Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

FAKULTÄT FÜR !NFORMATIK

SELBSTADAPTIERENDE PERSÖNLICHE AGENTEN ZUR UBIQUITOUS BUSINESS MEETING UNTERSTÜTZUNG

ausgeführt am Institut für Distributed and Multimedia Systems der Universität Wien

DIPLOMARBEIT

zur Erlangung des akademischen Grades
Magister rerum socialium oeconomicarumque
Magister der Sozial- und Wirtschaftswissenschaften
(Mag. rer. soc. oec.)

im Rahmen des Studiums
Wirtschaftsinformatik

eingereicht von:
Stefan Nießner

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Dipl.-Ing. Dr. techn. Karin Anna Hummel

Wien, November 2008

Unterschrift Verfasser

Unterschrift Betreuerin

Kurzfassung

Die hier beschriebene Diplomarbeit umfaßt einen theoretischen und einem praktischen Teil. Die Arbeit befasst sich mit der Implementierung eines persönlichen Agenten in Ubiquitous Computing Umgebungen, der die Benutzerunterstützung mittels eines “goal-oriented” Ansatzes realisieren soll. Dieser Ansatz wurde deshalb gewählt, weil er menschliche Intentionen gut abbilden kann. Dabei soll der Agent durch a) Beobachtung, b) Lernen, c) explizite Interaktion mit dem Benutzer die Assistenzfunktionen wahrnehmen und laufend adaptiert werden. Des Weiteren werden einige, bereits vorhandene verwandte Arbeiten aus dem Bereich “Software Agents” und “Implizite Interaktion und Personalisierung” vorgestellt.

Das hier entwickelte Programm baut auf Java und dessen Agentenframework JADE auf. Zielplattform sollen Personal Digital Assistants und SmartPhones sein. Es werden drei Use Cases für Meetings beschrieben, die bei dem hier entwickelten Programm zum Einsatz kommen. Die Funktion des Agenten umfasst neben einfachen Aufgaben wie das Darstellen einfacher Informationen auch komplexere Unterstützung ausgewählter Use Cases.

Es sind drei Use Cases für Meetings vorgesehen, die eine gekapselte Funktion (Erinnerung eines Benutzers), eine kooperative Funktion (Änderung der Meeting Agenda), sowie ein Fall, der langfristige Ziele betrifft (Karriereleiter) dargestellt. Bei der Erinnerungsfunktion geht es darum, dass der Vortragende vom Agenten notifiziert wird, wenn er in Gefahr läuft, seine ihm zugewiesene Redezeit, zu überschreiten. Beim Use Case Agenda wird die Unterstützung des Agenten bei der abgestimmten Einbringung von Agendapunkten beschrieben. Die Aufgaben umfassen konsistente, koordinierte Änderung zwischen mehreren Benutzer-Agenten und Notifizierung der Benutzer. Beim Use Case Karriereleiter wird eine Funktion zur Unterstützung langfristiger Ziele, wie der berufliche Aufstieg, durch einen Agenten vorgeschlagen.

Abstract

This diploma thesis consists of a theoretical part and a practical part.

This work deals with the implementation of a personal agent in Ubiquituous Computing environments, which should realize the support of users in a goal-oriented way. I have chosen this approach, because it is the best way to give an impression of human intentions. The agent should fulfill its tasks through a) observation, b) learning, and c) explicit interaction with the user. Furthermore, there should be continuos adaption of the agents' behaviour. In addition, some existing projects from the domains "Software Agents" and "Implicit Interaction and Personalization" will be presented.

The functionality of the agent will be described, as will the scenerios in which it is used. From the still known characteristics and achievements, the user is assigned a certain role. Depending on the role, the agent has to fulfill different tasks.

The implementation of the assistant is built up on Java and its agent framework JADE. The target platform is a PDA, which is often used in industry. A meeting agent will serve as a practical example. The activities of this assistant include simple tasks such as showing information about the users and meetings and giving an overview of the following meetings and their participants, as well as more complex support described by selected use cases. Three different use cases are foreseen for meetings: an encapsulated function (reminder of a user), a cooperative function (change and reorganization of the meeting agenda), and a case which concerns long-term targets (such as careers). With the Reminder function Members of a meeting will be warned by the system if they run the risk of exceeding their speech time limit. The use case Agenda includes the timely coordination during meetings as well as notifications of changes in the agenda. The aim of the use case Career is to model and support the professional career of the user.

Riassunto

Questa tesi di laurea, che è scritto in tedesco, consiste in una parte teorica ed una parte pratica.

Soggetto di questa tesi è l'implementazione di un agente personale in un ambiente di Ubiquitous Computing, che dovrebbe realizzare l'assistenza personale con un'impostazione del tipo "Goal-oriented". Quest'impostazione è stata scelta perchè può raffigurare l'intenzioni umane in un modo tanto soddisfacente. L'agente deve assistere l'utente con a) osservazione, b) studi e c) interazione esplicita. Quando qualcosa è modificato, l'agente si deve adattare di nuovo. Oltra a ciò sono presentate tesi, che già esistono su questa tema.

La funzione dell'agente e gli scenari, dove si ha bisogno di questa sistema, sono descritti.

L'assistente è implementato nella lingua Java e JADE. JADE è il sistema per implementare degli agenti in Java. Il programma deve funzionare con un PDA. Il PDA è stato scelto, perchè è molto usato nell'industria. Nell'attività dell'assistente sono incluse la presentazione delle informazioni dell'utente, delle presentazioni e delle conferenze.

Sono previsi tre azioni del tipo "Use Case": una funzione incapsulata (reminiscenza del utente), una funzione cooperativa (cambio dell'agenda delle conferenze) ed una funzione a lunga scadenza (carriera): La reminiscenza dell'utente assicura che quando il presentatore è in pericolo di non finire la conferenza nel tempo che è scritto nell'agenda, deve essere ricordato che deve finire il più presto possibile. L'agente anche può cambiare l'agenda delle conferenze. Quá la organizzazione e riorganizzazione delle conferenze è incentrata. Quando ci sono modificazione dell'agenda, l'utente deve essere informato. Nell'use case carriera l'oggetto delle analisi è come il problema della carriere a lunga scadenza può essere sostenuto con gli agenti.

Inhaltsverzeichnis

1 Einleitung **1**

2 Software Agenten **3**

2.1 “Agent” als Zuschreibung 3

2.2 “Agent” als Beschreibung 5

2.3 Kommunikation 10

2.3.1 Foundation for Intelligent Physical Agents (FIPA) 11

2.3.2 ACL Sprache 11

3 Grundlagen **13**

3.1 Goal-oriented Design von Agenten 13

3.2 Prädikatenlogik 13

3.2.1 Syntax und Semantik 14

3.3 Das Planungsproblem 16

3.3.1 Heuristische Suchfunktion 16

3.3.2 Zerlegung des Problems 17

3.3.3 Begriffe von Planungsproblemen 17

3.4 Geordnetes Planen 19

3.5 Implizite Interaktion und Personalisierung 22

3.5.1 Kontext 23

3.5.2 Identifikation von impliziter Mensch-Computer-Interaktion 25

3.5.3 Wahrnehmung / Awareness 25

4 Verwandte Arbeiten **27**

4.1 EasyMeeting 27

4.1.1 Context Broker Architecture 28

4.1.2 Semantic Web Ontologien in CoBrA 29

4.1.3 Gemeinsamkeiten bzw. Unterschiede zu dieser Arbeit 30

4.2 Context-Aware Office Assistant 30
4.2.1 Funktionsweise 31
4.2.2 Gemeinsamkeiten bzw. Unterschiede zu dieser Arbeit 31
4.3 Helsinki Prototype System 32
4.3.1 Ziele des Projekts 32
4.3.2 Anforderungen 32
4.3.3 Gemeinsamkeiten bzw. Unterschiede zu dieser Arbeit 33
4.4 Digital-Me 35
4.4.1 Entwurf der Digital-Me Architektur 35
4.4.2 Der persönliche Digital-Me Agent 36
4.4.3 Digital-Me Autonomie 37
4.4.4 Interaktion mit dem Benutzer 38
4.4.5 Gemeinsamkeiten bzw. Unterschiede zu dieser Arbeit 39
4.5 Treffen, Versammlungen und Events in smarten Umgebungen 40
4.5.1 M4: Multi-Modal Meeting Manager 41
4.5.2 AMI: Augmented Multi-party Interaction 42
4.5.3 Gemeinsamkeiten bzw. Unterschiede zu dieser Arbeit 43

5 Beschreibung der verwendeteten Technologien 44
5.1 Java 44
5.2 JADE und Leap-Mechanismen und API 45
5.2.1 Hauptpakete von JADE 45
5.2.2 Container und Platform 47
5.2.3 Agent-Management-System 47
5.2.4 Directory-Facilitator 48
5.2.5 Aufbau von Agentenklassen 48
5.2.6 Behaviours 49
5.3 Weitere verwendete Technologien 50
5.3.1 Software 50
5.3.2 Device 51

6 Beschreibung des Programms 52
6.1 Benutzerklassen 52
6.2 Use Cases 53
6.2.1 Agenda 54
6.2.2 Karriereleiter 55

6.2.3 Meeting Support 60
6.2.4 Zeitplan umorganisieren 61
6.2.5 Benutzer zu Meeting einladen 64
6.3 Systemvoraussetzungen und StartUp 66
6.4 Implementierung und Konzept 68
6.4.1 Aufgaben des Agenten 68
6.4.2 Meetingsupport 69
6.4.3 Bewertung der Präsentation 69
6.4.4 Reorganisation der Meetings 72
6.4.5 Karriereleiter 73
6.4.6 Benutzer zu Meeting einladen 74

7 Programmablauf und GUI 76
7.1 Benutzergesteuerte Funktionen 77
7.1.1 Eigene Meetings und eigene Präsentationen abrufen 77
7.1.2 An- und Abmeldung von Meetings 79
7.1.3 Mein Profil 79
7.1.4 Benutzer einladen 80
7.1.5 Links zu Meetings abrufen 82
7.2 Agentengesteuerte Funktionen 82
7.2.1 Meetingsupport 82
7.2.2 Meeting evaluieren 83

8 Zusammenfassung 85

Anhang A: Installationsanleitung I
A.1: Installation des Servers I
A.2: Installation des Clients II

Anhang B: Partial-Order-Plans III

Anhang C: Algorithmenverzeichnis VII

Anhang D: Abbildungsverzeichnis VII

Anhang E: Abkürzungsverzeichnis IX

Literaturverzeichnis XI

Danksagungen

An dieser Stelle möchte ich mich bei einigen Personen bedanken, die mit ihrer Unterstützung wesentlich zum Gelingen dieser Abschlussarbeit beigetragen haben. Mein herzlicher Dank gilt:

Meiner Betreuerin Dipl.-Ing. Dr. techn. Karin Hummel, die mich in dieser - mittlerweile doch recht langen - Zeit betreut hat.

Den beiden Technikern des Institutes für Distributed and Multimedia Systems an der Universität Wien, Rudolf Hürner und Ewald Hotop, die mir bei technischen Problemen mit Rat und Tat zur Seite gestanden sind.

Johanna Henninger, Elizabeth Andersen und Alessandro Miasciasci fürs Korrekturlesen.

Sowie natürlich meiner Familie und meinen Freunden, die mich auf meinem Weg begleitet, bestätigt und gestützt haben.

1 Einleitung

In den letzten Jahren hat sich ein nicht zu übersehender Trend in Richtung Meetingunterstützung und Automatisierung von Abläufen bemerkbar gemacht.

Herkömmliche Terminplaner gibt es schon in ausreichender Menge. Deswegen soll mit dieser Applikation ein Terminplaner mit meetingunterstützender Wirkung, der auf die Bedürfniss der User eingeht, im Vordergrund entwickelt werden. Anhand einiger Use Cases soll gezeigt werden, dass eine Applikation mehrere selbstadaptierende, lernende und zugleich unterstützende Funktionen wahrnehmen kann.

In dieser Arbeit werden verschiedene Definitionen von “Software Agenten” vorgestellt, sowie ein kurzer Überblick darüber gegeben, wie die Kommunikation zwischen Agenten funktioniert. Im nächsten Kapitel werden die Grundlagen von Prädikatenlogik und Planen erklärt. Dieses Kapitel beinhaltet eine Einführung in Begriffe wie heuristische Suchfunktion, geordnetes Planen und implizite Interaktion und Personalisierung.

Danach folgt die Vorstellung einiger verwandter Arbeiten. *EasyMeeting* ist einer der ersten Smart Meeting Rooms, die unter der Verwendung der Foundation for Intelligent Physical Agents (FIPA)-Agententechnologie entwickelt wurden. Ziel des *Context-Aware Office Assistant* ist es, den Inhaber eines Büros in Bezug auf die Besucher, die zu ihm kommen und deren Termine zu verwalten. Das *Helsinki Prototype System* ist das erste System zur Unterstützung von Besprechungen, das bei bilateralen diplomatischen Treffen zum Einsatz gekommen ist. *Digital-Me* soll die ubiquitäre Interaktion mit “aktiven Umgebungen” unterstützen. Im letzten Punkt dieses Kapitels werden noch einige weitere Anwendungen vorgestellt, die in smarten Umgebungen zum Einsatz kommen.

Im Kapitel über verwendete Technologien wird eine Einführung Java und dessen Agentenframework JADE gegeben. Des Weiteren werden MySQL, Apache und Leap vorgestellt. Im letzten Punkt dieses Kapitels werden die technischen Eigenschaften des Nokia N91 beschrieben, für das die Applikation entwickelt wurde.

In den letzten beiden Kapiteln wird die Funktionalität des Agenten beschrieben,

sowie die Szenarien, in denen er eingesetzt werden soll und Programmabläufe. Aus den bereits bekannten Eigenschaften bzw. Errungenschaften eines Benutzers wird ihm vom System eine bestimmte Rolle zugewiesen. Abhängig davon, welche Rolle der Benutzer inne hat, ist der Agent für verschiedene Aufgaben zuständig.

Die Implementierung des Assistenten baut auf Java und dessen Agentenframework JADE auf. Zielplattform soll der in der Industrie häufig verwendete Personal Digital Assistant (PDA) oder SmartPhones werden. Die Tätigkeit dieses Assistenten umfasst einfache Aufgaben, wie das Anzeigen von Informationen über die jeweiligen Benutzer bzw. Meetings, eine Übersicht, wann die nächsten Vorträge sind und welche Benutzer daran aktiv teilnehmen, aber auch komplexere Unterstützung ausgewählter Use Cases. Es sind drei Use Cases für Meetings vorgesehen, die eine gekapselte Funktion (Erinnerung eines Benutzers), eine kooperative Funktion (Änderung der Meeting Agenda), sowie ein Fall, der langfristige Ziele betrifft (Karriereleiter):

Erinnerungsfunktion: Hierbei geht es darum, dass der Vortragende vom Agenten notifiziert wird, wenn er in Gefahr läuft, seine ihm zugewiesene Redezeit, zu überschreiten.

Agenda: Hier wird die Unterstützung des Agenten bei der abgestimmten Einbringung von Agendapunkten beschrieben. Die Aufgaben umfassen konsistente, koordinierte Änderung zwischen mehreren Benutzer-Agenten und Notifizierung der Benutzer.

Karriereleiter: Hier wird eine Funktion zur Unterstützung langfristiger Ziele, wie der berufliche Aufstieg, durch einen Agenten vorgeschlagen.

Zum Schluß folgt noch ein kurzer Ausblick darauf, wie die Erkenntnisse dieser Arbeit in Zukunft für die Entwicklung smarter Devices mit autonomen Handeln verwendet werden können.

2 Software Agenten

Jeffrey Bradshaw verfolgt in seinem Buch “Software Agents” zwei Ansätze, wie ein Software Agent definiert werden kann - entweder als Zuschreibung oder als Beschreibung [12].

2.1 “Agent” als Zuschreibung

Das American Heritage Dictionary[1] definiert einen Agenten als “jemanden, der agiert oder die Macht bzw. Authorität hat zu agieren ... oder jemand anderen zu repräsentieren” oder als “Mittel, durch das etwas gemacht oder verursacht wurde; Instrument”. Das Wort wird vom lateinischen Verb *agere* abgeleitet: antreiben, führen, agieren oder tun.

Normalerweise würde man von einem Software Agenten erwarten, dass er mit Rücksicht auf andere agiert und eine bestimmte Aufgabe durchführt, die man ihm zugewiesen hat. Aber da es aufwendig ist dem Agenten jedes Detail mitzuteilen, sollten Fähigkeiten vorgesehen werden, eigene Schlussfolgerungen aus Anweisungen zu ziehen. Dazu ist es aber notwendig, dass Agenten den Inhalt der Anfrage deuten können (Semantik). Die besten Agenten müssen daher nicht nur eine bestimmte Übung trainieren, sondern auch Besonderheiten über den Benutzer und die Situation im Allgemeinen in Betracht ziehen und erkennen können.

In diesem Zusammenhang erfüllt der Agent eine Rolle, die Nicholas Negroponte als “digitale Schwiegertochter” bezeichnet [48]:

> *When I want to go out to the movies, rather than read reviews, I ask my sister-in-law. We all have an equivalent who is both an expert on movies and an expert on us. What we need to build is a digital sister-in-law.*

Negroponte gibt hier ein sehr persönliches Beispiel: Bei gewissen Dingen, so wie z.B. Rat bzgl. Kinofilmen, fragt er sehr oft seine Schwiegertochter, da diese ihn selbst

[1] http://www.bartleby.com/

gut kennt und weiß, was ihm gefallen wird und was nicht. Daraus schließt er, dass es wohl das Beste wäre, eine "digitale Schwiegertochter" zu entwickeln, die die Intentionen des Menschen möglichst gut abbilden kann.

Allerdings gibt es einige Behauptungen, dass viele Softwareagenten relativ wenig "agentenähnliches" an sich haben. Leonard Foner hat im Jahr 1993 in seiner Fallstudie "What's An Agent, Anyway?" folgendes beobachtet [32]:

> *I find little justification for most of the commercial offerings that call themselves agents. Most of them tend to excessively anthropomorphize the software, and then conclude that it must be an agent because of that very anthropomorphization, while simultanously failing to provide any sort of discourse or "social contract" between the user and the agent. Most are barely autonomous, unless a regularly-scheduled batch job counts. Many do not degrade gracefully, and therefore do not inspire enough trust to justify more than trivial delegation and its concomitant risks.*

Foner kritisiert, dass jede Software vermenschlicht wird und daraus folgt, dass sie ein Agent sein muss. Seiner Meinung nach verfügen die Agenten über viel zu wenig Autonomie. Des Weiteren ist die soziale Bindung zwischen Mensch und Agent sehr gering.

Daniel Dennett beschreibt drei verschiedene Einstellungen, die Leute gegenüber dem System haben können. Naturgemäß werden die Leute immer das auswählen, was die einfachste aber trotzdem zuverlässigste Erklärung eines Verhaltens liefert. Für natürliche Systeme (zum Beispiel die Kollision von Billiardkugeln) wird das Verhalten basierend auf den physikalischen Charakteristiken und Gesetzen vorhergesagt. Wenn wir genügend Verständnis vom System haben, ist es uns möglich, das Verhalten aufgrund von Funktionen vorherzusagen. Dennett definierte drei verschiedene Einstellungen, die man bzgl. Vorhersagen haben kann [27]:

Physikalische Einstellung: Vorhersage aufgrund von physikalischen Charakteristiken und Gesetzen

Design-Einstellung: Vorhersage aufgrund der Motivation

Intentionale Einstellung: Vorhersage aufgrund von Annahmen basierend auf rationalem Handeln

M. P. Singh listet mehrere pragmatische und technische Gründe dafür auf, so dass Agenten als intentionale Systeme erscheinen [56]:

- Sie erscheinen uns - als Designer und Analytiker - natürlich.
- Sie beinhalten eine kurze Beschreibung, helfen uns zu verstehen und erklären das Verhalten komplexer Systeme.
- Sie stellen gewisse Gleichmäßigkeiten und Aktionsmuster zur Verfügung, die unabhängig von der exakten physikalischen Implementation des Agent im System sind.
- Sie können als Agenten gegenseitig Schlüsse über sich selbst ziehen.

2.2 "Agent" als Beschreibung

Yoav Shoham hat im Jahre 1997 folgende technische Definition von Softwareagenten gefunden [55]:

> *A Software entity which functions continously and autonomously in a particular environment, often inhabited by other agents and processes.*

Das Verlangen nach Kontinuität und Autonomie kommt daher, dass ein Agent seine Aktivitäten in einer Art und Weise ausführen sollte, die es ihm ermöglicht flexibel und intelligent zu agieren und auf Veränderungen der Umwelt weitgehend ohne menschliches Zutun zu reagieren. Der optimale Agent sollte idealerweise über eine längere Periode aus seiner eigenen Erfahrung lernen können. Zusätzlich sollte er in einer Umgebung aktiv sein, die er mit anderen Agenten teilt und mit diesen auch kooperieren und kommunizieren kann. Eventuell sollte er sich auch in dieser Umgebung von einem Ort zu einem anderen bewegen können.

Jeder Agent kann über folgende Attribute in mehr oder weniger großem Ausmaß verfügen [28, 33]:

Reaktivität: die Möglichkeit zu erkennen und entsprechend der Erkenntnis zu handeln

Autonomie: zielgerichtetes, proaktives und selbststartendes Verhalten

Gemeinschaftliches Verhalten: kann zusammen mit anderen Agenten arbeiten um ein gemeinsames Ziel zu erreichen

Kommunikationsmöglichkeit auf Wissensebene: Die Fähigkeit mit anderen Personen und anderen Agenten in einer Sprache zu kommunizieren, die menschlichen Sprechakten ähnlicher ist als typische Programm zu Programm Protokolle auf Symbolebene.

Fähigkeit, Schlußfolgerungen ziehen zu können: Der Agent kann - mit dem Wissen von früheren Zielen und bevorzugten Methoden - abstrakte Spezifikationen von Aufgaben bearbeiten um Flexibilität zu erreichen; geht über die gegebene Information hinaus und kann explizite Modelle von sich selbst, dem Benutzer, der Situation und/oder anderen Agenten haben.

temporäre Kontinuität: Persistenz von Identität und Zustand über einen längeren Zeitraum

Personalität: die Möglichkeit Attribute eines “glaubhaften” Charakters - so wie zum Beispiel Emotion - zu manifestieren

Adaptivität: die Fähigkeit aus Erfahrung zu lernen und sich zu verbessern

Mobilität: die Fähigkeit aus eigenem Antrieb von einer Host-Plattform zu einer anderen zu migrieren

Eine einfachere Charakterisierung von Agententypen würde darin bestehen, dass man jede einzelne Kombination dieser Attribute beschreibt. Von der Gemeinschaft der Agentenforscher wurden dazu verschiedene Schemen und Taxonomien vorgeschlagen.

Zum Beispiel unterscheidet man Forscher in der künstlichen Intelligenz zwischen *schwacher* und *starker Vorstellung* von Agenten: Agenten mit der kleinsten Vielfalt werden entwickelt um explizite mentale oder emotionelle Qualitäten zu besitzen. Laut B. Moulin und B. Chaib-draa werden Agenten, zum Beispiel, anhand ihrer Fähigkeiten Probleme zu lösen, unterschieden [47]:

> *A reactive agent reacts to changes in its environment or to messages from other agents... An intentional agent is able to reason on its intentions and beliefs, to create plans of actions, and to execute those plans... In addition to intentional agent capabilities, a social agent possesses explicit models of other agents.*

Moulin und Chaib-draa beschreiben, welche besonderen Fähigkeiten unterschiedlich ausgeprägte Agenten haben: Ein reaktiver Agent reagiert auf Veränderungen in seiner Umwelt, ein intentionaler Agent kann Schlüsse aus seinem eigenen Handeln ziehen und ein sozialer Agent kann explizite Modelle von anderen Agenten besitzen.

In einem einflussreichen Artikel von IBM werden intelligente Agenten anhand von drei Dimensionen, die in Abbildung 2.1 graphisch dargestellt werden, beschrieben [35]:

Wirkung: Wirkung ist der Grad der Autonomie und Authorität mit der der Agent versehen ist und kann zumindest qualitativ von der Natur der Interaktion zwischen Agent und anderen Entitäten im System gemessen werden. Der Agent muss zumindest asynchron laufen. Der Grad der Wirkung wird verbessert, indem der Agent einen Benutzer in irgendeiner Art und Weise repräsentiert. Ein fortgeschrittener Agent kann über Daten, Applikationen, Services oder mit anderen Agenten interagieren.

Intelligenz: Intelligenz ist der Grad des Schließens und des gelernten Verhaltens, d.h., die Fähigkeit des Agenten, die Behauptungen des Benutzers bzgl. seiner Ziele zu akzeptieren und diejenigen Aufgaben auszuführen, die ihm aufgetragen worden sind. Zumindest sollten die Nutzerpräferenzen bekannt sein. Auf einer höheren Ebene inkludiert Intelligenz ein Benutzermodell. Noch weiter oben auf der Intelligenzskala befinden sich Systeme, die sowohl lernen, als auch die Fähigkeit besitzen, sich ihrer Umwelt anzupassen.

Mobilität: Die Mobilität gibt an, in welchem Ausmaß ein Agent fähig ist durch das Netzwerk zu reisen. Mobile Skripte können auf einer Maschine erstellt, verschickt und auf einer anderen Maschine ausgeführt werden. Sie können auch während ihrer Ausführung von einer Maschine zur nächsten wandern. Weiters wird unterschieden in starke und schwache Mobilität unterschieden. Bei starker Mobilität kann das Segment des Programmes, das gerade ausgeführt wird auch von einem Ort zum nächsten migrieren. Ein laufender Prozess kann gestoppt werden, migrieren und seine Ausführung an einem anderen Ort fortsetzen. Bei schwacher Mobilität hingegen werden nur das Codesegment und die Initialisierungsdaten transferiert. Ein Programm, das von einem Ort an einen anderen transferiert wurde, startet an seiner neuen Lokation immer von Beginn an [60].

Hyacinth Nwana hat im Jahre 1996 eine Agententopologie vorgeschlagen, die andere Dimensionen der Klassifizierung identifiziert [51]:

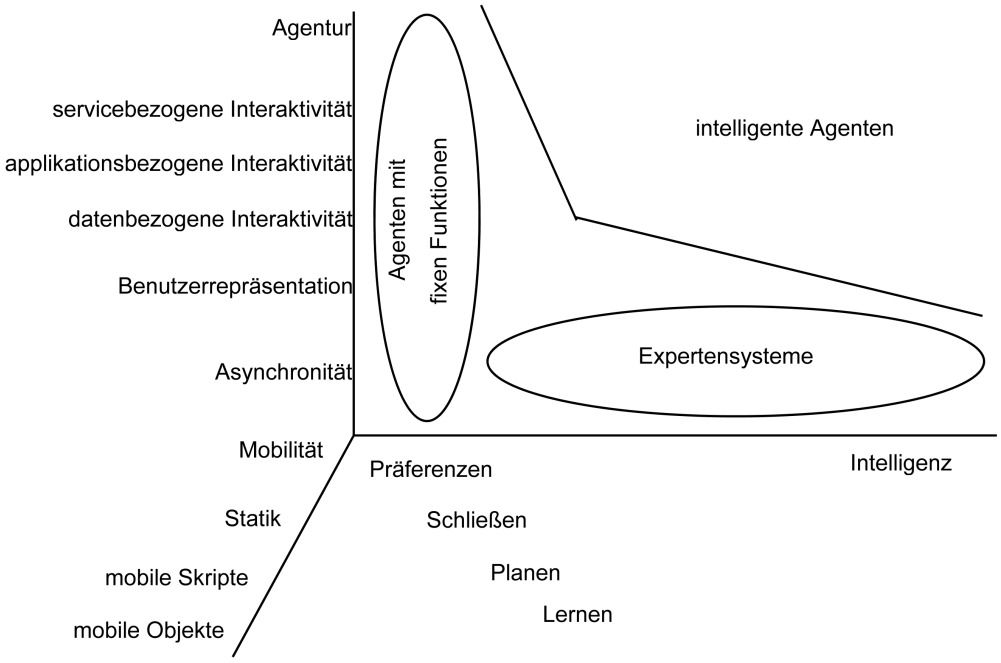

Abbildung 2.1: Anwendungsbereich von intelligenten Agenten (adaptiert nach Gilbert et. al., 1995) [35]

Mobilität: Agenten können entweder statisch oder mobil sein.

Vorhandensein eines symbolischen Schlußfolgerungsmodells: Agenten verfolgen entweder ein abwägendes oder ein reaktives Schlußfolgerungsmodell.

Darlegung von Ideal- und Primärattributen: Beispiele hierfür wären Autonomie, Kooperation und Lernen. Aus diesen Charakteristiken hat Nwana vier Agententypen abgeleitet: *kollaborative*, *kollaborativ lernende*, *Interface* und *smarte Agenten*

Rollen: Agenten werden manchmal nach ihren Rollen, die sie einnehmen, identifiziert. Es gibt zum Beispiel World Wide Web (WWW) Informationsagenten. Ihre Hauptaufgabe ist es, riesige Informationsmengen in Wide Area Networks, so wie dem Internet, zu managen.

hybride Agenten: Von hybriden Agenten spricht man, wenn zwei oder mehrere Agentenphilosophien (vgl. die von Nwana abgeleiteten Agententypen) kombiniert werden.

Sekundärattribute: so wie Vielseitigkeit, Gutmütigkeit, Aufrichtigkeit, Zuverlässig-

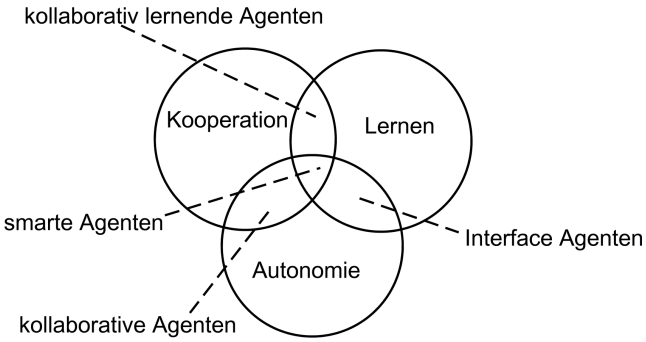

Abbildung 2.2: Agententypologie adaptiert nach Nwana [51]

keit, zeitliche Kontinuität, sanfte Degradierung der Systemfunktionalität (engl. graceful degradation), sowie mentale und emotionale Qualitäten.

Nach der Entwicklung dieser Klassifizierung, beschreibt Nwana sieben weiterführende Kategorien: kollaborative Agenten, Interface Agenten, Mobile Agenten, Information/Internetagenten, reaktive Agenten, hybride Agenten und smarte Agenten. Eine graphische Darstellung dieser Typologie findet man in Abbildung 2.2.

Stan Franklin und Arthur Graesser gaben 1996 in [33] ihre eigene Definition eines Agenten:

> *An autonomous agent is a system situated within and part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.*

Ihrer Meinung nach tastet ein Agent seine Umgebung von Anfang an ab und, da er lernfähig ist, weiß er mit der Zeit, wie seine Agenda in Zukunft auszuschauen hat. Eine graphische Darstellung der Agententaxonomie findet man in Abbildung 2.3.

Unterhalb der Anfangsklassifikation schlagen sie vor, dass Agenten durch Kontrollstrukturen, Umgebungen (z.B. Datenbank, Dateisystem, Netzwerk, Internet) und Sprache, in der sie geschrieben sind, sowie Anwendungen kategorisiert werden können.

Charles Petrie unternahm einen Versuch, Intelligenz und Autonomie zu beschreiben. Durch diese Eigenschaftszuschreibung müssen viele “Agenten” zur Suche und Filterung als simple “Frage-Antwort-Programme” oder Server bezeichnet werden. Ähnlich verhält es sich mit “mobilen Prozessen” und “mobilen Agenten” für jene Java-Applets, deren einzig agentenähnliche Funktion es ist, Prozesse auf fremden Maschinen sicher und stabil laufen zu lassen.

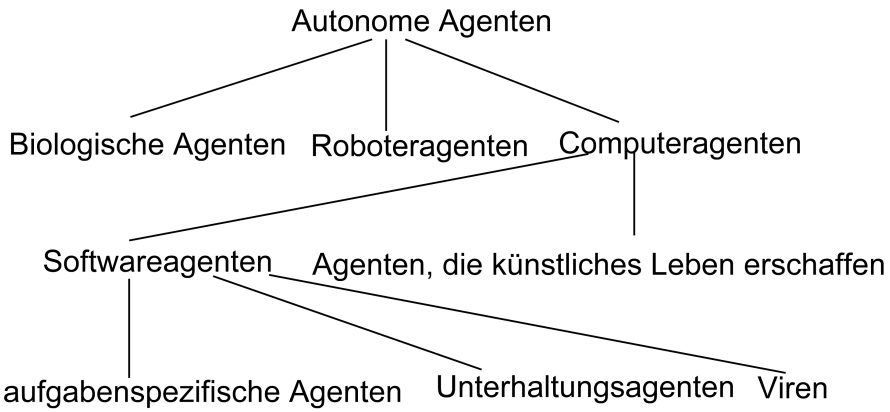

Abbildung 2.3: Agententaxonomie adaptiert nach Franklin und Graesser [33]

Im Gegensatz zu früheren Versuchen, die die Charakteristiken von Agenten nur generell beschrieben, spezialisierte sich Petrie auf einen einzigen Agententyp: *typisierte Nachrichtenagenten*. Diese unterscheiden sich von anderen Softwaretypen durch ihre Fähigkeit mit der Gemeinschaft der anderen Agenten über ein gemeinsames Nachrichtenprotokoll, nämlich die Knowledge Query and Manipulation Language (KQML)[2] , zu kommunizieren. Hierbei sind zumindest einige Teile der Nachrichtensemantik typisiert und applikationsunabhängig.

Ein weiteres Merkmal ist, dass die Kommunikation nicht nach dem Client-Server-Prinzip, sondern Peer-to-Peer stattfindet. Ein individuelles Softwaremodul ist kein Agent, wenn es mit anderen Agenten nur über ein Client-Server-Protokoll kommunizieren kann.

2.3 Kommunikation

Die Kommunikation zwischen Agenten baut auf asynchronem Message Passing auf. Unter asynchronem Message Passing versteht man eine Art des Nachrichtenaustausches zwischen einem Sender und Empfänger, bei der das Senden und Empfangen zeitversetzt stattfindet. Während bei synchronem Message Passing der sendende bzw. empfangende Prozess immer wartet, bis der Nachrichtenaustausch komplett abgeschlossen wurde, so ist dies bei der asynchronen Variante nicht der Fall. Das wohl am häufigsten verwendete Beispiel für asynchronen Nachrichtenaustausch zwischen Menschen ist die Kommunikation via E-Mail.

[2] http://www.csee.umbc.edu/kqml/

Jeder JADE-Agent verfügt über eine Mailbox, die *Agent Message Queue* (AMQ), in der die von anderen Agenten gesendeten Nachrichten gespeichert werden. Sobald eine Nachricht in der AMQ gespeichert wurde, wird der Agent, der die Nachricht empfangen soll, benachrichtigt [62].

2.3.1 Foundation for Intelligent Physical Agents (FIPA)

Die FIPA[3] ist eine Organisation der IEEE[4] Computergesellschaft, die sich um Standards der agentenbasierten Technologien kümmert. Des Weiteren stellt sie die Interoperabilität dieser Standards mit anderen Technologien sicher. Die FIPA wurde 1996 in der Schweiz gegründet. Am häufigsten werden die Standards zum Management von Agenten und zur Kommunikation zwischen den Agenten verwendet. Als Kommunkationssprache wird die Agent Communication Language (ACL) eingesetzt [31].

2.3.2 ACL Sprache

Die ACL Sprache wurde von der FIPA entwickelt und ist internationaler Standard für die Interoperabilität zwischen Agenten.

Dieser Standard beschäftigt sich mit Nachrichten basierend auf der Agent Communication Language, Protokollen zum interaktiven Nachrichtenaustausch, theoriebasierten Sprechakten zur Kommunikation und Repräsentationen der Sprache in der Inhalte ausgedrückt werden.

ACL ist an KQML angelehnt. Der Aufbau von KQML findet man in Algorithmus 1. Eine FIPA-ACL-Nachricht besteht aus einem Satz oder mehreren Parametern. Die Anzahl der Parameter ist situationsabhängig. Der einzige verpflichtende Parameter ist das Performativ, also die Absicht, warum der kommunikative Akt gesendet wurde. Nicht verpflichtend, aber in den meisten Fällen vorhanden, sind die Parameter Sender, Empfänger und Inhalt [31]. Eine Auflistung aller Parameter findet man in Tabelle 2.1.

Soll eine Nachricht gesendet werden, müssen zuerst die Felder der ACLMessage befüllt werden. Anschließend kann mit der *send()*-Methode die Nachricht vom sendenden Agenten an die private Nachrichtenschlange des empfangenden Agenten gesendet werden. Mittels der *receive()*-Methode wird nun eine Nachricht nach der anderen aus der Nachrichtenschlange ausgelesen und somit vom Agenten weiter-

[3] http://www.fipa.org/
[4] http://www.ieee.org/

Algorithmus 1 allgemeiner Aufbau von KQML

```
1: (<Performative>
2:    :content <Sprechakt>
3:    :sender <Name>
4:    :receiver <Name>
5:    :language <Text>
6:    :ontology <Text>
7: )
```

verarbeitet werden. Sobald ein Agent mit der Weiterverarbeitung einer Nachricht aus der Nachrichtenschlange begonnen hat, wird diese Nachricht aus der Nachrichtenschlange entfernt. Der Sender kann nun entweder auf eine Antwort warten - indem er die Methode *blockingReceive()* ausführt - oder er kann seine Arbeit fortsetzen. Wird *blockingReceive()* ausgeführt, blockiert der Sender solange bis er eine Nachricht vom Empfänger erhält und setzt erst nach erfolgreichem Empfangen seine Arbeit fort.

Mittels ACL-Nachrichten wird sowohl die Kommunikation zwischen Client und Server als auch Peer-to-Peer-Kommunikation - zwischen den Clients - unterstützt.

Feld	*Beschreibung*
sender	Sender der Nachricht
receiver	Liste der Empfänger der Nachricht
performative	die Kommunikationsabsicht, warum die Nachricht gesendet wurde
content	Inhalt der Nachricht
language	Sprache des Inhalts
ontology	Ontologie
reply-to	Die AID des Agenten, an den die Antwort geschickt werden soll.
encoding	Gibt die Codierung des Inhalts der Nachricht an.
protocol	Das Interaktionsprotokoll, das der sendende Agent mit dieser Nachricht verbindet.
conversation-id	Die ID der Konversation
reply-with	Wird verwendet, um die Antwort des Agenten zu identifizieren.
in-reply-to	Gibt an, auf welche Nachricht diese Nachricht eine Antwort ist.
reply-by	Gibt den spätestmöglichen Zeitpunkt an, zu dem der Empfänger die Nachricht erhalten will.

Tabelle 2.1: Felder von ACL - Nachrichten

3 Grundlagen

In diesem Kapitel werden relevante Aspekte und Grundlagen der künstlichen Intelligenz betrachtet, die für selbstadaptierende Agenten einen Nutzen darstellen.

3.1 Goal-oriented Design von Agenten

Goal-oriented heißt, dass eine Lösung nicht durch prozedurale Einzelaktivität ausgedrückt wird, sondern als zu erfüllende Bedingung, die dann aufgelöst bzw. erfüllt wird. Diese Erfüllung kann mit Hilfe eines Algorithmus automatisch erreicht werden. Ein Goal ist ein nichtoperationales Ziel, das vom System erreicht werden kann und ein präzises Kriterium zur ausreichenden Erfüllung der Anforderungsspezifikation zur Verfügung darstellt. Nichtoperational bedeutet, dass das Ziel nicht mit Hilfe von Objekten und Aktionen, die einem Agenten im System zur Verfügung stehen, erreicht werden kann [40].

3.2 Prädikatenlogik

Prädikatenlogik (im Englischen: First Order Logic) ist Teil der formalen Logik (vgl. Kapitel III.8 in [58]).

Sie basiert auf der Verwendung von *Gegenständen* (*Subjekten*, *Objekten*), *Prädikaten* und sog. *quantifizierende Redeteile*, so wie “für alle” und “es gibt”. Unter Subjekten versteht man Namen für Dinge, die aus einer bestimmten vorgegebenen Individuenmenge stammen. Prädikate bezeichnen Namen für Relationen auf dieser Individuenmenge [34]. Als Funktionen bezeichnet man Relationen, für die es einen Wert für einen gegebenen Input gibt.

Beispiele für Aussagen der Prädikatenlogik sind:

- *1 + 2 = 3:* Objekte: Eins, Zwei, Drei, Eins plus Zwei. Relation: ist. Funktion: plus.

- *Der böse König John regierte England um 1200.* Objekte: John, England, 1200. Relation: regierte. Prädikate: böse, König.

3.2.1 Syntax und Semantik

Syntax und Semantik umfassen mehrere Unterpunkte, die im Folgenden detailliert beschrieben werden.

Objekte und Relationen

Die Domäne eines Modells ist eine Sammlung von Objekten, die das Modell beinhaltet. Diese Objekte werden Domainelemente genannt. Diese Domainelemente stehen zueinander in Relation (vgl. Kapitel III.8 in [58]). Eine Darstellungsweise für die Bruderbeziehung zwischen König John und Richard Löwenherz ist zum Beispiel:

<Richard Löwenherz, König John>, <König John, Richard Löwenherz> .

Im Wortschatz der Prädikatenlogik existieren folgende Wörter: *Konstantsymbole* stehen für Objekte, *Prädikatsymbole* für Relationen (bzw. *logische Funktionssymbole* für Funktionen). Die Namen von Konstanten und Prädikaten sollten immer mit einem Großbuchstaben beginnen. Damit die Semantik Sätze zu Modellen zuordnen kann, um diese zu verifizieren, wird eine fixe Interpretation dieser Sätze benötigt. Damit wird festgelegt welche Objekte, Relationen und Funktionen mit welchen Symbolen assoziiert werden. In unserem Beispiel wäre das:

- *Richard* bezieht sich auf Richard Löwenherz und *John* bezieht sich auf den bösen König John.
- *Bruder* bezieht sich auf die Bruderschaftsrelation, die eine Menge von Tupeln von gegeben Objekten in der Gleichung ist.
- *Person* und *König* bezieht sich auf eine Menge von Objekten, die Personen und Könige sind.

Terme / Prädikate

Ein *Term* ist ein logischer Ausdruck, der sich auf ein Objekt bezieht. Setzt man Terme, die sich auf Objekt- und Prädikatsymbole beziehen, mit Relationen in Verbindung, erhält man einen *atomaren Satz*. Beispiel für einen atomaren Satz ist:

Bruder(Richard, John).

Dieser Satz sagt, dass Richard und John Brüder sind.

Des Weiteren können atomare Sätze auch komplexe Argumente haben und ineinander verschachtelt werden.

Verheiratet(Vater(Richard), Mutter(John))

Dieser Satz drückt aus, dass der Vater von Richard Löwenherz mit der Mutter von König John verheiratet ist.

Logische Verbindungsoperatoren

Ein logischer Operator ist eine Funktion, die einen Wahrheitswert liefert, mit denen es möglich ist komplexere Sätze zu erstellen. Einige Anwendungsbeispiele logischer Operatoren findet man in Tabelle 3.1.

komplexer Satz	*Bedeutung*
Bruder(John, Richard) $\wedge$ Bruder(Richard, John)	John ist ein Bruder von Richard UND Richard ist ein Bruder von John.
König(Richard) $\vee$ König(John)	Richard ist der König ODER John ist der König. (nicht exklusiv)
$\neg$ König(Richard) $\Rightarrow$ König(John)	Wenn NICHT Richard der König ist, dann ist John der König.

Tabelle 3.1: Anwendungsbeispiele logischer Operatoren.

Quantoren

Sollen alle oder nur eine Instanz eines Objekts angesprochen werden, können Quantoren verwendet werden. Die Prädikatenlogik stellt hierfür einen *Existenzquantor* und einen *Allquantor* zur Verfügung. Eine Überisicht über Quantoren findet man in Tabelle 3.2.

Quantor	*Name*	*Bedeutung*
$\forall$	Allquantor	Mit dem Allquantor ($\forall$) kann ausgedrückt werden, dass ein Satz für alle Elemente einer Menge gilt.
$\exists$	Existenzquantor	Dieser sagt aus, dass ein Element in der Menge existiert, für das der Satz zutrifft.

Tabelle 3.2: Quantoren

Einige Anwendungsbeispiele von Quantoren findet man in Tabelle 3.3.

Beispiele	*Bedeutung*
$\forall$ x (König(x) $\Rightarrow$ Person(x))	Für alle x gilt, wenn x ein König ist, dann ist x eine Person.
$\exists$ x (Krone(x) $\wedge$ aufKopf(x, John))	Es existiert ein x für das gilt, x ist eine Krone und x ist auf dem Kopf von John.
$\forall$ x $\forall$ y (Bruder(x, y) $\Rightarrow$ Geschwister(x, y))	Es gilt für alle x und alle y, dass wenn x und y Brüder sind, dann sind x und y Geschwister.
$\forall$ x ($\exists$ y (liebt(x,y)))	Es gilt für alle x, dass ein y existiert, für das gilt, dass x y liebt.

Tabelle 3.3: Anwendungsbeispiele Quantoren

3.3 Das Planungsproblem

Hier wird ein Überblick über Planungsprobleme gegeben. Für komplizierte Aufgabenstellungen ist es nicht ausreichend, auf einfache Suchalgorithmen, so wie A*- oder Depth-First-Search, zurückzugreifen. Ein Ansatz zur Lösung ist die Verwendung eines Planungsagenten, der mit menschlicher Hilfe (Konfigurationen, Heuristiken, Training) dann autonom Lösungen und Lösungswege vorschlägt.

3.3.1 Heuristische Suchfunktion

Heuristische Suche befasst sich damit für ein Suchproblem eine gute, aber nicht optimale Lösung zu finden. Diese Verfahren dienen zur Reduzierung des mit einer Suchaufgabe verbundenen Aufwandes [42]. Heuristiken unterscheiden sich von Verfahren

zur mathematischen Optimierung dadurch, dass bei der Heuristik die Optimalität einer Lösung nicht feststellbar ist.

Das wohl bekannteste Beispiel für die Verwendung einer Heuristik ist das "Problem des Handlungsreisenden" (engl. Traveling Salesman Problem, TSP[1]). Hier geht es darum, dass für einen Reisenden die kürzestmögliche Strecke für den Besuch mehrerer Orte und anschließender Rückkehr zum Ausgangsort gewählt werden soll. In den meisten Fällen wird die Methode zur Berechnung des nähesten Nachbarn verwendet. Hier wird in jeder Stadt die nähestgelegene Stadt als Folgeort angenommen. Dies wird so lange wiederholt, bis alle Städte besucht worden und der Reisende sich wieder am Ausgangspunkt befindet.

3.3.2 Zerlegung des Problems

Ein Problem ist dann zerlegbar, wenn es sich in mehrere Unterprobleme zerteilen lässt. Unterprobleme lassen sich meist leichter lösen, da sie eine weniger abstrakte Beschreibung des Problems bieten. Die Aufgabe wird damit für den Problemlöser konkreter und greifbarer.

Will man zum Beispiel ein Übernachtungspaket für jedes Urlaubsziel in Österreich finden (dies ist das Hauptproblem), so ist es sinnvoll zuerst den nähesten Flughafen für jedes Urlaubsziel zu finden (erstes Unterproblem) und dann die nähesten Übernachtungsmöglichkeiten für jeden Flughafen (zweites Unterproblem). Das Hauptproblem wurde hier in zwei Unterprobleme zerlegt.

3.3.3 Begriffe von Planungsproblemen

Im Allgemeinen besteht die Sprache von Planungsproblemen aus der Repräsentation von *Zuständen*, *Zielen* und *Aktionen*.

Zustände: Die Welt wird hierbei in logische Bedingungen zerlegt und stellt einen Zustand als eine Zerlegung von atomaren Sätzen dar. Zum Beispiel kann *arm* $\wedge$ *unglücklich* den Zustand eines unglücklichen Menschen darstellen.

Ziele: Ein Ziel ist ein teilweise spezifizierter Zustand, der als eine Verbindung von positiven Grundliteralen dargestellt wird, zum Beispiel *reich* $\wedge$ *berühmt*.

[1] http://www.tsp.gatech.edu/

Aktionen: Eine Aktion wird durch Terme spezifiziert, denen eine Vorbedingung zu Grunde liegt und die ein Ergebnis zufolge haben, welches eintritt, wenn die Aktion ausgeführt wurde.

Aktionsschema: Es stellt eine Anzahl von verschiedenen Aktionen dar, die dadurch abgeleitet werden können, dass die Variablen (z. B. *p*, *von*, *nach*) mit verschiedenen Konstanten instanziert werden.

Ein Beispiel für ein Aktionsschema wird in Algorithmus 2 gezeigt. Es beschreibt den *Flug* eines Objektes *p* von einem Ort an einen anderen. Vorbedingungen dieser Aktion sind, dass sich *p* im Ort *von* befindet, dass *p* ein *Flugzeug* ist und *von* und *nach Flughäfen* sind. *Effekt* dieser Aktion ist, dass sich *p nicht mehr in von* befindet und *p* sich *in nach* befindet.

Ein Aktionsschema besteht aus folgenden drei Teilen:

Aktionsname und Parameterliste: z.B.: *Flug(p, von, nach)*. Dient dazu, um die Aktion zu identifizieren.

Vorbedingung: Die Vorbedingung ist eine Verbindung von funktionsfreien positiven Literalen (im Sinne von atomaren Sätzen, die keine Funktionen beinhalten und wahr sind), die angeben, welche Bedingungen vor Ausführung der Aktion wahr sein müssen. Jede Variable in der Vorbedingung muss auch in der Parameterliste erscheinen.

Ergebnis: Das Ergebnis ist ein Verbindung von funktionsfreien atomaren Sätzen, die beschreiben, wie sich der Zustand geändert hat wenn die Aktion ausgeführt wurde. Ist im Ergebnis ein positives Literal P vorhanden, so handelt es sich um ein positives Ergebnis. Ist P negativ, hat man es mit einem negativen Ergebnis zu tun. Jede Variable, die im Ergebnis vorkommt, muss auch in der Parameterliste stehen.

Algorithmus 2 Beispiel für ein Aktionsschema

ACTION(Flug(p, von, nach),
PRECOND: In(p, von) $\wedge$ Flugzeug(p) $\wedge$ Flughafen(von) $\wedge$ Flughafen(nach),
EFFECT: $\neg$ In(p, von) $\wedge$ In(p, nach)).

3.4 Geordnetes Planen

Im Gegensatz zur total geordneten Plansuche - Beispiele hierfür sind die vorwärts bzw. rückwärts gerichtete Zustandssuche -, die Folgen von strikt linear geordnete Aktionen untersucht, schafft man es mittels teilweise geordnetem Planen, sich die Zerlegung des Problems zunutze zu machen. Die strikt linear geordneten Aktionen bei der total geordneten Plansuche sind entweder mit einem Ausgangspunkt oder einem Ziel verbunden und können dadurch aus der Zerlegung des Problems in mehrere Unterprobleme keinen Vorteil ziehen. Hier wird für jedes Teilproblem die optimale Lösung gesucht. Danach werden diese Teillösungen zusammengesetzt und ergeben so die optimale Lösung des ganzen Problems. Ein weiterer Vorteil dieses Ansatzes ist, dass sich der Problemlöser zuerst “wichtigen” Entscheidungen widmen kann und nicht auf die chronologische Abfolge der Teilprobleme Rücksicht nehmen muss. Man spricht hierbei vom Prinzip der “geringsten Bindung”. Der Grad der Bindung wird formal nicht definiert (vgl. Kapitel IV.11 in [58]). Die graphische Darstellung aller linear geordneten Aktionen bei der total geordneten Plansuche nennt man “*Vollständig geordneten Plan*” (engl. Totally Ordered Plan, TOP).

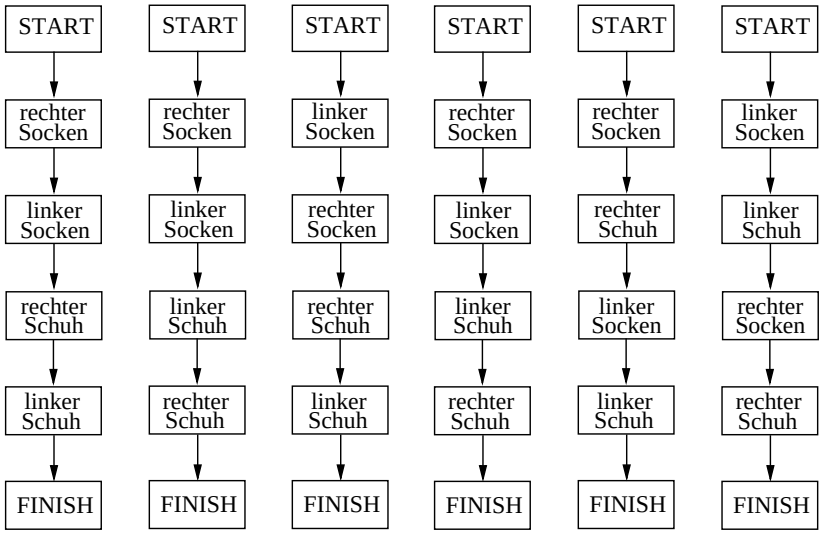

Abbildung 3.1: Vollständig geordneter Plan - graphische Darstellung

Als Beispiel für einen teilweise geordneten Plan kann das Anziehen von Schuhen angeführt werden. Um das Ziel zu erreichen ist es von keinerlei Bedeutung, ob zuerst der linke oder der rechte Schuh angezogen wird. Dieses Problem kann in zwei Teilprobleme zerlegt werden: linken Schuh anziehen und rechten Schuh anziehen.

Zuerst sollte die jeweilige Socke angezogen werden und danach der Schuh. Eine formale Darstellung dieses Problems findet man in Algorithmus 3.

Algorithmus 3 Geordnetes Planen: Beispiel

GOAL(*rechterSchuhAn* $\wedge$ *linkerSchuhAn*).
INIT()
ACTION(*rechterSchuh*,
PRECOND: *rechterSockeAn*,
EFFECT: *rechterSchuhAn*).
ACTION(*rechterSockeAn*,
PRECOND: ,
EFFECT: *rechterSockeAn*).
ACTION(*linkerSchuh*,
PRECOND: *linkerSockeAn*,
EFFECT: *linkerSchuhAn*).
ACTION(*linkerSockeAn*,
PRECOND: ,
EFFECT: *linkerSockeAn*).

Den Graphen von Aktionen der teilweise geordneten Plansuche (vgl. Abbildung 3.2), nennt man "*Teilweise geordneten Plan*" (engl. Partial Order Plan, POP). Des Weiteren wurden die Aktionen "Start" und "Finish" eingeführt, um den Beginn und das Ende des Planes zu markieren. Bei einem total geordneten Plan braucht man für dieses Beispiel sechs verschiedene Pläne um alle möglichen Abfolgen der Aktionen darzustellen (vgl. Abbildung 3.1. Man spricht hierbei auch von der Linearisierung des teilweise geordneten Planes.

Um das Ziel bestmöglich zu erreichen, definieren wir einen Partial Order Plan (POP)-Algorithmus. Dieser besteht aus folgenden Komponenten:

- Mit einer Menge von *Aktionen* definieren wir die einzelnen Schritte des Plans. Diese leiten sich direkt von den Aktionen beim Planungsproblem ab. Der leere Plan besteht nur aus den "Dummy"-Aktionen *Start* und *Finish*, die den Anfang bzw. das Ende des Planes markieren. *Start* hat keine Vorbedingungen und wirkt sich direkt auf den Anfangsstatus des Planungsproblemes aus. *Finish* hat das Ziel des Problems als Vorbedingung aber keine Effekte, da es den letzten Schritt darstellt.

- Eine Menge von *Ordnungsbeschränkungen*, die angeben in welcher Reihenfolge welche Schritte ausgeführt werden. $A \prec B$ wird als *A vor B* gelesen und

impliziert, dass A irgendwann vor B ausgeführt wird.

- Eine Menge von *kausalen Verknüpfungen*. Eine kausale Verknüpfung zwischen zwei Aktionen A und B wird als $A \xrightarrow{p} B$ geschrieben und als “A wird vor B ausgeführt” gelesen. Zum Beispiel bedeutet die kausale Verknüpfung rechter-Socken $\xrightarrow{rechterSockenAn}$ rechterSchuh, dass rechterSockenAn ein Effekt von der rechterSocken-Aktion und eine Vorbedingung von rechterSchuh ist.
- Eine Menge von *offenen Vorbedingungen*. Man spricht von einer offenen Vorbedingung, wenn die Vorbedingung durch keine Aktion im Plan ausgeführt wird.

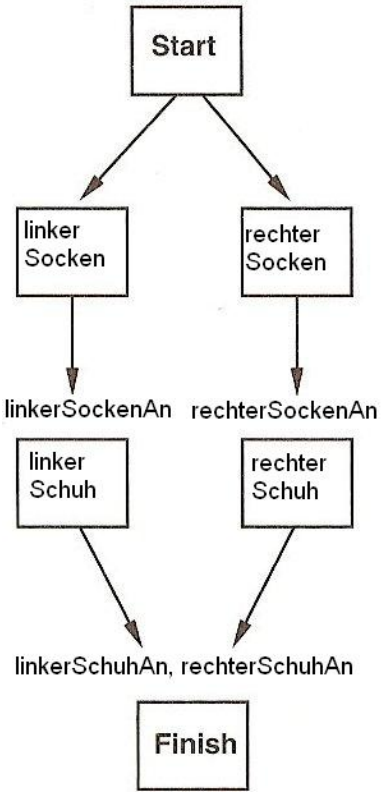

Abbildung 3.2: Teilweise geordneter Plan - graphische Darstellung

Als letzten Punkt in diesem Kapitel soll der *konsistente Plan* beschrieben werden. Ein konsistenter Plan zeichnet sich dadurch aus, dass er keine Zyklen in den Ordnungsbeschränkungen und auch keine Konflikte mit den kausalen Verknüpfungen aufweist. Verfügt ein konsistenter Plan über keine offenen Vorbedingungen, bezeichnen wir ihn als Lösung.

Mit all diesen besprochenen Punkten ist es nun möglich das Suchproblem zu definieren, das der POP-Algorithmus im Allgemeinen lösen soll:

- Der Anfangsplan beinhaltet "*Start*" und "*Finish*", die Ordnungsbeschränkung *Start* $\prec$ *Finish* sowie keine kausalen Verknüpfungen. Des Weiteren treten alle Vorbedingungen von Finish als offene Vorbedingungen auf.
- Die Nachfolgefunktion sucht sich eine offene Vorbedingung *p* für einen Aktion *B* heraus, die zufällig gewählt wird. Damit erstellt sie einen Nachfolgeplan für jede mögliche konsistente Lösung, die es ermöglicht nach Wahl einer Aktion *A p* zu erreichen. Konsistenz kann mit folgenden Mitteln erreicht werden:
  1. Der kausale Link $A \xrightarrow{p} B$ und die Ordnungsbeschränkung $A \prec B$ werden dem Plan hinzugefügt. Aktion A kann entweder schon im Plan existieren oder eine neue Aktion sein. Wenn sie neu ist, wird sie - so wie $Start \prec A$ und $A \prec Finish$ - zum Plan hinzugefügt.
  2. Alle Konflikte zwischen neuen kausalen Verknüpfungen sowie allen existierenden Aktionen und zwischen der Aktion A (wenn sie neu ist) und den bereits bestehenden kausalen Verknüpfungen müssen gelöst werden. Ein Konflikt zwischen $A \xrightarrow{p} B$ und C wird gelöst, indem C irgendwo außerhalb der kausalen Verknüpfung in Erscheinung tritt, entweder dadurch, dass $B \prec C$ oder $C \prec A$ hinzugefügt wird. Manchmal werden kausale Verknüpfungen auch Schutzintervall genannt, weil die Verknüpfung $A \xrightarrow{p} B$ *p* davor schützt ausserhalb des Intervall A bis B zunichte gemacht zu werden.
- Mit dem Ziel eines Planes wird überprüft, ob der Plan eine Lösung für das originale Planungsproblem ist. Da nur konsistente Pläne erstellt werden, reicht es aus zu testen, ob es offenen Vorbedingungen gibt oder nicht.

3.5 Implizite Interaktion und Personalisierung

Implizite Interaktion ist ein Teilgebiet der Mensch-Computer-Interaktion (im Englischen: Human Computer Interaction - HCI). In der HCI-Forschung werden direkte und indirekte Interaktion unterschieden. Von direkter Interaktion spricht man, wenn sich die Akteure zur gleichen Zeit am gleichen Ort befinden. Als Beispiel direkter Kommunikation kann man zum Beispiel Face-to-Face-Kommunikation anführen [38].

Bei indirekter Interaktion sind die Aktoren zeitlich und/oder örtlich verschieden situiert [15]. Kommunizieren mehrere Menschen miteinander wird oft implizite Information, also die Art von Information, die nicht direkt ausgedrückt wird, übertragen. Dazu zählen verschiedene Gesten. So wird z. B. mit einem Nicken Zustimmung signalisiert.

Mit den derzeit gegebenen technischen Möglichkeiten wird die Interaktion mit Computern standardmäßig noch explizit durchgeführt. Der Benutzer teilt dem Computer mit einem gewissen Abstraktionsgrad mit, welche Aktionen er ausführen soll. Dies kann über eine Kommandozeile, Sprachbefehle oder direkte Manipulation einer graphischen Benutzeroberfläche geschehen.

Im Gegensatz dazu steht die implizite Interaktion: Sie wird laut [54] als eine Aktion definiert, die zwar vom Benutzer ausgeführt wird, aber in erster Linie keine Interaktion mit dem computerisierten System anstrebt. Nichtsdestotrotz wird diese Aktion vom System als Input verstanden. Es wird vorausgesetzt, dass der Computer diese Aktion abhängig vom jeweiligen Verhalten in der jeweiligen Situation interpretieren kann. Um implizite Interaktion richtig einzusetzen, kann man sich dreier Konzepte bedienen:

1. die Fähigkeit eine Vorstellung von der Verwendung, der Umgebung und den Umständen zu haben,
2. Mechanismen um die von Sensoren gelieferten Werte zu interpretieren und diese Schlußfolgerungen über die Situation zu ziehen,
3. Anwendungen, die diese Informationen nutzen können.

(1) und (2) können als situationsbezogene Inhalte beschrieben werden, bei (3) handelt es sich um Anwendungen, die auf Inhalten aufbauen.

3.5.1 Kontext

In der vorliegenden Arbeit verwenden wir “situationsbezogene Inhalte”. Sie beschreiben implizit Fragmente der verwendeten Interaktion, wie zum Beispiel

- Einsatzort,
- Umgebungsbedingungen oder
- den Zustand eines Devices.

Mit dem Wort “Kontext” kann eine Vielzahl von Dingen beschrieben werden. Verschiedene Sparten der Computerwissenschaften, so wie Künstliche Intelligenz, Verarbeitung von natürlicher Sprache, Bilderkennung und in letzter Zeit natürlich immer mehr Mobile Computing, haben ihre eigene Definition von “Kontext”. Anbei zwei Definitionen von “Kontext” :

- Das, was etwas umgibt und für jemand anderen eine Bedeutung hat.[2]
- 1: Diskurs, der eine Spracheinheit umgibt und hilft ihre Interpretation festzulegen. 2: Eine Menge von Tatsachen oder Umständen, die eine Situation, oder ein Ereignis umgeben.[3]

Seit dem vermehrten Einsatz von mobilen Endgeräten (z.B. Notebooks, PDA's , Handhelds, etc.) kann eine deutliche Reduktion der Arbeitszeit am PC festgestellt werden. Ein typisches Anwendungsszenario beim traditionellen Mobile Computing wäre ein Vertreter, der mit seinem mobilen Endgerät zur potentiellen Kundschaft geht und auf diesem seine Produkte präsentiert. Des Weiteren werden mobile Anwendungen oft genutzt, während gleichzeitig etwas anderes erledigt wird. Dadurch ist es wünschenswert, dass die explizite Mensch-Maschine-Interaktion auf ein Minimum reduziert wird. Daher ist das Wissen über situationsbezogene Kontexte von größter Bedeutung für die Anwendung. Man kann folgende Eigenschaften einer Anwendung feststellen:

1. Die Anwendung läuft auf einem bestimmten Gerät (Typ, CPU, Memory, Diplaygröße, Betriebssystem, etc.),
2. zu einer bestimmten Zeit,
3. wird von einem oder mehreren Benutzern verwendet (single use, shared use),
4. in einer bestimmten physikalischen Umgebung (Ort, etc.),
5. in einem sozialen Umfeld und
6. um eine bestimmte Aufgabe zu lösen.

(1) bis (6) werden als die Grundbausteine des Kontexts bezeichnet. Für mobile Applikationen sind v.a. (4) und (5) von großem Interesse. Hierbei kann sich auch die phyische Umgebung ändern während eine Applikation ausgeführt wird. Ein Beispiel hierfür wäre, dass man telefoniert während man vom Büro in die Garage geht.

[2] Free Online Dictionary Of Computing, http://foldoc.org/
[3] WordNet 3.0, http://wordnet.princeton.edu/

3.5.2 Identifikation von impliziter Mensch-Computer-Interaktion

Implizite HCI kann anhand folgender Fragen identifiziert werden:

- Was passiert um die Applikation herum, während diese ausgeführt wird? Gibt es überhaupt Veränderungen?
- Trägt das Umfeld der Applikation (Verhalten, Umgebung, Umstände) wertvolle Informationen mit sich? Haben sie Einfluss auf die Applikation?
- Gibt es Mittel um die Information zu extrahieren, so dass sie in einem akzeptablen Rahmen bzgl. z.B. Sensorkosten, Prozesskosten, etc. liegt?
- Wie wird die Information verstanden (Semantik)? Welche Interpretation und Argumentation ist möglich und brauchbar? Was ist eine geeignete Reaktion für die Applikation?

3.5.3 Wahrnehmung / Awareness

Um ein Gerät mit den Fähigkeiten auszustatten, die Umgebung wahrzunehmen, gibt es mehrere Möglichkeiten. Dies können simple Sensoren sein, die die Lage eines Gerätes feststellen können, bis hin zu komplexer Audio- und Videoanalyse.

Die Anwendungen, die auf den Geräten laufen, kann man bzgl. ihrer Anwendung grob in folgende Gruppen einteilen:

- Datenbanken (Notepad: Aufschreiben von Notizen, Kalender: Nachschauen, ob und was an einem bestimmten Datum eingetragen ist, ToDo-Listen, Adressbücher, Profile, etc.)
- aktive Umgebungen deren Inhalte durch die Verwendung von Sensoren erkannt werden (Sensor Badges, GPS, Infrarot-Netzwerke, Kameras, etc.)

A. Dey und G. Abowd haben Context-Awareness wie folgt definiert [7]:

> *A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.*

Sie behaupten, dass ein System context-aware, also kontextsensitiv, ist, wenn es dem Benutzer Information und/oder Services zur Verfügung stellt, wobei die Relevanz der Services von der Aufgabe des Benutzers abhängt. Ein solches System stellt

dem Benutzer verschiedene Informationen zur Verfügung, die es über Sensoren (Kameras, Bewegungsmelder, Thermometer, etc.) in einer smarten Umgebung erhält. In der hier entwickelten Anwendung ist ein Beispiel für ein Service, das zur Verfügung gestellt wird, der Meetingsupport. Informationen über Meetings und andere Benutzer werden dem Benutzer nach expliziter Anforderung zur Verfügung gestellt. Dadurch soll eine höhere Usability (Anwendbarkeit) dieser Systeme erreicht werden.

4 Verwandte Arbeiten

Da die Arbeit sich mit Agentensystemen beschäftigt, werden nun einige schon existierende Anwendungen von Agenten vorgestellt.

EasyMeeting ist einer der ersten Smart Meeting Rooms, die unter der Verwendung der FIPA-Agententechnologie entwickelt wurden. Der *Context-Aware Office Assistant* hat es sich zum Hauptziel gemacht, den Inhaber eines Büros in Bezug auf die Besucher, die zu ihm kommen und deren Termine zu verwalten. Das *Helsinki Prototype System* ist das erste System zur Unterstützung von Besprechungen, das bei bilateralen diplomatischen Treffen zum Einsatz gekommen ist. *Digital-Me* soll die ubiquitäre Interaktion mit “aktiven Umgebungen” unterstützen. Im letzten Punkt dieses Kapitels werden noch einige weitere Anwendungen vorgestellt, die in smarten Umgebungen zum Einsatz kommen.

4.1 EasyMeeting

EasyMeeting ist ein Smart Meeting Room, der an der Universtität von Maryland im Jahre 2004 entwickelt wurde um die Verwendung der FIPA-Agententechnologie, Semantic Web Ontologien, logischem Schließen und Sicherheits- sowie Datenschutzrichtlinien zu demonstrieren [21].

Ein Smart Meeting Room ist ein verteiltes System, das aus mehreren Gemeinschaften von intelligenten Agenten, Services, Geräten und Sensoren besteht, die ein gemeinsames Ziel verfolgen, nämlich den Meetingteilnehmern relevante Dienste und Informationen zur Verfügung zu stellen. Dies können je nach Zusammenhang Redner, Zuhörer und Organisatoren sein. Typische Anwendungen eines Smart Meeting Rooms sind:

- automatisches Darstellen von Informationen auf einem Whiteboard,
- Sammeln von Informationen über die Teilnehmer,
- automatisches Reorganisieren von Meetingplänen sowie

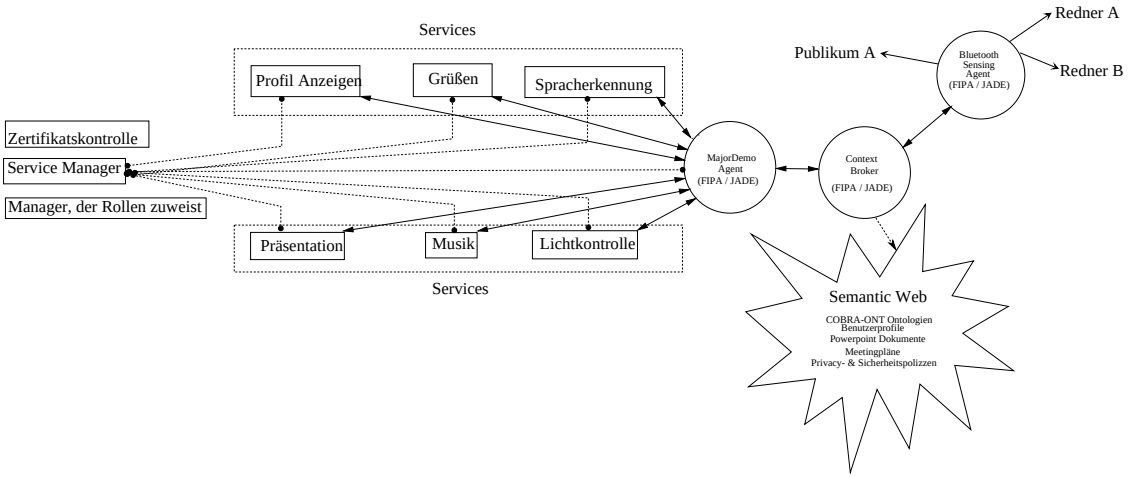

Abbildung 4.1: Übersicht Easy Meeting nach [21].

- Sammeln von Informationen zu den Meetings.

Um das Modellieren des Inhaltes, das Teilen von Wissen, Context Sharing und den Schutz der Sicherheitsrichtlinien der User zu vereinfachen, wurde die Context Broker Architecture (CoBrA) entwickelt.

In Abbildung 4.1 wird ein Überblick über die EasyMeeting-Architektur gegeben. Der Context Broker teilt sein Wissen mit dem MajorDomo-Agent. Aus diesem Wissen sucht sich der MajorDomo-Agent relevante Services aus und ruft geeignete Services des Service Managers auf, der diese relevanten Dienste dem Publikum und dem Redner zur Verfügung stellt.

4.1.1 Context Broker Architecture

CoBrA ist eine Agentenarchitektur, die inhaltsbewusste Systeme in smarten Umgebungen unterstützt. Mittelpunkt dieser Architektur ist ein Context Broker. Das ist ein intelligenter Agent, der auf einem stationären Computer mit ausreichend Ressourcen läuft. Er ist für die Akquisition Beibehaltung von Wissen und Zusammenhängen verwantwortlich. Des Weiteren kümmert er sich um Wissen und Informationen, die nicht direkt von den Sensoren abgelesen werden können sowie um Inkonsistenzen im gespeicherten Wissen.

CoBrA besteht aus mehreren Komponenten:

Context Knowledge Base: In der Wissensbasis wird das Wissen über den Kontext persistent gespeichert. Das komplette Wissen wird in Resource Description Fra-

mework (RDF)-Tripel (z.B. Subjekt, Eigenschaft, Objekt) gespeichert. Diese Tripel werden in einer relationalen Datenbank gespeichert. RDF ist eine relationale Sprache zur Bereitstellung von Metadaten im World Wide Web.

Context Reasoning Engine: Die Context Reasoning Engine ist eine regelbasierte Inferenzmaschine, die es ermöglicht aus kontextrelevanten Informationen Folgerungen zu ziehen. Wissen kann auf zwei Arten abgeleitet werden, entweder über Ontologieinferenzen oder über Heuristiken aus dem Anwendungsgebiet.

Context Acquisition Module: Dieses Modul ist eine Bibliothek von Prozeduren, die Informationen aus unterschiedlichen Quellen sammeln. Ziel ist es eine Middleware auf Abstraktionsebene zu generieren um die niedrige Komplexität bei der Akquisition von Inhalten zu verstecken.

Policy Management Module: Dieses Modul beinhaltet Inferenzregeln um zu entscheiden, ob ein Agent Zugriff auf einen bestimmten Typ von Informationen über einen User hat oder nicht.

4.1.2 Semantic Web Ontologien in CoBrA

Die CoBrA-Ontologie wurde mit der Web Ontologie Sprache OWL definiert. Sie hilft unabhängig voneinander entwickelten Agenten ihr Wissen zu teilen und die Kosten um Zusammenhänge zu erkennen und zu minimieren.

Benutzerprofil: Das Benutzerprofil kann verwendet werden, um nützliche Informationen für Schlußfolgerungen bezüglich der Rolle des Benutzers und seiner Aktivitäten zu ziehen. Ein typisches Benutzerprofil beinhaltet Hintergrundinformationen des Benutzers - also Kontaktdaten, Informationen über den Beruf, Vorlieben, soziale Beziehungen zu anderen Personen und Informationen zu den mobilen Geräten des Benutzers - sowie des persönlichen Agenten. Des Weiteren sollten Informationen über den täglichen Meetingplan, sowie den damit verbundenen Rollen und Prioritäten bei der Planung von Meetings berücksichtigt werden.

Meetingpläne: Wenn der Context Broker über Informationen bezüglich des Zeitplans der Meetings verfügt, kann dies hilfreich sein, damit Rückschlüsse über Zusammenhänge zwischen den Meetings gezogen werden können. Dies beinhaltet den aktuellen Zustand eines Meetings, die Zahl der Teilnehmer, sowie deren Rollen im Meeting.

Zeit und Raum: Hiermit kann der Context Broker zum Beispiel feststellen, in welcher zeitlichen Abfolge welche Meetings in welchen Raum stattfinden und somit diese nach bestimmten Kriterien ordnen. Grundlegend hierbei ist die Entität *Zeit*. Sie kann entweder ein fixer Zeitpunkt oder ein Intervall sein. Ein einfaches Beispiel für eine räumliche Beziehung ist die Relation eines Teil zum Gesamten, in welcher Entitäten zueinander in Beziehung stehen.

4.1.3 Gemeinsamkeiten bzw. Unterschiede zu dieser Arbeit

EasyMeeting verfügt über viele Gemeinsamkeiten mit der in dieser Arbeit vorgestellten, entwickelten Anwendung. Bei beiden Arbeiten handelt es sich um ein Meetingszenario, bei der die FIPA-Agententechnologie Verwendung findet. Beide Anwendungen verfügen über ähnliche Funktionen wie das zur Verfügung stellen von meetingrelevanten Informationen für die Benutzer oder das Reorganisieren von Meetingplänen. Was EasyMeeting fehlt, ist die Unterstützung von Langzeitzielen, wie die Implementierung einer Karriereleiter, bei der die einzelnen Benutzer - abhängig von der Anzahl der besuchten Meetings, gehaltenen Präsentationen, durchschnittliche Bewertung, etc. - vom System eine Position auf der Karriereleiter zugewiesen bekommen. Des Weiteren wird bei der vorliegenden, neuen Anwendung die Zeit im Meeting in die Aktionen miteinberechnet. Sollte ein Meeting überzogen werden, so wird überprüft, ob es zu irgendwelchen Überschneidungen mit anderen Meetings kommt, und sollte dies der Fall sein, wird der Zeitplan so umorganisiert, dass dies nicht mehr der Fall ist.

4.2 Context-Aware Office Assistant

Der Context-Aware Office Assistant soll den Eigentümer eines Büros unterstützen, indem er mit den Besuchern des Büros interagiert und den Zeitplan des Eigentümers regelt [64]. Hauptaufgabe dieses Assistenten ist es, den Besuchern Informationen über ihre Termine im Büro bereitzustellen bzw. diese auszumachen sowie den elektronischen Kalender auf den neuesten Stand zu bringen. Grundlage für die Entwicklung dieses Agenten ist eine Studie über das Verhalten von Leuten an der Tür des Büros. Ergebnis war, dass das Verhalten sowohl mit dem Status der Bürotür (offen, halb offen, geschlossen), als auch mit dem Verhalten der Leute und ihrer Rolle gegenüber dem Besitzer des Büros zusammenhängt. Wenn die Tür geschlossen ist,

wählen die meisten Besucher die Aktion “anklopfen und warten”, “Überprüfen des Status” oder “Hinterlassen einer Nachricht”.

4.2.1 Funktionsweise

Der Agent befindet sich auf einem Computer außerhalb des Büros und sammelt Daten, damit er mit den Besuchern interagieren kann. Ein zweites Programm läuft auf dem Computer des Büroeigentümers und interagiert mit ebendiesem, während es über eine TCP/IP-Verbindung mit dem Computer außerhalb des Büros kommuniziert. Der Kalender des Büroeigentümers kann vom Agenten eingesehen und verändert werden.

Auf jeder Seite der Bürotür befindet sich eine Matte, die mit Drucksensoren ausgestattet ist. Damit kann festgestellt werden, ob ein Benutzer gerade auf dem Weg ins Büro oder aus dem Büro ist, oder ob er auf der Türschwelle steht. Dadurch weiß der Agent auch, wie viele Besucher sich im Moment im Büro aufhalten.

Durch kontextbezogene Informationen kann der Agent sein Verhalten gegenüber einem Besucher anpassen. Sollte ein Besucher ohne vorherigen Termin im Büro auftauchen, überprüft der Agent, ob sich der Eigentümer des Büros gerade in einem Meeting befindet (feststellbar dadurch, dass mehr als eine Person im Büro anwesend ist). Sollte dies der Fall sein, wird der Agent den Besucher bitten, eine Nachricht zu hinterlassen und/oder einen Termin auszumachen. Ist der Eigentümer des Büros alleine, wird der Agent ihn Fragen, ob den Besuch empfangen will, oder nicht.

Kontextbezogene Informationen in diesem System beinhalten:

- die Identität des Besuchers
- den Status des Zeitplans des Büroeigentümers
- den Status des Beschäftigungsgrades des Büroeigentümers
- die Bereitschaft, ob der Büroeigentümer den Besuch empfangen will, oder nicht.

4.2.2 Gemeinsamkeiten bzw. Unterschiede zu dieser Arbeit

Dieser Agent kann unabhängig vom Benutzer (in diesem Fall der Inhaber des Büros) mit den Gästen kooperieren. Mit seinem Verhalten unterstützt er den Inhaber des Büros und kann für die Gäste die jeweils optimale Aktion auswählen. Des Weiteren stellt das System den Terminkalender für den Inhaber des Büros bereit. Der Agent

verhandelt mit den Benutzern über neue Termine und Änderung von bestehenden Terminen. Anhand dieser Aktionen passt der Agent den Terminkalender ständig an und sorgt somit dafür, dass es keine Überschneidungen im Meetingzeitplan gibt.

Die Gemeinsamkeit zwischen den Arbeiten besteht darin, dass der Agent sich um den Terminplan des Benutzers kümmert. Des Weiteren wissen beide Agenten immer ob und wie viele Personen sich in Reichweite der Agenten befinden. Ein Karrieremodus ist beim Context-Aware Office Assistant nicht implementiert. Der Agent des Context-Aware Office Assistants bezieht seine Informationen über die Anwesenheit von Benutzern aus den Sensoren der Fussmatte. In der hier vorgestellten Arbeit nimmt der Agent an, dass sich der Benutzer in der Meeting-Umgebung befindet, sobald er sich über das WLAN beim Server angemeldet hat.

4.3 Helsinki Prototype System

Beim Helsinki Prototype System handelt es sich um das erste Meeting Support System, das multilaterale diplomatische Treffen unterstützen soll [45]. Es wurde 1995 an der Universität von Jyväskylä in Finnland von Petri Maaranen und Kalle Lyytinen in Zusammenarbeit mit dem finnischen Außenministerium entwickelt. Erstmals eingesetzt wurde es beim vierten Nachfolgetreffen der Konferenz für Sicherheit und Zusammenarbeit in Europa[1].

4.3.1 Ziele des Projekts

Dieses Projekt verfolgt zwei Hauptziele: zum einen soll die Funktionalität von Gruppen, ihren Aufgaben und personenbezogenen Eigenschaften in Bezug auf multilaterale Diplomatie begriffen werden, zum anderen soll die Entwicklung von unterstützenden Computersystemen für diese Treffen erfolgreich abgeschlossen werden.

4.3.2 Anforderungen

In einem Raum gab es mehrere Arbeitsplätze, die jeweils mit einem Computer ausgestattet waren. Darüber hinaus waren zwei große Bildschirme angebracht, auf denen die ausgearbeiteten Ergebnisse des Treffens dargestellt wurden. Die Bildschirme wurden so angebracht, dass sie von allen teilnehmenden Personen gut gesehen wer-

[1] http://www.osce.org/

den konnten. Der Leiter des Meetings steuerte, was gerade auf den Bildschirmen dargestellt wurde.

Da sich die Entwickler in einem sehr straffen Zeitplan befanden, kam ein bereits vorhandenes Softwarepaket, die GroupSystems Software[2], zum Einsatz. Nichtsdestotrotz sollte die Software je nach Anonymität der Benutzer angepasst werden können. Die Delegierten in diesem Meeting nehmen keinen Einfluss auf größere Entscheidungen während des Meetings. Wichtiger für sie war eine endgültige Antwort, ob ihre Vorschläge abgelehnt oder angenommen wurden, oder Vorschläge, wie sie ihre Verhandlungen fortführen sollten. Die unterstützende Rolle des Programms beinhaltete Funktionen wie Eingabe von Text, Benachrichtigungen und Darstellen von Informationen. Aufgrund dieser Anforderungen kam die GroupSystems - Software, entwickelt an der Universität von Arizona, zum Einsatz. Eine Übersicht über die Design Prinzipien für das Helsinki Prototype System findet man in Tabelle 4.1.

4.3.3 Gemeinsamkeiten bzw. Unterschiede zu dieser Arbeit

Da es sich beim Helsinki Prototype System um das erste meetingunterstützende System handelt, kann es quasi als Prototyp für alle Systeme dieser Art verstanden werden. Mit Hilfe von verschiedenen Geräten, so wie zum Beispiel Bildschirmen, die relevante Informationen darstellen, können die Benutzer im Meeting unterstützt werden.

Beim Helsinki Prototype System ist allerdings keine Möglichkeit vorgesehen, bei Bedarf den Zeitplan umzuorganisieren. Des Weiteren sind die Benutzerrollen fix vorgegeben. Während beim Helsinki Prototype System nur zwischen Meetingleitern und Meetingteilnehmern unterschieden wird, kann in der hier vorgestellten Arbeit jeder Meetingteilnehmer durch Besuch von Meeting, Halten von Vorträgen, Besuchen von externen Veranstaltungen, etc. seine Position auf der Karriereleiter verbessern und so über mehrere Zwischenstufen bis zum Abteilungsleiter aufsteigen.

Beide Anwendungen haben aber gemeinsam, dass sie über bestimmte Devices Benutzer mit Informationen versorgen. Beim Helsinki Protoype System sind dies die Bildschirme, auf denen meetingbezogene Informationen dargestellt werden, bei der hier vorgestellten Arbeit, sieht der Vortragende die Restzeit des Meetings auf seinem mobilen Device und wird durch Vibrieren des Gerätes auf das nahende Ende des Meetings hingewiesen.

[2] http://www.groupsystems.com/

Design Prinzipien für das Helsinki Prototyp System	**spezifiziertes Design**
Sicherheit	ein nicht zu anderen Netzwerken verknüpftes LAN
komfortables und interessantes Innendesign	keine sichtbaren Computer aufeinander abgestimmte Farben ergonomisches Umfeld keine sichtbaren Kabel
Benutzertransparenz	Benutzer sehen nur Devices (Monitor, Tastatur, Laptops) Bildung von Untergruppen durch die Delegierten
einfache Benutzbarkeit	geleitete Sessions um die volle Teilnahme auch ohne Computer zu gewährleisten nicht zu viele Softwaretools Konzentration auf Textmanagement
optionale Verwendung von Computern	Computer sind verborgen, wenn sie nicht benutzt werden
Visualisierung von Texten	über zwei Videobildschirme anpassbare Bildschirme auf beiden Seiten des Tisches Ändern der Textfarben auf dem Hauptbildschirm
Kontrolle des Raums	Platz für mehr als eine Person pro Land kein fixer Platz für den Meetingleiter
Sichtbarkeit	flügelförmige Tische Laptops anpassbares Licht
akustische Bedingungen	Lautsprecher auf der Decke und an den Wänden Peripheriegeräte (z. B. Computer) werden nicht mehr innerhalb des Raumes positioniert
verschiedene Benutzungssituationen	Modularität der Tische Platz um die Formierung von Untergruppen zu ermöglichen
Dokumentation	Drucker und Kopiergeräte in angrenzenden Räumen vorhanden
Flexibilität des Systems	unterschiedliche Situationen benötigen unterschiedliche Tools; Software liefert viele Tools mit

Tabelle 4.1: Überblick über das Design des Helsinki Prototyps

4.4 Digital-Me

Bei Digital-Me handelt es sich um ein Multiagentensystem, das ubiquitäre und personalisierte Interaktion mit aktiven Umgebungen unterstützt. Entwickelt wurde es an der Universität von Bari in Italien [25].

Dieses Projekt verfolgt zwei Hauptziele: Zum einen soll die Darstellung von Information möglichst an Charakteristiken des Benutzers (dazu zählen Hintergrundwissen, Vorlieben, Geschlecht, etc.) angepasst werden, zum anderen soll die Erfüllung der zur Verfügung gestellten Services bestmöglich erreicht werden.

Dieses System besteht aus zumindest zwei interagierenden Entitäten: einer Umgebung und einem Agent. Die Umgebung bietet verschiedene Services an und interagiert mit einem oder mehreren mobilen Benutzern über einen Personal Agent - ein digitales alter Ego. Als erster Prototyp wurde eine To-Do-Liste mit eingebauter Intelligenz entwickelt. Eine To-Do-Liste ist ein typisches Beispiel einer Applikation, die Personalisierung erfordert und Vorteile aus der Modellierung von Benutzern und Inhalten ziehen kann. Zum Beispiel, wenn die To-Do-Liste die Aufgabe “kaufe Essen vor der Rückkehr nach Hause” enthält und der Benutzer auf dem Heimweg bei einem Supermarkt vorbeikommt, dann wäre es nützlich, wenn der Benutzer daran erinnert werden würde, Essen zu kaufen.

4.4.1 Entwurf der Digital-Me Architektur

Um die Erfüllung der Services zu gewährleisten, wurde eine To-Do-Liste erstellt, in der alle Aufgaben eingetragen werden, die der Benutzer erledigen muss. Sie gibt dem Digital-Me Personal Agent die Möglichkeit diese Aufgaben ganz oder nur in Teilen zu erledigen. Wenn sich der Benutzeragent in einer smarten Umgebung aufhält, die nützliche Services für die Ausführung der Aufgaben bereitstellt, kann der Agent benutzerbezogene Informationen verwenden und so personalisierte Ergebnisse erzeugen. Zum Beispiel kann dies eine Bestätigungsnachricht sein, die erzeugt wird, wenn eine Aufgabe ausgeführt wird, bei der der Agent nicht über die nötige Autonomie verfügt; eine Erinnerungsnachricht wird erzeugt, wenn es ausführbare Aufgaben gibt, etc.

Um all diese Services zur Verfügung stellen zu können, wurde die smarte Umgebung als Organisation von mehreren spezialisierten Agenten modelliert. Das Zusammenspiel dieser Agenten wird in Abbildung 4.2 graphisch dargestellt:

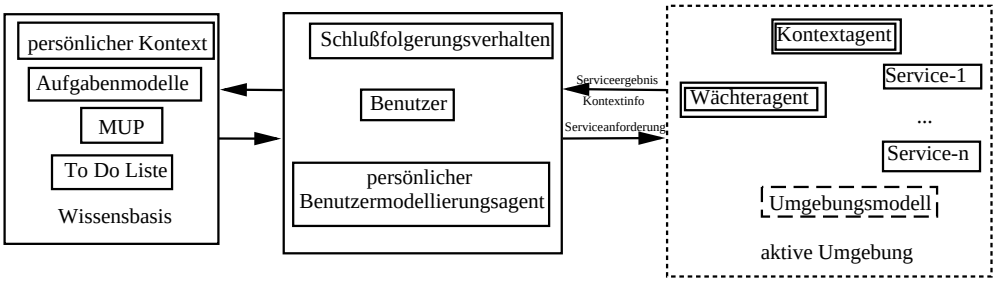

Abbildung 4.2: Digital-Me Architektur nach [25]

Service Agenten: Sie sind darauf spezialisiert, Services zur Verfügung zu stellen und Aufgaben auszuführen.

D-Me Personalagenten: Sie stammen von anderen Personen, die sich in der gleichen Umgebung befinden.

Wächteragent: Der Wächteragent agiert als Moderator und weiß, welche Agenten sich gerade in der Umgebung aufhalten.

Kontextagent: Der Kontextagent kann die Daten im Zusammenhang mit der Umgebung verarbeiten.

Benutzer können auch mit Services aus anderen Umgebungen kommunizieren. Dazu wird ein spezielles Modul, der *EnvKeeper*, benötigt, der das Zusammenspiel zwischen den einzelnen Umgebungen verwaltet.

4.4.2 Der persönliche Digital-Me Agent

Der Digital-Me Agent verfügt über folgende Funktionen:

- vollständige oder teilweise Ausführung der Aufgaben, die in der To-Do-Liste des Benutzers spezifiert sind,
- Erstellung neuer Aufgaben - sofern nötig,
- Beziehung von benutzerrelevanten Informationen um die Ausführung der Aufgaben anzupassen,
- Beziehung von inhaltsrelevanten Informationen um die Ausführung der Aufgaben in einen Zusammenhang zu bringen,

- Kommunikation von persönlichen Ergebnissen.

Um diese gewünschten Funktionen zu erreichen, greift der Agent auf folgende Wissensquellen zu. Eine To-Do-Liste, die die Beschreibung der Aufgaben und ihre Beschränkungen bzgl. Aktivierungsbedingungen, Priorität und Autonomielevel beinhaltet. Eine formale Beschreibung der Aufgabe, die der Agent verwenden kann um diese auszuführen. Das mobile Benutzerprofil beinhaltet situationsbezogene Information über den Benutzer. In der Kontextsituation der Umgebung werden Aussagen über Licht, Lärm, etc. getroffen. Im persönlichen Kontext bzgl. der Applikation werden Ergebnisse, die von den Geräten, die der Benutzer trägt, abgelesen werden (Herzschlag, Temperatur, etc.), gespeichert.

4.4.3 Digital-Me Autonomie

Die Eigenständigkeit eines Agenten wird in folgenden Bereichen sichergestellt:

Eigenständigkeit bei der Ausführung: Aufgaben, Unteraufgaben und Serviceanforderungen, die in der To-Do-Liste gespeichert sind, werden vom Digital-Me-Agenten eigenständig ausgeführt.

Eigenständigkeit bei der Kommunikation: Die Eigenständigkeit bei der Kommunikation bezieht sich auf das Level der Aufdringlichkeit bei der Kommunikation mit dem Benutzer. Es wird der Frage nachgegangen, ob der Agent die Initiative eine Aktion auszuführen jederzeit ergreifen kann, oder ob es bzgl. der Ausführung irgendwelche Beschränkungen gegenüber dem Benutzer oder dem Inhalt gibt. Des Weiteren ist es wichtig, wie viel Einfluss die Nachricht auf das Ergreifen der Initiative hat.

Eigenständigkeit der Verteilung persönlicher Daten: Die Eigenständigkeit der Verteilung persönlicher Daten bezieht sich auf die Autonomie bei der Ausführung von Aufgaben, die die Verteilung persönlicher Daten brauchen, so wie diejenigen Daten, die das Benutzerprofil beinhaltet.

Eigenständigkeit von Ressourcen: Der Agent kann kritische Benutzerressourcen verwenden um Aufgaben auszuführen (Kreditkartennummer, Zeit um Verabredungen zu treffen, etc.).

Jeder dieser hier beschriebenen Dimensionen ist ein Wert zwischen *null* und *hoch* zugewiesen. Ist einer Dimension der *Null-Wert* zugewiesen, so verfügt sie über keine Autonomie. In diesem Fall macht das System nur genau das, was vom Benutzer explizit verlangt wird. Es können keinerlei Schlüsse über Information oder Entscheidungen die Aufgabe zu modifizieren vom System gezogen werden, ohne dass beim Benutzer nachgefragt wird.

Wenn einer Dimension der Wert *hoch* zugewiesen ist, dann verfügt das System über größtmögliche Eigenständigkeit. Es steht dem Agenten vollkommen frei über seine Aktionen zu entscheiden, solange er sich an die vorgegebenen Beschränkungen (z.B. Budgetbeschränkungen, zeitliche Beschränkungen, etc.) hält.

4.4.4 Interaktion mit dem Benutzer

Das Kommunikationsverhalten des Agenten wird verwendet, um mit dem Benutzer zu interagieren und anderen Benutzern die Ergebnisse von Aufgaben, welche Information oder eine für die Ausführung der Aufgaben notwendige Bestätigung verlangen, mitzuteilen. Es werden folgende Gruppen von Kommunikationsaufgaben unterschieden:

Inputanforderungen: zum Beispiel, wenn die To-Do-Liste eine Aufgabe beinhaltet, die zusätzliche Information benötigt um ausgeführt zu werden.

Bereitstellung von Information: Information kann dargestellt werden, wenn sie vom Benutzer explizit angefordert wird oder proaktiv vom Digital-Me Agenten dargestellt wird, weil sie zur ausführenden Aufgabe in einer Beziehung steht.

Bestätigungsanforderung: Wenn die Aufgabe einen Schritt miteinbezieht, der eine Aktion des Digital-Me Agenten verlangt und der Agent nicht über volle Autonomie verfügt, so wird der Benutzer aufgefordert, einen entsprechenden Input zu tätigen.

Notifikationsnachrichten: Der Digital-Me Agent wird bei der proaktiven Ausführung von Nachrichten benachrichtigt. Zum Beispiel, wenn der Agent eine Aktion eigenständig ausführen kann, braucht er den Benutzer nicht um Erlaubnis fragen, ob er diese Aktion ausführen darf, sondern wird ihn nur benachrichtigen.

Erinnerungsnachrichten: Ein typisches Anwendungsbeispiel für das Einkaufsszenario in dem zu Beginn dieses Abschnitts angeführten Beispiel.

Folgende benutzer- und inhaltsbezogene Faktoren werden bei der Kommunikation zwischen Agent und Benutzer berücksichtigt:

Vorlieben des Benutzers: Informationen, die beschafft wurden, werden gefiltert, geordnet und entsprechend den Vorlieben des Benutzers, die in seinem Profil gespeichert sind, dargestellt. Dies können entweder Interessen oder Know-how über ein bestimmtes Gebiet sein.

Aktivität: Wenn ein Benutzer lieber etwas anderes macht als eine der Aufgaben aus der To-Do-Liste, dann wird die Nachricht die Zustellung der Nachricht solange verschoben, bis er die aktuelle Tätigkeit beendet hat.

Standort des Benutzers in der Umgebung: Texte, Bilder und andere Medien können der Umgebung, in der sich der Benutzer gerade befindet, angepasst werden. Bei den Umgebungen wird zwischen öffentlich und privat, laut oder leise, dunkel oder hell, etc. unterschieden.

Emotionen: Der emotionale Status eines Benutzer beeinflusst die Art, in der ihm Informationen präsentiert werden. Zum Beispiel werden in stressigen Situationen kurze Nachrichten bevorzugt [17].

Device: Die Größe des Displays bestimmt, auf welche Weise Information ausgewählt, strukturiert und dargestellt wird. Zum Beispiel sind natürlichsprachliche Texte mehr oder weniger wortreich, komplexe Abbildungen können vermieden oder mit geschriebenen oder gesprochenen Kommentaren substituiert werden.

4.4.5 Gemeinsamkeiten bzw. Unterschiede zu dieser Arbeit

Bei Digital-Me handelt es sich um einen Agenten, der eigenständig Aktionen ausführen kann. Da er genau weiß, welche Aufgaben vom Benutzer erledigt werden müssen, kann er mit Hilfe von Erinnerungsfunktionen die Aktionen des Benutzers bestimmen. So wie in dieser Arbeit vorgestellten Anwendung ist hier ein aktives Eingreifen des Benutzers in das Programm relativ selten nötig, da der Agent von jedem Benutzer ein Benutzerprofil hat, weiß welche Aufgaben zu erledigen sind und mit anderen Agenten bzw. Benutzern kommunizieren kann. Eine weitere Gemeinsamkeit der beiden Applikationen ist, dass sie mit Hilfe des Java Agenten Frameworks JADE implementiert wurden.

Da es sich bei Digital-Me um einen persönlichen Agenten handelt, dessen Hauptziel es ist, den Benutzer ortsungebunden und nicht nur in einer bestimmten Umgebung, so wie bei der hier vorgestellten Arbeit, zu unterstützen, ist die Implementierung einer Karriereleiter nicht nötig. Ein weiterer Grund dafür, dass es bei Digital-Me keine Karrierefunktion gibt, ist, dass es sich um nur zwei beteiligte Entitäten handelt: den Agenten und den Benutzer. Des Weiteren ist bei Digital-Me keine Zeitüberwachungsfunktion implementiert. Da die Ausführung von Aufgaben bei Digital-Me zeitlich nicht gebunden ist, gibt es hier auch keine Termine und Events, die zeitlich umorganisiert werden müssen.

4.5 Treffen, Versammlungen und Events in smarten Umgebungen

In der Arbeit von Anton Nijholt wird der Einsatz von smarten Umgebungen bei Treffen, Versammlungen und Events unter dem Gesichtspunkt der Ambient Intelligence beobachtet [50]. Unter Ambient Intelligence (AmI) versteht man den gemeinsamen Einsatz von ubiquitärem Computing und sozialen sowie intelligenten Schnittstellen. Des Weiteren baut AmI auf Context-Awareness und natürlicher Interaktion auf. Hierbei müssen die Schnittstellen nicht sichtbar sein. Die Umgebung entspricht dem Interface. Es können aber genauso addressierbare Objekte darin vorkommen. Der persönliche Assistent des Benutzers wird meist auf einem PDA oder Tablet PC ausgeführt. Die Umgebung muss über folgende Kriterien verfügen:

Interpretation von Events und Aktivitäten: Soziale und intelligente Interaktionen müssen in einer Umgebung zwischen Menschen, Menschen und Objekten sowie Menschen und eigenständigen Agenten modelliert werden. Der Input kann von Sensoren, die auf Sound, Bilder und Haptik reagieren, kommen. Die Interpretation benötigt eine Zusammenführung aller Modalitäten auf ein Level von Kommentaren und semantischen sowie pragmatischen Darstellungen, die eine weitere Verarbeitung ermöglichen.

Unterstützung in Echtzeit: Basierend auf der Interpretation und den daraus resultierenden Darstellungen der Umgebung müssen die virtuellen Bewohner und ihre smarten Objekte den menschlichen Benutzern und Besuchern Unterstützung in Echtzeit anbieten. Diese Unterstützung kann entweder explizit von den Benutzern angefordert werden oder das Programm bestimmt eine Gruppe von

Benutzern, die Unterstützung erhalten und aus dem vorher gewonnen Wissen Nutzen ziehen können.

Multimedia: Informationen, die von diversen Audio- und Videoquellen bezogen werden, sollen interpretiert und verschmolzen werden, damit die Teilnehmer der Meetings in Echtzeit unterstützt werden können.

Autonome und semiautonome Agenten Autonome Agenten sollten Teil der Ambient Intelligence Umgebung sein. So können zum Beispiel virtuelle Individuen, die von Menschen kontrolliert werden, zu Agenten werden und ihren Besitzer repräsentieren, der sich an einem entfernten Ort befindet.

Kontrolle der Umgebung und ihrer Bewohner: Verwendet man Ambient Intelligence oder smarte Meetingumgebungen so kann man die teilnehmenenden Entitäten beobachten. Alle Aktivitäten, die ein Benutzer durchführt, werden erfasst und diese Informationen können dann verwendet werden, um die beteiligten Entitäten zu kontrollieren und zu beeinflussen.

4.5.1 M4: Multi-Modal Meeting Manager

M4[3] ist ein Projekt, das von der Europäischen Union im Rahmen des 5. Framework Programmes finanziert wurde[4]. Es beschäftigt sich mit der Konstruktion eines Demonstrationssystems, das Strukturierung, Browsen und Abfragen eines automatisch analysierten Meetings ermöglicht. All diese Meetings finden in einem Raum, der mit multimodalen Sensoren ausgestattet ist, statt.

Außer verbaler und nonverbaler Kommunikation zwischen den Teilnehmern, finden viele Ereignisse statt, die für die Interaktion zwischen den Teilnehmern relevant sind. Teilnehmer äußern sich in irgendeiner Form zu jeder Art von Ereignis, die in einem Meeting stattfindet. Daher ist es Ziel des Meetingmanagers, den Input von Mikrophonen und Kameras aufzuzeichnen, und diese Ergebnisse zusammenzufassen und den Teilnehmern zur Verfügung zu stellen. Zum momentanen Stand kann die Meetingumgebung registrieren und interpretieren was gerade stattfindet, ist aber noch nicht aktiv am Geschehen beteiligt. Die Meetingumgebung ist achtsam aber noch nicht proaktiv. Partizipation von der Umgebung in Echtzeit benötigt nicht nur Aufmerksamkeit und Interpretation, sondern auch intelligentes Feedback und proaktives Verhalten. Des

[3] http://www.dcs.shef.ac.uk/spandh/projects/m4
[4] http://www.europartnersearch.net/ist/communities/index.php?page=home

Weiteren wird auch eine Darstellung der Umgebung von multimedialen Informationen von den Teilnehmern verlangt.

4.5.2 AMI: Augmented Multi-party Interaction

Dieses Projekt[5] beschäftigt sich mit neuen multimodalen Technologien um Interaktion von mehreren Parteien untereinander, im Sinne menschlicher Interaktion, im Rahmen von Meetingunterstützung und ferngesteuerter Meetingassistenz zu unterstützen. Hauptziel ist es aus den aufgezeichneten Daten einen größtmöglichen Nutzen bzgl. der Interaktion zu erzielen. Dies soll durch die Entwicklung neuer Werkzeuge für computerunterstützten Support erreicht werden. Folgende Problemstellungen werden im Folgenden kurz vorgestellt:

Meetings verstehen: Welche Meetingcharakteristiken spielen eine Rolle um die Interaktion von mehreren Parteien zu verstehen? Wie sehr hängen sie von Charakteristiken wie Größe, Status, Zielen und Aufgaben der Parteien ab?

Uni- und multimodale Erkennung: In smarten Umgebungen gibt es verschiedene Soundquellen. Darüberhinaus kann es Sprecher geben, die nicht ihre Muttersprache benutzen. Dies sind zwei Probleme bei der Sounderkennung. Videoerkennung hängt von der Rolle, der Position und der Haptik der Personen sowie den Gegebenheiten im Raum ab. All diese Einflüsse sind in der Interpretation des Benutzerverhaltens zu berücksichtigen.

Abstraktion von Inhalt und Multimediapräsentation: Um Meetings interpretieren und erkennen zu können, benötigt man eine “natürliche Strukturierung” des Inhalts. Diese “natürliche Strukturierung” wird durch Erkennung und Interpretation von Meetingfolgen und Indizierung von multimodalen Aufzeichnung erreicht. Zum Beispiel sollten folgende Fragen beantwortet werden können: “Wer sind die Teilnehmer?” “Welche Gebiete werden mit der Agenda abgedeckt?” “Welchen Fortschritt konnte man mit der Diskussion erzielen?” “Wie war die Atmosphäre?”

Ferngesteuerter Meetingassistent: Dies ist ein Hauptziel bei diesem Projekt. Das System erlaubt einem Teilnehmer Events in einem Meeting aus der Ferne zu beobachten, oder an interessanten Punkten automatisch einen Alarm auszulösen.

[5] http://www.amiproject.org/

4.5.3 Gemeinsamkeiten bzw. Unterschiede zu dieser Arbeit

Bei beiden hier vorgestellten Projekten liegt das Hauptaugenmerk darauf, Meetings möglichst ohne aktives Eingreifen von Benutzern zu unterstützen. Da eigenständiges Arbeiten von Programmen auch bei diesem hier vorgestellten Projekt zu den Hauptzielen gehört, wurden diese zwei Projekte ausgewählt. Was die zwei vorgestellten Anwendungen diesem hier vorgestellten Projekt allerdings voraus haben, ist, dass sie hauptsächlich aufgrund von Audio- und Videoerkennung Aktionen der Benutzer interpretieren und dies in den Unterstützungsprozess einfließen lassen zu können.

Eine Einteilung der Benutzer nach verschiedenen Rollen aufgrund ihrer Errungenschaften ist weder bei M4 noch bei AMI vorgesehen. Des Weiteren ist es bei keinem der beiden Projekte möglich, dass der Meetingzeitplan automatisch umorganisiert wird. Die Zeitkomponente wird bei beiden Projekten vernachlässigt.

5 Beschreibung der verwendeteten Technologien

Im Folgenden werden jene Technologien beschreiben, die zur Ausführung des Programms notwendig sind.

5.1 Java

Java[1] ist eine objektorientierte Programmiersprache, die von der Firma Sun Microsystems[2] entwickelt wurde. Durch ihre Plattformunabhängigkeit ist sie auf allen Computern ohne größere Adaptierungen einsetzbar. Der Server läuft mit der Java 2 Platform, Standard Edition, Version 1.5 (J2SE). Auf den mobilen Endgeräten, auf denen der Client läuft, ist die Java MicroEdition (J2ME)[3] installiert.

Die Java MicroEdition ist eine abgespeckte Javavariante. Sie benötigt weniger Ressourcen und wurde speziell für die Verwendung auf Mobiltelefonen und PDA's entwickelt. Definiert wird sie in den Java Specification Requests[4] 30 (Connected Limited Device Configuration(CLDC)) und 37 (Mobile Information Device Profile for the J2ME Platform(MIDP)). Bei der Entwicklung der hier vorgestellten Anwendung traten die größten Probleme bei der Programmierung des Graphical User Interfaces (GUI) und beim Umwandeln von Datentypen in einen anderen auf, da die J2ME nicht alle Methoden der J2SE unterstützt.

Als Basiskonfiguration wird die CLDC häufig eingesetzt, da sie nahezu auf allen mobilen Endgeräten zusammen mit dem Mobile Information Device Profile eingesetzt wird. Bei unserem Beispiel wird MIDP Version 2.0 mit der CLDC 1.1 eingesetzt.

[1] http://java.sun.com
[2] http://www.sun.com
[3] http://java.sun.com/javame/
[4] http://www.jcp.org/en/jsr/detail?id=30, http://www.jcp.org/en/jsr/detail?id=37

5.2 JADE und Leap-Mechanismen und API

Bei JADE handelt es sich um eine Java-basierte Middleware, die die Entwicklung von Multi-Agenten-Systemen ermöglicht [3]. Dies beinhaltet folgende Komponenten:

- Eine Laufzeitumgebung, in der JADE-Agenten ausgeführt werden können. Sie muss bereits aktiv sein bevor ein oder mehrere Agenten eine Anweisung ausführen können.
- Eine Bibliothek von Klassen, die Programmierer dazu benutzen können, Agenten zu entwickeln. Diese Klassen werden weiter unten im Detail beschrieben.
- Ein Set von graphischen Tools, die Administration und Beobachtung von Aktionen der laufenden Agenten ermöglichen.

Einen Überblick über die interne Architektur eines JADE-Agenten findet man in Abbildung 5.1. Die verteilte Architektur wird in Abbildung 5.2 dargestellt.

5.2.1 Hauptpakete von JADE

JADE ist in Java geschrieben und besteht aus verschiedenen Java-Paketen, die dem Anwender sowohl fertige Routinen, als auch Schnittstellen, die nach eigenen Bedürfnissen angepasst werden können, zur Verfügung stellen. Die Pakete werden im Folgenden beschrieben:

jade.core: Das Paket jade.core implementiert den Kernel des Systems. Darin befindet sich die Agentenklasse, die von einem Programmierer implementiert werden muss. Das Verhalten dieses Agenten wird im jade.core.behaviours Packet spezifiziert. Dieses Paket enthält die Behaviour-Klassenhierarchie. Damit werden Aufgaben oder Absichten eines Agenten realisiert.

jade.lang.acl: Hiermit wird die Agent Communication Language, die Kommunikation zwischen Agenten gemäß den FIPA-Spezifikationen, realisiert.

jade.content: Dieses Paket stellt einige Klassen zur Verfügung, die vom Benutzer definierte, inhaltsbeschreibende Ontologien und Sprachen unterstützen.

jade.content.lang.sl: Dieses Paket beinhaltet den SL-Codec (Parser und Encoder).

jade.domain: Dieses Paket beinhaltet alle Entitäten, die für das Management eines FIPA konformen Agenten zuständig sind. Im Speziellen sind das die AMS- und DF-Agenten (siehe weiter unten).

jade.gui: In diesem Paket sind alle Klassen zusammengefasst, die die JADE-GUI erzeugen. Hiermit können Agent-Identifier, Agentenbeschreibungen, ACL-Nachrichten, etc. bearbeiten werden.

jade.mtp: Stellt das Java Interface zur Verfügung, das jedes Message Transport Protocol implementieren sollte, um korrekt in das JADE Framework eingebunden zu werden.

jade.proto: Dieses Paket enthält alle Klassen, die eine FIPA-konforme Interaktion ermöglichen (fipa-request, fipa-query, fipa-contract-net, fipa-subscribe, etc.)

jade.tools: Hierbei handelt es sich Paket von Anwendungen, die die Administration der Agentenplattform vereinfachen. Folgende Tools sind im Moment verfügbar:

Remote Management Agent (RMA): Der RMA ist eine graphische Konsole für das Management und die Kontrolle der Plattform.

Dummy Agent: Der Dummy Agent ist ein Monitoring und Debugging Tool, das aus einem graphischen Benutzerinterface besteht und auf einem JADE-Agenten aufbaut.

Sniffer: Der Sniffer ist ein Agent, der ACL-Nachrichten während sie von einem Agenten zum nächsten unterwegs sind, abhört. Des Weiteren stellt er diese Nachrichten graphisch dar, indem er eine Notation verwendet, die Sequenzdiagrammen in der Unified Modelling Language sehr ähnlich sind.

Introspector: Der Introspector ist ein Agent, der den Lebenszyklus eines Agenten, die Nachrichten, die er sendet und empfängt, sowie die Verhalten, die er gerade ausführt, überwacht.

DF GUI: Die Directory-Facilitator (DF)-GUI stellt die “gelben Seiten” der Agentenplattform dar.

LogManagerAgent: Dies ist ein Agent, der jedes Ereignis innerhalb der Agentenplattform mitloggt.

jade.wrapper: Dieses Paket stellt höhere JADE-Funktionalitäten zur Verfügung, die es dem User ermöglichen JADE als Bibliothek zu benützen.

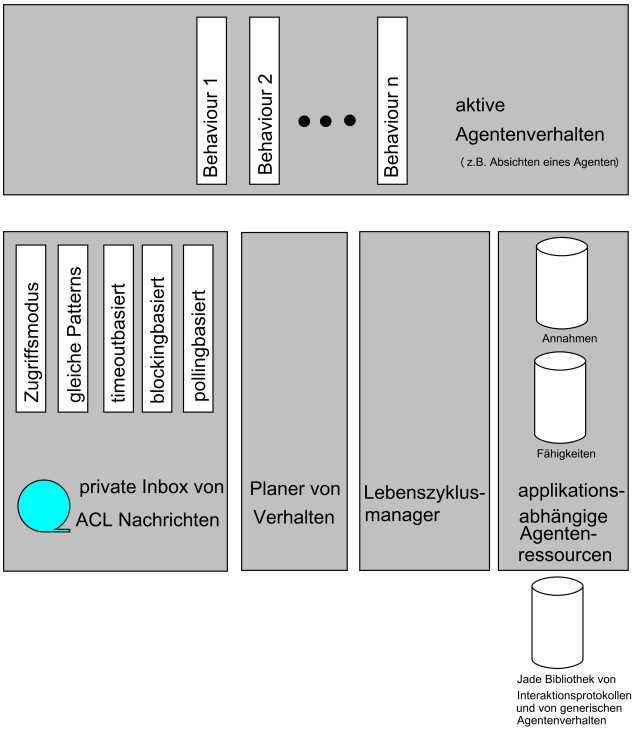

Abbildung 5.1: interne Architektur eines JADE-Agenten

5.2.2 Container und Platform

Als Container wird eine Instanz einer laufenden JADE-Umgebung bezeichnet, die mehrere Agenten beinhalten kann. Mehrere Container werden als Plattform bezeichnet. In jeder Plattform gibt es einen speziellen Hauptcontainer, bei dem sich alle anderen Container registrieren, sobald sie aktiv sind.

5.2.3 Agent-Management-System

Das Agent-Management-System (AMS) ist ein spezieller Agent, der ein Naming Service zur Verfügung stellt. Er stellt damit sicher, dass jeder Agent einen eindeutigen Namen besitzt. Beim Starten eines Agenten wird von JADE vor Ausführung der *setup()*-Methode der Agent automatisch beim Agent Management System registriert. Wird der Agent beendet, so wird nach Ausführung der *takedown()*-Methode der Agent aus der Registrierung des Agent Management Systems gelöscht.

5.2.4 Directory-Facilitator

Der Directory-Facilitator (DF) stellt so etwas wie die “Gelben Seiten” der Plattform dar, d.h. dass ein Agent hiermit andere Agenten suchen kann, die bestimmte Dienste zur Verfügung stellen, damit der Agent seine Ziele erreichen kann. Der Directory Facilitator ist in den Klassen des Pakets *jade.domain.DFService* implementiert. Dieses Paket beinhaltet Methoden zur Registrierung, Löschen aus der Registrierung, Modifikation und Suchen.

5.2.5 Aufbau von Agentenklassen

Jeder Agent besitzt einen eigenen Identifier, der von der Klasse jade.core.AID abgeleitet wird. Mit der Methode *getAID()* wird auf diesen Identifier zugegriffen. Jeder Name eines JADE-Agenten hat folgende Form:

<nickname>@<platform-name>.

Jeder Agent wird von der Klasse jade.core.Agent abgeleitet und implementiert eine *setup()*-Methode, die den Agenten zum Leben erweckt. Soll ein Agent terminieren, wird die *doDelete()*-Methode aufgerufen. Ein Agent kann sich in folgenden Zuständen befinden:

INITIATED: Der Agent wurde bereits erzeugt, hat sich aber selbst noch nicht beim Agent Management System registriert. Er hat weder einen Namen noch eine Adresse und kann nicht mit anderen Agenten kommunizieren.

ACTIVE: In diesem Zustand hat sich der Agent bereits beim Agent Management System registriert, verfügt über einen Namen und eine Adresse und kann alle Funktionen, die JADE zur Verfügung stellt, nutzen.

SUSPENDED Die Ausführung des Agenten wurde gestoppt. Ebenso werden keine Behaviours des Agenten ausgeführt.

WAITING: Der Agent ist blockiert und wartet auf eine Aktion oder Nachricht von einem anderen Agenten.

DELETED: Befindet sich der Agent in diesem Zustand, so ist er definitiv tot. Es werden keine Aktionen mehr an oder mit ihm ausgeführt und der Agent ist nicht mehr beim Agent Management System registriert.

TRANSIT: Wenn mobile Agenten von einem Ort zum nächsten wandern, so befinden sie sich während der Reise im TRANSIT-Zustand. Erhält ein Agent in diesem Zustand Nachrichten, so buffert diese das System und sendet die Nachrichten dem Agenten an seinen neuen Aufenthaltsort nach, sobald dieser dort eingetroffen ist.

JADE stellt Methoden bereit, mit denen der Agent von einem Zustand in einen anderen überführt werden kann.

5.2.6 Behaviours

Das Verhalten eines Agenten wird mit Hilfe eines Behaviours beschrieben, das von der Klasse *jade.core.behaviours.Behaviour* abgeleitet ist. Dazu muss das Behaviour mit der *addBehaviour()*-Methode der Agenten-Klasse hinzugefügt werden. Dies kann zu jeder Zeit entweder in der *setup()*-Methode oder innerhalb eines anderen Behaviours geschehen. Jede Klasse, die von Behaviour abgeleitet ist, muss die *action()*-Methode, die die Operationen spezifiziert, implementieren, sowie eine *done()*-Methode, die darüber Auskunft gibt, ob das Behaviour schon vollständig abgearbeitet wurde, oder nicht.

Es gibt 3 Arten von Verhalten:

One-Shot-Behaviour: One-Shot-Behaviour terminieren sofort nach ihrer Ausführung. Sie werden genau einmal ausgeführt.

Cyclic-Behaviour: Behaviours dieses Typs terminieren niemals. Die *action()*-Methode führt immer exakt dieselbe Operation aus - sofern sie ausgeführt wird.

Generic-Behaviour: Diese Art von Behaviour hängt vom momentanen Zustand des Agenten ab. Je nachdem werden verschiedene Operationen ausgeführt. Es terminiert, wenn eine bestimmte Bedingung erfüllt ist.

Natürlich kann die Ausführung der Verhalten auch zeitlich festgelegt bzw. verzögert werden. Methoden hierzu sind:

Waker-Behaviour: Terminiert, nachdem ein bestimmtes Timeout abgewartet wurde.

Ticker-Behaviour: Führt eine Operation alle n Millisekunden aus.

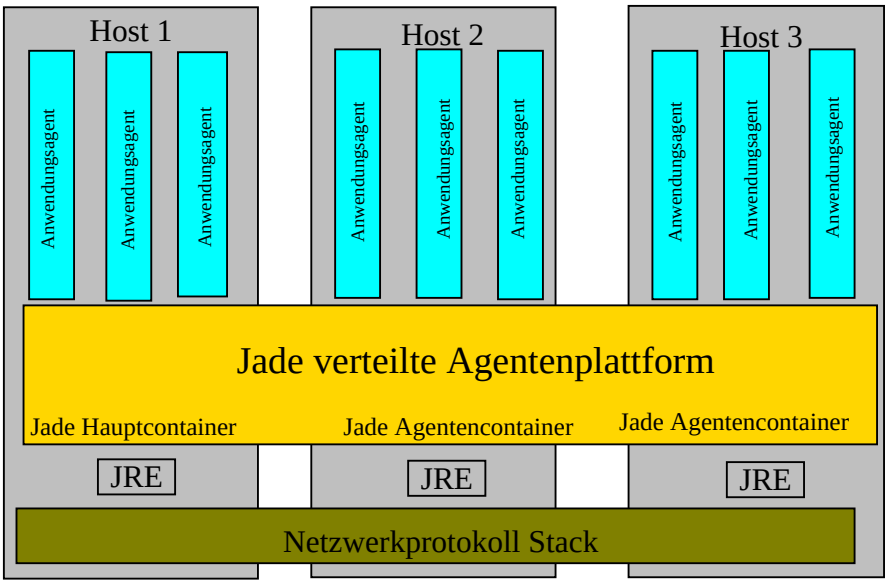

Abbildung 5.2: verteilte Archtitektur eines JADE-Agenten

5.3 Weitere verwendete Technologien

Neben den zwei bereits beschriebenen Technologien - Java und JADE - werden folgende Technologien verwendet, die in zwei Bereiche unterteilt werden können - Device und Software.

5.3.1 Software

In diesem Abschnitt werden das LEAP-AddOn , MySQL und der Apache Webserver überblicksmäßig beschrieben.

LEAP-AddOn: Das LEAP Add-On ermöglicht es JADE-Agenten auf mobilen Plattformen (PDAs und Mobiltelefone) ausgeführt zu werden. Da JADE einen sehr hohen Ressourcenverbrauch aufweist (oft mehrere MB), wurde mit LEAP eine abgespeckte Version entwickelt, die den geringen Rechenleistungen mobiler Plattformen gerecht wird. JADE benötigt mindestens das JDK 1.4. Auf mobilen Endgeräten wird Personal Java oder die MIDP verwendet.

MySQL: MySQL ist ein relationales Datenbankverwaltungsystem, das von der schwedischen Firma MySQL AB[5] entwickelt wurde. Damit ist die Verwaltung von großen

[5]wurde 2008 von Sun Microsystems (http://www.sun.com) übernommen

Datenmengen in einem Netzwerk möglich. Eine Datenbank besteht aus einer oder mehreren Tabellen, in der die Datensätze gespeichert werden können.

In unserem Beispiel sind alle Meeting- und User-relevanten Daten in der Datenbank gespeichert [5].

Apache: Bei Apache handelt es sich um einen Webserver der von der Apache Software Foundation entwickelt wurde. Die Tatsache, dass er auf nahezu allen Betriebsystemen einsetzbar ist, machte ihn zum beliebtesten und meistgenutzten Webserver. Der Apacheserver wird bei uns als Schnittstelle zwischen den Clients und der Datenbank eingesetzt [2].

5.3.2 Device

Das Programm wurde für die Verwendung auf dem N91 von Nokia konzipiert. Das N91 ist ein Smart Device mit integrierter bis zu 4 GB großer Festplatte, WLAN, Bluetooth sowie mit speziellen Tasten für die Musikwiedergabe und einer USB 2.0-Schnittstelle . Es verfügt über ein Farbdisplay mit 262144 Farben bei einer Auflösung von 176 x 208 Pixel. Des Weiteren verfügt es über einen XHTML-Browser für den mobilen Internetzugriff [1].

6 Beschreibung des Programms

In diesem Kapitel wird die grundlegende Funktionalität des Programms beschrieben. Zu Beginn wird kurz auf die verschiedenen Benutzerklassen eingegangen, danach werden alle Use Cases sowie Partial Order Plans und verwendete Algorithmen vorgestellt.

6.1 Benutzerklassen

Für das Meeting-System sind folgende User-Rollen vorgesehen:

Teammitglied: Wird ein Benutzer im System neu angelegt, so wird ihm die Rolle eines “Teammitglieds” zugewiesen.

Projektmitarbeiter: Durch den Besuch einer bestimmten konfigurierbaren Anzahl von Vorträgen, sowie der Erlangung von Spezialwissen wird das Teammitglied nach einer gewissen Zeit zum Projektmitarbeiter.

Vortragender: Hat das Teammitglied eine bestimmte konfigurierbare Anzahl von Vorträgen besucht, einen (externen) Rhetorikkurs besucht, sowie eine bestimmte konfigurierbare Anzahl von Artikeln zu einem Thema abgegeben, steigt er in der Karriereleiter zum potentiellen “Vortragenden” auf.

Projektmanager: Um zum Projektmanager aufzusteigen muss der Projektmitarbeiter über Erfahrung in Organisation und Führung sowie über Spezialwissen in der Projektdomäne verfügen und bereits bei einer bestimmten konfigurierbaren Anzahl von Projekten mitgearbeitet haben.

Meetingmanager: Um vom Vortragenden zum Meetingmanager aufzusteigen, muss der Benutzer eine bestimmte konfigurierbare Anzahl von Vorträgen gehalten haben, sowie bei der Bewertung von den Meetingteilnehmern einen Schnitt von unter 2,0 aufweisen.

Abteilungsleiter: Haben der Projekt- bzw. der Meetingmanager eine bestimmte konfigurierbare Anzahl von Projekten bzw. Meetings geleitet und sie haben nachgewiesen, dass sie über genügend Kundenkontakte verfügen, können sie zum Abteilungsleiter aufsteigen. Somit wurde die höchste, hier vorgesehene, Karrierestufe erreicht.

Die Rolle der Zuhörer bzw. Gäste wird vom System unterstützt, in dem das System ihnen Vorträge und dazu relevanten Links anzeigt.

Auf der nächsten Stufe befindet sich der Vortragende. In diese Position kommt man, wenn man schon einige Vorträge zu einem bestimmten Thema besucht hat bzw. von Beginn an über das Grundwissen verfügt. Das System kann dem Vortragenden helfen, indem es ihm den Besuch eines (externen) Rhetorikkurses bzw. Vorträge, die für ihn interessant sein könnten, um sein nächstes berufliches Ziel zu erreichen, vorschlägt.

Um das Endziel Projekt- bzw. Abteilungsleiter zu erreichen, muss noch eine weitere Zwischenstufe passiert werden. Durch den Besuch einiger Vorträge zu einem bestimmten Gebiet, erlangt der Mitarbeiter Spezialwissen.

Des Weiteren soll es für keinen der Beteiligten möglich sein, sofort nach Einstieg in das Projektteam Projektleiter zu werden. Abhängig von den Qualifikationen und der Anzahl der erfüllten Punkte (s.o.) muss der Mitarbeiter eine bestimmte Zeit im Team verbringen.

Die hier vorgeschlagenen Positionen auf der Karriereleiter sollen als Hilfe für Mitarbeiter in einem Unternehmen dienen. Ihnen sollen damit die Ziele, die notwendig sind, um sich auf der Karriereleiter zu verbessern näher gebracht und vorgeschlagen werden. Da diese Ziele frei konfigurierbar sind, können sie an die Bedürfnisse des Unternehmens angepasst werden. Des Weiteren ist es damit möglich, die Entwicklung der Mitarbeiter zu überwachen.

6.2 Use Cases

In diesem Kapitel werden die einzelnen Use Cases des Programms beschrieben. Für jeden Use Case wurden ein Use Case Diagramm und ein teilweise geordneter Plan (Partial Order Plan) erstellt. Entscheidungen werden in den POP-Diagrammen mittels einer Raute, Aktionen mittels eines Rechtecks dargestellt.

Vorgesehen sind der Use Case *Agenda*, der beschreibt wie man einen neuen Punkt in der Agenda anlegt, sowie die *Karriereleiter*, die die Möglichkeiten zeigt wie man das

endgültige Karriereziel Abteilungsleiter erreichen kann. Der Use Case *MeetingSupport* beinhaltet Meeting unterstützende Funktionen der Applikation.

Im Use Case *Zeitplan umorganisieren* wird die Möglichkeit beschrieben, bei Überziehung oder einer anderen Änderung im Ablauf der Meetings, den Zeitplan umzuorganisieren. Der Use Case *Benutzer einladen* beschreibt den Ablauf, welche Schritte gemacht werden müssen, damit ein User andere zu einem Meeting einladen kann.

6.2.1 Agenda

Um einen neuen Punkt in die Agenda des Meetingplans aufzunehmen, müssen folgende Schritte durchgeführt werden:

Antrag auf neuen Agendapunkt stellen: Auslösendes Ereignis bei diesem Use Case ist, dass ein Benutzer einen Antrag auf einen neuen Agendapunkt stellen will. Soll dieser in die Agenda eingefügt werden, so muss er einen Antrag an das System richten. Nachbedingung dieses Use Cases ist, dass der Benutzer einen Antrag an das System gestellt hat und dieser noch vom System verifiziert werden muss.

Neuen Agendapunkt verifizieren: Nachdem der Benutzer einen Antrag auf einen neuen Agendapunkt gestellt hat, muss dieser erst vom System auf Zulässigkeit geprüft werden, Hier wird entschieden, ob er alle Kriterien erfüllt oder nicht. Stürzt der Server ab, oder erfüllt der Agendapunkt nicht alle notwendigen Kriterien, so wird der Vorgang abgebrochen. Wurde er verifiziert, so kann er angelegt werden.

Agendapunkt anlegen: Ist der Agendapunkt vom System verifiziert worden, so wird er als neuer Eintrag in der Datenbank gespeichert. Sollte bei der Kommunikation mit der Datenbank ein Fehler auftreten, so erhält der Benutzer, der den Antrag stellt, eine Fehlermeldung. Tritt kein Fehler auf, wird der Agendapunkt als neuer Datensatz in der Datenbank gespeichert.

Benachrichtigung aller Beteiligten: Ist der Agendapunkt erfolgreich in der Datenbank gespeichert worden, werden alle beteiligten Benutzer vom System automatisch benachrichtigt. Sollte beim Erstellen oder Versenden der Nachricht ein Fehler auftreten, so wird der Prozess von Neuem gestartet. Wurde die Nachricht erfolgreich versendet, wissen alle beteiligten Benutzer über den neuen Eintrag Bescheid.

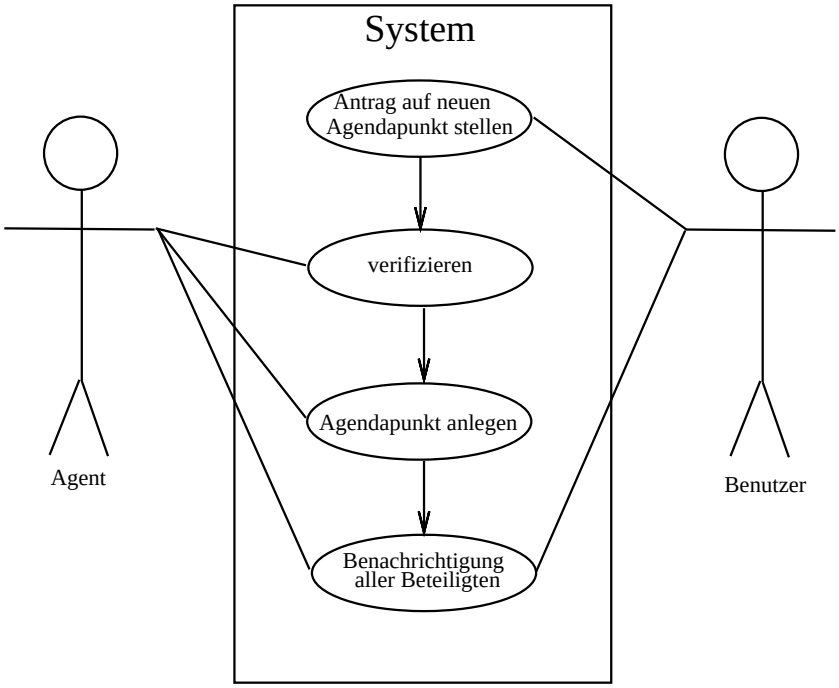

Abbildung 6.1: Use Case: Agenda

Der Benutzer ist nur bei *Antrag auf neuen Agendapunkt stellen* und *Benachrichtigung aller Beteiligten* involviert. *Verifizieren*, *Agendapunkt anlegen* wird vom Agenten alleine ausgeführt und bei der *Benachrichtigung aller Beteiligten* wird die Nachricht vom Agenten an alle Benutzer versendet.

Das zugehörige Use Case Diagramm ist in Abbildung 6.1, der Partial Order Plan in Abbildung 6.2 graphisch dargestellt. Den zum POP gehörenden Algorithmus findet man in Algorithmus 4.

6.2.2 Karriereleiter

Auf der Karriereleiter sind mehrere Karrierestufen vorgesehen, die vom Benutzer durchlaufen werden müssen um das endgültige Ziel, *Abteilungsleiter*, zu erreichen. Hierzu gibt es zwei Möglichkeiten:

Entweder wird er zuerst *Vortragender* und dann *Manager eines Meetings*, oder er schlägt den anderen Weg ein und wird zuerst *Projektassistent* und danach *Projektleiter*. Beide Wege haben die Karrierestufe *Abteilungsleiter* als Endstatus. Aber nun zur Beschreibung der einzelnen Stufen im Detail:

Wird man als neuer Benutzer angelegt, so ist man automatisch “einfaches” *Team-*

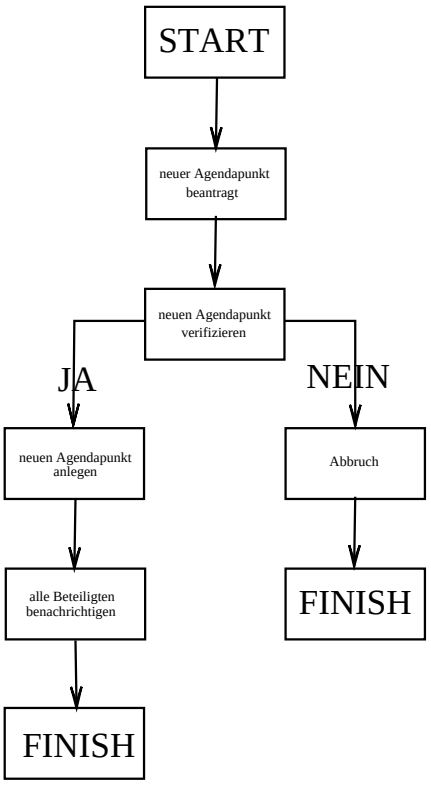

Abbildung 6.2: Partial Order Plan: Agenda

Algorithmus 4 Partial Order Plan: Agenda

Init(NeuerAgendaPunktBeantragt(U)).
Goal(AlleBeteiligtenBenachrichtigt(A)).

Action(AntragAufNeuenAgendaPunktStellen(U),
PRECOND: NeuerAgendaPunktBeantragt(U),
EFFECT: AntragAufNeuenAgendaPunktGestellt(U)).

Action(NeuenAgendaPunktVerifizieren(U, A),
PRECOND: AntragAufNeuenAgendaPunktGestellt(U),
EFFECT: NeuerAgendaPunktVerifiziert(U, A) $\vee$ NeuerAgendaPunktNichtVerifiziert(U, A)).

Action(NeuenAgendaPunktAnlegen(U, A),
PRECOND: NeuerAgendaPunktVerifiziert(U, A),
EFFECT: NeuerAgendaPunktAngelegt(U, A)).

Action(AlleBeteiligtenBenachrichtigen(U, A),
PRECOND: NeuerAgendaPunktAngelegt(U, A),
EFFECT: AlleBeteiligtenBenachrichtigt(U, A)).

Action(Abbruch(U,A)),
PRECOND: NeuerAgendaPunktNichtVerifiziert(U, A),
EFFECT: BeantragungVonNeuemAgendaPunktAbgebrochen(U, A)).

mitglied. Um die Möglichkeit zu erhalten, dass man selbst Vorträge halten kann, muss man darüber hinaus folgende Bedingungen erfüllen. Dies können beispielsweise der Nachweis des Besuches eines Rhetorikkurses, Abgabe von Artikeln zum Thema der besuchten Vorträge sowie eine bestimmte Anzahl von besuchten Vorträgen sein.

Der Nachweis über den Besuch eines Rhetorikkurses sowie der Nachweis über die Abgabe von Papers müssen im System gesondert bekannt gegeben werden.

Um vom *Vortragenden* zum *Meeting Manager* aufzusteigen, muss man einige konfigurierbare Kriterien erfüllen Dazu zählen eine bestimmte Anzahl an gehaltenen Vorträgen, eine durchschnittliche Benotung von > 2 als Vortragender, Erbringung eines Nachweises über Ausbildung in Organisation und Führung sowie geleistete Pressearbeit.

Schlägt man den anderen Weg ein, so ist das nächste Karriereziel *Projektassistent*. Dafür muss man über Spezialwissen in der Projektdomäne verfügen, einige Artikeln, die Themen in der Projektdomäne behandeln, abgegeben haben sowie eine bestimmte Anzahl an besuchten Vorträgen aufweisen können.

Um vom *Projektassistenten* zum *Projektleiter* aufzusteigen, muss den Nachweis von Spezialwissen in einem Bereich, der für das Projekt nützlich ist, erbringen. Des Weiteren muss man eine gewisse Zeit Mitglied im Team sein, über eine Ausbildung in Organisation und Führung verfügen sowie Pressearbeit geleistet haben.

Um das endgültige Ziel *Abteilungsleiter* zu erreichen muss man entweder *Projektleiter* oder *Meeting Manager* sein. Des Weiteren muss man bereits eine bestimmte Anzahl an Meetings gehalten haben bzw. eine bestimmte Anzahl an geleiteten Projekten vorweisen können.

Bei diesem Use Case findet die Überprüfung, ob ein Benutzer bereits eine neue Stufe auf der Karriereleiter einnehmen darf, nach folgendem Muster statt:

Position alt: Nachdem der Benutzer seine aktuelle Position auf der Karriereleiter eingenommen hat, wird vom Programm überprüft, ob er die Kriterien zum Aufstieg in die nächste Position erfüllt.

Überprüfen, ob Kriterien für neue Position erreicht: Abhängig von der aktuellen Position wird an verschiedenen Kriterien überprüft, ob der Benutzer in der Karriereleiter eine Stufe nach oben klettern kann. Ist der Benutzer bereits auf der obersten Stufe in der Karriereleiter angekommen, so ist kein weiterer Aufstieg möglich und es wird daher auch nichts mehr überprüft.

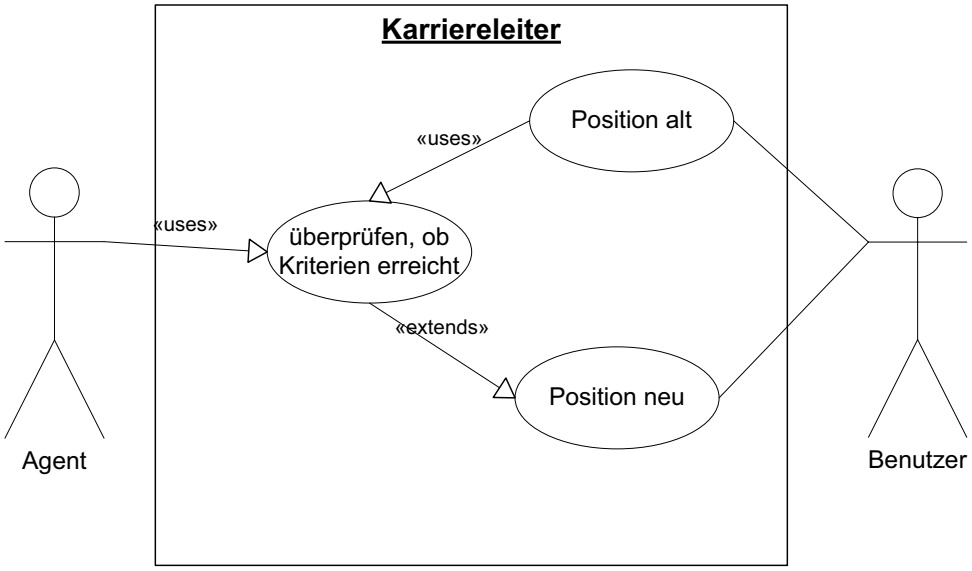

Abbildung 6.3: Use Case: Karriereleiter

Position neu: Nach erfolgreicher Prüfung wurde festgestellt, dass der Benutzer eine neue Position auf der Karriereleiter einnehmen kann. Sollte er die Kriterien für den Aufstieg nicht erfüllen, so bleibt er auf seiner alten Stufe. Hat der Benutzer seine neue Position eingenommen, so kann weiter überprüft werden, ob er bereits die Kriterien für den Aufstieg in die nächste Position erfüllt. Dieser Vorgang wird so lange wiederholt, bis der Benutzer die oberste Stufe auf der Karriereleiter erreicht hat.

Bei diesem Use-Case wird ein goal-orientierter Ansatz verfolgt. Das Hauptziel Abteilungsleiter kann über die Unterziele *Projektassistent* und *Projektleiter* bzw. *Vortragender* und *Meeting Manager* erreicht werden.

Der Use-Case Karriereleiter ist in Abbildung 6.3 graphisch dargestellt, der Partial Order Plan in Abbildung 6.4. Den Algorithmus zum Partial Order Plan für den Use Case Karriereleiter findet man in Anhang B.

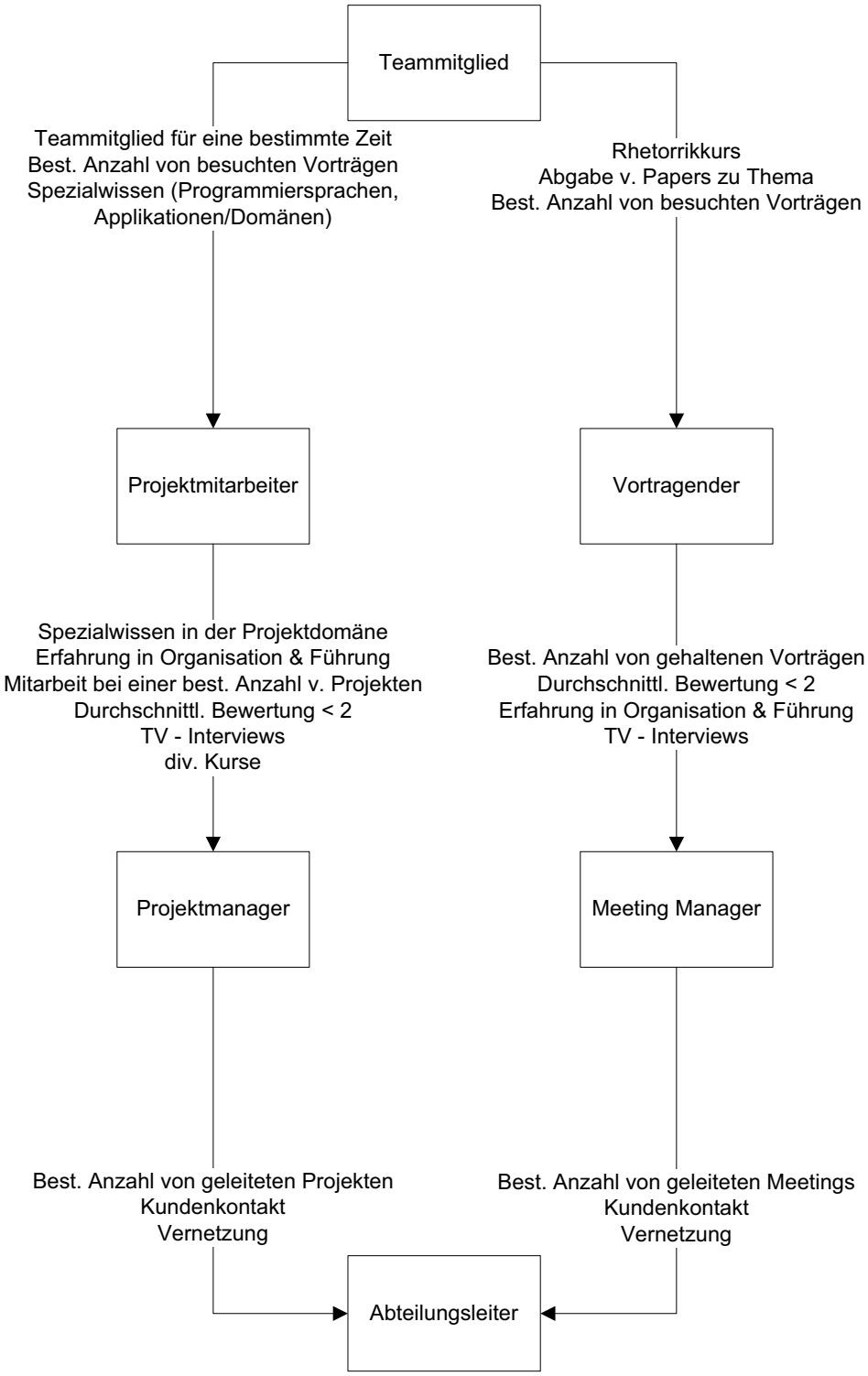

Abbildung 6.4: Partial Order Plan: Karriereleiter

6.2.3 Meeting Support

Hierbei handelt es sich um den wohl kompliziertesten Use Case in diesem hier vorgestellten Meeting unterstützendem System. Dieser Use Case umfaßt folgende Aktionen:

Meeting Support starten: Nachdem das Programm gestartet wurde, meldet sich der Benutzer mit seinem Benutzernamen im System an. Wird der Anmeldevorgang erfolgreich abgeschlossen, so ist der Benutzer im System angemeldet und kann das Programm verwenden. Als erstes wird überprüft, ob eine Einladung für den Benutzer vorliegt. Sollte dies der Fall sein, erscheint ein Bildschirm, über den man auswählen kann, ob man die Einladung annimmt oder nicht. Für die weitere Vorgehensweise bei der Abarbeitung der Einladung vergleiche *Benutzer zu Meeting einladen* weiter unten. Danach wird in regelmäßigen Abständen überprüft, ob gerade ein Meeting anfängt, für das der Benutzer angemeldet ist. Tritt dieser Fall ein, so wird der Use Case *MeetingSupport* gestartet.

Überprüfen ob Restzeit vorhanden: Findet ein Meeting statt, so wird im Hintergrund mittels eines CyclicBehaviours die aktuelle Restzeit berechnet und sowohl graphisch auf einem Zeitbalken, als auch in absoluten Zahlen dargestellt. Der Zeitpunkt, zu dem das Smart Device den Vortragenden notifiziert ist frei wähl- und konfigurierbar. Dies kann entweder nach einer festgesetzten Prozentzahl p nach Ablauf der Zeit, oder ein absoluter Wert (z.B. in Minuten) sein. In der Implementierung wurden die Werte 90%, 95% und 99% gewählt. Die Notifizierung findet mittels Vibrieren des Smart Devices statt. Die Werte wurden so gewählt, da der Vortragende somit immer weiß, dass schon ein bestimmter Prozentsatz *p* der Zeit des Meetings vergangen ist und ihm nur mehr (100-*p*)% Redezeit verbleiben.

Sollte es der Vortragende nicht schaffen, innerhalb der vorgegebenen Zeit das Meeting zu beenden, so wechselt die Hintergrundfarbe des Displays des mobilen Devices auf rot.

Benutzer benachrichtigen: Nach der Beendigung des Meetings durch den Vortragenden werden alle beteiligten Benutzer davon informiert und jeder erhält die Möglichkeit, den Vortragenden nach dem Schulnotensystem zu bewerten. Haben alle Beteiligten ihre Stimme abgegeben, sieht der Vortragende auf seinem Display, welche Note er wie oft bekommen hat, den Durchschnittswert sowie den

Durchschnittswert aller bisher erhaltenen Bewertungen. Dieser Wert hat auch Einfluss auf die Position in der Karriereleiter. Sollte der Vortragende seine Redezeit überzogen haben, wird die Zeit, die überzogen wurde, in der Datenbank gespeichert.

Überprüfen ob letztes Meeting erreicht: Nach Beendigung des Meetings wird zuerst überprüft, ob bereits das letzte Meeting an diesem Tag erreicht wurde. Sollte dies der Fall sein, so muss - auch wenn die Dauer überzogen wurde - nichts umorganisiert werden und der Zeitplan kann wie gewohnt am nächsten Tag fortgesetzt werden. Finden aber an diesem Tag noch weitere Meetings statt, so überprüft das Programm, welche Meetings wie verschoben werden können, um Überschneidungen im Zeitplan zu verhindern.

Zeitplan umorganisieren: Vom Programm wurde festgestellt, dass an diesem Tag noch weitere Meetings stattfinden. Zur genaueren Beschreibung siehe Use Case *Zeitplan umorganisieren*.

In Abbildung 6.5 wird das Zusammenspiel der einzelnen Aktionen im Use Case Meeting Support graphisch dargetellt, der Partial Order Plan in Abbildung 6.6. Den Algorithmus zum Partial Order Plan für den Use Case Meeting Support findet man im Anhang B.

6.2.4 Zeitplan umorganisieren

Ist ein Meeting vom Vortragenden überzogen worden, prüft das Programm automatisch, ob es sich durch das neue Ende, dieses Meeting mit anderen Meetings überschneidet. Sollte dies der Fall sein, so kann entweder die Dauer verkürzt bzw. verlängert oder auch nachfolgende Meetings verschoben werden. Hierbei werden vom Agenten folgende Aktionen durchgeführt:

Feststellen der Zeit, die überzogen wurde: Dauert das Meeting länger als seine ursprünglich festgesetzte Dauer, so wird zunächst überprüft, ob noch weitere Meetings an diesem Tag in diesem Raum stattfinden. Sollte dies der Fall sein, muss der Zeitplan umorganisiert werden.

Notifizierung: Benutzer werden vom Programm benachrichtigt, dass die maximale Dauer überschritten wurde.

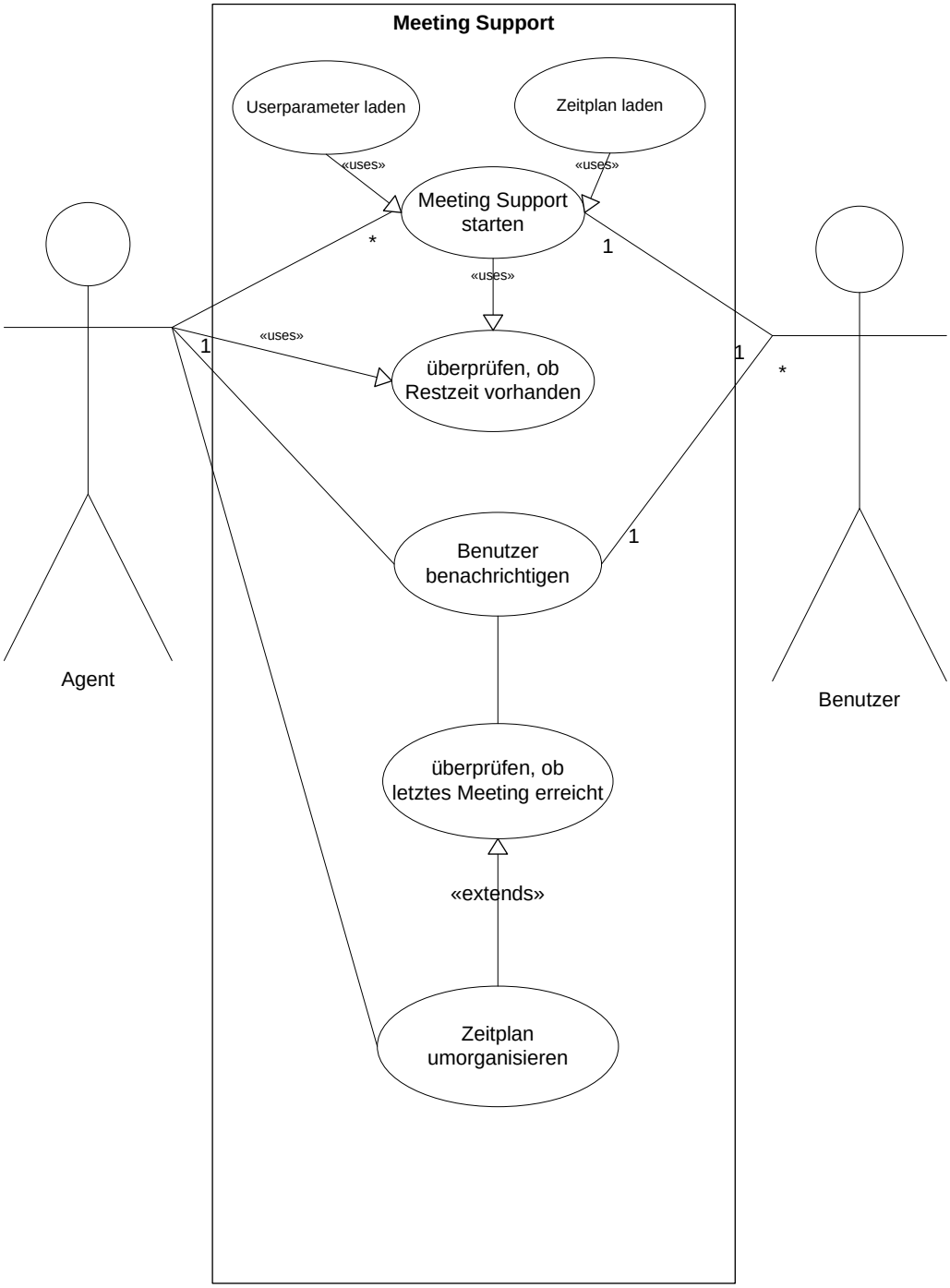

Abbildung 6.5: Use Case: Meeting Support

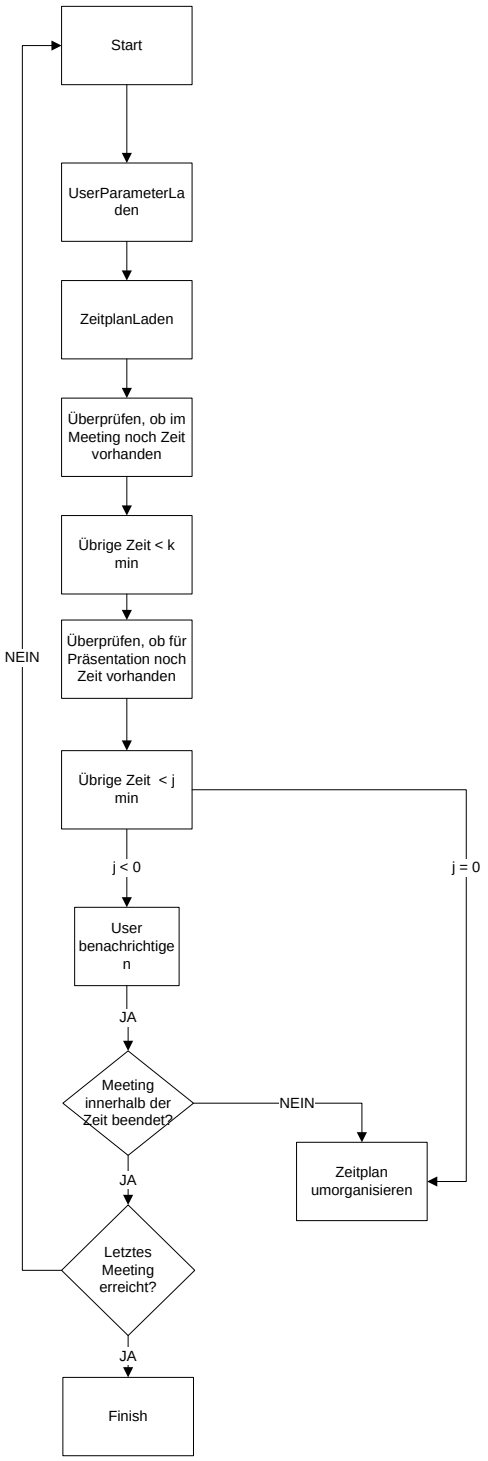

Abbildung 6.6: Partial Order Plan: Meeting Support

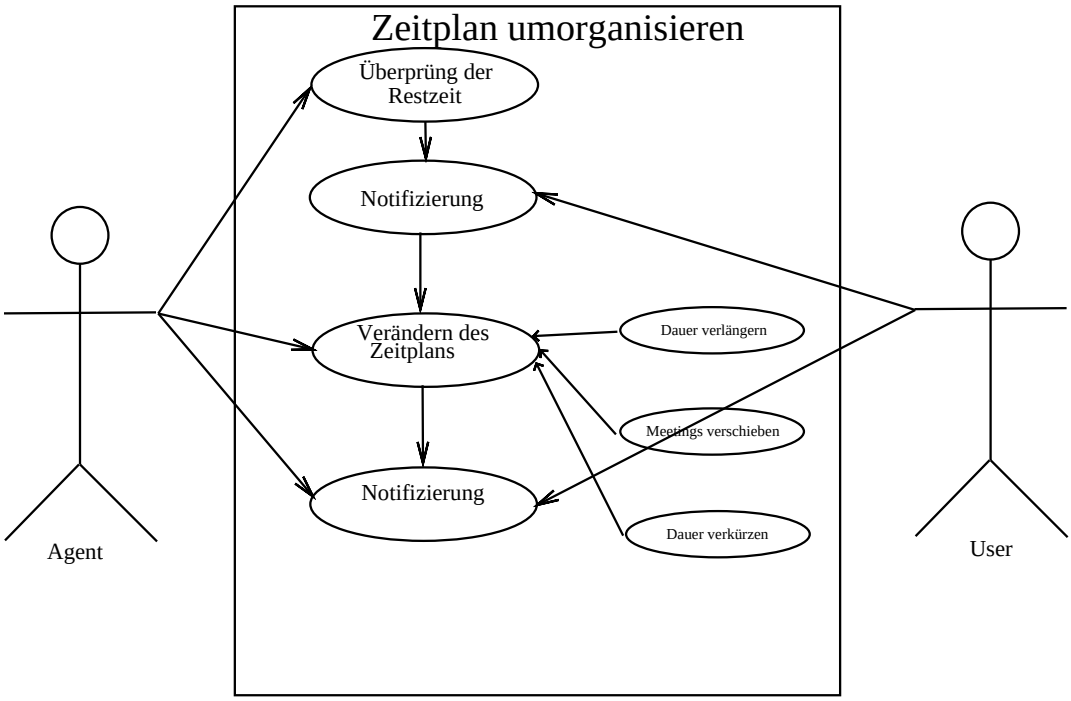

Abbildung 6.7: Use Case: Zeitplan umorganisieren

Verändern des Zeitplans: Wurde das Meeting überzogen, so muss der Zeitplan verändert werden. Dies kann entweder dadurch geschehen, dass nachfolgende Meetings entweder verkürzt oder verschoben werden. Sollte am aktuellen Tag kein Meeting mehr stattfinden, so wird der Meetingplan am folgenden Tag automatisch fortgesetzt. Finden aber noch weitere Meetings an diesem Tag in diesem Raum statt, so muss ein neuer Zeitplan erstellt werden.

Notifizierung: Wurde der Zeitplan verändert, so werden alle beteiligten Benutzer über die neuen Zeiten informiert. Danach kennen alle Teilnehmer den neuen, veränderten Zeitplan.

In Abbildung 6.7 werden die Abläufe graphisch dargestellt, wenn es nötig ist, dass der Zeitplan umorganisiert werden muss, in Abbildung 6.8 findet man den Partial Order Plan. Den Algorithmus zum Partial Order Plan für den Use Case Zeitplan umorganisieren findet man im Anhang **??**.

6.2.5 Benutzer zu Meeting einladen

Dieser Use Case umfasst folgende Aktionen:

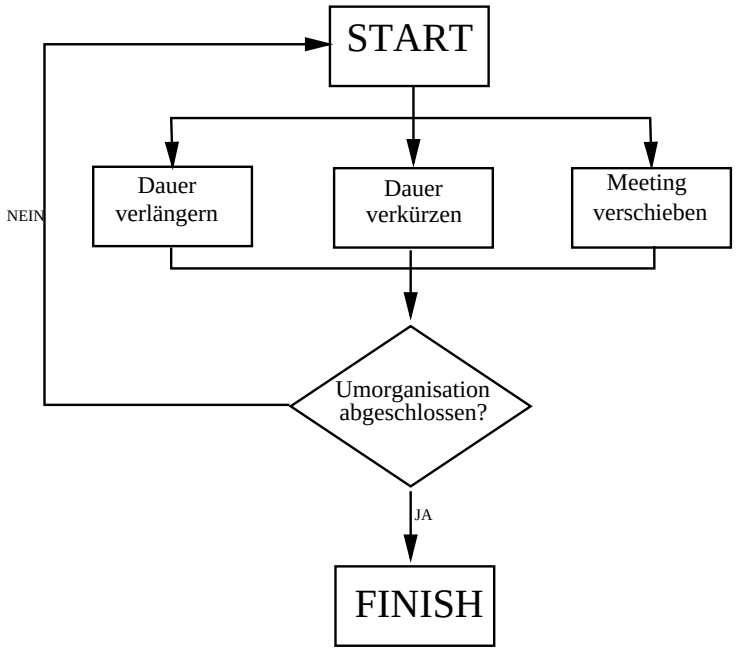

Abbildung 6.8: Partial Order Plan: Zeitplan umorganisieren

Auswählen der Benutzer: Jeder Benutzer hat die Möglichkeit andere Benutzer für ein oder mehrere Meetings einzuladen.

Auswählen des Meetings: Nachdem zumindest ein Benutzer ausgewählt wurde, werden nun die Meetings ausgewählt.

Einladungen verschicken: Nachdem Benutzer und Meetings zu denen eingeladen werden soll ausgewählt wurden, werden nun Einladungen an alle eingeladenen Benutzer verschickt. Jeder Benutzer, der eingeladen wurde, hat eine Einladung erhalten. Diese Einladung wird bis zu einer Reaktion des Benutzers als Eintrag in der Datenbank gespeichert.

Wenn sich ein Benutzer, der eingeladen worden ist, das nächste Mal im System anmeldet, wird er über die Einladung informiert und kann dann entscheiden, ob er die Einladung annehmen will oder nicht. Weist er die Einladung zurück, so wird der Eintrag mit der Einladung aus der Datenbank gelöscht. Entscheidet er sich jedoch die Einladung anzunehmen, so wird er als Teilnehmer des Meetings in der Datenbank gespeichert.

In Abbildung 6.9 werden die einzelnen Aktionen im Use Case Benutzer einladen graphisch dargestellt, der Partial Order Plan in Abbildung 6.10. Den Algorithmus zum

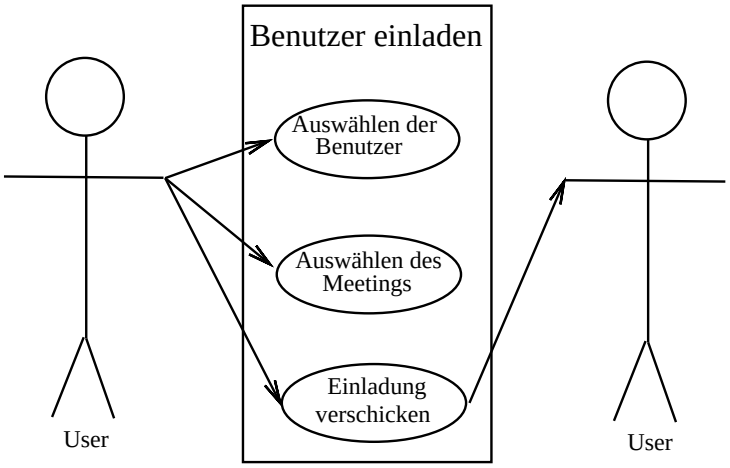

Abbildung 6.9: Use Case: Benutzer einladen

Partial Order Plan für den Use Case Benutzer einladen findet man im Anhang B.

6.3 Systemvoraussetzungen und StartUp

Damit das Programm zum Einsatz kommen kann, müssen folgende technischen Voraussetzungen erfüllt sein:

- Unterstützung der Java Micro Edition durch die mobilen Endgeräte,
- die mobilen Endgeräte müssen WLAN-fähig sein,
- der Server muss mit einem WLAN-Access-Point verbunden sein,
- auf dem Server muss ein Apache Webserver sowie eine MySQL-Datenbank installiert sein,
- auf dem Server muss Java installiert sein und
- als Bindeglied zwischen Java und MySQL muss der JDBC-MySQL-Connector installiert sein.

Bevor ein Meeting stattfinden kann, müssen die Daten der Benutzer sowie der Meetings in der Datenbank gespeichert werden. Dies geschieht in der aktuellen Version des Programms noch direkt über die SQL-GUI.

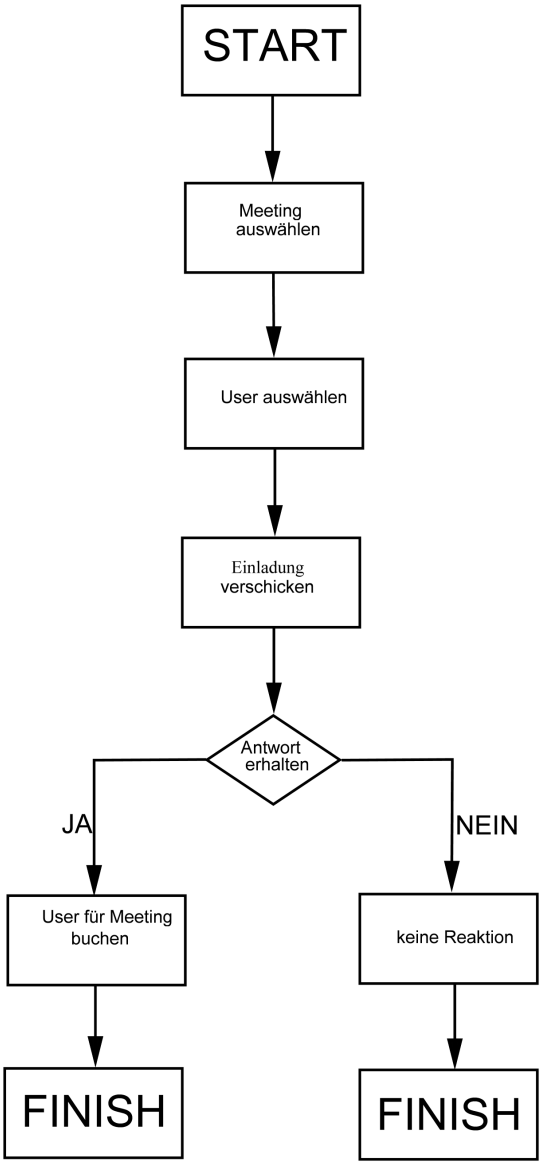

Abbildung 6.10: Partial Order Plan: Benutzer einladen

Zunächst muss ein Server hochgefahren werden, auf dem der Serveragent läuft. Dieser weiß zu jedem Zeitpunkt, welche Benutzer gerade im System angemeldet sind.

Sobald sich ein Benutzer am System anmeldet und in Reichweite des WLANs ist, wird ein Agent auf Clientseite initialisiert. In der *setup()*-Methode des Agenten wird dem Server mittels einer ACL-Message mitgeteilt, dass sich der betreffende Benutzer angemeldet hat. Des Weiteren starten zwei Behaviours, die jeweils in regelmäßigen Abständen überprüfen, ob gerade ein Meeting anfängt, für das der Benutzer registriert ist und ob für den Benutzer eine Einladung für ein Meeting vorliegt. Hat der Agent ein Meeting gefunden, so wird überprüft, ob alle Benutzer, die an diesem Meeting teilnehmen, bereits im System angemeldet sind. Sollte dies nicht der Fall sein, so wird der Anfang des betreffenden Meetings um zehn Minuten nach hinten verschoben und dann wird wieder überprüft, ob schon alle Benutzer angemeldet sind. Diejenigen Benutzer, die bereits angemeldet sind, bekommen vom Server eine Nachricht, dass sich der Beginn des Meetings um zehn Minuten nach hinten verschoben hat.

6.4 Implementierung und Konzept

In diesem Kapitel wird neben den Aufgaben des Agenten auf die Implementation der einzelnen Use Cases eingegangen. Des Weiteren werden zwei unterschiedliche Methoden zur Berechnung der durchschnittlichen Bewertung eines Benutzers vorgestellt.

6.4.1 Aufgaben des Agenten

In unserem Meeting unterstützendem System gibt es zwei Arten von Agenten:

Server-Agent: Der Agent, der auf dem Server implementiert ist, sorgt für die Kommunikation der Clients mit dem Server. Darüber hinaus steuert er den Zugriff auf die Datenbank.

Client-Agent: Sobald die Anwendung auf einem der mobilen Geräte gestartet wurde, wird ein Agent erzeugt, der die meetingunterstützende Funktion wahrnimmt. Des Weiteren steuert er die Kommunikation der einzelnen Clients sowohl untereinander, als auch mit dem Server.

6.4.2 Meetingsupport

Meldet sich der Benutzer im System an, so wird in der *setup()*-Methode des clientseitigen Agenten ein TickerBehaviour gestartet. Dieses Behaviour sendet im Abstand von einer Sekunde regelmäßig Anfragen in Form einer ACL-Message an den serverseitigen Agenten. Dieser überprüft, ob gerade ein Meeting anfängt für das der Benutzer angemeldet und das in der Datenbank gespeichert ist. Hat der Agent in der Datenbank ein Meeting gefunden, so wird zunächst überprüft, ob alle Benutzer, die daran teilnehmen sollen, auch schon eingeloggt sind. Den Algorithmus zum Starten des Meetingsupports findet man in Algorithmus 12, den Algorithmus für den Partial Order Plan in Anhang B.

Ist dies nicht der Fall, so wird vom serverseitigen Agenten eine ACL-Message erstellt. Diese enthält eine Benachrichtigung, dass noch nicht alle vorgesehenen Benutzer eingeloggt sind und dass das Meeting um zehn Minuten nach hinten verschoben wurde. Nachdem in der Datenbank die neue Beginnzeit gespeichert wurde, wird diese Nachricht an alle bereits eingeloggten Benutzer gesendet. Jeder Empfänger dieser Nachricht wird auf seinem Display über diese Änderungen informiert.

Sind alle Benutzer im System angemeldet, so kann der Meetingsupport beginnen. Dazu öffnet sich bei jedem beteiligten Benutzer ein Canvas, in dem Titel, Beginn und Ende des Meetings sowie die Restzeit des Meetings stehen. Unterhalb der textuellen Darstellung dieser Informationen befindet sich ein Balken, der die Restzeit graphisch darstellt. Solange das Meeting noch nicht überzogen wurde, ist der Hintergrund dieses Canvas weíß. Wird die vorhergesehene Zeit überschritten, so färbt er sich rot. Kurz vor Ende des Ablaufs der regulären Zeit bekommt der Vortragende zusätzlich durch Vibration des Smart Devices ein Signal, dass das Ende des Meetings unmittelbar bevorsteht.

Um das Meeting zu beenden, drückt der Vortragende eine Taste auf seinem Smart Device um das Ende dem Programm mitzuteilen. Danach findet die Bewertung der Präsentation statt.

Den Algorithmus zur Ausführung des Meetingsupports findet man in Algorithmus 13, den Algorithmus für den Partial Order Plan in Anhang B.

6.4.3 Bewertung der Präsentation

Die Bewertung des Benutzers, der die Präsentation gehalten hat, findet nach dem Schulnotensystem statt. Bei jedem Beteiligten wird nach Ende eines Meetings ein

Algorithmus 5 starte Meeting Support

```
1: starte Behaviour, das überprüft, ob ∃ Meeting
2: repeat
3:    sende Nachricht an Datenbank
4: until Meeting m für Benutzer b gefunden
5: if Meeting m für Benutzer b gefunden then
6:    überprüfe, ob alle Teilnehmer angemeldet
7:    if alle Teilnehmer angemeldet then
8:       starte Meeting Support
9:    else
10:      verschiebe Meeting um 10 Minuten nach hinten und informiere bis jetzt angemeldete Benutzer
11:      starte Behaviour, das überprüft, ob ∃ Meeting
12:   end if
13: end if
```

Algorithmus 6 Meeting Support

```
1: zeige Screen mit Titel, Beginn, Ende und Restzeit des Meetings
2: repeat
3:    Restzeit = Restzeit - 1
4: until Meeting = beendet
5: if Meeting überzogen then
6:    sende Zeit, die überzogen wurde, an Datenbank
7:    speichere Zeit, die überzogen wurde, in der Datenbank
8:    überprüfe, ob es Überschneidungen mit anderen Meetings gibt
9:    if ∃ Überschneidung then
10:      reorganisiere Meetings
11:      sende Nachricht, dass es neue Beginnzeiten gibt an alle beteiligten Clients
12:   else
13:      sende Nachricht, dass keine Reorganisation nötig an alle beteiligten Clients
14:   end if
15: end if
16: Evaluation
17: if Evaluation fertig then
18:    zeige dem Vortragenden Ergebnis der Bewertung
19:    bei allen beteiligten Clients für Evaluation bedanken
20: end if
```

Bildschirm dargestellt, auf dem er seine Bewertung abgeben kann. Während die Bewertung stattfindet, sieht der Vortragende des Meetings auf seinem Display, dass er auf deren Ende warten soll. Haben alle Benutzer ihre Bewertung abgegeben, werden die gesammelten Ergebnisse an die serverseitige Datenbank übertragen, dort gespeichert und ausgewertet. Der Vortragende sieht auf seinem Display ein Balkendiagramm mit der Häufigkeit der abgegebenen Bewertungen. Des Weiteren werden die Zahl der insgesamt abgegebenen Bewertungen, deren Durchschnitt und der Durchschnitt aller bisher erhaltenen Bewertungen angezeigt.

Der Durchschnitt aller bisher erhaltenen Bewertungen kann auf zwei Arten berechnet werden:

Berechnung des Durchschnitts ohne Alterung von Daten

Diese Methode - bei der das arithmetische Mittel herangezogen wird - ist im Programm implementiert. Zur Berechnung werden alle bisher abgegebenen Bewertungen aufsummiert und durch die Anzahl der Bewertungen dividiert. Ein Beispiel wird in Tabelle 6.1 angeführt.

Note	*Häufigkeit*
1	4
2	5
3	4
4	1
5	2
Summe	40
n	16

Tabelle 6.1: Durchschnitt ohne Alterung von Daten

Formel zur Berechnung des arithmetischen Mittels angewandt auf das Beispiel: $\bar{x} = \frac{\sum_{i=0}^{n} x_i}{n} = \frac{40}{16} = 2,5$

Berechnung des Durchschnitts mit Alterung der Daten

Bei dieser Art der Berechnung des Durchschnitts werden die arithemitischen Mittel der Bewertungen zum Zeitpunkt T_i mit einem Gewicht, abhängig vom Alter der Bewertungen, versehen. Das Alter der Bewertungen wird mit $T_i > 0$ angenommen, wobei T ein logischer Zeitpunkt nach dem Zeitpunkt Z_0 ist. Das Gewicht der einzelnen Bewertungen wird nach der Formel $t_i = \frac{1}{T_i}$ berechnet. Die gewichtete Bewertung

ist $t_i \times \bar{b_i}$. $\bar{b_i}$ ist das arithmetische Mittel aller Bewertungen zum Zeitpunkt T_i. Dadurch wirken sich kürzlich abgegebene Bewertungen stärker auf den Durchschnitt aus als Bewertungen, die schon länger zurückliegen.

Um nun eine durchschnittliche Bewertung in Abhängigkeit vom Alter der Daten zu berechnen wird wie folgt vorgegangen:

1. Berechnung des arithmetischen Mittels aller Bewertungen zum Zeitpunkt i. siehe *Berechnung des Durchschnitts ohne Alterung von Daten*
2. Berechnung der Gewichte der einzelnen arithmetischen Mittel nach der Formel $t_i = \frac{1}{T_i}$.
3. Multiplikation der arithmetischen Mittel zum Zeitpunkt i mit ihrem Gewicht $t_i \times \bar{b_i}$
4. Aufsummierung der zuvor berechneten Produkte
5. Aufsummierung der einzelnen Gewichte
6. Division von der Summe der Produkte mit der Summe der Gewichte

Ein Beispiel mit Berechnung der einzelnen Zwischenergebnisse wird in Tabelle 6.2 angeführt. Zur Berechnung des gewichteten arithmetischen Mittels wird folgende Formel herangezogen:

$\bar{x} = \frac{\sum_{i=0}^{n} t_i \times \bar{b_i}}{\sum_{i=0}^{n} t_i} = \frac{8{,}136}{2{,}929} = 2,777$

6.4.4 Reorganisation der Meetings

Bei dem Prototyp unseres Programms ist eine Möglichkeit der Reorganisation implementiert.

Sollte ein Vortragender sein Meeting um k Minuten überziehen, so wird zunächst überprüft, ob es zu Überschneidungen mit nachfolgenden Meetings kommt. Ist dies nicht der Fall, so kann mit der Agenda ohne Änderung fortgefahren werden. Kommt es aber zu Überschneidungen, so werden alle nachfolgenden Meetings an diesem Tag in diesem Raum um die überzogene Zeit von k Minuten nach hinten verschoben. Sind alle Änderungen erfolgreich in der Datenbank gespeichert, so bekommen alle beteiligten Benutzer eine Nachricht, dass sich die Zeiten der Meetings verschoben haben und sie sich über die neuen Beginnzeiten über einen Menüpunkt auf ihrem Display informieren sollen. Das Schema des zugehörigen Algorithmus ist in Algorithmus 7 dargestellt. Den Algorithmus für den Partial Order Plan findet man in Anhang B.

Zeitpunkt T(i)	*Gewichtung* $t_i = \frac{1}{T_i}$	$\bar{b_i}$	$t_i * \bar{b_i}$
1	1,000	2,5	2,500
2	0,500	4,2	2,100
3	0,333	1,9	0,633
4	0,250	2,0	0,500
5	0,200	1,7	0,340
6	0,167	2,6	0,433
7	0,143	3,7	0,528
8	0,125	4,1	0,512
9	0,112	3,4	0,378
10	0,100	2,1	0,210
Summe	2,929		8,136

Tabelle 6.2: Durchschnitt mit Alterung von Daten

6.4.5 Karriereleiter

Die Berechnung der Karriereleiter findet aufgrund der gespeicherten Einträge in den Tabellen *User* und *evaluation* statt. In der Tabelle *User* sind für jeden Benutzer in der der Spalte *meetingsVisited* die Anzahl der Meetings, die bereits vom Benutzer besucht wurden, und in der Spalte *participationsHeld* die Anzahl der Präsentationen, die bereits vom Benutzer gehalten wurden, gespeichert. In der Tabelle *evaluation* werden für jeden Benutzer die erhaltenen Bewertungen gespeichert.

Im Prototyp dieses Programms ist der Weg zum *Abteilungsleiter* mit den Zwischenstufen *Vortragender* und *Meeting Manager* implementiert (siehe Algorithmus 8). Die andere Möglichkeit den Status eines *Abteilungsleiter* zu erreichen, nämlich über die Zwischenstufen *Projektmitarbeiter* und *Projektmanager*, ist im Moment nicht implementiert.

Die Karriereleiter wurde mit folgenden Werten implementiert:

- Anzahl der besuchten Meetings als Meetingteilnehmer m = 10,
- Anzahl der besuchten Meetings als Vortragender v = 25,
- Anzahl der gehaltenen Präsentationen p = 25,
- durchschnittliche Bewertung e = 2.

Algorithmus 7 Reorganisation von Meetings

```
1: if überzogene Zeit erfolgreich in Datenbank gespeichert then
2:    prüfe, ob letztes Meeting an diesem Tag in diesem Raum erreicht
3:    if letztes Meeting nicht erreicht then
4:       verschiebe jedes nachfolgende Meeting an diesem Tag in diesem Raum um
         die Zeit, die überzogen wurde, nach hinten
5:       if neue Zeiten erfolgreich in Datenbank gespeichert then
6:          sende Nachricht, dass Zeiten von Meetings verschoben wurden an alle
            beteiligten Benutzer
7:       else
8:          sende Fehlernachricht
9:       end if
10:   else
11:      sende Nachricht, dass keine Reorganisation nötig
12:   end if
13: else
14:    sende Fehlernachricht
15: end if
```

Algorithmus 8 Karrierleiter

```
1: rolle = “meeting participant”
2: if besuchte Meetings ≥ m then
3:    rolle = “presentator”
4:    if besuchte Meetings ≥ v ∧ gehaltene Präsentationen ≥ p ∧ durchschnittliche
      Bewertung ≥ e then
5:       rolle = “meeting manager”
6:    end if
7: end if
```

6.4.6 Benutzer zu Meeting einladen

Jeder Benutzer hat die Möglichkeit andere Benutzer zu einem Meeting einzuladen.

Dazu wählt er das Meeting aus, zu dem eingeladen werden soll. Danach kann der Benutzer die potentiellen Teilnehmer auswählen und mittels Knopfdruck die Einladung in der Datenbank speichern (vgl. Algorithmus 9).

Loggt sich ein Benutzer im System ein, wird in regelmäßigen Abständen überprüft, ob eine Einladung für ihn vorliegt. Den Algorithmus zur Überprüfung von Einladungen findet man in Algorithmus 10, den Algorithmus für den Partial Order Plan in Anhang B.

Algorithmus 9 Einladung aussprechen

```
1: wähle Meeting zu dem eingeladen werden soll, aus
2: if Meeting ausgewählt then
3:     wähle einen oder mehrere Benutzer aus, die eingeladen werden sollen
4:     if Benutzer ausgewählt then
5:         schicke Einladungen an Server
6:     end if
7: end if
```

Algorithmus 10 auf Einladung überprüfen

```
1: repeat
2:     überprüfe, ob Einladung(en) in der Datenbank gespeichert ist
3: until Einladung(en) gefunden
4: schicke Einladung(en) an Client
5: stelle Einladung(en) am Display des Clients dar
6: if Einladung akzeptiert then
7:     sende Antwort an Server
8:     speichere Benutzer als Teilnehmer in der Datenbank
9:     lösche Einladung aus Datenbank
10: end if
11: if Einladung negiert then
12:     sende Antwort an Server
13:     lösche Einladung aus Datenbank
14: end if
```

7 Programmablauf und GUI

Alle hier beschriebenen Funktionen sind mit dem Sun Wireless Toolkit Emulator für das Nokia N91 getestet worden [1, 59]. Die Benutzung des Prototyps ist auf dem Nokia N91 vorgesehen. Die meisten Funktionen wurden auch bereits auf dem Nokia N91 getestet. Offen sind noch die Vibrationsfunktion und das Öffnen einer URL mit dem Browser des Smart Devices. Darüberhinaus traten bei der Entwicklung Probleme bei der Zeitsynchronisierung zwischen Client und Server auf.

Beim Design der Applikation wurde darauf geachtet, dass für den Benutzer alle Menüs und Menüpunkte selbsterklärend sind. Dadurch sollte er keine Schwierigkeiten haben das Programm zu bedienen. Sendet das Programm eine Nachricht an den Datenbankserver oder einen anderen Agenten ist im Normalfall mit einer Antwort im Bereich von einigen Zehntelsekunden zu rechnen. Sollte bei der Interaktion mit anderen teilnehmenden Entitäten ein Fehler auftreten, so erhält der Benutzer eine Nachricht mit der Fehlermeldung.

Wird die Applikation am mobilen Endgerät gestartet, so kommt man zu einem Bildschirm, wo man seinen Benutzernamen eingeben muss. Nach erfolgreichem Login kommt man in das Hauptmenü des Programms, wo alle Funktionalitäten, die ein direktes Handeln des Benutzers erfordern, aufgelistet sind (siehe Abbildung 7.1). Die wichtigsten Funktionen sind:

- Anzeigen aller Meetings, für die man angemeldet ist
- Anzeigen aller Meetings, bei denen man selbst Vortragender ist
- Anmeldung und Abmeldung von Meetings
- das eigene Benutzerprofil
- Menüpunkt, über den Personen zu Meetings eingeladen werden können sowie
- das Anzeigen externer Links, zu Meetings

Abbildung 7.1: Screenshot: Hauptmenü

7.1 Benutzergesteuerte Funktionen

Dazu werden Funktionen gezählt, die kein autonomes Handeln eines Agenten benötigen. Sie werden nur auf expliziten Befehl des Benutzers ausgeführt. Diese Funktionen ermöglichen es dem Benutzer meeting- und benutzerrelevante Informationen von der Datenbank abzufragen und sich diese darstellen zu lassen. Jede dieser Funktionen wurde so implementiert, dass die abgefragten Informationen in maximal fünf Bedienungsschritten angezeigt werden. Dies soll dem Benutzer eine einfache und unkomplizierte Bedienung des Programmes erlauben.

7.1.1 Eigene Meetings und eigene Präsentationen abrufen

Wählt man einen dieser beiden Menüpunkte, so werden alle Meetings, für die man selbst registriert bzw. bei denen man Vortragender ist, aufgelistet.

Hat man diesen Menüpunkt ausgewählt, so sieht man zunächst nur die Titel der einzelnen Meetings. Wurde eines ausgewählt, so werden am Display alle relevanten Informationen zu ebendiesem aufgelistet (siehe Abbildung 7.2, 7.3). Dazu gehören:

- die ID des Meetings
- der Titel des Meetings
- den Vortragenden des Meetings
- die Beginnzeit

- das Ende
- den Raum, in dem das Meeting stattfindet sowie
- die Anzahl der Teilnehmer.

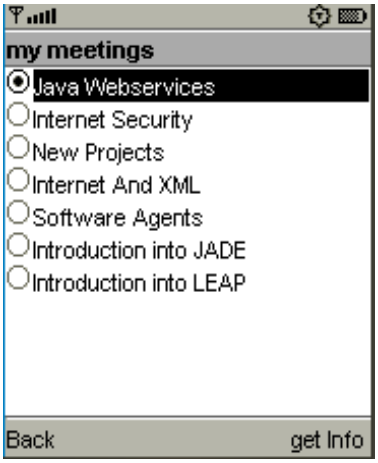

Abbildung 7.2: Screenshot: eigene Meetings bzw. Präsentationen

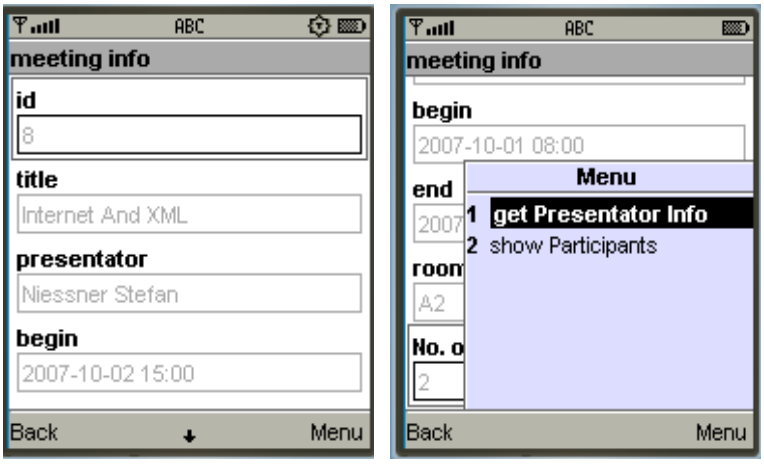

Abbildung 7.3: Screenshot: Meeting Info

Werden diese relevanten Informationen aufgelistet, so findet der Benutzer auf seinem Screen einen Menüpunkt über den man sich die Teilnehmer des Meetings, sowie persönliche Informationen über diese und den Vortragenden selbst, anzeigen lassen kann. Mit den Cursortasten kann die entsprechende Funktion ausgewählt werden. Wurde eine Funktion ausgewählt, so wird eine Nachricht mit der Anfrage an den Datenbankserver gesendet. Nachdem dort die Anfrage angekommen ist, wird sie vom

Serveragenten abgearbeitet. War dies erfolgreich, so wird das gewünschte Ergebnis an den Client zurückgesendet und am Display des Benutzers dargestellt.

7.1.2 An- und Abmeldung von Meetings

Hier sieht man alle Meetings, die im Moment in der Datenbank gespeichert sind, sowie den Status, ob man angemeldet ist oder nicht. Mittels der Navigatortaste kann man bei jedem Meeting seinen Status ändern. Führt man die Funktion “update” aus, so werden die aktuellen Stati der einzelnen Meetings an die Datenbank geschickt und dort gespeichert. War das Speichern erfolgreich, so erhält der Benutzer eine entsprechende Nachricht auf dem Display des SmartPhones (siehe Abbildung 7.4).

Abbildung 7.4: Screenshot: Registration

7.1.3 Mein Profil

Nach Auswahl dieses Menüpunktes sieht man seine persönlichen Daten, die in der Datenbank gespeichert sind. Daten - wie Adresse, Telefonnummer oder E-Mail-Adresse - kann man hier eigenständig ändern. Nach Ausführung der Funktion “update” werden die geänderten Daten an die Datenbank geschickt und dort gespeichert. War dieser Vorgang erfolgreich, so kommt vom Server eine Nachricht mit einer positiven Rückmeldung zurück, die dann auf dem Display des Benutzers erscheint.

Neben den persönlichen Daten sieht man hier noch die aktuelle Rolle, die man im Karriereprozess inne hat; des Weiteren sind noch die Zeit, die man als Vortragender bereits überzogen hat, die Anzahl der besuchten Meetings, die Anzahl der gehaltenen Präsentationen sowie die durchschnittliche Bewertung als Vortragender aufgelistet.

Aufgrund all dieser Eigenschaften kann sich der Benutzer zu jedem Zeitpunkt einen Überblick verschaffen, wie weit er auf der Karriereleiter bereits nach oben geklettert ist und in welchen Bereichen eine Verbesserung noch verbessern wäre. Diese Daten kann man nicht eigenständig ändern, da sie sich aus dem Verhalten des Benutzers zusammensetzen (siehe Abbildung 7.5).

Abbildung 7.5: Screenshot: Benutzerprofil

7.1.4 Benutzer einladen

Wie der Name der Funktion schon sagt, hat hier jeder Benutzer nach Auswahl des “invite Users”-Punktes die Möglichkeit andere Benutzer, die bereits im System registriert sind, zu Meetings einzuladen.

Nach Auswahl dieses Menüpunktes wird eine Nachricht mit der Anfrage an den Server gesendet. Sobald diese Nachricht am Server eingetroffen ist, beginnt der Da-

tenbankagent nach allen gespeicherten Meetings zu suchen. Wurde die Suche abgeschlossen, so werden sie zurück an den Client gesendet und deren Titel am Display dargestellt. Der Benutzer hat nun die Möglichkeit ein Meeting auszuwählen, zu dem eingeladen werden soll (vgl. Abbildung 7.6. Hat man ein Meeting ausgewählt, so kommt man durch Auswahl der “Select”-Option zum nächsten Screen. Der Auswählen dieser Option wird eine Nachricht an den Server gesendet und nachdem der Datenbankagent seine Arbeit getan hat, werden alle Benutzer, die zwar im System registriert sind aber noch nicht als Teilnehmer für das Meeting vorhergesehen sind, in einer Nachricht gespeichert. Wurde die Suche abgeschlossen, wird die Antwortnachricht zurück an den Client gesendet. Dieser erhält nun die Möglichkeit zumindest einen Benutzer auf seinem Display mittels der Cursortasten auszuwählen. Hat der Benutzer seine Auswahl beendet, so werden die gesammelten Einladungen an den Server verschickt und in der Datenbank gespeichert.

Jeder der eingeladenen Benutzer erhält bei der nächsten Aktion, die er ausführt, eine Mitteilung, dass eine oder mehrere Einladungen für ihn vorliegen. Hier hat der Benutzer die Möglichkeit die Einladung entweder anzunehmen oder abzulehnen. Akzeptiert er sie, so wird er zu den Teilnehmern des Meetings hinzugefügt (also in einer Tabelle in der Datenbank gespeichert) und die Einladung wird aus der entsprechenden Tabelle in der Datenbank gelöscht. Nimmt der Benutzer die Einladung nicht an, so erlischt sie und wird ebenfalls aus der entsprechenden Tabelle in der Datenbank gelöscht (vgl. Abbildung 7.7.

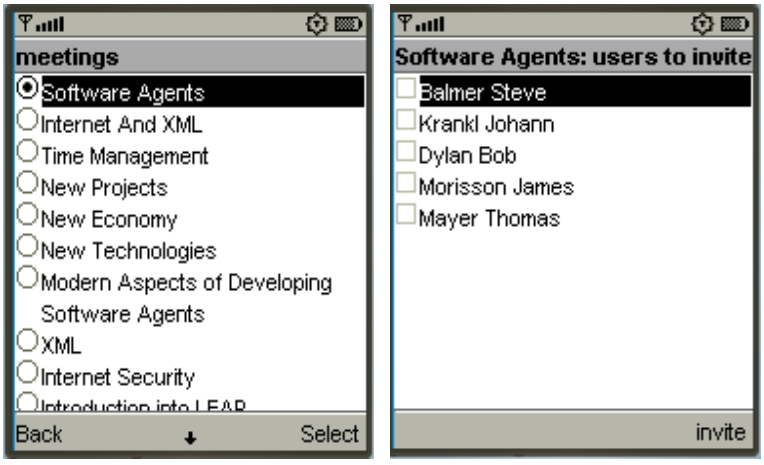

Abbildung 7.6: Screenshot: Benutzer einladen

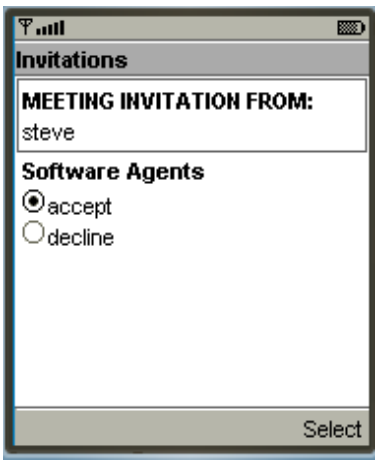

Abbildung 7.7: Screenshot: Einladung

7.1.5 Links zu Meetings abrufen

Zu jedem Meeting kann eine beliebige Anzahl von externen Links in der Datenbank gespeichert werden. Um diese abzurufen, wählt der Benutzer den Menüpunkt “external Links” aus. Zunächst bekommt der Benutzer alle in der Datenbank gespeicherten Meetings aufgelistet. Wählt er ein Meeting aus, so sieht er alle Links, die zu diesem Meeting gespeichert sind. Hier kann der Benutzer nun einen Link auswählen und sich im Browser des mobilen Endgerätes anzeigen lassen. Bei meinen Tests hat dies leider nur am Emulator funktioniert, obwohl strikt nach den Vorgaben und Dokumentationen der Java API vorgegangen wurde. In der Endversion dieser Applikation sollte dies ohne Probleme funktionieren. Ein Beispiel für das Abrufen von Links zu einem Meeting findet man in Abbildung 7.8.

7.2 Agentengesteuerte Funktionen

In diesem Kapitel werden Funktionen des Programms besprochen, die großteils durch autonomes Handeln der Agenten ausgeführt werden. Ein aktives Eingreifen des Benutzers wird auf ein Minimum reduziert. Die Kommunikation zwischen den Agenten der einzelnen Clients sowie mit dem Serveragenten funktioniert eigenständig.

7.2.1 Meetingsupport

Loggt sich der Benutzer im Programm ein, so wird im Abstand von einer Sekunde überprüft, ob ein Meeting für den Benutzer vorliegt. Durch die hohe Frequenz, in der

Abbildung 7.8: Screenshot: externe Links

auf ein Meeting überprüft wird, ist es im Test zu keinerlei Verzögerungen gekommen. Daher ist davon auszugehen, dass dies auch in der Praxis ohne Probleme funktionieren wird.

Hat der Client vom Server ein Signal bekommen, dass ein Meeting anfängt, so wird ein Canvas erzeugt, der relevante Informationen zum Meeting enthält. Dazu zählen Titel, Beginn, Ende sowie Restzeit. Des Weiteren wird der Fortschritt der vergangenen Zeit im Meeting grafisch mit einem Balken, der sich mit Fortdauer des Meetings prozentuell entsprechend der abgelaufenen Zeit füllt, dargestellt. Wurde das Meeting noch nicht überzogen, so sind die graphischen Elemente auf weißem Hintergrund dargestellt (vgl. Abbildung 7.9. Ist die Dauer jedoch überschritten worden, so färbt sich der Hintergrund rot. Der Vortragende des Meetings kann das Meeting jederzeit über einen entsprechenden Knopf beenden (vgl. Abbildung 7.10.

7.2.2 Meeting evaluieren

Ist das Meeting beendet, so erhält jeder Benutzer die Möglichkeit den Vortragenden zu bewerten. Zur Vereinfachung wird in der Beschreibung hier ein Vortrag mit einem Meeting gleichgesetzt.

Hat der Vortragende das Meeting beendet, so sieht jeder der Teilnehmer ein Formular mit nach dem Schulnotensystem auf seinem Display. Dort kann er dann auswählen, mit welcher Note er den Vortragenden bewerten will (vgl. Abbildung 7.12). Hat er seine Entscheidung getroffen, wird dieses Ergebnis an den Agenten, der auf dem mobilen Endgerät des Vortragenden läuft, geschickt. Die Bewertungen fließen direkt

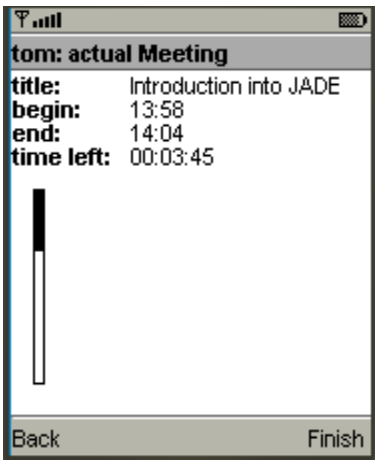

Abbildung 7.9: innerhalb der Zeit

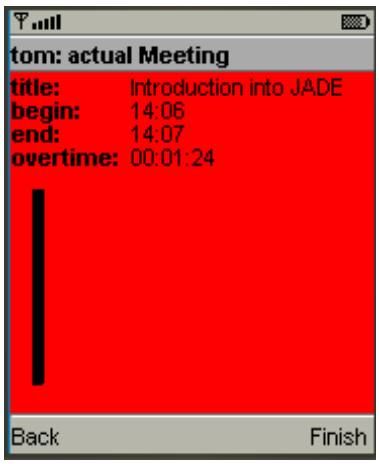

Abbildung 7.10: Dauer überzogen

Abbildung 7.11: Screenshot: Meetingsupport

in die Position auf der Karriereleiter ein. Haben alle Beteiligten ihre Bewertung abgegeben, so sieht der Vortragende das Ergebnis der Bewertung auf seinem Display. Außer dem Ergebnis wird noch die aktuelle Position auf der Karriereleiter dargestellt (vgl. Abbildung 7.13).

Abbildung 7.12: Teilnehmer

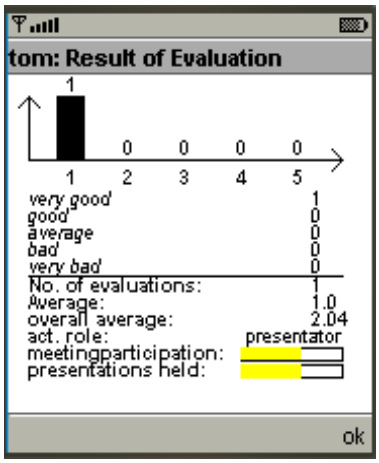

Abbildung 7.13: Vortragender

Abbildung 7.14: Screenshot: Evaluation

8 Zusammenfassung

Ziel dieser Arbeit war es den Einsatz und Machbarkeit von selbstadaptierenden Agenten im Einsatzgebiet der Meeting Assistant Umgebungen zu demonstrieren.

Dies sollte mit Hilfe drei verschiedener Use Cases vorgeführt werden. Dazu gehören eine gekapselte Funktion (Erinnerung eines Benutzers), eine kooperative Funktion (Änderung der Meeting Agenda), sowie ein Fall, der langfristige Ziele betrifft (Karriereleiter).

Das Programm ist mit Hilfe der Programmiersprache Java und dessen Agentenframework JADE entwickelt worden. Um autonome Funktionen wie logisches Schließen zu implementieren, wurde ein goal-orientierter Ansatz gewählt, der auf Prädikatenlogik aufbaut. In der jetzigen Version ist das Programm für den Einsatz auf dem Java Wireless Toolkit Emulator programmiert worden. In einer Endversion soll es auf jedem beliebigen smart Device, das Java unterstützt und WLAN-fähig ist, lauffähig sein.

Zu Beginn der Arbeit ist auf Software Agenten als solche eingegangen worden. Danach ist eine Einführung in Prädikatenlogik, Planen, goal-orientiertes Design sowie implizite Interaktion und Personalisierung gegeben worden. Im nächsten Kapitel sind einige verwandte Arbeiten vorgestellt worden, die in Bezug zu dem hier vorgestellten Projekt stehen.

Nach einem Überblick über die verwendeteten Technologien, so wie Java, JADE, Leap, MySql und Apache sowie dem Nokia N91, für das das Programm entwickelt wurde, ist das Programm vorgestellt worden, das im Zuge dieser Arbeit entwickelt wurde.

Zu guter Letzt sind noch der Programmablauf und die GUI im Einsatz vorgestelltworden. Darüberhinaus ist kurz auf die Usability eingegangen worden.

Mit dieser Arbeit ist die Machbarkeit automatisierter computergesteuerter Unterstützung, die gerade dabei ist an immer größerer Bedeutung zu gewinnen, von Menschen in Ubiquitous Business Meeting Umgebungen gezeigt worden. Diese Arbeit kann in Zukunft dazu verwendet werden, ein noch autonomeres Handeln von smart Devices zu ermöglichen und den Entwicklungsprozess zu unterstützen. Damit soll

der Benutzer noch mehr entlastet und von einem Programm ohne eigenes Zutun unterstützt werden. Dem Benutzer sind einige Aufgaben, so wie die Organisation und Reorganisation von Meetings, abgenommen worden und er kann sich somit auf andere Aufgaben konzentrieren.

Je nach Rolle des Benutzers könnten in Zukunft auch weitere Mechanismen, die über mobile Business Meeting Unterstützung hinausgehen, integriert werden. Da mit dem Use Case “Karriereleiter” ein Leitfaden für die Position eines Benutzers vorgegeben worden ist, könnte man diese Applikation in ein Förderprogramm für Firmen einbinden. V.a. für junge karrierehungrige Mitarbeiter, die in einem Unternehmen erst seit Kurzem arbeiten, könnte dieses Programm eine Hilfe bei der Orientierung auf dem Weg in Richtung Spitze der Karriereleiter sein.

Des Weiteren könnte man die Automatisierung von Abläufen noch stärker in den Vordergrund stellen. Mittels Activity Recognition kann die Applikation automatisch das Verhalten des Benutzers beobachten und aufzeichnen. Die dadurch gewonnenen Daten kann das Programm wieder in den Support des Benutzers einfließen lassen und ihm z.B. Meetings und Vorträge vorschlagen, die für ihn interessant sein könnten. Mit Hilfe von Sensoren und Sensing der Umgebung könnte man den Loginvorgang in der Applikation automatisieren indem das Programm in regelmäßigen Zeitabständen überprüft, ob sich der Benutzer in der Nähe eines Sensors befindet, der an den Server-Agenten angebunden ist. Kommt der Benutzer mit seinem mobilen Device in die Nähe eines solchen Sensors, könnte das Programm den Meeting Support ohne Zutun des Benutzers starten. Eine bestimmten Zeitpunkt vor Beginn des Meetings würde der Benutzer dann über den baldigen Beginn eines Meetings informiert werden und das Programm teilt ihm dann mit, wohin er sich zu begeben hat.

Anhang A: Installationsanleitung

Hier findet man eine Anleitung zum Installieren des Programmes. Im ersten Teil werden die notwendigen Programme und Schritte zur Installation des Servers beschrieben. Im zweiten Teil finden wird auf die Installation des Clients eingegangen.

Eine aktuelle Version der benötigten Programme (Stand November 2008) findet man auf der beigelegten CD im Ordner *programme*.

Installation des Servers

Auf dem Server, der über eine WLAN-Verbindung verfügen muss, muss folgende Software installiert sein:

- AppServ (aktuelle Version 2.6.10) ⇒ `http://www.appserv.com`
- Java (aktuelle Version Java SE 6 Update 10 RC) ⇒ `http://java.sun.com`
- JADE (aktuelle Version 3.6) ⇒ `http://jade.tilab.com/`

Sind aus dem Appserv-Package Server und Datenbank installiert, so muss man die Datenbank samt ihren Tabellen erzeugen. Die passenden SQL-Skripte finden sich in den Dateien *personal_assistance_db_struktur.sql* und Beispieldaten befinden sich im Skript *personal_assistance_daten.sql* im Ordner *SQL*. Im Serverpackage muss die Datei *\src\manager\DBInterface.java* mit einem geeigneten Texteditor (zum Beispiel VIM ⇒ Download geöffnet werden um die Zugangsdaten zur Datenbank anzupassen.

Um die JADE-Umgebung zu installieren, muss man diese zuerst downloaden oder die Datei *JADE-all-3.6.zip* aus dem Ordner Programme entpacken. Danach müssen die .lib-Dateien aus dem Ordner in den die zip-Datei entpackt wurde in den Ordner *j2se\lib* kopiert werden. Sollte dieser Ordner nicht vorhanden sein, so muss er erstellt werden.

Kompiliert wird das Server-Programm mittels eines ANT-Skripts. Downloadmöglichkeit und Installationsanleitung findet man unter `http://ant.apache.org/`. Die ANT-

Anweisung mit ihren Optionen findet man in den Dateien *client.bat* und *server.bat* im Ordner *ant-skripts*.

Zum Starten des Server-Programms muss das Skript *startServer.bat* im Ordner *src* ausgeführt werden.

Installation des Clients

Bevor das Programm kompiliert wird, muss die Datei *meeting_support.manifest* im Ordner src mit einem Texteditor geöffnet werden und bei LEAP-Host die IP-Adresse des Servers auf dem der Serveragent installiert und gestartet wurde eingetragen werden.

Danach kann das Client-Programm mittels beiliegendem ANT-Skript kompiliert werden.

Über die Nokia PC-Suite kann das Programm nun am Smart Device installiert werden. Dazu bitte einfach den Anweisungen am Bildschirm folgen.

Anhang B: Partial-Order-Plans

In diesem Kapitel findet man die Algorithmen zu den Partial Order Plans der in dieser Arbeit beschriebenen Use Cases.

Algorithmus 11 Partial Order Plan: Karriereleiter

Init(Karrierestart(U, C)).
Goal(Abteilungsleiter(U)).
Action(TeammitgliedWerden(U),
PRECOND: Karrierestart(U, C),
EFFECT: IstTeammitglied(U)).
Action(VortragenderWerden(U, M),
PRECOND: IstTeammitglied(U) $\wedge$ RhetorikKursBesuchen(U) $\wedge$ AbgabeVonPapers(U, M) $\wedge$ bestAnzahlAnBesuchtenVorträgen(U),
EFFECT: IstVortragender(U)).
Action(MeetingManagerWerden(U),
PRECOND: IstVortragender(U) $\wedge$ bestAnzahlAnGehaltenenVorträgen(U) $\wedge$ DurchschnittlBewertung < 2 $\wedge$ AusbildungInOrg&Führung $\wedge$ Pressearbeit,
EFFECT: IstMeetingManager(U)).
Action(ProjektAssistantWerden(U),
PRECOND: IstTeammitglied(U) $\wedge$ Spezialwissen $\wedge$ AbgabeVonPapers(U, M) $\wedge$ bestAnzahlAnBesuchtenVorträgen(U),
EFFECT: IstProjektAssistant(U)).
Action(ProjektManagerWerden(U, M),
PRECOND: IstProjektAssistant(U) $\wedge$ gewisseZeitMitglied(U) $\wedge$ Spezialwissen $\wedge$ Org&Führung $\wedge$ Pressearbeit,
EFFECT: IstProjektManager(U, M)).
Action(AbteilungsleiterWerden(U),
PRECOND: IstMeetingManager(U) $\vee$ IstProjektManager(U) $\wedge$ (bestAnzahlAnGeleitetenMeetings $\vee$ bestAnzahlAnGeleitetenProjekten)
EFFECT: IstAbteilungsleiter(U)).

Algorithmus 12 Partial Order Plan: Start Meeting Support

Init(Login(U)).
Goal(MeetingSupportGestartet(U, M)).
Action(Login(U),
PRECOND:,
EFFECT: Eingeloggt(U)).
Action(ÜberprüfenObMeetingVorhanden(U, M),
PRECOND: Eingeloggt(U) $\vee$ KeinMeetingVorhanden(M) $\vee$ IstNichtMeetingteilnehmer(U, M),
EFFECT: MeetingVorhanden(U, M) $\vee$ KeinMeetingVorhanden(U, M)).
Action(ÜberprüfenObMeetingTeilnehmer(U, M),
PRECOND: MeetingVorhanden(M) $\vee$ IstMeetingteilnehmer(M),
EFFECT: IstMeetingteilnehmer(U, M) $\vee$ IstNichtMeetingteilnehmer(U, M)).
Action(Vorbereitungen(M),
PRECOND: IstMeetingteilnehmer(U, M),
EFFECT: MeetingSupportGestartet(M)).

Algorithmus 13 Partial Order Plan: Meeting Support

Init(MeetingSupportStarten(U, M)).
Goal(MeetingSupport(U, M)).
Action(MeetingSupportStarten(U, M),
PRECOND:,
EFFECT: MeetingSupportGestartet(U, M)).
Action(UserParameterLaden(U),
PRECOND: MeetingSupportGestartet(U, M),
EFFECT: UserParameterGeladen(U)).
Action(ZeitplanLaden(U, M),
PRECOND: UserParameterGeladen(U),
EFFECT: ZeitplanGeladen(U, M)).
Action(ÜberprüfenObZeitImMeetingVorhanden(M),
PRECOND: ZeitplanGeladen(U, M),
EFFECT: ÜberprüftObZeitImMeetingVorhanden(M)).
Action(ÜberprüfenObZeitFürPräsentationVorhanden(M),
PRECOND: ÜberprüftObZeitImMeetingVorhanden(M),
EFFECT: ÜberprüfObZeitFürPräsentationVorhanden(M)).
Action(UserBenachrichtigen(U),
PRECOND: Restzeit(M) < j min,
EFFECT: UserBenachrichtigt(U)).
Action(ÜberprüfenObMeetingInnerhalbDerZeitBeendet(U, M),
PRECOND: UserBenachrichtigt(U),
EFFECT: MeetingRechtzeitigBeendet(U, M) $\vee$ MeetingNichtRechtzeitigBeendet(U, M)).
Action(ÜberprüfenObLetztesMeetingErreicht(U, M),
PRECOND: MeetingRechtzeitigBeendet(U, M),
EFFECT: LetztesMeetingErreicht(U, M) $\vee$ LetztesMeetingNichtErreicht(U, M)).
Action(ZeitplanUmorganisieren(U, M),
PRECOND: MeetingNichtRechtzeitigBeendet(U, M),
EFFECT:ZeitplanUmorganisiert(U, M)).
Action(MeetingSupportBeenden(U, M),
PRECOND: LetztesMeetingErreicht(U, M),
EFFECT: MeetingSupportBeendet(U, M)).

Algorithmus 14 Partial Order Plan: Zeitplan umorganisieren

Init(MeetingSupportGestartet(U, M)).
Goal(ZeitplanUmorganisiert(U, M)).
Action(UmorganisierungStarten(U, M),
PRECOND: MeetingSupportGestartet(U, M) $\vee$ UmorganisationNichtAbgeschlossen(M),
EFFECT: DauerVerlängern(M) $\vee$ DauerVerkürzen(M) $\vee$ MeetingVerschieben(M)).
Action(DauerVerlängern(M),
PRECOND: UmorganisationGestartet(M),
EFFECT: DauerVerlängert(M)).
Action(DauerVerkürzen(M),
PRECOND: UmorganisationGestartet(M),
EFFECT: DauerVerkürzt(M)).
Action(MeetingVerschieben(M),
PRECOND: UmorganisationGestartet(M),
EFFECT: MeetingVerschoben(M)).
Action(ÜberprüfenObUmorganisationAbgeschlossen(M),
PRECOND: DauerVerlängert(M) $\vee$ DauerVerkürzt(M) $\vee$ MeetingVerschoben(M),
EFFECT: UmorganisationAbgeschlossen(M) $\vee$ UmorganisationNichtAbgeschlossen(M)).

Algorithmus 15 Partial Order Plan: Benutzer einladen

Init(MeetingSupportGestartet(U, M)).
Goal(UserEinladen(U, M)).
Action(MeetingAuswählen(M),
PRECOND: EinladungGestartet(U, M)).
EFFECT: MeetingAusgewählt(M)).
Action(UserAuswählen(U),
PRECOND: MeetingAusgewählt(M),
EFFECT: UserAusgewählt(U)).
Action(EinladungSchicken(U, M),
PRECOND: MeetingAusgewählt(M) $\wedge$ UserAusgewählt(U),
EFFECT: EinladungBestätigt(U, M) $\vee$ EinladungZurückgewiesen(U, M)).
Action(UserFürMeetingBuchen(U, M),
PRECOND: EinladungBestätigt(U, M),
EFFECT: UserEingeladen(U, M)).

Anhang C: Algorithmenverzeichnis

1 allgemeiner Aufbau von KQML 12

2 Beispiel für ein Aktionsschema 18
3 Geordnetes Planen: Beispiel 20

4 Partial Order Plan: Agenda 56
5 starte Meeting Support 70
6 Meeting Support 70
7 Reorganisation von Meetings 74
8 Karrierleiter 74
9 Einladung aussprechen 75
10 auf Einladung überprüfen 75

11 Partial Order Plan: Karriereleiter III
12 Partial Order Plan: Start Meeting Support IV
13 Partial Order Plan: Meeting Support V
14 Partial Order Plan: Zeitplan umorganisieren VI
15 Partial Order Plan: Benutzer einladen VI

Anhang D: Abbildungsverzeichnis

2.1 Anwendungsbereich von intelligenten Agenten (adaptiert nach Gilbert et. al., 1995) [35] 8
2.2 Agententypologie adaptiert nach Nwana [51] 9
2.3 Agententaxonomie adaptiert nach Franklin und Graesser [33] 10

3.1 Vollständig geordneter Plan - graphische Darstellung 19
3.2 Teilweise geordneter Plan - graphische Darstellung 21

4.1 Übersicht Easy Meeting nach [21]. 28
4.2 Digital-Me Architektur nach [25] 36

5.1 interne Architektur eines JADE-Agenten 47
5.2 verteilte Archtitektur eines JADE-Agenten 50

6.1 Use Case: Agenda 55
6.2 Partial Order Plan: Agenda 56
6.3 Use Case: Karriereleiter 58
6.4 Partial Order Plan: Karriereleiter 59
6.5 Use Case: Meeting Support 62
6.6 Partial Order Plan: Meeting Support 63
6.7 Use Case: Zeitplan umorganisieren 64
6.8 Partial Order Plan: Zeitplan umorganisieren 65
6.9 Use Case: Benutzer einladen 66
6.10 Partial Order Plan: Benutzer einladen 67

7.1 Screenshot: Hauptmenü 77
7.2 Screenshot: eigene Meetings bzw. Präsentationen 78
7.3 Screenshot: Meeting Info 78
7.4 Screenshot: Registration 79
7.5 Screenshot: Benutzerprofil 80
7.6 Screenshot: Benutzer einladen 81
7.7 Screenshot: Einladung 82
7.8 Screenshot: externe Links 83
7.9 innerhalb der Zeit 84
7.10 Dauer überzogen 84
7.11 Screenshot: Meetingsupport 84
7.12 Teilnehmer 84
7.13 Vortragender 84
7.14 Screenshot: Evaluation 84

Anhang E: Abkürzungsverzeichnis

ACL ..... Agent Communication Language

AMQ ..... Agent Message Queue

CLDC ..... Connected Limited Device Configuration

DAI ..... Distributed Artificial Intelligence (= verteilte künstliche Intelligenz)

FIPA ..... Foundation for Intelligent Physical Agents

FIPA ..... Foundation for Intelligent Physical Agents

HCI ..... Human Computer Interaction

IBM ..... International Business Machines Corporation

IEEE ..... Institute of Electrical and Electronics Engineers, Inc.

IP ..... Internet Protocol

JADE ..... Java Agent Development Environment

KQML ..... Knowledge Query and Manipulation Language

MB ..... Megabyte

MIDP ..... Mobile Information Device Profile

OWL ..... Web Ontology Language

PDA ..... Personal Digital Assistant

RDF ..... Resource Description Framework

SQL ..... Structure Query Language

TCP ..... Transmision Control Protocol

WLAN ..... Wireless Local Area Network

XHTML ..... eXtensible Hypertext Markup Language

Literaturverzeichnis

[1] *Nokia N91*. http://www.nokia.at/A4265288, 2005.

[2] *Apache Webserver*. http://httpd.apache.org/, 2008.

[3] *JADE*. http://jade.tilab.com/, 2008.

[4] *Java*. http://java.sun.com/, 2008.

[5] *MySQL*. http://www.mysql.de/, 2008.

[6] *Sun*. http://www.sun.com/, 2008.

[7] ABOWD, GREGORY D.: *Towards a Better Understanding of Context and Context-Awareness*. In: *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, Seiten 304–307, London, UK, 1999. Springer-Verlag.

[8] ANDERSON, BEN: *Minimising Conceptual Baggage: Making Choices about Metaphor*. In: *IX, Proceedings of HCI 1994, Cockton*, Seiten 179–194. University Press, 1994.

[9] ANDERSON, BEN: *Everyday Theories, Cognitive Anthropology and User-centered System Design*. In: *System Design, People and Computers X, Proceedings of HCI 1995*, Seiten 121–135. University Press, 1995.

[10] ANUPAM, JOSHI: *Intelligent Agents Meet the Semantic Web*, 2004.

[11] BIXBY, ROBERT: *Doc. Math. J. DMV 645 On the Solution of Traveling Salesman Problems*, 1998.

[12] BRADSHAW, JEFFREY: *Software Agents*. MIT Press, 1999.

[13] BROWN, PETER: *Context-aware applications: from the laboratory to the marketplace*. IEEE Personal Communications, Seiten 4–5, 1997.

[14] BYLANDER, TOM: *The computational complexity of propositional STRIPS planning*. Artificial Intelligence, 69:165–204, 1994.

[15] CAPPABALLO, STEPHAN: *Die strukturationstechnische Analyse von Branchen*, Seite 26. Gabler Edition Wissenschaft, 2005.

[16] CARROLL, JEREMY J.: *Jena: Implementing the semantic web recommendations*, 2004.

[17] CAVALLUZZI, ADDOLORATA: *Emotional Dialogs with an Embodied Agent*. In: *Lecture Notes in Computer Science*, Seite 145. Springer-Verlag, 2003.

[18] CHANG, AI-MEI: *Design of an argumentation-based negotiation support system*. In: *HICSS '95: Proceedings of the 28th Hawaii International Conference on System Sciences*, Seite 242, Washington, DC, USA, 1995. IEEE Computer Society.

[19] CHEN, GUANLING: *A Survey of Context-Aware Mobile Computing Research*. Technischer Bericht, Department of Computer Science, Dartmouth College, 2000.

[20] CHEN, HARRY: *An Ontology for Context-Aware Pervasive Computing Environments*. Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, 18:197–207, 2003.

[21] CHEN, HARRY: *Intelligent Agents Meet Semantic Web in a Smart Meeting Room*. In: *3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, 2004.

[22] CHEN, HARRY: *Semantic Web in the Context Broker Architecture*. In: *In Proceedings of PerCom 2004*, Seiten 277–286, 2004.

[23] CHOUDHRI, AMIT: *PatientService: Electronic Patient Record Redaction and Delivery in Pervasive Environments*. Fifth International Workshop on Enterprise Networking and Computing in Healthcare Industry, 2003.

[24] COOK, DIANE: *Smart Environments: Technology, Protocols and Applications (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, 2004.

[25] COZZOLONGO, GIOVANNI: *A Personal Agent Supporting Ubiquitous Interaction.* In: *WOA*, Seiten 55–61. Pitagora Editrice Bologna, 2004.

[26] DARDENNE, ANNE: *Goal-Directed Requirements Acquisition*. Science of Computer Programming, 20(1-2):3–50, 1993.

[27] DENNETT, DANIEL: *The Intentional Stance*. MIT Press, Cambridge, Mass., 1987.

[28] ETZIONI, OREN: *Intelligent agents on the internet: Fact, fiction, and forecast*. IEEE Expert, 10:44–49, 1995.

[29] FALCONE, RINO: *Tuning the Collaboration Level with Autonomous Agents: A Principled Theory*. In: *AI*IA 01: Proceedings of the 7th Congress of the Italian Association for Artificial Intelligence on Advances in Artificial Intelligence*, Seiten 212–224, London, UK, 2001. Springer-Verlag.

[30] FIKES, RICHARD E.: *STRIPS: A new approach to the application of theorem proving to problem solving*. Artificial Intelligence, 2:189–208, 1971.

[31] FIPA: *Agent Communcation Language Specifications*. FIPA, http://www.fipa.org/repository/aclspecs.html, 2000.

[32] FONER, LEONARD N.: *What's an Agent Anyway? - A Sociological Case Study*. FTP Report - MIT Media Lab, May 1993.

[33] FRANKLIN, STAN: *Is It an Agent, or Just a Program?: A Taxonomy for Autonomous Agents*. In: *Third International Workshop on Agent Theories, Architectures, and Language*, Seiten 21–35. Springer-Verlag, 1996.

[34] FRITZ REINHARDT, HEINRICH SOEDER: *dtv - Atlas der Mathematik*, Seite 17. dtv - Deutscher Taschenbuchverlag, 1998.

[35] GILBERT, DON: *White paper on intelligent agents*, 1995.

[36] GREEN, SHAW: *Software Agents: Review*. Technischer Bericht TCS-CS-1997-06, Trinity College Dublin, Dublin, 1997.

[37] HAMILTON, A. G.: *Logic for mathematicians*. Cambridge University Press, 1978.

[38] HOFFMANN, ANTJE: *Interaktion zwischen Anbietern und Nachfragern bei der Vermarktung und Beschaffung innovativer Dienstleistungen*, Seite 123. Gabler Edition Wissenschaft, 2006.

[39] KALLE LYYTINEN, PETRI MAARANEN: *Groups are not always the same: An analysis of group behaviours in electronic meeting systems*. CSCW: An International Journal, 2(4):263–286, 1994.

[40] KHALLOUF, JASON: *Towards goal-oriented design of agent systems*. Quality Software, 2005. (QSIC 2005). Fifth International Conference on, Seiten 389–394, 2005.

[41] KNUUTTILA, JUHA S.: *DeleGate: towards a conference diplomat's multidimensional workstation*. In: *HICSS '95: Proceedings of the 28th Hawaii International Conference on System Sciences*, Seite 513, Washington, DC, USA, 1995. IEEE Computer Society.

[42] KOPFER, HERBERT: *Heuristische Suche in Operations Research und Künstlicher Intelligenz*, Seite 19. Habilitationsschrift: Freie Universität Berlin, 1989.

[43] LAMSWEERDE, AXEL VAN: *Goal-oriented requirements engineering: from system objectives to UML models to precise software specifications*. In: *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, Seiten 744–745, Washington, DC, USA, 2003. IEEE Computer Society.

[44] LYYTINEN, KALLE: *Unusual business or Business as Usual: an Investigation of Meeting Support Requirements in Multilateral Diplomacy*. Accounting, Management and Information Technology, 3(2), 1993.

[45] MAARANEN, PETRI: *Designing meeting support systems in a user-centered manner: the case of the Helsinki prototype system*. In: *COCS '95: Proceedings of conference on Organizational computing systems*, Seiten 336–346, New York, NY, USA, 1995. ACM.

[46] MOIZUMI, KATSUHIRO: *The mobile-agent planning problem*. Doktorarbeit, Thayer School of Engineering Dartmouth College, Hanover (New Hampshire), 1998.

[47] MOULIN, BERNARD: *An Overview of Distributed Artificial Intelligence*. Foundations of Distributed Artificial Intelligence, Seiten 3–56, 1996.

[48] NEGROPONTE, NICHOLAS: *Being Digital*. Hodder and Stroughton, Seite 155, 1995.

[49] NICHOLAS R. JENNINGS, MICHAEL J. WOOLDRIDGE: *Applications of Intelligent Agents*. In: JENNINGS, NICHOLAS R. und MICHAEL J. WOOLDRIDGE (Herausgeber): *Agent Technology: Foundations, Applications, and Markets*, Seiten 3–28. Springer-Verlag: Heidelberg, Germany, 1998.

[50] NIJHOLT, ANTON: *Meetings and meeting modeling in smart environments.* AI Soc., 20(2):202–220, 2006.

[51] NWANA, HYACINTH S.: *Software agents: An overview*. Knowledge Engineering Review, 11(3):205–244, 1996.

[52] RAO, ANAND S.: *BDI-agents: from theory to practice*. In: *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.

[53] REIF, GERALD: *Moderne Aspekte der Wissensverarbeitung*. Gerald Reif, 2000.

[54] SCHMIDT, ALBRECHT: *Implicit human computer interaction through context*. Personal and Ubiquitous Computing, 4(2-3):191–199, 2000.

[55] SHOHAM, YOAV: *An overview of agent-oriented programming*. Software agents, Seiten 271–290, 1997.

[56] SINGH, MUNINDAR P.: *Multiagent Systems*, Band 799. Springer Verlag, Berlin, 1994.

[57] STOLYAR, ABRAM A.: *Introduction to Elementary Mathematical Logic*. Dover Publ, 1983.

[58] STUART, RUSSELL: *Artificial Intelligence - A Modern Approach*, Seiten 240–253. Prentice Hall, 2003.

[59] SUN: *Sun Wireless Toolkit*. http://java.sun.com/products/sjwtoolkit/, 2008.

[60] TANENBAUM, ANDREW S.: *Verteilte Systeme*. Pearson Studium, 2., Aufl. Auflage, 2002.

[61] UNDERCOFFER, JEFFREY: *A Secure Infrastructure for Service Discovery and Access*. In: *The Journal of Special Issues on Mobility of Systems, Users, Data and Computing*, Seiten 113–125, 2003.

[62] VAUCHER, JEAN: *JADE Tutorial and Primer*. http://www.iro.umontreal.ca/ vaucher/Agents/Jade/JadePrimer.html, 2003.

[63] VELDE, WALTER VAN DE: *Cognitive architectures - from knowledge level to structural coupling*. In: *The Biology and Technology of Intelligent Autonomous Agents*, Seiten 197–221. Springer-Verlag, 1995.

[64] YAN, HAO: *Context-aware office assistant*. In: *Intelligent User Interfaces*, Seiten 276–279, 2000.