Software-Infrastruktur und Entwicklungsumgebung für selbstorganisierende multimediale Ensembles in Ambient-Intelligence-Umgebungen

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

DISSERTATION

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)

von

Dipl.-Phys. Michael Hellenschmidt
geboren in Heilbronn

Referenten der Arbeit:
Prof. Dr. José L. Encarnação, Technische Universität Darmstadt
Prof. Dr. Dieter Rombach, Technische Universität Kaiserslautern

Tag der Einreichung: 16.05.2007
Tag der mündlichen Prüfung: 28.06.2007

Darmstädter Dissertation, 2007
D 17

Vorwort

Die vorliegende Dissertation entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Fraunhofer-Institut für Graphische Datenverarbeitung (Fraunhofer IGD) in Darmstadt. Dort war ich in den Abteilungen „e-Learning & Knowledge Management“ und „Interaktive Multimedia Appliances“ in mehreren Projekten zum Themenfeld Ambient Intelligence beschäftigt.

Mein besonderer Dank gilt zahlreichen Personen, die in unterschiedlicher Art und Weise zum Gelingen dieser Arbeit beigetragen haben.

Ganz an erster Stelle möchte ich hier Prof. Dr.-Ing. Dr. h.c. Dr. E.h. José L. Encarnação danken, der mit hohem persönlichen Einsatz das Themenfeld Ambient Intelligence in Deutschland und Europa vorangetrieben hat und mir stetig die Bedeutung meiner wissenschaftlichen Arbeit vermittelt hat. Ihm sind das Gelingen dieser Arbeit und alle positiven Erfahrungen, die ich damit gemacht habe, in erster Linie zu verdanken.

Ich möchte mich bei Herrn Prof. Dr. Dieter Rombach bedanken, der sofort die Bereitschaft erklärt hat als Referent meiner Arbeit mitzuwirken. Ich bin mir der Bedeutung dieser Geste sehr bewusst.

Ganz besonders möchte ich Prof. Dr. Thomas Kirste danken. Er war technischer Projektleiter im EMBASSI-Projekt und später von Oktober 2003 bis Anfang 2005 mein Abteilungsleiter am Fraunhofer-IGD. Dies war die intensivste Phase meiner wissenschaftlichen Arbeit, die mit einem hohem Lerntempo und einer hohen Lernkurve verbunden war. Für diese Zeit möchte ich aufrichtig Danke sagen.

Ich bedanke mit bei meinem Abteilungsleiter Dr. Reiner Wichert für die tolle Zusammenarbeit, die Unterstützung während dem Abfassen dieser Dissertation und vor allem für das mir entgegen gebrachte Vertrauen in Bezug auf die von mir verantworteten Projekte und Arbeiten.

Zuletzt bedanke ich mich bei allen Kollegen im Haus der Graphischen Datenverarbeitung, die mir in Phasen von Stress und Zweifeln zur Seite gestanden haben.

Schließlich möchte ich mich bei meinen Eltern bedanken, bei meinem Vater Günter und meiner Mutter Elfriede († 30. April 1996), die mich bis hierher nach besten Kräften unterstützt haben,

und vor allem bei meiner lieben Frau Christine, die das Leben an der Seite eines wissenschaftlichen Mitarbeiters mit längeren Dienstreisen, der Arbeit an Wochenenden und den Aufs und Abs zwischen Enthusiasmus und Frust bei technischen Entwicklungen mit Toleranz und Großmut erträgt, mich stetig bestärkt und mich dabei wissen lässt, dass sie stolz auf mich ist und mich liebt.

Darmstadt im Juni 2007 Michael Hellenschmidt

Zusammenfassung

Den Umgang mit Technologie zu verändern ist die Hauptidee von Ambient Intelligence. Nicht mehr der Benutzer bedient die Technik, nachdem er sich vorher seiner Ziele bewusst wurde, diese Ziele selbst in Funktionen umgedacht hat und diese Funktionen einzeln unterschiedlichen Geräten zugewiesen hat, sondern die Technik wird den Menschen vielmehr maximal in der Durchführung seiner Ziele und Wünsche entlasten. Die Ideen des Ambient Intelligence stellen so den Menschen in den Mittelpunkt neuer technischer Entwicklungen. Schon frühzeitig hat die IST Advisory Group, die die europäische Kommission in Fragen von Zukunftstechnologien berät, hierzu eine Reihe von Schlüsseltechnologien identifiziert, die für den Aufbau von intelligenten Umgebungen und von Benutzungsassistenz wichtig sind.

Diese Arbeit stellt Lösungen für einen Teil der dieser Schlüsseltechnologien vor. Der erste Teil der Arbeit spezifiziert eine verteilt implementierbare Software-Infrastruktur, die eine dynamische Orchestrierung von Geräten ermöglicht und mittels Konfliktlösungsstrategien die selbstorganisierende Kommunikation und Kooperation von autonomen Geräteeinheiten realisiert. Gegenüber anderen Software-Lösungen zur Koordination von Geräteensembles weist die vorgestellte Lösung die Vorteile einer erhöhten Unterstützung von Dynamik auf. Der Verzicht auf zentrale Entitäten und die Möglichkeit domänenabhängige Konfliktlösungsstrategien zu definieren machen die Verwendung der Software-Infrastruktur in einer Vielzahl von selbstorganisierten Szenarios möglich.

Der zweite Teil stellt eine Komponententopologie für Ambient-Intelligence-Szenarios für die Realisierung von multimodaler Interaktion in intelligenten Umgebungen vor und definiert die dafür nötigen Selbstorganisationsstrategien. Auf Basis dieser Grundlagen werden multimodale und assistive Anwendungen im Bereich Unterhaltungselektronik und Automobil vorgestellt und diskutiert. Im Anschluss daran werden auf Basis dreidimensionaler Visualisierungen verschiedene Entwicklungswerkzeuge zur Kontrolle des Informationsflusses innerhalb eines Komponentenensembles beschrieben, sowie Regelinterpreter zur Konfiguration intelligenter Umgebungen eingeführt und deren Anwendung beschrieben.

Die vorgestellte Arbeit umfasst somit die von der ISTAG beschriebenen Schlüsseltechnologien zur erweiterten Infrastruktur, zur funktionalen Koordination und intelligenter Planung, sowie zur benutzerbasierten Assistenz und reflexiver Systeme. Nach Diskussion der Ergebnisse werden Inhalte momentaner und zukünftiger Projekte im industriellen, landwirtschaftlichen und häuslichen Umfeld auf Basis der hier vorgestellten Arbeit beschrieben, sowie Hinweise auf neue wissenschaftliche Herausforderungen gegeben.

Abstract

The main idea of the Ambient Intelligence vision is a paradigm chance of traditional human-computer-interaction. Yet, humans have to handle technical devices manually and they have to be aware of the devices' functions and their interrelationships and cooperation possibilities. In future, Ambient Intelligence will assist users in achieving their goals and will thus disburden humans from excessive demands and cognitive overload in technical or foreign environments. First of all, Ambient Intelligence puts the human being in the centre of future technical developments. Already at an early stage of Ambient Intelligence the IST Advisory Group identified a variety of key technologies that are important for the assembling of smart environments and for the implementation of personal user assistance.

This thesis presents solutions for some key technologies for Ambient Intelligence environments. The first part of the thesis specifies a distributed software infrastructure that allows a dynamic orchestration of devices and realizes self-organized communication and cooperation of autonomous device entities by means of execution of conflict resolution strategies. There are some advantages of the presented software infrastructure compared to other software solutions. That is primarily an increased support of dynamic ensemble behaviour. This is achieved by abandonment of any central component and the application of domain-dependent conflict resolution strategies. Thus the presented software infrastructure could be used in a variety of self-organized Ambient Intelligence scenarios.

The second part introduces a component topology for Ambient Intelligence that supports the implementation of multimodal interaction in smart environments. Furthermore the necessary strategies for self-organization for multimodal environments are defined. Based on these fundamentals multimodal and assistive applications in the home entertainment as well in the automobile domain are described and discussed. Directly following, based on visualization techniques, developing tools for information flow control within a component ensemble are presented. Additionally a rule interpreter that allows the configuration of smart environment behaviour as well as its application is described and discussed.

The presented thesis comprises solutions for key technologies as defined by the ISTAG. The main focus of the presented work relies on enhanced infrastructure, functional co-ordination, intelligent scheduling, and user profiling respectively reflexive systems. After an in-depth discussion of the results current projects and their applications based on this thesis' results are introduced. They are dealing with industrial, agricultural and home scenarios and demonstrate the large impact of Ambient Intelligence on today's information technology. This thesis ends with an outlook on future scientific work.

Inhaltsverzeichnis

1 Einleitung

Es war ein langer Weg seit der damalige Vorsitzende von IBM Thomas Watson im Jahre 1943 die Meinung äußerte, dass es auf der Welt einen maximalen Bedarf an vielleicht fünf Computern gäbe. Auch die Meinung von Ken Olsen, der Gründer und damalige Präsident der Digital Equipment Company, von 1977, dass es keinen Grund gäbe, warum irgend jemand in der Zukunft einen Computer bei sich zu Hause haben sollte, hat sich sehr bald darauf als grundlegend falsch erwiesen. In den nächsten Jahrzehnten setzte sich die Durchdringung der Technik im Alltags- und Berufsleben des Menschen immer mehr durch. Während sich noch in der Phase von 1950 bis 1970 mehrere Menschen einen Computer teilen mussten und dieser ausschließlich für komplexe berufliche Aufgaben eingesetzt wurde, startete ab 1980 der Personal Computer seinen Siegeszug vom rein beruflichen Einsatz in die Wohnzimmer und privaten Schreibzimmer. Der Einsatz von PCs fand jedoch singulär getrennt voneinander in den Büros statt. Erst in den 90ern des vorigen Jahrhunderts fand mit der Verbreitung der Internettechnologie auch die Vernetzung von Computern statt. Zeitgleich fanden Mikroprozessoren in immer neuen Geräten (und auch in neuen Formen bereits bekannter Geräte) immer stärkere Verbreitung und ermöglichten neue Funktionalitäten. Somit hat Marc Weiser im Jahre 1993 nicht untertrieben [215] als er feststellte, dass jeder Erwachsener in einem industriellen modernen Staat täglich mit mehr als 40 Geräten mit intern eingebauten Mikrochips konfrontiert ist (siehe Abbildung 1). Der Mensch hat somit viele Kontakte – bewusst und unbewusst – mit Technologie und Mikrotechnologie. Zwei Jahre zuvor schon schrieb Weiser den heute noch wegweisenden Aufsatz „The Computer of the 21st Century“ [213] in dem er die immer weitere Durchdringung des menschlichen Lebens durch elektronische Geräte vorhersah. Viel weiter sah Weiser die Möglichkeit einer Vielzahl von Geräten, deren der Mensch einzeln nicht mehr bewusst sein kann. Diese Durchdringung sei nahtlos und führe nahezu zum Verschwinden der Technik. Mit diesem Papier führte Weiser den Begriff des *Ubiquitous Computing* – allgegenwärtige Technik[1] – ein. Aber schon in diesem Papier beschreibt Weiser auch, dass – neben der rein technologischen Anforderungen dieser Vision – auch die kognitiven und psychologischen Anforderungen sowohl an den Menschen als Benutzer als auch an die Techniker und Entwickler nicht unterschätzt werden dürfen. Erst wenn es erreicht sei, dass „machines that fit the human environment instead of forcing humans to enter theirs will make using a computer as refreshing as a walk in the woods“ [213]. Später erweiterte Weiser diese Aussage und bezog ausdrücklich die Notwendigkeit der Arbeit von Sozialwissenschaftlern, Philosophen und Anthropologen in seine Ausführungen mit ein [214]. Durch Interdisziplinarität soll es erreicht werden, eine radikale Änderung dessen zu erreichen, das bis dato als Stand der Technik in Bezug auf Netzwerkfähigkeit von Geräten und die menschliche Interaktion galt[2]. Im Folgenden wurde den beiden grundsätzlichen

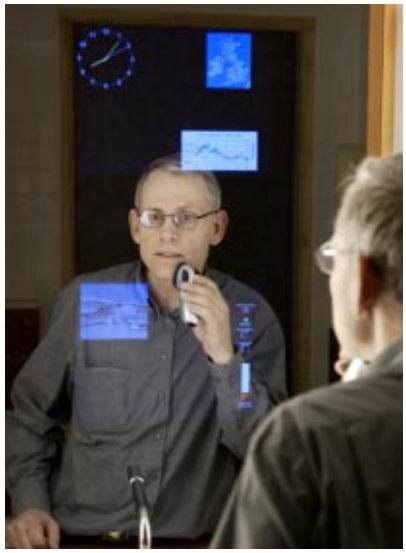
[1] Weiser definiert: „Ubiquitous computing is the method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user.“

[2] Aufgrund der Eindrücklichkeit soll das Zitat von Weiser hier im Original wiedergegeben werden: „Inspired by the social scientists, philosophers, and anthropoligists at PARC (Bemerkung: Weiser meint hier die Forschungslabor des Xerox Parc), we have been trying to take a radical look at what computing and networking ought to be like. We believe that people live through their practices and tacit knowledge so that

Der Mensch ist umgeben von Geräten

Auch kleine Geräte bieten immer mehr Funktionen

Mikroprozessoren werden immer kleiner und leistungsfähiger



Geräte integrieren sich ineinander und adaptieren sich auf den Benutzer

Abbildung 1: Oben: Eindrücke von Gerätelandschaften (Fotos: CeBIT 2006). Unten: Kleinste Mikroprozessoren machen Computer in Alltagsgegenständen Realität (Fotos: Philips Research).

Forderungen von Weiser – Unterstützung des Benutzers durch Änderung der Interakionsmetaphern und notwendige technische Voraussetzungen – besonders die Weiterentwicklung der Technologie verfolgt. Satyanarayanan [185] von der Carnegie Mellon University definiert die Herausforderungen die an diese Forschungsrichtung des sich aus der Idee

the most powerful things are those that are effectively invisible in use. This is a challenge that affects all of computer science. Our preliminary approach: Activate the world. Provide hundreds of wireless computing devices per person per office, of all scales. This has required new work in operating systems, user interfaces, networks, wireless, displays, and many other areas. We call our work ubiquitous computing.“

des Ubiquitous Computing entstandenen *Pervasive Computing* gestellt werden. Pervasive Computing soll sich hierbei aus einer Entwicklungskette beginnend von Forschungserfolgen auf den Gebieten der Verteilten Systeme und endend bei Technologien für Mobile Computing entwickeln. Verteilte Systeme seien hierbei durch die Möglichkeit zur Kommunikation zu entfernten Komponenten, durch Fehlertoleranz, durch die Garantie hoher Verfügbarkeit und des Zugriffs auch auf entfernt liegende Daten sowie durch hinreichende Schutz- und Sicherheitsmechanismen definiert. Dies wird ergänzt mit Technologien des *Mobile Computing*, die gekennzeichnet sind durch die Unterstützung mobiler Netzwerke, durch den mobilen Zugriff auf Daten und durch adaptive Applikationen[3]. Erst wenn diese technischen Voraussetzungen gegeben seien, könne die Forschungsrichtung des Pervasive Computing sich auf Neuerungen der Mensch-Maschine-Interaktion konzentrieren[4]. Dies seien besonders Anforderungen an die Unterstützung von Multimodalität, an die Entwicklung intelligenter Software-Assistenten für die Sicherstellung von proaktivem Systemverhalten und die Weiterentwicklung von Methodiken der künstlichen Intelligenz wie Expertensysteme, Planungssysteme und Entscheidungstheoreme. Aber schon 1988 erkannte Donald Norman [166], dass es nicht die Technik ist, die sich dem Menschen anpasst. Die Menschen seien bereit ihre Verhaltensweisen zu verändern, alleine wenn sie daran glaubten, dass die dabei benutzte Technologie ihr Leben verbessern könnte. Die Technologie sei aber nur nutzbar, wenn der Mensch verstünde wie mit ihr umzugehen ist. Norman schließt aber aus Berichten und Erfahrungen von Menschen im Umgang mit komplizierten Alltagsgegenständen wie Videorekordern oder Waschmaschinen, dass vielmehr Frust entstehe. Darüber hinaus würden Menschen viele technische Geräte gar nicht mehr benutzen, aus Angst einen Schaden anzurichten. Zugespitzt formuliert Norman dies als Technophobie. Der Mensch setze seine Hoffnungen in neue Technologien, wird dann aber aufgrund komplizierter Interaktionsmetaphern und komplizierter Funktionalität in der richtigen Benutzung der Geräte gehindert. Norman macht hier jedoch nicht die Menschen als Benutzer für die aufkommende Technophobie verantwortlich. Vielmehr seien sich die Entwickler und Designer ihrer Aufgabe nicht bewusst, die Technik nicht der Funktionalität willens zu entwickeln, sondern immer im Hinblick auf die Menschen, die sie benutzen sollen. Norman bekräftigt hier seine Vorstellung, dass Alltagsgeräte leicht zu bedienen sein müssten und darüber hinaus deren Nutzen und deren Funktionalitäten leicht und offensichtlich zu verstehen sein müssten. Man kann Normans Worten somit auch als Aufforderung verstehen: Technologie muss selbsterklärend sein und den Menschen darin unterstützen seine Ziele und Wünsche zu erreichen. Marc Weiser hatte also recht, als er treffend formulierte: „Applications are of course the whole point“ [215].

[3]Satyanarayanan definiert hier die Fähigkeit adaptiver Applikationen jedoch nicht aus Benutzersicht, sondern aus Systemsicht. Adaptivität bedeute die Fähigkeit einer Applikation sich auf äußere technische Umstände wie Netzwerklast, das Vorhandensein von Proxies oder begrenzte Ressourcen wie Batteriekapazitäten zu adaptieren.

[4]Satyanarayanan spricht hier selbst von „research challenges in areas outside computer systems“

1.1 Ambient Intelligence: Ideen und Visionen

Während sich die Anstrengungen und Entwicklungen auf den Gebieten des Ubiquitous und Pervasive Computing auf rein technologische Fortschritte konzentrierten (Aarts und Encarnação sprechen hier von „innovations by technology-push“ [4]) stellte die IST Advisory Group[5], die die europäische Kommission in Fragen von Technologie und Forschung berät, diesen die Idee des *Ambient Intelligence* entgegen[6]. Diese Ideen trugen und tragen weit in das Framework Programme 5, 6 und 7 der europäischen Forschungsprojekte hinein[7]. Schon in der Beschreibung der Ambient-Intelligence-Vision ist erkennbar, dass hier Technologie nicht für sich selbst entwickelt werden soll, sondern immer im Einklang mit dessen Nutzen und im Einklang mit den Bedürfnissen des Benutzers (Aarts und Encarnação sprechen hier von „innovations by user-pull“ [4])[8]. *Ambient Intelligence* macht es dem Benutzer möglich, alle Möglichkeiten, die sich dem Benutzer in seiner technischen Umgebung bieten, auch nutzen zu können. Der Benutzer befindet sich hierbei in einer technischen Umgebung, die aufmerksam ist für die gegenwärtige Situation des Benutzers, für dessen Ziele und Wünsche und die in der Lage ist die Persönlichkeit des Benutzers und dessen Präferenzen zu berücksichtigen. Die ISTAG definiert dies als den Ambient-Intelligence-Space mit dem der Benutzer intuitiv interagieren kann. Die Geräte der Umgebung „verschmelzen“ somit zu einem integrierten Gerät, mit dem der Benutzer auf natürliche Art und Weise interagiert. Der Benutzer wird dabei in den Mittelpunkt aller Betrachtungen gestellt [68]. Laut Emile Aarts wird diese neue Herangehensweise Technik für den Menschen und dessen Bedürfnisse zu entwickeln die Art und Weise Forschung und Technologie voranzubringen revolutionieren. Von Anfang jeder technischen Entwicklung an, wird der Nutzen einer Applikation oder einer Technologie im Vordergrund stehen [5][9]. Nicht die künstliche Intelligenz, so wie sie in der Vergangenheit interpretiert wur-

[5]Information Society and Technology Advisory Group, http://www.cordis.lu/ist/istag.htm

[6]Bei Streitz [199] finden sich hierzu die Erläuterung, dass Pervasive Computing „system-oriented, importunate smartness“ ist und Ambient Intelligence „people-oriented, empowering smartness“. Ambient Intelligence folge somit der Metapher, dass „smart spaces make people smarter“.

[7]Dabei verfolgt Ambient Intelligence neben dem Fortschritt in Technologie und Forschung auch in besonderer Art und Weise die von der europäischen Kommission lancierte Lissabon-Strategie. Hierzu findet sich in [121]: „Start creating an ambient intelligence landscape (for seamless delivery of services and applications) in Europe relying also upon test-beds and open source software, develop user-friendliness, and develop and converge the networking infrastructure in Europe to world-class“.

[8]Das Paradigma von Ambient Intelligence definiert sich gemäß der IST Advisory Group folgendermaßen [122]:„The concept of Ambient Intelligence is used where intelligence is pervasive and unobtrusive in the surrounding environment. Such an environment is sensitive to the presence of living creatures (persons, groups of persons´ and maybe even animals) in it, and supports their activities. It remembers and anticipates in its behaviour. The human and physical entities – or their cyber representatives – together with services share this new space which encompasses the physical and virtual world, the AmI Space. This space needs to be engineered so it has predictable behaviours, so that services can be offered through it, and so it can manage complicated many-to-many relationships. The AmI Space could be seen as the integration of functions at the local level across the various environments, enabling the direct natural and intuitive dialogue of the user with applications and services spanning collections of environment, as well as at the cyberspace level, enabling knowledge and content organization and processing.“

[9]Die ISTAG definiert dieses frühzeitige Einbeziehen von Benutzern in den Entwicklungsprozess in seinem Papier „Involving Users in the Development of Ambient Intelligence“ [124], welches eines der Basis-

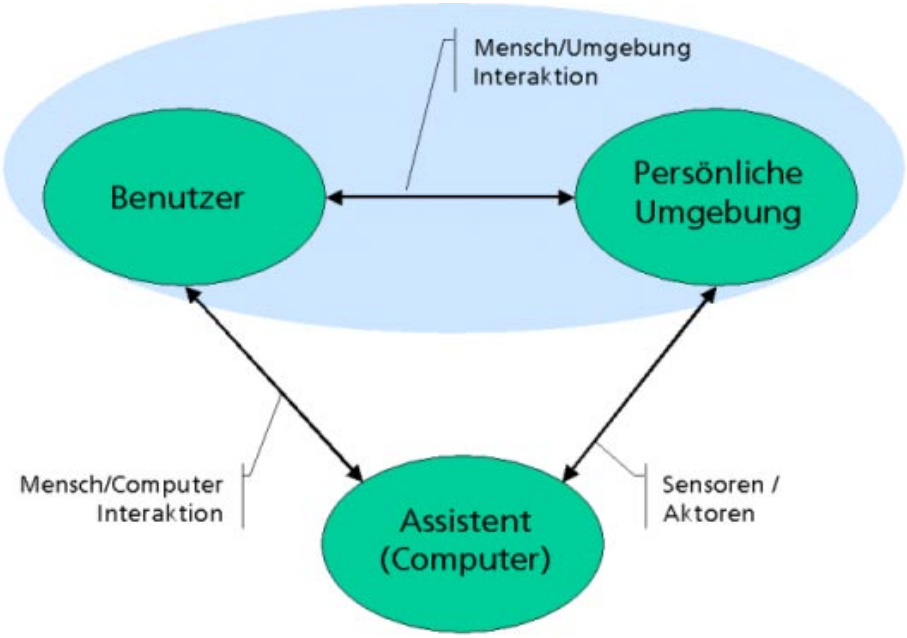


Abbildung 2: Ambient Intelligence stellt den Menschen in seiner persönlichen Umgebung in den Mittelpunkt. Als „Human in the Loop“ kann er auf natürliche Weise mit seiner Umgebung interagieren und wird dabei unterstützt und assistiert.

de, stehe im Mittelpunkt zukünftiger Entwicklungen und Forschungen, sondern die Einbeziehung von sozialer Intelligenz. Dies meint der Zusatz *Ambient*. Fünf Eigenschaften identifiziert Aarts [5] als Wesentlich für die Beschreibung von Ambient Intelligence: (1) eingebettet – viele (unsichtbare) verteilte Geräte innerhalb der Umgebung, (2) bewusst – eine Umgebung, die sich ihres aktuellen Zustandes bewusst ist, (3) personalisiert – ein Verhalten, das gemäß den persönlichen Bedürfnissen angepasst werden kann, (4) adaptiv – ein Verhalten, das proaktiv auf die Interaktionen des Benutzers reagieren kann und (5) antizipativ – eine Umgebung, die die Wünsche und Bedürfnisse des Benutzers vorausahnen kann [10]. Abbildung 2 illustriert hierbei die durch Ambient Intelligence angeregte Vorgehensweise. Mittels natürlicher und natürlich-sprachiger Interaktion ist es dem Benutzer möglich, innerhalb seiner persönlichen Umgebung zu interagieren. Die einzelnen Geräte und Funktionen sind gewissermaßen für den Benutzer unsichtbar. Unbemerkt vom Benutzer interagiert er mit intelligenten Assistenten, die die expliziten und impliziten Interaktionen des Benutzers wahrnehmen und interpretieren können. Encarnação spricht hier vom „Human in the Loop“ [67]. Dieser Paradigmenwechseln macht es möglich,

dokumente des Forschungsprogrammes des 7. Frameworks bildet. Die Ausschreibungen zu Projekten des 7. Frameworks beginnen im Frühjahr 2007.

[10] Der Klarheit wegen sollen diese fünf Punkte hier in Original aufgeführt werden: (1) embedded – many inivisible distributed devices throughout the environment, (2) context aware – that know about their situational state, (3) personalized – that can be tailored towards your needs and can recognize you, (4) adaptive – that can change in response to you and you environment, (5) anticipatory – that anticipate your desires without conscious mediation.

dass intelligente Umgebungen der Zukunft die zielorientierte Interaktion des Benutzers möglich machen. Der Benutzer muss nicht mehr konkrete Funktionen einzelner Geräte und Applikationen bedienen, sondern kann in seiner persönlichen Umgebung seine konkreten Wünsche und Ziele definieren.

Die grundsätzlichen Anforderungen, die an eine intelligente Umgebung zu stellen sind, sind laut Aarts und Encarnação wie folgt definiert [4]:

1. Die Umgebung (und damit die in ihr befindlichen Geräte und Applikationen) müssen sich der Situation, in der der Benutzer sich befindet, bewusst sein. Gleichzeitig muss die Umgebung in der Lage sein, die Interaktionen des Benutzers zu erfassen und in einem Kontext zur gegenwärtigen Umgebungssituation und Benutzersituation zu stellen.

2. Zusätzlich muss die Umgebung in der Lage sein, die Benutzerinteraktionen zu möglichen Benutzer- und Umgebungszielen zu interpretieren. Die Interpretationen haben auf Basis der Voraussetzung zu geschehen, dass der Benutzer in seinen Tätigkeiten und seinen Zielen maximal unterstützt wird.

3. In einem letzten Schritt setzt die Umgebung die zuvor gefundenen Ziele und Aktionen um. Hierzu verfügt sie über geeignete Strategien, wie auf Basis der vorhandenen Geräte und Applikationen die Umgebung im Sinne des Benutzers verändert werden kann.

Ambient Intelligence unterstützt somit den Menschen bei ihrer Interaktion mit der persönlichen Umgebung und setzt dabei die zur Verfügung stehenden Funktionen eines allgegenwärtigen IuK-Geräteensembles kooperativ ein. Dabei agiert der Geräteverbund als „Mittler“ zwischen dem Benutzer und seiner persönlichen Umgebung. Die Definition von Ambient Intelligence von Emile Aarts [3] soll hier im Zitat wiedergegeben werden: „Ambient Intelligence refers to the presence of a digital environment that is sensitive, adaptive, and responsive to the presence of people. Within a home environment, ambient intelligence will improve the quality of life of people by creating the desired atmosphere and functionality via intelligent, personalized inter-connected systems and services. Ambient intelligence can be characterized by the following basic elements: ubiquity, transparency, and intelligence. Ubiquity refers to a situation in which we are surrounded by a multitude of interconnected embedded systems. Transparency indicates that the surrounding systems are invisible and moved into the background of our surroundings. Intelligence refers to the fact that the digital surroundings exhibit specific forms of intelligence, i. e., it should be able to recognize the people that live in it, adapt themselves to them, learn from their behaviour, and possibly show emotion.“. Ambient-Intelligence-Umgebungen sind somit gekennzeichnet durch ihre Reaktivität bzgl. der von einem Benutzer gemachten Interaktionen und Äußerungen und durch ihre Proaktivität hinsichtlich des Unterstützungsbedarfes des Benutzers bei dessen Aktivitäten. Der Teilbegriff „Ambient“ in Ambient Intelligence bezieht sich hierbei auf die weit reichende Einbindung von Technologien in Alltagsgegenstände und Umgebungen. Der Teilbegriff „Intelligence“

wiederum spiegelt wieder, dass die technische Umgebung spezielle intelligente Mechanismen und Strategien besitzt, um die Interaktion mit dem Menschen zu unterstützen und intelligentes Eigenverhalten daraus abzuleiten. Bis heute haben zahlreiche Institutionen und Forschungsinitiativen die Ideen von Ambient Intelligence aufgegriffen und arbeiten an ihrer Verwirklichung. Die Fraunhofer-Gesellschaft hat 2004 zwölf Leitinnovationen vorgestellt, die die Chancen für den Zukunftsmarkt Deutschland erhöhen sollen. Mehrere Themen dieses Technik-Portfolios sind hierbei stark von Ambient Intelligence getrieben. So definiert das Thema „Internet der Dinge“ selbstorganisierende autonome Objekte, „Intelligente Produkte und Umgebungen“ definiert die Integration von Gegenständen, Infrastrukturen und Umgebungen in selbstständige selbstorganisierende Ensembles zur Bereitstellung umfassender Assistenz und das Thema „Mensch-Maschine-Interaktion“ definiert die Unterstützung der Interaktion mit technischen Gerätestrukturen[11]. International seien hier die Tätigkeiten der „Ambient Intelligence Research & Development“ der Firmen Philips (Niederlande) und Thomson (Frankreich) und der Forschungsinstitutionen INRIA (Frankreich) und der Fraunhofer-Gesellschaft (Deutschland) erwähnt[12]. In dieser Initiative findet ein regelmäßiger Austausch von Forschungsergebnissen und Ideen statt, sowie die gemeinsame Planung von Ambient-Intelligence-Projekten.

1.2 Realisierung von Ambient Intelligence

Ambient Intelligence führt einen grundsätzlichen Metaphernwechsel in der Interaktion des Menschen mit seiner Umgebung und den darin befindlichen Geräten ein (siehe Abbildung 3). Bis heute muss der Benutzer sich jedes Gerätes in seiner Umgebung bewusst sein und dieses zum Zwecke der Umsetzung seiner Wünsche und Ziele einsetzen und bedienen.

Abstrakt lässt sich ein Benutzerziel als Wunsch nach einer Veränderung der persönlichen Umgebung wie folgt definieren:

$$E(x_i) \rightarrow E^{'}(x_i^{'})$$

Dabei zeigen die Parameter x_i an, dass eine Umgebung durch eine endliche Liste an internen Variablen bestimmt ist. Hat nun der Benutzer das Ziel die gegenwärtige Umgebung $E(x_i)$ zur veränderten Umgebung $E^{'}(x_i^{'})$ zu manipulieren, so muss er sich hierfür eine geeignete Umsetzungsstrategie S überlegen:

$$E^{'}(x_i^{'}) = S(E(x_i))$$

Dabei besteht eine solche Strategie (oBdA) aus einer Abfolge geeigneter Funktionsaufrufe an verschiedene Geräte (vgl. Abbildung 3 links), in der Art dass:

$$S = f_j(E_{j-1}), j = 0..n$$

[11] Mehr Informationen hierzu sind zu finden unter: http://www.fraunhofer.de/fhg/company/perspectives/index.jsp.

[12] AIR&D Joint Virtual Laboratory for Ambient Intelligence, http://www.air-d.org, Stand Frühjahr 2007.

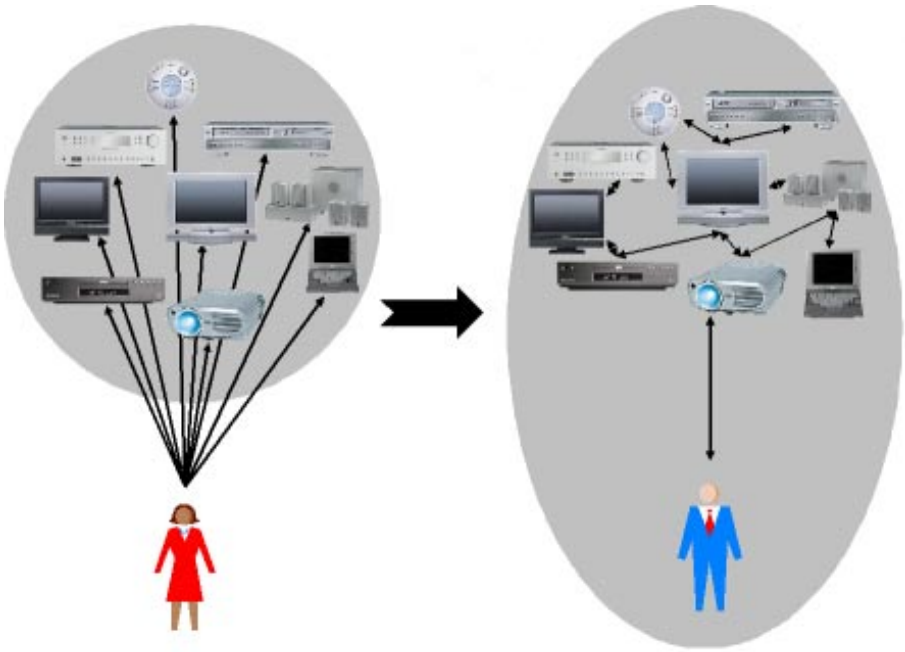


Abbildung 3: Illustration des Metaphernwechsels durch Ambient Intelligence in der Interaktion mit den Geräten der persönlichen Umgebung.

und

$$x_i^{'} = f_j(x_i), j = 0..n$$

Hierbei ist berücksichtigt, dass jede vorgenommene Aktion die vorhandene Umgebung bereits verändert. Der Benutzer muss somit nicht nur eine Funktionsreihe auf Basis des von ihm gewünschten Zieles und auf Basis der vorhandenen Geräte definieren, sondern zusätzlich bei der Definition dieser Funktionsreihe die Rückkopplungen auf mögliche Zwischenzustände der Umgebung berücksichtigen[13]. Ein einfaches Beispiel soll das hier Beschriebene verdeutlichen. In „normal“ ausgestatteten Wohnzimmern sind zumeist die drei Unterhaltungsgeräte Fernsehgerät, Satellitenempfänger und ein Aufnahmegerät (ein Videorekorder oder ein DVD-Rekorder) zu finden. Alle drei sind mit jeweils einer Fernbedienung zur manuellen Kontrolle der geräteeigenen Funktionen ausgestattet. Möchte der Benutzer nun eine Sendung aufzeichnen so muss er sich nicht nur überlegen, welche Geräte (aus diesen dreien) er dazu benötigt, sondern auch noch welches Gerät welche Funktion auszuführen hat. Kurz gefasst, muss der Benutzer sich entscheiden, dass für einen Aufnahmewunsch das Fernsehgerät unwesentlich ist[14], sondern am Satellitenrecei-

[13] Spätestens bei dieser rein formalen Überlegung wird klar, warum das Bedienen von Geräteensembles in fremden Umgebungen nahezu unmöglich ist. Der Mensch schreckt intuitiv davor zurück ihm fremde Geräte zu bedienen. Dies geschieht aus Furcht die Rückkopplungseffekte nicht berücksichtigen und einplanen zu können. Nach Norman [166] ist hier die Angst etwas unwiderruflich kaputt zu machen größer als der erhoffte Zugewinn.

[14] Schon bei diesem Schritt wird klar, dass und warum viele Menschen mit der Technik heutzutage überfordert scheinen. Es erscheint willkürlich und nicht intuitiv, dass gerade das Fernsehgerät nicht gebraucht

ver der gewünschte Programmkanal einzustellen ist und zugleich das Aufnahmegerät mit dem korrekten Datum zu programmieren ist. Gleichzeitig muss der Benutzer sicherstellen, dass die beiden Geräte korrekt miteinander verkabelt sind. Strenggenommen „wissen“ die beteiligten Geräte nichts voneinander und auch nichts davon, dass sie gemeinsam kooperieren, weil der Benutzer ein entsprechendes Ziel hatte. Der Tatsache, dass der Satellitenreceiver aus seinen Signalausgang ein Signal gibt, dass zur „richtigen“ Zeit von einem Aufnahmegerät gelesen wird, ist sich ausschließlich der Benutzer bewusst. Aus der Ausführung eines sehr konkreten – und auch einfachen – Zieles eine Sendung aufzuzeichnen, sind diverse Entscheidungen und aufeinanderfolgende Funktionsaufrufe geworden. Die Ideen des Ambient Intelligence (vgl. Abbildung 3 rechts) werden diese Art und Weise mit Geräten interagieren zu müssen ändern. Encarnação formuliert diese Änderung der Bedienungsmetaphor und auch die Änderung in der Erwartungshaltung der Benutzer wie folgt [68]: „Die Umgebung muss das machen, was der Benutzer von ihr erwartet. Dabei müssen wir auch erwarten, dass der Benutzer nicht die gesamten Technologien und Geräte, die ihm zur Verfügung stehen, in diesem Kontext selbst orchestriert, sondern das System muss das für ihn selbst organisieren und das Ganze muss multimodal sein, denn wir wollen nicht nur multimedial sein, sondern wir wollen Sprachein- und ausgabe realisieren. Das heißt, wir müssen kohärent sein, wir müssen von der Technologie, die die meisten zur Verfügung haben und ggf. durch Assistenzsysteme unterstützt ist, zu einer selbstorganisierenden Funktionalität kommen.“. Aarts und Encarnação definieren drei wichtige technischen und wissenschaftliche Fragen [4], die zur Realisierung von Ambient Intelligence gelöst werden müssen (vgl. Abbildung 3 rechts):

- Die Interaktion des Benutzers mit seiner Umgebung: Sobald der Benutzer nicht mehr gezwungen ist, mit jedem Gerät einzeln zu interagieren um deren Funktionen zu bedienen, muss es das Ziel sein, ihn in der Verwendung natürlichsprachiger Interaktion zu unterstützen. Die Interaktion wird sich somit von einer rein funktions- und geräteorientierten Interaktion zu einer zielorientierten Interaktion verändern. Dabei wird die Umgebung die Sprache des Benutzers (z. B. durch Gestikerkennung und Spracherkennung) sprechen und nicht mehr umgekehrt. Es bedarf somit großer technischer und auch sozio-wissenschaftlicher Fortschritte, um diese Kehrtwendung in der Interaktion zu erreichen. Sowohl Fortschritte in der Multimodalität als auch Fortschritte in der Interpretation von Benutzeräußerungen mit Einbeziehung situativer Kontextdaten müssen hier erreicht werden[15].

- Die Kooperation eines Geräteensembles: In Zukunft kann von einem Benutzer nicht mehr erwartet werden, dass er eine Vielzahl von Geräten in einem Geräteensembles organisiert und orchestriert. Um dies zu erreichen müssen die Geräte vielmehr selbst geeignete Strategien zur Selbstorganisation und zur Sicherung von Interope-

wird, um ein Fernsehprogramm aufzuzeichnen.

[15] Bei einer früheren Debatte von Shneiderman und Maes [195], die sich mit der Relation von direkter Manipulation gegen Delegation auseinandersetzten, ergab sich, dass unterschiedliche Szenarios und Kontexte auf Basis unterschiedlicher Interaktionsmetaphern realisiert werden können. Ambient Intelligence darf sich daher nicht auf eine bestimmte Interaktionsmetapher beschränken.

rabilität und Kooperation besitzen und bereitstellen können. Diese Selbstorganisation und Kooperation hat hierbei ad-hoc und autonom zu geschehen und nicht durch den Benutzer. Dieser muss hierbei vollständig von Konfigurations- und Koordinationstätigkeiten entlastet werden. Geeignete Software-Infrastrukturen müssen somit gewährleisten können, dass ein Geräteensemble in einer kohärenten Art und Weise interagieren kann.

- Die Reaktion eines Geräteensembles auf Umgebungsänderungen: Neben den rein kommunikativen und kooperativen Fähigkeiten der Geräte muss auch erreicht werden, dass sie sich der momentanen Umgebungssituation und der Situation des Benutzers „bewusst“ sein können und daraus intelligente Schlüsse ziehen können. Die Umgebung soll auf eine vernünftige Art und Weise auf die Interaktionen und Wünsche des Benutzers reagieren. Konsequenterweise sind hier Ambient-Intelligence-Technologien nötig, die selbst kontextbewusst und vernünftig handeln können.

Bereits in Dokumenten der IST Advisory Group finden sich konkrete Hinweise und Aufgabenbeschreibungen, welches Technologien für die Realisierung von Ambient-Intelligence-Szenarios von Wichtigkeit sein werden. Hierzu seien drei Schlüsseltechnologien weiterzuentwickeln und letztendlich miteinander zu integrieren (vgl. [123]). Diese seien *Ubiquitous Computing*, *Ubiquitous Communication*, und *Intelligent User-Friendly Interfaces*[16]. Erst wenn diese Konvergenz der drei Schlüsseltechnologien erreicht sei, kann davon gesprochen werden, dass der Mensch von (benutzerfreundlichen) miteinander vernetzten und kooperativen Geräten umgeben ist, die sich der Gegenwart des Benutzers, seiner individuellen Persönlichkeit und seiner Bedürfnisse und Wünsche bewusst sind. Aufgeschlüsselt in konkrete Technologien sind dies [122]: Intelligente Benutzerprofilbildung, Verteilte Datenbanken, Sicherheitstechnologien, Lernende und Reflexive Systeme, Sensoren und Aktuatoren, Mensch-Machine Schnittstellen zur Unterstützung natürlichsprachiger Interaktion, Kommunikationsunterstützung von intelligenten Objekten, Koordinations- und Kooperationsmechanismen und Software-Infrastrukturen. Besonders die letzten Punkte werden im Dokument „Grids, distributed systems and software architectures“ [125] der Arbeitsgruppe 8 nochmals herausgestellt. Essentiell für die Realisierung von Ambient Intelligence seien Fortschritte in der Kommunikation individueller Geräte, die über die reine Bereitstellung von Netzwerkfunktionalität hinausgehen und die Applikationen in der Kommunikation und Kooperation unterstützen. Die Vielfalt von heterogenen

[16] Schon aus dieser Definition wird klar, dass Ambient Intelligence weit über die Ideen des Ubiquitous Computing hinausgeht. Ubiquitous Computing wird relativiert zu einer Basistechnologie. An dieser Stelle soll nochmals die ISTAG zitiert werden, um die Relationen deutlich zu machen: „humans will, in an Ambient Intelligent Environment, be surrounded by intelligent interfaces supported by computing and networking technology that is embedded in everyday objects such as furniture, clothes, vehicles, roads and smart materials – even particles of decorative substances like paint. AmI implies a seamless environment of computing, advanced networking technology and specific interfaces. This environment should be aware of the specific characteristics of human presence and personalities; adapt to the needs of user; be capable of responding intelligently to spoken or gestured indications of desire; and even result in systems that are capable of engaging in intelligent dialogue. Ambient Intelligence should also be unobtrusive – interaction should be relaxing and enjoyable for the citizen, and not involve a steep learning curve.“

Technologien die im Kontext von AmI zum Einsatz kommen bedingen die Bereitstellung von geeigneten Middleware-Technologien für spontan vernetzte Systeme, die zum einen die Verschiedenartigkeit bestimmter Technologien berücksichtigen und integrieren können, zum anderen auch Strategien und Mechanismen für die Unterstützung von Gerätekooperation und Gerätekommunikation anbieten. Nur dadurch kann gewährleistet werden, dass Geräte ein spontanes Geräteensemble zu bilden im Stande sind und einen kohärenten und koordinierten Verbund schaffen. Die Kooperation der unterschiedlichen Geräte dieses Systems wird ermöglicht durch verteilte Strategien, die die Selbstorganisation und Selbstkonfiguration von Geräteensembles unterstützen und hierbei eine Adaption an die jeweiligen Bedürfnisse oder Wünsche der unterschiedlichen Akteure ermöglichen.

1.3 Inhalt und Gliederung der Arbeit

Diese Arbeit beschäftigt sich mit den von Aarts und Encarnação definierten Herausforderungen zur Realisierung von Ambient-Intelligence-Szenarios. Hierbei wird sich in besonderer Art und Weise mit dem Thema einer verteilt implementierten Software-Infrastruktur zur Gewährleistung von spontaner Kooperation und Kommunikation von Geräte und deren Ensembles beschäftigt. Darüber hinaus werden Methodiken entwickelt, die die Kommunikation und die Kooperation von verteilten Komponenten- und Geräteensembles befördern. Ein weiterer Teil dieser Arbeit ist der Realisation von intelligentem[17] Verhalten von einzelnen Komponenten und von Komponentenensembles gewidmet. Beide Teile zusammen sind geeignet Ambient-Intelligence-Umgebungen und -Szenarios aufzubauen und zu entwickeln. Ein besonderes Augenmerk gilt hier der Unterstützung von prototypischen Verfahren zum schnellen Aufbau von solchen Umgebungen. Dem Entwickler (bzw. Gruppen von Entwicklern) sollen Werkzeuge und Methodiken an die Hand gegeben werden können, die es erlauben in vertretbarer Zeit intelligente Umgebungen bereitzustellen.

Die Kapitel dieser Arbeit sind daher wie folgt gegliedert. Im Folgenden (vgl. Kapitel 2) werden die Ideen und Realisierungen wesentlicher Projekte und Forschungsinitiativen im Ambient-Intelligence-Umfeld im Detail besprochen und analysiert. Die Analyse konzentriert sich hierbei auf die beiden wesentlichen Fragestellungen der Kooperation und Kommunikation der beteiligten Geräte und der Realisation von intelligentem Verhalten der implementierten Szenarios. Zum Abschluss dieses Kapitels werden die in den unterschiedlichen Projekten identifizierten Anforderungen bezüglich der darunterliegenden Software-Infrastrukturen analysiert und zusammengefasst. Auf Basis dieser Anforderungsliste – die für die Realisierung einer Software-Infrastruktur für Ambient-Intelligence-Szenarios unverzichtbar ist – werden in Kapitel 3 bekannte Software-Infrastrukturen analysiert und hinsichtlich der Anforderungen diskutiert. Im Anschluss daran wird das dieser Arbeit zugrunde liegende SODAPOP-Modell eingeführt (vgl. Kapitel 4) und dessen Prinzipien für den Aufbau von kooperierenden Geräteensembles besprochen. Kapitel 5 definiert und spezifiziert die verteilte Implementierung einer Software-Infrastruktur auf Basis des SODAPOP-Modells. Dabei führt Kapitel 5 zuerst in die wesentliche Begriffe und Abstraktionen einer solchen Software-Infrastruktur ein und stellt in zwei Kapiteln die Realisierung von verteilten Komponentenensembles (siehe Kapitel 5.1) und verteilten Geräteensembles (siehe Kapitel 5.2) vor. Ein besonderer Schwerpunkt dieses Kapitels liegen in der Möglichkeit der Software-Infrastruktur Konfliktlösungsstrategien auszuführen. Dies stellt eine wesentliche Voraussetzung für die Realisierung spontaner ad-hoc Kommunikation und Kooperation dar. Ein weiterer Schwerpunkt liegt in der Beschreibung der Programmierschnittstelle, die es Entwicklern erlaubt, eigene Komponentenensembles zu definieren und zu programmieren. Die Entkopplung der unterlagerten Kommunikation von der Komponentenfunktion wird in diesem Kapitel genau beschrieben. Direkt im Anschluss daran werden in Kapitel 6 Typen an Komponenten für Ambient-Intelligence-Szenarios identifiziert und mögliche Konfliktlösungsstrategien zur Unterstützung des Aufbaus von Applikationen auf Basis der identifizierten Komponen-

[17] Der Autor möchte hier lieber von vernünftigem Verhalten sprechen.

tentypen vorgestellt. Die Ergebnisse von Kapitel 5 und Kapitel 6 stellen eine komplette Software-Infrastruktur zur verteilten Implementation von kommunizierenden und kooperierenden Komponenten- und Geräteensembles dar. Basierend auf diesen Arbeiten werden in Kapitel 7 Anwendungen aus unterschiedlichen Projekten erläutert. Der adaptive Auswahlassistent aus dem EMBASSI-Projekt[18] stellt eine intelligente Komponente im Sinne der in diesem Abschnitt definierten kontext- und benutzerabhängigen Komponenten dar. Aus den Anwendungen des DYNAMITE-Projektes[19], in dessen Kontext wesentliche Teile dieser Arbeit entstanden, stellt Kapitel 7.2 die wichtigsten Eigenschaften der Software-Infrastruktur für verteilte Komponentenensembles vor. Durch Experimente wird die Arbeitsweise der in Kapitel 6 eingeführten Konfliktlösungsstrategien für die Gewährleistung von spontaner ad-hoc Konnektivität dargelegt. Zusätzlich beschreibt dieser Abschnitt die Implementation einer Dialogmanager-Komponente für die Festlegung von Benutzerzielen und beschreibt die Berücksichtigung von Multimodalität in dieser Arbeit. Die letzte in Kapitel 7.3 beschriebene Anwendung in diesem Kapitel spezifiziert eine regelgestützte Komponente zur kontext-abhängigen Assistenz von Benutzern im automobilen Umfeld. Kapitel 8 beschreibt einen weiteren Schwerpunkt dieser Arbeit – eine Sammlung an Werkzeugen und Methoden zur Unterstützung des Entwicklungsprozesses beim Aufbau intelligenter Umgebungen. Dies ist ein Visualisierungswerkzeug zur Darstellung des Datenflusses von verteilt agierenden Komponenten sowie der Einsatz von dreidimensionalen Abbildungen der Umgebung, die es intelligent zu machen gilt. Ein Schwerpunkt dieses Kapitels ist die Spezifikation einer regelbasierten Inferenzmaschine zur Konfiguration intelligenten Verhaltens einer Umgebung. Die hier vorgestellte Inferenzmaschine kann zur Realisierung intelligenten Verhaltens in vielfältiger Art und Weise eingesetzt werden. Die vorliegende Arbeit gibt Beispiele in Form von Reiseassistenz (vgl. Kapitel 7.3) und in Form von funktionsbasierter Raumsteuerung (vgl. Kapitel 8.2.2 und Kapitel 8.2.4). Zuletzt werden in dieser Arbeit die Ergebnisse besprochen und ausführlich diskutiert. Kapitel 9 geht hier vor allem auf zwei wesentliche Aspekte ein. Die Benutzbarkeit und die Leistungsfähigkeit der in dieser Arbeit entstandenen Software-Infrastruktur sowie die Verwendbarkeit der in dieser Arbeit realisierten intelligenten Komponenten. Der Ausblick in Kapitel 10 stellt notwendige zukünftige Arbeiten zur Fortentwicklung von Ambient Intelligence vor und beschreibt momentan laufende Projekte, die auf Teilen der hier vorgestellten Arbeit basieren.

[18] EMBASSI (Elektronische Multimediale Bedien- und Service Assistenz) wurde gefördert vom Bundesministerium für Bildung und Forschung (BMB+F)- FKZ 01 IL 904 U8. Laufzeit: Juli 1999 - Juni 2003.

[19] DYNAMITE (Dynamisch adaptive multimodale IT-Ensembles) wurde gefördert vom Bundesministerium für Bildung und Forschung (BMB+F) - FKZ 01 ISC 27A. Laufzeit: Oktober 2003 - September 2006.

2 Szenarios und Anforderungen

Für die Realisierung intelligenter Umgebungen auf der Basis heterogener Geräten und Komponenten sind Software-Infrastrukturen notwendig, die die dynamische Vernetzung, die ad-hoc Kommunikation sowie die Kooperation von Ensembles von Geräten und Komponenten ermöglichen. Mit der Analyse von bekannten Projekten, die den Aufbau von intelligenten Umgebungen realisiert haben, und den sich daraus ergebenden Anforderungen an eine zugrunde liegende Software-Infrastruktur wird im Folgenden der Begriff der Selbstorganisation von Komponentenensembles definiert. Diese Definitionen bilden den Ausgangspunkt der weiteren Arbeiten.

2.1 Szenarios intelligenter Umgebungen

Schon Mitte 1999 starteten die 6 Leitprojekte der Mensch-Technik-Interaktion des Bundesministeriums für Bildung und Forschung (BMB+F) [54]. Diese auf jeweils vier Jahre angelegten Projekte nahmen in den Projekten EMBASSI, MAP und SmartKom in ihren Anwendungsszenarios schon einige Ideen der integrierten Assistenz für den Benutzer und der Interaktion mit einem Geräteensemble, die sich auch in Ambient Intelligence wiederfinden, vorweg[20]. Auch in anderen Forschungslaboren weltweit wurde in Projekten an der Realisierung intelligenter Umgebungen geforscht. Bekannte Arbeiten daraus wurden für die Analyse ihrer Szenarios für die Erstellung einer Anforderungsliste für Software-Infrastrukturen herangezogen. Besonders die Arbeiten in den Projekten EMBASSI und DYNAMITE flossen in die Definition der Selbstorganisation von verteilten Komponentenensembles für die Realisierung von intelligenten Umgebungen ein. Sie bilden daher den „Rahmen“ der in diesem Kapitel besprochenen Projekte.

2.1.1 EMBASSI

EMBASSI [61, 65] (für: Elektronische Multimediale Bedien- und Service Assistenz) hatte sich zum Ziel gesetzt Szenarios in drei unterschiedlichen Anwendungsdomänen – Privathaushalt, Kraftfahrzeug und Terminalsysteme – zu entwickeln. Die Bedienassistenz für den Privathaushalt in EMBASSI (siehe Abbildung 4 links) basiert in Teilen auf dem folgenden Szenario [62]:

„Es ist 7 Uhr. Die Stimme des Home-Assistenten weckt mich mit meiner Lieblingsmusik und erinnert mich an einen ersten wichtigen Termin um 8 Uhr. Auf dem Weg ins Bad weise ich das System an, die Kaffeemaschine einzuschalten und die Jalousien hochzufahren. Die Musik folgt mir mit ins Bad und wird dort nahtlos fortgesetzt. Während ich mein Frühstück bereite, lasse ich mir am Küchenradio die heutigen Termine anzeigen. Mittels Sprachbefehlen initiiere ich ein Telefonat zu einem Geschäftspartner – dass das Gespräch statt auf das Telefongerät auf die eingebauten Lautsprecher und Mikrophone in der Küche weitergeleitet werden soll, weiß das System selbst. Nach dem Gespräch setzt automatisch

[20] Informationen zur abschließenden Statustagung, die am 3./4. Juni 2003 in Berlin stattfand, sowie die Vorträge der einzelnen MTI-Leitprojekte Arvika, EMBASSI, Invite, MAP, Morpha und SmartKom finden sich unter http://informatiksysteme.pt-it.de/mti-2/cd-rom/index.html (Stand Januar 2006).

die Musik wieder ein – meine neue Audioanlage hat sich problemlos in das vorhandene Gerätesystem eingefügt. Die Funktionen der neuen Anlage wurde allen Assistenzsystemen in der Wohnung mitgeteilt, jetzt nutzt der Fernseher ihr Surround-System bei Musiksendungen und Spielfilmen. Das geräteeigene Bedien-Assistenz-System nutzt umgekehrt auch die Ausgabegeräte (wie Sprachausgabe) der übrigen Systeme. Ich überfliege noch kurz das heutige Fernsehprogramm in der elektronischen Programmzeitschrift. Das Fußballspiel heute Nachmittag soll aufgezeichnet werden. Weil der Fernsehton noch läuft, nutze ich nicht die Sprachsteuerung, sondern zeige auf die angezeigte Sendung und den Videorekorder und bestätige den rückgemeldeten Auftrag. Das System weist mich darauf hin, dass die Videokassette fast voll ist und schlägt als Alternative die Festplatte im Server als Speichermedium vor. Natürlich nutze ich für die meisten Standardbedienfunktionen noch die verschiedenen klassischen I/O-Geräte, aber in manchen Fällen ist ein intelligenter Assistent doch hilfreich. Er hat sich auch sehr gut auf mich eingestellt."

Der EMBASSI-Anwendungsbereich Kraftfahrzeug unterscheidet drei Anwendungsfälle [63]. Das Szenario „Ich möchte Popmusik hören" zielt darauf ab, den Benutzer von der Aufgabe zu entlasten, aus einer sehr hohen Angebotsmenge aussuchen zu müssen. Hier muss nicht mehr explizit ein Gerät wie z. B. das Radio oder der CD-Spieler und der entsprechende Sender oder Musiktitel angesprochen werden, sondern hier ist ein Dialog auf einer höheren Abstraktionsebene vorgesehen, bei dem der Fahrer das Musikgenre angeben kann. Ziel ist es, dem Gerät einen verbal geäußerten Benutzerwunsch mitzuteilen. Hierbei kann zunächst das Musikgenre angegeben werden z. B. „Ich möchte Popmusik hören". Der Fahrerwunsch wird dann an eine Assistenzebene weitergeleitet. Das System sucht dann mit Hilfe eines geeigneten Agenten beispielsweise nach einem Radioprogramm, das dem Musikgeschmack bzw. dem geäußerten Benutzerwunsch entspricht, wobei der Entertainment Agent in einer Datenbank Informationen über die aktuell verfügbaren Musiktitel z. B. aus dem Radio oder im CD-Wechsler gespeichert hat. Werden mehrere Treffer gefunden, kann ein Auswahldialog mit dem Benutzer geführt werden oder der Agent entscheidet aufgrund von Nutzerpräferenzen und Kontextinformationen eigenständig. Das System kann, zum Beispiel, Auskunft darüber geben, welche Radiosender zur Verfügung stehen und welche hauptsächlichen Musikformen momentan gesendet werden. Die Information über den Erfolg oder Misserfolg bei der Erfüllung des Fahrerwunsches ist im Sinne der Transparenz des Systems sinnvoll.

Ein weiteres Teilszenario widmet sich der „Fahreraufmerksamkeitskontrolle": Der Fahrer eines KFZ befindet sich zum Beispiel auf der Autobahn oder einer Landstraße. Während der Fahrt wird kontinuierlich über eine Abstandsmessung die Distanz zum vorausfahrenden Fahrzeug ermittelt. Parallel zu dieser Messdatenerfassung wird über ein geeignetes Kamera-System die Blickrichtung des Fahrers ermittelt. Hierbei sind Blicke nach rechts zum Beifahrer oder aus dem Fenster, sowie Blicke nach links aus dem Fenster registrierbar. Um die Fahrsicherheit zu gewährleisten, soll das System nun bei zu langem Abwenden des Blickfeldes vom Straßenverkehr eine geeignete Warnung erzeugen, die den Fahrer dazu veranlassen soll, sich wieder der Fahrtätigkeit zu widmen. Dabei sollen zusätzlich die Abstandsdaten verwendet werden, um die Warnung auszulösen, denn bei großem Abstand zum Vordermann kann ein größerer Zeitraum der Ablenkung akzep-

EMBASSI - Privathaushalt

EMBASSI - Kraftfahrzeug

Abbildung 4: Zwei der Demonstratoren des EMBASSI-Projektes. Links der Demonstrator für den Privathaushalt wie er auf der Statustagung der MTI-Leitprojekte am 3./4. Juni 2003 in Berlin ausgestellt wurde. Rechts ein Demonstrator des EMBASSI-Szenarios Kraftfahrzeug auf der Mensch & Computer-Konferenz (M&C) am 5.–8. März 2001 in Bad Honnef.

tiert werden, als bei sehr geringem Abstand oder auch bei sich schnell reduzierendem Abstand. Im ersten Demonstrator wird dann eine audiovisuelle Warnung generiert. Ein Agent überwacht also permanent die aktuelle Blickrichtung und dessen zeitlichen Verlauf. Dabei wird parallel die Abstandsinformation im zeitlichen Verlauf berücksichtigt, um nach einem vorgeschriebenen Regelwerk gegebenenfalls eine Aktion zu veranlassen.

Im dritten und letzten Teilszenario des Anwendungsbereiches Kraftfahrzeug wird die „em Reaktion auf eine Störungsmeldung“ behandelt. In diesem Szenario soll der Einsatz von multimodalen Ausgaben in Abhängigkeit der Situation gezeigt werden. Die Situation ergibt sich aus der Entfernung zu einem Störungsort und der Fahrzeugumgebung. Die Störungsmeldungen (z. B. Verkehrsmeldungen, Glatteiswarnungen) erreichen das Fahrzeug und werden in Abhängigkeit der Situation unterschiedlich ausgegeben. Dazu wird vom Fahrzeug die Fahrzeugposition kontinuierlich erfasst. Störungsmeldungen erreichen das Fahrzeug über mobile Kommunikationskanäle. Es erfolgt eine Relevanzprüfung für die aktuelle Strecke, auf der sich das Fahrzeug befindet. Die Assistenzkomponente erstellt eine Strategie zur Aufbereitung der Information für den Fahrer. Das Fahrzeug wertet zusätzlich die fahrzeugeigenen Sensoren aus und ermittelt so verkehrs- und umfeldbedingte Gefahrenmomente. So ist es z. B. möglich, dass das Fahrzeug selbst Nebel oder Glatteis erkennt. Diese sich verändernden Fahrsituationen fließen in die Strategie zur Aufbereitung der Information für den Fahrer ein. So wird bei Nebel früher gewarnt und es kann eine vorsichtigere Annäherung an die Gefahrenstelle erfolgen.

2.1.2 SmartKom

Als ein weiteres Projekt der MTI-Leitprojekte erforschte SmartKom [177, 178, 210] im Schwerpunkt innovative Konzepte zur multimodalen Interaktion von Mensch und Technik. Statt der traditionellen Desktop-Metapher steht in SmartKom das situierte, dele-

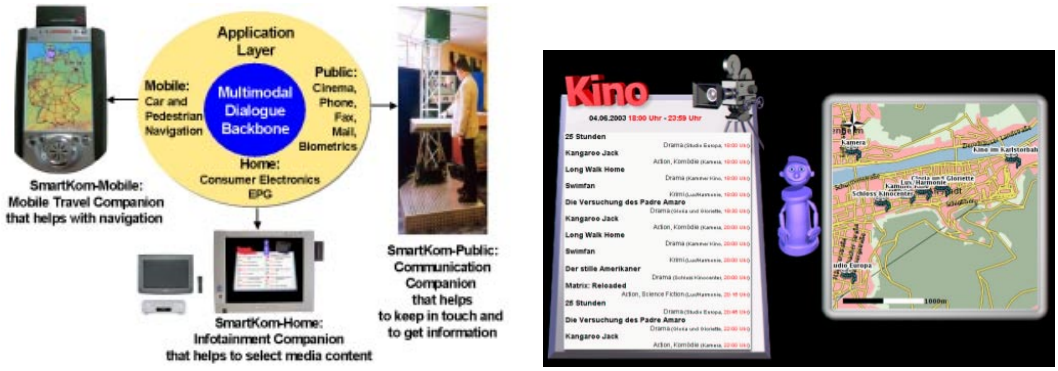


Abbildung 5: Links die Illustration der drei Anwendungsszenarios von SmartKom [177]. Die Abbildung rechts [178] visualisiert die Systemergebnisse auf die Benutzeranfrage „Zeig mir das Kinoprogramm heute abend“.

gationszentrierte Dialogparadigma im Zentrum der Benutzerinteraktion. Mittels Gestik, Mimik und Sprache interagiert der Benutzer mit dem System. Dabei muss er nicht alle Lösungsschritte vorgeben, sondern kann das Problem auch wieder an das System zurückgeben. Die Interaktion mit dem System findet dialogartig statt. SmartKom widmet sich hierbei drei unterschiedlichen Anwendungsszenarios (siehe Abbildung 5): einem öffentlichen Informationskiosk, einer Interaktionszentrale für das Wohnzimmer und einem Kommunikationsassistenten für die Wegsuche in mobilen Anwendungen. *SmartKom Public* unterstützt den Benutzer bei der Informationssuche in Flughäfen, Bahnhöfen und anderen öffentlichen Plätzen. Hier kann der Benutzer nach örtlichen Kinoprogrammen suchen, öffentliche Einrichtungen einsehen und personalisierte Kommunikation mittels Telefon, Fax oder Email benutzen. Spracheingabe wird mittels eines gerichteten Mikrophons aufgenommen, Mimik und Gestik mittels einer Kamera. *SmartKom Home* ist ein multimodaler Infotainment-Gefährte [178] der bei der Auswahl geeigneter Medien hilft und dem Benutzer bei der Bedienung der zahlreichen Gerätefunktionen unterstützt. Zusätzlich verfügt der Benutzer über eine elektronische Programmzeitschrift (Electronic Program Guide - EPG) um die Programme des Fernsehers und die Programmierung des Videorekorders zu ermöglichen. *SmartKom Mobile* ist ein mobiler Assistent für die Navigation und für ort-basierte Dienstleistungen. Es benutzt einen PDA (Personal Digital Assistant) und ermöglicht Reiseplanung und Routenplanung auf der Basis der kürzesten Distanz. Hier in diesem Szenario wird Multimodalität durch die Kombination von Sprache und Eingabestift erreicht.

2.1.3 MAP

MAP (Multimedia Arbeitsplatz der Zukunft) [78, 217] war das dritte Leitprojekt, das sich mit assistenzbasierter Mensch-Technik-Interaktion im Sinne der Ideen von Ambient Intelligence befasst hat. Der Schwerpunkt der Arbeiten lag auf der Unterstützung des mobilen Benutzers im Arbeitsumfeld. Ein mobiler Assistent erlaubt den mobilen Zugriff auf berufliche Daten, auf Termine und ermöglicht Dienstleistungen wie Routenplanung oder das

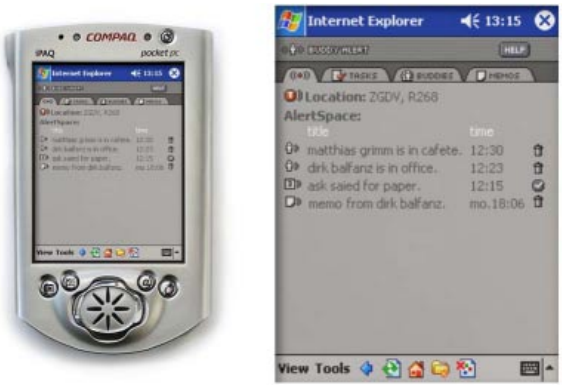


Abbildung 6: Die graphische Benutzeroberfläche der Anwendung BuddyAlert in MAP (aus [24]).

Buchen von Flügen, Taxis und Hotels. Dabei agiert der Assistent auch proaktiv und erinnert an Termine und schlägt selbstständig mögliche Flüge und Hotels vor. Die Interaktion mit dem Assistenten erfolgt entweder per Stifteingabe oder per Spracheingabe. Ebenso ist die Ausgabe per Sprache oder per Graphischer Oberfläche möglich. Dies kann abhängig von der gegenwärtigen Situation sein. Die BuddyAlert-Anwendung [24] folgt dieser Idee den Benutzer mit einer permanenten Erinnerungsstütze zu assistieren (siehe Abbildung 6). Hier kann der Benutzer Aufgaben definieren, die mit bestimmten Personen oder Dokumenten verknüpft sein können. Trifft der Benutzer eine dieser Personen, so erfolgt eine Erinnerung. Hierbei werden die Dokumente auf die gegenwärtige Situation des Benutzers adaptiert.

2.1.4 Interactive Workspaces

Das Interactive-Workspaces-Project [130] der Stanford University erforscht neue Möglichkeiten der Zusammenarbeit von Gruppen. Diese Zusammenarbeit soll stark von Technologie assistiert werden (siehe Abbildung 7). Ursprünglich wurde das Projekt ins Leben gerufen um die Interaktionsmöglichkeiten von Informationsarbeitern im Umgang mit großen Bildschirmen zu untersuchen. Hierzu wurde der *interactive Room (iRoom)* konzipiert, der zusätzlich die Integration von mobilen Geräten erlaubt. Die Realisierung des iRooms erfolgte nach Prinzipien die den Prinzipien des Ambient Intelligence ähnlich sind. Nicht das technisch Mögliche wird realisiert, sondern das was die assistivsten und nutzbarsten Szenarios umsetzbar macht. So wurde der Raum schon in der Aufbauphase für regelmäßige Arbeitsgruppentreffen genutzt, sowohl um die bereits vorhandenen Möglichkeiten des Raumes zu testen, als auch um neue Funktionalitäten für die Zukunft festzulegen („Practice what we preach“-Prinzip). Auf der anderen Seite wird versucht, die technischen Realisierungen so einfach wie möglich zu halten („Keep it simple“-Prinzip). Die betrifft vor allem die Mensch-Technik-Schnittstelle. Diese sollte so einfach gehalten sein, dass sie vom Benutzer auch wirklich angewendet wird (und angewendet werden kann). Ausgestattet mit drei berührungsempfindlichen White-Boards, einer Wandprojek-

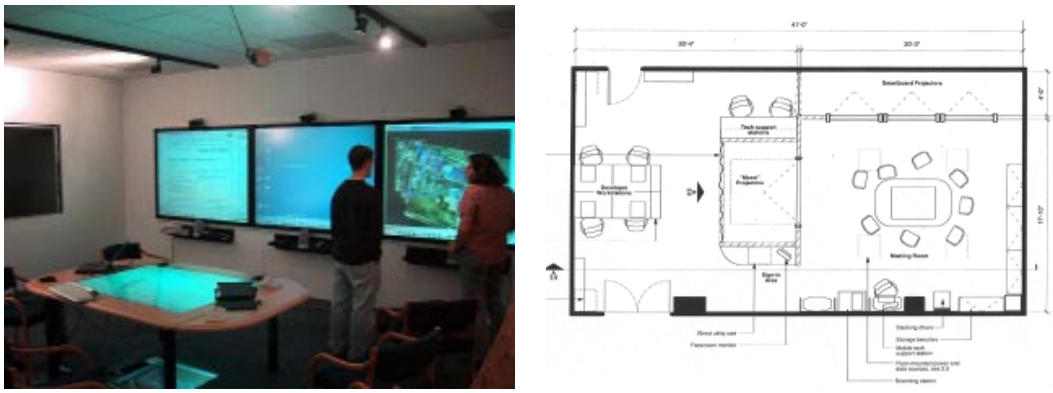


Abbildung 7: Der Interactive Room (iRoom) der Stanford University links, und dessen innen-architektonischer Aufbau rechts. Die Bilder sind der Veröffentlichung von Johanson et al. [130] entnommen.

tionsfläche (siehe Abbildung 7 rechts), einem Tisch mit eingebautem Display und einigen Kameras und Mikrophonen wurden die Art der Aktivitäten in so einem interaktiven Raum untersucht. Dabei wurden die folgenden drei Szenarios als relevant identifiziert: Das „Moving data“-Szenario erlaubt es den Benutzern Dokumente von Bildschirm zu Bildschirm zu verschieben. Hierzu kommen auch mobile Bildschirme – wie von den Benutzern mitgebrachte PDAs – in Betracht. Im Szenario „Moving Control“ soll jeder Teilnehmer einer Besprechung in der Lage sein jedes Gerät im Raum von seiner gegenwärtigen Position aus steuern zu können. Die Adaption von Dokumenten zur Nutzbarkeit auf unterschiedlichen Geräten ist wiederum Aufgabe des Szenarios „Dynamic application coordination“. Die Evaluationen der Szenarios fanden statt in Gruppen von 2 bis 15 Teilnehmern, wobei davon ausgegangen wird, dass diese über mitgebrachte Dokumente diskutieren und dies hauptsächlich am Hauptbildschirm (siehe Abbildung 7 rechts, „mural“ projectors) stattfindet. Eine Besprechung wird hierbei von einer vorher bestimmten Person geleitet. Interaktion findet statt, indem der Benutzer ein Dokument benennt, danach die geeignete Anwendung benennt, die für die Darstellung ausgewählt werden soll und dann das Display, auf dem das Dokument dargestellt werden soll. Zur Manipulation von Beleuchtung, Auswahl von Displays etc. wird der *Room Controller* eingesetzt. Hierbei handelt es sich um eine graphische Benutzeroberfläche, die mittels der bekannten Raumdaten für verschiedene Endgeräte (z. B. Java Swing, Palm Handheld) erzeugt werden kann. Hierzu wird die Infrastruktur-Software *iCrafter* [175] eingesetzt. Ein verkleinertes Szenario findet sich im *eClass*-Projekt [8] (vormals *Classroom 2000*) des Georgia Institute of Technology. Hier werden die Interaktionen einer Lehrkraft an einem interaktiven Whiteboard anhand der vom ihm erstellten Materialen dokumentiert und den Studenten nach der Vorlesung übers das Netzwerk zur Verfügung gestellt.

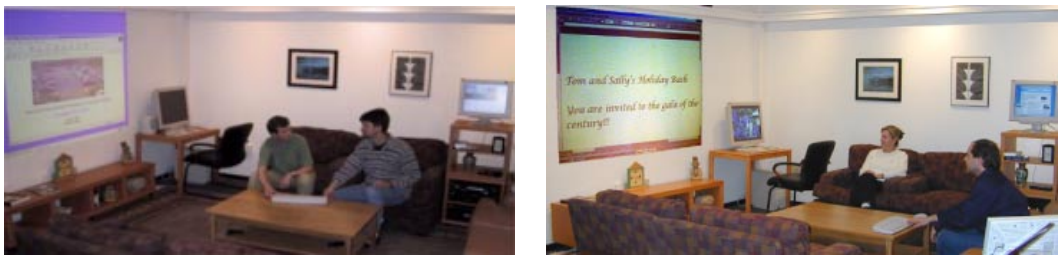


Abbildung 8: Die Umgebung des Easy-Living-Projekts von Microsoft besteht aus mehreren Monitoren und PCs, die entweder direkt oder durch einen zentralen Controller auf Basis eines statischen Umgebungsmodells und regelbasieren Inferenzmechanismus angesteuert werden. Die Abbildungen sind [33] und [57] entnommen.

2.1.5 Easy Living

Das Easy-Living-Projekt von Microsoft [57] beschäftigt sich mit der Entwicklung von Architekturen und Technologien für die dynamische Aggregation von Eingabe- und Ausgabegeräten. Das dem Aufbau zugrunde liegende Szenario ist wie folgt beschrieben [33]: „Tom kommt nach Hause. Nachdem er das Wohnzimmer betreten hat, setzt er sich an den PC, der in der Ecke steht. Hier wählt er ein paar MP3-Dateien aus und fügt diese seiner Playlist hinzu. Dann setzt er sich auf die Couch. Seine Sicht auf den Computer folgt ihm auf den großen Bildschirm, der der Couch gegenübersteht (siehe Abbildung 8). Dieser Bildschirm wurde vom System ausgewählt, da dieser verfügbar ist und sich in Toms Sichtfeld befindet. Tom nimmt die Fernbedienung vom Wohnzimmertisch und bedient die Raumsteuerung mit Hilfe des Trackballs. Diese Raumsteuerung wird ihm in einem Extrafenster auf dem großen Bildschirm dargestellt. Darauf wird ein Plan des Raumes dargestellt mit den Beleuchtungseinrichtungen, die bedienbar sind. Tom benutzt diese Schnittstelle um das Licht zu dimmen. Danach öffnet er seine Playlist und spielt sie ab. Die Musik aus den großen Lautsprechern. Jetzt kommt Sally in das Wohnzimmer und geht zum PC hinüber. Sie loggt sich manuell in das System des Raumes ein. Sie hat ein Word-Dokument dabei, bei es sich um eine Einladung für sie und Tom zu einer Feier handelt. Sie fragt Tom, ob sie den großen Bildschirm in der Mitte des Raumes benutzen kann. Tom gibt diese Benutzung mittels seiner Fernbedienung frei und Sally bewegt das Dokument mittels des PCs zum großen Bildschirm“ (siehe Abbildung 8, rechts). Im Mittelpunkt der Arbeiten stehen hier die Untersuchungen der unterschiedlichen technischen Möglichkeiten zur Kontexterfassung. In [146] sind die Wichtigsten aufgeführt: Personenidentifizierung (-und authentifizierung), Personenlokalisierung, Aktivitätserkennung (aufgrund von Bewegungen), Objekterkennung, Gestik und Sprache.

2.1.6 Intelligent Classroom

Der Intelligent Classroom der Northwestern University (Abteilung Intelligent Information Laboratory) [120] möchte Professoren, Geschäftsleute oder allg. Menschen, die eine Präsentation zu halten haben, unterstützen. Nötig sei hierzu eine Umgebung, die dazu in

Abbildung 9: Eine menschliche Geste interagiert implizit mit dem Intelligent Classroom.

der Lage ist, unterschiedliche Arten an Medien verarbeiten zu können. Zudem soll sie den Vortragenden von so viel persönlichem Aufwand entlasten können wie möglich. Der Intelligent Classroom verwendet Videokameras und Mikrophone, um die Gesten, Bewegungen und gesprochenen Sätze des Sprechers aufzuzeichnen (vgl. [80]). Der Raum übernimmt daraufhin die Kontrolle über Beleuchtung, die Kontrolle von Präsentationsfolien oder das Abspielen von Videos. Der Raum ist in der Lage [80] einen Film über eine gesamte Vorlesung zu produzieren, eine PowerPoint-Präsentation ablaufen und steuern zu lassen und einen Videorekorder zum Abspielen von Filmen zu steuern. Somit weist dieses Szenario viele Ähnlichkeiten zum *eClass*-Projekt (vgl. [8]) der Stanford University auf. Der besondere Schwerpunkt liegt bei diesem Projekt aber in der Unterstützung von Multimodalität. Dabei bedient sich die Systemstruktur zwei zentraler Informationsquellen, nämlich der persönlichen Präsentationsstragie des Vortragenden und temporal wichtiger physikalischer Ereignisse, wie Ortsveränderung des Sprechers, bestimmte Zeigegesten oder vokalischen Schlüsselwörtern. Damit ist der Intelligent Classroom in der Lage aus der Kenntnis der persönlichen Präsentationsstrategien in Verbindung mit der aktuellen Interaktion des Sprechers, die aktuellen Präsentationsziele zu inferieren und die Umgebung daraufhin gezielt zu manipulieren. Somit ist eine explizite Interaktion des Sprechers nicht mehr nötig. Das „Zeigen“ auf einen Gegenstand ist gleichbedeutend mit „Heranzoomen der Kamera“. Die Interaktion ist implizit. (siehe Abbildung 9). Der Intelligent Classroom verwendet einen zweistufigen Inferenzansatz. In einer ersten Interpretation werden die Interaktionen des Sprechers interpretiert, und erst in einer zweiten Interpretation aus den interpretierten Interaktionen Schlüsse auf das Verhalten des Gesamtsystems gezogen. Hierdurch werden gezielt Mehrdeutigkeiten verhindert, da z. B., zwei völlig unterschiedliche Handlungen des Sprechers dennoch dieselbe Ausführungsintension bedeuten können. Aber auch gleiche Interaktionen von verschiedenen Sprechern können gänzlich unterschiedliche Intentionen bedeuten und damit unterschiedliche Reaktionen des Raumes bedingen (vgl. die Ausführungen von Franklin und Flachsbart in [84, 85]).

2.1.7 Intelligent Room

Im Rahmen der Oxygen-Initiative [171] wurde am Massachusetts Institute of Technology (MIT) das Intelligent-Room-Projekt realisiert. Ähnlich wie das *Interactive-Workspaces*-Projekt und das *Easy-Living*-Projekt hat es zum Ziel, dass Computersysteme nicht mehr direkt bedient werden müssen, sondern aufgrund der Kombination von expliziten und im-

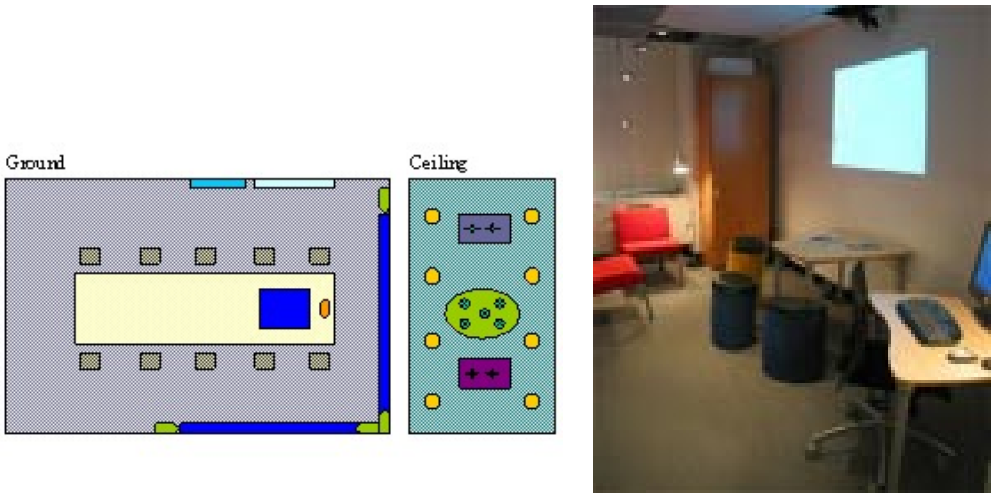


Abbildung 10: Der *Intelligent Room* des MIT. Abbildung des intelligenten Arbeitsplatzes (rechts) und Skizze der Gesamt-Innenarchitektur (aus [171])).

pliziten Benutzeräußerungen automatisch „das Richtige“ tun. Brooks [35] spricht hier von der Invertierung des Verhältnisses von Mensch und Computer. Der Raum (siehe Abbildung 10) verfügt über keine direkten Interaktionsmöglichkeiten wie Tastaturen oder Computermäuse[21]. Die Interaktionsmöglichkeiten konzentrieren sich somit auf Spracheingabe und Gestenerkennung. Den Forschungsaktivitäten im Intelligent Room liegen zwei Szenarios zugrunde, die unterstützend im Falle von beratenden Arbeitsgruppen wirken können. Das erste Szenario handelt von Planungen im Falle eines Katastropheneinsatzes, das Zweite bietet virtuelle Touren durch das MIT Artificial Intelligence Lab. Im ersten Szenario sitzen verschiedene Experten zusammen, die versuchen alle nötigen und verfügbaren Informationen für einen Katastrophenfall zusammen zu suchen. Der Raum beobachtet die in ihm befindlichen Personen. Mit den Worten „Computer, show me a map of the disaster area“ zeigt das System eine Lageplan auf dem Bildschirm an, der der Person am nächsten ist. Zeigt die Person nun auf die Karte und sagt „Computer, how far is the hurricane from here“, gibt der Computer die nötigen zusätzlichen Informationen. Ebenso sind Fragen wie: „Computer, where is the nearest airport“ oder „Computer, what is the weather like“ möglich. Das zweite Szenario ist für Besucher vorgesehen, die Hilfe bei der Orientierung im MIT Laboratory benötigen. Aber auch Fragen wie „Computer, tell me about the Gog project“ sind möglich (ergänzende Dialoge hierzu finden sich bei Coen [44]). Die Informationen sind nicht statisch auf Datenbanken verteilt, sondern werden dynamisch aus den Webseiten der Labormitglieder extrahiert. Im erweiterten Rahmen der *Oxygen*-Initiative sind ergänzende Schlüsselszenarios definiert worden. In einer *Business Conference* besprechen sich Projektmitarbeiter unterschiedlicher Länder und unterschiedlicher Muttersprachen. Die dabei benutzten Telefoneinrichtungen sind in

[21] Brooks [35] vergleicht die Realisierung mit der Fernsehserie *Star Trek, the next Generation*. Tatsächlich wird – wie auf den zur Webpräsenz der Oxygen-Initiative einsehbaren Videofilmen zu erkennen – der Computer im Intelligent Room dann aktiv wenn er mit dem Wort „computer“ aktiviert wurde. Die Wörter „go to sleep“ sind in der Lage den Computer zu deaktivieren.

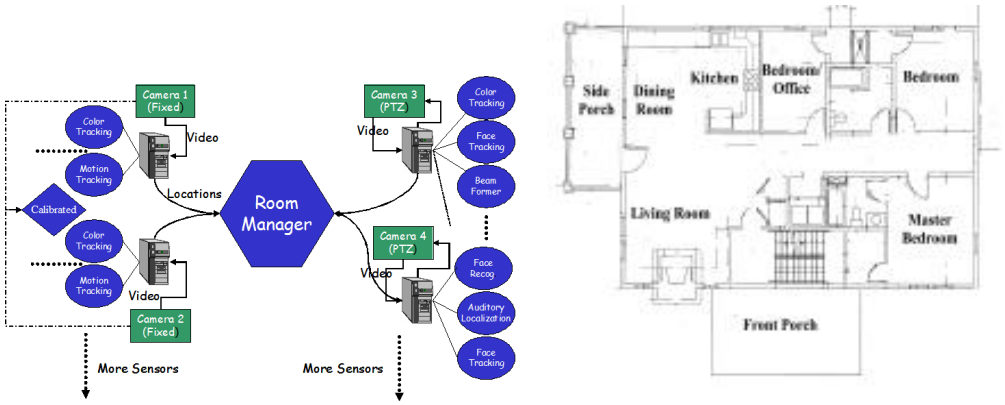


Abbildung 11: Die System-Architektur des Aware Homes (links) und der architektonische Aufbau des ersten Stocks (aus [134]).

der Lage in der jeweiligen Muttersprache des Anrufes Auskunft über Ab- und Anwesenheit des gewünschten Gesprächspartners zu geben. Wollen die Projektmitarbeiter ein gemeinsames Treffen verabreden, gleicht Oxygen die verschiedenen Kalender ab, schlägt mögliche Termine vor und erledigt die nötigen Hotel- und Flugreservierungen. Danach sind die persönlichen Handhelds in der Lage, den jeweiligen Mitarbeitern den Weg zum richtigen Gebäude und innerhalb des Gebäudes zum richtigen Raum zu weisen (in Analogie zum zweiten Szenario des Interactive-Workspaces-Projekts). Gleichzeitig sind die Handhelds in der Lage, die nötigen Dateien für das Meeting bei Bedarf zu organisieren. Das Szenario *Guardian Angel* soll vor allem das Leben von älteren und/oder behinderten Menschen erleichtern und ihnen ein automones Leben garantieren.

Die Bezeichnung *Oxygen* leitet sich im übrigen aus der Bezeichnung des Elementes Sauerstoff ab. Denn genauso wie Sauerstoff den Menschen umgibt, soll Oxygen die Vision des Pervasive Computing realisieren, dass der Mensch von allgegenwärtigen Computern umgeben ist, die ihn gewissermaßen umhüllen.

2.1.8 Aware Home

Das Aware-Home-Projekt – ebenfalls vom Georgia Institute of Technology – installiert eine Hausumgebung, um die Möglichkeiten allgegenwärtiger Technologie in der Unterstützung alltäglicher Handlungen zu untersuchen [134]. Das Aware Home besteht aus zwei identischen Stockwerken (siehe Abbildung 11, rechts), mit je zwei Schlafräumen, zwei Bädern, einem Büro, einer Küche, einem Speisezimmer, einem Wohnzimmer und einem Waschraum. Zur Ausstattung gehören zusätzlich eine Home-Entertaiment-Zone und ein Kontrollraum, in dem die computergesteuerten Dienste zentral zusammenlaufen (siehe Abbildung 11, links)[22]. Als Sensoren verfügt diese Umgebung über Positionser-

[22] Die Abbildung der System-Architektur ist der Präsentation „Building an Aware Home“ von Irfan Essa entnommen, die am 1. Juli 1999 im Mitsubishi Electric Research Laboratory gehalten wurde. Diese Präsentation ist unter der Webpräsenz des Aware-Home-Projekts unter http://www.awarehome.gatech.edu/ frei verfügbar.

kennung durch Ultraschallsensoren, RF-Technologie and Videoaufnahmen, sowie über im Boden eingelassene Sensoren. Mittels dieser Technologien sei es möglich, generelle Informationen (wie Lokation) über einen Bewohner zu erfassen und somit aufgrund der Interpretation von Umgebungsdaten den erweiterten Kontext der Bewohner zu identifizieren. Im Vordergrund der Forschungsbemühungen stehen Szenarios zur Unterstützung älterer Bewohner und deren Autonomie. Eine der Anwendungen hierbei ist ein System zum Auffinden verlorener Objekte. Objekte, wie Schlüssel, Brillen, oder Fernbedienungen sind mit kleinen RF-Tags versehen und können so mittels LCD-Bildschirmen, die den Bewohnern zur Verfügung stehen, gesucht werden. Das System unterstützt den Benutzer dabei auch akustisch („Die Schlüssel sind im Schlafzimmer"). Andere Anwendungen seien die Bereitstellung von Kommunikationsmethoden, und die Identifikation von kritischen Situationen, in denen Hilfe von außen ins Haus geholt werden muss.

2.1.9 HomeLab

Das HomeLab von Philips Research geht im Vergleich zum Aware-Home-Projekt noch einige Schritte weiter. Geplant im Jahr 2000 wurde es am 24. April 2002 eröffnet [2]. Es ist ein zweistöckiges Gebäude mit einem Wohnzimmer, einer Küche, zwei Schlafzimmern, einem Badezimmer und einem Arbeitszimmer. Zusätzlich ist es mit 34 Beobachtungskameras ausgestattet, die von 4 verschiedenen Räumen aus beobachtet werden können. Das HomeLab ist somit ein geeignetes Laboratorium, das das Testen von innovativen Technologien in einer natürlichen Umgebung erlaubt [153]. Ebenso gibt es einen Beobachtungsraum, der den direkten Blick in die Räume des HomeLabs gestattet. Hier werden von Verhaltensforschern der Umgang von Testpersonen mit neuen Technologien und Produkten evaluiert. Wenn so ein Experiment geplant wird, so erstellen die Forscher ein Schema, das alle prototypischen Verhaltensweisen die während des Experiments zu erwarten sind, aufzeichnen kann. Somit seien wirkungsvolle Wizard-of-Oz-Studien möglich (siehe Abbildung 12[23]). Ziel aller Untersuchungen im HomeLab ist es Technologie so unsichtbar wie möglich in die Umgebung zu integrieren. Philips selbst beschreibt die Aufgaben des HomeLabs wie folgt (siehe auch [183]): „Ambient Intelligence defines how Philips envisions the future. It describes technology that understands and anticipates your needs and reacts appropriately. All of the prototypes being tested at HomeLab are Ambiently Intelligent because they put people at the center of their functionality. The prototypes can think on their own and make your life easier, by acting with subtle or no direction. Ambient Intelligence technologies will be in your home, your car and even on you personally in the form of wearable electronics. Philips believes that the concept of Ambient Intelligence will be pervasive in our lives by the year 2020, if not sooner.". Eine Reihe von Szenarios (siehe hierzu [184]), die im HomeLab realisiert wurden, beschreibt diesen Realisierungsansatz den Mensch in den Mittelpunkt aller Forschungsbemühungen zu stellen. Das *DreamScreen*-Projekt untersucht die Nutzung und Nutzbarkeit von Bildschirmen, die sich über eine gesamte Wand oder zumindest ein Fenster erstrecken. Dies soll eine Art

[23] Bilder sind unter http://www.research.philips.com/newscenter/pictures/misc-homelab.html, dem Internetauftritt des HomeLabs, entnommen. Sie stehen hier der Öffentlichkeit frei zur Verfügung.

Abbildung 12: Das 2002 eröffnete HomeLab von Philips Research. Mit mehr als 30 Überwachungskameras ausgestattet (rechts oben) erlaubt es die Evaluierung von Prototypen (links unten) und die Erprobung neuer Technologien und Szenarios (rechts unten).

von Haustheater ermöglichen oder gar die Illusion erzeugen, an einem ganz anderen Ort zu sein [5]. Im *Interactive-Mirror*-Projekt werden verschiedene Konzepte für die Anwendungsbereiche Haushalt und Beruf untersucht. Im Badezimmer kann (während der Rasur etc.) in diesem Spiegel der aktuelle Wetterbericht oder Verkehrsinformationen abgerufen werden (siehe auch Abbildung 1). Mit speziellen Applikationen sei es möglich auf diesem Spiegel neue Haarfrisuren auszuprobieren, sowie durch im Bad eingebaute Kameras seine eigene Rückenansicht zu sehen. Eingeblendete Kontrollfunktionen können mittels Gestikerkennung aktiviert und deaktiviert werden. Im *interactiveCat (iCat)*-Projekt wird der Frage nachgegangen, in wie weit eine zentrale Figur als Ansprechpartner für die technischen Funktionen der Umgebung eingesetzt werden kann. Die *iCat*-Figur bietet hierbei zusätzlich Zugriff auf persönliche Daten (wie Kalender) und ist in der Lage Tipps zu geben bezüglich des täglichen Speiseplans und der Freizeitgestaltung. In diesem Projekt wird zusätzlich der Einfluss von Emotionen auf die Bedienung evaluiert (siehe [153]). Die Figur ist in der Lage die Gesichtmimik von überrascht zu fröhlich, von ängstlich zu traurig und von verärgert zu empört zu wechseln. Per Beobachtung der Benutzer wird hier das emotionale Zusammenspiel von Mensch und Technik evaluiert.

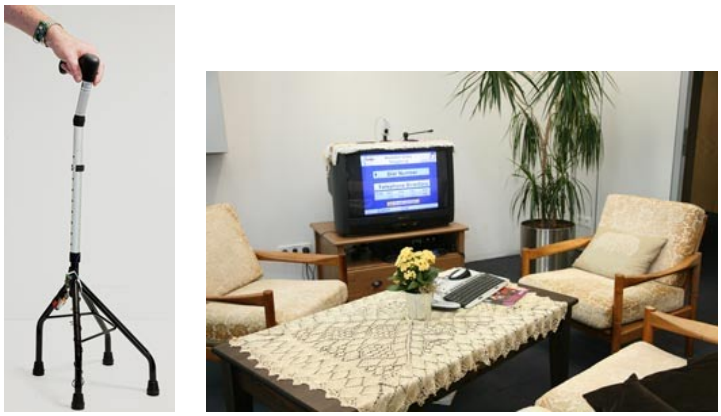


Abbildung 13: BelAmI: Gehhilfe mit Kippsensor zur Falldetektion, rechts ein Ausschnitt des Ambient Assisted Living Labors am Fraunhofer IESE.

2.1.10 BelAmI

Das BelAmI-Projekt (für Bilateral German-Hungarian Collaboration Project on Ambient Intelligence Systems) [30] ist ein deutsch-ungarisches Kollaborationsprojekt an dem von deutscher Seite vor allem das Fraunhofer-Institut für Experimentelles Software Engineering (Fraunhofer IESE) und die TU Kaiserslautern beteiligt sind. In gemeinsamen Aktivitäten und Teilprojekten werden Ambient-Intelligence-Technologien für den Aufbau von Assisted-Living-Szenarios erforscht und eingesetzt. Die Motivation dieser Initiative leitet sich aus vor allem der Tatsache ab, dass Europa sich in einem Alterungsprozess befindet. Den dabei einsetzenden Limitationen der physischen Fähigkeiten und der Bedürfnisse älterer Menschen nach medizinischer Versorgung und Sicherheit gerecht zu werden steht dabei im Mittelpunkt der Forschungsaktivitäten [181]. Ausgewählte Forschungsgebiete in BelAmI sind (vgl. [30]) z. B. *Beobachtung und Assistenz*, *Interaktion* und *Medikation*. Mittels Beobachtung und Assistenz soll es älteren Menschen ermöglicht werden, ein strukturiertes Alltagsleben erhalten zu können. Hierbei werden sie – durch IuK-Technologien unterstützt – erinnert, rechtzeitig ihre Medizin einzunehmen, zu trinken und zu essen. Lebenswichtige Vitalfunktionen werden regelmäßig zum Hausarzt gesendet. Andere Systeme sind in der Lage kritische Situationen zu erkennen, z. B. falls ein Bewohner gefallen ist oder einen Herzinfarkt erleidet (siehe Abbildung 13 links). Entsprechende Systeme sind durch die Fraunhofer-Gesellschaft auf der CeBIT 2007 [38] vorgestellt worden. Neuartige Interaktionstechnologien haben zum Ziel situationsabhängig den Kontakt und die Kommunikation von älteren Personen mit Freunden und Verwandten, aber auch mit Ärzten oder medizinischem Personal zu etablieren. Zuletzt sollen Forschungsaktivitäten auf dem Gebiet der Medikation ältere Personen bei der Einnahme ihrer notwendigen Medikamente unterstützen. BelAmI verfolgt somit das von Aarts und Encarnação in [4] beschriebene „innovations by technology-push“ (vgl. Rombach [180]) indem darauf abgezielt wird durch den Einsatz von IuK-Technologien das selbstbestimmte Leben älterer Menschen zu verlängern. Durch den Einsatz von Techniken der künstlichen Intelligenz, vernetzter Systeme und verteilter Sensoren können Basisdaten über den Tagesablauf

von Bewohnern eines Hauses gesammelt werden und durch anschließende Analyse von Situationen proaktiv Warnmeldungen, Hinweise, Erinnerungen oder das Auslösen von Notfallreaktionen bewirkt werden. Als Beispiele werden gegeben (vgl. [181]): Einsatz von RFID in Teppichen zur genauen Bestimmung des aktuellen Aufenthaltsorts, Überwachung regelmäßiger Flüssigkeitsaufnahme oder intelligente Kühlschränke. Die Ambient-Assisted-Living-Initiative BelAmI identifiziert auch Herausforderungen an Wissenschaft und Technologie, die sich aus den assistiven Szenarios ergeben. Hier steht die „Schaffung eines Rahmens in dem neue Technologien einfach und zuverlässig zu schlüssigen Gesamtkonzepten verbunden werden können“ (Zitat Thomas Kleinberger, Projektleiter am Fh-IESE [181]) neben der Realisierung offener Kommunikations- und Interaktionsplattformen an erster Stelle. Auch seien weitere Fortschritte in Techniken des Informationsmanagements und in der Kontext-, Situations- und Zielerkennung notwendig.

2.1.11 AMIGO

Die Arbeiten im Projekt AMIGO (Ambient Intelligence for the networked Home Environment) [11] begannen mit einem szenario-gestützten Ansatz um von Benutzern gezielte Meinungen bezüglich des gewünschten Verhaltens und der gewünschten Dienstleistungen von intelligenten Hausumgebungen zu erhalten. Die Studien hierbei wurden an sechs verschiedenen Orten in fünf europäischen Ländern (Spanien, Frankreich, Italien, Deutschland und Niederlande) durchgeführt. In AMIGO werden vier unterschiedliche Szenarios einer Familie (Mario, Jerry und ihre Kinder Robert und Pablo) in ihrer häuslichen Umgebung diskutiert [182]: Im Szenario 1 *Being Followed by Content* wird Maria von ihrer Lieblingsmusik geweckt, die ihr auf ihrem Weg durch das Haus folgt. Zur gleichen Zeit sieht Jerry die von ihm bevorzugten Nachrichten in einem anderen Raum. Als Maria beginnt ein eigenes Lied zu singen, beginnt das System dieses auch abzuspielen. Wenn sich Maria und Jerry in einem Raum treffen, stoppt das System. Trennen sie sich wieder in getrennte Räume, werden die bevorzugten Medien wieder für Beide abgespielt. Im Szenario 2 *Playing Games* ist das System in der Lage die Altersprüfung vorzunehmen und gegebenenfalls die Erlaubnis der Eltern einzuholen und ist in der Lage die Beleuchtung im Raum dem Inhalt des Spiels anzupassen. Das Spiel läuft dabei auf einem wandgroßen Bildschirm ab. Dabei interagieren die Spieler per Sprache und Gestik. Das dritte Szenario trägt den Namen *Home Caring*. Hier ist eine intelligente Tür in der Lage Familienmitglieder und Freunde zu erkennen und ihnen den Zutritt ins Haus zu erlauben. Gleichzeitig ist für Befugte von außen einsehbar, wer sich im Haus befindet. In der Küche ist es möglich Küchenrezepte einzusehen und den Aufenthaltsort der anderen Hausinsassen zu lokalisieren. Probleme bei technischen Geräten, z. B. der Waschmaschine, werden erkannt und angezeigt. Andere Geräte, wie der Geschirrspüler, werden situationsgerecht automatisch eingeschalten. Im Wohnzimmer wird, falls Filme angesehen werden sollen, die Beleuchtungsumgebung automatisch angesteuert. Das letzte Szenario *Sharing Ambiance* widmet sich der sozialen Interaktion. Sowohl Maria als auch ihr Vater können sich in getrennten Wohnungen gegenseitig sehen und auch miteinander sprechen. Mittels Befragungen und Diskussionen auf Basis der vier Anwendungsszenarios (unter insgesamt 55 Testpersonen) ergab sich eine Prioritätsliste, welche Eigenschaften ein Ambient-Intelligence-Haus ha-

ben sollte (für Details zur Evaluation und den Ergebnissen siehe [182]). Am höchsten priorisiert ist, dass der Benutzer sich im Besitz und Gefühl der Kontrollfähigkeit befinden will. Die potentiellen Bewohner wollen das System steuern können und nicht umgekehrt. An zweiter Stelle folgt die Entlastung von Information. Das System soll zu jeder Zeit an jedem Ort situationsgerecht die persönlich passenden Informationen zur Verfügung stellen können. Dritthöchst priorisiert sind Hausüberwachungsszenarios, die den Benutzer von der Aufgabe die technischen Geräte überwachen zu müssen entlasten können. Erst dann folgen Wunsche nach Haussicherheit (vor Eindringlingen von außen), Koordination von Haus und Büro, automatische Haussteuerung auf Basis von Kontextinformationen und die Befolgung von sozialen Regeln und die Ermöglichung von Awareness-Funktionen wie das Einblenden von Bildern und Ortsinformationen von Familienangehörigen.

2.1.12 ISTAG-Szenarios

Inspiriert von der Ambient-Intelligence-Idee entwickelte die IST Advisory Group (ISTAG) Szenarios, die den Menschen in die vorderste Reihe der technologischen Entwicklungen stellen (siehe [122]). Das erste Szenario – *Maria–Road Warrior* – beschreibt die nötigen Unterstützungen die Ambient Intelligence einer Geschäftsreisenden bieten kann. Hier kann Maria mit nur einem kleinen Gerät, einem sogenannten P-Com, statt mit einer Vielzahl an Geräten – wie Laptop, Mobiltelefone, PDAs, etc.) reisen. Das P-Com übernimmt hier eine Vielzahl an Aufgaben, die Maria sonst eine Menge an Zeit und Mühen gekostet hätten. Es funktioniert als Sicherheitsschlüssel am Flughafen, ist in der Lage den Zugang zu einem gemieteten Auto zu erlauben und öffnet die Türe im reservierten Hotel. Während der Fahrt benutzt Maria mittels des P-Com das Navigationssystem des Autos. Eventuell anfallende Mautgebühren werden ebenfalls über das P-Com geregelt. Nachdem Maria ihr Hotelzimmer betreten hat, passt sich das Zimmer ihren persönlichen Präferenzen gemäß an. Dies betrifft sowohl die Raumtemperatur, als auch die Beleuchtung und die Auswahl der Medien in Form präferierter Musik und Video. Maria kann aber auch mittels Sprachsteuerung die Lichter anpassen und sich im Badezimmer das Badewasser einlassen. Auch die Ansteuerung der Fernsehkanäle ist weiterhin manuell möglich. Vom Hotelzimmer aus hat sie vollen – sicheren – Zugriff auf ihre Daten, die auf dem Firmenserver gespeichert sind. Vom Hotelzimmer aus kann Maria ihre Präsentation für den nächsten Tag noch einmal überarbeiten. Telekommunikation steht im Zentrum des zweiten Szenarios – *Dimitrios–The Digital Me (D–Me)*. Von Ambient Intelligence unterstützte Telekommunikation soll hierbei bestehende Beziehungen zu Verwandten und Freunden befördern und ebenso dabei helfen neue Kontakte und Beziehungen zu knüpfen. Hierzu trägt Dimitrios in seiner Kleidung einen digitalen Avatar, der ihn repräsentiert. Der D–Me lernt ständig von Dimitrios Verhaltensweisen und Interaktionen und bietet Dimitrios Methoden für die Kommunikation, für Bedienungen und für Entscheidungen an. Durch das Lernen von Dimitrios Eigenschaften und Vorlieben ist der D-Me in der Lage, sich mit einer Vielzahl anderen D–Me's zu verbinden und Daten auszutauschen. Hierbei werden neue Kontakte geknüpft, Spiele initiiert und Interessen ausgetauscht. Aber das D–Me ist auch in der Lage in Notfällen nach Hilfe zu suchen. Dies kann ein direkter Hilferuf sein oder die Suche nach Informationen wie z. B. dem Weg zur Apotheke oder ein geeignetes

Medikament. Das dritte Szenario – *Carmen: traffic, sustainability & commerce* – stellt das Leben inmitten einer Ambient-Intelligence-Umgebung in einem intelligenten Haushalt in den Mittelpunkt der Handlungen. Ein solcher Haushalt verfügt über Spracheingabemöglichkeit und ist in der Lage externe Dienste zu suchen, nach persönlichen Präferenzen auszuwählen und automatisch zu beauftragen. Smarte Geräte im Haus sind in der Lage intelligente Dienste anzubieten. Der Kühlschrank listet fehlende Nahrungsmittel in Abhängigkeit bestimmter Kochrezepte auf und erlaubt die Vervollständigung einer Einkaufsliste. Auch ist er in der Lage die Einkaufsliste komplett an den Nahrungsmittellieferanten abzuschicken. Mittels eines persönlichen Gerätes – dem PAN–personal area network – kann Carmen jederzeit in einen Bus oder Zug einsteigen. Die Bezahlung geschieht automatisch und wegabhängig. Mit dem Gerät werden auch personalisierte Medien gemäß den persönlichen Präferenzen empfangen. Im letzten Szenario – *Annette and Solomon in the Ambient for Social Learning* – wird die Notwendigkeit des lebenslangen Lernens betont. Ambient Intelligence ist dabei in der Lage einen Raum so anzusteuern, dass Kommunikationen von Kleingruppen so möglich sind, dass andere nicht gestört werden. Ein Ambient-Intelligence-Gruppenraum ist auch in der Lage auf Basis der Anwesenheitsliste (die automatisch erzeugt wird) einen Tagesplan aufzustellen und die Geräte, wie Projektoren und Beleuchtung, situationsgerecht zu schalten. Der Raum weiß um die Erfahrung einzelner Teilnehmer und welche Erfahrung und welches Wissen für andere Teilnehmer von Nutzen sein könnten. Auf Basis erstellt der Raum die Tagespläne und sortiert die vorhandenen Dokumente.

2.1.13 DynAMITE

Im Projekt DYNAMITE [56] wurden zwei unterschiedliche Anwendungsszenarios als Basis für die weiteren Arbeiten ausgewählt: Das Wohnzimmer und der Hörsaal bzw. Besprechungsraum (siehe Abbildung 14 links). Das Szenario Wohnzimmer von DYNAMITE besteht aus 6 Anwendungsfällen: „1. Nach dem Arbeitstag kommt Herr Meier nach Hause. Bei sich trägt er seinen persönlichen Organizer (PDA), auf dem er in der Mittagspause die Urlaubsfotos eines Kollegen abgespeichert hat. Zuhause möchte Herr Meier die Urlaubsfotos seiner Frau zeigen. Im Wohnzimmer werden die Fotos direkt auf dem Fernseher angezeigt, während Herr Meier seinen PDA bedient, da der Fernseher über die besten Display-Eigenschaften aller Geräte im Raum verfügt. Außerdem führt er auf dem Handy gespeicherte Fotos auf dem Fernseher vor. Dazu bedient er lediglich den PDA und muss das Handy nicht aus seiner Jackentasche nehmen. 2. Nach dieser Diashow möchte Herr Meier Nachrichten ansehen, die über das Fernsehen ausgestrahlt werden. Mitten in einem Beitrag klingelt das Handy. Herr Meier nimmt das Telefongespräch an und die Nachrichten werden angehalten. Nach dem Telefonat wird die Wiedergabe nahtlos fortgesetzt. Um den Beitrag nochmals von vorne zu sehen, spult Herr Meier einige Sekunden zurück. 3. Anschließend entscheidet er sich, die gesamte Sendung permanent zu archivieren. Der Fernseher bietet ihm hierfür mehrere Möglichkeiten an: Die integrierte Festplatte des TV oder den DVD-Rekorder der Hifi-Anlage. Da die Nachrichtensendung zu groß für die Speicher des Handys und PDAs sind, werden diese Geräte nicht zur Auswahl angeboten. Er wählt den DVD-Brenner der Hifi-Anlage. 4. Herr Meier möchte sich jetzt über

Abbildung 14: DYNAMITE realisiert zwei unterschiedliche Anwendungsszenarios. Links dargestellt die Domäne Wohnzimmer (im Bild das Verkaufsstudio der Firma Loewe in Berlin), rechts dargestellt der große Hörsaal am Fraunhofer-Institut für Graphische Datenverarbeitung in Darmstadt.

Spielfilme informieren, die am Abend im Fernsehen kommen. Auf seinem PDA werden Informationen zu mehreren Filmen angeboten. Herr Meier fragt Informationen zum Film Rio Bravo ab, der ihn interessiert. Auf dem kleinen Bildschirm des PDAs werden dazu eine Reihe von Szenenbildern und Texten präsentiert zusammen mit Szenenausschnitten und Trailern, die an die niedrige Auflösung des PDAs angepasst sind. Da im Hintergrund das TV-Gerät läuft werden keine akustischen Ausgaben generiert. Herr Meier ist der Bildschirm des PDA allerdings zu klein und er leitet mittels Drag&Drop die Ausgabe auf den TV-Bildschirm um. Auf dem größeren Bildschirm können jetzt alle Informationen auf einmal angezeigt werden. Nun werden die Texte über eine Sprachausgabe akustisch wiedergegeben. Herr Meier schaltet zusätzlich einen 17 Zoll großen digitalen Bilderrahmen hinzu. So kann er jetzt jeweils zwei Bilder parallel anzeigen lassen und schnell die Bilder von Rio Bravo überfliegen. 5. Anschließend verlässt Herr Meier das Wohnzimmer. Frau Meier sieht sich die folgende Quizshow an. Nach einer Weile verlässt Sie mit dem Handy das Wohnzimmer. Die Sendung wird automatisch auf das Display des Handys umgelenkt. So kann Frau Mayer die Sendung auf der Terrasse weiter verfolgen. 6. Am späten Abend beschließt die ganze Familie, einen Spielfilm von DVD im Wohnzimmer zu sehen. Für die Bildwiedergabe wählt Herr Meier den Projektor aus. Der Ton wird automatisch über die Musikanlage ausgegeben.“ Herr Meier bewegt sich nicht nur in seinem eigenen Wohnzimmer. Während seiner Arbeit interagiert er mit anderen Umgebungen, so z. B. mit den Geräten in einem Vortragssaal (siehe Abbildung 14 rechts) oder in einem Besprechungsraum. Sein Arbeitsalltag besteht aus zwei komplexen Anwendungsfällen *Vortrag* und *Besprechung*. „1. Am Vormittag hat Herr Meier einen Vortrag in einem ihm unbekannten Hörsaal zu halten. Er setzt sich in die erste Reihe, und sein Laptop verbindet sich mit dem hausinternen W-LAN. Als Herr Meier mit seinem Vortrag an der Reihe ist, steht er auf und geht nach vorne. Sobald er das Vortragspult erreicht, schaltet sich das Licht auf Präsentationsambiente und der auf seinem Laptop gespeicherte Vortrag startet. Hierzu schaltet der Raum die Projektoren selbstständig. Zudem erkennt der Raum ein-

fallendes Sonnenlicht und verdunkelt die Rollos entsprechend, um im Zusammenspiel mit der Beleuchtungseinrichtung für optimale Ausleuchtung der Präsentationsfläche zu sorgen. 2. Nachdem alle Vortragenden ihre Ausführungen beendet haben, treffen sich einzelne Arbeitsgruppen. Sie treffen sich mitsamt ihren Laptops in einem Raum und haben von da an Zugriff auf projektrelevante Berichte und Protokolle. Befindet sich zudem noch ein Projektor im Raum, wird dieser automatisch zur Darstellung der Protokolle des gerade Sprechenden geschaltet.“ In DYNAMITE ist zusätzlich noch ein *Mobiles Szenario* definiert, das die Situationen Präsentation, Automatische Aufnahme und Zwischenfall in einem Anwendungsfall dynamischer Ensembles illustriert: „1. Nach der Arbeit besucht Herr Meier seinen Freund Herr Schneider. Zu Beginn des Besuchs führt Herr Meier die Urlaubsfotos von seinem PDA vor. Wie schon im Wohnzimmer von Herrn Meier (Szenario „Wohnzimmer“), werden auch hier die Fotos direkt auf dem Fernseher angezeigt, da der Fernseher von Herrn Schneider ebenfalls über ein besseres Display als die anderen vorhandenen Geräte (Laptop und PDA) verfügt. Da das Wohnzimmer von Herr Schneider über eine Beleuchtungssteuerung verfügt, wird das Licht automatisch gedimmt und Helligkeit und Kontrast des TV-Bildes automatisch angepasst. 2. Nach dem Ende des Films verlassen Herr Schneider und Herr Maier die Wohnung zu einem Restaurantbesuch. Vorher legt Herr Schneider noch eine DVD in das Laufwerk seines Laptops ein und startet die automatische Aufnahme eines Fußballspiels am TV. Das Geräteensemble nimmt die Sendung automatisch auf dem DVD-Rekorder auf, d.h. ohne eine Benutzerschnittstelle zur Auswahl des Aufnahmegerätes anzubieten. 3. Durch einen unvorhergesehen Zwischenfall (Verlängerung mit Elfmeterschießen) reicht der Speicherplatz auf der DVD nicht aus und das System findet automatisch eine Ersatzlösung – die Aufnahme auf der Festplatte des TV.“

2.2 Anforderungen an Software-Infrastrukturen für intelligente Umgebungen

Die von der ISTAG eingerichtete Arbeitsgruppe 8 *Grids, Distributed Systems and Software Architectures* [125] identifizierte, dass für die Realisierung der Ideen von Ambient Intelligence die „Middleware“ zwischen der reinen Netzwerkfunktionalität und den eigentlichen Applikationen die ausschlaggebende Komponente sein wird. Die Middleware realisiert diejenigen Funktionalitäten, die die Interoperabilität von mobilen und stationären Geräten garantieren aber auch für Funktionen der Sicherheit und Zuverlässigkeit zuständig sind. Die Arbeitsgruppe hat hierbei verschiedene Schwerpunkte identifiziert. Diese betreffen die Bereiche „Interoperabilität und Integration“, das „Management von komplexen Softwaresystemen“, „Semantik und Wissensverarbeitung“, „Werkzeuge für die Sofwareentwicklung, Methodiken und Standards“, „Vertrauen, Sicherheit und Datenschutz“, sowie die Fragen der „sozialen und kommerziellen Akzeptanz“. Vor allem der Veränderung von einer starren und ortsfesten Welt zu einer sich bewegenden verändernden Welt müsse hier Rechnung getragen werden. Es gibt keine zentrale Kontrolleinheit und der Zugriff auf Daten und Geräte ist allgegenwärtig. Vielmehr werden Systeme aus vielen Komponenten bestehen – seien es Recheneinheiten, Kommunikati-

onseinheiten oder Sensoren – die ad-hoc miteinander verbunden werden. Diese Verbindungen können auch nur für kurze Zeit bestehen oder periodisch sein. Ein Geräteensemble verhält sich somit in der Zukunft sehr dynamisch, einzelne Mitglieder schließen sich kurzzeitig einem bereits bestehenden Ensemble an, andere Mitglieder verlassen es wieder. Die Arbeitsgruppe sieht zwei miteinander verbundene Forschungsthemen in diesem Zusammenhang als sehr wichtig an. Diese sind eine offene Middleware-Infrastruktur um die Kommunikationsprozesse der beteiligten Komponenten zu bewerkstelligen sowie die Entwicklung von Semantiken um Dienste und Wissensverarbeitung von unterschiedlichen Gebieten und Domänen auszutauschen. Bei Michael H. Coen vom MIT Articial Intelligence Lab finden sich hierzu die Erfahrungen aus dem Intelligent-Room-Projekt, dass es zu vermeiden ist, dass ein monolithischer Kontroller zur Koordination aller Komponenten und Funktionen eingesetzt wird [43]. Coen selbst spricht hier von einem „big messy c program“. Ein solch monolithischer Kontroller als zentrale Funktionseinheit behindere das schnelle Hinzufügen von neuen Funktionen, das Modifizieren bereits vorhandener Funktionen sowie die manuelle Festlegung von Interaktionen und Handlungsprozessen. Die mit jeder neuen Funktion zunehmende Unübersichtlichkeit begünstige zudem das Aufkommen von Konflikten von bereits vorhandenen Funktionen und neuen Funktionen. Coen rät aus diesem Grund für intelligente Umgebungen zu einer Infrastruktur die ihre Funktionen in Schichten anbietet. Als weitere Anforderungen an Software-Infrastrukturen für intelligente Umgebungen definiert Coen [45], dass diese in Echtzeit lauffähig sein müssen, dass es möglich sein muss zur Laufzeit dynamisch neue Komponenten einem bereits bestehenden Komponentenensemble hinzuzufügen und auch Komponenten wieder zu entfernen und dass es – in einer logischen Folge davon – möglich sein muss, Komponenten auszutauschen. Eine offensichtliche Konsequenz dieser Anforderungen ist es, dass somit Komponenten verteilt (auf verschiedenen Recheneinheiten) und parallel lauffähig sein müssen. Weiterhin fordert Coen, dass ein System für kooperierende Geräte ein Ressourcen-Management vorhalten muss. Dieses müsse greifen, falls mehrere Komponenten dieselbe andere Komponente benutzen wollen und dieses hierbei nicht in der Lage ist, auf mehrere Anfragen gleichzeitig positiv zu reagieren (z. B: ein Display). Tatsächlich scheint diese Forderung aber der ursprünglichen Forderung auf eine zentrale Komponente zu verzichten zu widersprechen. Der Grund liegt darin, dass Coen Komponenten, die Dienste und Funktionen bereitstellen, als passive Geräte ansieht, deren Funktionen von außen aufgerufen werden. Die Frage nach einem Ressourcen-Management in der von Coen geforderten Art adressiert vielmehr die Frage nach der Autonomie der einzelnen Geräte und der Selbstorganisation von Geräteensembles. Nicht ein zentrales Ressourcen-Management muss die Frage nach der Zuteilung von Aufgaben und Ressourcen übernehmen, sondern die Geräte haben mit gemeinsamen Strategien diese Aufgabe gemeinsam (und damit verteilt) zu übernehmen. Die Forderung nach der Erweiterbarkeit von Geräteensembles findet sich konkretisiert bei Brad Johanson von der Stanford Universität aus den Erfahrungen im Interactive-Workspaces-Projekt in der Formulierung, dass „users should only have to plug in a device or bring it into a physical space for it to become part of the corresponding software infrastructure. User configuration should be simple and prompted by the space. . . . The logical extension of this is to allow ad hoc inter-

active workspaces to form wherever a group of devices are gathered.“ (aus [130]). Diese Aussage kann kombiniert werden mit der Vision von Mark Weiser, dass Umgebungen die ausgestattet sind mit vielen Geräten sich immer dynamisch – sowohl auf der Softwareseite, als auch auf der Hardwareseite – verhalten werden und daher niemals ad-hoc abgeschaltet und wieder hochgefahren werden können (da man das sonst immer tun müsste). Marc Weiser formuliert dies treffend mit dem Satz (vgl. [213]), dass „...new software for new devices may be needed at any time, and you'll never be able to shut off everything in the room at once...“. Geräteensembles sind somit durch einfaches Verbinden zu erzeugen, neue Geräte sind dabei in der Lage einem bereits bestehenden Geräteensembles beizutreten. Ebenso einfach muss das Entfernen von Geräten aus deinem Ensemble heraus sein[24]. Dies alles muss in Echtzeit in einem weiter laufenden System geschehen können. Diese Forderung wiegt umso schwerer, wenn man sich den Benutzer in einer fremden Umgebung vorstellt (in der er die Geräte, die er ein- und ausschalten müsste, keinesfalls kennen kann) oder er sich zusätzlich noch in einer Umgebung befindet, die auch über versteckte eingebettete Geräte (die eigentliche Idee des Ubiquitous Computing von Mark Weiser) verfügt. Sollte dies realisierbar sein, sehen Gregory D. Abowd und Elizabeth Mynatt vom Georgia Institute of Technology und Tom Rodden von der University of Nottingham die folgende weitere Herausforderung (vgl. [7]): „The challenge is to coordinate across many output locations and modalities without overwhelming our limited attention spans.“ Weiter sagen sie: „Negotiation and resolution strategies must integrate information from competing context services when more than one service concurrently provides the same piece of context“. Die Autoren sehen, dass im Falle dynamischer Geräteensembles leicht die Situation auftreten kann, dass es konkurrierende Geräte geben wird. Und nicht nur, wie Coen meint, um das eine Gerät welches eine bestimmte Funktion anbietet, sondern auch konkurrierende Situationen von Geräten die gleiche oder ähnliche Dienstleitungen anbieten. Hierzu müssen geeignete Koordinierungsmechanismen oder Konfliktlösungsstrategien bereitgestellt werden. Barry Brumitt von Microsoft hat hierzu im Bezug auf das Easy-Living-Projekt konkrete Ideen veröffentlicht (siehe [33]): „This screen is selected, because it is available and in Tom's field of view.“. Demnach besitzen Geräte nicht nur messbare Eigenschaften wie Dienstleistungen, sondern können je nach Kontext mal mehr oder mal weniger für die Erbringung einer Dienstleistung geeignet sein. Hier wird vorgeschlagen, das Display zu nehmen, welches am idealsten zu Toms Blickfeld orientiert ist. Im Projekt DYNAMITE werden ähnliche Lösungen präferiert. In den Szenarios findet sich, dass das Display eines Fernsehers dem Bildschirm eines PDAs aufgrund seiner Größe für die Ansicht von Urlaubsfotos vorgezogen wird. In EMBASSI wird für die Musikausgabe selbstverständlich die Ausgabe über die Hifi-Anlage vorgezogen, obwohl auch andere Lautsprecher verfügbar wären. Gregory Abowd et al. ziehen aus der Analyse ihrer Szenarios den wichtigen Schluss, dass in intelligenten Umgebungen sich die Mensch-Technik-Interaktion dramatisch verändern wird(vgl. [7]). Weg von der herkömmlichen Maus- und Tastaturbedienung wird sie sich mehr und mehr zu natürlichsprachiger Interaktion mit der umgebenden Welt verschieben. Nicht mehr nur die explizite Interaktion

[24]Häufig wird die Eigenschaft, dass Geräte sich gewissermaßen frei bewegen können auch als nomadisch bezeichnet.

durch den Benutzer, sondern auch die impliziten Aktionen auf die die intelligente Umgebung passend reagieren muss, werden die Anforderungen an die Software-Infrastruktur und die dabei beteiligten Komponenten und Geräte stellen. Ist das realisiert werden sich die Erfahrungen und Erwartungen der Benutzer in Bezug auf Eingaben und Ausgaben von Systemereignissen drastisch ändern.

Auch bei einer näheren Analyse der Szenarios findet sich dieser Mix von expliziten und impliziten Interaktionen. In einigen Anwendungsbeispielen findet zudem keinerlei Interaktion statt, sondern die Handlung der intelligenten Umgebung wird durch eine Kontextänderung (z. B. Zeit, Temperatur) hervorgerufen. Im EMBASSI-Projekt wird der Benutzer mittels seiner Lieblingsmusik geweckt. Hierbei wird die Musik entsprechend des Benutzerprofils ausgewählt. Explizite Interaktionen sind möglich durch Spracherkennung zur direkten Befehlseingabe oder durch Kombination von Spracheingabe und Gestik (EMBASSI) oder von Spracheingabe und graphischen Benutzeroberfläche (SmartKom). Informationen von Positionssensoren veranlassen eine Änderung des Verhaltens der Umgebung (EMBASSI, DYNAMITE, Oxygen, ISTAG-Szenarios). Die Betrachtung der Szenarios zeigt, dass es eine nicht bestimmbare Menge an Möglichkeiten zur expliziten und impliziten Interaktion mit dem Gesamtsystem geben kann. Diese veranlassen in ihrer Interpretation das System zu Verhaltensänderung oder zum Aufrufen von Funktionen. Die Szenarios beschreiben auch eindeutig, dass ein bestehendes Geräteensemble erweiterbar sein muss (z. B. wird in EMBASSI eine neue Audioanlage angeschlossen) und das System in der weiteren Laufzeit dieses Gerät in die folgenden Handlungen mit einbeziehen muss. Grundsätzlich ist immer ein Gerät für Aufgaben auszuwählen, welches der Situation gemäß am besten geeignet ist. So wird in EMBASSI das beste Ausgabegerät für die Musikwiedergabe verwendet, sowie in DYNAMITE das am besten geeignete Display für die Wiedergabe von Urlaubsfotos verwendet. Auch die Ausgabe von Information an den Benutzer geschieht in Abhängigkeit der dafür vorhandenen Ausgabegeräte (z. B. in DYNAMITE). Im Interactive-Workspaces-Projekt werden hier die Benutzerschnittstellen an die vorhandenen verschiedenen Displays angepasst, sowie das Vorhandensein und die Kombination von verschiedenartigen Ein- und Ausgabemodalitäten (z. B. Sprache) berücksichtigt. In den analysierten Szenarios lassen sich unterschiedliche Arten an Komponenten identifizieren. Sensoren für implizite Interaktion und Komponenten für explizite Interaktion werden in jedem Szenario genannt. Einige (z. B. SmartKom, Interactive Workspaces) nennen zudem die multimodale Ausgabe an den Benutzer als eine wichtige Klasse an Komponenten. Die Geräte, wie Fernsehapparate, Hifi-Anlagen, Beleuchtungseinrichtungen, Displays oder Projektoren können als eine andere Klasse an Komponenten (die Aktuatoren) identifiziert werden. In der Verbindung dazwischen lässt sich eine Klasse an Komponenten identifizieren, die man als intelligente Komponenten bezeichnen kann (siehe die Diskussion des *Intelligent Classrooms*). Sie sind in der Lage, die impliziten und expliziten Nutzeräußerungen sowie andere Kontextinformationen zu interpretieren, Systemausgaben zu veranlassen und die Aktuatoren so zu manipulieren, dass ihre Funktionsausführung den interpretierten Zielen am nächsten kommt. Der Hinweis, dass eine Software-Infrastruktur diesen unterschiedlichen Gruppen an Komponenten Rechnung tragen sollte, findet sich bei Robert Grimm von der University of Washington (vgl. [91]),

der hier von einem „Amalgam“ von autonomen und unabhängigen Domänen spricht. Es scheint eine offensichtliche Ableitung dieser Beobachtung zu sein, wenn auch eine Software-Infrastruktur dieser Tatsache gerecht wird, indem diese Gruppen an Komponenten getrennt voneinander behandelt werden. Grimm betont dies, indem er fordert, dass „our architecture needs a way to group computations and data, to change such groupings, to support interaction between different groupings“. Eine Software-Infrastruktur benötigt somit Mechanismen die Gruppierung von Komponenten gleichartiger Domänen zu unterstützen, deren Kommunikation zu befördern und die Kommunikation von Gruppe zu Gruppe möglich zu machen. Bei Grimm (siehe [91]) findet sich auch eine Forderung die den Erfahrungen entspricht, die im Rahmen der Erstellung dieser Arbeit in den Projekten EMBASSI und DYNAMITE gemacht wurden, sich jedoch in der Literatur – vielleicht aus scheinbaren Trivialitätsgrunden – selten findet: Die Software-Infrastruktur zur Realisierung von verteilten Komponentenensembles sollte von Dritten – i.A. sind dies Software-Entwickler und Programmierer – einfach in eigenen Projekten einsetzbar sein[25]. Sehr treffend (und wertfrei) pointiert dies Grimm in einer weiteren Veröffentlichung mit den Worten (vgl. [92]): „application developers can focus on the actual application logic and on making their applications adaptable“. Eine Software-Infrastruktur muss es Entwicklern ermöglichen, „to design, develop and deploy application“, und dies auf möglichst einfache Art und Weise.

2.3 Definition der Selbstorganisation von Ensembles

Die folgenden Punkte fassen die gesammelten Anforderungen an eine Software-Infrastruktur für heterogene intelligente Umgebungen zusammen. In ihrer Gesamtheit definieren sie die Möglichkeit zur Selbstorganisation von Komponenten- und Geräteensembles (vgl. [110]):

Erweiterbarkeit: Ein bereits bestehendes Ensemble an Geräten und/oder Komponenten muss zu jeder Zeit durch neue Geräte und/oder Komponenten erweiterbar sein. Die neuen Geräte müssen sich nahtlos in die vorhandene Kommunikationsstruktur des bereits bestehenden Ensembles einfügen können. Dabei darf es nicht nötig sein, Geräte die bereits vorhanden sind, für die Dauer der Konnektierung des neuen Gerätes abschalten zu müssen. Dies muss auch in dem Falle gelten, dass ein Gerät aus einem bestehenden Ensemble entfernt wird. Auch dies muss zur Laufzeit des Geräteensembles ohne eine Veränderung bei anderen Geräten vornehmen zu müssen, geschehen können.

Austauschbarkeit: Aus der Forderung der Erweiterbarkeit eines bereits bestehenden Geräteensembles durch neue Geräte und/oder Komponenten und der Entfernbar-

[25] Projekte mit verteilten Komponentenensembles, in denen unterschiedliche Projektpartner für die Realisierung verschiedener Komponenten verantwortlich sind, zeigen deutlich, dass viel Aufwand und Kompetenz in die Realisierung der Funktionalitäten der eigenen Komponenten investiert wird. Ressourcen um die gemeinsame Infrastruktur zu konzipieren oder komplizierte Integrationsvorgänge zu realisieren sind dann zumeist nicht mehr vorhanden.

keit von Geräten und/oder Komponenten ergibt sich trivialerweise die Forderung, dass Geräte nahtlos austauschbar sein müssen.

Autonomie: Stellt man sich ein Komponentenensemble vor als eine Ansammlung an Geräten die ad-hoc miteinander verbunden wurden, so kann offensichtlich kein Gerät Anforderungen an die Anwesenheit anderer Geräte stellen. Ein Gerät muss somit intern alle nötigen Funktionen mitbringen, um auch autonom funktionsfähig zu sein. Vor allem für die Kommunikation mit anderen Geräten dürfen keine externen Voraussetzungen bestehen.

Dezentralität: Soll ein Geräteensemble jederzeit dynamisch erweiterbar sein, so schließt dies die Möglichkeit des Vorhandenseins einer zentralen – ausgezeichneten – Komponenten aus. Trivialerweise müsste diese Komponente immer die Basis eines dann erweiterbaren Ensembles bilden. Ebenso dürfte diese zentrale Komponente niemals ein sich noch in der Laufzeit befindliches Ensemble verlassen. Die Forderung nach Dezentralität und verteilter Implementierung ist somit unbedingte Voraussetzung für die Bildung von dynamischen ad-hoc Geräteensembles.

Konfliktlösung: Während der Laufzeit eines Geräteensembles kann es zu Konflikten konkurrierender Geräte/Komponenten kommen. Geräte gleichartiger Funktionalität konkurrieren um Aufträge oder um Ressourcen. Für Konflikte passende Mechanismen und Strategien, sog. Konfliktlösungsstrategien, auf der Ebene der Software-Infrastruktur zur Verfügung zu stellen bzw. die Möglichkeit zu deren Implementierung anzubieten ist aus Gründen der Transparenz innerhalb eines Ensembles unabdingbar. Konflikte können hierbei nicht an einer zentralen Stelle (siehe Forderung nach Dezentralität) oder in den Komponenten selber (siehe Forderung Benutzbarkeit und Austauschbarkeit) bearbeitet werden.

Benutzbarkeit: Um die Realisierung komplexer Anwendungen auf der Basis verteilter Komponentenensembles zu ermöglichen, muss die Benutzung einer Software-Infrastruktur auch im Rahmen eines Projektes mit mehreren Partnern möglich sein. Dies wird ermöglicht durch logisch aufgebaute Programmierschnittstellen sowie durch das Vermeiden von zusätzlichen Belastungen für den Software-Entwickler. Hierzu gehört vor allem die Entlastung von Entscheidungsprozessen, die sich aufgrund der verteilten Entwicklung ergeben (siehe Konfliktlösung).

Echtzeit: Intelligente Umgebungen sollen in Abhängigkeit des aktuellen Kontextes, sowie in Abhängigkeit der expliziten und impliziten Interaktion des Benutzers in der Lage sein die Geräte der Umgebung in einer zielgerichteten und situationsgemäßen Art und Weise anzusteuern. Die Forderung nach Echtzeit ergibt sich hier offensichtlich. Eine Software-Infrastruktur zur Realisierung intelligenter Umgebungen sollte die Komponenten nicht zusätzlich rechenintensiv belasten und die Kommunikation der Komponenten in reeller Zeit ermöglichen.

Im folgenden Abschnitt werden die in diesem Kapitel aufgestellten Anforderungen mit den gängigsten und bekanntesten Lösungen für Software-Infrastrukturen verglichen.

3 Software-Infrastrukturen für verteilte Applikationen

Zur Realisierung von intelligenten Umgebungen und zur Sicherstellung der Kommunikation von verteilten Komponenten sind in der Vergangenheit in unterschiedlichen Projekten verschiedene Software-Infrastrukturen entstanden. Dieses Kapitel beschreibt und analysiert Software-Infrastrukturen und Middleware-Lösungen, die aufgrund ihrer Verbreitung, ihres Einsatzes in bekannten Projekten oder ihres generellen Bekanntheitsgrades eine Bedeutung erlangt haben. Unter ihnen sind bekannte Technologien wie die *Open Agent Architecture*, die seit mehr als 7 Jahren eingesetzt und weiterentwickelt wird oder die Resultate des *AMIGO*-Projektes, welches im europäischen Raum am aktuellsten an Bedeutung gewinnt. Auch Spezifikationen von semantischen Agentenkommunikationssprachen, wie der *KQML*-Initiative und die Arbeiten der *FIPA*-Organisation, werden hier besprochen. Diese waren die ersten Initiativen, die die bis dahin dominierenden Lösungen auf Basis von Methodenaufrufen entfernter Objekte (remote method invocations) durch asynchrone Kommunikation auf der Basis ausgetauschter Nachrichten ersetzt haben. Einige Software-Infrastrukturen zeichnen sich weniger durch ihre hohe Verbreitung als vielmehr durch interessante Innovationen und Lösungen für spezielle Anwendungsbereiche und die sich daraus ergebenden Konflikte aus. Hier werden als Beispiele die *Jaspis*-Infrastruktur oder die Lösungsansatze im *DIANE*- und im *3PC*-Projekt detailliert beschrieben. Neben Software-Infrastrukturen, die die Kommunikation zwischen verteilten Komponenten auf semantischer Ebene bewerkstelligen, werden auch die bekanntesten Middleware-Technologien (wie Universal Plug and Play, Jini oder HAVi) besprochen. Die Aufgabe solcher Middleware-Lösungen ist die Vernetzung heterogener Geräte und die Bereitstellung einer einheitlichen Schnittstelle zur Bearbeitung von Nachrichten und der Zugriffsmöglichkeiten auf angebotene Dienste und Funktionen.

3.1 AMIGO

Das EU-IST-Projekt Amigo [11] hat sich zum Ziel gesetzt eine offene, standardisierte und interoperable Middleware zu entwickeln. Zusätzlich sollen im Bereich Haushalt und Sicherheit, Hausinformation und Unterhaltung Demonstratoren realisiert werden (siehe auch Kapitel 2.1.11). Die Middleware in Amigo soll hierzu die Interoperabilität von heterogenen Geräten und Applikationen sicherstellen und deren Dienste innerhalb der vernetzten Umgebung durch die Nutzung von Standard-Technologien verfügbar machen. Das Ziel der Amigo-Architektur (siehe Abbildung 15 und [14] S.80f) ist es die Komposition von abstrakt beschriebenen Diensten möglich zu machen. Diese sollen hierbei dynamisch eingebunden werden können. Die wichtigste Eigenschaft hier ist die Unterstützung von Heterogenität. Dies wird in Amigo durch einen zweistufigen Ansatz erreicht: Im ersten Schritt wird nach semantischer Übereinstimmung gesucht. Dies führt zu einer Auswahl an Diensten die für die weitere Integration im zweiten Schritt ausgewählt werden können. Amigo verfolgt hier keinen agenten-basierten Ansatz im herkömmlichen Sinne, sondern stellt eine dienste-orientierte Middleware (engl. service-oriented architecture (SOA)) dar. Amigo zielt weniger darauf ab, eine innovative Software-Infrastruktur zu erstellen, son-

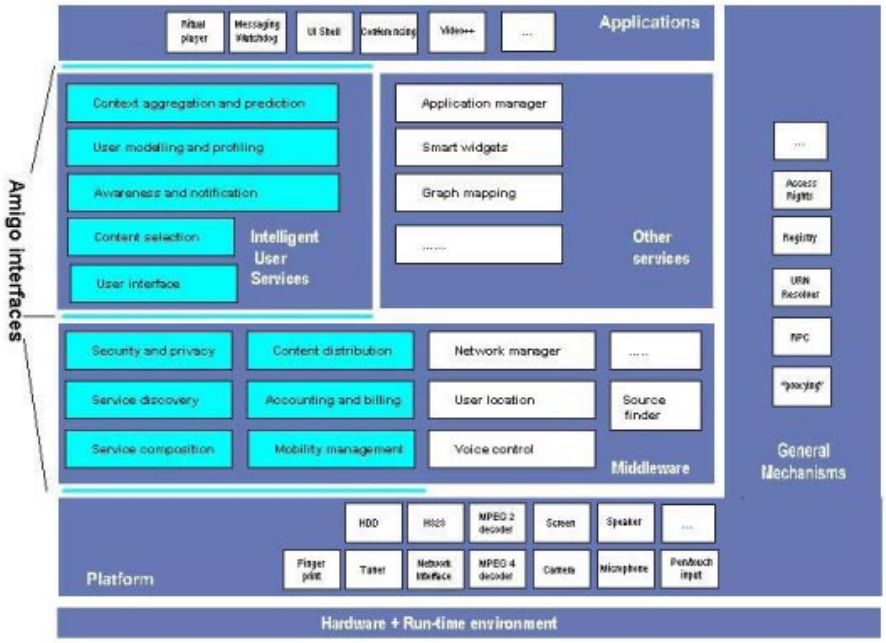


Abbildung 15: Die interne Architektur eines Amigo-Gerätes (aus [16]).

dern will so viele bekannte Gerätetechnologien wie möglich integrierbar machen (siehe auch Abbildung 16 rechts). So soll Amigo als service-oriented middleware andere gleichartige Architekturen wie UPnP, OSGi oder WebServices einbinden. Gleiches gilt für die unterstützten Protokolle. Auch hier sollen so viele wie möglich berücksichtigt werden (wie z. B. SLP, Jini oder SSDP als service discovery protocols). Hier wird Wert auf Interoperabilität dieser Dienste und Applikationen gelegt, um so heterogene Geräte und heterogene Netzwerke miteinander verbinden zu können (vgl. [12] S.80f). Die *Ambience Reference Architecture* (siehe Abbildung 15) definiert hierbei die nötigen Funktionalitäten pro Gerät bzw. pro Komponente. Unter anderem sind hier Funktionen für Sicherheit und Vertraulichkeit, für Dienstefindung (*Service Discovery*) und für Dienstekomposition (*Service Composition*). Aber auch interne Funktionen für eventuelle Buchungs- und Rechnungsprozesse (*Accounting and Billing*) sind vorgesehen[26]. Jede Komponente bzw. jedes Gerät ist somit für die Realisierung von funktionalen Methoden für die Ausführung von Service Discovery, Service Composition und die Implementation für Methoden zur Sicherstellung von Interoperabilität verantwortlich. Konflikte können auftreten im Falle der Mehrdeutigkeit der Service-Discovery-Funktion (im Falle konkurrierender Serviceanbieter) oder wenn ein Dienst aus mehreren Teildiensten zusammengestellt werden soll. Für die Ausführung dieser Funktion ist der *Enhanced Service Discovery* definiert (siehe Ab-

[26]Die in Abbildung 15 illustrierte interne Architektur eines Amigo-Gerätes weist darüber hinaus starke Ähnlichkeiten mit der *Ambience Reference Architecture* auf, die im ITEA-Projekt Ambience (http://www.hitech-projects.com/euprojects/ambience/) definiert wurde. Die Laufzeit dieses Projektes war von 2001 bis 2003.

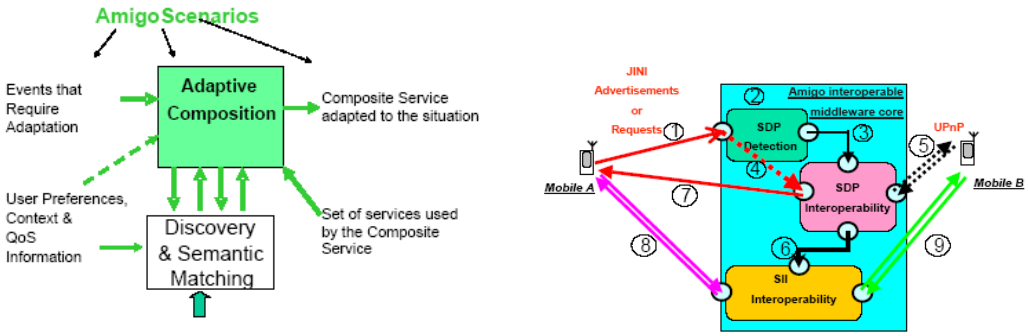


Abbildung 16: Die prinzipielle Vorgehensweise der adaptiven Service Composition in Amgio (links aus [15] S. 44ff) und die Illustration der Interoperabilität des Kerns der Amigo-Middleware (rechts aus [13]).

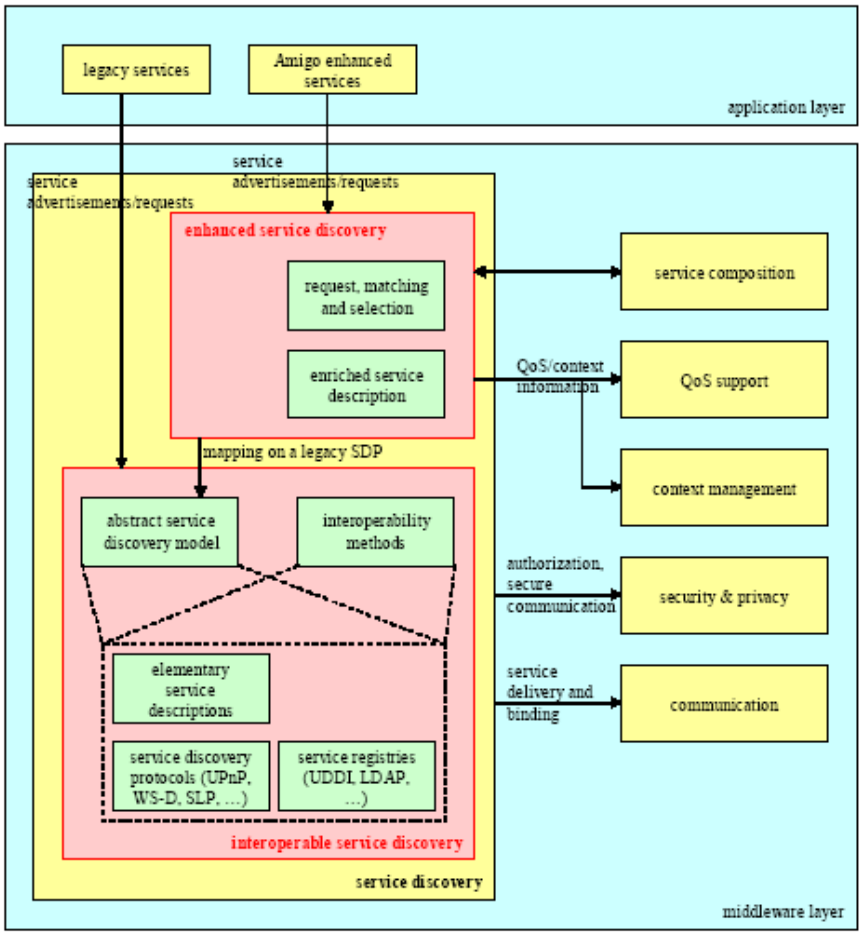


Abbildung 17: Die abstrakte Service-Discovery-Architektur eines Amigo Gerätes (aus [14] S.61f).

bildung 17). Dieser muss in der Lage sein (vgl. [14] S.63f) die angebotenen Dienste von mehreren Diensteanbietern miteinander vergleichen zu können und – wenn möglich – den am besten geeigneten Diensteanbieter auszuwählen. Das sog. *Service Binding* muss somit von jeder Komponente, die nach Diensten fragt, selbst ausgeführt werden können. Genaue Algorithmen hierfür werden nicht spezifiziert. In Anlehnung an bekannte Herangehensweisen des Service-Retrievals werden aber vier unterschiedliche Ansätze zum Auffinden geeigneter Diensteanbieter (nach Klein und Bernstein [139]) in Amigo diskutiert. Dies ist zum einen der schlüsselwort-basierte Ansatz, indem nach bestimmten Teilen in der Anfrage gesucht wird. Dieser wird jedoch aufgrund seiner Ungenauigkeiten nicht präferiert. Ein tabellen-basierter Ansatz – wie er auch in Jini (vgl. Kapitel 3.12.1) verwendet wird – besteht aus Attribut-Wert-Paaren, die die typischen Diensteeigenschaften beinhalten. Typischerweise sind dies der Name, die Beschreibung, sowie die Eingabe- und Ausgabeparameter und einige Angaben bezüglich der Leistungsfähigkeit eines Dienstes. Der konzept-basierte Ansatz versorgt die anfragende Komponente mit den nötigen Ontologien, die ein Service-Discovery über die Syntax hinaus mit den nötigen Semantiken möglich machen. Dieser Ansatz weist eine stark erhöhte Präzision im Vergleich zum schlüsselwort-basierten Ansatz auf. Die vierte Herangehensweise, die in Amigo diskutiert wird, ist der deduktive Ansatz, der Dienstanfragen mit formeller Semantik ausdrückt. In Amigo wird der tabellen-basierte Ansatz präferiert (vgl. [14] S.75ff). Hierbei werden bei den Ein- und Ausgabeparametern vier verschiedene Vergleichsalgorithmen unterschieden: exakt („exact“), austauschbar („plug in“ im Sinne, dass ein Parameter einen anderen ersetzen kann), nicht vollständig austauschbar („subsumes“ im Sinne, dass ein Parameter einen anderen nicht vollständig ersetzen kann) und ausgefallen („fail“).

Als dienste-orientierte Architektur definiert Amigo jede Komponente als einen Dienst und die Verbindungen zwischen den einzelnen Diensteanbietern als das Interaktionsprotokol dazwischen. Die Definition von Konfliktlösungsstrategien und Algorithmen für die Komposition von Diensten hat nicht die höchste Priorität in Amigo. Stattdessen sollen die verschiedensten Standards mittels einem integrierten Service-Discovery-Protocols (Abbildung 16 rechts) vereinigt werden. Diese Heterogenität der Geräte zu integrieren ist Ziel der Arbeiten bezüglich der Interoperabilität. Dies ist das erste Prinzip der Funktionsweise der Amigo-Architektur. Im zweiten Schritt wird es ermöglicht, die heterogenen Domänen und die damit verbundenen Technologien geeignet zu repräsentieren und dann im letzten Schritt erst geeignete Interoperabilitätsmethoden zu definieren. Auf diese Art soll das Amigo-System die dynamische Integration, das Auffinden und die Komposition von Diensten unterstützen. Amigo definiert zusätzlich drei unterschiedliche Möglichkeiten der Publikation von Diensten durch Diensteanbieter: Publikation von Diensten mittels einer zentralen Registry oder eines zentralen Repositories, mittels eines Indexes oder durch peer-to-peer-Kommunikation, indem Diensteanbietern anderen Komponenten direkt ihre Dienste mitteilen. Somit konzentrieren sich die Arbeiten bzgl. der Interoperabilität auf die drei Basisvoraussetzung: Publikation von Diensten, und die Anfrage und das Auffinden von Diensten (und Diensteanbietern).

3.2 Super Distributed Objects

Die Initiative *Super Distributed Objects* einer Gruppe des OMG[27] – mit den Partnern Universität Tokia, Hitachi Ltd, Fraunhofer Fokus, NTT Japan, dem Computer Science Laboratory Lille und der University of California – verfolgte die Idee, dass in einer standardisierten Computer-Infrastruktur die Entitäten der realen Welt als Objekte modelliert und diese dann in hoch-verteilten Umgebungen eingesetzt werden können [186]. Diese Objekte sind dann in der Lage miteinander zu kooperieren und die Benutzer in der Erledigung ihrer Ziele zu unterstützen. *Verteilt* meint hierbei eine sehr hohe Anzahl an Objekten, die ohne jede Art an zentraler Instanz autonom miteinander kommunizieren und kooperieren können. Super Distributed Objects (SDOs) sind hierbei definiert als die logische Repräsentation einer Hardware oder Software die eine genau bestimmte Funktionalität anbietet. Ein SDO ist hierbei nicht auf die Abstraktion einer physikalischen Hardware beschränkt. Auch bestimmte WebServices oder Inhalte können SDOs sein. Eine besondere Klasse an Objekten sind die Service-Logic-Objekte (SL), die definieren können Wiedurch, Wann und Wie ein oder mehrere SDO-Objekte auf gegebene Situationen reagieren. Ein solches Objekt kann für jedes SDO definiert sein (siehe Abbildung 18 links) oder auch für eine Vielzahl an SDOs (siehe Abbildung 18 rechts). Die Interaktion von mehreren SDOs untereinander oder SDOs und SL-Objekten miteinander definiert einen Application-Service. Ein solcher Application-Service lässt sich im Allgemeinen als ein Anwendungsszenario beschreiben. Gemäß den Voraussetzungen ist ein Application-Service auf Basis einer Vielzahl an SDOs dezentralisiert. Es gibt keine ausgezeichnete Komponente, die für die Kontrolle, die Koordination und Kommunikation der einzelnen SDOs zuständig ist. Die Empfehlungen der OMG-Arbeitsgruppe lassen den Punkt der Dezentralisierung jedoch den jeweiligen Entwicklern offen. Abbildung 18 macht deutlich, dass die Implementierungsentscheidung für ein Service-Logic-Objekt ausserhalb eines SDOs einem zentralen Ansatz entsprechen würde. Der Einsatz von Service-Logic-Objekten außerhalb eines SDOs beschränkt hierbei jedoch automatisch die Autonomie der von diesem Service-Logic-Object manipulierten SDOs. In [186] wird die Problematik des Service Discovery in dynamischen Umgebungen angesprochen. Es ist jedoch kein Algorithmus oder Vorgehensweise definiert, wie ein SDO einen Dienst suchen kann bzw. im Falle mehrerer Diensteanbieter einen solchen Dienst auswählen und bestimmen kann. Hier ist auf die Methode der (zentralen) Service-Logic-Objekte zurückzugreifen. Die SDO-Referenz-Architektur (siehe Abbildung 19) definiert hier genau ein Service-Logic-Objekt (mit Zugriff auf wichtige Daten wie Benutzerprofile etc.) das in einer dynamischen SDO-Umgebung gemäß den Benutzerinteraktionen die von den verfügbaren SDOs angebotenen Funktionen kombiniert. In [220] wird eine prototypische Implementation beschrieben. Der *I-centric*-Ansatz (vgl. [176]) des Fraunhofer-Fokus erweitert diesen Ansatz der SDOs und stellt Dienste von Geräten und Umgebungen nicht nur auf Basis der verfügbaren Technologien zusammen, sondern besonders auf Basis der individuellen Präferenzen der Benutzer (siehe Abbildung 20 links). Die *I-Centric*-Kommunikations-

[27] Genauer: der Object Management Group: Super Distributed Objects Domain Special Interest Group (SDO DSIG)

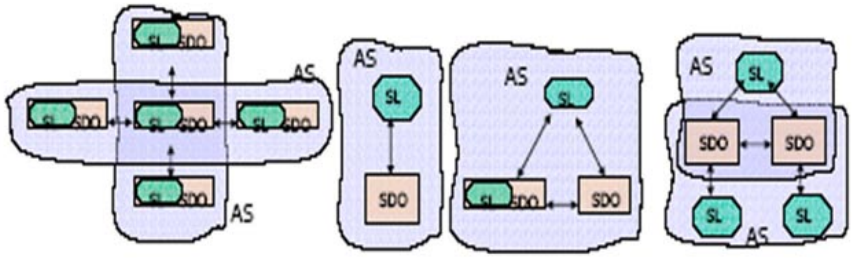


Abbildung 18: Illustration (aus [186]) verschiedener Kombinationsmöglichkeiten von SDOs, von Service-Logic-Objekten (SL) und Application-Service-Objekten (AS).

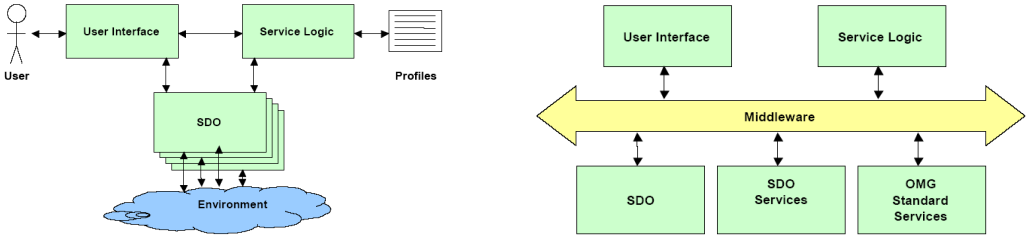


Abbildung 19: Links die wichtigsten funktionalen Komponenten, rechts die funktionale Struktur der SDO-Referenz-Architektur (aus [186]).

Architektur (rechtes Bild in Abbildung 20) illustriert die Vorgehensweise. Sensor- und Kontextdaten, die von speziellen Sensor- und Kontrollgeräten der Umgebung erfasst werden, werden von den sie repräsentierenden SDO-Objekten in einem Kontext-Server erfasst. Ähnliche SDOs sind im Übrigen für die Manipulation der Umgebung verantwortlich. Eine spezielle Komponente für die Kontextinterpretation ist für die sinnvolle Interpretation und Aufbereitung dieser Kontextdaten verantwortlich. Somit kann die *I-Centric-Service*-Komponente diese angereicherten Kontextdaten (in Kombination mit persönlichen Präferenzen) zur Bestimmung von situationsgerechten Funktionsaufrufen auf Basis der verfügbaren SDOs verwenden. Um die Dynamik einer SDO-Umgebung zu garantieren werden in [176] die Eigenschaften der SDOs weiter konkretisiert. Ein SDO ist in der Lage die verfügbaren Dienste einer bereits vorhandenen SDO-Umgebung zu entdecken (Service Discovery) und die Zustände anderer SDOs zu beobachten. Hierzu bietet jedes SDO spezifische Schnittstellen an. Arbanowski et al. beschreiben in [22] eine Anwendung auf Basis des *I-Centric*-Ansatzes. Zwei Benutzer sind mit je einem IR-Badge zur Identifikation ausgestattet und auf zwei Rechnern werden die Infrastrukturen von je einem Raum mit Klimaanlage, Jukebox, digitalem Bilderrahmen, Beleuchtung und Dimmer simuliert. „Betreten“ nun die Benutzer einen Raum, so wird dieser gemäß den individuellen Präferenzen gesteuert. Im Falle dass der zweite Benutzer den Raum betritt, in dem der erste Benutzer schon ist, so werden die Präferenzen des später Hereingekommenen nur dann ausgeführt, wenn diese nicht im Widerspruch zu den Präferenzen des schon Anwesenden

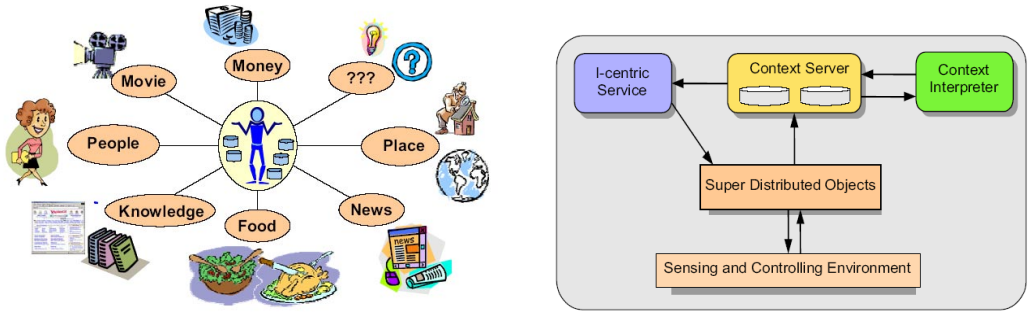


Abbildung 20: Links die Illustration der Vision des *I-Centric*-Ansatzes und die *I-Centric*-Kommunikations-Architektur (aus [176]).

sind. Verlässt ein Benutzer jedoch den Raum, so werden sofort die präferierten Einstellungen des noch verbleibenden Benutzers aktiv.

3.3 Oxygen

Oxygen ist kein Projekt des Massachusetts Institute of Technology (MIT) im eigentlichen Sinne. Es laufen hier vielmehr verschiedene Forschungsaspekte und Arbeitsgruppen des MITs zusammen. Gemeinsam entwickelte Szenarios stellen die Klammer dar, um verschiedene Technologien für *Pervasive Computing* zu entwickeln und zu erproben [223]. Eine Gesamtintegration ist bis zu diesem Zeitpunkt nicht absehbar. Oxygen hat sehr wichtige Beiträge im Bereich der Definition von *Ambient Intelligence* und *Pervasive Computing* geliefert. Wichtige Schlüsselwörter wie *embedded*, *nomadic* oder *adaptable* sind hier definiert und ausformuliert worden. Im Rahmen der Oxygen-Initiative des MIT werden neben Anwendungsszenarios (siehe das Intelligent-Room-Projekt in Kapitel 2.1.7) in anderen Teilprojekten auch unterschiedliche Basistechnologien erforscht und entwickelt. Ein Beispiel ist die Lokalisation von Menschen in Gebäuden unter Verwendung des *Cricket Location Support Systems*. Zusammen mit dem *International Naming Service (INS)* wird dann ein vom Benutzer gewünschter Dienst (z. B. das Abspielen von Musik oder der Ausdruck einer Datei) immer an dem Ort ausgeführt, an dem sich der Benutzer gerade befindet. Im *INS* werden Geräte nicht mehr nur durch Netzwerklokalisierungen, sondern auch durch deren Eigenschaften, deren Ressourcen und deren Daten beschrieben. Die Informationen hierüber werden in den sog. *Twine-Routern* (siehe Abbildung 21) gespeichert, von denen im Allgemeinen mehrere verfügbar sind. Dadurch ist eine gewisse Form der Dezentralisierung gegeben [23]. Von den in Oxygen eingesetzten Middleware-Technologien scheinen sich drei für szenarienübergreifende Anwendungen zu eignen: *CORE* [47], *Click* und *Metaglue*. *Metaglue* wird hierbei wegen seines Umfangs und seiner Bedeutung in Kapitel 3.4 in dieser Arbeit gesondert diskutiert. *CORE* dient innerhalb des MIT dem internen Einsatz zur Erprobung von neuen Komponenten. Es erlaubt einzelnen Geräten miteinander zu kommunizieren. *CORE* ist plattformunabhängig in Java implementiert und benutzt einen zentralen Router. *Click* verwendet ebenso zentrale Router innerhalb der Software-Architektur [143], der aus wieder verwendbaren Mo-

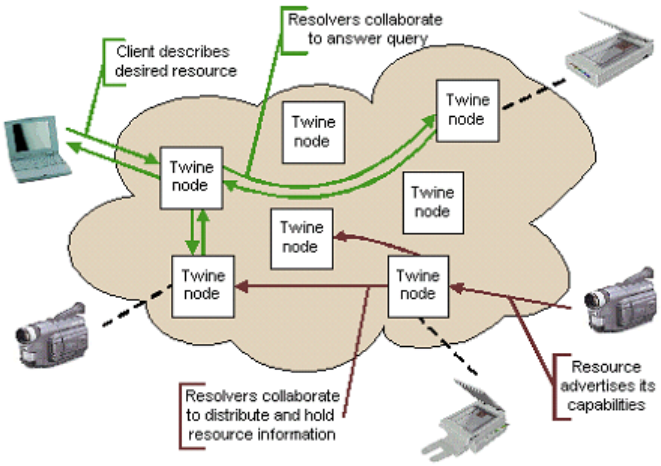


Abbildung 21: Schematische Darstellung des International Naming Service.

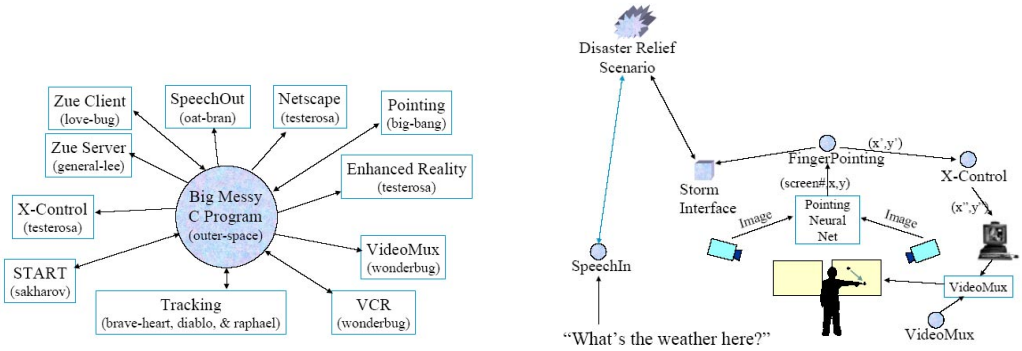


Abbildung 22: Komponentenarchitektur des *Intelligent Room* mit zentraler Kontrollkomponente und Weiterentwicklung auf Basis des „Scatterbrains“ (aus [43]).

dulen aufgebaut ist, die Kommunikationspakete, sog. *Elements* verarbeiten können. Der *Intelligent Room* am MIT wurde entwickelt auf Basis einer monolithischen Kontrollkomponente (siehe Abbildung 22 links) die Coen in [43] selbst „the big messy C program“ nennt. Um den einen zentralen Rechner, auf dem die monolithische Kontrollkomponente lief, zu entlasten und die Rechnerlast auf mehrere Rechner zu verteilten wurde daraufhin die Agentenplattform SodaBot [42] entwickelt. Hiermit waren Funktionalitäten des Intelligent Room verteilbar. Coen selbst nennt das Zusammenspiel dieser kleinen verteilten Komponenten den „Scatterbrain“ (siehe Abbildung 22 rechts). Der Scatterbrain (engl. für Wirrkopf) für den Intelligent Room besteht aus ungefähr 20 verschiedenen Komponenten, die statisch miteinander kommunizieren und auf insgesamt 10 Workstations verteilt sind. Die Kommunikationswege zwischen den einzelnen Komponenten sind hierbei statisch festgelegt. Die dynamische Erweiterung mit neuen Komponenten ist nicht ohne die Veränderung der Kommunikationsregeln an anderen Komponenten möglich. Bei Coen [43] finden sich daher die statischen Beschreibungen von mehreren verschiedenen Anwendungsszenarios.

3.4 Metaglue

Während der *Intelligent Room* des Massachusetts Institute of Technology (MIT) auf Basis der Technologie mit dem „Spitznamen“ Scatterbrain (siehe Kapitel 3.3) realisiert wurde, wurde das Nachfolgeprojekt *Hal* – benannt nach dem Computer im Film 2001: Odyssee im Weltraum – mit der Technologie *Metaglue* implementiert. *Metaglue* ist eine Erweiterung der Programmiersprache Java. Diese Erweiterung führt eine neue Klasse – Agent – ein [45]. Auf der *Metaglue Virtual Machine* werden die Agenten gestartet, die miteinander kommunizieren, indem sie gegenseitig spezifische Methoden aufrufen. Dies ähnelt den in Java gebräuchlichen Techniken des RMI (Remote Method Invocation) [127] und CORBA [46]. Jeder Agent, der im Gesamtsystem beteiligt ist, bringt hierbei seine eigene Runtime mit. Dadurch ist eine gewisse Form der Dezentralisierung und Flexibilisierung gegeben. Jedoch gibt es Grenzen der Flexibilität, die im Folgenden diskutiert werden:

Konfigurationsmanagement: Agenten greifen auf eigene SQL-Datenbanken zu, in denen sie Informationen über sich selbst sowie Informationen über andere Agenten ablegen können. *Metaglue* ist somit schwergewichtig, was die Portabilität auf andere Geräte (außer PCs) erschwert.

Agentenkommunikation: Benötigen Agenten Dienste, die von anderen Agenten bereitgestellt werden, benötigen sie eine Referenz auf diesen, und rufen danach Methoden von freigegebenen Schnittstellen auf. Die Virtual Machine versucht daraufhin einen entsprechenden Agenten zu finden. Über Auswahlprozesse bzw. das Auflösen von Konfliktsituationen (im Falle konkurrierender Dienstagenten) ist in Metaglue nichts bekannt. Es gilt das Prinzip des Erstgefundenen.

Metaglue bietet interessante Ansätze wie die Möglichkeit des Einfrierens von Agentenzuständen, die selbstständige Migration von Agenten und das Auffinden von Diensten. Ein Agent und dessen Funktionen sind also nicht an einen spezifischen Ort gebunden, oder in anderen Worten: ein Agent ist nicht lokalisiert. Er wird stattdessen an einem Ort gestartet, dessen Ressourcen diese Möglichkeit zulassen. Agenten/Komponenten rufen Methoden unbekannter Agenten auf, jedoch müssen diese Methoden bei der Entwicklung eines solchen Systems bekannt sein. Es muss somit eine genaue Spezifikation aller Funktionen und Methoden vor der Entwicklung eines verteilten Systems, welches auf *Metaglue* aufbaut, vorhanden sein. Dadurch ist *Metaglue* nicht zur Laufzeit mit Komponenten erweiterbar, die neue Funktionalitäten in das Gesamtsystem einbringen. Über Konfliktlösungsmechanismen oder Auswahlverfahren möglicher Zielagenten ist in Metaglue aber nichts bekannt. Hier scheinen der Ort der Ausführung oder die Metapher des „first comes, first serves“ zu bestimmen. Dahingehend ist die Intelligenz der verteilten Ausführung auf den auftraggebenden Agenten (d.h. derjenige, der einen Dienst sucht und die entsprechenden Methoden des Zielagenten aufruft) verlagert zu sein. Es ist unklar, in wie weit das MIT die Metaglue-Technologie noch weiter verfolgt. Die nächste Version war für Herbst 2002 angekündigt (vgl. [158]) und bisher (Stand Frühjahr 2007) nicht erschienen. Die Initiative dahingehend scheint weitgehend erlahmt zu sein.

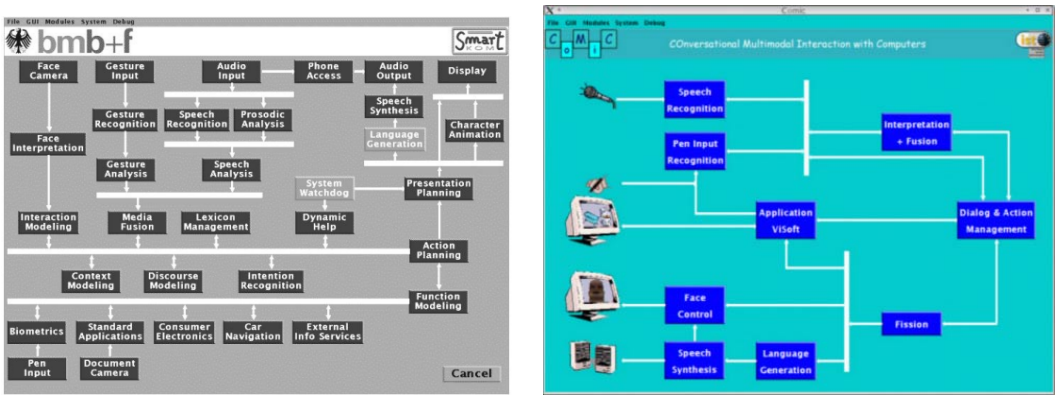


Abbildung 23: Mittels der Administration-GUI der Multiplatform dargestellte Komponentenarchitektur von SmartKom (links) und Comic (aus [117]).

3.5 Multiplatform

Multiplatform (für *Multiple Language Target Integration Platform for Modules*) wurde am Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI) entwickelt. Die Multiplatform wurde realisiert um die Entwicklung und Integration von multimodalen Dialogsystemen auf der Basis von verteilten Komponenten zu ermöglichen[28]. Zur Sicherung der Stabilität wird es in der Multiplatform nicht unterstützt, neue Komponenten zur Laufzeit hinzuzufügen [117]. Die Multiplatform unterscheidet verschiedene Arten an Komponenten. Die Recognizer-Komponenten sind für die Erfassung der Benutzeräußerungen und deren Weiterleitung zuständig. Diese Daten werden dann im Weiteren durch die Analyzer-Komponenten auf der semantischen Ebene interpretiert und dann mit Metadaten durch die Modeller-Komponenten angereichert. In den Generator-Komponenten werden daraufhin die weiteren Dialogstrukturen und Dialogvorgänge geplant. Die Synthesizer-Komponenten sind verantwortlich die Ausgabestrukturen der Generator-Komponenten gemäß den Ausgabekomponenten umzuwandeln. Device und Service sind sog. Connector-Module die den Zugang zu Hardware-Geräten und Applikationsfunktionen kapseln. Kommunikation zwischen Komponenten erfolgt in der Multiplatform auf Basis von Nachrichten und auf Basis des publish-subscribe-Verfahrens. Innerhalb der Multiplatform existieren hierfür spezielle Module. Der *Testbed Manager* ist hierbei verantwortlich für die Initialisierung des Systems sowie das Starten aller verteilt laufenden Komponenten. Dies geschieht auf Basis einer vordefinierten Konfiguration. Eine zweite wichtige Komponente ist die *Testbed GUI* (siehe Abbildung 23) die zur Systemüberwachung, zur Bedienung des Testbed Managers und zur Modifizierung der einzelnen Konfigurationen der verschiedenen Komponenten dient. Die in der Multiplatform zwischen den Komponenten ausgetauschten Nachrichten basieren auf der eigens entwickelten Beschreibungssprache M3L (Multimodal Markup Language) auf Basis des XML-Standards. Ab-

[28]Die Multiplatform wurde (in verschiedenen Entwicklungsstufen) in den Projekten Verbmobil (http://verbmobil.dfki.de/, Laufzeit: 1993-2000), SmartKom (http://www.smartkom.org/, Laufzeit: 1999-2003, siehe auch Kapitel 2.1.2) und dem EU-IST-Projekt Comic (http://www.hcrc.ed.ac.uk/comic/, Laufzeit: 2002-2004) eingesetzt.

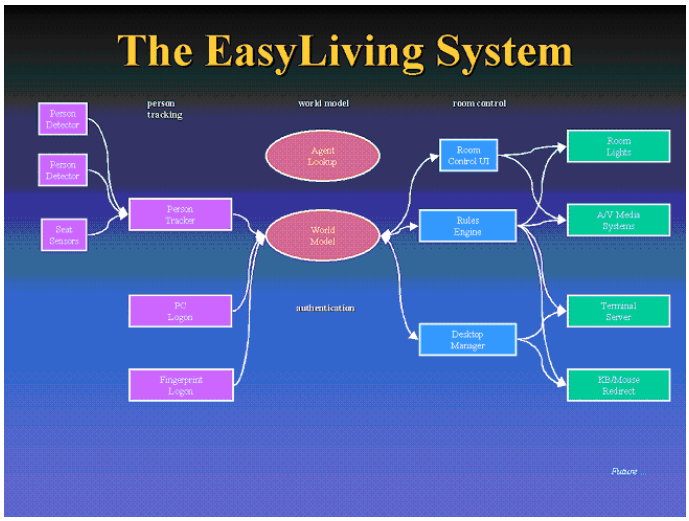


Abbildung 24: In der Easy-Living-Systemarchitektur sind die Komponenten *World Model* und *Rules Engine* die unverzichtbaren zentralen Komponenten.

bildung 23 zeigt links die Anordnung und Kommunikationsverbindungen im *Smartkom*-Projekt (siehe auch Kapitel 2.1.2) und rechts die entsprechende Architektur im EU-IST-Comic-Projekt.

3.6 Easy Living

Die Anwendungsfälle des Projekts *Easy Living* von Microsoft Research wurde schon in Kapitel 2.1.5 ausführlich analysiert. Es bietet aber auch in der Analyse der Realisierung einige interessante Aspekte. In *Easy Living* werden die von herkömmlichen Applikationen bekannten Aktuatoren (*Hardware Device Control*), Logikkomponenten (*Internal Computation Logic*) und Komponenten für die Benutzerinteraktion (*User Interface Presentation*) als tatsächlich getrennt unterschiedliche Komponenten realisiert, die dadurch allgemein zugänglich sind. Weiterhin erlaubt *Easy Living* dem Benutzer die physische Umgebung direkt zu manipulieren. Hierzu sind die Geräte im Raum benamt bzw. in einer Visualisierung direkt für den Benutzer zugänglich. Kernkomponenten der Easy-Living-Systemarchitektur (siehe Abbildung 24) ist das *World Model* und die *Rules Engine*. Innerhalb des World Models ist die Konfiguration des Raumes beschrieben. Zusammen mit den Person-Tracking-Daten ist somit die Lokalisation des Benutzers im Raum und die vom ihm manipulierbaren Geräte möglich. Diese Daten zusammen mit den Daten der *Room Control UI* und des *Desktop Managers* werden in der *Rules Engine* interpretiert. Interferenzprozesse steuern demnach die Geräte des Raumes. Die Systemarchitektur ist streng zentralistisch durch die Anwendung wichtiger zentraler Komponenten. Flexibilisierung im Falle neuer Komponenten ist ohne entsprechende Adaption des *World Models* und der *Rules Engine* nicht möglich. Das primäre Ziel des *Easy-Living*-Projektes liegt jedoch nicht in der Dezentralisierung und ad-hoc Integration von neuen Komponenten. Das Projekt verfolgt eher die Absicht, den herkömmlichen Desktop und die darin befind-

lichen Applikationen auf die Dimension eines Raumes auszubreiten. Hierzu werden die nötigen Interaktionsgeräte (z. B. Tastaturen, Fingerabdruckerkenner, etc.) allen Benutzers im gesamten Raum zugänglich gemacht.

3.7 Die Open Agent Architecture

Ziel der *Open Agent Architecture* (OAA) der SRI International (California)[29] war es, das Paradigma der fest definierten Agentensysteme hin zu flexiblen Systemen zu verschieben (vgl. [154]). Dazu sollten die Nachteile bekannter Ansätze der *Distributed Objects*, der *Conversational Agents* und der *Blackboards* vermieden werden. *Distributed Objects* haben den Nachteil der fest kodierten Objekt- und Methodenaufrufe (vgl. z. B. CORBA [46]). *Conversational Agents* auf Basis einer Agentenkommunikationssprache haben häufig den Nachteil der fest implementierten und damit statischen Agentenadressierung (siehe KQML und FIPA in Kapitel 3.10). *Blackboards* wiederum basieren auf dem Ansatz, dass alle angeschlossenen Komponenten alle vorhandenen Nachrichten und Ereignisse verarbeiten und somit eine kontrollierte vorhersagbare Zusammenarbeit nicht möglich ist. Die *Open Agent Architecture* verwendet das *Delegated Computing Model*, das es Komponenten erlaubt, ihre Anfragen an andere Komponenten in der Form „Was ist zu tun?“ zu richten, ohne eine direkten Adressanten angeben zu müssen. Hierzu wird die entsprechende Aufgabe an eine zentrale Stelle, den sogenannten *Facilitator*, gegeben (siehe Abbildung 25). Die Fähigkeiten der *Open Agent Architecture* sind an vier verschiedenen Typen von Komponenten lokalisiert [154]:

- Requesting Agent (Requester): dieser spezifiziert ein Ziel für den Facilitator und gibt gegebenenfalls Hinweise, wie dieses Ziel ausgeführt werden kann.
- Providers: sog. Service Agenten, die Ziele ausführen können und ihre Fähigkeiten beim Facilitator bekannt machen.
- Facilitator: verwaltet eine Liste an verfügbaren Service Agenten und verfügt über einen Satz an globalen Strategien, wie diese Ziele umgesetzt werden können.
- Meta-Agenten: verfügen über domänen- und zielspezifische Kenntnisse und Strategien. Diese Meta-Agenten werden vom Facilitator zum Auflösen von Konfliktsituationen verwendet.

Somit ist der Facilitator zuständig, mögliche Anfragen zu verwalten, die Leistungen der Provider-Agenten zu koordinieren und die Antworten an die anfragende Komponente zurückzusenden. Gleichzeitig (in der Analogie zu den *Distributed Objects* und den *Conversational Agents*) ist auch die direkte Adressierung von Agenten oder Broadcasts möglich. Dieses Delegationsmodell soll die Agenten von der Verantwortlichkeit der Aufga-

[29] Obwohl die ersten Versionen der Open Agent Architecture schon vor dem Jahr 2000 veröffentlicht wurden ist die OAA immer noch sehr aktuell. Im November 2005 wurde die neueste Version auf der Projektwebseite http://www.ai.sri.com/ oaa/ veröffentlicht (Stand: Frühjahr 2007). Die Projektwebseite listet mehr als 15 Applikationen und Projekte auf, die auf der OAA-Technologie basieren.

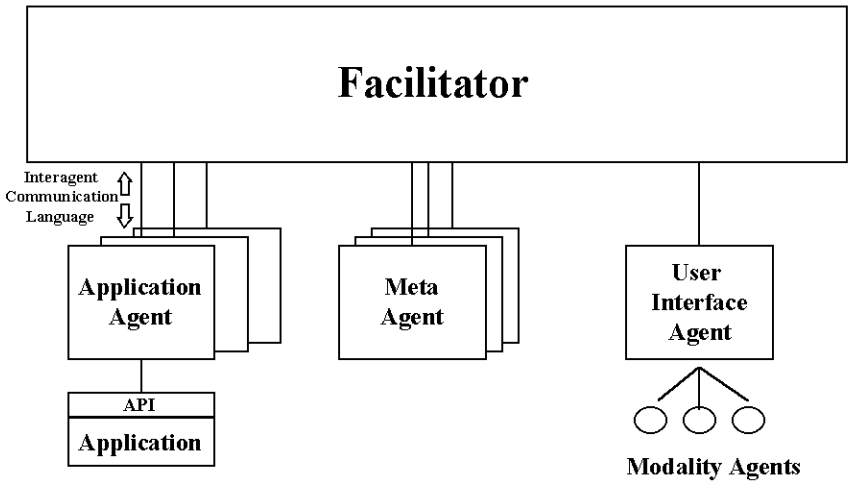


Abbildung 25: Die verschiedenen Komponententypen der Open Agent Architecture mit dem Facilitator als zentrale Vermittlungsstelle.

benplanung, Aufgabenzerlegung und der Ausführungsbeobachtung und Kontrolle entlasten. Jede Nachricht, die von Komponenten verschickt bzw. empfangen wird ist in der *Interagent Communication Language* beschrieben. Diese Sprache spezifiziert die Syntax der Ereignisse, die den Zielen, die Agenten erreichen können, entsprechen. Jedes Ereignis besteht aus einer Typenbezeichnung, einem Satz an Parametern und einem Inhalt. Wie beschrieben, kann jeder Diensteanfrager dem Facilitator Hinweise zur Erfüllung des von ihm gesendeten Zieles mitgeben. Diese betreffen z. B. Angaben über Zeitlimits, der Vorgabe, ob die Anfrage / das Ziel von einem Einzelnen oder im Verbund erfüllt werden sollen und Angaben, wie mögliche Antworten der Provider-Agenten gefiltert oder miteinander kombiniert werden können. Konflikte können in der OAA-Architektur geschehen, falls mehr als ein Dienstagent für die Ausführung eines gewünschten Dienstes zu Verfügung steht. Für diesen Fall verfügen an den Facilitator gebundene Meta-Agenten über Lösungsstrategien. Die Implementierung der Meta-Agenten ist hierbei nicht vorgegeben. Es sind jedoch vier verschiedene Typen definiert: lookup, prioritize, plan_query und execute_plan. Die Meta-Agenten machen sich beim Facilitator mit der Definition ihrer Lösungsfähigkeiten bekannt. Dies geschieht mittels der Angabe „meta(Type, +Goal, +Params, +FacInfo, -Result)“. „Type“ entspricht hierbei dem Typus des Meta-Agenten. Ein Meta-Agent vom Typ „lookup“ verfügt über eine interne Tabelle mit Listen von möglichen verfügbaren Agenten (auch Agenten, die noch nicht gestartet und damit nicht am Facilitator angemeldet sind, die aber gestartet werden können). Startet ein Meta-Agent dieses Typs einen anderen Agent aufgrund dessen dass dieser auf den gewünschten Dienst passt, so informiert der Meta-Agent den Facilitator mittels der Rückgabe eines „true“-Wertes. Meta-Agenten vom Typ „prioritize“ sind in der Lage eine Liste an Agenten nach ihrer Priorität zu ordnen. Diese ursprüngliche Liste bekommt der Meta-Agent vom Facilitator übergeben. Auf welche Weise der Meta-Agent diese Liste ordnet und priorisiert ist von der OAA nicht vorgegeben. Der dritte mögliche Typus eines Meta-Agenten ist der

Typ „plan_query“. Sie sind in der Lage aus einem zusammengesetzten Ziel eine Strategie zu entwickeln und dieses Ziel in Teilziele zu zerlegen. Das Ergebnis ist der modifizierte Plan, der die Teilziele enthält. Entsprechend ist ein Meta-Agent vom Typ „execute_plan“ in der Lage solche Listen an Teilzielen (d.h. Teilaufträge werden in einer Reihe an verschiedene Agenten übergeben) zu verwalten und an Stelle des Facilitators den Diensteagenten zu übergeben. Diese Vorgehensweise kann im Falle von Skalierungsproblemen sinnvoll sein, wenn die Rechenlast vom Facilitator zu den Meta-Agenten ausgelagert werden soll. Der Fall, dass Meta-Agenten im Konflikt zueinander stehen, ist in der OAA nach dem „first-come-first-serve“-Prinzip gelöst worden. Der Facilitator geht der Reihe nach die angehängten Meta-Agenten durch. Der erste, der in der Lage ist eine Konfliktlösung auszuführen, bekommt den entsprechenden Auftrag.

Die *Open Agent Architecture* bietet sehr interessante Ansätze und wird nicht zuletzt deswegen oftmals als wichtige Referenz angesehen. Als einer der ersten Agentenplattformen wird hier die Kommunikation von Agenten untereinander nicht mehr auf Basis von Agentennamen, sondern mittels Ereignissen realisiert. Die Weiterleitung der Ereignisse (z. B. an mögliche Empfänger) geschieht durch den Router / Facilitator auf Basis des Wissens von Meta-Agenten. Die *Open Agent Architecture* verfügt über mehrere zentrale Komponenten, die eine mögliche Flexibilisierung erschweren: den Facilitator und die diversen Meta-Agenten. Hierbei ist die Intelligenz über Auftragsausführungen sowohl in den Agenten (die mögliche strategische Hilfen mitübermitteln können) als auch in den Meta-Agenten oder sogar im Facilitator lokalisiert. Dies mag ein Ansatz sein, um das System flexibel zu gestalten, erschwert jedoch die Transparenz und auch die Entwicklung neuer Komponenten. Auch die Frage, wie die Intelligenz der Meta-Agenten adaptierbar ist, ist nicht hinreichend beantwortet. Häufig basieren die Regeln auf Expertensystemen, deren Erweiterung im Falle neu hinzukommender Agenten schwer ist.

3.8 Der Galaxy Communicator

Die *Galaxy Communicator Infrastructure* [87] ist auf eine Initiative der Spoken Language Systems Group am MIT Laboratory for Computer Science entwickelt worden. Erste Implementationen und Veröffentlichungen datieren aus den Jahren 1998 [190] und 1999 [191]. Aber die Galaxy-Infrastruktur ist immer noch aktuell und in stetiger Weiterentwicklung. Erst 2005 wurde eine spezielle Version in der Programmiersprache Java am Massachusetts Institute of Technology veröffentlicht [88]. Galaxy definiert eine Infrastruktur, die speziell dafür entwickelt wurde, um Sprachtechnologien zu integrieren. Hierzu wird eine Client-Server-Technologie entwickelt, mit der Benutzer mit dem Gesamtsystem kommunizieren können, indem sie kleine (light-weight) Komponenten benutzen, während spezielle Server die rechenintensiven Aufgaben, wie Spracherkennung, Sprachverständnis, Datenbankzugriffe und Sprachsynthese übernehmen. Die zentrale Komponente innerhalb der Architektur ist ein programmierbarer HUB (siehe Abbildung 26), der den Datenfluss zwischen den einzelnen Komponenten und den Servern kontrolliert und dabei den aktuellen Status und die Historie von Gesprächen verwaltet. Das Besondere an der Realisierung der zentralen HUB-Komponente ist, dass sie beim ersten Starten ein spe-

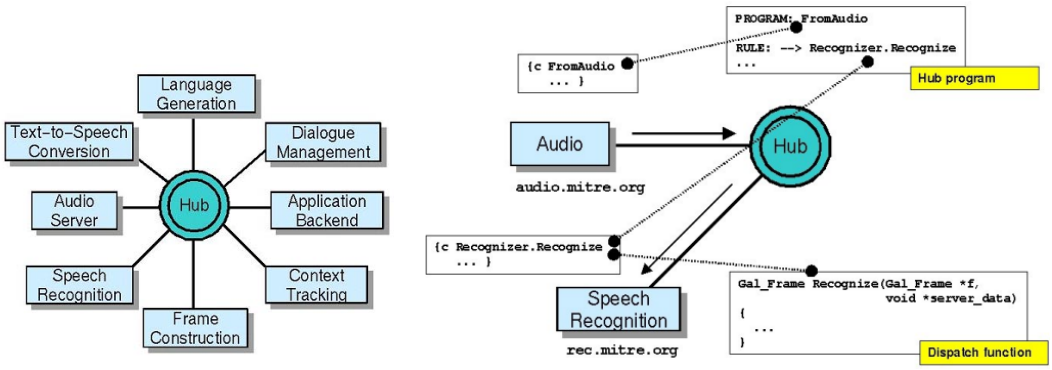


Abbildung 26: Links die prinzipielle Architektur der Galaxy Communicator Infrastructure mit einem zentralen HUB, rechts ein Beispiel der regelbasierten Nachrichtenbehandlung (aus [87]).

zielles File einlädt, in dem Informationen – wie IP-Adresse – der einzelnen Komponenten (in Galaxy heißen Komponenten eigentlich Server) verfügbar sind. Die die Komponenten verbindende HUB-Komponente wird dabei nach den Komponenten gestartet. Abbildung 26 zeigt verschiedene Serverkomponenten, die von Galaxy zur Verfügung gestellt werden und jeweils für spezielle komplexe Aufgaben innerhalb der Gesamtarchitektur verantwortlich sind. Zugleich wird dem HUB beim Start eine Menge an Regeln mitgegeben, die beschreiben in welchem Kontext welche Nachricht an welchen der verschiedenen Server zu versenden ist. Damit seien komplexe Dialogprozeduren möglich, die der Benutzer mittels eines graphischen Frontends, mittels Spracheingabe oder mittels speziellen Desktop-Agenten initiieren kann. Abbildung 26 illustriert im rechten Bild die regelbasierte Vorgehensweise der Nachrichtenbehandlung mittels der zentralen HUB-Komponente. Sendet z. B. eine Komponente, die einen Ton aufgezeichnet hat diesen mit dem Absender „FromAudio“ zum HUB, so sieht dieser in seinen internen Regeln nach. Im Beispiel wird hier gefunden, dass Nachrichten von der „FromAudio“-Komponente an die Komponente mit dem Namen „Recognizer“ weitergeleitet werden soll. Da sämtliche Routingregeln statisch in der zentralen HUB-Komponente vorliegen müssen, ist ein mit Galaxy realisiertes Komponentensystem sehr statisch und nur schwer erweiterbar. Die *Galaxy Communicator Infrastructure* ist stark unter dem Aspekt der Sprachanwendung gewachsen. In diesem Sinne kann hier sogar von einer integrierten Topologie gesprochen werden, da unterschiedliche Komponenten mit unterschiedlichen Aufgabenbereichen einer gewissen Position der Architektur zugeordnet werden können.

3.9 3PC und DIANE

Die Projekte 3PC und Diane beschäftigen sich im Rahmen des Schwerpunktprogramms 1140 der Deutschen Forschungsgemeinschaft (DFG) mit der Erforschung und Realisierung von *Basissoftware für Selbstorganisierende Infrastrukturen für Vernetzte Mobile Systeme*. Das *3PC*-Projekt [1] der Abteilung Verteilte Systeme an der Universität Stuttgart konzentriert sich auf energie-effizientes Service Discovery, Sicherheitsprotokolle für res-

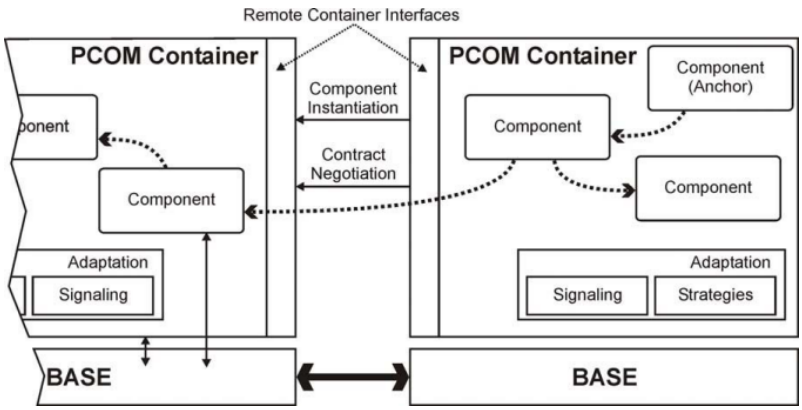


Abbildung 27: Die PCOM-Architektur auf Basis der BASE-Technologie (aus [28]).

sourcenarme Geräte und die automatische Konfiguration von Komponenten-basierten Applikationen. Das *PCOM*-System [28] erlaubt hierbei die Spezifikation von verteilten Applikationen auf Basis der Abhängigkeiten der einzelnen Komponenten. Eine solche Applikation kann dann ausgeführt werden, wenn alle beteiligten Komponenten ausgeführt werden können. Sind für eine Applikation mehrere passende Komponenten verfügbar und stehen somit in Konkurrenzsituationen zueinander, so werden adäquate Strategien angewendet. Adaptionen ohne Eingriff des Benutzers auf sich wechselnde Komponentenensembles sind somit möglich. Abbildung 27 illustriert die Architektur eines PCOM-Gerätes. Der PCOM-Container ist hierbei der Träger für die Komponenten zur Systemüberwachung und für die Ausführung von Adaptionsstrategien. Jeder Container hat dabei ein sog. *Remote Container Interface* definiert, welches die Dienste der internen Komponenten nach außen anbietet. Eine Applikation ist definiert als eine Baumstruktur an Komponenten, die miteinander agieren und kooperieren. Die von den Komponenten benötigten Dienste anderer Komponenten werden auch Abhängigkeiten genannt. Die Wurzel eines solchen Baumes ist dabei charakteristisch für die jeweilige Anwendung (vgl. den Anchor-Knoten in Abbildung 27). Applikationen sind hierbei weiterhin definiert durch die gegenseitigen Abhängigkeiten zur Ausführung der Applikation benötigen Komponenten. In sog. Contracts geben Komponenten die von ihnen zur Verfügung gestellten Dienste und die zur Ausführung des Dienstes benötigen Ressourcen an. Der Lebenszyklus einer Komponente findet innerhalb des PCOM-Containers des Gerätes statt. Um auf wechselnde Umgebungen, die Fluktuation von Komponenten und sich ändernde Ressourcenverhältnisse zu reagieren bietet die PCOM-Architektur einen Adaptionsmechanismus. Der hier zur Verfügung stehende Signalmechanismus teilt registrierten Komponenten die (neue) Verfügbarkeit von Diensten mit (im Falle des Startens neuer Komponenten bzw. der Detektion neuer PCOM-Geräte). Dies geschieht falls ein Dienst ausfallen sollte. Die Benachrichtigung der Komponenten durch den Adaptionsdienst in PCOM findet hierbei auf Basis der von den einzelnen Komponenten definierten Abhängigkeiten statt. Es gibt somit keine unnötigen Nachrichten über Dienste für die keine Nachfrage besteht. Da ein Contract aus der Dienstbeschreibung und der Leistungsfähigkeit besteht, wird eine Komponente auch über Änderungen in der gegenwärtigen Leistungsfähigkeit informiert. In PCOM

sind zwei unterschiedliche Mechanismen zur Adaption implementiert. In der *execution-discontinuation*-Strategie wird die beauftragende Komponente über den Ausfall der nachfolgenden Komponente (im Applikationsbaum) informiert. Diese kann (muss) dann eine andere Komponente beauftragen. Sollte dieser Mechanismus an der Wurzel der Applikation angekommen sein, wird die gesamte Applikation gestoppt. Der zweite Mechanismus – *component reselection* – unterstützt die Neuwahl von Komponenten zur Laufzeit. Hierbei können Strategien auf Basis von Benutzerpräferenzen definiert werden, so dass PCOM automatisch die hier am besten geeignete Komponente auswählen kann. Hierbei kann der Programmierer anwendungsspezifische Routinen implementieren. Wie in Abbildung 27 dargestellt setzt das PCOM-System auf der Middleware BASE auf (vgl. [29]). Während PCOM somit die Dienste der Geräte, ihrer Komponente und Strategien semantisch abstrahiert (siehe Beispiele in [28]) bietet BASE die nötigen adaptiven Kommunikationsmechanismen für die dynamische Auswahl von verschiedenen Protokollen. BASE bietet hier die Möglichkeit Geräte in heterogenen Netzwerken zu erkennen und die darüberliegende PCOM-Schicht mit den nötigen logischen Informationen zum Service Discovery zu versorgen.

Das Projekt *DIANE* [52] – Dienste in ad-hoc Netzen – wird in Kooperation der Universität Karlsruhe mit der Universität Jena im selben Schwerpunktprogramm bearbeit. In DIANE werden die integrierte, effiziente und effektive Nutzung von Ressourcen (und damit Diensten) in ad-hoc Netzwerken erforscht. Konsequenterweise werden daher vor allem die Fragestellungen nach Beschreibung, Ermittlung und Ausführung von Diensten untersucht. Die Dienstebeschreibungen müssen hierbei sowohl ausdrucksstark, als auch maschinenlesbar und maschinell vergleichbar sein. In dynamischen ad-hoc Netzwerken muss das Service Discovery wiederum verteilt und dennoch effizient ablaufbar sein. Der Abgleich in Hash-Tabellen (Anmerkung: analog zu der in Kapitel 3.1 beschrieben Vorgehensweise) ist in DIANE hierbei nicht die erste Wahl. Das Projekt erforscht hier zwei anderen Ansätze: den semantischen Ansatz mittels *Service Rings* und *Multi-Layer Clusters*, sowie Ansätze die weitere Abstraktionen der angebotenen Dienste vornehmen. Hier wird ein Ansatz verfolgt, der aufgrund seiner strukturellen Repräsentation, die entfernte Ähnlichkeit mit Fahrbahnen aufweist, „lanes“ [140] genannt wird. Die Forschungsarbeiten in DIANE widmen sich dem für dynamische Komponentensysteme wichtigen Thema der Diensteauffindung und vor allem der Diensteauswahl im Falle mehrerer passender Diensteanbieter, d.h. im Falle konkurrierender Komponenten. In [141] schlagen Klein und König-Ries ein Verfahren auf Basis persönlicher Präferenzen vor. Statt herkömmlicher „pattern-match“-Verfahren, die die Dienstanfrage mit den Dienstangeboten auf syntaktischer oder semantischer Ebene vergleichen und einen Ähnlichkeitswert in einem gewissen Zahlenintervall ergeben (vgl. Abbildung 28 links) gehen in das Vergleichsverfahren die persönlichen Präferenzen des Benutzers mit ein (vgl. Abbildung 28 rechts). In diesem Verfahren ist es einem Benutzer möglich zu definieren, welche Attribute einer gewünschten Funktion exakt erfüllt sein müssen und welche nur vage oder in einem gewissen Wertebereich erfüllt sein müssen. Um solche Anfragen mit Präferenzinformationen zu formulieren wurde eine Technik auf Basis sog. Fuzzy-Objekte entwickelt [142].

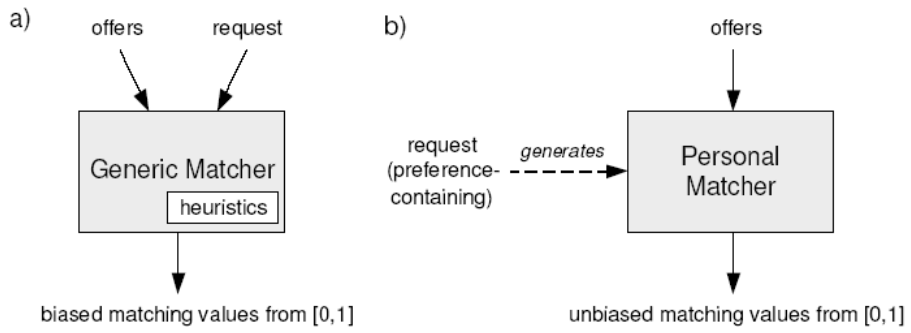


Abbildung 28: Vergleich der Verfahrensweisen generischer Vergleichsverfahren (links) und der in DIANE realisierten Vergleichsansatzes auf Basis persönlicher Präferenzen (aus [141]).

3.10 KQML, FIPA und LARKS

Die *Foundation for Intelligent Physical Agents (FIPA)* [75, 174] und die *Knowledge Query and Manipulation Language (KQML)* sind verschiedene miteinander konkurrierende Ansätze zur Realisierung einer allgemeinen Agentenkommunikationssprache. Sie weisen sehr viele Gemeinsamkeiten auf, so dass sie an dieser Stelle gemeinsam genannt werden können. *FIPA* ist eine internationale Organisation, die im Jahr 1996 gegründet wurde, um Softwarestandards für heterogene und interagierende Agenten sowie agentenbasierte Systeme zu definieren. Im Jahre 2002 waren bereits über 60 universitäre und industrielle Mitgliedsorganisationen in *FIPA* vertreten. *FIPA* wurde am 8. Juni 2005 durch die IEEE Computer Society Standards Organization als elftes Standardisierungskommittee aufgenommen[30]. Eine Agentenplattform gilt als *FIPA* kompatibel, wenn sie mindestens die Spezifikationen zum Agentenmanagement [77] und zur Agentenkommunikation [76] erfüllt. Das Agentenmanagement bildet ein Framework, innerhalb diesem *FIPA*-Agenten existieren und operieren. Es definiert ein logisches Referenzmodel für die Generierung, Registrierung, Lokalisierung, Kommunikation, Migration und das Ausscheiden der Agenten. Das Referenzmodel beinhaltet folgende logische Komponenten mit ihren jeweiligen Fähigkeiten:

- ein *Agent* ist ein Prozess, der die autonome und kommunikative Funktionalität einer Applikation implementiert. Agenten kommunizieren über eine Agentenkommunikationssprache und stellen einen oder mehrere Dienste zur Verfügung, die beim Facilitator (Yellow-Page-Dienst) bekannt gegeben werden können. Jeder Agent hat eine eineindeutige Kennung unter der er auch direkt angesprochen werden kann.

[30] Obwohl die Ursprünge von FIPA damit schon mehr als 10 Jahre zurück liegen ist dieser Quasi-Standard für die Agentenkommunikation sehr aktuell. Die bekannteste Implementierung einer Agentenplattform mit dem FIPA-Standard dürfte das *Java Agent DEvelopment Framework* (JADE, http://jade.tilab.com) sein. Die neueste Version datiert hierbei aus dem März 2005 (Stand Frühjahr 2007).

- ein *Agentenmanagementsystem – AMS* ist die obligatorische Komponente jeder Agentenplattform. Jeder Agent muss sich beim AMS anmelden, das die Agenten in den sog. White-Page-Diensten verwaltet (IP-Nummer, etc.).
- der *Directory Facilitator* ist eine optionale Komponente jeder Agentenplattform. Agenten können ihre Dienste hier registrieren und mögliche Adressagenten für gesuchte Dienste finden.

Jede *FIPA*-Nachricht setzt sich aus mehreren Parametern zusammen, die den Kommunikationsakt, die verschiedenen Teilnehmer der Kommunikation und den Inhalt der Nachricht definieren. *KQML*[31] definiert im Prinzip ähnliche Komponenten einer Agentenplattform. Die bekannteste Implementierung einer Agentenplattform auf Basis des KQML-Standards ist die JatLITE-Implementierung [128]. Die Unterschiede in der Spezifikation zu FIPA sind hierbei nur marginal und betreffen im Wesentlichen die Definition einzelner unterschiedlicher Sprechakte. Für nähere Details sei hier auf [73, 74, 149, 156, 157] verwiesen. Ein interessanter Ansatz in Bezug auf Konfliktlösungsmechanismen findet sich bei *KQML*. Barbuceanu et al. [25] definieren eine Ergänzung – KQML-Subset COOL – zu den bekannten Methoden einer Agentenkommunikationssprache um Sprechakte für Verhandlungen möglich zu machen. Konflikte können in *KQML* somit durch Verhandlungen von einzelnen Agenten gelöst werden. Die Intelligenz zum Lösen von Konflikten wird also komplett in die einzelnen Komponenten verlagert.

Bei *KQML* und *FIPA* handelt es sich genaugenommen nicht um eine Technologie, sondern um Protokollstandards, die die Sprechakte für die Kommunikation von verteilten Komponenten definieren. Ein Sprechakt besteht bei beiden Standards aus einem Performative und verschiedenen Parametern. Das Performative gibt den Verwendungszweck einer Nachricht an (z. B. eine Frage oder eine Anweisung), die Parameter sind zur näheren Beschreibung der Nachricht und für den eigentlichen Inhalt zu verwenden. Beide basieren auf dem Ansatz, jeder Komponente einen eineindeutigen Bezeichner zu geben. Dieser kann im Allgemeinen direkt mit der physikalischen IP-Adresse verknüpft sein. Dynamisierung, d.h. das Hinzufügen und Entfernen von Komponenten ist nur über den folgenden Mechanismus möglich:

- die einzelnen Agenten melden ihre Fähigkeiten beim Facilitator an
- jede Dienstanfrage wird an den Facilitator gegeben, der eine Auswahl möglicher Adresskomponenten anbietet
- interne komponenteneigene Strategien wählen den geeigneten Adressaten aus

Projekte auf Basis von *FIPA* oder *KQML* zeigen aber, dass selten ein Facilitator eingesetzt wird und die Implementation aller Komponenten mit festen Kommunikationswegen realisiert wird. Die wichtigsten Nachteile beider Ansätze in Stichpunkten:

- keinerlei Methodiken oder gar Ansätze für das Auflösen von Konkurrenzsituationen (z. B. für den Fall, dass der Facilitator auf Anfrage mehrere Diensteanbieter nennt)

[31] Erreichbar unter http://www.cs.umbc.edu/kqml/.

- keinerlei Ansätze für Problem Decomposition (z. B. eine Nachrichtenzerlegung in Teilnachrichten für den Fall, dass der Facilitator keinen passenden Diensteanbieter nennt)
- viele zentrale Komponenten (Router, White-Pages, Yellow-Pages, Facilitator)
- direkte Kommunikation von Agenten bedeutet immer, dass Intelligenz in den einzelnen Komponenten verfügbar sein muss. Dies macht verteiltes Implementieren nahezu unmöglich, und macht das Verhalten eines Gesamtsystems u.U. nicht zuverlässig vorhersehbar.

LARKS [201] steht für *Language for Advertisement and Request for Knowledge Sharing* und ist wie KQML oder FIPA eine Beschreibungssprache. LARKS erlaubt es Applikationen die sog. Capability von angebotenen Diensten zu beschreiben. Ein – im Vergleich zu KQML und FIPA – zusätzlicher *Matchmaker-Agent* ermöglicht es, angefragte Dienste mit den angebotenen Diensten zu vergleichen. Dieser *Matchmaking Process* moderiert somit zwischen den Diensteanfragern und den Diensteanbietern. Nach erfolgter Auswahl gibt der Matchmaker-Agent die Adresse des passenden Diensteanbieters an den Diensteanfrager weiter. Dieser Agent wirkt somit (aus Gründen der Vermeidung von Überlast) nicht als traditioneller Broker. LARKS definiert hier ein Beschreibungsformat, in dem der Kontext der aktuellen Anfrage, die Input- und Outputvariablen und nötige Einschränkungen (sog. Constraints) und andere notwendige Beschreibungen der Anfrage (definiert als Description) definiert sind. Um die Ontologie der Beschreibungen zu formulieren, verwendet LARKS dabei zusätzlich die Beschreibungssprache ITL (Information Terminology Language). Fünf verschiedene Strategien werden angeboten, um Anfragen und Angebote miteinander abzugleichen. Diese Strategien reichen von exaktem Match bis hin zu schwachen Match, in dem die letztlich nur noch die Einschränkungen aufeinander passen. Die Strategien liefern dabei nur eine geringe Anzahl an hoch gewerteten Diensten. Dies geschieht um eine Überforderung des Benutzers, der bei mehreren gleichwertigen Diensten die Endauswahl treffen muss, zu vermeiden [32].

3.11 Jaspis, iCrafter und INCA

Jaspis von der Universität Tampere in Finnland ist eine Architektur für Sprachanwendungen, die speziell für die adaptiven und multilingualen Anforderungen in Echtzeitumgebungen entwickelt wurde. Abbildung 29 illustriert den generellen Aufbau eines Jaspis-Systems. Die oberste Ebene einer solchen Struktur wird von sog. *Manager*-Komponenten gebildet, die mit der zentralen Managerkomponente verbunden sind. Die Kommunikation von Komponenten findet mit einem Client-Server-Ansatz statt. Eine Neuerung gegenüber Systemen wie der *Open Agent Architecture* oder dem *Galaxy Communicator* besteht in der Einführung des Agenten-Manager-Evaluator-Paradigmas (vgl. auch Abbildung 30).

[32] Zitat: „The matching process should be effective, that is, the set of matches should not be too large. For the user, typing in a request and receiving hundreds of matches is not necessarily very useful. Instead, we prefer a small set of highly rated matches to a given request.“

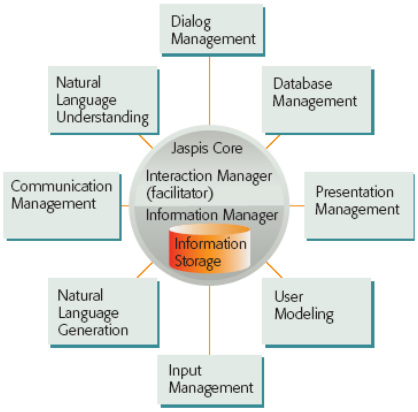


Abbildung 29: Die allgemeine Systemstruktur von Jaspis als Client-Server-Architektur (aus [204]).

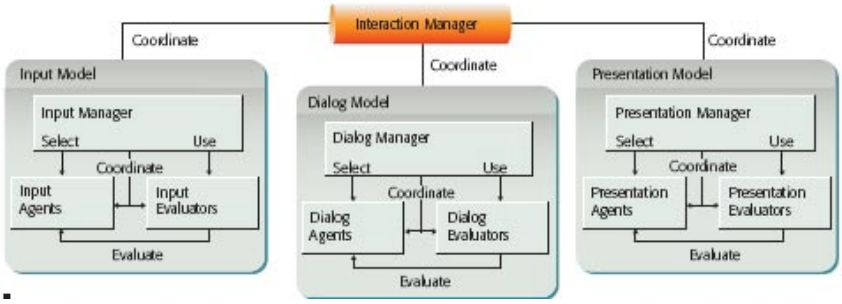


Abbildung 30: Die Architektur des Interaktions-Managements auf Basis von Agenten, Evaluatoren und Managerkomponenten (aus [204]).

Agenten in Jaspis sind definiert als Interaktionskomponenten für unterschiedliche Interaktionstechniken (z. B. Komponenten für Sprachausgabe oder Dialogentscheidungen). Die Evaluatoren sind dafür zuständig verschiedene Aspekte der Agenten zu evaluieren und festzustellen, wie geeignet ein Agent für eine bestimmte Aufgabe ist. Die Manager-Komponenten sind dafür zuständig die Handlungen der Evaluatoren und Agenten zu koordinieren. Agenten können mittels des zentralen Informationsrepositories auf gemeinsame Informationen zugreifen. Abbildung 30 illustriert das Interaktionmanager-Modell. Ein solches Modell besteht aus einem Eingabemodul für die Benutzerinteraktionen, einem Dialogmodul und einem Präsentationsmodul für die Systemausgaben an den Benutzer. Jedes Modul besteht wiederum aus einer lokalen Managerkomponente und kann aus mehreren Agenten und Evaluatoren bestehen. Welcher Agent in welcher Situation verwendet wird, wird lokal von der moduleigenen Managerkomponente entschieden. Diese können diese Entscheidung aber auch den Evaluatoren übertragen. Somit ist eine gewisse Form von Dezentralität gewährleistet. Das Konzept der Evaluatoren ist durch Jaspis eingeführte Neuerung. Es liegt in der Verantwortung eines Evaluators verschiedene Agenten (innerhalb desselben Moduls) miteinander zu vergleichen und jedem eine bestimmte Eva-

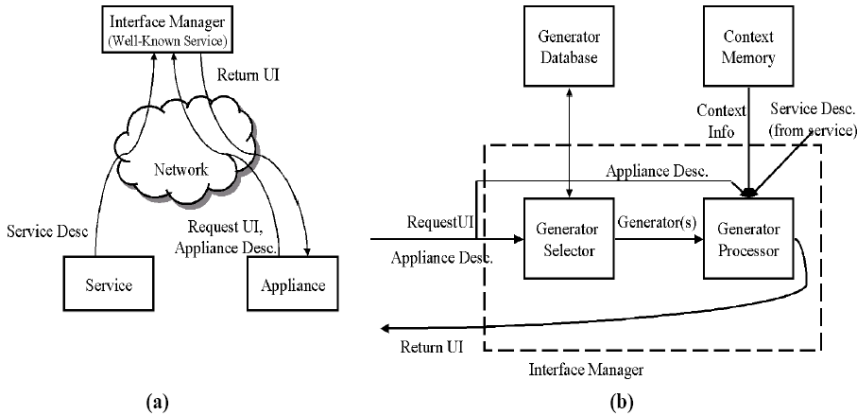


Abbildung 31: Illustration der *iCrafter*-Architektur: (a) Anfrage einer Applikation nach einer graphischen Benutzeroberfläche mittels des Interface-Managers. (b) Interne Struktur des Interface-Managers (aus [175]).

luierungspunktzahl in der jeweiligen Situation zu geben. Wie Agenten können hier auch Evaluatoren für bestimmte Situationen besser und andere Situationen schlechter geeignet sein. So evaluieren unterschiedliche Evaluatoren jeweils unterschiedliche Aspekte bei den Agenten. Turunen et al. geben als Beispiel (vgl. [204]) eine Situation, in der Dialogkomponenten Probleme des Benutzers bemerken und daher diejenigen Agenten eine höhere Punktzahl bekommen, die detaillierte und hilfreichere Dialogstrukturen benutzen. Das Verfahren verläuft wie folgt: Soll ein Systemmodul aktiv werden (d.h. ein Agent innerhalb eines Systemmoduls) so evaluiert jede Evaluatorkomponente jeden Agenten. Die moduleigene Managementkomponente multipliziert die verschiedenen Punktzahlen jedes Agenten. Derjenige Agent mit der nun höchsten Punktzahl muss die Aufgabe des Moduls ausführen.

Im Rahmen des *Interactive-Workspaces*-Projekt wurde (vgl. Kapitel 2.1.4) die *iCrafter*-Infrastruktur entwickelt und eingesetzt. Diese Infrastruktur hat zum Ziel dem Benutzer eine flexible Interaktion mit den angebotenen Diensten der Umgebung anzubieten. Dabei sollen eine Reihe von Eingabemodalitäten unterstützt werden [175]. Dienste werden hierbei als Funktionen von Geräten und als Applikationen verstanden. Mittels *iCrafter* erhalten die Entwickler des *iRooms* die Möglichkeit die Dienste ihrer Applikationen und Geräte zu entwickeln und Benutzerschnittstellen zu realisieren. In *iCrafter* wird hierbei das Ziel der Adaptivität verfolgt. Damit ist gemeint, dass unterschiedliche Applikationen sich unterschiedlichen Modalitäten anpassen können. Aber auch, dass entsprechend den Eingabemöglichkeiten sich Aussehen und Art der Benutzerschnittstellen einer Applikation ändern kann. Die Wiederverwendbarkeit und Anwendbarkeit wird in *iCrafter* erreicht durch die automatische Generation von graphischen Benutzerschnittstellen. Aggregationsfunktionen sollen es ermöglichen, Kombinationen von Funktionen anzubieten, z. B. für den Fall, wenn zwei Geräte zusammenkommen, deren Funktionen miteinander kombiniert neue Mehrwertfunktionen generieren. Abbildung 31 illustriert die prinzipielle Ar-

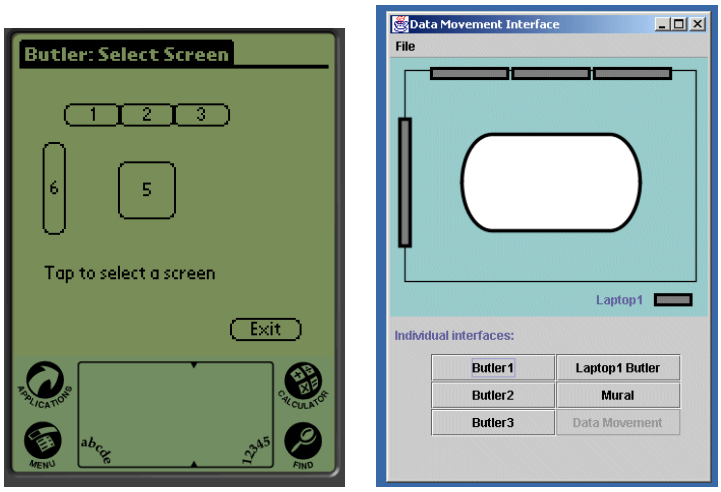


Abbildung 32: Von *iCrafter* produzierte graphische Oberflächen für Palm-PDAs (links) und für Java Swing (rechts) im *Interactive-Workspaces-Projekt*.

chitektur der *iCrafter*-Software. Hauptkomponenten ist ein zentraler Interface-Manager, der in der Lage ist Anfragen nach Benutzerschnittstellen zu verarbeiten und auf Basis von Kontextinformationen mittels eines UI-Generators graphische Benutzeroberflächen zu erstellen. Diese werden dann an den Anfrager zurückgesendet. Um dies zu erreichen werden im Interface-Manager die Dienstebeschreibungen der eingetragenen Geräte und Applikationen verwendet. Eine Dienstbeschreibung besteht aus der Dienstefunktion und einigen Parametern. Dazu gehören aber keine Namen von Geräten, IP-Adressen oder URLs. Für den Austausch von Diensten und die dazugehörigen Benutzerschnittstellen wird die XML-Beschreibungssprache verwendet. Für die Kontrolle der Funktionen ist hierfür ein EventHeap (in Abbildung 31 nicht dargestellt) auf der Basis von IBM TSpaces zuständig. *iCrafter* unterstützt als Beschreibung für Benutzerschnittstellen die Formate HTML, VoiceXML (für Sprachausgabe), MoDAL (für Palm-PDAs, siehe Abbildung 32 links) und SUIML (für die Implementation von Java Swing Oberflächen, siehe Abbildung 32 rechts).

INCA (Infrastructure for Capture and Access) des Georgia Institute of Technology ist eine Infrastruktur zum Erfassen und Wiedererlangen von Informationen, wie sie zum Beispiel bei gemeinsamen Arbeitstreffen entstehen [202]. Abbildung 33 illustriert die generelle Architektur. An einer zentralen Registry-Komponente können sich Module für die Informationsgewinnung (Capturer), für die Informationsspeicherung (Storage), für die Informationskonvertierung (Transduction) und für den Informationszugriff (Access) konnektieren. Die Anzahl an Komponenten von jedem Typ ist hierbei nicht begrenzt. Zur Vereinfachung der Benutzung von INCA gehören vorprogrammierte Module, die ein Entwickler weiterbenutzen oder erweitern kann (das erweiterte Architekturschema hierfür zeigt Abbildung 34). Ein Capture-Modul ist dafür zuständig Informationen aus der Umgebung zu sammeln, mit Metadaten anzureichern und in vorhandenen Storage-Modulen abzulegen. Storage-Module bieten neben der reinen Datenspeicherung zusätzliche Dienste wie eine automatische Benachrichtungsfunktion an. INCA legt hierbei die gespeicherten Daten entweder in einem Datenbankformat oder in einer Datei ab. Entsprechend sind

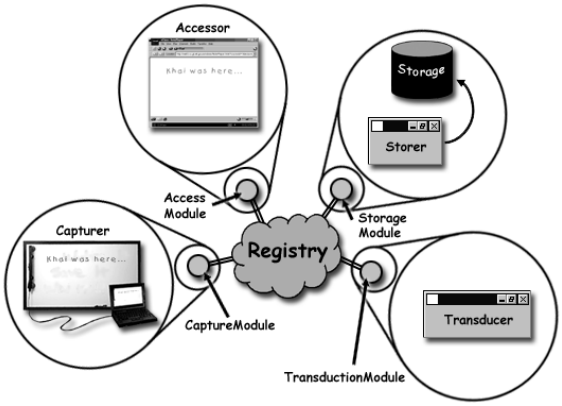


Abbildung 33: Illustration der *INCA*-Architektur mit der zentralen Registry als Kontaktpunkt seiner Komponenten (aus [202]).

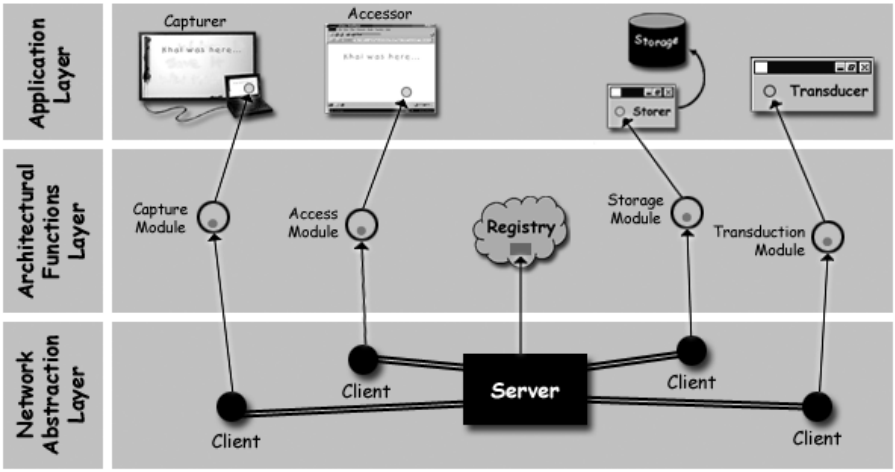


Abbildung 34: Die unterlagerte Schichtenarchitektur der *INCA*-Infrastruktur, die zwischen Netzwerkschicht, Funktionsschicht und Applikationsschicht unterscheidet.

Access-Module in der Lage sich für bestimmte Informationen bei einem Storage-Modul zu registrieren. Eine Access-Komponente kann aber auch aktiv nach Informationen fragen. Die Funktionen eines Transduction-Moduls werden bei INCA automatisch aufgerufen. da sich ein solches Modul beim zentralen Registry mit seinen Fähigkeiten (z. B. Konvertierung einer WAV-Datei in eine MP3-Datei) angemeldet hat. Falls Module nach Informationen in einem bestimmten Dateiformat fragen, können die Informationen somit auch dann erlangt werden, falls sie in einem anderen Dateiformat vorliegen sollten.

3.12 Middlewaretechnologien

Reine Technologie für die Middleware haben innerhalb eines Geräteensembles weniger die Kommunikationssemantik und -syntax, die für übergeordnete Mechanismen zur

Steuerung des Kommunikationsflusses verantwortlich sind, im Blick. Die unterschiedlichen Ansätze zur Middleware-Unterstützung haben vielmehr gemeinsam zum Ziel unterschiedliche heterogene Geräte oder Komponenten mit einem Geräteensemble zu verbinden. Teilweise bieten sie Mechanismen aus Agententechnologien, wie Look-up-Services (siehe *Jini* in Kapitel 3.12.1) oder Gruppierungsmöglichkeiten von Einzelkomponenten (siehe *JXTA* in Kapitel 3.12.3). Gemeinsam ist den verschiedenen Initiativen zur Definition von Middleware aber, dass sie Protokolle definieren, die die Bildung eines Geräteverbundes möglich machen und Gerätekommunikation ermöglichen. Strategien zur Konfliktlösung oder für die Auswahl eines passenden geeigneten Dienstes stehen nicht im Vordergrund dieser Technologien. Als Beispiel für die Trennung von Middleware-Funktionalitäten und darüber angesiedelter Software-Infrastruktur kann das BASE-PCOM-Zusammenspiel aus Kapitel 3.9 (vgl. auch Abbildung 27) dienen.

3.12.1 Jini

Jini [129, 211] ist eine von SUN geförderte Technologie, die auf Basis der Java 2 Plattform den Aufbau von Netzwerken durch Bereitstellung einer eigenen Infrastruktur fördern soll. Damit ist Jini vergleichbar mit einer Reihe anderer verteilter Systemarchitekturen, wie z. B. CORBA [46]. *Jini* hat zum Ziel, innerhalb eines Netzwerkes sowohl Software-Dienste als auch Hardware-Geräte zu spontanen Gemeinschaften zusammenzufassen. Dabei können Hardware-Dienste jedes beliebige Gerät (sog. Embedded-Devices) sein. Die Gemeinsamkeiten – und damit auch die Nachteile – mit CORBA sind offensichtlich: Jeder Software- und Hardware-Dienst wird durch Objekte repräsentiert, auf welche über spezielle Java-Interfaces zugegriffen wird. Hier wird direkt Programmcode über das Netzwerk übertragen (mittels der Serialisierung von Objekten oder RMI [127]. *Jini* definiert somit keinerlei Protokolle (im Gegensatz zu den Agentenplattformen), sondern verfolgt den Ansatz der verteilten Java-Implementierung unter der Verwendung von Objekt-Schnittstellen. Grundsätzlich unterteilt sich *Jini* in drei Teile. Ein verteiltes Sicherheitssystem, welches im Folgenden nicht weiter diskutiert werden soll, das *Discovery-/Join*-Protokoll und den *Lookup*-Service. Um einer Einheit (Dienst oder Anwendung) die Teilnahme an einem Netzwerk zu ermöglichen, sucht diese mittels des *Discovery*-Protokolls (siehe Abbildung 35) eine bereits vorhandene Jini-Gemeinschaft. Das Ergebnis dieser Netzwerksuche kann eine oder mehrere Referenzen auf verfügbare Look-up-Services sein. Jedes Gerät kann sich mit den Schnittstellen zu seinen Diensten an solchen Look-up-Services registrieren. Dabei wird die Schnittstelle eines Dienstes von seiner Implementierung getrennt. So kann es passieren, dass ein Client auf eine Dienstanfrage mehrere Schnittstellen zur Verfügung bekommt, die intern unterschiedlich implementiert sind (z. B. mit RMI (von *Jini* empfohlen), aber auch mittels Webservices, CORBA, DCOM, etc.). Mittels der Schnittstelle wird mit dem Diensteanbieter direkt kommuniziert. Jini bietet einige Grunddienste wie Persistenzunterstützung oder Ereignismodelle zur Erkennung von ausfallenden Geräten, die für ein selbstorganisiertes System wichtig sind, um die Zuverlässigkeit von spontanen Geräteensembles zu garantieren. Jedoch verwendet *Jini* zentrale Look-up-Services, die wie bei anderen Ansätzen auch als Facilitatoren agieren. Die reine Implementation in Java nicht nur auf Geräteebene, sondern

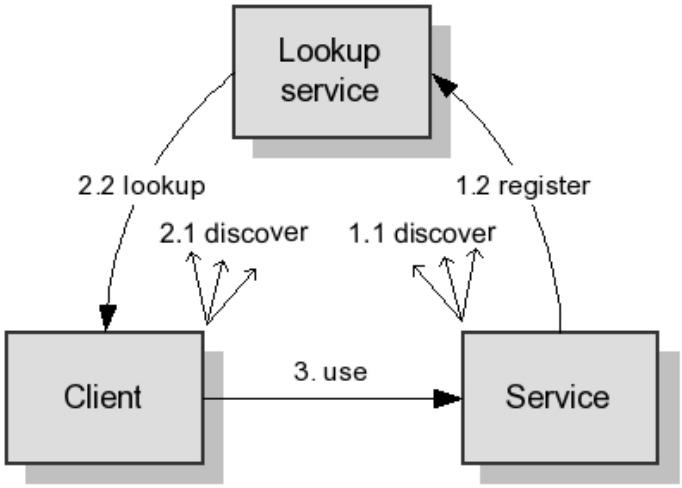


Abbildung 35: Funktionsweise des Look-up-Services bei Jini.

auch auf Schnittstellenebene, erschweren die Portabilität auf heterogene Geräte. Zumindest eine Java Virtual Machine muss immer verfügbar sein. *Jini* bietet keinerlei Möglichkeiten intelligenter Kommunikationsmechanismen wie Konfliktlösungsstrategien im Falle konkurrierender Dienste oder das Aufteilen von Aufträgen in Teilaufträge. Das Anwendungsszenario ist klar begrenzt. Es wird ein Dienst gesucht, die verfügbaren Dienste werden angezeigt, und müssen daraufhin vom Benutzer selektiert werden (z. B. Suche eines Druckers im Netzwerk).

3.12.2 Universal Plug and Play

Die Initiative *Universal Plug and Play* [206] wurde von Microsoft mit dem Ziel gegründet, das Konzept *Plug and Play* vom einzelnen PC auf Heimnetzwerke zu übertragen. *Universal Plug and Play* soll den Benutzern die Installation und Bedienung vernetzter Geräte vereinfachen durch

- die automatische Geräteerkennung und möglichst konfigurationslose Installation
- die Garantie der Interoperabilität und der komfortablen Bedienung aller angeschlossenen Geräte
- die Verbindung von computernaher Hardware, Unterhaltungselektronik und Heimautomatisierung.

Grundsätzlich (siehe Abbildung 36) werden in *UPnP* drei Arten von Komponenten unterschieden (vgl. [159]):

Device (Gerät): Geräte sind Container für eingebettete Services und wiederum für andere Geräte. Alle Eigenschaften und Services sind in standardisierten Gerätebeschreibungen in XML beschrieben.

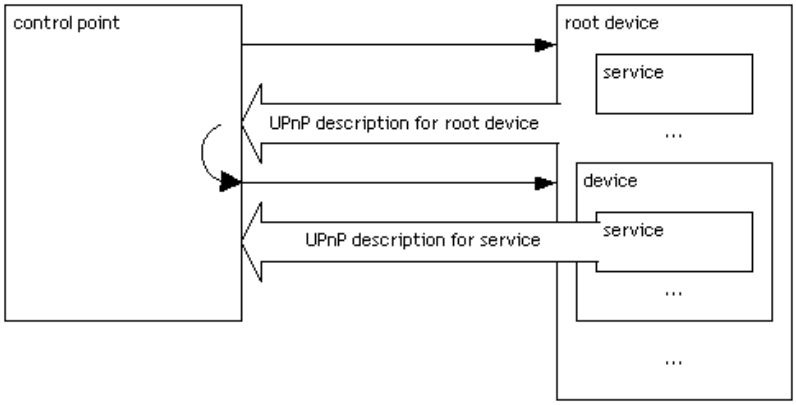


Abbildung 36: Schematische Darstellung der unterschiedlichen Arten der UPnP-Komponenten und deren Kommunikationsmechanismen.

Service (Dienste): Ein Service bietet den Benutzern den Zugriff auf eine Gerätefunktionalität. Hierbei ist ein Dienstezustand beschrieben durch seine Statusvariablen und die Aktionen, die diese Variablen beeinflussen können und wie diese abzufragen sind.

Control-Point: Ein Control-Point steuert Geräte über Services entweder automatisch oder bietet dem Benutzer die Möglichkeit der Steuerung.

Im Gegensatz zu *HAVi* oder *Jini* verwendet *UPnP* standardisierte, auf TCP/IP-basierende Protokolle. Hierbei werden nur Daten, aber keine Programme oder Objekte ausgetauscht. Aus diesem Grund ist *UPnP* generischer und unabhängiger von Programmiersprachen. Die wichtigsten Protokolle seien hier kurz erwähnt: das *Simple Service Discovery Protocol (SSDP)* wird für das Auffinden von Geräten und Diensten verwendet, die *Generic Event Notification Architecture (GENA)* definiert das Konzept über Subscriber und Publisher von Ereignissen und das *Simple Object Access Protocol (SOAP)* definiert die Art der Benutzung von XML und HTTP für Remote Procedure Calls. Hierbei wird der de-facto-Standard SOAP [197] für die Kontrollnachrichten – diese enthalten die jeweils auszuführende Aktion und die zugehörigen Parameter – und die entsprechenden Antwortnachrichten. Sechs Phasen wurden zum Betrieb einer UPnP-Komponente definiert:

Adressierung: Nach dem Start erhält jedes Gerät eine eindeutige IP-Adresse.

Discovery: Ein Control Point sucht über eine fest definierte Multicast-Adresse nach interessanten Geräten oder Diensten. Geräte und Dienste antworten auf diese Anfrage, woraufhin der Control Point die gefundenen Dienste in seiner Datenbasis einträgt. Geräte geben sich nach dem Start durch Broadcast-Nachrichten bekannt. In regelmäßigen zeitlichen Abständen werden aus diesem Grund SSDP-Nachrichten ausgesendet, die jeden verfügbaren Service beschreiben. Diese Nachrichten werden von allen Control Points empfangen und in den entsprechenden Datenbasen eingetragen.

Description: Während der Discovery-Phase kann ein Control Point weitere Informationen von jedem Gerät erfragen. Das sind: detailliertere Gerätebeschreibungen, herstellerspezifische Informationen, die zugehörigen URLs zur Steuerung von Aktionen oder Präsentationsbeschreibungen für graphische Oberflächen.

Control: Nachdem jeder Control Point die notwendigen Informationen zur Steuerung eines Gerätes besitzt, kann es mittels SOAP Anfragen und Befehle versenden. Innerhalb von 30 Sekunden muss jedes Gerät geantwortet haben, um sicherzustellen, dass der Auftrag angenommen wurde und das Gerät noch vorhanden ist.

Eventing: Mittels des Eventing-Mechanismus können alle Control Points über aktuelle Gerätezustände informiert werden. Hierzu meldet ein Control Point sein Interesse für einen bestimmten Service an und wird danach über alle Änderungen benachrichtigt.

Präsentation: Zusätzlich zu Control kann ein Gerät in der Gerätebeschreibung die URL einer HTML-Seite angeben. Diese HTML-Seite kann ein Control Point mittels HTTP in einen Browser laden und so dem Benutzer die Möglichkeit geben, das Gerät direkt per graphischer Benutzeroberfläche zu kontrollieren.

In verschiedenen Bereichen der Geräteindustrie (vgl. [206]) werden Geräteprofile entwickelt. Hierbei werden neben den Profilen auch die Dienste, die Statusvariablen und deren Semantik definiert. Es existieren entsprechende Standards für Drucker, Scanner und Router aber auch für weiße Ware wie Heizungs- und Klimaanlagen. In der nahen Zukunft ist mit entsprechenden Protokollen vor allem für Unterhaltungsgeräte zu rechnen, da hier der weitaus größte kommerzielle Markt zu erwarten ist. Eine Einschränkung hierbei ist zu machen: *UPnP* wird nicht die Übertragung und das Format von AV-Medien definieren. Das ist per Definition nicht Aufgabe des Standardisierungsgremiums, sondern Aufgabe der einzelnen Geräte. Es kann somit innerhalb eines Netzwerkes zu Inkompatibilitäten kommen. Im Vergleich zu anderen Technologien der Spontanvernetzung wie *Jini* oder *HAVi* erweist es sich als großer Vorteil von *UPnP*, dass hier keine spezifische Plattform oder Programmiersprache vorausgesetzt wird. Da *UPnP* die Beschreibung und die Steuerung von Geräten und Diensten standardisiert ohne weiter reichende Anforderungen zu stellen, sind diesem Gremium schon viele Hersteller beigetreten (z. B. Loewe, Bose, Nokia, Phillips, Samsung, etc.), so dass in sehr naher Zukunft mit den ersten Geräten und darauf aufbauenden Geräteensembles zu rechnen sein wird. *UPnP* bietet aber keine Lösung für Selbstorganisation an. *UPnP* soll es dem Benutzer „nur“ ermöglichen, mit jedem zur Verfügung stehendem Geräat eines Geräteverbundes interagieren zu können. Jeder Dienst soll an jedem Ort zur Verfügung stehen. Daher kann *UPnP* durch die mächtige semantische Beschreibung der Geräte und Dienste für die Entwicklung selbstorganisierter Geräteensembles eine gute Ausgangsbasis sein. Hierbei würde jedes Geräte eine *UPnP*-Beschreibung vorhalten.

3.12.3 JXTA

JXTA [132] wurde im Mai 2001 von Sun Microsystems in einem Open Source Forschungsprojekt gestartet. Das wesentliche Ziel war es, eine Plattform zu entwickeln, die insbesondere Interoperabilität, Plattformunabhängigkeit und Geräteunabhängigkeit unterstützt. Offenbar hatte man bei SUN aus den Erfahrungen der geringen Akzeptanz von *Jini* gelernt. Dabei wurde ein komplettes Peer-to-Peer-Netzwerk ohne eine zentrale Stelle angestrebt. Die *Interoperabilität* soll es miteinander verbundenen Peers (Teilnehmern) ermöglichen, andere Verbundpartner zu lokalisieren, mit ihnen zu kommunizieren (direkte Peer-to-Peer-Verbindung) und gemeinsam Aktivitäten (wie Dateiaustausch) zu ermöglichen. Die *Plattformunabhängigkeit* sollte garantieren, dass auch Geräte, die in einer anderen Programmiersprache als Java implementiert wurden, Teil des Verbundnetzes sein können und somit auch light-weight-Geräte (wie Telefone, PDAs, etc.) angesprochen werden können (*Geräteunabhängigkeit*). Aus diesem Grund besteht *JXTA* im Gegensatz zu *Jini* aus einer Menge von offenen Protokollen, die die Kommunikation und Kooperation von miteinander verbundenen Geräten beschreiben. Wie bei *UPnP* sind die Protokolle hierbei in XML formuliert. Ein JXTA Netzwerk besteht aus einer Menge miteinander verbundenen Peers. In JXTA definierte Protokolle bestimmen dabei, wie andere Peers gefunden werden können, wie die Organisation von Peers in sog. Peergruppen stattfinden soll und wie das Veröffentlichen und Lokalisieren von Ressourcen beschrieben wird (vgl. [90, 207]). Ein Peer kann u.a. ein Prozess, ein Gerät oder auch ein menschlicher Benutzer sein. Dabei arbeitet jeder Peer unabhängig von anderen Peers, muss aber durch eine eindeutige ID identifizierbar sein. Ein Peer veröffentlicht sogenannte Endpunkte, an denen es für die Kommunikation mit anderen Peers eine Schnittstelle anbietet. Dabei müssen Peers nicht direkt miteinander kommunizieren. Kommunikation kann auch über zwischengeschaltete Peers stattfinden, die die Nachrichten weiterleiten müssen. Ein zentrales Konzept von JXTA ist die Definition von Peergruppen. Hierdurch können Peers (spontan) gruppiert werden um miteinander zu kommunizieren und zeitlich beschränkt zu kooperieren. Hierbei beschreibt die Spezifikation nur, wie eine Gruppe gebildet, veröffentlicht und gefunden wird. Der eigentliche Mechanismus ist Sache der Implementierer eines *JXTA*-Netzwerkes. Peers sollen sich selbstständig gruppieren und können auch zeitgleich Mitglied in verschiedenen Peergruppen sein. Der Austausch von Nachrichten findet in Kanälen statt. Dabei ist ein Kanal ein asynchroner unidirektionaler Nachrichtentransfermechanismus und unterstützt den Transport beliebiger Daten (Anmerkung: dies können auch Java-Objekte sein). Da Kanäle unidirektional sind, wird zwischen Ausgabekanälen (output pipes) und Eingabekanälen (input pipes) unterschieden. Ein Peer muss somit mindestens zwei Endpunkte vorhalten können. Die Kanäle sind jedoch nur virtuell zu verstehen und haben keine physikalische Repräsentation. Die folgenden Eigenschaften werden von *JXTA* definiert:

Nachrichten (Messages): Nachrichten werden analog zu *KQML* oder *FIPA* mittels einem Datagramm verschickt. Hierbei besteht eine Nachricht aus einem „Umschlag“ mit Protokollkopfzeilen (wie Absender, Adressat, etc.) und dem eigentlichen Nachrichteninhalt.

Visitenkarten (Advertisements): In sog. Visitenkarten wird in einem XML Dokument die Eigenschaft von Peers, von Peergruppen und von Kanälen beschrieben. Hat ein Peer die Visitenkarte eines anderen Peers erhalten, kann er die darin enthaltene Beschreibung zur direkten Kommunikation und zur Sendung von Steuerungsnachrichten verwenden.

Identifier: Peers, Peergruppen und Kanäle müssen durch eindeutige Bezeichner gekennzeichnet sein.

Protokolle: Mittels des *Peer Discovery Protocol* kann jeder Peer seine eigenen Dienste beschreiben. Das Gegenstück davon ist das *Peer Resolver Protocol* mittels dessen ein Peer eine generische Anfrage stellen kann um einen geeigneten Peer für einen gesuchten Dienst zu finden. Das *Peer Information Protocol* erlaubt es, Statusinformationen von jedem angeschlossenen Peer zu erhalten. Diese drei semantischen Protokolle der Dienstbeschreibungen und Dienstaufrufe werden ergänzt durch das *Pipe Binding Protocol*, welches die virtuellen Kommunikationskanäle zwischen verschiedenen Peers aufbaut, dem *Endpoint Routing Protocol*, welches verwendet werden kann, um Wege zu Netzwerkschnittstellen anderer Peers zu finden (z. B. im Falle zwischengeschalteter Firewalls oder unterschiedlicher grundsätzlicher Netzwerkprotokolle). Zuletzt definiert das *Rendevous Protocol*, wie Nachrichten an alle Mitglieder derselben Peergruppe versendet werden können (z. B. Anwendung von Multicast und Broadcast).

JXTA definiert mächtige Protokolle zum Auffinden und Lokalisieren von Diensten. Vor allem die Gruppierung von Geräten in unterschiedlichen Gruppen ist neu. Es fehlen jedoch bisher völlig eine versprochene plattformunabhängige Implementierung (bei [132] findet sich eine Beispielimplementierung, die dieselben Nachteile wie *Jini* aufweist – völlige Abhängigkeit von der Programmiersprache Java durch serialisierten Objektaustausch), sowie eine Beschreibung von Geräten. *JXTA* bietet die Möglichkeit Geräte und Dienste zu beschreiben. Eine Standardisierung wie bei *UPnP* fehlt jedoch völlig, so dass unterschiedliche *JXTA*-Netzwerke in der Zukunft wohl nicht zueinander kompatibel sein werden. Konfliktlösungsmechanismen sowie die Selbstorganisation von Geräten wird völlig dem Implementierer einer *JXTA*-Komponente überlassen bzw. dem Benutzer selber – als letzte Instanz der Kontrolle eines Peers. Insofern kann *JXTA* tatsächlich nur die reine Middleware-Aufgabe der Netzwerkschicht übernehmen, die dann die Basis für intelligentere Mechanismen der Selbstorganisation stellt.

3.12.4 HAVi

Im Jahr 1997 trafen sich acht führende Hersteller (u.a. Grundig, Mitsubishi) von Unterhaltungselektronik um einen Branchen-Standard für die Interoperabilität von Geräten zu schaffen. Die erste Spezifikation des *Home Audio Video Interoperability (HAVi)* [96] erschien im Jahre 1999 und wurde 2001 von der bis jetzt geltenden Version 1.1 abgelöst. Auf Basis von IEEE 1394 (Firewire) und IEC 61883 sollte ein Standard für die Vernetzung und Kommunikation von Geräten geschaffen werden. Dabei kann *HAVi* in einer

beliebigen Programmiersprache auf jedem Betriebssystem implementiert werden. *HAVi* spezifizierte einen Satz an APIs (Application Programming Interfaces), die die Entwicklung von interoperablen HAVi-Geräten ermöglichen sollte, sowie eine Spezifikation von Java-Applikationen, die die direkte Interaktion des Benutzers mit den angeschlossenen Geräten erlaubt. In *HAVi* wird zwischen Kontrollgeräten und Zielgeräten unterschieden, die in vier Kategorien unterteilt werden:

Full AV Device: ein komplettes AV-Gerät, das die vollständige *HAVi*-Middleware und eine Java-Laufzeitumgebung (inklusive graphischer Benutzeroberfläche) implementiert. Beispiele hierfür wären Set-Top-Boxen.

Intermediate AV Device: ebenso ein komplettes AV-Gerät, jedoch ohne Java-Laufzeitumgebung.

Base AV Device: das sind zu kontrollierenden Zielgeräte, die eine Software zur Ansteuerung mittels Java-Bytecode bereitstellen (z. B. ein Videorecorder, ein Radio).

Legacy AV Device: das sind die derzeitigen Geräte der Unterhaltungselektronik, die ein Gateway bzw. eine Schnittstellen benötigen, um mit anderen *HAVi*-Geräten zu interagieren.

HAVi definiert ebenso unterschiedliche Software-Elemente, die die nötigen Netzwerkdienste, Systemdienste, Gerätedienste und die Dienste für graphische Benutzeroberflächen bereitstellen. Diese unterscheiden sich in ihren Prinzipien nicht von den hier auch besprochenen anderen Middleware-Technologien wie *UPnP*, *Jini* oder *JXTA*, so dass sie hier nicht im Detail aufgezählt werden sollen. Heute ist *HAVi* eher bedeutungslos geworden. Viele Gerätehersteller haben sich aus der Anwendung dieses Standards zurückgezogen. Die Gründe hierfür sind relativ offensichtlich: Noch immer bieten viele Geräte keine Java-Laufzeitumgebung, die Komplexität der *HAVi*-Spezifikation ist sehr groß und dadurch sehr restriktiv und bis heute fehlen Geräte, die diesen Standard tatsächlich implementieren. Insofern bietet *HAVi* dieselben Nachteile der Ansätze von *Jini* – völlige Java-Abhängigkeit – ohne die Vorteile von *UPnP* – die rein semantische Beschreibung von Diensten – zu bieten.

3.13 Zusammenfassung

Für eine abschließende Analyse der beschriebenen Software-Infrastrukturen und Middleware-Lösungen soll der Begriff Middleware im Sinne von Emmerich [64] verwendet werden. Emmerich definiert eine Middleware als eine spezifische Software-Infrastruktur, die die Interaktion von verteilten Software-Modulen möglich macht. Sie ist architektonisch eingegliedert zwischen dem Betriebssystem einen Gerätes und der Applikationsschicht, welche die Logik und Funktionalität einer Komponente oder eines Gerätes definiert. Zugleich ist die Middleware verantwortlich für die grundlegenden Kommunikationsmechanismen in so einem verteilten Netzwerk. Eine Middleware ist somit in der Lage auch

ein heterogenes Geräteensemble, d.h. Geräte auf Basis unterschiedlicher Betriebssysteme und Kommunikationsprinzipien, zu vernetzen und miteinander interagieren zu lassen. Gemäß dieser Definition entsprechen die Resultate des *AMIGO*-Projektes exakt den Anforderungen einer Middleware. Oberhalb einer (physikalisch wirkenden) Middleware sind die Funktionen einer Software-Infrastruktur angesiedelt, die das Auffinden von gewünschten Diensten, das Publizieren eigener Dienste sowie die eigentliche Kommunikation zwischen den verteilten Komponenten möglich machen. Ein Beispiel für diese Art der Arbeitsteilung ist im Kapitel 3.9 in der Analyse der *3PC*- und *BASE*-Architektur beschrieben. Typische Vertreter dieser reinen Form der Softwarelösungen sind die *Open Agent Architecture*, *Metaglue* oder die *Multiplatform*. Die in Kapitel 3.12 besprochenen Technologien sind gemäß der Middleware-Definition von Emmerich aufgrund ihrer technologischen Einschränkungen nur bedingt als Middleware-Technologien zu identifizieren. *Jini* und *JXTA* sind zu sehr auf die Programmiersprache Java zugeschnitten (*HAVi* zeigt diese Eigenschaft in der tatsächlichen Realisierung auch), während *UPnP* darauf abzielt einen eigenen Standard zu definieren. Aufgrund ihrer Anlage und ursprünglichen eigenen Spezifikationen werden sie dennoch als Middleware-Technologien verstanden.

Oberhalb der reinen Middleware-Schicht finden sich unterschiedliche Ansätze für die Kommunikation, für die Publizierung der Dienstebeschreibung, das Auffinden von Diensten, die Unterstützung von Kommunikation und die Herangehensweisen für die Ermöglichung von Dynamik und damit nötige Strategieanwendung im Falle konkurrierender Komponenten oder Ambiguitäten. Laut Herzog et al. [117] sind hier Techniken der Methodenaufrufe entfernter Objekte (remote procedure call bzw. remote method invocation, vgl. die Implementierung von RMI in der Programmiersprache Java [127]) zunehmend abgelöst worden von der asynchronen Kommunikation die auf dem Austausch semantischer Nachrichten beruht. Dies führt zu einem erhöhten Maß an Flexibilität aber auch zu vermehrter Autonomie der beteiligten Komponenten. Die Definition von Agentenkommunikationssprachen wie *KQML* oder *FIPA* (vgl. Kapitel 3.10) ohne jede Festlegung auf eine etwaige Implementierung war hierzu der nächste konsequente Schritt. Zwei grundsätzliche Schemata der Nachrichtenkommunikation können jedoch unterschieden werden (vgl. [216]). Die Punkt-zu-Punkt-Kommunikation von einem Absender an einen ihm bekannten Empfänger. Dies ist die herkömmliche Vorgehensweise in Multiagentensystemen aber auch beim Aufbau von Anwendungen auf der Basis verteilter Komponenten. Typische Vertreter sind die Spezifikationen von *FIPA* und *KQML*, die im Nachrichtenprotokoll immer einen Verweis auf den Absender und Empfänger haben, die Realisation des *Intelligent Rooms* auf der Basis der „Scatterbrain“-Software (vgl. Abbildung 22 rechts) und die Anwendung spezifischer Routingregeln in der *Open Agent Architecture* und dem *Galaxy Communicator*. Das zweite bekannte Schema der Nachrichtenkommunikation ist der publish-subcribe-Mechanismus, in dem sich Komponenten, die sich für bestimmte Nachrichten interessieren, an einer bestimmten – meist zentralen – Stelle für diese Nachricht registrieren (subscribe) und diese dann bekommen, sobald ein Sender eine solche Nachricht an diese zentrale Registrystelle versendet hat (publish). Dies ähnelt somit dem Event-Mechanismus, der in modernen objekt-orientierten Sprachen (z. B. Java, C++) verwendet wird. Häufig wird diese Form der Kommunikati-

on auch als *Blackboard*-Architektur bezeichnet. Die ersten Formen dieser Art entstammen dem Themenfeld der Künstlichen Intelligenz (vgl. das Hearsy-II-System bei Erman et al. [69]). In den Middleware-Lösungen und Software-Infrastrukturen, die in diesem Kapitel besprochen werden, finden sich reine publish-subscribe-Ansätze nur zu einem Teil. *KQML* und *FIPA* definieren solche Sprechakte, die dann einem zentralen Facilitator bzw. Router voraussetzen. Beide Kommunikationsmechanismen sind in dynamischen Umgebungen mit wechselnden Ensemblemitgliedern vorsichtig einzusetzen. Die *Punkt-zu-Punkt-Kommunikation* scheitert spätestens in dem Moment, wenn ein einzelnes Kommunikationsmitglied das Ensemble verlässt. Neuen Ensemblemitgliedern ist es beinahe unmöglich, an einer Kommunikation von Komponenten teilzuhaben[33]. Der reine *publish-subscribe-Mechanismus* ist in dynamischen Ensembles ebenfalls problematisch. Abgesehen von den hier einzusetzenden zentralen Komponenten, die die Nachrichtenbehandlung verwaltet, kann es hier zu konkurrierenden Nachrichtenverarbeitungen kommen. So sind zum Einen „bösartige“ Komponenten denkbar, die Nachrichten im Widerspruch zu bereits vorhandenen Komponenten verarbeitet, z. B. eine Komponente die Geräte an- statt ausschaltet und umgekehrt. Es sind aber auch Komponenten denkbar, die exakt dasselbe Verhalten aufweisen wie bereits vorhandene Komponenten. Hier wäre dann z. B. das doppelte Aufnehmen von Fernsehfilmen oder das Abspielen einer Präsentation vorstellbar.

Eine ganz ähnliche Art wie die Herausforderung des Versendens einer Nachricht an passende Konsumenten liegt in der Problemstellung einer Komponente einen passenden Diensteanbieter zu finden. Für die Problematik des Service Discovery haben sich zwei grundsätzliche Ansätze durchgesetzt. Im *aktiven Service Discovery* – requester pull – kontaktiert eine Komponente einen (zentralen) Dienst und erfragt von diesem die Bezeichner möglicher Diensteanbieter. In einer dezentralen Version sendet die anfragende Komponente einen Multicast aus und wartet auf die Antworten aller Angesprochenen. Das *passive Service Discovery* – provider push – folgt einem ganz ähnlichen Schema. Diensteanbieter informieren einen (zentralen) Dienst, der daraufhin alle ihm bekannten Komponenten über die angebotenen Dienste und den Diensteanbieter informiert. Alternativ sendet ein Diensteanbieter die Informationen über sich und die vom ihm angebotenen Dienste per Multicast an alle vorhandenen Knoten aus.

Neben der Fragenstellung der Handhabung später hinzukommender Komponenten (hier: Wie informieren sie sich über bereits vorhandene Dienste bzw. werden informiert) stellt sich in allen Herangehensweisen die Frage, wie Komponenten mit einer Liste von möglichen Diensteanbietern umgehen. Oder anders ausgedrückt: Auf welche Art und Weise kann ein Dienst an einem möglichen Diensteanbieter gebunden werden? Oreizy et al. haben hierzu die drei prinzipiellen Adaptionsmodelle für diese Fragestellung des *Service Bindings* im Falle konkurrierender Geräte bzw. Komponenten aufgestellt (vgl. [167]): Bei der *manuellen Adaption* nimmt der Benutzer die letztendliche Auswahl vor. Hier präsentiert das System dem Benutzer das gefundene Gerät bzw. die gefundenen

[33] Dies ist nur dann denkbar, wenn der Programmierer einer neuen Komponente die Bezeichner der bereits vorhandenen Komponenten kennt. Dann kann die von ihm entwickelte Komponenten aktiv kommunizieren und gegebenfalls – abhängig von der internen Implementation der bereits vorhandenen Komponenten – Antworten erhalten.

Geräte zur Auswahl. Somit hat der Benutzer als letzte Entscheidungsinstanz die Aufgabe des Service-Bindings. Diese Herangehensweise wird in den in diesem Kapitel vorgestellten Middlewaretechnologien *UPnP*, *HAVi* und *Jini*, aber auch durch die Software *iCrafter* im *Interactive-Workspaces*-Projekt verfolgt. In dynamischen Systemen (oder in Systemen mit versteckten / unbekannten Komponenten) scheint diese Vorgehensweise jedoch aufgrund der hohen mentalen Belastung für den Benutzer nicht praktikabel. Bei der *anwendungsspezifischen automatischen Adaption* nehmen die Applikationen entsprechend des Szenarios die Auswahl aufgrund ihrer Implementation vor. Beispiele für diese Art des Vorgehensmodells sind die Regeln der Rule Engine in *Easy Living*, die Implementation der Kommunikationswege im *Intelligent-Room*-Projekt, die Regeln im *Galaxy-Communicator*-HUB, aber auch die Spezifikation eines *Enhanced Service Discovery* in der *AMIGO*-Middleware. Für die Agentenkommunikationssprache KQML besteht das Subset COOL [25], welches Sprechakte für etwaige Verhandlungen von zwei Komponenten definiert. In dieser Herangehensweise ist somit der Entwickler einer Komponente für die Konzeption und Implementation möglich gut geeigneter Strategien zum Service-Binding verpflichtet. Das letzte der drei Adaptionsmodelle ist die *generische automatische Adaption*. Hier werden die funktionalen Eigenschaften eines Dienstes ohne Angabe eines Diensteanbieters beschrieben, während die Zugrunde liegende Software-Infrastruktur daraufhin in der Lage ist, einen möglichen Diensteanbieter (und/oder Nachrichtenkonsument) zu bestimmen. In dieser Form der Adaption wird somit das sog. *Service Binding* von der Infrastruktur übernommen. Diese Art des Service-Bindings findet sich in den Komponenten der *Galaxy Communicator Infrastruktur*, den Komponenten der *Open Agent Architecture*, in *Jaspis* und *DIANE*. Die Spezifikation der *Super Distributed Objects* bildet hier einen Sonderfall, da je nach Implementation der *Service-Logic*-(SL)-Objekte zwischen den beiden zuletzt genannten Adaptionsmodellen differenziert werden kann. Gibt es von SDOs getrennte Service-Logic-Objekte, so hat man es mit zentralen Kommunikationsroutern zu tun, die eine stark vereinfachte Implementation der SDOs möglich machen. Dies jedoch auf Kosten zentraler SL-Objekte. Besitzt jedes SDO ein SL-Objekt so würde dies dem *AMIGO*-Ansatz ähneln. Jeder Entwickler einer SDO-Komponente ist damit wiederum für die Implementation geeigneter Service-Bindung-Strategien verantwortlich.

Werden nun die Eigenschaften der hier beschriebenen Lösungen für Software-Infrastrukturen und Middleware-Technologien mit den in Kapitel 2.3 definierten Anforderungen für selbstorganisierende Ensembles verglichen so findet sich, dass vor allem der Punkt der Dezentralität ein schwierig umzusetzender ist. Sowohl die *Multiplatform*, die Implementation von *Easy Living*, die *Open Agent Architecture* als auch der *Galaxy Communicator* basieren auf zentralen ausgezeichneten Komponenten. In Galaxy ist zudem durch die Anlage der Regeln ein definiertes Komponentenensemble zur Laufzeit nicht erweiterbar. Dies gilt auch für die Multiplatform. Die Komponenten der Open Agent Architecture sind zur Laufzeit erweiterbar, hierbei wird das Routing der Nachrichten nicht statisch durch Regeln bestimmt, sondern durch Regelmechanismen die in sog. Meta-Agenten implementiert sind. Hierdurch ist Flexibilität des Komponentenensembles zur Laufzeit gegeben, jedoch stellen auch die Meta-Agenten eine Form von zentralen Komponenten dar,

die gegebenenfalls im Erweiterungsfall zusätzlich ergänzt werden müssen. Ganz analog gelten diese Eigenschaften auch für die Evaluator-Komponenten der *Jaspis*-Architektur. Im *DIANE*-Ansatz stellt der Regelmechanismus auf Basis von Persönlichkeitsprofilen eine solche zentrale Komponente dar. *AMIGO* und die *Super Distributed Objects* sind komplett dezentral spezifiziert (im Falle der SDOs gilt dies, falls jedes SDO über eine Service-Logic-Komponente verfügt). Hier können die Eigenschaften Dezentralität, Austauschbarkeit, Autonomie und Erweiterbarkeit als gegeben vorausgesetzt werden. Ein Sonderfall stellt die *PCOM*-Infrastruktur dar, die ebenso aus dezentralen Einheiten, den Containern, besteht. Kommunikationswege sind hier jedoch statisch in Form von Anwendungen beschrieben, so dass der Punkt Autonomie hier in Frage gestellt werden muss. Die Frage der Konfliktlösung im Falle konkurrierender Diensteanbieter wird aber nicht abschließend gelöst. Die hierfür nötigen Mechanismen sind in jeder einzelnen Komponente zu implementieren. Vorschriften hierfür sind nicht gegeben. Es liegt somit in der Verantwortung jedes Software-Entwicklers für die von im entwickelten Komponenten geeignete Strategien zu implementieren. Die Anforderung der einfachen Benutzbarkeit und der schnellen Integrierbarkeit von bereits vorhandenen funktionalen Komponenten ist hier dann nicht gegeben. Der wichtige Punkt der Benutzbarkeit ist ein starkes Argument der *INCA*-Infrastruktur. Hier sind klare Schnittstellen vorhanden, um leicht Komponenten zu programmieren (vgl. [202]). Leider ist der Anwendungsbereich dieser Infrastruktur sehr begrenzt. Die einfache Bedienbarkeit macht es aber zu einer wichtigen Referenzimplementation.

Resümee: Nach der Analyse der hier beschriebenen Middleware-Lösungen und Software-Infrastrukturen würde eine Software-Infrastruktur den Anforderungen entsprechen, die die Flexibilität und Dezentralität der AMIGO-Architektur aufweist. Zugleich müsste sie über eingebaute Konfliktlösungsstrategien zum Service-Discovery und Service-Binding verfügen, was den Entwickler von Softwarekomponenten maximal entlasten würde. Das Absenden (und Empfangen) von Nachrichten sollte daher so gestaltet sein, wie in den Lösungen die auf einer zentralen regelbasierten Routerkomponente aufbauen. Dieser offenbare Widerspruch wäre dadurch aufzulösen, indem jede Komponente auf einer Software-Infrastruktur aufbauen würde (vgl. die AMIGO-Architektur in Abbildung 15 und die interne Architektur in PCOM in Abbildung 27), die die Fähigkeit der verteilten Konfliktlösung auf Basis gleicher Strategien mit sich bringen würde. Diese Software-Infrastruktur müsste dann verteilt auf jedem physikalischen Gerät laufen.

4 Das SODAPOP-Modell für selbstorganisierende Ensembles

Das Projekt EMBASSI [114, 115][34] hatte sich mit 19 Partnern aus Industrie, Wissenschaft und angewandter Forschung zum Ziel gesetzt, Assistenztechnologien für Alltagsgegenstände zu entwickeln und technologisch in Alltagsgegenstände zu integrieren. In diesem Projekt wurde der Schwerpunkt auf drei verschiedene Anwendungsszenarios in den Bereichen Unterhaltungselektronik, Automobile und Terminalsysteme gesetzt. Schon die hohe Anzahl der unterschiedlichen Partner und die ausgewählten Szenarios bestimmten einige Anforderungen an die zugrunde liegende Software-Infrastruktur. Sie sollte vor allem die dynamische Integration von Komponenten zu einem Ensemble unterstüzen, sowie die Kommunikationsmöglichkeiten der Komponenten untereinander garantieren können. Besonders im Bereich der Unterhaltungselektronik[35], wo Benutzer in der Lage sein sollten, Geräte auch von verschiedenen Herstellern miteinander zu verbinden, ist diese Art der Dynamik erforderlich. Für eine Software-Infrastruktur, die die in EMBASSI definierten Szenarios unterstützt, gelten die folgenden Eigenschaften (siehe auch [97, 104, 105, 110]):

Unabhängigkeit der Einzelkomponenten: Die Fähigkeiten von Komponenten sollten nicht von der Anwesenheit anderer Komponenten abhängig sein. Weniger streng formuliert bedeutet dies, dass Basisfunktionalitäten von Geräten in einem Ensemble dem Benutzer immer bereit stehen sollten.

Dynamische Erweiterbarkeit: In ein bereits bestehendes Ensemble von Geräten bzw. Komponenten sollten sich ad-hoc zur Laufzeit neue Geräte bzw. Komponenten hinzufügen lassen (und auch entfernen lassen). Dies bedeutet, dass sich Komponenten dynamisch in den Kommunikationsfluss eines bereits bestehenden Ensembles integrieren lassen.

Vermeidung von zentralen Komponenten: Die Forderung nach dynamischer Ensemblebildung – genau dies bedeutet im Umkehrschluss die Forderung nach dynamischer Erweiterbarkeit[36] – schließt automatisch das Vorhandensein einer zentralen Komponente aus.

Unterstützung verteilter Implementation: Verteilte Realisierung ist eine unabdingbare Anforderung, die sich aus der Anforderung zentrale Komponenten zu vermeiden,

[34] EMBASSI – Abkürzung für Elektronische multimediale Bedien- und Service-Assistenz (Multimodal Assistance for Infotainment & Service Infrastructures) – war eines von 6 Leitprojekten im Rahmen des Forschungsthemas *Mensch-Technik-Interaktion* des Bundesministeriums für Bildung und Forschung (BMB+F). Projektzeitraum: Juli 1999 - Juli 2003.

[35] In EMBASSI waren die Firmen Grundig (zusammen mit dem Fraunhofer-Institut für Graphische Datenverarbeitung (Fh-IGD) Konsortialführer des Projektes), die Loewe Opta GmbH und Sony beteiligt.

[36] Dynamische Erweiterbarkeit bedeutet auch den Sonderfall, dass ein Komponentenensemble mit null Komponenten beginnt und sich ad-hoc aus Einzelkomponenten bildet.

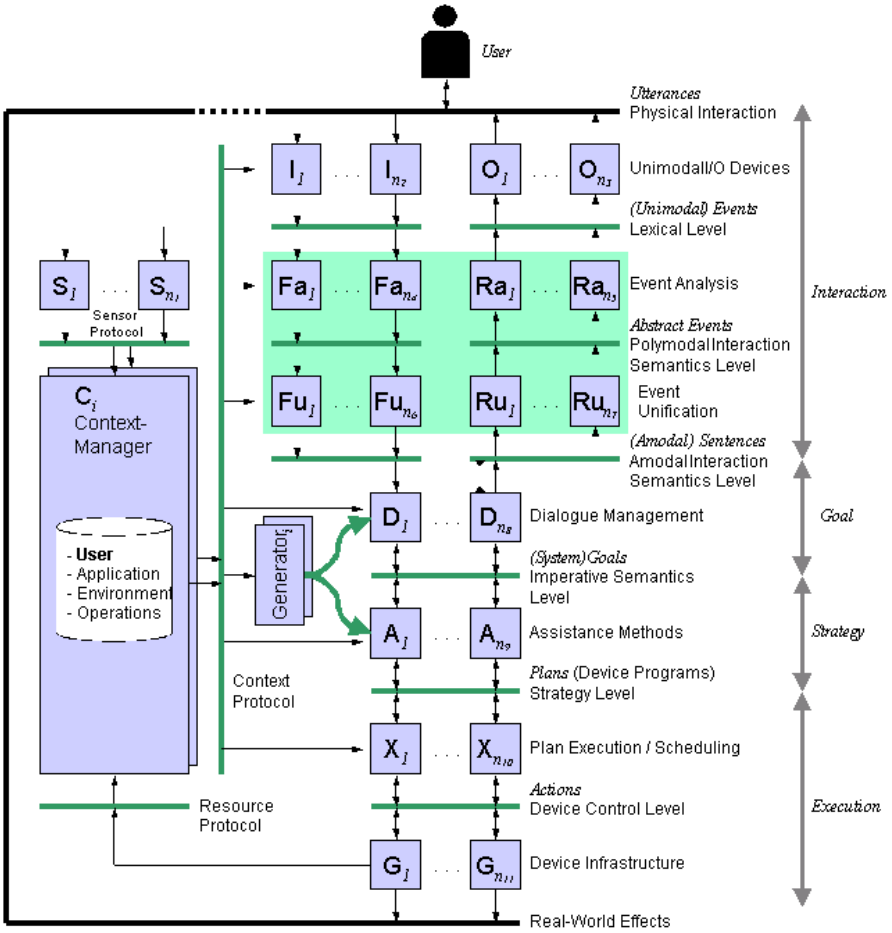


Abbildung 37: Die generische EMBASSI-Topologie. Benutzeräußerungen werden in Ziele interpretiert (oberer Bereich), diese Ziele werden in Strategien zur Änderung der Umgebung umgewandelt. Letztlich werden Funktionen gefunden, die Gerätevariablen manipulieren.

ergibt. Gibt es keine zentralen Komponenten, die den Datenfluss und die Kooperation von Komponenten verwalten, so muss die zugrunde Software-Infrastruktur verteilt sein. Die verteilte Implementation einer solchen Software-Infrastruktur ist Voraussetzung für die Selbstorganisation der beteiligten Komponenten.

Wiederverwendbarkeit von Komponenten: Komponenten müssen sich aus einem Ensemble entfernen lassen können, wieder integrieren lassen können, bzw. auch in anderen Ensembles wieder verwenden lassen können.

Austauschbarkeit von Komponenten: Der Austausch von Komponenten im laufenden System sollte immer möglich sein. Voraussetzung hier ist eine klare Definition des Funktionsumfang der betroffenen Komponenten und damit verbunden eine klare Schnittstellendefinition.

Neben diesen Anforderungen, die sich aus den dynamischen Szenarios – dynamische Ensemblebildung aus Komponenten unterschiedlicher Hersteller – ergaben, war das zweite wichtige Ziel des Projektes EMBASSI der Metaphernwechsel des Umganges des Menschen mit technischen Infrastrukturen. Nicht mehr die funktions-orientierte Interaktion mit von Geräten angebotenen Funktionen, sondern die ziel-orientierte Interaktion sollte im Mittelpunkt der Mensch-Technik-Interaktion stehen. Dies bedeutet auch die Überwindung von uni-modalen menü-basierten Interaktionsmetaphern zu multimodalen Dialogstrukturen. Die generische EMBASSI-Komponententopologie in Abbildung 37 verdeutlicht diesen Ansatz. Diese Topologie verfolgt den Ansatz des Datenflusses. Ausgehend von Äußerungen des Benutzers (bzw. seinen Eingaben) werden mögliche Benutzungsziele interpretiert und mittels vorhandener Gerätefunktionen umgesetzt. Jede Ebene der Topologie repräsentiert eine Verarbeitungsstufe innerhalb des EMBASSI-Datenflusses. In Abbildung 37 repräsentiert ein Kästchen eine funktionale Komponente, während ein Bus einen „Austauschplatz“ symbolisiert, in dem Nachrichten gleichartiger Ontologie[37] zwischen den Komponenten kommuniziert werden. Die EMBASSI-Topologie (vgl. [97]) besitzt hier Ähnlichkeiten mit dem ARCH-Modell [26, 27, 144] für Mensch-Maschine-Interaktion. Deren Ebenen – *Interaction Toolkit*, *Presentation Component*, *Dialogue Component*, *Domain Adaptor Component* und *Domain Specific Component* – lassen sich entsprechend auf die *I/O*-Ebene für Eingabe und Ausgabe, die *F/R*-Ebene für Datenfilterung, die *D*-Ebene für Dialogmanagement, die *A*-Ebene für Assistenz und Strategie und die *X bzw. G*-Ebene für die Gerätefunktionen abbilden.

4.1 Die SODAPOP-Grundlagen

Im Kontext des Projektes EMBASSI wurde unter Berücksichtigung der oben formulierten Anforderungen die Prinzipien einer selbstorganisierenden Software-Infrastruktur definiert[38]. Das SODAPOP-Modell [97, 136] – für: Self-Organizing Data-flow Architectures suPporting Ontology-based problem decomPosition – bildete dann die Grundlage für die im Projekt DYNAMITE entwickelte verteilte Software-Infrastruktur für dynamische Komponentenensembles [106, 107, 108]. Die Realisierung wird in den folgenden Kapiteln 5 und 6 besprochen. Das SODAPOP-Modell basiert auf der Annahme, dass eine Software-Infrastruktur für die Selbstorganisation von Komponenten und Geräten die internen Verarbeitungsprozesse der Komponenten innerhalb eines Gerätes berücksichtigen und unterstützen muss. Abbildung 38 soll mit einem virtuellen Blick in das „Innenleben“ eines typischen Unterhaltungsgerätes die grundlegenden Ideen des SODAPOP-Modells illustrieren. Die meisten Geräte verfügen zumindest über eine physikalische Verbindung zur Außenwelt, das heißt zu ihrer direkten Umgebung auf deren Eingaben bzw. Änderun-

[37] Ontologie wird hier verstanden als ein Satz an semantischen Objekten, die auf den entsprechenden Ebenen ausgetauscht werden können.

[38] Tatsächlich liefen die Arbeiten an den Komponenten und den Demonstratoren sowie die Definition des hier beschriebenen Modells einer selbstorganisierenden Software-Infrastruktur zeitlich parallel. Die generische EMBASSI-Topologie wurde auf Basis eines peer-to-peer-Kommunikationssystems unter Verwendung der Kommunikationssprache KQML [145] realisiert. Hierzu wurde ein eigenes entwickelter Kommunikationsrouter auf Java-Basis implementiert [83].

gen sie mit der Ausführung von Funktionen reagieren. Hierbei handelt es sich meistens um eine Schnittstelle zum Benutzer (z. B. über graphische Bedienoberflächen, Schalter oder Fernbedienungen) oder um Sensoren, die bestimmte Variablen der Umgebung beobachten (z. B. Licht- oder Bewegungssensoren). Abbildung 38 illustriert den angenommenen abstrahierten Datenfluss innerhalb eines Gerätes. Benutzerschnittstellen nehmen die Interaktionen direkt auf und reichen diese als Ereignisse an die darunter liegende Verarbeitungsschicht weiter. Somit werden physikalische Ereignisse in Ereignissignale umgewandelt. Diese werden von einer (oder mehreren) Verarbeitungsebenen innerhalb von Steuerapplikationen verarbeitet, die daraus physikalische Funktionsaufrufe ableiten, die tatsächlich von Effektoren ausgeführt werden. Ohne Beschränkung der Allgemeinheit kann angenommen werden, dass offensichtlich alle bekannten Geräte über eine solche prinzipielle Art der Datenverarbeitung verfügen (Anmerkung: auch wenn es in gewissen Fällen recht trivial sein kann, man denke z. B. an eine Lampe, die per Schalter aktiviert werden kann, und deren Steuerungsapplikationen aus einem simplen Draht bestehen kann). Es ist somit möglich, in physikalischen Geräten sowohl *Typen an Komponenten* zu identifizieren und zu definieren als auch *Verbindungsebenen* zwischen Teilkomponenten zu bilden.

Abbildung 39 oben illustriert, wie sich somit Ebenen der Datenverarbeitung gleicher Art zwischen unterschiedlichen Geräten einziehen lassen, die damit auch geräte-übergreifende Schnittstellen definieren, auf denen Nachrichten gleicher Art publiziert werden (siehe Abbildung 39 unten). Dieses Vorgehen macht es möglich, dass Ereignisse einer Benutzerschnittstelle auch von weiterführenden Applikationen anderer Geräte interpretiert werden können. Die Voraussetzungen für die Kooperation von Komponenten unterschiedlicher physikalischer Geräte und damit verbunden die Selbstorganisation von ad-hoc Ensembles sind somit durch diesen Ansatz gegeben. Es ist zum Beispiel eine Dialogkomponente vorstellbar, die auch die Ereignisse anderer Geräte verstehen kann, oder eine Kontrollkomponente, die auch in der Lage ist mehrere Geräte gleichzeitig anzusteuern. Indem diese internen Schnittstellen zu öffentlichen Schnittstellen werden, ist eine architektonische Gesamtintegration möglich.

Das SODAPOP-Modell führt somit zwei fundamentale Organisationsebenen [97] ein:

- grob-gekörnte Selbstorganisation, die auf dem Ansatz der Aufteilung des Datenflusses basiert
- fein-gekörnte Selbstorganisation für funktionsgleiche Komponenten unter Anwendung passender Strategien zur Zuteilung von Ereignissen und Daten

Die grob-gekörnte Art der Selbstorganisation besteht damit in der Identifikation von Ebenen gleichartiger Komponenten und deren Verbindung durch geräte-übergreifende Kanäle. Die fein-gekörnte Art der Selbstorganisation besteht in der Folge in den kanal-abhängigen Strategien in der Verarbeitung und Zuteilung von Nachrichten.

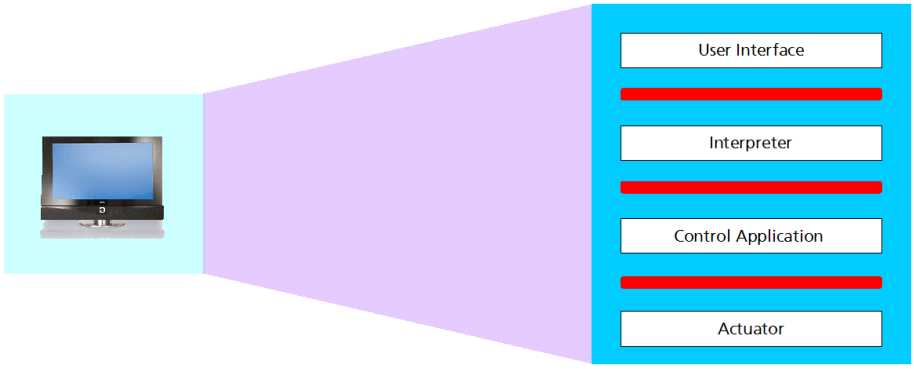


Abbildung 38: Illustration des „Innenlebens“ eines Fernsehgerätes. Es lassen sich unterschiedliche Typen an Komponenten identifizieren die gemäß eines Datenflusses zeitlich voneinander abhängige Teilaufgaben ausführen.

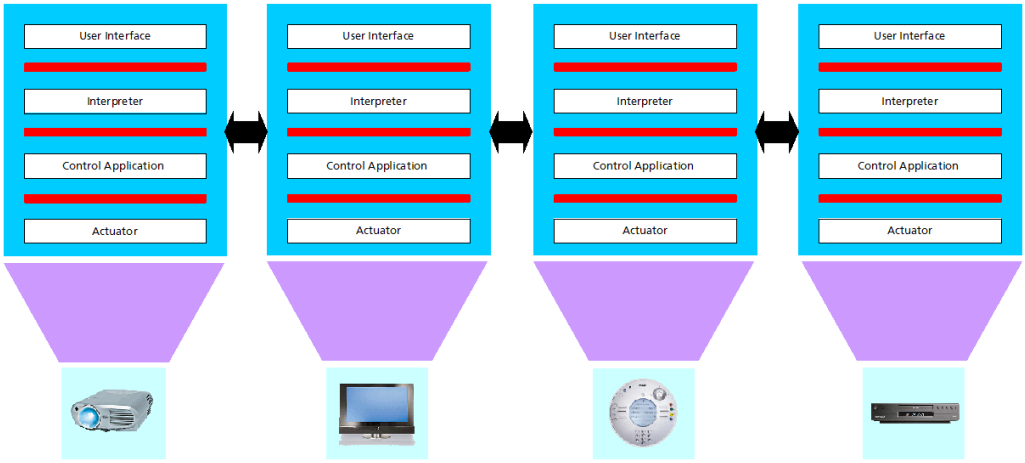
4.2 Komponententypen: Transducer und Kanäle

Wie Abbildung 39 nahelegt sind für die Selbstorganisation im SODAPOP-Modell zwei grundsätzlichen Arten an Komponenten definiert:

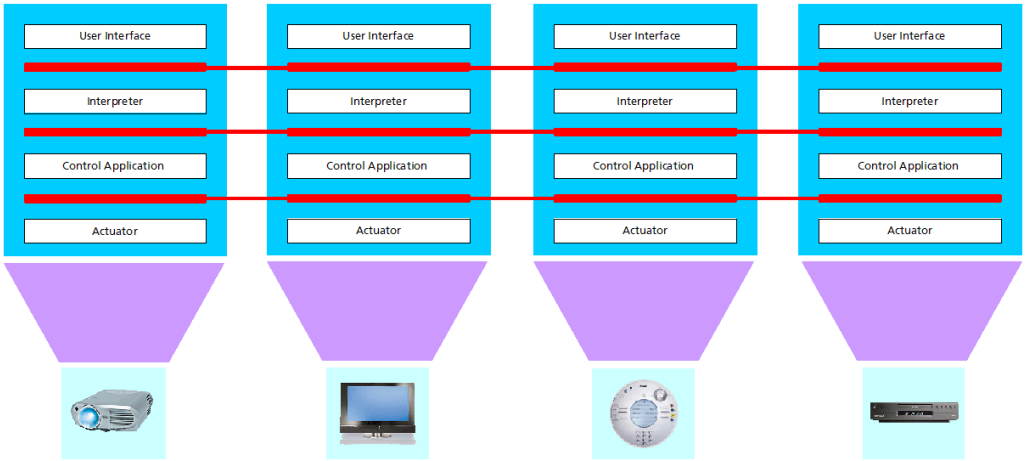
Kanäle (Channels) lesen eine Nachricht zu einem gewissen Zeitpunkt, bilden die Nachricht auf eine (oder mehrere) Nachrichten ab und weisen diese zu den angeschlossenen Komponenten der folgenden Verarbeitungsstufe zu. Kanäle besitzen keinen Speicher, sind verteilt und müssen jede Nachricht akzeptieren und weiterverarbeiten. Die Kanäle sorgen somit für eine räumliche Verteilung von Einzelereignisse an eine Vielzahl an Transducer. Kanäle arbeiten – konzeptuell – ohne Zeitverzögerung.

Transducer (Komponenten) sind die funktionalen Komponenten in einem SODAPOP-System. Sie können eine oder mehrere Nachrichten während eines Zeitintervalls lesen und diese auf eine (oder mehrere) Ausgabeereignisse abbilden. Transducer sind nicht verteilt, da sie per Definition an ein physikalisches Gerät gebunden sind. Sie können Speicherplatz belegen und müssen nicht jede Nachricht, die von den Kanälen publiziert wird, akzeptieren. Transducer können somit für eine zeitnahe Vereinheitlichung von mehreren Ereignissen zu einem einzelnen Ausgabeereignis sorgen. Im Allgemeinen können Transducer mehrere Eingabe- und Ausgabekanäle besitzen. Sie sind nicht wie die Transducer der *Benutzerschnittstelle* oder der *Steuerungsapplikationen* in Abbildung 38 an eine $1:1$ Kanalzuordnung gebunden.

Kanäle akzeptieren jede Art Nachricht, die von einem Typ t sind, Transducer aber wandeln Nachtrichten eines Types t in Nachrichten eines Types t' um. Ein Gesamtsystem ist somit definiert als ein Satz an Kanälen und einen Satz an Transducertypen. Kanäle arbeiten auf Nachrichten derselben Ontologie – *Kanäle sind Ontologietreu* – während Transducer Nachrichten von einer Ontologie in eine andere abbilden – *Transducer sind Ontologiewandler*. Kanäle werden identifiziert durch ihre Kanalbeschreibung. Diese Kanalbeschreibung sollte die verwendete Ontologie enkodieren, so dass Transducer die mit dieser Ontologie arbeiten, sich automatisch damit verbinden können.



Geräte mit gleichartiger interner Komponentenstruktur . . .



. . . bilden Kanäle für gemeinsame Nachrichtenverarbeitung aus.

Abbildung 39: Unterschiedliche physikalische Geräte mit gleichartigen Verarbeitungsebenen werden ad-hoc miteinander verbunden. Die Verarbeitungsebenen (Kanäle) bilden geräte-übergreifend Ebenen gemeinsamer Verarbeitungsstufen.

Mehrere Gründe sprechen für die klare Aufteilung in Transducer und Kanälen:

Vorhandene Applikationen: Die verschiedenen Applikationen der Verarbeitungsstufen von Ereignissen sind in den Geräten bereits vorhanden, sind daher streng lokalisiert und verfügen daher auch über die Möglichkeit Speicherplatz zu allokieren. Diese vorhandenen Applikationen sind somit – prinzipiell – leicht auf ein SODAPOP-System implementierbar.

Geräteübergreifende Datenverarbeitung: Kanäle verbinden die einzelnen Ebenen der Datenverarbeitung geräteübergreifend. Dies ist für (herkömmliche) Geräte neu. Daher müssen hier geeignete Strukturen geschaffen werden. Kanäle stellen dabei die eigentlichen Mechanismen der Selbstorganisation bereit.

Konfliktlösungsstrategien: Kanäle üben mittels bestimmter Kanalstrategien die *Intelli-*

genz des selbstorganisierten Geräteensembles aus. Wie die Illustration in Abbildung 39 vermuten lässt, kann es zu Konflikten von Transducern in der Bearbeitung von Nachrichten kommen. Diese Konflikte zu lösen und geeignete Empfänger für Nachrichten zu finden ist Aufgabe des Kanals. Hierfür wendet er Konfliktlösungsstrategien (allg. Kanalstrategien) an. Diese Strategien definieren die eigentlichen Fähigkeiten zur Selbstorganisation eines dynamischen Komponentenensembles. Diese Intelligenz muss verteilt ausführbar sein, um zentrale Komponenten zu vermeiden und damit auch unabhängig von der Intelligenz einzelner Applikationen und/oder Geräte zu sein. Die Intelligenz eines selbstorganisierten Systems lässt sich somit definieren als das Gesamtverhalten eines dynamischen Geräteensembles und ist damit abhängig von der Art der Zuweisung von Ereignissen an bestimmte einzelne Komponenten, d.h. von der Steuerung des Datenflusses.

Topologiebildung: Die Topologie eines Geräteensembles gibt die verschiedenen Stufen der Signalverarbeitung wieder. Das SODAPOP-Modell macht keine Voraussetzungen in Art und Anzahl der zu definierenden Kanäle und Transducertypen.

4.3 Kommunikationsmuster

Das SODAPOP-Modell unterscheidet zwischen zwei unterschiedlichen Kommunikationsmustern:

Events (Ereignisse): Ereignisse durchlaufen gemäß des Datenflusses die verschiedenen Kanäle und Transducer. Dabei erwartet ein Transducer auf ein gesendetes Ereignis *E* keine Antwort. Er kann vielmehr davon ausgehen, dass die anderen Systemkomponenten dieses Ereignis weiterverarbeiten. Ein Beispiel wäre ein Lichtsensor, der in regelmäßigen zeitlichen Abständen Informationen über die aktuelle Raumhelligkeit in einen Eingabekanal sendet.

Remote Procedure Calls: Wenn ein Transducer einen RPC benutzt, erwartet er darauf eine Antwort. Somit wird die spätere Verarbeitung der Ereignisse von diesem Transducer bestimmt. Ein Beispiel ist ein Dialogmanager, der mittels multi-modaler Ausgabegeräte einen Dialog mit dem Benutzer steuert.

Events und Remote Procedure Calls beschreiben unterschiedliche Semantiken des Durchlaufens von Nachrichten durch eine Topologie-Pipeline. Der Datenfluss in den hier abgebildeten Illustrationen in Abbildung 38 und Abbildung 39 von einer Benutzerschnittstelle hin zu Applikationen, die mittels Funktionen Gerätevariablen oder Umgebungsvariablen ändern, spiegelt eine typische Event-Pipeline wider. Das SODAPOP-Modell interpretiert einen Remote Procedure Call als ein aktives Nachfragen nach Ereignissen. Somit stellt ein Kanal, der Events verarbeitet, auch gleichzeitig einen inversen RPC-Kanal dar. Transducer, die Events auf einen Kanal publizieren können, sind somit die Adressaten von Remote Procedure Calls. Im Gegensatz dazu sind Transducer, die RPCs an einen Kanal absenden, die potentiellen Empfänger von Events. In Kapitel 5 werden diese Eigenschaften des SODAPOP-Modells für die verteile Implementierung im Detail ausspezifiziert und deren Verwendung diskutiert.

4.4 Zuteilung von Nachrichten

In einem dynamischen System können Nachrichten, wie Ereignisse und Remote Procedure Calls nicht mit einer spezifischen Adressinformation versendet werden. Es obliegt vielmehr der Verantwortung eines Kanals, an den eine Nachricht verschickt wird, mittels geeigneter Konfliktlösungsstrategien diese Nachricht geeignet zu zerteilen und/oder geeignete Adressaten für eine Nachricht (oder die Menge der zerteilten Nachrichten) zu finden.

Transducer müssen somit den Kanal an dem sie partizipieren mit einer gewissen Menge an Informationen versorgen. Im Einzelnen sind dies Informationen über:

- der Satz an Nachrichten, den er versteht und weiterverarbeiten kann (Sprachschatz)
- für jede Art der Nachrichten die Güte, mit der er diese Art Nachricht verarbeiten kann (Quality of Service)
- ob er in der Lage ist, die Nachricht parallel zu anderen Transducern zu verarbeiten (Parallelverarbeitung)
- und ob er in der Lage, die Nachricht in Zusammenarbeit mit anderen Transducern zu verarbeiten (Kooperationsfähigkeit)

Diese Informationen beschreibt ein Transducer beim Anmelden an einen Kanal mittels seiner *UtilityValue-Funktion*. Dies ist eine Funktion, die Nachrichten auf einen Wert abbildet, der eine Aussage darüber liefert, in welcher Güte der entsprechende Transducer diese Nachricht weiterverarbeiten kann. In welcher Art und Weise diese UtilityValue-Funktion kodiert wird und nach welchen Strategien der Kanal den passenden Empfänger einer Nachricht auswählt, ist abhängig von der Ontologie, die auf diesem Kanal verwendet wird. Die verteilte Implementation der Ausführung von Kanalstrategien wird in Kapitel 5 im Detail diskutiert und spezifiziert, während mögliche Kanalstrategien in Kapitel 6 definiert und diskutiert werden. Abbildung 40 zeigt die prinzipielle Vorgehensweise, wie eine Nachricht unter Verwendung der UtilityValue-Funktionen der angeschlossenen Transducer (Abbildung 40 a)) an einen Satz passender Adressaten von dem zuständigen Kanal verteilt wird. Nachdem im Kanal eine Nachricht zur Weiterverarbeitung eingegangen ist (Abbildung 40 b)), evaluiert er die UtilityValue-Funktionen der angeschlossenen Transducer in Bezug auf die Nachricht (Abbildung 40 c)) und entscheidet nach Ausführung der kanaleigenen Konfliktlösungsstrategie (Abbildung 40 d)) welcher Transducer (bzw. welche Transducer) die Nachricht letztlich zur Weiterverarbeitung erhalten (Abbildung 40 e)). Die Art und Weise, wie ein Kanal Nachrichten zerlegt und wie ein Kanal einen Satz an passenden Empfängern bestimmt, ist Aufgabe der Kanalstrategie. Es sind somit auch Kanalstrategien denkbar, die eine Nachricht in mehrere Nachrichten zu zerlegen, die dann parallel von mehreren Transducern verarbeitet werden. Kooperation von mehreren Transducern zur Bearbeitung desselben Auftrages scheint somit möglich.

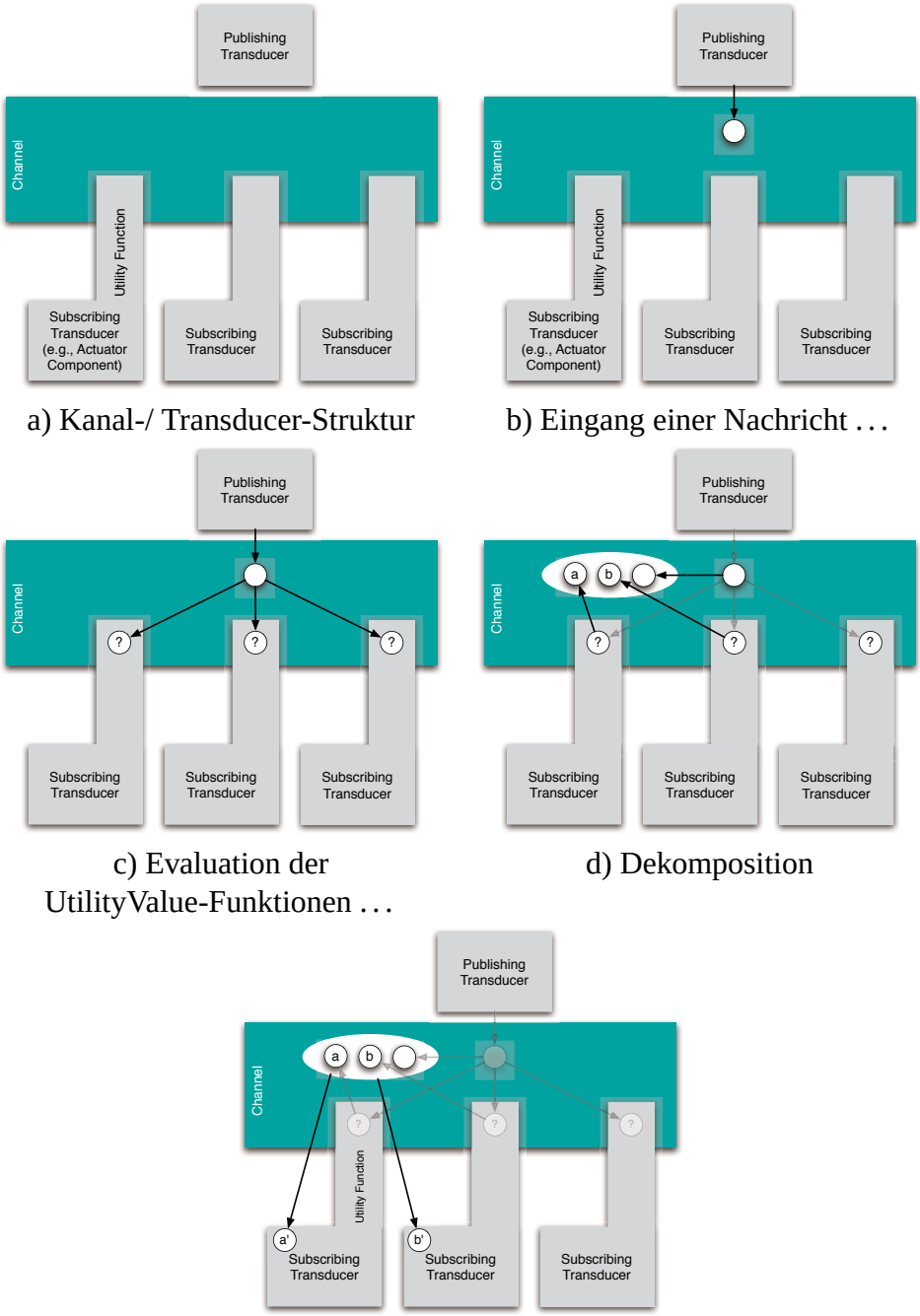


Abbildung 40: Prinzipielle Nachrichtenbehandlung in einem Kanal im SODAPOP-Modell.

4.5 Diskussion des SODAPOP-Modells

Verglichen mit anderen Software-Infrastrukturen, die die Kommunikation von heterogenen Komponentenensembles unterstützen (siehe Kapitel 3) führt das SODAPOP-Modell einige neue Prinzipien ein. Obwohl der Ansatz des „pattern-matchings“ der in der Aus-

wertung der UtilityValue-Funktionen innerhalb der Kanäle zum Tragen kommt nicht neu ist (ähnliche Konzepte finden sich in der Open Agent Architecture [154] und der Galaxy Infrastruktur [88, 190] aber auch schon in frühen Arbeiten zu Prolog [41] oder dem in [203] definierten *Pattern-Matching Lambda Calculus*) bietet das SODAPOP-Modell wesentliche Änderungen. Es ergänzt das statische Auswerten von Tabellen durch Routing-Komponenten (in denen sich Komponenten für Nachrichten mittels publish-subscribe-Mechanismen registriert haben) durch die dynamische zeitnahe Auswertung von Utility-Value-Funktionen, die den jeweiligen Kanälen von den Transducern zur Verfügung gestellt werden. Neben der Dynamik der Auswertung findet auch eine Separierung aufgrund der Definitionsmöglichkeiten unterschiedlicher Kanäle statt.

Die wichtigen Unterschiede sind somit der zwei-stufige Ansatz von SODAPOP, Kanäle – die für unterschiedliche Nachrichtensätze zuständig sind – definieren zu können und auf diesen Kanälen Strategien anwenden zu können. Dieser verteilte Ansatz durch unterschiedliche Kanäle unterscheidet SODAPOP von Technologien wie HAVi [96], Jini [129], der Open Agent Architecture und dem Galaxy Communicator, die auf singulären Einheiten zur Verwaltung des Nachrichtenflusses basieren. HAVi und Jini setzen auf einen einfachen Subskriptionsmechanismus auf einem globalen Bus. Mögliche Konflikte verschiedener Komponenten werden hier nicht erkannt und nicht gelöst. Die Open Agent Architecture wendet an einer zentralen Stelle prolog-basierte Routing-Strategien an. Ähnlich die Galaxy Infrastruktur, bei der interne Routingregeln für die Zuteilung von Nachrichten zuständig sind. Neben diesen Infrastrukturen, die die Kommunikation mittels „schwergewichtigen“ zentralen Komponenten ermöglichen existieren die peer-to-peer-Ansätze, für die die Architektur des AMIGO-Projektes [11, 12] exemplarisch steht. Hier verfügt jede Komponente im Komponentenensemble über Funktionseinheiten die zuständig sind für *Service Discovery*, *Service Composition* und *Service Publishing*. Die Art des Vorgehens einer Komponente ist somit für die Entwickler anderer Komponenten vollkommen intransparent. Das Beispiel EMBASSI herangezogen, in dem verschiedene Anbieter von Unterhaltungselektronik beteiligt waren, bestünde hier die große Gefahr dass Komponenten eines Anbieters nur mit Komponenten desselben Anbieters kommunizieren.

Die Möglichkeit transparente Kanalstrategien zur Konfliktlösung und zur Garantie des Nachrichtenflusses anzuwenden hat somit zwei wichtige Effekte: Es entlastet den Komponentenentwickler von der Aufgabe eigene Service-Discovery- und Service-Composition-Strategien entwickeln zu müssen und bietet ihm somit die Möglichkeit sich völlig auf die Funktionalitäten der zu entwickelnden Komponenten zu konzentrieren. Der zweite Effekt besteht im erhöhten transparenten Verhalten eines Komponentenensembles.