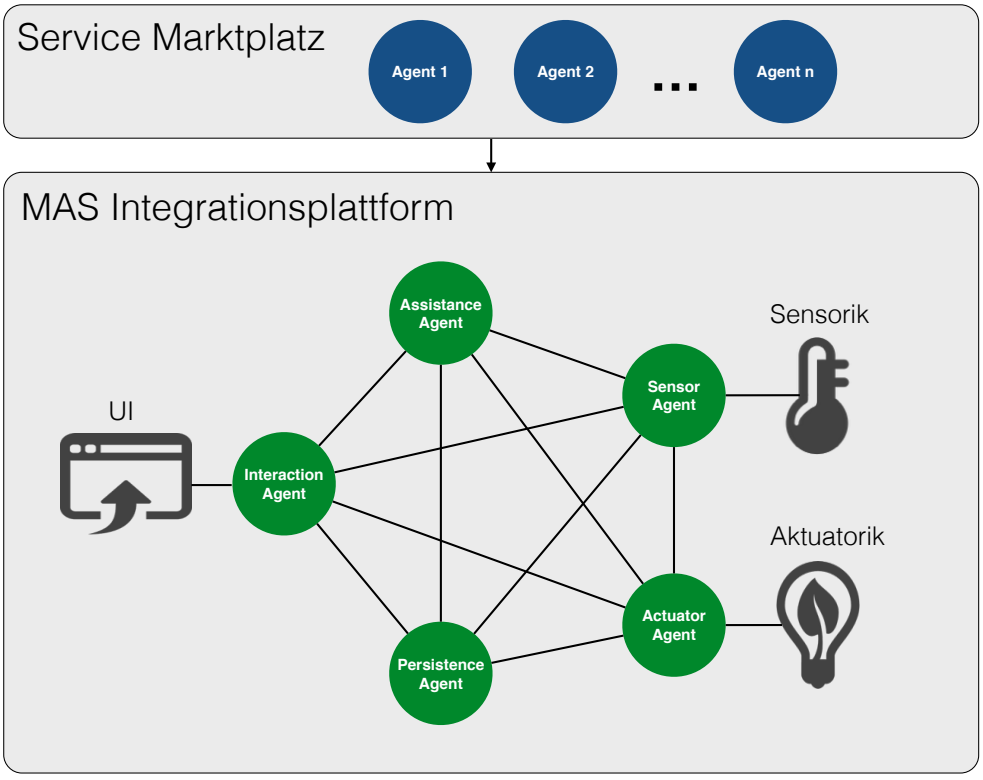


Abbildung 7.12: Übersicht über die abstrakte Zielarchitektur einer auf einem *Multiagentensystem* basierenden Integrationsplattform für *Smart Services*.

Im Vergleich zum vorigen prototypischen Ansatz ist das Gesamtsystem als einheitliches *Multiagentensystem* realisiert worden. Der *Interaction Agent* ersetzt den *UIS-Agent*, wohingegen *Sensor Agent* und *Acutator Agent* die Funktionalitäten des *UCH-Agents* abdecken. *Assistance Agent* und *Persistence Agent* stellen Mehrwertdienste zur Benutzerassistenz und zur Speicherung der Daten bereit. Über einen *Service Marktplatz* können neue Mehrwertdienste geschaffen und in Form von *Smarten Service Agenten* bereitgestellt werden.

Die vorgestellte Zielarchitektur bietet zwar eine einfache Strukturierung, ist zugleich aber vielseitig einsetzbar. Zur Verdeutlichung erläutern die beiden

folgenden Abschnitte, wie das vorgestellte Konzept sowohl in die *Smart Service Welt* als auch in die OpenURC-Architektur einzuordnen ist.

7.8 Einordnung in die Smart Service Welt

Wie in Kapitel 3 ausführlich beschrieben, gliedert sich die Architektur der *Smart Service Welt* in die vier dedizierte Schichten *Serviceplattform*, *Software-definierte Plattform*, *Vernetzte physische Plattform* und schließlich die *Technische Infrastruktur*. Abbildung 7.13 skizziert die zugrunde liegende Architektur in Kombination mit dem vorgestellten Plattformkonzept.

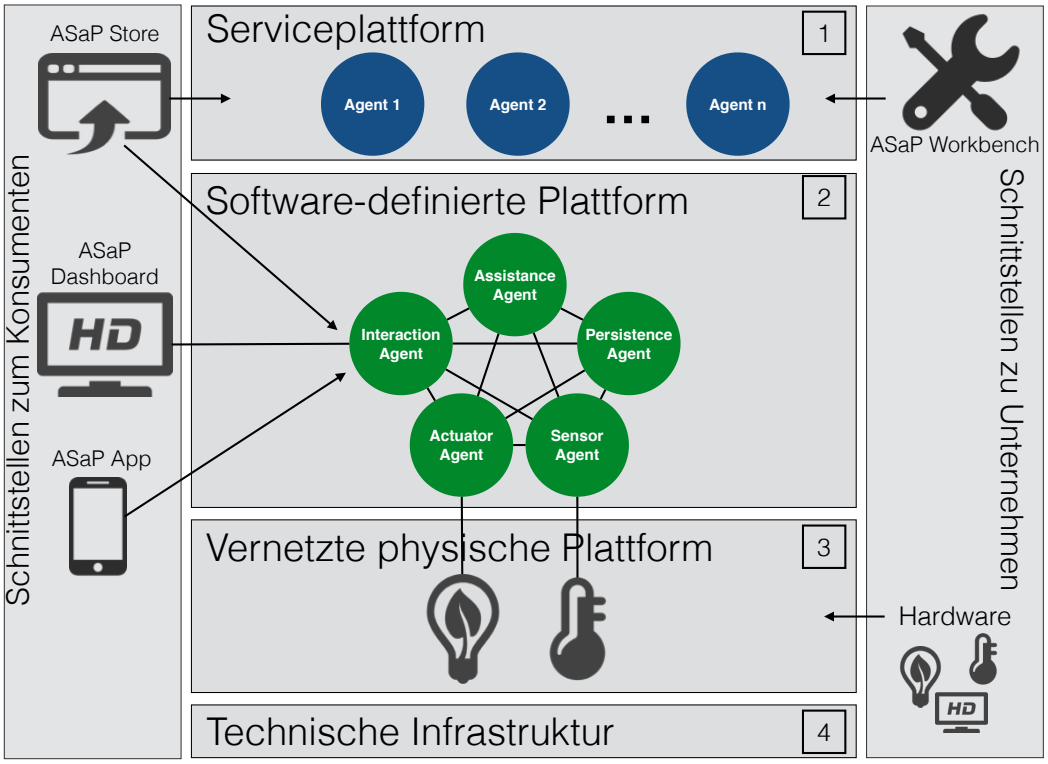


Abbildung 7.13: Einordnung des Plattformkonzeptes in die Architektur der *Smart Service Welt*.

Die Kernkomponenten der Integrationsplattform sind auf der Ebene der *Software-definierten Plattform* angesiedelt. Durch die entsprechenden *Sensor Agents* und *Actuator Agents* können *Smart Products* von unterschiedlichen Herstellern auf der Ebene der *Vernetzten physischen Plattform* integriert werden. *Interaction Agents* bieten jeweils Konsumentenschnittstellen zur Kontrolle und zur Überwachung der Umgebung. Auf der übergeordneten *Serviceplattform* können neue Mehrwertdienste erstellt und über die definierten semantischen Schnittstellen mit anderen Diensten kombiniert werden. Diese Ebene stellt somit eine Kollaborationsplattform im Sinne des *Smart Service* Ansatzes dar, sowohl für Endkunden als auch für dienst- und hardwareanbietende Unternehmen und Entwickler. Auf Endkundenseite wird durch ein Marktplatzportal

der Zugriff auf die geschaffenen Mehrwertdienste in Form von *Smart Service Agenten* geboten. Auf Unternehmens- und Entwicklerseite wird durch eine integrierte Entwicklungsumgebung (IDE) eine durchgängige Werkzeugkette zur Unterstützung der Serviceentwicklung bereitgestellt.

7.9 Einordnung in die OpenURC-Architektur

Wie in Kapitel 6 beschrieben setzt sich die UCH-Architektur aus drei dedizierten und modular aufgebauten Schichten zusammen. Abbildung 7.14 skizziert die zugrunde liegende Architektur in Kombination mit dem vorgestellten Plattformkonzept.

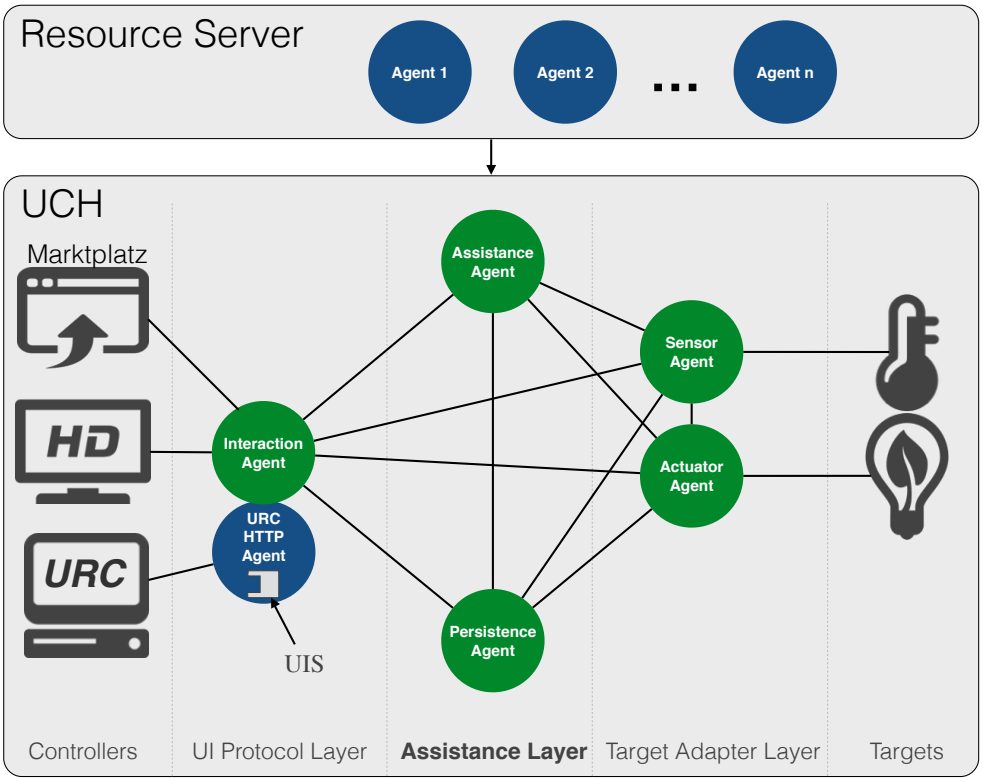


Abbildung 7.14: Einordnung des Plattformkonzeptes in die OpenURC-Architektur.

Die Kernkomponenten der Integrationsplattform sind auf alle Schichten verteilt. Die *Sensor Agents* und *Actuator Agents* übernehmen die Funktionalitäten der *Target Adapters* und sind somit für die Integration der *Targets* verantwortlich. Die *Interaction Agents* ersetzen die *User Interface Protocol Modules* und ermöglichen die Anbindung unterschiedlicher *Controller*. Da die Kommunikation der Agenten auf einer semantischen Modellierung basiert, entfällt der *Socket Layer* als plattforminternes Kommunikationsprotokoll. Die abstrakte Benutzerschnittstellenbeschreibung (UIS) findet sich in einem speziellen *Interaction Agent* wieder. Dieser übersetzt die plattforminternen semantischen Modelle in eine URC konforme UIS-Beschreibung und stellt sie

möglichen URC Clients zur Verfügung. Anstelle des *Socket Layers* tritt nun ein neuer *Assistance Layer*, welcher für die Einbindung der *Smart Service Agenten* verantwortlich ist. Der *Resource Server* repräsentiert den Service Marktplatz und verwaltet die global verfügbaren *Smart Service Agenten*. Er bietet wiederum Schnittstellen für ein entsprechendes Marktplatzportal und eine integrierte Entwicklungsumgebung.

7.10 Anwendungsszenarien

Zur Motivation der vorgestellten Architektur werden im Folgenden zwei Beispielszenarien zur Anwendung der entwickelten Konzepte vorgestellt, sowohl aus Endkunden- als auch aus Dienstanbieter- und Entwicklersicht.

7.10.1 Szenario 1: Auffinden und Installation

Das erste Szenario zeigt das Auffinden und die Installation neuer digitaler Mehrwertdienste basierend auf der entwickelten Integrationsplattform für *Smart Services*, aus Endnutzer- beziehungsweise Konsumentensicht. Es wird hierbei davon ausgegangen, dass der Kunde sein *Smart Home* mit einer initialen Version der Plattform ausgestattet hat. Eine minimale technische Infrastruktur wäre beispielsweise ein Computer mit Internetzugang innerhalb eines lokalen Funknetzes. Über einen speziellen *Interaction Agent* wird dem Kunden ein personalisierter Zugriff auf den *Smart Service Marktplatz* angeboten. Über ein Portal kann der Kunde den aktuellen Zustand seiner Umgebung verwalten sowie neue, auf seine aktuelle Instrumentierung passende Services finden. Hierzu werden die semantischen Abhängigkeiten der *Input* und *Output* Beziehungen der verfügbaren *Smart Services* analysiert und diese im Anschluss entsprechend gefiltert. Der Marktplatz kann so dem Kunden aufzeigen, welche Voraussetzungen für bestimmte Services erfüllt werden müssen oder sogar Empfehlungen über neue und auf die Instrumentierung passende *Smart Services* abgeben. Auf diese Art und Weise kann ein transparentes Marktmodell den Kunden schon beim Auffinden von für ihn geeigneten *Smart Services* und somit beim Bilden von Serviceketten unterstützen. Eine Servicekette aus der Energieforschung könnte zum Beispiel durch das folgende Szenario abgebildet werden: Damit lokale Energieanbieter ihre Stromkontingente auf dem Energiemarkt effizienter und kostengünstiger handeln können, ist es zum Beispiel notwendig, dass der Energieverbrauch von Privathaushalten besser prognostiziert wird (siehe Kapitel 11.2.3). Durch den Einsatz von *Smart Metern* mit einen hohen Messfrequenz können bereits genauere Daten für die Stromerzeuger erfasst werden. Darauf aufbauend können für die jeweiligen Haushalte individuelle Prognosedienste in Form von *Assistance Services* implementiert werden. Die Prognose wird umso genauer, je mehr Informationen über den zugrunde liegenden Haushalt und dessen Bewohner vorliegen. Durch geeignete Verfahren zur Aktivitätserkennung können regelmäßige Handlungsmuster der Bewohner erkannt werden und so die Genauigkeit der Energieprognose verbessert werden.

Ein *Assistance Service* zur Erkennung von Benutzeraktivitäten basiert wiederum auf einer Vielzahl von unterschiedlichen *Sensor Services*, *Actuator Services* und *Persistence Services*. In Kapitel 11.5.2 wird hierzu ein *Assistance Service* auf der Grundlage eines soundbasierten Analyseverfahrens vorgestellt. Die Ergebnisse der Energieprognose können wiederum durch entsprechende *Interaction Services* den Bewohnern der Haushalte mitgeteilt werden, beziehungsweise können durch weitere *Assistance Services* auch Handlungsempfehlungen zum Einsparen von Energiekosten gegeben werden. Darüber hinaus können *Smart Services* gezielte Installationsanweisungen zur Verfügung stellen, welche dem Kunden während eines eventuell notwendigen Installationsprozesses die benötigten Hilfestellungen anzeigen.

7.10.2 Szenario 2: Entwicklung und Wirkbetrieb

Das zweite Szenario betrachtet die Entwicklung und die Bereitstellung von *Smart Services* aus Unternehmens- beziehungsweise Entwicklersicht. Im Fokus steht hierbei die optimale Unterstützung während des gesamten Entwicklungszyklus eines *Smart Service* durch den Einsatz einer integrierten Entwicklungsumgebung. Aufbauend auf der semantischen Modellierung der Umgebung können durch den Einsatz von unterschiedlichen Editoren, Wizards und Codegeneratoren schnell und unkompliziert *Smart Service Templates* erstellt werden, welche dann durch den Entwickler auf den jeweiligen Anwendungsfall angepasst werden können. Durch die direkte Anbindung an den Marktplatzes können die entwickelten Services direkt veröffentlicht und so den Endkunden zur Verfügung gestellt werden. Weiterhin können statistische Auswertungen der semantischen *Smart Service* Beschreibungen auf dem Marktplatz dem Unternehmen Anhaltspunkte für potentiell vielversprechende Produkte liefern. So lässt zum Beispiel eine hohe Zahl von auf dem Marktplatz verfügbaren *Actuator Services* zur Lichtsteuerung erkennen, dass die Realisierung von *Assistance Services* zur automatischen Lichtsteuerung zu einer hohen Kundenakzeptanz und -resonanz führen könnte.

7.11 Zusammenfassung und Fazit

Das Kapitel hat gezeigt, wie digitale Mehrwertdienste in Form von *Smart Services* anhand eines einheitlichen Plattformkonzeptes in eine *Intelligente Umgebung* integriert werden können. Hierzu wurde ein Architekturmodell entwickelt, mit dessen Hilfe sich *Smart Services* als *Softwareagenten* innerhalb eines *Multiagentensystems* realisieren lassen. Das Modell umfasst Konzepte und Lösungen zum strukturellen Aufbau, zur Typisierung unterschiedlicher *Smart Services* sowie die entsprechenden Kommunikations- und Kooperationsmechanismen in *Smart Home* Szenarien. Darüber hinaus wurde gezeigt, wie sich das beschriebene Plattformkonzept sowohl in die *Smart Service Welt* als auch in die OpenURC-Architektur einordnen und darin abbilden lässt. Abschließend dienen zwei Anwendungsszenarien als roter Faden für die nächsten Kapitel und motivieren somit die weitere Vorgehensweise innerhalb der Arbeit.

Modelle sollten sich bemühen, dem Porträt ähnlich zu sehen.

Salvador Dali

8 Semantische Repräsentation von Intelligenten Umgebungen

8.1 Übersicht

In diesem Kapitel wird auf den bereits vorgestellten verwandten Arbeiten in Kapitel 5.5 aufgebaut und eine semantische Repräsentation von *Smart Services* im Kontext von *Intelligenten Umgebungen* beschrieben (siehe auch Eberhart (2003)). Diese bildet die Grundlage für die im nächsten Kapitel vorgestellte Implementierung einer *Agentenbasierten Smart Service Plattform (ASaP)*.

Das übergeordnete Ziel ist hierbei, eine leichtgewichtige und semantische Repräsentation bereitzustellen, welche die Umgebung mit den darin enthaltenen Objekten, den möglichen Ereignissen (*Events*) sowie den zur Verfügung stehenden Diensten (*Smart Services*) erfassen und modellieren kann. Besonders die jeweiligen Abhängigkeiten der *Services* untereinander sollen berücksichtigt werden. Die Modellierung dieser Abhängigkeiten entspricht der inhaltlichen Struktur eines *Smart Services*, bestehend aus *Input* und *Output* Abhängigkeiten (siehe Kapitel 7.3). In Kapitel 7.4 wurde der zugrunde liegende Kommunikationsmechanismus für *Smart Services* auf der Basis unterschiedlicher kommunikativer Akte eingeführt. In den folgenden Abschnitten werden durch die Einführung einer initialen und erweiterungsfähigen *ASaP-Ontology* schließlich die Inhalte dieser Akte anhand von semantischen Modellen detailliert beschrieben.

In Abschnitt 8.2 wird zunächst eine allgemeinen Einführung in das *Eclipse Modeling Framework* (EMF) gegeben und dessen Verwendung als Modellierungssprache innerhalb des in der vorliegenden Arbeit entwickelten Plattformkonzepts motiviert. Das erarbeitete EMF Modell stellt zudem die Grundlage der in Kapitel 10 beschriebenen Werkzeugkette dar. In Abschnitt 8.3 wird eine semantische Repräsentation der *Smart Services* und der zuvor definierten *Smart Service* Typen erarbeitet. Darüber hinaus wird der Zusammenhang zu den möglichen *Events* innerhalb einer *Intelligenten Umgebung* sowie den entsprechenden Kontextinformationen hergestellt. Abschnitt 8.4 beschreibt die Modellierung der *Events* im Detail und unterscheidet zwischen Informationen und Zustandsveränderungen innerhalb der Umgebung. Abschließend werden in Abschnitt 8.5 unterschiedliche Modelle zur Repräsentation des Kontext einer *Intelligenten Umgebung* beschrieben. Hierbei wird im Wesentlichen zwischen

räumlichen (*Spatial*), physischen (*Physical*) und benutzerspezifischen (*Behavioural*) Modellen unterschieden. Die folgende Übersicht fasst die Anforderungen an die *ASaP-Ontology* und die verwendete Modellierungssprache nochmals zusammen:

Modellierungsmächtigkeit
- Repräsentation der *Smart Services* innerhalb einer Umgebung - Repräsentation der Kontextinformationen innerhalb einer Umgebung - Repräsentation der Ereignisse innerhalb einer Umgebung
Modellierungssprache
- Leicht verständliche Repräsentationssprache - Leichte Integrierbarkeit in moderne Entwicklungsumgebungen - Leichte Erweiterbarkeit durch geeignete Werkzeugunterstützung

8.2 Exkurs: Eclipse Modeling Framework (EMF)

In der vorliegenden Arbeit wurde das Eclipse Modeling Framework (EMF) als Modellierungssprache zur Repräsentation einer *Intelligenten Umgebung* eingesetzt. Steinberg et al. (2009) beschreiben EMF pragmatisch betrachtet als Modellierungsframework, welches auf den Funktionalitäten der Eclipse Plattform aufbaut und diese optimal ausnutzt. EMF erlaubt die Modellierung in drei unterschiedlichen Formaten: XML, UML und Java. Aufbauend auf der erstellten Modellierung können die anderen Formate durch den Einsatz entsprechender Codegenerierungswerkzeuge erzeugt werden. EMF ist ein wichtiger Bestandteil der auf Eclipse basierenden *Modellgetriebenen Softwareentwicklung* (MDSD). Um ein spezifisches Modell erstellen zu können, ist eine allgemeine Terminologie zur Beschreibung des Modells erforderlich. Man benötigt also wiederum ein Modell, um ein EMF Modell beschreiben zu können. Ein solches abstraktes Modell wird als Metamodell bezeichnet. EMF verwendet Ecore als zugrunde liegendes Metamodell. Um ein EMF-Modell zu erstellen, werden die folgenden vier Klassen des Ecore-Metamodells benötigt:

- **EClass:** Die *EClass* Klasse repräsentiert eine modellierte Klasse innerhalb des Modells. Sie verfügt über einen Namen, mehrere Attribute und Referenzen zu anderen Klassen.
- **EAttribute:** Die *EAttribute* Klasse modelliert ein Attribut. Sie besitzt einen Namen und eine Typbeschreibung.
- **EReference:** Die *EReference* Klasse repräsentiert das Ende einer Beziehung zu einer anderen Klasse. Sie hat einen Namen, eine boolesche Variable, welche anzeigt, ob die Beziehung ein *Containment* beinhaltet, sowie eine Klasse, die das Ziel der Beziehung darstellt.

- **EDataType:** Die *EDataType* Klasse repräsentiert den Datentyp eines Attributes. Sie kann sowohl einen primitiven als auch einen objektbasierten Datentyp darstellen.

Abbildung 8.1 zeigt einen Auszug aus dem Ecore Metamodell[1].

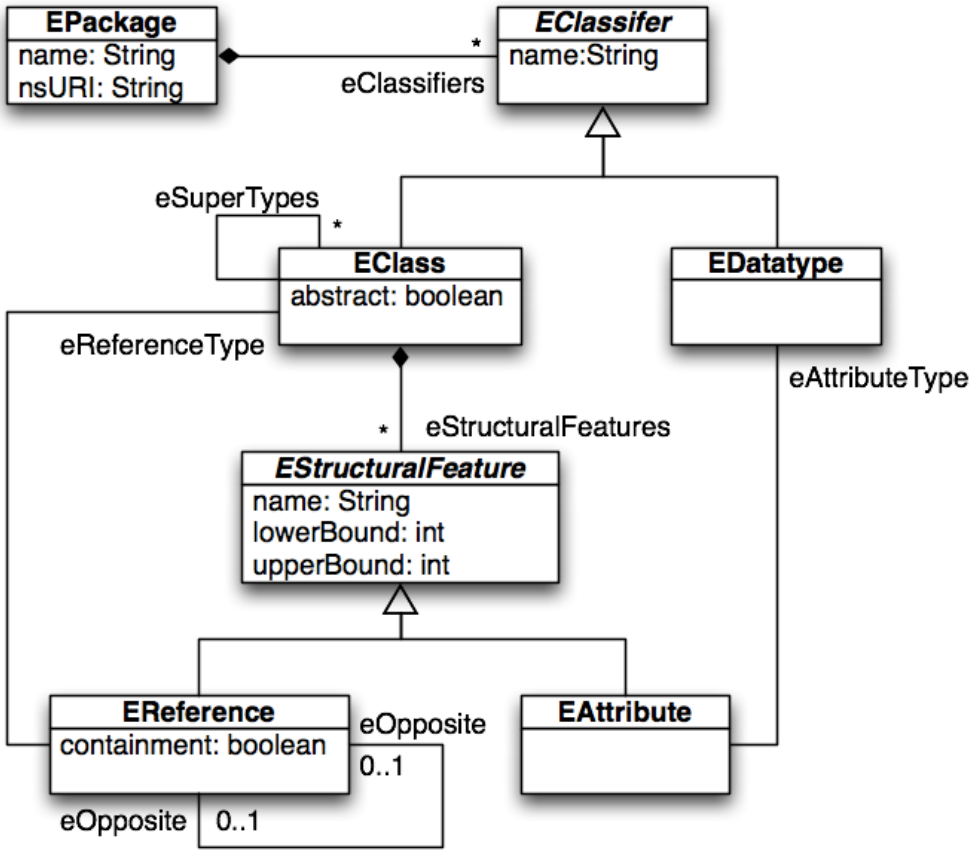


Abbildung 8.1: Übersicht über das zugrunde liegende EMF Ecore Metamodell (aus Steinberg et al. (2009).

Hillairet et al. (2008) fassen einige der Unterschiede zwischen einem objektorientierten EMF Modell und der *Web Ontology Language* OWL[2] zusammen:

- **Klassenzugehörigkeit:** In objektorientierten Sprachen wird ein Objekt aus genau einer Klasse instanziiert. Diese Klassenzugehörigkeit ist unveränderlich und für die gesamte Lebensdauer eines Objektes gültig. In OWL hingegen kann ein Resource ein Mitglied mehrerer Klassen sein.
- **Klassenhierarchie:** OWL Klassen können von mehreren Oberklassen abgeleitet werden. In objektorientierten Sprachen ist dies in der Regel nicht möglich. EMF unterstützt trotzdem multiple Vererbung durch die Generalisierung der Klasseninterfaces während der Codegenerierung.

[1] http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.xtext.doc%2Fcontents%2F210-emf-integration.html [Letzter Zugriff: 23.12.2014]

[2] http://www.w3.org/TR/owl-ref/ [Letzter Zugriff: 22.02.2015]

- **Strukturelle Vererbung:** In objektorientierten Sprachen erben Objekte die Attribute ihrer Oberklasse. In OWL ist diese Eigenschaft nicht enthalten, da *Properties* als eigenständig betrachtet werden.

Hillairet et al. (2008) und Schneider (2010a) beschreiben weiterhin jeweils eigene Ansätze und Konzepte, um die Unterschiede von EMF und OWL auszugleichen, beispielsweise durch den Einsatz von entsprechenden Modelltransformationen. Neßelrath und Feld (2014) haben hingegen gezeigt, dass EMF Modelle eine vergleichbare Repräsentationsmächtigkeit wie beispielsweise die von Carpenter (1992) eingeführten *Typed Feature Strutures* (TFS) besitzen und somit die Verwendbarkeit von EMF im Kontext von multimodalen Dialogsystemen motiviert. Darüber hinaus werden die semantischen Modelle der im Rahmen dieser Arbeit vorgestellten Integrationsplattform für *Smart Services* im Wesentlichen für die Wissensrepräsentation und weniger für das automatisierte Schließen verwendet, was abschließend EMF als zugrunde liegende Modellierungsmethode nahelegt.

In den folgenden Abschnitten werden die einzelnen, zugrunde liegenden Modellelemente des in Kapitel 7 eingeführten theoretischen Plattformkonzeptes auf der Basis von EMF allgemein beschrieben. Eine detaillierte und vollständige Darstellung der im Rahmen der Arbeit entstandenen Modellierung ist in Anhang B zu finden.

8.3 Repräsentation von Smart Services

Der zentrale Bestandteil der *ASaP-Ontology* ist die Repräsentation von *Smart Services*. Wie alle Klassen innerhalb der Ontologie leitet sich die `SmartService` Klasse von der Oberklasse `OntologyObject` ab. Sie enthält darüber hinaus mehrere Attribute, welche allgemeine Eigenschaften der Klasse definieren, zum Beispiel den Namen des *Services*, einen beschreibenden Text, den entsprechenden Entwickler des *Services* sowie, falls vorhanden, ein konkretes zugrunde liegendes Produkt oder einen Verweis auf eine zum *Service* kompatible Anwendung. Daneben verfügt die `SmartService` Klasse über Relationen zu den drei folgenden Klassen. Die `ServiceType` Klasse spezifiziert über eine `type` Relation zunächst den Typ des *Smart Services* (siehe Kapitel 7.5). Über die `context` Relation können dem *Smart Service* ein oder mehrere optionale Umgebungskontexte durch entsprechende `Context` Klassen zugeordnet werden (siehe Abschnitt 8.5). Zuletzt definieren die `input` und `output` Relationen die Abhängigkeiten des *Smart Services* von entsprechenden Ereignissen innerhalb der Umgebung. Ereignisse werden hierbei durch die `Event` Klasse abgebildet und stellen somit in Kombination mit den in Kapitel 7.4 definierten kommunikativen Akten sowohl Eingabe- als auch Ausgabeparameter eines *Smart Services* dar. Abbildung 8.2 zeigt das zugehörige Klassendiagramm der beschriebenen `SmartService` Klasse.

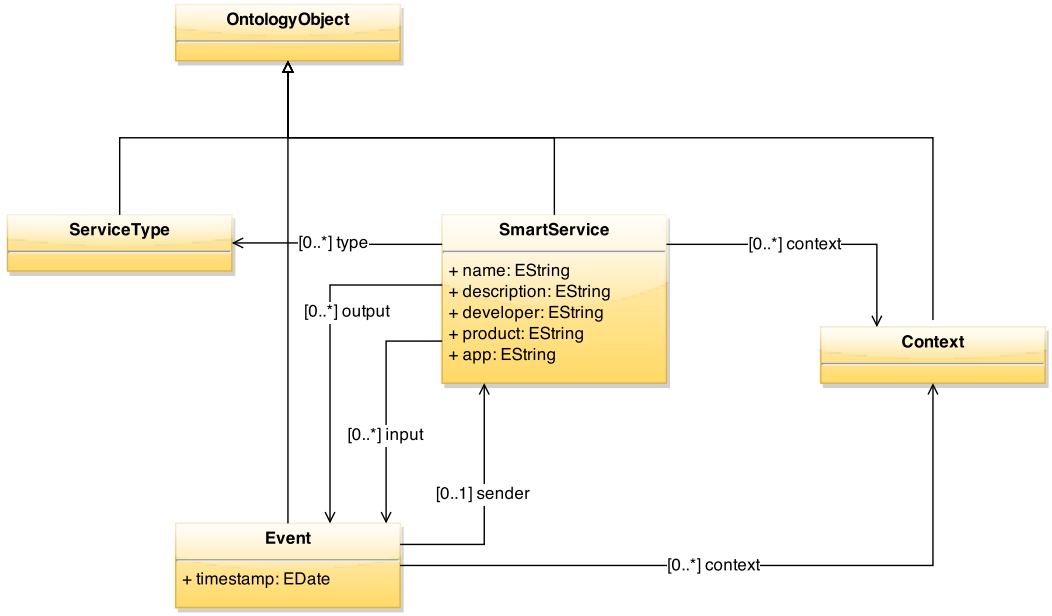


Abbildung 8.2: Auszug aus dem Klassendiagramm zur Repräsentation von *Smart Services*.

8.4 Repräsentation von Ereignissen

Wie in Kapitel 7.4 beschrieben wurde, basiert der Informationsaustausch innerhalb des in der vorliegenden Arbeit entwickelten Plattformkonzepts auf einer Teilmenge der in Kapitel 4.4 vorgestellten kommunikativen Akte. Kommunikative Akte definieren hierbei die Pragmatik, also die Wirkung einer einzelnen Informationseinheit. Die Übermittlung der Semantik, das heißt die tatsächliche Bedeutung einer Information, wird durch die Modellierung und die Einbettung von in der Umgebung vorkommenden Ereignissen in die jeweiligen kommunikativen Akte erreicht. Die entsprechende `Event` Klasse leitet sich wiederum von der Oberklasse `OntologyObject` ab. Des Weiteren lässt sich ein `Event` in zwei weitere Unterklassen ableiten. Die `Information` Klasse umfasst alle Ereignisse mit dem Zweck eines reinen Informationsaustausches, zum Beispiel Hilfestellungen (`Help`), Notfallsituationen (`Alarm`) oder `Streaming` Inhalte wie Video- und Audiosignale. Daneben umfasst die `State` Klasse alle Zustandsveränderungen innerhalb der Umgebung. Analog zu den *Smart Services* können über die `context` Relation dem `Event` zudem ein oder mehrere optionale Umgebungskontexte durch entsprechende `Context` Klassen zugeordnet werden. Ein `Event` besitzt darüber hinaus einen eindeutigen Zeitstempel um den Zeitpunkt des auslösenden Ereignisses zu definieren. Die `sender` Relation bildet schließlich die zugehörige `SmartService` Klasse ab, welche für das Auslösen des Ereignisses verantwortlich ist. Abbildung 8.3 zeigt das zugehörige Klassendiagramm der beschriebenen `Event` Klasse.

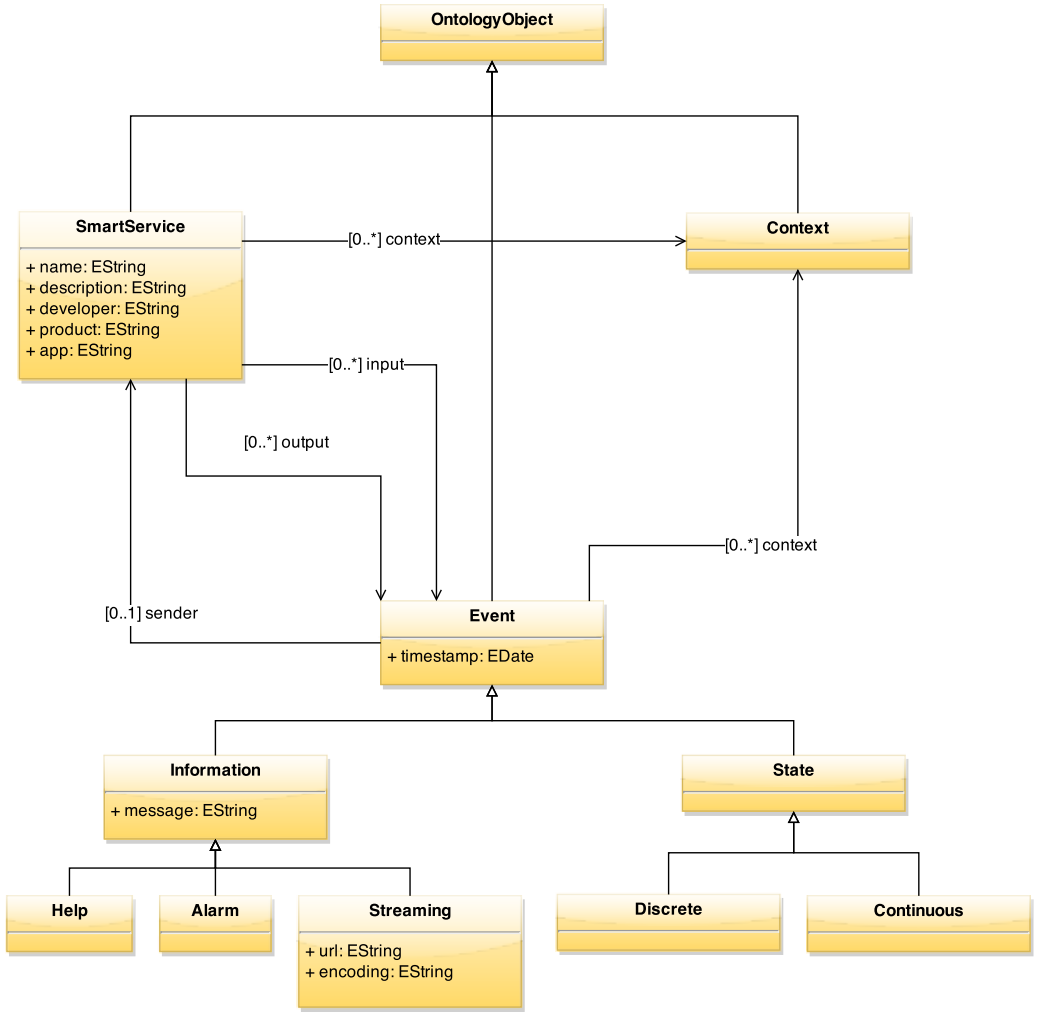


Abbildung 8.3: Auszug aus dem Klassendiagramm zur Repräsentation von Ereignissen innerhalb der Umgebung.

Ein wichtiger Bestandteil der `Event` Klasse ist die Modellierung von Zustandsveränderungen innerhalb der Umgebung. Wie bereits erwähnt, werden diese durch die `State` Klasse abgebildet. Ähnlich der in Kapitel 5.5.4 beschriebenen DogOnt Ontologie, kann die `State` Klasse wiederum in diskrete (`Discrete`) und kontinuierliche (`Continuous`) Zustandsveränderungen untergliedert werden. Die `Discrete` Klasse repräsentiert vordefinierte, statische Zustände innerhalb der Umgebung, wie zum Beispiel der `Power` Modus eines elektrischen Haushaltsgerätes oder die Öffnung eines Fensters (`OpenClosed`). Die `Continuous` Klasse umfasst dynamische Werte innerhalb eines bestimmten Wertebereiches. Hierbei wird zwischen frei wählbaren, absoluten Werten (`Absolute`), bestehend aus einem festen Wert und einem dazugehörigen Einheitsmaß, und prozentualen Wertebereichen (`Percentage`) unterschieden. Abbildung 8.4 zeigt einen Auszug aus der Klassenhierarchie der beschriebenen `State` Klasse.

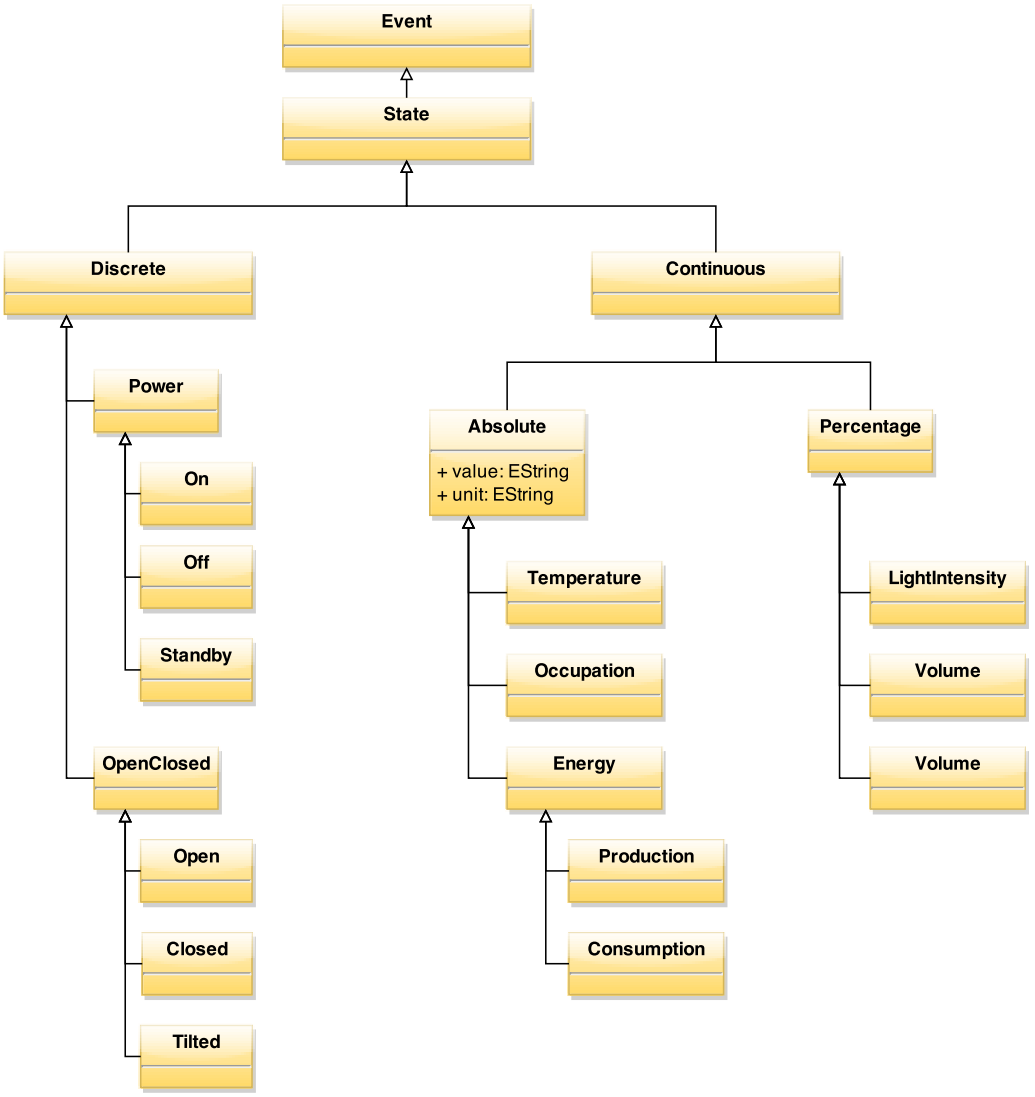


Abbildung 8.4: Auszug aus dem Klassendiagramm zur Repräsentation von Zustandsveränderungen innerhalb der Umgebung.

8.5 Repräsentation des Kontext

Die `Context` Klasse beschreibt allgemein betrachtet den Kontext einer *Intelligenten Umgebung*. Genauer betrachtet definiert die Klasse hierzu beispielsweise den gültigen Umgebungskontext eines *Smart Services* oder beschreibt die Umstände, unter denen ein bestimmtes Ereignis innerhalb der Umgebung ausgelöst wurde. Durch den Abgleich der jeweiligen `Context` Eigenschaften kann somit herausgefunden werden, ob ein konkretes `Event` für einen spezifischen `SmartService` relevant ist und folglich eine entsprechende Aktion beziehungsweise Reaktion nach sich zieht. Die `Context` Klasse lässt sich hierbei weiter untergliedern in einen räumlichen Kontext (`Spatial`), einen physi-

schen Kontext (`Physical`) sowie einen verhaltensorientierten Kontext in Bezug auf die Benutzeraktivität (`Behavioural`). Die Modellierung des `Spatial` Kontext baut hierbei auf der in Heckmann (2006) vorgestellten *Spatial Ontology* als Teil der *UbisWorld* auf. Neben den Konzepten `Building`, `Floor` und `Room` zur Einschränkung einer gegebenen *Smart Home* Umgebung werden darüber hinaus auch ein Konzept für Koordinaten im dreidimensionalen Raum (`CartesianCoordinates`) sowie ein Konzept für Aktivitätszonen (`ActivityZone`) vorgestellt. Die Modellierung von Aktivitätszonen (Frey et al., 2014) basiert auf der `Behavioural` Klasse zur Beschreibung von alltägliche Benutzeraktivitäten innerhalb eines *Smart Homes* (Reisberg et al., 2001). Die `Physical` Klasse erlaubt abschließend die Modellierung und Klassifizierung der Objekte innerhalb der physischen Umgebung, besonders im Hinblick auf die Art und die Eigenschaften des zugrunde liegenden *Smart Products*. Abbildung 8.5 zeigt das zugehörige Klassendiagramm der beschriebenen `Context` Klasse.

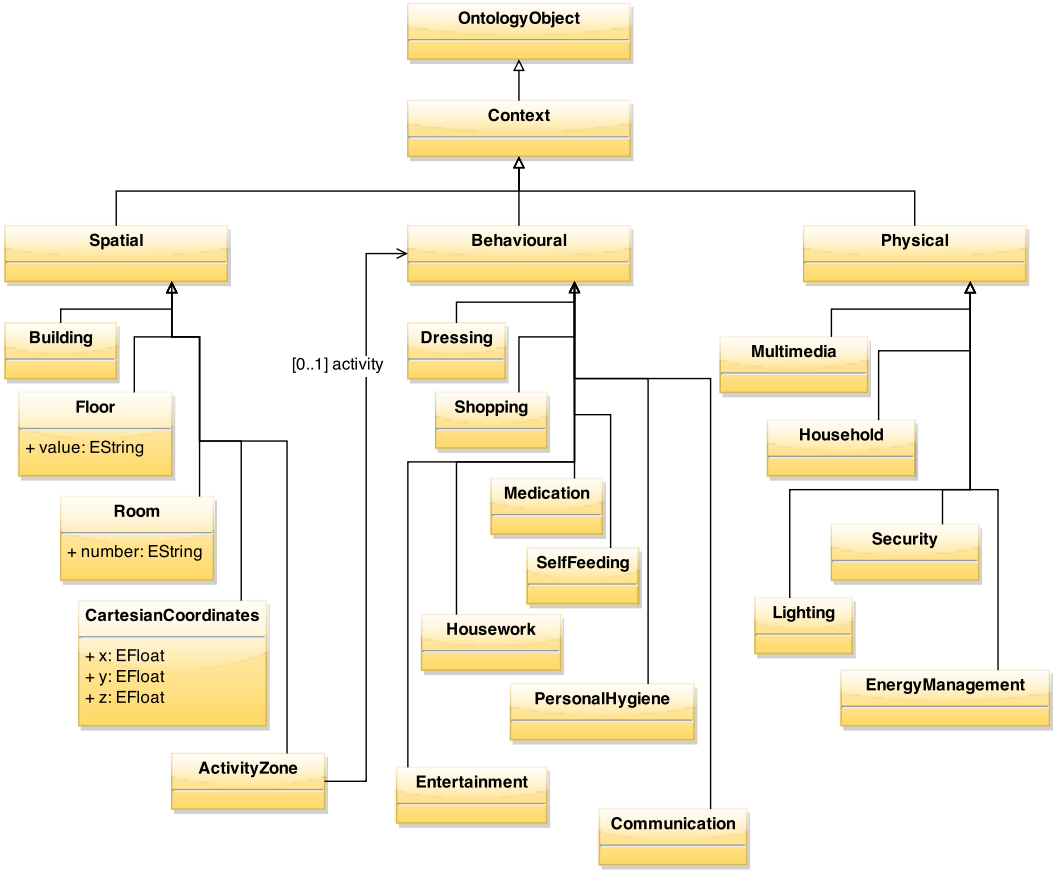


Abbildung 8.5: Auszug aus dem Klassendiagramm zur Repräsentation des Kontexts.

Die vorgestellten Konzepte bilden eine abstrakte Sicht auf die im Rahmen der Arbeit entwickelten *ASaP-Ontology*. Eine vollständige Darstellung der realisierten Modelle ist in Anhang B zu finden. Zur Verdeutlichung der beschriebenen Konzepte gibt der folgende Abschnitt ein einfaches und leicht nachvollziehbares Anwendungsbeispiel.

8.6 Anwendungsbeispiel

Im Folgenden soll eine Lichtsteuerung innerhalb eines *Smart Homes* betrachtet werden. Zur Vereinfachung wird davon ausgegangen, dass jede installierte Lampe einzeln durch einen dedizierten *Smart Service* repräsentiert wird, der jeweils das ein Ein- und Ausschalten der Lampe ermöglicht. Der zugrunde liegende *Smart Service* stellt hierbei eine konkrete Instanziierung `SmartService` Klasse dar. Der Typ der *SmartService* Instanz ist durch ein `ActuatorService` Objekt modelliert, welches über die `type` Relation dem *Service* zugeordnet wird. In Kapitel 7.5.2 wurde gezeigt, dass *Actuator Services* dazu in der Lage sind, <*propose*> Akte als Nachrichten zu empfangen und zu verarbeiten. Die möglichen semantischen Inhalt der Nachricht repräsentieren Ereignisse innerhalb der Umgebung und werden durch die `input` Relation der `SmartService` Klasse definiert. Da die Lampe ein Ein- und Ausschalten ermöglichen soll, wurde der `SmartService` so modelliert, dass er diskrete Zustandsveränderung in Form von `On` oder `Off` Instanzen als mögliche Ereignisse akzeptieren und entsprechend verarbeiten kann. Würde man beispielsweise nur die Lampen in einem bestimmten Raum ansteuern wollen, so müsste man die entsprechenden `SmartService` Instanzen derart modellieren, dass ihnen der entsprechende Raum als `Spatial` Kontext über die `context` Relation zugeordnet wird. Im Anschluss werden diese *Services* nur noch auf Ereignisse reagieren, die den gleichen *Spatial* Kontext besitzen.

8.7 Zusammenfassung und Fazit

In diesem Kapitel wurde eine semantische Repräsentation von *Smart Services* im Kontext von *Intelligenten Umgebungen* beschrieben. Diese bildet die Grundlage für die im nächsten Kapitel vorgestellten Plattformimplementierung und für die in Kapitel 10 beschriebene integrierte Entwicklungsumgebung. Zunächst wurde eine kurze Einführung in das *Eclipse Modelling Framework* gegeben und dessen Einsatz als Modellierungswerkzeug motiviert. Danach wurden die einzelnen Modellelemente anhand von *Smart Services*, den möglichen Ereignissen innerhalb eine Umgebung sowie den Kontextinformationen ausführlich beschrieben. In dem abschließenden Anwendungsbeispiel wurde der Einsatz der Modellierung anhand eines einfachen Einsatzszenarios demonstriert. Kapitel 7 und Kapitel 8 haben zwei wesentliche Bestandteile der *Smart Service Welt* vorgestellt: die technische Serviceorchestrierung und die semantische Servicebeschreibungen als Enabler-Bausteine (Kagermann et al., 2015) einer *Software-definierten Plattform*. Kagermann et al. (2015) unterteilt die Enabler in drei Kategorien, welche wie folgt definiert sind:

- Generische Enabler stellen die strukturell einfachsten Komponenten von *Software-definierten Plattformen* dar.
- Architekturmuster stellen innerhalb der *Software-definierten Plattformen* einen komplexeren Dienst dar, der sich durch das organisierte Zusam-

menspiel von interagierenden Komponenten definiert, die auch generische Enabler sein können.

- Leistungsbündel stellen in *Software-definierten Plattformen* strukturell komplexe Enabler-Komponenten dar, bei denen nicht mehr alleine die Immaterialität der Dienstleistung im Vordergrund steht, sondern die Integration von Sach- und Dienstleistungsanteilen unterstützt wird.

Das nächste Kapitel gibt einen Überblick über die praktische Implementierung der bisher beschriebenen theoretischen Ansätze anhand einer *Agentenbasierten Smart Service Plattform (ASaP)* als zentraler Bestandteil der *Software-definierten Plattform*. Abbildung 8.6 zeigt abschließend, wie mit Hilfe von *ASaP* unterschiedliche generische Enabler-Bausteine der *Smart Service Welt* realisiert werden können.

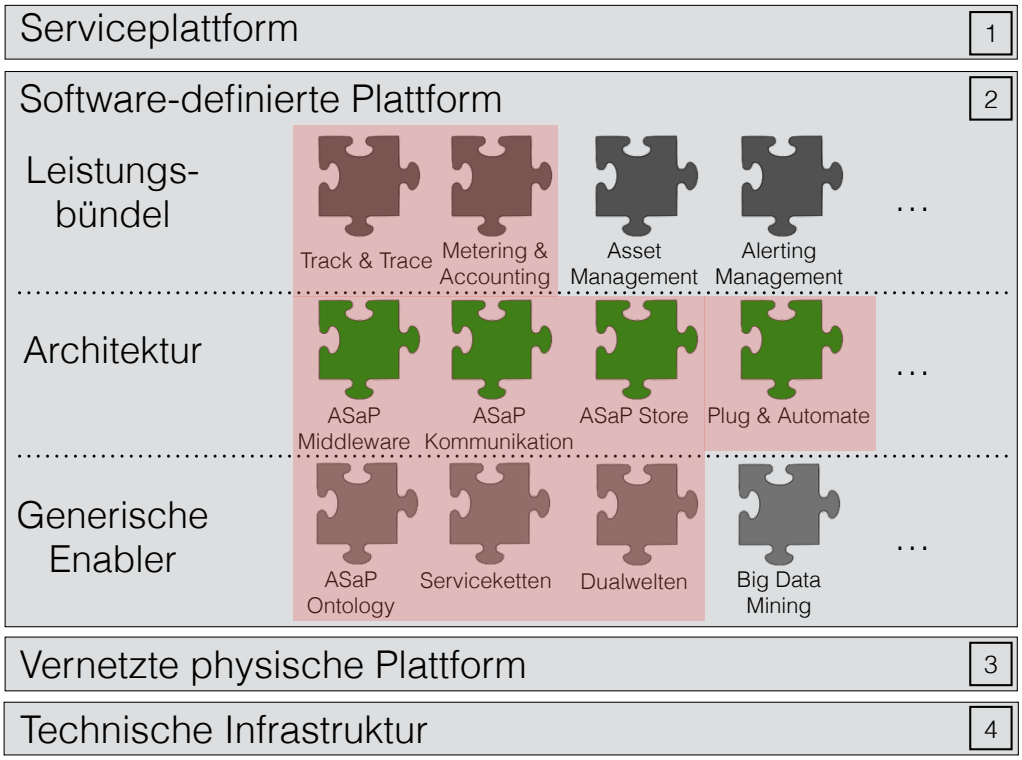


Abbildung 8.6: Einordnung von *ASaP* in die generischen Enabler, Architekturmuster und Leistungsbündel einer *Software-definierten Plattformen* innerhalb der *Smart Service Welt* (adaptiert nach Kagermann et al. (2015)).

Teil III

Realisierung und Anwendung

Suche nicht nach Fehlern, suche nach Lösungen.

Henry Ford

9

ASaP - Agentenbasierte Smart Service Plattform

9.1 Einleitung und Überblick

Das vorliegende Kapitel beschreibt die Implementierung einer *Agentenbasierten Smart Service Plattform* (*ASaP*) auf Grundlage der in Kapitel 7 und 8 eingeführten theoretischen Konzepte. Die grundlegende Idee folgt dem in Bianco et al. (2007) beschriebenen *Serviceorientierten Architekturansatz* (SOA) und adressiert hierbei die Abbildung von *Smart Services* auf einzelne *Agenten* innerhalb eines *Multiagentensystems*, wobei jeweils ein *Smart Service* als einzelner *Softwareagent* repräsentiert wird. Die *Smart Service Agenten* tauschen Informationen auf der Basis von ACL-Nachrichten aus, was einerseits die Verwendung kommunikativer Akte (siehe Kapitel 4.4) und andererseits auch die in Kapitel 8 eingeführte semantische Modellierung ermöglicht. Durch den Einsatz von OSGi als Ablaufumgebung können *Smart Services* als einzelne *OSGi Bundles* implementiert werden, was die Austauschbarkeit und die Integrationsfähigkeit einzelner *Smart Services* zur Laufzeit der Plattform gewährleistet.

Die Kernfunktionalität von *ASaP* wird in Abschnitt 9.2 beschrieben. Abschnitt 9.3 skizziert die Struktur der verwendeten Technologiekomponenten und deren Rolle innerhalb des Gesamtsystems. In Abschnitt 9.4 wird die Gesamtarchitektur der Plattform spezifiziert. In Abschnitt 9.5 werden die in der vorliegenden Arbeit realisierten und auf *ASaP* aufbauenden *Smart Service Agenten* kurz vorgestellt. Abschnitt 9.6 beschreibt abschließend eine technische Realisierung eines digitalen Marktplatzes zum Vertrieb und zum Auffinden von *Smart Services*.

9.2 Funktionalität

Die nachfolgende Aufzählung beschreibt die Kernfunktionalität der *Agentenbasierten Smart Service Plattform*:

- **Ablaufumgebung für *Smart Services*:** Die Hauptaufgabe von *ASaP* ist die Bereitstellung einer Ablaufumgebung für *Smart Services*. Entsprechend den Anforderungen an eine SOA (Bianco et al., 2007), sind die auf

Basis von *ASaP* entwickelten *Smart Service Bundles* modular und kohäsiv aufgebaut und können jeweils unabhängig voneinander installiert und benutzt werden.

- **Dynamische Rekonfigurierbarkeit:** *ASaP* bietet die Möglichkeit, die Funktionalität des Gesamtsystems durch die Integration neuer *Smart Services* zu erweitern. Die Verwendung von OSGi als Komponentenplattform erlaubt zudem das Hinzufügen und Entfernen von *Smart Service Bundles* während der Laufzeit von *ASaP*. Ein *Smart Service Bundle* ist wiederum dafür verantwortlich, *Smart Service Agenten* entsprechend zu erstellen und in *ASaP* zu integrieren.

- **Auffindbarkeit von *Smart Services*:** Die Auffindbarkeit von *Smart Services* innerhalb von *ASaP* ist durch unterschiedliche plattforminterne Mechanismen gewährleistet. Zunächst stellt die OSGi-Plattform eine lokale und innerhalb der aktuellen JVM gültige *Service Registry* zur Verfügung, mit deren Hilfe *Service Bundles* sowie deren Schnittstellen und Lebenszyklen verwaltet und überwacht werden können.

  Des Weiteren verfügt die verwendete JADE Plattform mit dem sogenannten *Directory Facilitator* (siehe Kapitel 4.3) über einen weiteren zentralen Verzeichnisdienst, mit dem alle angemeldeten *Smart Service Agenten* und deren semantische Beschreibungen auffindbar werden. Zuletzt ist durch den vollständig in das *ASaP*-Ökosystem integrierten *ASaP-Store* eine globale Kooperationsplattform gegeben, mit der sowohl Dienstanbieter und Entwickler als auch Endkunden dazu in die Lage versetzt werden, *Smart Services* zu finden, zu erstellen und miteinander zu kombinieren.

- **Verteilte Verarbeitung:** Durch den Einsatz von JADE als Multiagentenplattform ist die Ausführung eines *Smart Service Agenten* und somit dessen interne Verarbeitung nicht auf die lokale Infrastrukturumgebung der laufenden *ASaP*-Instanz beschränkt. Durch die Verwendung des *Agent Management Systems* (siehe Kapitel 4.3) wird eine Verbindung zu einer auf einem *Remote Server* installierten Agentenplattform aufgebaut. *ASaP* erfüllt somit die Anforderungen einer cloud-basierten Verarbeitung, indem beispielsweise die Speicherung von großen Datenmengen sowie *Smart Service Agenten* mit ressourcenintensiven Berechnungen ausgelagert werden können.

- **Dienstübergreifende Interoperabilität:** Durch die *ASaP* zugrunde liegende, semantische Modellierung einzelner *Smart Services* sowie deren Abhängigkeiten zueinander kann eine dienstübergreifende Interoperabilität der *Smart Services* sichergestellt werden. Wie in Kapitel 7 beschrieben, besitzt jeder *Smart Service* eine eindeutige Definition seiner *Input* und *Output* Parameter, wodurch beispielsweise eine Kombination und dadurch auch die Interoperabilität einzelner Mehrwertdienste realisiert werden können.

9.3 Plattformkomponenten

Abbildung 9.1 gibt einen Überblick über die technologischen Komponenten von *ASaP*. Die Plattform setzt sich, aufbauend auf Java als verwendete Programmiersprache, aus mehreren Basiskomponenten zusammen, welche in den folgenden Abschnitten detailliert beschrieben werden.

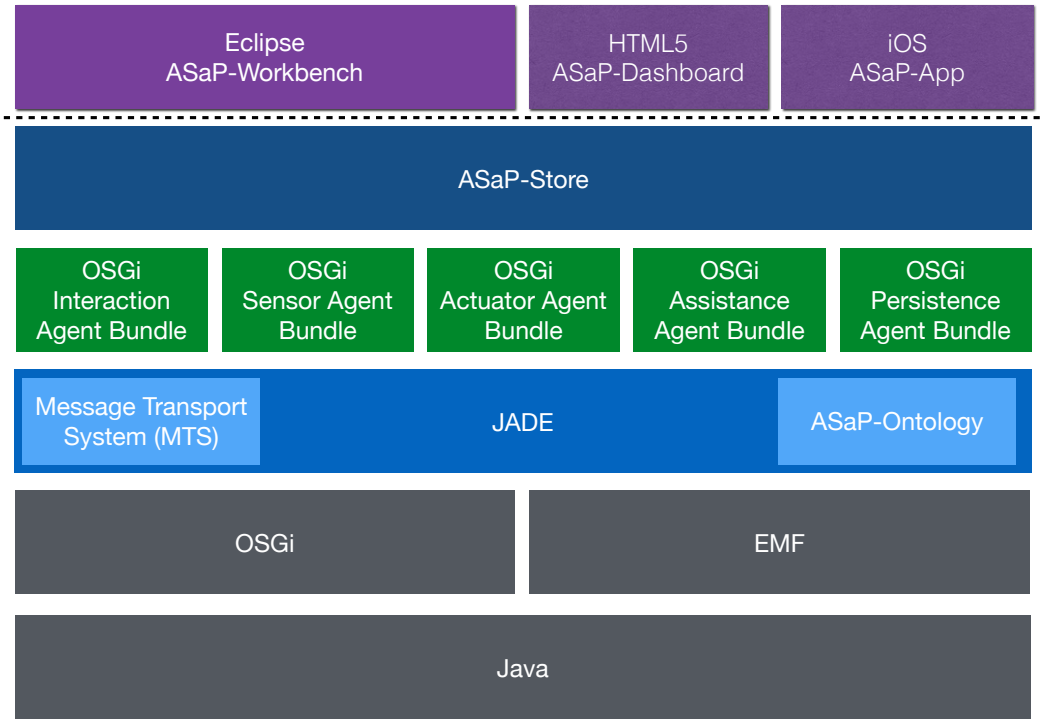


Abbildung 9.1: Übersicht über die in *ASaP* verwendeten Technologiekomponenten.

9.3.1 OSGi

In Kapitel 6.5.1 wurde bereits eine allgemeine Einführung in die Architektur und Funktionsweise des OSGi-Frameworks gegeben. Analog zu der dort beschriebenen UCH-Implementierung baut auch *ASaP* auf einem zugrunde liegenden OSGi-Komponentenmodell auf. OSGi hat seinen Ursprung in der Entwicklung eingebetteter Systeme. Durch die klare Modularisierung in einzelne *Bundles*, welche über definierte Serviceschnittstellen plattformintern adressierbar sind, kann ein OSGi-Framework jedoch auch als grundlegendes Komponentenmodell einer SOA dienen (Wang et al., 2010; Garcia Sanchez et al., 2013). Innerhalb einer SOA-Anwendung übernimmt OSGi hierbei die folgenden Aufgaben[1]:

[1]http://www.jboss.org/dms/javaone2008/JavaOne2008_DevelopingSOAappsOSGi.pdf [Letzter Zugriff: 31.01.2015]

- **Erweiterbarkeit:** Die Funktionalität der Anwendung kann durch das Hinzufügen neuer *Bundles* erweitert werden.
- **Kapselung:** Die Modularisierung in einzelne *Bundles* ermöglicht eine direkte Kontrolle der importierten und exportierten Softwarepakete.
- **Dynamik:** *Bundles* besitzen einen von der Java Virtual Machine (JVM) unabhängigen Lebenszyklus und können unabhängig voneinander ausgeführt, gestartet und gestoppt werden.
- **Abhängigkeiten:** Die Abhängigkeit zu anderen *Bundles* und Services können deklarativ definiert sowie zur Laufzeit überprüft und abgefragt werden.

In *ASaP* werden *Smart Services* durch einzelne *OSGi-Bundles* definiert. Das daraus resultierende *Service Bundle* kann hierbei einen oder mehrere Services in Form von *Softwareagenten* bereitstellen, welche nach Aktivierung des *Bundles* entsprechend in das vorliegende *Multiagentensystem* geladen oder bei Deaktivierung des *Bundles* wieder entfernt werden.

9.3.2 EMF

ASaP nutzt die in Kapitel 8 definierte, auf EMF basierende, semantische Modellierung einer *Intelligenten Umgebung* als Grundlage für die plattforminternen Kommunikations- und Kooperationsmechanismen der durch die *Service Bundles* zur Verfügung gestellten *Smart Service Agenten*. Die entwickelten EMF Modelle befinden sich hierzu in einem dedizierten *Ontology Bundle*, welches von anderen *Bundles* importiert werden muss, um vollen Zugriff auf die jeweiligen Modelle zu haben. Durch entsprechende *Utility Klassen* können Instanzen einzelner Klassen innerhalb des Modells erzeugt werden. Damit die erzeugten Objekte dann als *Content* Parameter innerhalb einer ACL-Nachricht übergeben werden können, müssen diese in eine serialisierbare Form übertragen werden. Dazu wurde das Softwarepaket *EMF Binding for JSON*[2] als *OSGI Dependency* integriert. Das Paket erlaubt die Serialisierung und Deserialisierung von EMF Objekten in eine korrespondierende *JavaScript Object Notation* (JSON). Listing 9.1 zeigt am Beispiel eines ausgelösten Feueralarms im Wohnzimmer die zugehörige JSON-Repräsentation innerhalb einer ACL-Nachricht.

```
{
  content:
    "eClass" : "http://www.jofrey.de/ontology#//Fire",
      "context" : [ {
      "eClass" : "http://www.jofrey.de/ontology#//LivingRoom"
      }, {
        "eClass" : "http://www.jofrey.de/ontology#//SmokeDetector"
      } ]
    ontology: default
```

[2] http://www.emfjson.org [Letzter Zugriff: 31.01.2015]

```
    performative: INFORM
    sender: Nest Sensor Agent
}
```

Listing 9.1: JSON-Serialisierung des Inhalts einer ACL-Nachricht am Beispiel eines ausgelösten Feueralarms im Wohnzimmer.

9.3.3 JADE

Im Kontext von *ASaP* stellt JADE im Zusammenspiel mit OSGi die technologische Basis zur Realisierung einer *Serviceorientierten Architektur* dar (Spanoudakis und Moraitis, 2008; Wang et al., 2012; Maximilien und Singh, 2004, 2005). JADE wurde bereits als *Agentenplattform* zur Implementierung von *Multiagentensystemen* in Kapitel 4 eingeführt und ausführlich beschrieben. Abbildung 9.2 zeigt eine Abbildung der *ASaP*-Infrastruktur auf das klassische Dreieckmodell einer *Serviceorientierten Architektur*.

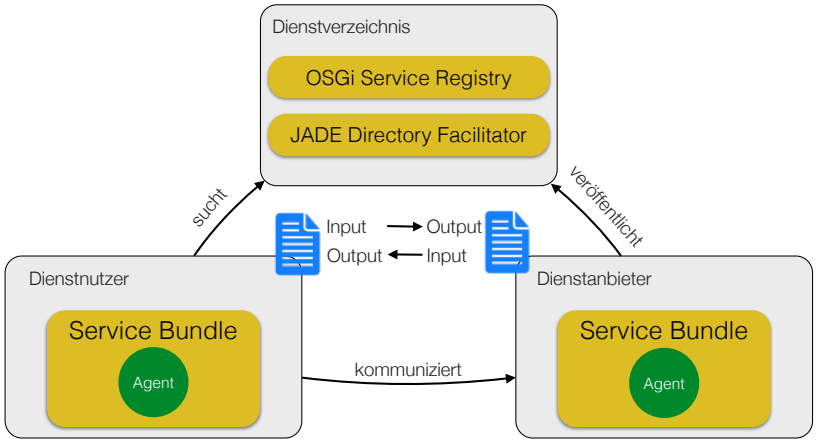


Abbildung 9.2: Übersicht über die Einbettung von OSGi und JADE in die *Serviceorientierte Architektur* von *ASaP*.

Dienstnutzer und Dienstanbieter werden jeweils durch einen *Smart Service Agenten* innerhalb eines *Service Bundles* realisiert. Beide verfügen über eine interne, semantische Beschreibung ihrer *Input* und *Output* Parameter. Hierbei besteht, wie in SOA gefordert, eine lose Kopplungsbeziehung zwischen Dienstnutzer und Dienstanbieter, das heißt, der tatsächliche Informationsaustausch beider *Smart Services* hängt von einem Abgleich zwischen den jeweiligen geforderten und vorhandenen *Input* und *Output* Eigenschaften ab. Dies führt dazu, dass Dienste ersetzbar und austauschbar sind, da der Dienstnutzer nicht von einer konkreten Ausprägung eines Dienstanbieters abhängig ist, sondern vielmehr von der semantischen Beschreibung.

9.4 Plattformarchitektur

Die vorgestellten Basiskomponenten OSGi, EMF und JADE bilden die Grundlage für die Entwicklung der *Smart Services*, des *ASaP-Stores* sowie der integrierten Entwicklungsumgebung. Abbildung 9.3 zeigt das Zusammenspiel der einzelnen *ASaP*-Komponenten anhand der vorhandenen Plattformarchitektur.

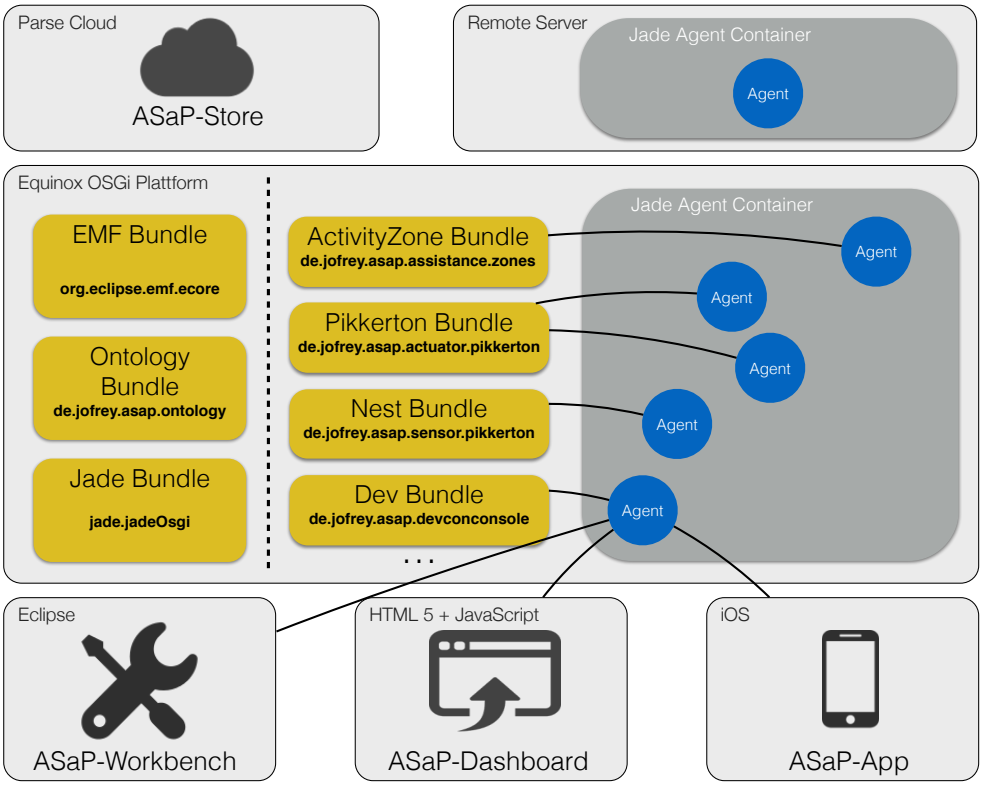


Abbildung 9.3: Übersicht über das Zusammenspiel der *ASaP*-Komponenten innerhalb der Plattformarchitektur.

Im Kern setzt die Plattform auf dem *Equinox OSGi Container* auf und besteht, wie bereits erwähnt, aus mehreren unabhängig laufenden *Smart Service Bundles*. Diese sind dazu in der Lage, *Smart Service Agenten* bei Bedarf zu instanziieren und plattformintern als Dienste zur Verfügung zu stellen. Ein *Bundle* ist allgemein eine Sammlung von Java Klassen sowie zusätzlichen Ressourcen. Abbildung 10.4 zeigt die Dateistruktur eines minimalen *Smart Service Bundles*.

Jedes *Smart Service Bundle* setzt sich demnach mindestens aus den folgenden Dateien zusammen:

- `MANIFEST.MF`: Eine MANIFEST-Datei enthält eine deklarative Beschreibung des zugrunde liegenden *OSGi Bundles*. Hier werden unter anderem der Name des *Bundles*, eine Versionsnummer, die verwendete Java Version sowie die importierten und exportierten *Bundles* spezifiziert. Die notwendigen Abhängigkeiten zu anderen *OSGi-Bundles* sind möglichst

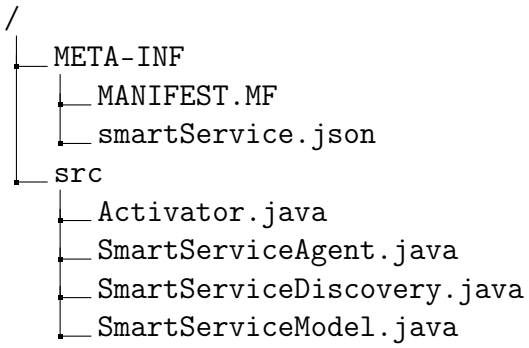


Abbildung 9.4: Dateistruktur eines minimalen *Smart Service Bundles*.

gering gehalten. So müssen in einer Minimalkonfiguration lediglich zwei externe *OSGi-Bundles* importiert werden: `jade.jadeOsgi` für die Einbindung von JADE und `de.jofrey.asap.ontology` für die Einbindung der semantischen Modelle. Letzteres hat wiederum eine Abhängigkeit auf *org.eclipse.emf.ecore* zur Einbindung der EMF Funktionalitäten. Listing 9.2 zeigt eine exemplarische MANIFEST-Datei.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Template
Bundle-SymbolicName: de.jofrey.asap.service.template
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: de.jofrey.asap.service.template.Activator
Bundle-RequiredExecutionEnvironment: JavaSE-1.8
Import-Package: org.osgi.framework;version="1.3.0"
Bundle-ActivationPolicy: lazy
Require-Bundle: jade.jadeOsgi;bundle-version="1.0.0",
 de.jofrey.asap.ontology;bundle-version="0.1.0"
```

Listing 9.2: Beispiel einer MANIFEST-Datei.

- `smartService.json`: Die JSON-Datei enthält eine deklarative Zusammenfassung des zugrunde liegenden *Smart Services* und beschreibt die entsprechenden semantischen Informationen. Die Datei wird primär innerhalb der *ASaP-Workbench* verwendet und dient insbesondere zum Austausch von Meta-Informationen während des Wirkbetriebs auf dem *ASaP-Store*. Listing 9.3 zeigt ein Beispiel anhand eines *Aktuators* zum Schalten einer Steckdose und zum Messen der daran verbrauchten Energie.

```
{
  "name": "Pikkerton Smart Service",
  "type": [ "Sensor", "Actuator"],
  "description": "Smart Service Bundle to integrate Pikkerton
    Smart Meter Plugs.",
  "context": ["PowerPlug"],
  "input": [ "On", "Off"],
```

```
  "output": [ "Consumption", "Power",],
  "app": [""],
  "product": ["Pikkerton Smart Energy Plug"],
  "ontology": "default",
  "developer": "jofrey"
}
```

Listing 9.3: Beispiel einer deklarativen Servicebeschreibung am Beispiel einer vernetzten Steckdose.

- `Activator.java`: Die `BundleActivator` Klasse wird aufgerufen, sobald ein *Smart Service Bundle* gestartet wird, und stellt somit den programmatischen Einstiegspunkt eines *Bundles* dar. Die Hauptaufgabe der Klasse ist, den `SmartServiceDiscovery` Mechanismus zu starten und gegebenenfalls wieder zu stoppen. Listing 9.4 zeigt ein entsprechendes Quellcodebeispiel.

```
@Override
public void start(BundleContext bundleContext) throws
    Exception {
  Activator.context = bundleContext;
  smartServiceDiscovery = new SmartServiceDiscovery();
  smartServiceDiscovery.startDiscorvery();
}

@Override
public void stop(BundleContext bundleContext) throws
    Exception {
  Activator.context = null;
  smartServiceDiscovery.stopDiscovery();
}
```

Listing 9.4: Auszug aus der `Activator` Klasse.

- `SmartServiceDiscovery.java`: Die `SmartServiceDiscovery` Klasse ist dafür verantwortlich, je nach Situation und Kontext, entsprechende *Smart Service Agenten* zu erzeugen und in JADE zu registrieren. Listing 9.5 zeigt ein entsprechendes Quellcodebeispiel.

```
public void startDiscorvery() {
    SmartServiceAgent agent = new SmartServiceAgent(ID);
    AgentController agentController = jadeRuntimeService.
   acceptNewAgent(ID, agent);
}

public void stopDiscovery() {...}
```

Listing 9.5: Auszug aus der `SmartServiceDiscovery` Klasse.

- `SmartServiceAgent.java`: Die `SmartServiceAgent` Klasse bildet die grundlegende Funktionalität eines *Smart Service Agenten* ab. Hierzu werden zwei Verhaltensweisen des *Agenten* durch entsprechende *Behaviours*

hinzugefügt. Die `ReceivingBehaviour` ist dafür verantwortlich, plattforminterne ACL-Nachrichten zu empfangen, zu verarbeiten und gegebenenfalls an das `SmartServiceModel` weiterzuleiten. Die optionale *UpdateBehaviour* ermöglicht die Realisierung eines Update Mechanismus. Hierbei wird der Agent aufgefordert, sein internes `SmartServiceModel` aufzufrischen, das heißt im Falle eines *Sensor Agenten*, aktuelle Werte des *Sensors* programmatisch abzufragen. Darüber hinaus stellt die `SmartServiceAgent` Klasse eine `publish` Methode bereit, welche das Versenden von ACL-Nachrichten an alle registrierten Agenten ermöglicht. Listing 9.6 zeigt einen Auszug eines entsprechenden Quellcodebeispiels.

```
class ReceivingBehaviour extends CyclicBehaviour {
  @Override
  public void action() {
    smartServiceModel.proposeOn();
  }
}

class UpdateBehaviour extends TickerBehaviour {...}

public void publish(ACLMessage message) {...}
```

Listing 9.6: Auszug aus der `SmartServiceAgent` Klasse.

- `SmartServiceModel.java`: Die `SmartServiceModel` Klasse stellt die eigentliche Schnittstelle zu der internen Funktionsweise dar. Betrachtet man beispielsweise den zuvor beschriebenen Aktuator zum Schalten einer vernetzten Steckdose, so bedient die zugehörige `SmartServiceModel` Klasse die zur Kontrolle des Aktuators notwendigen Programmierschnittstellen. Listing 9.7 zeigt die entsprechenden Methodenaufrufe, basierend auf der semantischen Beschreibung und der relevanten, kommunikativen Akte.

```
public void proposeOn() {...}

public void proposeOff() {...}

public void requestPower() {...}

public void requestConsumption() {...}
```

Listing 9.7: Auszug aus der `SmartServiceModel` Klasse.

9.5 Smart Services

Im Rahmen der vorliegenden Arbeit wurden auf der Basis von *ASaP* insgesamt neun unterschiedliche *Smart Services* realisiert, davon drei *Sensor Services*, ein *Actuator Service*, zwei *Assistance Services*, zwei *Interaction Services*

und schließlich ein *Persistence Service*. Die folgenden Abschnitte stellen die entsprechenden *Smart Service Agenten* anhand ihrer Funktionsweise und ihrer semantischen Beschreibung kurz vor.

9.5.1 Nest Sensor Agent

Nest Labs[3] ist ein US-amerikanisches Unternehmen mit Sitz in Palo Alto. Sein Schwerpunkt liegt auf der Entwicklung von *Smart Home* Technologien. Nest Labs wurde Anfang 2014 von Google übernommen und bietet mit einem selbstlernenden Thermostat sowie einem vernetzten Rauchmelder derzeit zwei Produkte auf dem Consumer Markt an. Die nachfolgende Spezifikation beschreibt die Funktionsweise des *Smart Services* auf der Grundlage des Rauchmelders[4].

- **Zweck:** Überwachung der Rauchentwicklung und des CO_2 Gehaltes innerhalb eines *Smart Homes* und gegebenenfalls Auslösen eines entsprechenden Alarms.
- **Typ:** SensorService
- **Output:** Alarm, Battery
- **Funktionsweise:** Der *Service* bietet reine *Sensor* Funktionalitäten an, das heißt, er ist dazu fähig, sowohl die Rauchentwicklung als auch den CO_2 Gehalt in der Luft zu überwachen und im Notfall eine entsprechende Alarmnachricht innerhalb von *ASaP* auszulösen.

9.5.2 Kinect Sensor Agent

Die Microsoft Kinect[5] ist ursprünglich ein Sensor zur Steuerung von Videospielen und Multimediaanwendungen. Durch den Einsatz einer Tiefenkamera ist sie dazu im Stande, Bewegungsmuster oder Gesten von Personen innerhalb ihrer Reichweite zu erkennen. Zusätzlich ist sie mit einem Mikrofonarray ausgestattet, wodurch auch Sprachkommandos verarbeitet werden können.

- **Zweck:** Erfassung von Bewegungsdaten eines Benutzers innerhalb eines relativen kartesischen Koordinatensystems sowie von Audiosignalen über ein Mikrofonarray.
- **Typ:** SensorService
- **Output:** CartesianCoordinates, Audio
- **Funktionsweise:** Der *Service* bietet reine *Sensor* Funktionalitäten an, indem er die kartesischen Koordinaten mehrerer Benutzer sowie erkannte Audiosignale in *ASaP* propagieren kann.

[3]http:www.nest.com [Letzter Zugriff: 19.01.2015]
[4]https://nest.com/smoke-co-alarm/life-with-nest-protect [Letzter Zugriff: 19.01.2015]
[5]http://www.microsoft.com/en-us/kinectforwindows/ [Letzter Zugriff: 19.01.2015]

9.5.3 Hagleitner Sensor Agent

Die Firma Hagleitner[6] ist ein führender Komplettanbieter für professionelle Hygiene und bietet eine umfassende Palette an Reinigungsprodukten und -geräten an. Mit der senseMANAGEMENT Serie stellt Hagleitner erstmals vernetzte Seifen- und Papierspender zur Realisierung eines intelligenten Waschraums zur Verfügung.

- **Zweck:** Überwachung der Füll- und Batteriestände der installierten Waschraumspender und gegebenenfalls auslösen eines entsprechenden Alarms.
- **Typ:** SensorService
- **Output:** Battery, Level
- **Funktionsweise:** Der *Service* bietet reine *Sensor* Funktionalitäten an, indem er den Füllstand und den Batteriestatus eines Waschraumspenders überwacht und rechtzeitig eine entsprechende Alarmnachricht innerhalb von *ASaP* auslöst.

9.5.4 Pikkerton Actuator Agent

Pikkerton[7] ist ein mittelständisches Unternehmen mit Sitz in Deutschland, dass sich auf die Entwicklung und den Vertrieb von Lowpower Funknetzwerkbaugruppen spezialisiert hat. Der *Pikkerton Smart Service Agent* kann mehrere auf ZigBee[8] basierende *Sensoren* und *Aktuatoren* in *ASaP* zu integrieren. Die nachfolgende Spezifikation beschreibt die Funktionsweise des *Smart Services* auf der Grundlage des *ZigBee Smart Energy Meters / Smart Plugs*[9].

- **Zweck:** Erfassung von Stromverbrauchsdaten sowie die Möglichkeit zum Ein- und Ausschalten von angeschlossenen Geräten.
- **Typ:** SensorService, ActuatorService
- **Input:** On, Off
- **Output:** Consumption, Power
- **Funktionsweise:** Der *Service* umfasst sowohl *Actuator* als auch *Sensor* Funktionalitäten. Einerseits bietet er die Möglichkeit zur Aktivierung und zur Unterbrechung der Stromzufuhr des angeschlossenen Geräts. Andererseits kann wiederum der Zustand der Stromzufuhr sowie der aktuelle Stromverbrauch des Gerätes abgefragt werden.

[6] http://www.hagleitner.com [Letzter Zugriff: 15.04.2015]
[7] http://www.pikkerton.de [Letzter Zugriff: 19.01.2015]
[8] http://www.zigbee.org [Letzter Zugriff: 19.01.2015]
[9] http://www.pikkerton.de/_objects/1/6.htm [Letzter Zugriff: 19.01.2015]

9.5.5 Activity Zone Assistance Agent

Im Rahmen der vorliegenden Arbeit wurde ein *Smart Service* entwickelt, um, basierend auf unterschiedlichen Sensorinformationen, sogenannte Aktivitätszonen innerhalb eines *Smart Homes* zu berechnen und dreidimensional zu visualisieren (Frey et al., 2014). Der hierfür realisierte *Activity Zone Assistance Agent* benötigt eine relative Positionierung des Benutzers innerhalb eines dreidimensionalen kartesischen Koordinatensystems und Zustandsinformationen der Umgebung, um daraus die entsprechenden Aktivitätszonen berechnen und visualisieren zu können.

- **Zweck:** Berechnung und Visualisierung von dreidimensionalen Aktivitätszonen innerhalb der Umgebung.
- **Typ:** AssistanceService
- **Input:** CartesianCoordinates, State
- **Output:** ActivityZone
- **Funktionsweise:** Der *Service* bietet *Assistance* Funktionalitäten an, indem er aus einer Kombination der erkannten Benutzerbewegungen und der zugehörigen Zustandsveränderungen innerhalb der Umgebung entsprechende dreidimensionale Aktivitätszonen berechnet und visualisiert (siehe Kapitel 11.5.1).

9.5.6 Sound-based Device Recognition Assistance Agent

Die vorliegende Arbeit zeigt eine weitere Entwicklung eines *Smart Services*, um, basierend auf erfassten Audiosignalen innerhalb der Umgebung, die verursachende Benutzeraktivität beziehungsweise das entsprechende Haushaltsgerät erkennen zu können (Dimitrov et al., 2014). Nach einer erfolgreichen Trainingsphase ist das System in der Lage, aus einfachen Audiosignalen entsprechende Gerätetypen oder Benutzeraktivitäten abzuleiten.

- **Zweck:** Erkennung einer Benutzeraktivität oder einer Gerätenutzung innerhalb der Umgebung.
- **Typ:** AssistanceService
- **Input:** Audio
- **Output:** Physical, Behavioural
- **Funktionsweise:** Der *Service* bietet *Assistance* Funktionalitäten an, indem er durch geeignete Verfahren die spezifische Soundcharakteristik von erfassten Audiosignalen extrahiert und durch den Einsatz von maschinellen Lernverfahren dazu fähig ist, die zugrunde liegende Benutzeraktivitäten und Gerätenutzung zu klassifizieren (siehe Kapitel 11.5.2).

9.5.7 URC Interaction Agent

Das Ziel des *URC Interaction Agent* ist die Einbindung von existierenden *URC-Controllern* zur Kontrolle und zur Überwachung der in *ASaP* installierten *Smart Services*. Aufbauend auf den Ideen von Zimmermann et al. (2013, 2014) wird hierbei ein generativer Ansatz verfolgt, indem eine zur semantischen Beschreibung des jeweiligen *Service* passende *User Interface Socket* Beschreibung erzeugt wird und durch ein URC-HTTP-Protokoll (siehe Kapitel 6.4.4) bereitgestellt wird.

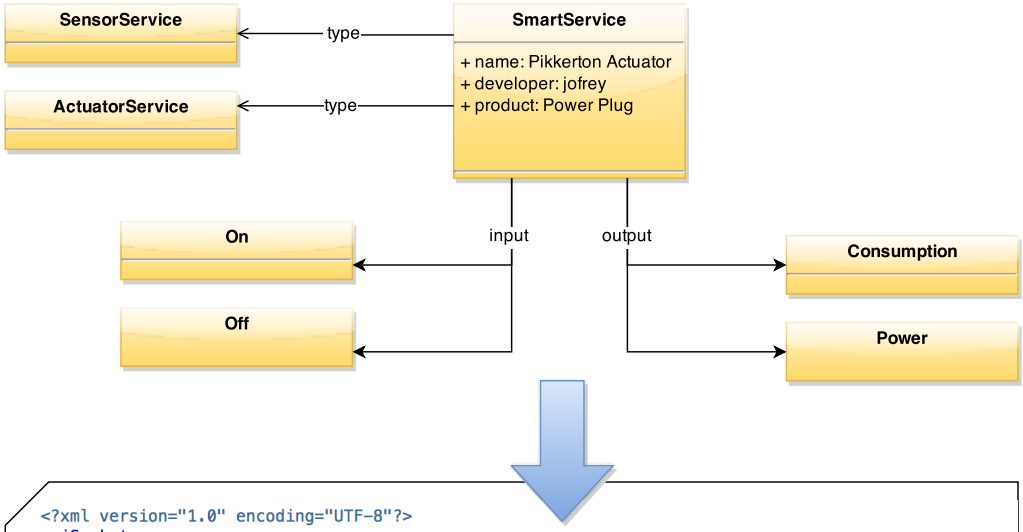
- **Zweck:** Einbindung von existierenden URC-Clients in die *ASaP*-Infrastruktur.
- **Typ:** InteractionAgent
- **Input:** Event
- **Output:** Event
- **Funktionsweise:** Der *Service* bietet *Interaction* Funktionalitäten an. Durch eine zugrunde liegende Modelltransformation wird aus der semantischen Beschreibung des *Services* eine entsprechende UIS-Beschreibung generiert und durch das URC-HTTP-Protokoll (siehe Kapitel 6.4.4) verfügbar gemacht.

Abbildung 9.5 zeigt einen Auszug aus der semantischen Beschreibung des *Pikkerton Actuator Agents* und der dazugehörigen UIS-Beschreibung innerhalb des URC-Ökosystems.

9.5.8 Devconsole Interaction Agent

Der *Devconsole Interaction Agent* bietet eine einfache, interne Schnittstelle zu den Funktionalitäten der laufenden *ASaP*-Instanz. Die Schnittstelle wird überwiegend von den in Kapitel 10 vorgestellten Anwendungen verwendet.

- **Zweck:** Bereitstellung einer standardmäßig verfügbaren, einfachen Interaktionsmöglichkeit mit der *ASaP*-Infrastruktur.
- **Typ:** InteractionAgent
- **Input:** Event
- **Output:** Event
- **Funktionsweise:** Der *Service* bietet zwei grundlegende *Interaction* Funktionalitäten an. Auf der einen Seite wird ein leichtgewichtiger, auf Jetty basierender Webserver bereitgestellt, welcher beispielsweise d as *ASaP-Dashboard* beherbergt (siehe Kapitel 10.4). Auf der anderen Seite wird ein WebSocket erzeugt, der eine kommandozeilenbasierte Interaktion mit der Plattform sowie den installierten *Smart Services* bietet. Diese Schnittstelle wird beispielsweise von der *ASaP-Workbench* (siehe Kapitel 10.3) sowie der *ASaP-App* (siehe Kapitel 10.5) genutzt.




```
<?xml version="1.0" encoding="UTF-8"?>
<uiSocket
      about="http://de.jofrey.asap.actuator.pikkerton"
      id="socket"
      sufficient="false"
      complete="false"
      xmlns="http://myurc.org/ns/uisocketdesc"
      xmlns:uis="http://myurc.org/ns/uisocketdesc"
      xmlns:dc="http://purl.org/dc/elements/1.1/"
      xmlns:dcterms="http://purl.org/dc/terms/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://myurc.org/ns/uisocketdesc http://myurc.org/ns/uisocketdesc
        http://purl.org/dc/elements/1.1/ http://dublincore.org/schemas/xmls/qdc/2006/01/06/dc.xsd
        http://purl.org/dc/terms/ http://dublincore.org/schemas/xmls/qdc/2006/01/06/dcterms.xsd">
      <dc:creator>jofrey</dc:creator>
      <dc:publisher>Pikkerton</dc:publisher>
      <dc:title xml:lang="en">Pikkerton Actuator Power Plug</dc:title>

    <!-- ================================================================== -->
    <!-- Variables                                                          -->
    <!-- ================================================================== -->
    <variable id="Consumption" type="xsd:float"></variable>
    <variable id="Power" type="xsd:boolean"></variable>

    <!-- ================================================================== -->
    <!-- Commands                                                           -->
    <!-- ================================================================== -->
    <command id="On" type="basicCommand"></command>
    <command id="Off" type="basicCommand"></command>

</uiSocket>
```

Abbildung 9.5: Abbildung der semantischen Beschreibung des *Pikkerton Actuator Agents* auf die URC-konforme *User Interface Socket* Beschreibung.

9.5.9 SQLite Persistence Agent

Der *SQLite Persistence Agent* bietet eine einfache Möglichkeit, die Ereignisse und Nachrichten innerhalb einer *ASaP*-Instanz in einer lokalen SQLite-Datenbank[10] abzuspeichern.

- **Zweck:** Persistierung der Ereignisse und Nachrichten innerhalb einer *ASaP*-Instanz.

[10] http://www.sqlite.org [Letzter Zugriff: 15.04.2015]

- **Typ:** PersistenceAgent
- **Input:** Event
- **Output:** Event
- **Funktionsweise:** Der *Service* bietet *Persistence* Funktionalität an, indem er alle Ereignisse, Zustandsinformationen oder Kontextinformation in einer lokalen Datenbank abspeichert und mit einem eindeutigen Zeitstempel verknüpft. Sowohl Endnutzer als auch andere *Smart Services* innerhalb der *ASaP*-Instanz können daraufhin gezielt auf die gespeicherten Verlaufsinformationen zugreifen und diese für ihre eigenen Berechnungen verwenden.

9.6 ASaP-Store

Der Einsatz von digitalen Marktplätzen, wie beispielsweise *Apple App Store* oder *Google Play*, hat die allgemeine Art und Weise wie Kunden mit Softwareanwendungen umgehen, wie sie diese erwerben und im Alltag einsetzen können, entscheidend verändert(Jansen und Bloemendal, 2013). So sind digitale Marktplätze beispielsweise dazu in der Lage die Vertriebskosten von Softwareanwendungen erheblich zu reduzieren und somit eine effizientere Marktstrategie der Anbieter zu ermöglichen. Auch das Bereitstellen von neuen Inhalten oder Produktupdates wird durch den Einsatz von digitalen Vertriebswegen erheblich beschleunigt und vereinfacht. Nach Hyrynsalmi et al. (2012) sind *App Stores* hierbei ein entscheidender Erfolgsfaktor beim Aufbau eines jeden softwarebasierten Ökosystems.

Definition 35 (*App Store*)

> *Ein App Store ist ein digitaler Onlinemarktplatz innerhalb eines Softwareökosystems, welcher es Entwicklern und Anbietern ermöglicht ihre Softwareprodukte innerhalb einer definierten Plattforminfrastruktur an die entsprechenden Kunden zu verkaufen (Jansen und Bloemendal, 2013).*

Der vollständig in das *ASaP*-Ökosystem integrierte *ASaP-Store* bietet hierzu eine Kooperationsplattform, mit der sowohl Dienstanbieter und Entwickler als auch Endkunden dazu in die Lage versetzt werden, *Smart Services* zu finden, zu erstellen und miteinander auszutauschen. Die Kernfunktionalität des Marktplatzes umfasst hierbei die Möglichkeit, *Smart Service Bundles* zusammen mit ihrer semantischen Spezifikation zu verwalten und potentiellen Endkunden zur Installation anzubieten.

Die technologische Basis für die Umsetzung des Marktplatzes baut auf einem *Backend-as-a-Service* (Baas) Ansatz auf. Darunter versteht man eine extern verwaltete Backend-Infrastruktur, welche sich dynamisch auf die Anforderungen der Anwendungen anpassen kann. Hierzu stehen bei den meisten

BaaS Anbietern unterschiedliche Preismodelle zur Auswahl, welche beispielsweise eine spätere Skalierung auf steigende Nutzerzahlen möglich macht. Der entscheidende Vorteil von BaaS ist, dass die Wartung des Backends durch den Anbieter übernommen wird und die Kosten sich an dem tatsächlichen Nutzungsverhalten orientierten. Es existieren bereits eine Vielzahl unterschiedlicher BaaS-Anbieter[11], die sich aber in ihrem Funktionsumfang nur wenig unterscheiden. In der Regel stellen die Systeme Entwicklungsumgebungen und Schnittstellenbeschreibungen für unterschiedliche Zielplattformen zur Verfügung. In der vorliegenden Arbeit wurde Parse[12] als zugrunde liegendes BaaS verwendet, und durch den Einsatz der bereitgestellten REST-Schnittstelle in die im nächsten Kapitel vorgestellten Werkzeuge integriert. Abbildung 9.6 zeigt die von Parse zur Verfügung gestellte Weboberfläche zur Verwaltung der veröffentlichten *Smart Service Bundles*.

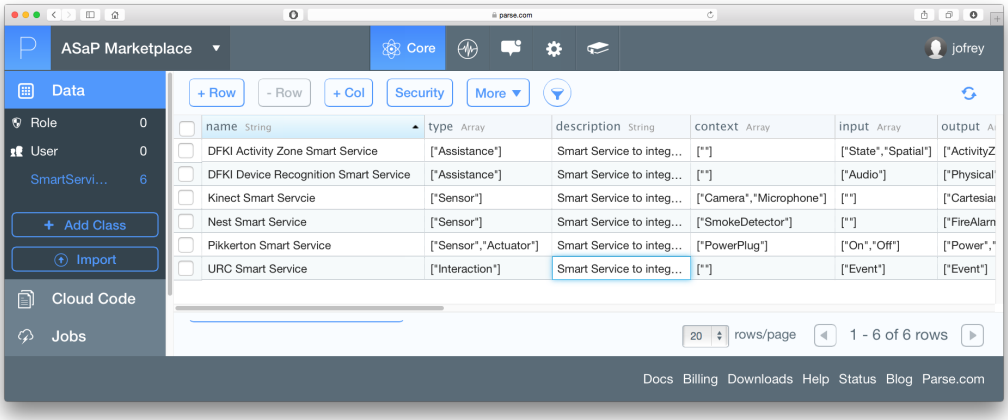


Abbildung 9.6: Übersicht über die von Parse zur Verfügung gestellte Weboberfläche.

In Kapitel 10.4 und 10.5 werden zudem eine webbasierte und eine mobile Schnittstelle für die Interaktion mit dem *ASaP-Store* vorgestellt.

9.7 Zusammenfassung und Fazit

In diesem Kapitel wurde eine Implementierung der *Agentenbasierten Smart Service Plattform* (*ASaP*) vorgestellt. Die Hauptaufgabe von *ASaP* ist die Bereitstellung einer Ablaufumgebung für *Smart Services*. Es wurde hierbei gezeigt, wie die Plattform durch die Integration neuer *Smart Services* dynamisch erweitert werden kann. Neben der Auffindbarkeit von *Services* durch entsprechende Verzeichnisdienste, wurde auch die Möglichkeit zur verteilten Verarbeitung beschrieben. Im Anschluss wurden die technologischen Basiskomponenten von *ASaP* detailliert beschrieben und in eine einheitliche Plattformarchitektur überführt. Neben der internen Struktur eines *Smart Service Bundles* wurden

[11] http://de.wikipedia.org/wiki/Backend_as_a_Service [Letzter Zugriff: 19.01.2015]
[12] http://parse.com [Letzter Zugriff: 19.01.2015]

die im Rahmen der Arbeit entwickelten *Smart Service Agenten* anhand ihrer Spezifikation und Funktionsweise kurz vorgestellt. Abschließend wurde die technologische Basis des entwickelten *ASaP-Store* erklärt. Das nächste Kapitel beschäftigt sich schließlich mit der Fragestellung, wie eine durchgängige Werkzeugkette realisiert werden kann, die sowohl Softwareentwickler, Dienstanbieter als auch Endnutzer während des kompletten Lebenszyklus eines *Smart Services* unterstützt und zudem die Möglichkeiten zu späteren Erweiterungen bereitstellt.

Im Leben kommt es darauf an, Hammer oder Amboss zu sein - aber niemals das Material dazwischen.

Norman Mailer

10 Smart Service Werkzeuge

10.1 Einleitung

In den vorigen Kapiteln wurde mit *ASaP* sowohl ein theoretisches Konzept als auch eine praktische Realisierung einer Plattform zur Integration von Smart Services in *Intelligenten Umgebungen* vorgestellt. Das beschriebene Konzept basiert auf einer semantischen Beschreibung der Umgebung, der Objekte und der entsprechenden *Smart Services* mit den jeweiligen Abhängigkeiten zueinander. In diesem Kapitel wird nun mit der *ASaP-Workbench* (Abschnitt 10.3) eine Sammlung von Werkzeugen beschrieben, welche es Unternehmern und Entwicklern erleichtern soll, digitale Mehrwertdienste in Form von *Smart Services* zu entwickeln und auf einem entsprechenden Marktplatz zu vertreiben. Weiterhin soll den Endkunden ein *ASaP-Dashboard* (Abschnitt 10.4) sowie eine *ASaP-App* (Abschnitt 10.5) zur Verfügung gestellt werden, um ihre eigene *ASaP*-Installation zu verwalten und über den Marktplatz durch die Installation neuer *Smart Services* dynamisch zu erweitern. Alle in diesem Kapitel vorgestellten Werkzeuge basieren auf einem dedizierten *Interaction Service*, welcher Grundbestandteil einer jeden *ASaP*-Installation ist. Der *Devconsole Smart Service* stellt hierzu eine bidirektionale WebSocket Verbindung zwischen der Plattform und den jeweiligen Werkzeugen zur Verfügung. Abbildung 10.1 gibt einen Überblick über die zugrunde liegende Softwarearchitektur.

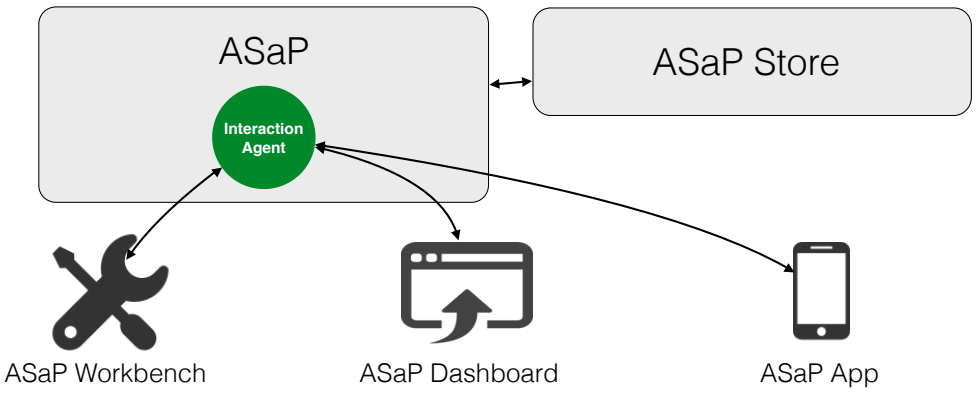


Abbildung 10.1: Überblick über die Anbindung der *Smart Service* Werkzeuge innerhalb der *ASaP*-Gesamtarchitektur.

10.2 Eclipse Plattform als Basis für die ASaP-Workbench

Die in diesem Kapitel vorgestellten Werkzeuge bauen auf Eclipse 4.4 auf und setzen sich aus verschiedenen Softwarekomponenten zusammen. In der Eclipse Terminologie werden einzelne Softwarekomponenten als *Plugins* bezeichnet. Da die Eclipse Technologie als Laufzeitkomponente auf Equinox OSGi (siehe Kapitel 6.5.1) basiert, können *Plugins* auch als *Bundles* bezeichnet werden.

Eclipse[1] ist primär eine weit verbreitete, quelloffene integrierte Entwicklungsumgebung (IDE) für eine Vielzahl unterschiedlicher Programmiersprachen und Softwareprojekte. Eclipse bietet den Entwicklern darüber hinaus mit der *Plug-in Development Environment* (PDE) die Möglichkeit, eigene Weiterentwicklungen der IDE zu implementieren und Eclipse somit auf die eigenen Bedürfnisse anzupassen (Gamma und Beck, 2003). Darüber hinaus wird Entwicklern durch die *Rich Client Plattform* (RCP) ein Framework an die Hand gegeben, um eigenständige und von der Eclipse IDE unabhängige Anwendungen zu realisieren (McAffer et al., 2010). Grundsätzlich besteht jede, mit Hilfe des Eclipse Frameworks erstellte Anwendung aus den folgenden vier visuellen Konzepten:

- **Sichten:** Sichten oder *Views* werden typischerweise verwendet, um Entwicklungsressourcen zu verwalten, zu navigieren und deren Eigenschaften zu modifizieren. Aktionen und Veränderungen innerhalb eines *Views* werden direkt wirksam und erfordern keine aktive Speicherung durch den Benutzer.

- **Editoren:** Editoren ermöglichen das Editieren spezifischer Ressourcen, wie beispielsweise das Verändern von Quelltexten oder das Manipulieren eines zugrunde liegenden Modells. Damit Änderungen wirksam werden, müssen diese explizit durch den Benutzer gespeichert werden.

- **Perspektiven:** Perspektiven stellen eine zweckorientierte Sammlung und Anordnung unterschiedlicher *Views* und Editoren bereit und definieren somit jeweils die aktuell sichtbare Bedienoberfläche einer Eclipse Anwendung.

- **Wizards:** *Wizards* bieten die Möglichkeit, die für den Ablauf der Anwendung notwendigen Benutzerinformationen in einer strukturierten Art und Weise vom Benutzer zu erfragen. *Wizards* bestehen in der Regel aus Eingabe- oder Ausgabefeldern und können thematisch in mehrere *Wizard Pages* gegliedert sein.

Abbildung 10.2 gibt einen Überblick über die Kernkomponenten einer Eclipse IDE, auf deren Basis die nachfolgend beschriebene *ASaP-Workbench* entwickelt wurde. Ein detaillierte Beschreibung der einzelnen Komponenten der Eclipse Plattform ist in Vogel (2013) zu finden.

[1] http://www.eclipse.org [Letzter Zugriff: 31.01.2015]

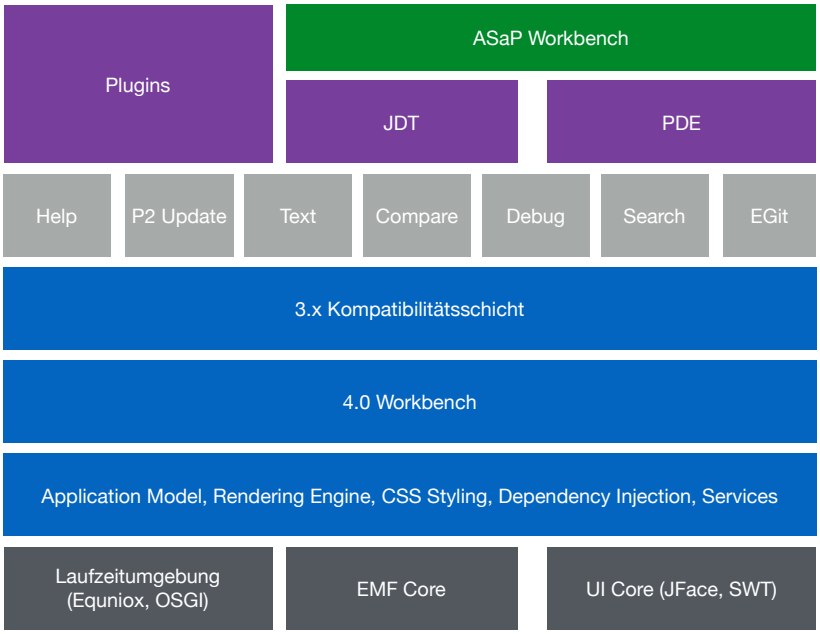


Abbildung 10.2: Überblick über die zugrunde liegenden Eclipse Technologiekomponenten der *ASaP-Workbench* (adaptiert nach Vogel (2013)).

Die folgenden Abschnitte zeigen, wie die *ASaP-Workbench* die Eclipse Plattform erweitert, um Entwicklern und Dienstanbietern eine durchgängige Werkzeugkette für die Entwicklung von *Smart Services* bereitzustellen. Eclipse stellt hierfür zwei Erweiterungsmechanismen zur Verfügung. *Plugins* können zum einen sogenannte *Extension Points* nutzen, um die Funktionalität anderer, bereits verfügbarer *Plugins* zu erweitern. Die zweite Möglichkeit zur Erweiterung ist die Erstellung von *Fragmenten*, welche das grundlegende *Application Model* der Anwendung ergänzen können. *Fragmente* sind wiederum kleine *Application Models* und beschreiben die Elemente, welche zu der Hauptanwendung hinzugefügt werden sollen.

10.3 ASaP-Workbench

Der Erfolg und die Verbreitung einer Softwareplattform hängt in hohem Maße von der Verfügbarkeit einer modernen Werkzeugunterstützung ab. Betrachtet man beispielsweise Android[2] und iOS[3] als die beiden Marktführer auf dem Gebiet der mobilen Softwareplattformen, so liegt laut Statista[4] die Anzahl der verfügbaren *Apps* für beide Plattformen bei jeweils über 1,2 Millionen. Beide Plattformen bieten ausgereifte Entwicklungswerkzeuge innerhalb eines kompletten Ökosystems an, um Entwicklern den Einstieg und die Erstellung von *Apps* so einfach wie möglich zu gestalten. Dies führt wiederum zu einer erhöhten Zahl von verfügbaren *Apps*, da mehr Entwickler sich damit beschäftigen,

[2]http://developer.android.com [Letzter Zugriff: 31.01.2015]
[3]http://developer.apple.com [Letzter Zugriff: 31.01.2015]
[4]http://de.statista.com/statistik/daten/studie/208599/umfrage/anzahl-der-apps-in-den-top-app-stores/ [Letzter Zugriff: 13.01.2015]

Mehrwerte für den Endnutzer zu generieren. Der Nutzer entscheidet sich nicht zuletzt aufgrund der Vielfalt der ihm angebotenen Mehrwertdienste zugunsten einer bestimmten Plattform. Ein ähnlicher Ansatz soll nun auch basierend auf dem in der vorliegenden Arbeit vorgestellten Plattformkonzept *ASaP* realisiert werden. Durch den Einsatz der *ASaP-Workbench* sollen *Smart Service* Entwickler während des gesamten Entwicklungszyklus ihrer Anwendungen unterstützt werden. Hierzu umfasst die *ASaP-Workbench* sowohl Entwicklungs- als auch Laufzeitwerkzeuge. Abbildung 10.3 gibt einen Überblick über die graphische Bedienoberfläche der *ASaP-Workbench*.

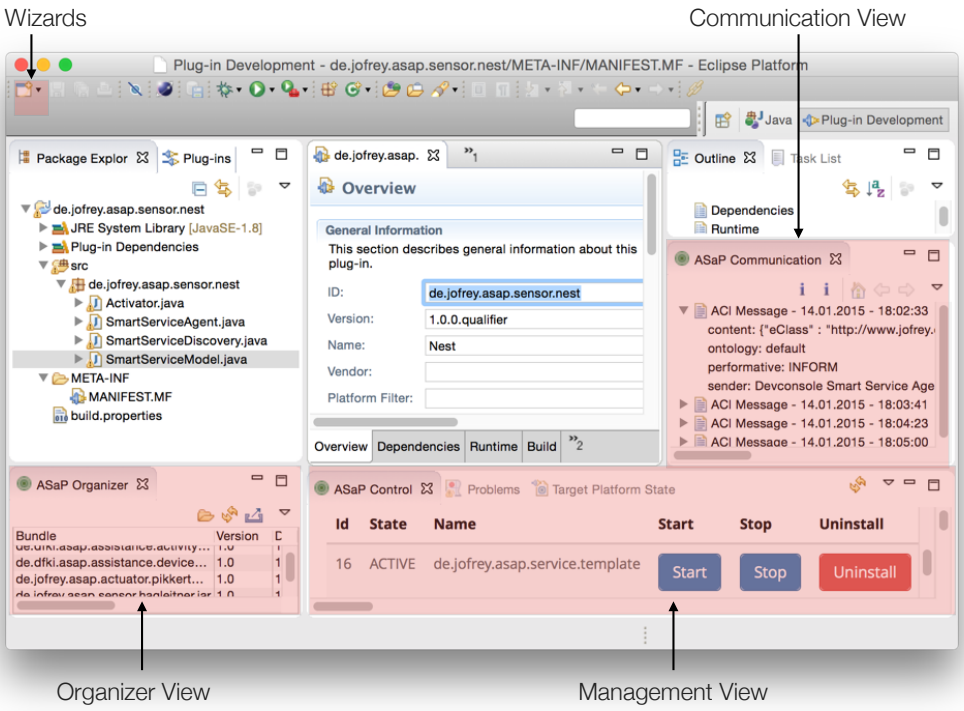


Abbildung 10.3: Übersicht über die graphische Bedienoberfläche der *ASaP-Workbench* basierend auf der Eclipse IDE.

Die Oberfläche stellt eine Erweiterung der *Plug-in Development Perspective* dar und ergänzt diese durch entsprechende *Views*, Editoren und *Wizards*, welche in den folgenden Abschnitten detailliert beschrieben werden. Die beschriebene Werkzeugsammlung ist zusammen mit der zugrunde liegenden Plattform als Open-Source Technologie verfügbar.

10.3.1 Entwicklungswerkzeuge

Entwicklungswerkzeuge kommen während der Designzeit eines *Smart Services* zum Einsatz und unterstützen den Entwickler bei der Konzeption, der Erstellung, der Verwaltung oder dem Wirkbetrieb der Anwendung.

New Smart Service Project Wizard

Im ersten Schritt sollen die Entwickler bei der Erstellung eines neuen *Smart Service* Projekts unterstützt werden. Hierzu kommen *Code Templates* zum Einsatz, mit deren Hilfe der Entwickler, gemäß dem zugrunde liegenden semantischen Modell, ein Grundgerüst eines *Smart Service* Projekts erstellen kann. Abbildung 10.4 zeigt die Dateistruktur eines minimalen *Smart Service* Projekts.

```
/
├── META-INF
│   ├── MANIFEST.MF
│   └── smartService.json
└── src
    ├── Activator.java
    ├── SmartServiceAgent.java
    ├── SmartServiceDiscovery.java
    └── SmartServiceModel.java
```

Abbildung 10.4: Überblick über die Dateistruktur eines minimalen *Smart Service* Projekts.

Durch geeignete Codegeneratoren werden hierzu die entsprechenden Dateien in Form von Java Klassen und Konfigurationsdateien erzeugt. Damit ein Projekt entsprechend generiert werden kann, muss der Entwickler Metainformation über den zu realisierenden *Smart Service* manuell bereitstellen. Die verfügbaren Eigenschaften eines *Smart Services* basieren auf der in Kapitel 8 vorgestellten, semantischen Modellierung einer *Intelligenten Umgebung*. Durch die Verwendung von EMF wird hierbei eine durchgängige Integration in die Werkzeugkette gewährleistet. Ändern sich beispielsweise die zugrunde liegenden ontologischen Konzepte, so wird diese Veränderung automatisch innerhalb der *ASaP-Workbench* repräsentiert. Zur strukturierten Eingabe der benötigten Metainformationen durch den Entwickler wird im Folgenden die Implementierung eines *New Smart Service Project Wizards* beschrieben. Nach Abschluss der Benutzereingabe wird durch den verwendeten Codegenerator ein Grundgerüst des Projektes angelegt und innerhalb der Eclipse IDE als standardmäßiges *Plugin Project* behandelt. Der Entwickler kann wie gewohnt sofort mit der Implementierung der Servicefunktionalitäten beginnen. Die entsprechende Kommunikationslogik des zugrunde liegenden *Service Agenten* ist vollständig vorhanden, das heißt der Agent ist bereits dazu in der Lage, die gemäß seiner semantischen *Input* Beschreibung für ihn relevanten ACL-Nachrichten zu empfangen und weiterzuverarbeiten. Der Entwickler kann sich hierdurch voll und ganz auf die Implementierung der internen Logik des Services konzentrieren, wie beispielsweise das Analysieren eingehender Sensordaten oder die Anbindung und Kontrolle eines spezifischen Haushaltsgeräts. Abbildung 10.5 zeigt eine Übersicht der entsprechenden Bedienoberflächen des realisierten *New Smart Service Project Wizards*.

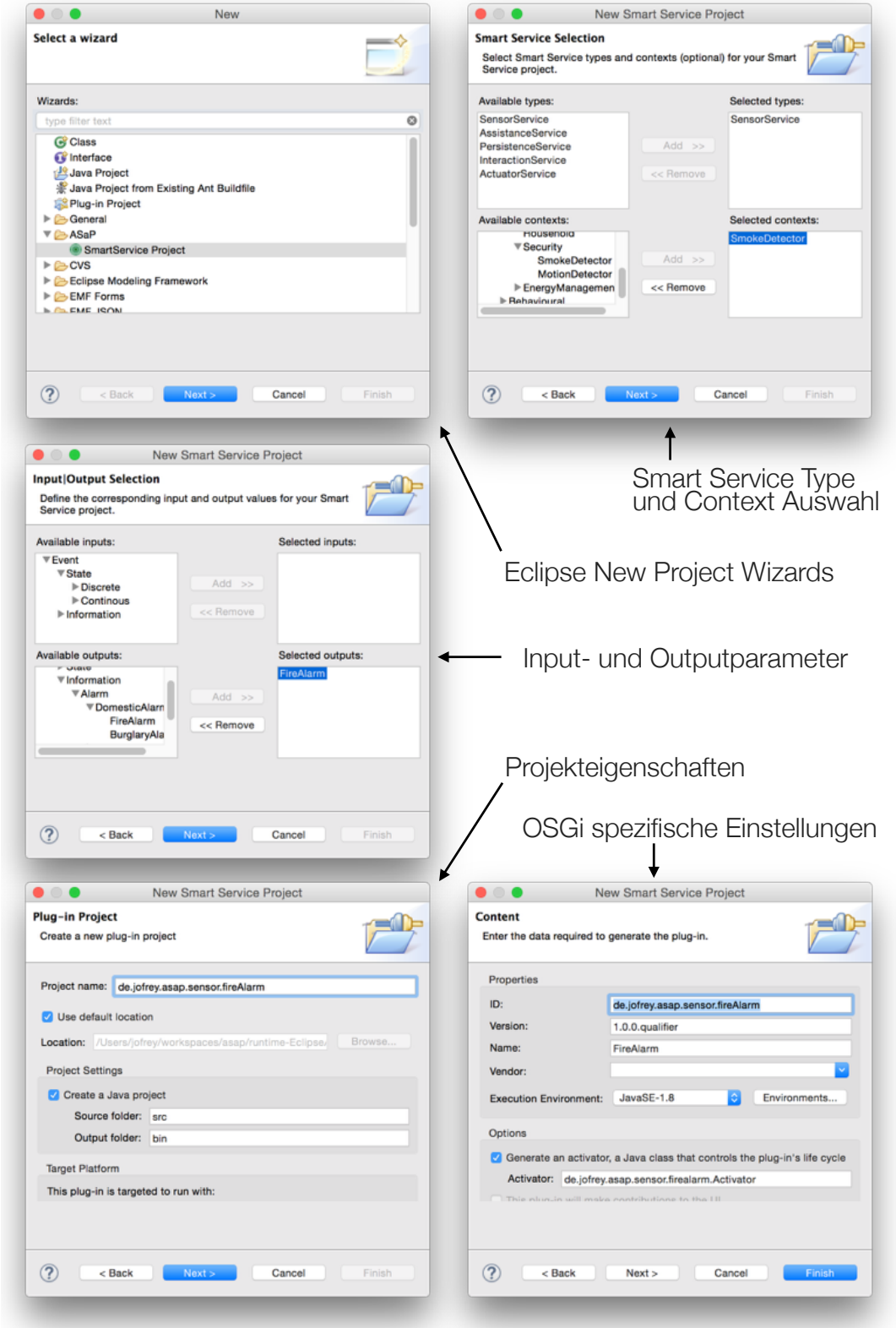


Abbildung 10.5: Überblick über die Bedienoberflächen des *New Smart Service Project Wizards*.

Organizer View

Nach abgeschlossener Implementierung eines *Smart Services* kann das entsprechende Projekt durch den Einsatz des verfügbaren Exportmechanismus der Eclipse IDE als *Deployable plug-ins and fragements* exportiert werden. Mit Hilfe des *Organizer Views* können die exportierten Versionen einzelner *Smart Service* Projekte verwaltet werden. Abbildung 10.6 zeigt eine Übersicht über die graphische Oberfläche des *Organizer Views*.

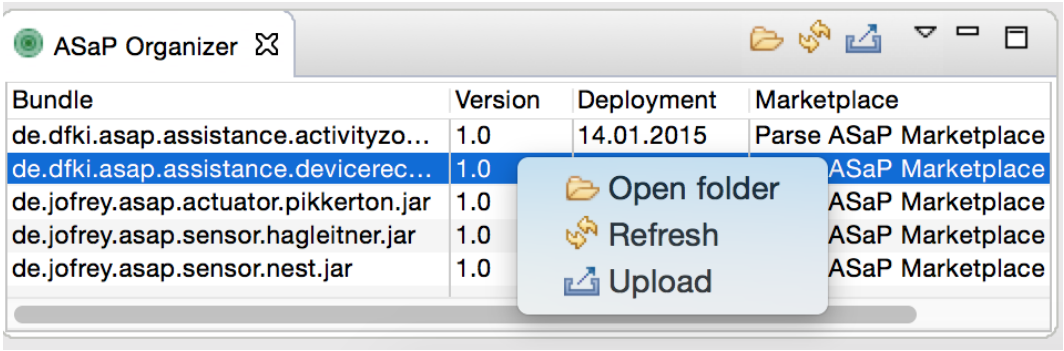


Abbildung 10.6: Bedienoberfläche des *Organizer Views* zur Verwaltung, Versionierung und Bereitstellung der eigenen *Smart Service* Bundles.

Der *ASaP-Organizer* zeigt neben dem Namen der entwickelten *Smart Service Bundles* auch die zugehörige Versionierungsnummer, das Datum einer Veröffentlichung und den Namen des entsprechenden Marktplatzes an. Über ein Kontextmenü können die Services schließlich durch den Einsatz eines *Upload Smart Service Bundle Wizards* auf unterschiedlichen Marktplätzen veröffentlicht werden.

Upload Smart Service Bundle Wizard

Um Medienbrüche zu vermeiden, können exportierte Projekte direkt aus der *Organizer View* der *ASaP-Workbench* auf einem globalen Service Marktplatz veröffentlicht werden. Zur Unterstützung des Entwicklers wird hierfür wiederum ein entsprechender *Upload Smart Service Bundle Wizard* bereitgestellt. Der Entwickler kann zunächst einen der verfügbaren Marktplätze auswählen. Durch die Implementierung geeigneter Konnektoren können unterschiedliche, zur Plattform kompatible Marktplätze integriert werden. Im Anschluss werden die entsprechenden Metainformationen des *Service Bundles* zusammengefasst und dem Entwickler somit eine abschließende Möglichkeit zum Bearbeiten und zum Hinzufügen eines beschreibenden Kommentars gegeben. Abbildung 10.7 zeigt die relevanten graphischen Bedienoberflächen des Wizards.

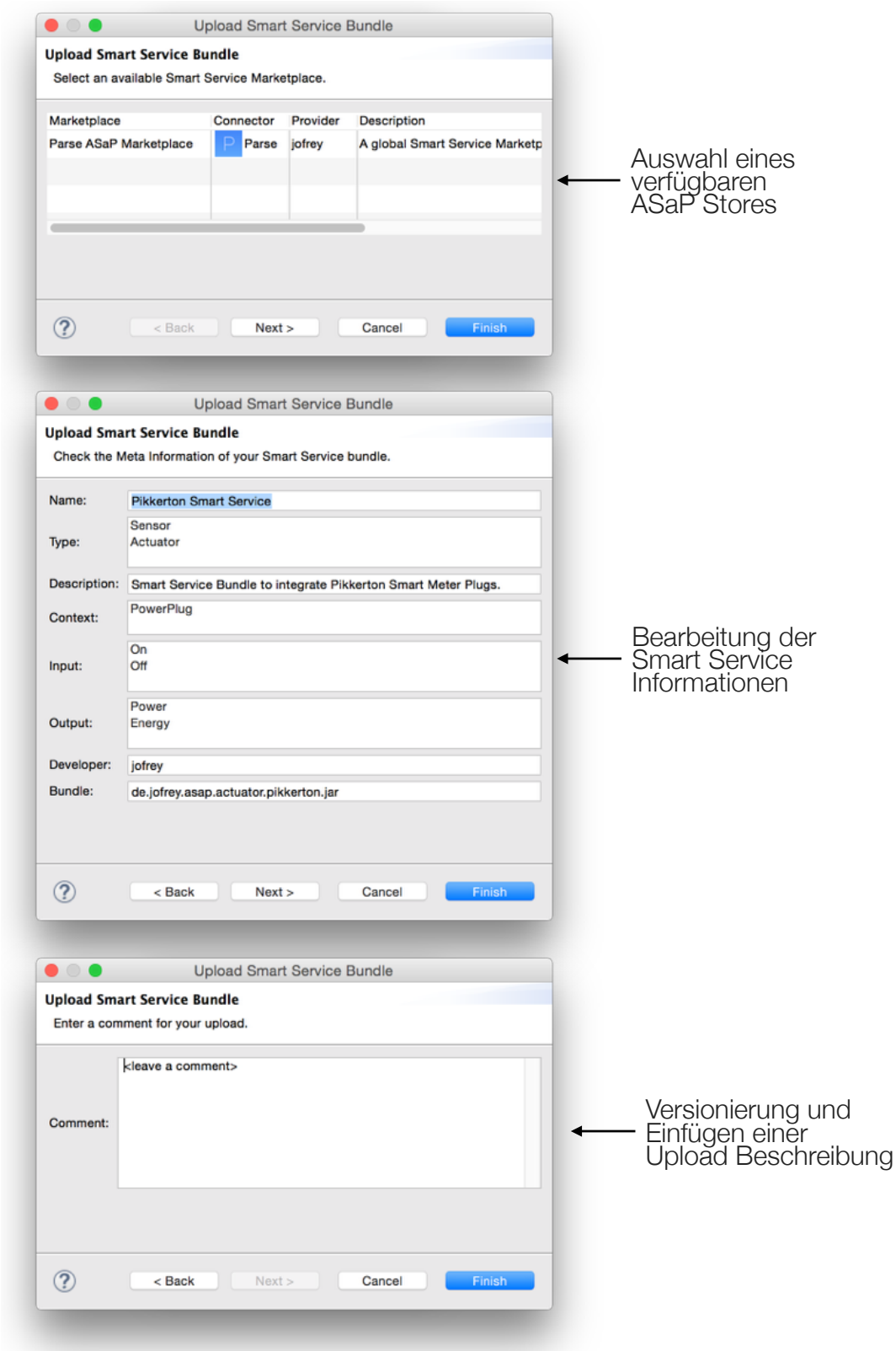


Abbildung 10.7: Überblick über die Bedienoberflächen des *Upload Smart Service Bundle Wizards*.

Ontologieeditor

Durch die Verwendung von EMF und der entsprechenden *Modeling Perspective* innerhalb der *ASaP-Workbench* werden zwei standardmäßige Editoren zum Bearbeiten der zugrunde liegenden semantischen Modelle zur Verfügung gestellt. EMF bietet hierbei innerhalb der Eclipse IDE sowohl einen baumartigen als auch einen klassendiagrammartigen Editor an. Abbildung 10.8 zeigt die graphische Bedienoberfläche beider Editoren.

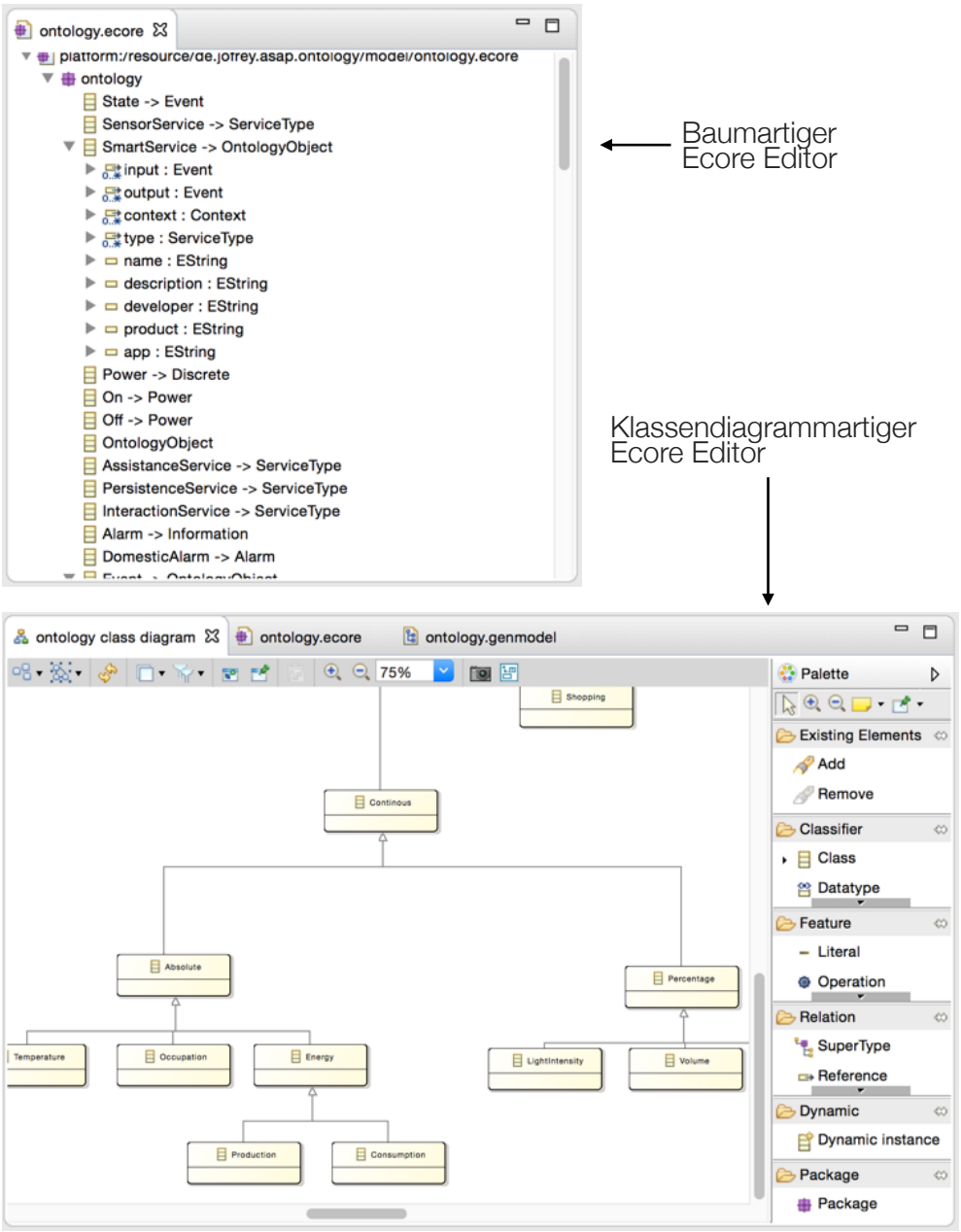


Abbildung 10.8: Überblick über die verfügbaren und innerhalb der *ASaP-Workbench* integrierten EMF Editoren.

Darüber hinaus ist durch die *Modeling Perspective* eine vollständige Werkzeugsammlung für EMF Modelle vorhanden, so dass der komplette Generierungsvorgang des semantischen Modells abgebildet werden kann. Insbesondere wird sowohl dem Plattform- als auch dem Serviceentwickler die Möglichkeit gegeben bestehende Konzepte innerhalb der *ASaP-Ontology* zu erweitern. Da alle in diesem Kapitel vorgestellten Werkzeuge auf den Konzepten der *ASaP-Ontology* aufbauen, führt eine Veränderung des Ecore Modells auch zu einer entsprechenden Anpassung der jeweiligen Werkzeuge.

10.3.2 Laufzeitwerkzeuge

Laufzeitwerkzeuge kommen während der Laufzeit einer *ASaP-Plattform* zum Einsatz und unterstützen den Entwickler beim Testen, Überwachen und bei der Fehlerbehebung der von ihm entwickelten *Smart Services*.

Management View

Zur Laufzeit von *ASaP* kann es zur Fehlersuche notwendig sein, die der Plattform zugrunde liegende OSGi Schicht zu überwachen und zu verwalten. Durch den *Management View* wird dem Entwickler eine Möglichkeit geboten, die installierten *Smart Service Bundles* der laufenden *ASaP*-Instanz aus der *ASaP-Workbench* heraus zu verwalten und zu manipulieren. Der Entwickler kann hierzu direkt den Zustand einzelner Bundles auslesen sowie diese aktiv starten, stoppen und wieder von der Plattform entfernen. Abbildung 10.9 zeigt die graphische Bedienoberfläche des *Management Views*.

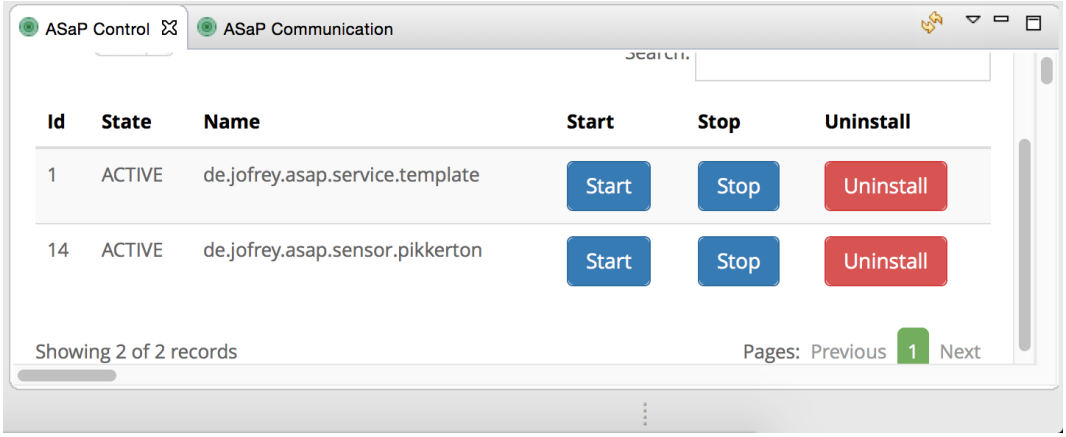


Abbildung 10.9: Überblick über die Bedienoberfläche des *Management Views* zur Verwaltung und zur Kontrolle der installierten *Smart Service Bundles*.

Communication View

Der *Communication View* gibt dem Entwickler Zugriff auf die plattforminterne Kommunikation einer *ASaP*-Instanz. Hierzu können alle durch *Smart Service Agenten* versendeten ACL-Nachrichten angezeigt sowie deren Inhalte,

wie Sender, kommunikativer Akt und das semantische Modell visualisiert werden. Auf diese Art und Weise werden die internen Vorgänge einzelner Agenten für den Entwickler sichtbar und nachvollziehbar, wodurch insgesamt eine bessere Debuggingmöglichkeit gegeben ist. Abbildung 10.10 zeigt die graphische Bedienoberfläche des *Communication Views*.

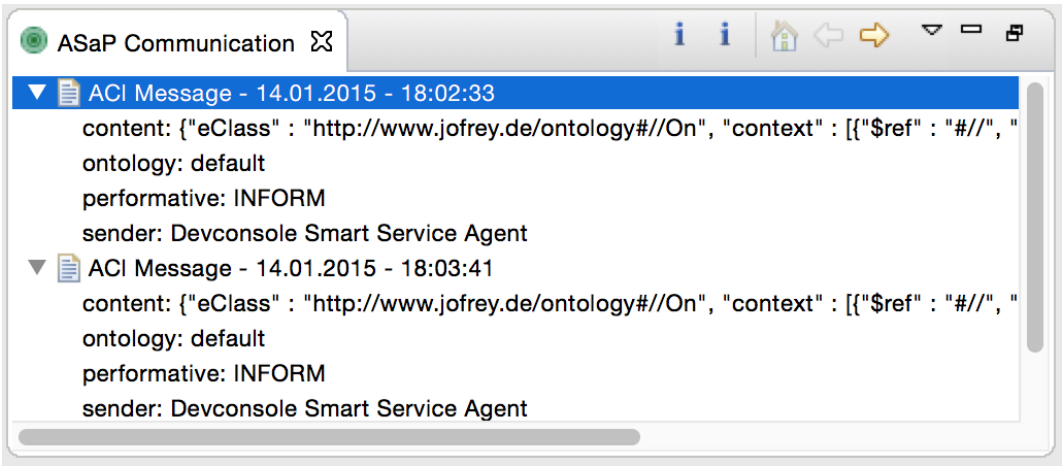


Abbildung 10.10: Überblick über die Bedienoberfläche des *Communication Views* zur Visualisierung der plattforminternen ACL-Nachrichten.

Neben der beschriebenen *ASaP-Workbench* zur Unterstützung der Entwickler von *Smart Services* werden im nächsten Schritt den Endkunden entsprechende Werkzeuge an die Hand gegeben, mit deren Hilfe sie jederzeit in der Lage sind, ihre *ASaP-Plattform* zu verwalten. Hierbei sollen die folgenden Anforderungen erfüllt werden:

- **Installation neuer *Smart Services*:** Dem Benutzer muss hierzu einen Zugriff zu einem verfügbaren *Smart Service Marktplatz* angeboten werden. Über die Schnittstelle soll er in die Lage versetzt werden, neue und auf seine Plattform passende *Smart Services* zu finden und direkt zu installieren. Darüber hinaus sollen eventuelle Abhängigkeiten zu anderen *Smart Services* transparent angezeigt werden.

- **Management und Verwaltung der laufenden *ASaP*-Instanz:** Der Benutzer muss dazu befähigt werden, seine installierten *Smart Services* zu verwalten. Das bedeutet, dass er jederzeit die Möglichkeit haben muss, Services zu stoppen, wieder zu starten oder vollständig von seinem System zu deinstallieren.

- **Manipulation und Überwachung des Umgebungszustandes:** Dem Benutzer soll eine einfache Möglichkeit zur Verfügung stehen, den Zustand seiner Umgebung zu überwachen und gegebenenfalls zu manipulieren.

10.4 ASaP-Dashboard

Das *ASaP-Dashboard* baut technologisch auf HTML5 und JavaScript auf und implementiert somit die zuvor genannten Anforderungen anhand einer webbasierten Browseranwendung. Als Server dient hierbei der Rechner, auf dem die jeweilige *ASaP*-Instanz installiert wurde. Das Dashboard ist im gesicherten lokalen Netzwerk der Umgebung unter der folgenden Adresse verfügbar: `http://<ASaP Host IP>:1112`. Neben einer grundlegenden Seitennavigation hat der Benutzer die Möglichkeit einen oder mehrere *Smart Service* Marktplätze zu durchsuchen. Durch die direkte Anbindung der Schnittstelle an die laufende *ASaP*-Instanz kann zudem die Suche auf *Smart Services* eingeschränkt werden, welche entsprechend ihrer *Input* und *Output* Abhängigkeiten zu der zugrunde liegenden Infrastruktur passen. Weiterhin können mit Hilfe der Managementschnittstelle des Dashboards die ausgewählten Services direkt installiert, gestartet, gestoppt und auch wieder vollständig vom System entfernt werden. Abschließend wurde eine konsolenartige universelle Interaktionsschnittstelle realisiert, die sowohl dem Entwickler zu Debuggingzwecken als auch dem versierten Endnutzer die Möglichkeit gibt, die laufende *ASaP*-Instanz sowie die installierten *Smart Services* über einen Kommandozeileninterpreter zu kontrollieren und zu manipulieren. Tabelle 10.1 gibt eine Übersicht über die verfügbaren Kommandos innerhalb des *ASaP-Dashboards*.

Kommando	**Beschreibung**
lb	Listet alle installierten *Smart Service Bundles* auf.
ls	Listet alle verfügbaren *Smart Services* auf.
start	Startet ein installiertes *Smart Service Bundle*. Als Parameter wird die entsprechende *Bundle ID* übergeben.
stop	Stoppt ein installiertes *Smart Service Bundle*. Als Parameter wird die entsprechende *Bundle ID* übergeben.
install	Installiert ein *Smart Service Bundle*. Als Parameter wird die entsprechende *Bundle URL* übergeben.
uninstall	Deinstalliert ein *Smart Service Bundle*. Als Parameter wird die entsprechende *Bundle ID* übergeben.
help	Listet die ausführbaren Zustandsveränderungen und die entsprechenden Umgebungskontexte auf.

Tabelle 10.1: Übersicht über die verfügbaren Kommandos innerhalb des *ASaP-Dashboards*.

Das *ASaP-Dashboard* erfüllt die Anforderungen des *Responsive Web Designs* (Marcotte, 2011), das heisst, der gesamte Inhalt und das Layout der Anwendung passt sich automatisch an das entsprechende Ausgabegerät an. Abbildung 10.11 gibt eine Übersicht über die unterschiedlichen Bedienoberflächen des *ASaP-Dashboards*.

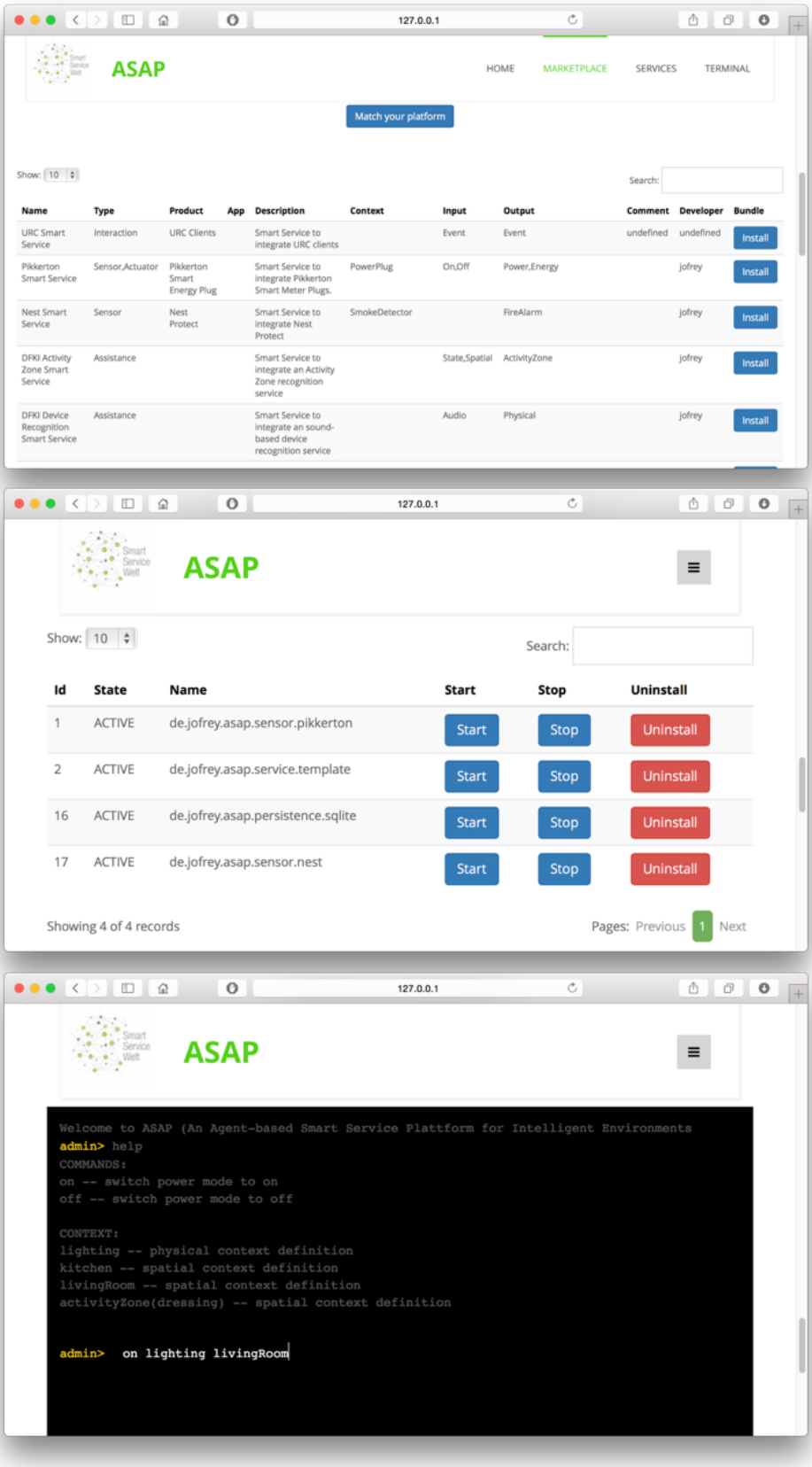


Abbildung 10.11: Überblick über die Bedienoberfläche des *ASaP-Dashboards*.

10.5 ASaP-App

Neben der webbasierten Anwendung wurde auch eine nativ implementierte, iOS-basierte, mobile Anwendung zur Verwaltung einer laufenden *ASaP*-Instanz erstellt. Der Funktionsumfang erfüllt die zuvor gestellten Anforderungen und entspricht somit größtenteils den Funktionen des *ASaP-Dashboards*. Im Gegensatz zu der zuvor verwendeten und eher auf Entwickler ausgelegten, kommandozeilenbasierten Interaktion wird dem Benutzer innerhalb der mobilen Applikation eine intuitive und auf Touchgesten basierende Möglichkeit zur Kontrolle der *Smart Services* und damit der zugrunde liegenden *Smart Home* Umgebung geboten. Abbildung 10.12 zeigt die zur Verfügung gestellten Navigationselemente der *ASaP-App*.

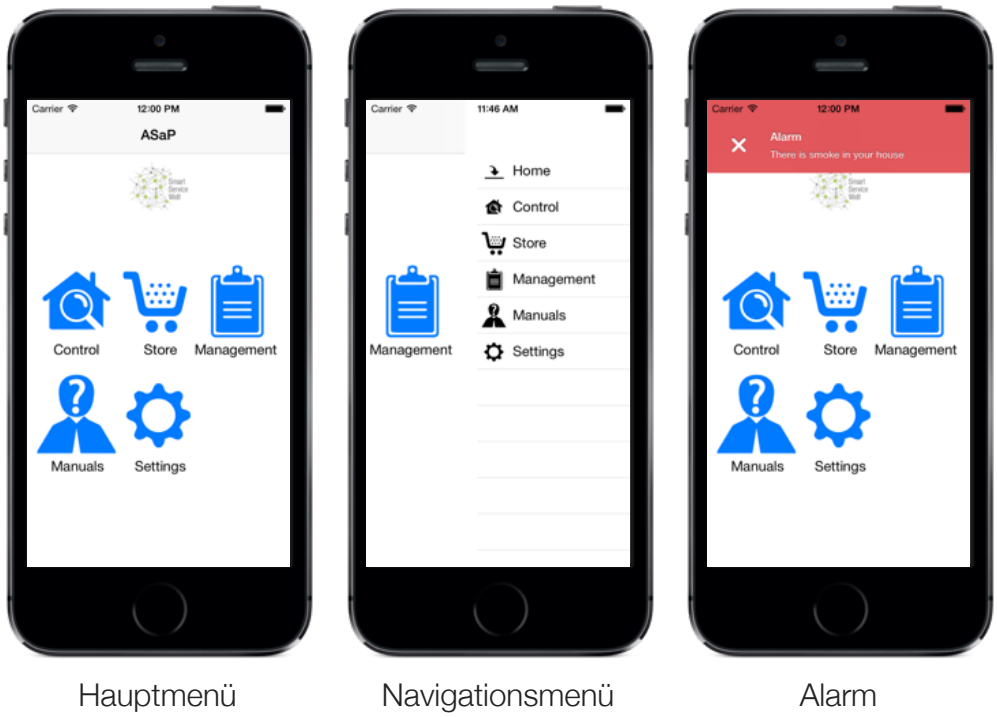


Abbildung 10.12: Überblick über die Navigationselemente der *ASaP-App*.

Die Startseite enthält das Hauptmenü und bietet somit Zugriff auf die einzelnen Funktionalitäten der *ASaP-App*. Durch den Einsatz eines seitlichen Navigationsmenüs, welches durch eine entsprechende Wischgeste aufgerufen werden kann, hat der Benutzer jederzeit einen schnellen Zugriff auf die jeweiligen Menüpunkte. Darüber hinaus können wichtige und für den Benutzer relevante Benachrichtigungen und Informationen, wie zum Beispiel ein Alarm, direkt innerhalb der Bedienoberfläche der *ASaP-App* visualisiert werden. Die Hauptfunktionalität der *ASaP-App* ist die Kontrolle und die Manipulation der *Smart Home* Umgebung durch die Interaktion mit den installierten *Smart Services*. Abbildung 10.13 zeigt eine exemplarische Bedienoberfläche zur Kontrolle der verfügbaren *Sensoren* und *Aktuatoren*. Die jeweilige Darstellung ist hier-

bei abhängig von der tatsächlichen Instrumentierung und basiert auf der in Kapitel 8 vorgestellten semantischen Modellierung der Umgebung.

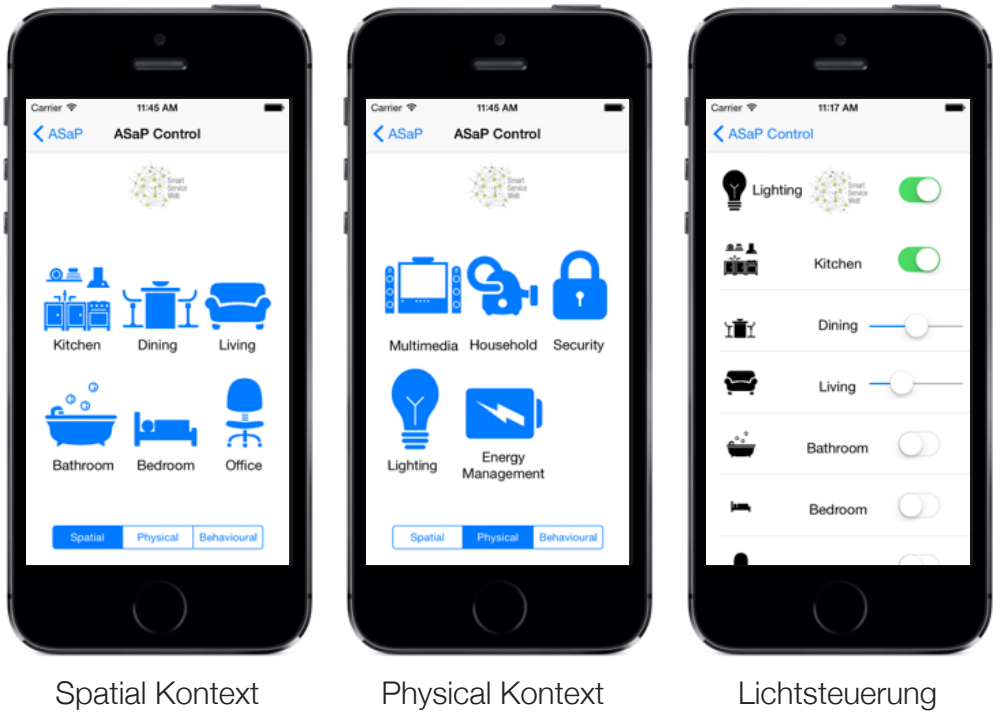


Abbildung 10.13: Überblick über Kontrollfunktionalitäten der *ASaP-App*.

Die verfügbaren *Events* der installierten *Smart Services* sind entsprechend ihrer Kontextbeschreibung in *Spatial*, *Physical* oder *Behavioural* Gruppen sortiert. Die konkrete Kontrolloberfläche unterscheidet wiederum zwischen den möglichen Zuständen eines angeschlossenen *Sensors* oder *Aktuators*. Im Beispiel der gezeigten Lichtsteuerung eines kompletten *Smart Homes* können folglich sowohl normale Lampen mit diskreten Zuständen ein- oder ausgeschaltet werden, aber auch dimmbare Lichtinstallationen angesteuert werden.

Die *ASaP-App* bietet dem Benutzer einen direkten Zugang zu einem oder mehreren verfügbaren *ASaP-Stores*. Analog zu dem bereits vorgestellten *ASaP-Dashboard* kann der Benutzer neue *Smart Services* suchen oder entsprechend seiner vorhandenen Plattform filtern. Durch eine seitliche Wischgeste in der tabellarischen Übersicht wird ein Kontextmenü aufgerufen, über das Detailinformationen des *Smart Services* angezeigt oder der *Smart Service* direkt auf der lokalen *ASaP*-Instanz installiert werden kann. Wurde der *Smart Service* erfolgreich installiert, erhält der Benutzer eine entsprechende Benachrichtigung innerhalb der Anwendung. Die *ASaP-App* bietet dem Benutzer schließlich auch die Möglichkeit, die installierten *Smart Services* zu verwalten, das heißt entsprechende *Service Bundles* starten, stoppen oder vollständig entfernen zu können. Abbildung 10.14 gibt eine Übersicht über die beschriebenen Verwaltungsoberflächen.

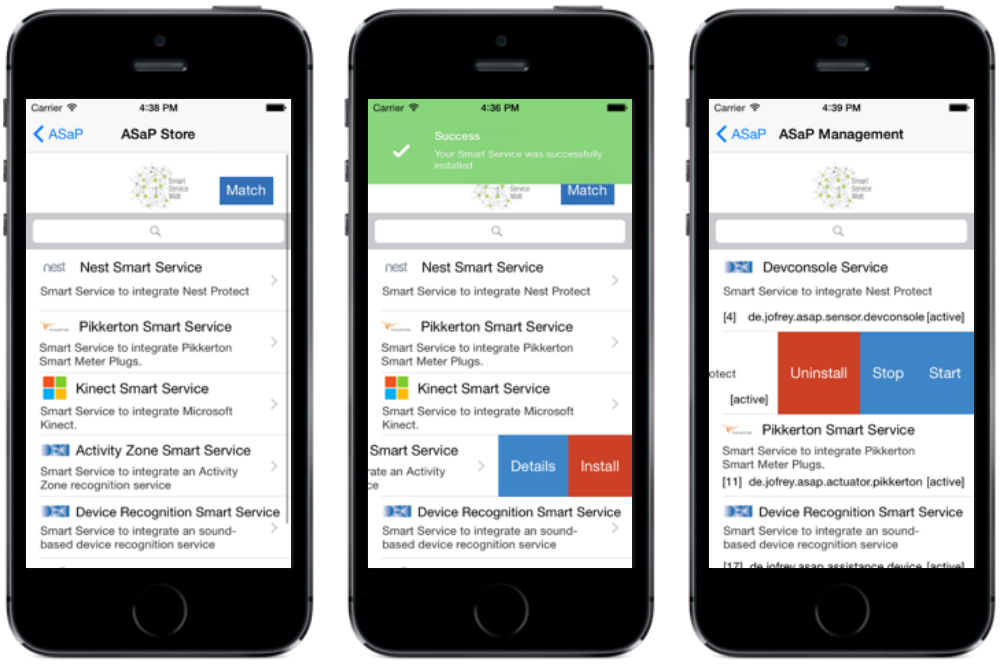


Abbildung 10.14: Überblick über Verwaltungsfunktionalitäten der *ASaP-App*.

10.6 Zusammenfassung und Fazit

In diesem Kapitel wurde mit der *ASaP-Workbench* eine durchgängige Werkzeugkette für Dienstanbieter beschrieben, die es Entwicklern erleichtern soll, digitale Mehrwertdienste in Form von *Smart Services* zu entwickeln und auf einem entsprechenden *ASaP-Store* zu vertreiben. Die *ASaP-Workbench* baut technologisch auf der Eclipse IDE auf und erweitert hierzu die *Plug-in Development Perspective* durch die folgenden *ASaP*-spezifischen Entwicklungs- und Laufzeitwerkzeuge:

- ***New Smart Service Wizard*** zur Unterstützung der Entwickler bei der Erstellung von neuen *Smart Service* Projekten
- ***Upload Smart Service Wizard*** zur Unterstützung der Entwickler beim Veröffentlichen der realisierten *Smart Services*
- ***Organizer View*** zur Verwaltung, Versionierung und Wirkbetrieb von *Smart Service Bundles*
- ***Management View*** zur Verwaltung und Kontrolle der innerhalb einer *ASaP-Middleware* installierten *Smart Service Bundles*
- ***Communication View*** zur Visualisierung der plattforminternen ACL-Nachrichten

- ***Ontologieeditor*** zum Bearbeiten der zugrunde liegenden semantischen Modelle der *ASaP-Ontology*

Weiterhin wurde ein browserbasiertes *ASaP-Dashboard* sowie eine mobile *ASaP-App* bereitgestellt. Beide Anwendungen ermöglichen dem Benutzer, seine vorhandene *ASaP*-Instanz zu verwalten, zu überwachen, zu manipulieren und über einen *ASaP-Store* durch die Installation neuer *Smart Services* dynamisch zu erweitern. Im folgenden Kapitel werden abschließend konkrete Anwendungsgebiete und Einsatzbeispiele der entwickelten Plattformen vorgestellt.

Holzhacken ist deshalb so beliebt, weil man bei dieser Tätigkeit den Erfolg sofort sieht.

Albert Einstein

11 Anwendungsgebiete und Einsatzbeispiele

11.1 Einleitung und Überblick

Um die praktische Relevanz und die Benutzbarkeit einer Plattform bewerten zu können, müssen konkrete Anwendungen implementiert werden. Im folgenden Kapitel werden daher konkrete Einsatzbeispiele der zuvor beschriebenen Konzepte und Realisierungen gegeben. Die einzelnen Anwendungen sind hierbei den entsprechenden Forschungsprojekten zugeordnet. Die Grundlage aller Demonstratoren und Prototypen bilden die in dieser Arbeit vorgestellten Plattformkonzepte, welche je nach Anwendungsfall eine benutzerzentrierte oder serviceorientierte Zielsetzung verfolgen. Die Strukturierung der Einsatzbeispiele lässt zudem die chronologische Entwicklung der vorgestellten Plattformkonzepte erkennen und verdeutlicht auf diese Art und Weise auch die Weiterentwicklung der initialen URC-Plattform hin zu einer agentenbasierten Plattform für *Smart Services* in *Intelligenten Umgebungen*. Die im Folgenden vorgestellten Arbeiten wurden größtenteils im Rahmen des *Kompetenzzentrums für Ambient Assisted Living* (Frey et al., 2010d) am Deutschen Forschungszentrum für Künstliche Intelligenz[1] (DFKI) umgesetzt.

Abschnitt 11.2 gibt zunächst eine kurze Einführung in die zugrunde liegenden Forschungsprojekte, in deren Rahmen die nachfolgenden Demonstratoren umgesetzt wurden. In Abschnitt 11.3 werden im Anschluss mehrere Anwendungen auf der Basis der in Kapitel 6 vorgestellten URC-Architektur aufgezeigt. Abschnitt 11.4 skizziert einen *Smart Service* aus dem Bereich des dezentralen Handels von erneuerbaren Energien. In Abschnitt 11.5 werden schließlich zwei Demonstratoren auf der Basis der in Kapitel 7, 8 und 9 eingeführten *ASaP*-Plattform beschrieben.

[1] http://www.dfki.de [Letzter Zugriff: 31.01.2015]

11.2 Relevante Forschungsprojekte

11.2.1 i2home

Das Ziel des europäischen Forschungsprojekts i2home (Alexandersson et al., 2006; Alexandersson, 2008) war die Konzeption und die Realisierung einer offenen und auf dem *Universal Remote Console* Standard basierenden Plattform für die intuitive multimodale Interaktion mit Heimelektronik. Im Fokus der Forschung standen hierbei Benutzerschnittstellen für Personen mit besonderen Bedürfnissen, beispielsweise ältere Menschen oder Menschen mit leichten kognitiven Behinderungen. Trotz der Fokussierung auf diese Zielgruppen wurde entsprechend eines *Design-for-All-Ansatzes* die Entwicklung eines Systems angestrebt, welches für alle Benutzer geeignet und unabhängig von unterstützender Technologie anwendbar ist. Damit stellt das Projekt die technologische Basis für viele Anwendungen zur Verfügung, die aufbauend auf der URC-Infrastruktur entwickelt werden können. Die im Folgenden beschriebenen Anwendungen fokussieren auf die Entwicklung von graphischen und multimodalen Benutzerschnittstellen auf Basis der *Universal Remote Console* Infrastruktur und wurden im Rahmen von Benutzerstudien mit realen Benutzern auf ihre Bedienfreundlichkeit evaluiert. Die während der Entwicklungs- und Evaluationsprozesse gesammelten Erfahrungen und Ergebnisse (Zimmermann et al., 2010b) flossen direkt in die Konzeption und die Implementierung des in Kapitel 6 beschriebenen *Universal Control Hubs* mit ein.

11.2.2 SmartSenior

SmartSenior (Balasch et al., 2012) war mit insgesamt 28 Partnern aus Industrie und Forschung deutschlandweit eines der größten Forschungsprojekte auf dem Gebiet der altersgerechten Assistenzsysteme. Die Zielstellung des Projekts war die Entwicklung von Technologien und Dienstleistungen, mit denen Senioren unterstützt werden können, ihre eigene Selbständigkeit, Gesundheit, Mobilität und Sicherheit so lange wie möglich zu erhalten. Besonders in den Bereichen multimodale Interaktion und Lokalisierung, standardbasierte Interaktionsmodelle, Benutzerfreundlichkeit und Überwindung von Sprachbarrieren suchte SmartSenior nach Möglichkeiten und Strategien, um Senioren ein unabhängiges Leben zu ermöglichen. Ältere Menschen sollen hierbei wirtschaftliche, soziale und gesundheitstechnische Unterstützung erfahren, damit sie weiter in ihrer familiären Umgebung leben können. SmartSenior wurde sowohl für Senioren konzipiert, die im Großen und Ganzen ein unabhängiges Leben führen, als auch für schwerkranke oder chronisch kranke Menschen, die Pflege und Hilfe benötigen. Zusammengefasst adressieren die Forschungsschwerpunkte von SmartSenior die folgenden drei Lebensbereiche: länger selbständig im häuslichen Umfeld leben, sicher unterwegs sein sowie gesund werden und bleiben.

11.2.3 Peer Energy Cloud

Das Ziel von Peer Energy Cloud (PEC) (Brandherm et al., 2013) ist die Entwicklung innovativer Erfassungs- und Prognoseverfahren für die Lastgangsentwicklung mit dem Ziel der Lastflussoptimierung sowie die Etablierung eines virtuellen Markplatzes für den Stromhandel innerhalb eines sogenannten Smart Microgrids. Dieser Marktplatz ermöglicht einen lokalen Ausgleich zwischen Erzeugung und Verbrauch von Elektrizität innerhalb eines Subnetzes, was unmittelbar zu einer Entlastung übergeordneter Netzebenen führt. Dieser Ansatz trägt so zur Verbesserung der Netzwerk-Stabilität bei und verringert den Bedarf an einem kostenintensiven Ausbau überregionaler Stromnetze. Aktuell ist der tatsächliche Strombedarf zu einem bestimmten Zeitpunkt für die Stromerzeuger nicht verbrauchergenau voraussagbar. Die Standardlastprofile sollen durch den Einsatz von geeigneter Sensorik und intelligenten Prognoseverfahren durch individuelle Lastprofile ersetzt werden, was die Energieanbieter dazu in die Lage versetzt, ihren eigenen Stromeinkauf besser voraussagen und und somit besser planen zu können.

11.2.4 AdAPT

Das übergeordnete Ziel des durch den Softwarecampus[2] geförderten und in Kooperation mit der Deutschen Telekom AG durchgeführten Projekts AdAPT ist die Realisierung eines universellen Ansatzes zur Erkennung von Aktivitäten innerhalb eines *Smart Homes*. Die Grundidee ist dabei, dass sich die *Intelligente Umgebung* dynamisch und kontinuierlich auf die Benutzer und die sich ändernden Gegebenheiten anpasst. Hierzu wurde im Projekt das in der vorliegenden Arbeit vorgestellte agentenbasierte Plattformkonzept entwickelt und realisiert, welches die Integration von neuartigen *Smart Services* in eine *Intelligente Umgebung* ermöglicht. Auf *ASaP* aufbauend werden im Folgenden zwei unterschiedliche *Smart Services* zur Erkennung von Benutzeraktivitäten innerhalb eines *Smart Homes* detailliert beschrieben.

11.3 Demonstratoren und Beispielanwendungen auf der Basis von URC

11.3.1 Multimodales Dialogsystem zur Kontrolle und Steuerung von Haushaltsgeräten

Abbildung 11.1 gibt eine Übersicht über die in i2home realisierte mobile, multimodale Benutzerschnittstelle. Die Entwicklung der erforderten Interaktionskonzepte folgt hierbei einem *User-Centred-Design* Ansatz und ist in enger Zusammenarbeit mit realen Anwendern entstanden. Die Schnittstelle basiert auf einem multimodales Dialogsystem zur Unterstützung von Personen mit leichten kognitiven Behinderungen (Frey et al., 2010c). Aus funktionaler Sicht

[2] http://www.softwarecampus.de [Letzter Zugriff: 31.01.2015]

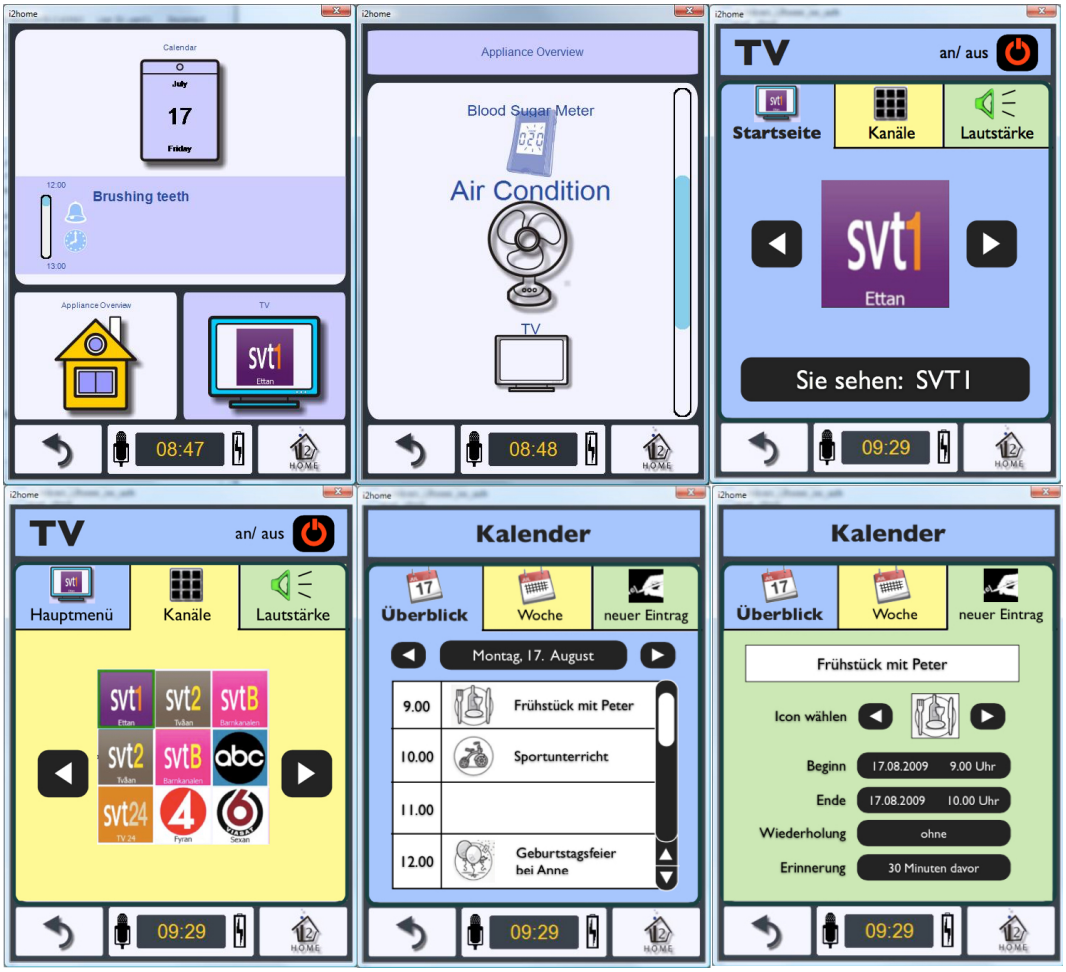


Abbildung 11.1: Multimodales Dialogsystem zur Steuerung von Haushaltsgeräten und zur Unterstützung des Tagesablaufs von Personen mit leichten kognitiven Behinderungen.

verfügt das System über Sprach- und Gestikinteraktionen, und bietet Hilfestellungen bei alltäglichen Handlungsroutinen, wie zum Beispiel die Steuerung der umgebenden Heimelektronik, das Überwachen medizinischer Vitalparameter oder das Verwalten von wichtigen Terminen und Erinnerungen. Technologisch basiert das Dialogsystem auf den Arbeiten von Pfleger (2008) und der daraus resultierenden *Ontology Dialogue Platform*, welche mittlerweile durch die Firma SemVox[3] zur Produktreife gebracht wurde und kontinuierlich weiterentwickelt wird. Darüber hinaus wurde eine Referenzimplementierung des *Universal Control Hubs* integriert, um die Anbindung der *Sensoren* und *Aktuatoren* und der jeweiligen abstrakten Benutzerschnittstellen zu realisieren.

[3]http://www.semvox.de [Letzter Zugriff: 31.01.2015]

11.3.2 Aufgabenbasierter Kalender zur Einhaltung der Therapietreue

Weiterhin wurde ein aufgabenbasierter Kalender realisiert, welcher im Zusammenspiel mit Googles Kalender Service und speziell angefertigten Aufgabenmodelle dazu in der Lage ist, kognitiv eingeschränkte Menschen in ihrem Alltag zu unterstützen. Neßelrath et al. (2011a) beschreiben zum Beispiel die Kombination aus einem intelligenten Medikamentenblister und einem interaktiven Kalender, um ältere Menschen bei ihrer täglichen Medikamenteneinnahme optimal zu unterstützen. Abbildung 11.2 zeigt den daraus resultierenden Demonstrator bestehend aus einem Informationsservice, welcher neben dem aufgabenbasierten Kalender auch über einen Einnahmeassistenten sowie über eine Anzeige zur persönlichen Medikamentenunverträglichkeit verfügt.

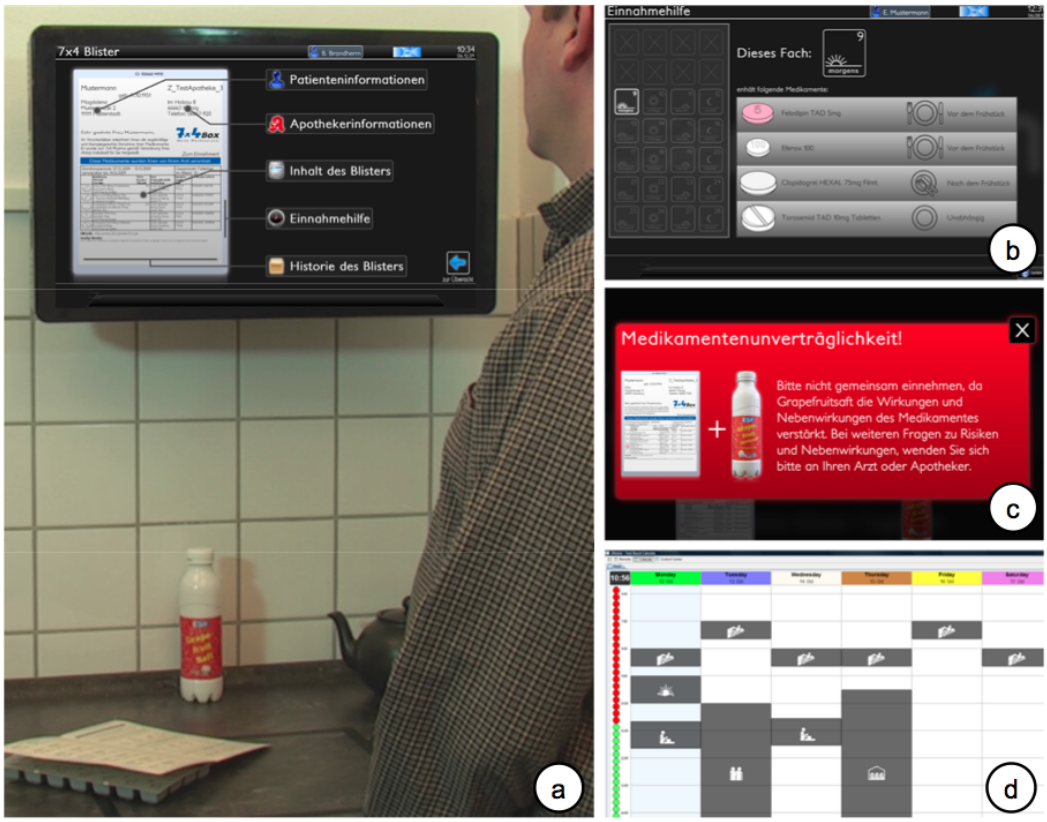


Abbildung 11.2: Informationsservice zur Unterstützung der Medikamenteneinnahme innerhalb der Smart Kitchen Umgebung: (a) Dashboard, (b) Einnahmeassistenz, (c) Medikamentenunverträglichkeitsanzeige, (c) Aufgabenbasierter Kalender zur Unterstützung der täglichen Routine der Benutzer.

Zur Realisierung des beschriebenen Szenarios wurde die in Kapitel 6 vorgestellte UCH-Implementierung wie folgt erweitert. Im ersten Schritt wurden drei neue *Target Adapters* realisiert, um neben dem Zugriff auf den *Google Kalender Service* und mehreren in der Umgebung verteilten *RFID-Readern* auch eine Schnittstelle zu *Digitalen Produktgedächtnissen* und dem entsprechenden *Object Memory Server* (Haupert, 2013) zu schaffen. Im zweiten Schritt wurde

die zugrunde liegende UCH-Architektur um eine socketübergreifende Pluginschicht erweitert. Das *SemProM Medication Plugin* beinhaltet hierbei die relevanten Regeln zur Medikamenteneinnahme und zur Medikamentenverträglichkeit und koordiniert den Informationsaustausch mit den zuvor genannten *Targets*. Konkret erlaubt das Plugin, dass innerhalb des beschriebenen Szenarios Einnahmehinweise und Daten aus dem Gedächtnis des Blisters gelesen, automatisch in den Kalender des Patienten eingetragen und entsprechend visualisiert werden können.

11.3.3 Dual und Synchronized Reality

SmartCase Simulationsumgebung

Im Rahmen von SmartSenior und des *Kompetenzzentrums für Ambient Assisted Living* am DFKI wurde mit SmartCase (Frey et al., 2011a) eine maßstabsgetreue Nachbildung einer realen Wohnung entworfen und vollständig instrumentiert. Das zugrunde liegende Konzept der Modellierung realer *Intelligenter Umgebungen* wird hierbei durch die folgenden drei Gesichtspunkte motiviert:

- Das erstellte Modell dient als Test- und Demonstrationsumgebung für *Smart Home* Technologie sowie für entwickelte Benutzerschnittstellen. Hierbei wird zudem der zuvor erwähnte Gedanke aufgegriffen, den Testnutzern die Effekte und Auswirkungen der Schnittstellen zu visualisieren und somit leichter verständlich zu machen.

- Ein dreidimensionales Modell der Umgebung kann Designer und Architekten in ihrem Entscheidungsprozess unterstützen, zum Beispiel im Hinblick auf die Fragestellung, welche Art von *Sensoren* und *Aktuatoren* verwendet werden und wo diese räumlich platziert werden sollen.

- Im Kontext von *Dual Reality* (Kahl, 2014) kann das Modell als Werkzeug genutzt werden, um entweder Zustände der Umgebung darzustellen oder umgekehrt diese zu manipulieren.

Auf der Basis der technischen Infrastruktur wurde darüber hinaus ein Mechanismus entwickelt, um automatisiert mobile und intuitive Benutzerschnittstellen zur Kontrolle und zur Überwachung der angeschlossenen Geräte zu generieren (Gholamsaghaee et al., 2012). Der SmartCase besteht aus zwei Teilen. Der untere Teil enthält das Modell einer realen Wohnung im Maßstab 1:20. Zur funktionalen Repräsentation der Haushaltsgeräte wie Herd, Ofen oder Beleuchtung wurden LED Dioden verbaut, welche über einen entsprechenden *Target Adapter* an die zugrunde liegende UCH-Architektur angeschlossen wurden. Die verwendeten UIS-Beschreibungen entsprechen hierbei denen von tatsächlichen Küchengeräten. Weiterhin wurden auf EnOcean[4] Technologie basierende *Sensoren* und *Aktuatoren* verbaut, um den Zustand von Türen und Fenstern zu

[4]http://www.enocean.com [Letzter Zugriff: 10.01.2015]

überwachen oder externe Geräte über eine vernetzte Steckdosenleiste anzusteuern. Der obere Teil des Koffers verfügt über einen Monitor, um Fernsehszenarien zu adressieren oder graphische Benutzerschnittstellen zu visualisieren. Abbildung 11.3 gibt eine Übersicht über den beschriebenen Demonstrator.

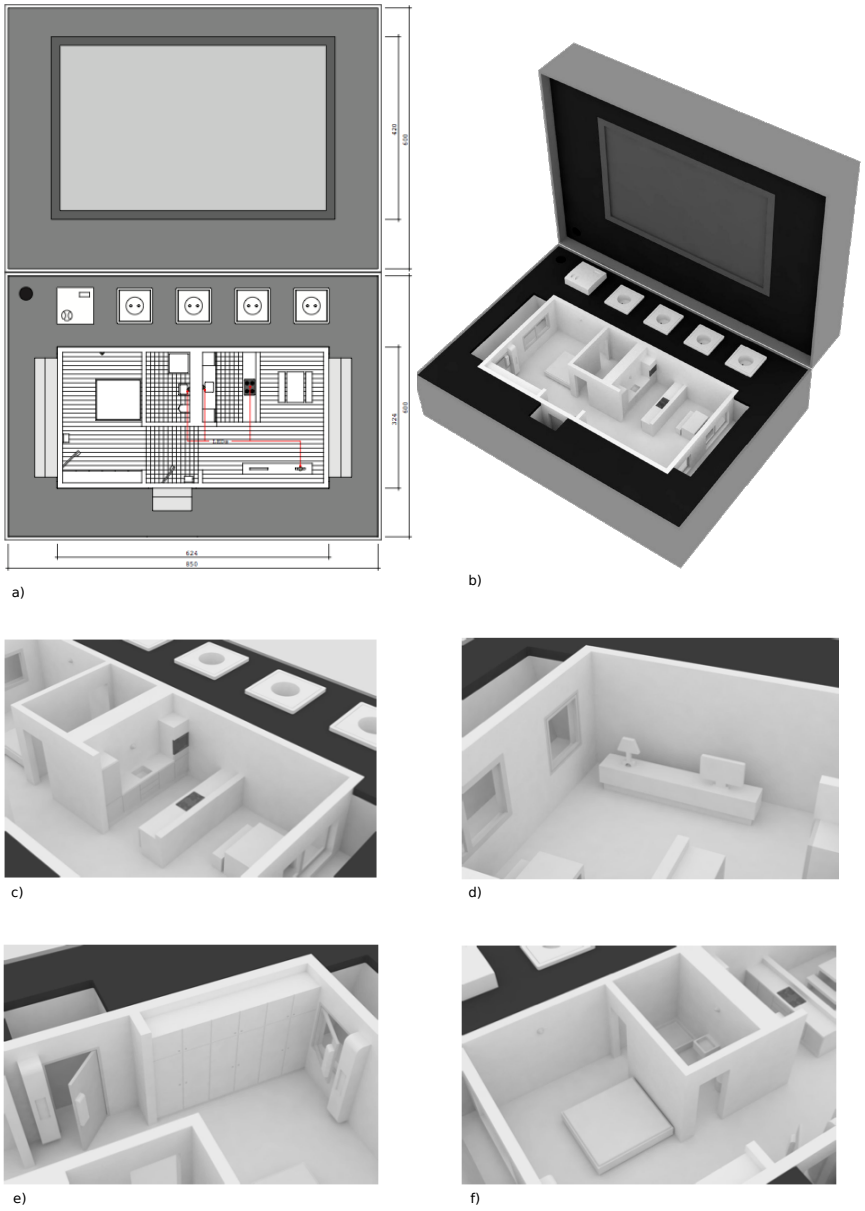


Abbildung 11.3: Übersicht über die SmartCase Simulationsumgebung (a). Der Koffer enthält ein maßstabsgetreues Modell einer realen Wohnung (b) bestehend aus einer Küche (c), einem Wohnzimmer (d), einem Schlafzimmer (e) und einem Badezimmer (f).

Synchronisierung von URC-Implementierungen

Aufbauend auf dem *Dual Reality* Konzept stellen Stahl et al. (2011) eine Entwicklungsmethodik vor, die einen *Synchronized Reality* Ansatz zeigt. Der be-

schriebene Ansatz betrachtet nicht nur die gegenseitige Abbildung einer realen Umgebung innerhalb eines virtuellen Modells, sondern adressiert hierbei auch die Synchronisierung von mehreren, realen *Instrumentierten Umgebungen*. Tsujita et al. (2008) haben beispielsweise gezeigt, dass die funktionale Verknüpfung von alltäglichen Objekten innerhalb der jeweiligen Wohnungen von entfernt lebenden Paaren ihr Zusammengehörigkeitsgefühl und ihre Verbundenheit steigern kann. Als technologische Grundlage wurde eine Erweiterung der vorgestellten UCH-Architektur vorgenommen (Frey et al., 2011b). Hierzu wurde, basierend auf einer socketübergreifenden Architekturebene, ein Plugin zur Synchronisation von *Targets* entwickelt. Dieses Plugin ist dazu in der Lage, die abstrakten UIS-Beschreibungen der *Targets* miteinander zu verknüpfen und entsprechende Regeln zur Synchronisation zu spezifizieren. Hierbei spielt es keine Rolle, ob es sich um *Targets* innerhalb einer realen oder einer virtuellen Umgebung handelt. Abbildung 11.5 gibt eine Übersicht über die beschriebene Architekturerweiterung.

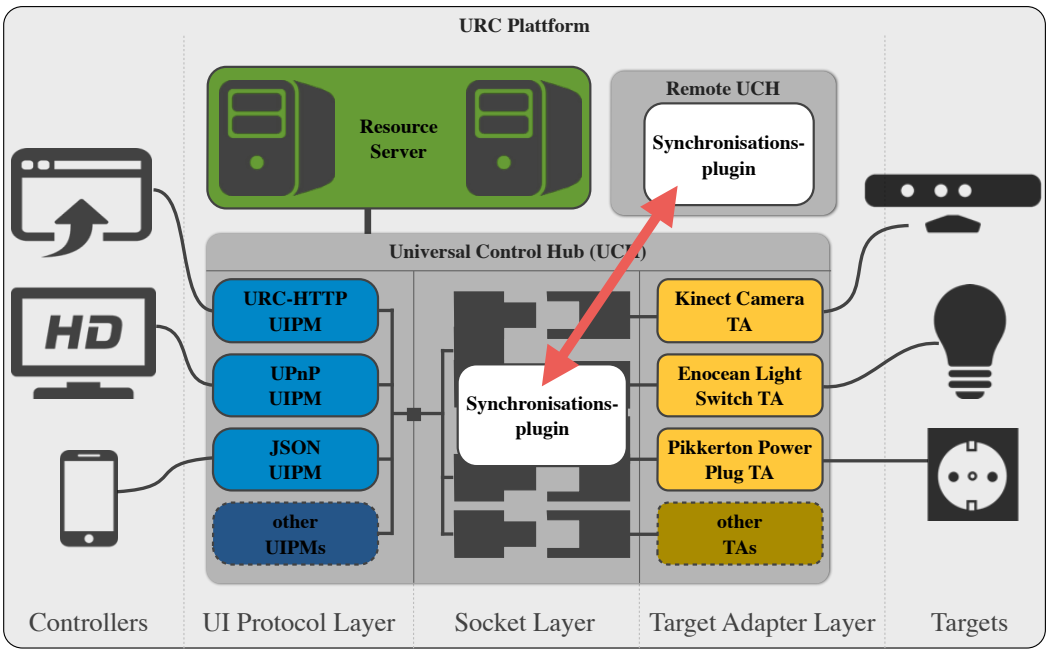


Abbildung 11.4: Überblick über die Erweiterung der UCH-Architektur durch ein Plugin zur Synchronisation virtueller und realer Umgebungen.

Als integrierter Demonstrator wurde eine bidirektionale Synchronisierung der in Kapitel 6.6.1 vorgestellten *Smart Kitchen* und dem in Kapitel 6.6.2 vorgestellten *Bremen Ambient Assisted Living Lab* realisiert. Darüber hinaus wurde ein virtuelles Modell basierend auf dem YAMAMOTO Toolkit (Stahl und Schwartz, 2010) in die Gesamtarchitektur integriert. Abbildung 11.5 zeigt die *Smart Kitchen* zusammen mit einem virtuellen dreidimensionalen YAMAMOTO Modell.

Abbildung 11.5: Demoaufbau in der *Smart Kitchen* mit dem zugehörigen virtuellen, dreidimensionalen YAMAMOTO Modell.

11.3.4 Benutzerzentrierte Werkzeuge zur Erstellung von graphischen Benutzerschnittstellen

Das Entwickeln von intuitiven Benutzerschnittstellen stellt eine nichttriviale Aufgabe dar, besonders im Hinblick auf ältere Menschen oder Menschen mit kognitiven Einschränkungen. Nicht selten gerät der Endbenutzer aus dem Fokus und es wird an den konkreten Bedürfnissen vorbei entwickelt. Gestützt auf den Erfahrungen von i2home wurden mit Hilfe der URC Infrastruktur Werkzeuge und Anwendungen entwickelt, um den Endnutzer möglichst früh an dem Entwicklungsprozess einer graphischen Benutzerschnittstelle teilhaben zu lassen (Schuler und Namioka, 1993).

Tool zur Erstellung socket-basierter Anwendungen

Mit TESA[5] wurde in (Gard, 2010; Frey et al., 2010a) ein Konzept zur benutzerzentrierten Erstellung von auf *User Interface Sockets* basierenden, interaktiven Anwendungen realisiert. Die Grundidee hierbei ist das Bereitstellen eines leicht verständlichen graphischen Editors, mit dem Entwickler und Endnutzer in die Lage versetzt werden, möglichst schnell funktionale Benutzerschnittstellen auf Basis der URC-Technologie zu erstellen. Hierzu liest TESA im ersten Schritt die verfügbaren UIS-Elemente einer laufenden UCH-Installation aus. Im zweiten Schritt können die entsprechenden Elemente mittels einer vorgefertigten Bibliothek mit graphischen Benutzerschnittstellenobjekten verknüpft werden.

[5] *Tool zur Erstellung Socketbasierter Anwendungen*

Das Resultat ist letztlich eine voll funktionale und direkt einsetzbare Interaktionsmöglichkeit mit der zugrunde liegenden *Instrumentierten Umgebung*. Abbildung 11.6 gibt eine Übersicht über die Bedienoberfläche von TESA.

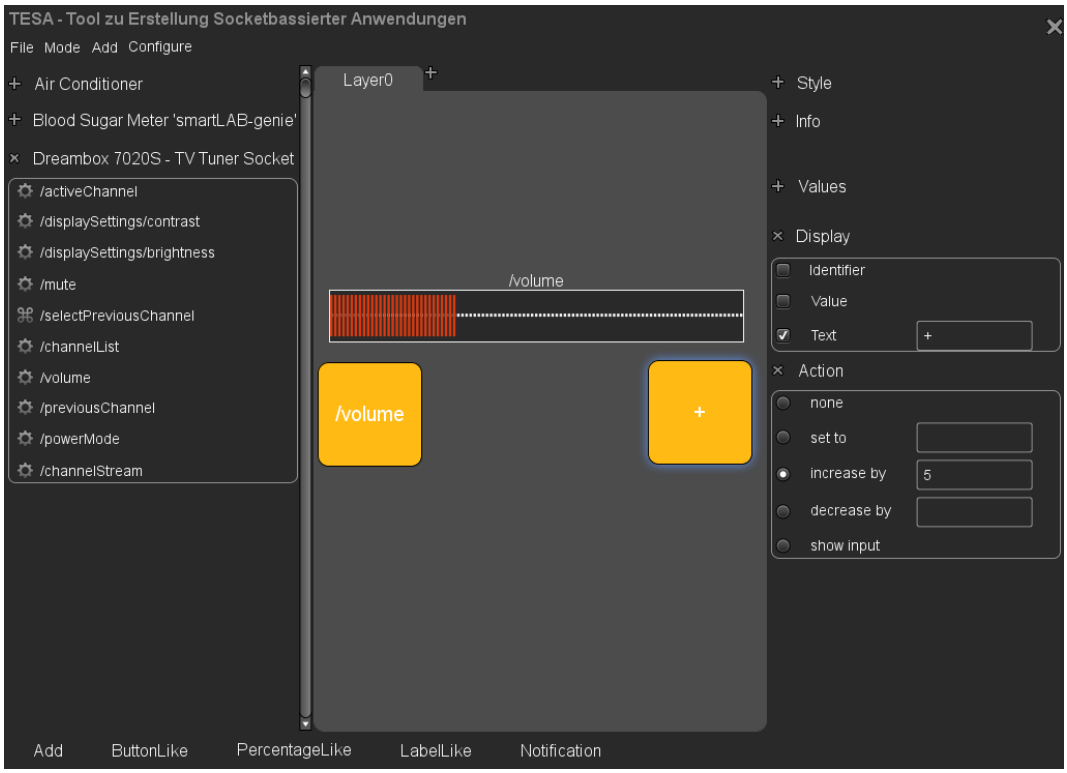


Abbildung 11.6: Überblick über die graphische Bedienoberfläche von TESA zur Erstellung von socketbasierten Anwendungen auf der Basis des URC Standards.

SketchyMate

Paper Prototyping (Snyder, 2003) ist eine verbreitete Methode zur benutzerzentrierten Gestaltung von graphischen Benutzerschnittstellen. Sie ist deswegen so beliebt, weil sie ohne komplexe Hilfsmittel auskommt und von jedermann grundsätzlich direkt verstanden werden kann. Während des i2home Projekts hat sich jedoch herausgestellt, dass viele Benutzer Schwierigkeiten hatten, sich die Funktionalitäten der auf Papier gezeichneten Schnittstellen vorstellen zu können, insbesondere die aus der Interaktion resultierenden Effekte innerhalb der Umgebung. Aufbauend auf den Ideen und Konzepten von TESA wurde in der im Rahmen des *Kompetenzzentrums für Ambient Assisted Living* betreuten Masterarbeit mit SketchyMate (Hörz, 2012) ebenfalls das Ziel verfolgt, funktionale graphische Prototypen erstellen zu können. Der entscheidender Unterschied zu TESA ist jedoch, dass hierbei die Vorteile des *Paper Prototyping* erhalten bleiben sollten. Die Architektur von SketchyMate ähnelt der von TESA, mit dem Unterschied, dass zur Erstellung der prototypischen Benutzerschnittstelle kein graphischer Editor verwendet wird, sondern diese

mit Hilfe eines digitalen Stiftes[6] entworfen wird. Zur Verknüpfung mit den entsprechenden *User Interface Sockets* Elementen kommt ein einfaches Dialogsystem zum Einsatz, wodurch die auf Papier gezeichneten Elemente entsprechend mit Funktionalitäten versehen werden können. Abbildung 11.7 zeigt die grundlegende Idee von SketchyMate, aus gezeichneten Entwürfen funktionale Prototypen zu erzeugen.

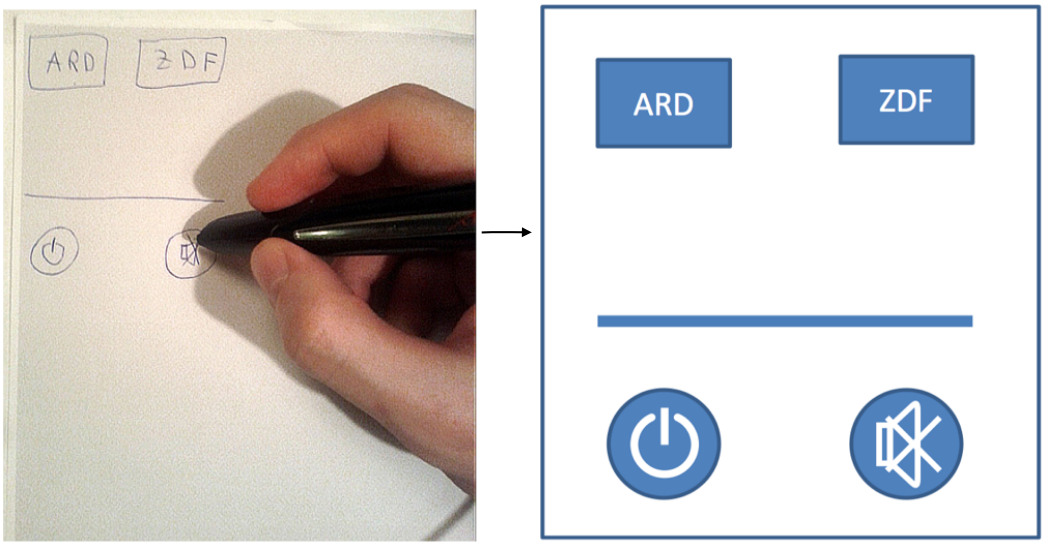


Abbildung 11.7: Paper Prototyping zur Erstellung von funktionalen Benutzerschnittstellen auf der Basis des URC Standards.

11.4 Beispielanwendung eines *Smart Services* zum dezentralen Energiehandel

In Peer Energy Cloud wurde ein *Multiagentensystem* entwickelt, das einen Energiehandel von Privathaushalten auf einen öffentlichen Bürgermarktplatz ermöglicht (Frey et al., 2012). Der dezentrale und autonome Energiehandel ist hierbei als neuartiger digitaler Mehrwertdienst zu verstehen und lässt sich innerhalb des in dieser Arbeit vorgestellte *Smart Service* Konzeptes als *Assistance Service* kategorisieren. Abbildung 11.8 gibt einen Überblick über die zugrunde liegende Architektur des Gesamtsystems.

Die Grundidee hierbei ist, dass jedem beteiligten Haushalt ein dedizierter *Smart Home Agent* zugeordnet wird, welcher wiederum aus drei verschiedenen Funktionalitäten beziehungsweise Verhaltensweisen (*Behaviour*) besteht:

- **Energieprognose:** Die *Forecasting Behaviour* ist für die Vorhersage des zukünftigen Energieverbrauchs, beziehungsweise, im Falle des Vorhandenseins einer Photovoltaikanlage, der zukünftigen Energieerzeugung

[6] http://www.anoto.com [Letzter Zugriff: 10.01.2015]

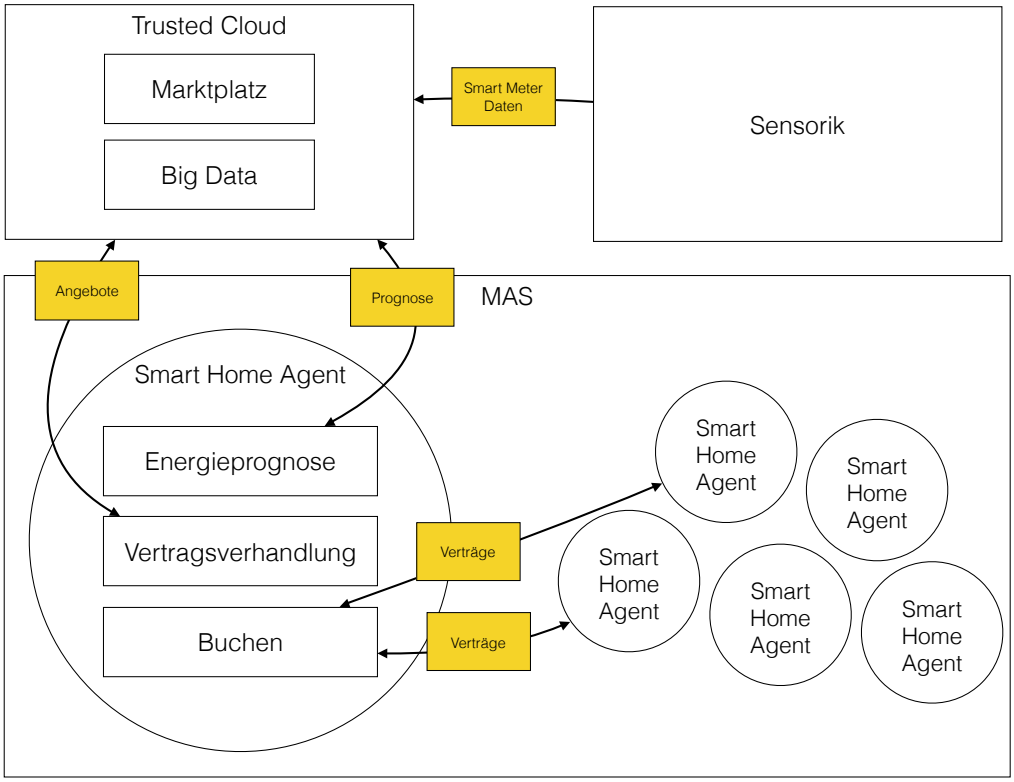


Abbildung 11.8: Übersicht über die Architektur des PEC Multiagentensystems zum Handel von Energieüberschüssen von Privathaushalten.

verantwortlich. Die Berechnung erfolgt durch die Anwendung maschineller Lernverfahren auf der Basis historischer Energiedaten der jeweiligen Haushalte.

- **Vertragsverhandlung:** Aufbauend auf der prognostizierten Energiedifferenz können entsprechende Handelsstrategien angewendet werden, das heißt die *Smart Home Agents* versuchen die Energiebilanz des zugeordneten Privathaushalts für den nächsten Tag durch einen Handel mit anderen *Agenten* zu optimieren. Jeder *Agent* verfolgt hierbei eine eigene Optimierungsstrategie, welche direkt oder indirekt durch den Benutzer konfiguriert werden kann. Die *Trading Behaviour* kommuniziert ausschließlich mit dem Marktplatz und stellt Angebote ein oder trifft Vereinbarungen auf existierende Angebote.

- **Buchen:** Im Gegensatz zur *Trading Behaviour* basiert die *Booking Behaviour* nicht auf den Vorhersagedaten, sondern auf den tat-sächlichen Energieverbrauchsdaten der letzten fünfzehn Minuten. Hierzu versucht die *Booking Behaviour*, aufbauend auf den am Vortag getroffenen Vereinbarungen, rückwirkend seinen Energieverbrauch zu decken. Hierzu kommuniziert der *Smart Home Agent* durch den Einsatz von ACL-Nachrichten direkt mit anderen Agenten.

Das *Multiagentensystem* wurde in JADE realisiert und läuft innerhalb einer

sogenannten *Trusted Cloud* auf den Servern des Energieanbieters. Hierdurch können insbesondere Aspekte wie Datenschutz und Datensicherheit des Gesamtsystems gewährleistet werden.

11.5 Demonstratoren und Beispielanwendungen auf der Basis von *ASaP*

11.5.1 Berechnung und Visualisierung dreidimensionaler Aktivitätszonen

Kontextabhängige Aktivitätszonen (Koile et al., 2003) beschreiben räumliche Bereiche innerhalb einer *Intelligenten Umgebung*, in denen vergleichbare Benutzeraktivitäten auftreten, welche durch Beobachtung des menschlichen Verhaltens erlernt werden können. Aktivitätszonen können als Grundlage für die Ableitung von detaillierten, dreidimensionalen räumlichen Modellen verwendet werden oder als Kontextinformation für Heimautomationsszenarien dienen. So kann zum Beispiel die Beleuchtung in einem Raum bedarfsgerecht gesteuert werden, Audiosignale mit Hilfe von Beamforming gezielt erkannt werden oder Bildschirmanzeigen in Abhängigkeit zu der jeweiligen Aktivitätszone dynamisch anpassen.

Auf dem in dieser Arbeit vorgestellten Plattformkonzept wurde ein *Assistance Service* zur Berechnung und zur Visualisierung dreidimensionaler Aktivitätszonen implementiert (Frey et al., 2014; Neurohr, 2013). Die interne Verarbeitung des beschriebenen *Assistance Service* ist als dreistufiger Prozess strukturiert:

- Im ersten Schritt werden alle für die Erkennung relevanten Statusänderungen der Umgebung erkannt und analysiert, welche innerhalb des Kommunikationsmechanismus mit einem Zeitstempel versehen und durch einen dafür verwantwortlichen *Persistance Service* gespeichert werden.
- Im zweiten Schritt werden die erkannten Ereignisse gruppiert und dreidimensionale Hüllkörper berechnet.
- Im dritten und letzten Schritt werden die Aktivitätszonen schließlich durch eine graphische Oberfläche visualisiert.

Die Berechnung verfolgt hierbei einen inkrementellen Adaptionsprozess, das bedeutet, aktuelle Ereignisse innerhalb der Umgebung werden ständig berücksichtigt. Dies führt folglich dazu, dass die Relevanz bestimmter Ereignisse entweder erhöht oder verringert werden muss. Hierzu wurden zwei gegenläufige Mechanismen implementiert. Auf der einen Seite wird die Relevanz eines Ereignisses verringert, je älter das Ereignis ist. Auf der anderen Seite müssen neu auftretende Ereignisse wiederum die Relevanz von räumlich benachbarten und gleichartigen Ereignissen erhöhen. Hierzu wurde ein *Spreading Activation* (Crestani, 1997) Algorithmus implementiert und entsprechend integriert.

Abbildung 11.9 gibt eine Übersicht über die Bedienoberfläche und den darin enthaltenen Konfigurationsmöglichkeiten.

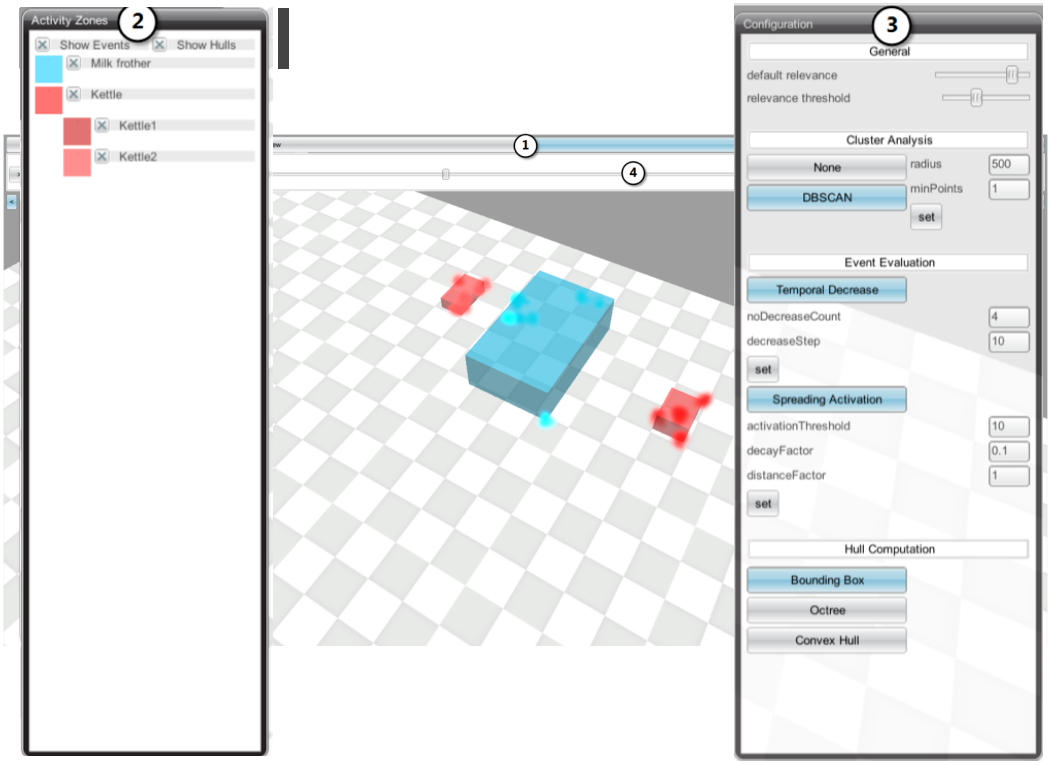


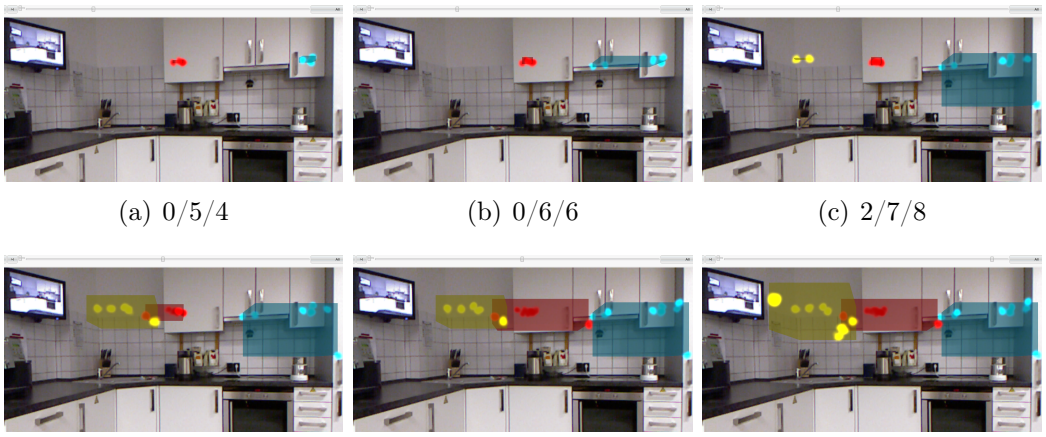
Abbildung 11.9: Ausschnitt der graphischen Bedienoberfläche zur Visualisierung dreidimensionaler Aktivitätszonen.

Abbildung 11.10 zeigt die Entwicklung von drei unterschiedlichen Aktivitätszonen innerhalb der *Smart Kitchen* in Abhängigkeit von der Zeit und den bisher aufgetretenen Ereignissen.

11.5.2 Soundbasierte Geräte- und Aktivitätserkennung

Eine Schwierigkeit bei der Entwicklung von Assistenzsystemen in *Smart Homes* ist die korrekte Erfassung des Zustands der Umgebung basierend auf einheitlichen Sensordaten. Dies beinhaltet nicht nur den Zustand der darin befindlichen Objekte und Geräte, sondern auch die Aktivitäten des jeweiligen Benutzers. *Smart Homes* unterscheiden sich in der Regeln in der Art und Weise ihrer Instrumentierung. Darüber hinaus existieren Benutzeraktivitäten, welche beispielsweise nur schwer durch spezifische Sensoren erkannt werden können. Durch die Verbreitung von Microsoft Kinect[7] oder Amazon Echo[8] sind zukünftig auch visuelle beziehungsweise auditive Verfahren zur Erkennung der Benutzeraktivität denkbar. So können soundbasierte Erkennungsverfahren eingesetzt werden, um Anomalien in der häuslichen Pflege und Betreuung von älteren

[7] http://www.microsoft.com/en-us/kinectforwindows/ [Letzter Zugriff: 12.01.2015]
[8] http://www.amazon.com/echo [Letzter Zugriff: 12.01.2015]



(a) 0/5/4 (b) 0/6/6 (c) 2/7/8

(d) 5/10/8 (e) 5/12/10 (f) 12/12/10

Abbildung 11.10: Entwicklung von dreidimensionalen Aktivitätszonen innerhalb der *Smart Kitchen*. Die Zahlen symbolisieren die entsprechende Anzahl der erkannten Statusveränderungen für das jeweilige Gerät (Fernseher/Wasserkocher/Milchaufschäumer).

Menschen zu erfassen, indem Abweichungen von der typischen Tagesroutine, wie zum Beispiel das Fehlen des abendlichen Fernsehgeräusches, erkannt werden. Im Folgenden wird die Entwicklung eines soundbasierten *Assistance Service* beschrieben, welcher auf der Basis eines einzelnen Standardmikrophons dazu in der Lage ist, die zugrunde liegende Nutzung eines Gerätes zu erkennen (Dimitrov et al., 2014). Abbildung 11.11 zeigt die graphische Bedienoberfläche der beschriebenen Anwendung.

Hierzu wurde im ersten Schritt eine flexible Menge an für den Anwendungsfall charakteristischen Features extrahiert. Durch den Einsatz von maschinellen Lernverfahren konnten einzelnen Geräten spezifische akustische Fingerprints zugeordnet und in einem entsprechenden Korpus gespeichert werden. Im letzten Schritt wurde eine Live-Erkennung der in der Umgebung auftretenden Geräusche implementiert. Durch eine abschließende Evaluation konnte die Anwendbarkeit, die Performanz und die Erkennungsgenauigkeit des Ansatzes bewertet werden. Eine ausführliche und detaillierte Beschreibung der verwendeten Techniken ist in (Dimitrov, 2014) zu finden.

11.6 Zusammenfassung und Fazit

In diesem Kapitel wurden Anwendungen und Demonstratoren aus unterschiedlichen Forschungsprojekten vorgestellt, die mit Hilfe der im Rahmen dieser Arbeit entwickelten Plattformkonzepte entworfen und implementiert werden konnten. Die Beispiele decken ein breites Spektrum von Einsatzmöglichkeiten der in Kapitel 6 beschriebenen UCH-Plattform und der in Kapitel 7, 8 und 9 beschriebenen *ASaP*-Plattform ab. Die beschriebenen Beispiele wurden auf nationalen und internationalen Konferenzen sowie auf DFKI-Veranstaltungen erfolgreich demonstriert. Die Anwendungen belegen die praktische Relevanz

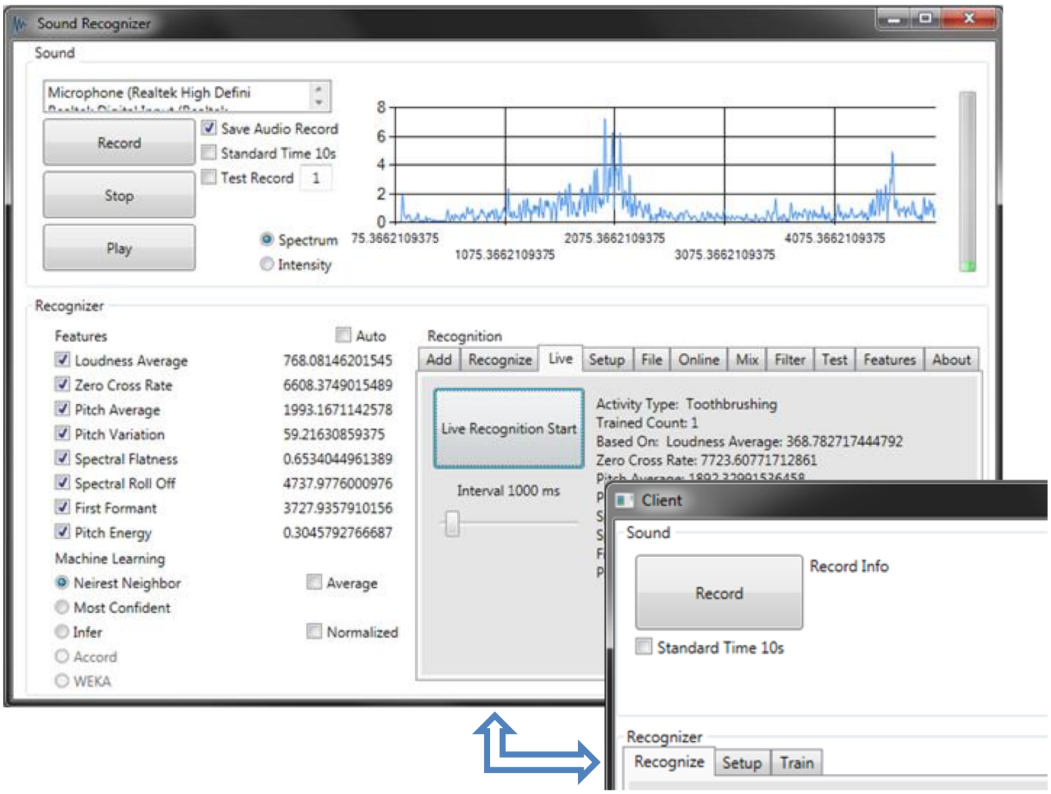


Abbildung 11.11: Übersicht über die graphische Bedienoberfläche zur soundbasierten Erkennung von Geräten und Benutzeraktivitäten.

der Arbeit wie auch eine flexible Einsatzmöglichkeit in unterschiedlichen Anwendungsszenarien. Das folgende und letzte Kapitel fasst abschließend die wissenschaftlichen und praktischen Beiträge der vorliegenden Arbeit zusammen, gibt einen Überblick über die im Rahmen dieser Arbeit entstandenen, wissenschaftlichen Veröffentlichungen und schließt mit einem Ausblick auf mögliche zukünftige Arbeiten, welche die Nutzbarkeit und Effektivität der erarbeiteten Konzepte weiter steigern können.

Teil IV

Diskussion

Objektivität: Alles hat zwei Seiten. Aber erst wenn man erkennt, dass es drei sind, erfasst man die Sache.

Heimito von Doderer

12 Zusammenfassung und Ausblick

12.1 Zusammenfassung

Das Ziel der vorliegenden Arbeit war die Konzeption, der Entwurf und die Realisierung einer Plattform und der dazugehörigen durchgängigen Werkzeugkette zur Integration digitaler Mehrwertdienste in *Intelligenten Umgebungen*. Die im Rahmen der Arbeit erarbeiteten theoretischen Konzepte sowie die entwickelten Softwaremodule stellen wichtige Bausteine zur Einführung und Weiterentwicklung der in Kapitel 3 vorgestellten *Smart Service Welt* dar.

Auf der Basis des in Kapitel 6 eingeführten Industriestandards *Universal Remote Console* (URC) wurde zunächst eine Implementierung des *Universal Control Hubs* (UCH) als zentrale *Smart Home Middleware* umgesetzt. Der Hauptfokus von UCH liegt auf der Bereitstellung von personalisierten Benutzerschnittstellen zur Überwachung und zur Kontrolle eines *Smart Homes*. Aufbauend auf den entwickelten Konzepten wurde im nächsten Schritt die Integration von Mehrwertdiensten in Form von sogenannten *Smart Services* adressiert. Hierzu wurde in Kapitel 7 ein theoretisches Plattformkonzept zur Integration von *Smart Services* in *Intelligenten Umgebungen* erarbeitet. Zur dienstübergreifenden Kommunikation wurde sowohl eine leicht erweiterbare semantische Modellierung der *Intelligenten Umgebung* und der darin befindlichen Objekte und Beziehungen als auch ein auf der *Agent Communication Langauge* (ACL) und auf kommunikativen Akten aufbauender Kommunikationsmechanismus bereitgestellt. Mit der in Kapitel 9 beschriebenen *ASaP-Middleware* wurde abgeleitet von dem vorgestellten Konzept eine dienstübergreifende *Smart Service Plattform* auf der Basis eines verteilten *Multiagentensystems* entworfen und realisiert. Darüber hinaus wurde im Rahmen der Arbeit ein prototypischer Marktplatz für den Endkunden (siehe Kapitel 9.6) als auch eine durchgängige Werkzeugkette für Unternehmen und Entwickler von *Smart Services* (siehe Kapitel 10) zur Verfügung gestellt. Die wissenschaftliche Relevanz des vorgestellten Ansatzes sowie einzelner Teile zeigt sich unter anderem an der Akzeptanz referierter Beiträge auf nationalen und internationalen Konferenzen und Fachtagungen. Durch die Entwicklung von Prototypen und Demonstratoren konnte zudem die praktische Nutzbarkeit des Ansatzes sichergestellt werden. Abbildung 12.1 gibt einen Überblick über die einzelnen, im Rahmen der Arbeit entwickelten Bausteine zur Einführung und zur Weiterentwicklung der *Smart Service Welt*.

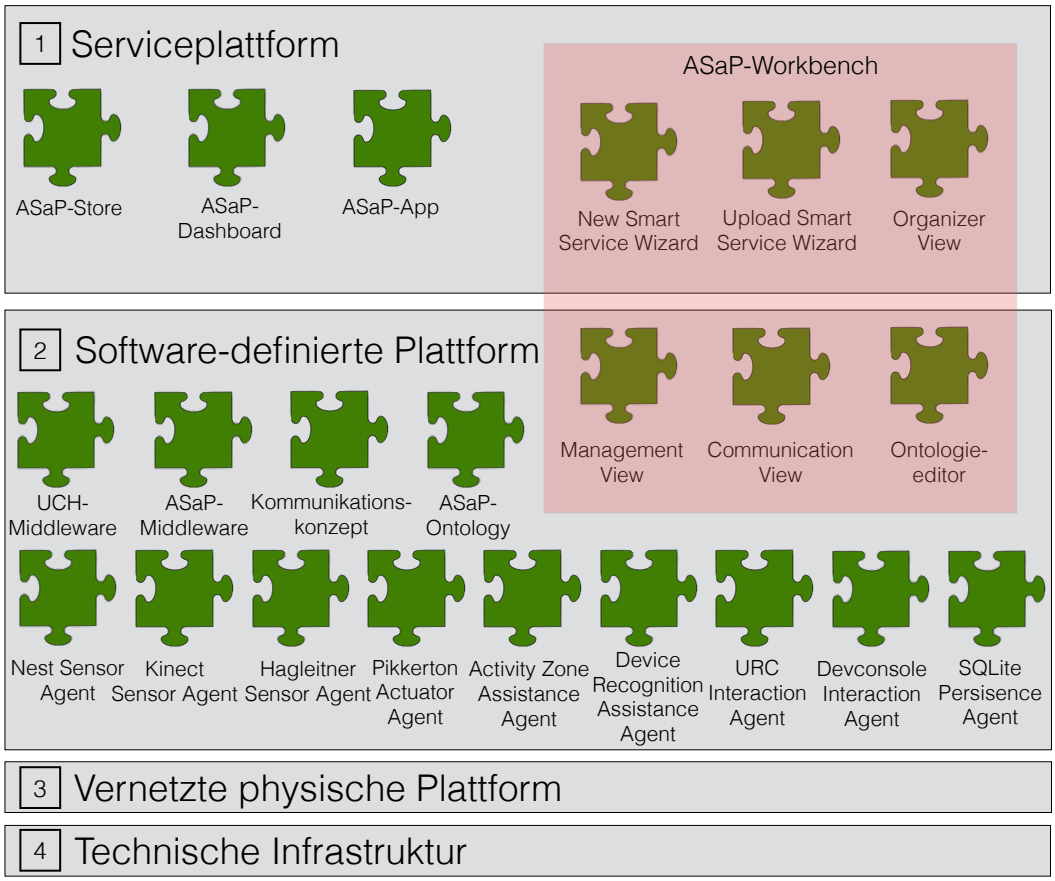


Abbildung 12.1: Übersicht über die im Rahmen der Arbeit entwickelten Softwarebausteine zur *Smart Service Welt*.

In Abschnitt 12.2 wird eine Übersicht über die wissenschaftlichen und technischen Beiträge und die relevanten Ergebnisse der Arbeit gegeben. Abschnitt 12.3 zeigt mögliche, zukünftige Weiterentwicklungen und Forschungsrichtungen auf.

12.2 Beiträge und Ergebnisse

Die im Rahmen der Arbeit entwickelten Beiträge gliedern sich in zwei untergeordnete Teilbereiche auf. Abschnitt 12.2.1 beschreibt die wissenschaftlichen und technischen Ergebnisse. Dies beinhaltet nicht nur die erarbeiteten, theoretischen Konzepte, sondern auch die realisierten, praktischen Softwarelösungen. Abschnitt 12.2.2 fasst schließlich die im Rahmen der Arbeit akzeptierten wissenschaftlichen Publikationen zusammen.

12.2.1 Wissenschaftliche und technische Ergebnisse

Die Aufzählung der wissenschaftlichen und technischen Beiträge der Arbeit orientiert sich an den in Kapitel 1.2.2 aufgeführten wissenschaftlichen Fragestellungen.

Die ersten drei Fragestellungen beziehen sich auf die Konzeption, den Entwurf und die Realisierung einer Integrationsplattform für:

1. ***Sensoren* und *Aktuatoren*:** *Wie können unterschiedlichste Sensoren und Aktuatoren in eine Intelligente Umgebung integriert werden?*

   Laut einer Studie von BITKOM und Deloitte[1] sind Verbraucher grundsätzlich zurückhaltender bei der Entscheidung für geschlossene Plattformen. Um dem entgegenzuwirken, wurden in der Arbeit zwei aufeinander aufbauende offene Architekturkonzepte zur Integration von *Sensoren* und *Aktuatoren* in eine *Intelligente Umgebung* vorgestellt. In beiden Fällen lag der Fokus der Entwicklung auf einem modular aufgebauten Komponentensystem, welche zur Laufzeit dynamisch um weitere Funktionalitäten, wie beispielsweise die Integration neuer *Sensoren* und *Aktuatoren* von unterschiedlichen Herstellern, erweitert werden kann. Während die ***UCH-Middleware*** primär zum Ziel hat, dem Endnutzer einen personalisierten Zugriff auf seine Umgebung anzubieten (siehe Kapitel 6), lag der Fokus bei der ***ASaP-Middleware*** auf der dienstübergreifenden Integration und der Verkettung digitaler Mehrwertdienste in Form von *Smart Services* (siehe Kapitel 7). In beiden Ansätzen wurden auf OSGi basierende, modulare Softwarekomponenten zur Integration von *Sensoren* und *Aktuatoren* entwickelt. Wohingegen bei UCH unterschiedliche *Target Adapters* in Kombination mit standardisierten *User Interface Socket* Beschreibungen entwickelt wurden, lag der Fokus bei *ASaP* auf dem Einsatz entsprechender *Sensor Services* und *Actuator Services* und deren Zusammenspiel auf Basis einer einheitlichen, semantischen Modellierung. Die einzelnen *Smart Services* wurden hierzu durch entsprechende *Smart Service Agenten* innerhalb eines *Multiagentensystems* repräsentiert. Wie in Kapitel 9 beschrieben, wurden zur Integration von *Sensoren* und *Aktuatoren* die folgenden *Smart Service Agenten* im Rahmen der Arbeit realisiert und semantisch modelliert:

   - ***Nest Sensor Agent*** zur Integration von auf Google Nest basierenden Rauchmeldern
   - ***Kinect Sensor Agent*** zur Integration der Microsoft Kinect Kamera
   - ***Pikkerton Actuator Agent*** zur Integration von auf ZigBee basierenden *Smart Energy Plugs*

2. ***Smart Services*:** *Wie können Smart Services in eine Intelligente Umgebung integriert und eine dienstübergreifende Interoperabilität gewährleistet werden?*

[1] http://www2.deloitte.com/content/dam/Deloitte/de/Documents/technology-media-telecommunications/TMT-Deloitte-Bitkom-Marktaussichten-SmartHome.pdf [Letzter Zugriff: 07.01.2015]

Eine wichtige Eigenschaft einer *Intelligenten Umgebung* ist die Möglichkeit zur Erweiterung und somit zur schrittweisen Anpassung an den Benutzer. Die Umgebung muss hierbei dem Benutzer stets einen tatsächlichen und auf den konkreten Anwendungsfall bezogenen Mehrwert anbieten können. In der Arbeit wurde ein Konzept zur Klassifizierung der verfügbaren Mehrwertdienste innerhalb einer Umgebung vorgestellt. Hierzu wurden in Kapitel 7.5 fünf unterschiedliche Typen von *Smart Services* definiert. Neben den im vorigen Abschnitt aufgeführten *Sensor Services* und *Actuator Services* zur Anbindung der jeweiligen *Sensoren* und *Aktuatoren* werden zudem *Persistence Services* zur Speicherung der Umgebungs- und Plattforminformationen, *Interaction Services* zur Kommunikation mit dem Benutzer und *Assistance Services* zur Realisierung intelligenter Assistenzdienste innerhalb der Umgebung beschrieben. Des Weiteren wurde ein plattforminternes *Kommunikationskonzept* basierend auf der *Agent Communication Language* (ACL) und kommunikativen Akten entworfen. Zudem wurden einfache Möglichkeiten zur Kooperation und zur Verkettung einzelner *Smart Services* realisiert. Es wurde darüber hinaus gezeigt, wie sich die vorgestellte Plattform in die Standardarchitektur von OpenURC und auch in die vorgeschlagene Referenzarchitektur der *Smart Service Welt* einordnen und abbilden lässt. Die Realisierung des vorgestellten Plattformkonzepts zur Integration von *Smart Services* in *Intelligenten Umgebungen* hat gezeigt, wie JADE als zugrunde liegende *Agentenplattform* genutzt werden kann, um die erarbeiteten theoretischen Konzepte praktisch umzusetzen. Weiterhin wurde gezeigt, wie die entwickelten *Smart Services* in modulare Komponenten zergliedert und somit wiederum als *OSGi Bundles* verteilt und dynamisch zur Laufzeit in ein bestehendes System integriert werden können. Im Rahmen der Arbeit wurde zudem zu jedem *Smart Service* Typ mindestens eine konkrete Ausprägung realisiert, womit die spezifizierten Kommunikationsprotokolle wie auch der realisierte Kollaborationsmechanismus als Verkettung von *Smart Services* demonstriert werden konnte. Neben den im vorigen Abschnitt bereits erwähnten *Sensor Services* und *Actuator Services* wurden die folgenden *Assistance Services* und *Interaction Services* durch entsprechende *Softwareagenten* implementiert:

- ***Activity Zone Assistance Agent*** zur Erkennung und Visualisierung dreidimensionaler Aktivitätszonen
- ***Device Recognition Assistance Agent*** zur soundbasierten Geräte- und Aktivitätserkennung
- ***Devconsole Interaction Agent*** zur Integration der entsprechenden *ASaP-Workbench*
- ***URC Interaction Agent*** zur Transformation der plattforminternen semantischen Modellierung von *Smart Services* in URC kompatible *User Interface Socket* Beschreibungen

3. **Personalisierte Benutzerschnittstellen:** *Wie kann eine Vielzahl von*

Benutzerschnittstellen mit minimalem Aufwand in eine Intelligente Umgebung integriert werden?

Personalisierte Benutzerschnittstellen haben im Vergleich zu *one size fits all* Schnittstellen einen weitaus höheren Entwicklungsaufwand. Es hat sich aber gezeigt, dass gerade für Menschen mit Einschränkungen oder besonderen Bedürfnissen solche Schnittstellen unumgänglich sind. Aber auch für die breite Masse spielen zunehmend individualisierte und technisch ansprechende Designs eine wichtige Rolle. Um den Entwicklungsaufwand für personalisierte Benutzerschnittstellen möglichst gering zu halten, werden standardisierte Schnittstellenbeschreibungen eingesetzt. Ausgehend von dem Industriestandard *Universal Remote Console* wurde mit der ***UCH-Middleware*** eine auf Java und OSGi basierende Referenzimplementierung des *Universal Control Hubs* realisiert. Die modular aufgebaute Plattform erlaubt eine dynamische Integration einer Vielzahl von verfügbaren *Sensoren* und *Aktuatoren* in Form von *OSGi Bundles*. Jedes *Target Bundle* beinhaltet hierbei sowohl den Quellcode zur Adressierung der konkreten Hardware als auch eine abstrakte Funktionsbeschreibung in Form eines *User Interface Sockets*. Die entwickelte ***UCH-Middleware*** wird aktuell in mehreren Projekten am DFKI verwendet sowie im Rahmen des Projektes SUCH derzeit entsprechend standardisierter IT-Sicherheitsrichtlinien neu implementiert. Im Rahmen der in Kapitel 9 beschriebenen *ASaP-Middleware* wurde darüber hinaus auf einer semantischen Modellierung der zugrunde liegenden plattforminternen Kommunikation aufgebaut, welche andererseits durch entsprechende Modelltransformationen in eine URC konforme abstrakte Schnittstellenbeschreibung überführt werden konnte. Durch den Einsatz des ***URC Interaction Agents*** wird somit die direkte Anbindung einer Vielzahl von bereits existierenden und zu URC kompatiblen Benutzerschnittstellen an die ***ASaP-Middleware*** ermöglicht. Darauf aufbauend wurden im Rahmen der Arbeit weitere Anwendungen und Demonstratoren realisiert.

Die letzten drei Fragestellungen betrachten die Konzeption, den Entwurf und die Realisierung einer semantischen Modellierung, eines *Smart Service Marktplatzes* und einer durchgängigen Werkzeugunterstützung innerhalb einer *Intelligenten Umgebungen*.

4. **Semantische Modellierung:** *Wie kann eine Intelligente Umgebung semantisch repräsentiert werden, insbesondere das Zusammenspiel von Sensoren, Aktuatoren, Smart Services und Benutzerschnittstellen?*

   Die semantische Modellierung der Umgebung sowie der darin enthaltenen Objekte, Dienste und Zustände verfolgt im Wesentlichen drei Zielsetzungen. Zunächst dient sie als plattforminternes Kommunikationsprotokoll zur Kollaboration von *Smart Services*. Des Weiteren erleichtert ei-

ne semantische Dienstbeschreibung das Auffinden von geeigneten *Smart Services* innerhalb eines entsprechenden Marktplatzes sowie die Bereitstellung von Entwicklungswerkzeugen, beispielsweise zur automatischen Generierung von Code Templates. Zuletzt ermöglicht die semantische Modellierung der Umgebung eine darauf aufbauende manuelle oder automatische Erstellung von abstrakten oder konkreten Benutzerschnittstellen. Der zentrale Bestandteil der in Kapitel 8 vorgestellten ***ASaP-Ontology*** ist die Repräsentation von *Smart Services*, *Ereignissen* und des *Kontext* innerhalb einer *Intelligenten Umgebung*. Die technologische Grundlage der ***ASaP-Ontology*** bildet das *Eclipse Modeling Framework* (EMF). Die erarbeiteten EMF Modelle stellen die Grundlage der ***ASaP-Workbench*** dar.

5. ***Smart Service Marktplatz*:** *Wie können neuartige digitale Geschäftsmodelle in Form von Smart Services auf Basis eines einheitlichen Marktplatzkonzeptes realisiert und vertrieben werden?*

   Aufbauend auf *ASaP* wurde ein prototypischer webbasierter Marktplatz entwickelt, um sowohl Unternehmen und einzelnen Entwicklern die Möglichkeit zu bieten, ihre *Smart Services* anzubieten als auch den Endkunden ein zentrales Portal zur Verfügung zu stellen, um neue und gezielt auf ihre aktuelle Situation und jeweilige Instrumentierung passende digitale Mehrwertdienste auffinden und installieren zu können. Der vollständig in das *ASaP*-Ökosystem integrierte ***ASaP-Store*** bietet hierzu eine globale Kooperationsplattform, mit der sowohl Dienstanbieter und Entwickler als auch Endkunden dazu in die Lage versetzt werden *Smart Services* zu finden, zu erstellen und miteinander auszutauschen. Die Kernfunktionalität des in Kapitel 9.6 vorgestellten ***ASaP-Stores*** umfasst hierbei die Möglichkeit *Smart Service Bundles* zusammen mit ihrer semantischen Spezifikation zu verwalten und potentiellen Endkunden zur Installation anzubieten. Die technologische Basis für die Umsetzung des ***ASaP-Store*** baut auf einem *Backend-as-a-Service* (Baas) Ansatz auf. In der vorliegenden Arbeit wurde Parse[2] als zugrunde liegendes BaaS verwendet und durch den Einsatz der bereitgestellten REST Schnittstelle in die im nächsten Abschnitt beschriebene durchgängige Werkzeugkette integriert.

6. **Durchgängige Werkzeugunterstützung:** *Wie kann eine durchgängige Werkzeugkette konzipiert werden, um die Entwicklung von Smart Services in Intelligenten Umgebungen zu unterstützen?*

   Zur Unterstützung der *Smart Service* Entwickler wurde eine durchgängige Werkzeugkette implementiert, die den Entwickler entlang des gesamten Entwicklungszyklus unterstützt. Die einzelnen Entwicklungs- und Laufzeitwerkzeuge wurden in einer auf Eclipse basierenden ***ASaP-Workbench*** zusammengefasst und als Open Source bereitgestellt. Die ***ASaP-Workbench*** umfasst hierbei die folgenden Softwarekomponenten:

[2] http://parse.com [Letzter Zugriff: 19.01.2015]

- ***New Smart Service Wizard*** zur Unterstützung der Entwickler bei der Erstellung von neuen *Smart Service* Projekten
- ***Upload Smart Service Wizard*** zur Unterstützung der Entwickler beim Veröffentlichen der realisierten *Smart Services*
- ***Organizer View*** zur Verwaltung, Versionierung und Wirkbetrieb von *Smart Service Bundles*
- ***Management View*** zur Verwaltung und Kontrolle der innerhalb einer ***ASaP-Middleware*** installierten *Smart Service Bundles*
- ***Communication View*** zur Visualisierung der plattforminternen ACL-Nachrichten
- ***Ontologieeditor*** zum Bearbeiten der zugrunde liegenden semantischen Modelle der ***ASaP-Ontology***
- ***ASaP-Dashboard*** zur Installation neuer *Smart Services*, zum Management und zur Verwaltung der laufenden *ASaP-Middleware* und zur Manipulation und Überwachung des Umgebungszustandes durch den Benutzer
- ***ASaP-App*** zur Bereitstellung einer mobilen Schnittstellen zu einer laufenden *ASaP-Middleware* und den verfügbaren *ASaP-Stores*

Auf der Basis der genannten wissenschaftlichen und technischen Ergebnisse wurden prototypische Anwendungen aus den Bereichen *Ambient Assisted Living* und *Smart Energy* realisiert. Die dabei gewonnenen Erkenntnisse haben die Weiterentwicklung der Plattformen und der entsprechenden Werkzeuge maßgeblich beeinflusst, aber auch die praktische, wissenschaftliche und wirtschaftliche Relevanz der vorgestellten Konzepte verdeutlicht.

12.2.2 Wissenschaftliche Veröffentlichungen

Dieser Abschnitt fasst die im Rahmen der Arbeit eingereichten und akzeptierten wissenschaftlichen Publikationen zusammen. Die Aufzählung gliedert sich in nationale sowie internationale Journal-, Konferenz- und Workshopbeiträge.

Journal-Artikel

Teile dieser Arbeit wurden im folgenden Journal veröffentlicht:

- **Journal of Ambient Intelligence and Smart Environments 2011**: Stahl, C., Frey, J., Alexandersson, J., und Brandherm, B. (2011). Synchronized Realities. *Journal of Ambient Intelligence and Smart Environments*, 3(1):13–25

Konferenzen

Teile dieser Arbeit wurden auf folgenden nationalen und internationalen Konferenzen veröffentlicht:

- **3rd International Conference on Health Informatics 2010**:
  Frey, J., Schulz, C. H., Neßelrath, R., Stein, V., und Alexandersson, J. (2010c). Towards Pluggable User Interfaces for People with Cognitive Disabilities. In *3rd International Conference on Health Informatics*, pages 428–431

- **International Conference on Language Resources and Evaluation 2010**:
  Frey, J., Neßelrath, R., Schulz, C. H., und Alexandersson, J. (2010b). SensHome: Towards A Corpus for Everyday Activities in Smart Homes. In *International Conference on Language Resources and Evaluation*, pages 137–139

- **1st International Joint Conference on Ambient Intelligence 2010**:
  Frey, J., Stahl, C., Röfer, T., Krieg-Brückner, B., und Alexandersson, J. (2010d). The DFKI Competence Center for Ambient Assisted Living. In *Proceedings of the First International Joint Conference on Ambient Intelligence (AmI'10)*, pages 310–314. Springer-Verlag

- **4. Deutscher AAL-Kongress 2011**:

  - Schulz, C. H., Zinnikus, I., Kapahnke, P., Frey, J., Neßelrath, R., und Alexandersson, J. (2011). Universally Accessible Interactive Services on TV: A Case Study on the Provisioning of Internet Services to Elderly People. In *Ambient Assisted Living, 4 AAL-Kongress*, Berlin. Springer
  - Neßelrath, R., Lu, C., Schulz, C. H., Frey, J., und Alexandersson, J. (2011b). A Gesture Based System for Context-Sensitive Interaction with Smart Homes. In *Ambient Assisted Living, 4 AAL-Kongress*, Berlin

- **7th International Conference on Intelligent Environments 2011**:

  - **(Best Demo Award)** Frey, J., Bergweiler, S., Alexandersson, J., Gholamsaghaee, E., Reithinger, N., und Stahl, C. (2011a). SmartCase: A Smart Home Environment in a Suitcase. In *Proceedings of the 7th International Conference on Intelligent Environments (IE)*, pages 378–381, Nottingham, UK
  - **(Best Video Award)** Neßelrath, R., Haupert, J., Frey, J., und Brandherm, B. (2011a). Supporting Persons with Special Needs in their Daily Life in a Smart Home. In *Proceedings of the 7th International Conference on Intelligent Environments (IE)*, pages 370–373, Nottingham, UK

- **5. Deutscher AAL-Kongress 2012**:
  - Gauterin, A., Alexandersson, J., Neßelrath, R., Schulz, C. H., und Frey, J. (2012). Accessible Elevator. In *Technik für ein selbstbestimmtes Leben - 5. Deutscher AAL-Kongress 01/24/2012 - 01/25/2012 at Berlin*
  - Gholamsaghaee, E., Reithinger, N., und Frey, J. (2012). Automatic Generation of Intuitive User Interfaces for Controlling and Monitoring of Home Appliances for Mobile Devices. In *Technik für ein selbstbestimmtes Leben - 5. Deutscher AAL-Kongress 01/24/2012 - 01/25/2012 at Berlin*
- **8th International Conference on Intelligent Environments 2012**: Brandherm, B., Baus, J., und Frey, J. (2012). Peer Energy Cloud – Civil Marketplace for Trading Renewable Energies. In *Proceedings of the 8th International Conference on Intelligent Environments (IE)*, pages 375–378, Guanajuato, Mexico. IEEE Computer Society
- **9th International Conference on Intelligent Environments 2013**: Frey, J. (2013). AdAPT - A Dynamic Approach for Activity Prediction and Tracking for Ambient Intelligence. In *9th International Conference on Intelligent Environments (IE)*, pages 254–257, Athens, Greece
- **10th International Conference on Intelligent Environments 2014**:
  - Neurohr, C. (2013). EvA - Event-basierte Erkennung und Visualisierung dreidimensionaler Aktivitätszonen in Instrumentierten Umgebungen . Master's thesis, Saarland University, Dept. of Computer Science
  - Britz, J., Frey, J., und Alexandersson, J. (2014). Bridging the Gap between Smart Home and Agents. In *Proceedings of the 10th International Conference on Intelligent Environments (IE)*, pages 31–38, Shanghai, China
- **European Conference on Ambient Intelligence 2014**: Dimitrov, S., Britz, J., Brandherm, B., und Frey, J. (2014). Analyzing Sounds of Home Environment for Device Recognition. In *Proceedings of the European Conference on Ambient Intelligence*, pages 1–16, Eindhoven, Netherlands

Workshops

Teile dieser Arbeit wurden auf folgenden internationalen Workshops veröffentlicht:

- **Workshop on future standards for Model-Based User Interfaces 2010**:
  Alexandersson, J., Zinnikus, I., Frey, J., Zimmermann, G., Epelde, G.,

Bund, J., und Rosa, B. (2010). Convergence of Internet of Services and Internet of Appliances: Extending the Universal Remote Console. In *Workshop on future standards for Model-Based User Interfaces*, Rome, Italy

- **Workshop on Tool-Support for Mobile and Pervasive Application Development 2010**:
  Frey, J., Gard, T., und Alexandersson, J. (2010a). User-Centred Tool Support: Developing User Interfaces for Persons with Special Needs. In *Workshop on Tool-Support for Mobile and Pervasive Application Development*

- **Workshop on Location Awareness for Mixed and Dual Reality 2011**:
  Frey, J., Neßelrath, R., und Stahl, C. (2011b). An Open Standardized Platform for Dual Reality Applications. In *Proceedings of the Second International Workshop on Location Awareness for Mixed and Dual Reality. Workshop on Location Awareness for Mixed and Dual Reality (LAMDa-11), 2nd, located at International Conference on Intelligent User Interfaces*, pages 1–4, Palo Alto, CA, USA

12.3 Zukünftige Arbeiten

Das Hauptziel der vorliegenden Arbeit war die Konzeption, der Entwurf und die Realisierung einer Plattform und der dazugehörigen durchgängigen Werkzeugkette zur Integration digitaler Mehrwertdienste in *Intelligenten Umgebungen*. Das Gebiet der *Smart Service Welt* stellt hierbei ein sehr breit gefächertes und interdisziplinär aufgestelltes Forschungsgebiet dar. Folglich wurden grundsätzlich viele interessante Forschungsfelder adressiert, von denen im Rahmen der Arbeit aber nur einige ausführlich betrachtet werden konnten. Die folgende Aufzählung gibt einen Überblick über mögliche Weiterentwicklungen und zukünftige Forschungsrichtungen im Kontext der Entwicklung einer Integrationsplattform für *Smart Services* in *Intelligenten Umgebungen*.

- **Wie kann eine einheitliche semantische Beschreibung für *Intelligente Umgebungen* etabliert werden, beziehungsweise wie können unterschiedliche semantische Beschreibungssprachen ineinander überführt werden?**

  Der in Kapitel 8 vorgestellte Ansatz zur semantischen Beschreibung beinhaltet eine definierte und vorgegebene Modellierung der Objekte und Beziehungen innerhalb einer *Intelligenten Umgebung*. Durch die entwickelten Werkzeuge wird eine Weiterentwicklung des zugrunde liegenden Modells zwar erleichtert, ein einheitliches Konzept zur Homogenisierung mit bereits existierenden Ansätzen oder sogenannten *Top-level Ontologies* konnte hingegen im Rahmen dieser Arbeit nicht adressiert werden.

Um ein möglichst hohes Maß an Interoperabilität unter allen Marktteilnehmern zu erreichen, muss einerseits untersucht werden, ob es möglich ist, eine allgemein akzeptierte semantische Modellierung einer *Intelligenten Umgebungen* zu etablieren, oder andererseits geeignete Verfahren entwickelt werden, um unterschiedliche semantische Beschreibung ineinander zu überführen oder aufeinander abzubilden (Kalfoglou und Schorlemmer, 2003).

- **Wie kann eine dienstübergreifende Koordination und Kooperation von *Smart Services* erzielt werden, besonders in Hinblick auf konkurrierende Zielsetzungen einzelner Agenten?**

  Der in Kapitel 7.6 vorgestellte Kooperationsmechanismus basiert auf einer einfachen, dynamischen Verkettung von *Smart Services* entsprechend ihrer jeweiligen *Input* und *Output* Parameter. Das vorgestellte Konzept eignet sich in erster Linie für einfache Mehrwertdienste im privaten *Smart Home* Umfeld mit einer geringen Anzahl von *Smart Services*. Eine Erhöhung sowohl der Komplexität einzelner Mehrwertdienste als auch der Anzahl der insgesamt installierten *Smart Services* impliziert somit auch eine gesteigerte Wahrscheinlichkeit konkurrierender Zielsetzungen einzelner *Smart Service Agenten*. Ein einfaches Beispiel für konkurrierende Zielsetzungen im *Smart Home* Kontext ist die Bereitstellung von Komfortfunktionen für den Benutzer bei gleichzeitiger Minimierung des benötigten Energieverbrauchs. Um solche Szenarien realisieren zu können, muss *ASaP* um Konfliktlösungsstrategien in Form von Kollaborationsmechanismen für *Multiagentensysteme* erweitert werden (Wilsker, 1996).

- **Wie können *Reasoning* Mechanismen eingesetzt werden, um ein semantisches *Matchmaking* von digitalen Mehrwertdiensten zu realisieren, und somit eine optimale Auffindbarkeit von passenden *Smart Services* auf einem Marktplatz zu gewährleisten?**

  In Kapitel 7.10 wurde ein Anwendungsszenario zur Interaktion mit einem *Smart Service Marktplatz* vorgestellt. Über den in Kapitel 9.6 realisierten *ASaP-Store* und mit Hilfe des in Kapitel 10 beschriebenen *ASaP-Dashboards* und der *ASaP-App* kann der Benutzer die auf seine aktuelle Instrumentierung passenden *Smart Services* auffinden. Hierzu werden die semantischen Abhängigkeiten der *Input* und *Output* Beziehungen der verfügbaren *Smart Services* analysiert und diese im Anschluss entsprechend gefiltert. Umgekehrt kann der Marktplatz dem Kunden aufzeigen, welche Voraussetzungen für die Installation einzelner *Smart Services* erfüllt werden müssen. Der vorgestellte Ansatz stellt einen praktikablen aber einfachen Ansatz dar, der sich unter Umständen nicht auf komplexere Dienstbeschreibungen, die zudem unterschiedlich semantisch modelliert sein können, übertragen lässt. Um diese Aufgabenstellung zu lösen, soll der Einsatz von *Reasoning* Mechanismen zur Realisierung eines semantischen *Matchmakings* (Di Noia et al., 2007; Klusch und Kapahnke,

2012) von *Smart Services* untersucht werden, um auf diese Art und Weise eine verbesserte und automatisierte Auffindbarkeit der Mehrwertdienste zu gewährleisten.

A

FIPA ACL-Spezifikation

A.1 FIPA Protokollspezifikation

Parameter	**Beschreibung**
performative	Spezifiziert den zugrunde liegenden kommunikativen Akt
sender	Identifiziert den Sender der Nachricht
receiver	Identifiziert die Empfänger der Nachricht
reply-to	Identifiziert die *Agenten*, welche, anstelle des Senders, mögliche weitere Nachrichten innerhalb der Diskussion empfangen sollen
content	Beschreibt den Inhalt der Nachricht
language	Beschreibt die Sprache, in welcher der Inhalt verfasst wurde und gewährleistet somit eine einheitliche Interpretation des Inhalts
encoding	Spezifiziert die Kodierung des Inhalts
ontology	Definiert die zugrunde liegende Ontologie der Nachrichten und gewährleistet somit eine einheitliche Interpretation des Inhalts
protocol	Definiert ein Interaktionsprotokoll, in dessen Kontext die zugrunde liegenden ACL-Nachricht generiert wurde
conversation-id	Identifiziert die Zugehörigkeit von ACL-Nachrichten zu einer bestimmten Konversation innerhalb des Systems
reply-width	Führt einen beschreibenden Ausdruck ein, mit dem mögliche Antworten der empfangenden *Agenten* gekennzeichnet werden können
in-reply-to	Kennzeichnet die Zugehörigkeit einer Antwort zu einer zuvor definierten Aktion eines anderen Agenten
reply-by	Spezifiziert einen zeitlichen Rahmen, in welchem der Empfänger spätestens eine Antwort erhalten will

Tabelle A.1: Parameter einer FIPA-konformen ACL-Nachricht.

A.2 FIPA Kommunikative Akte

Parameter	Beschreibung
Accept Proposal	Der Sender teilt dem Empfänger mit, dass der Sender ein vorhergehendes *Proposal* unter den angegebenen Bedingungen akzeptieren wird und, dass die Aktion ausgeführt wird, sobald die Vorbedingungen erfüllt sind.
Agree	Der Sender teilt dem Empfänger mit, dass der Sender einem vorhergehenden *Request* unter den angegebenen Bedingungen zustimmt.
Cancel	Der Sender teilt dem Empfänger mit, dass das Ausführen einer Aktion nicht mehr gefordert wird.
Call for Proposal	Initiiert typischerweise eine Konversation, indem es den Empfänger dazu auffordert, *Proposals* an den Sender zu stellen.
Confirm	Der Sender teilt dem Empfänger mit, dass eine Aussage wahr ist, und beabsichtigt hiermit, den Empfänger davon zu überzeugen. Der Sender geht hierbei davon aus, dass der Empfänger sich über den Wahrheitsgehalt der Aussage unsicher ist.
Disconfirm	Der Sender teilt dem Empfänger mit, dass eine Aussage falsch ist, und beabsichtigt hiermit, den Empfänger ebenfalls davon zu überzeugen. Der Sender geht hierbei davon aus, dass der Empfänger sich über den Wahrheitsgehalt der Aussage unsicher ist.
Failure	Der Sender teilt dem Empfänger mit, dass eine Aktion grundsätzlich möglich ist, aber nicht vollständig ausgeführt werden konnte.
Inform	Der Sender teilt dem Empfänger mit, dass eine Aussage wahr ist, und beabsichtigt hiermit, den Empfänger davon zu überzeugen. Der Sender geht hierbei davon aus, dass der Empfänger bisher keine Kenntnis über die Aussage besitzt.
Inform If	Stellt eine Erweiterung zu *Inform* dar, wobei der Inhalt der Nachricht von dem Wissensstand des Senders abhängt, das heißt, der Sender beabsichtigt, den Empfänger von seiner Einstellung gegenüber der zugrunde liegenden Aussage zu überzeugen.
Inform Ref	Stellt eine Erweiterung zu *Inform* dar, wobei der Inhalt der Nachricht aus einem Objekt und einem eindeutigen Deskriptor zusammengesetzt ist.
Not Understood	Der Sender teilt dem Empfänger mit, dass er eine Aktion nicht verstanden hat.

Propagate	Der Sender fordert den Empfänger auf, die enthaltene Nachricht an die spezifizierten Empfänger weiterzuleiten.
Propose	Der Sender schlägt dem Empfänger vor, unter den angegebenen Bedingungen eine bestimmte Aktion auszuführen.
Proxy	Der Sender fordert den Empfänger dazu auf, die übermittelte Nachricht an die angegebenen Agenten zu senden.
Query If	Der Sender fragt den Empfänger, ob eine Aussage wahr ist. Der Sender hat hierbei kein eigenes Wissen über den Wahrheitsgehalt der Aussage, glaubt aber, dass der Empfänger über dieses Wissen verfügt.
Query Ref	Der Sender fragt den Empfänger nach dem durch den Bezeichner referenzierten Objekt.
Refuse	Der Sender teilt dem Empfänger einerseits mit, dass eine Aktion nicht durchgeführt werden konnte, und gibt andererseits die zugrunde liegenden Gründe hierfür an.
Reject Proposal	Stellt eine Ablehnung eines zuvor übermittelten *Proposal* Akts dar und drückt aus, dass der Sender nicht mehr die Absicht hat, dass der Empfänger eine bestimmte Aktion ausführt.
Request	Der Sender fordert den Empfänger auf, eine bestimmte Aktion auszuführen.
Request When	Der Sender fordert den Empfänger auf, eine bestimmte Aktion unter den gegebenen Vorbedingungen auszuführen.
Request Whenever	Der Sender fordert den Empfänger auf, eine bestimmte Aktion immer dann durchzuführen, wenn die Vorbedingungen erfüllt sind.
Subscribe	Der *Subscribe* Akt stellt eine persistente Version des *Query Ref* Akts dar, in dem der Sender den Empfänger auffordert, ihn kontinuierlich über Änderungen des referenzierten Objekts zu informieren.

Tabelle A.2: Kommunikative Akte innerhalb einer FIPA-ACL-Nachricht.

B

ASaP-Ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/
    XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="
    ontology" nsURI="http://www.jofrey.de/ontology" nsPrefix="
    ontology">
  <eClassifiers xsi:type="ecore:EClass" name="State" eSuperTypes="
    #//Event"/>
  <eClassifiers xsi:type="ecore:EClass" name="SensorService"
    eSuperTypes="#//ServiceType"/>
  <eClassifiers xsi:type="ecore:EClass" name="SmartService"
    eSuperTypes="#//OntologyObject">
    <eStructuralFeatures xsi:type="ecore:EReference" name="input"
    upperBound="-1"
        eType="#//Event"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="output"
     upperBound="-1"
        eType="#//Event"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="context
    " upperBound="-1"
        eType="#//Context"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="type"
    upperBound="-1" eType="#//ServiceType"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
    eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
    EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="
    description" eType="ecore:EDataType http://www.eclipse.org/emf
    /2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="
    developer" eType="ecore:EDataType http://www.eclipse.org/emf
    /2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="product
    " eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore
    #//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="app"
    eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
    EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Power" eSuperTypes="
    #//Discrete"/>
```

```
21  <eClassifiers xsi:type="ecore:EClass" name="On" eSuperTypes="#//
      Power"/>
    <eClassifiers xsi:type="ecore:EClass" name="Off" eSuperTypes="
      #//Power"/>
23  <eClassifiers xsi:type="ecore:EClass" name="OntologyObject"/>
    <eClassifiers xsi:type="ecore:EClass" name="AssistanceService"
      eSuperTypes="#//ServiceType"/>
25  <eClassifiers xsi:type="ecore:EClass" name="PersistenceService"
      eSuperTypes="#//ServiceType"/>
    <eClassifiers xsi:type="ecore:EClass" name="InteractionService"
      eSuperTypes="#//ServiceType"/>
27  <eClassifiers xsi:type="ecore:EClass" name="Alarm" eSuperTypes="
      #//Information"/>
    <eClassifiers xsi:type="ecore:EClass" name="DomesticAlarm"
      eSuperTypes="#//Alarm"/>
29  <eClassifiers xsi:type="ecore:EClass" name="Event" eSuperTypes="
      #//OntologyObject">
      <eStructuralFeatures xsi:type="ecore:EReference" name="context
      " upperBound="-1"
31        eType="#//Context"/>
      <eStructuralFeatures xsi:type="ecore:EAttribute" name="
      timestamp" eType="ecore:EDataType http://www.eclipse.org/emf
      /2002/Ecore#//EDate"/>
33    <eStructuralFeatures xsi:type="ecore:EReference" name="sender"
       eType="#//SmartService"/>
    </eClassifiers>
35  <eClassifiers xsi:type="ecore:EClass" name="Spatial" eSuperTypes
      ="#//Context"/>
    <eClassifiers xsi:type="ecore:EClass" name="Context" eSuperTypes
      ="#//OntologyObject"/>
37  <eClassifiers xsi:type="ecore:EClass" name="Physical"
      eSuperTypes="#//Context"/>
    <eClassifiers xsi:type="ecore:EClass" name="ActuatorService"
      eSuperTypes="#//ServiceType"/>
39  <eClassifiers xsi:type="ecore:EClass" name="MedicalAlarm"
      eSuperTypes="#//Alarm"/>
    <eClassifiers xsi:type="ecore:EClass" name="FireAlarm"
      eSuperTypes="#//DomesticAlarm"/>
41  <eClassifiers xsi:type="ecore:EClass" name="BurglaryAlarm"
      eSuperTypes="#//DomesticAlarm"/>
    <eClassifiers xsi:type="ecore:EClass" name="Building"
      eSuperTypes="#//Spatial"/>
43  <eClassifiers xsi:type="ecore:EClass" name="Floor" eSuperTypes="
      #//Spatial">
      <eStructuralFeatures xsi:type="ecore:EAttribute" name="value"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
      EString"/>
45  </eClassifiers>
    <eClassifiers xsi:type="ecore:EClass" name="FallAlarm"
      eSuperTypes="#//MedicalAlarm"/>
47  <eClassifiers xsi:type="ecore:EClass" name="HeartAttackAlarm"
      eSuperTypes="#//MedicalAlarm"/>
    <eClassifiers xsi:type="ecore:EClass" name="Room" eSuperTypes="
      #//Spatial">
49    <eStructuralFeatures xsi:type="ecore:EAttribute" name="number"
       eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore
```

```
    #//EString"/>
  </eClassifiers>
51 <eClassifiers xsi:type="ecore:EClass" name="ActivityZone"
    eSuperTypes="#//Spatial">
    <eStructuralFeatures xsi:type="ecore:EReference" name="
    activity" eType="#//Behavioural"/>
53 </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Bathroom"
    eSuperTypes="#//Room"/>
55 <eClassifiers xsi:type="ecore:EClass" name="Bedroom" eSuperTypes
    ="#//Room"/>
  <eClassifiers xsi:type="ecore:EClass" name="DiningRoom"
    eSuperTypes="#//Room"/>
57 <eClassifiers xsi:type="ecore:EClass" name="LivingRoom"
    eSuperTypes="#//Room"/>
  <eClassifiers xsi:type="ecore:EClass" name="Kitchen" eSuperTypes
    ="#//Room"/>
59 <eClassifiers xsi:type="ecore:EClass" name="Lobby" eSuperTypes="
    #//Room"/>
  <eClassifiers xsi:type="ecore:EClass" name="Lighting"
    eSuperTypes="#//Physical"/>
61 <eClassifiers xsi:type="ecore:EClass" name="Multimedia"
    eSuperTypes="#//Physical"/>
  <eClassifiers xsi:type="ecore:EClass" name="Household"
    eSuperTypes="#//Physical"/>
63 <eClassifiers xsi:type="ecore:EClass" name="Security"
    eSuperTypes="#//Physical"/>
  <eClassifiers xsi:type="ecore:EClass" name="SmokeDetector"
    eSuperTypes="#//Security"/>
65 <eClassifiers xsi:type="ecore:EClass" name="Energy" eSuperTypes=
    "#//Absolute"/>
  <eClassifiers xsi:type="ecore:EClass" name="Production"
    eSuperTypes="#//Energy"/>
67 <eClassifiers xsi:type="ecore:EClass" name="Consumption"
    eSuperTypes="#//Energy"/>
  <eClassifiers xsi:type="ecore:EClass" name="Temperature"
    eSuperTypes="#//Absolute"/>
69 <eClassifiers xsi:type="ecore:EClass" name="MotionDetector"
    eSuperTypes="#//Security"/>
  <eClassifiers xsi:type="ecore:EClass" name="Volume" eSuperTypes=
    "#//Percentage"/>
71 <eClassifiers xsi:type="ecore:EClass" name="OpenClose"
    eSuperTypes="#//Discrete"/>
  <eClassifiers xsi:type="ecore:EClass" name="Open" eSuperTypes="
    #//OpenClose"/>
73 <eClassifiers xsi:type="ecore:EClass" name="Close" eSuperTypes="
    #//OpenClose"/>
  <eClassifiers xsi:type="ecore:EClass" name="Tilted" eSuperTypes=
    "#//OpenClose"/>
75 <eClassifiers xsi:type="ecore:EClass" name="Battery" eSuperTypes
    ="#//Percentage"/>
  <eClassifiers xsi:type="ecore:EClass" name="LightIntensity"
    eSuperTypes="#//Percentage"/>
77 <eClassifiers xsi:type="ecore:EClass" name="EnergyManagement"
    eSuperTypes="#//Physical"/>
```

```
  <eClassifiers xsi:type="ecore:EClass" name="SmartMeter"
    eSuperTypes="#//EnergyManagement"/>
79  <eClassifiers xsi:type="ecore:EClass" name="PowerPlugs"
    eSuperTypes="#//EnergyManagement"/>
  <eClassifiers xsi:type="ecore:EClass" name="ServiceType"
    eSuperTypes="#//OntologyObject"/>
81  <eClassifiers xsi:type="ecore:EClass" name="Standby" eSuperTypes
    ="#//Power"/>
  <eClassifiers xsi:type="ecore:EClass" name="Discrete"
    eSuperTypes="#//State"/>
83  <eClassifiers xsi:type="ecore:EClass" name="Continous"
    eSuperTypes="#//State"/>
  <eClassifiers xsi:type="ecore:EClass" name="Absolute"
    eSuperTypes="#//Continous"/>
85  <eClassifiers xsi:type="ecore:EClass" name="Percentage"
    eSuperTypes="#//Continous"/>
  <eClassifiers xsi:type="ecore:EClass" name="Information"
    eSuperTypes="#//Event">
87    <eStructuralFeatures xsi:type="ecore:EAttribute" name="message
    " eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore
    #//EString"/>
  </eClassifiers>
89  <eClassifiers xsi:type="ecore:EClass" name="Help" eSuperTypes="
    #//Information"/>
  <eClassifiers xsi:type="ecore:EClass" name="Advice" eSuperTypes=
    "#//Help"/>
91  <eClassifiers xsi:type="ecore:EClass" name="
    InstallationInstruction" eSuperTypes="#//Help"/>
  <eClassifiers xsi:type="ecore:EClass" name="Occupation"
    eSuperTypes="#//Absolute"/>
93  <eClassifiers xsi:type="ecore:EClass" name="Streaming"
    eSuperTypes="#//Information">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="url"
    eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
    EString"/>
95    <eStructuralFeatures xsi:type="ecore:EAttribute" name="
    encoding" eType="ecore:EDataType http://www.eclipse.org/emf
    /2002/Ecore#//EString"/>
  </eClassifiers>
97  <eClassifiers xsi:type="ecore:EClass" name="CartesianCoordinates
    " eSuperTypes="#//Spatial">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="x"
    eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
    EFloat"/>
99    <eStructuralFeatures xsi:type="ecore:EAttribute" name="y"
    eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
    EFloat"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="z"
    eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//
    EFloat"/>
101 </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Behavioural"
    eSuperTypes="#//Context"/>
103 <eClassifiers xsi:type="ecore:EClass" name="Audio" eSuperTypes="
    #//Streaming"/>
```

```
    <eClassifiers xsi:type="ecore:EClass" name="Visual" eSuperTypes=
      "#//Streaming"/>
105 <eClassifiers xsi:type="ecore:EClass" name="Camera" eSuperTypes=
      "#//Multimedia"/>
    <eClassifiers xsi:type="ecore:EClass" name="Microphone"
      eSuperTypes="#//Multimedia"/>
107 <eClassifiers xsi:type="ecore:EClass" name="PersonalHygiene"
      eSuperTypes="#//Behavioural"/>
    <eClassifiers xsi:type="ecore:EClass" name="Dressing"
      eSuperTypes="#//Behavioural"/>
109 <eClassifiers xsi:type="ecore:EClass" name="SelfFeeding"
      eSuperTypes="#//Behavioural"/>
    <eClassifiers xsi:type="ecore:EClass" name="Housework"
      eSuperTypes="#//Behavioural"/>
111 <eClassifiers xsi:type="ecore:EClass" name="Communication"
      eSuperTypes="#//Behavioural"/>
    <eClassifiers xsi:type="ecore:EClass" name="Entertainment"
      eSuperTypes="#//Behavioural"/>
113 <eClassifiers xsi:type="ecore:EClass" name="Medication"
      eSuperTypes="#//Behavioural"/>
    <eClassifiers xsi:type="ecore:EClass" name="Washing" eSuperTypes
      ="#//PersonalHygiene"/>
115 <eClassifiers xsi:type="ecore:EClass" name="TeethBrushing"
      eSuperTypes="#//PersonalHygiene"/>
    <eClassifiers xsi:type="ecore:EClass" name="Cooking" eSuperTypes
      ="#//SelfFeeding"/>
117 <eClassifiers xsi:type="ecore:EClass" name="Eating" eSuperTypes=
      "#//SelfFeeding"/>
    <eClassifiers xsi:type="ecore:EClass" name="Shopping"
      eSuperTypes="#//Behavioural"/>
119 <eClassifiers xsi:type="ecore:EClass" name="GoingToToilet"
      eSuperTypes="#//PersonalHygiene"/>
    <eClassifiers xsi:type="ecore:EClass" name="Cleaning"
      eSuperTypes="#//Housework"/>
121 <eClassifiers xsi:type="ecore:EClass" name="Laundry" eSuperTypes
      ="#//Housework"/>
    <eClassifiers xsi:type="ecore:EClass" name="Shaving" eSuperTypes
      ="#//PersonalHygiene"/>
123 <eClassifiers xsi:type="ecore:EClass" name="WatchingTV"
      eSuperTypes="#//Entertainment"/>
    <eClassifiers xsi:type="ecore:EClass" name="ListeningMusic"
      eSuperTypes="#//Entertainment"/>
125 <eClassifiers xsi:type="ecore:EClass" name="Playing" eSuperTypes
      ="#//Entertainment"/>
    <eClassifiers xsi:type="ecore:EClass" name="HomeImprovement"
      eSuperTypes="#//Housework"/>
127 <eClassifiers xsi:type="ecore:EClass" name="Gardening"
      eSuperTypes="#//Housework"/>
    <eClassifiers xsi:type="ecore:EClass" name="Television"
      eSuperTypes="#//Multimedia"/>
129 </ecore:EPackage>
```

Listing B.1: ASaP-Ontology als Ecore XML Export

Literaturverzeichnis

Abchiche-Mimouni, N., Andriatrimoson, A., Colle, E., und Galerne, S. (2013). Multidimensional Adaptiveness in Multi-Agent Systems. *International Journal On Advances in Intelligent Systems*, 6(1):124–135.

Ahlers, E., Bachfeld, D., Endres, J., Hansen, S., Hilgefort, U., Klan, F., und Schuster, J. (2014). Smart Home - Praxisratgeber für intelligentes Wohnen. *c't Wissen*.

Alexandersson, J. (2008). i2home - Towards a universal home environment for elderly and disabled. *KI - Künstliche Intelligenz, German Journal on Artificial Intelligence - Organ des Fachbereiches "Künstliche Intelligenz"der Gesellschaft für Informatik e.V.*, 22(3):66–68.

Alexandersson, J., Bund, J., Carrasco, E., Epelde, G., Klíma, M., Urdaneta, E., Vanderheiden, G., Zimmermann, G., und Zinnikus, I. (2011). openURC: Standardisation towards "User Interfaces for Everyone, Everywhere, on Anything". In Wichert, R. und Eberhardt, B., editors, *Ambient Assisted Living*, pages 117–125. Springer Berlin Heidelberg.

Alexandersson, J., Richter, K., und Becker, S. (2006). I2Home: Benutzerzentrierte Entwicklung einer offenen standardbasierten Smart Home Plattform. In *Proceedings of USEWARE 2006 - Nutzergerechte Gestaltung technischer Systeme*, Düsseldorf. VDI/VDE-Gesellschaft für Mess- und Automatisierungstechnik, VDI-Verlag.

Alexandersson, J., Zinnikus, I., Frey, J., Zimmermann, G., Epelde, G., Bund, J., und Rosa, B. (2010). Convergence of Internet of Services and Internet of Appliances: Extending the Universal Remote Console. In *Workshop on future standards for Model-Based User Interfaces*, Rome, Italy.

Amft, O. und Lukowicz, P. (2009). From Backpacks to Smartphones: Past, Present, and Future of Wearable Computers. *IEEE Pervasive Computing*, 8(3):8–13.

Andriatrimoson, A., Abchiche-Mimouni, N., Colle, E., und Galerne, S. (2012). An Adaptive Multi-agent System for Ambient Assisted Living. In *Proceedings of the Fourth International Conference on Adaptive and Self-Adaptive Systems and Applications*, pages 85–92.

Augusto, J. (2010). Past, Present and Future of Ambient Intelligence and Smart Environments. In Filipe, J., Fred, A., und Sharp, B., editors, *Agents and Artificial Intelligence*, pages 3–15. Springer Berlin Heidelberg.

Augusto, J., Callaghan, V., Cook, D. J., Kameas, A., und Satoh, I. (2013). Intelligent Environments: a manifesto. *Human-centric Computing and Information Sciences*, 3(1):1–18.

Augusto, J. C. (2007). Ambient Intelligence: The Confluence of Ubiquitous/-Pervasive Computing and Artificial Intelligence. In *Intelligent Computing Everywhere*, pages 213–234. Springer London.

Augusto, J. C., Huch, M., Kameas, A., Maitland, J., McCullagh, P. J., Roberts, J., Sixsmith, A., und Wichert, R. (2012). *Handbook of Ambient Assisted Living: Technology for Healthcare, Rehabilitation and Well-being (Ambient Intelligence and Smart Environments)*. IOS Press, Berlin.

Augusto, J. C. und O'Donoghue, J. (2009). Context-Aware Agents - The 6Ws Architecture. In Filipe, J., Fred, A., und Sharp, B., editors, *International Conference on Agents and Artificial Intelligence*, pages 591–594.

Augusto, J. C., Zheng, H., Mulvenna, M., Wang, H., Carswell, W., und Jeffers, P. (2011). Design and Modelling of the Nocturnal AAL Care System. In Novais, P., Preuveneers, D., und Corchado, J., editors, *Ambient Intelligence - Software and Applications*, pages 109–116. Springer Berlin Heidelberg.

Austin, J. L. (1962). *How to do Things with Words*. Oxford University Press, New York.

Aztiria, A., Izaguirre, A., und Augusto, J. C. (2010). Learning patterns in ambient intelligence environments: a survey. *Artificial Intelligence Review*, 34(1):35–51.

Balasch, M. C., Budych, K., Bußer, J. U., Carius-Düssel, C., Cornils, M., Downes, R., Gövercin, M., Jansen, J. P., Kiselev, J., Kuhlmann, M., Reukauf, T., Schlösser, M., Schultz, M., Shin, I. H., Trachterna, A., Voigt, B., und Winnig, K. (2012). SmartSenior – Intelligente Dienste und Dienstleistungen für Senioren. In Kunze, H. und Mutze, S., editors, *Telemedizin: Jahrbuch HealthCapital*. Oldenbourg Wissenschaftsverlag, Berlin-Brandenburg.

Bellifemine, F. L., Caire, G., und Greenwood, D. (2007). *Developing Multi-Agent Systems with JADE*. Wiley Series in Agent Technology. Wiley & Sons.

Bianco, P., Kotermanski, R., und Merson, P. (2007). Evaluating a Service-Oriented Architecture. Technical report.

Bodden, E., Schneider, M., Kreutzer, M., Mezini, M., Hammer, C., Zeller, A., Achenbach, D., Huber, M., und Kraschewski, D. (2013). Entwicklung sicherer Software durch Security by Design. *Fraunhofer Verlag, SIT Technical Reports*.

Böhmer, M., Ganev, L., und Krüger, A. (2013). AppFunnel: A Framework for Usage-centric Evaluation of Recommender Systems That Suggest Mobile Applications. In *Proceedings of the 2013 International Conference on Intelligent User Interfaces*, pages 267–276, New York, NY, USA. ACM.

Boman, M., Davidsson, P., und Younes, H. L. S. (2013). Artificial Decision Making Under Uncertainty in Intelligent Buildings. *CoRR*, abs/1301.6680.

Bonino, D., Castellina, E., Corno, F., und Di Torino, P. (2008). DOG: An Ontology-Powered OSGi Domotic Gateway. In *ICTAI 20 the IEEE Int Conference on Tools with Artificial Intelligence*, pages 157–160.

Bonino, D. und Corno, F. (2008). DogOnt - Ontology Modeling for Intelligent Domotic Environments. In *In 7th International Semantic Web Conference, Lecture Notes on Computer Science*, pages 790–803. Springer-Verlag.

Bordini, R. H., Hubner, J. F., und Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak Using Jason*. Wiley Series in Agent Technology. Wiley & Sons.

Brandherm, B., Baus, J., und Frey, J. (2012). Peer Energy Cloud – Civil Marketplace for Trading Renewable Energies. In *Proceedings of the 8th International Conference on Intelligent Environments (IE)*, pages 375–378, Guanajuato, Mexico. IEEE Computer Society.

Brandherm, B., Baus, J., und Frey, J. (2013). Peer Energy Cloud - Marketplace for Renewable Energy. *Germany Trade & Invest*, (1).

Bratman, M. (1987). *Intention, plans, and practical reason*. Harvard University Press.

Braubach, L., Pokahr, A., und Lamersdorf, W. (2005). Jadex: A BDI-Agent System Combining Middleware and Reasoning. In Unland, R., Calisti, M., und Klusch, M., editors, *Software Agent-Based Applications, Platforms and Development Kits*, pages 143–168. Birkhäuser Basel.

Britz, J., Alexandersson, J., und Stephan, W. (2015). UCH goes EAL4 — The foundation of an Eco System for Ambient Assisted Living: ISO/IEC 15408 Common Critera based Implementation of the ISO/IEC 24752 Universal Control Hub Middleware . In *8. AAL Kongress, 29. - 30. April 2015*, Frankfurt/Main.

Britz, J., Frey, J., und Alexandersson, J. (2014). Bridging the Gap between Smart Home and Agents. In *Proceedings of the 10th International Conference on Intelligent Environments (IE)*, pages 31–38, Shanghai, China.

Brooks, R. (1991). Intelligence Without Representation. *Artificial Intelligence*, 47:139–159.

Caire, G., Gotta, D., und Banzi, M. (2008). WADE: A Software Platform to Develop Mission Critical Applications Exploiting Agents and Workflows. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Industrial Track*, pages 29–36, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Callaghan, V., Clarke, G., Colley, M., und Hagras, H. (2001). Embedding intelligence: Research issues for ubiquitous computing. In *Proceedings of the 1st Equator IRC Workshop on Ubiquitous Computing Environments*, pages 110–130.

Callaghan, V., Clarke, G., Colley, M., und Hagras, H. (2002). A Soft-Computing based Distributed Artificial Intelligence Architecture for Intelligent Buildings. In Loia, S. und Sessa, S., editors, *Soft Computing agents: New Trends for Designing Autonomous Systems*, pages 117–145. Springer.

Carpenter, B. (1992). *The Logic of Typed Feature Structures*. Cambridge University Press, New York, NY, USA.

Carswell, W., Augusto, J., Mulvenna, M., Wallace, J., Martin, S., McCullagh, P. J., Zheng, H., Wang, H., McSorley, K., Taylor, B., und Jeffers, W. P. (2011). The NOCTURNAL Ambient Assisted Living system. In *Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2011 5th International Conference on*, pages 208–209.

Cervantes, L., Lee, Y.-S., Yang, H., Ko, S.-h., und Lee, J. (2007). Agent-Based Intelligent Decision Support for the Home Healthcare Environment. In Szczuka, M., Howard, D., Ślęzak, D., Kim, H.-k., Kim, T.-h., Ko, I.-s., Lee, G., und Sloot, P. A., editors, *Advances in Hybrid Information Technology*, pages 414–424. Springer Berlin Heidelberg.

Chen, H. (2004). *An Intelligent Broker Architecture for Pervasive Context-Aware Systems* . PhD thesis, Faculty of the Graduate School of the University of Maryland.

Chen, H., Finin, T., und Joshi, A. (2005). The SOUPA Ontology for Pervasive Computing. In *Ontologies for Agents: Theory and Experiences*, pages 233–258. BirkHauser.

Cook, D. J. (2009). Multi-agent Smart Environments. *Journal of Ambient Intelligence and Smart Environments*, 1(1):51–55.

Cook, D. J. und Das, S. K. (2007). How Smart Are Our Environments? An Updated Look at the State of the Art. *Pervasive Mob. Comput.*, 3(2):53–73.

Cook, D. J. und Krishnan, N. (2014). Mining the Home Environment. *Journal of Intelligent Information Systems*, 43(3):503–519.

Cook, D. J., Youngblood, M., Heierman, III, E. O., Gopalratnam, K., Rao, S., Litvin, A., und Khawaja, F. (2003). MavHome: An Agent-Based Smart Home. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, pages 521–521, Washington, DC, USA. IEEE Computer Society.

Crestani, F. (1997). Application of Spreading Activation Techniques in Information Retrieval. *Artificial Intelligence Review*, 11:453–482.

Dastani, M., van Birna Riemsdijk, M., und Meyer, J.-J. (2005). Programming Multi-Agent Systems in 3APL. In Bordini, R., Dastani, M., Dix, J., und El Fallah Seghrouchni, A., editors, *Multi-Agent Programming*, pages 39–67. Springer US.

Davidsson, P. und Boman, M. (2000). Saving Energy and Providing Value Added Services in Intelligent Buildings: A MAS Approach. In *Proceedings of the Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents*, pages 166–177, London, UK. Springer-Verlag.

Decker, K. und Sycara, K. (1997). Intelligent Adaptive Information Agents. *Journal of Intelligent Information Systems*, 9(3):239–260.

Dey, A. K. (2001). Understanding and Using Context. *Personal Ubiquitous Computing*, 5(1):4–7.

Di Noia, T., Di Sciascio, E., und Donini, F. M. (2007). Semantic Matchmaking As Non-monotonic Reasoning: A Description Logic Approach. *Journal of Artificial Intelligence Research*, 29(1):269–307.

Dillon, T. (2006). Pervasive and ubiquitous computing. *http://www2futurelaborguk/resources/publications-reports-articles/web-articles/Web-Article497*.

Dimitrov, S. (2014). Framework for Analyzing Sounds of Home Environment for Device Recognition . Master's thesis, Saarland University, Dept. of Computer Science.

Dimitrov, S., Britz, J., Brandherm, B., und Frey, J. (2014). Analyzing Sounds of Home Environment for Device Recognition. In *Proceedings of the European Conference on Ambient Intelligence*, pages 1–16, Eindhoven, Netherlands.

Durfee, E. H., Lesser, V. R., und Corkill, D. D. (1989). Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):63–83.

Eberhart, A. (2003). *Ontology-based Infrastructure for Intelligent Applications*. PhD thesis, Saarland University, Dept. of Computer Science.

Epelde, G. (2014). *User Interface Abstraction for enabling TV set based Inclusive Access to the Information Society*. PhD thesis, University of the Basque Country, Computer Architecture and Technology Department, San Sebastian.

Epelde, G., Carrasco, E., Zimmermann, G., Alexandersson, J., Neßelrath, R., und Dubielzig, M. (2013). Universal Remote Console-based next-generation accessible television. *Universal Access in the Information Society*, 12(1):73–87.

Fagerberg, G., Kung, A., Wichert, R., Tazari, M.-R., Jean-Bart, B., Bauer, G., Zimmermann, G., Furfari, F., Potortì, F., Chessa, S., Hellenschmidt, M., Gorman, J., Alexandersson, J., Bund, J., Carrasco, E., Epelde, G., Klíma, M., Urdaneta, E., Vanderheiden, G., und Zinnikus, I. (2010). Platforms for AAL Applications. In Lukowicz, P., Kunze, K., und Kortuem, G., editors, *Smart Sensing and Context*, pages 177–201. Springer Berlin Heidelberg.

Fensel, A., Tomic, S., Kumar, V., Stefanovic, M., Aleshin, S. V., und Novikov, D. O. (2013). SESAME-S: Semantic Smart Home System for Energy Efficiency. *Informatik Spektrum*, 36(1):46–57.

Fensel, D., Hendler, J. A., Lieberman, H., und Wahlster, W., editors (2003). *Spinning the Semantic Web*. MIT Press, Cambridge, MA.

Finin, T., Fritzson, R., McKay, D., und McEntire, R. (1994). KQML As an Agent Communication Language. In *Proceedings of the Third International Conference on Information and Knowledge Management*, pages 456–463, New York, USA. ACM.

FIPA (2002a). FIPA Abstract Architecture Specification . FIPA TC Architecture.

FIPA (2002b). FIPA ACL Message Structure Specification . FIPA TC Communication.

FIPA (2002c). FIPA Communicative Act Library Specification . FIPA TC Communication.

FIPA (2002d). FIPA Device Ontology Specification . FIPA TC Gateways.

Fleisch, E. und Mattern, F., editors (2005). *Das Internet der Dinge: Ubiquitous Computing und RFID in der Praxis*. Springer, Berlin.

Frey, J. (2013). AdAPT - A Dynamic Approach for Activity Prediction and Tracking for Ambient Intelligence. In *9th International Conference on Intelligent Environments (IE)*, pages 254–257, Athens, Greece.

Frey, J., Bergweiler, S., Alexandersson, J., Gholamsaghaee, E., Reithinger, N., und Stahl, C. (2011a). SmartCase: A Smart Home Environment in a Suitcase. In *Proceedings of the 7th International Conference on Intelligent Environments (IE)*, pages 378–381, Nottingham, UK.

Frey, J., Brandherm, B., und Baus, J. (2012). Peer Energy Cloud - Trading Renewable Energies. In *Proceedings of the 35th German Conference on Artificial Intelligence*, Saarbrücken, Germany.

Frey, J., Gard, T., und Alexandersson, J. (2010a). User-Centred Tool Support: Developing User Interfaces for Persons with Special Needs. In *Workshop on Tool-Support for Mobile and Pervasive Application Development*.

Frey, J., Neßelrath, R., Schulz, C. H., und Alexandersson, J. (2010b). SensHome: Towards A Corpus for Everyday Activities in Smart Homes. In *International Conference on Language Resources and Evaluation*, pages 137–139.

Frey, J., Neßelrath, R., und Stahl, C. (2011b). An Open Standardized Platform for Dual Reality Applications. In *Proceedings of the Second International Workshop on Location Awareness for Mixed and Dual Reality. Workshop on Location Awareness for Mixed and Dual Reality (LAMDa-11), 2nd, located at International Conference on Intelligent User Interfaces*, pages 1–4, Palo Alto, CA, USA.

Frey, J., Neurohr, C., und Brandherm, B. (2014). EvA - Self Adaptable Event-based Recognition Framework for Three-Dimensional Activity Zones. In *Proceedings of the 10th International Conference on Intelligent Environments (IE)*, pages 87–94, Shanghai, China.

Frey, J., Schulz, C. H., Neßelrath, R., Stein, V., und Alexandersson, J. (2010c). Towards Pluggable User Interfaces for People with Cognitive Disabilities. In *3rd International Conference on Health Informatics*, pages 428–431.

Frey, J., Stahl, C., Röfer, T., Krieg-Brückner, B., und Alexandersson, J. (2010d). The DFKI Competence Center for Ambient Assisted Living. In *Proceedings of the First International Joint Conference on Ambient Intelligence (AmI'10)*, pages 310–314. Springer-Verlag.

Friedewald, M. und Da Costa, O. (2003). Science and Technology Roadmapping: Ambient Intelligence in Everyday Life (AmI@Life). Technical report.

Furfari, F., Sommaruga, L., Soria, C., und Fresco, R. (2004). DomoML: The Definition of a Standard Markup for Interoperability of Human Home Interactions. In *Proceedings of the 2Nd European Union Symposium on Ambient Intelligence*, pages 41–44, New York, NY, USA. ACM.

Gamma, E. und Beck, K. (2003). *Contributing to Eclipse: Principles, Patterns, and Plugins*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Garcia Sanchez, P., Garcia Arenas, M., Mora, A. M., Castillo, P. A., Fernandes, C., de las Cuevas, P., Romero, G., Gonz a lez, J., und Merelo, J. J. (2013). Developing Services in a Service Oriented Architecture for Evolutionary Algorithms. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, pages 1341–1348, New York, NY, USA. ACM.

Gard, T. (2010). TESA: Tool zur Erstellung Socket-basierter Anwendungen . Master's thesis, Saarland University, Dept. of Computer Science.

Gard, T. (2012). S3PO - Creating Ontology-based Spatial Semantic Models Using Deictic Pointing and Spoken Language. Master's thesis, Saarland University, Dept. of Computer Science.

Gauterin, A., Alexandersson, J., Neßelrath, R., Schulz, C. H., und Frey, J. (2012). Accessible Elevator. In *Technik für ein selbstbestimmtes Leben - 5. Deutscher AAL-Kongress 01/24/2012 - 01/25/2012 at Berlin*.

Gholamsaghaee, E., Reithinger, N., und Frey, J. (2012). Automatic Generation of Intuitive User Interfaces for Controlling and Monitoring of Home Appliances for Mobile Devices. In *Technik für ein selbstbestimmtes Leben - 5. Deutscher AAL-Kongress 01/24/2012 - 01/25/2012 at Berlin*.

Gomez-Pérez, A., Fernandez-Lopez, M., und Corcho, O. (2007). *Ontological Engineering: With Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web. (Advanced Information and Knowledge Processing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Gruber, T. R. (1995). Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal of Human-Computer Studies - Special issue: the role of formal ontology in the information technology*, 43(5-6):907–928.

Gruber, T. R. (2009). Ontology. In *Encyclopedia of Database Systems*, pages 1963–1965.

Grütter, R. (2006). Software-Agenten im Semantic Web. *Informatik Spektrum*, 29(1):3–13.

Hagras, H., Callaghan, V., Colley, M., Clarke, G., Pounds-Cornish, A., und Duman, H. (2004). Creating an Ambient-Intelligence Environment Using Embedded Agents. *Intelligent Systems, IEEE*, 19(6):12–20.

Hanke, S., Mayer, C., Hoeftberger, O., Boos, H., Wichert, R., Tazari, M.-R., Wolf, P., und Furfari, F. (2011). universAAL – An Open and Consolidated AAL Platform. In Wichert, R. und Eberhardt, B., editors, *Ambient Assisted Living*, pages 127–140. Springer Berlin Heidelberg.

Haupert, J. (2013). *DOMeMan : Repräsentation, Verwaltung und Nutzung von digitalen Objektgedächtnissen*, volume 339 of *Dissertationen zur Künstlichen Intelligenz*. Akademische Verlagsgesellschaft AKA GmbH, Berlin.

Heckmann, D. (2006). *Ubiquitous User Modeling*, volume 297 of *Dissertationen zur Künstlichen Intelligenz*. Akademische Verlagsgesellschaft AKA GmbH, Berlin.

Heuser, L. und Wahlster, W., editors (2011). *Internet der Dienste*. acatech diskutiert. Springer, Berlin.

Hillairet, G., Bertrand, F., und Lafaye, J. Y. (2008). Bridging EMF Applications and RDF Data Sources. In *4th International Workshop on Semantic Web Enabled Software Engineering*.

Hirsch, B., Konnerth, T., Burkhardt, M., und Albayrak, S. (2010). Programming Service Oriented Agents. In Calisti, M., Dignum, F. P., Kowalczyk, R., Leymann, F., und Unland, R., editors, *Service-Oriented Architecture and (Multi-)Agent Systems Technology*, Dagstuhl, Germany. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

Hörz, M. (2012). SketchyMate: Participatory User Interface Design with Interactive Paper Prototypes . Master's thesis, Saarland University, Dept. of Computer Science.

Hudson, D. L. und Cohen, M. E. (2002). Use of Intelligent Agents in the Diagnosis of Cardiac Disorders. In *Computers in Cardiology, 2002*, pages 633–636.

Hyrynsalmi, S., Mäkilä, T., Järvi, A., Suominen, A., Seppänen, M., und Knuutila, T. (2012). App Store, Marketplace, Play! An Analysis of Multi-Homing in Mobile Software Ecosystems. In Jansen, S., Bosch, J., und Alves, C. F., editors, *IWSECO@ICSOB*, pages 59–72. CEUR-WS.org.

IEC (2014). Electricity metering data exchange - The DLMS/COSEM suite. *Inernational Electrotechnical Commission (IEC)*.

ISO (2008a). Information Technology - User Interfaces - Universal Remote Console - Part 1: Framework. *International Organization for Standardization (ISO)*, pages 1–56.

ISO (2008b). Information Technology - User Interfaces - Universal Remote Console - Part 2: User Interface Socket Description. *International Organization for Standardization (ISO)*, pages 1–51.

ISO (2008c). Information Technology - User Interfaces - Universal Remote Console - Part 3: Presentation Template. *International Organization for Standardization (ISO)*, pages 1–18.

ISO (2008d). Information Technology - User Interfaces - Universal Remote Console - Part 4: Target Description. *International Organization for Standardization (ISO)*, pages 1–14.

ISO (2008e). Information Technology - User Interfaces - Universal Remote Console - Part 5: Resource Description. *International Organization for Standardization (ISO)*, pages 1–49.

ISO (2009). Information technology – Security techniques – Evaluation criteria for IT security. *International Organization for Standardization (ISO)*.

Jakob, R. und Weiß, G. (2004). *Agentenorientierte Softwareentwicklung: Methoden und Werkzeuge*. Springer.

Jamshidi, M. (2011). *System of Systems Engineering: Innovations for the Twenty-First Century*. Wiley Series in Systems Engineering and Management. Wiley.

Jansen, S. und Bloemendal, E. (2013). Defining App Stores: The Role of Curated Marketplaces in Software Ecosystems. In Herzwurm, G. und Margaria, T., editors, *Software Business. From Physical Products to Software Services and Solutions*, pages 195–206. Springer Berlin Heidelberg.

Jennings, N. R. und Wooldridge, M. (1998). Applications of Intelligent Agents. In *Agent Technology: Foundations, Applications, and Markets*, pages 3–28. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Kagermann, H., Riemensperger, F., Hoke, D., Helbig, J., Stocksmeier, D., Wahlster, W., und Scheer, A.-W., editors (2014). *Smart Service Welt.* Umsetzungsempfehlungen für das Zukunftsprojekt Internetbasierte Dienste für die Wirtschaft. acatech - Deutsche Akademie der Technikwissenschaften, Berlin.

Kagermann, H., Riemensperger, F., Hoke, D., Schuh, G., Scheer, A.-W., Spath, D., Leukert, B., Wahlster, W., Rohleder, B., und Schweer, D., editors (2015). *Smart Service Welt.* Umsetzungsempfehlungen für das Zukunftsprojekt Internetbasierte Dienste für die Wirtschaft (Abschlussbericht). acatech - Deutsche Akademie der Technikwissenschaften, Berlin.

Kagermann, H., Wahlster, W., und Helbig, J., editors (2013). *Deutschlands Zukunft als Produktionsstandort sichern: Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0, Abschlussbericht des Arbeitskreises Industrie 4.0.* Forschungsunion im Stifterverband für die Deutsche Wirtschaft e.V., Berlin.

Kahl, G. (2014). *Dual Reality Framework – Basistechnologien zum Monitoring und Steuern von Cyber-Physischen Umgebungen.* PhD thesis, Saarland University, Dept. of Computer Science.

Kahl, G. und Bürckert, C. (2012). Architecture to Enable Dual Reality for Smart Environments. In *Proceedings of the 8th International Conference on Intelligent Environments (IE)*, pages 42–49, Guanajuato, Mexico. IEEE.

Kalfoglou, Y. und Schorlemmer, M. (2003). Ontology Mapping: The State of the Art. *The Knowledge Engineering Review*, 18(1):1–31.

Klusch, M. (2001). Information Agent Technology for the Internet: A Survey. *Data & Knowledge Engineering - Special issue on heterogeneous information resources need semantic access*, 36(3):337–372.

Klusch, M. (2008). *On Agent-Based Semantic Service Coordination.* Habilitation, Saarland University, Dept. of Computer Science.

Klusch, M. und Kapahnke, P. (2012). The iSeM Matchmaker: A Flexible Approach for Adaptive Hybrid Semantic Service Selection. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 15:1–14.

Koehler, J. und Ottiger, D. (2002). An AI-based Approach to Destination Control in Elevators. *AI Magazine*, 23(3):59–78.

Koile, K., Tollmar, K., Demirdjian, D., Shrobe, H., und Darrell, T. (2003). Activity Zones for Context-Aware Computing. In *Proceedings of 5th International Conference on Ubiquitous Computing*, pages 90–106, Seattle, WA, USA. Springer-Verlag.

Krüger, F., Kasparick, M., Mundt, T., und Kirste, T. (2014). Where are my colleagues and why? Tracking multiple persons in indoor environments. In *Proceedings of the 10th International Conference on Intelligent Environments (IE)*, pages 190–197, Shanghai, China.

Krumm, J., editor (2010). *Ubiquitous Computing Fundamentals*. CRC Press, Boca Raton, FL, USA.

Lackes, R. und Siepermann, M. (2014a). Middleware. *Gabler Wirtschaftslexikon* *http://wirtschaftslexikongablerde/Archiv/78647/middleware-v8html*.

Lackes, R. und Siepermann, M. (2014b). Ubiquitous Computing. *Gabler Wirtschaftslexikon* *http://wirtschaftslexikon.gabler.de/Archiv/76604/ubiquitous-computing-v8.html*.

Laney, D. (2001). 3-D Data Management: Controlling Data Volume, Velocity and Variety. *Application Delivery Strategies by META Group Inc.*, 949.

Lanzola, G. und Boley, H. (2002). Experience with a Functional-logic Multiagent Architecture for Medical Problem Solving. In Grütter, R., editor, *Knowledge Media in Healthcare*, pages 17–37. IGI Global, Hershey, PA, USA.

Lee, G.-H., Yoon, Y.-J., Lee, S.-H., Choi, K.-H., und Shin, D.-R. (2008). Design of Directory Facilitator for Agent-Based Service Discovery in Ubiquitous Computing Environments. In Fyfe, C., Kim, D., Lee, S.-Y., und Yin, H., editors, *Intelligent Data Engineering and Automated Learning – IDEAL 2008*, pages 412–419. Springer Berlin Heidelberg.

Loskyll, M., Heckmann, D., und Kobayashi, I. (2009). UbisEditor 3.0: Collaborative Ontology Development on the Web. In *Proceedings of the Workshop on Web 3.0: Merging Semantic Web and Social Web*.

Lukowicz, P., Amft, O., Roggen, D., und Cheng, J. (2010). On-Body Sensing: From Gesture-Based Input to Activity-Driven Interaction. *IEEE Computer*, 43(10):92–96.

Lukowicz, P., Kirstein, T., und Tröster, G. (2004). Wearable systems for health care applications. *Methods of Information in Medicine*, 43:232–238.

Lukowicz, P., Pentland, S., und Ferscha, A. (2012). From context awareness to socially aware computing. *IEEE Pervasive Computing*, 11(1):32–41.

Lutzenberger, M., Kuster, T., Konnerth, T., Thiele, A., Masuch, N., Heßler, A., Keiser, J., Burkhardt, M., Kaiser, S., Tonn, J., Kaisers, M., und Albayrak, S. (2013). A Multi-agent Approach to Professional Software Engineering. In Cossentino, M., El Fallah Seghrouchni, A., und Winikoff, M., editors, *Engineering Multi-Agent Systems*, pages 156–175. Springer Berlin Heidelberg.

Lyre, H. (2002). *Informationstheorie : Eine philosophisch naturwissenschaftliche Einführung*. UTB für Wissenschaft. Fink, München.

MacDougall, W. (2013). Industrie 4.0: Smart Manufacturing for the Future. *Germany Trade and Invest Gesellschaft für Außenwirtschaft und Standortmarketing mbH*.

Marcotte, E. (2011). *Responsive Web Design*. Brief Books for People Who Make Websites. A Book Apart.

Mattern, F., editor (2007). *Die Informatisierung des Alltags: Leben in smarten Umgebungen*. Springer, Berlin.

Matuszek, C., Cabral, J., Witbrock, M., und Deoliveira, J. (2006). An introduction to the syntax and content of Cyc. In *Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, pages 44–49.

Maximilien, E. M. und Singh, M. P. (2004). A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing*, 8(5):84–93.

Maximilien, E. M. und Singh, M. P. (2005). Multiagent System for Dynamic Web Services Selection. In *Proceedings of 1st Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE at AAMAS*, pages 25–29.

Maybury, M. T. und Wahlster, W., editors (1998). *Readings in Intelligent User Interfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

McAffer, J., Lemieux, J.-M., und Aniszczyk, C. (2010). *Eclipse Rich Client Platform*. Addison-Wesley Professional, 2nd edition.

McNaull, J., Augusto, J. C., Mulvenna, M. D., und McCullagh, P. J. (2011). Multi-agent Interactions for Ambient Assisted Living. In *7th International Conference on Intelligent Environments, IE 2011, Nottingham, United Kingdom, July 25-28, 2011*, pages 310–313.

Moore, G. E. (1965). Cramming More Components onto Integrated Circuits. *Electronics*, 38(8):114–117.

Müller, J. P. und Fischer, K. (2014). Application Impact of Multi-agent Systems and Technologies: A Survey. In Shehory, O. und Sturm, A., editors, *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*. Springer, Berlin.

Murukannaiah, P. K. und Singh, M. P. (2014). Xipho: Extending Tropos to Engineer Context-aware Personal Agents. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, pages 309–316, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Nakashima, H., Aghajan, H., und Augusto, J. C. (2009). *Handbook of Ambient Intelligence and Smart Environments*. Springer Publishing Company, Incorporated, 1st edition.

Neßelrath, R. und Feld, M. (2014). SiAM-dp: A Platform for the Model-Based Development of Context-Aware Multimodal Dialogue Applications. In *Proceedings of the 10th International Conference on Intelligent Environments*, Shanghai, China. IEEE.

Neßelrath, R., Haupert, J., Frey, J., und Brandherm, B. (2011a). Supporting Persons with Special Needs in their Daily Life in a Smart Home. In *Proceedings of the 7th International Conference on Intelligent Environments (IE)*, pages 370–373, Nottingham, UK.

Neßelrath, R., Lu, C., Schulz, C. H., Frey, J., und Alexandersson, J. (2011b). A Gesture Based System for Context-Sensitive Interaction with Smart Homes. In *Ambient Assisted Living, 4 AAL-Kongress*, Berlin.

Neurohr, C. (2013). EvA - Event-basierte Erkennung und Visualisierung dreidimensionaler Aktivitätszonen in Instrumentierten Umgebungen . Master's thesis, Saarland University, Dept. of Computer Science.

Ottosson, H., Akkermans, H., Ygge, F., und EnerSearch (1998). *The ISES Project: Information/society/energy/system*. EnerSearch.

Pfleger, N. (2008). *Context based multimodal interpretation: an integrated approach to multimodal fusion and discourse processing.* PhD thesis, Saarland University.

Plattner, H. (2013). Big Data. *Enzyklopädie der Wirtschaftsinformatik* *http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/daten-wissen/Datenmanagement/Datenmanagement--Konzepte-des/Big-Data*.

Poslad, S. (2007). Specifying Protocols for Multi-agent Systems Interaction. *ACM Transactions on Autonomous and Adaptive Systems*, 2(4).

Preece, J., Rogers, Y., und Sharp, H. (2002). *Interaction Design*. Wiley & Sons, New York, NY, USA, 1st edition.

Ram, R., Furfari, F., Girolami, M., Ibañez-Sánchez, G., Lázaro-Ramos, J.-P., Mayer, C., Prazak-Aram, B., und Zentek, T. (2013). universAAL: Provisioning Platform for AAL Services. In van Berlo, A., Hallenborg, K., Rodríguez, J. M. C., Tapia, D. I., und Novais, P., editors, *Ambient Intelligence - Software and Applications*, pages 105–112. Springer International Publishing.

Rao, A. S. und Georgeff, M. P. (1995). BDI Agents: From Theory to Practice. In *In Proceedings of the First International Conference on Multiagent Systems (ICMAS'95)*, pages 312–319.

Reisberg, B., Finkel, S., Overall, J., Schmidt-Gollas, N., Kanowski, S., Lehfeld, H., Sclan, S. G., Wilms, H. U., Heininger, K., Hindmarch, I., Stemmler, M., Poon, L., Kluger, A., Cooler, C., Bergener, M., Hugonot-Diener, L., Robert, P. H., Antipolis, S., und Erzigkeit, H. (2001). The Alzheimer's disease activities of daily living international scale (ADL-IS). *International Psychogeriatrics*, 13(2):163–181.

Röfer, T., Laue, T., und Gersdorf, B. (2009a). iWalker - An Intelligent Walker providing Services for the Elderly. In *Technically Assisted Rehabilitation 2009. European Conference on Technically Assisted Rehabilitation (TAR-09), March 18-19, Berlin, Germany*. VDe/VDI.

Röfer, T., Mandel, C., und Laue, T. (2009b). Controlling an Automated Wheelchair via Joystick/Head-Joystick Supported by Smart Driving Assistance. In *Proceedings of the 2009 IEEE 11th International Conference on Rehabilitation Robotics*, pages 743–748. IEEE.

Russell, S. J. und Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 3 edition.

Salber, D., Dey, A. K., und Abowd, G. D. (1999). The Context Toolkit: Aiding the Development of Context-enabled Applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 434–441, New York, NY, USA. ACM.

Schäfer, U., Arnold, F., 0002, S. O., und Reifers, S. (2013). Ingredients and Recipe for a Robust Mobile Speech-Enabled Cooking Assistant for German. In Timm, I. J. und Thimm, M., editors, *Lecture Notes in Artificial Intelligence*, pages 212–223. Springer.

Schneider, C. (2010a). Towards an Eclipse Ontology Framework: Integrating OWL and the Eclipse Modeling Framework . In *Proceedings of the 3rd Workshop on Transforming and Weaving Ontologies in Model Driven Engineering*, Málaga, Spain.

Schneider, M. (2010b). *Resource-Aware Plan Recognition in Instrumented Environments*. PhD thesis, Saarland University, Dept. of Computer Science.

Schuler, D. und Namioka, A., editors (1993). *Participatory Design: Principles and Practices*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.

Schulz, C. H., Zinnikus, I., Kapahnke, P., Frey, J., Neßelrath, R., und Alexandersson, J. (2011). Universally Accessible Interactive Services on TV: A Case Study on the Provisioning of Internet Services to Elderly People. In *Ambient Assisted Living, 4 AAL-Kongress*, Berlin. Springer.

Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge.

Sernani, P., Claudi, A., palazzo, L., dolcini, G., und dragoni, A. F. (2013a). Towards the Definition of a Multi-Agent Architecture For Ambient Assisted Living: The Virtual Carer Project. *Egyptian Computer Science Journal*, 37.

Sernani, P., palazzo, L., Claudi, A., dolcini, G., Biancucci, G., Trentalange, G., und dragoni, A. F. (2013b). Virtual Carer: A BDI Agent System for Ambient Assisted Living. In *Proceedings of the 2nd International Workshop on Artificial Intelligence and NetMedicine*.

Shehory, O. und Sturm, A., editors (2014). *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*. Springer, Berlin.

Slaney, J. und Thiébaux, S. (2001). Blocks World Revisited. *Artificial Intelligence*, 125(1-2):119–153.

Smirek, L., Zimmermann, G., und Ziegler, D. (2014). Towards Universally Usable Smart Homes – How Can MyUI, URC and openHAB Contribute to an Adaptive User Interface Platform? In *CENTRIC 2014 : The Seventh International Conference on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services*, pages 29–38, Nice, France.

Snyder, C. (2003). *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*. Interactive Technologies Series. Morgan Kaufmann.

Sommaruga, L., Formilli, T., und Rizzo, N. (2011). DomoML: An Integrating Devices Framework for Ambient Intelligence Solutions. In *Proceedings of the 6th International Workshop on Enhanced Web Service Technologies*, pages 9–15, New York, NY, USA. ACM.

Sommaruga, L., Perri, A., und Furfari, F. (2005). DomoML-env: an ontology for Human Home Interaction. In Bouquet, P. und Tummarello, G., editors, *SWAP*. CEUR-WS.org.

Spanoudakis, N. und Moraitis, P. (2008). An ambient intelligence application integrating agent and service-oriented technologies. In *Research and Development in Intelligent Systems XXIV*, pages 393–398. Springer London.

Stahl, C. (2009). *Spatial Modeling of Activity and User Assistance in Instrumented Environments*. PhD thesis, Saarland University, Dept. of Computer Science, Saarbrücken.

Stahl, C., Frey, J., Alexandersson, J., und Brandherm, B. (2011). Synchronized Realities. *Journal of Ambient Intelligence and Smart Environments*, 3(1):13–25.

Stahl, C. und Schwartz, T. (2010). Modeling and simulating assistive environments in 3-D with the YAMAMOTO toolkit. In *Proceedings of the 2010 International Conference on Indoor Positioning and Indoor Navigation*, pages 1–6. IEEE Xplore.

Steinberg, D., Budinsky, F., Paternostro, M., und Merks, E. (2009). *EMF: Eclipse Modeling Framework 2.0.* Addison-Wesley Professional, 2nd edition.

Sturm, A. und Shehory, O. (2014). The Evolution of MAS Tools. In Shehory, O. und Sturm, A., editors, *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*. Springer, Berlin.

Sun, Q., Yu, W., Kochurov, N., Hao, Q., und Hu, F. (2013). A Multi-Agent-Based Intelligent Sensor and Actuator Network Design for Smart House and Home Automation. *Journal of Sensor and Actuator Networks*, 2(3):557–588.

Thielscher, M. (2005). FLUX: A Logic Programming Method for Reasoning Agents. *Theory and Practice of Logic Programming*, 5(4-5):533–656.

Tsujita, H., Tsukada, K., und Siio, I. (2008). SyncDecor: Communication Appliances for Couples Separated by Distance. In *Mobile Ubiquitous Computing, Systems, Services and Technologies, 2008. UBICOMM '08. The Second International Conference on*, pages 279–286.

Vogel, L. (2013). *Eclipse 4 RCP: The Complete Guide to Eclipse Application Development.* Vogella series. Vogella.

Wahlster, W. (2006). Dialogue Systems Go Multimodal: The SmartKom Experience. In Wahlster, W., editor, *SmartKom: Foundations of Multimodal Dialogue Systems*, pages 3–27. Springer.

Wahlster, W. (2008). SmartWeb — Ein multimodales Dialogsystem für das semantische Web (2004–2007). In *Informatikforschung in Deutschland*, pages 300–311.

Wahlster, W. (2013). The Semantic Product Memory: An Interactive Black Box for Smart Objects. In Wahlster, W., editor, *SemProM: Foundations of Semantic Product Memories for the Internet of Things*, pages 3–21. Springer Berlin Heidelberg.

Wahlster, W. (2014). Semantic Technologies for Mass Customization. In Wahlster, W., Grallert, H.-J., Wess, S., Friedrich, H., und Widenka, T., editors, *Towards the Internet of Services: The THESEUS Program*, pages 3–13. Springer International Publishing.

Wahlster, W. und Kobsa, A. (1986). Dialog-based User Models. *Journal Proceedings of the IEEE, Special Issue on Natural Language Processing*, pages 948–960.

Walls, C. (2009). *Modular Java: Creating Flexible Applications with OSGi and Spring.* Pragmatic Bookshelf, Raleigh, NC.

Wang, X., Wong, T. N., und Wang, G. (2012). Service-oriented architecture for ontologies supporting multi-agent system negotiations in virtual enterprise. *J. Intelligent Manufacturing*, 23(4):1331–1349.

Wang, Y., Song, M., und Song, J. (2010). An extended distributed OSGi architecture for implementation of SOA. In *Advanced Intelligence and Awarenss Internet (AIAI 2010), 2010 International Conference on*, pages 416–419.

Weiser, M. (1991). The Computer for the 21st Century. *Scientific American*, 265(3):66–75.

Weiss, G. (2000). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press.

Wemlinger, Z. und Holder, L. (2011). The COSE Ontology: Bringing the Semantic Web to Smart Environments. In *Proceedings of the 9th International Conference on Toward Useful Services for Elderly and People with Disabilities: Smart Homes and Health Telematics*, pages 205–209, Berlin, Heidelberg. Springer-Verlag.

Wilsker, B. (1996). A Study of Multi-Agent Collaboration Theories. In *ISI Research Report*.

Winikoff, M. (2005). Jack^{TM} Intelligent Agents: An Industrial Strength Platform. In Bordini, R., Dastani, M., Dix, J., und El Fallah Seghrouchni, A., editors, *Multi-Agent Programming*, pages 175–193. Springer US.

Wooldridge, M. (2002). Intelligent Agents: The Key Concepts. In *Proceedings of the 9th ECCAI-ACAI/EASSS 2001, AEMAS 2001, HoloMAS 2001 on Multi-Agent-Systems and Applications II-Selected Revised Papers*, pages 3–43, London, UK, UK. Springer-Verlag.

Wooldridge, M. (2009). *An Introduction to Multiagent Systems*. Wiley, Chichester, UK, 2 edition.

Wooldridge, M. und Jennings, N. R. (1995). Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10:115–152.

Zimmermann, G., Alexandersson, J., Buiza, C., Urdaneta, E., Diaz, U., Carrasco, E., Klima, M., und Pfalzgraf, A. (2010a). Meeting the Needs of Diverse User Groups: Benefits and Costs of Pluggable User Interfaces in Designing for Older People and People with Cognitive Impairments. In *Intelligent Technologies for Bridging the Grey Digital Divide*, pages 80–93. IGI Global.

Zimmermann, G., Alexandersson, J., Buiza, C., Urdaneta, E., Diaz, U., Carrasco, E., Klima, M., und Pfalzgraf, A. (2010b). Meeting the Needs of Diverse User Groups: Benefits and Costs of Pluggable User Interfaces in Designing for Older People and People with Cognitive Impairments. In *Intelligent Technologies for Bridging the Grey Digital Divide*, pages 80–93. IGI Global.

Zimmermann, G., Jordan, J. B., Thakur, P., und Gohil, Y. (2013). GenURC: Generation Platform for Personal and Context-driven User Interfaces. In *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility*, pages 6:1–6:4, New York, NY, USA. ACM.

Zimmermann, G., Jordan, J. B., Thakur, P., und Gohil, Y. (2014). Abstract User Interface, Rich Grouping and Dynamic Adaptations – A Blended Approach for the Generation of Adaptive Remote Control User Interfaces. *Assistive Technology: From Research to Practice*, 33:1289–1297.

Zimmermann, G. und Vanderheiden, G. (2007). The Universal Control Hub: An Open Platform for Remote User Interfaces in the Digital Home. In Jacko, J., editor, *Human-Computer Interaction. Interaction Platforms and Techniques*, pages 1040–1049. Springer Berlin Heidelberg.

Zimmermann, G. und Vanderheiden, G. (2010). A dream... The Universal Remote Console . *ISO Focus+*, pages 11–13.

Zimmermann, G. und Wassermann, B. (2009). Why We Need a User Interface Resource Server for Intelligent Environments. In *Intelligent Environments (Workshops)*, pages 209–216.