Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Dennis Hollatz

Entwicklung einer nachrichtenbasierten Architektur für Smart Homes

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Dennis Hollatz

Entwicklung einer nachrichtenbasierten Architektur für Smart Homes

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Informatik (Master of Science)
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

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

Abgegeben am 28. Juni 2010

Dennis Hollatz

Titel der Arbeit

Entwicklung einer nachrichtenbasierten Architektur für Smart Homes

Stichworte

Ambient Intelligence, Asynchrone Kommunikationsmodelle, Complex Event Processing, Middleware, Model Driven Architecture, Pervasive Computing, Smart Homes, Software Engineering, Ubiquitous Computing

Zusammenfassung

Anwendungen im Rahmen von Smart Homes erfordern ein hohes Maß an Dynamik, Verlässlichkeit und Flexibilität. Sie sollen dem Nutzer eine sichere, nahtlose Interaktion ermöglichen, ohne ihn mit technologiespezifischen Eigenheiten abzulenken. Diese ambitionierten Anforderungen an das Gesamtsystem stellen allerdings auch große Anforderungen an die Anwendungsentwickler einzelner Teilkomponenten. Im Rahmen dieser Arbeit werden die Anforderungen an eine Softwarearchitektur eines Smart Homes am Beispiel des Living Place Hamburg untersucht und darauf aufbauend ein Architekturvorschlag erarbeitet. Das Ziel der Architektur ist es, die effiziente Implementierung flexibler und leichtgewichtiger Dienste zu ermöglichen. Der Entwicklungsprozess soll somit von komplexen und fehleranfälligen Infrastrukturanpassungen befreit werden.

Das „Living Place Hamburg“ ist ein Labor, in dem zukünftig praxisnahe Realexperimente und Langzeit-Usability-Studien auf dem Gebiet der Smart Homes durchgeführt werden. Das Labor befindet sich auf dem Campus am Berliner Tor der Hochschule für Angewandte Wissenschaften Hamburg. Mit der Fertigstellung des ersten Bauabschnitts (voraussichtlich im Herbst 2010) wird das Living Place Hamburg eine ca. 140 qm große Laborwohnung, sowie zusätzlich ca. 100 qm administrative Arbeitsflächen bieten.

Dennis Hollatz

Title of This Paper

Design of a Message Oriented Architecture for Smart Homes

Keywords

Ambient Intelligence, Asynchronous Communication, Complex Event Processing, Middleware, Model Driven Architecture, Pervasive Computing, Smart Homes, Software Engineering, Ubiquitous Computing

Abstract

Software infrastructures for Smart Homes are required to provide a very high level on viability, reliability, and flexibility. Users expect to be able to seamlessly interact with a set of various cooperating devices, while aspects of technological heterogenity step behind. Thus building applications that deal with these requirements, tend to bear enormous challenges in the application development process. This thesis analyses these challenges and presents a software architecture for a message based system that is capable of unburdening application developers from complex, time consuming and error-prone implementation tasks that deal with these complex infrastructure aspects.

Living Place Hamburg is a laboratory at the University of Applied Sciences in Hamburg, Germany. It offers the possibilities to a broad variety of research topics and is intended to act as a testbed and showroom for applications developed by the related iFlat laboratory. Living Place Hamburg is designed as a regular apartment enhanced with high-end technology. It offers around 140 sqm of living space and about 100 sqm of office space.

Inhaltsverzeichnis

Abbildungsverzeichnis **8**

Tabellenverzeichnis **9**

1 Motivation **10**
1.1 Die Vision eines Smart Homes . . . 11
1.2 Zielsetzung und Abgrenzung . . . 12
1.3 Herausforderungen des Software Engineering . . . 13
1.4 Herangehensweise und Aufbau dieser Arbeit . . . 14

2 Ubiquitous Computing und Smart Homes **16**
2.1 Ubiquitous Computing . . . 16
2.2 Pervasive Computing . . . 19
2.3 Ambient Intelligence . . . 25
2.4 Smart Environments und Smart Homes . . . 28
2.5 Diskussion . . . 32

3 Systemanalyse im Living Place Hamburg **33**
3.1 Problemstellung . . . 33
3.2 Living Place Hamburg . . . 34
3.2.1 iFlat . . . 35
3.2.2 Living Place . . . 37
3.3 Softwareinfrastrukturen für Ambient Intelligence . . . 39
3.3.1 DynAMITE . . . 40
3.3.2 Oxygen . . . 41
3.3.3 Personal Information Environments . . . 41
3.3.4 Agentenbasierte Systeme . . . 43
3.3.5 EventHeap . . . 44
3.3.6 Diskussion . . . 48
3.4 Szenariobasierte Analyse . . . 48
3.4.1 Wecker 2.0 . . . 48
3.4.2 Gegensprechanlage . . . 50
3.4.3 Cooking Agent . . . 52

3.4.4 Zusammenfassung der Szenarien 54
3.5 Anforderungen aus Entwicklersicht 55
3.5.1 Kommunikationsrelevante Anforderungen 55
3.5.2 Konfigurationsrelevante Anforderungen 57
3.5.3 Sicherheitsrelevante Anforderungen 58
3.5.4 Nichtfachliche Anforderungen 59
3.6 Zielsetzung: Eine Middleware für Smart Homes 59

4 Das Smart Home als verteiltes System 62
4.1 Modellierung nebenläufiger Systeme 63
4.2 Middleware in verteilten Systemen 64
4.2.1 Einordnung 65
4.2.2 Middleware als technologische Voraussetzung 70
4.2.3 Einschränkungen für den Einsatz von Middleware 71
4.2.4 Zusammenfassung 71
4.3 Complex Event Processing 72
4.3.1 Anwendungsgebiete 72
4.3.2 Definitionen und Begriffsabgrenzung 73
4.3.3 Ereignisanfragen 77
4.3.4 Erzeugung von Events 79
4.3.5 Ereignisabfragesprachen 79
4.3.6 Zusammenfassung 81
4.4 Fazit 81

5 Systementwurf einer nachrichtenbasierten Architektur für Smart Homes 83
5.1 Vorüberlegungen 83
5.1.1 Systemorganisation 83
5.1.2 Abstraktionsebenen 84
5.1.3 Unterstützte Aufgaben 85
5.1.4 Aktormodell und Software-Agenten 86
5.1.5 Diskussion 87
5.2 Vision: Eine agentenbasierte Basisarchitektur für Smart Homes 88
5.3 Grobkonzeption der Middlewarekomponente 90
5.4 Zentrale Architekturentscheidungen 91
5.4.1 Anlehnung an das Tuplespaces-Architekturmodell 92
5.4.2 Nachrichtenbasierte Kommunikation nach dem Aktormodell 93
5.4.3 Peer-to-Peer im koodinierten Overlay-Netzwerk 93
5.4.4 Separation of Concerns 96
5.5 Architekturentwurf 97
5.6 Möglichkeiten zur technischen Realisierbarkeit 100
5.6.1 Aufbau der Laufzeitumgebung 100

5.6.2 Integration des Message Service 102
5.6.3 Veröffentlichen und Abonnieren von Events 102

6 Schlussbetrachtung **104**
6.1 Zusammenfassung 104
6.2 Mögliche erste Schritte einer iterativen Entwicklung 105
6.3 Fazit 106

Literaturverzeichnis **108**

Abbildungsverzeichnis

2.1 Eine alte Sicht auf intelligente Systeme . . . 29
2.2 Eine neue Sicht auf intellligente Systeme . . . 29

3.1 Einflussbereiche des Living Place Hamburg . . . 34
3.2 Zusammenspiel der Labore Livingplace Hamburg und iFlat . . . 35
3.3 Grundriss des Living Place Hamburg . . . 38
3.4 Impressionen des Living Place Hamburg . . . 39
3.5 DynAMITE – Demo-Applikation . . . 40
3.6 Überblick Oxygen . . . 41
3.7 Personal Information Environment – Mindmap . . . 42
3.8 iROS Komponenten . . . 45
3.9 Schematische Darstellung einer Blackboard-Architektur . . . 46
3.10 Grundprinzipien von Tuplespaces . . . 47
3.11 Aktivitätsdiagramm: Gegensprechanlage . . . 51
3.12 Situationen im Szenario Gegensprechanlage . . . 51
3.13 Sequenzdiagramm: Cooking Agent . . . 53
3.14 Concept-Map der Szenarien . . . 54

4.1 Middleware zur Maskierung der Heterogenität des Betriebssystems . . . 66
4.2 Middleware zur Maskierung der Verteilung eines Systems . . . 66
4.3 Klassifizierung von Events . . . 75
4.4 Unterschiede zwischen Datenbank- und Ereignisstromanfragen . . . 76
4.5 Event Operators . . . 78

5.1 Systemorganisation als Smart Environment . . . 88
5.2 Vereinfachtes Komponentendiagramm einer Middlewarearchitektur für Smart Homes . . . 91
5.3 Vereinfachtes Sequenzdiagramm eines Handshake zwischen Peer und Middleware . . . 92
5.4 Kommunikation im Peer-to-Peer-Netz ohne Koordinator . . . 94
5.5 Kommunikation im Peer-to-Peer-Netz mit Koordinator . . . 94
5.6 Kommunikation im Peer-to-Peer-Mesh-Netz (über Koordinatoren) . . . 96

5.7 Erweitertes Komponentendiagramm einer Middlewarearchitektur für Smart Homes 98

Tabellenverzeichnis

2.1 Merkmale und Herausforderungen von Pervasive Computing 21
2.2 Schwerpunktthemen von Pervasive Computing und Ambient Intelligence . . 25

3.1 Übersicht studentischer Projekte mit Bezug zum iFlat 36

1 Motivation

Bereits Anfang der 1990er Jahre formulierte Weiser [1991] seine Vision von allgegenwärtigen und unsichtbaren Computern:

> *„Hundreds of computers in a room could seem intimidating at first, just as hundreds of volts coursing through wires in the walls did at one time. But like the wires in the walls, these hundreds of computers will come to be invisible to common awareness. People will simply use them unconsciously to accomplish everyday tasks. “* – [Weiser, 1991]

Diese Prognose, die damals eher an Science-Fiction grenzte, rückt heute in immer greifbarere Nähe. Die Entwicklung wird beispielsweise in Form der zunehmenden Verbreitung von Smartphones und PDAs deutlich. Financial Times Deutschland berichtet, dass nach einer gemeinsamen Studie der Unternehmen Google und Otto der Anteil der Smartphonebesitzer an der gesamtdeutschen Bevölkerung ca. elf Prozent beträgt. Für die nächsten zwei Jahre wird sogar ein Anstieg auf bis zu 25 Prozent erwartet [Maatz, 2010].

Analog zu diesen Entwicklungen zeigt sich, dass der Merkt für *Ambient Intelligence* (s. Abschnitt 2.3) von der Industrie immer ernster genommen wird. Beispielsweise veranstaltete der Industrieverband Bitkom[1] im Mai 2010 zum ersten Mal der Fachkongress „ConLife Conference & Exhibition“. Die Veranstaltung richtete sich nach eigenen Angaben vornehmlich an Besucher „die sich beruflich mit den Themen Heimvernetzung und -automatisierung sowie Connected Life-Lösungen beschäftigen oder beschäftigen werden“[2]. In der Pressemeldung zur Ankündigung von ConLife gibt Bitkom eine geschätzte Verkaufszahl von ca. zwei Millionen intenetfähigen Fernsehern für das Jahr 2010 an [BITKOM e.V., 2010]. Im Zusammenhang mit diesen Entwicklungen, warnt Weber u. a. [2005a] allerdings davor, dass Ambient Intelligence auch das Potential habe, sich zu einer *self-fulfilling Prophecy* (einer sich selbst erfüllenden Prophezeiung) zu entwickeln.

[1] Bitkom: Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V.

[2] s. Website der Veranstaltung: `http://conlife-cologne.de/`, Abrufdatum: 20.06.2010

1.1 Die Vision eines Smart Homes

Die Entwicklungen im Bereich der Smart Homes zeigen eine verstärkte Verfügbarkeit leistungsfähiger Geräte, es fehlt allerdings an akzeptierten offenen Infrastrukturstandards, die eine Integration der intelligenten Einzelgeräte fördern. Es kommen mehr und mehr Geräte auf den Markt, die zwar untereinander physisch vernetzt werden können oder die Fähigkeit besitzen über das Internet zu kommunizieren, denen es allerdings an Fähigkeiten zur Interaktion mit anderen Geräten im lokalen Umfeld mangelt. Es handelt sich dabei im Consumer-Bereich beispielsweise um Spielekonsolen, Fernseher oder Internetradios oder außerhalb des Consumer-Bereichs um zentrale Heizungs- und Lichtsteuerungen oder auch Digitale Stromzähler. Intelligente Haushaltsgeräte, wie Waschmaschinen, Spülmaschinen, Kaffeemaschinen oder auch Kühlschränke tauchen ebenfalls vermehrt auf dem Endverbrauchermarkt auf. Die Vernetzung der einzelnen Geräte begrenzt sich häufig auf einen herstellerspezifischen einfachen Kommunikationsbus. Es fehlt ein Standard, der es beliebigen Geräten ermöglicht, miteinander in Verbindung zu treten und so ein *Netz der Dinge* formen zu können Norbisrath [2007].

Im Rahmen des Living Place Hamburg zeigt Urich [2009] den Nutzen einer möglichen Anwendung für die Vernetzung intelligenter Haushaltsgeräte zur Erstellung eines Menüplanes. Der Prototyp (*Cooking Agent*) zeigt, wie der Benutzer bei der Suche nach einem passenden Kochrezept unterstützt werden kann. Bei der Rezeptermittlung werden u. a. der Kühlschrankinhalt und die Vorlieben der Gäste beachtet.

Gregor u. a. [2009] beschreiben die grundlegenden Eigenschaften des Living Place Hamburg als ein konkretes Smart Home, das an der Hochschule für Angewandte Wissenschaften Hamburg eingerichtet wird. Es handelt sich dabei um eine intelligente Wohnung, in der ein sehr hoher Vernetzungsgrad zwischen möglichst vielen Geräten in der Wohnung bestehen soll. Ein hoher Grad der Vernetzung bewirkt allerdings auch eine hohe Komplexität der Anwendungen. Die Bereitstellung einer entsprechenden Infrastruktur kann helfen, die Komplexität zu senken. Dabei ist es wichtig, dass Infrastruktur sowohl über die Leistungsfähigkeit der einzelnen Geräte, als auch mit deren steigender Anzahl skaliert. Der in Hollatz [2008a,b] beschriebene und für die in Hollatz [2008c] und Urich [2009] beschriebenen Anwendungen erfolgreich eingesetzte Event Heap ist ein erster Schritt in diese Richtung.

Der Einsatz des Event Heap in einer derart komplexen Umgebung birgt allerdings einige Nachteile. Er ist einerseits als Middleware-Komponente eines intelligenten Arbeitsraumes ausgelegt und daher nur für eine relativ kleine Anzahl von angeschlossenen Geräten in einem Konferenzraum ausgelegt, es fehlen daher Funktionalitäten, wie Persistenz oder Lastverteilung. Die Kommunikation ist in der vorliegenden Realisierung auf Java-basierte Geräte beschränkt. Diese Einschränkung betrifft derzeit insbesondere Embedded Devices, die nicht die Leistungsfähigkeit zum Betrieb einer Java Virtual Machine (JVM) haben. Eckert

und Bry [2009] geben einen Überblick des Themengebiets Complex Event Processing, das es ermöglichen soll, Schlussfolgerungen aus einem Strom von Events zu ziehen, um darauf basierend entsprechende Aktionen auslösen zu können. Die Erkennung solch komplexer Ereignisse sollte ebenfalls eine Kernfunktionalität der Middleware sein, die angeschlossenen Anwendungen sollen über eine Abfragesprache (ähnlich der SQL) den Event-Strom filtern können.

1.2 Zielsetzung und Abgrenzung

Es soll im Rahmen dieser Arbeit eine Architektur entwickelt werden, auf deren Grundlage die Infrastruktur eines Smart Homes aufgebaut und betrieben werden kann. Die Architektur soll dabei flexibel genug gestaltet werden, so dass sie für den Laborbetrieb geeignet ist und den zu erwartenden massiven Veränderungen der Anforderungen standhalten kann.

Der Architekturentwurf soll dabei den Einsatz einer zentralen Middleware vorsehen, die die asynchrone nachrichtenbasierte Kommunikation der Endgeräte innerhalb des Systems unterstützt. Ähnlich zu dem in Johanson [2002] vorgestellten und in Hollatz [2007, 2008a,b] näher evaluierten iROS soll somit eine zentrale Komponente die Koordination und Vermittlung von Nachrichten übernehmen. Der Sender übergibt eine Nachricht ungerichtet an die Middleware, die sie daraufhin an die von ihr identifizierten Empfänger weiterleitet. Die Auswahl der Empfänger soll dabei auf der Grundlage flexibler Regeln geschehen, die die Empfänger der Middleware selbst mitteilen können. Auf diese Weise sollen die angeschlossenen Teilnehmer von den aufwendigen Aufgaben der Adressierung an Empfänger entlastet werden.

Die auf der Grundlage der Erkenntnisse aus dieser Arbeit angestrebte Entwicklung einer Middleware soll primär die Vernetzung von intelligenten Geräten ermöglichen. Die Aufgaben der Middleware umfassen dabei lediglich die Weiterleitung von Nachrichten. Die Kontrolle von Geräten, sowie die Verarbeitung von Daten soll ausschließlich an den Endpunkten des Systems geschehen. Somit werden typische Aufgaben, wie beispielsweise die Lichtsteuerung in einem Haushalt, von den betreffenden Schaltern und Lampen behandelt. Die Middleware übernimmt in diesem Zusammenhang lediglich die Übermittlung der Schaltereignisse am Lichtschalter an die zugehörigen Lampen.

Es wird Arbeit eine umfassende Analyse der Anforderungen im Sinne des Software Engineering vorgestellt. Anhand der Ergebnisse wird eine Architektur entwickelt, die den o. g. Rahmenbedingungen entspricht. Der Schwerpunkt dieser Arbeit liegt auf der Analyse und dem Entwurf eines Architekturvorschlags, um eine breite Grundlage für die nachfolgenden Implementierung der Architekturkomponenten zu schaffen. Es kann daher nicht das Ziel dieser Arbeit sein, zusätzlich zum Entwurf eine praktische Evaluierung der Architektur durchzuführen, da diese den Rahmen der Arbeit deutlich übersteigen würde. Um die technische

Realisierbakeit dennoch abschätzbar zu machen, wird im Rahmen der Vorstellung des Designs auch eine Abschätzung bezüglich der technologischen Realisierbarkeit der Anwendung abgegeben.

1.3 Herausforderungen des Software Engineering

Die Entwicklung einer Softwarearchitektur ist eine Disziplin des Software Engineering. Aus diesem Grund wird hier eine kurze Einordnung des Begriffs, sowie einige Ergänzungen gegeben, die für diese Arbeit gelten sollen.

Eine allgemeine Definition des Begriffs Software-Engineering nach Kahlbrandt [2005] lautet folgendermaßen:

> *„Software-Engineering ist die Entwicklung, die Pflege und der Einsatz qualitativ hochwertiger Software unter Einsatz von wissenschaftlichen Methoden, wirtschaftlichen Prinzipien, geplanten Vorgehensmodellen, Werkzeugen und quantifizierbaren Zielen.“* – [Kahlbrandt, 2005, S. 5f.]

Es umfasst somit den vollständigen Lebenszyklus einer Anwendung. Jeder einzelne Schritt dieses Vorgehens muss daher angemessen vorbereitet und geplant sein, damit diese Herausforderungen des Software Engineering gelöst werden können.

In Ergänzung zu den klassischen Aspekten des Software Engineering stellt Paspallis [2009] weitere Aspekte vor, die für die Entwicklung von Anwendungen im Bereich „Ubiquitous Computing“ gelten sollen.

Komplexität. Die Handhabung der Komplexität einer Anwendung stellt eine der größten Herausforderungen an das Software Engineering verteilter Systeme im Rahmen des Ubiquitous Computing dar. Die Heterogenität der eingesetzten Geräte, ihre Möglichkeiten der Interaktion, die unterschiedlichen Anforderungen im Hinblick auf den Kontext, sowie deren Benutzung durch den Benutzer werden als die Hauptgründe für die gesteigerte Komplexität gesehen.

Heterogenität und Interoperabilität. Ubiquitäre Systeme sind per Definition angepasst auf den Einsatz jenseits des Schreibtischs. Der Einsatz von mobilen und spezialisierten Geräten und Softwarelösungen führt dazu, dass ein gesteigerter Bedarf an Kommunikations- und Interoperabilitätsanforderungen gedeckt werden muss.

Eingeschränkte Ressourcenverfügbarkeit. Bedingt durch die weitere Verbreitung eingebetteter und mobiler Geräte ergibt sich, dass diese Geräte oftmals mit deutlich beschränkten Ressourcen umgehen können müssen. Es kann sich hierbei um die Größe der verfügbaren Bildschirmauflösung, den zur Verfügung stehenden Arbeits- und Festspeicher, aber auch um eingeschränkte Energiereserven handeln.

Modularität. Die letztgenannten zwei Anforderungen führen zur Forderung nach modular aufgebauten Architekturen und Systemen. Die Gestaltung ubiquitärer Systeme, die auf der Grundlage vieler höchst heterogener Geräte, Anwendungen und Protokolle basiert, stellt umfangreiche Anforderungen an die zugrundeliegende Architektur. In einer solchen Umgebung wird es von Paspallis als unwahrscheinlich angesehen, dass es eine Universallösung gibt, die sich auf alle Szenarien anwenden lässt.

1.4 Herangehensweise und Aufbau dieser Arbeit

Die vorliegende Arbeit gliedert sich in sechs Kapitel und orientiert sich so an den breit gefächerten Anforderungen, die an die Architekturentwicklung eines Smart Homes gestellt werden.

In Kapitel 2 wird die Entstehung des Konzepts Smart Home vorgestellt. Ausgehend von Mark Weisers Visionen des Ubiquitous Computing, werden Weiterentwicklungen dieses Konzepts unter den Überschriften Pervasive Computing und Ambient Intelligence vorgestellt. Im Kontext von Ambient Intelligence wird schließlich auf die Entstehung und Bedeutung des Begriffs Smart Home eingegangen und es werden einige Beispiele konkreter Arbeiten auf diesem Gebiet gegeben.

Um die Möglichkeit einer fundierten Entwicklung zu schaffen, wird in Kapitel 3 eine umfassende Analyse der bestehenden Umgebung, Vorleistungen und Anforderungen an die Architektur vorgenommen. In diesem Kapitel erfolgt auch eine detaillierte Beschreibung der beteiligten Labore iFlat und Living Place, sowie eine Vorstellung verwandter Ansätze. Ausgehend von diesen Grundlagen wird eine szenarienbasierte Anwendungsanalyse durchgeführt, auf deren Grundlage die Identifikation der Anforderungen an die Architektur vorgenommen wird.

In Kapitel 4 folgt vorbereitend für das Architekturdesign die Einordnung eines Smart Homes in das Gebiet der verteilten Systeme. Diesbezüglich werden hier zunächst die allgemeinen Anforderungen an das Design verteilter Systeme vorgestellt und daraufhin Lösungsvorschläge für die wichtigsten Herausforderungen eines Smart Homes vorgestellt: Modellierung von Nebenläufigkeit, Umgang mit Heterogenität, sowie das Erkennen und Verarbeiten von komplexen Ereignissen.

Das Architekturdesign baut auf der Zusammenführung der Ergebisse der vohergehenden Kapitel auf. In Kapitel 5 wird der Aufbau der Architektur in Form eines Systementwurfs beschrieben, sowie Hinweise zur technischen Realisierbarkeit gegeben. Eine ausführliche Diskussion der Ergebnisse, die auch einen Ausblick auf zukünftige Entwicklungen mit einschließt, wird in Kapitel 6 vorgenommen.

2 Ubiquitous Computing und Smart Homes

Das Forschungsgebiet *Smart Homes* bildet ein noch relativ junges Forschungsgebiet. Erste Veröffentlichungen zu diesem Thema finden sich in den frühen 2000er-Jahren [z. B. Tolmie u. a., 2002], eines der ersten praktischen Forschungsprojekte stellt das Philips HomeLab dar, welches ebenfalls in dieser Zeit eingerichtet wurde [vgl. Aarts u. a., 2003]. Zuvor existierende Ansätze für intelligente Häuser beschränkten sich in der Regel auf einfache Heimautomation, bzw. zentrale Steuerung der Heizungsanlage, Fenster, Licht und Alarmanlagen. Erst in den letzten Jahren wurden vermehrt die Ideen des Ubiquitous Computing (Abschnitt 2.1) und der Ambient Intelligence (Abschnitt 2.3) mit einbezogen.

In diesem Abschnitt werden die relevanten Forschungsgrundlagen, sowie bedeutende Entwicklungen auf vergleichbaren Gebieten vorgestellt. Es soll so ein objektiver Einstieg in das Thema Smart Homes ermöglicht werden. Dieses Kapitel orientiert sich dabei an der historischen Entwicklung aus den Ideen des Ubiquitous Computing bis hin zu Pervasive Computing und Ambient Intelligence. Die vorgestellten Inhalte sollen ein verbessertes Verständnis der späteren Überlegungen von Systemanalyse (Kapitel 3) und Systemdesign (Kapitel 5) ermöglichen.

2.1 Ubiquitous Computing

Die Wahrnehmung von Computern hat sich in den letzten Dekaden stark gewandelt. So war es in den 60er und 70er Jahren noch eine weit verbreitete Meinung, dass mit fortschreitender Entwicklung die künstliche Intelligenz die natürliche ersetzen würde. Die Untersuchungen auf dem Gebiet der künstlichen Intelligenz waren zudem stark von der Idee geprägt, dass die Prinzipien künstlicher und natürlicher Intelligenz eng miteinander verknüpft seien. Es wurde angenommen, dass gewonnene Erkenntnisse auf einem dieser Gebiete so auch zu Fortschritten auf dem jeweils anderen Gebiet führen würden. Spätere Forschungsprojekte haben sich mehr und mehr von dieser Sichtweise abgewandt und der Erforschung künstlicher Intelligenz damit einen eigenen Stellenwert zugesprochen. In Winograd und Flores [1990] werden dieser Wandel beschrieben und die Möglichkeiten präsentiert, wie Computer

in diesem Zusammenhang sinnvoller eingesetzt werden können. Dabei werden eine Reihe von Punkten herausgestellt, die eine angemessene Interaktion von Mensch und Computer ermöglichen sollen.

Die im Kontext dieser Arbeit wichtigsten Punkte werden im Folgenden kurz dargestellt:

Readiness-to-Hand. Winograd und Flores beschreiben diesen Punkt als ein wichtiges Merkmal bei der Nutzung von Werkzeugen (im Allgemeinen). Ein Hammer wird von jemandem, der es gewohnt ist, diesen zu benutzen, nicht als Fremdkörper wahrgenommen. Auch wird die Person nicht darüber nachdenken, wie dieser benutzt werden muss, um einen Nagel in die Wand zu bekommen, sie wird ihn einfach benutzen.

Ähnlich verhält es sich mit Computerprogrammen. Bei der Benutzung einer Textverarbeitung denkt der Benutzer nicht über jeden einzelnen Tastendruck nach, sondern formuliert einen Text in Sätzen und Abschnitten. Das Werkzeug an sich wird erst wahrgenommen, wenn es nicht so funktioniert, wie es sollte. Im Alltag kann es sich dabei z. B. um einen defekten oder verschwundenen Hammer handeln. Analog wird auch ein Computerprogramm, mit dem man täglich arbeitet, erst wieder aktiv wahrgenommen, wenn es abstürzt oder eine für den Anwender unverständliche Nachricht ausgibt.

Social Embedding. Das Anpassen von Computersystemen an die entsprechende Zielgruppe stellt für Winograd und Flores eine Schlüsselrolle für den Erfolg desselben dar. Viele Programme seien technisch sehr weit fortgeschritten, allerdings nicht erfolgreich, da sie nicht in den Kontext passen, für den sie geschaffen wurden [Winograd und Flores, 1990, S. 83ff.].

Entscheidungsfindung und -unterstützung. Die automatische Entscheidungsfindung und die Entscheidungsunterstützung werden von Winograd und Flores als problematisch angesehen, da der Einsatz von Computersystemen, die solche Dienste anbieten, oft auf falschen Annahmen beruhe. Beispiele hierfür sind die *Wahl der Ausrichtung* (S. 153), *Annahmen über Relevanz* (S. 153f.), *unbeabsichtigte Verlagerung von Macht* (S. 154), *unerwartete Seiteneffekte* (S. 154f.), *Verschleierung von Verantwortung* (S. 155f.), sowie *falscher Glaube in die Objektivität des Systems* (S. 156f.).

Winograd und Flores [1990] rücken das Bild der Computertechnologie in eine andere Perspektive. Es geht hier nicht um die Funktionalitäten, die ein einzelner Computer haben kann. Der Fokus liegt viel mehr auf den Möglichkeiten, die jeweiligen Stärken von Menschen und Computern zu nutzen, um somit die Produktivität des Einzelnen zu steigern. Diese Überlegungen lassen sich durch die Ideen in Weiser [1991] anreichern, in dem Weiser seine Gedanken zum *Ubiquitous Computing* formuliert. In Weiser und Brown [1995] wird hierauf aufbauend die Idee unaufdringlicher Technologien als ein Teilaspekt des *Ubiquitous Computing* formuliert.

Weiser [1991] nennt ein kurzes Beispielszenario, das anhand des Tagesablaufs einer fiktiven Person *Sal* die Möglichkeiten des *Ubiquitous Computing* verdeutlichen soll (durch den Autor auf die für diese Arbeit wesentlichen Punkte gekürzt):

> Sal interagiert während eines typischen Tages nahezu ununterbrochen mit intelligenter Technologie, wie ihrem Wecker, der sie in einer Wachphase unaufdringlich fragt, ob sie Kaffee möchte. Am Frühstückstisch liest sie die Papierversion der Tageszeitung. Trotz des nicht-elektronischen Formats ist sie in der Lage, einen Artikel mit ihrem Stift zu markieren, um diesen als Email zu erhalten. Ihr Büro ist virtuell mit dem Büro ihres Kollegen verknüpft, so dass zwischen beiden eine natürliche Interaktion möglich ist, ohne dass sie sich physisch im selben Raum befinden müssen. Informationen sind grundsätzlich zugänglich und leicht zu beschaffen. Sal ist z. B. auf der Suche nach einer Kollegin, die sie vor mehreren Wochen bei einem Meeting getroffen hat. Sie kann sich zwar nicht mehr an den Namen der Person erinnern, ist aber dennoch in der Lage, diese Information aus den Teilnehmerlisten der vergangenen Meetings abzurufen. Sie erhält zusätzlich Zugriff auf den beruflichen Hintergrund der Kollegin und kann alle notwendigen Kontaktdaten einsehen, da diese von der Kollegin entsprechend für die anderen Teilnehmer des Meetings freigegeben wurden. So kann Sal schließlich ohne größere Mühen mit ihr in Kontakt zu treten.

Weiser weist darauf hin, dass die beschriebene Form der individuellen Datensammlung einen hohen Anspruch an die Sicherheitsarchitektur des Systems stelle und somit von Anfang an in das Anwendungsdesign mit eingeflochten werden solle. Weiser vergleicht die fortschreitende Entwicklung der Verbreitung von Computern mit der des Radios oder der Entwicklung von Drucksystemen hin zu Desktop-Publishing-Lösungen. In beiden Fällen waren anfangs nur einige wenige und teure Geräte auf dem Markt. Im Laufe der Jahre stieg die Marktdurchdringung immer weiter an. Heutzutage gibt es in nahezu jedem Haushalt ein Radio und viele handelsübliche PCs werden bereits mit einer Textverarbeitungssoftware ausgeliefert.

Die Kombination der Schlussfolgerungen aus den Arbeiten zu den Themen *Ubiquitous* und *Calm Computing* [Weiser, 1991; Weiser und Brown, 1995] sind schon sehr nah an den Anforderungen, die an ein Smart Home gestellt werden. Insbesondere die nahtlose und unaufdringliche Integration von Technologie in unseren Alltag stellt noch immer eine große Herausforderung dar.

Computer Supported Collaborative Work (CSCW[1]) stellt ein wichtiges Einflussgebiet des Ubiquitous Computing dar. CSCW hat sich bereits Anfang der 1980er Jahre als ein eigenständiges Forschungsgebiet entwickelt [Greif, 1988, S. 5f.] und hat in seiner Natur der Sichtweise auf einen computergestützten Arbeitsplatz starken Einfluss auf die Strömungen des

[1] früher auch als *Computer Supported Cooperative Work* bezeichnet, vgl. Greif [1988]

Ubiquitous Computing. Im Rahmen der CSCW werden dabei die Möglichkeiten der computergestützten Interaktion von Individuen untersucht, um die Produktivität der Zusammenarbeit am selben oder auch an getrennten Orten zu steigern Chapanis [1988]. Die Untersuchungen sind dabei nicht nur technischer, sondern auch soziologischer Natur. Knoblauch und Heath [1999] stellen anhand von Studien zum Thema *Workplace Studies* das Verhalten von üblicherweise verteilt arbeitenden Teams vor. Es handelt sich hier um Teams, die z. B. die Sicherheit an Flughäfen regeln oder die die Einsätze von Ambulanzen in einem Stadtgebiet organisieren. Im Rahmen der Arbeit wurde herausgestellt, dass es ein häufiges Problem bei der Einführung von computergestützten Systemen sei, die Bedürfnisse der Arbeiter auch zu erfüllen. Häufig würde der Einsatz des Computers nur einen peripheren Teil der Arbeit abdecken. Das führt wiederum dazu, dass ergänzende (auch improvisierte) Mittel eingesetzt werden, um die Unzulänglichkeiten der Technik auszugleichen. Es handelt sich dabei beispielsweise um selbsterstellte Papierformulare, die als eine Art Laufzettel für den jeweiligen Vorgang mitgeführt werden. Der Computer wird in diesen Fällen nicht selten als eine zusätzliche Belastung wahrgenommen und nicht als die Arbeitserleichterung, als die er eigentlich gedacht ist.

Jüngere Projekte zum Thema CSCW befassen sich zunehmend mit der Arbeit in *Collaborative Workspaces*. Dieser Begriff wird häufig im Zusammenhang mit der Unterstützung von Kommunikations- und Arbeitsprozessen verstanden [Neumann, 2006] und wird mit sehr unterschiedlichen technischen Ansätzen zur Erreichung dieses Ziels wissenschaftlich untersucht [Russell u. a., 2005]. Die Arbeiten in diesem Umfeld gliedern sich dabei größtenteils in Infrastrukturuntersuchungen (z. B. Johanson und Fox [2002, 2004]; Fraunhofer IGD [2003]; Tandler [2004]) und Arbeiten, die sich zum Ziel setzen, eine natürliche Interaktion zwischen Mensch und Maschine zu erreichen (z. B. Hinrichs u. a. [2005]; Scott [2005]; Agarawala [2006]; Gregor u. a. [2009]; Isenberg u. a. [2009]).

2.2 Pervasive Computing

Die Begriffe *Ubiquitous Computing* und *Pervasive Computing* werden häufig synonym verwendet. Tatsächlich gehen beide Ansätze aus denselben Paradigmen hervor, wie sie von Weiser formuliert wurden. Der Unterschied zwischen *Ubiquitous Computing* und *Pervasive Computing* liegt daher laut Mattern [2001] viel mehr in der Akzentuierung der Schwerpunkte: Weisers Vision des allgegenwärtigen Computers stelle eine eher akademisch idealistische Sichtweise dar, die eine langfristige Entwicklung erfordert. Der Begriff *Pervasive Computing* wurde daher von der Industrie geprägt, um eine kurzfristige Realisierung von eindringender und allgegenwärtiger Informationsverarbeitung im Rahmen von eCommerce- und Web-basierten Szenarien zu erreichen. Diese Prägung zeigt sich v. a. auch in der Ausrichtung von

Veröffentlichungen in diesem Themengebiet, die meist den Anspruch erheben, praxisorientierte Lösungen präsentieren zu können.

Satyanarayanan [2001] sieht Pervasive Computing als Kombination und Weiterentwicklung verteilter und mobiler Systeme. Zusätzlich zu den für diese Systeme geltenden Anforderungen stellt Satyanarayanan vier weitere Herausforderungen auf, die für die Entwicklung pervasiver Systeme gelten: *effektive Nutzung intelligenter Lebensräume*, *Unsichtbarkeit*, *lokale Skalierbarkeit*, sowie *das Verbergen ungleicher Voraussetzungen*. Eine detailliertere Beschreibung findet sich in Hansmann u. a. [2003]. Dort werden die Design-Prinzipien des Pervasive Computing nach folgenden Gesichtspunkten dargestellt: *Dezentralisation*, *Diversifikation*, *Konnektivität* und *Einfachheit*. Die Bedeutung dieser Schlagworte soll im Folgenden kurz erläutert werden:

Dezentralisation. Während der Mainframe-Ära wurde die Hauptrechenleistung in einem verteilten System von einem sehr leistungsfähigen Supercomputer erbracht. Die Etablierung von PCs und Client-Server-Architekturen hat eine Verlagerung der Rechenkapazitäten und der Aufgabenverteilung zu den Clients bewirkt. Beim Pervasive Computing werden die Aufgaben über eine große heterogene Menge kleiner autonomer Geräte verteilt. Die Herausforderungen der Dezentralisation sind die Organisation und Synchronisation der verteilten Daten und Anwendungen.

Diversifikation. Pervasive Computing bewegt sich weg von dem Paradigma des universell einsetzbaren Computers, indem es für spezialisierte Aufgaben nicht nur spezialisierte Software, sondern auch spezialisierte Geräte fordert. Ein Benutzer wird dabei mehrere Geräte parallel verwenden, die sich zum Teil in ihren Funktionalitäten überlappen können. Hansmann u. a. nennen hier als Beispiel mehrere Wege, über die man Zugang zum Internet zu bekommen kann: wie z. B. unterwegs mit einem Smart Phone oder zuhause mit dem PC. Der Umgang mit den Fähigkeiten und Einschränkungen verschiedenster Geräte ist eine große Herausforderung, der sich Entwickler pervasiver Anwendungen stellen müssen. Die Unterschiede bestehen dabei nicht nur in der ungleichen Rechenleistung, sondern berühren auch Themen wie unterschiedliche Eingabemethoden, die Verfügbarkeit von Daten und die Internetverbindung.

Konnektivität. Konnektivität und Interoperabilität zwischen vielfältigen Geräten stellt nach Hansmann u. a. [2003] das dritte Paradigma des Pervasive Computing dar. Der nahtlose Austausch von Daten und die Verwendung von Diensten über Gerätegrenzen hinweg ist eine wichtige Kernfunktionalität für dessen Erfolg. Die Herausforderungen dieses Paradigmas sind zum Einen die Schaffung einheitlicher und offener Kommunikationsstandards, aber auch die Fähigkeit des Datenaustausches zwischen verschiedenen Anwendungen. Hansmann u. a. sehen die Lösungsansätze für diese Heraus-

forderungen in plattformunabhängigen und skalierenden Sprachen wie Java[2], universellen Austauschformaten wie z. B. XML oder Konzepten wie Jini und UPnP.

Einfachheit. Pervasive Geräte sollen überschaubar und leicht zu bedienen sein. Im Gegensatz zum universell einsetzbaren PC bieten kleine Geräte wie Smartphones nur einen begrenzten Funktionsumfang. Vom Blickpunkt der Usability aus erledigen sie diese Aufgaben allerdings sehr effizient. Die in diesem Zusammenhang genannten Stichworte sind *Verfügbarkeit*, *Zuvorkommnis* und *Einfachheit in der Benutzung*. Es soll problemlos sein und wenig Zeit benötigen, die Funktionsweise einer pervasiven Anwendung zu erlernen. Die Schaffung von schnell zu verstehenden und effizient zu bedienenden Benutzerschnittstellen, die zusätzlich die Komplexität der zugrundeliegenden Anwendung maskieren, ist die große Herausforderung dieses Aspekts des Pervasive Computing [Hansmann u. a., 2003].

Merkmal	**Herausforderungen**
Eingeschränkte Ressourcen	Anpassungsfähigkeit Ressourcen- und Energieverwaltung
Mobilität	Schlechte Qualität drahtloser Verbindungen Verbindungsabbrüche Rekonfigurierbarkeit
Benutzerzentrierung	Context-awareness Proaktivität Skalierbarkeit Unaufdringlichkeit und Unsichtbarkeit Datenschutz
Heterogenität	Interoperabilität Service Discovery Portabilität
Spontane Interaktion	Benennung von Ressourcen Sicherheit Vertrauenswürdigkeit

Tabelle 2.1: Merkmale und Herausforderungen von Pervasive Computing

Riva [2007] stellt die Merkmale der praktischen Realisierung von Pervasive Computing ihren Herausforderungen in ihren einzelnen Gesichtspunkten gegenüber (s. a. Tabelle 2.1):

[2] Java ist darauf ausgelegt, auf einer Vielzahl von Plattformen, wie z. B. Smart Cards, Handhelds oder leistungsfähigen PCs zu laufen.

Eingeschränkte Ressourcen. Der Umgang mit eingeschränkten Ressourcen spezialisierter Geräte erfordert ein hohes Maß an Anpassungsfähigkeit der Anwendung und eine durchdachte Verwaltung derselbigen, sowie eine leistungsfähige Energieverwaltung. Die Geräte müssen ihre Umgebung beobachten, um Änderungen des verfügbaren Dienstangebots zu erkennen und darauf reagieren zu können. In Riva [2007] wird eine Einteilung in Geräteklassen vorgenommen, die auch schon in Kehr u. a. [1999] vorgenommen wurde:

- *Sensoren* stellen die Haupteingabegeräte in einer pervasiven Umgebung dar. Sie bieten in der Regel wenig Rechenleistung und sind über eine geringe Bandbreite an das System angeschlossen.
- *Aktuatoren (auch Aktoren)* werden komplementär zu Sensoren gesehen. Sie sind in Lage, Befehle zu empfangen und bestimmte Entitäten zu kontrollieren. Beispiele für Aktuatoren sind Bildschirme oder Eingabegeräte.
- *Informationsverarbeitende Knoten* haben die Aufgabe, benutzererstellte Inhalte aufzunehmen, zu verwalten und weiterzuleiten. Der Benutzer in einem pervasiven System ist somit nicht nur Informationsempfänger, er gibt auch selbst Informationen in das System. Einerseits kann es sich dabei um aktiv selbst erstellte Dokumente und Medien handeln, andererseits kann das System auch im Hintergrund die Handlungen des Benutzers beobachten, um sich zuvorkommend verhalten zu können. Weiser und Brown [1997] führte in diesem Zusammenhang bereits den Begriff der *Thin Servers* ein. Es handelt sich dabei um kleine und günstige Geräte, die Informationen zu einem gewissen Zweck bereitstellen und verarbeiten können.

Mobilität. Bedingt durch die Mobilität der Geräte müssen diese zusätzlich mit der wechselnden Verfügbarkeit drahtloser Verbindungen umgehen können und sollten einen möglichst unaufdringlichen Mechanismus zur nahtlosen Anpassung bieten. Kleinrock [2003] nennt diesen Aspekt *„Nomadic Computing“* oder auch *„Nomadicity“*. Nomadic Computing beschreibt den transparenten Umgang mit wechselnden und spontan verfügbaren Netzzugängen, sowie deren Bandbreiten und der erforderlichen Konfiguration der Geräte.

Satyanarayanan [2001] nennt verschiedene Möglichkeiten des Umgangs mit diesen Herausforderungen: Zunächst kann die Anwendung auf den Umgang mit knappen Ressourcen angepasst sein. Sie kann beispielsweise weniger Bandbreite benötigen oder auch (temporär) ohne Netzverbindung arbeiten. Bei der Wahl dieser Alternative würde gegebenenfalls die Qualität der Anwendung beeinträchtigt werden. Eine zweite Möglichkeit wäre es, eine erforderliche Zusicherung der benötigten Ressourcen durch die Umgebung einzufordern. Ein Client könnte so sicherstellen, dass er unterbrechungsfrei funktionieren kann. Diese Möglichkeit verlagert das Problem jedoch nur,

da bei Nichtverfügbarkeit der angeforderten Ressource ebenfalls eine Handlungsalternative gefunden werden muss. Die dritte von Satyanarayanan vorgestellte Lösung ist die Delegation der Fragestellung an den Benutzer. Für diesen ginge somit zwar die Transparenz der Verbindung verloren, er hätte jedoch selbst Einfluss darauf, ob eine eingeschränkte Ressourcenverfügbarkeit seinen Ansprüchen genügt. Riva [2007] sieht alternativ noch Möglichkeiten einer regelbasierten Steuerung der Ressourcenbelegung, sowie der Lernfähigkeit des Systems, um die Anforderungen von Anwendungen vorauszusehen und so eine effiziente Rekonfiguration zu erreichen.

Benutzerzentrierung. Das Erreichen von *Benutzerzentrierung* hängt davon ab, wie gut sich eine Anwendung auf die Anforderungen des Nutzers einstellen kann. Sie muss dazu kontextbezogen, proaktiv und dennoch unaufdringlich sein. Im Hintergrund spielen die Skalierbarkeit auf verschiedene Geräteklassen, sowie Datenschutz wichtige Rollen. Arbanowski u. a. [2004] argumentieren, dass der Benutzer und seine Handlungen im Mittelpunkt des Designprozesses mobiler (und damit auch pervasiver) Anwendungen stehen soll.

Pervasive Computing findet in sogenannten *Smart Spaces* statt [Rosenthal und Stanford, 2000]. Es handelt sich hierbei um Orte, wie beispielsweise einen mit entsprechender Technologie ausgestatteten Arbeitsplatz oder auch einen Parkplatz [Riva, 2007]. Dadurch wird es diesen Orten ermöglicht, autonom Handlungen zu erfassen und auszuführen, wodurch sie intelligent (*„smart“*) werden. In Kindberg und Fox [2002] wird die Integration von pervasiver Technologie in Alltagsgegenstände wie z. B. Kaffeetassen oder auch der Aufbau eines intelligenten Arbeitsraums beschrieben. Der Zweck hinter diesen Visionen ist die Schaffung einer Umgebung, die sich auf die jeweilige Situation und das Handeln des Benutzers einstellt, sich also kontextbezogen (*„context aware“*) verhält.

Diese Aspekte der Context Awareness werden in Chen und Kotz [2000] auch unter dem Begriff *Context-Aware-Computing* aufgeführt. Chen und Kotz unterscheiden dabei zwischen aktiver und passiver Context-Awareness. Aktive Context-Awareness bedeutet, dass sich das Verhalten einer Anwendung aktiv mit dem Erkennen eines neuen Kontextes verändert. Passive Context-Awareness-Anwendung informiert den Nutzer lediglich über eine Änderung des Kontextes und erfordert somit mehr Interaktion.

Heterogenität. Der Umgang mit der *Heterogenität* von Umgebung und verbundenen Systemen soll ebenfalls möglichst gut vor dem Nutzer verborgen bleiben. Dieses Ziel soll durch die durchgängige Ausrichtung auf Selbstkonfiguration, sowie Interoperabilität und Portabilität durch die Verwendung offener Standards erreicht werden. In Riva [2007] wird davon ausgegangen, dass eine kompatible Implementierung verschiedener intelligenter Umgebungen nicht erreichbar ist. Die Herausforderungen im Umgang

mit der Heterogenität mobiler Geräte werden in Brewer u. a. [1998] diskutiert und entsprechende Lösungsansätze vorgestellt. Satyanarayanan [2001] bezeichnet Heterogenität auch als *„Uneven Conditioning“*.

Die Heterogenität in pervasiven Systemen entsteht zum einen durch die Verwendung unterschiedlicher Netzwerktechnologien und -topologien, sowie zum Anderen durch den Einsatz verschiedenartiger Interaktionsmethoden. Die Heterogenität des Netzwerks entsteht einerseits durch das Angebot mehrerer Protokolle (z. B. WLAN, Bluetooth oder Mobilfunknetz), aber auch durch die Anwendung unterschiedlicher Netzwerkmanagement-Methoden (z. B. infrastrukurbasiert oder Ad-hoc-Netze bei WLAN). Diese Heterogenität bewirkt das Vorhandensein mehrerer voneinander isolierter Netzwerke. Ein Gerät kann sich jedoch in der Regel nur mit einer begrenzten Anzahl von Netzwerken verbinden – häufig ist sogar nur eine einzige Verbindung möglich. Dienste, die über eine andere Verbindung angeboten werden, können dann nicht erreicht werden. Die Heterogenität der Interaktionsmethoden entsteht u. a. durch den Einsatz unterschiedlicher Middleware-Plattformen wie z. B. JINI, UPnP oder Webservices. Diese Systeme verwenden verschiedene Methoden der Datenrepräsentation und Kommunikationsprotokolle, was zur Folge haben kann, dass der Benutzer nur mit Diensten interagieren kann, die von seinen Geräten unterstützt werden [Raverdy u. a., 2006].

Riva sieht die Kernherausforderung im Umgang mit Heterogenität in pervasiven Umgebungen in der Fähigkeit der Anwendungen unter *unchoreographierten Bedingungen* („unchoreographed conditions“, [Riva, 2007, S. 18]) zu funktionieren[3].

Spontane Interaktion. Als letzter Punkt wird das Merkmal der *spontanen Interaktion* angeführt. Die Herausforderungen liegen für diesen Punkt v. a. in der einheitlichen Benennung der von den Interaktionspartnern gemeinsam verwendeten Ressourcen und der Gewährleistung von Sicherheit und Datenschutz. Kindberg und Fox [2002] charakterisieren *spontane Interaktion* als die Fähigkeit eines Dienstes oder Programms, in einer flüchtigen Infrastruktur zu funktionieren. Spontan interagierende Geräte stellen innerhalb eines Systems Dienste bereit. Das im System verfügbare Angebot, sowie die Eigenschaften der angebotenen Dienste können ohne Vorankündigung wechseln oder auch gänzlich wieder verschwinden. In einer solchen Umgebung ist es insbesondere im Umgang mit mobilen Geräten häufig der Fall, dass ein Gerät zum ersten (oder einzigen) Mal mit einem Dienst interagiert. Dieser Umstand stellt hohe Anforderungen an die Vertrauenswürdigkeit, Sicherheit und Kooperationsfähigkeit der beteiligten Geräte. Die Möglichkeit der spontanen Interaktion kann beispielsweise auch das Bereitstellen der Geoposition des Gerätes (bzw. des Nutzers) oder die Preisgabe sensitiver Kon-

[3] vgl. hierzu die Erläuterung der Begriffe Choreographie und Orchestrierung zur Komposition von Webservices im Abschnitt 4.2.1 unter der Überschrift „ Middleware für Service-Oriented-Computing“

textinformationen erfordern und stellt damit komplexe Anforderungen an den Schutz und die Steuerung der Freigabe privater Daten [Buttyán und Hubaux, 2003].

2.3 Ambient Intelligence

Der Begriff Ambient Intelligence beschreibt Konzepte, die auf den Ansätzen von Ubiquitous Computing, Ubiquitious Communication und Intelligent User Interfaces basieren. Ubiquitous Computing stellt die Grundlage der Metapher dar, indem es die allgegenwärtige verfügbarkeit von Computern in Form von intelligenten Gegenständen beschreibt. *Ubiquitious Communication* ermöglicht es diesen intelligenten Gegenständen miteinander zu kommunizieren. *Intelligent User Interfaces* bilden eine natürliche Schnittstelle zwischen Mensch und Maschine, mit deren Hilfe der Benutzer auf natürliche Weise (z. B. über Gesten oder Sprache) mit dem System interagieren kann [Alcañiz und Rey, 2005]. Eine andere Art der Differenzierung zwischen Pervasive Computing und Ambient Intelligence wird in Tabelle 2.2 dargestellt. Während beim Pervasive Computing die Herausforderungen von Verteilung und Interaktion im Fokus liegen, rücken bei den Untersuchungen zu Ambient Intelligence verstärkt Themen der Einbettung in das tägliche Leben, Personalisierung und Benutzerunterstützung in den Vordergrund [Aarts und de Ruyter, 2009].

Pervasive Computing	–	**Ambient Intelligence**
Allgegenwärtigkeit	–	Einbettung
Interaktivität	–	Context-Awareness
Interoperabelität	–	Personalisierung
Verteilung	–	Anpassungsfähigkeit
Skalierbarkeit	–	Zuvorkommnis

Tabelle 2.2: Schwerpunktthemen von Pervasive Computing und Ambient Intelligence

Während mit den Ideen des Ubiquitous Computing eine sehr langfristige Vision beschrieben wird und die Entwicklungen des Pervasive Computing eher kurzfristig und technologiegetrieben orientiert sind, wird mit Ambient Intelligence eine deutlich architekturgeprägte Vision vorgestellt. Veröffentlichungen zu diesem Thema (z. B. [Riva u. a., 2005] oder Weber u. a. [2005b]) stellen für gewöhnlich einen ganzheitlichen Architekturaspekt in den Vordergrund, aus dem die technische Realisierung abgeleitet wird. Weber u. a. [2005a] charakterisieren Ambient Intelligence als eine Umgebung,

- in der die Technologie eingebettet und verborgen ist,
- die sich der Anwesenheit von Personen und Gegenständen anpasst,

- die Tätigkeiten des Benutzers unaufdringlich und im Hintergrund ausführt, sowie
- für Sicherheit, Datenschutz und Vertrauen sorgt, indem sie in angebrachter Weise benötigte und sinnvolle Informationen bereitstellt.

Alcañiz und Rey [2005] beschreiben die technologischen Herausforderungen von *Ambient Intelligence* und gliedern diese in die drei Bereiche *Ubiquitous Computing* , *Ubiquitous Communication* und *Intelligent User Interfaces*. Für die Bereiche *Ubiquitous Computing* und *Communication* werden die folgenden technischen Punkte herausgestellt:

Benutzerschnittstelle. Jede Art von Gerät soll eine intuitive Benutzerschnittstelle bieten, die dem Benutzer eine natürliche Form der Interaktion ermöglicht.

Günstige Geräte. Da jeder Benutzer Zugriff auf eine Vielzahl von Geräten hat, die miteinander interagieren, ist es erforderlich, dass jedes einzelne Gerät günstig in der Anschaffung ist. Da die Geräte jeweils einen speziellen Zweck erfüllen, sind hohe Hardwareanforderungen, wie sie heute an PCs gestellt werden, nicht mehr erforderlich.

Hohe Bandbreite. Da mit einer Vielzahl von Geräten auch der Anteil der Kommunikation unter den Geräten zunehmen wird, ist eine hohe Bandbreite zur Vermeidung von Netzwerklatenzen sinnvoll.

„Unsichtbare“ Dateisysteme. Wenn ein Nutzer einen Computer verwendet, ist die Organisation und das Speichern von Daten und Dateien ein zentraler Bestandteil im Umgang mit dem System. Es benötigt verhältnismäßig viel Aufwand zu lernen, wie ein Dateisystem (aus Benutzersicht) organisiert ist und wo bestimmte Datentypen abgelegt werden sollten. Die Aufmerksamkeit des Benutzers wird so von der Erledigung der eigentlichen Aufgabe abgelenkt. Ein solches Handeln steht im Widerspruch zu der Idee des unsichtbaren Computers, da es die Auseinandersetzung des Benutzers mit den Eigenschaften des benutzten Systems erfordert. Der Nutzer sollte daher Zugriff auf seine Daten haben, ohne konkret über Dateinamen, Pfade oder Dateitypen informiert zu sein[4].

Automatische Installation. In *Ubiquitous Computing*-Umgebungen sollte es keinen Grund mehr geben, Programme zu installieren. Programme sollten sich einfach von einem Gerät auf ein anderes übertragen lassen, ohne dass ein Nutzer eine aufwendige Installationsroutine durchlaufen muss. Alcañiz und Rey führen hier Java als Beispiel einer plattformunabhängigen Programmiersprache auf, die diese Anforderung erfüllen kann.

[4] Ein Beispiel aus heutiger Sicht wäre z. B. der Speicherort von Mailbox-Nachrichten oder E-Mails. Der Zugriff wird in der Regel durch eine spezielle Benutzerschnittstelle gesteuert.

Personalisierbare Informationen. Ein Benutzer sollte die Repräsentation von Informationen in den von ihm verwendeten Anwendungen personalisieren können. So kann eine auf seine Bedürfnisse zugeschnittene Benutzererfahrung erreicht werden.

Schutz privater Daten. Der Schutz privater Daten ist einer der größten Kritikpunkte und somit auch eine der größten Herausforderungen von Ubiquitous Computing und Ambient Intelligence. Die Übergabe hochsensibler Daten an ein vernetztes Computersystem macht das System interessant für potentielle Angreifer. Alcañiz und Rey fordern daher – wie auch zuvor Weiser [1991] – den frühen Einbezug von Datenschutz- und Sicherheitsmechanismen in das Anwendungsdesign.

Der weitere Punkt *Intelligent User Interfaces* umfasst zusätzlich die folgenden Punkte, mit denen ein *Ambient Intelligence*-System umgehen können soll. Alcañiz und Rey beziehen sich dabei auf diese in Maybury und Wahlster [1998] herausgestellten Punkte:

Multimodale Eingabe. Es handelt sich um Eingaben, die sich aus mehreren, teilweise unpräzisen Eingabeformen zusammensetzen. Diese können z. B. aus Text, Gesten, Sprache oder Blicken bestehen.

Multimodale Ausgabe. Die Ausgabe ist an den Kontext angepasst. Je nachdem, welche Ausgabeform am besten passt, kann die Ausgabe beispielsweise aus beliebigen Kombinationen von Sprache, Text, Symbolen oder auch durch animierte, menschenähnliche Agenten geschehen.

Interaction Management. In einer Umgebung, in der häufig zwischen Akteuren und Aufgaben wechselnde Eingaben an ein System gegeben werden, ist es wichtig, dass sich dieses auch in angemessener Weise verhält. Ein solches System kann als ein *Context Aware*-System bezeichnet werden [s. a. Dey, 2001]. Diese neue Form von Benutzerschnittstelle soll einen Dialog mit dem Nutzer ermöglichen und ihm z. B. beim Fehlschlagen einer Aktion Wege aufzeigen können, mit denen das Ziel doch erreicht werden kann.

Die vorgestellten Aspekte finden sich auch in Philips Research [2010] wieder. Die dort gegebene Darstellung unterteilt das Thema Ambient Intelligence ebenfalls in unterschiedliche Kategorien, die denen in Alcañiz und Rey [2005] sehr ähneln. In der feineren Untergliederung finden sich zusätzlich noch die Aspekte der *Context Awareness*, *Personalisierung*, des *Einflusses auf die Umgebung* und *allgemeinen Anpassungsfähigkeit* des Systems. Diese Punkte finden sich auch bei ISTAG [2003] und der Zusammenstellung von Arbeiten in Riva u. a. [2005] wieder. Der Einbezug der Context-Awareness führt dazu, dass ein *Ambient Intelligence*-System sich entsprechend des aktuellen Kontextes verhalten kann und somit angepasst reagieren kann. Wenn beispielsweise eine zweite Person einen Raum betritt, ändern sich die Bedingungen für die Preisgabe privater Daten durch das System. Auf der anderen Seite kann sich das System aber auch auf besondere Vorlieben seiner Benutzer

einstellen und so beispielsweise Musik und Beleuchtung entsprechend einstellen. Der intelligente Umgang mit Benutzeranfragen wird durch Riva u. a. [2005] ebenfalls thematisiert. Das System sollte intelligent auf Anfragen antworten, bestenfalls multimodal, so dass immer der Kommunikationskanal verwendet wird, der für den Nutzer am angenehmsten ist. Es soll sich unaufdringlich verhalten, die Benutzung unkompliziert und möglichst ohne Vorwissen zu bewältigen sein. Der Begriff *Ambient Intelligence* erweitert und konkretisiert somit die Gedanken, die Mark Weiser bereits 1991 formulierte.

Abschließend soll hier noch ein weiterer Aspekt zur Einordnung von Ambient Intelligence gegeben werden. Riva [2005] sieht Ambient Intelligence (AmI) als Gegenpol zur *Virtual Reality*:

> *„AmI is the direct extension of today's concept of ubiquitous computing, the integration of microprocessors into everyday objects. More in detail, AmI may be roughly described as the opposite of virtual reality: virtual reality puts people inside a computer generated world; AmI puts the computer inside the world to help us.“* – [Riva, 2005, S. 18f]

Riva leitet aus dieser Abgrenzung die Anforderung an den Designprozess eines Ambient Intelligence-Systems ab, dass dieser sich nicht ausschließlich an den technologischen Anforderungen, sondern viel mehr an den Bedürfnissen der Nutzer orientieren soll. Aarts und de Ruyter [2009] schlagen u. a. aus diesem Grunde ein *Experience Research* genanntes Konzept zur Entwicklung und Untersuchung von Lösungen vor, das die folgenden drei Kernbereiche in den Fokus der Untersuchungen stellt: *Experience@Context*, *Experience@Lab* und *Experience@Field*. *Experience@Context* beschreibt die gesellschaftliche Sichtweise, mit der mögliche Trends und Bedürfnisse der Nutzer identifiziert werden sollen. Die Entwicklung und Untersuchung von Lösungen in einer kontrollierten Laborumgebung wird unter *Experience@Lab* zusammengefasst. Hier werden Nutzungskonzepte und Prototypen nach dem Paradigma des User-Centered-Design entwickelt. Die Integration der Lösungen wird unter dem Aspekt *Experience@Field* durchgeführt, mit dem Ziel, unter möglichst realitätsnahen und unkontrollierten Bedingungen Langzeittests und Versuche durchführen zu können.

2.4 Smart Environments und Smart Homes

Bereits in dem in Weiser [1991] vorgestellten Szenario um die Person „Sal“ spielte das Smart Home (die intelligente Wohnung) neben intelligenten mobilen Geräten und dem intelligenten Arbeitsplatz eine zentrale Rolle. Die Ideen dieser sogenannten *„Smart Spaces“* werden auch in den Konzepten des Pervasive Computing und der Ambient Intelligence immer wieder aufgegriffen [vgl. Satyanarayanan, 2001; Kleinrock, 2003; Riva, 2007]. Die Idee, ein funktionsfähiges Smart Home als eine realitätsnahe Testumgebung zu implementieren, ist eine

Entwicklung, die das u. a. von Aarts und de Ruyter [2009] beschriebene Konzept des *Experience@Field* umsetzt. Die Entstehung von Smart Homes in der Forschung lässt sich aber auch auf die geänderte Sichtweise auf intelligente Systeme und deren Interaktion mit der Umgebung zurückführen.

Eine alte Sichtweise intelligenter Systeme ist ausschließlich auf die Verarbeitung von Symbolen konzentriert, was zu einem Systemdesign nach dem Schema *Eingabe-Verarbeitung-Ausführung* oder auch *Erkennen-Verarbeiten-Handeln* führt (s. Abbildung 2.1). Eine neuere Sichtweise erkennt die Umgebung als Teil des Systems. Die Symbolverarbeitung tritt dabei in den Hintergrund und die Prozesse des Erkennens, Verarbeitens und Handelns werden parallel ausgeführt (s. Abbildung 2.2). Diese Entwicklung bringt eine deutlich stärkere Betonung der Umgebung mit sich. In einem solchen System agiert nicht mehr ein einzelner Agent, sondern ganze Teams von Agenten sind für die Bearbeitung von Ereignissen verantwortlich [Augusto u. a., 2010].

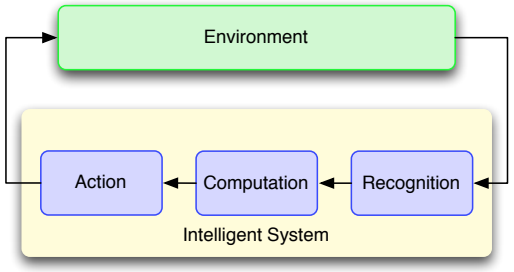

Abbildung 2.1: Eine alte Sicht auf intelligente Systeme
[In Anlehnung an Augusto u. a., 2010]

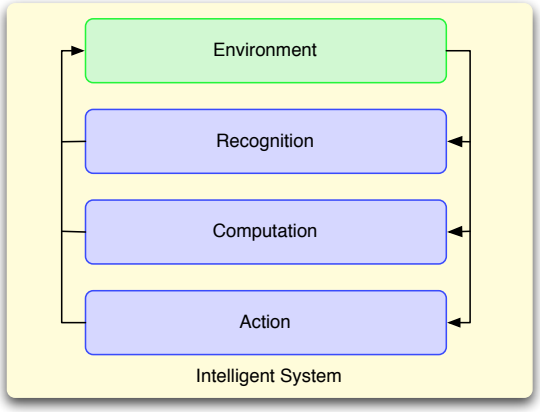

Abbildung 2.2: Eine neue Sicht auf intellligente Systeme
[In Anlehnung an Augusto u. a., 2010]

In einer Zusammenstellung auf [aal-deutschland.de, 2010] werden weltweit ca. 33 Projekte[5] unter der Überschrift Living Labs aufgelistet, die sich mit dem Thema Ambient Intelligence und der Spezialisierung auf Smart Homes beschäftigen. Um die vielfältigen Aspekte dieses Forschungsbereichs zu zeigen, werden im Folgenden einige kurze Beispiele für konkrete Projekte aus dem Bereich Smart Homes gegeben:

Philips HomeLab (Eindhoven, Niederlande). Das Philips Homelab wurde im April 2002 der Öffentlichkeit vorgestellt. Ziel des Labors war es, die Vision einer vollständigen Integration elektronischer Haushaltsgeräte zu erproben, wie sie von Aarts u. a. bis zum Jahr 2020 erwartet wurde. Das Hauptaugenmerk der Forschung lag darin, (1) das

[5] Davon 17 in Deutschland, acht in Europa, sechs in den USA und zwei in Japan.

soziale Zusammenleben, sowie den persönlichen Austausch von Erfahrungen zu fördern, (2) die Möglichkeiten für Unterhaltung und Entspannung zu verbessern, sowie (3) das Bedürfnis nach einem ausgeglichenen und organisierten Leben zu decken. Diese Ziele sollten u. a. durch den Einsatz von stationären und mobilen Bildschirmen, stimmungsabhängiger Lichtsteuerung, intelligenter Fernbedienungen, sowie durch nahtlose Integration dieser Technologien mit einfach zu bedienenden und dennoch leistungsfähigen Benutzerschnittstellen erreicht werden. Zur Überprüfung der Ergebnisse waren ständige Usability Tests und Methoden des Data Mining zur Erkennung von Mustern in Aktionen der Benutzer vorgesehen [Aarts u. a., 2003].

T-Com-Haus Berlin (Berlin). Das T-Com-Haus ist in einer Kooperation von Deutsche Telekom, WeberHaus, Siemens und Neckermann geplant und gebaut worden. Die Fertigstellung war im März 2005. Das T-Com-Haus war im Gegensatz zum Philips HomeLab für die breite Öffentlichkeit zugänglich. Von April 2005 bis Juli 2006 wurde jede Woche einer anderen Gruppe von Menschen die Möglichkeit gegeben in dem Haus zu leben. Das T-Com Haus war ein reales Haus im Zentrum Berlins, das mit einer Vielzahl vernetzter elektronischer Geräte ausgestattet war, die der Unterstützung der Haushaltsführung oder zum Steuern von Multimedia-Features dienen. Die zentrale Bedienung der Geräte geschah über einen PDA oder einen Multimediabildschirm. Das System ermöglichte beispielsweise mit Hilfe des Mobiltelefons, den Ofen vorzuheizen oder die Waschmaschine auf „Schleudern“ zu schalten. Diese Funktionen standen dem Benutzer auch außerhalb des Hauses, z. B. beim Einkaufen, zur Verfügung [aal-deutschland.de, 2010][6].

inHaus-Zentrum (Duisburg). Das inHaus-Projekt teilt sich in zwei Forschungsanlagen. Die erste, das inHaus 1, ist ein bereits 2001 fertiggestelltes Doppelhaus (250 qm Nutzfläche), mit dem Ziel ressoucenschonendes Wohnen, sowie eine zentrale Haus- und Multimediasteuerung zu ermöglichen. Die zweite Immobilie, das inHaus 2 (ca. 3500 qm Nutzfläche) wurde im Juni 2008 fertiggestellt. Es handelt sich dabei um eine Nutzimmobilie, auf deren Flächen intelligente Raum- und Gebäudesysteme untersucht werden. Die Betrachtungen beziehen sich beispielsweise auf Technologien für neuartige Pflegeheime, Krankenhäuser, Bürokomplexe oder auch Hotels [aal-deutschland.de, 2010]. Das inHaus-Zentrum wird von sieben Fraunhofer-Instituten und über 100 Wirtschaftspartnern für neue Technologie-, Produkt- und Anwendungslösungen in Wohn- und Nutzimmobilien unterstützt [inHaus-Zentrum, 2010].

BAALL – Bremen Ambient Assisted Living Lab (Bremen). Das BAALL dient der Erprobung von Konzepten zur altersgerechten technologischen Unterstützung für das Wohnen in gewohnter Umgebung. Es befasst sich somit mit einem Teilaspekt von Smart

[6] Weitere Informationen finden sich auf der Homepage des Projekts unter http://www.t-com.weberhaus.de, Zugriffsdatum: 05.01.2010

Homes, dem sogenannten *Ambient Assisted Living* (AAL). Das BAALL umfasst auf ca. 60 qm die üblichen Wohnbereiche Ankleide-, Schlaf-, Wohn-, Esszimmer, Küche, Bad/Sanitär, Heimbüro. Die Kernpunkte des Labors beziehen sich auf die Integration eines intelligenten Rollstuhls („Rolland“) bzw. einer intelligenten Gehhilfe („iWalker“), Assistenz in der Umgebungssteuerung, die Kontrolle und Steuerung von Haushaltsgeräten, sowie der Ablage und des Findens von Gegenständen (wie z. B. Kleidung) oder auch von Lebensmitteln im Kühlschrank. Eine wichtige Rolle spielen dabei u. a. die Integration von anpassungsfähigen Möbeln, wie beispielsweise einer vollständig und individuell höhenverstellbaren Küchenzeile oder eines intelligenten Bettes in Kombination mit einer Lichtsteuerung. Ein weiterer wichtiger Punkt ist die Schaffung einer intelligenten, adaptiven, leicht und multimodal zu bedienenden Benutzerschnittstelle, die die Komplexität des Systems vor dem Benutzer verbirgt [Krieg-Brückner u. a., 2009].

OFFIS IDEAAL (Oldenburg). Die *Integrated Development Environment for Ambient Assisted Living* (IDEAAL) des Oldenburger Forschungs- und Entwicklungsinstituts für Informatik (OFFIS) befindet sich an der Carl-von-Ossietzky-Universität in Oldenburg. Das Projekt sieht seine Motivation im demographischen Wandel begründet, der eine deutliche Zunahme des Anteils älterer Menschen an der Gesamtbevölkerung vorhersagt [OFFIS e.V., 2010, IDEAAL Konzept » Motivation]. Das Konzept des Projekts sieht – ähnlich wie das BAALL (s. o.) – die Kombination assistiver Technik und neuen Informationstechnologien vor, um so eine selbstbestimmte Lebensführung bis ins hohe Alter zu ermöglichen. Im Jahr 2005 wurde eine Wohnung zu Forschungs- und Demonstrationszwecken eingerichtet und im Jahr 2008 grundlegend umgebaut. Das IDEAAL-Senioren-Appartement ist heute eine 48 qm 2-Zimmer-Wohnung mit eigenständiger Küche und Badezimmer [OFFIS e.V., 2010, IDEAAL Konzept » IDEAAL].

Connected Living (Berlin). Der Verein Connected Living e.V. wird vom Bundesministerium für Wirtschaft und Technologie gefördert. Er hat es sich zum Ziel gesetzt, maßgebliche Partner aus den verschiedenen Anwendungsbranchen für Smart Homes einzubinden, um so die Möglichkeiten der intelligenten Heimvernetzung zu erproben, zu standardisieren und tragfähige Geschäftsmodelle für diese Technologie zu entwickeln. Die Durchführung der Forschungsschwerpunkte wird über das DAI-Labor der TU-Berlin koordiniert. Es werden die Programmpunkte „Kommunikation und Entertainment“, „Energiebewusstsein und Effizienz“, „Haushalt und Versorgung“, „Konsumelektronik und Komfort“, „Gesundheit und Homecare“. Der Verein sieht die Rolle des DAI-Labors dabei in der Entwicklung und Bereitstellung von *„enabling Technologies“* in diesen Bereichen [Connected Living e.V., 2010, *Ziele*][7].

Living Place Hamburg (Hamburg). Das Living Place Hamburg ist ein seit Anfang 2009 betriebenes Labor auf dem Campus der Hochschule für Angewandte Wissenschaften

[7] vgl. hierzu auch [Runge u. a., 2009] und [Blumendorf, 2009]

Hamburg. Es gliedert sich in die zwei Teilprojekte *iFlat* und *Living Place*. Das iFlat bildet einen Laborteil für Experimente und Grundlagenuntersuchungen [Stegelmeier u. a., 2009]. Hier sollen Lösungen entwickelt werden, die bei entsprechender Eignung in den Kontext des Living Place übernommen werden. Das Living Place befindet sich zur Zeit im Aufbau. Es soll nach seiner Fertigstellung als Test- und Präsentationsraum für die aus dem iFlat übertragenen Lösungen und Anwendungen dienen. Der Fokus der Untersuchungen liegt dabei auf den verschiedenen Möglichkeiten der Interaktion in einem Smart Home. Es ist als eine vollständige Wohnung geplant und soll interessierten Personen die Möglichkeit bieten, hier auch testweise zu wohnen. Auf diese Weise sollen Langzeit-Usability-Tests ermöglicht werden [Gregor u. a., 2009][8].

Die Entwicklungen auf dem Gebiet der Smart Homes zeigen die Bestrebungen in Industrie und Forschung, sowohl greifbare Anwendungsfälle, als auch tragbare Geschäftsmodelle zu entwickeln. Als Zielgruppen rücken dabei verstärkt ältere Menschen in den Fokus, die eine Unterstützung für ein selbstgeführtes Leben im Alter benötigen. Auf der anderen Seite sollen aber auch junge Menschen angesprochen werden, die ein gesteigertes Interesse an der Integration der technischen Möglichkeiten haben. In nahezu jedem der vorgestellten Projekte ist es vorgesehen, eine integrierte, systemweite Lösung zu schaffen und sich so vom derzeitigen Stand zu entfernen, der von de Ruyter und Aarts [2004] als *„fragmented with features“* bezeichnet wird.

2.5 Diskussion

Im vorliegenden Kapitel wurde die Entwicklung von den fast utopisch wirkenden Vorstellungen Mark Weisers zu Ubiquitous Computing, über eine geräte- und funktionalitätsgetriebene Weiterentwicklung in Gestalt des Pervasive Computing, bis hin zu einer ernsthaften und eher architekturgetriebenen Entstehung des Forschungsgebiets Ambient Intelligence beschrieben. Aus dem letzgenannten der Forschungsschwerpunkt „Smart Homes“ hervor.

Die Entwicklung hat gezeigt, dass durch die Verfügbarkeit immer günstiger und leistungsfähiger werdender mobiler und eingebetteter Geräte vielfältige Möglichkeiten ergeben. Die Funktionalität des Gesamtsystems ergibt sich dabei aus der Zusammenstellung der Fähigkeiten der einzelnen Geräte. Es ist wichtig, dass dabei gesehen wird, dass nicht nur die technologischen Fähigkeiten der einzelnen Geräte wichtig ist, sondern auch die infrastrukturellen Möglichkeiten zur Integration der Geräte an Bedeutung gewinnt.

[8] Eine tiefergehende Beschreibung des Living Place Hamburg wird im Rahmen der Systemanalyse in Kapitel 3 vorgenommen

3 Systemanalyse im Living Place Hamburg

In diesem Abschnitt werden die Anforderungen an eine Infrastruktur für das Living Place Hamburg erörtert. Zu diesem Zweck wird zunächst eine nähere Eingrenzung der Problemstellung, sowie eine Beschreibung der beteiligten Labore und ihrer Beziehungen zueinander vorgenommen. Es folgt eine kurze Vorstellung vergleichbarer Arbeiten. Aufbauend auf der Beschreibung der Infrastruktur werden anhand von drei realitätsnahen Anwendungsszenarien die Anforderungen an eine Systemarchitektur abgeleitet. Die Szenarien beschreiben dabei die Sicht eines Benutzers auf das System, aus der die Anforderungen für den Entwickler abgeleitet werden.

3.1 Problemstellung

Die Implementierung eines verteilten Systems ist eine komplexe Aufgabe. Insbesondere die Entwicklung von Anwendungen für ein Smart Home stellt hohe Anforderungen an die nahtlose Integration einzelner Anwendungen und an das Bedienkonzept. Dienste sollen anwendungsübergreifend verwendet werden können und den Fokus des Benutzers nicht von seiner eigentlichen Handlung ablenken. Die an der Hochschule für Angewandte Wissenschaften Hamburg betriebenen Labore *iFlat* und *Living Place Hamburg* (s. a. Abschnitt 3.2) stellen eine Infrastruktur zur integrativen Entwicklung von Anwendungen für Smart Homes bereit. Neue Ideen können zunächst prototypisch im iFlat erprobt werden, bevor sie in einem Gesamtzusammenhang im *Living Place Hamburg* integriert werden.

Aus den Anwendungsfällen der prototypischen Entwicklung (“rapid prototyping“), sowie des Technologietransfers aus dem iFlat ins Living Place Hamburg ergibt sich die Schwierigkeit, dass diese Prozesse nur unzureichend durch die bestehenden Arbeitsabläufe abgedeckt sind. Mit der Einrichtung des iFlat wurde zunächst der Weg der Entwicklung einzelner Prototypen verfolgt, die eher zwanglos auf ähnlichen Architekturmustern beruhten. Das Living Place Hamburg sieht allerdings vor, dass ein integriertes Gesamtsystem „Wohnung“ geschaffen wird, in dem ein hoher Grad der Interaktion zwischen den Endgeräten möglich

ist. Zu diesem Zweck soll eine gemeinsame Infrastruktur für iFlat und Living Place geschaffen werden, mit deren Hilfe die Komplexität der Anwendungsentwicklung für Smart Homes, sowie des Technologietransfers zwischen den Laboren verringert wird.

3.2 Living Place Hamburg

Das Department Informatik der Hochschule für Angewandte Wissenschaften Hamburg führt verstärkt Projekte im Themenbereich Ambient Intelligence durch, in denen das Konzept der intelligenten, sich dem jeweiligen Kontext anpassenden Wohnung aufgegriffen wird. Die primären Einflüsse ergeben sich zu großen Teilen aus interdisziplinären Kooperationen der Bereichen Informatik, Design und Medientechnik (vgl. Abbildung 3.1). Der Hauptarbeitsbereich des Forschungsschwerpunktes Ambient Intelligence liegt auf dem Gebiet einfacher Interaktion mit allgegenwärtiger und nahtlos integrierter Technologie.

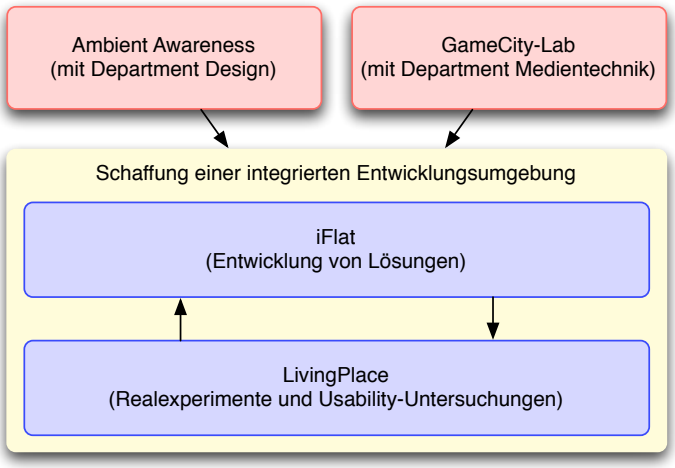

Abbildung 3.1: Einflussbereiche des Living Place Hamburg

Um diese Aspekte zu untersuchen, wurden zwei Labore gegründet, die in enger Kooperation die Forschung auf den Gebieten Ambient Intelligence und Smart Homes durchführen. Abbildung 3.2 zeigt das Zusammenspiel der Labore, sowie deren Schwerpunkte und den stattfindenden Informationsaustausch. Die Entwicklung und Erforschung neuer Lösungen und Ideen findet zunächst im iFlat-Entwicklungslabor statt (Abschnitt 3.2.1). Wenn sich eine Entwicklung als potentiell nützlich und stabil genug für den täglichen Einsatz erweist, kann die isolierte Lösung daraufhin im Living Place Hamburg in einen Gesamtkontext integriert werden, um sie dort in Langzeit-Usability-Tests auf ihre Alltagstauglichkeit zu überprüfen (Abschnitt 3.2.2). Die Ergebnisse der Evaluation beeinflussen daraufhin wieder in die Neuentwicklungen im Rahmen des iFlat.

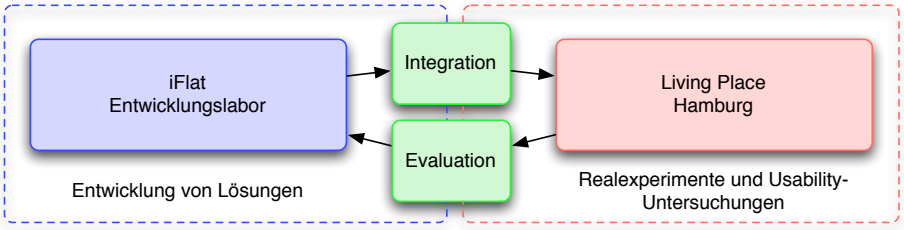

Abbildung 3.2: Zusammenspiel der Labore Livingplace Hamburg und iFlat

Das iFlat ist somit primär als Experimentier- und Entwicklungslabor ausgelegt, während das Living Place Hamburg als eine tatsächlich benutzbare Wohnung geplant ist, die zu Evaluations- und Demonstrationszwecken verwendet werden soll. Die entstehenden Wechselwirkungen zwischen den Laboren sollen eine deutliche Steigerung der Anwendungsqualität bewirken.

3.2.1 iFlat

Das 2007 entstandene iFlat-Labor dient der Umsetzung von Projekten, die der Metapher „intelligente Wohnung“ zugeschrieben werden. Herzstück dieses Labors war für lange Zeit eine Infrastruktur, die einen Blackboard-basierten Servicebus zur Kommunikation der Endgeräte zur Verfügung stellt [Stegelmeier u. a., 2009]. Tabelle 3.1 ordnet frühe studentische Arbeiten mit Bezug zum iFlat einzelnen Themenbereichen zu. Diese befassen sich mit konkreten Fragestellungen bezüglich der Infrastruktur, Positionsbestimmung und Context-Awareness, Ambient Assisted Living, Datenorganisation, sowie der Untersuchung multimodaler Interaktionsformen.

Im iFlat werden Anwendungsfälle untersucht, die sich an realitätsnahen Szenarien für ein Smart Home orientieren. Zur Realisierung dieser Anwendungsfälle werden die im iFlat bereitstehenden Komponenten verwendet. Es handelt sich dabei um mehrere leistungsfähige Desktopcomputer, sowie einige Eingabe- und Sensorkomponenten, wie z. B. ein Indoor-Location-Systen, Touchscreens, Webcams oder RFID-Reader.

Das Indoor-Location-System kann aktive Tokens (winzige elektronische Geräte) durch Messungen im Ultra-Wideband-Frequenzbereich orten. Die Messergebnisse basieren dabei auf Feldstärken-Messungen zwischen vier Sendeantennen. Das im iFlat eingesetzte Produkt wird von der Firma Ubisense[1], das laut Herstellerangaben die Ortung von Gegenständen im dreidimensionalen Raum mit einer Genauigkeit von 15 cm ermöglicht. Bei

[1] Website: http://www.ubisense.de, Abrufdatum: 14.06.2010

Themenbereich	–	Kurzbeschreibung	–	Studentische Arbeiten
Infrastruktur	–	Schaffung einer Infrastruktur mit dem Ziel, die transparente Interaktion vieler intelligenter Endgeräte zu ermöglichen.	–	[Dreyer, 2007], [Hollatz, 2007, 2008a,c,b]
Positionsbestimmung und Context-Awareness	–	Erkennen von Handlungs- und Positionskontexten zur Ableitung von Handlungsinitiativen des Systems	–	[Urich, 2007, 2008a, 2009], [Vollmer, 2007, 2008], [Gregor, 2007]
Ambient Assisted Living	–	Unterstützung eingeschränkter Personen mit dem Ziel, eine möglichst barrierearme selbstständige Lebensführung zu ermöglichen.	–	[Bernin, 2007], [Meißner, 2007, 2008a,c,b]
Datenorganisation	–	Organisieren und Finden von Daten und Personen in verteilten Systemen.	–	[Gärner, 2007, 2008b,a], [Mählmann, 2007, 2008b,a], [Urich, 2008b]
Interaktion	–	Untersuchung neuartiger Eingabemethoden mit dem Ziel, eine sowohl intuitive, als auch effiziente Handhabung komplexer Systeme zu erreichen.	–	[Hamann, 2007, 2008b,c,a], [Dreyer, 2008], [Gehn, 2007, 2008b,a], [Rödiger, 2007]

Tabelle 3.1: Übersicht studentischer Projekte mit Bezug zum iFlat

den Touchscreens handelt es sich um kapazitive Desktop-Touchscreens und große 42-Zoll-Bildschirme mit Infrarotrahmen (multitouch-fähig). Die eingesetzten RFID-Reader zur Nearfield-Kommunikation sind Entwicklungsprototypen der Firma Frosch-Elektronik[2]. Sie erlauben das Auslesen von passiven RFID-Tags bis zu einer Entfernung von einem Meter. Als Softwareinfrastruktur des iFlat kommt in den meisten Projekten der Event Heap des Stanford iROS-Systems zum Einsatz[3].

Im Folgenden soll die Verwendung einiger dieser Komponenten kurz vorgestellt werden.

Intelligentes TV-System. Ein Intelligentes TV-System besteht aus mehreren unabhängigen Komponenten zum Empfang des TV-Signals, sowie Wiedergabe des Bild- und Tonmaterials. Das System nutzt zusätzlich die Fähigkeiten des Ubisense-Systems, um beispielsweise das Fernsehbild automatisch auf den dem Zuschauer am nächsten stehenden Bildschirm wiederzugeben. In diesem Zusammenhang wurde in [Dreyer,

[2]Website: http://www.froschelectronics.com/, Abrufdatum: 14.06.2010
[3]Eine nähere Beschreibung des Event Heap wird in Abschnitt 3.3.5 vorgenommen.

2009] ein agentenbasiertes System entwickelt, das in der Lage ist, dem Benutzer basierend auf seinem Fernsehverhalten Programme von Interesse vorzuschlagen.

smart:shelf. Das Intelligente Regalsystem *smart:shelf* ist im Rahmen eines Masterprojekts [Hollatz, 2008c; Meißner, 2008c; Urich, 2008b] entwickelt worden. Es handelt sich dabei um ein System, das in der Lage ist, den Inhalt zweier Regale basierend auf RFID-Technologie zu bestimmen. Es stehen dem System somit essentielle Informationen bereit, die es z. B. erlauben, einen intelligenten Menüplaner [Urich, 2009] zu implementieren.

Türklingel. Die intelligente Türklingel ist die Simulation einer Videogegensprechanlage, die mit dem TV-System interagieren kann. Wenn jemand die Türklingel betätigt, wird das Bild der Türkamera auf dem Fernseher dargestellt. Ein gegebenenfalls noch laufendes Fernsehprogramm wird unterbrochen und im Hintergrund aufgezeichnet. Sollte der Benutzer sich entschließen, die Tür zu öffnen, wird die Aufzeichnung fortgeführt und das Programm kann zu einem späteren Zeitpunkt angesehen werden. Sollte die Tür nicht geöffnet werden, wird Programmwiedergabe ab der Stelle der Unterbrechung wieder abgespielt[4].

3.2.2 Living Place

Mit Beginn des Jahres 2009 wurde mit der Planung zum Aufbau des Living Place Hamburg begonnen. Das Erreichen der ersten Ausbaustufe ist für Herbst 2010 geplant. Es soll als Labor zur Überprüfung der Forschungsergebnisse aus dem Bereich intelligente Wohnungen in Realexperimenten dienen und ist daher als eine vollständig ausgestattete Wohnung konzipiert. Es werden so im Rahmen von Experimenten und Usability-Studien Laboruntersuchungen bis zu einer Dauer von mehreren Tagen ermöglicht. Es soll so gezeigt werden können, ob sich die eingesetzten Technologien als tauglich für den Alltag erweisen.

Der Grundriss des Labors (Abbildung 3.3) zeigt einen fast 140 qm großen Wohnbereich (inkl. Bad), der das eigentliche Labor darstellt, sowie drei Arbeitsräume mit einer Gesamtfläche von ca. 53 qm und einem ca. 17 qm großen Kontrollraum. An diesen sind eine Reihe Arbeitsräume angeschlossen, in denen Entwicklertätigkeiten durchgeführt werden. In der konkreten Ausprägung des Living Place Hamburg wird auch die Kooperation mit Hamburger Firmen angestrebt. In Kombination mit dem iFlat-Labor stellt das Living Place Hamburg eine realistische Test- und Evaluationsumgebung für neuartige Technologien dar [Gregor u. a., 2009].

[4] Im Rahmen der Analyse in Abschnitt 3.4 wird ein erweitertes Beispielszenario dieser Anwendung vorgestellt.

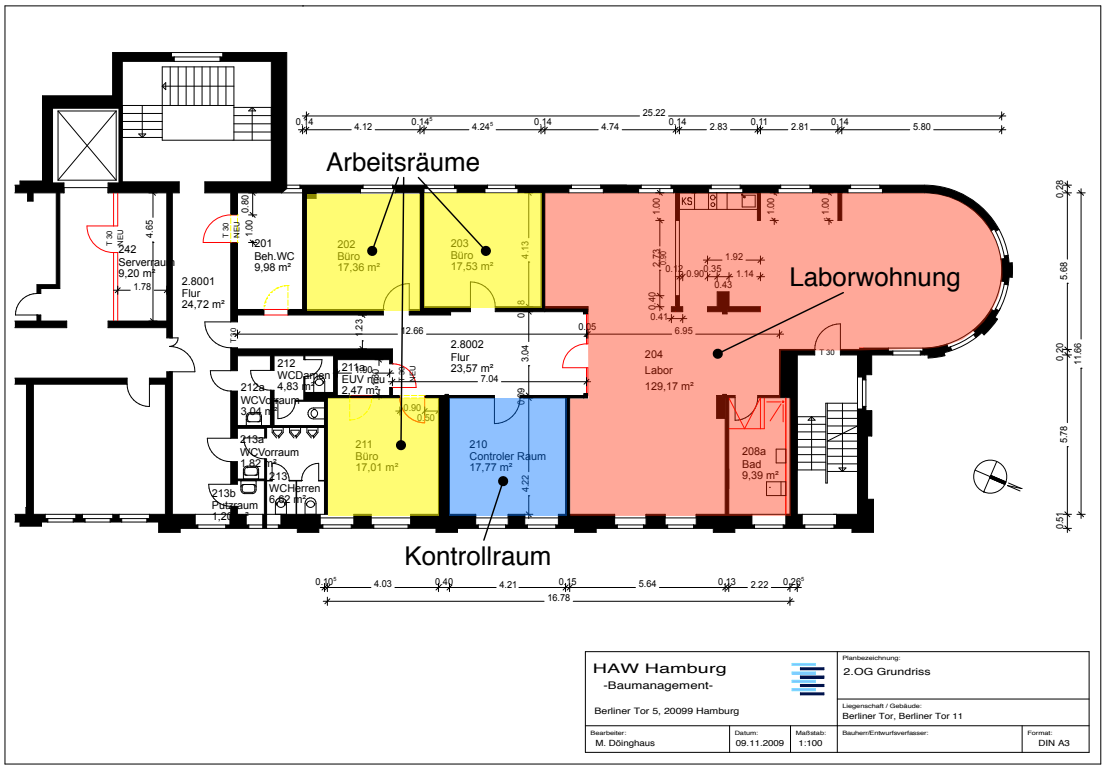

Abbildung 3.3: Grundriss des Living Place Hamburg

Für den Rohbau des Living Place Hamburg wurde ein denkmalgeschütztes Bestandsgebäude aus dem Jahr 1928 auf dem Campus Berliner Tor der Hochschule für Angewandte Wissenschaften Hamburg umgebaut. Es wurde ein Badezimmer eingebaut, sowie Vorbereitungen für den Einbau einer Küche geschaffen. Nahezu die gesamte Fläche des Wohnbereichs ist mit einem Doppelboden versehen, wodurch Kommunikations- und Datenleitungen flexibel und versteckt verlegt werden können. Der Doppelboden ermöglicht zusätzlich die Installation weiterer Technologien, wie z. B. eines Indoor-Positioning-Systems. Das Labor kann somit ohne großen Aufwand umgestaltet und an sich ändernde Anforderungen angepasst werden. Abbildung 3.4 zeigt einige Impressionen des fertiggestellten Rohbaus des Living Place. Zu sehen sind die Außenansicht des Gebäudes, ein Blick in die Rotunde und die weiteren Bereiche des Wohnraums, sowie das Badezimmer. Die Wohnung ist so gestaltet, dass man sich in ihr barrierearm bewegen kann [Rahimi und Vogt, 2010].

Abbildung 3.4: Impressionen des Living Place Hamburg
(Oben links: Außenansicht; oben rechts: Rotunde; unten links: Blick in den Raum; unten rechts: Badezimmer)

3.3 Softwareinfrastrukturen für Ambient Intelligence

Das Design einer Middleware für Ambient Intelligence wird auch in einigen anderen Projekten behandelt. In diesem Abschnitt werden Projektaspekte präsentiert, die es erlauben, Rückschlüsse für die Entwicklung einer nachrichtenbasierten Architektur für Smart Homes zu ziehen. Es werden hier bewusst sehr unterschiedliche Modelle bespielhaft vorgestellt. Es soll dem Leser so ermöglicht werden, einen Eindruck der jeweiligen Entwicklungsrichtungen im Großkontext *Smart Homes* zu erhalten.

3.3.1 DynAMITE

Das DynAMITE-Projekt[5] des Fraunhofer IGD ist ein Peer-to-Peer-System für *Ambient Intelligence*. Hellenschmidt [2005b] beschreibt es als eine selbstorganisierende Middleware für *Ambient Intelligence* ohne eine zentrale Koordinationskomponente. Die Grundlage für dieses System bildet das in [Hellenschmidt und Kirste, 2004; Hellenschmidt, 2005a] beschriebene *SodaPop*-Konzept[6]. Es beschreibt eine zentralisierte Middleware, die es so genannten *Device Ensembles* ermöglicht, sich dynamisch finden und interagieren zu können. Des weiteren werden Konfliktlösungsstrategien aufgezeigt, wie sich widersprechende Anweisungen des Systems aufgelöst werden können.

Die Fokus des DynAMITE-Projekts liegt hierbei insbesondere auf dem Umgang mit häufig wechselnden Teilnehmern. Es wird davon ausgegangen, dass die Verbindungen zwischen Geräten häufig wechseln und so genannte *flüchtige Funktionsensembles* sich spontan zusammenfinden können, um den gewünschten Nutzen zu erfüllen.

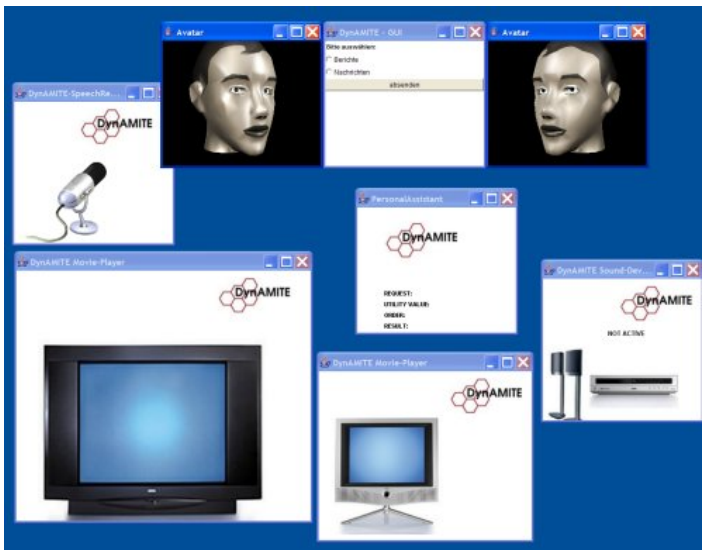

Abbildung 3.5: DynAMITE – Demo-Applikation
(Quelle: http://www.dynamite-project.org)

Von der Homepage des Projekts[7] lassen sich Beispielprogramme der Anwendung herunterladen. Teil der zur Verfügung stehenden Software ist auch eine Simulationsumgebung, die den Test der Architektur erlaubt, ohne das konkrete Geräteinstallationen vorhanden sind. Abbildung 3.5 zeigt einen Screenshot dieser Demo-Applikation.

[5] DynAMITE: Dynamisch Adaptive Multimodale IT-Ensembles
[6] SodaPop: Self-Organizing Data-flow Architectures suPporting Ontology-based problem decomPosition
[7] http://www.dynamite-project.org

3.3.2 Oxygen

Das Oxygen-Projekt des MIT [MIT Project Oxygen, 2004] stellt den Menschen und seine Interaktion mit Computern in den Mittelpunkt der Untersuchungen. Es wird hier versucht, ausgehend von den Bedürfnissen des Menschen, eine ganzheitliche Lösung zu schaffen. Anders als die anderen bisher vorgestellten Ansätze bezieht die Oxygen-Architektur bewusst auch die Außenwelt und Kommunikation über große Distanzen mit ein.

Die hierbei in Betracht gezogenen Geräteklassen umfassen sowohl Audio- und Videokommunikation, Handheld- und Embedded-Devices, als auch Mobiltelefone und stationäre PCs. Abbildung 3.6 zeigt einen Überblick der Komponenten.

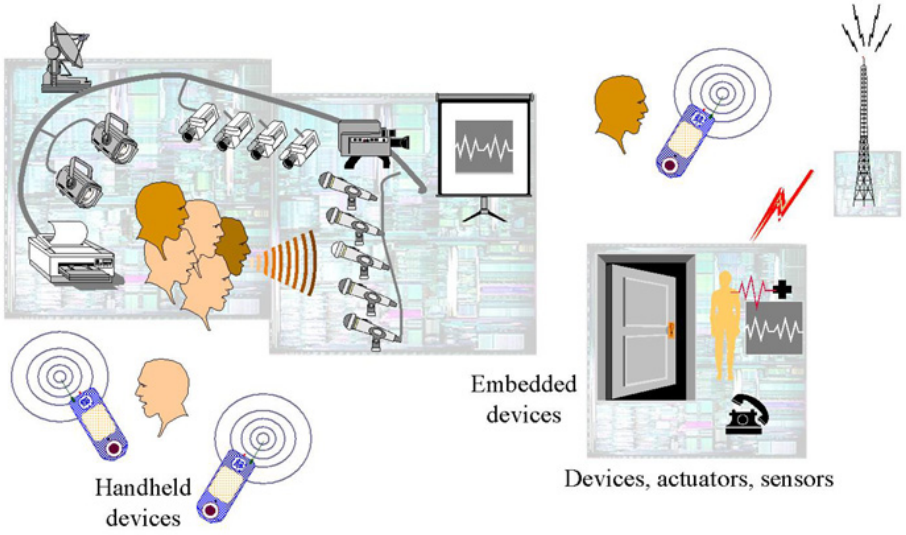

Abbildung 3.6: Überblick Oxygen
[Quelle: MIT Project Oxygen, 2004]

Das Oxygen Projekt teilt sich in folgende Bereiche auf: Oxygen device technologies, Oxygen network technologies, Oxygen software technologies, Oxygen perceptual technologies und Oxygen user technologies. Es wird versucht, alle wichtigen Aspekte des Human-Centered-Computing zu erfassen.

3.3.3 Personal Information Environments

Pierce und Nichols [2007] beschreiben in ihrem Projekt *Personal Information Environments* eine Architektur, mit deren Hilfe die immer präsenter werdenden Herausforderungen durch die Verteilung von Daten auf mehrere Geräte bewältigt werden sollen. Bei der Verwendung

heterogener Geräte ist nicht jedes Gerät für jede Form von Daten geeignet. Ein Mobiltelefon mit kleinem Display ist für gewöhnlich nicht gut geeignet, einen Satz Präsentationsfolien anzuzeigen. Für diese Aufgabe eignet sich eher ein großer Bildschirm. Mit dem Mobiltelefon ließe sich allerdings sehr gut steuern, auf welchem Gerät die Daten gespeichert werden sollen, so dass sie später betrachtet werden können. Die derartige Vernetzung aller persönlichen Geräte stellt den Grundgedanken einer *Personal Information Environment* dar. Eine Übersicht der Zusammenhänge zwischen Personen, Aktivitäten, Geräten und Daten zeigt Abbildung 3.7 in Form einer Mindmap.

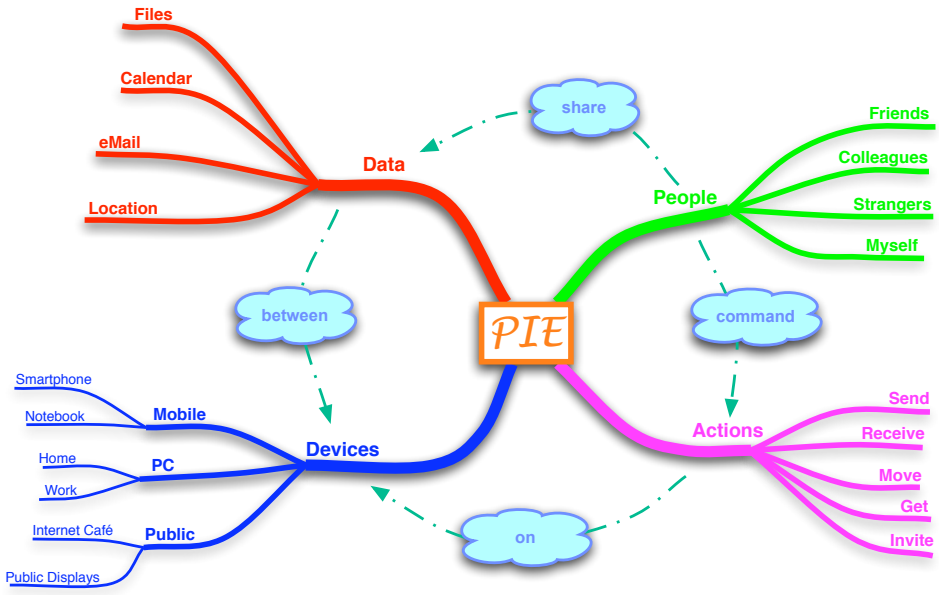

Abbildung 3.7: Personal Information Environment – Mindmap

Praktisch gesehen handelt es sich bei diesem System um einen *Instant-Messaging-Dienst* zur Gerätekommunikation. Die einzelnen Geräte erhalten durch diesen Dienst die Möglichkeit untereinander zu kommunizieren. Über diesen Dienst wird es dem Benutzer ermöglicht, Dokumente über mehrere Geräte hinweg zu verteilen und zu synchronisieren, entfernte Bildschirmausgaben zu steuern oder Nachrichten zu verschicken.

Die Kernpunkte einer *Personal Information Environment* können mit den Begriffen *Zugriff auf eigene Geräte*, *Zugriff auf fremde Geräte*, *Datei- und Informationsverteilung* und *Sicherheit* zusammengefasst werden:

Zugriff auf meine Geräte. In einer vollständig vernetzten Welt benötigt man einfachen Zugriff auf seine Geräte. Dieser soll entsprechend den Gedanken des *Ubiquitous Computing* verborgen im Hintergrund geschehen. In Pierce und Nichols [2007] wird eine

Umgebung als *Personal Information Environment* beschrieben, in der alle Geräte eines Benutzers miteinander kommunizieren und Daten austauschen können. Die Lautsprecher eines Raums können so z. B. verwendet werden, um ein Telefonat zu führen.

Zugriff auf fremde Geräte. Der Zugriff auf fremde Geräte gestaltet sich ähnlich zu dem Zugriff auf eigene Geräte. Dieser Fall ist wichtig, wenn beispielsweise Fotos von einem Handy auf einem großen Fernsehbildschirm dargestellt werden sollen.

Möglichkeit, Dateien und Informationen zu verteilen. Die Möglichkeit, Ressourcen anderer Geräte zu nutzen, soll auch in der Möglichkeit bestehen, auf die Dateien dieser Geräte zuzugreifen. Es wäre somit möglich, beispielsweise in der Küche auf die private Musiksammlung zuzugreifen und die Musik dort wiederzugeben.

Sicherheit. Ein System, das einen nahtlosen Zugriff sowohl auf Geräte, als auch auf Dateien ermöglicht, benötigt ein flexibles Sicherheitssystem. Es soll verhindern, dass Unbefugte Zugriff auf sensible Daten und Funktionen des Systems erlangen können und es ermöglichen, dass der Besitzer je nach Vertrauensstand anderen den Zutritt zum System erlauben kann.

3.3.4 Agentenbasierte Systeme

Unter dem Begriff Software-Agenten werden Computerprogramme zusammengefasst, die zu einem gewissen eigenständigen Handeln fähig sind. Ein Software-Agent wird von Jennings und Wooldridge folgendermaßen beschrieben:

> *„[A] self-contained program capable of controlling its own decision-making and acting, based on its perception of its environment, in pursuit of one or more objectives.“* – [Jennings und Wooldridge, 1996]

Aus dieser Definition leiten Jennings und Wooldridge die folgenden vier Charakteristika von Software-Agenten ab [s. a. Ndumu und Nwana, 1997]:

Autonomität. Die weitestgehend unabhängige Arbeitsweise des Programms. Ein Softwareagent sollte seine Aufgaben möglichst ohne die Steuerung durch einen Benutzer oder anderen Agenten erfüllen können.

Soziale Fähigkeiten. Die Fähigkeit mit anderen Agenten zu kommunizieren. Ein Software-Agent ist in der Lage, mit anderen Agenten in Kontakt zu treten, wenn es die Erfüllung seiner Aufgabe erfordert.

Reaktivität. Die Fähigkeit auf Änderungen der Umgebung zu reagieren. Die Umgebung kann z. B. die physische Welt, der Benutzer, eine Gruppe anderer Agenten oder auch das Internet sein.

Proaktivität. Die Fähigkeit Aktionen aus eigener Initiative zu starten. Ein Agent soll so in der Lage sein, zielgerichtete Handlungen auszuführen wann immer es ihm angebracht erscheint.

Die Entwicklung von Softwareagenten ist an keine besonderen technologischen Bedingungen geknüpft. Es bietet es sich dennoch an, ein Framework zur Entwicklung von agentenbasierten Systemen zu verwenden, da dieses die Entwicklung unter den o. g. Gesichtspunkten deutlich vereinfacht. Jade[8] ist ein Beispiel für ein solches Fraemwork. Es ist vollständig in Java implementiert und bietet Unterstützung der FIPA Agent Communication Language (ACL), ein Protokollstandard zur Kommunikation zwischen Agenten [Bellifemine u. a., 1999]. Jade stellt zwar keine Architektur für *Ambient Intelligence* im engeren Sinne dar, wird aber dennoch hier mit aufgeführt, da den Clients in einer intelligenten Umgebung die o. g. Kriterien für Software-Agenten zugesprochen werden können.

Eine Erweiterung für Jade ist Jadex. Die Erweiterung des Frameworks ermöglicht es, deliberative Agenten nach dem BDI-Modell zu implementieren. BDI steht dabei für *Belief, Desire und Intention*. BDI-Agenten haben ein Bild von ihrer Umgebung (Belief) und handeln selbstständig (Intention) auf der Grundlage von Plänen (Desire). Die Auswahl der Pläne geschieht selbsttätig auf Grundlage der Kombination ihrer Ziele und ihres Umweltwissens. Jade und Jadex können auch als verteilte Plattform für Agentensysteme verwendet werden, bei der die Agenten des Systems auf mehrere Rechner verteilt sein können [Pokahr u. a., 2003].

3.3.5 EventHeap

Der *EventHeap* ist Teil des Stanford iROS[9]. Er stellt als zentrale Kommunikationsinstanz einen der wichtigsten Bestandteile des Systems. Abbildung 3.8 zeigt den Aufbau des Systems aus seinen einzelnen Komponenten.

Das vom *Event Heap* verwendete Modell zur losen Kopplung geht auf ein als *Blackboard* bezeichnetes Architekturmodell zurück. Eine der ersten Implementierungen, einer Blackboard-Architektur beruht auf dem von Erman u. a. [1980] beschriebenen *Hearsay-II Speech-Understanding System*. Blackboard-Systeme funktionieren, wie bereits durch den Namen angedeutet, ähnlich einer Tafel oder Pinnwand. Sie bieten einen öffentlichen Datenspeicher, auf dem jedes angeschlossene Teilsystem[10] frei lesen und schreiben darf. Ein so genannter *Blackboard Monitor* beobachtet die auf den Daten des Blackboards durchgeführten Aktionen. Relevante Änderungsinformationen werden an einen *Scheduler* übergeben, der die Aktualisierungshinweise an die *Knowledge Sources* übermittelt, die daraufhin wieder selbst aktiv werden können (vgl. Abbildung 3.9).

[8] http://jade.tilab.com, Abrufdatum: 14.06.2010
[9] iROS: intelligent Room Operating System
[10] hier als *Knowledge Source* bezeichnet

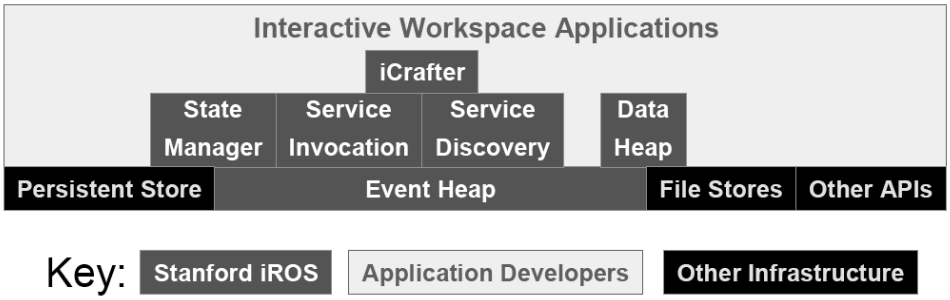

Abbildung 3.8: iROS Komponenten
[Quelle: Johanson und Fox, 2004]

Die *Knowledge Sources* bearbeiten und verändern die Daten auf dem Blackboard, wodurch ein neuer Zustand geschaffen wird, der wiederum von anderen *Knowledge Sources* weiterverarbeitet werden kann. Hearsay-II sollte so gesprochene Sätze analysieren und deren Bedeutung erkennen. Es gab demnach Knowledge Sources, die zunächst eine Hypothese über die Zerlegung in einzelne Satzteile und Worte vornahmen, die die jeweiligen Wave-Formen der Worte analysierten und sie dem System bekannten Vokabeln zuordneten und die darauf aufbauend versuchten, die Bedeutung des Satzes zu interpretieren.

Eine Weiterentwicklung der Blackboard-Architektur stellt das Tuplespaces-Modell dar, das auf den Ideen des Blackboard aufbaut und es um die Definition einer Reihe von Aktionen erweitert, die auf dem *Tuplespace* ausgeführt werden dürfen: Post, Read und Remove. (vgl. Abbildung 3.10). Ein Tuple stellt eine sortierte Menge von Schlüssel-Werte-Paaren dar, der zu einem Schlüssel hinterlegte Wert kann entweder leer sein oder auf einen konkretes Datum zeigen. Das ursprüngliche Tuplespaces-Modell wird in Ahuja u. a. [1986] beschrieben. Der ursprüngliche Zweck der Architektur ist es, die Koordination parallel laufender Prozesse zu ermöglichen. Implementationen von Tuplespaces sind z. B. Limbo [Davies u. a., 1997], JavaSpaces [Freeman u. a., 1999] oder T-spaces [Lehman u. a., 1995].

Das Modell der *EventHeap*-Architektur erweitert das Konzept von Tuplespaces (s.a. Abbildung 3.10). Ein Sender kann eine Nachricht in einem Tuplespace ablegen, mögliche Empfänger können eine Kopie der Nachricht abrufen oder die Nachricht vom *EventHeap* entfernen (Johanson und Fox [2002] und Johanson und Fox [2004][11]). Die Erweiterungen des *EventHeap* beziehen sich allerdings noch noch auf weitere Aspekte, wie z. B. die Möglichkeit, sich für Events zu registrieren, um bei deren Auftreten automatisch informiert zu werden. Die Anpassungen für den *EventHeap* im Vergleich zum *Tuplespace*-Modell wurden an die speziellen Anforderungen der eines Collaborative Workspace angepasst. Da sich das

[11] vgl. hierzu auch Erman u. a. [1980]

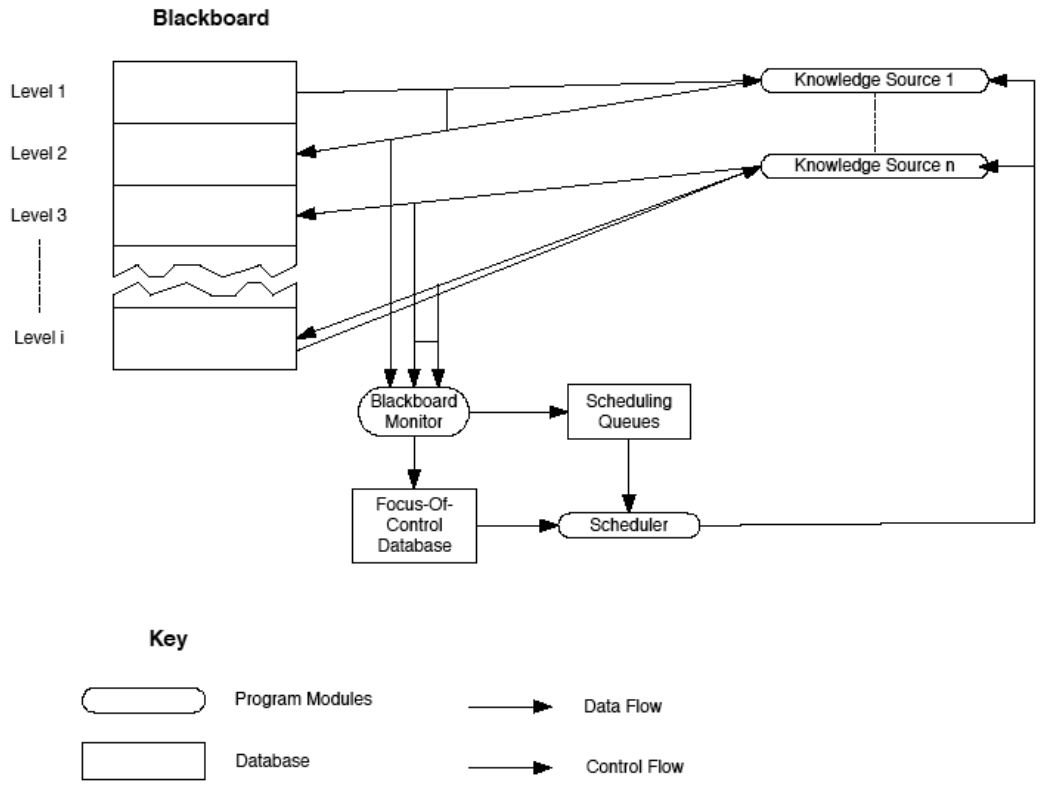

Abbildung 3.9: Schematische Darstellung einer Blackboard-Architektur
[Quelle: Erman u. a., 1980]

resultierende Modell erkennbar vom *Tuplespace*-Modell unterscheidet, bezeichnen Johanson und Fox die Tupel als *Events*. Die Anpassungen umfassen insbesondere die folgenden Punkte:

Selbstbeschreibend. Die Events sollen selbstbeschreibend sein, um bei der Fehlersuche leichter erkennen zu können, welcher Event wo verarbeitet bzw. erwartet wurde.

Flexible Typisierung. Die flexible Typisierung von Events und der enthaltenen Daten soll gewährleisten, dass Fehlinterpretationen ähnlicher Events nicht zu einem unerwarteten Verhalten führen. Sie sollen zudem ermöglichen, dass eine erweiterte Version eines Events auch noch verwendet werden kann, wenn die ursprüngliche Version erwartet wird[12].

Standard Routing Felder. sollen bewirken, dass gewisse Events nur an bestimmte Empfänger ausgeliefert werden. So ist die Weitergabe eines Events zur Darstellung von

[12] z. B. das Einfügen eines Felds Zoomfactor in ein Tupel MapLocation

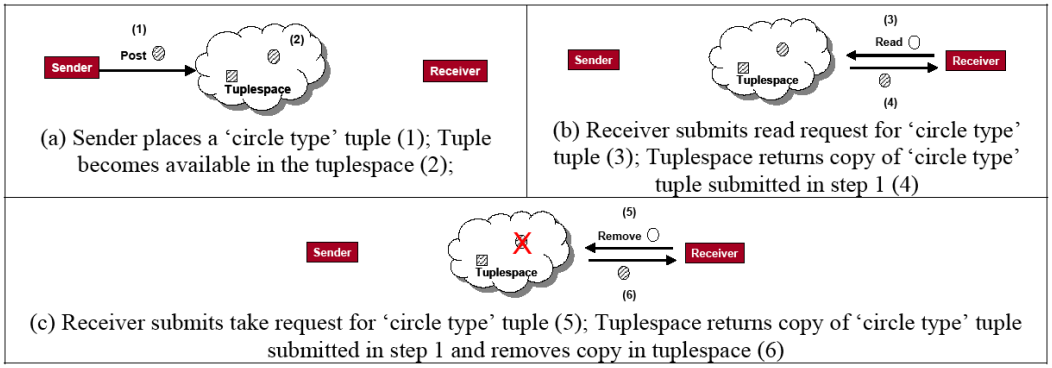

Abbildung 3.10: Grundprinzipien von Tuplespaces
(Quelle: Johanson und Fox [2004], S. 11)

Grafik an ein bildschirmloses Gerät in den seltensten Fällen sinnvoll und soll auf diese Weise verhindert werden können. Die Einstellung kann sowohl auf der Empfänger- als auch auf der Senderseite vorgenommen werden.

Begrenzte Lebensdauer von Events. Durch den im vorhergegangenen Punkt beschriebene Filtermechanismus können Situationen entstehen, in denen bestimmte Events nicht mehr vom Heap entfernt werden. Um dies zu verhindern, werden Events automatisch nach einer festgelegten Zeit vom Heap entfernt, die gewöhnlich vom Sender festgelegt wird.

Abonnieren von Events. Es ist Empfängern möglich, sich für bestimmte Ereignistypen beim *EventHeap* zu registrieren. Die Empfänger werden informiert, wenn ein entsprechendes Ereignis auf den Heap gelegt wurde.

FIFO, At Most Once. Im ursprünglichen Tuplespace Modell ist keine Reihenfolge festgelegt, in der ein Empfänger Tupel aus dem Tupelspace bekam. Es ist zudem möglich, dass dasselbe Tupel auch mehrfach an den selben Empfänger abgegeben wird. Im Gegensatz hierzu werden Events immer in der Reihenfolge ausgegeben, in der sie auf den Heap gelegt wurden. Ein bereits ausgelieferter Event wird kein zweites Mal an denselben Empfänger gegeben.

Modulare Inbetriebnahme. Der Ausfall eines Teils des Systems soll nicht dazu führen, dass das Gesamtsystem neugestartet werden muss. Es soll somit für jede Komponente (auch den Koordinator) möglich sein, nach einem Ausfall wieder in das System integriert zu werden.

3.3.6 Diskussion

In diesem Abschnitt wurden einige zunächst sehr unterschiedlich wirkende Bespiele für Softwareinfrastrukturen vorgestellt, die im Kontext eines Smart Homes eine Rolle spielen können. Die vorgestellten Projekte weisen jedoch auch viele Gemeinsamkeiten auf, da sie versuchen die Heterogenität der Verteilung auf einzelne Geräte vor dem Benutzer zu verbergen. Dieser Zustand wird – wie bereits zuvor herausgestellt – als *„fragmented with features“* bezeichnet [de Ruyter und Aarts, 2004][13].

Die vorgestellten Infrastrukturen und Frameworks versuchen diese Unterteilung bezogen auf ihren jeweiligen Anwendnungsfall zu minimieren. *DynAMITE*, *Oxygen* und *iROS* (Event Heap) schaffen eine Dienstinfrastruktur, die es den angeschlossenen Geräten möglich macht, flexible Dienste anzubieten und zu verwenden. Das Projekt Oxygen zielt dabei auf einen weit verteilten Kontext ab, der das System über die Grenzen von Gebäuden und Räumen hinweg konzipiert. Die von Pierce und Nichols entwicklelten *Personal Information Environments* integrieren ebenfalls den Einsatz mobiler Geräte durch den Benutzer, beschränken sich jedoch auf die Organisation von Daten und Dateien. *Jade* nimmt im Rahmen der vorgestellten Beispiele eine Sonderrolle ein, da es sich nicht um ein fertiges System, sondern vielmehr um ein Framework für Softwareagenten handelt. Jade bietet ein gutes Beispiel, da es ein standardkonformes Kommunikationsprotokoll (FIPA ACL) unterstützt und somit die Implementierung der Kommunikation zwischen Softwareagenten erheblich erleichtert. Die aus der Vorstellung der Beispiele gezogenen Erkenntnisse sollen in die weiteren Betrachtungen der Systemanalyse mit einbezogen werden.

3.4 Szenariobasierte Analyse

Im Folgenden sollen drei kurze Szenarien beschrieben werden, um das Bild einer intelligenten Wohnung weiter zu verdeutlichen. Die vorgestellten Szenarien stellen einen kleinen Ausschnitt der Möglichkeiten dar, die in den bestehenden Planungen, Abschluss- oder Seminararbeiten erarbeitet wurden. Sie zeigen, wie die Interaktion zwischen Mensch und System und den Systemkomponenten untereinander erfolgt. Es wird dabei auch die Kommunikation mit externen Dienstanbietern (z. B. übers Internet) mit berücksichtigt.

3.4.1 Wecker 2.0

Der Wecker 2.0 wird in [Ellenberg, 2010] als ein kontextsensitiver Wecker beschrieben, der in der Lage ist, die Weckzeit kontextbezogen für seinen Benutzer zu ermitteln. In Kombination

[13] vgl. hierzu auch Abschnitt 2.4

mit dem in [Hardenack, 2010] beschriebenen Sensorbett ergibt sich somit die Möglichkeit, einen Wecker so zu gestalten, das dieser aus einer Vielzahl von Informationen, wie z. B. aus dem Terminplaner und denen des aktuellen Schlafzustands die optimale Weckzeit ermittelt.

> Das Bett im Schlafzimmer einer intelligenten Wohnung ist mit Drucksensoren ausgestattet, die es in die Lage versetzen, Bewegungen während des Schlafes zu registrieren und so auch einzelne Schlafphasen zu unterscheiden. Diese Daten stehen einem intelligenten Wecker zur Verfügung, der diese und weitere Informationen nutzt, um den perfekten Weckzeitpunkt zu ermitteln. Die im Terminkalender eingetragenen Ereignisse liefern die Basis für die zu ermittelnde Weckzeit, der Ort und die Art des Termins bestimmen die benötigte Vorlaufzeit. Der Weckagent weiß[14], wie viel Zeit der Benutzer nach dem Wecken benötigt, um das Haus zu verlassen und kann unter Zuhilfenahme von Verkehrsinformationen (ÖPNV-/Bahnverbindungen oder Routenplanung mit Staumeldungen) die benötigte Dauer des Weges ermitteln. Aus diesen Informationen wird der spätestmögliche Weckzeitpunkt ermittelt. In einer bestimmten Zeitspanne vor dem Weckzeitpunkt beginnt der Weckagent die aktuelle Schlafphase zu überwachen, sobald sich die Person in einer Leichtschlafphase befindet, wird die Person geweckt. Sollte der spätestmögliche Weckzeitpunkt erreicht werden, so wird ebenfalls ein Weckereignis ausgelöst.

Der Wecker nutzt mehrere Dienste des Smart Homes. Es gibt allerdings zwei entscheidende Unterschiede: Zum Einen wird der Wecker indirekt durch das Eintragen von Terminen aktiviert und andererseits wartet der Wecker während des Wartezeitfensters auf das Eintreten eines komplexen Ereignisses. Die Ermittlung des Weckzeitfensters erfolgt zudem in hohem Maße unter Zuhilfenahme von externen Systemen. Es werden zu diesem Zweck Fremdsysteme, wie z. B. die aktuellen Verkehrswarnungen oder die Fahrplanabfrage des öffentlichen Personennahverkehrs über das Internet mit einbezogen.

Aus Entwicklersicht erfordern die beschriebenen Eigenschaften die Integration eines internen Benachrichtigungsdienstes. Es soll möglich sein, die beteiligten Komponenten über zuvor definierte Ereignisse anderer Komponenten informieren zu können. In diesem Fall würde das Speichern eines Kalendereintrags veranlassen, dass die Weckzeit aktualisiert wird. Zudem wäre es wünschenswert, wenn es für diese Art von Benachrichtigungen eine vereinbarte Beschreibungssprache gäbe, so dass nur relevante Ereignisse übertragen werden. In diesem Szenario würde es konkret bedeuten, dass der Wecker z. B. regelmäßig jeden Abend die Ereignisse für den kommenden Tag abfragt und aus den anstehenden Terminen die Weckzeit berechnet. Sollten im Anschluss noch neue Termine eingetragen werden, die die Weckzeit beeinflussen, so wird der Wecker über diese Änderungen informiert und kann

[14] z. B. durch Lernen oder Konfiguration

diese entsprechend anpassen. Während der Wartephase direkt vor dem Weckereignis wird der Wecker dann nur über relevante Änderungen der Schlafphase [15] informiert. Die transparente Einbindung von Fremdsystemen stellt dabei eine weitere Herausforderung dar. Es soll für den Anwendungsentwickler keinen besonderen Unterschied darstellen, ob mit einem internen Dienst[16] kommuniziert wird oder ein externer Dienst in Anspruch genommen wird. Auf der anderen Seite stellt das Verwenden externer Angebote auch ein Stabilitäts- und Sicherheitsrisiko dar, das es abzufedern gilt. Es kann beispielsweise geschehen, dass die Schnittstelle eines externen Dienstes geändert und somit unbrauchbar wird. Es können auf diesem Wege aber auch schadhafte oder falsche Daten und Programme in das System des Smart Homes eingeschleust werden.

3.4.2 Gegensprechanlage

Das Szenario beschreibt Ereignisse im Wohnzimmer und an der Eingangstür einer intelligenten Wohnung.

> Im Wohnzimmer wird ferngesehen. Das Licht ist gedimmt und das Telefon läutet nur für *wichtige* Anrufer. Diese Einstellungen treffen die betroffenen Agenten automatisch, sobald das entsprechende Programm eingeschaltet wurde und auf dem Sofa registriert wird, dass sich der Bewohner hingesetzt hat.
>
> Als es an der Tür klingelt, wird der Bewohner unaufdringlich über eine kurze Einblendung auf dem Bildschirm darauf hingewiesen. Er entscheidet sich, den Video- und Audio-Stream von der Gegensprechanlage direkt am Fernseher entgegen zu nehmen. Das laufende Programm pausiert und kann später über die Timeshift-Funktion des Fernsehers weiter gesehen werden. Auf dem Fernseher erscheint das Bild des Besuchers, über die Lautsprecheranlage und ein Mikrofon im Raum kann der Bewohner mit dem Besucher sprechen. Es handelt sich um einen guten Freund und er entscheidet sich, ihn hereinzulassen.
>
> Die Wohnzimmersteuerung nimmt das Öffnen der Tür als Signal, das Setup des Wohnzimmers von *Film* in *Besuch* zu ändern: Es wird das Licht eingeschaltet, passende Musik ausgewählt, sowie die Klingel und das Telefon auf *laut* gestellt. Der Besuch bleibt für ein paar Stunden, andere Sendungen, die der Bewohner üblicherweise ansieht, werden ebenfalls aufgenommen. Später, wenn der Besuch die Wohnung verlassen hat und der Bewohner erneut den Fernseher einschaltet, wird er auf die aufgenommenen Sendungen hingewiesen. Bei Bedarf schaltet das System zurück in das Profil *Film*.

[15] In diesem Fall: der Wechsel in eine Leichtschlafphase

[16] Hiermit sind Dienste gemeint, die in der Wohnung für die Wohnung angeboten werden.

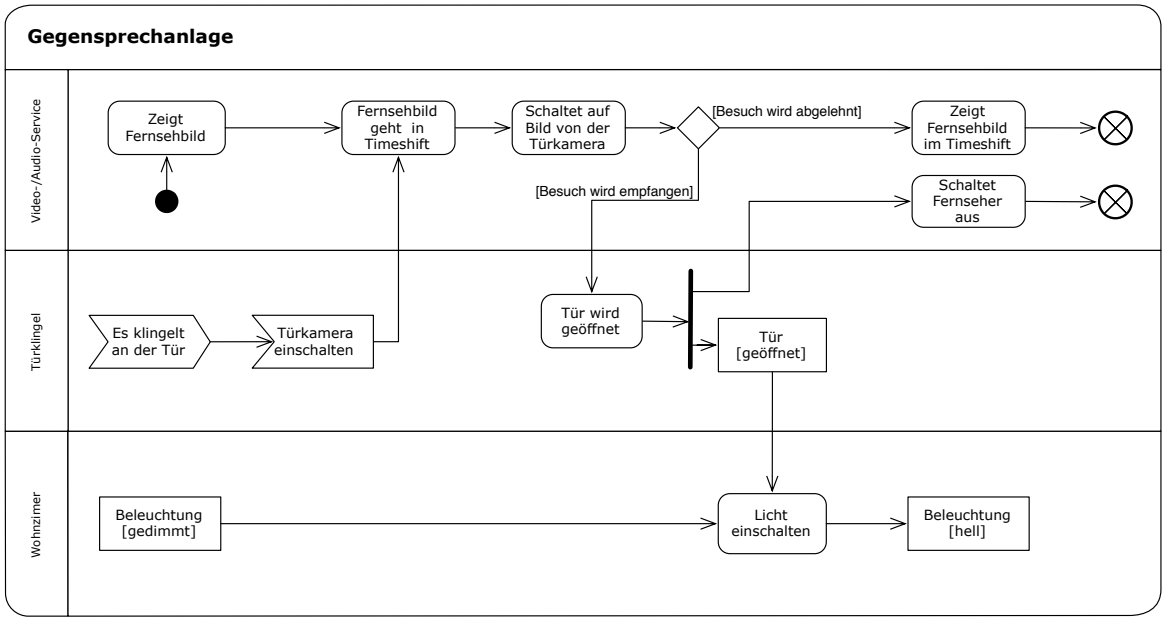

Abbildung 3.11: Aktivitätsdiagramm: Gegensprechanlage

Das Szenario beschreibt eine Reihe von Aktionen und Entscheidungspunkten, die Einfluss auf das Verhalten des Systems haben (vgl. Abbildung 3.11). Es sind hier eine Vielzahl vernetzter Geräte beteiligt, deren Verhalten nicht zentral gesteuert oder koordiniert wird. Es ist auch nicht klar geregelt, welche Geräte Anweisungen an andere Geräte geben dürfen. Jedes Gerät kann Anfragen stellen und so das Verhalten der übrigen Geräte beeinflussen. Zusätzlich werden typische Aktionen aus dem Verhalten des Bewohners abgeleitet.

Der Ablauf aus Systemsicht lässt sich in drei Situationen einteilen: Fernsehen, Gegensprechanlage und Besuch (s. Abbildung 3.12). Das System muss in der Lage sein, sicher aus den Aktionen des Benutzers zu erkennen, in welcher Situation es sich zum jeweiligen Zeitpunkt befindet. Der Modus Gegensprechanlage ermöglicht zusätzlich noch eine Unterscheidung in der Darstellungsform zwischen *privat* und *öffentlich*. Es kann dabei die Darstellung des Audio- und Video-Streams von der Gegensprechanlage beispielsweise über den Großbildschirm und die Audioanlage (öffentlich) oder über das Smartphone (privat) erfolgen.

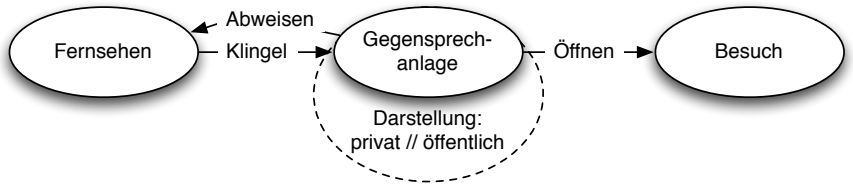

Abbildung 3.12: Situationen im Szenario Gegensprechanlage

Das Szenario gibt einen Überblick des Zusammenspiels einzelner Komponenten und Dienste. Der Fokus liegt dabei auf der automatischen und kontextsensitiven Aggregation von Diensten. Der Entwickler soll bei der Erstellung von Anwendungen von den Aufgaben der Kommunikation zwischen den Systemkomponenten entlastet werden. Er soll auch keine Kenntnis über die Service-Discovery-Methoden haben müssen. Diese zentralen Aufgaben sollen bereits in der Basisarchitektur des Systems verankert sein. Es werden zwischen den jeweiligen Anwendungen Prioritäten vergeben, nach denen das System entscheidet. Das Fernsehprogramm kann beispielsweise je nach Inhalt eine sehr hohe Priorität besitzen, die nur von der Priorität der Türklingel übertroffen wird. Die Systemarchitektur sollte diese Priorisierung von Nachrichten direkt berücksichtigen, da sie Einfluss auf alle Komponenten des Systems hat.

3.4.3 Cooking Agent

In Urich [2009] wird ein agentenbasiertes System beschrieben, das den Benutzer bei der Auswahl eines geeigneten Menüs für seine Gäste, sowie der Erstellung einer Einkaufsliste unterstützt.

> Zur Vorbereitung eines Essens mit Freunden soll vom Gastgeber eine Menüauswahl getroffen werden. Es sollen dabei die speziellen Vorlieben der Gäste, die Jahreszeit und der Anlass, sowie das vorhandene Budget berücksichtigt werden.
>
> Eine Reihe von Softwareagenten sind in der Lage, diese Aufgabe weitestgehend zu bearbeiten und entsprechend Vorschläge zu unterbreiten:
>
> - Der Kalenderagent kann Informationen bezüglich des Zeitpunkts, der Gäste und des Anlasses ermitteln.
> - Der Kühlschrankagent kennt die vorhandenen Vorräte[17].
> - Der Rezeptagent hat eine Datenbank mit möglichen Rezepten und kann diese den Vorlieben der Gäste und der Jahreszeit zuordnen.
>
> Die Arbeit der Basisagenten wird durch den Cooking-Agent initiiert, der gleichzeitig im direkten Kontakt mit dem Benutzer steht. Er ermittelt mit der Hilfe der anderen Agenten eine passende Rezeptauswahl und kann basierend auf der Auswahl des Gastgebers eine Einkaufsliste erstellen, die darauf dem Einkaufszettelagenten übergeben wird. Im Anschluss an das Essen kann der Cooking-Agent beim Gastgeber erfragen, ob den Gästen das Essen geschmeckt hat und die Bewertung für den nächsten Besuch speichern. Auf diese Weise lässt sich die Rezeptauswahl für den nächsten Besuch noch weiter verfeinern.

[17] vgl. hierzu auch Hollatz [2008c], Meißner [2008c] und Urich [2008b]

Auch in diesem Szenario wird die Zusammenarbeit in mehreren Teilsystemen beschrieben, die durch die Initiative des Gastgebers aktiv werden. Es wurde im Rahmen von Urich [2009] ein Prototyp erstellt, der in der Lage ist, die im Szenario beschriebenen Aufgaben zu erfüllen. Die Kommunikation der Agenten untereinander wird hier über den in Johanson und Fox [2002] beschriebenen Event Heap gewährleistet. Die Agenten untereinander sind lose gekoppelt, sie wissen lediglich, wie sie ihre Informationsanfragen an den Event Heap stellen, jedoch nicht zwingend, welche Instanzen diese bearbeiten. Die Kommunikation zwischen den Agenten folgt festgelegten Konventionen, das Kommunikationsprotokoll zur Erledigung gewisser Aufgaben wurde bereits zur Entwicklungszeit festgelegt.

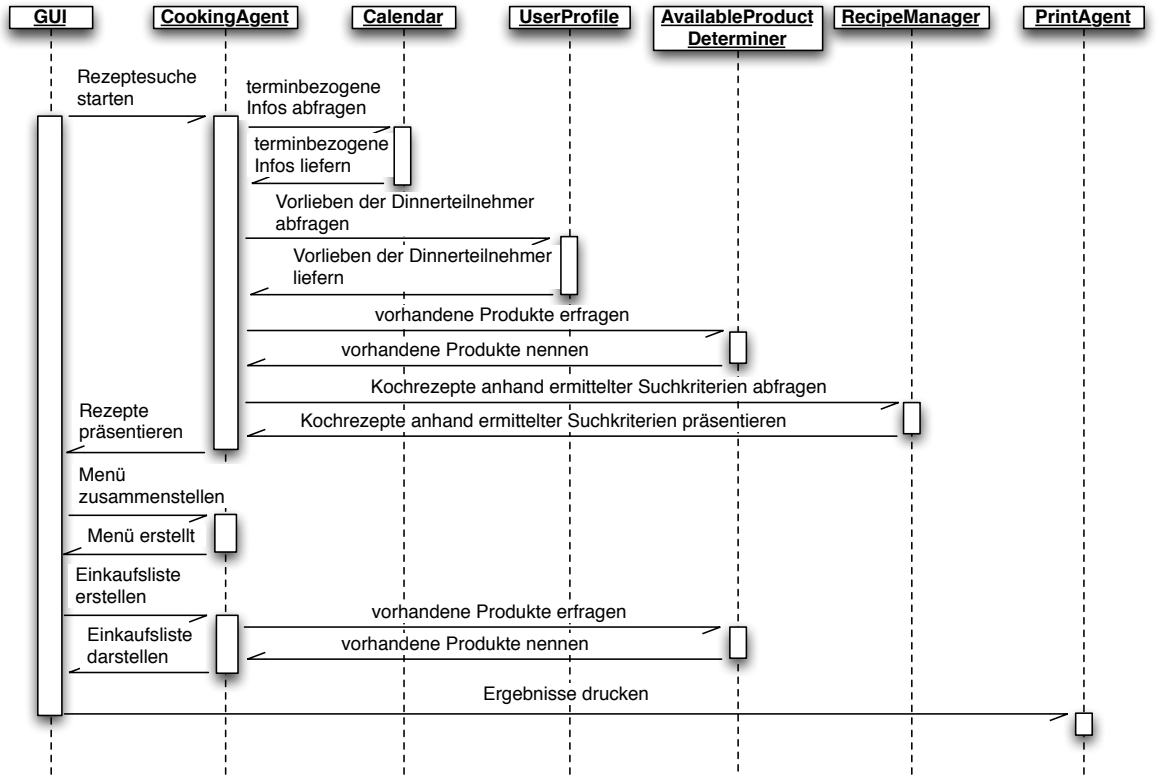

Abbildung 3.13: Sequenzdiagramm: Cooking Agent
[Quelle: Urich, 2009, S. 83]

Abbildung 3.13 fasst die beschriebenen Aktivitäten zusammen. Das Szenario spricht als neuen Aspekt die Verarbeitung von Daten an. Die Vorlieben der Gäste stehen hier im Vordergrund. Es werden Informationen aggregiert und der Gastgeber dadurch interaktiv in seiner Entscheidungsfindung unterstützt. Die Komponenten des Cooking Agent verwenden – falls erforderlich – jeweils eine private Datenbank, auf die nur sie selbst zugreifen können (vgl. [Urich, 2009, Abschnitt 5]). Die Wiederverwendbarkeit der gespeicherten Daten ist allerdings eine wünschenswerte und auch wichtige Anforderung an die Architektur. Es wird er-

wartet, dass die Abfrage von Daten eine der häufigsten Aktionen sein wird. Sie sollte daher über eine leistungsfähigen Abstraktion verfügbar sein, um das Anlegen redundanter Daten zu verhindern und somit die Konsistenz und Wartbarkeit der gespeicherten Informationen zu verbessern.

3.4.4 Zusammenfassung der Szenarien

Die drei vorgestellten Szenarien beleuchten einige Aspekte der in einen Smart Home möglichen Funktionalitäten. Sie stellen sowohl die Kommunikation innerhalb des Systems, sowie mit systemfremden Komponenten dar. Es wurden ebenfalls Aspekte der Datenverwaltung und -verarbeitung angesprochen. Die Abbildung 3.14 zeigt die Zusammenhänge der vorgestellten Komponenten. Sie verdeutlicht die Komplexität des Systems, das bisher lediglich drei Aspekte eines Smart Homes abdeckt.

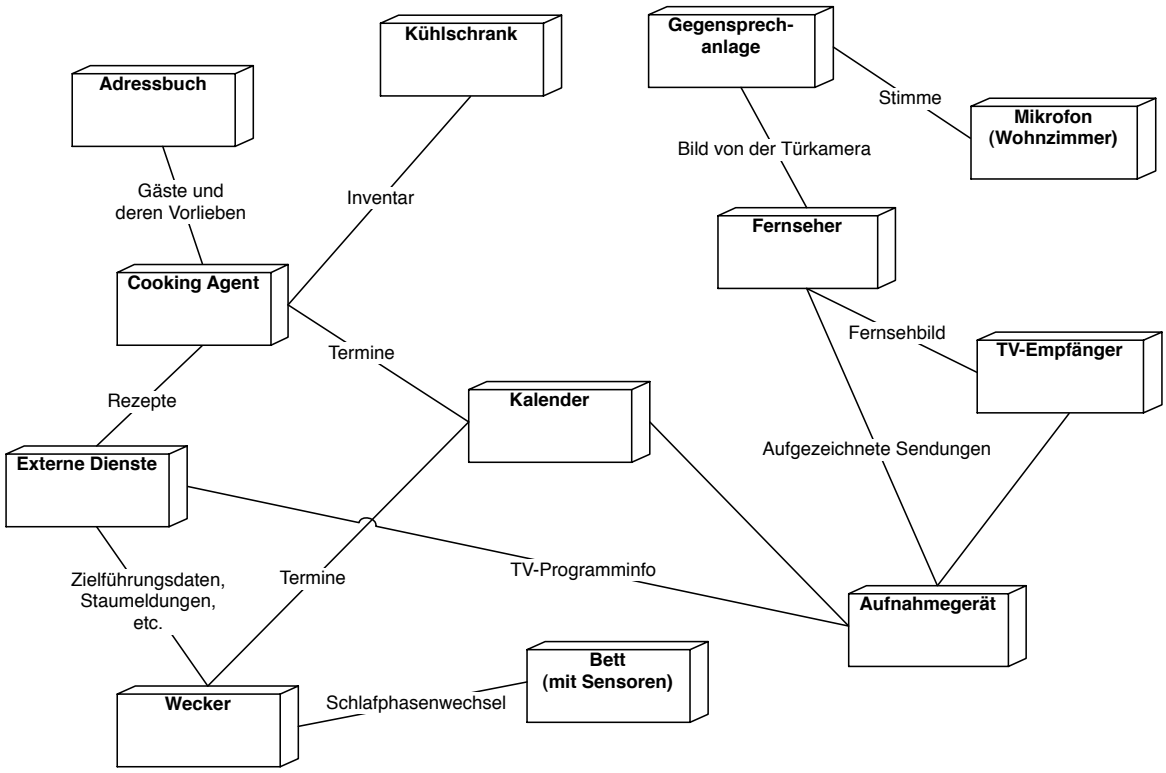

Abbildung 3.14: Concept-Map der Szenarien

Die in den Szenarien angesprochenen Komponenten sind fast ausschließlich auf Wiederverwendbarkeit ausgelegt. Im Sinne von Service Orientierten Architekturen [18] choreografiert

[18] s. a. Erläuterungen im Unterabschnitt „Middleware für Service-Oriented-Computing“ (S. 68f.)

ein Dienst mehrere andere Basisdienste, um ein Ziel zu erreichen. Die Basisdienste stehen dem aufrufenden Dienst allerdings nicht exklusiv zur Verfügung, sondern werden auch von anderen Diensten verwendet. Die eindeutige Definition von Schnittstellen ist daher sehr wichtig.

Die nahtlose Nutzung und die flexible Kombination sensibler Dienste erfordert ein Sicherheitsmodell, das eine systemweite Authentifikation und Autorisierung von Zugriffen erlaubt. Eine redundante Pflege von Zugriffskonten an jedem Endpunkt ist nicht zumutbar, so dass eine zentrale Verwaltung von Zugriffsrechten als sinnvoll erscheint.

3.5 Anforderungen aus Entwicklersicht

Aus den in Abschnitt 3.4 vorgestellten Szenarien werden im folgenden Abschnitt die Anforderungen eines Entwicklers abgeleitet. Es handelt sich dabei um die wichtigsten funktionalen Anforderungen an eine Middleware im Smart Home.

3.5.1 Kommunikationsrelevante Anforderungen

Die Architektur eines Smart Homes dient in erster Linie zur Unterstützung der Kommunikation zwischen den jeweiligen Diensten und Ressoucen. In diesem Abschnitt werden diesbezüglich Anforderungen formuliert, die das Ziel haben, eine flexible und transparente Kommunikationsgrundlage zu schaffen und gleichzeitig die Komplexität des Systemzugriffs für die Teilnehmer zu senken.

Empfangen von Ereignisinformationen. Das Verarbeiten und Weiterleiten von Ereignissen, die innerhalb des Systems auftreten, ist die Kernaufgabe des Systems. Einfache Ereignisse können beispielsweise aktiv aus Sensordaten, als „Schaltereignisse“ oder auch zeitgesteuert auftreten. Das System muss in der Lage sein, das Auftreten dieser Ereignisse in einer hohen Frequenz aufzunehmen und zu verarbeiten.

Flexible Zustellung von Ereignisinformationen. Auftretende Ereignisse sind für gewöhnlich ungerichtet. Sie beschreiben in der Regel Zustandswechsel, die wiederum von Interessenten dieser Ereignisse empfangen und verarbeitet werden. Ob es sich dabei um einen einzelnen oder eine ganze Gruppe von Empfängern handelt, ist zunächst nicht relevant. Im Einzelfall kann es für den Sender eines Ereignisses von Interesse sein, ob und wie ein Ereignis vom Empfänger behandelt wird. Es wird somit ein Mechanismus benötigt, über den der Sender (oder auch ein Beobachter) Einzelheiten der Verarbeitung der von ihm ausgehenden Ereignisse erhalten kann.

Erkennen komplexer Ereignisse. Ereignisse können nicht nur einfacher Natur sein. Es ist auch denkbar, dass das Auftreten mehrerer Ereignisse ein weiteres *komplexes* Ereignis hervorruft. So kann z. B. nach dem Aufstehen und Verlassen des Schlafzimmers das Weckerradio abgeschaltet werden und beim Betreten des Badezimmers das dortige Radio auf der selben Frequenz eingeschaltet werden.

Abfrage des aktuellen Zustands. Ein Großteil der in einem Smart Home auftretenden Ereignisse haben Einfluss auf oder beschreiben den Zustand von Entitäten. Um die Stabilität und Verlässlichkeit des Systems zu gewährleisten, müssen neu in das System integrierte Geräte in der Lage sein, auch ohne auf das Eintreffen eines Ereignisses, die aktuellen Zustände abzufragen. Eine neu installierte Klimaanlage kann sich so auf die gewünschte Temperatur einstellen, auch ohne dass diese vorher neu eingestellt werden muss.

Unterstützung von Sprechakten. Es ist nicht zu erwarten, dass die Kommunikation zwischen den Endgeräten lediglich mittels Ereignismitteilungen geschehen kann. Wie im Szenario „Gegensprechanlage“ beschrieben, kann es erforderlich sein, die Verwendbarkeit von (bereits belegten) Ressourcen auszuhandeln. Die Abbildung von Sprechakten, die es den Kommunikationspartnern erlauben, sich auf vorhergehende Nachrichten zu beziehen, erleichtern die Verarbeitung komplexer Ressourcenanfragen.

Transparenter Zugriff auf Dienste und Ressourcen. Die Komplexität des Gesamtsystems soll dem Programmierer möglichst verborgen bleiben. Bei der Entwicklung sollen Dienste verwendet und kombiniert werden können. Die verwendeten Dienste sind daraufhin selbstständig in der Lage, die benötigten Geräte entsprechend den Rahmenbedingungen auszuwählen. Die genaue Adressierung der Ressourcen wird daraufhin von den jeweiligen Diensten übernommen.

Verlässlichkeit. Es soll sichergestellt sein, dass Ereignisse garantiert verarbeitet werden, sofern sie es erfordern. Sollte ein Ereignis nicht verarbeitet werden können, so ist der Benutzer angemessen darüber zu informieren. Sollte z. B. am Abend das automatische Einstellen der Weckzeit nicht erfolgen, so ist der Benutzer möglichst noch vor dem Zubettgehen darauf hinzuweisen (s. Szenario „Wecker 2.0“).

In einer dynamischen Umgebung gibt es Unterschiede in der Bewertung von Verfügbarkeiten. Ein Mobiltelefon oder auch ein Notebook können nur erreicht werden, wenn sie sich auch im Netzwerk des Smart Homes befinden. Eine gegebenenfalls geplante Synchronisation von Kalender und Dokumenten kann jedoch auch zu einem späteren Zeitpunkt erfolgen. Ein Fernsehbildschirm kann nur verwendet werden, wenn er nicht schon bereits das Bild einer anderen Quelle anzeigt (s. Szenario „Gegensprechanlage“). Während eine nicht durchgeführte Programmierung des Weckers einen kritischen Ausfall signalisieren kann, ist es denkbar, dass die zuletzt genannten Beispiele bei sporadischem Auftreten zunächst ignoriert werden können.

3.5.2 Konfigurationsrelevante Anforderungen

Die Ausrichtung auf den Heim- und Laborbetrieb erfordert eine einfache Konfiguration. Im realen Leben hat ein Anwender kaum Zeit und Motivation, sich mit der komplizierten Einrichtung seiner Wohnung zu befassen. Im Laborbetrieb wird davon ausgegangen, dass die Konfiguration des Systems fortlaufend angepasst wird. Die nachfolgend formulierten Anforderungen zielen daher insbesondere auf die Vereinfachung des Konfigurationsprozesses ab.

Netzwerkintegration, Zero-Config. Insbesondere im Laborbetrieb wird damit gerechnet, dass sehr häufig neue Geräte und Dienste in das System integriert werden, bzw. wieder aus dem System herausgenommen werden. Es ist daher von großer Wichtigkeit, dass dieser Schritt einfach und robust funktioniert. Idealerweise fügt sich ein Gerät gänzlich konfigurationslos oder mit Hilfe nur weniger Konfigurationsschritte in das System ein. Es ist so möglich, Lösungen einfach zu testen und gegebenenfalls auf einfache Weise im iFlat weiter zu evaluieren.

Kombination von Diensten, Meta-Dienste. Dienste und Ressourcen sollen frei miteinander kombinierbar sein. Die Kombination mehrerer Sensoren, Aktoren und Dienste soll auch ohne die physische Repräsentation durch ein Objekt (z. B. Kühlschrank) möglich sein (vgl. Weckdienst und Kalender im Szenario „Wecker 2.0“ oder das Zusammenfassen von Diensten im Szenario „Cooking Agent“)

Beschreibung von Diensten und Ressourcen. Um eine abstrakte Kombination von Diensten zu ermöglichen, sollen die Eigenschaften von Diensten beschrieben werden können. So ist es möglich, auch alternative Dienste finden zu können, sollte ein Dienst einmal nicht verfügbar sein. Es wäre beispielsweise auch denkbar, dass die Vorschläge nach einem alternativen Dienst von einer eigenen Komponente erstellt werden.

Einbindung externer Dienste. Es ist denkbar, dass die Funktionalität durch Fremdsysteme erweitert wird. Diese Systeme sind nicht Teil des eigenen Netzwerks, sondern werden z. B. über das Internet eingebunden. So kann beispielsweise ein Wetter-, Stau- oder die Routenplanungsdienst bei der Ermittlung der Weckzeit herangezogen werden („Wecker 2.0“)., Die Abfrage nach Sonderangeboten im Lebensmittelgeschäft um die Ecke wird mit in die Ermittlung von Rezepten einbezogen („Cooking Agent“) oder Informationen über das Fernsehverhalten in Kombination mit den Daten anderer Fernsehzuschauer ermöglichen es dem Videorecorder, bestimmte Sendungen als „Vorschlag“ aufzunehmen (vgl. [Dreyer, 2009], am Rande auch das Szenario „Gegensprechanlage“).

Einteilung in virtuelle Netze. Die Einteilung in virtuelle Netzwerke soll es ermöglichen, dass im Laborbetrieb einzelne Systemkomponenten isoliert in einer kontrollierten Umgebung getestet werden können, um so Fehlerquellen sicher ausschließen oder identifizieren zu können. Die flexible Auswahl der für einen Dienst oder eine Ressource sichtbaren Systemteilnehmer erscheint daher mehr als sinnvoll.

Kommunikation mit Gastsystemen Die Einbindung von Gastgeräten in das System zur Erfüllung eines bestimmtes Zwecks (z. B. Anzeige von Fotos, die auf einem Mobiltelefon gespeichert sind) soll spontan möglich sein. Einem Gastgerät soll ohne besonderen Konfigurationsaufwand die Möglichkeit zur Interaktion innerhalb eines eng definierten Rahmens gegeben werden.

3.5.3 Sicherheitsrelevante Anforderungen

Ein System, das so eng mit unserem Leben verbunden sein soll, wie ein Smart Home stellt besonders gesteigerte Anforderungen an Sicherheit, Verlässlichkeit und Schutz vor Manipulation. Die Allgegenwärtigkeit von Computern und die vereinfachten Bedienungsmethoden erschweren die Absicherung des Systems gegen Schadsoftware und erleichtern es der selbigen somit, unser Netzwerk zu befallen Eckert [2007].

In diesem Abschnitt werden die zentralen sicherheitsrelevanten Anforderungen formuliert, die ein solches System erfüllen sollte. Die vorgestellten Anforderungen seien als Ergänzung zu den allgemeinen Schutzzielen „Authentizität“, „Datenintegrität“, „Informationsvertraulichkeit“, „Verbindlichkeit“, sowie „Anonymisierung und Pseudomisierung“ [Eckert, 2009, Kapitel 1.2] für Informationssysteme zu sehen. Die in der folgenden Liste aufgeführten Anforderungen entsprechen den wichtigsten Anforderungen, die sich mittelbar aus den Anwendungsszenarien ableiten lassen.

Authentifikation. Die Authentifikation hat zum Ziel, dass sich weder fremde Benutzer, noch fremde Geräte mit dem System verbinden und so interagieren können. Bei der Umsetzung dieser Anforderung ist es wichtig, dass sie einen sicheren Schutz gegen fremde Zugriffe schafft, aber gleichzeitig die Anforderung nach einer einfachen Integration mit dem Netzwerk nicht zu sehr einschränkt.

Schutz privater Daten. Die Absicherung gegen Fremdeinwirkung nimmt an Gewicht zu, je mehr persönliche Daten über die Bewohner des Haushalts gespeichert werden. Die Sicherheit der Daten, auch vor Mitgliedern des gleichen Haushalts, muss gewahrt sein. Die Schaffung von Transparenz bezüglich der Herkunft abgeleiteter Daten ist wünschenswert.

Rechtevergabe und Sichtbarkeiten. In einem System, das mehreren Personen und auch fremden Diensten Zugriff auf Daten und Ressourcen erlaubt, ist es sinnvoll, die Vergabe von Rechten und Sichtbarkeit sensibler Daten zu steuern. Fremdsystemen soll nur der Zugriff auf das System gewährt werden, den sie zur Erfüllung ihrer Aufgabe dringend benötigen. Das gleiche gilt für Gastgeräte, die vorübergehend Zugriff auf bestimmte Ressourcen des Systems erhalten.

Die Vergabe von Rechten hat allerdings auch zum Zweck, dass die Lokalität von Aktionen gewahrt bleibt. Beispielsweise könnte man allen Lichtschaltern den Zugriff auf „sich nicht im selben Raum befindliche“ Lampen verweigern (Sandboxing).

3.5.4 Nichtfachliche Anforderungen

Zusätzlich zu den bereits aufgeführten Anforderungen existieren noch eine Vielzahl nichtfachlicher Anforderungen an das System. Die allgemeinen Anforderungen, wie beispielsweise Skalierbarkeit, Ausfallsicherheit, etc. sollen hier nicht näher beschrieben werden. Für eine tiefergehende Auseinandersetzung mit diesen Themen sei auf die einschlägige Fachliteratur [Coulouris u. a., 2005; Tanenbaum und Steen, 2006] verwiesen. In der nachfolgenden Liste sollen ein paar Besonderheiten herausgestellt werden, die sich durch die besondere Ausrichtung des Systems ergeben.

Stabilität der Schnittstellen. Die Ausrichtung des Systems als Labor- und Testumgebung bedingt häufige Änderungen der Systemumgebung. Es ist bei der Spezifikation der Schnittstellen darauf zu achten, dass diese flexibel und umfassend sind, da eine nachträgliche Änderung an den Schnittstellen Einfluss auf jeden Dienst innerhalb des Systems haben kann. Der dadurch erwartete zusätzliche Integrations- und Testaufwand sollte somit dringend vermieden werden.

Minimierung der Anzahl entfernter Aufrufe. Der Aufruf von lokalen Methoden ist in der Regel sehr schnell. Im Gegensatz hierzu sind entfernte Methodenaufrufe (z. B. RMI, RPC, Webservices) durch den entstehenden Overhead von Marshalling, Un-Marshalling, sowie Netzwerklatenzen sehr langsam. Es sollte zur Bewahrung der Leistungsfähigkeit des Systems darauf geachtet werden, dass der Aufruf entfernter Methoden nur sehr sparsam und effizient geschieht [Fowler, 2002, Kapitel 7.1].

3.6 Zielsetzung: Eine Middleware für Smart Homes

Es soll im Rahmen dieser Arbeit eine Architektur entworfen werden, die die effiziente Entwicklung von Software für ein Smart Home ermöglicht. Es sollen zu diesem Zweck die größ-

ten Hürden bei der Entwicklung verteilter Systemene entschärft werden und dem Entwickler Werkzeuge bereitgestellt werden, mit denen er sich verstärkt auf die Umsetzung der Kernfunktionalitäten konzentrieren kann.

Aus der Analyse der Szenarien geht hervor, dass zwischen den Endpunkten des Systems eine hohe Frequenz ungerichteter Nachrichten (Ereignissinformationen) übertragen werden. Die Datenmenge der jeweils übertragenen Nachricht ist dabei als sehr gering einzustufen. Ein Versender kann dabei im Regelfall nur unter großem Aufwand die Empfänger seiner Nachricht ermitteln. Aus diesem Grund soll das System eine Umkehr der Adressierung implementieren: nicht der Sender der Nachricht bestimmt den Empfänger, sondern der Empfänger wählt die Nachrichten aus, die ihn interessieren. Diese Form einer Publisher-/Subscriber-Architektur wurde Bereits am Beispiel des Event-Heap (Abschnitt 3.3.5) erläutert. In dieser Publisher-/Subscriber-Architektur werden Aufträge und Anfragen ungerichtet an eine zentrale Komponente des Systems gesendet. Diese Komponente steuert den Nachrichtenverkehr und verwaltet die Regeln für Auswahl der Empfänger.

Für gewöhnlich werden nachrichten asynchron versendet und der Sender erwartet keine Rückantwort. Es kann jedoch vorkommen, dass für manche Ereignisse besondere Anforderungen für die Beantwortung von Anfragen bestehen. So kann z. B. geregelt sein, dass auf das Einschalten der Herdplatte nach spätestens vier Stunden automaisch ein Ausschaltereignis gesendet wird, um die erhöhte Brandgefahr durch ein versehentliches vergessen der eingeschalteten Herdplatte ausschließen zu können. Ein zweiter wichtiger Punkt ist die Datenhaltung, die grundsätzlich durch die beteiligten Komponenten selbst erfolgen soll (vgl. Abschnitt 3.4.3). Einen weiteren wichtigen Punkt stellen die Möglichkeiten des Testens und Debuggings verteilter Anwendungen dar [Kakousis u. a., 2010]. Es ist für den Entwickler eines verteilten Systems schwierig, die Kommunikation zwischen einzelnen Komponenten nachzuvollziehen. Diese Anforderung erfordert häufig umfangreiche Hilfsimplementationen an den Endpunkten der Kommunikation.

Diese Aufgaben lassen sich durch die Einführung einer geeigneten Middleware abdecken, die eine Komponente zur Dienstfindung bereitstellen und die Zustellung der Nachrichten garantieren kann. Die Kommunikation, sowie deren Adressierung läuft dann für gewöhnlich über diesen zentralen Knoten, wodurch eine deutliche Verringerung der Komplexität – insbesondere der Client-Logik – ermöglicht wird. Es wird so eine Möglichkeit zur Beobachtung des Nachrichtenflusses an einer zentralen Stelle geschaffen, die den Entwicklern während der Entwicklungsphase wichtige Informationen bereitstellt.

Die Middleware soll den Entwicklungsprozess neuer Anwendungen beschleunigen. Es ist nicht das Ziel, mit der Middleware alle Funktionalitäten eines Smart Homes abzubilden, sie soll jedoch flexibel genug sein, um an künftige Anforderungen angepasst zu werden. Die weitere Betrachtung konzentriert sich daher auf die Sicht der Systementwicklung, da diese die Zielgruppe zur Verwendung der Basisarchitektur repräsentiert. Es ist auf dieser Ebene

der Architektur nur bedingt erforderlich und möglich, auf die Anforderungen der Endbenutzer einzugehen, da diese nur mittelbar mit der Architektur des Systems in Berührung kommen. Sie nehmen das System durch das Verhalten der Endanwendungen wahr. Basierend auf den in Coulouris u. a. [2005] aufgeführten Transparenzkriterien für verteilte Systeme soll die tatsächliche Art und Weise der Verteilung des Systems für den Endanwender nicht von Bedeutung sein.

4 Das Smart Home als verteiltes System

Die Analyse in Kapitel 3 hat gezeigt, dass ein Smart Home ein massiv verteiltes System ist. Die Anforderungen an die Implementierung verteilter Systeme unterscheiden sich bedeutend von denen isolierter, nichtverteilter Anwendungen. Coulouris u. a. [2005] identifizieren sieben Aspekte als die größten Herausforderungen des Designs und der Umsetzung eines verteilten Systems:

1. **Heterogenität.** Der Einsatz einer Vielzahl unterschiedlicher Netzwerkprotokolle, Betriebssysteme, Hardware-Komponenten, Programmiersprachen oder Implementationen führt zu teilweise erheblichen Unterschieden zwischen den beteiligten Geräten. Die Folgen der Heterogenität können durch die Vereinbarung von einheitlichen Protokollen, sowie durch den Einsatz von Middleware vermindert werden.

2. **Offenheit.** Die Erweiterbarkeit verteilter Systeme ist ein wichtiges Merkmal. Durch die Spezifikation von offenen Schnittstellen können Endgeräte ihre Dienste anbieten. Die Schaffung einer einheitlichen Datenrepräsentation für die beteiligten Komponenten, sowie die Korrektheit der Funktionsweise des Systems stellen allerdings eine große Hürde dar.

3. **Sicherheit.** Die Absicherung des Systems gegen Fremdeinwirkungen wird um so wichtiger, je zugänglicher ein System für *fremde* Personen und andere Systeme ist. Dabei gilt es nicht nur, die öffentlichen Schnittstellen gegen eine unbefugte Benutzung abzusichern, sondern generell die fortlaufende Funktionsfähigkeit des Systems zu gewährleisten. Es gelten in diesem Zusammenhang die allgemeinen Schutzziele für IT-Systeme (s. a. Abschnitt 3.5.3).

4. **Skalierbarkeit.** Verteilte Systeme sollen mit der Zahl der Zugriffe und Benutzer möglichst unterlinear skalieren können. Es sollen Engpässe vermieden werden,indem z. B. eine hierarchische Strukturierung der Datenhaltung gewählt wird. Replikation häufig benötigter Datensätze kann ebenfalls von Vorteil sein.

5. **Fehlerbehandlung.** Es besteht zu jeder Zeit und bei jedem beteiligten Gerät in einem Netzwerk die Gefahr, dass es ausfällt. Aus diesem Grund sollten die Komponenten eines verteilten Systems mit diesem Umstand umgehen können und alternative Handlungsmöglichkeiten haben, falls eine kritische Ressource ausgefallen ist.

6. **Nebenläufigkeit.** Durch die Verteilung der Ressourcen und der Möglichkeit des parallelen Zugriffs auf Ressourcen, kann es geschehen, dass Ressourcenzugriffe in Konkurrenz zueinander stehen. Die betroffenen Ressourcen müssen in der Lage sein, damit umzugehen und so einen konsistenten Zustand bewahren.

7. **Transparenz.** Mit der Schaffung von Transparenz wird erreicht, dass bestimmte Aspekte der Verteilung bei der Programmierung nicht mehr wahrgenommen werden. Es soll dem Anwendungsentwickler so ermöglicht werden, sich auf die Problemstellung zu konzentrieren, ohne durch die Heterogenität des Systems abgelenkt zu werden.

In den folgenden Abschnitten werden die Herausforderungen der Modellierung nebenläufiger Systeme, der Maskierung von Heterogenität mit Hilfe einer Middlewareschicht, sowie der Verarbeitung komplexer Ereignisse behandelt.

4.1 Modellierung nebenläufiger Systeme

Die nebenläufige Ausführung von Prozessen ist ein Merkmal verteilter Systeme. Nebenläufigkeit gibt es allerdings auch in nichtverteilten Systemen. Ein System wird als parallel bezeichnet, wenn seine Ausführungsreihenfolge bestenfalls als *partielle Ordnung* beschrieben werden kann [Ghosh, 2006]. Nach dieser Definition können verteilte Systeme als eine Sonderform paralleler Systeme gesehen werden. Ghosh weist darauf hin, dass diese Sichtweise im Detail zu Problemen führen kann, da beide Systemtypen ungleichen Ansprüchen gerecht werden sollen. So ist es das Ziel von Designparadigmen verteilter Systeme, die Heterogenität der Verteilung zu handhaben, während die Zielsetzung paralleler Systeme eher auf dem Aspekt des Effizienzgewinns durch verteilte Programmausführung liegt. Das beide Konzepte aber dennoch miteinander vereinbar sind, zeigt beispielsweise Googles[1] sowohl verteilt, als auch parallel arbeitendes MapReduce [Dean und Ghemawat, 2004].

Formale Beschreibung

Die Programmierung nichttrivialer nebenläufiger Systeme gestaltet sich häufig schwierig. Fehlerfälle werden durch die vielen Möglichkeiten in der partiellen Ordnung von Abläufen provoziert. Selbst erfahrene und vorsichtige Programmierer sind davon nicht ausgenommen Sutter und Larus [2005]. Die Schwierigkeiten liegen dabei laut Lee [2006] nicht zwingend in der Nebenläufigkeit an sich, da Menschen es gewohnt seien, nebenläufige Prozesse in der realen Welt zu beobachten und zu beeinflussen. Die Schwierigkeiten würden oftmals aus der Wahl der unübersichtlichen Abstraktionsform in Form von Threads resultieren. Lee schlägt

[1] Website: `http://www.google.com`, Abruf: 14.06.2010

als alternative Repräsentationsform für nebenläufige Prozesse eine grafische Notation (analog zu UML) vor.

Andere Möglichkeiten zur formalen Beschreibung nebenläufiger Systeme sind Petri-Netze. Sie dienen der Beschreibung nebenläufiger Prozesse in der Form endlicher gerichteter Graphen [Petri, 1962]. Auf dem Gebiet der Petrinetze wurden in den vergangenen Jahrzenten viele Erweiterungen zu ihrer ursprünglichen Form entwickelt, die die Modellierung und Simulation nebenläufiger Systeme auf einer höheren Abstraktionsebene erlauben [Reisig, 2010]. Eine weitere Formalisierungsform sind Prozessalgorithmen, wie z. B. Communicating Sequential Processes (CSP). Mit Hilfe von CSP lässt sich die komplexe nebenläufige Interprozesskommunikation unabhängiger Systeme auf einer hohen Abstraktionsebene beschreiben. Tony Hoare entwarf CSP zunächst als eine imperative Programmiersprache [Hoare, 1978], entwickelte allerdings später zu einer formalen Prozessalgebra weiter [Brookes, Hoare und Roscoe, 1984].

Das Aktor-Modell

Das von Hewitt u. a. [1973] vorgestellte Aktormodell ist ein Modell zur Beschreibung nebenläufiger Prozesse. Die in ihm enthaltenen Paradigmen dienen der Modellierung nebenläufiger Systeme. Ein Aktor hat die Möglichkeit (1) Nachrichten zu empfangen, (2) lokale Entscheidungen zu treffen, (3) andere Aktoren zu erstellen und (4) Nachrichten an andere Aktoren zu senden. Die Kommunikation zwischen Aktoren ist nachrichtenbasiert und somit asynchron. Ein Aktor führt den Programmablauf direkt nach dem Versenden einer Nachricht fort und blockiert so auch bis zum Eintreffen einer Rückmeldung. In einer weiteren Präzisierung beschreiben [Hewitt und Baker, 1977] weitere Aspekte von Event-getriebenen Aktorsystemen, wie beispielsweise die partielle Ordnung von Ereignissen oder die Erzeugung von Aktoren. Das Aktormodell wird von einigen modernen Programmiersprachen, wie z. B. Erlang (s. Armstrong [1997][2]) oder Scala[3] direkt abgebildet.

4.2 Middleware in verteilten Systemen

Eine Middleware stellt in einem verteilten System eine Abstraktionsschicht des Programmiermodells dar. Sie schafft Transparenz bei der Kommunikation in verteilten Systemen, da mit ihrer Hilfe die Heterogenität der zugrunde liegenden Komponenten, wie z. B. des Netzwerks, der Hardware, Betriebssysteme oder Programmiersprachen gekapselt wird (Coulouris u. a. [2005]). Die Middleware verwaltet dabei nicht die Ressourcen der einzelnen Knoten, diese

[2] Eine ausführliche Einführung findet sich in Armstrong u. a. [1996].
[3] The Scala Programming Language: http://www.scala-lang.org/, Abruf: 20.06.2010

Aufgabe bleibt dem Betriebssystem überlassen. Sie stellt allerdings eigene Schnittstellen bereit, die die Verwendung von plattformeigenen Schnittstellen durch die verteilte Anwendung weitestgehend verhindern sollen (Tanenbaum und Steen [2006]).

In diesem Abschnitt wird eine kurze Einordnung des Themas *Middleware* gegeben. Zu diesem Zweck werden zunächst mögliche Ausprägungen als Beispiele für den Einsatz von Middleware in verteilten Systemen gegeben und anschließend gezeigt, in welchen Systemen der Einsatz einer Middleware nur bedingt sinnvoll ist.

4.2.1 Einordnung

In [Coulouris u. a., 2005] wird die Middleware-Schicht in einem verteilten System zwischen der Plattform (Betriebsystem und Hardware) und der Anwendungsschicht eingeordnet. Sie dient somit zunächst in erster Linie der Abstraktion der zugrundeliegenden Hardware (vgl. Abbildung 4.1). Der Programmierer nutzt beim Einsatz einer Middleware nur noch die von ihr angebotenen Funktionalitäten und gestaltet seine Anwendung somit plattformunabhängig. Die Kommunikation im Netzwerk und die über dieses angebotenen Dienste werden einheitlich transparent über die Middleware angesprochen. Der Einsatz einer Middleware zur Schaffung von Transparenz in verteilten Systemen wird von Tanenbaum und Steen [2006] noch einmal deutlicher hervorgehoben. In Abbildung 4.2 wird gezeigt, wie eine Anwendung auf einer über mehrere Rechnersysteme verteilt ist. Die Verteilung des Systems wird dabei durch den Einsatz der Middleware verborgen. Der Anwendungsentwickler ereicht somit im Idealfall die Verteilung seiner Anwendung kann, ohne sich mit den Besonderheiten verteilter Systeme auseinandersetzen zu müssen.

Die Kategorisierung von Middleware ist in der Literatur nicht eindeutig. In [Coulouris u. a., 2005] wir eine stark objektorientierte Sicht auf das Thema Middleware gegeben, die vorgestellten Beispiele beschränken sich nahezu ausschließließlich auf dieses Gebiet. Tanenbaum und Steen [2006] hingegen geben eine eher historische Einordnung von Paradigmen. Sie stellen die historeische Entwicklung im Bereich der Middleware für verteilte Anwendungen beginnend mit dateibasierten Zugriffen in verteilten Dateisystemen, über RPC (Remote Procedure Calls), bis hin zu verteilter Objekthaltung vor. Eine thematische Aufgliederung der Klassifikation geben Emmerich u. a. [2008]. Diese soll im Folgenden kurz vorgestellt werden.

Remote Procedure Calls

Remote Procedure Calls stellen historisch eine sehr frühe Form der Verteilung von Anwendungen dar. Coulouris u. a. [2005] stellen einen Vergleich mit dem Konzept der zuvor beschriebenen *Remote Method Invocation* her. Ein Server, der eine RPC-Schnittstelle anbietet,

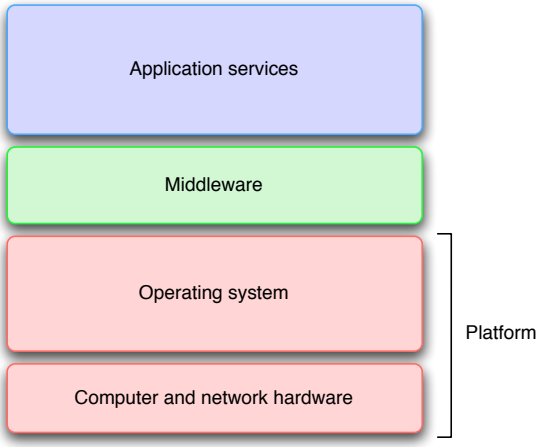

Abbildung 4.1: Middleware zur Maskierung der Heterogenität des Betriebssystems [In Anlehnung an Coulouris u. a., 2005]

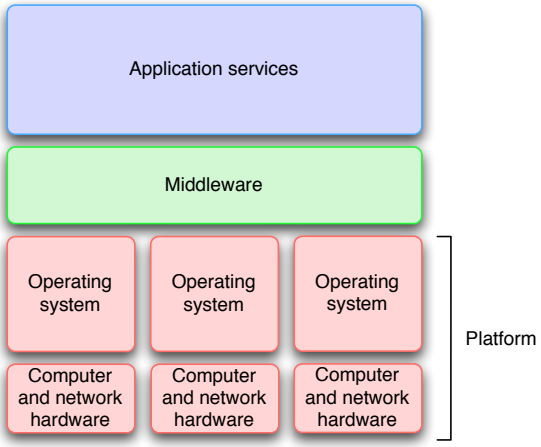

Abbildung 4.2: Middleware zur Maskierung der Verteilung eines Systems [In Anlehnung an Tanenbaum und Steen, 2006]

sei in diesem Zusammenhang ähnlich zu einem einzelnen *Remote Object*. Der Server kann ebenfalls verschiedene Zustände speichern. Der zentrale Unterschied bestehe in der fehlenden Möglichkeit, neue Instanzen von Objekten zu erzeugen oder Daten innerhalb dieser zu kapseln. Emmerich u. a. [2008] heben außerdem noch hervor, dass RPC-Middleware für gewöhnlich bereits von allen etablierten Betriebssystemen angeboten wird. RPC sei in sehr grundlegenden Betriebssystemfunktionen verankert. So würden z. B. Dienste wie DNS oder DHCP über RPCs realisiert.

Distributed Objects

Mit dem Einzug der Objektorientierung haben sich objektorientierte Middleware-Systeme vermehrt etabliert. Sie alle verfolgen das Ziel das objektorientierte Programmiermodell transparent auf verteilten Systemen anzuwenden. Im Folgenden sollen *Java RMI* und *CORBA* als zwei Beispiele für *Distributed Objects*-Middleware gegeben werden.

CORBA. Die *Common Object Request Broker Architecture* (CORBA) ist auf den Umgang mit der Heterogenität zwischen verschiedenen Betriebssystemen, Netzwerken oder auch Datenrepräsentationen ausgelegt. Die von verteilten Prozessen verwendeten Remote-Schnittstellen werden daher in einer plattformunabhängigen IDL (Interface Definition Language) beschrieben und entsprechend in die Sprache und Datenrepräsentation des Zielsystems übersetzt [Emmerich u. a., 2008].

Java RMI. Die Java *Remote Method Invocation* (Java RMI; [Sun Microsystems, Inc., 2006]) ist das in der Programmiersprache *Java* angebotene Programmiermodell für entfernte Methodenaufrufe. *Java RMI* ermöglicht es Objekte über mehrere Prozesse und Maschinen zu verteilen und diese transparent wie lokale Java-Objekte verwenden zu können. Die Verwendung von *Java RMI* ist dabei auf Anwendungen beschränkt, die innerhalb einer Java Virtual Machine ausgeführt werden können. *Java RMI* besitzt somit zwar oberflächliche Ähnlichkeiten zu CORBA oder DCOM, behandelt die Heterogenität von Netzwerken und Betriebssystemen auf einer völlig anderen Ebene [Waldo, 1998].

Sowohl *Java RMI*, als auch *CORBA* ermöglichen den verteilten und transparenten Einsatz objektorientierter Programmiertechniken. Beide ermöglichen es dem Programmierer entfernte Methodenaufrufe genau wie lokale Methodenaufrufe zu behandeln. Der Unterschied in beiden Modellen besteht grundsätzlich in der Handhabung der Heterogenität. Bei CORBA ist der Umgang mit Heterogenität expliziter Bestandteil der Spezifikation. Java RMI behandelt das Thema eher implizit, in dem es die Abstraktion der Plattform mit Hilfe der virtuellen Maschine übernimmt.

Verteilte Transaktionen

Emmerich, Aoyama und Sventek teilen den Bereich der verteilten Transaktionen in zwei Bereiche ein, die zum einen von der *Java Enterprise Edition* (Java EE) und zum anderen von *CORBA Object Transactions* abgedeckt werden (Emmerich u. a. [2008]). Es handelt sich bei diesen Ausprägungen von Middleware um Technologien, die heterogene Transaktionskonzepte für Entwickler transparent gestalten sollen.

CORBA Object Transactions. Das Ziel bei der Entwicklung der *CORBA OTS* [OMG, 1998, Kapitel 10] war die Protokolle zur verteilten Transaktionskontrolle in objektorientierten Umgebungen zu kapseln und dabei die in bereits etablierten Standards zur Interoperabilität unterschiedlicher Transaktionsmonitore und Datenbanken zu erhalten (Emmerich u. a. [2008]). Emmerich, Aoyama und Sventek stellen weiterhin heraus, dass *CORBA OTS* in sehr engem Zusammenhang mit dem *CORBA Concurrency Control Service* (CCS) [OMG, 1998, Kapitel 7] stehen, das von OTS verwendet wird, um Transaktionssicherheit zu gewährleisten.

Java EE. Die *Java Enterprise Edition* (Java EE) bildet eine Softwareplattform, die auf die Entwicklung von hochverfügbaren, sicheren, verlässlichen und skalierbaren Systemen ausgerichtet ist (Chinnici und Shannon [2009]). Die *Java EE*-Spezifikation beschreibt eine Reihe von Teilkomponenten, zu denen u. a. Java Server Pages (JSP), Servlets, das Java Messaging System (JMS), das Java Transaction API (JTA), das Java Persistence API (JPA) und Enterprise Java Beans (EJB) zählen. Die EJB-Spezifikation (s.

Saks u. a. [2009]) bildet den Kern von *Java EE*. EJB definiert ein serverseitiges Komponentenmodell, das den transparenten Einsatz Kernaspekte von Java EE erlaubt.

Verteilte Transaktionstransparenz wird vor allem durch das Zusammenspiel von JTA (Cheung und Matena [2002]), JTS (Java Transaction Service; [Cheung, 1999]) und JPA (DeMichiel u. a. [2009]) erreicht. Die EJB-Spezifikation bietet zwei Modelle des Transaktionsmanagements an: containergesteuert und applikationsgesteuert. Im ersten Modell handhabt der EJB-Container Beginn und Ende von Transaktionen. Im zweiten Modell nimmt der Anwendungsentwickler mit Hilfe der in der JTA angebotenen Schnittstellen Einfluss auf die Transaktionslaufzeit. Bereits in der Einleitung von [Cheung, 1999] wird auf die Interoperabilität und Portabilität von JTS hingewiesen, die durch die Implementierung von Schnittstellen der *CORBA Object Transactions* (OTS) erreicht werden.

Middleware für Service-Oriented-Computing

Service-orientierte Architekturen (SOA) stellen ein abstraktes Architekturmuster lose gekoppelter Systeme dar. Sie sind entstanden aus dem immer stärker gewichteten Anteil der IT in Unternehmen, sowie der mangelnden Flexibilität und Interoperabilität der vorhandenen IT. Systeme, die auf objektorientierter Middleware beruhen, sind für die Kommunikation zwischen Unternehmen häufig zu schwergewichtig und abstrahieren meist nicht ausreichend von den verwendeten Programmiersprachen, Sicherheits- oder Transaktionsmodellen (Weerawarana u. a. [2005]).

Webservices stellen laut Emmerich u. a. [2008] die am weitetsten verbreitete Ausprägung Service-orientierter Architekturen dar. Sie verwenden daher insbesondere Webservices als konkretes Beispiel und stellen die verwendeten Protokolle und Beschreibungssprachen in den Mittelpunkt ihrer Betrachtungen.

XML. Die *eXtensible Markup Language* [Bray u. a., 2006] ist eine allgemeine und erweiterbare Beschreibungssprache für strukturierte Inhalte. Sie liefert die Basis für die in Webservices verwendeten Sprachen. *WSDL*, *SOAP* und *BPEL* sind jeweils als XML-Dialekt spezifiziert.

WSDL. Die *Web Services Description Language* (WSDL; Booth und Liu [2007], sowie Chinnici u. a. [2007b,a]) ist eine Beschreibungssprache für Schnittstellen, die von einem Webservice für die externen Dienstnutzer (auch Consumer) angeboten werden. Ein WSDL-Dokument besteht typischerweise aus einem abstrakten und einem konkreten Teil. Der abstrakte Teil beschreibt das Angebot und die Struktur der auszutauschenden Daten. Der konkrete Teil gibt Auskunft über die Beschreibung der konkreten Schnittstelle und der verwendeten Protokolle [Weerawarana u. a., 2005, Kapitel 6].

SOAP. Das SAOP-Protokoll[4] (Mitra und Lafon [2007], sowie Gudgin u. a. [2007a,b]) spezifiziert ein Nachrichtenformat für entfernte Aufrufe von Objekten und der zugehörigen Antworten, die in XML-Nachrichten kodiert werden. SOAP-Nachrichten können sowohl über synchrone (z. B. HTTP) als auch asynchrone (z. B. SMTP) Netzwerkprotokolle übertragen werden [Emmerich u. a., 2008].

BPEL. Die *Business Process Execution Language* (BPEL; Alves u. a. [2007]) ist eine erweiterbare Beschreibungssprache, die es erlaubt, eine bedingte arbeitsablauforientierte Ausführungsreihenfolge von Diensten zu beschreiben [Weerawarana u. a., 2005, Kapitel 14].

Bei der Ausführung von Diensten werden zwei Arten der Komposition von Webservices unterschieden: Orchestrierung und Choreographie. Der grundsätzliche Unterschied zwischen diesen Kompositionsarten besteht in der Sichtweise auf die Dienstanbieter. Bei der Choreographie wird die Zusammenarbeit aller Teilnehmer in Betracht gezogen und eine globale Sicht auf das System gegeben. Bei der Orchestrierung von Webservices hingegen liegt das Hauptaugenmerk auf der Integration von Geschäftsprozessen, weshalb nur die Sichtweise einer Partei in Betracht gezogen wird. BPEL ist ein Werkzeug zur Orchestrierung von Webservices [Emmerich u. a., 2008].

Die vorgestellten Technologien von Webservices führen zu einer lose gekoppelten flexiblen Architektur, die unabhängig von Programmiersprachen, Datenrepräsentationen, Ausführungsort und Kommunikationsprotokollen betrieben werden können. Die beschriebenen Standards werden von vielen Anbietern Service-orientierter Middleware-Produkte implementiert und angeboten [Emmerich u. a., 2008].

Message-Oriented-Middleware

Message Oriented Middleware (MOM) wird in vielen Application Servern zur nachrichtenbasierten asynchronen Kommunikation verwendet [Emmerich u. a., 2008]. So enthalten die Spezifikationnen *Java Enterprise Edition* ein Dokument, das den *Java Messaging Service* (JMS; Hapner u. a. [2002]) beschreibt. Dieser wird folgendermaßen Charakterisiert:

> *„JMS provides a common way for Java programs to create, send, receive and read an enterprise messaging system's messages."* – [Hapner u. a., 2002, S. 13]

Der Einsatz von JMS ist insbesondere in komplexen Softwareprojekten sinnvoll, so beschreiben beispielsweise Krasemann und Spindel [2004] die Realisierung einer Softwarearchitektur für die hochautomatisierten Abläufe des Containerterminals Altenwerder in Hamburg. Die

[4] Das Akronym SOAP für *Simple Object Access Protocol* gilt seit der Version 1.2 des Protokolls nicht mehr. SOAP ist somit zu einem eigenständigen Kunstwort geworden [Weerawarana u. a., 2005, Kapitel 4.1].

beschriebene Architektur setzt an zentraler Stelle auf den Einsatz einet MOM auf Java-EE-Basis. Krasemann und Spindel heben insbesondere die Entkopplung der Lastverteilung in der Kommunikation (Lastspitzen werden über Message-Queues aufgefangen), sowie die Gewährung von Zustellgarantien (der Sender weiß, dass der Empfänger die Nachricht erhält) als zentrale Vorteile einer auf JMS basierenden Middleware hervor.

Die im Rahmen des in [Krasemann und Spindel, 2004] beschriebenen Systems verwendete Implementation des JMS ist SwiftMQ[5]. Weitere Implementationen sind z. B. IBM WebsphereMQ[6] oder Apache ActiveMQ[7].

4.2.2 Middleware als technologische Voraussetzung

Der Einsatz einer Middleware in einem verteilten System ist in vielerlei Hinsicht sinnvoll. Sie reduziert die Dauer von Entwicklungszyklen, den Gesamtentwicklungsaufwand, sowie die Komplexität der Entwicklung von hochwertigen, flexiblen, interoperablen und eingebetteten Systemen. Der Aufbau eines verteilten Systems aus wiederverwendbaren Komponenten trägt erheblich zur Erreichung dieses Ziels bei [Schantz und Schmidt, 2001]. Um die geforderten Kriterien erreichen zu können, kann es notwendig sein, eine Middleware selbst in mehrere austauschbare Komponenten aufzuteilen [Gokhale u. a., 2002].

Paspallis [2009] stellt die wesentlichen Möglichkeiten heraus, die sich durch den Einsatz einer Middleware ergeben:

Component-based Design. Im Systemdesign entwickelt sich aus der Sichtweise der Objektorientierung eine komponentenorientierte Sichtweise. Sie hat den Vorteil, dass Komponenten mit wohldefinierten Schnittstellen als ihre Funktionsweise als sogenannte „Black Boxes“ verborgen werden können. Diese Komponenten können unabhängig von anderen Komponenten realisiert werden.

Computational reflection. Der Begriff *Computational reflection* bezieht sich auf die Fähigkeit eines Programms, sein Verhalten bei Bedarf anzupassen. Diese Anpassung kann struktureller, aber auch verhaltensbasierter Natur sein. Strukturell bedeutet in diesem Zusammenhang, dass Zuordnungen innerhalb von Klassenhierarchien oder auch Objektverknüpfungen neu geordnet werden können. Verhaltensbasierte Änderungen beziehen sich auf Änderungen des Programmablaufs.

[5] http://www.swiftmq.com, Zugriffsdatum: 16.05.2010
[6] http://www.ibm.com/webspheremq, Zugriffsdatum: 16.05.2010
[7] http://activemq.apache.org/, Zugriffsdatum: 16.05.2010
Im Rahmen der Architekturdiskussion des Living Place Hamburg (s. a. Abschnitt 3.2) ist ActiveMQ zur Zeit eine angestrebte Kerntechnologie für den Ersatz des derzeit verwendeten EventHeap (weitere Erläuterungen folgen in Abschnitt 5.6.2)

Separation of concerns. Das Konzept der *Separation of concerns* beschreibt eine Methode, nach der die Funktionalitäten einer Software in eigenständige Komponenten unterteilt werden, die sich nach Möglichkeit nur minimal überlappen. Paspallis und Papadopoulos [2006] stellen heraus, dass die Trennung der Anwendungsentwicklung in funktionale und nicht-funktionale Anforderungen den Entwickungsprozess deutlich beschleunigen kann.

4.2.3 Einschränkungen für den Einsatz von Middleware

Coulouris, Dollimore und Kindberg beschreiben in [Coulouris u. a., 2005, Kapitel 2.2.1] die Einschränkungen beim Einsatz einer Middleware. Sie heben dabei ein Argument hervor, das in Saltzer u. a. [1984] als das *„end-to-end-argument“* bezeichnet wird. Saltzer, Reed und Clark beschreiben hier, dass besondere Umstände wie z. B. des Netzwerks oder der Verbindungsqualität es erfordern können, eine besondere Berücksichtigung in der Softwarearchitekur zu integrieren. Sie weisen dabei darauf hin, dass insbesondere die verbindenden Komponenten (in diesem Fall die Middleware) nicht zu sehr auf spezielle Anwendungsfälle spezialisiert sein sollen. In Reed [2000] wird dieser Gedanke erneut aufgegriffen. Reed mahnt an, dass eine zunehmende Spezialisierung der Verbindungskomponente zwischen zwei Endpunkten zukünftige Entwicklungen erheblich behindern kann.

4.2.4 Zusammenfassung

In diesem Kapitel wurde ein Überblick zum Thema Middleware gegeben. Es wurden gezeigt, wie eine Middleware innerhalb einer Softwarearchitektur eingegliedert ist und nach welchen Gesichtspunkten verschiedene Arten von Middleware unterschieden werden können.

Im Kern beschreibt der Begriff *Middleware* eine Zwischenschicht der Softwarearchitektur, die es erlaubt von verschiedenen Aspekten der Plattform zu abstrahieren. Konkret wurden hier die Beispiele der Verteilung der Ausführung von Anwendungen und Transaktionen, der Steigerung von Flexibilität der Abbildung von Geschäftsprozessen oder die Einführung asynchroner Kommunikation in komplexen verteilten Systemen gegeben. Abschließend wurden die Einschränkungen angedeutet, die die Möglichkeiten der Übernahme von Aufgaben einer Middleware einschränken.

4.3 Complex Event Processing

Complex Event Processing ist ein Sammelbegriff unter dem die Verarbeitung von Ereignissen und die dazu benötigten Methoden, Technologien und Werkzeuge zusammengefasst werden. Die Verarbeitung von Ereignissen geschieht dabei kontinuierlich und zeitnah, während diese auftreten. Es wird dabei das Ziel verfolgt, wertvolles und höheres Wissen aus den erfassten Aktivitäten abzuleiten, das sich nur in Form von komplexen Ereignissen, d. h. aus der Kombination mehrerer Ereignisse erkennen lässt (Eckert und Bry [2009]). Der Begriff *Complex Event Processing* wurde hauptsächlich in der ersten Auflage von Luckham [2005][8] geprägt. Die Wurzeln dieses Themengebiets in der Forschung liegen dabei laut Eckert und Bry [2009] vor allem in den Bereichen der ereignisorientierten Simulation, aktiven Datenbanken oder auch dem temporalen Schließen in der KI.

Die in einem Smart Home auftretenden Ereignisse sollen schnell und möglichst direkt verarbeitet werden. Ein häufiger Anwendungsfall ist die Ableitung eines Kontextes aus einer Vielzahl von Ereignissen.

In diesem Abschnitt wird die Bedeutung des Complex Event Processing im Kontext eines Smart Homes erläutert. Zu diesem Zweck wird zunächst der Event-Begriff erläutert. Aufbauend folgt die Theorie der Erzeugung von Events, sowie der Erkennung von komplexen Ereignissen. Abschließend folgt eine Erläuterung der gängigsten Konzepte, die heutzutage im Complex Event Processing eingesetzt werden.

4.3.1 Anwendungsgebiete

Die Ursprünge des *Complex Event Processing* sind vielseitig, ähnlich ist es mit den Anwendungsgebieten. Während die in Chakravarthy und Mishra [1994] beschriebenen praktischen Anwendungen noch sehr (aber nicht ausschließlich) auf das Gebiet der Datenbanken begrenzt ist, bieten Luckham [2005] und Eckert und Bry [2009] schon eine deutlich breitere Auswahl an Anwendungsgebieten.

Luckham nennt dabei sehr allgemein orientierte verteilte Systeme, wie z. B. ein Börsensystem oder ein militärisch orientiertes *Command-and-Control-System*. Er nutzt diese Beispiele dabei um auf die allgemeinen Vorteile des Complex Event Processing hinzuweisen, die er in einer deutlichen Vereinfachung der Kommunikationsstrukturen und Überwachung systemkritischer Zustände sieht.

Eckert und Bry geben zusätzlich eine Reihe sehr unterschiedlicher Anwendungsgebiete, die die Flexibilität der Technologie verdeutlichen:

[8] Die erste Ausgabe des Buches erschien im April 2002.

Business Activity Monitoring. Erkennung von Problemen und Chancen durch Überwachung von Geschäftsprozessen und unternehmenskritischer Parameter.

Sensor-Netzwerke Übermittlung von Messdaten an die Außenwelt. Es können z. B. gleichartige Daten aggregiert werden, um Messfehler zu verringern oder um Interpretationen aus verschiedenen Datenquellen zu schließen (z. B. *Brand* aus den Daten der Rauchmelder und Temperatursensoren).

Markdaten Unter dem Stichwort *Algorithmic Trading* kann bei der kontinuierlichen Analyse komplexer Marktdaten automatisch reagiert werden.

Die beschriebenen Beispiele zeigen, dass die Anwendungsgebiete des Complex Event Processing v. a. in verteilten komplexen Umgebungen liegen. Sie beziehen sich dabei auf die Erkennung von Ereignissen, die mit vorhandenen Beschreibungsmöglichkeiten nur schwierig zu erfassen sind.

4.3.2 Definitionen und Begriffsabgrenzung

In Luckham [2005] wird erwähnt, dass die Grundlagen im Umgang mit Events für gewöhnlich sehr vertraut erscheinen, da sie im Allgemeinen dem natürlichen Umgang mit Computern und dem Internet entsprächen. Es wird dabei darauf hingewiesen, dass dieses nur oberflächlich erscheine und die Techniken des Complex Event Processing dieses *Gefühl* deutlich präzisieren würden.

Chakravarthy und Adaikkalavan [2008][9] definieren einen *Event* als ein unmittelbares und atomares Ereignis von Interesse:

> *„An event is defined to be an instantaneous, atomic (happens completely or not at all) occurrence of interest.“*

Unabhängig hiervon definiert Luckham einen ähnlichen *Event*-Begriff ([Luckham, 2005, S. 88]):

> *„An Event is an object that is a record of an activity in a system. The event signifies the activity. An event may be related to other events.“*

[9] in Anlehnung an Chakravarthy und Mishra [1994]

Unterscheidung zwischen Aktivitäten und Events

Die vorgestellten Definitionen erscheinen zunächst ähnlich. Die Definition von Chakravarthy und Adaikkalavan ist von allgemeiner Natur und beschreibt ein Event lediglich als ein „unmittelbares und atomares Ereignis von Interesse“. Luckham nimmt dahingegen eine Differenzierung zwischen den Begriffen *Aktivität* („activity“) und Event vor. Eine Aktivität ist nach seiner Definition eine natürliche Handlung oder ein physisches Ereignis. Ein Event wird in diesem Zusammenhang als ein Objekt[10] bezeichnet, das geeignet ist eine Aktivität in ihren charakterisierenden Eigenschaften innerhalb eines Gesamtsystems zu formalisieren. Events besitzen somit eine Aussagekraft über die von ihnen beschriebene Aktivität. Sie können zudem explizit in Bezug zu anderen Events gesetzt sein [Luckham, 2005].

Luckham nennt drei Aspekte zur Beschreibung von Events:

Form (*„Form“*). Die Form eines Events ist die eines Objekts. Events können Attribute oder andere Daten tragen. Die Form eines Events kann beispielsweise einem Text-String entsprechen oder auch durch ein Tupel verschiedenster Daten repräsentiert werden[11].

Aussagekraft (*„Significance“*). Ein Event kennzeichnet die Aussage einer Aktivität. Events beinhalten normalerweise Informationen die die Aktivität, die den Event auslöst, beschreiben.

Bezug (*„Relativity“*). Aktivitäten stehen zueinander im Bezug durch *zeitliche Abfolge („time“)* , *Kausalität („causality“)* und *Häufung („aggregation“)*. Events stehen auf die selbe Art zueinander in Bezug, wie es auch die Aktivitäten tun, die von ihnen gekennzeichnet werden.

Es handle sich daher beim Event-Processing nicht um das Verarbeiten von Aktivitäten, sondern um die Verarbeitung von Objekten, die diese Aktivitäten beschreiben. Weiterhin greift Luckham die häufige Verwechslung zwischen *Event* und *Nachricht („Message“)* auf. Ein Event sei keine Nachricht[12], wenngleich ein Event mit Hilfe von Nachrichten übermittelt werden kann. Bei der Reduzierung des Event-Begriffs auf die reine Nachrichtenübermittlung würden die zusätzlichen charakterisierenden Aspekte (s. o.) von Events verloren gehen.

Abgrenzung verschiedener Event-Typen

Sowohl Luckham, als auch Chakravarthy und Adaikkalavan unterscheiden zwischen primitiven (direkt aus einer Aktivität abgeleiteten) und komplexen (aus mehreren Events abgeleite-

[10] im Sinne des Paradigmas der objektorientierten Programmierung
[11] vgl. hierzu auch Johanson und Fox [2004]
[12] im Sinne von asynchroner Kommunikation innerhalb eines Systems

ten) Events[13]. Eine einfache Unterscheidung zwischen komplexen und primitiven Events ist, dass das Auftreten primitiver Events auf einen Zeitpunkt eingegrenzt werden kann, während komplexe Events meist über ein Zeitintervall verlaufen (Luckham [2005]). Eine Möglichkeit der Klassifizierung verschiedener Event-Typen zeigt Abbildung 4.3. Die sehr an der Entwicklung auf dem Gebiet der Datenbanken orientierte Ausrichtung von Chakravarthy und Adaikkalavan [2008] spiegelt sich zwar in der Abbildung wider, sie gibt dennoch eine Idee der komplexen Klassifizierungsmöglichkeiten von Events. Das präzise Ableiten komplexer Events aus den Zusammenhängen von anderen (primitiven und komplexen) Ereignissen ist die Kernaufgabe des Complex Event Processing[14].

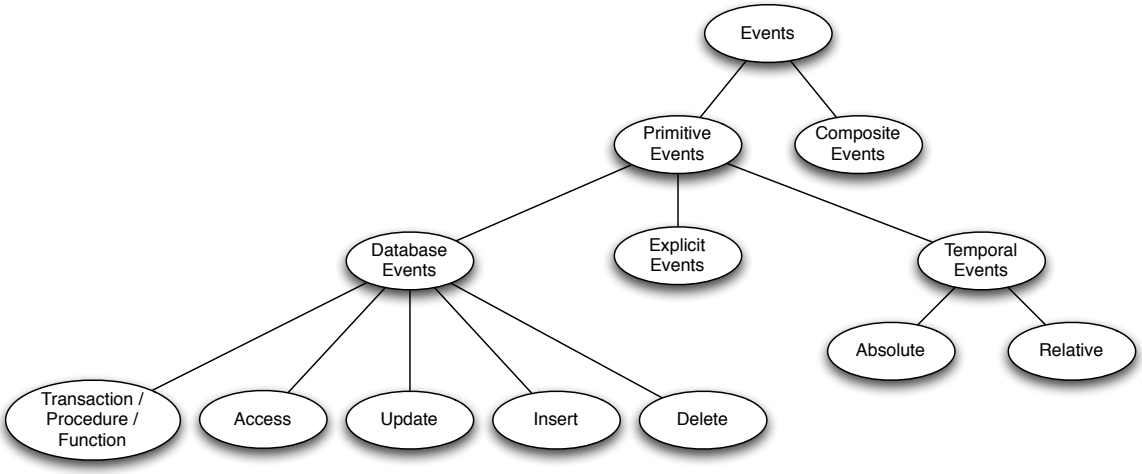

Abbildung 4.3: Klassifizierung von Events
[In Anlehnung an Abbildung 2 in Chakravarthy und Mishra, 1994]

Abgrenzung zu Data Mining und Knowledge Discovery

Die Ereignisverarbeitung wird beim Complex Event Processing gegen einen theoretisch unendlichen Strom von Daten ausgeführt. Das Ziel ist es mit geeigneten Mitteln Muster und Zusammenhänge innerhalb des Datenstroms zu erkennen und darauf basierend Aktionen ableiten zu können.

[13] Luckham verwendet hierbei durchgängig den Begriff *„complex Events“*. Chakravarthy und Adaikkalavan führen zusätzlich die Begriffe *„abstract Events“* und *„composite Events“* ein. Die drei Formulierungen können im Kontext dieser Arbeit als Synonym betrachtet werden.

[14] Es sei an dieser Stelle abschließend auf die abweichende Verwendung des Event-Begriffs von Johanson und Fox [2002, 2004] (s. Abschnitt 3.3.5) hingewiesen. Die dort beschriebenen Events bezeichnen zwar ebenfalls Aktivitäten der realen Welt, jedoch wird der Begriff im dortigen Zusammenhang definiert, um eine Abgrenzung zum Tuple-Spaces-Modell zu erreichen. Er bezeichnet daher eine konkrete Datenstruktur innerhalb der iROS-Architektur.

Von Fayyad u. a. [1996] werden die folgenden Definitionen der Begriffe Data Mining und Knowledge Discovery in Databases (KDD) gegeben:

> *„KDD is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.“*
>
> *„Data mining is a step in the KDD process that consists of applying data analysis and discovery algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns (or models) over the data.“*

Aus den Definitionen wird ersichtlich, dass bei Data Mining und KDD die Verarbeitung und Transformation einer persistenten und endlichen Datenmenge im Vordergrund steht, während Complex Event Processing Ereignisanfragen an einen flüchtigen Datenstrom gestellt werden.

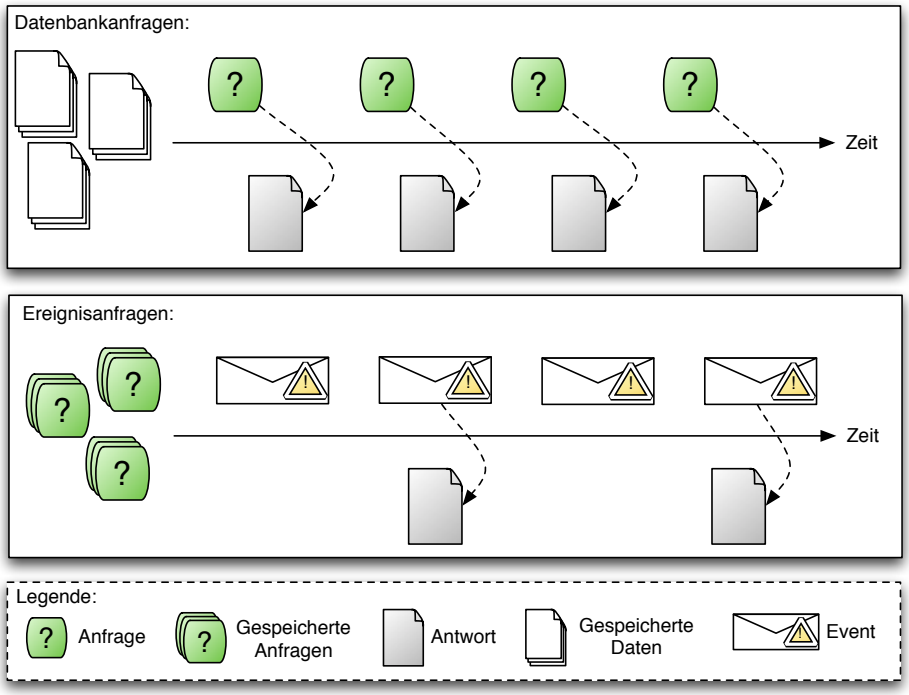

Abbildung 4.4: Unterschiede zwischen Datenbank- und Ereignisstromanfragen [In Anlehnung an Eckert und Bry, 2009]

Auf der Kommunikationsebene mit dem Datenbank-, bzw. dem Event-System wird deutlich inwiefern sich die beiden Konzepte bereits in der Art von Anfrage- und Antwortverhalten unterscheiden. In Abbildung 4.4 sind die bedeutendsten Unterschiede zwischen Ereignis- und

Datenbankabfragen dargestellt. Ein Datenbanksystem kann auf Anfragen unmittelbar (synchron) aus der Menge der ihm bekannten Daten antworten. Die Anfragen sind spontan und flüchtig, bei einer Änderung der Daten (und gegebenfalls auch der Antwortmenge) obliegt es in der Regel dem Benutzer diese Änderung mittels einer erneuten Anfrage festzustellen. Im Gegensatz hierzu werden beim Complex Event Processing die Ereignisanfragen hinterlegt und permanent gegen den Strom von Events ausgewertet. Sobald ein auf eine Anfrage passendes Muster erkannt wird, liefert das System eine Antwort (asynchron). Ob der Benutzer auf eine Anfrage eine (oder mehrere) Antworten auf seine Anfrage erhält, kann zum Zeitpunkt der Anfragestellung kaum zugesichert werden [Eckert und Bry, 2009]. Vereinfacht lässt sich sagen, dass in einem Datenbanksystem die Daten gespeichert werden und über einen kontinuierlichen Strom von Daten abgefragt werden, während beim Complex Event Processing die Daten kontinuierlich (in Form von Ereignissen) geliefert werden und gegen hinterlegte Anfragen geprüft werden.

4.3.3 Ereignisanfragen

Ereignisanfragen an ein System werden grundsätzlich durch die folgenden Aspekte abgedeckt (Eckert und Bry [2009], vgl. hierzu auch [Luckham, 2005, S. 94f.]):

Extraktion von Daten. Ereignisse beinhalten Daten (vgl. Aspekt der *Form* in Abschnitt 4.3.2). Diese Daten bilden die Grundlage für den Informationsgehalt eines Events. Die Daten eines Events müssen daher für die Abfrage von Events zur Verfügung stehen.

Komposition. Der Begriff Komposition beschreibt Beziehungen zwischen Events. Bei einer Verteilung von Events über die Zeit muss es möglich sein, die Zusammenhänge von Events datenbezogen auswerten zu können. Luckham nennt hierzu ein Beispiel: Eine Aktivität wird mittels eines Events A beschrieben, der wiederum aus den Events $B_1, B_2, B_3, \ldots$ besteht. A ist somit eine Komposition aller Events B_i. Im Umkehrschluss ist jeder Event B_i ein Element von A. A beschreibt eine komplexe Aktivität und ist somit ein höherwertiger *komplexer Event*, der aus Events der aggregierten Teilaktivitäten besteht.

Kausalität. Wenn eine von Event A beschriebene Aktivität auftreten muss, damit eine andere durch Event B beschriebene Aktivität ebenfalls auftritt, so wurde B verursacht durch A. Diese Kausalität ist definiert als eine Abhängigkeitsbeziehung zwischen Aktivitäten innerhalb des Systems. Ein Event hängt von einem anderen Event ab, wenn dieser nur eintritt, weil der andere Event zuvor eingetreten ist (hinreichende Bedingung). Zwei Events sind unabhängig voneinander, wenn keiner von beiden der Verursacher des anderen ist (Luckham [2005]).

Zeitliche Zusammenhänge. Die Formulierung von Ereignisanfragen kann eine bestimmte Abfolge oder zeitliche Häufung von Events beinhalten. Es ist auch möglich Anfragen nur über ein bestimmtes Zeitfenster zu stellen. Luckham bezeichnet die Zeit als die Beziehung, die Events in eine Abfolge ordnet. Beim Auftreten einer Aktivität würde ein Event erzeugt und mit einem Zeitstempel versehen. Da es in einem System mehrere Uhren geben kann, kann es vorkommen, dass ein Event mehrere Zeitstempel trägt – einen für jede Uhr.

Akkumulation. Es ist vorstellbar, dass Anfragen bezüglich des Fehlens eines Ereignisses gestellt werden. Denkbar wäre das Prüfen einer Rückmeldung, ob eine Aktion erfolgreich ausgeführt wurde. Eine andere Ausprägung wäre die Anfrage auf eine besondere Häufung von Events. Beide Anfragen lassen sich in unendlichen Datenströmen nur beantworten, wenn sie auf ein definiertes Zeitfenster gestellt werden.

Eckert und Bry betonen zusätzlich noch zwei Arten von Regeln nach denen Events erzeugt werden: deduktive und reaktive Regeln. *Deduktive Regeln* erzeugen Ereignisse aus Ereignisanfragen. Die Autoren vergleichen diese dabei mit dem Konzept der *Views* in Datenbanken. *Reaktive Regeln* hingegen spezifizieren wiederum, welche Aktionen beim Erkennen komplexer Ereignisse ausgeführt werden sollen. Hierbei kann es sich beispielsweise um Datenbankänderungen oder auch Methodenaufrufe handeln (Eckert und Bry [2009]).

Chakravarthy und Mishra [1994] geben einen Überblick der Operationen zur Abfrage einfacher Events (vgl. Abbildung 4.5). Die Knoten des dargestellten Baumes geben ein Beispiel für Operationen mit denen die Abfrage von Events gesteuert werden kann.

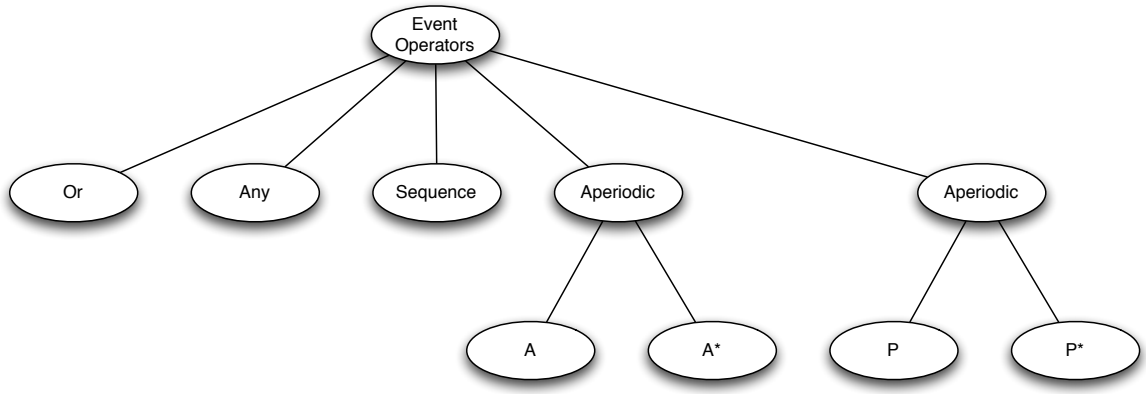

Abbildung 4.5: Event Operators
[In Anlehnung an Abbildung 3 in Chakravarthy und Mishra, 1994]

4.3.4 Erzeugung von Events

Zur Erzeugung von Events beschreibt Luckham [2005] zwei Schritte, in denen eine Aktivität beobachtet wird und schließlich in Form eines Events beschrieben wird.

1. Beobachtung. Aktivitäten müssen innerhalb des Systems so beobachtbar sein, dass sie das Verhalten des Systems nicht beeinflussen.

2. Adaption. Es müssen Adapter existieren, die die beobachteten Aktivitäten in Form von Events beschreiben können, so dass diese vom Complex Event Processing verarbeitet werden können.

Events können prinzipiell aus drei unterschiedlichen Quellen erzeugt werden (Luckham [2005]):

IT-Schicht. Die IT-Schicht ist für gewöhnlich auf allen Ebenen beobachtbar. Somit lässt sich hier eine Vielzahl von Events erzeugen und erfassen. Events, die in der *IT-Schicht* erzeugt werden, haben ihren Ursprung häufig in der Kommunikation zwischen den einzelnen Komponenten, wie z. B. einer Message Oriented Middleware oder einer Datenbank. Hier lassen sich beispielsweise entfernte Methodenaufrufe oder Datenbankabfragen in Events übersetzen.

Messinstrumente. Die Aktivitäten der Komponenten eines Zielsystems lassen sich ebenfalls in Events übersetzen. Hierbei kann es sich beispielsweise um Temperaturdaten, Netzwerkverkehr oder auch Statusmeldungen oder Alarme jeglicher Ursache handeln, die in Events übersetzt werden.

Complex Event Processing. Events, die bei der Verarbeitung komplexer Events erzeugt werden, werden ebenfalls als neue Events in das System eingespeist.

4.3.5 Ereignisabfragesprachen

Im vorhergehenden Unterabschnitt wurden die grundsätzlichen Eigenschaften von Events und damit auch die Möglichkeiten der Differenzierung und Abfrage von Events vorgestellt. In Folgenden sollen Konzepte vorgestellt werden, auf deren Basis die Formulierung von Ereignisabfragen möglich ist. In Eckert und Bry [2009] werden drei Grundideen vorgestellt, die geeignet sind, Ereignisabfragen im Sinne des Complex Event Processing zu stellen.

Kompositionsoperatoren. Anfragen nach komplexen Ereignissen bestehen hier aus miteinander verknüpften Anfragen nach Einzelereignissen. Es stehen für gewöhnlich Operatoren für Konjunktion (Ereignisse zu unterschiedlichen Zeitpunkten), Sequenz

(Abfolgen von Ereignissen) oder Negation (ein Ereignis tritt nicht im Zeitraum zwischen zwei anderen Ereignissen auf). Das Konzept der Kompositionsoperatoren entstammt ursprünglich aus der Entwicklung aktiver Datenbanksysteme (Paton und Díaz [1999][15]), wird allerdings auch unabhängig von Datenbanken eingesetzt (s. Adi und Etzion [2004]). Der Einsatz von Kompositionsoperatoren zur Ereignisabfrage ist laut Eckert und Bry [2009] nicht sonderlich weit verbreitet. Ferner bieten sie hauptsächlich die Möglichkeit der Analyse *von außen*, wobei die Daten eines Events vernachlässigt werden.

Datenstrom-Anfragesprachen. Datenstrom-Anfragesprachen haben sich auf der Grundlage von Datenbankabfragesprachen entwickelt. Sie arbeiten nach dem Grundprinzip, einen Event-Strom innerhalb eines endlichen Zeitfensters (z. B. *„alle Events der letzten Stunde“* oder *„die letzten 100 Events“*) in Relationen zu transformieren, auf denen SQL-ähnliche Abfragen möglich sind. Ein Beispiel für eine Datenstrom-Anfragesprachen ist die *Continuus Query Language* (CQL; s. Arasu u. a. [2006]). Der Einsatz von etablierten Datenstrom-Anfragesprachen bietet einen sehr datenzentrierten Ansatz, Anfragen nach Negation oder temporalen Zusammenhängen sind oft nur umständlich zu realisieren.

Produktionsregeln. Produktionsregeln werden heute häufig in Business Rule Management Systemen eingesetzt. Beispiele hierfür stellen u. a. Drools[16] oder auch ILOG JRules[17] dar. Diese Frameworks stellen im Grunde keine Ereignisanfragesprachen dar, können allerdings für diese Zwecke verwendet werden. Im Kern verwalten diese Systeme Zustände und Regeln, die ausgeführt werden, wenn ein entsprechender Zustand erkannt wird.

Die Auswertung von Produktionsregeln geschieht inkrementell (z. B. nach dem Rete-Algorithmus; s. Forgy [1982]). Diese Form der Auswertung eignet sich ebenfalls für die Auswertungen des Complex Event Processing. Durch das Erzeugen von Events werden auch entsprechende Fakten erzeugt. Ereignisanfragen werden dann als Bedingungen über diese Fakten formuliert. Eckert und Bry stellen die Produktionsregeln als eine sehr flexible Grundlage für Complex Event Processing dar, sie betonen dabei auch die gute und breite Anbindung an bestehende Programmiersprachen. Sie bemängeln in diesem Zusammenhang jedoch die etwas unnatürliche Abstraktionsebene, da hier zustands- und nicht ereignisorientiert gearbeitet wird. Da zusätzlich auf einer sehr niedrigen Ebene gearbeitet wird, kann es umständlich sein Aggregation und Negation von Events auszudrücken. Weiterhin benötigt ein solches System einen

[15] Eckert und Bry verweisen zu diesem Thema auf auf Paton u. a. [1998]
[16] http://www.jboss.org/drools, Abruf: 12.05.2010
[17] http://www.ibm.com/software/integration/business-rule-management/jrules/, Abruf: 12.05.2010

Mechanismus zur Garbage Collection, die das Entfernen nicht mehr benötigter Events aus dem Working Memory übernimmt.

4.3.6 Zusammenfassung

Complex Event Processing ist ein Konzept der kontinuierlichen Informationsverarbeitung. Es bietet Möglichkeiten, komplexe Zusammenhänge aus Ereignisströmen abzufragen und zu neuen Ereignissen weiterzuverarbeiten. Im Gegensatz zur Verarbeitung historischer Daten, wie es beim Einsatz von Datenbanken der Fall ist, werden hier Ereignisse (sog. *Events*) analysiert und verarbeitet, während diese passieren. Die Ursprünge eines Events können dabei natürlicher (z. B. Temperaturmessungen oder Türklingel) oder logischer Natur (z. B. hoher Wasserverbrauch bei geschlossenen Wasserhähnen deutet auf einen Rohrbruch hin) sein.

Die Abfrage von Events bezieht sich dabei zum einen auf die äußere Form, sowie temporale oder kausale Zusammenhänge zwischen Events, andererseits aber auch auf die mit den Events transportierten Nutzdaten. Die Laufzeit von Anfragen kann dabei unbestimmt sein oder mit dem Eintreten eines bestimmten Events oder Zeitpunkts enden.

Es gibt eine Reihe von Ansätzen, die für das Complex Event Processing eingesetzt werden. Diese basieren meist auf der historischen Entwicklung des Complex Event Processing beispielsweise aus den Bereichen der aktiven Datenbanken oder auch regelbasierten Systemen. Der Einsatz dieser Techniken erscheint unter dem historischen Gesichtspunkt zunächst als fragwürdig, erhält aber unter Einbezug der in [Luckham, 2005] vorgestellten allgemeinen Anwendungsgebiete neue Impulse. Während die Verarbeitung primitiver Ereignisse (z. B. Lichtsteuerung) sehr offensichtlich erscheint, ist auch die Verarbeitung komplexer Ereignisse denkbar. Beispiele für die Erzeugung solch komplexer Ereignisse werden in den analysierten Szenarien in Abschnitt 3.4 gegeben.

4.4 Fazit

Ausgehend von den Eigenschaften verteilter Systeme und den in Kapitel 3 analysierten Szenarien wurden in diesem Kapitel die Kerneigenschaften verteilter Systeme und deren Herausforderungen erläutert. Es wurde beschrieben, welche Kernherausforderungen die Modellierung einer Architektur für Smart Homes beeinflussen.

Die Modellierung nebenläufiger Systeme in Form von Threads birgt große Nachteile, da sie für den Programmierer ein nur schwer nachvollziehbares mentales Modell begünstigen. Andere Formen der Modellierung sind hier vorzuziehen [vgl. Lee, 2006]. Die Möglichkeit ein

nebenläufiges System mit Hilfe von Petri-Netzen oder Prozessalgebren (z. B. CSP) auf abstrakter Ebene formal zu Beschreiben und so das Design abzusichern. Die Implementation nach dem Aktormodell, z. B. mit Erlang oder Scala, stellt eine asynchrone und leichtgewichtige Möglichkeit Modellierung von Nebenläufigkeit dar.

Der Einsatz einer Middleware in einem verteilten System ermöglicht es viele Aspekte der Heterogenität des Systems transparent zu gestalten. Ziel soll es dabei sein, dem Entwickler die Anpassung auf die infrastrukturellen Eigenschaften des Systems möglichst vollständig abzunehmen. Es existiert eine Vielzahl von Middlewareprodukten, die für den Einsatz in bestimmten Systemtypen und die Erfüllung bestimmter Eigenschaften vorgesehen sind [vgl. Coulouris u. a., 2005; Tanenbaum und Steen, 2006; Emmerich u. a., 2008]. Middleware kann aber auch Systemfunktionalitäten und Designprozesse erst ermöglichen (z. B. Component-based Design, Computational Reflection oder Reflection of Concerns, [s. Paspallis, 2009]). Es ist allerdings auch Gründe, die gegen den universellen Einsatz einer Middlewareschicht sprechen. Dies ist beispielsweise der Fall, wenn die Middleware Aufgaben übernehmen soll, die ein präzises Wissen der Anwendung an den jeweiligen Endpunkten der Kommunikation vorraussetzen [s. Saltzer u. a., 1984; Reed, 2000].

Die Analyse der Anwenderszenarien hat zudem ergeben, dass es sich bei einem Smart Home um ein hochgradig ereignisbasiertes System handelt. Die identifikation komplexer Ereignissfolgen und deren Verarbeitung wird als Complex Event Processing bezeichnet (CEP) [Luckham, 2005; Eckert und Bry, 2009]. Im Gegensatz zum Data Mining [Fayyad u. a., 1996] wird beim CEP eine möglichst unmittelbare Erkennung von Ereignismustern angestrebt, die eine asynchrones Auftreten von Ergebnismengen auf Ereignisanfragen zur Folge hat [Eckert und Bry, 2009]. Der Inhalt diser Anfragen kann sich sowohl auf die Ausprägungen einzelner als auch auf Muster in Folgen von Events beziehen.

5 Systementwurf einer nachrichtenbasierten Architektur für Smart Homes

In Kapitel 3 wurden die Anforderungen an eine Basisarchitektur für Smart Homes analysiert. Kapitel 4 nahm darauf aufbauend die Einordnung von Smart Homes in das Gebiet der verteilten Systeme vor. Auf diese Weise wurden die speziellen und allgemeinen Anforderungen an eine Architektur für Smart Homes herausgestellt.

Dieser Abschnitt stellt einen auf den identifizierten Anforderungen basierenden Architekturentwurf vor. Zu diesem Zweck erfolgt zunächst die Formulierung der Systemvision und anschließend werden der Entwurf der Architektur und des Sicherheitskonzepts entwickelt. Es wird dabei auch auf die Spezifikation des Kommunikationsprotokolls zwischen den Systemkomponenten eingegangen. Abschließend werden die Möglichkeiten der formalen Verifikation des Systems, sowie einige die Realisierung betreffende Besonderheiten diskutiert.

5.1 Vorüberlegungen

Bevor die eigentliche Systemarchitektur vorgestellt wird, gibt dieses Kapitel einen Überblick der Vorüberlegungen, die bei der Entwicklung einer Systemarchitektur von Bedeutung sind und setzt sie in Relation zu den bisher erarbeiteten Ergebnissen der Analyse. Die vorgestellten Grundlagen orientieren sich an den in [Schiele u. a., 2010] vorgestellten Einflussfaktoren auf das Design pervasiver Systeme.

5.1.1 Systemorganisation

Die Organisationsform eines Systems lässt sich nach Schiele u. a. [2010] durch zwei unterschiedliche Sichtweisen beschreiben: *Smart Environments* und *Smart Peers*.

Smart Environments. Die Organisationsform der *Smart Environments* definiert sich für gewöhnlich über einen eingegrenzten Raum, wie z. B. einen Büroraum oder eine Wohnung, die mit Sensoren und Aktoren ausgestattet sind. Dieses führt häufig dazu, dass Geräte einer *Smart Environment* fest installiert oder nur bedingt mobil sind. Die Integration mobiler Geräte ist möglich. Die Art der Integration wird für gewöhnlich über den Standort des Geräts bestimmt [Schiele u. a., 2010, S. 202f.]. In Jensen u. a. [2009] werden verschiedene Möglichkeiten der Ortung von mobilen Geräten in Gebäuden beschreiben. Die Mechanismen zur Ortung können dabei spezielle Anforderungen an die Infrastruktur des Gebäudes oder Hardware des Clients stellen.

Schiele u. a. beschreiben weiterhin, dass *Smart Environments* häufig über zentralisierte Server- und Middlewarekomponenten verfügen. Die Begrünung hierfür wird in der ohnehin schon stationären Ausprägung dieser Organisationsform gesehen.

Smart Peers. Ausgehend von dem Gedanken sich dynamisch formender Netzwerke mobiler Geräte, hat sich die Organisationsform der *Smart Peers* herauskristallisiert. Smart Peers setzen den Menschen in den Mittelpunkt, mit dem Grundgedanken, dass ein pervasives System aus einer Sammlung von Computern besteht, die um eine Person herum angeordnet sind. Eines der Ergebnisse dieser Sichtweise ist, dass es keine zentrale Steuerung des Systems geben kann, da zu keinem Zeitpunkt sichergestellt sein kann, dass ein Gerät ununterbrochen verfügbar ist. Die beteiligten Geräte müssen in sich die Fähigkeit zur Selbstorganisation tragen [Schiele u. a., 2010, S. 203f.].

Die Organisation in Smart Peers erscheint zunächst als eine sehr flexible Systemform. Schiele u. a. weisen jedoch darauf hin, dass Smart Peers ihren Wirkungsrahmen häufig auf die unmittelbare Umgebung beschränken, um mit den gesteigerten Anforderungen an die Geräte eines solchen Systems umgehen zu können. Die erforderlichen Verwaltungsmechanismen können dabei häufig zu komplex für einen effizienten Einsatz sein.

5.1.2 Abstraktionsebenen

Die Entwicklung einer Middleware hat das Ziel Transparenz für die von ihr abgedeckten Problemstellungen zu erreichen. Sie automatisiert somit Abläufe, die ohne den Einsatz einer Middleware vom Entwickler implementiert werden müssten. Der Grad der erreichbaren Transparenz lässt sich in drei Ebenen unterteilen:

Vollständige Transparenz. Eine ideale Middleware sollte aus der Perspektive eines Systementwicklers vollständige Transparenz bezüglich ihrer Aufgaben bieten (s. a. Abschnitt 4.2). Um dieses Ziel erreichen zu können muss eine Middleware auf die Problemstellung und die Anwendung angepasst sein. Eine Middleware muss eine gene-

rische Lösung liefern, die unter allen denkbaren Szenarien einer Anwendung funktioniert. Anwendungsspezifische Middleware liefert daher häufig nur einen eng mit der Problemstellung verwobenen Funktionsumfang.

Konfigurierbare Automation. Der Versuch eine vollständige Transparenz zu erreichen erfordert ein hohes Maß an Automation durch das System. Hierdurch wird die Flexibilität in der Wahl der Szenarien oftmals erheblich eingeschränkt. Abhilfe kann die Schaffung eines Konfigurationsrahmens bringen. Dieser vermindert zwar die Transparenz der Middleware, führt jedoch im Gegenzug Kontrollmöglichkeiten ein, die die Einsatzmöglichkeiten der Infrastruktur deutlich steigern können. Es besteht jedoch die Gefahr, dass zu komplexe Konfigurationsmöglichkeiten die Vorteile der Automation überwiegen.

Aufzeigen von Lösungswegen. Wenn auch die konfigurierbare Automation nicht mehr zu brauchbaren Ergebnissen führen kann, ist es hilfreich, dem Benutzer zumindest alle zur Bewältigung der Herausforderung notwendigen Informationen aufzubereiten und anzubieten. Ausgehend von dem Wissen, dass einige Aufgaben nicht vollständig erfasst werden können oder auch einen zu großen Lösungsraum haben, kann es sein, dass eine solche Methode die einzige Option darstellt.

Eine feine Abstimmung des Grades der Transparenz beeinflusst die Systementwicklung in großem Maße. Das Verbergen der Verteilung des Systems schafft in der Programmierung eine sehr anwendungsorientierte Umgebung, kann aber beispielsweise aufgrund des mangelnden Bewusstseins für die Verteilung ebenfalls zu Schwierigkeiten führen. Fowler [2002] stellt heraus, dass Aufrufe entfernter Methoden im Vergleich zu lokalen Aufrufen sehr viel langsamer seien. Die Leistungsfähigkeit einer verteilten Anwendung kann dadurch stark negativ beeinflusst werden. Es ist daher darauf zu achten, dass die Kommunikation über die Grenzen der Komponenten effizient und mit wenigen Aufrufen möglich ist. Hierzu ist es allerdings wichtig, dass dem Anwendungsentwickler der Charakter einer Methode (entfernt oder lokal) bewusst ist. Dieses Schaffung dieses Bewusstseins und die damit verbundene Unterscheidung von Methodentypen steht allerdings dem Aspekt der Transparenz der Verteilung entgegen (vgl. Abschnitt 5.1.2).

5.1.3 Unterstützte Aufgaben

Die Aufgaben, die ein System unterstützen soll haben meist den größten Einfluss auf dessen Design. Im Folgenden werden die drei am häufigsten unterstützten Funktionalitäten pervasiver Systeme vorgestellt.

Spontane Interaktion. Unabhängig von der zugrundeliegenden Systemorganisation (Abschnitt 5.1.1), ist es für pervasive Systeme wichtig, dass sie die spontane Interaktionen der verbundenen Geräte untereinander unterstützen. Es muss den Teilnehmern ermöglicht werden, sich gegenseitig zu finden und miteinander zu kommunizieren. In einem pervasiven System kommen verschiedenste Geräte zum Einsatz. Dieses Eigenschaft umfasst zum einen klassische Heterogenitätskriterien, wie Soft- und Hardwareplattformen (vgl. [Coulouris u. a., 2005, S. 16f.]) und zum anderen die Heterogenität der verfügbaren Ressoucen [Schiele u. a., 2010, S. 206f.].

Kontext-Management. Es wird von einer pervasiven Anwendung erwartet, dass sie nicht nur die bloße Interaktion unterstützt, sondern auch fähig ist, die Eigenschaften der umgebenden realen Welt mit einzubeziehen. Hierzu ist es erforderlich, dass die Anwendung in der Lage ist, die Umgebung über entsprechende Sensoren wahrzunehmen. Es kann ebenfalls erforderlich sein, die Daten von auf mehrere Geräte verteilten Sensoren ermitteln zu müssen. Die Anwendung muss dabei nicht nur mit wechselnden Verfügbarkeiten der Sensoren umgehen können, sondern auch mit gegebenenfalls unterschiedlichen Datenrepresentationen, die sich aus der Verwendung unterschiedlicher Sensortypen ergeben [Schiele u. a., 2010]. Die zentrale Unterstützung von Kontext-Management-Features durch eine Middleware ist sinnvoll, da hier alle kontextrelevanten Informationen zusammenlaufen [Mikalsen u. a., 2006b,a].

Adaptionsfähigkeit der Anwendung. Die sich durch die Kombination vieler unterschiedlicher Geräte ergebene Heterogenität der verwendeten Endgeräte, sowie die massive Verteilung pervasiver Anwendungen erfordern ein hohes Maß an Anpassungsfähigkeit. Damit eine Anwendung in einem solchen System funktionieren kann, muss sie in der Lage sein, sich auf die unterschiedlichsten Eigenschaften und Verfügbarkeiten von Ressourcen des Systems einzustellen. Die Anpassungsfähigkeit kann seitens der Middleware durch die Möglichkeit der *konfigurierbaren Automation* (Abschnitt 5.1.2) erreicht und vereinfacht werden [Schiele u. a., 2010]. Ein komponentenbasiertes Design wird von Geihs u. a. [2006] als die vielversprechendste Möglichkeit angesehen, Adaptionsfähigkeit durch ein System erreichen zu können.

5.1.4 Aktormodell und Software-Agenten

Das Aktormodell wurde erstmals in Hewitt und Baker [1977] definiert[1]. Es beschreibt eine asynchrone nachrichtenbasierte Kommunikation zwischen parallelen Prozessen. Das Aktormodell wurde beispielsweise in Programmiersprachen wie Erlang oder Scala (s. a. Hinweise in Abschnitt 4.1) aufgegriffen, die es als ein zentrales Programmiermodell unterstützen.

[1] vgl. hierzu auch die in Minsky [1988] vorgestellten Ideen zur *Society Of Mind*

Agentenbasierte Systeme, wie sie z. B. mit dem Framework Jade (Pokahr u. a. [2003]) realisert werden können verwenden ebenfalls das Aktormodell. Haller und Odersky [2007] beschreiben, wie mit Hilfe des Aktormodells leichtgewichtige, multithreaded und eventbasierte Systeme in Scala realisiert werden können.

5.1.5 Diskussion

Die von Schiele u. a. [2010] hervorgehobenen Charakterisierungen einer Middleware für pervasive Umgebungen geben einen Überblick der für die Entwicklung einer Architektur für Smart Homes wichtigen Designentscheidungen. Die Anwendbarkeit auf die Gebiete der Ambient Intelligence und Smart Homes ist möglich, da die hier vorgestellten Vorüberlegungen lediglich die gemeinsamen Grundlagen ubiquitärer Systeme berühren (vgl. Kapitel 2).

Aufgrund der räumlichen Begrenzung auf den Bereich einer Wohnung, bietet sich bei der Wahl der Organisationsform (Abschnitt 5.1.1) eines Smart Homes das Paradigma der *Smart Environments* an. Diese Einordnung wird durch die Art der in Abschnitt 3.2 vorgestellten Arbeiten und Planungen im Rahmen der Labore iFlat und Living Place Hamburg weiter gestützt[2]. Die Sicht des Systems ist somit hauptsächlich auf die Interaktion des Benutzers mit stationär installierten Geräten ausgerichtet. Die spontane Einbindung mobiler Geräte soll möglich sein und ist erwünscht, stellt allerdings – gemessen an der Anzahl der eingesetzten Geräte – nicht den Regelfall dar.

Die Wahl der angestrebten Abstraktionsebene (Abschnitt 5.1.2) ergibt sich aus den Anforderungen (Abschnitt 3.5), dass sowohl eine vollständige Transparenz angestrebt wird („Transparenter Zugriff auf Dienste und Ressourcen“, S. 56), aber auch die Notwendigkeit einer konfigurierbaren Automation als erforderlich angesehen wird („Konfigurationsrelevante Anforderungen“, Abschnitt 3.5.2).

Die Wahl der direkt durch das Systemdesign unterstützten Aufgaben (Abschnitt 5.1.3) lässt sich ebenfalls aus den Anforderungen ableiten. So bilden die *kofigurationsrelevanten Anforderungen* Abschnitt 3.5.2) fast ausschließlich die Forderung nach spontaner Interaktion ab[3] Im Rahmen der kommunikationsrelevanten Anforderungen (Abschnitt 3.5.1) wurden indirekt Anforderungen nach der Bereitstellung von Kontextinformationen gestellt. Diese wurden mit der Einführung des Konzepts des Complex Event Processing (Abschnitt 4.3) noch weiter gefestigt. Zusätzlich wird die Anpassungsfähigkeit der Anwendung durch den Laborbetrieb gefordert, der die laufende Fluktuation von Komponenten und Funktionalitäten innerhalb des Systems zur Folge hat. Die Architektur muss demnach der hohen Dynamik der Umgebung angepasst sein. Eine Möglichkeit des Umgangs mit Dynamik in verteilten und nebenläufigen Systemen ist die Implementierung der Systemkomponenten als in sich geschlossene

[2] vgl. hierzu auch die Analyse der Anwendungsszenarien in Abschnitt 3.4

[3] Einzige Ausnahme: „Einteilung in virtuelle Netze“, S. 58f.

Software-Agenten. Diese Agenten sollen in der Lage sein, ein gemeinsam verstandenes Protokoll zu unterstützen (Abschnitt 3.3.4). Die Kommunikation mittels asynchroner Nachrichtenübermittlung, wie sie Teil des Aktormodells ist, ist zudem hilfreich, die nebenläufigen Aktionen des Systems verständlicher zu machen (Abschnitt 4.1).

5.2 Vision: Eine agentenbasierte Basisarchitektur für Smart Homes

Ausgehend von der Idee, dass das Verhalten eines Smart Homes sich emergent aus der Verknüpung und Interaktion seiner verbundenen Einzelsysteme ergibt und nicht zentral geplant und gesteuert wird, soll eine Infrastruktur geschaffen werden, die es den angeschlossenen intelligenten Agenten (vgl. Abschnitt 3.3.4) ermöglicht nahtlos Informationen und Anfragen miteinander auszutauschen. Es wird daher der Einsatz einer Middleware als fest installierte Vermittlungskomponente des Smart Homes (als Beispiel für eine Smart Environment) vorgeschlagen, mit deren Hilfe Dienstnachfrager und -anbieter miteinander in Kontakt treten können. Mobile Benutzer und Geräte sollen mit der stationär installierten Infrastruktur interagieren können. Abbildung 5.1 zeigt eine schematische Darstellung dieser Organisationform für Smart Environments.

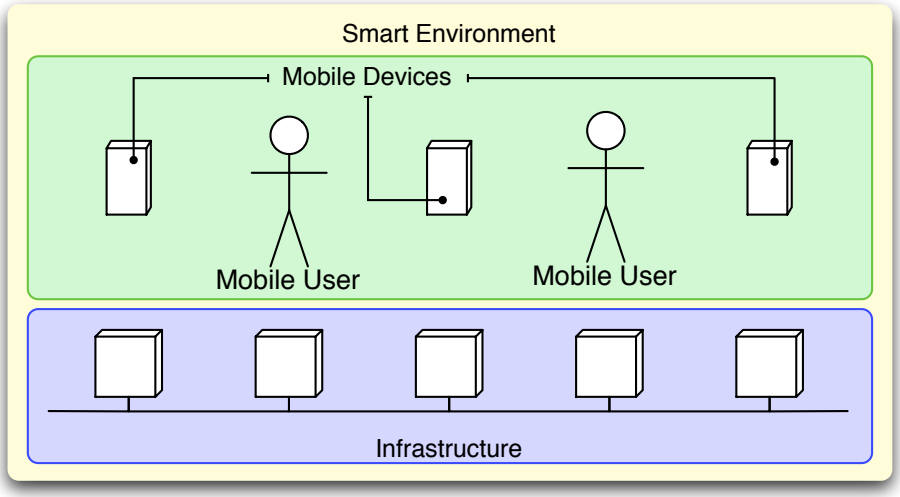

Abbildung 5.1: Systemorganisation als Smart Environment

Die Architektur soll eine Middlewarekomponente enthalten, deren Aufgabe es ist, Dienste kontextsensitiv zwischen den Peers (auch Agenten) zu vermitteln. Auftretende Events sollen über dynamische und statische Filterregeln an die angeschlossenen Peers übermittelt

werden. Die Middleware bietet den Clients eine Schnittstelle an, über die dynamisch eine umfangreiche Beschreibung von Events und Filterregeln möglich ist. Es soll so gewährleistet sein, dass ein Client auch nur die Events erhält, die er verarbeiten kann und soll. Jeder Peer teilt der Middleware zusätzlich eine Selbstbeschreibung („Aktor, Sensor oder hybrid“, „Ort der Installation“, „Grad der Mobilität“, etc.) mit, so dass ebenfalls eine kontextbezogene Auswahl möglich ist, die unabhängig von aktiv angeforderten Events ist. Die kontextbezogene Auswahl von Ressourcen kann auch hilfreich sein, wenn es um die Bewertung der zur Auswahl stehenden Geräte geht. Im Regelfall ist der Fernseher im Wohnzimmer die Beste Wahl zur Anzeige des Bilds eines Video-Chats. Sollte der Benutzer allerdings nicht alleine im Raum sein, wäre die Anzeige auf dem Bildschirm des Smartphones aufgrund der Mobilität des Geräts und besseren Wahrung der Privatsphäre empfehlenswert (s. Abschnitt 5.1.3, sowie Szenario „Gegensprechanlage“ in Abschnitt 3.4.2). Ein derartiges Systemverhalten kann zu einer deutlichen Flexibilisierung des Gesamtsystems führen (s. a. [Paspallis und Papadopoulos, 2006; Paspallis, 2009]).

Die Kommunikation zwischen den einzelnen Peers soll nur in Ausnahmefällen unter Umgehung der Middleware und somit direkt möglich sein. Ein solcher Fall kann eintreten, wenn z. B. große Datenmengen zwischen zwei Endpunkten übertragen werden sollen und die Umleitung über einen Koordinatorknoten zu ineffizient wäre. Die Adressierung von Peers soll über die Angabe von Zielen geschehen, so dass für die Dienstsuche nicht erst die betroffenen Geräte ermittelt werden müssen[4]. Es kann dabei vorkommen, dass bestimmte Ziele die Erreichung anderer Ziele blockieren oder verhindern. Ein Beispiel wäre z. B. ein durch das Fernsehprogramm blockierter Bildschirm. Die Gründe für die Nichtverfügbarkeit von Ressourcen können vielfältiger Natur sein, die Architektur muss daher die Behandlung dieser Konflikte adäquat abbilden und auflösen können.

Aus Sicht des Systems ergibt sich ein Netzwerk aus vielen Agenten, die zum einen die physischen Möglichkeiten ihrer zugehörigen Geräte repräsentieren(vgl. Szenario „Cooking Agent“ in Abschnitt 3.4.3) und zum anderen als reine Softwareagenten zur Erfüllung einer Aufgabe existieren. Durch die zusätzliche Integration von Softwareagenten wird beispielsweise auch die Bündelung einzelner Agenten ermöglicht (vgl. Szenario „Wecker 2.0“ in Abschnitt 3.4.1). Beispiele für Softwareagenten können ein Kalender sein oder auch die Gegensprechanlage, die aus einer Kamera, einem Mikrofon, dem Klingelknopf, sowie aus dynamisch zu wählenden Ausgabegeräten besteht[5]. Für den Benutzer tritt die Technologie der Wohnung im Idealfall vollständig in den Hintergrund. Er nimmt eine normale Wohnung wahr, in der die Technik zwar allgegenwärtig ist, jedoch lediglich ein Hilfsmittel in seinem Handeln darstellt (vgl. Kapitel 2).

[4] z. B. *„Informiere Carla, dass es an der Tür geklingelt hat.“* im Gegensatz zu *„Spiele Audiosignal der Türklingel über die Lautsprecher in Küche und Wohnzimmer.“*

[5] weitere Beispiele können den Szenarien in Abschnitt 3.4 entnommen werden

5.3 Grobkonzeption der Middlewarekomponente

Die Grobkonzeption der Middlewarekomponente ist der erste Schritt des iterativen Systementwurfs. Es werden im Folgenden die einzelnen Komponenten der Architektur vorgestellt, sowie ihre jeweiligen Aufgaben und Beziehungen untereinander beschrieben. Der Entwurfsprozess orientiert sich dabei am komponentenbasierten Design [s. Geihs u. a., 2006].

Die vorhandenen Projekte in den Laboren iFlat und Living Place Hamburg bestehen bereits überwiegend aus einer Vielzahl unabhängiger und lose gekoppelter Einzelkomponenten. Werden jetzt neue Szenarien umgesetzt, kann dabei häufig nur ein sehr geringer Anteil der bereits bestehenden Infrastruktur unverändert wiederverwendet werden. Dieses Vorgehen schränkt die Möglichkeit der Interaktion zwischen einzelnen Systemkomponenten untereinander deutlich ein und erschwert die Implementierung eines Gesamtsystems, da häufig verwendete Teilkomponenten der Infrastruktur, wie Kommunikationsschicht, Dienstfindung und Dienstvermittlung im Rahmen jedes einzelnen Projekts eigenständig definiert und angepasst werden müssen. Der Architekturentwurf sieht daher die Wiederverwendbakeit der häufig verwendeten Infrastrukturdienste vor. Es soll eine Basis geschaffen werden, auf der eine effiziente Nutzung des Gesamtsystems zur Umsetzung von Ideen möglich ist, ohne dabei die Flexibilität bei der Entwicklung der Einzellösungen einzubüßen (s. a. Abschnitt 3.1). Die aus Hollatz [2008c] gezogenen Rückschlüsse[6] zeigen, dass die Verwendung eines Blackboard-ähnlichen Systems das geforderte Maß an Flexibilität und Performance bietet, wenn auch die verwendete Implementation des iROS EventHeaps fehleranfällig ist[7].

Die Middleware abstrahiert Standardaufgaben der Infrastruktur und ermöglicht es so den Entwicklern, den Fokus der Arbeit auf die eigentliche Problemstellung zu lenken. Sie soll zudem flexibel einsetzbar und erweiterbar sein. Zu ihren Kernaufgaben zählen die Verwaltung, Abfrage und Routing von Events, Dienstentdeckung, sowie Gewährleistung von Ausfallsicherheit (vgl. Abschnitt 3.5). Sie soll somit als zentrale Schnittstelle zwischen Agenten, Aktuatoren und Sensoren agieren. Der erste Architekturansatz der Middleware berücksichtigt bereits die Bereitstellung dieser Basisdienste (s. Abbildung 5.2). Zur Kommunikation mit den angeschlossenen Teilnehmern soll eine flexible und möglichst zustandslose Message Queue verwendet werden. Diese bildet die Einheitliche Kommunikationsschnittstelle für alle angeschlossenen Clients und soll gewährleisten, dass Änderungen im Kern der Infrastruktur möglichst keine Anpassungen an den bestehenden Clients erfordern. Diese Stabilitätsanforderung ist wichtig, da bei der zu erwartenden Vielzahl von Agenten im Living Place eine alle Clients betreffende Anpassung sehr aufwendig wäre (vgl. Abschnitt 3.5.4). Darauf aufbauend stellt ein zustandsbehafteter Event Handler die Möglichkeit der Vorverarbeitung von Events bereit. Als dritte Basiskomponente soll ein Service-Discovery-Dienst eingesetzt werden, der

[6] vgl. hierzu auch Hollatz [2007],Hollatz [2008a] und Hollatz [2008b], Meißner [2008c], Urich [2008b] und Urich [2009]

[7] persönliche Mitteilung, Sören Voskuhl, 20.04.2010

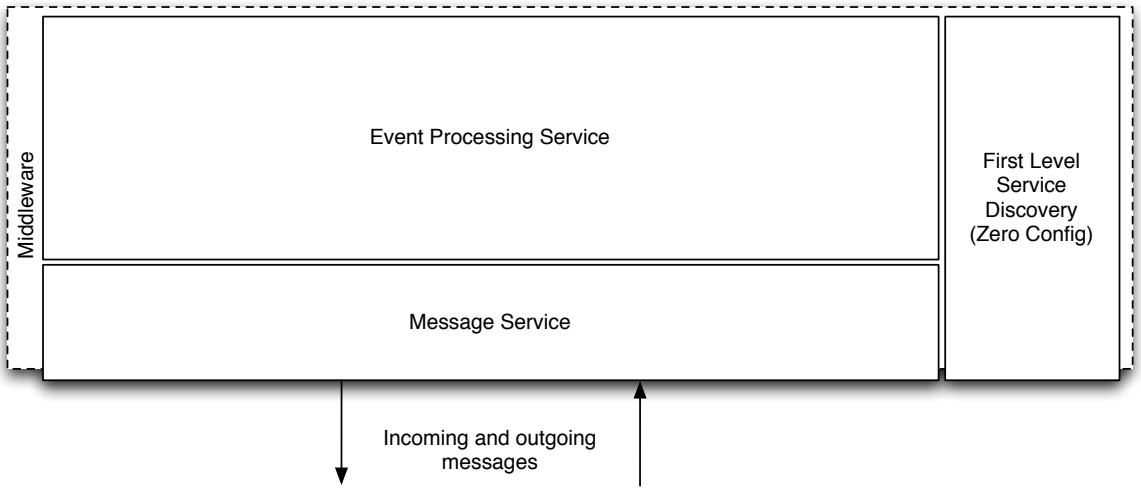

Abbildung 5.2: Vereinfachtes Komponentendiagramm einer Middlewarearchitektur für Smart Homes

es den Clients ermöglicht, die Middleware im Netzwerk zu finden und sich mit dieser ohne eine manuelle Konfiguration verbinden zu können.

Der erste Zugriff an die Middleware soll über die Schnittstelle der Service-Discovery erfolgen, diese liefert dem anfragenden Peer die notwendigen Informationen um sich an der Middleware anzumelden. Der Zugriff auf die Dienste der Middleware soll nur mittels Nachrichten über die Kommunikationsschnittstelle („Message Service“) erfolgen. Diese identifiziert die Art der Nachricht und leitet sie an den Event Handler, der die weitere Verarbeitung der Nachrichten vornimmt. Die konkrete Implementation der hinter der Schnittstelle angespreochenen Dienste ist für einen Client nicht von Bedeututng. Die angeschlossenen Clients bestehen ihrerseits wiederum aus physischen Geräten (Ereignisquellen und/oder -senken) die über eine eigene Agentenlogik gesteuert werden. Die Kommunikation mit anderen Agenten erfolgt zwischen den Kommunikationsschnittstellen des Peers und der Middleware (s. Abbildung 5.3).

5.4 Zentrale Architekturentscheidungen

Bevor der eigentliche Architekturentwurf entwickelt wird, sollen hier einige grundlegende Entscheidungen, die im Rahmen des Architekturentwicklung getroffen wurden vorgestellt werden. Diese haben direkten Einfluss auf das weitere Vorgehen zur Entwicklung der Architektur. Sie betreffen im Einzelnen den grundlegenden Aufbau des Netzwerks, des Kommunikationsprotokolls, der Nachrichtenverteilung sowie der technischen Architektur der Middleware.

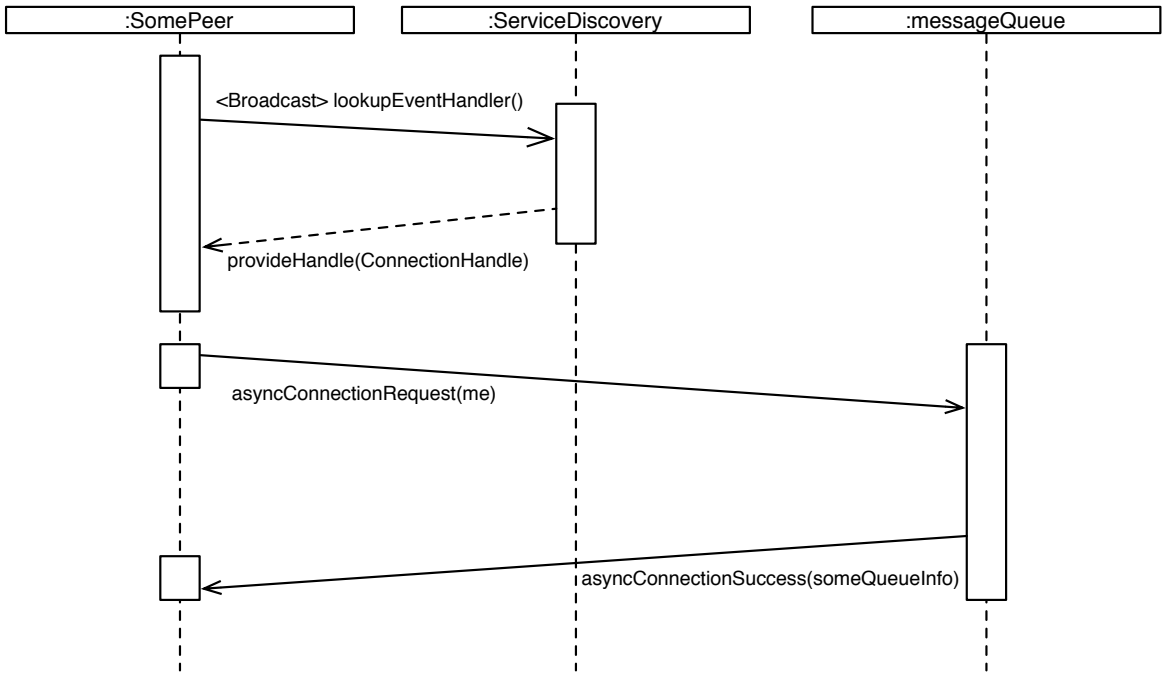

Abbildung 5.3: Vereinfachtes Sequenzdiagramm eines Handshake zwischen Peer und Middleware

5.4.1 Anlehnung an das Tuplespaces-Architekturmodell

In Erman u. a. [1980] wird das Blackboard-Architekturmodell beschrieben, dessen Weiterentwicklung zum Tuplespaces-Modell in Ahuja u. a. [1986] erläutert wird. Johanson und Fox [2004] zeigen ergänzend dazu die Erweiterung von Tuplespaces zum Konzept des Event Heap, der u. a. in Hollatz [2008c] zum Einsatz kam.

Blackboard-basierte Systeme bieten den Vorteil, dass nicht der Sender einer Nachricht den Empfänger adressiert, sondern die mögliche Empfänger die für sie interessanten Nachrichten abrufen können. Das Konzept ähnelt dem eines schwarzen Bretts, an dem Nachrichten angeschlagen werden. Dieses Modell der Kommunikation ist in einem Smart Home wie dem Living Place Hamburg sehr vorteilhaft, da es ermöglicht, auf einfache Weise eine Vielzahl gleichartiger Geräte anzusprechen. So muss ein Lichtschalter, der im Wohnzimmer alle Lampen schalten soll, nicht jede Lampe einzeln adressieren, sondern lediglich das *„Licht im Wohnzimmer“* schalten und die betreffenden Geräte können die entsprechende Nachricht abrufen, indem sie Nachrichten mit dem Thema *„Licht“* am Ort *„Wohnzimmer“* abfragen.

Damit nicht jedes Gerät ständig den Koordinator abfragen muss, ob neue Events für ihn vorliegen, besteht die Möglichkeit, bestimmte Event-Muster zu registrieren. Wenn der Koor-

dinator ein entsprechendes Muster erkennt, so sendet er die Nachricht direkt an den Empfänger.

5.4.2 Nachrichtenbasierte Kommunikation nach dem Aktormodell

Das Living Place Hamburg beschreibt ein stark verteiltes Event-basiertes System. Ein Event ist eine Nachricht, die einem oder mehreren Empfängern zugänglich gemacht werden soll. Wenn man nun die einzelnen Peers im Netzwerk als Aktoren einordnet die ihre Events dem Koordinator mitteilen, der diese dann an interessierte Empfänger weiterleitet. Dieses Kommunikationsmodell ermöglicht auf einfache Weise die übersichtliche Schaffung einheitlicher und flexibler Kommunikationsstrukturen.

5.4.3 Peer-to-Peer im koodinierten Overlay-Netzwerk

Die in Abschnitt 4.1 vorgestellte Form der Modellierung nebenläufiger Systeme bietet es an, dass die Endpunkte eines Systems prinzipiell als gleichberechtigte Knoten (Peers) eingeordnet werden können. Gestützt wird diese Einordnung durch die folgende Definition von Androutsellis-Theotokis und Spinellis:

> *„Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority.“*
>
> – [Androutsellis-Theotokis und Spinellis, 2004]

Dabei ist zu beachten, dass sich der Term „global centralized server or authority“ auf eine Komponente Bezieht, die zentralen Einfluss auf die Funktionalität des Systems nimmt. Buford [2008] erwähnt, dass der Einsatz eines zentralen Koordinators nicht schädlich für die Definition eines Peer-to-Peer-Netzes sein muss, sondern sogar sinnvoll sein kann, um z. B. die zentralen Funktionalitäten des Netzwerks zu koordinieren. Ergänzend hierzu wurde in Abschnitt 4.2 die Möglichkeiten zum Einsatz einer Middleware erläutert und der Einfluss einer Middleware auf ein verteiltes System analysiert. Die Abbildungen 5.4 und 5.5 zeigen den Unterschied in der Anzahl der Verbindungen an den Peers zwischen einem vollvermaschten Peer-to-Peer-Netz (Abbildung 5.4), in dem die Komplexität der Anzahl der Verbindungen mit $O(N)$ ansteigt und der Art der indirekten Verbindung über einen Koordinator (Abbildung 5.5), der die Komplexität der Verbindungen der Peers konstant mit $O(1)$ hält.

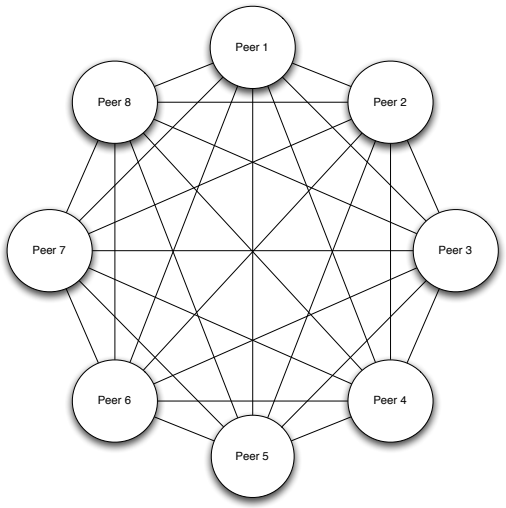

Abbildung 5.4: Kommunikation im Peer-to-Peer-Netz ohne Koordinator

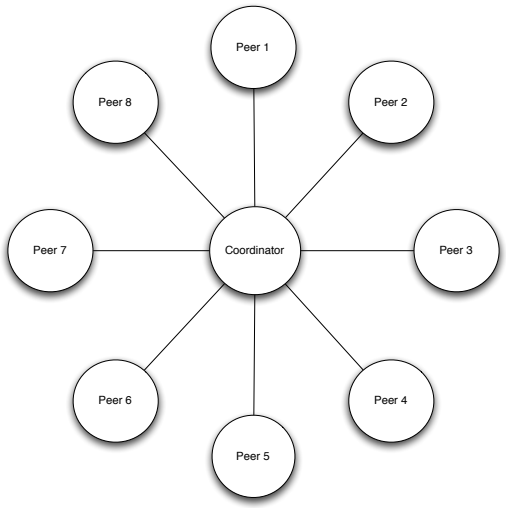

Abbildung 5.5: Kommunikation im Peer-to-Peer-Netz mit Koordinator

Zusätzlich werden durch den Einsatz eines Koordinators weitere Möglichkeiten eröffnet, die nicht nur für die Entwicklung in einer Laborumgebung von Bedeutung sein können:

Zentrale Beobachtung von Events. Durch die Verteilung der Events mittels eines zentralen Koordinators können die auftretenden Ereignisse beobachtet und aufgezeichnet werden. Es wird dadurch eine Möglichkeit des Debuggings geschaffen, mit deren Hilfe sich das Verhalten des Gesamtsystems leichter nachvollziehen lässt.

Zentrale Haltung von Zuständen. Die vorgeschlagene Architektur bietet die Möglichkeit, allgemeingültige Zustände zentral zu verwalten. Es kann sich dabei beispielsweise um abgeleitete Zustände handeln, die durch das Auftreten mehrerer Events entstehen. So lässt sich die Kontextinformation *„Es ist jemand im Haus“* aus einer regen Verwendung der Internetverbinung in den letzten 30 Sekunden in Verbindung mit dem eingeschalteten Radio schließen.

Vereinfachtes Sicherheitskonzept. Da die Peers zur Kommunikation mit dem Koordinator verbunden sein müssen lässt sich an dieser Stelle auch ein zentrales Sicherheitskonzept implementieren. Das Absichern des Nachrichtenverkehrs ist in einer realen Umgebung von Bedeutung kann im Labor aber häufig vernachlässigt werden. Eine mögliche Anwendung im Labor wäre die Segmentierung des Koordinators für verschiedene Anwendungsfälle. Es ließen sich auf diese Weise verschiedene Teilsysteme getrennt voneinander untersuchen, ohne sie explizit auf eine alternative Installation des Koordinators zu konfigurieren.

Modularisierung / Plug-In-Architektur möglich. Insbesondere eine sich in der Erfor-

schung befindliche Umgebung sollte in der Lage sein, schnell und unkompliziert auf neue Anforderungen angepasst zu werden. Die Organisation einzelner Komponenten in einer Plug-In-Architektur erscheint so als sinnvoll (vgl. hierzu auch Abschnitt 5.4.4).

Vereinfachtes Routing und Lastverteilung möglich. Bei einer Skalierung des Architekturmodells mit einer Unterteilung in verschiedene Subnetze mit mehreren Koordinatoren benötigt die Clientlogik keine Anpassung, da diese Veränderung lediglich die Logik der Koordinatoren betrifft. Ein Routing von Events zwischen den Subnetzen kann somit vollkommen transparent erfolgen. Das gleiche gilt für eine Lastverteilung, die auf verschiedene Weise bereits durch die Kommunikationsschicht durchgeführt werden könnte. Beispiele für eine Segmentierung der Aufgaben könnte dabei die Art oder Lokalität von Events sein.

Der Einsatz einer zentralen Middleware kann demnach die Komplexität des Nachrichtenverkehrs, sowie die der Client-Logik signifikant senken und bringt zusätzliche Vorteile mit sich. Diese Punkte gelten um so mehr in einer stark verteilten Umgebung wie dem Living Place Hamburg, in der mit sehr vielen vernetzten Clients gerechnet werden kann. Das Routing des gesamten Nachrichtenverkehrs über einen zentralen Koordinator kann allerdings zu Schwierigkeiten der Ausfallsicherheit und Skalierbarkeit mit sich ziehen (Single Point of Failure). Während die in einem vollvermaschten Netz von Peers die Komplexität nur theoretisch mit $O(N)$ ansteigt, zeigt sich am Beispiel des Koordinators, dass hier die gleichen Komplexitätsbedingungen gelten, die Verbindungen zu den Peers aber auch garantiert genutzt werden.

Abhilfe kann in diesem Fall der Einsatz eines Overlay-Mesh-Netzwerks schaffen, das den Einsatz mehrerer miteinander vernetzter Koodinatoren vorsieht. Abbildung 5.6 zeigt schematisch, wie ein derartiges Netzwerk organisiert sein kann. Der Begriff „Ovelay“ bezieht sich dabei auf die Implementation des Netzwerks in der Nutzdatenebene der Netzwerkinfrastruktur. Buford und Yu definieren ein Overlay-Netzwerk folgendermaßen:

> *„An application layer virtual or logical network in which end points are addressable and that provides connectivity, routing, and messaging between end points. Overlay networks are frequently used as a substrate for deploying new network services, or for providing a routing topology not available from the underlying network. Many peer-to-peer systems are overlay networks that run on top of the Internet.“* – [Buford und Yu, 2010]

Die Aufteilung auf mehrere Koordinatoren kann beispielsweise auf Grundlage des Orts oder der Zugehörigkeit der einzelnen Peers erfolgen. Das Routing zwischen den Koordinatoren kann auch in diesem Fall transparent gestaltet werden. Der Nachteil der Verteilung der Koordinatoren ist, die erschwerte Ermittlung eines globalen Systemzustands.

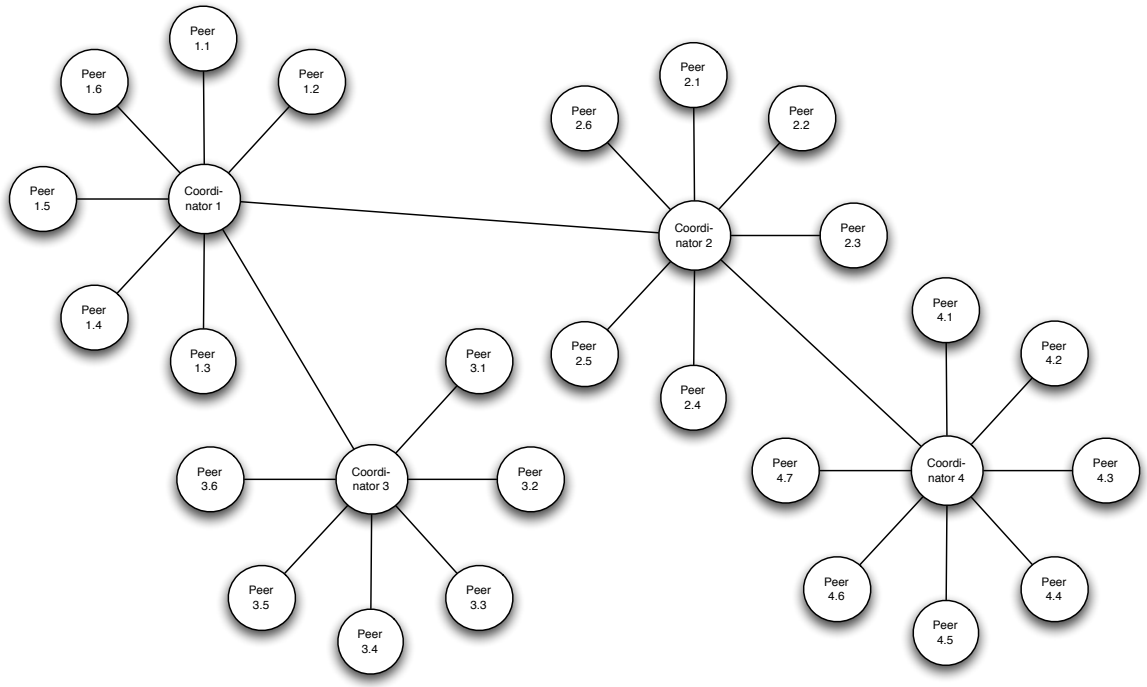

Abbildung 5.6: Kommunikation im Peer-to-Peer-Mesh-Netz (über Koordinatoren)

5.4.4 Separation of Concerns

Analog zu dem von Paspallis und Papadopoulos [2006] beschriebenen Konzept der „Separation of Concerns“ soll das Architekturkonzept es zulassen, die Funktionalitäten des Systems getrennt voneinander entwickeln zu können. In Abschnitt 5.4.3 wurden bereits die Vorteile einer Plug-In-Architektur für den Koordinator teilweise erläutert. Plug-In-Architekturen finden ihre Anwednung für gewöhnlich in Systemen, in denen ein hohes Maß an Anpassungsfähigkeit erwartet wird. Ein weiteres Argument für den Einsatz dieses Architekturmusters ist die Modulariesierung der Entwicklung komplexer Anwendungen. Plug-In-basierte Systeme bestehen meist aus einem kleinen Kern, der die Laufzeitumgebung der Komponenten bereitstellt. Jede Komponente definiert ihrerseits Schnittstellen, die von anderen Komponenten genutzt werden können. Die klare Definition von Schnittstellen zwischen den Komponenten ermöglicht es den Entwicklern bereits das Dienstangebot anderer Komponenten anzusprechen, ohne dass eine konkrete Implementation der jeweiligen Komponente existiert. Diese Mock-Ups können dann später durch funktionsfähige Versionen des Plug-Ins ersetzt werden.

Ein Beispiel für eine Plug-In basierte Plattform ist die Eclipse Rich-Client-Plattform (RCP), auf deren Basis die Eclipse IDE und andere Anwendungen entwickelt werden. Die Eclipse

RCP basiert auf der OSGi-Spezifikation, die ein Framework definiert, das die Unterteilung von Software in unterschiedliche Komponenten (Bundles) erlaubt[8].

5.5 Architekturentwurf

Die im vorhergehenden Abschnitt beschriebene erste Systemvision soll als Ausgangspunkt für den weiteren Entwurf der Middleware dienen. Es wurde hier noch keine eigene Funktionalität des Smart Homes berücksichtigt. Es ist z. B. denkbar, dass von der Middleware auch eine Laufzeitumgebung für Meta-Agenten angeboten wird. Der Begriff Meta-Agenten bezeichnet dabei Agenten die keine physische Repräsentation besitzen. Es kann sich dabei z. B. um einen Terminplaner oder Videorekorder handeln, der die Aktionen verschiedener Agenten koordiniert. Weiterhin wurden Aspekte der Sicherheit und der Erweiterung des Event Handlers bisher nicht betrachtet. Abbildung 5.7 zeigt einen erweiterten Architekturentwurf, der den ersten Ansatz um eben diese Punkte erweitert. Die gezeigten Komponenten sollen im Folgenden kurz beschrieben werden.

Abbildung 5.2 stellt den ersten Zielentwurf der Middleware dar. Es wird hierbei die Idee eines zentralen Ansprechpartners verfolgt, der das Erscheinungsbild der Middleware nach außen prägt. Der Entwurf zeigt die Komponenten, die in auf Grundlage der Analyse in Kapitel 3 ermittelt wurden. Das *Communication Interface* und das *Event Board* bilden die Basisdienste der Architektur. Sie stellen die zentralen Dienste und Schnittstellen bereit, auf denen alle weiteren Komponenten aufbauen.

Message Queue

Die Message Queue bildet die Schnittstelle zur Kommunikation der Middleware mit externen Systemen. Aufgabe der Message Queue ist es, den reibungslosen und verlässlichen Ablauf der Kommunikation zu gewährleisten. Ihre Aufgaben beschränken sich dabei auf das bloße weiterleiten von Nachrichten zwischen den angeschlossenen Komponenten. Die Vereinheitlichung der Kommunikation innerhalb des Systems mit Hilfe der Message Queue ermöglicht eine deutliche vereinfachte Möglichkeit der Entwicklung im Team, da jeder Entwickler gegen eine wohldefinierte Schnittstelle entwickeln kann. Die Entscheidung für eine einheitliche Kommunikationsschnittstelle zwischen Teilnehmern ermöglicht zusätzlich die Bereitstellung von Daten aus unterschiedlichen Quellen an zentraler Stelle.

Um die Transparenz der Kommunikation wahren zu können, sollte zwischen der Beschreibungsprache für den Zugriff auf die Message Queue, sowie der konkreten Implementation

[8] Eine kurze Einführung in OSGi findet sich in Tavares und Valente [2008]

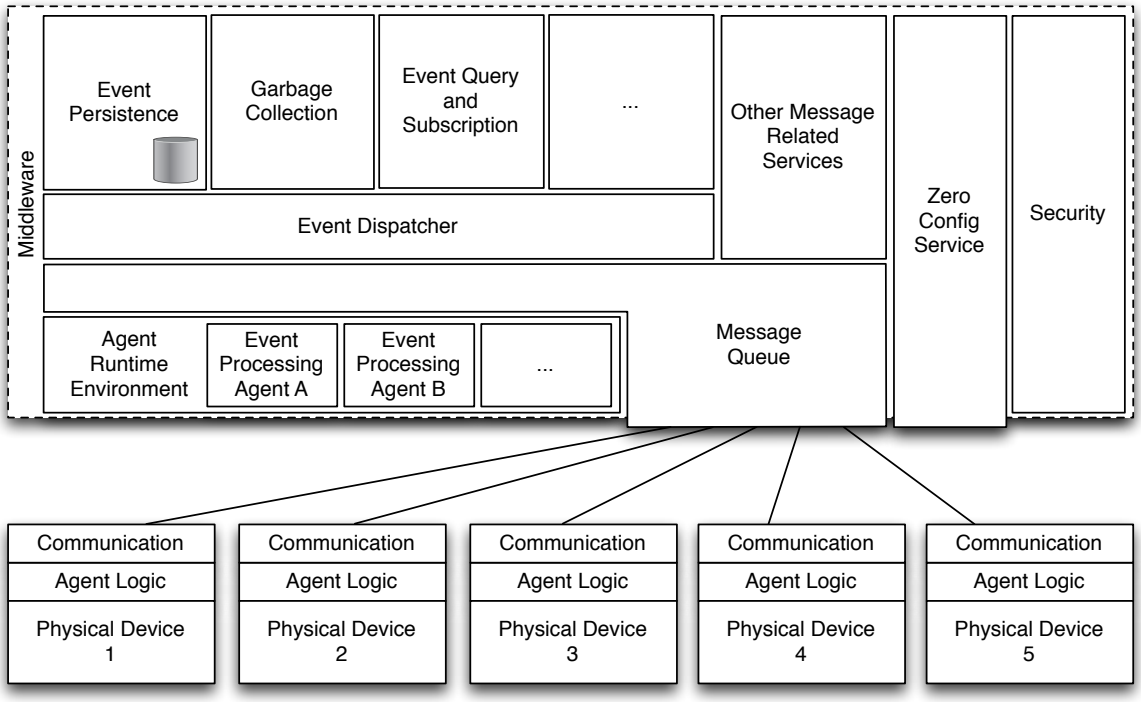

Abbildung 5.7: Erweitertes Komponentendiagramm einer Middlewarearchitektur für Smart Homes

des Zugriffs unterschieden werden. Der Einsatz Pluggable-Communication-Layers, analog zu JDBC oder JNDI wäre daher sinnvoll.

Event Dispatcher

Der Event Dispatcher stellt die zentrale Schnittstelle innerhalb der Middleware dar. Der Event Dispatcher nimmt die von der Message Queue empfangenen Events auf und leitet sie weiter. Im Event Dispatcher werden – ähnlich wie auf einem Arbeitstisch – Events abgelegt, geordnet und weiterverarbeitet. Die Weiterverarbeitung geschieht durch Arbeitermodule, die vom Dispatcher benutzt werden können[9]. Im Folgenden soll eine beispielhafte Beschreibung möglicher Module gegeben werden Diese beeinflussen direkt die Manipulation von Events und sind somit entscheidend das verhalten der Middleware.

Event Persistence. Es handelt sich hierbei um einen Festspeicher, der die Ausfallsicherheit des Systems ermöglichen soll. Alle auftretenden Events werden in dieser Datenbank

[9] Vgl. *Knowledge Sources* in Erman u. a. [1980]

gespeichert, so dass das System auch nach einem plötzlichen Ausfall seinen Zustand nicht verliert.

Garbage Collection. Der Begriff „Garbage Collection“ ist aus dem Bereich der automatischen Speicherverwaltung entlehnt, wie sie beispielsweise in einigen objektorientierten Programmiersprachen angeboten wird. Er wird allerdings auch im Zusammenhang mit anderen Speicherformen, wie Festspeichern verwended. Sockut und Iyer [2009] beschreiben, dass „Garbage Collection“-Techniken sinnvoll in Datenbank- und Data-Mining-Systemen eingesetzt werden können. Walzer u. a. [2008] nennt den Begriff sogar explizit im Zusammenhang mit „Complex Event Processing“.

Das Ziel einer Garbage Collection im Event Board ist es, die hinterlegten Events in Event Board und Persistence Store zu überwachen. Ungültige, überschriebene oder abgelaufene Events werden identifiziert und gelöscht. Die Garbage Collection kann außerdem Eventquellen darüber informieren, falls ein Event nicht abgerufen oder bearbeitet wurde.

Event Query and Subscription. Die Komponente *Event Query and Subscription* bietet den Agenten die Funktionalität, komplexe Anfragen zu Events und Event-Folgen zu stellen. Sie ermöglicht es den Teilnehmern einerseits aktiv nach gewissen Events zu suchen, andererseits bietet sie die Möglichkeit Events, die einem bestimmten Muster entsprechen zu abonnieren. Die aktive Suche nach Events hat den Vorteil, dass ein Teilnehmer den für ihn aktuell relevanten Status abfragen kann. Es kann sich dabei zum Beispiel um Licht- oder Temperatureinstellungen handeln oder auch um anstehende Kalendereregnisse. Das Abonnieren von Events ermöglicht es wiederum den Teilnehmern direkt auf Ereignisse zu reagieren, sobald diese auftreten, ohne ständig aktiv danach fragen zu müssen.

Chakravarthy und Adaikkalavan [2008] stellen in diesem Zusammenhang das Konzept eines *Local Event Detectors* (LED) vor. Der LED kann das Auftreten von Events anhand von Graphenalgorithmen analysieren und aufbereiten. Zusätzlich wird eine Pattern-Specification-Language vorgestellt, um komplexe Auftrittsmuster beschreiben und abfragen zu können..

Agent Runtime Environment

Die Agent Runtime Environment bietet die Möglichkeit, Agentenlogik direkt innerhalb der Middleware auszuführen. Die Idee dieser Komponente ist, dass hier zentrale Dienste in Form von Meta-Agenten angeboten werden, die keinem einzelnen physischen Gerät zugeordnet werden können. So kann hier beispielsweise ein Kalenderagent das Überwachen und Manipulieren von Kalendereregnissen verwalten oder ein Kontextagent anhand von komplexen

Ereignisfolgen kontextbezogene Ereignisse auslösen oder steuern. Agenten innerhalb der Laufzeitumgebung können mit anderen (internen oder externen) Agenten über das Event Board kommunizieren.

Service Discovery

Die Service-Discovery-Komponente dient der konfigurationslosen Einbindung von Clients in das System. Sie soll die aufwendige Konfiguration der Middlewareanbindung möglichst vollständig automatisieren. Zusätzlich ist es denkbar, dass diese Komponente zur Sicherung des nahtlosen Weiterbetriebs bei einem Systemausfall eingesetzt wird, um die Clients bei Bedarf auf ein bestehendes Backupsystem zu konfigurieren.

Sicherheit

Die Middleware übernimmt die Verwaltung von äußerst sensiblen Daten, die den für gewöhnlich persönlichsten Bereich – die eigene Wohnung – berühren. Es gilt diese Daten vor Fremdeinwirkungen und -zugriffen zu schützen. Die Kommunikationsschicht kann hierbei eine Point-to-Point-Sicherheit in Form von Verschlüsselung und Authentifizierung der direkten Gesprächspartner bieten. Die Aufgabe der gesonderten Sicherheitskomponete betrifft dabei vor allem die End-to-End-Sicherheit in einem weiteren Umfeld, das beispielsweise den Zugriff auf bestimmte (Teil-)Informationen schützt und auch in einer komplexen Umgebung, in der Daten zwischen mehreren Event Boards ausgetauscht und geroutet werden, den autorisierten Datenzugriff sicherstellt

Durch die Einführung einer flexiblen Sicherheitsarchitektur kann ein nützlicher Nebeneffekt erziehlt werden: Eine Segmentierung in unabhängige Teilnetze. Bei der Entwicklungsarbeit in einer Laborumgebung kann es häufig sinnvoll sein, nur eine begrenzte Anzahl von Teilkomponenten miteinander in Kontakt treten zu lassen, um Seiteneffekte ausschließen zu können. Eine Sicherheitsarchitektur kann diese Funktionalität bereitstellen, indem sie Steuerungsmechanismen für die Sichtbarkeit einzelner Eventquellen bereitstellt.

5.6 Möglichkeiten zur technischen Realisierbarkeit

5.6.1 Aufbau der Laufzeitumgebung

Das Ziel des Architekturentwurfs ist die Überführung in eine lauffähige Middleware. Die nächsten Iteration der Entwicklung sollten demnach der Implementation eines lauffähigen

Prototypen dienen. Der erste Schritt in diesem Prozess sollte die Wahl eines entsprechenden Komponenten-Frameworks sein. Die Wahl der Sprachplattform (Java, .NET, etc.) ist hierbei eher als sekundär zu betrachten. Die Middleware sollte jedoch aus Gründen der Übersichtlichkeit auf einer einheitlichen Plattform beruhen. Es werden im Folgenden zwei Beispiele für die Umsetzung der vorgestellten Architektur gegeben:

Application-Server. Die Verwendung eines Application-Servers (z. B. JBoss[10], basierend auf Java EE) liefert in einem Paket viele Komponenten der Basisarchitektur. Komponenten, wie z. B. ein Nachrichtendienst, eine Datenbankabstraktion oder auch Clustering-Mechanismen sind bereits in den Application-Server integriert.

Application-Server sind darauf ausgelegt, hochkomplexe und hochverfügbare Unternehmensanwendungen zu bedienen. Das hat zur Folge, dass eine solche Serveranwendung häufig nicht besonders leichtgewichtig ist. Es wird daher bereits eine leistungsfähige Hardware benötigt, auf der der Server ausgeführt werden kann. Hinzu kommt, dass die Integration vieler Standardtechnologien in das Auslieferungspaket die Flexibilität der Architektur begrenzen. Der Austausch, bzw. die Ergänzung einzelner Dienste ist zwar häufig möglich aber nicht immer trivial.

Verwendung eines *reinen* Komponenten-Frameworks. Die Wahl eines reinen Komponentenframeworks (z. B. basierend auf OSGi[11]) ermöglicht es, die Ausgestaltung der Architektur von Grund auf selbst zu gestalten. Die Spezifikation von OSGi beschreibt den Aufbau der Architektur aus Komponenten, sog. *Bundles*. Jedes Bundle unterliegt einer Versionierung und besteht aus einer in sich geschlossenen Implementierung, inkl. der benötigten Bibliotheken. Ein Bundle definiert seine Schnittstelle zu anderen Bundles und kann in seiner Spezifikation die Abhängigkeit zu anderen Bundles definieren.

Es ist auf diese Weise möglich, den Funktionsumfang der Anwendung selbst zu gestalten. Der Zugewinn an Flexibilität hat allerdings zum Nachteil, dass die Schaffung der Basisinfrastruktur zunächst aufwendiger wird.

Die Wahl zwischen einem Application-Server oder dem Aufbau einer Lösung auf einem reinen Komponenten-Framework lässt sich im Hinblick auf die Entwicklung des Zielsystems nur mittelbar beantworten. Kriterien für die Entscheidung können z. B. die Zahl der zur Verfügung stehenden Entwickler, die Leistungsfähigkeit des Host-Computers der Middleware und der gewünschte Grad der Selbstbestimmung des Basis-Funktionsumfangs sein.

[10] Website: http://jboss.org, Abruf: 14.06.2010
[11] Website: http://osgi.org, Abruf: 14.06.2010

5.6.2 Integration des Message Service

In Abschnitt 4.2.1 (Message-Oriented-Middleware) wurden bereits Beispiele für Java-basierte Frameworks auf Grundlage der JMS-Spezifikation gegeben. Die Spezifikation beschreibt einen Standard, nach dem Nachrichten asynchron zwischen entfernten Prozessen ausgetauscht werden können [Hapner u. a., 2002]. Es gibt einige Implementationen für JMS: wie z. B. SwiftMQ[12], IBM WebsphereMQ[13], JBoss hornetQ[14] oder Apache ActiveMQ[15]. Die aufgeführten Produkte unterscheiden sich beispielsweise in der Form ihrer Lizensierung (kommerziell, open-source, etc.) oder Leistungsfähigkeit über die JMS-Spezifikation hinaus (stand-alone oder eingebetteter Betrieb möglich, Schnittstellen zu anderen Sprachen/Frameworks).

JMS definiert die asynchrone Kommunikation zwischen Cients in zwei Kategorien: Queue und Topic. Queues werden zur Kommunikation zwischen zwei Partnern verwendet, was zur Folge hat, dass jede Nachricht nur einmal zugestellt wird. Topics repräsentieren ein Versendesystem, an dem sich mehrere Teilnehmer anmelden können. Der Server verteilt jede Nachricht an alle Empfänger des Topics.

Eine mögliche Implementation als Middlewarekomponente kann für jeden Peer eine eigene Queue vorsehen und Topics für Klassen von Peers (Licht, Heizung, etc.). Auf diese Weise ist eine rudimentäre Routingmöglichkeit für Events bereits mit dem Message Service gegeben.

5.6.3 Veröffentlichen und Abonnieren von Events

Das Sammeln und Verteilen von Events in einem Publisher-/Subscriber-System wird von Johanson und Fox [2002, 2004] für den Event Heap auf einer einfachen syntaktischen Weise beschrieben. Der Event Heap empfängt Nachrichtentupel, sogenannte Events, und kann diese anhand ihrer Eigenschaften (Vorhandensein bestimmter Felder oder gesetzter Werte) identifizieren und potentiellen Empfängern zuordnen. Der Event Heap speichert die empfangenen Events bis sie aktiv gelöscht werden oder auslaufen. Events haben einen Timeout-Wert, der die Dauer ihrer Gültigkeit angibt. Mit diesem Konzept soll eine Form von Ausfallsicherheit erreicht werden, indem Clients jederzeit den aktuellen Zustand des Systems abfragen können, nachdem sie sich (neu) verbunden haben.

[12] http://www.swiftmq.com, Zugriffsdatum: 16.05.2010
[13] http://www.ibm.com/webspheremq, Zugriffsdatum: 16.05.2010
[14] http://www.jboss.org/hornetq, Abrufdatum: 16.06.2010
[15] http://activemq.apache.org/, Zugriffsdatum: 16.05.2010

Der Umgang mit komplexen Ereignissen nach [Luckham, 2005] und [Eckert und Bry, 2009] wurde bereits in Abschnitt 4.3 vorgestellt und Chakravarthy und Adaikkalavan [2008] beschreiben, wie sich Eventmuster mit Hilfe eines auf der Graphentheorie basierenden Local Event Detectors erkennen lassen.

Zur Abfrage von Events wird von [Luckham, 2005] die RAPIDE-Pattern-Language beschrieben, eine deklarative Programmiersprache, die die Auswertung von Event-Strömen in Echtzeit ermöglicht. Weitere Varianten der Ereignisanfrage wurden bereits in Abschnitt 4.3.5 erwähnt [s. a. Eckert und Bry, 2009].

6 Schlussbetrachtung

Mit dieser Arbeit wurde ein flexibles Architekturmodell entwickelt, dass die Kommunikation der Komponenten in einem Smart Home ermöglicht. In diesem Kapitel werden noch einmal die wichtigsten Erkenntnisse zusammengefasst und bewertet.

6.1 Zusammenfassung

Die Entwicklung einer Architektur für ein Smart Home unter den Aspekten des Software Engineering ist eine große Herausforderung. Dabei sind viele Aspekte aus den Bereichen der verteilten, mobilen und intelligenten Systeme, sowie der IT-Sicherheit zu beachten. Der ausgeprägte Laborcharakter des Zielsystems erfordert zudem eine hochgradig flexible und anpassungsfähige Infrastruktur mit umfangreichen Monitoring-Möglichkeiten, die das Verhalten der implementierten Komponenten beobachtbar machen.

Im Rahmen der Forschungsrichtung Smart Homes (auch Smart Environments) wird daran gearbeitet, die Idee unsichtbarer und allgegenwärtiger technologischer Unterstützung zu verwirklichen, wie sie bereits von Mark Weiser Anfang der 1990er Jahre unter dem Begriff „Ubiquitous Computing“ formuliert wurde (Abschnitt 2.1). Aus dieser Idee ging später das „Pervasive Computing“ hervor, dass mit dem Ansatz kurzfristiger Umsetzbarkeit insbesondere die Entwicklung im Bereich des „Mobile Computing“ mit einbezog (Abschnitt 2.2). Mit Beginn der 2000er Jahre wurde vermehrt der Begriff „Ambient Intelligence “ geprägt (Abschnitt 2.3). Forschungsprojekte in diesem Umfeld legen schon seit den frühen Anfängen ihren Fokus verstärkt auf die Lebensbereiche des Menschen – insbesondere auf seine Wohnsituation. [s. a. Aarts u. a., 2003]. Der Einbezug der Umgebung in das Systemdesign deutet dabei eine wesenliche Veränderung in der Bewertung der technologischen Entwicklung an [vgl. Augusto u. a., 2010]. Die ursprüngliche Ansicht der starren und sequentiellen Verarbeitung von Ereignissen aus der Umwelt wird abgelöst durch die Forderung dass sich beide Welten - Umgebung und Technologie - gegenseitig beeinflussen (Abschnitt 2.4).

Die beiden Labore des Living Place Hamburg bilden diese Entwicklungen mit ab. Es ist ihr Ziel, ein dynamisches Zusammenspiel zwischen Entwicklung, Integration, Usability-Untersuchungen und Evaluation der Ergebnisse zu liefern. Mit seiner Inbetriebnahme wird

das Living Place Hamburg auf einer Laborfläche von ca. 140 qm das lebensgroße Modell einer Wohnung der Zukunft repräsentieren (Abschnitt 3.2). Es soll untersucht werden, ob und wie neuartige Technologien und Interaktionsformen in den Alltag mit eingebunden werden können.

Der Laborbetrieb hat den Anspruch, eine möglichst realitätsnahe Integration des Systems zu schaffen. Hierdurch werden hohe Anforderungen an die zugrundeliegende Softwareinfrastruktur gestellt, die sowohl in der bestehenden Lösung, als auch in den Architekturen vergleichbarer Projekte nur unzureichend abgedeckt werden. Anhand von Anwendungsszenarien, die sich an studentischen Projekten im Rahmen des Living Place Hamburg orientieren, wurden die Anforderungen an eine Infrastruktur für ein Smart Home identifiziert. Aus den allgemeinen Anforderungen an verteilte Systeme, die außerdem gelten, haben sich in der Analyse die Aspekte der Nebenläufigkeit, Heterogenität und Verarbeitung komplexer Ereignisse herausgestellt. Zusätzlich ergeben sich insbesondere durch den Laborbetrieb des Systems hohe Anforderungen an die Erweiterbarkeit und Anpassungsfähigkeit des Systems, ohne dass diese den Betrieb der bestehenden Anwendungen beeinträchtigen (Kapitel 4).

Die in der Analyse gewonnenen Erkentnisse finden im Systemdesign ihre Abbildung in Form einer komponentenbasierten Architektur, die nach dem Prinzip des „Separation of Concerns“ in einzelnen Komponenten organisiert ist, die sich in ihrer jeweiligen Funktionalität möglichst wenig überschneiden (Abschnitt 5.4.4). Im Kern trägt die Architektur eine Laufzeitumgebung, in der die Komponenten integriert werden können. Jede Komponente definiert Schnittstellen, über die ihre Funktionalität den anderen Komponenten angeboten wird. Zur Koordination der Kommunnikation wurde ein an Tuple-Spaces-Architekturen angelehntes Modell gewählt (vgl. Abschnitt 3.3.5), da dieses sowohl die asynchrone Kommunikation fördert, als auch die Abbildung eines globalen Systemzustands vereinfacht (Abschnitt 5.4.1). Die Modellierung der Nebenläufigkeit des Systems nach dem Aktormodell ermöglicht zudem für den Entwickler ein besseres Verständnis der Nebenläufigkeit innerhalb des Systems, mit dem erwarteten Ergebnis, die infrastrukturell bedingte Komplexität der Anwendungsentwicklung rapide zu senken (s. Abschnitt 5.4.2 im Zusammenhang mit Abschnitt 4.1).

6.2 Mögliche erste Schritte einer iterativen Entwicklung

Der erste Iterationsschritt bei der Entwicklung des Prototypen soll der Überprüfung der eingeplanten Konzepte dienen. Es soll sich zeigen, ob die vorgeschlagene Architektur in ihrer Grundstruktur tragbar ist und ob sich aus ihr eine stabile Softwarelösung entwickeln lässt. Es werden in der folgenden Auflistung der Iterationsschritte nur die Entwicklungen der Middleware beschrieben. Die Schaffung der notwendigen Client-seitigen Anpassungen wird vorausgesetzt, aber nicht explizit erwähnt.

Erste Iteration. Ziel des Prototypen der ersten Iteration ist es, die Grundlagen des Architekturentwurfs tragfähig zu implementieren. Es sollte in diesem Entwicklungsschritt eine Laufzeitumgebung aufgebaut werden, mit deren Hilfe sich die Anwendung in möglichst disjunkte Teilkomponenten untergliedern lässt. Um die Grundfunktionalität der Anwendung testen zu können, sollten auch bereits erste funktionale Komponenten implementiert werden. Empfohlen werden hier die Message Queue und der Event Dispatcher. Die Integration der Message Queue kann im Kern eine bereits fertige Produktlösung enthalten. Es müssen dann lediglich die notwendigen Schnittstellen für die Interoperabilität mit der Laufzeitumgebung implementiert werden. Die Implementation des Dispatchers soll zunächst über minimale Funktionalität verfügen, um über diese Schnittstelle Nachrichten senden und empfangen zu können.

Zweite Iteration. In der zweiten Iteration soll die Entwicklung der Grundfunktionalitäten des Event Dispatchers vorangetrieben werden. Es sollen dazu die Schnittstellen implementiert werden, die es Arbeitermodulen ermöglichen, Events von Event Dispatcher abzurufen und auch wieder an diesen abzugeben. Um dieses Ziel erreichen zu können, ist es notwendig zunächst die benötigten Schnittstellen zu identifizieren. Am Ende dieser Iteration soll die erste Implementation eines einfachen Arbeitermoduls vorliegen, das geeignet ist, die Funktionsfähigkeit eines erweiterbaren Event Dispatchers zu prüfen.

Dritte und folgende Iterationen. Mit Beginn der dritten Iteration wird damit begonnen, die Arbeitermodule des Event Dispatchers zu implementieren. Es sollte in jeder Iteration, in der mit der Implementation einer neuen Komponente begonnen wird, auch das Ziel sein, eine erste lauffähige Version dieser Komponente zu schaffen. In weiteren Iterationsschritten kann dann der Funktionsumfang der jeweiligen Komponenten erweitert werden.

Die iterative Entwicklung des Systems ermöglicht es in kurzen Abständen signifikante Erfolge des Entwicklungsprozesses beobachtbar zu machen. Es wird somit möglich, Fehlentwicklungen frühzeitig entdecken zu können und entsprechend gegensteuernde Maßnahmen zu treffen. Durch den komponentenbasierten Aufbau der Architektur, kann in zusätzlichen, parallel verlaufenden Iterationen die Einführung weiterer Anwendungsteile, wie z. B. die Sicherheitsarchitektur und Zero-Config-Komponenten, vorangetrieben werden.

6.3 Fazit

In dieser Arbeit wurde eine nachrichtenbasierte Architektur für Smart Homes entwickelt und vorgestellt. Es wurde dabei der Anspruch verfolgt, einen Architekturentwurf vorzustellen, der als Grundlage für eine weitere Entwicklung einer nachrichtenbasierten Middleware für Smart

Homes dienen kann. Der Fokus des entwickelten Entwurfs liegt dabei auf der Schaffung einer dynamisch erweiterbaren Architektur. Um diesem Anspruch gerecht zu werden, basiert die Architektur auf einem komponentenbasierten Modell. Die jeweiligen Komponenten definieren öffentliche Schnittstellen, die von anderen Komponenten angesprochen werden können. Neben dieser *internen* Erweiterbarkeit des Funktionsumfangs der Middleware, wird die Funktionalität des Gesamtsystems hauptsächlich durch das Hinzufügen von Peers (Aktoren, Sensoren) in das System erreicht. Die Middlewarearchitektur ist geeignet, komplexe Strukturen und Regeln der Event-Verbreitung im Smart Home transparent aus der Perspektive der Peers zu gestalten. Das Event-Konzept erlaubt es dabei, dass ein Client ungerichtet Informationen über registrierte Aktivitäten mitteilt. Die Middleware übernimmt daraufhin die Auswahl der Empfänger der Nachrichten auf der Grundlage von komplexen Filterregeln und den ihr bekannten Eigenschaften der Peers (z. B. Position, verfügbare Ressourcen, Kontext). Die Beschreibung von Events kann dabei durch die in einer entsprechenden Beschreibungssprache formulierten Filterregeln vorgenommen werden. Die Middleware ist so für die zuverlässige Zustellung der Events verantwortlich.

Es wird empfohlen, den vorgestellten Architekturentwurf in einer praktischen Umsetzung weiter zu evaluieren.

> *„There is more information available at our fingertips during a walk in the woods than in any computer system, yet people find a walk among trees relaxing and computers frustrating. Machines that fit the human environment, instead of forcing humans to enter theirs, will make using a computer as refreshing as taking a walk in the woods.“* – Weiser [1991]

Literaturverzeichnis

[aal-deutschland.de 2010] AAL-DEUTSCHLAND.DE: *Ambient Assisted Living Deutschland – Living Labs*. Juni 2010. – URL http://www.aal-deutschland.de/informationen/living-labs. – Zugriffsdatum: 05.06.2010

[Aarts und de Ruyter 2009] AARTS, Emile ; RUYTER, Boris de: New research perspectives on Ambient Intelligence. In: *J. Ambient Intell. Smart Environ.* 1 (2009), Nr. 1, S. 5–14. – ISSN 1876-1364

[Aarts u. a. 2003] AARTS, Emile ; RUYTER, Boris de ; JANSEN, Leonie ; SLUIS, Richard van de ; VERBERKT, Mark ; DIEDERIKS, Elmo ; HOONHOUT, Jettie ; LASHINA, Tatiana ; LOENEN, Evert van ; HOVEN, Elise van den ; HOLLEMANS, Gerard ; HARTSON, Rex ; NOLDUS, Lucas ; TSCHELIG, Manfred ; GREEN, Josephine ; RUYTER, Boris de (Hrsg.): *365 Days Ambient Intelligence In Homelab*. Mai 2003. – URL http://www.research.philips.com/technologies/download/homelab_365.pdf. – Zugriffsdatum: 06.06.2010

[Adi und Etzion 2004] ADI, Asaf ; ETZION, Opher: Amit - the situation manager. In: *The VLDB Journal* 13 (2004), Nr. 2, S. 177–203. – ISSN 1066-8888

[Agarawala 2006] AGARAWALA, Anand: *Enriching the Desktop Metaphor with Physics, Piles and the Pen*, University of Toronto, Masters Thesis, 2006. – URL http://honeybrown.ca/Pubs/Thesis-BumpTop.pdf. – Zugriffsdatum: 28.02.2008

[Ahuja u. a. 1986] AHUJA, Sudhir ; CARRIERO, Nicholas ; GELERNTER, David: Linda and friends. In: *Computer* 19 (1986), Nr. 8, S. 26–34. – ISSN 0018-9162

[Alcañiz und Rey 2005] ALCAÑIZ, Mariano ; REY, Beatriz: *Emerging Communication*. Bd. 6: *New Technologies for Ambient Intelligence*. Siehe [Riva u. a., 2005]. – URL http://www.vepsy.com/communication/volume6.html. – Zugriffsdatum: 28.02.2008

[Alves u. a. 2007] ALVES, Alexandre ; ARKIN, Assaf ; ASKARY, Sid ; BARRETO, Charlton ; BLOCH, Ben ; CURBERA, Francisco ; FORD, Mark ; GOLAND, Yaron ; GUÍZAR, Alejandro ; KARTHA, Neelakantan ; LIU, Canyang K. ; KHALAF, Rania ; KÖNIG, Dieter ; MARIN, Mike ; MEHTA, Vinkesh ; THATTE, Satish ; RIJN, Danny van der ; YENDLURI, Prasad ; YIU, Alex: *Web Services Business Process Execution Language Version 2.0*. 11. April 2007.

– URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. – Zugriffsdatum: 16.05.2010

[Androutsellis-Theotokis und Spinellis 2004] ANDROUTSELLIS-THEOTOKIS, Stephanos ; SPINELLIS, Diomidis: A survey of peer-to-peer content distribution technologies. In: *ACM Computing Surveys* 36 (2004), Nr. 4, S. 335–371. – ISSN 0360-0300

[Arasu u. a. 2006] ARASU, Arvind ; BABU, Shivnath ; WIDOM, Jennifer: The CQL continuous query language: semantic foundations and query execution. In: *The VLDB Journal* 15 (2006), Nr. 2, S. 121–142. – ISSN 1066-8888

[Arbanowski u. a. 2004] ARBANOWSKI, Stefan ; BALLON, Pieter ; DAVID, Klaus ; DROEGEHORN, Olaf ; EERTINK, Henk ; KELLERER, Wolfgang ; KRANENBURG, Herma van ; RAATIKAINEN, Kimmo ; POPESCU-ZELETIN, Radu: I-centric communications: personalization, ambient awareness, and adaptability for future mobile services. In: *Communications Magazine, IEEE* 42 (2004), sept., Nr. 9, S. 63 – 69. – ISSN 0163-6804

[Armstrong 1997] ARMSTRONG, Joe: The development of Erlang. In: *ICFP '97: Proceedings of the second ACM SIGPLAN international conference on Functional programming*. New York, NY, USA : ACM, 1997, S. 196–203. – URL http://www.erlang.org/doc.html. – Zugriffsdatum: 04.05.2010. – ISBN 0-89791-918-1

[Armstrong u. a. 1996] ARMSTRONG, Joe ; VIRDING, Robert ; WIKSTRÖM, Claes ; WILLIAMS, Mike: *Concurrent Programming in ERLANG*. Second Edition. Englewood Cliffs, New Jersey 07632 : Prentice Hall, 1996

[Augusto u. a. 2010] AUGUSTO, Juan C. ; NAKASHIMA, Hideyuki ; AGHAJAN, Hamid: Ambient Intelligence and Smart Environments: A State of the Art. In: [Shen u. a., 2010], S. 3–31. – ISBN 0387097503, 9780387097503

[Bellifemine u. a. 1999] BELLIFEMINE, Fabio ; POGGI, Agostino ; RIMASSA, Giovanni: JADE - A FIPA-compliant agent framework / Telecom Italia. Italia, April 1999. – internal technical report. – URL http://jade.tilab.com/papers-2001.htm. – Zugriffsdatum: 14.06.2010

[Bernin 2007] BERNIN, Arne: *Body Monitoring*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2007/vortraege.html. – Zugriffsdatum: 14.06.2010

[BITKOM e.V. 2010] BITKOM E.V.: *Ein Haus, eine Fernbedienung – für alles (Pressemitteilung)*. 17. Mai 2010. – URL http://www.bitkom.org/63886_63882.aspx. – Zugriffsdatum: 24.06.2010

[Blumendorf 2009] BLUMENDORF, Marco: *Multimodal Interaction in Smart Environments A Model-based Runtime System for Ubiquitous User Interfaces*, Technische Universität Berlin, Dissertation, 2009. – URL http://opus.kobv.de/tuberlin/volltexte/2009/2325/pdf/blumendorf_marco.pdf. – Zugriffsdatum: 08.06.2010

[Booth und Liu 2007] BOOTH, David ; LIU, Canyang K.: *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*. 26. Juni 2007. – URL http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/. – Zugriffsdatum: 15.05.2010

[Bray u. a. 2006] BRAY, Tim ; PAOLI, Jean ; SPERBERG-MCQUEEN, C. M. ; MALER, Eve ; YERGEAU, François ; COWAN, John: *Extensible Markup Language (XML) 1.1 (Second Edition)*. 29. September 2006. – URL http://www.w3.org/TR/2006/REC-xml11-20060816. – Zugriffsdatum: 15.05.2010

[Brewer u. a. 1998] BREWER, Eric A. ; KATZ, Randy H. ; CHAWATHE, Yatin ; GRIBBLE, Steven D. ; HODES, Todd ; NGUYEN, Giao ; STEMM, Marc ; HENDERSON, Tom ; AMIR, Elan ; BALAKRISHNAN, Hari ; FOX, Armando ; PADMANABHAN, Venkata N. ; SESHAN, Srinivasan: A network architecture for heterogeneous mobile computing. In: *Personal Communications, IEEE* 5 (1998), oct, Nr. 5, S. 8 –24. – ISSN 1070-9916

[Brookes u. a. 1984] BROOKES, S. D. ; HOARE, C. A. R. ; ROSCOE, A. W.: A Theory of Communicating Sequential Processes. In: *J. ACM* 31 (1984), Nr. 3, S. 560–599. – ISSN 0004-5411

[Buford 2008] BUFORD, John F.: Management of peer-to-peer overlays. In: *International Journal of Internet Protocol Technology* 3 (2008), Nr. 1, S. 2–12. – ISSN 1743-8209

[Buford und Yu 2010] BUFORD, John F. ; YU, Heather: *Peer-to-Peer Networking and Applications: Synopsis and Research Directions*. Kap. Part 1, S. 3 – 45. Siehe [Shen u. a., 2010]. – ISBN 0387097503, 9780387097503

[Buttyán und Hubaux 2003] BUTTYÁN, Levente ; HUBAUX, Jean-Pierre: Stimulating cooperation in self-organizing mobile ad hoc networks. In: *Mob. Netw. Appl.* 8 (2003), Nr. 5, S. 579–592. – ISSN 1383-469X

[Chakravarthy und Mishra 1994] CHAKRAVARTHY, S. ; MISHRA, D.: Snoop: An expressive event specification language for active databases. In: *Data & Knowledge Engineering* 14 (1994), Nr. 1, S. 1 – 26. – URL http://www.sciencedirect.com/science/article/B6TYX-47X2F5Y-6/2/870e6e72c683e480b4f4d000dc9fb979. – ISSN 0169-023X

[Chakravarthy und Adaikkalavan 2008] CHAKRAVARTHY, Sharma ; ADAIKKALAVAN, Raman: Events and streams: harnessing and unleashing their synergy! In: *DEBS '08: Proceedings of the second international conference on Distributed event-based systems*. New York, NY, USA : ACM, 2008, S. 1–12. – ISBN 978-1-60558-090-6

[Chapanis 1988] CHAPANIS, Alphonse: *Interactive human communication*. Kap. 6, S. 127–141. Siehe [Greif, 1988]. – ISBN 0-934613-57-5

[Chen und Kotz 2000] CHEN, Guanling ; KOTZ, David: A Survey of Context-Aware Mobile Computing Research / Dartmouth College. Hanover, NH, USA, 2000. – Forschungsbericht. – URL http://www.cs.dartmouth.edu/reports/TR2000-381.pdf. – Zugriffsdatum: 19.05.2010

[Cheung 1999] CHEUNG, Susan ; SUN MICROSYSTEMS, INC. (Hrsg.): *Java Transaction Service (JTS)*. Dezember 1999. – URL http://java.sun.com/javaee/technologies/jts/. – Zugriffsdatum: 15.05.2010

[Cheung und Matena 2002] CHEUNG, Susan ; MATENA, Vlada ; SUN MICROSYSTEMS, INC. (Hrsg.): *Java Transaction API (JTA)*. November 2002. – URL http://java.sun.com/javaee/technologies/jta/. – Zugriffsdatum: 15.05.2010

[Chinnici u. a. 2007a] CHINNICI, Roberto ; HAAS, Hugo ; LEWIS, Amelia A. ; MOREAU, Jean-Jacques ; ORCHARD, David ; WEERAWARANA, Sanjiva: *Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts*. 26. Juni 2007. – URL http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626. – Zugriffsdatum: 15.05.2010

[Chinnici u. a. 2007b] CHINNICI, Roberto ; JACQUES MOREAU dJean ; RYMAN, Arthur ; WEERAWARANA, Sanjiva: *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. 26. Juni 2007. – URL http://www.w3.org/TR/2007/REC-wsdl20-20070626. – Zugriffsdatum: 15.05.2010

[Chinnici und Shannon 2009] CHINNICI, Roberto ; SHANNON, Bill ; SUN MICROSYSTEMS, INC. (Hrsg.): *Java™ Platform, Enterprise Edition (Java EE) Specification, v6*. Dezember 2009. – URL http://jcp.org/aboutJava/communityprocess/final/jsr316/index.html. – Zugriffsdatum: 15.05.2010

[Connected Living e.V. 2010] CONNECTED LIVING E.V.: *iZ Connected – Innovationszentrum „Vernetztes Leben – Connected Living“*. 2010. – URL http://www.connected-living.org. – Zugriffsdatum: 08.06.2010

[Coulouris u. a. 2005] COULOURIS ; DOLLIMORE, Jean ; KINDBERG, Tim: *Distributed Systems: Concepts and Design (4th Edition) (International Computer Science)*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2005. – URL http://www.cdk4.net/. – Zugriffsdatum: 28.02.2008. – ISBN 0321263545

[Davies u. a. 1997] DAVIES, N. ; WADE, S. P. ; FRIDAY, A. ; BLAIR, G. S.: Limbo: a tuple space based platform for adaptive mobile applications. In: *ICODP/ICDP '97: Proceedings of the IFIP/IEEE international conference on Open distributed processing and distributed*

platforms. London, UK, UK : Chapman & Hall, Ltd., 1997, S. 291–302. – ISBN 0-412-81230-4

[Dean und Ghemawat 2004] DEAN, Jeffrey ; GHEMAWAT, Sanjay: MapReduce: Simplified Data Processing on Large Clusters. In: *OSDI'04: Sixth Symposium on Operating System Design and Implementation*. San Francisco, CA, Dezember 2004. – URL http://labs.google.com/papers/mapreduce.html. – Zugriffsdatum: 14.06.2010

[DeMichiel u. a. 2009] DEMICHIEL, Linda ; VROOM, Jeff ; BOUSCHEN, Michael ; SAMSON, Eric ; BRAKKEE, Erik ; SUTTER, Kevin ; PODDAR, Pinaki ; BENOIT, Florent ; KEITH, Michael ; YORKE, Gordon ; LINSKEY, Patrick ; ANUPALLI, Deepak ; KING, Gavin ; BERNARD, Emmanuel ; SCHWEIGKOFFER, Rainer ; GOERLER, Adrian ; ADAMS, Matthew ; SAKS, Kenneth ; IRELAND, Evan ; KIM, Wonseok ; CHUNG, Eugene ; BIEN, Adam ; GONCALVES, Antonio ; MAKI, Chris ; SUN MICROSYSTEMS, INC. (Hrsg.): *JSR 317: Java^{TM} Persistence API, Version 2.0*. Dezember 2009. – URL http://jcp.org/aboutJava/communityprocess/final/jsr317/index.html. – Zugriffsdatum: 15.05.2010

[Dey 2001] DEY, Anind K.: Understanding and Using Context. In: *Personal Ubiquitous Comput.* 5 (2001), Nr. 1, S. 4–7. – URL http://dx.doi.org/10.1007/s007790170019. – Zugriffsdatum: 28.02.2008. – ISSN 1617-4909

[Dreyer 2007] DREYER, Markus: *Architekturen für ein Collaborative Worksplace*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2007/vortraege.html. – Zugriffsdatum: 14.06.2010

[Dreyer 2008] DREYER, Markus: *Your Home in Your Hand*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08/vortraege.html. – Zugriffsdatum: 14.06.2010

[Dreyer 2009] DREYER, Markus: *Ein nutzeradaptierendes agentenbasiertes TV System als Teil eines intelligenten Hauses*, Hochschule für Angewandte Wissenschaften Hamburg, Masters Thesis, 2009. – URL http://users.informatik.haw-hamburg.de/~ubicomp/papers.html. – Zugriffsdatum: 14.06.2010

[Eckert 2007] ECKERT, Claudia: Ambient Intelligence: Neue Herausforderungen für die IT-Sicherheit. In: *TU-Darmstadt, thema Forschung* (2007), Januar, S. 22–27. – ISSN 1434-7768

[Eckert 2009] ECKERT, Claudia: *IT-Sicherheit : Konzepte - Verfahren - Protokolle*. 6., überarb. und erw. Aufl. München : Oldenbourg, 2009. – ISBN 978-3-486-58999-3

[Eckert und Bry 2009] ECKERT, Michael ; BRY, François: Complex Event Processing (CEP). In: *Informatik-Spektrum* 32 (2009), April, Nr. 2, S. 163–167

[Ellenberg 2010] ELLENBERG, Jens: *Ein Wecker in einem ubicom Haus*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2010. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-aw1/vortraege.html. – Zugriffsdatum: 10.03.2010

[Emmerich u. a. 2008] EMMERICH, Wolfgang ; AOYAMA, Mikio ; SVENTEK, Joe: The impact of research on the development of middleware technology. In: *ACM Trans. Softw. Eng. Methodol.* 17 (2008), Nr. 4, S. 1–48. – ISSN 1049-331X

[Erman u. a. 1980] ERMAN, Lee D. ; HAYES-ROTH, Frederick ; LESSER, Victor R. ; REDDY, D. R.: The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. In: *ACM Comput. Surv.* 12 (1980), Nr. 2, S. 213–253. – ISSN 0360-0300

[Fayyad u. a. 1996] FAYYAD, U. ; PIATETSKY-SHAPIRO, G. ; SMYTH, P.: From data mining to knowledge discovery in databases. In: *AI Magazine* 17 (1996), S. 37–54. – URL http://www.aaai.org/aitopics/assets/PDF/AIMag17-03-2-article.pdf. – Zugriffsdatum: 10.06.2010

[Forgy 1982] FORGY, Charles L.: Rete: A fast algorithm for the many pattern/many object pattern match problem. In: *Artificial Intelligence* 19 (1982), Nr. 1, S. 17 – 37. – URL https://cit-server.cit.tu-berlin.de/~battre/db.rdf.forgy.90.rete.pdf. – Zugriffsdatum: 12.05.2010. – ISSN 0004-3702

[Fowler 2002] FOWLER, Martin: *Patterns of Enterprise Application Architecture*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2002. – ISBN 0321127420

[Fraunhofer IGD 2003] FRAUNHOFER IGD: *Software-Infrastructures for Ambient Intelligence*. 12 2003. – URL http://www.igd.fraunhofer.de/igd-a1/projects/ambientintelligence/. – Zugriffsdatum: 28.02.2008

[Freeman u. a. 1999] FREEMAN, Eric ; ARNOLD, Ken ; HUPFER, Susanne: *JavaSpaces Principles, Patterns, and Practice*. Essex, UK, UK : Addison-Wesley Longman Ltd., 1999. – ISBN 0201309556

[Gärner 2007] GÄRNER, Sven: *Verteilte Dateisysteme und mobile Clients*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2007/vortraege.html. – Zugriffsdatum: 14.06.2010

[Gärner 2008a] GÄRNER, Sven: *Coda und mobile Clients*, Hochschule für Angewandte Wissenschaften Hamburg, Projektbericht, 2008. – URL http://users.

informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-proj/berichte.html. – Zugriffsdatum: 14.06.2010

[Gärner 2008b] GÄRNER, Sven: *Synchronisation und verteilte Dateisysteme*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-aw/vortraege.html. – Zugriffsdatum: 14.06.2010

[Gehn 2007] GEHN, Stefan: *Eingabeperformance von Multitouch-Displays am Beispiel von Spielen*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2007/vortraege.html. – Zugriffsdatum: 14.06.2010

[Gehn 2008a] GEHN, Stefan: *Evaluation einer infrarotbasierten Multitouch-hardware*, Hochschule für Angewandte Wissenschaften Hamburg, Projektbericht, 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-proj/berichte.html. – Zugriffsdatum: 14.06.2010

[Gehn 2008b] GEHN, Stefan: *Techniken für Interaktion mittels Bewegungen und Gesten*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-aw/vortraege.html. – Zugriffsdatum: 14.06.2010

[Geihs u. a. 2006] GEIHS, Kurt ; KHAN, Mohammad U. ; REICHLE, Roland ; SOLBERG, Arnor ; HALLSTEINSEN, Svein ; MERRAL, Simon: Modeling of component-based adaptive distributed applications. In: *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*. New York, NY, USA : ACM, 2006, S. 718–722. – ISBN 1-59593-108-2

[Ghosh 2006] GHOSH, Sukumar: *Distributed systems: an algorithmic approach*. Boca Raton, FL, USA : Chapman & Hall/CRC, 2006 (Computer and information science series 13). – ISBN 978-1-58488-564-1

[Gill und Balkishan 2008] GILL, Nasib S. ; BALKISHAN: Dependency and interaction oriented complexity metrics of component-based systems. In: *SIGSOFT Softw. Eng. Notes* 33 (2008), Nr. 2, S. 1–5. – ISSN 0163-5948

[Gokhale u. a. 2002] GOKHALE, Aniruddha ; SCHMIDT, Douglas C. ; NATARAJAN, Balachandran ; WANG, Nanbor: Applying model-integrated computing to component middleware and enterprise applications. In: *Communications of the ACM* 45 (2002), Nr. 10, S. 65–70. – ISSN 0001-0782

[Gregor 2007] GREGOR, Sebastian: *Indoor Positionierung - Anforderungen und technische Möglichkeiten*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2007/vortraege.html. – Zugriffsdatum: 14.06.2010

[Gregor u. a. 2009] GREGOR, Sebastian ; RAHIMI, Mohammad A. ; VOGT, Matthias ; SCHULZ, Thomas ; LUCK, Kai von: Tangible Computing revisited: Anfassbare Computer in Intelligenten Umgebungen. In: *4. Kongress Multimediatechnik* IFM Institut für Multimediatechnik gGmbH (Veranst.), URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/papers/MMWismar2009.pdf. – Zugriffsdatum: 02.12.2009, Oktober 2009

[Greif 1988] GREIF, Irene (Hrsg.): *Computer-supported cooperative work: a book of readings*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1988. – ISBN 0-934613-57-5

[Gudgin u. a. 2007a] GUDGIN, Martin ; HADLEY, Marc ; MENDELSOHN, Noah ; MOREAU, Jean-Jacques ; NIELSEN, Henrik F. ; KARMARKAR, Anish ; LAFON, Yves: *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. 27. April 2007. – URL http://www.w3.org/TR/2007/REC-soap12-part1-20070427/. – Zugriffsdatum: 16.05.2010

[Gudgin u. a. 2007b] GUDGIN, Martin ; HADLEY, Marc ; MENDELSOHN, Noah ; MOREAU, Jean-Jacques ; NIELSEN, Henrik F. ; KARMARKAR, Anish ; LAFON, Yves: *SOAP Version 1.2 Part 2: Adjuncts (Second Edition)*. 27. April 2007. – URL http://www.w3.org/TR/2007/REC-soap12-part2-20070427/. – Zugriffsdatum: 16.05.2010

[Haller und Odersky 2007] HALLER, Philipp ; ODERSKY, Martin: Actors That Unify Threads and Events. In: *Coordination Models and Languages* (2007), S. 171–190. – URL http://dx.doi.org/10.1007/978-3-540-72794-1_10

[Hamann 2007] HAMANN, Lennard: *Grundlagen des Eyetrackings*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2007/vortraege.html. – Zugriffsdatum: 14.06.2010

[Hamann 2008a] HAMANN, Lennard: *Eyetracker als Eingabemedien in der multimodalen Interaktion*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08/vortraege.html. – Zugriffsdatum: 14.06.2010

[Hamann 2008b] HAMANN, Lennard: *Multimodale Interaktion*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-aw/vortraege.html. – Zugriffsdatum: 14.06.2010

[Hamann 2008c] HAMANN, Lennard: *Validierung von Konzepten für die multimodale Eingabe*, Hochschule für Angewandte Wissenschaften Hamburg, Projektbericht,

2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-proj/berichte.html. – Zugriffsdatum: 14.06.2010

[Hansmann u. a. 2003] HANSMANN, Uwe ; MERK, Lothar ; NICKLOUS, Martin S. ; STOBER, Thomas: *Pervasive Computing : The Mobile World*. Springer-Verlag, August 2003. – ISBN 3540002189

[Hapner u. a. 2002] HAPNER, Mark ; BURRIDGE, Rich ; SHARMA, Rahul ; FIALLI, Joseph ; STOUT, Kate ; SUN MICROSYSTEMS, INC. (Hrsg.): *Java™ Message Service Specification*. 12. April 2002. – URL http://java.sun.com/products/jms/docs.html. – Zugriffsdatum: 16.05.2010

[Hardenack 2010] HARDENACK, Frank: *Das intelligente Bett - Interpretation von Schlafphasen als Beispiel für Bodymonitoring im Living Place Hamburg*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2010. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-aw1/vortraege.html. – Zugriffsdatum: 10.03.2010

[Hellenschmidt 2005a] HELLENSCHMIDT, Michael: Distributed implementation of a self-organizing appliance middleware. In: *sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence*. New York, NY, USA : ACM, 2005, S. 201–206. – URL http://doi.acm.org/10.1145/1107548.1107600. – Zugriffsdatum: 28.02.2008. – ISBN 1-59593-304-2

[Hellenschmidt 2005b] HELLENSCHMIDT, Michael: DynAMITE: Self-organizing middleware technologies for Ambient Intelligence. In: *Computer Graphik Topics* (2005), Nr. Vol. 17. – URL http://www.inigraphics.net/press/topics/2005/issue1/1_05a02.pdf. – Zugriffsdatum: 28.02.2008

[Hellenschmidt und Kirste 2004] HELLENSCHMIDT, Michael ; KIRSTE, Thomas: SodaPop: a software infrastructure supporting self-organization in intelligent environments. In: *Industrial Informatics, 2004. INDIN '04. 2004 2nd IEEE International Conference on*, 26-26 2004, S. 479 –486

[Hewitt und Baker 1977] HEWITT, Carl ; BAKER, Henry: *Actors and Continuous Functionals*. 07 1977. – URL http://hdl.handle.net/1721.1/6687. – Zugriffsdatum: 14.06.2010

[Hewitt u. a. 1973] HEWITT, Carl ; BISHOP, Peter ; STEIGER, Richard: A universal modular ACTOR formalism for artificial intelligence. In: *IJCAI'73: Proceedings of the 3rd international joint conference on Artificial intelligence*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1973, S. 235–245

[Hinrichs u. a. 2005] HINRICHS, Uta ; CARPENDALE, Sheelagh ; SCOTT, Stacey D. ; PATTISON, Eric: Interface Currents: Supporting Fluent Collaboration on Tabletop Displays. In: *5th Symposium on Smart Graphics*. Berlin Heidelberg : Springer-Verlag, August 2005, S. 185–197. – URL http://www.utahinrichs.de/work/smartGraphics/sg_paper_final.pdf. – Zugriffsdatum: 18.02.2007

[Hoare 1978] HOARE, C. A. R.: Communicating sequential processes. In: *Commun. ACM* 21 (1978), Nr. 8, S. 666–677. – ISSN 0001-0782

[Hollatz 2007] HOLLATZ, Dennis: *Konzepte für interaktive Räume*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2007/vortraege.html. – Zugriffsdatum: 28.02.2008

[Hollatz 2008a] HOLLATZ, Dennis: *Managing Information - Infrastructures for Ambient Intelligence*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-aw/vortraege.html. – Zugriffsdatum: 28.02.2008

[Hollatz 2008b] HOLLATZ, Dennis: *Managing Information - Personal Information Environments auf der Basis von iROS*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08/vortraege.html. – Zugriffsdatum: 28.02.2008

[Hollatz 2008c] HOLLATZ, Dennis: *smart:shelf – Projektbericht*, Hochschule für Angewandte Wissenschaften Hamburg, Projektbericht, 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projects.html. – Zugriffsdatum: 28.02.2008

[inHaus-Zentrum 2010] INHAUS-ZENTRUM: *inHaus – Innovagtionszentrum der Fraunhofer Gesellschaft*. Juni 2010. – URL http://www.inhaus-zentrum.de/. – Zugriffsdatum: 07.06.2010

[Isenberg u. a. 2009] ISENBERG, Tobias ; HINRICHS, Uta ; CARPENDALE, Sheelagh: Studying Direct-Touch Interaction for 2D Flow Visualization. In: *Proceedings of the Workshop on Collaborative Visualization on Interactive Surfaces (CoVIS 2009, October 11, 2009, Atlantic City, USA)*, 2009. – To appear in the »Technical Reports« series of the Department of Media Informatics of the Ludwig-Maximilians-University of Munich, Germany

[ISTAG 2003] ISTAG: Ambient Intelligence: from vision to reality / Information Society Technologies Advisory Group (ISTAG). URL http://cordis.europa.eu/ist/istag-reports.htm. – Zugriffsdatum: 28.02.2008, September 2003. – Forschungsbericht

[Jennings und Wooldridge 1996] JENNINGS, Nick ; WOOLDRIDGE, Michael: Software agents. In: *IEE Review* 42 (1996), 18, Nr. 1, S. 17 –20. – ISSN 0953-5683

[Jensen u. a. 2009] JENSEN, Björn ; KRUSE, Ralf ; WENDHOLT, Birgit: Application of indoor navigation technologies under practical conditions. In: *Positioning, Navigation and Communication, 2009. WPNC 2009. 6th Workshop on*, 19-19 2009, S. 267 –273

[Johanson 2002] JOHANSON, Bradley E.: *Application Coordination Infrastructure for Ubiquitous Computing Rooms*, Department Of Electrical Engineering Of Stanford University, Dissertation, Dezember 2002. – URL http://graphics.stanford.edu/~bjohanso/dissertation/. – Zugriffsdatum: 28.02.2008

[Johanson und Fox 2002] JOHANSON, Bradley E. ; FOX, Armando: The Event Heap: a coordination infrastructure for interactive workspaces. In: *Mobile Computing Systems and Applications, 2002. Proceedings Fourth IEEE Workshop on* (2002), S. 83–93. – URL http://ieeexplore.ieee.org/search/wrapper.jsp?arnumber=1017488. – Zugriffsdatum: 28.02.2008

[Johanson und Fox 2004] JOHANSON, Bradley E. ; FOX, Armando: Extending tuplespaces for coordination in interactive workspaces. In: *J. Syst. Softw.* 69 (2004), Nr. 3, S. 243–266. – URL http://dx.doi.org/10.1016/S0164-1212(03)00054-2. – Zugriffsdatum: 28.02.2008. – ISSN 0164-1212

[Kahlbrandt 2005] KAHLBRANDT, Bernd: *Software-Engineering mit der Unified Modeling Language*. Version 3.3.2. Springer, 2005. – Herausgeber und ISBN beziehen sich auf die 2001 im Springer-Verlag erschienene 2. Auflage des Buches; dem Autor liegt eine elektronische Fassung des Werks vom 13. Juni 2005 vor. – ISBN 3-540-41600-5

[Kakousis u. a. 2010] KAKOUSIS, Constantinos ; PASPALLIS, Nearchos ; PAPADOPOULOS, George A. ; RUIZ, Pedro A.: Testing self-adaptive applications with simulation of context events. In: *3rd international workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS)*. Amsterdam, Netherlands : EASST, 2010. – URL http://www.cs.ucy.ac.cy/~paspalli/publications.html. – Zugriffsdatum: 17.06.2010

[Kehr u. a. 1999] KEHR, Roger ; ZEIDLER, Andreas ; VOGT, Harald: Towards a Generic Proxy Execution Service for Small Devices. In: *Proceedings of the Workshop on Future Services for Networked Devices*. Heidelberg, Germany, Oktober 1999

[Kindberg und Fox 2002] KINDBERG, Tim ; FOX, Armando: System software for ubiquitous computing. In: *Pervasive Computing, IEEE* 1 (2002), Januar–März, Nr. 1, S. 70 – 81. – ISSN 1536-1268

[Kleinrock 2003] KLEINROCK, Leonard: An Internet vision: the invisible global infrastructure. In: *Ad Hoc Networks* 1 (2003), Nr. 1, S. 3 – 11. – ISSN 1570-8705

[Knoblauch und Heath 1999] KNOBLAUCH, Hubert ; HEATH, Christian: Technologie, Interaktion und Organisation: die Workplace Studies. In: *Schweizerische Zeitschrift für Soziologie* 25 (1999), Nr. 2, S. 163–181. – URL http://nbn-resolving.de/urn:nbn:de:0168-ssoar-8392. – Zugriffsdatum: 01.06.2010. – ISSN 0379-3664

[Krasemann und Spindel 2004] KRASEMANN, Hartmut ; SPINDEL, Ulrich: *Softwarearchitektur für einen Containerterminal – Vom Informationssystem in J2EE bis zur Gerätesteuerung in Java*. 2004. – URL http://users.informatik.haw-hamburg.de/~sarstedt/AKOT1/stprot48.html. – Zugriffsdatum: 16.05.2010

[Krieg-Brückner u. a. 2009] KRIEG-BRÜCKNER, Bernd ; GERSDORF, Bernd ; DÖHLE, Mathias ; SCHILL, Kerstin: Technik für Senioren in spe im Bremen Ambient Assisted Living Lab. In: *Ambient Assisted Living, 2. Deutscher AAL-Kongress 2009. Deutscher AAL-Kongress*. Berlin, Germany : VDE-Verlag, January 2009

[Lee 2006] LEE, E.A.: The problem with threads. In: *Computer* 39 (2006), may, Nr. 5, S. 33 – 42. – ISSN 0018-9162

[Lehman u. a. 1995] LEHMAN, Tobin J. ; MCLAUGHRY, Steve ; WYCKOFF, Peter: *T Spaces: The Next Wave*. 1995. – URL http://www.almaden.ibm.com/cs/TSpaces/html/TSRole.html. – Zugriffsdatum: 14.06.2010

[Luckham 2005] LUCKHAM, David C.: *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. 3. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2005. – ISBN 0-201-72789-7

[Maatz 2010] MAATZ, Björn: Studie von Google und Otto: Handelskonzerne setzen auf Smartphone-Boom. In: *Financial Times Deutschland (Online)* (2010), Juni. – URL http://www.ftd.de/it-medien/medien-internet/:studie-von-google-und-otto-handelskonzerne-setzen-auf-smartphone-boom/50131532.html. – Zugriffsdatum: 21.06.2010

[Mählmann 2007] MÄHLMANN, Lars: *A „useful“ profile*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2007/vortraege.html. – Zugriffsdatum: 14.06.2010

[Mählmann 2008a] MÄHLMANN, Lars: *Deliever who I mean*, Hochschule für Angewandte Wissenschaften Hamburg, Projektbericht, 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-proj/berichte.html. – Zugriffsdatum: 14.06.2010

[Mählmann 2008b] MÄHLMANN, Lars: *Deliver who I mean*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2008. – URL http://users.

informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-aw/vortraege.html. – Zugriffsdatum: 14.06.2010

[Mattern 2001] MATTERN, Friedemann: *Pervasive Computing / Ubiquitous Computing*. 2001. – URL http://www.vs.inf.ethz.ch/publ/papers/UbiPvCSchlagwort.pdf. – Zugriffsdatum: 19.05.2010

[Maybury und Wahlster 1998] MAYBURY, Mark T. (Hrsg.) ; WAHLSTER, Wolfgang (Hrsg.): *Readings in intelligent user interfaces*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1998. – ISBN 1-55860-444-8

[Meißner 2007] MEISSNER, Stefan: *Barrierefreiheit mithilfe von Ambient Intelligence*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2007/vortraege.html. – Zugriffsdatum: 28.02.2008

[Meißner 2008a] MEISSNER, Stefan: *Ambient Assisted Living - Accessibility*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-aw/vortraege.html. – Zugriffsdatum: 14.06.2010

[Meißner 2008b] MEISSNER, Stefan: *Ambient Assisted Living - Accessibility: Ambient awareness*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08/vortraege.html. – Zugriffsdatum: 28.02.2008

[Meißner 2008c] MEISSNER, Stefan: *Projektbericht – smart:shelf*, Hochschule für Angewandte Wissenschaften Hamburg, Projektbericht, 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-proj/berichte.html. – Zugriffsdatum: 18.01.2010

[Mikalsen u. a. 2006a] MIKALSEN, Marius ; FLOCH, Jacqueline ; STAV, Erlend ; PASPALLIS, Nearchos ; PAPADOPOULOS, George A. ; RUIZ, Pedro A.: Putting Context in Context: The Role and Design of Context Management in a Mobility and Adaptation Enabling Middleware. In: *Mobile Data Management, 2006. MDM 2006. 7th International Conference on*, Mai 2006, S. 76 – 76. – ISSN 1551-6245

[Mikalsen u. a. 2006b] MIKALSEN, Marius ; PASPALLIS, Nearchos ; FLOCH, Jacqueline ; STAV, Erlend ; PAPADOPOULOS, George A. ; CHIMARIS, Akis: Distributed context management in a mobility and adaptation enabling middleware (MADAM). In: *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*. New York, NY, USA : ACM, April 2006, S. 733–734. – ISBN 1-59593-108-2

[Minsky 1988] MINSKY, Marvin: *The Society of Mind*. New York, NY, USA : Simon & Schuster, March 1988. – 339 S. – ISBN 0671657135

[MIT Project Oxygen 2004] MIT PROJECT OXYGEN: *MIT Project Oxygen – Pervasive, Human-Centered Computing*. 06 2004. – URL http://oxygen.csail.mit.edu. – Zugriffsdatum: 28.02.2008

[Mitra und Lafon 2007] MITRA, Nilo ; LAFON, Yves: *SOAP Version 1.2 Part 0: Primer (Second Edition)*. 27. April 2007. – URL http://www.w3.org/TR/2007/REC-soap12-part0-20070427/. – Zugriffsdatum: 16.05.2010

[Ndumu und Nwana 1997] NDUMU, D.T. ; NWANA, H.S.: Research and development challenges for agent-based systems. In: *Software Engineering. IEE Proceedings- [see also Software, IEE Proceedings]* 144 (1997), feb, Nr. 1, S. 2 –10. – ISSN 1364-5080

[Neumann 2006] NEUMANN, Carola: *Effizienzsteigerung von Diskussionsprozessen in einem neu gestalteten Konferenzraum*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2006. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/diplom/neumann.pdf. – Zugriffsdatum: 01.06.2010

[Norbisrath 2007] NORBISRATH, Ulrich: *Konfigurierung von eHome-Systemen - Configuring eHome Systems*. Aachen, RWTH Aachen, Dissertation, 09. August 2007. – URL http://phd.ulno.net. – Zugriffsdatum: 14.06.2010

[OFFIS e.V. 2010] OFFIS E.V.: *OFFIS ideAAL Projekt*. 2010. – URL http://www.ideaal.de/. – Zugriffsdatum: 08.06.2010

[OMG 1998] OMG ; OBJECT MANAGEMENT GROUP (Hrsg.): *CORBAservices: Common Object Services Specification*. Dezember 1998. – URL http://www.omg.org/cgi-bin/doc?formal/98-12-09.pdf. – Zugriffsdatum: 15.05.2010

[Paspallis und Papadopoulos 2006] PASPALLIS, N. ; PAPADOPOULOS, G.A.: An Approach for Developing Adaptive, Mobile Applications with Separation of Concerns. In: *Computer Software and Applications Conference, 2006. COMPSAC '06. 30th Annual International* Bd. 1, 17-21 2006, S. 299 –306. – ISSN 0730-3157

[Paspallis 2009] PASPALLIS, Nearchos: *Middleware-Based Development Of Context-Aware Applications With Reusable Components*, University of Cyprus, Dissertation, September 2009. – URL http://www.cs.ucy.ac.cy/~paspalli/phd/phd.html. – Zugriffsdatum: 18.05.2010

[Paton und Díaz 1999] PATON, Norman W. ; DÍAZ, Oscar: Active database systems. In: *ACM Comput. Surv.* 31 (1999), Nr. 1, S. 63–103. – ISSN 0360-0300

[Paton u. a. 1998] PATON, Norman W. (Hrsg.) ; SCHNEIDER, F. (Hrsg.) ; GRIES, D. (Hrsg.): *Active Rules in Database Systems*. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 1998. – ISBN 0387985298

[Petri 1962] PETRI, Carl A.: *Kommunikation mit Automaten.* Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962. – URL http://www.informatik.uni-hamburg.de/TGI/publikationen/public/biblio_petri.html. – Zugriffsdatum: 14.06.2010

[Philips Research 2010] PHILIPS RESEARCH: *Ambient Intelligence*. 2010. – URL http://www.research.philips.com/technologies/projects/ami/. – Zugriffsdatum: 16.05.2010

[Pierce und Nichols 2007] PIERCE, Jeff ; NICHOLS, Jeff: *Personal Information Environments*. 2007. – URL http://www.almaden.ibm.com/cs/projects/pie/. – Zugriffsdatum: 28.02.2008

[Pokahr u. a. 2003] POKAHR, Alexander ; BRAUBACH, Lars ; LAMERSDORF, Winfried: Jadex: Implementing a BDI-Infrastructure for JADE Agents. In: *EXP - In Search of Innovation (Special Issue on JADE)* 3 (2003), September, S. 76–85. – URL http://vsis-www.informatik.uni-hamburg.de/projects/jadex/pubs.php. – Zugriffsdatum: 28.02.2008

[Rahimi und Vogt 2010] RAHIMI, Mohammadali ; VOGT, Matthias: *Aufbau des Living Place Hamburg*, Hochschule für Angewandte Wissenschaften Hamburg, Projektbericht, 2010. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-proj/berichte.html. – Zugriffsdatum: 15.06.2010

[Raverdy u. a. 2006] RAVERDY, Pierre-Guillaume ; RIVA, Oriana ; LA CHAPELLE, Agnès de ; CHIBOUT, Rafik ; ISSARNY, Valérie: Efficient Context-aware Service Discovery in Multi-Protocol Pervasive Environments. In: *Mobile Data Management, 2006. MDM 2006. 7th International Conference on*, 10-12 2006, S. 3 – 3. – ISSN 1551-6245

[Reed 2000] REED, David P.: *The End of the End-To-End Argument*. April 2000. – URL http://www.reed.com/dpr/locus/Papers/endofendtoend.html. – Zugriffsdatum: 16.05.2010

[Reisig 2010] REISIG, Wolfang: *Petrinetze: Modellierungstechnik, Analysemethoden, Fallstudien*. Vieweg+Teubner, 15 Juli 2010 (Leitfäden der Informatik). – URL http://www2.informatik.hu-berlin.de/top/pnene_buch/pnene_buch.pdf. – Zugriffsdatum: 22.06.2010. – 248 pages. – ISBN 978-3-8348-1290-2

[Riva u. a. 2005] RIVA, G. (Hrsg.) ; VATALARO, F. (Hrsg.) ; DAVIDE, F. (Hrsg.) ; ALCAÑIZ, M. (Hrsg.): *Emerging Communication*. Bd. 6: *Ambient Intelligence. The Evolution of Technology, Communication and Cognition Towards the Future of Human-Computer Interaction*. IOS Press, 2005. – URL http://www.vepsy.com/communication/volume6.html. – Zugriffsdatum: 28.02.2008

[Riva 2005] RIVA, Giuseppe: *Emerging Communication*. Bd. 6: *The Psychology of Ambient Intelligence: Activity, Situation and Presence*. Kap. 2. Siehe [Riva u. a., 2005]. – URL http://www.vepsy.com/communication/volume6.html. – Zugriffsdatum: 28.02.2008

[Riva 2007] RIVA, Oriana: *Middleware for Mobile Sensing Applications in Urban Environments*. Helsinki, Finland, Department of Computer Science, University of Helsinki, Dissertation, November 2007. – URL http://www.infotech.oulu.fi/GraduateSchool/LSeries/2007/riva.html. – Zugriffsdatum: 19.05.2010

[Rödiger 2007] RÖDIGER, Markus: *Multitouch - Technik und Technologie*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2007/vortraege.html. – Zugriffsdatum: 14.06.2010

[Rosenthal und Stanford 2000] ROSENTHAL, Lynne ; STANFORD, Vincent: NIST Smart Space: pervasive computing initiative. In: *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2000. (WET ICE 2000). Proeedings. IEEE 9th International Workshops on*, 2000, S. 6 –11

[Runge u. a. 2009] RUNGE, Mathias ; QUADE, Michael ; BLUMENDORF, Marco ; ALBAYRAK, Sahin: Towards a Common Smart Home Technology. In: *AmI-Blocks'09 – Smart Products: Building Blocks of Ambient Intelligence*. Ernst-Reuter-Platz 7, 10587 Berlin, 2009. – URL http://www.dai-labor.de/fileadmin/files/publications/AmI%202009%20-%20Towards%20a%20common%20smart%20home%20technology.pdf. – Zugriffsdatum: 08.06.2010

[Russell u. a. 2005] RUSSELL, Daniel M. ; STREITZ, Norbert A. ; WINOGRAD, Terry: Building disappearing computers. In: *Communications of the ACM* 48 (2005), Nr. 3, S. 42–48. – URL http://doi.acm.org/10.1145/1047671.1047702. – Zugriffsdatum: 28.02.2008. – ISSN 0001-0782

[de Ruyter und Aarts 2004] RUYTER, Boris de ; AARTS, Emile: Ambient intelligence: visualizing the future. In: *AVI '04: Proceedings of the working conference on Advanced visual interfaces*. New York, NY, USA : ACM, 2004, S. 203–208. – ISBN 1-58113-867-9

[Saks u. a. 2009] SAKS, Kenneth ; LINSKEY, Patrick ; KRISTIANSSON, Peter ; VAN DER VELDEN, Erik ; SCHNIER, Randy ; BARGHOUTHI, Soloman ; BENOIT, Florent ; HALEY, Jason ; KEITH, Michael ; SHINN, Matthew ; ANUPALLI, Deepak ; BURKE, Bill ; KING, Gavin ; DE WOLF, Carlo ; SIMEONOV, Ivo ; PESHEV, Peter ; IRELAND, Evan ; DEMICHIEL, Linda ; BYON, Miju ; KIM, Wonseok ; BIEN, Adam ; BLEVINS, David ; GONCALVES, Antonio ; IHNS, Oliver ; RAHMAN, Reza ; SUN MICROSYSTEMS, INC. (Hrsg.): *JSR 318: Enterprise JavaBeans™, Version 3.1 - EJB Core Contracts and Requirements*.

Dezember 2009. – URL http://jcp.org/aboutJava/communityprocess/final/jsr318/index.html. – Zugriffsdatum: 15.05.2010

[Saltzer u. a. 1984] SALTZER, Jerome H. ; REED, David P. ; CLARK, David D.: End-to-end arguments in system design. In: *ACM Trans. Comput. Syst.* 2 (1984), Nr. 4, S. 277–288. – ISSN 0734-2071

[Satyanarayanan 2001] SATYANARAYANAN, M.: Pervasive computing: vision and challenges. In: *Personal Communications, IEEE* 8 (2001), August, Nr. 4, S. 10 –17. – ISSN 1070-9916

[Schantz und Schmidt 2001] SCHANTZ, Richard E. ; SCHMIDT, Douglas C.: Middleware for distributed systems: Evolving the common structure for network-centric applications. In: MARCINIAK, J. (Hrsg.) ; TELECKI, G. (Hrsg.): *Encyclopedia of Software Engineering*. New York : John Wiley & Sons, Inc., 2001

[Schiele u. a. 2010] SCHIELE, Gregor ; HANDTE, Marcus ; BECKER, Christian: Pervasive Computing Middleware. In: [Shen u. a., 2010], S. 201–227. – ISBN 0387097503, 9780387097503

[Scott 2005] SCOTT, Stacey D.: *Territoriality in Collaborative Tabletop Workspaces*, Department of Computer Science, University of Calgary, Dissertation, 2005. – URL http://scripts.mit.edu/~sdscott/wiki/uploads/Main/scott_dissertation.pdf. – Zugriffsdatum: 28.02.2008

[Shen u. a. 2010] SHEN, Xuemin (Hrsg.) ; YU, Heather (Hrsg.) ; BUFORD, John (Hrsg.) ; AKON, Mursalin (Hrsg.): *Handbook of Peer-To-Peer Networking*. 1. Auflage. New York / Dordrecht / Heidelberg / London : Springer Publishing Company, Incorporated, 2010. – ISBN 0387097503, 9780387097503

[Sockut und Iyer 2009] SOCKUT, Gary H. ; IYER, Balakrishna R.: Online reorganization of databases. In: *ACM Comput. Surv.* 41 (2009), Nr. 3, S. 1–136. – ISSN 0360-0300

[Stegelmeier u. a. 2009] STEGELMEIER, Sven ; WENDT, Piotr ; LUCK, Kai von: iFlat - Eine dienstorientierte Architektur für intelligente Räume. In: *Ambient Assisted Living Kongress*. Berlin, Germany, 2009

[Sun Microsystems, Inc. 2006] SUN MICROSYSTEMS, INC.: *Java Remote Method Invocation Specification*. 2006. – URL http://java.sun.com/javase/6/docs/platform/rmi/spec/rmiTOC.html. – Zugriffsdatum: 14.05.2010

[Sutter und Larus 2005] SUTTER, Herb ; LARUS, James: Software and the Concurrency Revolution. In: *Queue* 3 (2005), Nr. 7, S. 54–62. – ISSN 1542-7730

[Tandler 2004] TANDLER, Peter: *Synchronous Collaboration in Ubiquitous Computing Environments*, Fachbereich Informatik der Technischen Universität Darmstadt, Dissertation, 2004. – URL http://elib.tu-darmstadt.de/diss/000506/. – Zugriffsdatum: 01.06.2010

[Tanenbaum und Steen 2006] TANENBAUM, Andrew S. ; STEEN, Maarten v.: *Distributed Systems: Principles and Paradigms (2nd Edition)*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 2006. – ISBN 0132392275

[Tavares und Valente 2008] TAVARES, Andre L. ; VALENTE, Marco T.: A gentle introduction to OSGi. In: *SIGSOFT Softw. Eng. Notes* 33 (2008), Nr. 5, S. 1–5. – ISSN 0163-5948

[Tolmie u. a. 2002] TOLMIE, Peter ; PYCOCK, James ; DIGGINS, Tim ; MACLEAN, Allan ; KARSENTY, Alain: Unremarkable computing. In: *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM, 2002, S. 399–406. – ISBN 1-58113-453-3

[Urich 2007] URICH, Jaroslaw: *Context-Aware Services: Multimedia-Unterstützung im Flugzeug*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2007/vortraege.html. – Zugriffsdatum: 28.02.2008

[Urich 2008a] URICH, Jaroslaw: *Context-Awareness: aktuelle Projekte*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-aw/vortraege.html. – Zugriffsdatum: 28.02.2008

[Urich 2008b] URICH, Jaroslaw: *Projektbericht – smart:shelf*, Hochschule für Angewandte Wissenschaften Hamburg, Projektbericht, 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp. – Zugriffsdatum: 28.02.2008

[Urich 2009] URICH, Jaroslaw: *Ein Menüplaner als Teil einer multiagentenbasierten Steuerung intelligenter Wohnumgebung*, Hochschule für Angewandte Wissenschaften Hamburg, Masters Thesis, 2009. – URL http://users.informatik.haw-hamburg.de/~ubicomp/papers.html. – Zugriffsdatum: 02.12.2009

[Vollmer 2007] VOLLMER, Sven: *Location-based Services*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2007/vortraege.html. – Zugriffsdatum: 14.06.2010

[Vollmer 2008] VOLLMER, Sven: *Dienstsuche im IntelliHome*, Hochschule für Angewandte Wissenschaften Hamburg, Seminararbeit, 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-aw/vortraege.html. – Zugriffsdatum: 14.06.2010

[Waldo 1998] WALDO, Jim: Remote procedure calls and Java Remote Method Invocation. In: *Concurrency, IEEE* 6 (1998), Juli – September, Nr. 3, S. 5 –7. – ISSN 1092-3063

[Walzer u. a. 2008] WALZER, Karen ; BREDDIN, Tino ; GROCH, Matthias: Relative temporal constraints in the Rete algorithm for complex event detection. In: *DEBS '08: Proceedings of the second international conference on Distributed event-based systems*. New York, NY, USA : ACM, 2008, S. 147–155. – ISBN 978-1-60558-090-6

[Weber u. a. 2005a] WEBER, Werner ; RABAEY, Jan M. ; AARTS, Emile: *Ambient Intelligence*. Kap. *„Introduction"*, S. 1–2. Siehe [Weber u. a., 2005b]. – ISBN 3540238670

[Weber u. a. 2005b] WEBER, Werner (Hrsg.) ; RABAEY, Jan M. (Hrsg.) ; AARTS, Emile (Hrsg.): *Ambient Intelligence*. Springer-Verlag (Berlin Heidelberg), April 2005. – ISBN 3540238670

[Weerawarana u. a. 2005] WEERAWARANA, Sanjiva ; CURBERA, Francisco ; LEYMANN, Frank ; STOREY, Tony ; FERGUSON, Donald F.: *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2005. – ISBN 0131488740

[Weiser 1991] WEISER, Mark: The computer for the twenty-first century. In: *Scientific American* September (1991), S. 94–100. – URL http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html. – Zugriffsdatum: 16.05.2010

[Weiser und Brown 1995] WEISER, Mark ; BROWN, John S.: *Designing Calm Technology*. 12 1995. – URL http://www.ubiq.com/hypertext/weiser/calmtech/calmtech.htm. – Zugriffsdatum: 28.02.2008

[Weiser und Brown 1997] WEISER, Mark ; BROWN, John S.: The coming age of calm technolgy. In: DENNING, P. J. (Hrsg.) ; METCALFE, R. M. (Hrsg.): *Beyond calculation: the next fifty years*. New York, NY, USA : Copernicus, 1997, S. 75–85. – ISBN 0-38794932-1

[Winograd und Flores 1990] WINOGRAD, Terry ; FLORES, Fernando: *Understanding Computers and Cognition: A New Foundation for Design*. 5. Auflage. 355 Chestnut Street, Norwood, New Jersey 07648, U.S.A. : Ablex Publishing Corp., 1990. – ISBN 0-89391-050-3

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 28. Juni 2010	
Ort, Datum	Unterschrift