Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Jaroslaw Urich

Ein Menüplaner als Teil einer multiagentenbasierten Steuerung intelligenter Wohnumgebung

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jaroslaw Urich

Ein Menüplaner als Teil einer multiagentenbasierten Steuerung intelligenter Wohnumgebung

Masterarbeit
im Studiengang Informatik (Master of Science)
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

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

Abgegeben am 6. November 2009

Jaroslaw Urich

Thema der Masterarbeit

Ein Menüplaner als Teil einer multiagentenbasierten Steuerung intelligenter Wohnumgebung

Stichworte

Intelligente Umgebung, Smart Home, Context-Awareness, Ubiquitous Computing

Kurzzusammenfassung

Mit der Entwicklung der Digital-Technik wurden mehr und mehr herkömmliche Geräte durch entsprechende digitale Lösungen ersetzt. Das betrifft nicht nur Geräte des Multimedia-Bereiches wie z.B. Fernseher oder Stereoanlagen, sondern auch andere Bereiche wie z.B. Haushaltsgeräte: Waschmaschinen, Elektroherde u.a. Diese Arbeit beschäftigt sich mit der Konzeption eines Systems für Wohnumgebungen, das die Steuerung solcher Geräte erleichtert und zum Teil sogar selbstständig durchführt. Dies soll das Wohlbefinden der Bewohner in ihrem Alltag steigern.

Jaroslaw Urich

Title of the paper

Menu planner as part of a multi-agent based control of intelligent living environment

Keywords

intelligent living environment, smart home, context awareness, ubiquitous computing

Abstract

Due to the development of digital technology many traditional devices have been replaced with corresponding digital solutions. It happened not only in the multimedia domain (e.g. TV sets or hi-fi systems) but also in other domains (for example housekeeping: washers or cookers). The aim of this paper is the development of a system for living places, that makes it easier for the inhabitants to control such devices and can even be partly autonomous. That should increase the wellness of the inhabitants and help them in their domestic life.

„Man sollte alles so einfach wie möglich sehen - aber auch nicht einfacher.“

Albert Einstein (∗14.03.1879 - †18.04.1955)

Inhaltsverzeichnis

1. Einleitung

Dank des Fortschritts in der Elektronik-Industrie werden die Computer immer leistungsstärker und kleiner. Diese Entwicklung lässt sich an mobilen Geräten beobachten. So passen Smartphones[1] heute in eine Hosentasche und bieten eine Vielzahl von Anwendungsmöglichkeiten.

Mit diesen neuen Geräten ist es möglich übliche Web-Anwendungen zu benutzen, die vielen Anwendern aus der Desktop-Nutzung bekannt sind. Diese mobilen Geräte verfügen beispielsweise über einen Webbrowser[2] und eine WLAN-Kommunikationsschnittstelle, z.B. das iPhone[3] und das HTC G1[4] .

Der immer weiter sinkende Preis für Rechner[5] beschleunigt deren Verbreitung im Alltag[6]. Nicht nur Hochleistungsgeräte wie Notebooks oder Desktop-PCs finden mehr und mehr Anwendung im täglichen Leben, sondern auch schwächere Geräte, die beispielsweise in der Unterhaltungsbranche[7] eingesetzt werden.

Dies alles zeigt, dass der Computer unser Leben relativ stark beeinflusst. Er wird sowohl für industrielle als auch für private Zwecke genutzt: als Kommunikationsmittel, im beruflichen Leben oder für die Unterhaltung.

---

[1] Ein Smartphone vereint den Leistungsumfang eines Mobiltelefons mit dem eines Personal Digital Assistants (PDA) [78]

[2] Webbrowser (oder allgemein auch Browser genannt) sind spezielle Computerprogramme zum Betrachten von Webseiten im World Wide Web (Wikipedia).

[3] ein Mobiltelefon von Apple (siehe mehr unter http://www.apple.com/de/iphone/)

[4] ein Smartphone von HTC (siehe mehr unter http://www.htc.com/www/product/g1/overview.html)

[5] Mini-Notebooks werden heutzutage bereits für 400 Euro angeboten [5].

[6] Im Jahr 2008 wurden über 11,6 Millionen Notebooks gekauft, was einem Wachstum von etwa 15 Prozent zum Vorjahr entspricht [5].

[7] Ein Beispiel hierfür sind Multimedia-Geräte wie ACTIVY von Fujitsu Siemens (mehr zu diesem Gerät ist unter http://www.fujitsu-siemens.de/home/products/entertainment/media_centers/index.html zu finden).

1.1. Motivation

Die Computer-Unterstützung findet in vielen Bereichen des menschlichen Alltags, z.B. in der Wirtschaft, im Gesundheitswesen, in der Forschung oder in der Unterhaltungsindustrie statt. Aufgrund neuer Anwendungen breitet sich die Verwendung immer weiter aus. So wird z.B. seit einiger Zeit an der Computer-Unterstützung im privaten Wohnraum geforscht.

In mehreren Laboren starteten verschiedene Forschungsprojekte, die im Grunde genommen ein gemeinsames Ziel verfolgen: einerseits das Leben im Haus (bzw. in der Wohnung) sicherer und interessanter zu gestalten und andererseits die Arbeit im Haushalt zu erleichtern. Diese Projekte sind unter dem deutschen Begriff *Intelligentes Wohnen*[8] (bzw. *intelligentes Haus* oder *intelligente Wohnung*) gebracht werden. International werden hierfür u.a. folgende Synonyme verwendet: *eHome*, *Smart House*, *Smart Home*, *Smart Living* [72].

Ein weiteres Ziel, das bei diesen Projekten verfolgt wird, ist die Erhöhung der Lebensqualität bedürftiger Menschen. Dies ist als *Ambient Assisted Living* bekannt. [9] definiert diesen Begriff wie folgt:

> Unter „Ambient Assisted Living“ (AAL) werden Konzepte, Produkte und Dienstleistungen verstanden, die neue Technologien und soziales Umfeld miteinander verbinden und verbessern mit dem Ziel, die Lebensqualität für Menschen in allen Lebensabschnitten zu erhöhen. Übersetzen könnte man AAL am besten mit „Altersgerechte Assistenzsysteme für ein gesundes und unabhängiges Leben“.

Die oben genannten Projekte verfolgen das Ziel diese Vorgaben zu erfüllen. Obwohl dieses Gebiet relativ neu ist, gibt es bereits erste Erfolge zu verzeichnen (vgl. [4], [18] oder [1]). Die meisten Projekte bieten aber keine umfassenden Lösungen an, sondern schlagen lediglich vor, wie ein solches System zu implementieren sei. Aufgrund der Komplexität gibt es keine universelle Lösung hierfür.

1.2. Zielsetzung

Das Ziel dieser Arbeit besteht in der Konzeption eines Systems für eine *intelligente Wohnumgebung* (siehe 1.1). In dieser Arbeit sollen die Hauptbegriffe aus diesem Gebiet erläutert und die wichtigsten Lösungsansätze einander gegenüber gestellt werden.

[8] Der Begriff *Intelligentes Wohnen* wurde laut [72] von ZVEI (Zentralverband Elektrotechnik- und Elektronikindustrie e.V.) eingeführt. Dieser Verband hat die gleichnamige Initiative (*Initiative Intelligentes Wohnen*) ins Leben gerufen, deren Teilnehmer (führende Hersteller und ihre Partner) *"vernetzte, bedarfsgerechte und zukunftssichere Lösungen für das Zuhause entwikeln"* [33].

Ein Prototyp des Systems soll als Beweis für die Lauffähigkeit der entwickelten System-Architektur dienen. Der Prototyp soll nicht den vollen Funktionsumfang des Systems aufweisen. Es wird lediglich ein mögliches Anwendungs-Szenario implementiert.

1.3. Gliederung

Kapitel 2 stellt die für diese Arbeit relevanten theoretischen Grundlagen dar. In diesem Zusammenhang werden die Hauptbegriffe, Konzepte und Modelle erläutert.

Kapitel 3 beschäftigt sich mit der Analyse der gestellten Aufgabe. Es präsentiert Labore der HAW und sonstiger Forschungsorganisationen, legt die Anforderungen an das zu entwickelnde System fest und stellt einige verwandte Arbeiten anderer Forscher vor.

Das Design und die Architektur des verfolgten Systems werden in Kapitel 4 beschrieben. Hierbei werden die einzelnen Komponenten dieses Systems geschildert und deren Zusammenarbeit erörtert.

Die realisierungsrelevanten Entscheidungen werden in Kapitel 5.1 dargestellt. Hier werden die wesentlichen Implementierung-Details beschrieben und an den Quellcode-Auszügen verdeutlicht. Kapitel 5.2 evaluiert das realisierte System und erläutert die Vor- und Nachteile der Architektur.

Den Abschluss der Arbeit bietet das Fazit über die erreichten Ziele und ein Ausblick auf weitere Vorhaben im Hinblick auf die Erweiterung des Systems bzw. dessen Architektur.

2. Grundlagen

In diesem Kapitel sollen die für diese Arbeit relevanten Begrifflichkeiten erläutert werden, soweit sie für das weitere Verständniss notwendig sind. Darüber hinaus vertiefende Informationen können jeweils in den dort angegebenen Literaturstellen gefunden werden.

2.1. Begriffsklärung

2.1.1. Ubiquitous Computing (UbiComp)

Der Begriff *Ubiquitous Computing*[1] wurde 1991 von Mark Weiser geprägt [70]. In seiner Arbeit *„The computer for the 21st century“* präsentiert er eine neue Sichtweise auf die Computer-Technologie. Nach Weiser sind *die profundesten Technologien* die, die *unsichtbar* bzw. *versteckt* sind.

Weiser löst sich von der Idee: *„nur ein Personal Computer (PC) pro menschlichen Benutzer“*. Dem Benutzer werden eher eine Vielzahl von *computing devices* zur Verfügung gestellt. Zudem sind diese *devices* unscheinbar, indem sie in die ohnehin verwendeten Gegenstände des Alltags eingebaut (bzw. integriert) werden[2].

Unter *Ubiquitous Computing* (auch als *UbiComp* bezeichnet) wird die *Allgegenwertigkeit der rechnergestützten Informationsverarbeitung*[3] verstanden. Die Idee dieser neuen Sichtweise liegt in der Unauffälligkeit der Computer-Ünterstützung. Hiermit wird versucht die Interaktion eines oder mehreren Anwender mit dem Computer zu vereinfachen (bzw. optimaler Weise unbewusst für den Anwender zu machen).

[1] zu deutsch *Uiquitäre Computertechnik*
[2] Diese *devices* werden als *smart devices* (zu deutsch *„intelligente Gegenstände“*) bezeichnet [73].
[3] vgl. [73]

2.1.2. Pervasive Computing

Pervasive Computing[4] wird u.a. als Synonym für *Ubiquitous Computing* verwendet. Im Gegensatz zu *UbiComp*, das eher einen akademischen Charakter hat, wurde *Pervasive Computing* von der Industrie geprägt [77].

Der Begriff *Pervasive Computing* besitzt eine etwas andere Akzentuierung als *Ubiquitous Computing* [43]. Wie bei *Ubiquitous Computing* geht es auch bei *Pervasive Computing* um die überall durchdringende und immer präsente Informationstechnologie. Jedoch wird hier laut [43] ein anderes Primärziel verfolgt: Die mit *Ubiquitous Computing* bezeichnete Technologie *„schon kurzfristig im Rahmen von Mobile-Commerce-Szenarien und Web-basierten Geschäftsprozessen nutzbar zu machen“*[5].

Der Einsatz von *Pervasive Computing* lässt sich bereits heute beobachten. Dies wird z.B. im Umfeld der E-Commerce deutlich. So können geschäftliche Aktivitäten mit Hilfe der Internet- und anderer Kommunikationstechnologien und der mobilen Geräte (wie Smartphones oder Handys) durchgeführt werden: eine Bestellung in einem Online-Shop[6] oder der Einkauf elektronischer Fahrscheine (sog. *Handy Tickets*)[7].

2.1.3. Ambient Intelligence (AmI)

So wie *Pervasive Computing* ist auch der Begriff *Ambient Intelligence*[8] mit *Ubiquitous Computing* verwandt. Hierbei geht es um die Allgegenwärtigkeit der Computer und deren angebotene Dienste im Alltag. Der Akzent bei *Ambient Intelligence* wird auf das Zusammenspiel der Hardware-, Software- und Kommunikationstechnologie gelegt[9].

Entscheidend für die *Ambient Intelligence* ist die Sensor-Technologie. Mit Hilfe der Sensoren lässt sich die Umgebung und sogar der Zustand des Anwenders erfassen. Somit können Systeme auf den gegebenen Zustand, bzw. dessen Änderung, sinngemäß reagieren (siehe Unterkapitel 2.2.1). Solche Sensoren können relativ kleine Formen annehmen, sodass sie sogar in die Kleidung angenäht werden können. Dieser spezielle Forschungsbereich ist als *Wearable Computing*[10] bekannt [71].

---

[4] zu deutsch *Rechnerdurchdringung* (von engl. *pervasive* - durchdringend)

[5] vgl. [43] und [77]

[6] Viele mobile Geräte verfügen heutzutage über einen funkbasierten Internetzugang (beispielsweise über WLAN oder UMTS). Somit können die Besitzer solcher Geräte jederzeit die üblichen Webanwendungen in Anspruch nehmen (vgl. mit Kapitel 1).

[7] Solche elektronischen Fahrscheine werden vom Hamburger Verkehrsverbund (HVV) eingesetzt (`http://www.hvv.de/handyticket/`).

[8] zu deutsch *Umgebungsintelligenz*

[9] vgl. [71] u. [13]

[10] zu deutsch *tragbare Datenverarbeitung*

2.1.4. Seamless Interaction

Die im vorigen Kapitel dargestellten Begriffe beschreiben Computer-Technologien, die einerseits überall verfügbar sind und andererseits unauffällig (bzw. unscheinbar) für den Menschen sind. Solche Systeme sind in der Lage eigenständig zu agieren. Es stellt sich jedoch die Frage wie diese bedienen werden sollen, wenn eine Anwender-Interaktion gewünscht ist.

Da solche Systeme in der Regel relativ komplex sind[11], kann deren Bedienung für einen Durchschnitts-Anwender unter Umständen relativ unverständlich und mühselig sein [67].

Abbildung 2.1.: HAW Multitouch Tabletop [53]

Seamless Interaction[12] ist ein Forschungsgebiet in der Informatik, das sich mit dem Thema „Benutzer-Interaktion“ auseinandersetzt. Mit *Seamless Interaction* wird versucht die Bedienung einfacher und intuitiver zu gestalten. Die entscheidenden Punkte für deren Design sind

[11] Meistens bestehen diese Systeme aus mehreren Komponenten, die über Netzwerk miteinander kommunizieren (vgl. mit Kapitel 2.3).

[12] zu deutsch *nahtlose Interaktion*

der Mensch mit seiner Physiologie und seinen alltäglichen Gewohnheiten und die reale Welt mit ihren physikalischen Gesetzen. Somit werden virtuelle Objekte mit den physikalischen Eigenschaften versehen, die die entsprechende Gegenstände aus der realen Welt aufweisen[13]. So wird die Bedienung vom Benutzer als intuitiv und vertraut empfunden (vgl. mit [67] und [53]).

Ein Beispiel für *Seamless Interaction* zeigt das an der Hochschule für Angewandte Wissenschaften Hamburg entwickelte *Multitouch Tabletop* (siehe Abbildung 2.1). Es ist ein Multitouch-Gerät, das sich mit üblichen Gesten bedienen lässt: Der Benutzer kann beispielsweise Objekte (in diesem Fall Fotos) drehen oder verschieben in der gleichen Art, wie er dies in der realen Welt tut. Zudem können mehrere Anwender zur gleichen Zeit mit dem Gerät arbeiten. Somit wird die Effizienz des Gerätes erhöht.

2.2. Kontext und Context-Awareness

2.2.1. Kontext

In der Literatur finden sich eine Vielzahl von Definitionen, die den Begriff *Kontext* (engl. *Context*) erörtern. Eine der meist zitierten stammt von Anind K. Dey[14]:

Kontext ist jede Information, die verwendet werden kann um die Situation einer Entity zu charakterisieren. Unter einer Entity wird eine Person, ein Standort oder ein beliebiges Objekt verstanden, das relevant für die Interaktion zwischen dem Benutzer und der Anwendung ist, einschließlich des Benutzers und der Anwendung selbst [14].

2.2.2. Modellierung von Kontext

Eine Kernaufgabe in diesem Zusammenhang ist die Modellierung von Kontext. Hierbei müssen folgende Fragen beantwortet werden[15]:

- *Welche Informationen sind für die jeweilige Interaktion relevant?*
  Es muss festgestellt werden, welche Daten für das System von Bedeutung sind.

[13] In solchen Fällen spricht man von Simulationen.
[14] Diese Zitat wurde von dem Autor dieser Arbeit aus dem Englischen ins Deutsche übersetzt (vgl. [14]).
[15] vgl. [65]

- *Wie können diese Informationen gewonnen werden?*
  Die Erfassung von Informationen ist entscheidend. Hierfür werden unterschiedliche Sensor-Techniken untersucht. Unter „Sensoren“ werden hier nicht nur physikalische sondern auch Software-Sensoren[16] verstanden.
- *Wie werden diese Informationen zu einem Kontext zusammengefasst, verwaltet und gespeichert?*
  Diese komplexe Frage befasst sich mit der Analyse und der Deutung der gewonnenen Informationen (vgl. die ersten beiden Fragen). Das Ergebnis der Analyse ist die Erkenntnis der aktuellen Situation (auch als *Kontext* bezeichnet)[17].

Es existieren bereits mehrere Strategien für die Kontext-Modellierung, z.B. *Activity-Centric Context*, *Schichtenmodell* und *Ontologie-basierte Kontext-Modelle*. Im Folgenden werden lediglich die Hauptkonzepte der jeweiligen Modelle präsentiert.

Activity-Centric Context

Nach [51] beginnt Kontext erst zu existieren, wenn eine Aktion ausgeführt wird (siehe Abbildung 2.2). Hier sollen zuvor die Relevanzkriterien für die Analyse der Information spezifiziert werden. So werden nur die relevanten Informationen bearbeitet und die irrelevanten ignoriert, was eine Überflutung mit Informationen verhindert.

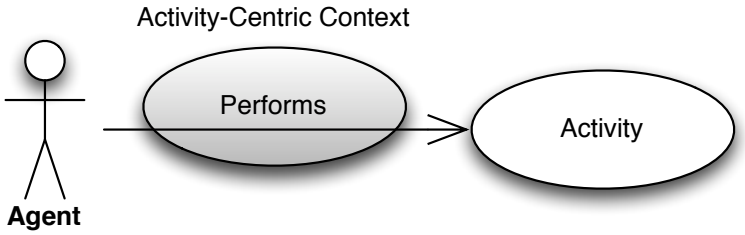


Abbildung 2.2.: Elemente des Activity-Centric Context [51]

Als eine der schwierigsten Aufgaben erweist sich die Identifikation der aktivitätsrelevanten Informationen. Dieses Problem wird mit dem so genannten *cascading contexts* Konzept gelöst[18]. Jeder Kontext lässt sich auf einen allgemeineren Kontext zurückführen[19]. Selbst die

[16] Applikationen oder Programme, die relevante Informationen liefern.
[17] Zur Veranschaulichung kann das folgende Beispiel dienen: *Die Sensoren erfassen, dass eine Person sich im Bett befindet und sich kaum bewegt. Es ist 2 Uhr nachts.* Diese Informationen deuten darauf hin, dass diese Person schläft. *'Die Person schläft'* wäre an dieser Stelle ein möglicher *Kontext*.
[18] vgl. [51]
[19] vgl. mit Vererbung in einer Objekt-Orientierten Sprache

domäneabhängig bereits definierten Kontexte lassen sich aus diesen ableiten. Solche *Root-Kontexte* enthalten alle Informationen, die von einer beliebigen Aktivität benötigt werden und in der jeweiligen Domäne stattfinden.

Der *Unter-Kontext* kann aus dem höherstehenden Kontext abgeleitet (bzw. *vererbt*) werden. So kann der Kontext einer Aktivität zu ihrem Startzeitpunkt ermittelt werden. Hiermit wird das Problem der Kontext-Identifikation einer Aktivität auf das Problem der Identifikation der Kontext-Beziehung reduziert, was wesentlich einfacher zu bewältigen ist[20].

Schichtenmodell

Wie der Name bereits verrät, beschreibt dieses Modell Schichten[21]. Diese Schichten bilden zusammen das so genannte *Kontext-Stack*. Die Verarbeitung dieses Stacks verläuft von unten nach oben. Jede Schicht analysiert die Input-Daten, modifiziert diese und gibt sie als Output an die höher stehende Schicht weiter.

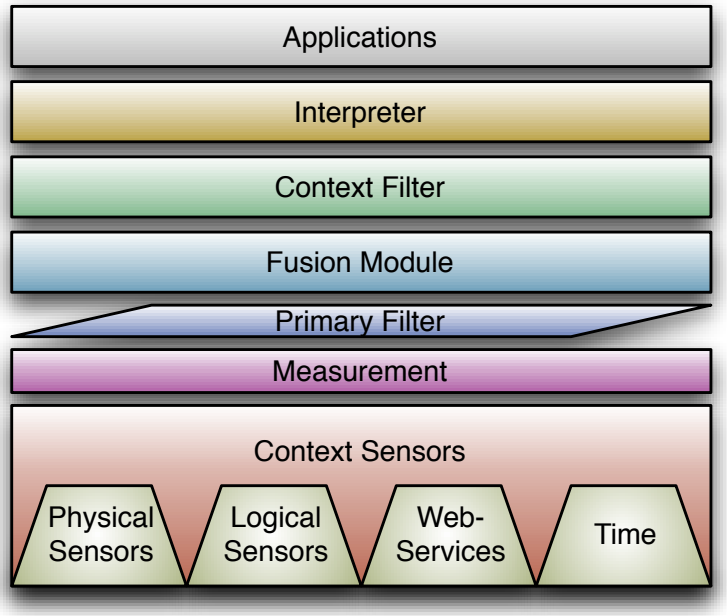


Abbildung 2.3.: Kontext-Stack [2] u. [49]

[20] vgl. mit [30]
[21] bzw. horizontalangeordnete Komponenten

Nach [2] und [49] besteht das *Kontext-Stack* aus sieben Schichten (siehe Abbildung 2.3). Die unterste Schicht nimmt die rohen Daten von Sensoren[22] entgegen. Die oberste dagegen beschreibt die Anwendungen (bzw. Services oder Agenten), die den ermittelten Kontext nutzen. Die Veränderung der Informationen und ihre Bildung zu einem Kontext finden zwischen diesen äußersten Schichten statt.

Weitere Informationen zu diesem Modell können unter [2], [49] oder [30] eingesehen werden.

Ontologie-basierte Kontext-Modelle

Der Begriff *Ontologie* stammt ursprünglich aus der Philosophie und bedeutet *„Lehre vom Sein“* [80]. Er wird verwendet um *die Existenz von Dingen in der Welt zu beschreiben und zu klassifizieren* [29]. In der Informatik bedeutet dieser Begriff *eine konzeptuelle Formalisierung von Wissensbereichen und Begriffsystemen* [62].

Mit Hilfe von *Ontologien* lassen sich Begriffe oder Objekte aus der realen Welt definieren (bzw. beschreiben). Hierbei können die Beziehungen zwischen den Objekten festgelegt werden[23]. Ontologien lassen sich voneinander ableiten. Durch die Anwendung von Operationen[24] lässt sich aus zwei Ontologien eine neue erschaffen.

Zur Spezifikation von Ontologien werden Beschreibungssprachen verwendet. Es gibt eine Reihe solcher Sprachen wie z.B. *CMS (Conseptual Modeling Language)* [54] oder *LOOM* [42]. Eine der meistverbreiteten ist die XML-basierte[25] Sprache *OWL (Web Ontology Language)*[26]. Diese Sprache wurde von der *W3C*[27]spezifiziert, was ihre Verbreitung beschleunigt hat.

Mit Hilfe von Ontologien lassen sich Kontext-Modelle erstellen. So wird von [47] ein auf OWL basiertes Kontext-Modell entwickelt, in dem *klassische Dimensionen* modelliert werden. Diese fünf Dimensionen (siehe Abbildung 2.4) sind: *Actor*, *Activity*, *Time*, *Location* und *Device*. Diese Dimensionen beinhalten Informationen mit deren Hilfe sich folgende Fragen beantworten lassen: *Wer*, *Was*, *Wann*, *Wo* und *Wie*? Hiermit können Situationen beschrieben und in einem Kontext zusammen gefasst werden.

[22] Hier sind nicht nur Hardware- sondern auch Software-basierte Sensoren gemeint (siehe oben).

[23] Es gibt unterschiedliche Arten von Ontologien. So können Begriffe und deren Beziehungen für einen bestimmten Anwendungsbereich mit Ontologien (bzw. so genannten *Domänenontologien*) beschrieben werden. Es gibt aber auch solche, die domänenübergreifend sind (diese sind als *Generelle Ontologien* bekannt). Die letztgenannten sind für mehrere (bzw. alle) Anwendungsgebiete gültig [29].

[24] Diese Operationen sind aus der Mengen-Theorie bekannt, z.B. *Vereinigung*, *Durchschnitt* oder *Differenz*.

[25] *„Die Extensible Markup Language (engl. für „erweiterbare Auszeichnungssprache“), abgekürzt XML, ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdaten“* [74].

[26] Eine ausführliche Beschreibung dieser Sprache kann unter [44] entnommen werden.

[27] Das World Wide Web Consortium ist das Gremium zur Standardisierung der Techniken des World Wide Web.

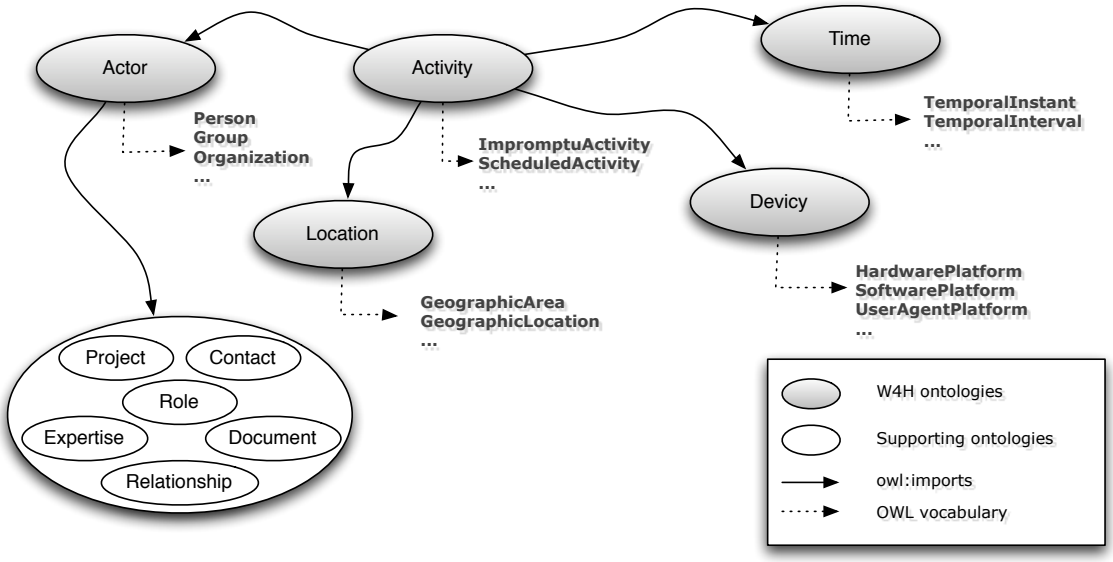


Abbildung 2.4.: Ontologie-basiertes Kontext-Modell [47]

Die Abbildung 2.4 zeigt neben den o.g. Dimensionen, die als fünf Ontologien dargestellt werden, auch einige personenbezogene Ontologien[28]. Diese Ontologien werden von *Actor* importiert. Somit können Personen mit den von ihnen verfassten Dokumenten semantisch verknüpft werden (durch *Document*).

Es gibt auch andere Ontologie-basierte Kontext-Modelle, die in dieser Arbeit nicht betrachtet werden. An dieser Stelle wird auf [68], [60] und [80] verwiesen.

2.2.3. Context-Awareness

Unter *Context-Awareness*[29] wird die Fähigkeit einer Anwendung (bzw. eines Systems oder eines Dienstes) verstanden, *Kontext*[30] wahrzunehmen und das Verhalten anhand dessen anzupassen[31].

Die Anwendung (bzw. der Dienst) kann die Erwartung des Benutzers in größerem Maße erfüllen, wenn diese (bzw. dieser) die Kontext-Informationen bei der Verarbeitung mit einbezieht. So kann das System gezielter auf die Wünsche des Anwenders unter Berücksichtigung der aktuellen Situation eingehen (bzw. diese erfüllen).

[28] z.B. *Project*, *Contact* u.a. (vgl. mit der Blase unten links)
[29] wird häufig als *Kontextsensitivität* ins Deutsche übersetzt
[30] also die Informationen über die Umgebung (siehe oben)
[31] vgl. [14], [15] und [80]

2.3. Architekturmodelle für verteilte Systeme

Die im Kapitel 1.1 vorgestellten Systeme[32] beinhalten bzw. steuern Geräte unterschiedlicher Art (z.B. Heizkörper, Fenster oder Fernseher). Diese Geräte sind meistens an Computer-Netzwerke angeschlossen und können somit nicht nur vom Anwender sondern auch computergesteuert bedient werden.

Die Geräte können frei entfernt oder hinzugefügt werden, ohne dass der Software-Entwickler das System programmatisch anpasst. Ein solches Gerät wird als Teil des Gesamtsystems[33] verstanden. Somit besteht das System aus mehreren Komponenten die miteinander interagieren. Solche Systeme werden unter dem Oberbegriff *„Verteilte Systeme“* in der Informatik behandelt. A. Tannenbaum und M. van Steen definieren ein solches System wie folgt [59]:

> Ein verteiltes System ist eine Menge voneinander unabhängiger Computer, die dem Anwender wie ein einzelnes, kohärentes System erscheinen.

Die Entwicklung eines solchen Systems wird aufgrund der Verteilung erschwert. Beispiele hierfür sind Komponenteninteraktion und Fehlersuche. Da die einzelnen Systemkomponenten auf unterschiedlichen Rechnern laufen können, interagieren sie miteinander über Netzwerke. Bestimmte Protokolle müssen verwendet (bzw. entwickelt) werden, um die Kommunikation dieser zu gewährleisten. Diese Kommunikation stellt zugleich eine potenzielle Fehlermöglichkeit dar. Aus diesem Grund wird die Fehlersuche aufwendiger, denn die Fehlerursache kann nicht nur in den einzelnen Komponenten liegen, sondern auch in deren Kommunikation oder in deren Zusammenarbeit.

Dennoch genießen solche Systeme eine weite Verbreitung in der IT-Welt, da sie auch ihre Vorteile besitzen, wie z.B. einen hohen Grad der Wiederverwendbarkeit der einzelnen Komponenten.

Dieses Kapitel stellt Architekturparadigmen für die Umsetzung eines verteilten Systems dar. Hierbei wird die Akzentuierung im Hinblick auf die in Kapitel 1.1 präsentierten Systeme gelegt.

2.3.1. Service-orientierte Architektur (SOA)

Service-orientierte Architektur (*SOA*) ist ein Architektur-Paradigma, das in der IT-Welt neuerdings verwendet wird. Der *Service* (zu deutsch *Dienst*) spielt hierbei eine zentrale Rolle.

[32] Hier sind Systeme gemeint, die als *eHome*, *Smart House*, *Smart Home*, *Smart Living* etc. bekannt sind (siehe Kapitel 1.1).

[33] Hier sind solche Geräte gemeint, die eine kleine Einheit bilden und nicht als einfache Sensoren genutzt werden.

Diese Services bieten bestimmte Funktionalitäten an und sind voneinander losgekoppelt. Die Wiederverwendbarkeit dieser Dienste nimmt in der SOA eine wichtige Rolle ein. Mit der ständig ändernden Situation in der Wirtschaft sind die Unternehmen gezwungen adäquat auf diese Änderungen zu reagieren. Somit müssen sie flexibel und anpassungsbereit sein. Die SOA bietet die gewünschte Flexibilität und Wiederverwendbarkeit an. Aus diesem Grund setzten immer mehr Unternehmen bei ihrer IT-Lösung auf die SOA.

In der Fachliteratur gibt es eine Vielzahl von Definitionen der SOA. Diese betonen unterschiedliche Aspekte der SOA. Bis jetzt existiert jedoch keine einheitliche Definition von SOA. W. Dostal et al. versuchen eine gewisse Generalität in ihrer Definition zu schaffen. Diese sieht wie folgt aus [19]:

> Unter einer SOA versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachunabhängige Nutzung und Wiederverwendung ermöglicht.

Die SOA ist nur ein Architektur-Paradigma und keine Technologie oder bestimmte Implementierung. Sie beschreibt lediglich das Konzept solcher Architekturen. Abbildung 2.5 stellt die Hauptbestandteile der SOA graphisch dar. Das Fundament des so genannten *SOA-Tempels* bilden *offene Standards*, *Sicherheit* und *Einfachheit*. *Verteilte Dienste*, *lose Kopplung*, *Verzeichnisdienst* und *prozessorientierte Struktur* sind die tragenden Säulen.

In der SOA können Beteiligte eine der drei folgenden Rollen annehmen:

- *Anbieter*,
- *Nutzer* oder
- *Vermittler*.

Die *Anbieter* bieten bestimmte Dienste an, die von den *Nutzern* in Anspruch genommen werden können. Hierbei kann ein Beteiligter u.U. verschiedene Rollen annehmen. So kann er beispielsweise Dienste anbieten, bei deren Realisierung andere Dienste verwendet werden. In einem solchen Fall ist er sowohl Anbieter als auch Nutzer. Der *Vermittler* spielt hierbei die Rolle eines Maklers, in dem er die Dienste verwaltet und bei der Suche nach jenen seine Hilfe anbietet.

Die Dienste werden in einer maschinenlesbaren Form (*Service Description*) beschrieben. Anhand dieser Beschreibung werden die Dienste registriert, verwaltet und und gegebenenfalls in Anspruch genommen. Ein Dienstanbieter registriert seine Dienste, in dem er deren Beschreibungen zu einem Dienstverzeichnis senden. Der Dienstnutzer schickt eine Anfrage für eine bestimmte Dienstart an das Dienstverzeichnis. Dieser antwortet ihm seinerseits mit dem Übersenden der Beschreibung des passenden Dienstes. Mittels dieser Beschreibung

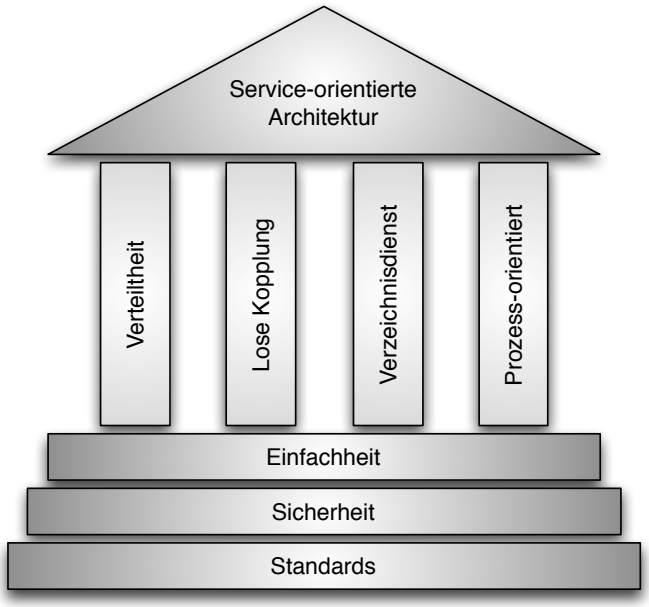


Abbildung 2.5.: SOA-Tempel

wird die Kommunikation zwischen dem Dienstnutzer und dem Dienstanbieter hergestellt, und der Dienst wird in Anspruch genommen. Abbildung 2.6 stellt diesen Prozess graphisch dar.

Für die Kommunikationserleichterung kann ein so genannter *Enterprise Service Bus (ESB)* eingesetzt werden. Der gesamte Datenfluss erfolgt über den ESB. Der ESB unterstützt verschiedene Kommunikationsarten, sodass die beteiligten Seiten nur eine der Kommunikationsarten anbieten müssen. Partner mit unterschiedlicher Kommunikationsschnittstelle (z.B. einer über SMTP[34] und der andere über HTTP[35]) können somit über ESB miteinander kommunizieren.

Es gibt bereits erfolgreich implementierte Instanzen der SOA. Zu den erfolgreichsten und meistverbreiteten zählt die *Web-Services-Architektur*. Diese Architektur nutzt eine Vielzahl von Standards für deren Realisierung. Vielleicht ist das der Grund ihres Erfolges.

Die Kommunikation erfolgt mittels Nachrichten, die über das Internet verschickt und empfangen werden. Diese Nachrichten sind XML-basierte Standards. So erfolgt z.B. die Kommunikation zwischen dem Dienstanbieter und dem Dienstnutzer über *SOAP*[36], wobei die Service

[34] SMTP (*Simple Mail Transfer Protocol*) ist ein Protokoll zum Austausch von E-Mails in einem Computer-Netzwerk.
[35] HTTP (*Hypertext Transfer Protocol*) ist ein Protokoll zur Übertragung von Daten über ein Computer-Netzwerk.
[36] SOAP (ursprünglich für *Simple Object Access Protocol*) ist ein Netzwerkprotokoll, mittels dessen Daten

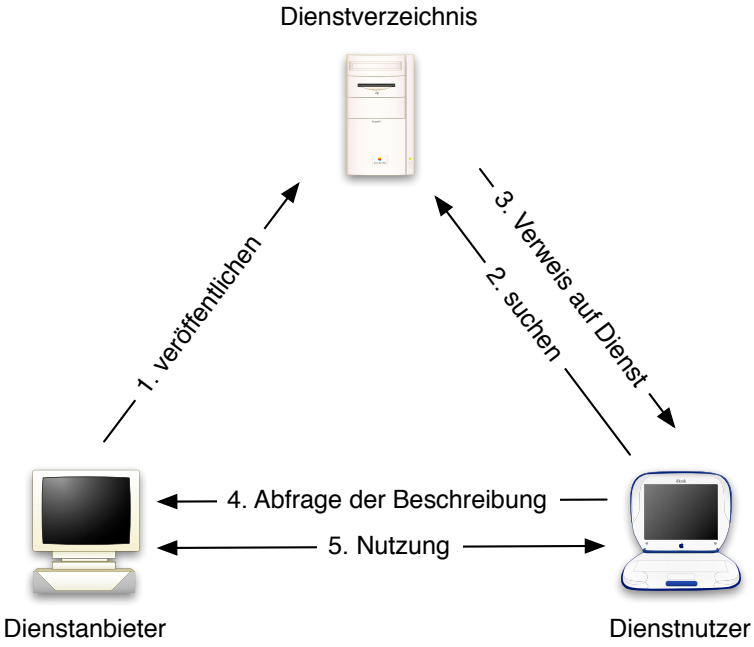


Abbildung 2.6.: Rollen und Aktionen in einer SOA (angelehnt an [19])

Description mit Hilfe von *WSDL*[37] stattfindet. Für die Dienstverwaltung wird eine standardisierte Verzeichnisstruktur *UDDI*[38] verwendet.

Web Services beinhaltet auch weitere Spezifikationen und Standards, deren Behandlung den Rahmen der Arbeit sprängen würde. Für eine ausführliche Behandlung der Web Services und SOA im allgemeinen wird der Leser gebeten, die Fachliteratur wie z.B. [19], [21] oder [39] zu lesen.

2.3.2. Multi-Agenten / -Architektur

Dieses Kapitel stellt Multi-Agenten-Systeme dar. Zunächst soll jedoch die Begrifflichkeit „Agent“ erläutert werden.

zwischen Applikationen bzw. Systemen ausgetauscht und RPC (*Remote Procedure Calls*) durchgeführt werden können.

[37] WSDL (*Web Services Description Language*) ist eine XML-basierte Beschreibungssprache für die Dienstbeschreibung in Web Services.

[38] UDDI (*Universal Description, Discovery and Integration*) spezifiziert eine Verzeichnisstruktur für die Verwaltung von Web-Services-Metadaten.

Agent

Agenten und deren Systeme werden in der Informatik im Rahmen der *Künstilchen Intelligenz (KI)* untersucht. Sie werden für die Lösung folgender Aufgaben eingesetzt: Verarbeitung natürlicher Sprache und Bilder, maschinelles Problemlösen, Entscheidungsunterstützung, Konstruktion (teil-)autonomer Systeme (z.B. Roboter).

Ein *Software-Agent*[39] ist ein Programm, das bestimmte Probleme löst, bzw. bestimmte Aufgaben erfüllt. Hier stellt sich jedoch die Frage: Was unterscheidet einen *Agenten* von einem *„gewöhnlichen“* Programm? Es gibt in der Literatur eine Reihe von Definitionen, die den Begriff *Software-Agent* erklären. Diese Definitionen betrachten diesen Begriff unter differenten Gesichtspunkten. Diverse Artikel sind deren vergleichenden Betrachtung gewidmet [26]. Im Folgenden wird eine Definition präsentiert, die die Arbeitsweise eines Agenten akzentuiert und zugleich die wichtigsten Aspekte eines Agenten wie *Andauernde Verfügbarkeit / Aktivität*, *Interaktion mit einer Umwelt* und *Autonomie* beinhaltet.

> Ein Software-Agent ist ein längerfristig arbeitendes Programm, dessen Arbeit als eigenständiges Erledigen von Aufträgen oder Verfolgen von Zeiten in Interaktion mit der Umwelt beschrieben werden kann [29].

Ein Software-Agent weist demnach folgende Eigenschaften auf:

- *autonom* - das Programm arbeitet ohne Benutzereingriffe
- *proaktiv* - das Programm führt Aktionen aufgrund eigener Initiative aus
- *reaktiv* - das Programm reagiert auf Umgebungsänderungen
- *sozial* - das Programm kommuniziert mit anderen Agenten
- *lernfähig/anpassungsfähig* - das Programm lernt aufgrund eigener Beobachtung

In der Literatur zur *Künstlichen Intelligenz* werden weitere Eigenschaften einem Software-Agenten zugewiesen, bzw. die vorgestellten weiter unterteilt. Diese können der Fachliteratur wie beispielsweise [29], [7] oder [22] entnommen werden.

Arbeitsweise eines Agenten und seine Bestandteile

Ein komplexer *Software-Agent* besteht aus mehreren Komponenten. Bei einer groben Betrachtung dieser lassen sich folgenden fünf unterscheiden:

- *Umweltkopplung*

[39] Für *Software-Agent* wird in der Informatik häufig der allgemeinere Begriff *Agent* verwendet.

- *Steuerung und Management*
- *Fähigkeiten*
- *Umweltmodell*
- *Entscheidungskomponente*

Die Komponente *Umweltkopplung* stellt eine Schnittstelle zwischen dem Agenten und der Umgebung bereit. Sie beinhaltet Sensoren, über die der Agent die Informationen von Außen bekommen kann (*Input*), und steuert die Ausgabe-Geräte, über die der Anwender die Lösung erhält (*Output*).

Eine der wichtigsten Aufgaben der Komponente *Steuerung und Management* ist die Synchronisation der einzelnen Phasen im Arbeitsprozess des Agenten. Ein komplexer Agent kann bei entsprechender Strukturierung als ein verteilt arbeitendes System angesehen werden, in dem die einzelnen Komponenten Nachrichten austauschen. *Steuerung und Management* hat je nach Agent zudem folgende Aufgaben: eine zentrale Ablauf-Steuerung mit Zuteilung von Rechenzeiten, Synchronisation, Vermeidung von Blockierungen, die Zusicherung von Fairness usw. analog zu anderen verteilten Systemen. Auch die allgemeinen Verwaltungs-Aufgaben wie Anmelden und Abmelden bei einer Zentrale oder die Initialisierung sind dem Arbeitsbereich von *Steuerung und Management* zuzuordnen.

Es wird empfohlen in den Komponenten *Umweltkopplung* und *Steuerung und Management* die notwendigen Middleware-Funktionen des Software-Agenten bereit zu stellen.

In der Komponente *Fähigkeiten* werden die Möglichkeiten festgehalten, über die der Agent verfügt. Diese werden oft als Pläne betrachtet, die für die Lösung eines bestimmten Problems eingesetzt werden können. Softwaretechnisch können diese Fähigkeiten als Prozeduren / Methoden oder Skripts angesehen werden.

Die Komponente *Umweltmodell* verwaltet die Informationen über die Außenwelt. Nach [29] ist es sinnvoller dieses Umweltmodell als *Annahme (engl. belief)* zu bezeichnen, um zu verdeutlichen, dass die Annahmen möglicherweise kein korrektes *Wissen (engl. knowledge)* darstellen. Das gewonnene Modell kann mit *Kontext* verglichen werden (siehe Kapitel 2.2.1).

Die *Entscheidungskomponente* wählt Handlungen aus. Entsprechend der gegebenen Situation, bzw. des aktuellen Umweltmodells, wird eine passende Fähigkeit (bzw. ein passender Plan) ausgesucht und ausgeführt.

Die Komponenten *Fähigkeiten*, *Umweltmodell* und *Entscheidungskomponente* repräsentieren somit die *Intelligenz* eines Agenten.

Grundsätzlich sind drei Phasen bei der Arbeit eines Agenten unterscheidbar [29]:

- *Phase der Informationsaufnahme*
  Der Agent nimmt Informationen aus seiner Umwelt auf. Er empfängt Nachrichten und nimmt die Aufträge entgegen. Er registriert bzw. beobachtet seine Handlungen.
- *Phase der Wissensverarbeitung und Entscheidung*
  Der Agent aktualisiert sein Wissen anhand der gewonnenen Informationen. Er analysiert die aktuelle Situation, die entgegengenommenen Aufträge und den Fortschritt bereits begonnener Handlungen. Er trifft Entscheidungen über seine unmittelbaren und zukünftigen Handlungen.
- *Phase der Aktionsausführung*
  Die getroffenen Handlungen werden ausgeführt (z.B. Versenden von Nachrichten, Ausführen von Berechnungen, Präsentation der Ergebnisse).

Listing 2.1 veranschaulicht die Arbeitsweise eines Agenten. Der Agent erhält Eingabe `S` z.B. vom Nutzer oder anderen Agenten und formt sie mit `sense(S)` in eine Wahrnehmung der Umwelt `W` um. In der Phase `thought(W)` wird anhand der Wahrnehmung eine Entscheidung über die nachfolgende Aktion[40] getroffen. Diese Aktion wird dann mit `act(C)` ausgeführt. Nach der Ausführung der Aktion ist der Agent bereit weitere Aufträge zu bearbeiten.

Listing 2.1: Abarbeitungsphasen eines Agenten [29]

```
repeat
        W := sense(S);
        C := thought(W);
        A := act(C);
forever
```

Laut dieser Arbeitsdefinition implementiert die Komponente *Umweltkopplung* die Funktionen `sense` und `act`, wobei die Funktion `thought` die *Intelligenz* des Agenten darstellt, die in den Komponenten *Fähigkeiten*, *Umweltmodell* und *Entscheidungskomponente* realisiert wird (siehe oben).

Agententypen

Je nach Fähigkeit und Arbeitsweise werden verschiedene Arten von Agenten unterschieden. Im Folgenden werden die wesentlichen Agententypen vorgestellt.

[40] vgl. mit der Komponente *Fähigkeiten* (siehe oben)

Deliberative Agenten *Deliberative Agenten* zeichnen sich durch zwei wesentliche Merkmale aus. Sie setzen ein explizites symbolisches Modell der Umwelt und die Fähigkeit zur logischen Schlussfolgerung als Grundlage für intelligentes Handel voraus [7].

Die Modellierung der Umwelt erfolgt in der Regel im Voraus und bildet eine Hauptkomponente der Wissensbasis eines Agenten. Dieser Vorgang stellt dabei zwei wesentliche Herausforderungen dar: Die Auswahl einer geeigneten Repräsentationssprache und der eigentliche Übersetzungsvorgang. Eine Vorgehensweise muss gefunden werden, mit der sich ein konkretes und inhaltlich ausreichendes Umweltmodell eines Agenten mit vertretbaren Aufwand erstellen lässt. Hierbei muss eine Repräsentationsform gewählt werden, die eine ausreichende Modellierungsfunktionalität bereitstellt. Angesichts der hohen Komplexität derartiger Repräsentation sind deliberative Agenten nur bedingt für den Einsatz in dynamischen Umgebungen geeignet. Sie sind nur schwer im Stande die Änderungen bezüglich ihrer Umwelt in ihr bereits bestehendes Umweltmodell zur Laufzeit aufzunehmen. Meistens fehlen ihnen hierzu das notwendige Wissen und die notwendige Ressourcen.

Im Zuge des Schlussfolgerungsprozesses modifiziert der Agent anhand des in seiner Umwelt enthaltenen Wissen seinen internen Zustand. Dieser interne Zustand wird häufig auch als *mentaler Zustand* bezeichnet und setzt sich aus drei Grundkomponenten zusammen: *Überzeugungen (engl. belief)*, *Wünsche (engl. desire)* und *Intentionen (engl. intention)*. Aus diesem Grund werden deliberative Agenten auch als *BDI Agenten*[41] bezeichnet. Neuere Ansätze erweitern das klassische Konzept um weitere zwei Faktoren: *Ziele (engl. goal)* und *Pläne (engl. plan)*. Somit repräsentieren die fünf Faktoren den mentalen Zustand eines deliberativen Agenten [7] (siehe Abbildung 2.7).

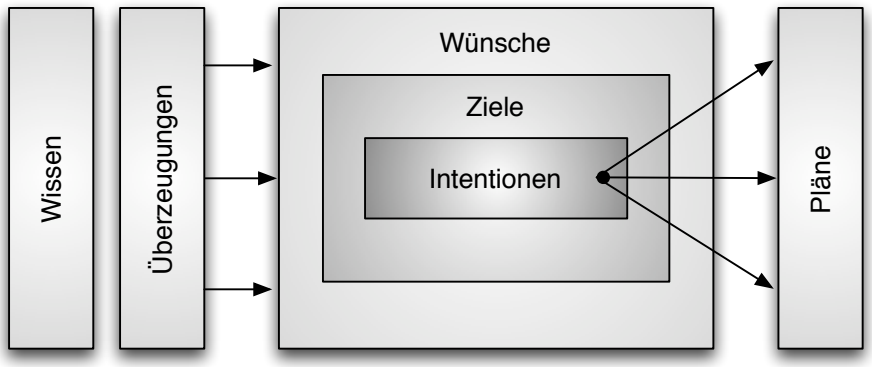


Abbildung 2.7.: BDI Struktur [7]

- ***Überzeugungen*** umfassen die grundlegende Ansichten eines Agenten hinsichtlich

[41] von engl. *belief, desire, intention*

seiner Umwelt. Mittels dieser drückt der Agent seine Erwartungen über mögliche bevorstehende Umweltzustände aus.

- ***Wünsche*** werden direkt aus den Überzeugungen abgeleitet. Sie beinhalten die Abschätzungen zukünftiger Umweltsituationen aus der Agentensicht. So kann beispielsweise ein Wunsch des Agenten das Eintreten bzw. das Nichteintreten eines bestimmten zukünftigen Umweltzustandes sein. Bei der Formulierung der Wünsche werden keine Aussagen darüber getroffen, ob und in wie weit diese Wünsche realisierbar sind. Es kann also vorkommen, dass ein Agent einen unrealistischen Wunsch hat. Auch in Konflikt miteinander stehende Wünsche sind nicht ausgeschlossen.
- ***Ziele*** stellen aus der Sicht des Agenten erfüllbare Wünsche dar. Im Unterschied zu den Wünschen eines Agenten sollten seine Ziele daher realistisch sein und nicht in Konflikt zueinander stehen. Die Ziele repräsentieren den potenziellen Handelsraum eines Agenten, denn sie stellen die zur Verfügung stehenden Handelsalternativen zu einem bestimmten Zeitpunkt dar.
- ***Intentionen*** bilden eine Untermenge der Ziele. Entscheidet sich ein Agent ein Ziel zu verfolgen, so wird aus einem Ziel eine Intention. Meistens kann nur ein Ziel zur Zeit verfolgt werden. Ein Grund hierfür könnte z.B. die Knappheit der notwendigen Ressourcen sein. Als Folge dessen werden die anstehenden Ziele priorisiert und ihrer Wichtigkeit nach bearbeitet.
- ***Pläne*** fassen die Intentionen zu koinzidierten Einheiten zusammen. Dabei besteht eine Korrelation zwischen Intentionen und Plänen: Intentionen stellen Teilpläne des Gesamtplans eines Agenten dar und die Menge aller Pläne reflektiert wiederum die Intentionen eines Agenten.

Die Erkenntnisse des BDI-Modells beeinflussen die architektonische Gestaltung eines deliberativen Agenten. Die Abbildung 2.8 verdeutlicht die zentralen Komponenten eines deliberativen Agenten.

Ein wesentlicher Bestandteil der Wissensbasis des Agenten ist das symbolische Umweltmodell. Aus diesem werden Wünsche, Ziele und Intentionen abgeleitet. Dies geschieht mit Hilfe der Schlussfolgerungskomponente (engl. *reasoner*). Die Intentionen werden vom Planer übernommen und zu einem Gesamtplan zusammengefasst. Der Planer ist im Stande die Abhängigkeiten zwischen den Intentionen zu erkennen und diese dann beim Erstellen des Gesamtplans zu berücksichtigen. So können z.B. Ergebnisse einer Intention als Eingabewerte einer anderen Intention verwendet werden. Mit dem Eintreffen neuer Intentionen werden Pläne kontinuierlich angepasst.

Die aktuellen Pläne werden an den Scheduler übergeben. Jeder Plan besteht aus einer oder mehreren Aktionen. Diese können sequenziell oder parallel abgearbeitet werden. Der Scheduler trifft die Entscheidung, welche Aktionen und zu welchen Zeitpunkt auszuführen sind.

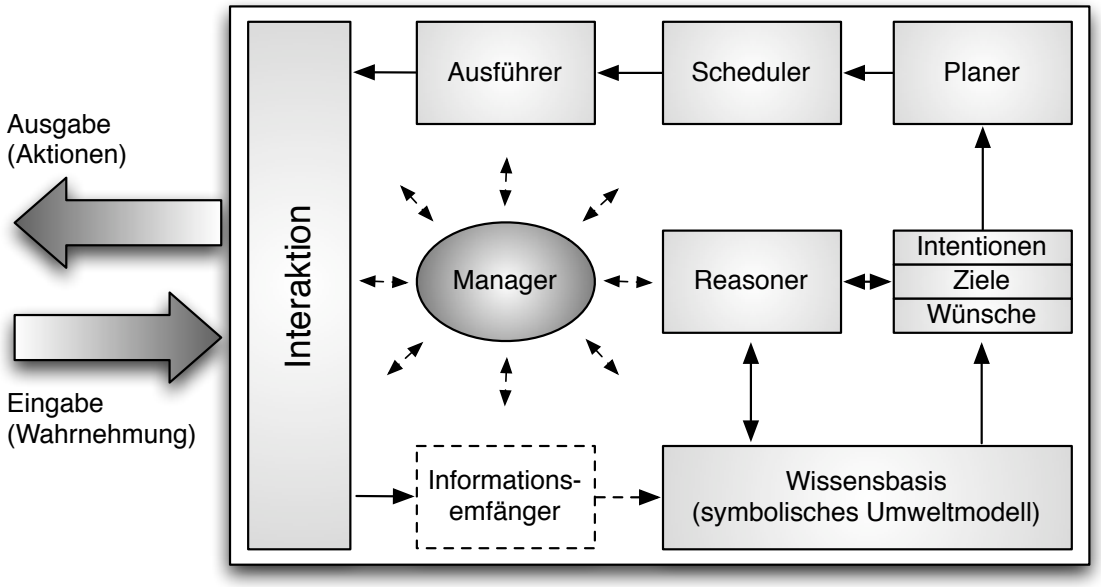


Abbildung 2.8.: Architektur eines deliberativen Agenten [7]

Die Aktionen werden demzufolge vom Scheduler mit einem optimalen und einem spätesten Ausführungszeitpunkt und den benötigten Ressoucen versehen. Um diese Entscheidungen treffen zu können, muss der Scheduler einen ständigen Überblick über die vorhandenen Ressourcen haben.

Die Aktionen mit den Informationen vom Scheduler werden vom Ausführer erhalten. Der Ausführer ist, wie sein Name verrät, für die Ausführen der Aktionen zuständig. Er leitet die Aktionen ein, überwacht deren Verarbeitung und beendet sie im Falle eines Erfolges. Falls die Ausführung länger dauert als es vom Scheduler vorgesehen wurde, wird diese vom Ausführer abgebrochen. Die Aktionen, die bis zum spätesten Zeitpunkt nicht gestartet werden können, werden an den Scheduler bzw. Planer zurück geschickt.

Die Abbildung 2.8 zeigt, dass eine dynamische Anpassung des internen Umweltmodells nur bedingt möglich ist. Die Agenten können zwar über einen Informationsempfänger verfügen, dieser wird aber in der Regel nur selten für die Modifikation des Wissensbasis verwendet[42].

Deliberative Agenten haben auf Grund ihrer Komplexität eine relativ starre Struktur. Dies ist einer der wichtigsten Kritikpunkte bei ihren Einsatz in dynamischen Umgebungen. Die ständig ändernde Umwelt verlangt vom Agenten adäquate Reaktionen auf die Änderungen.

[42] vgl. mit Anfang des Kapitels

Dies ist nur schwer erfüllbar, da die aktuelle Umweltsituation nur eingeschränkt berücksichtigt werden kann.

Reaktive Agenten *Reaktive Agenten*[43] verfolgen, im Gegensatz zu den deliberativen Agenten, einen anderen Ansatz. Sie besitzen kein Umweltmodell. Reaktive Agenten besitzen keine Fähigkeiten im Sinne eines Agenten und verzichten auf aufwendige Schlussfolgerungsprozesse[44]. Der Grund hierfür liegt im Wunsch einen kompakten, fehlertoleranten und hauptsächlich flexiblen Agenten zu entwickeln.

Ein reaktiver Agent bezieht seine Intelligenz aus der Interaktion mit seiner Umgebung. Folglich wird auf die Interaktion großen Wert gelegt. Einige Anhänger der reaktiven Agenten sind sogar überzeugt, dass die Intelligenz eines reaktiven Agenten durch die kontinuierliche Interaktion mit der ständig ändernden Umwelt entsteht[45].

Abbildung 2.9 stellt die Architektur eines reaktiven Agenten dar. Mit Hilfe von Sensoren nimmt der Agent Informationen auf. Diese werden, meistens im Rohformat, an die aufgabenspezifischen Kompetenzmodule weiter geleitet. Jedes Kompetenzmodul ist für einen relativ einfachen und klar definierten Aufgabenbereich zuständig. Alle für die Erfüllung seiner Aufgaben notwendigen Eigenschaften sind in den Kompetenzmodulen zusammengefasst. Somit sind die Kompetenzmodule voneinander unabhängig und können deshalb parallel betrieben werden. Das Ergebnis ihrer Arbeit sind Reaktionen, die unter Einsatz von Aktuatoren auf die Umwelt übertragen werden.

Ein reaktiver Agent zeichnet sich durch eine intensive Interaktion mit seiner Umwelt, eine schnelle Verarbeitung und einen relativ einfachen Aufbau aus. Ein Vorteil des reaktiven Agenten gegenüber eines deliberativen Agenten ist das Fehlen einer Zentralkomponente[46]. Somit kann ein reaktiver Agent im Falle eines Ausfalls eines der Kompetenzmodule seine Funktionalität weiter ausüben. Sie wird lediglich auf die Eigenschaften des ausgefallenen Moduls eingeschränkt.

Reaktive Agenten sind in ihrer Handelsweise eingeschränkt. Sie können beispielsweise nicht wie deliberative Agenten neue Handlungen erzeugen. Die möglichen Handelswege werden bei reaktiven Agenten in ihren Kompetenzmodulen gehalten.

Hybride Agenten Deliberative und reaktive Agenten sind in ihrer Arbeitsweise sehr verschieden. Sie haben ihre Vor- und Nachteile. Agenten, die die Vorteile zweier Agententypen in sich vereinen, werden als *hybride Agenten* bezeichnet.

[43] auch als *stimulus-response-Agenten* bezeichnet
[44] vgl. [7] und [29]
[45] vgl. [8] und [7]
[46] vgl. Reasoner oder Planer eines deliberativen Agenten

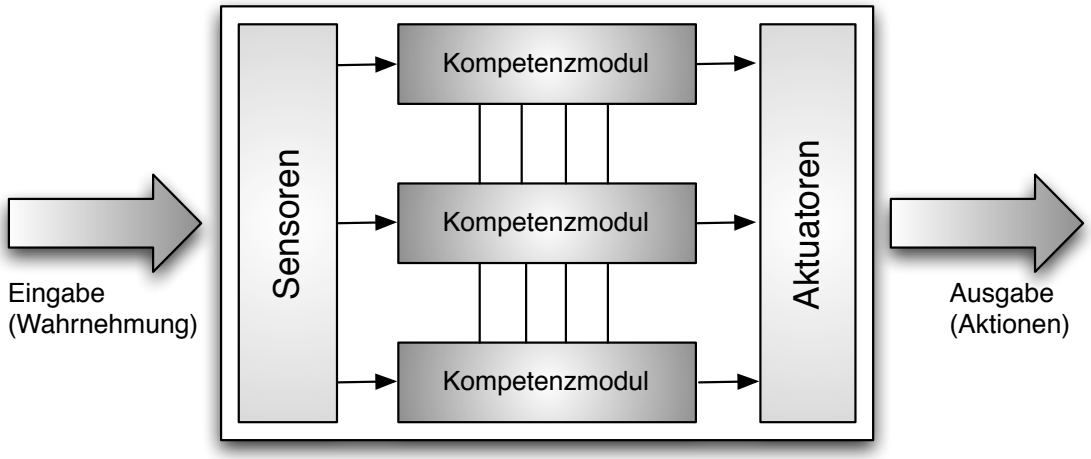


Abbildung 2.9.: Architektur reaktiver Agenten [7]

Diese Agenten besitzen sowohl eine reaktive als auch eine deliberative Komponente. Die reaktive Komponente wird für die Interaktion mit der Umwelt verwendet, wobei die deliberative Komponente mit ihrem internet Umweltmodel und ihrer aufwendigen Schlussfolgerungsfähigkeit für die Planung und Entscheidung zuständig ist. Inwieweit diese Komponente ausgebaut wird hängt von dem jeweiligen Einsatz und der Entscheidung des Systemarchitekten ab.

Für eine ausführlichere Behandlung der drei Agententypen wird hier auf die externe Literatur verwiesen wie z.B. [7], [29] oder [22].

Multi-Agenten-Systeme

Mehrere miteinander kooperierende Agenten werden als *Multi-Agenten-System (MAS)* bezeichnet. In solchen Systemen werden eine Vielzahl von Agenten für die Lösung eines Problems herangezogen. Die Agenten werden hier als kleine Einheiten betrachtet, die bestimmte individuelle Aufgaben lösen. Diese Aufgaben sind als Teilprobleme zu verstehen, sodass das Lösen dieser zur Bewältigung des Gesamtproblems beiträgt.

Bei einer verteilten Problemlösung müssen die folgenden vier Schritte ausgeführt werden [29]:

1. *Zerlegung des Problems in getrennt lösbare Teilprobleme*,
2. *Zuordnung der Teilprobleme zu einzelnen Agenten*,

3. *Lösung der Teilprobleme durch Agenten*,
4. *Zusammenführung der Resultate*.

Die verteilte Lösung ist jedoch mit einem zusätzlichen Aufwand verbunden. Dieser Aufwand rechtfertigt sich, wenn die einzelnen Teilprobleme großteils unabhängig voneinander parallel bewältigt werden können. In einem System mit mehreren Prozessoren können diese Teilprobleme parallel verarbeitet werden, was einen wesentlichen Performanzgewinn mit sich bringt.

Kooperation und Koordination Als anspruchsvoll im Zusammenhang mit MAS erweist sich die Aufgabenverteilung zwischen Agenten. Wenn die Aufgabenverteilung nicht statisch vorgegeben ist, muss sie von Agenten ausgehandelt werden. Einer der bekanntesten Ansätze hierfür ist das *Kontrakt-Netz-Protokoll*. Mit diesem Verfahren findet die Verteilung von Aufgaben entsprechend der aktuellen Belastung und der Kompetenz der Problemlöser statt.

Nach dem Kontrakt-Netz-Protokoll werden zwei Arten von Einheiten unterschieden: *Manager* und *Arbeiter*. Der Manager zerlegt ein Problem in Teilprobleme, macht Ausschreibungen[47] und nimmt Angebote von den Arbeitern entgegen. Anhand der Angebote ordnet er die Aufgaben den Agenten zu und überwacht deren Ausführung. Zum Schluss fasst er die Teilergebnisse zu einer Gesamtlösung zusammen. Ein Arbeiter bewirbt sich entsprechend seiner Fähigkeiten und der aktuellen Auslastung für die Ausführung der Aufgaben. Er löst das Problem und teilt seine Ergebnisse dem Manager mit. Ein Arbeiter kann u.U. die Rolle eines Managers übernehmen, indem er die ihm zugewiesene Aufgabe weiter unterteilt und deren Ausführung überwacht.

Einen anderen Ansatz bei der Aufgabenverteilung bieten die so genannten *Blackboard-Systeme*. Hier werden die Aufgaben an einer zentralen Stelle (*Blackboard*) abgelegt, die für alle Teilnehmer sichtbar ist. Diese Aufgaben werden von Agenten bearbeitet und die Lösungen wieder auf das Blackboard geschrieben. Die Ergebnisse anderer Agenten können bei der Lösung eigener Aufgaben mit einbezogen werden. Eine etwas ausführlichere Behandlung der Blackboard-Modelle findet in Kapitel 2.3.3 statt.

Weitere Theorien und Verfahren zur Kooperation und Koordination in Multi-Agenten-Systemen können z.B. [81] entnommen werden.

Kommunikation Kommunikation ist ein zentrales Thema in der Informatik. Viele Formalismen wurden für die Repräsentation von Kommunikation konkurrierender Systeme entwickelt [81]. Diese Formalismen sind auf die Probleme fokussiert, die bei der Interaktion miteinander handelnder Systeme auftreten.

[47] Hiermit ist die Freigabe der Aufgabe gemeint.

Das charakteristische Problem in der Erforschung der Kommunikation konkurrierender Systeme ist die *Synchronisation* mehrerer Prozesse[48]. Zwei Prozesse müssen synchronisiert werden, wenn sie sich bei ihrer Ausführung gegenseitig behindern können. Die Synchronisation von Prozessen kann mittels Kommunikation dieser Prozesse erreicht werden[49].

In einer objektorientierten Sprache, wie z.B. Java, kann die Kommunikation zwischen Prozessen mittels Methoden-Aufruf zweier Objekte realisiert werden. So kann ein Objekt eine Methode eines anderen Objektes aufrufen und die zu übertragenden Daten als Parameter der Methode mit übergeben.

In der agenten-orientierten Welt gibt es dieses Konzept, das den Agenten erlaubt, Methoden oder Prozeduren anderer Agenten aufzurufen, nicht. Der Grund liegt darin, dass Agenten autonom sind. Sie haben die Kontrolle über ihren Zustand und ihr Verhalten. Im Allgemeinen können die Agenten weder andere Agenten „zwingen“, bestimmte Aktionen auszuführen, noch Daten in den internen Zustand anderer Agenten zu schreiben. Das heißt aber nicht, dass die Agenten nicht miteinander kommunizieren können. Sie können bestimmte Aktionen (Kommunikation-Aktionen) ausführen, die das Verhalten anderer Agenten in ihrem Sinne beeinflussen.

Im Folgenden wird eine Theorie vorgestellt, die die Kommunikation als Aktion behandelt, und die mit ihrer Hilfe entstandenen Agenten-Kommunikationssprachen.

Sprechakttheorie Die *Sprechakttheorie* (*engl. Speech act theory*) hat mit der Arbeit des Philosophen John Austin begonnen. Nach seiner Auffassung können mit sprachlichen Äußerungen (Reden) nicht nur Sachverhalte beschrieben und Behauptungen aufgestellt, sondern darüber hinaus Handlungen (Akte) vollzogen werden, die eine Änderung des Zustands der Welt zur Folge haben [81], [79].

Austin identifizierte eine Reihe von *Performativen Verben*[50], die unterschiedlichen Sprechakten entsprechen. Beispiele für solche Verben sind *auffordern*, *informieren* und *versprechen* [81]. Außerdem hat Austin drei verschiedene Aspekte der Sprechakts ausgezeichnet:

- *Lokutionärer Akt* (*engl. locutionary act*) - Hiermit ist das bloße Aussprechen als Sprechakt gemeint (z.B. „Bitte mache mir etwas Tee“).
- *Illokutionärer Akt* (*engl. illocutionary act*) - Eine Aktion wird ausgeführt, indem etwas ausgesprochen wurde. Der Akt besteht in der Verwendung des Gesagten (z.B. „Er hat mich gebeten (bzw. aufgefordert) etwas Tee zu machen“).

[48] Dieses wurde in 1970-1980er Jahren intensiv untersucht [81].

[49] Methoden und Verfahren zur Synchronisation mehrerer Prozesse können der Fachliteratur für verteilte Systeme entnommen werden, wie z.B. [59] oder [12].

[50] Unter performativen Verben (*engl. performative verbs*) werden solche Verben verstanden, die implizit eine Handlung vollziehen.

- *Perlokutionärer Akt* (*engl. perlocution*) - das Erzielen einer Wirkung, die über den illokutionären Akt hinausgeht, wie beispielsweise Überzeugen, Umstimmen, Kränken, Beleidigen etc. (z.B. „Er hat mich überzeugt etwas Tee zu machen“).

[81], [79].

Austin erkannte drei wichtige Bedingungen, die erfüllt werden müssen, damit ein Performativ erfolgreich abgeschlossen wird [79]:

1. Es muss eine konventionelle Prozedur für den Performativ geben und die Umstände und Personen müssen in dieser Prozedur spezifiziert sein.
2. Die Prozedur muss korrekt und vollständig ausgeführt werden.
3. Der Akt muss aufrichtig sein, und daraus resultierende Verpflichtungen müssen erfüllt sein, sofern es möglich ist.

Die Arbeit von Austin wurde von John Searle 1969 vervollständigt. Searle hat mehrere Bedingungen identifiziert, die eingehalten werden müssen, damit ein Sprechakt erfolgreich zu Stande kommt:

- **Normale E/A Bedingungen** - Diese Bedingungen versichern, dass der Hörer in der Lage ist, Aufforderungen wahrzunehmen.
- **Vorbereitende Bedingungen** - Die vorbereitenden Bedingungen geben an, welche Gegebenheiten gelten müssen, damit der Sprecher den Sprechakt korrekt auswählt. In diesem Fall muss der Hörer in der Lage sein Aktionen auszuführen, und der Sprecher muss glauben, dass der Hörer in der Lage ist Aktion auszuführen. Es darf nicht offensichtlich sein, dass der Hörer Aktionen ohnehin ausführt.
- **Aufrichtigkeitsbedingungen** - Diese Bedingungen charakterisieren die aufrichtigen Aufforderungen. Eine nicht aufrichtige Aufforderung kann dann auftreten, wenn der Sprecher nicht will, dass eine bestimmte Aktion ausgeführt wird.

Searle hat außerdem versucht mögliche Typen der Sprechakte zu klassifizieren. Diese Type charakterisieren solche Sprechakte, mit denen es möglich ist:

- dem Hörer wahre Informationen zu geben (**Repräsentativa**),
- den Hörer zu einer bestimmten Aktion zu bewegen (**Direktiva**),
- den Sprecher zu einem bestimmten Verhalten zu verpflichten (**Kommissiva**),
- den psychologischen Zustand auszudrücken (**Expressiva**) und
- eine Änderung eines institutionellen Zustandes herbei zu rufen (**Declarativa**).

Weitere Informationen über Sprechakten und deren Verwendung (z.B. in Planungssystemen oder als rationale Aktionen) können der Fachliteratur, wie z.B. [81] oder [79], entnommen werden.

Die Sprechaktentheorie hat die Entwicklung der Agenten-Kommunikationssprachen beeinflusst. So finden sich die wichtigsten Ideen dieser Theorie in der Sprachen *KQML* und *FIPA ACL* wieder, die im Folgenden vorgestellt werden.

KQML In den frühen 1990er wurde im Auftrag der amerikanischen Defense Advanced Research Projects Agency (DARPA) das Konsortium Knowledge Sharing Effort (KSE), mit dem Ziel Protokolle für den Austausch vom repräsentierten Wissen zwischen autonomen Information-Systemen zu entwickeln, gegründet. Das Ergebnis der Forschung von KSE sind *KIF* und *KQML*.

KIF (*Knowledge Interchange Format*) ist eine Sprache zur Beschreibung von Wissen. Sie ist stark an die Prädikatenlogik der ersten Stufe angelehnt. KIF besitzt eine Grundmenge von Operatoren wie z.B. `and`, `or`, `not`, `forall` und `exists`. Mit dieser Sprache lassen sich beispielsweise folgende Eigenschaften einer Domäne beschreiben [81]:

- Eigenschaften der Dinge (z.B. „Michael ist Vegetarier“)
- Beziehungen zwischen den Dingen (z.B. „Michael und Janine sind verheiratet“)
- Allgemeine Eigenschaften (z.B. „Jeder hat eine Mutter“)

Listing 2.2 zeigt ein Beispiel für die des Junggesellen in KIF. Diese Definition besagt, dass ein Junggeselle ein Mann ist, der nicht verheiratet ist.

Listing 2.2: Definition von Junggesellen in KIF [81]

```
(defrelation bachelor (?x)) :=
  (and (man ?x))
    (not (married ?x)))
```

KIF, eine Sprache zur Beschreibung des Wissens, kann zur Ontologie-Beschreibung verwendet werden.

KQML (*Knowledge Query and Manipulation Language*) ist eine nachrichten-basierte Sprache für Agenten-Kommunikation. KQML definiert ein gemeinsames Format für Nachrichten. Jede Nachricht hat ein Performativ (siehe Sprechakttheorie) und eine Zahl von Parameters, die als Attribut/Wert-Paare repräsentiert sind.

Listing 2.3: Beispiel-Nachricht in KQML [81]

```
(ask-one
  :content  (PRICE IBM ?price)
  :receiver  stock-server
  :language  LPROLOG
  :ontology  NYSE-TICKS
)
```

Listing 2.3 zeigt eine Beispiel-Nachricht in KQML. Diese Nachricht repräsentiert eine Anfrage nach dem Preis der IBM-Aktie. Der Performativ ist `ask-one`, der als eine Frage eines Agenten an einen anderen Agenten verstanden wird, auf die exakt eine Antwort erwartet wird. Andere Komponenten dieser Nachricht sind ihre Attribute. Das wichtigste ist hier das Attribut `:content`, das den Inhalt der Nachricht spezifiziert. In diesem Fall wird nach dem Aktien-Preis gefragt. Das `:receiver` Attribut bestimmt den Empfänger der Nachricht und `:language` die Sprache, in der `:content` der Nachricht verfasst ist. Damit der Empfänger die Terminologie der Nachricht zuordnen kann, wird diese mit `:ontologie` (vgl. KIF und Kapitel 2.2.2) eingegeben. Weitere mögliche Parameter können [81] gefunden werden.

Es gibt mehrere Implementierungen dieser Sprache, die bereits erfolgreich eingesetzt wurden [81]. Jedoch wurde KQML stark kritisiert. Die Haupt-Nachteile von KQML sind [81]:

- Die Performativen wurden nicht genau spezifiziert, sodass verschiedene KQML-basierte Implementierungen nicht interoperabel sind.
- Die Transport-Art für KQML-Nachrichten wurde ebenfalls nicht präzise definiert, deswegen blieben die KQML-sprachigen Agenten inkompatibel zu einander.
- Die Semantik der KQML wurde nicht definiert, sodass nicht feststellbar ist, ob ein Agent diese Sprache richtig verwendet.
- Die Sprache vermisst die Klasse der Kommissiva (vgl. Sprechakttheorie), d.h. die Performativen für die Koordination der Agenten sind nicht spezifiziert.
- Die Menge der Performative in KQML ist sehr groß, die vermutlich ad hoc entstanden ist.

Die Kritik hat zur Entwicklung einer Sprache geführt, in der die Nachteile der KQML vermieden werden. Diese wird im Weiteren vorgestellt.

FIPA ACL 1995 begann die *Foundation for Intelligent Physical Agents* (*FIPA*) die Entwicklung von Standards für agentenbasierte Systeme. Das Gremium entwickelt Standards für Agentenkommunikation, Agentenmanagement und den Transport von Agentennachrichten. Diese Standards und aktuelle Forschungen können unter [25] eingesehen werden.

Die Sprache für die Agentenkommunikation, die von FIPA entwickelt wurde, ist *ACL* (*Agent Communication Language*) und sehr stark an KQML angelehnt. Sie definiert 20 Performative[51] (wie z.B. `inform`) zur Interpretation der Nachrichten.

Listing 2.4: Beispiel-Nachricht in FIPA ACL [81]

```
(inform
  :sender  agent1
  :receiver  agent2
  :content  (price good2 150)
  :language  sl
  :ontology  hpl-auction
)
```

Listing 2.4 präsentiert eine Beispiel-Nachricht in FIPA ACL. Mit dieser Nachricht informiert (vgl. `inform`) Agent `agent1` den Agenten `agent2` darüber, dass der Preis für `good2` 150 Geldeinheiten ist. Mit dem Attribut `:ontologie hpl-action` weiß der `agent2`, um welche Ware es sich handelt. Der Inhalt Nachricht (siehe `:content`) ist in der Sprache SL eingegeben.

Wie aus dem Listing 2.4 zu entnehmen ist, ähnelt die Syntax der FIPA ACL der KQML-Sprache. Die semantische Interpretation der Nachricht, die bei dem KQML-Modell fehlt, wird mit Hilfe der Sprache SL definiert. Diese Sprache erlaubt es, „Vorstellungen, Wünsche und unsichere Vorstellungen“ der Agenten zu beschreiben [81]. Somit können mit FIPA ACL nicht nur Fragen an die Agenten gestellt werden, sondern auch die momentanen Zustände anderer Agenten mitgeteilt werden.

FIPA ACL wurde am 06.12.2002 standardisiert und kann unter [25] eingesehen werden. Es gibt bereits Implementierungen dieser Sprache. Eine der meistverbreiteten ist *JADE* (*Java Agent Development Environment*). Diese Middleware erlaubt den Java-Entwicklern FIPA agentenbasierte Systeme mit einem geringerem Aufwand zu implementieren.

Eine detailliertere Behandlung der Multi-Agenten-Systeme und der hiermit verbundenen Themen können der Fachliteratur wie z.B. [29], [7], [24] oder [81] entnommen werden.

[51] Die komplette Liste der Performative können bei [81] oder [25] eingesehen werden.

2.3.3. Blackboard-Modell

Das Blackboard-Modell hat ihren Ursprung im Hearsay-II-Spracherkennungssystem [23]. Dieses System wurde in den Jahren 1971-1976 von Erman et al. an der Carnegie-Mellon University entwickelt. Dessen Aufgabe lag in der Erkennung natürlichsprachiger Anfragen an eine Literaturdatenbank.

Wie der Name des Modells bereits verrät, liegt die Metapher *Blackboard*[52] diesen Systemen zugrunde. Das Prinzip des Blackboard-Modells ist das folgende: Eine Gruppe von Spezialisten, von denen jeder für einen bestimmten Fachbereich kompetent ist, lösen kooperativ eine Aufgabe. Hierzu wertet jeder Spezialist aktuelle Lösungselemente und Lösungshypothesen des Blackboard aus und fügt neue Elemente und Hypothesen dem Blackboard hinzu. Dadurch werden andere Spezialisten aktiv.

Die Spezialisten werden im Blackbard-Modell als *Wissensquellen* (*WQ*) und das Blackboard als *Blackboard-Datenstruktur* bezeichnet. Die Wissensquellen sind funktional unabhängige Module, die das Wissen für die Lösung eines Problems besitzen. Das Blackboard verwaltet die Daten zum Zustand der Problemlösung, die von den Wissensquellen verändert bzw. erzeugt werden. Es ist einziges Kommunikationsmedium der Wissensquellen.

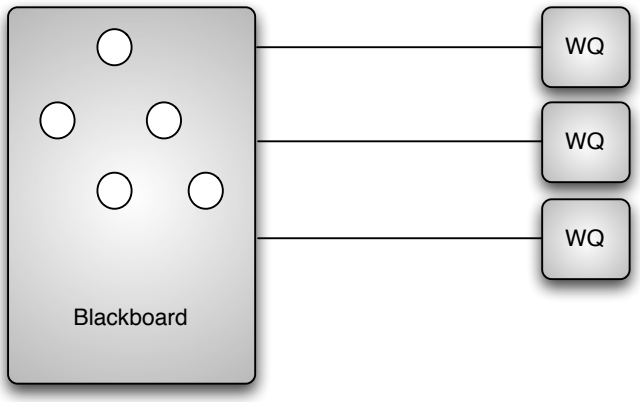


Abbildung 2.10.: Klassisches Blackboard-System

Die Abbildung 2.10 stellt ein klassisches Blackboard-System dar. Die Wissensquellen sind für ihre Aktivierung selbst verantwortlich. Sie bestehen aus zwei Teilen: *Spezifikationsteil* und *Aktionsteil*. Der Spezifikationsteil enthält die Angaben darüber, wann die Wissensquelle anwendbar ist, wobei der Aktionsteil die eigentliche Verarbeitung ausführt. Dieses Modell weist einige Nachteile auf. So kann es unter Umständen dazu kommen, dass das Blackboard überfüllt wird, wenn veraltete Daten nicht gelöscht werden oder sehr viele Daten identischen

[52] engl. *Schultafel*

Typs abgelegt werden. Inkonsistente Zustände des Gesamtsystems sind auch möglich, wenn z.B. ein Agent einen Datensatz ändert, der von einem anderen Agenten in seiner aktuellen Berechnung verwendet wird.

Eine Erweiterung des klassischen Blackboard-Modells um eine Steuerungskomponente[53] vermeidet das Problem der Inkonsistenz (siehe Abbildung 2.11). Die Steuerungskomponente führt die Aktivierung der Wissensquellen durch und überwacht deren Ausführung [10] .

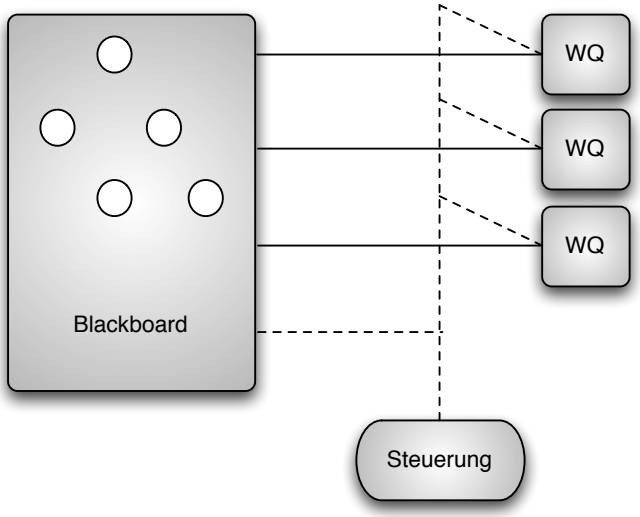


Abbildung 2.11.: Blackboard-System mit einer Steuerungskomponente (BB1)

Es existieren verschiedene Erweiterungen (bzw. Abwandlungen) des klassischen Blackboard-Modells. So werden beispielsweise weitere Komponenten eingeführt, die die Verwaltung und Überwachung von Wissensquellen und des Gesamtsystems übernehmen, während eine andere Abwandlung dafür sorgt, dass die Struktur des Blackboard unterschiedliche Ausprägungen aufnimmt[54]. Für eine ausführliche Beschreibung all dieser Modellen wird an dieser Stelle auf die Fachliteratur verwiesen wie z.B. [23], [10] oder [29]

[53] Dieses Blackboard-Modell ist unter dem Namen *BB1* bekannt.

[54] Manche Blackboard-Modelle sehen z.B. Blackboards vor, in denen die Datenstruktur einen hierarchischen Charakter aufweist. Verschiedene Blackboads, die Informationen unterschiedlicher Natur verwalten, sind auch denkbar.

3. Analyse

3.1. Intelligentes Haus in der Entwicklung

Der Wunsch nach einem intelligenten Haus, das das Wohnen in dessen sowohl einfacher als auch interessanter gestaltet, ist inzwischen weit verbreitet. So werden solche Systeme nicht nur in den Forschungsarbeiten der Wissenschaftler beschrieben, sondern sie werden bereits in realen Umgebungen aufgebaut und getestet. Zwar haben diese Versuche in erster Linie einen wissenschaftlichen Charakter[1], sie bieten jedoch eine gute Grundlage um Anwendern einen Einblick in solche Systeme zu geben.

Umgebungen, in denen solche Systeme entwickelt und getestet werden, sind vor allem unter dem Namen *Living Lab* bekannt. Diese Labore bieten alle nötigen Voraussetzungen, um Anwendungen[2] zu entwickeln und sie unter realen Bedingungen zu testen. Nicht selten können sich in Rahmen der Testdurchführungen neue Lösungsansätze oder sogar neue Anwendungen (bzw. Dienste) ergeben. Unter [41] sind eine Reihe dieser Labore gelistet.

Die meisten Anwendungen, die unter [41] genannt sind, stellen Lösungen für bestimmte in sich abgeschlossene Probleme dar. Somit können sie nur als Teilaspekte des intelligenten Hauses angesehen werden. Die Idee des intelligenten Hauses beinhaltet aber eine Reihe solcher Anwendungen, die zusammen eine Gesamtheit bilden. Diese einzelne Anwendungen müssen nicht autark voneinander laufen, sondern können miteinander kooperieren und somit einen höheren Mehrwert darstellen. Zurzeit existieren noch nicht viele derartige Labore (bzw. Systeme), die ein intelligentes Haus (bzw. intelligente Wohnung) mit allen seinen (bzw. ihren) Anwendungen als Ziel verfolgen. Beispielhaft seien genannt: *Philips HomeLab*, *T-Com Haus*, *inHaus* und *BAALL*.

3.1.1. Philips HomeLab

Philips HomeLab ist ein technisches Labor in Eindhoven (Niederlande). Es ist ein vollfunktionsfähiges Haus, in dem Philips seine Geräte-Prototypen und „Anwendungen der Zukunft“

[1] Meistens werden in solchen Umgebungen verschiedene Konzepte unter realen Bedingungen erprobt. Sie dienen u.a. der Erforschung und Verbesserungen der ergonomischen Aspekte solcher Systeme.

[2] Hier sind vor allem solche Anwendungen gemeint, die in einem intelligenten Haus installiert werden können.

testet und erforscht. Das Haus ist mit Videokameras und Mikrofonen versehen, sodass die Probanden rund um die Uhr überwacht werden können. Dies erlaubt den Forschern die Bedürfnisse und Motivation zur Benutzung neuer Technologien der „Bewohner“ besser zu verstehen.

Abbildung 3.1 zeigt als Beispiel ein Gerät-Prototyp aus diesem Labor. Es handelt sich dabei um einen Spiegel, der mit Display-Funktionalität ausgestattet ist. Unterschiedliche Anwendungen können so mit dem Anwender interagieren. So kann beispielsweise der Benutzer die Wettervorhersage oder den Börsenkurs jederzeit abfragen. Mit der neuen Anwendung, als „Double Vision“ bezeichnet, kann der Bewohner sich sogar von hinten betrachten.

Abbildung 3.1.: Philips HomLab: interaktiver Spiegel [50]

Mit HomeLab hofft Philips den Zyklus der Produktentwicklung zu beschleunigen. Weitere Informationen zu diesem Labor und dessen Anwendungen finden sich unter [50].

3.1.2. T-Com Haus

T-Com Haus ist ein gemeinsames Projekt mehrerer Unternehmen wie T-Com, WeberHaus, Siemens und Neckermann [69]. Das T-Com Haus ist ein reales Haus (siehe Abbildung 3.2), in dem eine Vielzahl elektronischer Geräte eingebaut sind. Sie sind miteinander vernetzt und lassen sich zentral bedienen (bzw. steuern). Die meisten Anwendungen, die hier installiert sind, unterstützen die Haushaltsführung und bieten Features aus dem Multimedia-Bereich an. So kann beispielsweise der Ofen per Handy angemacht, um ihn rechtzeitig vorzuheizen, oder die Waschmaschine auf „Schleudern“ geschaltet werden, während der Anwender im Laden einkauft. Für ein besseres Kinoerlebnis wird dank Mood Management eine passende Beleuchtung geschaltet. Weitere Anwendungen und allgemeine Informationen zu T-Com Haus bietet [69].

Abbildung 3.2.: T-Com Haus: Eingangsbereich [69]

3.1.3. inHaus

inHaus[3] ist ein Labor der Fraunhofer-Gesellschaft. Dieses Labor wird von neun Fraunhofer-Instituten und über 80 Wirtschaftspartnern für neue Technologie-, Produkt- und Anwendungslösungen in Wohn- und Nutzimmobilien unterstützt [27]. Es ermöglicht einzelnen Entwicklern oder Unternehmen, die über kein großes Budget verfügen, ihre Anwendungen (bzw. Systeme) unter realen Bedingungen zu testen. Hierfür stellen die Fraunhofer Institute zwei Anlagen zur Verfügung:

- *inHaus1* zum Testen der Anwendungen für Wohnimmobilien (siehe Abbildung 3.3(a)) und
- *inHaus2* zum Testen der Anwendungen für Nutzimmobilen (siehe Abbildung 3.3(b)).

[3] Das Präfix *in* steht für *intelligent* [27].

Mit der inHaus1-Anlage wurde am 3.4.2001 das gesamte inHaus-Zentrum eröffnet. Hier entstehen neue Technologie- und Anwendungslösungen für private Wohnhäuser aller Art und für Immobilien der Wohnwirtschaft. Einige Anwendungen, die im inHaus1 entwickelt und erprobt worden sind, finden mittlerweile in der Praxis eine erfolgreiche Anwendung [27].

Das inHaus2 wurde am 5.11.2008 in Betrieb genommen. In dieser Anlage wird an den Komponenten und Systemen für intelligente Räume und Gebäude der nächsten Generation gearbeitet. Hierfür werden verschiedene Labor- und Werkstattbereiche genutzt, in denen Anwendungen für beispielsweise Hotel-, Hospital- und Bürobereiche entwickelt werden.

(a) inHaus1: Smart Home

(b) inHaus2: Smart Building

Abbildung 3.3.: inHaus-Anlagen [27]: a) inHaus1, b) inHaus2

Die beiden Anlagen dienen nicht nur der technischen Entwicklung, sondern auch der Optimierung bereits existierender Lösungen. Ausführlichere Informationen zu inHaus finden sich unter [27].

3.1.4. BAALL

BAALL steht für (*Bremer Ambient Assisted Living Laboratory*) und ist ein Labor, in dem Anwendungen zur Erleichterung des Alltags älterer, kognitiv und physisch eingeschränkter Menschen entwickelt werden. Es wird in Kooperation mit der Universität Bremen im EU-

Projekt SHARE-it[4] und im SFB/TR8 Spatial Cognition[5] der Deutschen Forschungsgemeinschaft[6] geführt [3].

Das Labor wird durch eine alters- und behindertengerechte Wohnung repräsentiert. In dieser Wohnung sind technische Geräte installiert, die unauffällig (bzw. unsichtbar) für den Anwender sind. Mit Hilfe dieser werden Anwendungen entwickelt, die den eingeschränkten Menschen bei ihren alltäglichen Tätigkeiten helfen sollen. Abbildung 3.4 zeigt beispielsweise eine Küchenzeile, deren Oberschränke ab- und wieder aufwärts fahren können.

Abbildung 3.4.: BAALL: Küchenzeile [40]

Da das Labor wie eine Wohnung gebaut ist, können Probanden hier mehrere Tage wohnen um die installierte Technik zu erproben bzw. zu beurteilen. Weitere Informationen über BAALL können [3], [17] oder [40] entnommen werden.

3.1.5. Fazit

Die hier vorgestellten Labore verfolgen unterschiedliche Ziele. Während die Aufgaben des Philips HomeLab die Entwicklung und Erprobung einzelner Geräte für den Bereich intelli-

[4] *SHARE-it (Supported Human Autonomy for Recovery and Enhancement of cognitive and motor abilities using information technologies)* ist ein EU-Projekt, dessen Ziel es ist skalierbare und anpassungsfähige Systeme zur Erweiterung Sensor- und anderen Technologien zu entwickeln, die im Bereich intelligenter Häuser eingesetzt werden können [56].

[5] *SFB/TR 8 Spatial Cognition* ist ein Forschungszentrum, dessen Aufgabe darin besteht, menschenorientierte Systeme zur Umgebungsassistenz zu erforschen (bzw. zu entwickeln) [55].

[6] „Die *Deutsche Forschungsgemeinschaft (DFG)* ist die zentrale Selbstverwaltungseinrichtung der Wissenschaft zur Förderung der Forschung an Hochschulen und öffentlich finanzierten Forschungsinstitutionen in Deutschland“ [16].

gente Häuser sind, versuchen Telecom, Weber und weitere Unternehmen mit dem T-Com Haus ihre Produkte den Endbenutzern näher zu bringen. Hiermit soll das Interesse des potenziellen Kunden geweckt und somit die Kaufbereitschaft dieser Produkte erhöht werden.

Das Labor inHaus stellt lediglich die räumlichen und technischen Mittel zur Verfügung, um den Prozess der Erforschung und Entwicklung der Anwendungen (bzw. Systeme) für intelligente Häuser zu unterstützen. Somit ist es kein reines Living Lab, in dem ein Zielsystem kontinuierlich entwickelt und verfeinert wird. Das Labor BAALL ist hingegen ein echtes Living Lab. Sein Schwerpunkt liegt jedoch nicht in der Entwicklung eines Systems für ein Haus (bzw. eine Wohnung) einer durchschnittlichen Familie. Die Zielpersonen von BAALL sind ältere oder behinderte Menschen, deren Alltag durch ihre körperliche Einschränkungen erschwert ist. Hier werden daher AAL-artige Systeme (bzw. hierzu gehörige Anwendungen) entwickelt.

3.2. HAW Living Lab

Die *Hochschule für Angewandte Wissenschaften Hamburg (HAW Hamburg)* besitzt zwei Labore für die Forschung, Entwicklung und Testdurchführung im Bereich *intelligentes Haus*. Im Gegensatz zu den im Kapitel 3.1 vorgestellten Living Labs, ist der Zweck der HAW Labore ein einheitliches System zu entwickeln, das in einem intelligenten Haus eingesetzt werden kann. Als Zielgruppe wird in diesem Zusammenhang ein Ein-Person-Haushalt aber auch eine durchschnittliche Familie angesehen. Die Entwicklung soll einen kontinuierlichen Charakter aufweisen, sodass das System im Laufe des Entwicklungsprozesses immer weiter verfeinert und verbessert werden kann. Die Labore stellen räumliche Umgebungen zur Verfügung und sind mit technischen Mitteln ausgestattet, die die Verfolgung des genannten Ziels unterstützen. Im Weiteren werden die Labore genauer beschrieben.

3.2.1. iFlat

Das erste Labor, das für die Entwicklung des intelligenten Hauses (bzw. deren Anwendungen) von der HAW bereit gestellt wurde, trägt den Namen *iFlat*[7]. Es befindet sich Berliner Tor 7, Raum 1081 und hat 8 bis 10 Arbeitsplätze für die Entwickler. Dieses Labor ist ein reines Entwicklungslabor für Anwendungen und Systeme in intelligenten Häusern. Aufgrund der knappen zur Verfügung stehenden räumlichen Ressourcen[8] ist die Erprobung der entwickelten Anwendungen (bzw. Systeme) nur bedingt möglich. Im zweiten Labor (siehe Kapitel

[7] *iFlat* steht für *intelligente Wohnung*.
[8] siehe Abbildung 3.5

3.2.2) sollen ausführliche und unter realistischen Bedingungen stattfindende Tests durchgeführt werden.

Abbildung 3.5.: Das iFlat-Labor der HAW

Die Entwicklung einzelner Anwendungen für intelligente Häuser werden in diesem Labor nicht mit realen Geräten realisiert, sondern simuliert[9]. Jedes Gerät wird durch einen Rechner repräsentiert, der die Eigenschaften (bzw. Dienste) des jeweiligen Gerätes simuliert (bzw. anbietet). So stehen folgende Geräte bzw. deren Simulationen in iFlat zur Verfügung:

- *Fernseher*
  Ein Fernseher wird durch einen Monitor und einen Mac mini[10] mit einer installierten TV-Karte und einem entsprechenden Programm simuliert. An dieser Stelle wird auf eine Masterarbeit verwiesen [20], im Rahmen derer ein TV System für ein intelligentes Haus entwickelt wurde.

[9] Die Simulation ist nur dann sinnvoll, wenn sie die realen Geräte (bzw. Umgebungen) imitiert, sodass diese später ohne erhebliche Probleme durch reale Geräte ersetzt (bzw. in die realen Umgebungen integriert) werden kann. Eine Simulation ist meistens eine bessere Wahl als die Entwicklung unter den realen Bedingungen. Der Vorteil liegt darin, dass somit die Entwicklungskosten gesenkt werden können. Hier werden Geräte simuliert, die noch nicht existieren bzw. nur sehr aufwendig zu beschaffen sind.

[10] *Mac mini* ist ein Rechner des Herstellers Apple (http://www.apple.com/de/macmini/).

- *Stereo-Anlage*
  Dieses Gerät wird durch einen Mac mini mit einer eigebauten Sound-Karte, an der Lautsprecher angeschlossen sind, repräsentiert.

- *Smart:Shelf*
  Ein echtes Regal, das eine RFID-Antenne[11] besitzt, sodass mittels des an die Antenne angeschlossenen RFID-Readers[12] die Gegenstände[13] im Regal identifiziert werden können (siehe Abbildung 3.6). Dieser RFID-Reader wird von einem Mac mini gesteuert. Die Anwendung[14] für die Identifikation der Gegenstände wurde im Rahmen eines Projektes von HAW-Masterstudenten entwickelt [66].

- *Kühlschrank*
  Ein Kühlschrank wird durch ein weiteres Samart:Shelf repräsentiert.

- *Door bell*
  Door bell ist eine Türklingel mit einer Video-Kamera. Es wird durch eine Anwendung simuliert. Diese Anwendung läuft auf einem Mac mini, an dem eine Webcam[15] angeschlossen ist. Der Knopf der Klingel selbst wird durch einen Button in der GUI[16] der Anwendung repräsentiert.

- *Indoor Location Detector*
  Indoor Location Detector ist eine Anwendung zur Ermittlung der Position einer Person (oder auch eines beliebigen Gegenstandes) in geschlossenen Räumen. Hiermit kann beispielsweise der aktuelle Aufenthalt des Benutzers im Haus geortet werden. Die eigentliche Location-Ermittlung erfolgt mit Hilfe von vier UWB-Sensoren[17], die in einem bestimmten Abstand voneinander aufgestellt sind[18], und eines aktiven UWB-Tags[19] (siehe Abbildung 3.7(c)), der von diesen Antennen registriert wird. Die Anwendung

[11] Hier wird *IC SA70 Standard Antenne* von Frosch Electronics (www.froschelectronics.com) verwendet.

[12] Bei dem RFID-Reader handelt es sich um einen *Philips i.CODE*-kompatiblen Test-Reader, der ebenfalls von Frosch Electronics zur Verfügung gestellt wurde.

[13] Diese Gegenstände müssen hierfür einen passiven RFID-Tag besitzen. Solche Tags brauchen keine externe Stromquelle, da sie die Energie von den Antennen bekommen. Sie werden meistens in Form eines Aufklebers gebaut (siehe Abbildung 3.6), sodass sie auf einer einfachen Weise an die Gegenstände angebracht werden können.

[14] Diese Anwendung trägt den Namen *Smart:Shelf* [66].

[15] *Webcam* ist eine Kamera, die in bestimmten Zeitabständen Bilder macht, die direkt übers Internet verschickt oder auf einer Seite des World Weib Web (WWW) angezeigt werden können.

[16] GUI steht für *graphical user interface* und ist eine graphische Benutzeroberfläche für die Interaktionen des Programms mit dem Anwender.

[17] *UWB (ultra-wideband)* ist eine RF (radio frequency) ähnliche Technologie, die eine großräumigere und präzisere Location-Ermittlung ermöglicht als die herkömmliche RFID-Technologie [63].

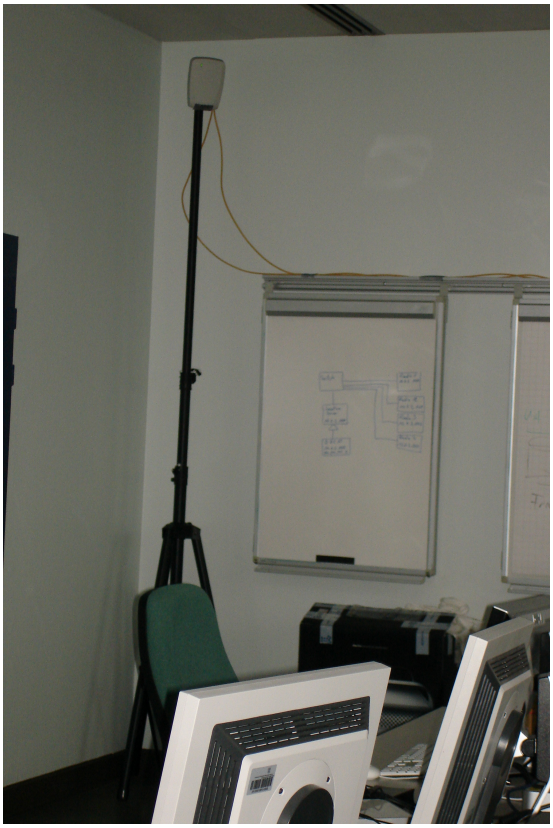
[18] Im iFlat-Labor sind sie in den Ecken des Raumes aufgestellt (siehe Abbildungen 3.7(a) und 3.7(b)).

[19] Ein aktiver UWB-Tag zeichnet sich dadurch aus, dass er die Identifikation einleitet [63]. Somit benötigen solche Tags eine externe Energie-Quelle (z.B. eine Batterie).

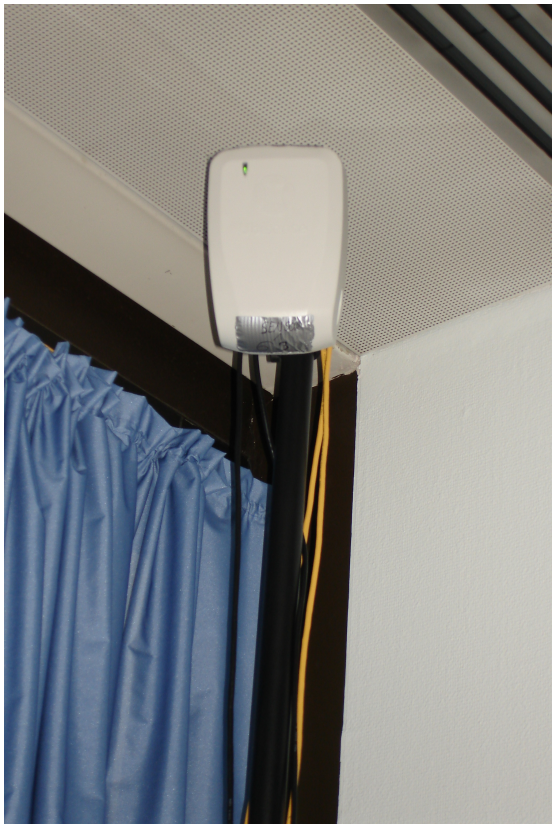
hierfür wurde von den wissenschaftlichen Mitarbeitern der HAW geschrieben und läuft auf einem separaten Mac mini.

Abbildung 3.6.: Foto des Smart:Shelf des iFlat-Labors

Die beschriebenen Geräte bilden eine Basis für die möglichen Anwendungen im Bereich der intelligenten Häuser. Das Equipment des Labors ist selbstverständlich nicht fest, sodass weitere Anschaffungen neuer Geräte denkbar und wünschenswert sind, um beispielsweise neue Anwendungen hervorzubringen oder die bestehende Anwendungen (bzw. Systeme) um diese Geräte zu erweitern. Im weiteren Verlauf dieser Arbeit wird ein Beispielsszenario beschrieben (siehe Kapitel 3.4), das auf den bereits vorgestellten Geräten aufbaut.



(a)



(b)

(c)

Abbildung 3.7.: Geräte des Indoor Location Detector: a) und b) UWB-Antenne, c) UWB-Tag

3.2.2. Living Place Hamburg

Das zweite Labor der HAW hat den Namen *Living Place Hamburg* und wird z.T. von der Hamburger Wirtschaftsbehörde finanziert. Es befindet Berliner Tor 11 (siehe Abbildung 3.8). Eine der wichtigsten Aufgaben dieses Labors ist die Testdurchführung der bereits entwickelten Anwendungen (bzw. Systeme) aus dem Bereich intelligentes Haus. Das Labor ist zum Zeitpunkt des Schreibens dieser Arbeit noch nicht fertig gestellt. Der zukünftige Aufbau des Labors ist jedoch bereits bekannt. Es wird zwei Räume haben (siehe Abbildung 3.9), in denen die Anwendungen (bzw. Systeme) gesteuert und getestet werden:

- Raum 210: Kontroll- bzw. Steuerraum und
- Raum 204: Wohnung.

(a) Außenansicht von links

(b) Außenansicht von rechts

Abbildung 3.8.: Außenansicht des HAW-Labor Living Place Hamburg

Der Kontrollraum ist für die Steuerung und Überwachung der Anwendungen (bzw. Systeme) und für die Beobachtung der Testpersonen zuständig. Hierzu werden Bildschirme installiert, die die einzelnen Bereiche der Labor-Wohnung anzeigen. So können die Probanden beobachtet, und damit Rückschlüsse über die Eigenschaften der Anwendung (bzw. des Systems), wie beispielsweise Bedienbarkeit und andere ergonomische Merkmale, gezogen werden. Während der Beobachtung können nicht nur Verbesserungsvorschläge zur jeweiligen Anwendungen gewonnen werden, sondern auch Ideen für neue Anwendungen entstehen. Somit ist der Kontrollraum ein wichtiger Bestandteil des Labors.

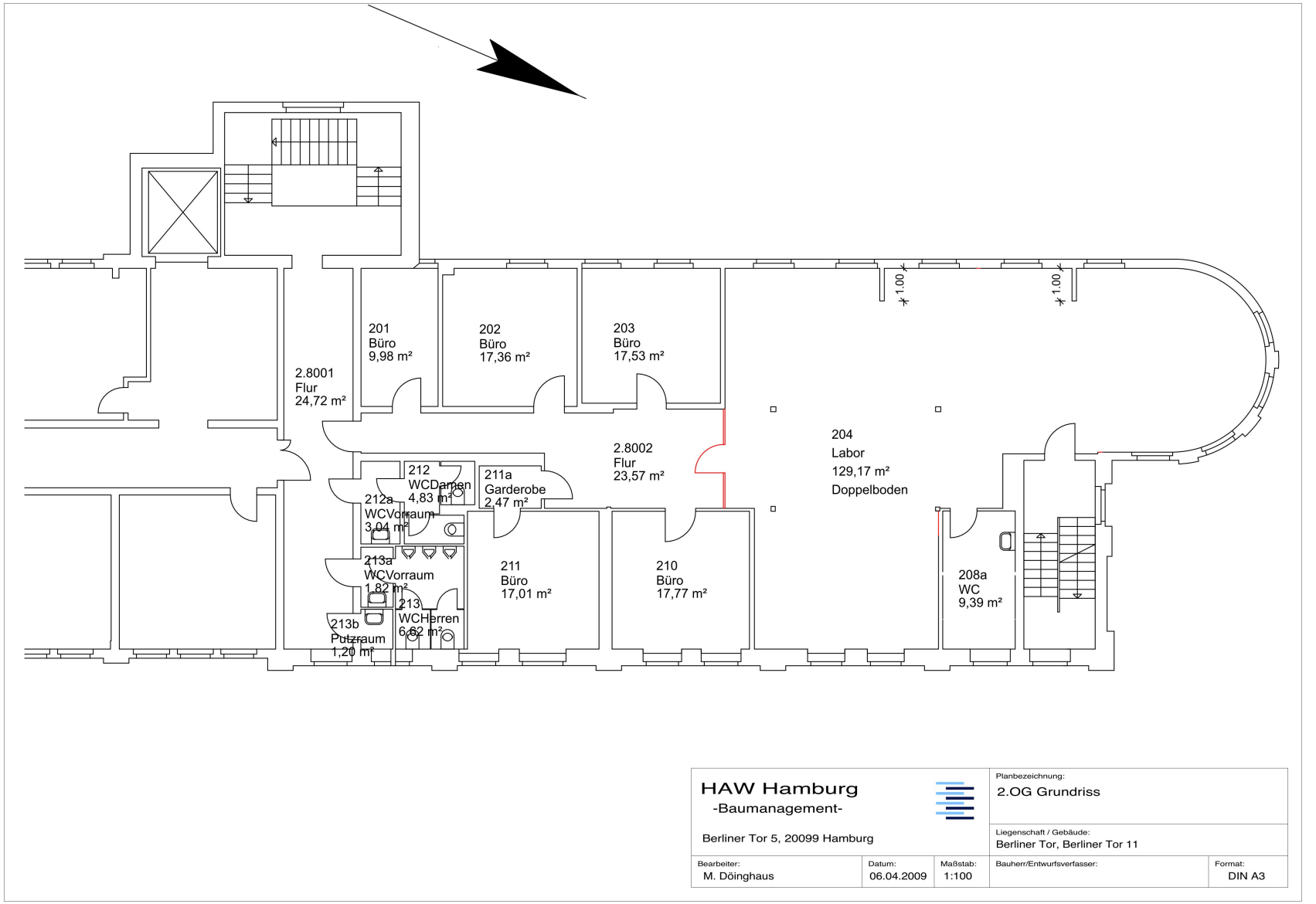


Abbildung 3.9.: Plan des Labor Living Place Hamburg (Quelle: HAW Hamburg)

Die Labor-Wohnung repräsentiert einen Wohnbereich, der in seinem Aufbau einer Wohnung ähnelt. Hier können recht realistische Bedingungen für die Testdurchführung der Anwendungen für intelligente Häuser hergestellt werden. Die Abbildung 3.10 zeigt das Baumodell dieses Labors. Die Wohnung wird folgende Räume (bzw. Bereiche) aufweisen:

- Wohnzimmer
- Schlafzimmer
- Flur
- Küche
- Essbereich
- Bad

Abbildung 3.10.: Das Baumodell des HAW-Labor Living Place Hamburg

Die Wohnung wird mit echten Möbeln und anderen Wohnungsutensilien ausgestattet sein, um den Grad der Authentizität dieser Wohnung zu erhöhen. In jedem Zimmer (bzw. Wohnbereiche) werden Mikrofone und Videokameras aufgestellt, mit deren Hilfe die Probanden

vom Kontrollraum aus beobachtet werden können. Die Labor-Wohnung ist somit ein zentraler Bereich dieses Labors, der nicht nur der Durchführung der Tests dienen wird, sondern sie stellt eine Präsentationswohnung dar, in der installierte Anwendungen Besuchern vorgeführt werden können.

3.3. Intelligentes Haus als Gesamtsystem

Die in Kapitel 3.1 vorgestellten Labore entwickeln Anwendungen für intelligente Häuser. Ihre bereits realisierten Lösungen können jedoch nicht oder sehr bedingt als vollständige Systeme für intelligente Wohnumgebung gesehen werden. Sie stellen eher Insel-Lösungen für bestimmte Probleme aus diesem Umfeld dar. In diesem Kapitel werden Arbeiten präsentiert, deren Ziel darin besteht, vollwertige Systeme zu entwickeln, die in intelligenten Häusern eingesetzt werden können.

3.3.1. CAMUS

CAMUS (*Context-aware Middleware for Ubiquitous Computing Systems*) ist ein Projekt der koreanischen Universität *Kyung Hee*[20]. Das Ziel dieses Projektes war eine Middleware zu entwickeln, mit deren Hilfe sich Context-Aware Systeme relativ einfach entwickeln lassen. So stellt CAMUS eine Zahl von Werkzeugen zur Verfügung, die es ermöglichen, die Umgebung zu erfassen, die somit gewonnenen Daten zu analysieren und anhand des Analyseergebnisses eine resultierende Aktion auszuführen.

Die Arbeitsweise von CAMUS ist wie folgt: Die von den Sensoren registrierten Rohdaten werden in das gewünschte Format transformiert. Anschließend werden sie analysiert und zu einem Kontext zusammengefasst. Anhand des gewonnenen Kontextes werden adäquate Aktionen generiert und anschließend ausgeführt. Dies wird aus der konzeptuellen Architektur von CAMUS deutlich (siehe Abbildung 3.11).

Zum Testen dieser Middleware wurde eine Beispielanwendung implementiert. Diese Anwendung sorgte dafür, dass beim Betreten der Räume jeweils der Fernseher mit der Lieblingssendung des Bewohners eingeschaltet wurde. Beim Verlassen des Zimmers wurde der Fernseher ausgeschaltet. So konnte der Bewohner durch die Wohnung sich bewegen ohne seine Lieblingssendung zu verpassen.

Weitere Informationen zu CAMUS können unter [45] oder [32] gefunden werden.

[20] Die Homepage dieses Projektes kann unter `http://oslab.khu.ac.kr/camus/` gefunden werden.

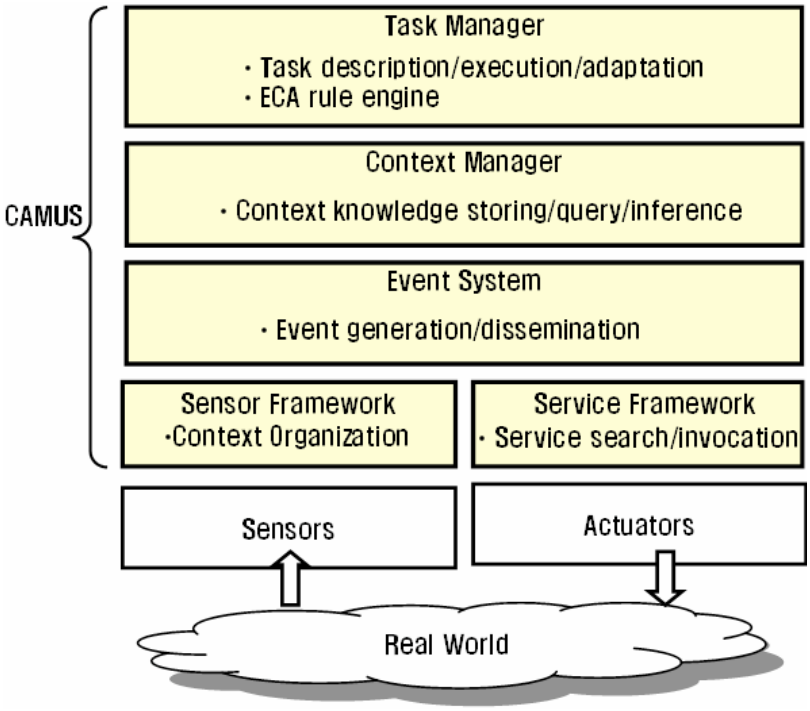


Abbildung 3.11.: Konzeptionelle Architektur von CAMUS [32]

3.3.2. Konfigurierung von eHome-Systemen

In seiner Dissertation „Konfigurierung von eHome-Systemen“ stellt Ulrich Norbisrath eine Möglichkeit für die Unterstützung aus softwaretechnischer Sicht der Prozesse der Einrichtung von sogenannten Smart-Home- oder eHome-Systemen dar [48]. Hierfür wurden graphische Werkzeuge entwickelt, die eine Konfiguration des gewünschten Systems ermöglichen bzw. unterstützen. Anhand dieser Konfiguration soll das System von Entwicklern realisiert werden.

Norbisrath bezeichntet den Prozess von der Spezifizierung bis zur Entwicklung eines eHome-Systems als *SCD-Prozess*, wobei *SCD* für *Spezifizierung*, *Konfigurierung* und *Deployment* steht (vgl. Abbildung 3.12). Demnach durchläuft dieser Prozess drei Phasen [48]:

1. Spezifizierung der eHome-Umgebung und der benötigten Dienste,
2. Automatische Konfigurierung der ausgewählten Dienste und
3. Deployment der Konfiguration auf das Service-Gateway im eHome.

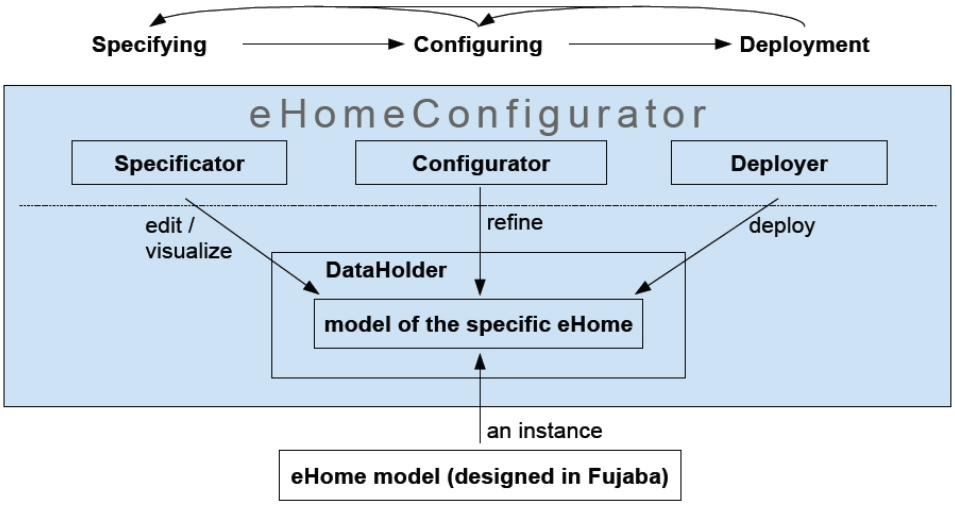


Abbildung 3.12.: Übersicht über den Aufbau des eHomeConfigurators [48]

Die entwickelten Werkzeuge unterstützen diesen Prozess. So können z.B. Bewohner ihre Wünsche mit Hilfe eines graphischen Tools eingeben. Diese werden dementsprechend in der Konfiguration abgebildet, die anschließend für die Entwicklung und das Deployment verwendet wird. Mit dieser Vorgehensweise hofft Norbisrath die Kunden in die Entwicklung derartiger Systeme mit einzubeziehen und somit die Entwicklungskosten zu senken.

Getestet wurde dieses Verfahren anhand der Entwicklung mehrerer Anwendungen. Diese und weitere Informationen zur Konfigurierung von eHome-Systemen können [48] entnommen werden.

3.3.3. MavHome

University of Texas at Arlington betreibt das Projekt *MavHome* (*Managing An Intelligent Versatile Home*), das sich mit der Forschung im Bereich Smart Home beschäftigt. Das Ziel dieses Projektes ist, eine Wohnumgebung zu entwickeln, die als ein intelligenter Agent agiert [11]. Sie soll anhand der Sensoren die Umgebung wahrnehmen und mittels bestimmter Steuergeräte (*Device Controller*) handeln. Dieser Agent soll zur Erhöhung der Lebensqualität des Bewohners beitragen.

MavHome besteht aus mehreren Agenten, die miteinander Kommunizieren können, falls sie eine bestimmte Aufgabe nicht selbständig bewältigen können. Ein solcher Agent ist durch ein Vier-Schichten-Modell repräsentiert:

- *Decision layer* entscheidet anhand der Informationen, die von den anderen Schichten übermittelt wurde, welche Aktionen auszuführen sind.

- *Information layer* verwaltet entscheidungswichtige Informationen.
- *Communication layer* ist für die Kommunikation zwischen den Agenten verantwortlich.
- *Physical layer* stellt die Hardware des Agenten, inklusive seiner individuellen und Netzwerk-Hardware, dar.

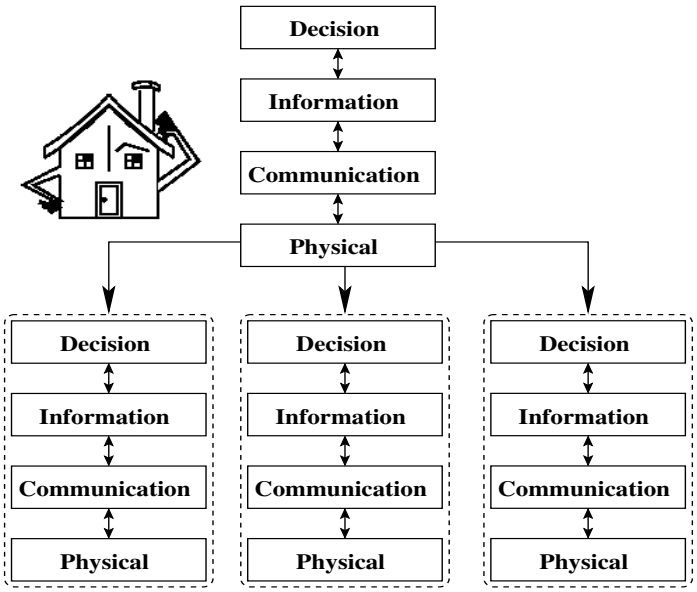


Abbildung 3.13.: Agenten-Architektur von MavHome [11]

Die Architektur von MavHome ist hierarchisch aufgebaut. Somit werden die einzelnen Schichten eines Agenten durch die entsprechenden Schichten des jeweiligen untergeordneten Agenten repräsentiert. Die Arbeitsweise dieser Architektur verläuft nach dem *bottom-up*-Prinzip. Abbildung 3.13 stellt diese Architektur graphisch dar.

Eine detaillierte Beschreibung des Projektes MavHome findet sich unter [11]

3.3.4. ACHE

Das System *ACHE* (*adaptive control of home environments*) entstand aus dem Forschungsprojekt *Adaptive House*. Dieses Projekt hat die Entwicklung eines Hauses, das sich durch Beobachtung selbsttätig an die Benutzer anpasst, im Fokus. ACHE beobachtet den Benutzer in seiner Tätigkeit und registriert sein Verhalten. Diese Informationen dienen zur Erstellung von Benutzer-Profilen. Anhand des vom System ermittelten Benutzer-Profils und dem aktuellen Wetter-Gegebenheiten wird z.B. die Raumtemperatur automatisch geregelt.

ACHE verfügt über Sensoren, die die Umgebungs-Eigenschaften des Hauses (wie z.B. das Licht und dessen Intensität, Raumtemperatur, Geräuschpegel u.a.) registrieren können. Die von ihnen erhaltenen Informationen gehen in die Entscheidung für die Steuerung des Hauses mit ein. Für die eigentliche Steuerung verwendet ACHE die entsprechenden Steuergeräte bzw. -einheiten.

Die Architektur von ACHE ist in Abbildung 3.14 abgebildet. Sie besteht aus fünf Hauptkomponenten:

- *State Transformation* führt die Statistiken über die Maximum-, Minimum- und Durchschnittswerte des Umgebungszustandes für das eingegebene Zeitfenster.
- *Occupancy Model* stellt fest, ob in einer bestimmten Wohnzone (üblich ein Zimmer) sich eine Person sich aufhält.
- *Predictors* beschäftigt sich mit der Analyse der von den ersten zwei Modulen übermittelten Informationen und berechnet den zukünftigen Umgebungszustand.
- *Setpoint Generator* ermittelt anhand des aktuellen und zukünftigen Zustandes der Umgebung den Einstellwert für die jeweiligen Steuergeräte.
- Bedient werden die Steuergeräte von jeweiligen *Devise Regulators*

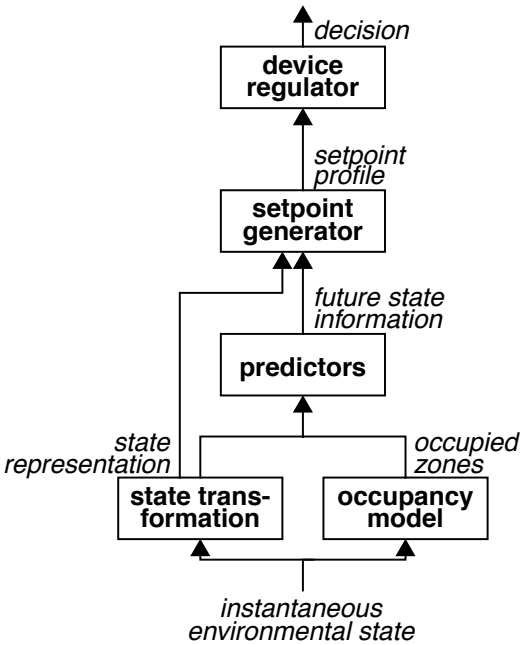


Abbildung 3.14.: System-Architektur von ACHE [46]

Bereits realisiert und getestet wurden die Heizungs- und Licht-Kontroler. Diese funktionieren ohne aufwendige Einstellungen seitens der Benutzer und agieren vollständig autonom.

Weitere Informationen zu diesem System können [46] entnommen werden.

3.3.5. Fazit

Die in diesem Kapitel vorgestellten Forschungsprojekte zeigen, dass bereits brauchbare Lösungen für Systeme intelligenter Wohnumgebungen entwickelt wurden. Sie unterscheiden sich zwar voneinander in ihrer Arbeitsweise, jedoch haben sie ein gemeinsames Ziel. Diese Projekte versuchen ein vollständiges System zu entwickeln, das den Anwendern das Bewohnen einer Wohnumgebung komfortabler gestaltet. Dieses soll die Bewohner von den aufwendigen heimischen Regulierungen entlasten und den Haushalts- und den Entertainment-Bereich unterstützen.

In den folgenden Kapiteln wird ein System für intelligente Wohnumgebungen mit seinen Merkmalen und Eigenschaften präsentiert. Hierbei wird ein mögliches Szenario beschrieben, anhand dessen die Anforderungen an das System gestellt werden. Die hieraus gewonnenen Erkenntnisse sollen in den späteren Architekturentwurf einfließen.

3.4. Gesamtszenario

In diesem Kapitel wird die Hauptvision des Projektes Living Place Hamburg vorgestellt. Im Folgenden werden die wichtigsten Merkmale detailliert beschrieben.

Eine klassische Wohnumgebung besteht aus mehreren Zimmern (bzw. Wohnbereichen), die je nach Bestimmung besondere charakteristische Eigenschaften aufweisen. So ist das Schlafzimmer ein Wohnbereich, in dem der Bewohner sich ausruhen kann. Ein solches Zimmer wird mit einem Bett und evtl. einem Wecker ausgestattet. Dieser Wohnbereich wird oft als Umkleideraum genutzt. Demnach befinden sich ein Kleiderschrank und ein Spiegel in diesem Raum.

Ein Wohnzimmer wird dagegen als Unterhaltungsraum verstanden. Somit wird dieser Raum mit Polstermöbeln und Unterhaltungsmedien wie Fernseher, Stereo-Anlage u.a. ausgestattet. Weitere Beispiele für spezielle Räume sind das Badezimmer, das für die hygienische Pflege genutzt wird, und die Küche, in der das Essen zubereitet wird. Diese Räume werden mit den nur für sie charakteristischen Utensilien ausgestattet.

Diese Beispiele zeigen, dass bestimmte Wohnbereiche für spezielle Zwecke im klassischen Sinne entworfen und aufgebaut werden. Der Mensch ist jedoch ein autonomes Wesen, das nicht immer diesem folgt. So wird oft ein Wohnzimmer als Schlafzimmer genutzt, z.B. wenn der Bewohner beim Fernsehen einschläft. Nicht selten wird auch im Wohnzimmer gekocht.

So wird beispielsweise häufig Silvester Fondue oder Raclette gemacht. Demnach bekommt das Wohnzimmer für diesen Moment die Funktionalität einer Küche.

Ein Ergebnis ist, dass die klassischen Bestimmungen der Räume aufgegeben werden können. Dies kann in einer Ein-Zimmer-Wohnung beobachtet werden. Wegen der räumlichen Enge wird ein Zimmer als Wohn-, Schlaf-, Ess- und Arbeitszimmer genutzt. Eine funktionale Flexibilität der Räume ist demnach nötig.

Die Hauptidee des Living Place Hamburg liegt darin, ein System zu entwickeln, das diese Flexibilität unterstützt. So soll ein Zimmer in der Lage sein, bei Bedarf, funktional flexibel genutzt werden zu können und nicht wie ein klassisches Zimmer nur einen bestimmten Zweck zu erfüllen. Um dies zu ermöglichen, muss das Gesamtsystem im Stande sein kontextuelle Informationen zu sammeln und zu deuten. Die semantische Interpretation der gesammelten Informationen kann dem System verhelfen auf die gegebene Situation adäquat zu reagieren.

Beispielhaft ist folgendes Szenario: Der Bewohner befindet sich im Wohnzimmer und sieht fern. Er schläft ein. Das System erkennt diese Situation und in Folge dessen führt es die folgenden Aktionen aus: der Fernseher wird ausgeschaltet, das Licht wird gedämmt oder ganz ausgeschaltet und die Jalousien werden geschlossen.

Zur Verdeutlichung der Gesamtvision wird ein weiteres Beispielszenario vorgestellt: Ein simples Ein- und Ausschalten des Lichts in Räumen kann auf eine einfachere und „intelligentere“ Weise erfolgen. So kann der Bewohner z.B. per Sprachsteuerung oder mit Hilfe seines mobilen Gerätes, das er immer bei sich trägt, den Auftrag geben, dass das Licht ein- bzw. ausgeschaltet werden soll. Das System erkennt dessen Aufenthaltsort und führt den Befehl für diesen Raum aus. Die gegebene Situation kann unter Umständen etwas komplexer sein. So können verschiedene umgebungsbezogene Faktoren die Ausführung des Kommandos beeinflussen. So könnte das Vorhandensein eines schlafenden Kindes das Einschalten des Lichts verhindern. Die Lichtintensität und derer Farbton kann an die äußerliche Umgebung (vgl. Wetterbedingungen oder Tageszeit) und an die Vorlieben der Bewohner angepasst werden. Das System soll dieses erkennen und hierauf adäquat reagieren.

Die Hauptvision zu verwirklichen bzw. zu implementieren ist zu diesem Zeitpunkt unrealistisch. Der gesamte Raum der möglichen Anwendungen kann zu diesem Zeitpunkt nicht (bzw. nur sehr bedingt) erfasst werden. Falls dies überhaupt möglich wäre, wäre der Entwicklungsaufwand sehr groß und somit im Rahmen dieser Arbeit nicht zu bewältigen. Die Entwicklung einer bestimmten Anwendung, die ein Bestandteil der Hauptvision sein kann, kann jedoch dazu beitragen die Komplexität und die damit verbundenen Entwicklungsschwierigkeiten besser zu verstehen bzw. diese zu bewältigen.

Das nächste Kapitel stellt eine konkrete Ausprägung der Gesamtvision vor.

3.5. Cooking Agent

Dieses Kapitel stellt ein Szenario vor, das eine mögliche Ausprägung der im Kapitel 3.4 beschriebenen Gesamtvision und deren Teilanwendung darstellen kann. Mit der Implementierung dieser Anwendung wird versucht, die Problematik, die mit der Entwicklung der Gesamtvision verbunden ist, zu verdeutlichen und eine Variante für deren Lösung zu finden.

Die Anwendung in diesem Kapitel trägt den Namen *Cooking Agent*[21]. Wie der Name bereits verrät, soll diese Anwendung die Bewohner bei der Essenzubereitung unterstützen. Die Anwendung soll geeignete Kochrezepte vorschlagen und falls notwendig auch eine Einkaufsliste der benötigten Produkte zusammenstellen. Diese Entscheidung soll auf verschiedenen Faktoren beruhen. Diese sind u.a. vorhandene Produkte (bzw. Zutaten), Bewohner- und Gästevorlieben und die verfügbare Kochzeit. Im Weiteren wird die Anwendung detaillierter beschrieben.

3.5.1. Mögliches Szenario

Cooking Agent soll mehrere Features aufweisen, die dem Bewohner bei der Entscheidung oder bei der Suche nach einem passenden Kochrezept unterstützen. Die gesamte Funktionalität dieser Anwendung umfasst diverse Szenarien. Eine Auflistung aller möglichen Anwendungsfälle würde den Rahmen dieser Arbeit sprengen. Im Folgenden wird ein mögliches Szenario beschrieben, das die wichtigsten Features und Eigenschaften des Cooking Agent verdeutlicht:

> *Der Bewohner möchte ein feierliches Dinner geben. Er hat bereits mehrere Personen hierzu eingeladen. Es stellt sich jetzt die Frage nach einem passenden Menü. Der Gastgeber möchte einerseits ein feierliches Essen mit mehreren Gängen zubereiten, das die geschmacklichen Vorlieben der Gäste trifft. Andererseits will der Bewohner die bevorstehende Zubereitungszeit und die Kosten möglichst genug halten.*
>
> *Der Gastgeber wünscht eine Beratung in dieser Situation und nimmt Cooking Agent zu Hilfe. Er fragt die Anwendung nach passenden Rezepten, indem er den dafür vorgesehenen Button der Benutzer-Schnittstelle betätigt. Der Anwendung ist bereits der Anlass des Essens und der Zeitpunkt bekannt, da diese als Termin vom System verzeichnet wurden. Cooking Agent kennt die Vorlieben des Gastgebers und die der Gäste und ist mit einem intelligenten Kühlschrank vernetzt, von dem eine Liste der vorhandenen Produkte erfragt werden kann.*

[21] aus dem Englischen *Kochagent*

Die Anwendung durchsucht die Rezepte-Datenbank nach geeigneten Rezepten. Hierbei werden die Vorlieben des Gastgebers und dessen Gäste, die Zubereitungszeiten, die Informationen über vorhandene Produkte, die die Anwendung von dem intelligenten Kühlschrank erhält, und die möglichen Gesamtkosten berücksichtigt. Als Ergebnis dieser Abfrage erhält der Bewohner eine Liste vorgeschlagener Kochrezepte und evtl. eine Einkaufsliste, auf der die im Haushalt fehlenden Produkte aufgelistet sind. Der Gastgeber wählt bestimmte Gerichte aus, worauf das System die gewünschten Kochrezepte und die hierzu passende Einkaufsliste zusammenstellt. Der Bewohner druckt diese Auswahl aus und kann mit dem Einkauf und der anschließenden Essenzubereitung beginnen.

Die spätere Anwendungsanalyse und das Architekturdesign und deren Implementierung werden auf diesem Szenario basieren.

3.5.2. Funktionale Anforderungen

Das im Kapitel 3.5.1 vorgestellte Szenario weist mehrere funktionalität-bezogene Features auf. Dieses Kapitel fasst diese zusammen und stellt somit die funktionalen Anforderungen an die Anwendung Cooking Agent.

Die Hauptaufgabe von Cooking Agent ist die Suche nach geeigneten passenden Kochrezepten. Somit muss die Anwendung eine große Anzahl verschiedener Rezepte verwalten können. Diese stellen dann die Basismenge für die spätere Entscheidung. Der eigentliche Rezeptvorschlag ist das Ergebnis der Analyse verschiedener Randbedingungen oder Faktoren wie z.B.: vorhandene Produkte, Einkaufsbereitschaft, Vorlieben des Gastgebers und der Gäste, die zur Verfügung stehende Zubereitungszeit, Schwierigkeitsgrad der Rezepte und Eigenschaften der Gerichte wie Kalorien-, Fett- oder Alkoholgehalt. Aus den gerade aufgelisteten Randbedingungen verdienen folgende Features eine besondere Berücksichtigung:

- Ermittlung vorhandener Produkte,
- Verwaltung und Analyse der Personenprofile,
- Analyse der Gerichte bzw. deren Rezepte,
- Verwaltung der Termine (vgl. Kalenderführung) und
- Berücksichtigung aller Rahmenbedingungen (bzw. Ermittlung der passenden Rezepte anhand dieser Rahmenbedingungen).

Die Anwendung soll Funktionen zur Erstellung des Menüs und die sich daraus notwendigerweise ergebene Erstellung der Einkaufsliste anbieten.

Cooking Agent soll über eine Benutzerschnittstelle verfügen, mit deren Hilfe der Bewohner mit der Anwendung interagieren kann. Diese soll in Form einer grafischen Oberfläche realisiert werden, die eine einfache Bedienung der Anwendung erlaubt und deren Ergebnisse grafisch darstellt. Zudem sollen die Ergebnisse druckbar sein, damit der Bewohner sie jederzeit zur Hand hat. Das ist vor allem für das zusammengestellte Menü, die Kochrezepte und die Einkaufsliste sinnvoll.

Das Anwendungsfall-Diagramm[22] 3.15 stellt die Features von Cooking Agent und deren Zusammenhänge aus der Sicht des Bewohners dar.

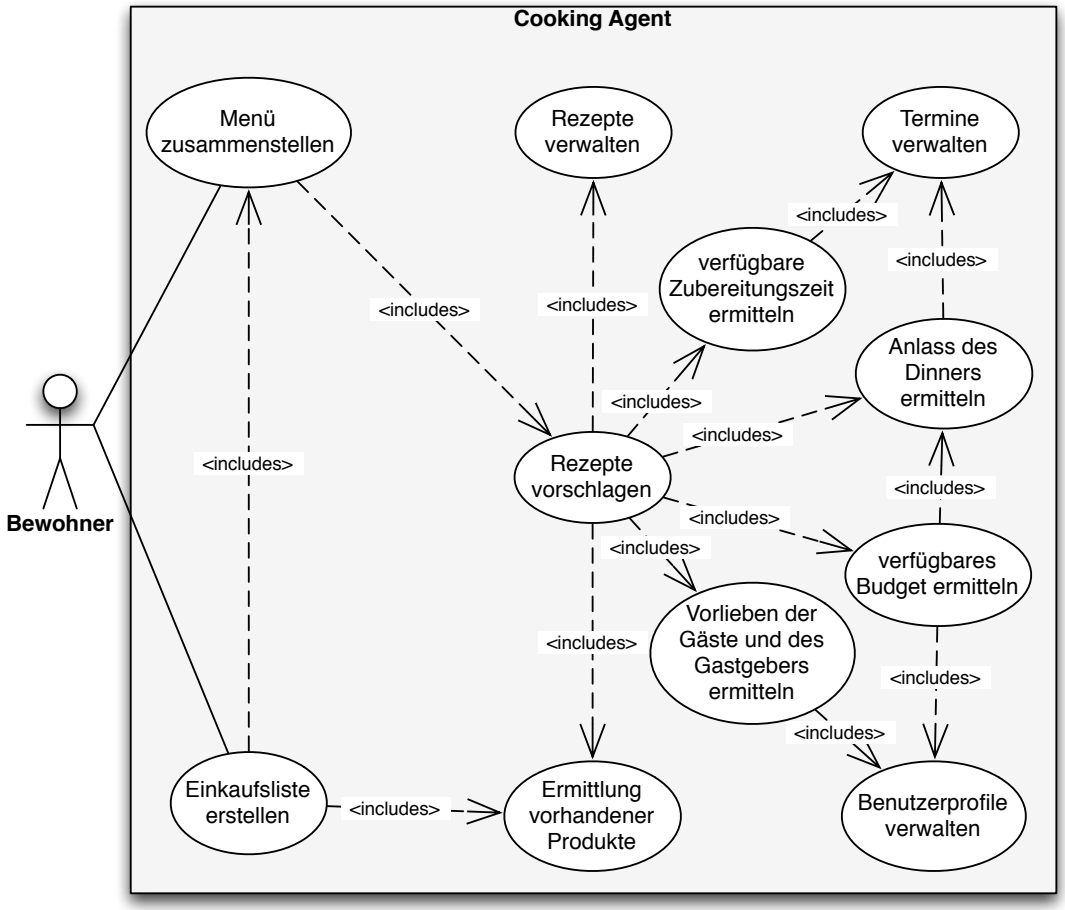


Abbildung 3.15.: Cooking Agent: Anwendungsfall-Diagramm

[22] Anwendungsfall-Diagramm (engl. *use case diagram*) ist eine Spracheinheit der UML (*Unified Modeling Language*), die das nach außen sichtbare Verhalten eines Elements (z.B. Benutzer oder System) zeigt.

Tabelle 3.1 fasst die hier ermittelten funktionalen Anforderungen noch einmal zusammen. Neben der Benennung der Features wird u.a. deren Funktionalität kurz erläutert. Hierbei soll beachtet werden, dass einige im Anwendungsfall-Diagramm 3.15 aufgeführten Features zu einer abstrakteren Funktionalität zusammengefasst wurden, sodass das allgemeinere Feature diese impliziert.

Diese funktionalen Anforderungen sollen nicht vollständigen Funktionsumfang von Anwendungen dieser Art verstanden werden. Sie haben lediglich einen exemplarischen Charakter. Somit kann die Funktionalität solcher Anwendungen geändert bzw. erweitert werden.

	Feature	kurze Beschreibung
1	Kochrezepte vorschlagen	Anhand der Rahmenbedingungen wie z.B. Anlass des Essens, verfügbare Zubereitungszeit, verfügbares Budget, Präferenzen der Gäste und des Gastgebers und bereits vorhandene Produkte soll Cooking Agent passende Kochrezepte finden.
2	Rezepte verwalten	Cooking Agent soll Rezepte, die die Basismenge für die Ermittlung der gewünschten Rezepte darstellen, verwalten können.
3	Termine verwalten	Eine Art Kalender, in dem Termine mit den zugehörigen Informationen verwaltet werden. Im Hinblick auf die Anwendung sollen hier das Datum, die Uhrzeit, der Anlass des Essens und Informationen über die erwarteten Gästen des geplanten Dinners registriert werden.
4	Benutzerprofile verwalten	Cooking Agent soll Benutzerprofile verwalten können, in denen außer den allgemeinen Informationen über die Personen auch deren Vorlieben, vor allem die essenbezogene Präferenzen gespeichert werden.
5	Ermittlung vorhandener Produkte	Die Anwendung soll in der Lage sein die bereits vorhandenen Produkte zu ermitteln.
6	Analyse der Rezepte	Cooking Agent soll in der Lage sein die Rezepte nach ihre Eigenschaften (wie beispielsweise Gerichtstyp, Zubereitungszeit, Schwierigkeitsgrad, Fett- oder Alkoholgehalt u.a.) zu analysieren. Diese Informationen sollen bei der Suche nach den passenden Rezepten berücksichtigt werden.
7	Menüzusammenstellung	Anhand der von Cooking Agent vorgeschlagenen Gerichte kann der Gastgeber ein Menü zusammenstellen. Dieser Vorgang soll von Cooking Agent unterstützt werden.
8	Erstellung der Einkaufsliste	Unter Berücksichtigung der bereits vorhandenen Produkte soll für das gewählte Menü eine Einkaufsliste mit den noch fehlenden Produkten erstellt werden.
9	Userinterface	Cooking Agent soll eine Benutzerschnittstelle für die Interaktion des Benutzers mit der Anwendung anbieten. Diese soll Werkzeuge zur Verfügung stellen, die das Bedienen der Anwendung erlauben und die Ergebnisse graphisch darstellen. Die wichtigsten Ergebnisse sollen ausgedruckt werden können.

Tabelle 3.1.: Feature-Übersicht von Cooking Agent

3.5.3. Nicht-funktionale Anforderungen

Neben den im Kapitel 3.5.2 präsentierten funktionalen Anforderungen werden eine Reihe von nicht-funktionalen Anforderungen an das System gestellt. Die nicht-funktionalen sind Anforderungen an die Umstände, unter denen die geforderte Funktionalität zu erbringen ist. Diese bestehen aus Leistungsanforderungen, Qualitätszielen und Randbedingungen [28].

Da es in dieser Arbeit um ein System geht, das im Rahmen einer Forschungsarbeit entwickelt wird, müssen nicht alle nicht-funktionalen Anforderungen erfüllt sein, die für dieses System sinnvoll sind[23]. Hier werden lediglich die wichtigsten nicht-funktionalen Anforderungen beschrieben, die aus der Sicht des Autors für die Entwicklung unabdingbar sind. Weitere Anforderungen werden hier zwar nicht erwähnt, jedoch werden sie bei der Entwicklung des Systems in Betracht gezogen, sodass jenes im Sinne einer qualitativen Software konzipiert wird. Solche Anforderungen und weitere Informationen zum Thema „Nicht-funktionale Anforderungen“ können der Fachliteratur entnommen werden z.B. [28] oder [37].

Einfache Modifikation bzw. Erweiterbarkeit

Cooking Agent ist ein Beispiel für ein System einer intelligenten Wohnumgebung. Diese Anwendung soll so gestaltet werden, dass deren Modifikation bzw. Erweiterung mit geringem Aufwand verbunden ist, sodass diese relativ einfach zu einem komplexeren System vervollständigt werden kann. Auf diese Weise kann aus Cooking Agent ein vollwertiges System für intelligente Häuser entstehen.

Plattform- und Sprachunabhängigkeit

Cooking Agent soll so entwickelt werden, dass es auf unterschiedlichen Plattformen lauffähig ist. Auf diese Weise kann die Anwendung in unterschiedlichen Umgebungen installiert bzw. integriert werden. Zudem soll das entwickelte System verschiedene Programmiersprachen unterstützen, bzw. unabhängig von ihnen sein. Dies kann die Integration weiterer Anwendungen in das Gesamtsystem (oder die Erweiterung der bereits bestehenden) vereinfachen, die in anderen Sprachen realisiert wurden.

[23] Das Hauptziel dieser Arbeit besteht nicht in der Entwicklung eines bestimmten Systems (bzw. Anwendung), sondern in der Erforschung einer möglichen Lösung für Systeme aus dem Bereich *intelligentes Haus*.

3.6. Zusammenfassung

In Kapitel 3 wurden Labore vorgestellt in denen Forscher an der Entwicklung intelligenter Wohnumgebungen arbeiten. Unter diesem Aspekt wurden auch Labore an der Hochschule für Angewandte Wissenschaften Hamburg eingerichtet. Kapitel 3.3 gab einen Überblick über Projekte, deren Ziel die Entwicklung vollwertiger Systeme für intelligente Wohnunumgebungen ist. Die Hauptvision dieser Arbeit und ein konkretes Szenario wurden in Kapiteln 3.4 und 3.5 präsentiert. Hierbei wurden die Eigenschaften eines solchen Systems erläutert und die Anforderungen an das konkrete zu entwickelnde System gestellt.

Im weiteren Verlauf dieser Arbeit wird ein Architekturentwurf für das System vorgeschlagen. Hierbei werden die Hauptüberlegungen an die Gesamtvision gerichtet. Eine konkrete Ausprägung dieser Architektur, indem eine Beispielanwendung entwickelt wird, dient zur Verdeutlichung der Bestandteile dieser Architektur und deren Zusammenspiel.

4. Design und Architektur

Dieses Kapitel beschäftigt sich mit der Konzeption einer Architektur für eine intelligente Wohnumgebung, deren allgemeine Beschreibung und konkrete Anforderungen in den Kapiteln 3.4 und 3.5 zu finden sind. In diesem Zusammenhang werden die grundlegenden architektonischen Überlegungen im Hinblick auf das Gesamtszenario beschrieben, wobei eine konkrete Ausprägung dieser Architektur am Beispiel von Cooking Agent verdeutlicht wird.

In den Kapiteln 3.4 und 3.5 wurden mögliche Szenarien für intelligente Räume beschrieben. Hierbei wurden die Interaktionen zwischen dem Bewohner und dem intelligenten Raum erläutert. Es wurde deutlich, dass die Kontexterkennung ein zentraler Bestandteil der Architektur ist[1]. Diese wird durch die Verarbeitung der von den Sensoren erhaltenen Rohdaten gewonnen.

Eine Eigenschaft von intelligenten Räumen ist die Vielfalt an unterschiedlichen Geräten in diesen Räumen. Ihre Handlungen werden von den Anwendern als „räumliche Intelligenz“ empfunden. Diese Geräte bieten bestimmte Dienste (Leistungen) an und sind meist eigenständig, sodass sie ohne Hilfe anderer arbeiten können. Geräte bzw. Dienste, die nur in Kooperation mit den weiteren Geräten oder Diensten lauffähig sind, sind auch in intelligenten Wohnumgebungen denkbar. Diese können andere Dienste erweitern oder als Dritt-Agenten fundieren, deren Aufgabe es ist, die Daten für die weitere Verarbeitung entsprechend vorzubereiten.

Die Granularität der einzelnen Dienste, deren Zusammenarbeit einen höheren Nutzen darstellt, deutet auf eine dienst-orientierte Architektur (SOA[2]) für Systeme dieser Art hin. Architekturen dieses Typs sind seit einigen Jahren in der Geschäftswelt verbreitet. Die bisher gemachten Erfahrungen können bei der Entwicklung eines Systems für intelligente Umgebungen behilflich sein. Hierbei müssen die spezifischen Charakteristika solcher Systeme berücksichtigt werden. Die meisten SOA-Lösungen sind prozess-orientiert, bei denen die Geschäftsprozesse abgebildet werden, wobei intelligente Räume, welche die Menschen in ihren Alltagsaufgaben unterstützen sollen [58], aktivität-orientiert sein müssen. Diese Unterschiede spiegeln sich auch in der Architektur wider.

[1] vgl. Beispiele aus dem Kapitel 3.4
[2] siehe Kapitel 2.3.1

4.1. Konzeptionelle Architektur

Es wurden bereits verschiedene Systemlösungen für intelligente Wohnumgebungen entwickelt. Hierbei wurden unterschiedliche Architekturansätze verfolgt, die jedoch folgende gemeinsame Teil-Ziele verfolgt haben:

- adäquate Reaktion auf die Änderung in der Umgebung und
- Möglichkeit zur Integration neuer Softwarekomponenten (Erweiterbarkeit des Systems).

Diese Punkte sind charakteristisch für intelligente Räume (vgl. [61], [1], [52], [18], [4] u.a.).

Eine der wichtigsten Eigenschaften intelligenter Häuser ist die Reaktion auf die Veränderungen im Raum bzw. eine adäquate Reaktion auf die eintreffenden Ereignisse. Diese Ereignisse (auch *Events* genannt) werden von den Sensoren erfasst. Die von den Sensoren gewonnenen Rohdaten werden weiter analysiert (bzw. verarbeitet) und zu einem Kontext zusammengefasst. Anhand des ermittelten Kontext werden bestimmte Aktionen ausgeführt, die aus der Sicht des Systems sinnvoll erscheinen. Das gerade beschriebene Vorgehen stellt die Abbildung 4.1 graphisch dar.

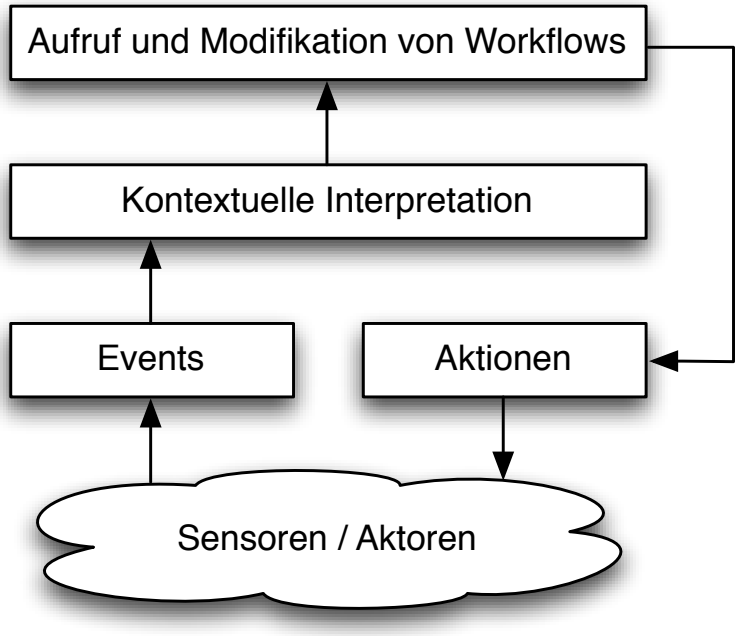


Abbildung 4.1.: Konzeptionelle Architektur (angelehnt an [58])

Hierbei ist zu berücksichtigen, dass eine Aktion sowohl eine für den Benutzer gewünschte Handlung (z.B. Ausschalten des Lichts im Wohnzimmer) als auch eine Erzeugung von einem weiteren Ereignis sein kann, das neue Handlungen auslöst. Dieses Gebilde kann beliebig klein gestaltet werden, sodass diese Softwarekomponente eine kleine, definierte und in sich abgeschlossene Aktion ausführt. Auf diese Weise wird der Grad der Wiederverwendbarkeit dieser Komponente erhöht. Ein weiterer Vorteil dieses Konzepts liegt darin, dass die Softwarekomponenten mittels Austausches von Events miteinander kommunizieren können. Somit wird eine Anwendung durch Handlungen mehrerer nacheinander folgender oder auch parallel laufender Softwarekomponenten repräsentiert. Diese Vorgehensweise entspricht auch dem SOA-Gedanken (vgl. Kapitel 2.3.1).

Entschieden werden muss ob diese Komponenten als Services oder als Softwareagenten entwickelt werden sollen. Diese beiden Komponentenarten sind kleine Programme, die bestimmte Aufgaben lösen. Jedoch unterscheiden sie sich in ihrer Arbeitsweise. Reine Services bieten bestimmte Dienste an und führen klar definierte Programmschritte aus. Sie müssen permanent einsatzbereit sein. Agenten dürfen dagegen temporär offline sein und können von sich aus bestimmte Aktionen auslösen. Agenten können u.a. die Veränderungen in ihrer Umgebung wahrnehmen und dies bei ihrer Entscheidung berücksichtigen (siehe Kapitel 2.3.2). Somit eignen sich Agenten für den Einsatz in intelligenten Räumen. Die Agenten sollen Ereignisse registrieren und sie entsprechend verarbeiten können. Eine Kombination von Services und Agenten ist denkbar. So können Agenten Dienste, die Services anbieten, in Anspruch nehmen. Zur einfacheren Handhabung ist der Einsatz von gleichartigen Softwarekomponenten sinnvoll. Demnach werden in dem hier konzipierten System die einzelnen Softwarekomponenten als Softwareagenten modelliert.

4.1.1. Agentenmodellierung

Den oben genannten Bedingungen folgend, sieht ein Agent wie folgt aus: Er besteht aus einem so genannten Verarbeitungsteil, das den „Körper“ des Agenten ausmacht, und einer Interaktionsschnittstelle. Alle Verarbeitungsschritte, die vom Agenten ausgeführt werden sollen, werden im Körper des Agenten implementiert. Die Interaktionsschnittstelle ist für die Kommunikation des Agenten mit seiner Umgebung zuständig. Somit werden in dieser Schnittstelle zwei Kanäle unterschieden (siehe Abbildung 4.2):

- *IN* - für die Erfassung der eingehenden Informationen und
- *OUT* - für den Transport der verarbeiteten Daten nach außen.

Durch den IN-Kanal können sowohl die vom Sensor gewonnenen Daten als auch die Events, die von den anderen Agenten erzeugt wurden, fließen. Die OUT-Schnittstelle besitzt zwei Ausprägungen: Die Ausgabedaten (bzw. Events) können durch diesen Kanal fließen oder

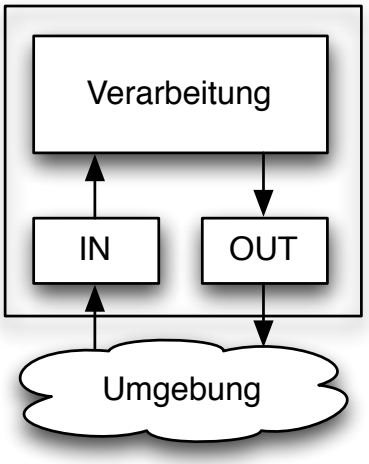


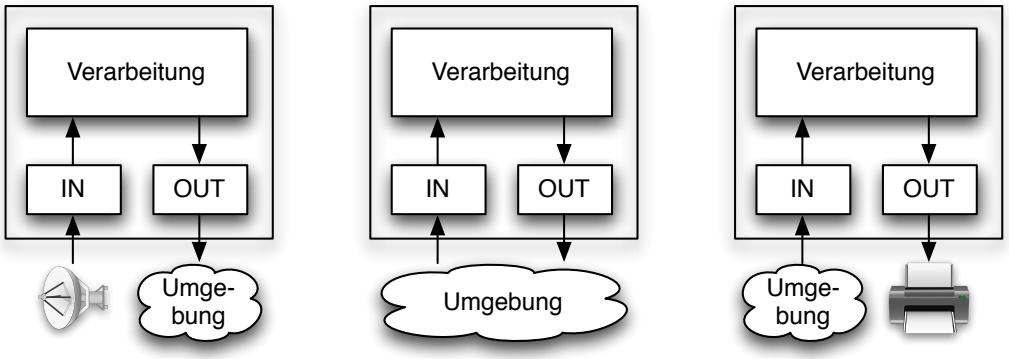
Abbildung 4.2.: Abstrakte Sicht des Agenten

die OUT-Schnittstelle kann die Steuerung der Endgeräte durchführen (vgl. das Beispiel vom Ein-/Ausschalten des Lichts). Es werden drei Arten von Agenten in diesem System unterschieden:

- *Sensor-Agenten* - diese Agenten besitzen physikalische Sensoren und schicken Events falls sie Ereignisse registriert haben.
- *Dritt-Agenten* - diese verarbeiten die eingegangenen Events und erzeugen neue als Ergebnis dieser Verarbeitung.
- *Aktion-Agenten* - diese erhalten Events von den anderen Agenten und führen bestimmte End-Aktionen aus (z.B. Einschalten des Lichts).

Abbildung 4.3 stellt die drei Agententypen schematisch dar. Hierbei werden Sensoren durch ein Antenne und Aktionen durch einen Drucker als Beispiel für ein gesteuertes Gerät repräsentiert.

Hierbei ist darauf zu achten, dass die Sensor-Agenten und die Aktion-Agenten, sowohl über den IN- als auch über den OUT-Kanal (so wie die Dritt-Agenten) Events empfangen und verschicken können. Dies kann sinnvoll sein, wenn z.B. die Sensor-Agenten auf Wunsch anderer Agenten die Umgebung mit ihren Sensoren abtasten (vgl. Request-Event und Response-Event) oder die Aktion-Agenten über die Ausführung bestimmter Aktionen einen Bericht liefern sollen. Der Unterschied zwischen den Sensor- und Aktion-Agenten und den Dritt-Agenten besteht darin, dass die Dritt-Agenten keine Geräte besitzen, die sie steuern können, sodass sie als „reine Software-Agenten“ gesehen werden können.



(a) (b) (c)

Abbildung 4.3.: Klassifikation der Agenten a) Sensor-, b) Dritt- und c) Aktion-Agent

Mit Hilfe dieser drei Agentenarten können Anwendungen für intelligente Räume entwickelt werden. Die Sensor-Agenten registrieren dabei die in der Umgebung eintreffenden Ereignisse. Diese Ereignisse werden von den Dritt-Agenten verarbeitet und von den Aktion-Agenten werden die daraus gewonnenen Ausführungsschritte durchgeführt. Dieser Ablauf könnte u.U. von nur einem Agenten durchgeführt werden, wenn er die Sensoren, die Endgeräte und die notwendige „Intelligenz“ besäße. Diese Methode würde jedoch das Wiederverwendbarkeitsprinzip verletzen bzw. sehr einschränken, da die einzelnen Dienste, die Teil-Aufgaben lösen, nach außen verborgen wären und somit nicht in Anspruch genommen werden könnten.

4.1.2. Kommunikation

In dieser Architektur wird eine indirekte Kommunikation zwischen den Agenten verwendet. Eine Vorteil dieser Art der Kommunikation, gegenüber der Kommunikation innerhalb des Multiagentensystems im klassischen Sinne (vgl. FIPA ACL in Kapitel 2.3.2), ist, dass das Auffinden benötigter Agenten (bzw. Dienste) unnötig ist. Die Agenten-Kommunikation findet hier mittels des Event-Austausches statt. Die Events können u.U. mit den FIPA ACL-Nachrichten verglichen werden. Im Gegensatz zu den letzteren beinhalten Events keine Angaben sowohl über ihren Sender als auch über den Empfänger. Die Agenten werden hier nicht direkt angesprochen, sie werden eher durch den Empfang bestimmter Events „angeregt“ und somit zum Leisten ihrer Dienste aufgefordert.

Da die Agenten nur mittels des Event-Austausches miteinander kommunizieren, soll diese Kommunikation durch einen entsprechenden Event-Vermittler (auch *Event-Manager* ge-

nannt) unterstützt werden. Die ganze Kommunikation erfolgt nur über den Event-Manager (siehe Abbildung 4.4).

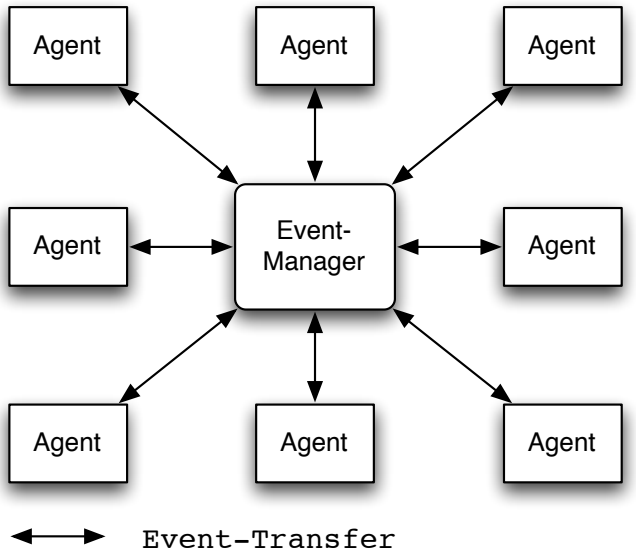


Abbildung 4.4.: Abstrakte Sicht des Agenten

Die semantische Interpretation der Events wird durch die Angabe ihrer Typen repräsentiert. So wie die Performativen in FIPA ACL müssen auch hier die Event-Typen klar spezifiziert sein, damit die Agenten die Events richtig interpretieren können. Die Empfänger der einzelnen Events werden durch die Unterscheidung der Event-Typen ermittelt. Der Sender übergibt Events bestimmten Typs an den Event-Manager, der seinerseits diese an die interessierten Agenten weitergibt. Die Empfänger-Agenten werden anhand des von ihnen registrierten Events ermittelt. Da mehrere Agenten bestimmte Events abonnieren können, wird es mehrere Empfänger dieser Events geben.

Die Nutzdaten (vgl. `:content` in FIPA ACL) werden mit den Events transportiert. So enthalten Events entsprechende Attribute, die diese Daten beinhalten.

Diese Kommunikationsart entspricht einem Blackboard-System (siehe Kapitel 2.3.3) mit dem Unterschied, dass die verteilten Daten nicht von den Agenten abgeholt werden, sondern ihnen zugestellt werden. Die Daten stehen aber wie in einem Blackboard-Modell allen Teilnehmern zur Verfügung. Damit ein Teilnehmer (bzw. Agent) nicht mit allen in dem System erzeugten Nachrichten überflutet wird, muss das Modell so erweitert werden, dass die einzelnen Teilnehmer nicht alle Nachrichten empfangen müssen, sondern nur die, die für sie wichtig sind. Dies kann durch die Einführung von Nachrichtentypen (vgl. Event-Typ) erreicht werden. Somit erleichtert der Event-Manager nicht nur die Kommunikation zwischen den

Agenten, sondern er kapselt auch die Agenten voneinander ab, sodass sie von ihrer Existenz keine Kenntnis haben. Auf diese Weise kann der Grad der Wiederverwendbarkeit und Erweiterbarkeit erhöht werden[3].

4.1.3. Generischer Agent

Anhand der in Kapitel 4.1.1 beschriebenen Agenteneigenschaften und der in Kapitel 4.1.2 erörterten Kommunikation zwischen den Agenten, wird ein generischer Agent wie folgt beschrieben. Ein Agent besteht aus einem Verarbeitungsteil und einem Kommunikationsteil. Das Verarbeitungsmodul leistet die wesentliche Arbeit des Agenten. Es enthält die Logik des Agenten und somit stellt seine Intelligenz dar. Leistungen des Agenten, die zur Bewältigung bestimmter Probleme führen, werden in Form von Diensten von dem Agenten angeboten. Das Kommunikationsmodul implementiert die Kommunikationsschnittstelle des Agenten mit seiner Umgebung. Über diese Schnittstelle werden Anfragen empfangen und deren Ergebnisse verschickt. Somit beschreibt diese Schnittstelle Dienste, die der Agent zur Verfügung stellt.

Um Anfragen erhalten zu können, muss der Agent sich bei einem Supervisor (vgl. Event-Manager) registrieren. Hierbei teilt er dem Event-Manager mit, welche Art von Anfragen (vgl. Event-Typ) er bearbeiten kann. Dieses Abonnieren für bestimmte Events kann als das Anbieten (bzw. Registrieren) seiner Dienste gesehen werden. Falls von einem Agenten eine Anfrage in Form eines Events gestellt wird, wird sie dem Agenten zugestellt, der sich für solche Events registriert hat. Nach der Verarbeitung der Anfrage wird das Ergebnis vom Agenten veröffentlicht. Dies geschieht indem er das Resultat in ein Event verpackt und es dem Supervisor übergibt. Die Agenten, die sich für solche Ergebnisse interessieren, d.h. die diese Events abonniert haben, werden die Ergebnisse von dem Supervisor zugestellt bekommen. Somit liegt die Verantwortung über die verschickten Events nicht bei den Agenten, die diese Events generieren, sondern es ist die Aufgabe des Supervisors (in dem Fall des Event-Managers), der diese Events letztendlich zustellt. Die Agenten, die diese Events erhalten wollen, müssen für solche beim Event-Manager registriert sein.

Agenten können evtl. verschiedene physikalische Geräte enthalten. So können Agenten über Sensoren oder verschiedenartige Endgeräte verfügen. Dieses Hardware kann von den Agenten genutzt werden um z.B. die Veränderungen in der Umgebung zu registrieren oder gewünschte Aktionen dieser Geräte (vgl. Gerätesteuerung) hervorrufen zu können (siehe Kapitel 4.1.1).

[3] Die ganze Kommunikation in diesem System findet über den Event-Manager statt, sodass die Agenten unabhängig voneinander arbeiten. Aus diesem Grund wird der Austausch bestimmter Komponenten (bzw. die Erweiterung des Gesamtsystems durch weitere Komponenten) erleichtert. Es müssen lediglich die Schnittstellen (vgl. Event-Typen), die durch die Events repräsentiert sind, eingehalten werden.

Die Struktur des Verarbeitungsteils eines Agenten ist nicht vorgeschrieben und den Entwicklern überlassen. Agenten können über Persistierungsmechanismen verfügen, um bestimmte Daten dauerhaft ablegen zu können. Möglich ist auch die Verwendung anderer Softwarelösungen, die für die Lösung bestimmter Probleme von Agenten in Anspruch genommen bzw. in jene integriert werden können.

Es ist u.a. nicht zwingend, dass die jeweiligen Agenten in einer bestimmten Programmiersprache implementiert sind. Die Plattformheterogenität und Sprachunabhängigkeit der Agenten ist in diesem System sogar erwünscht (vgl. nicht-funktionale Anforderungen in Kapitel 3.5.3). Da die Kommunikation mittels Event-Austausch erfolgt, müssen die Agenten lediglich in der Lage sein Events zu empfangen bzw. zu verschicken und diese richtig interpretieren zu können. Um dies zu erreichen müssen Events in einem vordefinierten Format vorliegen, das von jedem Agenten verstanden werden kann. Falls ein Agent andere Datenformate für seine interne Verarbeitung benötigt, die sich von denen unterscheiden, die das Event enthält, besteht seine Aufgabe darin, die Daten in das für ihn verständliche Format zu transformieren. Hierbei können bestimmte Adapter helfen, die das Empfangen und Senden der Events übernehmen, und die mit den Events übermittelten Daten in die für den Agenten verständliche Datenformate transformieren. In solchen Fällen ist auch eine Rücktransformation beim Senden der Events erforderlich. Solche Adapter können als Vermittler zwischen dem Supervisor und dem Agenten gesehen werden. Sie registrieren Agenten bei dem Supervisor für bestimmte Events, erhalten diese, transformieren sie in das für den Agenten verständliche Format und stellen dieses dann dem Agenten zu. Nach der Verarbeitung wird der entsprechende Gegenprozess angestoßen, bei dem die Ergebnisse des Agenten von dem Vermittler übersetzt und als Event dem Supervisor übergeben werden. Dieser Vermittler kann als ein weiterer Agent implementiert werden, dessen Dienste die gerade beschriebenen Leistungen eines Vermittlers darstellen. Solche Vermittler können als Brücke zwischen zwei unterschiedlichen „Welten“ (Umgebungen) verwendet werden, in denen eine bestimmte Technologie „herrscht“, sodass die Agenten aus verschiedenen Welten sich nicht direkt verständigen können. Die Entwicklung dieser Adapter wird in dieser Arbeit aus Zeitgründen nicht weiter verfolgt. Dies ist ein eigenständiges Forschungsgebiet geworden und wird an der HAW weiter vertieft.

Im Allgemeinen wird empfohlen die einzelnen Agenten, die kleine, aber in sich abgeschlossene, Aufgaben lösen können, so einfach wie möglich zu gestalten. So stehen ihre Dienste verschiedenen Agenten zur Verfügung und können somit in unterschiedlichen Anwendungen eingesetzt werden. Dies entspricht dem Wiederverwendbarkeitsprinzip.

4.1.4. Dienste

Dienste werden in diesem System als Leistungen der jeweiligen Agenten repräsentiert. Diese können von einem oder mehreren Agenten in Anspruch genommen werden, die auf ihre eigenen Zielen orientiert handeln. Eine Anwendung besteht aus einer Sammlung verschiedener Dienste, die von den hierzu gehörigen Agenten angeboten werden. Demnach wird eine Anwendung durch die Zusammenarbeit einer Gruppe von Agenten dargestellt.

In Kapitel 4.1.1 wurden drei Arten von Agenten identifiziert. Sie können Dienste folgender Art anbieten:

- Registrierung und Meldung von aufgetretenen Ereignissen
- Analyse und Interpretation der Ereignisse
- Erzeugung der Handlungsschritte (bzw. auszuführenden Aktionen)
- Entgegennahme von Anweisungen und Steuerung der End-Geräte

Der erste Punkt beschreibt die Erfassung der Änderungen in der Umgebung. Dies geschieht mittels Sensoren. Die erfassten Daten werden zur nächsten Verarbeitung weiter gereicht.

Punkt zwei beschreibt die Kontext-Erkennung, wobei der dritte Punkt die Erzeugung von Handlungsschritten anhand des ermittelten Kontextes darstellt. Dienste aus dem zweiten und dritten Punkt können eventuell von einem Agenten zur Verfügung gestellt werden. Im allgemeinen beschreibt jedoch der zweite Punkt die kontextuelle Interpretation der erfassten Ereignisse. Dies ist eine Aufgabe der Agenten, die für die Kontext-Ermittlung zuständig sind. Demnach sollen solche Dienste nur von diesen Agenten angeboten werden. Die Agenten sollen möglichst nur diese Aufgabe lösen, nämlich die Kontext-Erkennung. Hiermit wird erreicht, dass die Kontext-Ermitlung nur einmal stattfindet, die von unterschiedlichen Anwendungen genutzt werden kann.

Der letzte Punkt beschreibt die Ausführung von Aktionen, die für die gegebene Situation vom System als passend ermittelt wurde. Dies kann die Steuerung von Geräten (bzw. Ausgabegeräten) sein, wie z.B. einem Drucker, um dem Anwender die Ergebnisse in textueller Form auf dem Papier zu präsentieren, oder einem Lichtschalter, um beispielsweise das Licht Ein- oder Auszuschalten.

Es gibt keine feste Restriktion, welche Dienste möglich sind und wie sie auszusehen haben. Jedoch wird empfohlen die Agenten möglichst schlank für eine kleine in sich abgeschlossenes Aufgabe zu modellieren. Auf dieser Weise wird die Wiederverwendbarkeit dieses Agenten erhöht und somit die Erweiterung des Gesamtsystems erleichtert. Die Kommunikation zwischen den einzelnen Agenten findet nur über den Event-Manager statt. Dies Ermöglicht

einen einfachen Austausch der jeweiligen Agenten und erhöht die Flexibilität des Gesamtsystems[4].

4.1.5. Kontext-Ermittlung

Kontext-Erkennung erweist sich als zentrale Aufgaben in Systemen für intelligente Räume. Werden Ereignisse von den Sensoren erfasst, so werden die gewonnenen Rohdaten zur weiteren Verarbeitung an die *Kontext-Agenten* übermittelt. Diese analysieren die Daten und fassen sie zu einem Kontext zusammen. Dabei können Daten unterschiedlicher Quellen (bzw. Sensoren) ausgewertet und schließlich zu einem einzigen Kontext zusammengeführt werden. In diesem Prozess können u.U. mehrere Kontext-Agenten beteiligt sein, sodass diese auch unterschiedliche Kontext-Interpretationen liefern können, die sich aber gegenseitig nicht widersprechen sondern eher einander vervollständigen. Je nach Anwendungsart können unterschiedliche Kontext-Interpretationen gewünscht werden.

Ein Beispiel verdeutlicht den oben beschriebenen Vorgang. Ein RFID-Sensor registriert den Aufenthalt des Bewohners im Wohnzimmer. Der gewonnene Kontext wäre also „Bewohner befindet sich im Wohnzimmer“. Als Folge dessen wird der Fernseher mit dem Lieblingsprogramm des Bewohners angeschaltet. Ein weiterer Sensor meldet, dass es im Wohnzimmer dunkel ist. Die Kombination des bereits ermittelten Kontext und der neuen Daten ergibt einen weiteren Kontext „Bewohner sitzt im Dunkeln“. Auf Grund dieser Erkenntnis wird das Licht im Wohnzimmer vom System angeschaltet.

Wie an dem Beispiel zu erkennen ist, sind einheitliche Formate für den ermittelten Kontext erforderlich, damit verschiedene Anwendungen sie „verstehen“ können[5]. Diese sollen von den Entwicklern bereits zur Entwicklungszeit definiert werden. So kann der Kontext für die Location-Ermittlung folgende Felder besitzen:

- $ROOM\_ID$ - identifiziert das betroffene Zimmer
- $POSITION\_ID$ - beschreibt den Ort innerhalb des Raumes
- X - relative X-Koordinate
- Y - relative Y-Koordinate

Wie der Kontext letztendlich ermittelt wird, ist in dieser Entwicklungsphase nebensächlich. Zu diesem Zeitpunkt ist es wichtig sicher zu stellen, wie die Kommunikation zwischen

[4] vgl. die Anonymität der Agenten und somit eine vereinfachte Systemerweiterung

[5] Evtl. werden diese Kontext-Informationen zu weiteren Kontext-Erzeugungen berücksichtigt. Dies erfordert weiterhin geeignete einheitliche Datenformate.

den Sensor-Agenten, Kontext-Agenten, Aktion-Agenten und weiteren anwendungsrelevanten Dritt-Agenten auszusehen hat und welche Datenformate für die Kontext-Erfassung verwendet werden. Wie bereits oben beschrieben, findet die Kommunikation mittels des Events-Austausches durch den Event-Manager statt. Die Datenformate dieser Events können beliebig gestaltet werden. Sie müssen lediglich zuvor definiert werden, um dies in der weiteren Entwicklung berücksichtigen zu können.

Es gibt mehrere Möglichkeiten, wie die Kontext-Ermittlung realisiert werden kann (siehe Kapitel 2.2.1). Die Festlegung auf ein bestimmtes Verfahren ist zu diesem Zeitpunkt weder notwendig noch möglich, da dies ein aufwendiges Vorgehen ist und den Rahmen dieser Ausarbeitung sprengen würde. Durch die Gegebenheiten der gewählten Systemarchitektur können die hierfür zuständigen Agenten zur beliebigen Zeitpunkt ausgetauscht werden. Diese Architektur ermöglicht sogar die Unterstützung mehrerer Verfahren zur Kontext-Ermittlung, da die einzelnen Agenten unabhängig agieren und ihre Kommunikation nur mittels definierter Events erfolgt.

4.1.6. Anwendungen

Das System besteht aus einer Zahl von Agenten, die bestimmte Dienste anbieten. Die Agenten können je nach ihrer Dienstart der einen oder anderen Agenten-Gruppe angehören. Es werden vier Gruppen in dieser Architektur unterschieden:

- *Sensor-Agenten* - das sind Agenten, die Änderungen in der Umgebung registrieren und diese weiter melden.
- *Kontext-Agenten* - diese Agenten sind für die kontextuelle Interpretation der von den Sensor-Agenten empfangenen Daten zuständig (siehe Kapitel 4.1.5).
- *Prozess-Agenten* - diese analysieren den gewonnenen Kontext und generieren anhand dessen passende Handlungsschritte bzw. Aktionen. Sie sind wie die Kontext-Agenten vom Typ Dritt-Agenten.
- *Aktion-Agenten* - das sind die Akteure, die die von den Prozess-Agenten erzeugten Handlungsschritte ausführen.

Die Kommunikation zwischen den Agenten findet auf der Basis des Event-Austausches statt und wird von einem Event-Manager unterstützt (vgl. Kapitel 4.1.2). Abbildung 4.5 präsentiert die konzeptionelle Sicht dieser Architektur.

Eine Anwendung im Sinne der gewählten Architektur wird als ein Zusammenspiel bestimmter Sensor-, Kontext-, Prozess-, Aktion- und weiterer Dritt-Agenten verstanden. Dabei registrieren die Sensor-Agenten die Veränderungen in der Umgebung. Die Kontext-Agenten

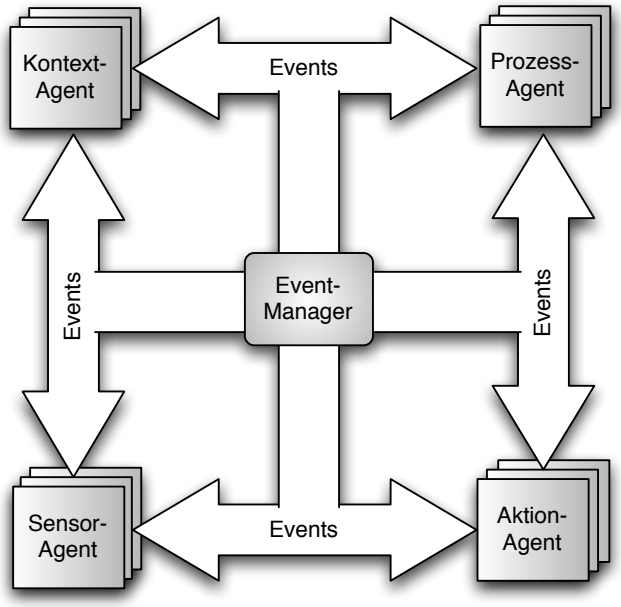


Abbildung 4.5.: Konzeptionelle Sicht der Architektur

analysieren die aufgetretenen Ereignisse und ermittelt daraus einen Kontext (vgl. kontextuelle Interpretation). Anhand des gewonnenen Kontextes werden von den Prozess-Agenten und evtl. mit Hilfe weiterer Dritt-Agenten passende Aktionen generiert, die von den Aktion-Agenten ausgeführt werden. Abbildung 4.6 stellt die Anwendungen schematisch dar. Hierbei repräsentieren Linien einer Farbe eine bestimmte Anwendung.

Wie in der Abbildung 4.6 zu sehen ist, können Agenten unterschiedlicher Anwendungen ihre Dienste anbieten (vgl. Wiederverwendbarkeitsprinzip). Um fest zu halten, für welche Anwendung der Agent gerade seine Arbeit leistet, können die Anwendungseingaben mit den Events übertragen werden. Auf dieser Weise können anwendungsspezifische Unterschiede in der Verarbeitung realisiert werden. So können beispielsweise Agenten gezielt für bestimmte Anwendungen unterbunden werden, die sonst ihre Arbeit in gleichen Fällen leisten.

Nach dem Wiederverwendbarkeitsprinzip können Agenten mehreren anderen Agenten ihre Dienste anbieten. Um die einzelnen Prozesse voneinander zu unterscheiden[6], müssen diese Prozesse identifizierbar sein. Somit werden die Events um ein weiteres Feld *Prozess-ID* erweitert. Auf diese Weise können Agenten die Prozesse eindeutig identifizieren.

[6]Die Kommunikation nach der gewählten Architektur verläuft mittels Event-Austausches (siehe 4.1.2). Deswegen kennen die Sender ihre Empfänger nicht und umgekehrt. Zudem können Agenten mehrere Clients unterhalten. Aus diesem Grund ist es wünschenswert für den Agenten zu wissen, ob das empfangene Event tatsächlich die Antwort auf die von ihm gestellte „Frage“ ist.

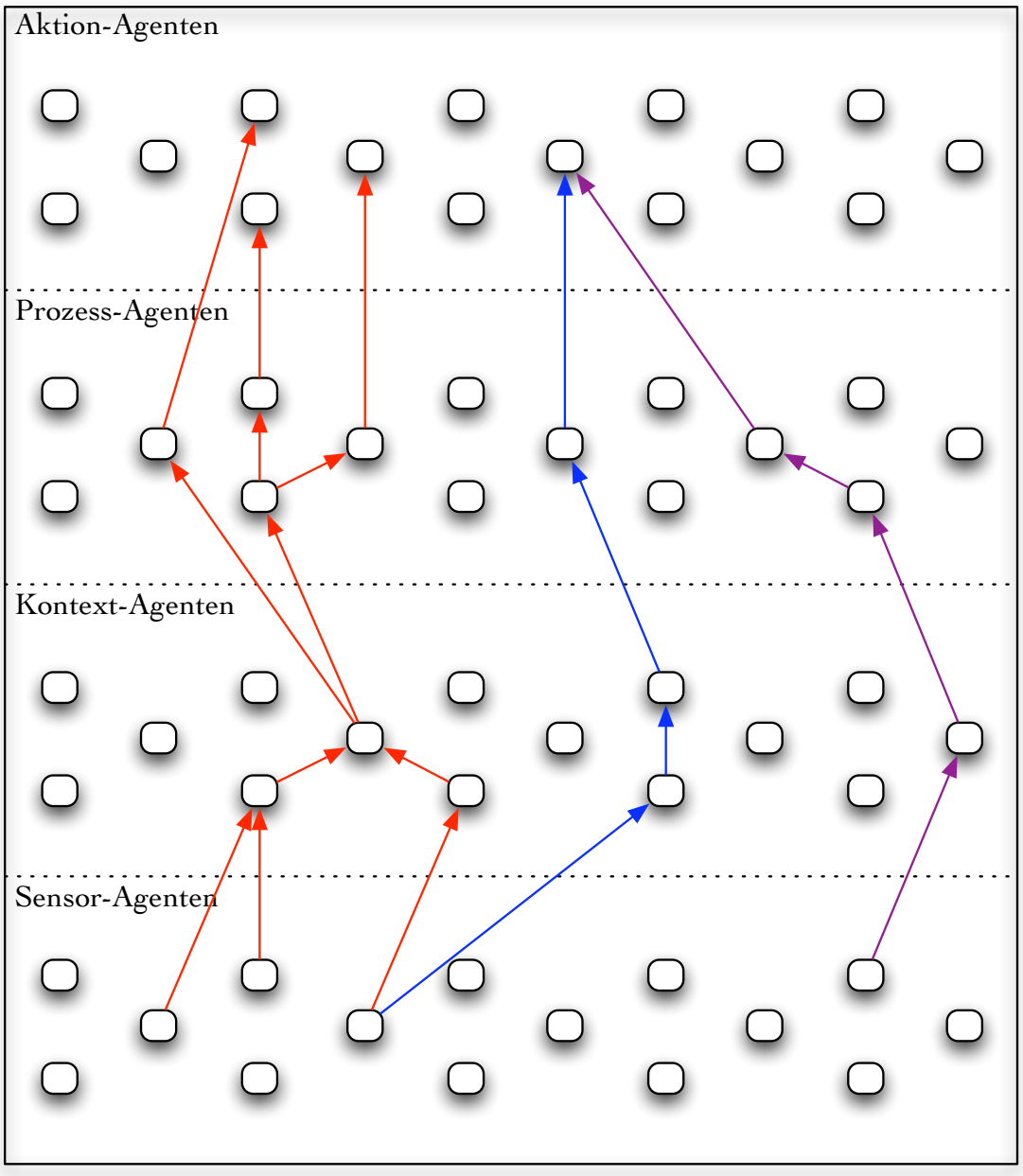


Abbildung 4.6.: Anwendung als Zusammenspiel verschiedener Agenten

Die in diesem Kapitel vorgestellte Architektur kann je nach Anwendungsfall modifiziert werden, um bestimmten Anforderungen gerecht zu sein. So kann beispielsweise gewünscht sein, dass nur Agenten mit bestimmten Charakteristika auf die Events reagieren. Zu solchen Eigenschaften eines Agenten kann z.B. seine aktuelle Position im Raum zählen. Ein Anwendungsbeispiel hierfür wäre Steuerung des Lichtschalters in einem bestimmten Raum oder sogar an einem bestimmten Ort innerhalb des Raumes. Um dies zu ermöglichen müssen die Agenten die Location in ihrer Verarbeitung miteinbeziehen. Die gewünschten Location-Eingaben müssen deshalb ein Bestandteil des Event sein (vgl. 4.1.5).

Auf die Gestaltung der Location-Eingaben kann in dieser Arbeit nicht weiter eingegangen werden. Die Location-Eingaben sollen im Allgemeinen einen hierarchischen Charakter aufweisen (vgl. baumartige Strukturen), sodass eine bestimmte Location unterschiedlichen Location-Eingaben entsprechen bzw. sie implizieren kann. Ein Beispiel hierfür wäre die Steuerung einer Stehlampe, die sich in einer Ecke des Wohnzimmers befindet. Diese Lampe kann sowohl durch die Eingabe des bestimmten Positionsortes (vgl. Ecke) als auch durch die Eingaben des Raumes insgesamt (vgl. Wohnzimmer) angesprochen werden. Diese detailliertere Location-Eingabe soll priorisiert werden, um unerwünschte Kollisionen zu vermeiden.

Außer Location sind auch andere Restriktionen denkbar (z.B. Zeit). Diese können durch die Einführung weiterer Felder in den Events und ihre entsprechende Verarbeitung von den Agenten realisiert werden.

Anhand der unterschiedlichen Kontexte können verschiedene Aktionen von verschiedenartigen Anwendungen generiert werden, die sich widersprechen können. Eine geeignete Lösung hierfür wäre die Prioritätenvergabe diesen Anwendungen. Die Anwendungen (bzw. Agenten dieser Anwendung) mit niedrigen Prioritäten werden unterdrückt und die Anwendungen mit höheren Prioritäten werden bevorzugt und somit ausgeführt.

Die hier herauskristallisierten allgemeinen und anwendungsspezifischen Restriktionen werden in den Events mit übertragen[7] und müssen von den Agenten berücksichtigt werden. Das sind:

- *APPLICATION_ID* - identifiziert die Anwendung
- *PRIORITY* - Priorität einer Anwendung
- *PROCESS_ID* - ordnet die Events einem Prozess zu
- *LOCATION_ID* - beschreibt die Location

[7] Hier sind die Informationen gemeint, die für die Anwendung und deren Prozesse im allgemeinen wichtig sind. Die eigentlichen Verarbeitungsdaten, die von den Agenten benötigt werden und somit ihre Schnittstelle darstellen, werden zusätzlich zu diesen Restriktionen übermittelt.

- weitere mögliche Restriktionen...

Somit sollen die Agenten konfigurierbar sein. Hierfür kann ein Agent eine Tabelle mit den eingegebenen Restriktionen führen und so in der Lage sein, die ein- und ausgehenden Events richtig zu interpretieren bzw. zu verarbeiten. Diese Art der Konfiguration ist sowohl manuell als auch durch die Unterstützung eines graphischen Editors denkbar.

4.1.7. Debugging und Fehlersuche

So wie in einem beliebigen verteilten System ist die Suche nach den Fehlern im System auch hier erschwert. Die Fehlerursachen können in der Kommunikation zwischen den Agenten (vgl. Event-Manager und seine Arbeitsweise), in der Anwendung (z.B. durch Orchestrierung verschiedener Agenten können Deadlocks oder Lifelocks entstehen (vgl. [59] oder [12])) und in den einzelnen Agenten selbst (Fehler in seinen Prozeduren) liegen.

Um die Gründe und die Verursacher dieser Fehler festzustellen bzw. zu identifizieren, können bestimmte Agenten entwickelt werden, die das Arbeiten des System überwachen und die einzelnen Schritte der Anwendungen registrieren und protokollieren. Solche Agenten können sich für alle Events einer bestimmten Anwendung oder sogar des gesamten Systems registrieren. Auf diese Weise können sie den gesamten Trafik für eine bestimmte Anwendung oder für das Gesamtsystem überwachen. Bei jedem Empfang eines Events registrieren sie wann dieses empfangen wurde und protokollieren alle Informationen, die mit diesem Event übertragen wurden. Anhand dieses Protokolls kann der Entwickler herausfinden, wo die vermuteten Fehler entstehen können.

Solche Agenten setzen voraus, dass die Kommunikation über den Event-Manager einwandfrei funktioniert, da sie diesen auch selbst nutzen. Somit kann der Fehler in der Kommunikation von diesen Agenten nicht direkt erkannt werden. Für andere Fehlerursachen, wie Anwendung- oder Agentenfehler (vgl. oben), können solche Agenten die Entwickler unterstützen.

4.1.8. Fazit

In diesem Kapitel wurde die Architektur für ein System für intelligente Räume konzipiert und ihre Bestandteile erläutert. Hierbei wurden vier Agenten-Arten erkannt, deren Dienste im Zusammenspiel verschiedene Anwendungen repräsentieren können. Als Kommunikationsart zwischen den Agenten wurde eine Kommunikation auf der Basis von Event-Austausch gewählt, die der Kommunikation in einem Blackboard-System ähnelt. Diese Kommunikation wird von einer Komponente namens Event-Manager unterstützt. Die Agenten selbst wurden abstrakt beschrieben, sodass ihre innere Struktur nicht notwendig gleich aufgebaut sein

muss. Es muss lediglich eine Schnittstelle auf der Basis von bestimmten Event-Typen definiert werden, über die die Agenten miteinander kommunizieren können. Im Kapitel 4.1.6 wurden Vorschläge gemacht, welche Daten die Events transportieren sollen, damit die Agenten ihre Arbeit im Sinne einer oder verschiedener Anwendungen leisten können.

Kapitel 4.1 beschreibt die Architektur auf einer abstrakten Ebene. Um die Architektur zu erproben, wird in den nächsten Kapiteln die im Kapitel 3.5 vorgestellte Anwendung realisiert. Hierbei wird eine detailliertere Architektur konzipiert, die die einzelnen Agenten definiert und deren Zusammenarbeit bestimmt. Diese Architektur wird an die in Kapitel 4.1 vorgestellte Architektur angelehnt bzw. dient als Beispiel-Architektur für diese.

4.2. Cooking Agent

Dieses Kapitel beschäftigt sich mit dem Design der Anwendung Cooking Agent (siehe Kapitel 3.5). Dabei wird eine Architektur entworfen, die auf den Erkenntnissen aus dem Kapitel 4.1 basiert. Die vorher beschriebene Architektur wird auf Cooking Agent übertragen, sodass die neu modellierte Architektur eine mögliche Instanz dieser darstellt.

Nach der Analyse der in Kapitel 3.5.2 definierten funktionalen Anforderungen können folgende Vorschläge für Software-Komonenten gemacht werden. Jede einzelne Funktionalität wird von einem Agenten angeboten. Falls eine Funktionalität sich als komplex erweist, kann sie in verschiedene Dienste unterschiedlicher Agenten aufgesplittert werden, sodass diese Agenten gemeinsam diese Funktionalität bereit stellen. Der Vorteil der Granularität der Agenten ist ihre Wiederverwendbarkeit sowohl in derselben als auch in anderen Anwendungen.

Nach der Auswertung der funktionalen Anforderungen sind folgende Komponenten bzw. Agenten zu entwickeln (vgl. Kapitel 3.5.2):

- *RecipeManager* - ein Agent, der Rezepte verwaltet und die Suche nach ihnen ermöglicht.
- *Calendar* - ein Termin-Verwalter, der für die Terminplanung zuständig ist.
- *UserPofile* - dieser Agent verwaltet Benutzerprofile mit den allgemeinen Personeninformationen und den Anwender-Vorlieben.
- *AvailableProductDeterminer* - ermittelt die bereits vorhandenen Produkte.
- *GUI* - graphische Benutzerschnittstelle für die Kommunikation des Anwenders mit dem System.

- *CookingAgent* - dieser Agent ist das Herzstück der Anwendung. Er schlägt Kochrezepte anhand des Essenanlasses, der Vorlieben der Gäste und des Gastgebers, der vorhandenen Produkte und weiteren Faktoren vor (vgl. die Anforderungen aus dem Kapitel 3.5.2). Er stellt mit der Anwenderhilfe ein Menü zusammen und ermittelt anhand dessen die Einkaufsliste.

Der Bewohner stellt eine Anfrage an den GUI-Agenten nach einen geeigneten Menü. Der GUI-Agent fragt seinerseits den CookingAgent nach möglichen Rezepten. Dieser analysiert die gegebene Situation (vgl. Essenanlass, Vorlieben der Dinner-Teilnehmer, vorhandene Produkte und weitere Faktoren aus dem Kapitel 3.5.2) und schlägt geeignete Rezepte vor. Diese Kochrezepte werden in einer Liste von dem GUI-Agenten präsentiert, aus der der Gastgeber ein Wunschmenü zusammenstellen kann. Anschließend ermittelt der CookingAgent die Einkaufsliste anhand der Kochrezepte, deren Gerichte das Menü ergeben, und der vorhandenen Produkte. Das Menü und die Einkaufsliste können von dem GUI-Agent ausgedruckt werden. Das Drucken beliebiger Dokumenten kann von einem *PrintAgent* übernommen werden. Dieser Agent kann seine Druck-Dienste allen anderen Agenten zur Verfügung stellen, sodass dies dem Wiederverwendbarkeitsprinzip entspricht.

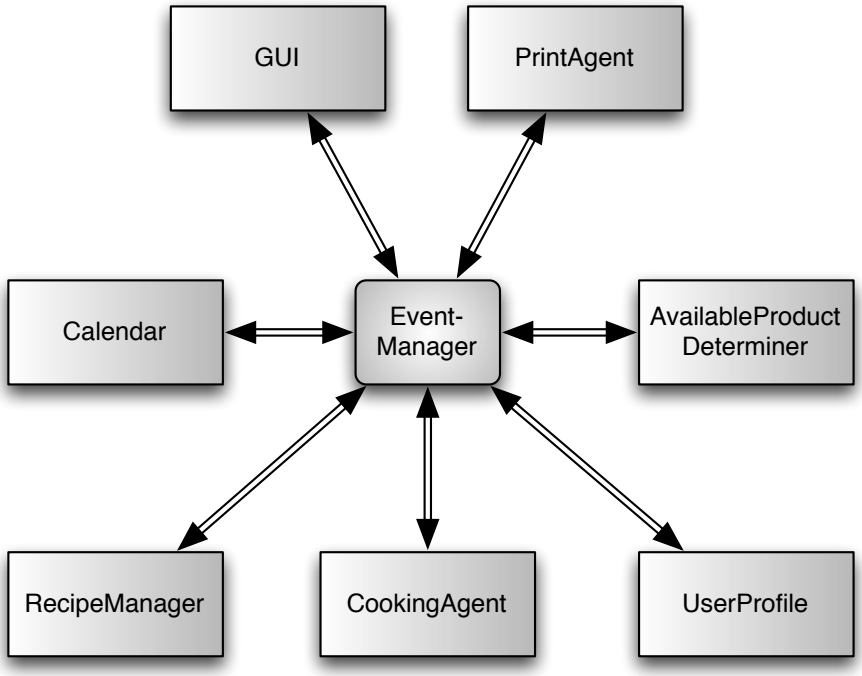


Abbildung 4.7.: Konzeptionelle Sicht der Architektur von Cooking Agent

Die Kommunikation zwischen den Agenten beruht auf dem Event-Austasch und wird von einem Event-Manager unterstützt bzw. verwaltet. Jeder Agent kann alle anderen Agenten mit-

tels Verschicken von Events erreichen. Das heißt, damit nur bestimmte Agenten auf bestimmte Anfragen reagieren und sie beantworten können, müssen Anfrage-spezifische Events definiert werden, für die nur die beteiligten Agenten sich registrieren und somit diese verarbeiten können. Die Definition solcher Events ist somit gleich der Schnittstellen-Spezifikation für diese Agenten. Dies soll in der Realisierungsphase geschehen und wird somit in dem nachfolgenden Kapitel beschrieben. Die hier ermittelten Agenten und deren Einsatz im System wird aus architektonischer Sicht in Abbildung 4.7 graphisch dargestellt.

Nach der Vergabe von Event-Typen bzw. einer Schnittstellendefinition ist ein möglicher Ablauf, wie er in Abbildung 4.8 in einem Sequenzdiagramm dargestellt ist. Hierbei soll beachtet werden, dass keine Aufrufe im Sinne von RPCs[8] stattfinden, sondern die Kommunikation zwischen den einzelnen Agenten asynchron veläuft und ausschließlich über die Events erfolgt.

Die hier vorgestellte Architektur nimmt erstmal keine Rücksicht auf bestimmte Programmiersprachen und die Implementierungsbesonderheiten der einzelnen Agenten. Die hiermit verbundenen Entscheidungen werden in den nächsten Kapiteln getroffen. Die Aufgabe dieses Kapitels bestand in der Identifizierung der beteiligten Agenten und der Erörterung deren Zusammenspiels in der Anwendung.

[8] *Remote Procedure Call* ist ein entfernter Prozeduraufruf, der bei der Realisierung von Interprozesskommunikation verwendet wird. Hierbei handelt es sich um eine direkte Kommunikation zwischen einem Client, dem Teilnehmer, der den Aufruf initiiert, und dem Server, der diese Prozedur anbietet [12].

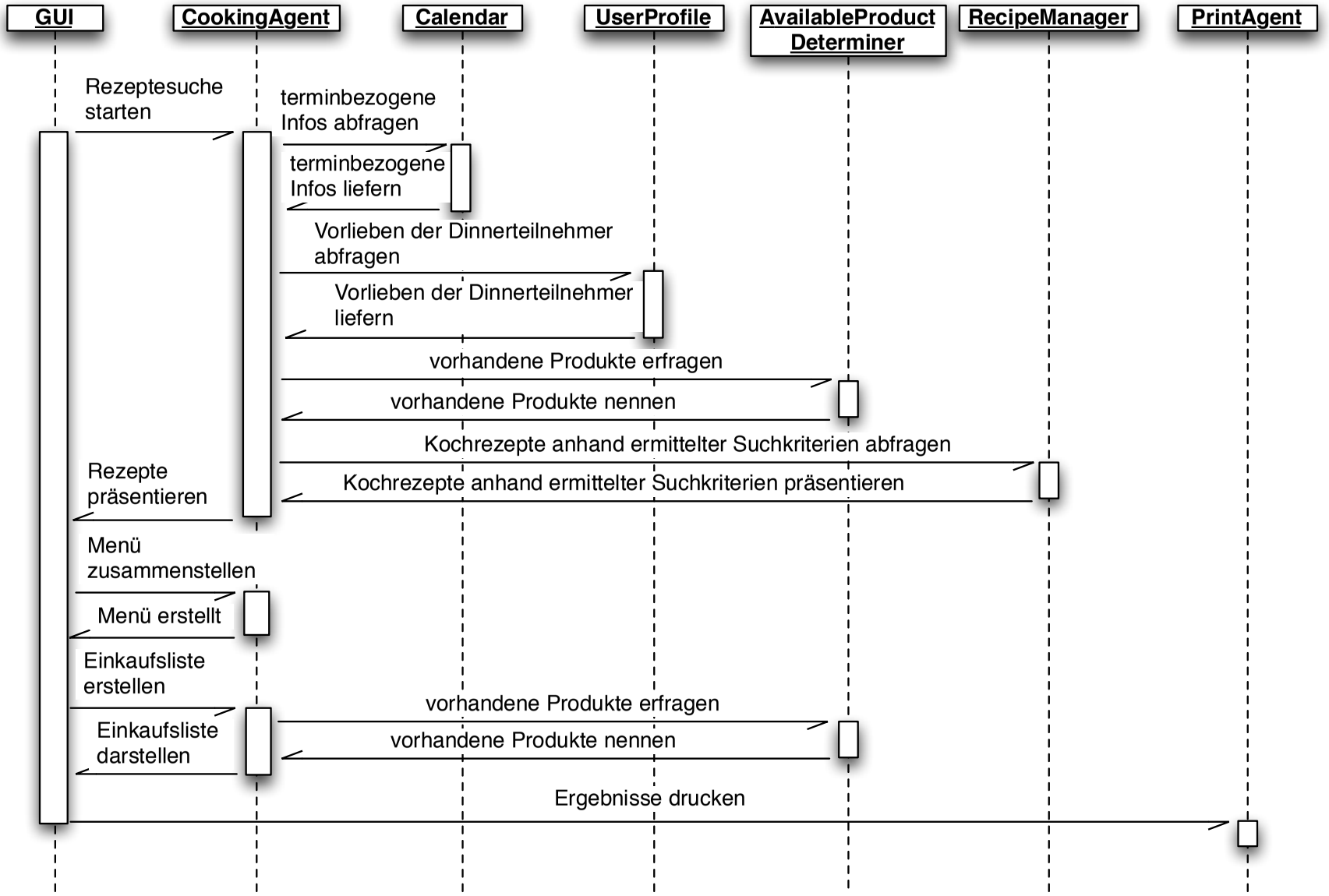


Abbildung 4.8.: Möglicher Ablauf der Cooking Agent Anwendung

5. Realisierung und Evaluation

5.1. Realisierung

In diesem Kapitel wird die Realisierung der einzelnen Agenten der Cooking Agent Anwendung beschrieben. Dabei werden die wesentlichen implementierungsrelevanten Entscheidungen erörtert.

5.1.1. Event-Manager

Als erstes muss der Event-Manager realisiert werden, da er eine zentrale Komponente des Systems darstellt. Der Event-Manager stellt die Kommunikation zwischen den Agenten her, bzw. unterstützt diese. Anhand der in Kapitel 4 beschriebenen Aufgaben des Event-Managers soll dieser folgende Eigenschaften aufweisen:

- *Agenten-Registrierung*: Der Event-Manager soll eine Schnittstelle anbieten für die An- und Abmeldung der Agenten.
- *Event-Registrierung*: Der Event-Manager soll den Agenten ermöglichen, sich für bestimmte Events zu registrieren.
- *Event-Zustellung*: Anhand der abonnierten Events soll der Event-Manager diese den Interessenten zustellen. Hierbei soll er u.a. die Event-Hierarchie berücksichtigen, sodass die Agenten, die sich für die höher in der Hierarchie stehenden Events registriert haben, auch die darunter liegenden erhalten[1].
- *Berücksichtigung prozess- und anwendungsspezifischer Event-Eigenschaften[2] bei der Event-Zustellung*: Events können sowohl für die Anwendung benötigte Nutz-Daten als auch allgemeine und anwendungsspezifischen Restriktionen enthalten (wie z.B.

[1] Wenn z.B. ein Agent sich für den Event *ALARM* registriert hat, soll er auch die Events vom Typ *FEUER_ALARM* empfangen können (vorausgesetzt, dass der Event-Typ *ALARM* oberhalb des Event-Typs *FEUER_ALARM* in der Event-Typ-Hierarchie steht).

[2] vgl. allgemeine und anwendungsspezifische Restriktionen in Kapitel 4.1.6

APPLICATION_ID, *PRIORITY*, *PROCESS_ID* und *LOCATION_ID* (siehe Kapitel 4.1.6)). Anhand der spezifizierten Restriktionen soll der Event-Manager die entsprechenden Events nur an die beteiligten Agenten zustellen. Auf diese Weise werden nur die Agenten solche Events empfangen, die sich sowohl für diesen Event-Typ registriert haben als auch an dem aktuellen Prozess beteiligt sind[3].

Nach einer Recherche wurde eine passende Software-Lösung für solchen Event-Manager gefunden[4]. Diese weist die wichtigsten beschriebenen Merkmale auf, sodass diese Lösung als geeignet für die zu implementierende Kommunikationskomponente gilt. Somit wird auf die eigene Implementierung des Event-Managers verzichtet und auf die bereits bestehende Lösung für diese Komponente zurückgegriffen. Im Folgenden wird diese Software-Lösung präsentiert.

iROS Event Heap

1999 wurde ein Projekt namens *iRoom* (*Interactive Room*) an der Stanford University ins Leben gerufen, dessen Anfangsziel darin bestand, einen interaktiven Raum zu entwickeln, der einer Gruppe von Personen bei ihrem Brainstorming unterstützen soll [57]. Solche Meetings sollten mit der Unterstützung von 5 SMART Boards mit einem Touch-Screen, einem Projektor und einem hochauflösenden Bildschirm durchgeführt werden [36]. In diesem intelligenten Raum befanden sich auch kaufübliche Geräte wie z.B. Videokamera, schnurlose Steuergeräte usw., die miteinander über WLAN verbunden wurden (siehe [36]). Diese Geräte stellten die Hardware-Basis für iRoom. Der intelligente Raum bat somit bestimmte Dienste an, die durch diese Geräte repräsentiert wurden. Die Aufgabe der iRoom-Entwickler bestand darin, diese Dienste miteinander zu kombinieren um einen höheren Nutzungsgewinn zu erzielen. So sollte es möglich sein mit einem PDA die im intelligenten Raum befindlichen Geräte zu steuern.

Im Rahmen dieses Projektes ist ein Framework entstanden, das die Koordination und Kommunikation innerhalb der Anwendungen unterstützt. Dieses Framework trägt den Namen *iROS* (*Interactive Room Operating System*). iROS ist ein Middleware, die die Koordinations-

[3] z.B. ein Agent, der für einen bestimmten Ort in einer bestimmten Anwendung Sonderhandlungen hat. Der Agent registriert sich für Events, die bestimmte Ort- und Anwendung-Informationen enthalten. Als Folge dessen wird dieser Agent nur die Events empfangen, die diesen bestimmten Ort als *LOCATION_ID* und diese bestimmte Anwendung als *APPLICATION_ID* beinhalten.

[4] Es gibt mehrere Implementierungen eines Event-Managers. In dieser Arbeit wird nur eine beschrieben, die auch für die Realisierung von Cooking Agent verwendet wurde. Die Betrachtung all solcher Implementierungen würde den Rahmen dieser Arbeit sprengen. Außerdem ist der Event-Manager nicht das Thema dieser Arbeit, sondern ist nur eine Software-Komponente, die lediglich die Kommunikation zwischen den Agenten herstellt. Somit können auch andere Implementierungen des Event-Managers für Systeme dieser Art verwendet werden.

und Kommunikationsinfrastruktur der Anwendungen des intelligenten Raums darstellt [6], [34]. Sie besteht aus drei Subsystemen [36]:

- *Data Heap* ist für die Datenverwaltung innerhalb der interaktiven Umgebung zuständig.
- *iCrafter* generiert die Benutzer-Interaktionsschnittstelle für die Steuerung beliebiger Geräte.
- *Event Heap* unterstützt die Koordination innerhalb dynamischer Anwendungen. Es bietet eine Kommunikationsinfrastruktur für Anwendungen der interaktiven Umgebung an.

Nach [36] ist Event Heap, der die Koordinations- und Kommunikationsinfrastruktur für Anwendungen innerhalb der interaktiven Umgebung darstellt, die einzige Komponente, die ein iROS-Programm verwenden muss.

Dem Event Heap liegt das *Tuplespaces-Modell* zu Grunde. Der Name *Tuplespace* (zu Deutsch *Tupelraum*) wurde von David Gelernter und Nicholas Carriero Mitte der 1980er Jahre mit der Entwicklung der Programmiersprache für verteilte Programmierung *Linda* eingeführt [75], [36]. Ein Tuplespace verwaltet *Tuples* (geordnete Werteansammlungen), die durch eine sortierte Menge von Schlüssel-Werte-Paaren repräsentiert sind. Diese Tuples können zu dem Tuplespace hinzugefügt und entfernt werden. Sie sind für alle Tuplespace-Anwender sichtbar, sodass sie von ihnen gelesen oder entfernt werden können (vgl. Blackboard-Modell in Kapitel 2.3.3). Mit Hilfe von Tuplespace können Prozesse innerhalb einer Anwendung miteinander kommunizieren, ohne identifizierende Informationen voneinander zu besitzen, und somit koordiniert werden.

Im Event Heap werden Tuples als Events bezeichnet. Sie erweitern das klassische Tuple. So sind Events hier selbstbeschreibende, typisierte Tuples. Die semantische Bedeutung der in dem Event enthaltenen Felder können aus ihren Namen gedeutet werden. Events mit gleichen Feldern aber vom unterschiedlichen Typ können andere semantische Bedeutungen aufweisen [36].

Auch der Event Heap wurde einigen Erweiterungen gegenüber dem klassischen Tuplespace unterzogen. Im Folgenden werden die wichtigsten Eigenschaften von iROS Event Heap zusammengefasst präsentiert:

Routing mit inhalt-basierter Adressierung Anhand der in den Events enthaltenen Felder können sie an die entsprechenden Empfänger zugestellt werden.

Verwendung von Routing-Standardfeldern Verwendung gleicher Routing-Felder sorgt für die Portabilität von Anwendungen in verschiedene interaktive Umgebungen.

Event-Zustellung-Garantie Event Heap garantiert die Zustellung von Events an die interessierten Empfänger. Im Gegensatz zum klassischen Tuplespace, wo die *pull*-Methode[5] verwendet wird, basiert die Zustellung der Events auf der *push*-Methode, bei der das System die Verantwortung für die Zustellung übernimmt.

Zeitbegrenzte Persistierung Events haben eine bestimmte Lebensdauer, sodass sie innerhalb dieses Zeitraums zugestellt werden müssen. Nach Überschreitung dieses Zeitlimits werden sie gelöscht.

Selbstbeschreibende Events Felder in den Events werden so benannt, dass deren semantische Bedeutungen aus ihren Namen ablesbar sind.

Flexible Event-Typisierung Flexible Event-Typisierung ermöglicht eine Unterscheidung gleichstrukturierter Events. Hiermit wird ihre Fehlinterpretation vermieden. Zudem gestattet dieses Feature Events neuer Versionen (aber vom selben Typ) zu akzeptieren.

Einfache Client-API Event Heap bietet eine einfache Schnittstelle für die Registrierung der Teilnehmer und Abonnierung von Events.

FIFO, At Most Once Reihenfolge Das klassische Tuplespace-Modell sieht keine Reihenfolge für den Empfang von Events vor. Zudem können die selben Events mehrfach an einen Empfänger zugestellt werden. Der Event Heap verhält sich anders: Die Events werden in der Reihenfolge zugestellt in der sie in dem Heap abgelegt worden sind. Der Empfänger wird höchstens ein Mal benachrichtigt.

Ausfall-Tolerierung Ausfall eines Teilnehmers (bzw. eines Sub-Systems) sorgt nicht für den Ausfall des Gesamtsystem.

Die Kommunikation zwischen dem Event Heap und den Clients findet mittels des *Wire Bundle Protocol* statt [35]. Dieses Protokoll baut auf dem Standard-Socket mit Verwendung von TCP/IP auf. Die Verbindung, über die *Wire Bundles* gemäß des Protokolls verschickt werden, ist unidirektional. Jeder *Wire Bundle* hat eine eigene semantische Bedeutung (vgl. Performativen in FIPA ACL), mit deren Hilfe die Funktionalität des iROS Event Heap realisiert ist (z.B. das Einfügen oder Löschen von Events). Eine ausführlichere Beschreibung dieses Protokolls findet sich bei [35].

Der erste Prototyp vom iROS Event Heap wurde in der Programmiersprache C++ geschrieben. Ab der zweiten Version namens Event Heap v1 wird er in Java implementiert, wobei diese Version noch eine C++ -Unterstützung anbietet. Die aktuelle Version Event Heap v2 wurde komplett in Java implementiert, sowohl Server als auch Client. Die Clients dürfen sich auf externen Rechnern befinden, sodass die Kommunikation zwischen dem Event Heap und den Clients über das Computer-Netzwerk erfolgt. Die aktuelle Version des iROS Event Heap

[5] Der Empfänger holt die Nachricht selbst ab.

ist 2.0 und unter http://mac.softpedia.com/get/Internet-Utilities/iROS.shtml erhältlich.

Da der iROS Event Heap eine Java-Schnittstelle anbietet, wird der erste Prototyp von Cooking Agent in der Programmiersprache Java realisiert.

Listings 5.1 und 5.2 präsentieren einen Beispiel-Code fürs Ablegen von Events auf dem Event Heap. Im ersten Listing wird ein Event erzeugt, das zwei Felder besitzt: AGE und NAME. Diese Felder werden mit den entsprechenden Werten gefüllt: **new** Integer(27) und "Tico␣Ballagas" (siehe Listing 5.1). Listing 5.2 zeigt das Hinzufügen eines Event-Templates zu dem Event Heap. Der Unterschied zum ersten Beispiels liegt darin, dass hier die Felder keine konkreten Werte enthalten, sondern nur deren Typen und die *Post-* und *Value-Types*. Die letzteren können eine der drei Ausprägungen sein: ACTUAL, FORMAL oder VIRTUAL [76]. ACTUAL bedeutet, dass sowohl der Feld-Name, der Feld-Typ als auch der Feld-Wert beim Matching-Prozess übereinstimmen müssen. Mit FORMAL werden nur der Feld-Name und der Feld-Typ beim Matching-Prozess berücksichtig[6]. VIRTUAL bedeutet, dass dieses Feld ignoriert wird, wenn ein Event mit dem Event-Template vergliechen wird.

Listing 5.1: Ablegen eines Events auf dem Event Heap [76]

```
import iwork.eheap2.*;

EventHeap eh = new EventHeap("localhost");

Event event = new Event("MyEventType");
event.addField("AGE", new Integer(27));
event.addField("NAME", "Tico␣Ballagas");

eh.putEvent(e);
```

[6] In diesem Beispiel muss nur sicher gestellt werden, dass Events die Felder AGE und NAME enthalten. Deren Werte werden nicht berücksichtigt.

Listing 5.2: Ablegen eines Event-Templates auf dem Event Heap [76]

```
import iwork.eheap2.*;

EventHeap eh = new EventHeap("localhost");

Event template = new Event("MyEventType");
template.addField("AGE", Integer.class,
   FieldValueTypes.FORMAL, FieldValueTypes.FORMAL);
template.addField("NAME", String.class,
   FieldValueTypes.FORMAL, FieldValueTypes.FORMAL);

eh.putEvent(e);
```

Listing 5.3 zeigt das Abonnieren eines Events von einem Client. Hierbei wird der Client (vgl. `this` im Beispiel) für einen bestimmte Event beim Event Heap registriert (siehe Listing 5.3). Um die gewünschten Events zu empfangen, muss der Client das Interface `EventCallback` und somit die Methode `returnEvent` implementieren (vgl. Listing 5.3).

Listing 5.3: Registrierung eines Clients für bestimmte Events [76]

```
Event templates[] = new Event[1];
templates[0] = event;
eh.registerForEvents(templates, this);

...

public boolean returnEvent(Event[] retEvents){
     try{
          System.out.println("registerForEvents: " +
          retEvents[0].getEventType());
     }catch(Exception e){
          e.printStackTrace();
     }
     return true;
}
```

Eine Alternative zur in Listing 5.3 aufgelisteten Registrierung stellt Listing 5.4 dar.

Listing 5.4: Alternative für Registrierung eines Clients für bestimmte Events [76]

```
Event templates[] = new Event[1];
templates[0] = event;
Event received = eh.waitForEvent(templates);
```

Die hier dargestellten Eigenschaften vom iROS Event Heap erweisen sich als ausreichend für dessen Verwendung in Cooking Agent. Er bietet Möglichkeiten zur Registrierung zum Empfang von gewünschten Events an und stellt sicher, dass diese auch zugestellt werden. Zudem ermöglicht er eine flexible Event-Typisierung, sodass eine Fehlinterpretation der Events ausgeschlossen wird. Diese und weitere Merkmale (siehe oben) des iROS Event Heaps sprechen für den Einsatz dieses Frameworks als Event-Manager in dem zu implementierenden System.

iROS Event Heap Adapter

Die in Kapitel 5.1.1 vorgestellten Methoden zum Verschicken und Abonnieren von Events müssen somit in jedem Client (in diesem Fall sind das Agenten) implementiert werden. Die Komponente, die das Senden und Empfangen von Events übernimmt, wird in diesem System als *iROS Event Heap Adapter* oder kurz *iROS Adapter* bezeichnet. Hiermit sollen alle beteiligten Agenten diesen beinhalten[7]. Das heißt, wenn eine bereits bestehende Applikation in das System integriert werden soll, muss sie lediglich einen solchen Adapter implementieren.

Mit der Auswahl des iROS Event Heaps kann das Architektur-Bild aus Kapitel 4.2 konkretisiert werden. Abbildung 5.1 zeigt die neu gewonnene Sicht auf Cooking Agent.

Der iROS Event Heap Adapter stellt eine Reihe von Methoden zur Verfügung, die das Abonnieren von Events und das Extrahieren der Nutzdaten aus diesen vereinfacht. Somit können Events auf eine einfache Art und Weise von Agenten abonniert werden. In Cooking Agent werden Event-Typen mit deren Feldern definiert, sodass sie für alle Agenten sichtbar sind. Diese werden in dem iROS Adapter implementiert. Das heißt, wenn ein neuer Agent in das System integriert werden soll, muss dessen Entwickler nur den iROS Adapter als Java-Projekt importieren um diesen an das System anzuschließen. Dem Entwickler stehen dann alle dem System bekannten Events zur Verfügung. Um den Überblick über alle möglichen

[7] Um die Unabhängigkeit von einer bestimmten Implementierung des Event-Managers zu gewährleisten, kann ein allgemeines Interface für solchen Adapter definiert werden. Auf dieser Weise kann der iROS Event Heap durch andere Implementierungen ersetzt werden. In diesem Fall wird nur eine Implementierung des entsprechenden Adapters notwendig sein. In Cooking Agent wird auf ein solches Interface aus zeitlichen Gründen verzichtet.

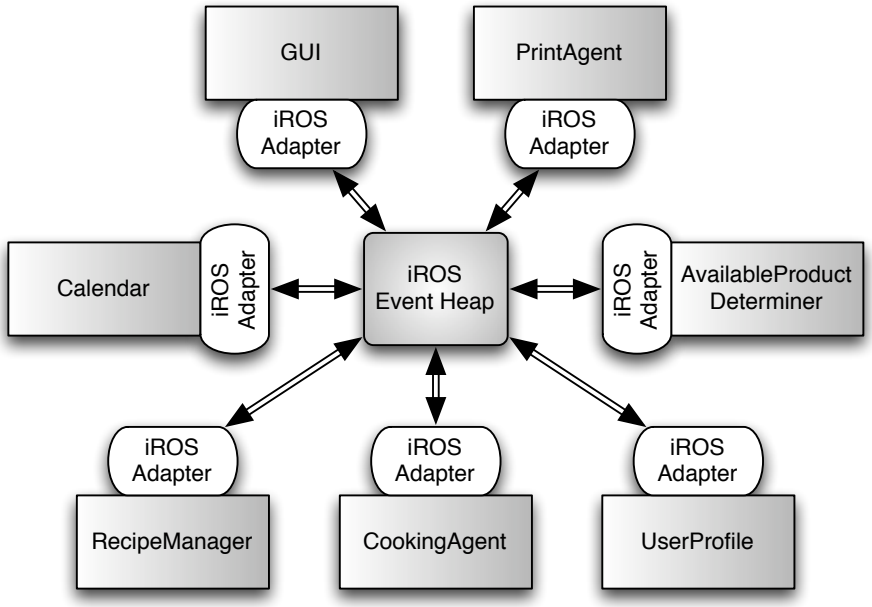


Abbildung 5.1.: Konzeptionelle Sicht der Architektur von Cooking Agent mit iROS Event Heap als Event-Manager

Events nicht zu verlieren, wurden die Events einer Anwendung in einem separaten Package abgelegt. Somit kann der Entwickler nur die gewünschten Packages importieren und muss nicht alle bereits existierenden Event-Typen laden.

Listing 5.5 zeigt eine Event-Hilfsklasse, die das Event initiallisiert und den Zugriff auf seine Felder erleichtert. Solche Event-Hilfsklassen werden in diesem Projekt als `Facade` (zu deutsch Fassade) bezeichnet. Jede Klasse beinhaltet einen Event vom bestimmten Typ (vgl. `TYPE_NAME`) und bietet die Zugriffsschnittstelle auf seine Felder an (in diesem Beispiel `FIELD_SHELF_ID`).

Listing 5.5: ShelfInventoryRequestEventFacade.java

```
import de.haw.iflat.eha.events.SimpleEventFacade;
import iwork.eheap2.*;

public class ShelfInventoryRequestEventFacade extends
   SimpleEventFacade {

 public static final String TYPE_NAME = "
    shelfInventoryRequestEvent";
 public static final String FIELD_SHELF_ID = "shelfID";
```

```
 public ShelfInventoryRequestEventFacade(Event event) throws
     EventHeapException {
   super(event);
   if (!event.getEventType().equals(TYPE_NAME))
     throw new RuntimeException("Illegal event type:" +
        event.getEventType());
 }

 public void setShelfID(String Id) throws
    EventHeapException {
   getEvent().setFieldValue(FIELD_SHELF_ID, Id);
 }

 public Object getShelfID() throws EventHeapException {
   return getEvent().getPostValue(FIELD_SHELF_ID);
 }
}
```

Diese Fassade ist für die Events vom Typ `shelfInventoryRequestEvent` zuständig. Solche Events repräsentieren eine Anfrage für vorhandene Produkte (vgl. Kapitel 5.1.7). Die Antwort auf diese Anfrage wird durch einen Event anderes Typs repräsentiert. In diesem Fall werden solche Events von der Fassade `ShelfInventoryResponseEventFacade` (siehe Listing 5.6) angeboten. Diese Events liefern die von einem RFID-Reader registrierten Artikel (bzw. deren RFIDs, vgl. `RFIDTags`) und die Regal-ID (vgl. `shelfID`), in dem diese registriert worden sind.

Listing 5.6: ShelfInventoryResponseEventFacade.java

```
import de.haw.iflat.eha.events.SimpleEventFacade;
import iwork.eheap2.*;

public class ShelfInventoryResponseEventFacade extends
   SimpleEventFacade {

  public static final String TYPE_NAME  = "
     shelfInventoryResponseEvent";
  public static final String FIELD_SHELF_ID = "shelfID";
  public static final String FIELD_RFID_TAGS = "RFIDTags";
```

```
  public ShelfInventoryResponseEventFacade(Event event)
     throws EventHeapException {
       super(event);
       if (!event.getEventType().equals(TYPE_NAME))
         throw new RuntimeException("Illegal␣event␣type:" +
            event.getEventType());
  }

  public void setShelfID(String iD) throws
     EventHeapException {
       getEvent().setFieldValue(FIELD_SHELF_ID, iD);
  }

  public String getShelfID() throws EventHeapException {
       return (String) getEvent().getPostValue(
          FIELD_SHELF_ID);
  }

  /**
   * Sets a List of RFIDTags.
   * @param tags
   * @throws EventHeapException
   */
  public void setRFIDTags(String[] tags) throws
     EventHeapException {
       getEvent().setFieldValue(FIELD_RFID_TAGS, tags);
  }

  /**
   * Returns a List of RFIDTags.
   * @return de.haw.smartshelf.reader.tags.RFIDTag
   * @throws EventHeapException
   */
  @SuppressWarnings("unchecked")
  public String[] getRFIDTags() throws EventHeapException {
     return (String[]) getEvent().getPostValue(
        FIELD_RFID_TAGS);
  }
}
```

Das hier vorgestellte Beispiel zeigt, dass zu jeder Anfrage zwei Event-Typen gehören: *Request-Events* und *Response-Events*. Die *Request-Events* repräsentieren eine Anfrage und enthalten Informationen, die für ihre Ausführung relevant sind. Agenten, die solche Anfragen verarbeiten können, müssen solche Events abonnieren, damit sie diese empfangen können. Listing 5.7 zeigt das Abonnieren von `ShelfInventoryRequestEvent`. Hierbei wird ein `EventFactory` verwendet, um einen solchen Event zu erzeugen (siehe Listing 5.7). Solche `Factory`-Klassen werden für jedem Adapter implementiert, der für eine bestimmte Anwendung zuständig ist und alle in dieser Anwendung benötigten Event-Typen enthält.

Listing 5.7: Abonnieren von ShelfInventoryRequestEvent

```
...
EventHeapAdapterConfig ehaConfig = null;
try {
  ehaConfig = EventHeapAdapterConfig.getConfig(EHA_CONFIG)
     ;
  eha = new EventHeapAdapter(ehaConfig);
} catch (Exception e) {
  throw new EventHeapException("Could␣not␣initialize␣
     EventHeapAdapter.␣" + e.getMessage());
}

eha.registerForEvent(EventFactory.
   createShelfInventoryRequestEvent(), this);
...
```

Die *Response-Events* stellen somit die Antwort auf eine Anfrage dar. Wenn also ein Agent sich für solche Events interessiert, muss er sich für diese bei dem Event Heap registrieren (siehe Listing 5.8). Solche Agenten müssen nicht unbedingt diejenigen sein, die eine Anfrage gestellt haben und deswegen auch eine Antwort auf diese erwarten. Sie können auch ihr Interesse für solche Events zeigen um z.B. eine Aktualisierung durch zu führen oder um anwendungsrelevante Änderungen zu registrieren.

Listing 5.8: Abonnieren von ShelfInventoryResponseEvent

```
...
try{
  this.eha = new EventHeapAdapter(new
     EventHeapAdapterConfig(System.getProperty("user.dir")
      + File.separator + "config" + File.separator + "
     eventheapadapter.properties"));
```

```
}
catch (Exception e){
  throw new EventHeapException("Could␣not␣initialize␣
     EventHeapAdapter.␣" + e.getMessage());
}

this.eha.registerForEvent(EventFactory.
   createShelfInventoryResponseEvent(), this);
...
```

Damit der Anfrage-Steller sicher sein kann, dass die empfangene Antwort auf seine Anfrage sich bezieht, wird eine Prozess-Identifikation mit dem Event (vgl. `eventId` in Listing 5.9) übergeben, die diese Transaktion markiert. Das Feld, das diese ID speichert und den Zugriff auf diese anbietet, wird in der Klasse `SimpleEventFacade` implementiert, die die Oberklasse von allen `Facade`-Klassen darstellt. Wenn das Feld nicht gesetzt ist, wird der Agent alle Antwort-Events bekommen, andernfalls wird die Antwort mit einer bestimmten `eventId` ihm zugestellt. Der Agent, der eine Anfrage stellt und die Antwort auf diese erhalten möchte, muss auch die `eventId` beim Abonnieren der Antwort angeben, die mit der von der Anfrage übereinstimmt. Listing 5.9 zeigt das Verschicken eines *Request-Events* und das Abonnieren des entsprechenden *Response-Events*.

Listing 5.9: Anfrage-Stellung und Abonnieren für deren Antwort

```
...
private void sendShelfInventoryRequestEvent(String eventId
   ) throws EventHeapException {
  try {
    Event shelfInventoryRequestEvent = EventFactory.
       createShelfInventoryRequestEvent();
    ShelfInventoryRequestEventFacade inventoryEventFacade
       = new ShelfInventoryRequestEventFacade(
       shelfInventoryRequestEvent);

    if(eventId != null) inventoryEventFacade.setEventId(
       eventId);

    eha.putEvent(shelfInventoryRequestEvent);
  }catch (Exception e) {
    throw new Exception("Could␣not␣send␣
       ShelfInventoryRequestEvent.␣" + e.getMessage());
```

```
    }
  }
  ...
```

Wenn weitere Restriktionen (vgl. Kapitel 4.1.6) für die Kommunikation zwischen bestimmten Agenten relevant sind, kann die Klasse `SimpleEventFacade` um diese entsprechend erweitert werden.

Der iROS Event Heap Adapter vereinfacht den Umgang mit dem Event und bietet Methoden an, die die Event-Zustellung präziser machen (vgl. Restriktionen). Er enthält eine Sammlung von Events, die in einer Anwendung möglich sind. Somit stehen sie für die Entwicklung neuer Agenten zur Verfügung. Die Kenntnis über die Event-Typen und deren enthaltenen Felder erleichtert die Erweiterung des Systems um neue Agenten. Wenn das System um eine komplett neue Anwendung erweitert werden muss, soll der iROS Adapter um die in dieser Anwendung verwendeten Events erweitert werden, damit diese global bekannt werden und in anderen Anwendungen (bzw. von anderen Agenten) verwendet werden können. Dies ermöglicht, dass aus den bereits vorhandenen Anwendungen neue Anwendungen geschaffen werden können.

5.1.2. GUI

Für die Interaktion zwischen dem Anwender und der Anwendung Cooking Agent wird ein GUI-Agent implementiert. Er soll eine graphische Oberfläche zur Benutzung der Anwendung zur Verfügung stellen. In dem ersten Prototyp von Cooking Agent wird eine webbasierte Anwendung entwickelt, die diese Benutzer-Interaktion anbietet. Eine Web-Anwendung hat einige Vorteile gegenüber einer Desktop-Anwendung. So wird z.B. eine Installation des Programms vermieden, da eine Web-Anwendung in einem Web-Browser läuft und somit auf allen Rechnern, die einen Web-Browser installiert haben, benutzt werden kann. Andere Vorteile der Web-Anwendung sind für die Prototyp-Entwicklung eher uninteressant, sie können aber unter [38] eingesehen werden. In den zukünftigen Versionen von Cooking Agent können auch Steuerungsprogramme anderer Art entwickelt werden, die auf die Endgeräte angepasst sind. Es sind weitere Methoden für die Interaktion mit dem Cooking Agent denkbar. So kann beispielsweise die Anwendung per Sprache gesteuert werden. Diese können in Zukunft realisiert und in das System integriert werden. Die Verwendung mehrerer Alternativen zur Steuerung von Cooking Agent ist auch möglich, sodass sie als Agenten parallel laufen können (vgl. Kapitel4).

In dieser Version von Cooking Agent wird eine Java-Web-Anwendung mit Verwendung des Servlet-Containers *Tomcat*[8] entwickelt. Diese kann als ein weiterer Agent von Cooking Agent gesehen werden. Die Eingaben, die der Anwender mit Hilfe der Web-Oberfläche macht, werden in die entsprechenden Anfrage-Events überführt und dem Event Heap übergeben. Diese werden vom Agenten CookingAgent (siehe 5.1.8) empfangen, da er sich für die Events dieser Art registriert hat. Der CookingAgent verarbeitet die Anfragen und schickt deren Ergebnisse als Events an den Event Heap. Letzterer stellt diese dem GUI-Agenten zu, weil er solche abonniert hat. Schließlich werden die Ergebnisse auf der Web-Oberfläche dem Benutzer präsentiert. Der Anwender hat die Möglichkeit diese Ergebnisse auszudrucken. Hierfür wird der GUI-Agent Events schicken, die diese Aufforderung und das zu druckende Dokument beinhalten. Diese Events sind für den *PrintAgent* (siehe Kapitel 5.1.3) gedacht, von dem die übertragenden Dokumente gedruckt werden. Somit übersetzt der GUI-Agent die Benutzer-Eingaben in die entsprechenden Events und schickt diese an den Event Heap. Er empfängt die Antwort-Events und stellt die darin enthaltenen Resultate auf der Oberfläche dar.

5.1.3. PrintAgent

Der PrintAgent kann als ein Aktion-Agent gesehen werden, da er ein Endgerät, nämlich einen Drucker, besitzt. Seine Aufgabe besteht darin, Dokumente zu drucken. Somit ist er für die Events registriert, die vom Typ `PrintRequestEvent` sind. Diese Events können sowohl Dokumente als auch Informationen, wo diese abgelegt sind, enthalten. Die zweite Variante ist für große Dokumente sinnvoll, da Events eine begrenzte Daten-Menge übertragen können. Aus den empfangenen `PrintRequestEvent` werden die zu druckenden Dokumente extrahiert und an den Drucker geschickt, der über einen Drucker-Treiber vom Agenten gesteuert wird. Auf diese Weise wird das Drucken in der Anwendung Cooking Agent realisiert.

Mit dem Abonnieren der Events vom Typ `PrintRequestEvent` werden die zu druckenden Dokumente an den PrintAgent zugestellt. Der PrintAgent, so wie alle anderen Agenten im System, kennt den Sender nicht. Die Zustellung dieser Events wird vom iROS Event Heap übernommen, der das Weiterleiten an die Empfänger anhand deren Abonnements durchführt. Hiermit können verschiedene Sender (bzw. Agenten) solche Events initiieren. Das heißt, dass der Druck-Service allen im System bekannten Agenten zur Verfügung steht[9].

[8] *Tomcat* ist eine Umgebung, die die Ausführung von Java-Code auf Webservern bereit stellt (`http://tomcat.apache.org/`).

[9] An dieser Stelle soll daran erinnert werden, dass die Sender den PrintAgent ebenfalls nicht kennen. Sie schicken einfach ihre Druck-Anfrage an den Event Heap, und er übernimmt die Zustellung an die Interessenten (vgl. Kapitel 4.1.2 und Kapitel 5.1.1).

5.1.4. Calendar

Der Calendar-Agent verwaltet Termine mit den hierzu gehörigen Informationen (wie z.B. Datum, Ort, Teilnehmer etc.). Hierfür besitzt der Agent eine Datenbank, in der diese Informationen gespeichert werden. Der Datenbank-Zugriff erfolgt mit der Unterstützung von *Hibernate*[10].

Zu diesem Zeitpunkt können nicht alle benötigten Termin-bezogenen Informationen vorausgesehen werden. Aus diesem Grund wird eine Schlüssel-Werte-Tabelle für die zusätzlichen Attribute modelliert, sodass eine Erweiterung um weitere Termin-Attribute möglich ist.

Für die Kommunikation mit anderen Agenten, stellt der Calendar-Agent eine Event-Schnittstelle zur Verfügung. So ist dieser Agent für die zu verarbeitenden Anfragen (`RequestEvents`) registriert, und übergibt die entsprechenden `ResponseEvents`, nach der Verarbeitung der Anfragen, an den iROS Event Heap. Die Funktionalität des Calendar-Agenten kann in Zukunft erweitert werden. In diesem Fall muss die Event-Schnittstelle um weitere Events erweitert werden.

5.1.5. UserProfile

Der UserProfile-Agent ist ähnlich dem Calendar-Agent aufgebaut. So werden auch hier die Informationen über Personen (wie z.B. ihre Namen, Adressen, Vorlieben etc.) in einer Datenbank verwaltet. So wie der Calendar-Agent ist auch der UserProfile-Agent erweiterbar. Die Anfragen und deren Ergebnisse werden mit Hilfe von Events transportiert, die die Event-Schnittstelle dieses Agenten darstellen (vgl. Calendar-Agent).

5.1.6. RecipeManager

Der RecipeManager-Agent repräsentiert ein Sammlung von Kochrezepten. Er ermöglicht die Suche nach Rezepten anhand deren Attribute wie z.B. Namen oder die benötigten Zutaten. Für die Realisierung dieses Agenten wurde eine Software namens *RMS* (*Recipe Management System*) verwendet [64]. Sie entstand in Rahmen einer Diplomarbeit und ist ein Kochrezepte-Verwaltungssystem, das die benötigte Funktionalität anbietet, die RecipeManager aufweisen soll. Es stellt Funktionen zu Verfügung, wie z.B. das Speichern oder Ändern von Kochrezepten und die Suche nach ihnen mittels derer Attribute (vgl. [64]).

Um RMS in die Cooking Agent Anwendung zu integrieren, wird ein entsprechender iROS Event Heap Adapter implementiert. Dieser Adapter nimmt die Anfragen entgegen, die als

[10] *Hibernate* ist ein Open-Source-Persistenz- und ORM (*Object-Relational Mapping*)-Framework für Java. Mit dessen Hilfe lassen sich Java-Objekte auf eine vereinfachte Weise persistieren [31].

Events bei dem Event Heap abonniert werden, und wandelt sie in die RMS-spezifischen Anfragen um. Die RMS-Ergebnisse auf diese Anfragen werden vom Adapter wieder in die entsprechende Events verpackt und dem Event Heap übergeben. RMS wurde in Java geschrieben, sodass die Integration dieser Software in die Cooking Agent Anwendung mit einem relativ geringen Aufwand verbunden ist.

5.1.7. AvailableProductDeterminer

Der Agent AvailableProductDeterminer ist für die Ermittlung vorhandener Produkte zuständig. Er wird durch *Smart:Shelf* repräsentiert. Smart:Shelf ist, wie bereits in Kapitel 3.2.1 vorgestellt, eine Anwendung, die im Rahmen eines Projektes an der HAW im Jahr 2008 entstanden ist [66].

Diese Anwendung besteht aus drei wesentlichen Komponenten: *SmartShelf Shelf*, *SmartShelf DB* und *SmartShelf WebClient* (vgl. [66]). SmartShelf Shelf besitzt einen RFID-Reader, der RFID-Tags mit Hilfe seiner Antenne registrieren kann. SmartShelf DB verwaltet Informationen über die Gegenstände, denen diese RFID-Tags gehören. Hierfür wird eine Datenbank verwendet, die mit Hilfe von Hibernate angesprochen wird. In dieser Datenbank werden zu jeder RFID die Attribute des zugehörigen Gegenstands (wie z.B. `TYPE`, `NAME` u.a.) festgehalten. Die Speicherung dieser Attribute ist ähnlich wie in den Agenten Calendar und UserProfile durch eine Schlüssel-Werte-Tabelle realisiert. Somit können verschiedene Eigenschaften der Gegenstände in der Datenbank abgelegt bzw. um neue erweitert werden (vgl. [66]). SmartShelf DB kann anhand der ID eines RFID-Tags ermitteln, um welchen Gegenstand es sich handelt und seine Eigenschaften liefern. Die Interaktionen zwischen der Anwendung und dem Benutzer bietet SmartShelf WebClient. SmartShelf WebClient ist eine Web-Anwendung, die die Steuerung von Smart:Shelf mittels eines Web-Browser erlaubt.

Die Kommunikation zwischen diesen drei Komponenten verläuft über den iROS Event Heap[11]. Somit verfügen diese Komponenten über einen iROS Event Heap Adapter, der die gewünschten `RequestEvents` abonniert und diese Empfängt und die entsprechenden `ResponseEvents` dem Event Heap übergibt.

Smart:Shelf erlaubt die Suche nach Gegenständen anhand derer Attribute und kann die Lokation dieser ermitteln[12]. Jedoch fehlt die Funktionalität für die Ermittlung aller von Smart:Shelf registrierbaren Gegenstände. Um diese Funktionalität muss diese Anwendung erweitert werden, da diese von großer Bedeutung für Cooking Agent ist (vgl. funktionale Anforderungen von Cooking Agent aus Kapitel 3.5.2). Als Folge dessen wird SmartShelf DB um die Events `ItemInventoryRequestEvent` und

[11] Smart:Shelf war ein Pilot-Projekt, um die Kommunikation mittels Events mit Verwendung von iROS Event Heap zwischen Software-Komponenten zu erproben.

[12] Die Lokation-Ermittlung ist zu diesem Zeitpunkt auf das jeweilige Regal begrenzt.

`ItemInventoryResponseEvent` erweitert. Somit ermöglicht die Erweiterung der Smart:Shelf-Schnittstelle Gegenstände, mit deren Attributen eines bestimmten Regals oder aller registrierter Regale im System, zu erfragen. Diese Erweiterung erweist sich als genügend für die Verwendung von Smart:Shelf in der Rolle des AvailableProductDeterminer-Agenten in Cooking Agent.

5.1.8. CookingAgent

CookingAgent ist der zentrale Agent der Cooking Agent Anwendung. Dieser Agent macht die Entscheidungen, welche Kochrezepte zum aktuellen Zeitpunkt mit der gegeben Situation, passend sind und fasst vom Benutzer ausgewählte Rezepte zu einem Menü zusammen. Zudem ist dieser Agent für die Erstellung der Einkaufsliste zuständig (vgl. funktionale Anforderungen aus Kapitel 3.5.2).

Um diese Funktionalität zu implementieren, benötigt CookingAgent die Zusammenarbeit weiterer Agenten wie Calendar, UserProfile, RecipeManager und AvailableProductDeterminer (vgl. das Sequenz-Diagramm in Abbildung 4.8). Die Kommunikation mit diesen Agenten verläuft mit Hilfe eines iROS Event Heap Adapter, mittels dessen die entsprechenden `RequestEvents` und `ResponseEvents` verschickt bzw. empfangen werden. So sendet CookingAgent Anfragen in Form von `RequestEvents`, die von den zuständigen Agenten abonniert wurden, an den iROS Event Heap und registriert sich bei ihm für die entsprechenden `ResponseEvents`, die als Antworten auf die Anfragen zu verstehen sind. Auf diese Weise erhält CookingAgent die für seine Tätigkeit notwendigen Informationen.

Die eigentliche Entscheidung über passende Kochrezepte wird wie folgt realisiert. Zu Beginn werden Kochrezepte ermittelt, die den geschmacklichen Präferenzen der bei dem Dinner beteiligten Teilnehmern entsprechen. Hierfür werden vom Calendar-Agenten die Teilnehmer und von dem UserProfile-Agenten deren Vorlieben ermittelt. Anhand dieser Informationen wird die entsprechende Anfrage an den RecipeManager gestellt, der die hierfür passenden Rezepte liefert. Als nächstes werden Kochrezepte für die Gerichte ermittelt, die mit vorhandenen Produkten zubereitet werden können. Hierfür liefert der AvailableProductDeterminer die von ihm registrierten Gegenstände. Diese Gegenstände werden vom CookingAgent nach essbaren Produkten gefiltert. Dies geschieht, indem CookingAgent ihren Typ (vgl. `TYPE` als Gegenstand-Attribut) analysiert. Produkte werden mit dem Typ `FOOD` in SmartShelf DB abgelegt (siehe AvailableProductDeterminer). Der RecipeManager ermittelt in Folge dessen Kochrezepte, die diese Produkte als Zutaten beinhalten. Die somit gewonnenen Kochrezepte bilden die Basis-Menge für den Entscheidungsprozess des CookingAgent.

Um die gewonnene Menge deutlich zu reduzieren, da u.U. diese Menge sehr groß und deshalb unübersichtlich für den Anwender werden kann, werden die Kochrezepte analysiert und

die am meisten passenden beibehalten. Das Ergebnis dieser Reduktion soll eine überschaubare Menge von Kochrezepten sein, die als Basis für die Menü-Erstellung dem Anwender präsentiert wird. Diese Menge wird nach Erfüllungsgrad der gegebenen Bedingungen sortiert, sodass die Rezepte, die vom CookingAgent als am meisten passend empfunden wurden, oben in der Ergebnisliste erscheinen.

Das Ergebnis der Kochrezept-Analyse ist eine Kennzahl, die den Erfüllungsgrad der gegebenen Bedingungen für das jeweilige Kochrezept beschreibt. Auf diese Weise können Kochrezepte quantitativ gemessen werden, sodass sie vom System anhand mehrerer Faktoren verschieden gewichtet werden. Diese Gewichtung (bzw. Kennzahl) ist für die Entscheidung, ob das jeweiliges Kochrezept passend ist, ausschlaggebend. So werden Kochrezepte mit höheren Kennzahlen als passender vom System vermerkt und sie dem Anwender, nach Kennzahlen sortiert, für die Menü-Auswahl vorgeschlagen.

Die Kennzahl wird, wie in Formel 5.1 dargestellt, berechnet. Sie setzt sich aus der Summe der Kochrezepte-Eigenschaften, multipliziert mit einem entsprechenden Faktor, zusammen.

$$KZ = V * f_V + Z * f_Z + E * f_E + K * f_K + ... + X * f_X \tag{5.1}$$

Die Eigenschaften sind die Randbedingungen wie z.B. Vorlieben `V`, Zubereitungszeit `Z`, Einkaufsbereitschaft `E`, Kosten `K` u.a.. Diese haben einen bestimmten Wert, der entweder aus dem Kochrezept direkt ablesbar ist oder von einem anderen Agenten ermittelt werden kann. So werden z.B. geschmackliche Vorlieben (bzw. die Priorität des Rezeptes) vom UserProfile-Agenten und Einkaufsbereitschaft mit Hilfe der Agenten UserProfile und Calendar, die aus den Informationen wie allgemeine Einkaufsbereitschaft des Bewohners und die zur Verfügung stehende Zeit sich zusammen setzt, ermittelt. Der Faktor $f_{Eigenschaft}$ steht für die Gewichtung der jeweiligen Eigenschaft, sodass diese durch einen höheren Faktor hervorgehoben wird. Er wird entweder zur Konfigurationszeit der Anwendung vordefiniert oder dynamisch zur Laufzeit ermittelt[13].

Die Analyse kann in Zukunft um weitere Summanden (Eigenschaften bzw. Randbedingungen und deren Faktoren) erweitert werden. Auf diese Weise kann sie verfeinert werden und so präzisere Aussage über die Priorität der Rezepte treffen.

Nachdem der Gastgeber anhand der von CookingAgent vorgeschlagenen Kochrezepte ein Menü zusammengestellt hat, kann die Einkaufsliste erstellt werden. Hierfür wird der AvailableProductDeterminer nach vorhandenen Produkten befragt. Die fehlenden Zutaten werden als eine Differenz-Menge zwischen den Zutaten der bereits bestimmten Kochrezepte und den vorhandenen Produkte ermittelt, die in einer Einkaufsliste zusammengefasst werden.

[13] In dieser Phase der Entwicklung werden die Faktoren zunächst fest definiert.

CookingAgent ist ein Agent, der seine Aufgaben mit Hilfe anderer Agenten bewältigt. Die Kooperation zwischen den Agenten erfolgt, so wie bei allen Agenten in diesem System, ausschließlich durch die Event-Kommunikation (vgl. iROS Event Heap Adapter in Kapitel 5.1.1).

5.1.9. Entwicklungsstand

Die in den letzten Kapiteln beschriebene Realisierung wird in die Tat umgesetzt. Die Implementierung der Anwendung dient als Bestätigung in der Praxis (*Proof of Concept*) der theoretischen Überlegungen und deren Architektur. Hierbei werden die wesentlichen Module der Architektur realisiert, die für den praktischen Beweis unabdingbar sind.

Für die Agenten-Kommunikation über den iROS Event Heap Adapter wurden so genannte Adapter verwendet, die diese Kommunikation zwischen dem Event Heap und dem jeweiligen Agenten herstellen. Somit muss jeder Agent einen solchen Adapter besitzen (vgl. Kapitel 5.1.1). Um die Implementierung des Adapters zu vereinfachen, wurde ein abstrakter Adapter implementiert, der die wesentliche Funktionalität eines Adapters beinhaltet, sodass die von ihm abgeleiteten Adapter nur um ihre zusätzliche Funktionalität (vgl. Schnittstelle mit Hilfe von Events) erweitert werden müssen. Dieser Adapter erlaubt die Integration externer Anwendungen in das System.

Agenten wie Calendar, UserProfile und PrintAgent werden für den praktischen Machbarkeitsnachweis der Architektur nicht implementiert, da deren Entwicklung nur wenig zum Nachweis beitragen kann[14]. Für die Realisierung von AvailableProductManager wurde die Anwendung Smart:Shelf verwendet. Sie wurde um weitere Funktionen erweitert, sodass dieser Agent aus der Sicht von Cooking Agent eine ausreichende Funktionalität aufweist (siehe Kapitel 5.1.7). RecipeManager wurde mit Hilfe von einem Kochrezepte-Verwaltungssystem RMS realisiert. Hierfür wurde ein entsprechender iROS Event Heap Adapter gebaut, der die Integration dieser Anwendung in Cooking Agent ermöglicht (vgl. 5.1.6). Da die Agenten wie Calendar und UserProfile noch fehlen, soll der Agent CookingAgent zunächst ohne Informationen, die diese Agenten liefern sollen, arbeiten. So wurden zunächst die Entscheidungen über passende Rezepte anhand vorhandener Produkte getroffen (vgl. sec:funktionale-anforderungen). Der Agent GUI bietet zur Zeit nur die Möglichkeit vorhandene Produkte und aus diesen passende Kochrezepte darzustellen. Diese Funktionalität genügt zunächst um die Ergebnisse der vorhandenen Features von Cooking Agent zu präsentieren. Auf diese Weise kann die Richtigkeit der vorgeschlagenen Rezepte visuell überprüft werden.

Abbildung 5.2 stellt die Anwendung Cooking Agent mit Berücksichtigung der Realisierungsdetails und des aktuellen Entwicklungsstands dar. Hierbei repräsentieren transparente Agen-

[14] Die Realisierung weiterer Agenten gilt als genügend für den Machbarkeitsnachweis.

ten die noch nicht realisierten Komponenten. Die umrahmten Komponenten zeigen die für ihre Realisierung verwendete Software.

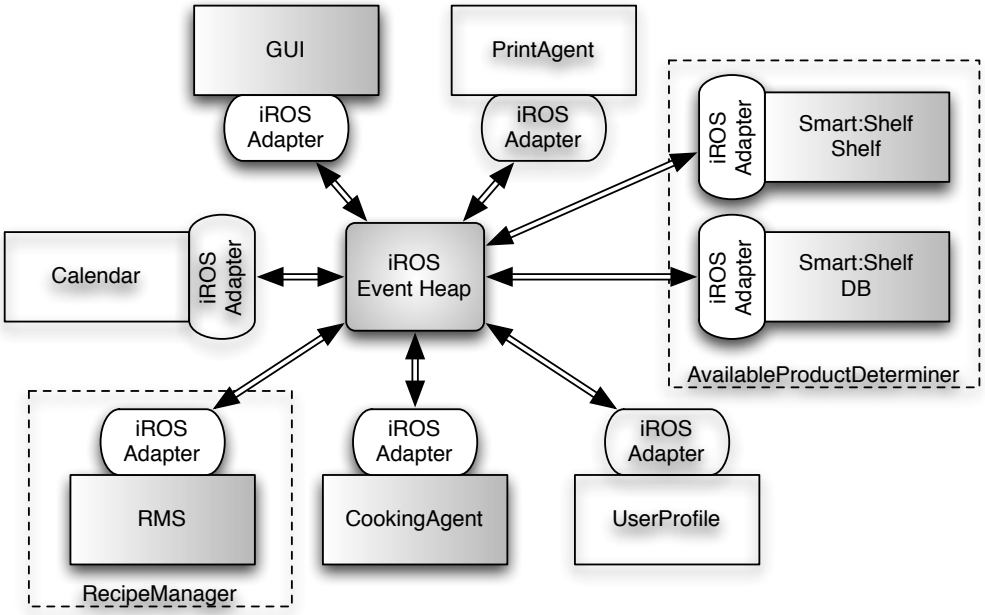


Abbildung 5.2.: Konzeptionelle Sicht der Architektur von Cooking Agent unter Berücksichtigung der Realisierungsdetails und des aktuellen Entwicklungsstands

Abbildung 5.3 zeigt die graphische Oberfläche von Smart:Shelf, die die von dem RFID-Reader registrierten RFID-Tags darstellt. Das in zwei Teile geteilte Fenster repräsentiert die jeweiligen zwei Smart:Shelf-Instanzen. In der Abbildung 5.4 wird die Oberfläche der RMS-Anwendung mit den eingespeisten Kochrezepten gezeigt. Abbildung 5.5 stellt die Oberfläche der Anwendung Cooking Agent dar. Diese präsentiert die vom System erkannten vorhandenen Produkte und die hierzu entsprechenden Kochrezepte.

Die Anwendung Cooking Agent befindet sich noch in der Entwicklungsphase. Jedoch sind bereits mehrere Features der Anwendung realisiert, sodass die Arbeitsweise der in Kapitel 4 beschriebenen Architektur erprobt werden kann. Diese Anwendung ist architektonisch so ausgelegt, dass sie in ein System für intelligente Häuser integriert werden kann. Diese Anwendung kann um weitere Funktionen erweitert werden, sodass das gewonnene System einem System für intelligente Wohnumgebungen entspricht (vgl. Kapitel 4).

Abbildung 5.3.: Remote GUI der Anwendung Smart:Shelf

Abbildung 5.4.: Anwendung RMS mit den vorhandenen Kochrezepten

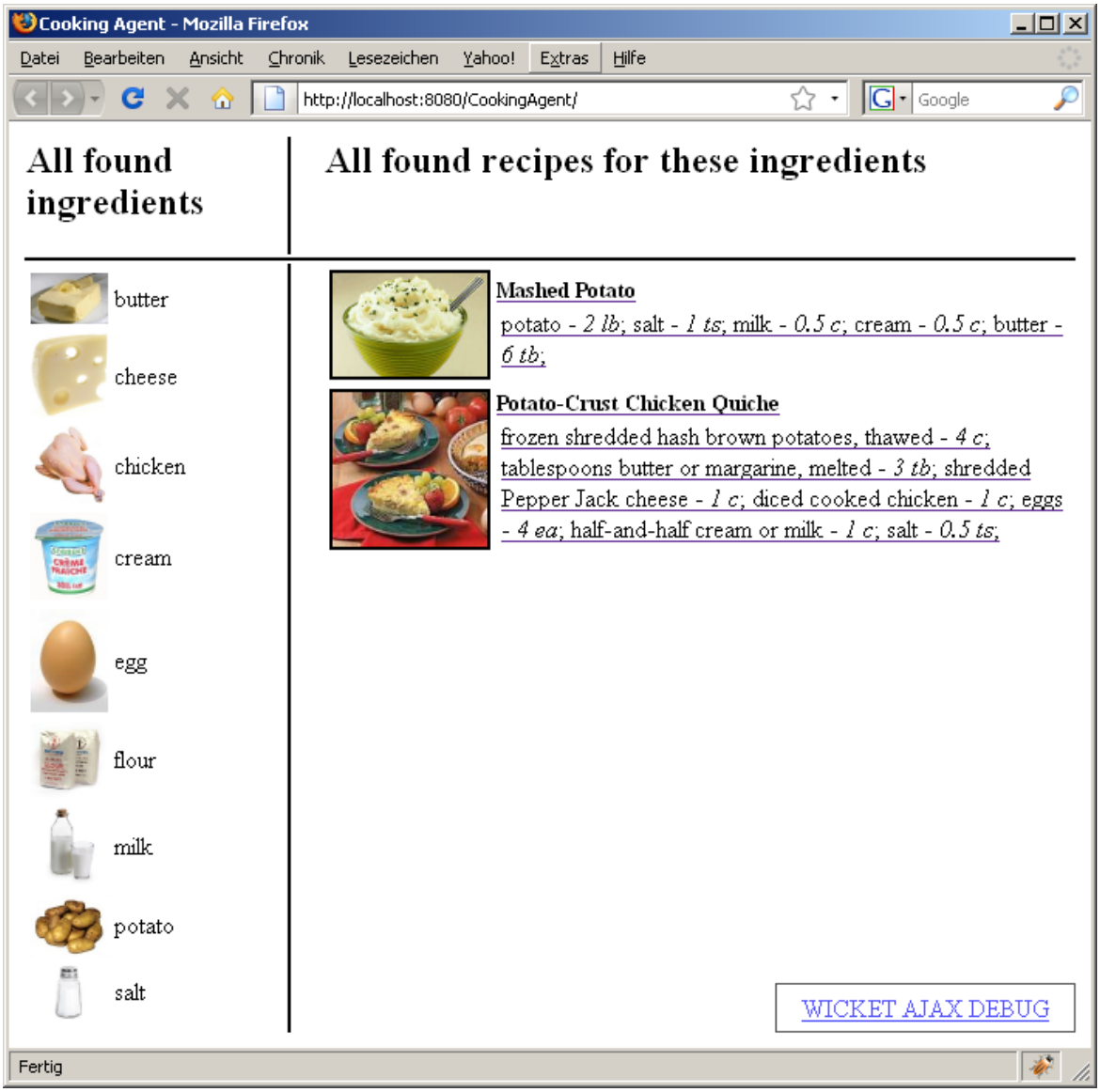


Abbildung 5.5.: Cooking Agent GUI mit den registrierten vorhandenen Produkten und den dazu passenden Kochrezepten

5.2. Evaluation

Mit der Entwicklung von Cooking Agent wurde versucht, die in Kapitel 4 erarbeitete Architektur an einem realen Beispiel zu erproben. Obwohl diese Anwendung noch nicht vollständig realisiert wurde, weist sie dennoch die wesentlichen Bestandteile einer solchen Architektur auf. Anhand dieser können die ersten Schlüsse über den Einsatz einer derartigen Architektur für Systeme für intelligente Wohnumgebungen gezogen werden.

In dieser Anwendung wurden mehrere Agenten implementiert, die wie folgt klassifiziert werden können (vgl. Kapitel 4.1.1):

- **Sensor-Agenten**: SmartShelf Shelf (als „Bestandteil“ von AvailableProductDeterminer), da dieser einen RFID-Reader besitzt.
- **Kontext-Agenten**: SmartShelf DB (als „Bestandteil“ von AvailableProductDeterminer), wo die RFIDs zu Artikel gemappt werden.
- **Prozess-Agenten**: CookingAgent, der den Ablauf der Cooking Agent Anwendung übernimmt.
- **Aktion-Agenten**: GUI und PrintAgent, die als Ausgabe-Medien zu verstehen sind.
- **weitere Dritt-Agenten**: RecipeManager, Calendar und UserProfile, die Basiswissen darstellen.

Agenten können u.U. Eigenschaften mehrerer Agenten-Arten aufweisen. Die in Kapitel 4.1.1 vorgeschlagene Klassifizierung der Agenten dient zu einer besseren Trennung der Aufgaben in einem System für intelligente Häuser, die an die wesentlichen Merkmale solcher Systeme angelehnt wurde. Eine klare Aufgaben-Trennung kann die Erweiterung bzw. Ersetzung von Teilen des System (vgl. Agenten und ihre Dienste) erleichtern. Diese Vorgehensweise soll als Leitfaden für die Entwicklung solcher Systeme verstanden werden. Eine kleine Abwandlung dieser Strategie kann in bestimmten Fällen gerechtfertigt sein. So kann z.B. ein Sensor-Agent (vgl. SmartShelf Shelf) von anderen Agenten gesteuert werden, sodass er in diesem Moment die Rolle eines Aktion-Agenten übernimmt, der die Anweisung anderer Agenten ausführt.

Die Kommunikation zwischen den Agenten verläuft ausschließlich mittels Event-Austausch mit Hilfe eines Even-Managers (vgl. Kapitel 4). Dieser stellt die ihm übergebenen Events an die Interessenten zu. In Cooking Agent übernimmt iROS Event Heap diese Rolle. Die Kommunikation zwischen dem Agenten und dem Event Heap wird von einem Event Heap Adapter unterstützt, der das Erzeugen, das Verschicken, das Registrieren und das Empfangen von Events übernimmt. In ihm werden Events definiert, die der Agent „verstehen“ soll. Auf diese Weise wird die Schnittstelle des Agenten definiert, die zugleich als dessen Dienst-Beschreibung fundiert.

Der Event-Austausch verläuft indirekt über den iROS Event Heap. Dieser nimmt die Events von einem Agenten entgegen und stellt sie den Agenten zu, die sich für solche Events registriert haben (vgl. Kapitel 4.1.2). Der Sender und der Empfänger dieser Events sind den beteiligten Agenten unbekannt. Somit gibt es u.U. mehrere Empfänger. Diese Tatsache ermöglicht einen anwendungsübergreifenden Einsatz der Agenten, was ihre Wiederverwendbarkeit erhöht.

Eine Anwendung ergibt sich aus einer Agenten-Kooperation, die zur Laufzeit zustande kommt. Eine Änderung des Systems durch beispielsweise einem Austausch, einer Hinzufügung oder Löschung von Agenten kann die bestehenden Anwendungen beeinflussen oder sogar neue erzeugen. Unerwünschte Handlungen von Agenten in bestimmten Anwendungen können unterbunden werden. Hierfür wurden Strategien erarbeitet, die dieses verhindern sollen. Dies kann durch die Einführung spezieller Event-Felder realisiert werden (siehe Kapitel 4.1.6).

Bei der Konzeptionierung und der anschließenden Realisierung wurden folgende Vor- und Nachteile dieser Architektur festgestellt:

Vorteile:

- *Keine Anfrage der Dienste* (*service detection*): Die Agenten teilen nur mit, was sie benötigen (vgl. `RequestEvent`) und was sie erwarten (vgl. `ResponseEvent`). Auf diese Weise werden ihre Anfragen von anderen Agenten bearbeitet und ihre Ergebnisse dem Anfragesteller mitgeteilt.
- *Keine komplexe Prozess-Engine*: Die Anwendung ergibt sich indirekt durch die Agenten-Interaktion. Somit wird die Intelligenz einer Anwendung auf die in dieser Anwendung beteiligten Agenten verteilt.
- *Keine komplexe Konfiguration*: Da die Kommunikation zwischen den Agenten ausschließlich nur über einen Event-Manager verläuft, muss lediglich der Zugang zum Event-Manager den Agenten bekannt sein.
- *Einfache Änderung / Erweiterung*: Durch Änderung, Löschung oder Hinzufügung der Agenten können Anwendungen (oder sogar das Gesamtsystem) geändert bzw. erweitert werden.
- *Wiederverwendbarkeit*: Die Agenten kennen die Anfragesteller nicht, somit können die Anfragen für die Agenten verschiedener Anwendungen von den selben Agenten bearbeitet werden.
- *Einfache Integration bestehender Anwendungen*: Bestehende Anwendungen können relativ einfach in das System integriert werden. Hierfür muss lediglich ein entsprechender Adapter implementiert werden, der die Schnittstelle für die Agenten-Interaktion mittels Event-Kommunikationsart anbietet (vgl. Integration von RMS in Kapitel 5.1.6).

- *Robustheit des Systems*: Ausfall eines der Agenten führt nicht zum Ausfall des Gesamtsystems. Für die ausfallsicheren Anwendungen können deren Agenten repliziert werden, sodass ihre Robustheit erhöht wird.

Nachteile:

- *Single Point of Failure*: Der Ausfall des Event-Managers führt zum Ausfall des Gesamtsystems, da die Agenten-Kommunikation unterbrochen wird. Eine Replikation des Event-Managers kann die Ausfallsicherheit des Systems erhöhen.
- *Empfindlichkeit des Systems*: Da die Anwendung sich indirekt durch die Agenten-Interaktion ergibt, kann die Änderung, Hinzufügung oder Löschung von Agenten unerwünschte Handlungen hevorrufen bzw. den erwarteten Anwendungsablauf stören oder unterbrechen. Strategien zur Vermeidung dieser unerwünschten Effekte wurden bereits in Kapitel 4.1.6 vorgeschlagen (z.B. Einführung neuer Event-Felder).
- *Anfälligkeit für allgemeine Probleme Verteilter Systeme*[15]: Probleme, die im Zusammenhang mit verteilten Systemen bekannt sind (z.B. Deadlock oder Lifelock), können in solchen Systemen auftreten. Um dies zu vermeiden, darf der Entwickler den Überblick über die beteiligten Agenten nicht verlieren und deren kooperative Handlungen kontrollieren.
- *Erschwerte Fehlersuche*: Wie in anderen verteilten Systemen ist hier die Fehlersuche erschwert. So können beispielsweise Fehler bei der Interagenten-Kommunikation, -Kooperation oder in den einzelnen Agenten auftreten. Eine Abhilfe bei der Suche nach den möglichen Fehlerursachen können sog. Protokoll-Agenten verschaffen (vgl. Kapitel 4.1.7). Sie registrieren jede nach außen sichbare Tätigkeit der Agenten (vgl. Events) und protokollieren diese. Hiermit wird ein sequenzieller Arbeitsablauf der Anwendungen durch eine Event-Liste erstellt. Dieser Ablauf wird von den Entwicklern analysiert und kann so bei der Fehlersuche behilflich sein. Bei diesem Verfahren wird allerdings davon ausgegangen, dass der Event-Manager fehlerfrei funktioniert.
- *Keine standardisierte Schnittstellen-Beschreibung*: Die Agenten-Schnittstelle wird in diesem System durch Events repräsentiert. Diese werden zugleich als Beschreibung für die von den Agennten angebotenen Dienste verstanden. Die Event-Definition ist den Entwicklern des jeweiligen Systems überlassen. Somit ist der Einsatz von Agenten anderer Systeme durch die Schnittstellen-Unterschiede nicht ohne weiteres möglich.
- *Störung durch böswillige Agenten*: Durch Hinzufügung feindlicher Agenten kann der System-Ablauf gestört werden. Zu Verhinderung dessen kann ein Sicherheitskonzept erarbeitet werden, dessen Regel nach die einzelnen Agenten sich authentifizieren

[15] Bekannte Probleme der nebenläufigen oder verteilten Programmierung (vgl. z.B. Programmiersprache *Erlang*) sind teilweise auf diese Architektur übertragbar. Für ihre Lösung können die Ansätze dieser Programmierungen verwendet werden.

müssen. Diese Authentifikation kann z.B. auf der Basis vom Austausch elektronischer Zertifikate geschehen.

Mit der Implementierung von Cooking Agent wurde gezeigt, dass die konzepierte Architektur für Systeme intelligenter Wohnumgebungen grundsätzlich geeignet ist. Die festgestellten Nachteile dieser Architektur müssen bei der Entwicklung eines solchen Systems berücksichtigt werden, um diese zu vermeiden.

Die Cooking Agent Anwendung wurde von zahlreichen Besuchern des iFlat-Labors als positiv empfunden. Dies bestätigt den möglichen Einsatz solcher Systeme in intelligenten Häusern.

6. Fazit und Ausblick

Mit den heutigen weiter wachsenden technologischen Errungenschaften wächst auch der Wunsch nach einer technischen Unterstützung im Wohnbereich. In dieser Arbeit wurde eine Architektur erarbeitet, die für den Einsatz in Systemen für intelligente Wohnumgebungen geeignet ist. Hierbei wurden ihre wesentlichen Komponenten und deren Zusammenarbeit erörtert.

Der Architektur liegen zwei Modelle zugrunde: Das Multiagenten- und das Blackboard-Modell. Einige Änderungen wurden gegenüber diesen klassischen Modelle vorgenommen, sodass eine Architektur entstand, die diese Modelle in sich vereinte. Des Weiteren wurde versucht die wesentlichen Merkmale einer SOA (z.B. Verteilheit, Lose Kopplung und Prozessorientiertheit, vgl. Kapitel 2.3.1) in die Architektur einfließen zu lassen.

Für den Einsatz der Architektur in intelligenten Wohnumgebungen wurden entsprechende Komponenten vorgesehen und beschrieben. Mittels dieser können die Hauptmerkmale eines solchen Systems umgesetzt werden. Diese Merkmale sind:

- Erfassung der Umgebung,
- Kontext-Ermittlung,
- Erstellung adäquater Aktionen und
- Ausführung dieser Aktionen.

Mittels dieser Komponenten können Anwendungen im Bereich der intelligenten Wohnumgebung realisiert werden (vgl. Kapitel 4.1.6).

Zur Erprobung der theoretischen Überlegungen in der Praxis wurde eine Anwendung mit dieser Architektur realisiert. Dabei wurden Vor- und Nachteile der Architektur festgestellt bzw. bestätigt (siehe Kapitel 5.2). Für die Vermeidung der Nachteile wurden an dieser Stelle entsprechende Strategien vorgeschlagen. Zahlreiche Besucher haben diese Anwendung als positiv empfunden, was den möglichen Einsatz von Anwendungen dieser Art in Wohnungen oder Häusern bestätigt.

Die erarbeitete Architektur eignet sich nicht nur für Systeme intelligenter Wohnumgebungen. Mit Hilfe dieser Architektur können z.B. Systeme für die Abwicklung von Geschäftsprozessen realisiert werden. Zur Zeit ist die am häufigsten gewählte Realisierungsarchitektur

die *Web-Services-Architektur*[1]. Nach dieser Architektur werden mittels einer geeigneten Beschreibungssprache (wie z.B. *BPEL*[2]) Geschäftsprozesse definiert. Die somit gewonnene Definition spielt die Rolle eines Steuerungsplanes, anhand dessen die Webservices der Reihe nach aufgerufen werden und somit einen Geschäftsprozess ausführen. Solche Geschäftsprozesse können durch bestimmte Anwendungen der in dieser Arbeit modellierten Architektur abgebildet werden. Durch die Schnittstellen-Festlegung der Agenten wird deren „Verknüpfung“ definiert. Auf diese Weise kann der Ablauf eines Geschäftsprozesses realisiert werden. Bei der Verwendung von Agenten mit ähnlichen Schnittstellen in verschiedenen Prozessen können die einzelnen Geschäftsprozesse, beispielsweise durch die Einführung zusätzlicher Event-Felder, unterschieden werden (vgl. Kapitel 4.1.6).

Diese Architektur eignet sich außerdem für Systeme, bei denen die System-Reaktion auf bestimmte Ereignisse gewünscht ist. Inwieweit sich der Einsatz dieser Architektur als geeignet erweist, liegt diese Entscheidung der jeweiligen Entwickler solcher Systeme.

Die in dieser Arbeit konzipierte Architektur wurde mittels Realisierung einer Anwendung getestet. Die Machbarkeit eines solchen Systems mit mehreren Anwendungen werden die zukünftigen Ergebnisse zeigen. Ein Ziel in diesem Zusammenhang wird sein ein vollständiges System für intelligente Wohnumgebungen zu realisieren. Hierfür soll die bereits vorhandene Anwendung Cooking Agent als Basis für zusätzliche Erweiterungen dienen. Mit Hilfe neuer Agenten soll das System um weitere Anwendungen ergänzt werden.

Weitere geplante Ziele im Zusammenhang mit der entworfenen Architektur sind:

- *Standardisierung der Agenten-Schnittstellen*
  Es soll eine Struktur der Agenten-Schnittstelle (vgl. z.B. FIPA ACL) definiert werden, die allgemein für solche Systeme geeignet wäre. Auf diese Weise können Agenten anderer Systeme ohne großen Aufwand in das bestehende System integriert werden. Ein weiterer Punkt in diesem Zusammenhang ist die semantische Interpretation solcher Schnittstellen. Sie sind meistens nur innerhalb einer Domäne eindeutig interpretierbar. Um die Integration von Agenten anderer Domänen zu vereinfachen, soll die Schnittstellen-Definition der Agenten domäneübergreifend gestaltet werden.

- *Sprach- und Plattformunabhängigkeit des Systems*
  Das System soll nicht von den technischen Einschränkungen der jeweiligen Umgebung beeinflusst sein. Aus diesem Grunde ist es erwünscht, dass das System plattformunabhängig sein soll. Des Weiteren soll es möglich sein, Agenten, die in verschiedenen Programmiersprachen realisiert wurden, in das System zu integrieren. Es wurden bereits Vorschläge zur Umsetzung der genannten Anforderungen gemacht, jedoch sollen diese genauer untersucht und anschließend realisiert werden.

[1] vgl. Kapitel 2.3.1

[2] *Business Process Execution Language* ist eine XML-basierte Beschreibungssprache für Geschäftsprozesse, deren einzelne Aktivitäten durch Webservices implementiert werden.

- *Erstellung von Anwendungen durch Konfiguration bereits vorhandener Agenten*
  Es ist wünschenswert ein System zu haben, das von den jeweiligen Anwender auf ihre Bedürfnisse angepasst werden kann. Dies soll anhand eines graphisch unterstützten Werkzeugs geschehen.

Diese und weitere Ziele stellen das Aufgabengebiet der Entwicklungen der Labore iFlat und Living Place Hamburg dar.

Im Allgemeinen lässt sich sagen, dass die in dieser Arbeit entworfene Architektur sich zur Erstellung von Systemen für intelligente Wohnumgebungen als geeignet erwiesen hat. Sie weist zwar einige Nachteile auf, die von den Entwicklern bei der Realisierung solcher Systeme berücksichtigt werden müssen, dennoch hat sie ein großes Potenzial für Systeme dieser Art. Die vorgestellten Erweiterungen sollen einen breiteren Einsatz dieser Architektur schaffen. In Wieweit diese Architektur tatsächlich für Systeme intelligenter Wohnumgebungen eingesetzt wird, bleibt noch abzuwarten.

Abbildungsverzeichnis

Listings

Literaturverzeichnis

[1] ALAMO, Reyes ; WONG, J.M. ; WONG, J.: Service-oriented middleware for Smart Home applications. In: *Wireless Hive Networks Conference, 2008. WHNC 2008. IEEE* (2008), Aug., S. 1–4

[2] AUSTALLER, G.: *Web Services als Bausteine für kontextabhängige Anwendungen*. 2004

[3] BAALL: *BAALL – Bremen Ambient Assisted Living Lab*. – URL http://www.baall.net/de. – Zugriffsdatum: 2009-06-11

[4] BISCHOFF, U. ; SUNDRAMOORTHY, V. ; KORTUEM, G.: Programming the smart home. In: *Intelligent Environments, 2007. IE 07. 3rd IET International Conference on* (2007), Sept., S. 544–551. – ISSN 0537-9989

[5] BITCOM: *Notebooks treiben PC-Absatz auf Rekordhoch*. 09 2008. – URL http://www.bitkom.org/de/presse/30739_54217.aspx. – Zugriffsdatum: 2008-09-28

[6] BORCHERS, J. ; RINGEL, M. ; TYLER, J. ; FOX, A.: Stanford interactive workspaces: a framework for physical and graphical user interface prototyping. In: *Wireless Communications, IEEE* 9 (2002), Dec., Nr. 6, S. 64–69. – ISSN 1536-1284

[7] BRENNER, Walter ; ZARNEKOW, Rüdiger ; WITTIG, Hartmut: *Intelligente Softwareagenten : Grundlagen und Anwendungen*. Springer Berlin, 1998. – ISBN 3-540-63431-2

[8] BROOKS, Rodney A.: *Intelligence without representation*. 1987. – URL http://people.csail.mit.edu/brooks/papers/representation.pdf. – Zugriffsdatum: 2009-04-20

[9] BUNDESMINISTERIUMS FÜR BILDUNG UND FORSCHUNG: *Ambient Assisted Living*. 2009. – URL http://www.aal-deutschland.de/. – Zugriffsdatum: 2009-010-01

[10] CARVER, Norman ; LESSER, Victor: *The Evolution of Blackboard Control Architectures*. 1992. – URL http://ksuseer1.ist.psu.edu/viewdoc/download?doi=10.1.1.43.6808&rep=rep1&type=pdf. – Zugriffsdatum: 2009-04-24

[11] COOK, Diane J. ; YOUNGBLOOD, Michael ; EDWIN O. HEIERMAN, III ; GOPALRATNAM, Karthik ; RAO, Sira ; LITVIN, Andrey ; KHAWAJA, Farhan: MavHome: An Agent-Based Smart Home. In: *Pervasive Computing and Communications, IEEE International Conference on* 0 (2003), S. 521. ISBN 0-7695-1893-1

[12] COULOURIS, George ; DOLLIMORE, Jean ; KINDBERG, Tim: *Verteilte Systeme, Konzepte und Design*. Pearson Studium, 2002. – ISBN 3-8273-7002-1

[13] DARMSTADT, TU: *Ambient Intelligence*. 2007. – URL http://www.it-gipfel.tu-darmstadt.de/media/illustrationen/referat_kommunikation/themaforschung/200701/07-01-gesamt.pdf. – [Online; Stand 5. Februar 2009]

[14] DEY, Anind K.: *Understanding and Using Context*. 2001. – URL http://www.cc.gatech.edu/fce/ctk/pubs/PeTe5-1.pdf. – Zugriffsdatum: 2009-02-06

[15] DEY, Anind K. ; ABOWD, Gregory D.: *Towards a Better Understanding of Context and Context-Awareness*. 2001. – URL http://www.cc.gatech.edu/fce/ctk/pubs/PeTe5-1.pdf. – Zugriffsdatum: 2009-02-06

[16] DFG: *Homepage der Deutschen Forschungsgemeinschaft*. – URL http://www.dfg.de/. – Zugriffsdatum: 2009-06-13

[17] DFKI: *BAALL - Bremen Ambient Assisted Living Laboratory*. – URL http://www.dfki.de/web/living-labs-de/baall-bremen-ambient-assisted-living-laboratory. – Zugriffsdatum: 2009-06-09

[18] DIMITROV, Todor ; PAULI, Josef ; NAROSKA, Edwin: A probabilistic reasoning framework for smart homes. In: *MPAC '07: Proceedings of the 5th international workshop on Middleware for pervasive and ad-hoc computing*. New York, NY, USA : ACM, 2007, S. 1–6. – ISBN 978-1-59593-930-2

[19] DOSTAL, W. ; JECKLE, M. ; MELZER, I. ; ZENGLER, B.: *Service-orientierte Architekuren mit Web Services: Konzepte-Standards-Praxis*. ELSEVIER Spektrum Akademischer Verlag, 2005. – ISBN 3-8274-1457-1

[20] DREYER, Markus: *Ein nutzeradaptierendes agentenbasiertes TV System als Teil eines intelligenten Hauses*. 2009. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/dreyer.pdf. – Zugriffsdatum: 2009-06-20

[21] ERL, Thomas: *Service-Orientied Architecture: Concepts, Technology, and Design*. Prentice Hall, 2005. – ISBN 0-13-185858-0

[22] EYMANN, Torsten: *Digitale Geschäftsagenten : Softwareagenten im Einsatz*. Springer Berlin, 2003. – ISBN 3-540-44019-4

[23] FATHI-TORBAGHAN, M. ; HÖFFMANN, A.: *Fuzzy Logik und Blackboard-Modelle in der technischen Anwendung*. R. Oldenbourg Verlag, 1994. – ISBN 3-486-22650-9

[24] FERBER, Jacques: *Multi-Agent Systems : An Introduction to Distributed Artificial Intelligence*. Addison Wesley, 1999. – ISBN 0-201-36048-9

[25] FIPA: *Foundation for Intelligent Physical Agents*. – URL http://fipa.org. – Zugriffsdatum: 2009-04-24

[26] FRANKLIN, Stan ; GRAESSER, Art: *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*. 1996. – URL http://www.msci.memphis.edu/~franklin/AgentProg.html. – Zugriffsdatum: 2009-04-01

[27] FRAUNHOFER: *Fraunhofer inHaus-Zentrum: Intelligente Raum- und Gebäudesysteme*. – URL http://www.inhaus-zentrum.de/site_de/. – Zugriffsdatum: 2009-06-09

[28] GLINZ, Martin: *Requirements Engineering I*. 2006

[29] GÖRZ, G. ; ROLLINGER, C.-R. ; SCHNEEBERGER, J.: *Handbuch der Künstlichen Intelligez*. Bd. 4. Auflage. Oldenbourg, 2003. – ISBN 3-486-27212-8

[30] HAIBER, André C.: *Context-Aware Services und Ubiquitous Computing*. 2006. – URL http://www.aifb.uni-karlsruhe.de/Lehre/Sommer2006/ws/AusarbeitungUbiCom

[31] HIBERNATE: *Relational Persistence for Java*. – URL https://www.hibernate.org/

[32] HONG, Chung-Seong ; LEE, Kang-Woo ; SUH, Young-Ho ; KIM, Hyoung-Sun ; KIM, Hyun ; LEE, Hyun-Chan: Developing Context-Aware System using the Conceptual Context Model. In: *Computer and Information Technology, International Conference on* 0 (2006), S. 238. ISBN 0-7695-2687-X

[33] IIW: Homepage der Initiative Intelligentes Wohnen. (2009). – URL http://www.intelligenteswohnen.com/iw_de/. – Zugriffsdatum: 2009-10-01

[34] JOHANSON, B. ; FOX, A.: 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

[35] JOHANSON, Bradley E.: *Appendix – The Event Heap Wire Protocol*. Dec. 2003. – URL http://graphics.stanford.edu/~bjohanso/dissertation/wire-protocol-appendix.pdf

[36] JOHANSON, Bradley E.: *Dissertation: Application Coordination Infrastructure for Ubiquitous Computing Rooms*. Dec. 2003. – URL http://graphics.stanford.edu/~bjohanso/dissertation/

[37] KAHLBRANDT, Bernd: *Software-Engineering. Objektorientierte Softwareentwicklung mit der Unified Modeling Language*. 1998

[38] KLOSSEK, Martin ; WLEKLINSKI, Fabian: *Vor- und Nachteile von Webanwendungen im Vergleich zu Lösungen auf Office Software Basis*. – URL www.eworks.de/research/2004/05/Web_vs.../Web_vs_Office.pdf. – Zugriffsdatum: 2009-08-30

[39] KRAFZIG, Dirk ; BANKE, Karl ; SLAMA, Dirk: *ENTERPRISE SOA: Service-Oriented Architecture Best Practices*. Prentice Hall, 2004. – ISBN 0-13-146575-9

[40] KRIEG-BRÜCKNER, Bernd ; GERSDORF, Bernd ; DÖHLE, Mathias ; SCHILL, Kerstin: *Technik für Senioren in spe im Bremen Ambient Assisted Living Lab*. 2009. – URL http://www.vde.com/de/technik/aal/steckbriefe/documents/aal2009-baall.pdf. – Zugriffsdatum: 2009-06-11

[41] LABS, Open L.: *European Network of Living Labs - a first step towards a new Innovation Systems*. – URL http://www.openlivinglabs.eu/. – Zugriffsdatum: 2009-06-11

[42] MACGREGOR, R.: *Inside the LOOM Classifier*. 1994

[43] MATTERN, Friedemann: *Ubiquitous Computing: Schlaue Alltagsgegenstände*. 2004. – URL http://www.vs.inf.ethz.ch/publ/papers/mattern2004_sev.pdf

[44] MCGUINNESS, Deborah L. ; VAN HARMELEN, Frank: *OWL Web Ontology Language*. 2004. – URL http://www.w3.org/TR/owl-features/. – Zugriffsdatum: 2009-02-24

[45] MOON, A. ; KIM, H. ; KIM, H. ; LEE, S.: *Context-Aware Active Services in Ubiquitous Computing Environments*. 2007. – URL http://etrij.etri.re.kr/Cyber/servlet/GetFile?fileid=SPF-1175650242476

[46] MOZER, Michael C.: The Neural Network House: An Environment that Adapts to its Inhabitants. In: *Proceedings of the American Association for Artificial Intelligence Spring Symposium on Intelligent Environments* 0 (1998), S. 110–114

[47] NETO, Renato B. ; TEIXEIRA, César A. C. ; DA GRAÇA CAMPOS PIMENTEL, Maria: A Semantic Web-Based Infrastructure Supporting Context-Aware Applications. In: *EUC* Bd. 3824, Springer, 2005, S. 900–909. – ISBN 3-540-30807-5

[48] NORBISRATH, Ulrich: *Konfigurierung von eHome-Systemen*, Rheinisch-Westfälischen Technischen Hochschule Aachen, Dissertation, 2007. – URL http://darwin.bth.rwth-aachen.de/opus3/volltexte/2007/1959/pdf/Norbisrath_Ulrich.pdf

[49] ORTIZ, V. S.: *A General Model for Context Aware Computing.* 2003

[50] PHILIPS: *Homepage des Philips HomeLab*. – URL http://www.research.philips.com/technologies/projects/homelab/index.html. – Zugriffsdatum: 2009-06-13

[51] PREKOP, Paul ; BURNETT, Mark: *Activities, Context and Ubiquitous Computing*. 2003. – URL http://arxiv.org/ftp/cs/papers/0209/0209021.pdf. – Zugriffsdatum: 2009-02-06

[52] RETKOWITZ, Daniel ; PIENKOS, Monika: Ontology-based configuration of adaptive smart homes. In: *ARM '08: Proceedings of the 7th workshop on Reflective and adaptive middleware*. New York, NY, USA : ACM, 2008, S. 11–16. – ISBN 978-1-60558-367-9

[53] ROSSBERGER, Philipp ; VON LUCK, Kai: *Seamless interaction in interactive rooms - some preliminary remarks*. 2008. – URL http://www.worldusabilityday.de/2008/hamburg/vortraege/WUD2008HH_Luck.pdf. – [Online; Stand 5. Februar 2009]

[54] SCHREIBER, A.Th. ; WELINGA, B. ; AKKERMANS, H. ; VAN DE VELDE, W. ; ANJEWIERDEN, A.: *CML: The Common KADS Conceptual Model Language*. 1991

[55] SFB/TR 8 SPATIAL COGNITION: *Home page of SFB/TR 8 Spatial Cognition*. – URL http://www.sfbtr8.spatial-cognition.de/. – Zugriffsdatum: 2009-06-13

[56] SHARE-IT: *Home page of SHARE-it*. – URL http://www.ist-shareit.eu/shareit. – Zugriffsdatum: 2009-06-13

[57] STANFORD UNIVERSITY: iRoom Interactivity Lab. (2002)

[58] STEGELMEIER, Sven ; WENDT, Piotr ; LUCK, Kai von: *iFlat - Eine dienstorientierte Architektur für intelligente Räume*. 2009. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/papers/aal2009.pdf. – Zugriffsdatum: 2009-08-03

[59] TANENBAUM, Andrew ; VAN STEEN, Marten: *Verteilte Systeme, Grundlagen und Paradigmen*. Pearson Studium, 2003. – ISBN 3-8273-7057-4

[60] TANG, Y. ; WU, Q.: *A Semantic and Adaptive Context Model for Ubiquitous Computing*. 2005

[61] TAYLOR, Alex S. ; HARPER, Richard ; SWAN, Laurel ; IZADI, Shahram ; SELLEN, Abigail ; PERRY, Mark: Homes that make us smart. In: *Personal Ubiquitous Comput.* 11 (2007), Nr. 5, S. 383–393. – ISSN 1617-4909

[62] TEACHING.ORG e: *Ontologie*. – URL http://www.e-teaching.org/glossar/ontologie. – Zugriffsdatum: 2009-02-06

[63] UBISENSE: *Ubisense Platform*. – URL http://www.ubisense.de. – Zugriffsdatum: 2009-06-20

[64] URICH, Jaroslaw: *Vergleich von Alternativen zur Speicherung von XML-Dokumenten an einem Anwendungsbeispiel*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, Sep. 2005

[65] URICH, Jaroslaw: *Context-Awareness: aktuelle Projekte*. 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-aw/urich/bericht.pdf. – Zugriffsdatum: 2008-02-13

[66] URICH, Jaroslaw: *smart:shelf, Projektbericht*. 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master07-08-proj/urich/report.pdf. – Zugriffsdatum: 2009-05-15

[67] WANDKE, Hartmut: *Die Zukunft der Usability-Forschung im Lichte allgegenwärtiger Computertechnologie*. – URL http://www.artop.de/5000_Archiv/5000_PDF_und_Material/Usability/Zukunft_der_Usability-Forschung_im_Lichte_allgegenwaertiger_Computertechnologie.pdf. – [Online; Stand 5. Februar 2009]

[68] WANG, X.H. ; ZHANG, D.Q. ; GU, T. ; PUNG, H.K.: Ontology based context modeling and reasoning using OWL. In: *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on* (2004), March, S. 18–22

[69] WEBERHAUS: *T-Com Haus: Einzigartiges Gemeinschaftsprojekt*. – URL http://weberhaus.de/t-com.html. – Zugriffsdatum: 2009-06-09

[70] WEISER, Mark: The computer for the 21st century. (1995), S. 933–940. ISBN 1-55860-246-1

[71] WIKIPEDIA: *Ambient Intelligence — Wikipedia, Die freie Enzyklopädie*. 2008. – URL http://de.wikipedia.org/w/index.php?title=Ambient_Intelligence&oldid=50944947. – [Online; Stand 5. Februar 2009]

[72] WIKIPEDIA: *Intelligentes Wohnen — Wikipedia, Die freie Enzyklopädie*. 2008. – URL http://de.wikipedia.org/w/index.php?title=Intelligentes_Wohnen&oldid=53625936. – [Online; Stand 11. Januar 2009]

[73] WIKIPEDIA: *Ubiquitous Computing — Wikipedia, Die freie Enzyklopädie*. 2008. – URL http://de.wikipedia.org/w/index.php?title=Ubiquitous_Computing&oldid=53639944. – [Online; Stand 24. Januar 2009]

[74] WIKIPEDIA: *Extensible Markup Language — Wikipedia, Die freie Enzyklopädie*. 2009. – URL http://de.wikipedia.org/w/index.php?title=Extensible_Markup_Language&oldid=56843293. – [Online; Stand 24. Februar 2009]

[75] WIKIPEDIA: *Linda (coordination language) — Wikipedia, The Free Encyclopedia*. 2009. – URL http://en.wikipedia.org/w/index.php?title=Linda_(coordination_language)&oldid=273115100. – [Online; accessed 25-February-2009]

[76] WIKIPEDIA: *Linda (coordination language) — Wikipedia, The Free Encyclopedia*. 2009. – URL hci.rwth-aachen.de/istuff/tutorial_eventheap.pdf. – [Online; accessed 25-February-2009]

[77] WIKIPEDIA: *Pervasive Computing — Wikipedia, Die freie Enzyklopädie*. 2009. – URL http://de.wikipedia.org/w/index.php?title=Pervasive_Computing&oldid=54849801. – [Online; Stand 24. Januar 2009]

[78] WIKIPEDIA: *Smartphone — Wikipedia, Die freie Enzyklopädie*. 2009. – URL http://de.wikipedia.org/w/index.php?title=Smartphone&oldid=55109556. – [Online; Stand 9. Januar 2009]

[79] WIKIPEDIA: *Sprechakttheorie — Wikipedia, Die freie Enzyklopädie*. 2009. – URL http://de.wikipedia.org/w/index.php?title=Sprechakttheorie&oldid=63085301

[80] WINKLER, Ronny: *Entwicklung eines ontologiebasierten Kontextmodells für kollaborative Web-Anwendungen*. 2007. – URL http://www-mmt.inf.tu-dresden.de/global/UploadedContent/Themen/207.pdf. – Zugriffsdatum: 2009-02-24

[81] WOOLDRIDGE, Michael: *An Introduction to MultiAgent Systems*. Second Edition. WILEY, 2009. – ISBN 978-0-470-51946-2

A. Inhalt der CD-ROM

Dieser Arbeit ist eine CD-ROM beigelegt. Sie enthält folgende Verzeichnisse:

Masterarbeit beinhaltet diese Arbeit in digitaler Form.

Source-Code enthält den Quelltext der entwickelten Anwendung *Cooking Agent* mit den für die Implementierung verwendeten Bibliotheken. Der Quelltext ist in mehrere Projekte unterteilt, die in die Entwicklungsumgebung *Eclipse* (ab Version 3.4) importiert werden können. Eine Distribution hat diese Anwendung nicht, sodass die Anwender gebeten werden Cooking Agent aus der Entwicklungsumgebung zu starten.

Versicherung ber Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 6. November 2009	
Ort, Datum	Unterschrift