Eine modulare Architektur für dienstbasierte Interaktionen zwischen Agenten

Ralf Sesseler

Eine modulare Architektur für dienstbasierte Interaktionen zwischen Agenten

vorgelegt von
Diplom-Informatiker
Ralf Sesseler

Von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –
genehmigte Dissertation

Promotionsausschuß:

Vorsitzender: Prof. Dr. Reinhold Orglmeister
Berichter: Prof. Dr. Hermann Krallmann
Berichter: Prof. Dr. Erhard Konrad

Tag der wissenschaftlichen Aussprache: 16. 01. 2002

Berlin 2002
D 83

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit am DAI-Labor der TU Berlin. Sie basiert vorwiegend auf den durchgeführten Arbeiten des drittmittelgeförderten Projekts AARFTA (Agent Architecture for the Realization of Future Telecommunication Applications). Ergebnisse aus diesem Projekt wurden bereits veröffentlicht in [Fricke et al. 2001], [Sesseler 2001], [Sesseler & Albayrak 2001a], [Sesseler & Albayrak 2001b] und [Sesseler & Albayrak 2002].

Mein Dank gilt dem Betreuer der Arbeit Professor Krallmann, dem Gutachter Professor Konrad und dem Leiter des DAI-Labor Dr. Albayrak für ihre Unterstützung bei der Erstellung dieser Arbeit. Weiterhin danke ich allen meinen Mitarbeitern am DAI-Labor und insbesonderen den an der Implementierung von JIAC beteiligten, allen voran Siegfried Ballmann.

Ralf Sesseler.

Überblick

Mit der zunehmenden Vernetzung vor allem durch das Internet entsteht eine Computerinfrastruktur, die neue Möglichkeiten zur Gestaltung von Dienstplattformen mit sich bringt, aber auch neuartige Anforderungen stellt. Anbieter, Vermittler und Konsumenten von Waren und Dienstleistungen profitieren von der hohen Flexibilität und Dynamik wie auch von dem orts- und zeitunabhängigen Zugang. Dieses Potential läßt sich jedoch nur dann voll ausschöpfen, wenn die Offenheit, Verteiltheit und Heterogenität der Computernetze durch neue Technologien voll beherrschbar und nutzbar gemacht werden kann.

Einen interessanten Ansatz hierfür bieten Softwareagenten, die als autonome Spezialisten in den Netzen agieren und untereinander interagieren. Das in dieser Arbeit konzipierte CASA ist eine Architektur für Multiagentensysteme, die sich aufgrund ihrer Modularität sowie der Flexibilität und Interoperabilität der damit erstellten Agenten besonders als Basisplattform zur Realisierung von Dienstumgebungen in elektronischen Netzen eignet. CASA umfaßt die drei Designebenen Agentenaufbau, Agentenkontrolle und Agentengesellschaft sowie die formale Sprache CAL zur Spezifikation von Agenten.

Einzelne Agenten werden je nach Bedarf und Aufgabe aus wiederverwendbaren Komponenten zusammengesetzt. Auch noch zur Laufzeit können Komponenten umkonfiguriert sowie hinzugefügt, entfernt oder ausgetauscht werden, ohne die Arbeit des Agenten zu beeinträchtigen. So lassen sich Agenten einfach erstellen und an veränderte Aufgaben anpassen.

Die Steuerung von Agenten beruht auf einem wissensbasierten Kontrollschema, das reaktives, zielgerichtetes und interaktives Verhalten zu einer flexiblen Verhaltenssteuerung integriert. Die Verwendung von formalen Wissensstrukturen bietet dabei einerseits eine abstrakte Ebene zur Spezifikation und Analyse von Agenten, andererseits bildet sie die Grundlage der autonomen Verhaltenskontrolle.

Zudem ermöglicht das formale Wissen die innerhalb von Agentengesellschaften benötigte Interoperabilität, indem es als Inhalt der Kommunikation und zur Beschreibung der Interaktionsmöglichkeiten der Agenten dient. Dabei werden Interaktionen als Dienste aufgefaßt, die Agenten anbieten und nutzen. Ein Dienst ist eine Handlung, die ein Agent für einen anderen ausführt. Zusammen mit einer Agenteninfrastruktur, die Agenten und Dienste verwaltet, erlaubt das Dienstschema eine flexible, dynamische Zusammenarbeit von Agenten und die Integration von Interaktionen in die Verhaltenssteuerung.

Die Sprache CAL ist eine Kombination aus formaler Logik und Programmiersprache, mit der sich sämtliche Wissensaspekte von Agenten ausdrücken lassen.

CASA ist eine offene und skalierbare Architektur, mit der sich nach dem Baukastenprinzip flexible, dynamische Multiagentensysteme erstellen lassen. Alle Architekturebenen unterstützen dazu ein hohes Maß an Modularität und Interoperabilität.

Abstract

Computer networks like the Internet provide a novel infrastructure bearing new possibilities for the realisation of service platforms, while raising new requirements as well. Suppliers, brokers, and consumers of goods and services profit from their flexibility and dynamics as well as from the independency of time and location. To benefit from this potential, the open, distributed, and heterogeneous character of the networks needs to be utilised by new technologies.

A promising approach are software agents, which are able to act and interact as autonomous specialists in the networks. CASA is an architecture for multi-agent systems that offers a modular structure and agents with a high degree of flexibility and interoperability. Thus, it provides a suitable basis for the realisation of service platforms in computer networks. CASA is divided into three design levels for agent composition, agent control, and agent societies, which are completed by the formal language CAL for agent specification.

Individual agents are assembled from reusable components depending upon their tasks. Even at run-time, components can be reconfigured as well as added, removed, or exchanged without affecting the current work of the agent. Thus, agents are created and adapted to changing tasks easily.

Agent control empoys a knowledge-based control scheme that integrates reactive, deliberative, and interactive behaviour. Formal knowledge structures provide an abstract layer for the specification and analysis of agents and form the basis of a flexible, autonomous behaviour control.

In addition, formal knowledge is utilised as the content of communications and to describe the interactive capabilities of agents, thereby enabling the interoperability required by agent societies. Interactions are represented as service acts provided and used by agents in order to delegate tasks. In combination with an agent infrastructure that administers agents and services, the service scheme allows flexible and dynamic collaborations among agents and the integration of interactions into the behaviour control.

The agent description language CAL is a combination of formal logic and programming language concepts. It covers all aspects of the representation, use, and communication of knowledge by agents.

CASA is an open and scalable architecture to build flexible, dynamic multi-agent systems that supports modularity and interoperability at all architectural layers.

Inhalt

Abbildungen

Definitionen

Abkürzungen

ACC	Agent Communication Channel
ACL	Agent Communication Language
AID	Agent Identifier
AMS	Agent Management System
AOP	Agentenorientierte Programmierung
AP	Agent Platform
BDI	Belief, Desire, Intention
CASA	Component Architecture for Service Agents
CAL	CASA Agent Language
CCL	Constraint Choice Language
CCM	CORBA Component Model
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CSCW	Computer Supported Cooperative Work
DCOM	Distributed Component Object Model
DF	Directory Facilitator
dMARS	distributed Multi-Agent Reasoning System
DVMT	Distributed Vehicle Monitoring Testbed
EJB	Enterprise JavaBeans
FIFO	First In, First Out
FIPA	Foundation for Intelligent Physical Agents
HAP	Home Agent Platform
HTML	Hypertext Markup Language
IDL	Interface Definition Language
IP	Internet Protocol
JADE	Java Agent Development Environment
JIAC IV	Java Intelligent Agent Componentware, Version 4
JVM	Java Virtual Machine
KA	Knowledge Area
KEE	Knowledge Engineering Environment
KI	Künstliche Intelligenz
KIF	Knowledge Interchange Format
KQML	Knowledge Query and Manipulation Language

KSE	Knowledge Sharing Effort
MASIF	Mobile Agent System Interoperability Facilities
MIDL	Microsoft Interface Description Language
MTS	Message Transport System
OA	Ontology Agent
OKBC	Open Knowledge Base Connectivity
OMG	Object Management Group
OOP	Objektorientierte Programmierung
ORB	Object Request Broker
PGP	Partial Global Planning
PLACA	Planning Communicating Agents
PRS	Procedural Reasoning System
PRS-CL	Procedural Reasoning System – Common Lisp
RADL	Reticular Agent Definition Language
RDF	Resource Description Framework
RMI	Remote Method Invocation
SL	Semantic Language
UM-PRS	University of Michigan – Procedural Reasoning System
URL	Uniform Resource Locator
VKB	Virtual Knowledge Base
VKI	Verteilte Künstliche Intelligenz
WML	Wireless Markup Language

1. Einleitung

1.1. Umfeld und Motivation

Computernetzwerke wie das Internet werden vermehrt zur Bereitstellung elektronischer Dienste genutzt, die über die reine Präsentation von Informationen weit hinausgehen, und auch im Telekommunikationsbereich beschränken sich die angebotenen Dienstleistungen nicht mehr nur auf das Herstellen von Verbindungen. Das Spektrum dieser Dienste umfaßt Angebote an den Endkunden wie interaktive Informationsdienste, Telematikdienste und elektronischen Handel (electronic commerce) sowie Infrastrukturdienste wie Netzwerkmanagement und Routing. Allen gemeinsam ist der Einsatz der neuen Technologien, um Dienste schnell und effizient und weitgehend unabhängig von Zeit und Ort der Nutzung bereitstellen und erbringen zu können.

Bei wachsendem Leistungsumfang der Dienste bringen diese elektronischen Dienstumgebungen neue Möglichkeiten und Anforderungen mit sich. Ein besonderes Merkmal dieser Systeme ist dabei ihre Heterogenität, Verteiltheit und Offenheit. Die Infrastruktur ist sehr heterogen, sie besteht aus verschiedenen Arten von Netzwerken, unterschiedlichen Endgerätetypen sowie einer Vielzahl an Betriebssystemen, Plattformen und Zugangssoftware. Die Ressourcen wie Hardware und Daten sind räumlich und logisch stark verteilt. Das Dienstangebot ist offen und verändert sich dynamisch: Dienste kommen neu hinzu, werden ersetzt oder in ihrer Funktionalität umgestaltet und erweitert. Deshalb muß eine entsprechende Interoperabilität gewährleistet sein, die einerseits die Heterogenität und Verteiltheit des Systems weitgehend transparent und vernachlässigbar macht und andererseits eine möglichst flexible Kombination von Diensten erlaubt. Interoperabilität auf der Infrastrukturebene macht Anbieter und Nutzer unabhängig von spezifischer Hard- und Software. Auf der Dienstebene ermöglicht die Interoperabilität eine bedarfsgerechte Verknüpfung von Diensten zu neuen Angeboten und Leistungen.

Weitere Anforderungen ergeben sich jeweils relativ zu den verschiedenen Rollen, die in den Anbietern von Infrastruktur und Diensten sowie in den Nutzern bestehen.

Bei den Infrastrukturanbietern kann nochmals unterschieden werden zwischen der eigentlichen Dienstplattform und erweiternden Funktionalitäten, z.B. zur Suche und Vermittlung von Diensten. Die angebotenen Dienste umfassen primäre Leistungen, die direkt vom Anbieter erbracht werden, und sekundäre Leistungen, die auf andere Dienste aufsetzen. Die Dienstnutzer sind sowohl die Endnutzer, als auch die Anbieter der Sekundärleistungen.

Sämtliche Anbieter von Leistungen benötigen Mittel zur Kontrolle ihres Angebots sowie einzelner Vorgänge während der Umsetzung der Dienste. Dies umfaßt die Konfiguration, Wartung und flexible Erweiterung des Angebots, wobei die Skalierbarkeit zur Anpassung an den jeweiligen Leistungsbedarf eine wichtige Rolle spielt. Weiterhin sind Managementfunktionalitäten, beispielsweise zur Fehlerbehebung und Performanzkontrolle, notwendig. Erbrachte Leistungen müssen nachvollziehbar protokolliert und verläßlich abgerechnet werden können. Insbesondere für die Anbieter von Diensten eröffnen sich neue Vermarktungsmöglichkeiten und neuartige Dienste und Funktionalitäten sowie eine verbesserte Qualität und Aktualität ihrer Angebote und Leistungen aufgrund der durch die neuen Technologien bereitgestellten Dynamik und Flexibilität. Für die Vermittler von Diensten hingegen ergibt sich ein Bedarf nach inhaltlichen Strukturierungsmöglichkeiten, um mit der enormen Menge und Vielfalt an Diensten und deren Dynamik effektiv umgehen zu können.

Für den Nutzer steht eine flexible Dienstauswahl und -konfiguration im Vordergrund, um die jeweiligen Ansprüche und Bedürfnisse individuell zu erfüllen. Dazu leistet die bedarfsgerechte und automatisierte Kombination von Diensten einen wichtigen Beitrag. Dabei muß aber eine hinreichende Kontrolle des Nutzers über die Dienstnutzung und insbesondere die anfallenden Kosten gewährleistet sein. Der Endnutzer verlangt zudem nach einem hohen Bedienungskomfort, wozu auch die Personalisierbarkeit der Dienste und die Möglichkeit der Verwendung mobiler Endgeräte gehört.

Hinzu kommen noch klassische Anforderungen wie die Stabilität und Effizienz des Systems sowie die Absicherung der im Zusammenhang mit Dienstnutzungen vorgenommenen geschäftlichen Transaktionen hinsichtlich der Vertraulichkeit und Integrität.

Herkömmliche Client/Server-Ansätze werden diesen neuen Anforderungen nicht hinreichend gerecht und lassen viele der neuen Möglichkeiten noch ungenutzt. Sie erweisen sich als zu unflexibel, um der Offenheit der Systeme voll Rechnung zu tragen. Die Interoperabilität beschränkt sich auf die Infrastrukturebene, Dienste lassen sich nicht dynamisch auswählen oder zu sekundären Leistungen kombinieren. Auch die Bereitstellung und Pflege von Diensten gestaltet sich aufwendig, da viele der benötigten Basisfunktionalitäten immer wieder neu realisiert werden müssen, weil sie gar nicht oder nur rudimentär von den Dienstplattformen unterstützt werden. Als Alternative bietet sich der Einsatz von Agenten an, die aufgrund ihrer Autonomie und ihrer Interaktionsfähigkeiten Dienstnutzungen flexibilisieren und automatisieren können.

1.2. Ansatz

Agenten sind Softwareeinheiten, die selbständig eine Aufgabe erfüllen können und dazu mit anderen Agenten über eine abstrakte Kommunikationssprache interagieren. Damit eignen sie sich besonders für den Einsatz in heterogenen, verteilten, offenen Systemen wie den oben beschriebenen Dienstplattformen. In der vorliegenden Arbeit wird mit CASA (Component Architecture for Service Agents) eine Agentenarchitektur für offene Dienstumgebungen vorgestellt, die eine flexible Bereitstellung und Nutzung von Diensten unterstützen soll.

Sämtliche Interaktionen zwischen Agenten geschehen deshalb in CASA auf der Basis von Dienstnutzungen. Ein Dienst wird dabei aufgefaßt als eine Handlung, die ein Agent für einen anderen ausführt, und enthält eine formale Beschreibung, die Vorbedingungen und Effekte einer Dienstnutzung explizit deklarieren. Auf diese Weise kann ein Agent die Dienste anderer Agenten in einer analogen Weise zu seinen eigenen Handlungen zur Erfüllung seiner Aufgaben verwenden. Zusammen mit einer Infrastruktur zur Vermittlung von Agenten und Diensten wird so die Offenheit des Systems für eine dynamische und automatisierte Dienstauswahl genutzt. Dabei bietet das Konzept des Dienstes als Grundlage der Interaktionen sowohl einen intuitiv verständlichen Rahmen für Nutzer und Programmierer, als auch eine explizite Basis zur Protokollierung, Abrechnung und Absicherung von Interaktionen zwischen Agenten. Der Verlauf dieser Interaktionen zur Nutzung von Diensten wird durch Konversationsprotokolle gesteuert und unter Verwendung von Sprechakten in einer formalen Kommunikationssprache mit frei definierbaren Ontologien als Vokabular durchgeführt. Diese Konzepte gewähren ein hohes Maß an flexibler Interoperabilität auf einer abstrakten Dienstebene.

Damit die Agenten mit den formalen Spezifikationen der Dienste umgehen können, basiert ihre interne Verhaltenssteuerung auf Mechanismen zum Speichern und Verarbeiten explizit repräsentierten formalen Wissens. Dabei wird strikt getrennt zwischen den deklarativen Wissensstrukturen, mit denen das Verhalten eines Agenten abstrakt beschrieben wird, und den prozeduralen Kontrollfunktionen, die auf diesen operieren und so das tatsächliche Verhalten bestimmen. Zusammen mit der Verwendung von formal spezifizierten Diensten ermöglicht dies eine flexible Steuerung des Verhaltens durch den Agenten zur Laufzeit. Zudem stellt das formale Wissen eine höherstufige Beschreibungsebene zur Entwicklung von Agenten und zur Analyse von deren Verhalten dar. Die Logik der Wissensrepräsentationssprache ist einerseits bei hinreichender Ausdrucksstärke möglichst einfach gehalten, um eine effiziente Verarbeitung zu ermöglichen, andererseits berücksichtigt sie explizit die subjektive Weltsicht des Agenten und das damit verbundene unvollständige Wissen. Diese Sprache bildet auch den Inhalt der Kommunikation zwischen Agenten.

Die Kontrollprozeduren werden realisiert durch Agentenkomponenten, von denen jede eine oder mehrere funktionale Rollen innerhalb des Agenten übernimmt.

Die Zusammensetzung eines Agenten aus Komponenten ist dabei offen und kann je nach Anforderungen und gemäß den Aufgaben eines Agenten frei bestimmt werden. Komponenten können auch noch zur Laufzeit hinzugefügt, ersetzt oder entfernt werden. Die dazu notwendige entkoppelte Interaktion zwischen den Komponenten wird durch einen Nachrichtenmechanismus gewährleistet, bei dem diese identifiziert werden anhand ihrer Rollen, die die jeweiligen Schnittstellen für Interaktionen festlegen. Diese Modularität im Aufbau der einzelnen Agenten sorgt für eine hinreichende Skalierbarkeit zur Konfiguration und Wartung eines Agenten und erleichtert die Integration zusätzlicher Funktionalitäten, beispielsweise für Sicherheit oder Management. Außerdem vereinfacht sie das Erstellen von Agenten, da gegebenenfalls auf bereits existierende Komponenten zurückgegriffen werden kann. Zur Realisierung von Agenten, die wie beschrieben auf der Basis formalen Wissens über Dienste mit anderen Agenten interagieren können, wird eine Standardarchitektur an Komponentenrollen vorgeschlagen, die reaktives, deliberatives und interaktives Verhalten unterstützt.

Insgesamt bilden der Aufbau der Agenten aus Komponenten, die wissensbasierte Verhaltenssteuerung und die Interaktion über formal spezifizierte Dienste die Grundlage für ein offenes, flexibles und dynamisches Agentensystem, das sich besonders zur Realisierung von elektronischen Dienstplattformen eignet.

1.3. Aufbau

Nachdem dieses Kapitel 1 die Motivation und den gewählten Ansatz der CASA-Agentenarchitektur vorgestellt hat, gibt das nachfolgende Kapitel 2 einen Überblick über Grundlagen und Techniken der Agententechnologie sowie über deren aktuellen Stand. Dazu werden als begrifflicher Rahmen die Konzepte des Agenten und der Agentenarchitektur erörtert. Anschließend werden bestehende Ansätze vorgestellt, auf denen die vorliegende Arbeit aufbaut. Abgeschlossen wird dieser Überblick durch Anwendungsfelder für Multiagentensysteme und eine kurze Vorstellung einiger mit CASA vergleichbarer Architekturen.

Mit Kapitel 3 beginnt die Beschreibung des entwickelten Agentensystems CASA. Es beinhaltet die Agentenhülle, die die interne Infrastruktur bereitstellt, anhand derer ein Agent aus Komponenten zusammengesetzt wird und diese miteinander interagieren können. Kapitel 4 entwirft eine Standardarchitektur an Komponenten für Agenten, deren reaktives, deliberatives und interaktives Verhalten auf einer abstrakten, deklarativen Ebene formalen Wissens spezifiziert werden kann. Für diese Standardarchitektur wird in Kapitel 5 ein Interaktionsschema ausgearbeitet, das auf Diensten mit formalen Beschreibungen basiert, und dieses in eine Infrastruktur für Agentengesellschaften integriert.

Das Kapitel 6 beschreibt eine Wissensrepräsentationssprache für die Standardarchitektur von CASA. Diese kombiniert objektorientierte Datenstrukturen mit einer dreiwertigen Logik, die die Unvollständigkeit des Wissens von Agenten berücksichtigt. Für diese Logik werden eine Syntax und die Semantik angegeben. Hinzu kommen noch Repräsentationsformalismen für prozedurales Wissen und zur Kommunikation.

Das abschließende Kapitel 7 faßt den vorgestellten Entwurf der Agentenarchitektur CASA zusammen und beschreibt kurz das darauf aufbauende, implementierte System JIAC IV. Den Schluß bildet eine Gegenüberstellung mit anderen Agentenarchitekturen und ein Ausblick auf mögliche Erweiterungen.

2. Agententechnologie

Die Agententechnologie hat viele verschiedene Ursprünge und Einflußgebiete, deren Ansätze und Ergebnisse sie aufgreift und zusammenfügt zur Konzeption und Realisierung von Agentensystemen. Nicht zuletzt aufgrund der Vielzahl an unterschiedlichen Einflüssen ist das Gebiet der Agententechnologie sehr verzweigt und uneinheitlich, so daß im folgenden zunächst einige vorherrschende Begriffe und Definitionen für Agenten näher diskutiert werden, um aus diesen den für CASA zugrundegelegten Agentenbegriff abzuleiten.

Anschließend werden einige für die vorliegende Arbeit relevante Techniken zu den Themen Architektur, Einzelagent und Agentengesellschaft vorgestellt. Der Aufbau und die Kontrollstruktur eines Agenten werden bestimmt durch seine Architektur, für die es verschiedene Gestaltungsansätze gibt. Zur Steuerung des Verhaltens des einzelnen Agenten dienen häufig in der Künstlichen Intelligenz (KI) entwickelte wissensbasierte Mechanismen. Und zur Bildung von Agentengesellschaften werden Organisationsstrukturen sowie Techniken zur Kommunikation, Koordination und Kooperation zwischen den Agenten verwendet. Auf dieser letztgenannten Ebene finden auch einige Standardisierungsbemühungen zur Interoperabilität verschiedener Agentensysteme statt.

Aufgrund ihrer spezifischen Charakteristika eignen sich derartige Multiagentensysteme besonders für bestimmte Anwendungsbereiche, die kurz vorgestellt werden. Den Abschluß bildet ein Überblick über einige existierende Architekturen für Multiagentensysteme.

2.1. Grundlagen

Zu den Grundlagen der Agententheorie zählen zum einen Begriffsbestimmungen und Definitionen, zum anderen abstrakte Charakterisierungen von Agenten über Eigenschaften und formale Modelle.

2.1.1. Der Agentenbegriff

In den letzten zehn bis fünfzehn Jahren wird in der Informatik zunehmend der Begriff des Agenten[1] als Grundlage der Entwicklung von komplexen und verteilten Softwaresystemen verwendet. Die Wurzeln dieses Paradigmas liegen vor allem in den Bereichen der Künstlichen Intelligenz und der verteilten Systeme – zusammengefaßt in der Verteilten Künstlichen Intelligenz (VKI) –, aber auch bei der objektorientierten Programmierung (OOP) und interaktiven Benutzerschnittstellen [Jennings et al. 1998]. Hinzu kommen noch Einflüsse aus so verschiedenen Bereichen wie der Organisationstheorie, der Entscheidungstheorie, der Kognitionswissenschaft und der Sprachphilosophie. Entsprechend vielfältig sind die Auffassungen, was genau ein Agent ist, ohne daß es dabei eine allgemein akzeptierte Ansicht zu geben scheint:

> *„Most agent developers have their own opinion on exactly what constitutes an agent – and no two developers appear to share the same opinion.“* [Wooldridge & Jennings 1998]

Zumindest lassen sich jedoch die folgenden Hauptrichtungen identifizieren, die sowohl einzeln, als auch in Kombination miteinander vertreten werden:

- *Der Agent als Objekt mit besonderer Kompetenz* [Shoham 1993]: Diese Sichtweise sieht den Agentenbegriff als eine Erweiterung des Objektbegriffs der OOP. Dabei werden Agenten als komplexe Objekte mit einem höheren Grad an Autonomie und sich daraus ergebenden flexibleren Interaktionen untereinander aufgefaßt.
- *Der Agent als Teil eines Multiagentensystems* [Genesereth & Ketchpel 1994]: Hierbei wird der Schwerpunkt des Agentenbegriffs auf die Interaktionen zwischen Agenten gelegt. Jeder Agent ist ein spezialisierter Problemlöser, der mit anderen Agenten kommuniziert und kooperiert, um gemeinsam eine Problemstellung zu bearbeiten.
- *Der Agent als autonom Handelnder* [Russel & Norwig 1995], [Franklin & Graesser 1996]: Dies ist die KI-Sicht auf Agenten. Dabei entscheidet ein Agent aufgrund seiner Wahrnehmung der Umgebung und seines Wissens über die Handlungen, die er in dieser Umgebung ausführt. Zur Umgebung gehören dabei i.a. auch andere Agenten, mit denen vermittels Kommunikation interagiert werden kann.
- *Der Agent als „mentales“ („intentionales“) System* [Shoham 1993], [Wooldridge & Jennings 1995]: Gemäß dieser Auffassung ist ein Agent jedes System, das sich sinnvoll mit Hilfe mentalistischer Begriffe wie „Wissen“, „Ziel“ usw. beschreiben läßt. Allerdings weicht die Verwendung dieser Begriffe oft nicht unerheblich von der üblichen Verwendung beim Menschen ab und wird als reine Zuschreibung von außen aufgefaßt.

---

[1] Agenten werden auch als autonome Agenten oder als intelligente Agenten bezeichnet. Jedoch ist Autonomie bereits im Begriff Agent enthalten (s.u.) und die Zuschreibung von Intelligenz selten uneingeschränkt gerechtfertigt, so daß zunehmend auf diese Zusätze verzichtet wird.

- *Der Agent als Stellvertreter* [Gilbert 1996]: Hier wird der Agent vor allem als Stellvertreter für seinen Benutzer angesehen. Er soll dabei selbständig die Interessen oder Ziele des Benutzers vertreten können, ohne daß dieser den Agenten permanent neu anzuweisen oder zu kontrollieren braucht.
- *Der Agent als Assistent* [Maes 1994]: Der Agent als Assistent beobachtet selbsttätig das Verhalten eines Benutzers und unterstützt ihn bei seiner Arbeit am Computer, indem er Aufgaben übernimmt und Hilfestellungen gibt. Dazu soll er sich jeweils an das Verhalten des Benutzers anpassen können und dessen Eigenschaften und Präferenzen lernen.

Auch wenn alle diese Sichtweisen verschiedene Gemeinsamkeiten untereinander aufweisen, so lassen sie sich doch nicht auf einen einheitlichen Begriff reduzieren, so daß, wenn die Rede von Agenten ist, jeweils genauer anzugeben ist, welcher oder welche Kombination dieser Agentenbegriffe tatsächlich gemeint ist. Das einzige allen Auffassungen von Agenten gemeinsame Kriterium scheint die Autonomie zu sein [Wooldridge 1999], diese allein reicht jedoch als Begriffsbestimmung nicht aus.

Das Agentenkonzept verspricht gegenüber anderen Softwarelösungen eine Reihe von Vorzügen, insbesondere im Bereich komplexer verteilter Systeme, die sich mit Agenten als elementaren Bestandteilen einfach und intuitiv modellieren lassen. Als eigenständig arbeitende Prozesse bieten Agenten Vorteile im Bereich der Effizienz durch parallele und verteilte Ausführung sowie der Robustheit und Ausfallsicherheit durch Redundanz. Aufgrund der Konzeption als offene, modulare, kommunikationsbasierte Systeme ergibt sich zudem ein Mehr an Flexibilität und Dynamik. Vor allem aber die Autonomie verspricht Computersysteme, die eine wesentlich komfortablere Art der Programmierung und Benutzung erlauben, da sie keiner so vollständigen und expliziten Kontrolle mehr bedürfen.

Gerade diese intuitive Vorstellung des Agenten als selbständig denkende und handelnde Einheit führt aber auch zu überzogenen Erwartungen und Anthropomorphismen, wie sie bereits der KI auf längere Sicht eher geschadet als genutzt haben. Der Agentenbegriff verkommt dabei zunehmend zu einem willkommenen, aber leeren Schlagwort für eine Vermarktungsstrategie, mit der zur Abwechslung der Mythos der „intelligenten“ Maschine propagiert werden soll – allerdings wie schon so oft ohne wirkliche Neuerungen oder Fortschritte außerhalb der Rhetorik.

Dabei wird nur allzu gerne übersehen, daß es sich bei allen genannten Auffassungen übereinstimmend um Forschungsziele und erste Ansätze und nicht um bereits erreichte Resultate und ausgereifte Technologien handelt. Realisierte Agentensysteme sind meist Prototypen und stellen nur erste Schritte in Richtung der hochgesteckten Zielsetzungen dar. Nicht zuletzt deshalb warnen Wooldridge und Jennings [1998] vor überzogenen Erwartungen bezüglich der agentenorientierten Programmierung (AOP) und verweisen auf den Forschungscharakter der meisten bestehenden Agentensysteme:

> *There are good arguments in favour of the view that agent technology will lead to improvements in the development of complex distributed software systems. But, as yet, these arguments are largely untested in practice. There is certainly no scientific evidence to support the claim that agents offer any advance in software development – the evidence to date is purely anecdotal. … agent technology is essentially immature and untested.*

2.1.2. Agentendefinitionen

Es gibt viele Versuche zu definieren, was genau ein Agent sein soll[2]. Diese unterscheiden sich allerdings sehr, je nachdem welche der oben genannten Sichtweisen auf den Agentenbegriff gewählt wird bzw. welche konkrete Realisierung eines Agentensystems ihr zugrundeliegt. Zudem sind viele Agentendefinitionen recht vage und intuitiv und orientieren sich oftmals weniger an dem technisch Möglichen als an Forschungszielen, deren Erreichen teilweise noch in weiter Ferne liegen kann. Deshalb sollen hier zunächst einige typische und weit akzeptierte Definitionen kritisch betrachtet werden, bevor eine eigenen Definition gegeben wird.

Oft wird bei der Definition von Agenten verlangt, daß diese sich mit Hilfe mentalistischer Ausdrücke wie „Wissen“ oder „Ziel“ beschreiben lassen [Wooldridge & Jennings 1995], oder dies wird sogar als einziges Kriterium verwendet:

> *An agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments.* [Shoham 1993]

Diese mentalistische Beschreibung ist aber nicht von objektivierbaren Kriterien abhängig, im Gegenteil, „Agenthood is in the mind of the programmer“, wie Shoham [1993] selbst betont. Und selbst wer diese Ansicht nicht teilt und die Zuschreibung mentaler Zustände von objektiven Kriterien abhängig machen möchte, steht vor dem Problem, daß es dafür keine allgemein akzeptierte Theorie gibt. In jedem Fall bleibt offen, wann und welchen Programmen mentale Zustände zugesprochen werden können (oder sollten), welche dies im einzelnen sind, sowie welchen Inhalt diese haben, so daß auch die Frage, ob etwas ein Agent ist oder nicht, weitgehend willkürlich zu entscheiden ist. Somit ist eine Definition, die explizit (oder sogar ausschließlich) auf mentale Zustände zurückgreift, beim gegenwärtigen Stand der Forschung nicht eben hilfreich, um Agenten zu definieren oder auch nur einen halbwegs klaren Agentenbegriff zu formulieren.

Ein ebenso ambitionierter und umfassender, aber leider auch ebenso interpretationsfähiger Ansatz versucht Agenten als autonom Handelnde zu definieren:

---

[2] Einen kommentierten Überblick über verschiedene Agentendefinitionen geben Franklin & Graesser [1996].

> *An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors. … A rational agent is one that does the right thing. … the right action is the one that will cause the agent to be most successful.* [Russell & Norwig 1995]

> *An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.* [Wooldridge 1999]

Natürlich ist damit jedes Programm ein Agent, sobald man Wahrnehmen mit Input und Handeln mit Output gleichsetzt und als Umgebung Bits und Bytes erlaubt (was für Softwareagenten allerdings unerläßlich ist). Und auch die zusätzliche Anforderung der Zielgerichtetheit oder Rationalität der Handlungen erlaubt in den seltensten Fällen eine klare Abgrenzung, da eigentlich jedes Programm einem bestimmten Zweck dient und somit ein entsprechendes „Ziel“ (hoffentlich) erfolgreich verfolgt und erfüllt. Jedenfalls hängt auch diese Definition vor allem von intuitiven Vorstellungen ab, ohne ein hinreichend objektives Kriterium liefern zu können.

Beide bisher diskutierten Agentendefinitionen orientieren sich bewußt weder an spezifischen Eigenheiten von Computersystemen, noch an konkreten Umsetzungen, sondern verwenden den Agentenbegriff zur Definition eines umfassenden Forschungsprogramms. Allerdings werden diese Definitionen dabei derartig intuitiv und vage, daß sie kaum zur Abgrenzung von Agentensystemen gegenüber anderen Computersystemen dienlich sind.

Eine andere weithin akzeptierte Definition versucht konkret, die Merkmale von Agenten als Computersystemen anzugeben. Wooldridge und Jennings [1995] wählen dabei die folgenden Eigenschaften als Konstituenten des Agentenbegriffs:

- *autonom*: Die Kontrolle über seine Aktivitäten liegt vor allem beim Agenten selbst und nicht in direkten Anweisungen von außen.
- *reaktiv*: Agenten nehmen Änderungen der Umgebung wahr und können angemessen und zeitgerecht auf diese reagieren.
- *proaktiv*: Agenten können zudem aus eigener Initiative heraus aktiv werden und sich zielgerichtet verhalten.
- *interaktiv*: Agenten können miteinander interagieren und ihr Verhalten aufeinander abstimmen, um eigene und gemeinsame Ziele zu erreichen.

Auch diese Eigenschaften sind noch recht intuitiv und lassen einigen Interpretationsspielraum. Zusammengenommen ergeben sie jedoch ein anwendbares Kriterium zur Beurteilung von Agentensystemen, insbesondere da diese Eigenschaften nicht absolut, sondern graduell sind, d.h. ein System erfüllt sie jeweils mehr oder weniger. Entsprechend wird auch das Konzept des Agenten graduell, d.h. ein gegebenes System entspricht um so mehr der Definition, je mehr es diese Eigenschaften besitzt. Es muß betont werden, daß diese Eigenschaften normative Kriterien zur Beurteilung

von Agentensystemen sind, daß aber nicht umgekehrt von der Bezeichnung Agent bereits auf diese Eigenschaften geschlossen werden kann.

Wesentlich mehr an der praktischen Realität orientiert sich die Definition der FIPA (Foundation for Intelligent Physical Agents), einer Vereinigung für Standardisierungen im Bereich der Softwareagenten:

> *An Agent is the fundamental actor on an AP [Agent Platform] which combines one or more service capabilities into a unified and integrated execution model that may include access to external software, human users and communications facilities.* [FIPA 2000: XC00023G]

Hier wird der Agent in erster Linie als gekapselte Einheit verstanden, die Dienste erbringt. Allerdings wird damit die Abgrenzung zu anderen Programmierparadigmen wie der OOP oder Middleware schwierig, denen ein ähnlicher Ansatz zugrundeliegt.

2.1.3. Definition des Begriffs Agent

Auch wenn die nachfolgende Definition mehrere der oben vorgestellten Agentenbegriffe in sich vereint, beansprucht sie keine Allgemeingültigkeit oder Übertragbarkeit auf andere Agentensysteme, sondern dient lediglich der Klärung des bei der Entwicklung von CASA zugrundegelegten Agentenbegriffs.

> *Ein Agent ist ein autonomer Spezialist, der innerhalb eines Systems mehrerer Agenten eine bestimmte Funktionalität bereitstellt oder eine bestimmte Aufgabe ausführt und dazu mit anderen Agenten vermittels einer formalen Kommunikationssprache interagieren kann.*

Zentral an dieser Definition ist die Fokussierung der einzelnen Agenten auf bestimmte gekapselte Funktionalitäten und Aufgaben, die deren Rolle innerhalb eines Multiagentensystems festlegen, in dem einander ergänzende Kompetenzen verteilt zusammenwirken. Die Autonomie der Agenten besteht einerseits in den losen, auf funktionale Rollen beschränkten Abhängigkeiten der Agenten untereinander, andererseits in der Eigenständigkeit bei der Erfüllung dieser Rolle, die eine direkte Kontrolle durch den menschlichen Benutzer oder andere Agenten nicht erfordert, aber auch nicht ausschließt. Für die Interaktion zwischen den Agenten wird eine formale Kommunikationssprache gefordert, die von der spezifischen Implementierung der Agenten abstrahiert, um die Interoperabilität möglichst offen zu halten. Die Interaktionen brauchen jedoch nicht auf die Agentenkommunikation beschränkt zu sein, sie können auch zusätzlich direkt oder indirekt über eine gemeinsame Umgebung stattfinden. Andere Arten von Interaktionspartnern wie Benutzer oder Datenbanken sind meistens auf bestimmte Agenten beschränkt und verwenden jeweils eine spezifische Interaktionsform, da sie anders als Agenten untereinander keine

gleichgestellten autonomen Partner darstellen[3]. Eine Beschränkung auf Softwareagenten ist nicht zwingend, auch wenn dies der übliche Fall ist.

Aus diesem Agentenbegriff ergibt sich bereits die Motivation für die drei zentralen Konzepte der CASA-Architektur. Die Wissensbasiertheit ermöglicht einen hohen Grad an Autonomie und Flexibilität im Verhalten der einzelnen Agenten. Die Komponentenbasiertheit führt durch ihren modularen Ansatz zu einer hohen Flexibilität und Skalierbarkeit beim Erstellen einzelner Agenten für spezifische Aufgaben. Und durch das Dienstkonzept wird eine Grundlage für eine flexible und dennoch verläßliche Interoperabilität zwischen verschiedenen Agenten innerhalb der Agentengesellschaft erreicht.

2.1.4. Eigenschaften von Agenten

Weiterhin lassen sich aus dieser Definition die vier oben genannten definierenden Eigenschaften für Agenten von Wooldridge und Jennings [1995] – autonom, reaktiv, proaktiv und interaktiv – ableiten, wenn auch eher als Anforderungen, denn als definierende Eigenschaften. Die Autonomie wird direkt gefordert, ergibt sich aber auch aus der Kapselung der Funktionalität zusammen mit den möglichst losen Abhängigkeiten zwischen Agenten. Reaktivität und Proaktivität dienen dem eigenständigen und zuverlässigen Erfüllen einer Aufgabe und die Interaktivität dem Anbieten und Nutzen der verteilten Kompetenzen.

Neben diesen definierenden Eigenschaften werden Agenten noch weitere typische Merkmale zugeschrieben, die meist als optional aufgefaßt werden (vgl. [Wooldridge & Jennings 1995], [Franklin & Graesser 1996]). Einige häufig genannte Eigenschaften sind:

- Ein Agent ist *mobil*, wenn er sich eigenständig durch ein Computernetzwerk bewegen kann.
- Ein Agent wird als *rational* bezeichnet, wenn sein Verhalten einem vorgegebenen Optimalitätskriterium entspricht (bzw. sich diesem annähert).
- Ein Agent ist *adaptiv* oder *lernfähig*, wenn er sein Verhalten an seinen Benutzer bzw. seine Umgebung anpassen kann.
- Agenten nennt man *kooperativ*, wenn sie zusammen gemeinsame Ziele verfolgen, und *kompetitiv* bei konkurrierenden Zielen.
- Ein Agent ist *aufrichtig*, wenn er nur solche Informationen anderen mitteilt, die er selbst für wahr hält.
- Agenten verhalten sich *wohlwollend*, wenn sie keine Aufträge von anderen Agenten für ausführbare Aufgaben ablehnen.

---

[3] Im Verhältnis Benutzer / Agent ist es i.a. der Benutzer, dem die Kontrolle zukommt, im Verhältnis Agent / nicht-Agenten-Software der Agent.

- Ein Agent ist *sozial*, wenn sich sein Handeln nach expliziten gesellschaftlichen Normen richten kann.

Da Agenten nicht nur per Definition verteilte Systeme sind, sondern oft auch real über verschiedene Rechner innerhalb eines Netzwerks verteilt operieren, kann es von Vorteil sein, wenn sie ihre physische Position im Netzwerk zur Laufzeit ändern können. Dazu bedienen sie sich einer eigens dafür geschaffenen Infrastruktur aus Plattformen, zwischen denen Agenten migrieren können. Diese dienen zudem als Laufzeitumgebung – zur Migration wird i.a. interpretierter Kode verwendet – und übernehmen die Übertragung des Agenten zu einer anderen Plattform. Durch die Mobilität sollen vor allem Effizienz und Flexibilität gesteigert werden [Harrison et al. 1995]. Beispielsweise kann ein mobiler Agent einen Interaktionspartner oder andere Ressourcen auf einem anderen Rechner aufsuchen und so die Kosten der Kommunikation über das Netzwerk verringern, falls die Menge der dann lokal ausgetauschten Informationen die Größe des migrierenden Agenten (hin und zurück zusammengenommen) übersteigt. Bei mobilen Endgeräten muß eine Verbindung zum Netz nur noch zur Migration des Agenten und nicht während seiner gesamten Existenzdauer bestehen, damit dieser mit Agenten oder Ressourcen im Netz interagieren kann.

Die Rationalität von Agenten bezieht sich auf ihre Zweckmäßigkeit. Die ihnen übertragenen Aufgaben sollen möglichst optimal durchgeführt werden, wofür der Agent aufgrund seiner autonomen Arbeitsweise selbst sorgen muß. Mit dem Begriff der *begrenzten Rationalität* [Russell & Subramanian 1995] wird die Beschränktheit der vorhandenen Ressourcen explizit mit in den Optimalitätsbegriff aufgenommen. Außerdem wird bei Multiagentensystemen unterschieden zwischen lokalen Optimalitätskriterien für einzelne Agenten und der globalen Optimalität des Gesamtsystems.

Lernfähige Agenten optimieren ihr Verhalten aufgrund einer Bewertung der Resultate ihrer bisherigen Handlungen. Diese Bewertung findet entweder anhand vorgegebener Optimalitätskriterien oder durch den Benutzer statt. Agenten mit differenzierten Benutzerinteraktionen sind adaptiv, wenn sie sich an das Benutzerverhalten anpassen können. Dieses modellieren sie anhand von Profilen, in denen persönliche Präferenzen, Interessen und Gewohnheiten des Benutzers enthalten sind.

Beim Verhältnis der Agenten untereinander sind kooperative und kompetitive Szenarien zu unterscheiden. Im ersten Fall arbeiten mehrere Agenten an einem gemeinsamen Ziel, während im zweiten Fall eigene und auch konkurrierende Ziele verfolgt werden. Konflikte zwischen Zielen können in beiden Fällen auftreten, werden aber u.U. durch verschiedene Strategien behandelt. Während kooperative Agenten i.a. sämtliche Aufgaben anderer Agenten übernehmen, werden bei kompetitiven Agenten meist Gegenleistungen verlangt.

Die Interaktionen zwischen Agenten können besonderen Anforderungen unterliegen. So ist es für viele – vor allem kooperative – Szenarien sinnvoll und hilfreich,

von allen Agenten zu verlangen, daß sie aufrichtig und wohlwollend sind, so daß bei stattfindenden Interaktionen nicht die Möglichkeit der Täuschung oder Verweigerung berücksichtigt zu werden braucht, was die Programmierung erheblich vereinfacht. Bei sozialen Agenten sind diese gesellschaftlichen Normen nicht fest vorprogrammiert und garantiert, sondern basieren auf Konventionen, die explizit eingehalten werden.

2.1.5. Abstrakte Agentenmodelle

Zur Klassifizierung von Agenten lassen sich abstrakte Agentenmodelle verwenden. Im folgenden werden kurz zwei Ansätze vorgestellt, die von dem Begriff des Agenten als eines autonom Handelnden ausgehen und dessen interne Prozesse modellieren, mit denen aus Wahrnehmungen Handlungen abgeleitet werden.

Das Agentenmodell von Wooldridge [1999] basiert auf dem von Genesereth und Nilsson [1987]. Die Umgebung des Agenten wird in diesem Modell als eine Folge separater Zustände aufgefaßt, wobei sich die Zustandsübergänge (u.U. nichtdeterministisch) aus den Handlungen des Agenten ergeben. Ein Agent ist umgekehrt eine Abbildung der Umgebung auf seine Handlungen. Die Welt ist somit ein permanentes Wechselspiel aus Handlungen des Agenten und den sich daraus ergebenden Veränderungen in der Umgebung.

In diesem Rahmen lassen sich Agenten auf drei Detailebenen modellieren. Im einfachsten Fall ist ein Agent eine einzige Abbildung der bisherigen Zustände der Umgebung auf die Handlungen, die dieser als nächstes ausführt. Ein Spezialfall hiervon sind rein reaktive Agenten, die lediglich den aktuellen Zustand der Umgebung berücksichtigen. Auf der nächsten Ebene wird die Wahrnehmungstätigkeit explizit in das Modell mit aufgenommen, so daß zunächst aus dem aktuellen Zustand der Umgebung eine interne Repräsentation gebildet wird. Erst aus dieser spezifischen Sicht des Agenten ergeben sich dann dessen Handlungen. Auf der detailliertesten Ebene wird schließlich noch zusätzlich der interne Zustand des Agenten explizit gemacht. Nunmehr wird aus der jeweils aktuellen Wahrnehmungsrepräsentation und dem momentanen internen Zustand ein neuer interner Zustand abgeleitet, und erst dieser aktualisierte Zustand dient als Grundlage des Handelns.

Auch in dem Agentenmodell von Russel und Norwig [1995] ist ein Agent über die Wahrnehmung der Umgebung und daraus abgeleiteten Handlungen zur Veränderung der Umgebung definiert. Hier werden vier Arten von Agenten aufgrund von funktionalen Kriterien unterschieden.

Bei Reflex-Agenten werden die Handlungen aufgrund von Bedingung/Aktion-Regeln direkt aus der momentanen Wahrnehmung abgeleitet. Besitzt ein Agent zudem einen internen Zustand, der durch die jeweilige Wahrnehmung aktualisiert wird, und werden dann erst aus diesem internen Zustand die Handlungen abgeleitet, so kann das Verhalten des Agenten auch bereits vergangene Zustände berücksichtigen. Ein zielgerichteter Agent verwendet Zielbeschreibungen und Wissen über

die Auswirkungen seiner Handlungen in der Umgebung, um sein Handeln auf das Erreichen dieser Zielzustände auszurichten. Berücksichtigt ein zielgerichteter Agent auch den Nutzen seiner Handlungen, so ist er nutzenorientiert.

Beide Ansätze sind jedoch sehr abstrakt und deshalb nur bedingt zur Analyse oder Klassifizierung von (natürlichen wie künstlichen) Agenten im Sinne von autonom Handelnden geeignet. Insofern eine interne Struktur und funktionale Modularisierung postuliert wird, ist diese nicht allgemeingültig. Beispielsweise werden hybride Ansätze nicht berücksichtigt, die reaktive und zielgerichtete Aspekte nicht einfach nur parallel zueinander besitzen, sondern sorgsam aufeinander abgestimmt kombinieren. Auch fehlen soziale Aspekte wie Kommunikation und Kooperation, wozu andere Agenten als ein ausgezeichneter Teil der Welt mit besonderen Interaktionsmöglichkeiten modelliert werden müssen.

2.2. Agentenarchitekturen

Die Architektur legt die Struktur eines Agenten und damit den während der Laufzeit i.a. unveränderlichen Anteil fest. Dabei wird zwischen zwei Aspekten unterschieden. Zum einen bestimmt die Architektur den internen Aufbau eines Agenten aus funktionalen Bestandteilen und die Kontrollstruktur zwischen diesen:

> *Its architecture characterises the agent by modularisation, i.e., the identification of functional roles, and by the (structured) interactions between these modules.* [Jung & Fischer 1998]

Zum anderen dient die Architektur der Verhaltenskontrolle, indem sie die Prinzipien zur Steuerung des Verhalten im Sinne des Einwirkens auf die Umgebung des Agenten festlegt:

> *Agent architectures … are … software architectures for decision making systems that are embedded in an environment.* [Wooldridge 1999]

2.2.1. Verhaltenskontrolle

Bezüglich der Verhaltenskontrolle[4] kann zwischen den folgenden Architekturtypen unterschieden werden (vgl. [Wooldridge & Jennings 1995]):

- Bei einem *reaktiven* Agenten werden seine Handlungen direkt durch dessen Wahrnehmungen ausgelöst.

[4] Im Zusammenhang mit der Verhaltenskontrolle ist es sinnvoll, sich auf die Sichtweise des Agenten als einem autonom Handelnden zu beschränken.

- Ein *deliberativer* Agent kann abhängig von seinem Wissen über seine Umgebung selbst seine Handlungen bestimmen, um seine Ziele zu erreichen.
- Ein *interaktiver* Agent kann auf einer inhaltlichen Ebene mit anderen Agenten interagieren.
- Eine *hybride* Agentenarchitektur kombiniert mehrere dieser Architekturtypen miteinander.

Bei reaktiven Agenten hängen sämtliche Handlungen direkt von der momentanen Wahrnehmung ab, auf einen zu berücksichtigenden internen Zustand wie eine Repräsentation der Welt wird häufig ganz verzichtet. Sie befinden sich ständig in einem sehr engen Wechselspiel zu ihrer Umgebung, was als Situiertheit bezeichnet wird. Komplexeres Verhalten ergibt sich aus der gegenseitigen Kontrolle verschiedener rein reaktiver Prozesse [Brooks 1986]. Reaktive Agenten eignen sich besonders für sehr dynamische Umgebungen, in denen kaum verläßliche Annahmen über zukünftige Weltzustände möglich sind, so daß das Verhalten permanent an die aktuelle Situation angepaßt werden muß. Es ist jedoch schwierig, auf diese Weise ein auf längerfristige, komplexe Aufgaben hin ausgerichtetes Verhalten zu realisieren.

Deliberative Agenten hingegen verwenden explizite Repräsentationen ihrer Umgebung und ihrer Fähigkeiten, um ihr Verhalten auf das Erreichen von Zielen hin zu steuern. Dazu werden aufwendige Inferenz- und Planungsmechanismen verwendet, um unter Annahme der Richtigkeit des Wissens und der erfolgreichen Ausführung von Handlungen logisch korrekte Schlußfolgerungen zu ziehen und Handlungssequenzen zum Erreichen von Zielen aufzustellen. Diese formal-logischen Verfahren setzen jedoch ein vollständiges Wissen von der Umgebung und ihrer zukünftigen Entwicklung voraus, damit die logisch korrekten Ableitungen auch in der Realität valide sind und bleiben. Gerade in komplexen, dynamischen Umgebungen führen jedoch unvollständiges initiales Wissen und unvorhergesehene Ereignisse leicht zu Abweichungen zwischen angenommenen und tatsächlichen Weltzuständen, so daß die logischen Ableitungen korrigiert oder sogar völlig neu vorgenommen werden müssen. Deliberative Agenten zeigen ein zielgerichtetes Verhalten und können auch komplexe Aufgaben lösen, je nach Domäne können jedoch Aufwand und Fehlerwahrscheinlichkeit sehr hoch werden.

Bei interaktiven Agenten machen Interaktionen mit anderen Agenten den wesentlichen Bestandteil ihres Verhaltens aus. Diese Interaktionen finden vor allem auch auf einer inhaltlichen Ebene des Informationsaustauschs statt. Neben der Kommunikation werden Organisationsstrukturen und Koordinationsmechanismen verwendet, um die Interaktionen zu steuern. Architekturen für rein interaktive Agenten dienen dem Erstellen von Multiagentensystemen, bei denen das Verhalten des Gesamtsystems im Vordergrund steht.

Da von Agenten üblicherweise reaktive, deliberative und interaktive Verhaltensweisen verlangt werden, werden in hybriden Architekturen die Ansätze mehrerer Architekturtypen so verknüpft, daß sie sich gegenseitig ergänzen, um auf diese Weise deren Vorzüge miteinander zu kombinieren [Müller 1996]. Beispielsweise soll

die Verbindung von reaktiven und deliberativen Mechanismen zielgerichtetes Verhalten flexibel und fehlertolerant auch in dynamischen Umgebungen ermöglichen. Die Integration der verschiedenen Ansätze ist jedoch nicht trivial, weil sie auf unterschiedlichen und zum Teil inkompatiblen Prinzipien beruhen. Sie geschieht entweder auf der theoretischen Ebene durch Kombination der Ansätze oder auf der Systemebene, indem auf den verschiedenen Ansätzen basierende Teilsysteme kombiniert und untereinander koordiniert werden.

2.2.2. Aufbau

Vom Aufbau her lassen sich folgende Architekturtypen unterscheiden:

- Eine *monolithische* Architektur besitzt keine explizite Strukturierung.
- Eine *modulare* Architektur gliedert den Agenten in einzelne funktionale Komponenten.
  - Bei einer *zentralistischen* Architektur geschieht die Koordinierung zwischen den Komponenten über ein besonderes Zentralmodul.
  - Bei einer *geschichteten* Architektur stellen diese Module jeweils eine separate Verarbeitungsebene dar.
- Eine *konnektionistische* Architektur besteht aus sehr vielen einfachen, gleichartigen Einheiten.

Anders als monolithische Architekturen nehmen modulare Architekturen (Abbildung 1, links) eine explizite Trennung zwischen ihren einzelnen Bestandteilen vor. Die Architektur bestimmt dabei nicht nur die funktionale Gliederung in einzelne Komponenten, sondern auch die Abhängigkeiten und Interaktionsmöglichkeiten zwischen diesen. Die Struktur der Architektur ist fest oder variabel, je nachdem ob die Menge der Komponenten fest vorgegeben oder offen ist.

Besitzt eine modulare Architektur eine zentrale koordinierende Komponente, so ist sie zentralistisch (Abbildung 1, rechts). Dieses Zentralmodul kann wie bei der Blackboard-Architektur [Hayes-Roth 1985] ein gemeinsam genutzter Wissensspei-

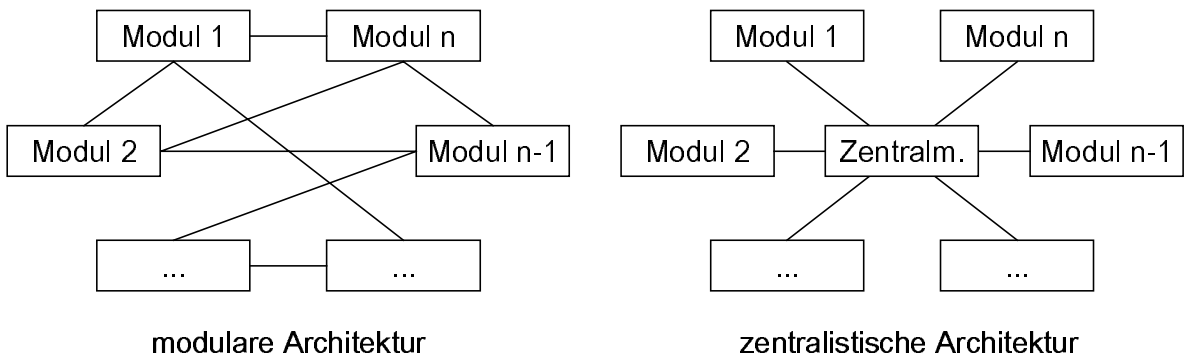

Abbildung 1: Arten von modularen Architekturen

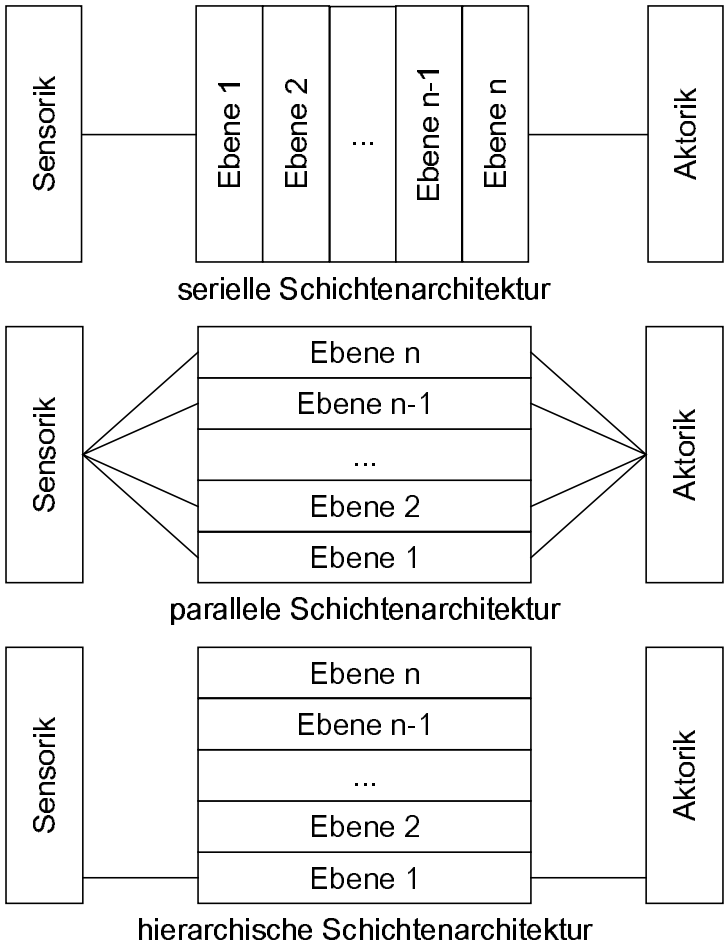

Abbildung 2: Arten von Ebenenarchitekturen

cher autonom arbeitender Module sein, oder es dient als zentrale Kontrolleinheit (Supervisor), die die anderen Komponenten überwacht und koordiniert.

Stellen die Module einzelne Verarbeitungsebenen dar, die jeweils nur mit den direkten Nachbarebenen Informationen austauschen, so handelt es sich um eine geschichtete Architektur. Bei der seriellen Schichtenarchitektur (Abbildung 2, oben), die auch als Kaskadenarchitektur bezeichnet wird, übernimmt jede Schicht jeweils einen Schritt innerhalb einer Verarbeitungssequenz. In einer parallelen Schichtenarchitektur (Abbildung 2, Mitte) arbeiten alle Ebenen parallel zueinander, weshalb ihre Ergebnisse zusätzlich miteinander koordiniert werden müssen. In der Subsumption-Architektur [Brooks 1986] geschieht diese Koordinierung, indem Ebenen die Ergebnisse anderer Ebenen überschreiben oder unterdrücken, in der Touring-Machines-Architektur [Ferguson 1992] durch zentrale Zensoren, die einer Ebene die Eingabe vorenthalten, und Suppressoren, die die Ausgabe einer Ebene unterdrükken. Bei der hierarchischen Schichtenarchitektur (Abbildung 2, unten) kontrolliert die unterste Ebene Sensorik und Aktorik des Agenten, während die höheren Ebenen die jeweils darunterliegende kontrollieren. Beispielsweise besitzt die Architektur von INTERRAP [Müller 1996] eine reaktive unterste Ebene und darüber eine lokale und eine kooperative Planungsebene.

Konnektionistische Architekturen bestehen aus Netzstrukturen vieler gleichartiger Einheiten, die sehr einfach aufgebaut und anders als bei modularen Architekturen auf keine bestimmte Aufgabe spezialisiert sind. Die Funktionalität des Systems ergibt sich dabei aus der komplexen Struktur der Verbindungen zwischen diesen Einheiten. Ein Beispiel hierfür sind neuronale Netze.

2.2.3. Komponententechnologie

Eine Form der strukturellen Gliederung von Softwarearchitekturen ist die Modularisierung durch Komponenten, die eine bestimmte Funktionalität kapseln. Bei der Konzeption von Agentenarchitekturen wird in der Regel beim Entwurf eine statische Einteilung in Komponenten auf einer abstrakten Ebene zur Festlegung der Kontrollstruktur vorgenommen, die jedoch vorwiegend der Strukturierung dient und zur Laufzeit unveränderlich ist.

Andere Programmierparadigmen bedienen sich der funktionalen Kapselung in Komponenten vor allem zur Wiederverwendung generischer Funktionalitäten, um so durch Redundanzvermeidung den Entwicklungsaufwand zu verringern. Dies beginnt bei Kodebibliotheken, deren Funktionalität statisch bei der Kompilierung oder dynamisch zur Laufzeit in ein Programm eingebunden werden kann, und setzt sich fort in der objektorientierten Programmierung, die die Kapselung von Funktionalität in Objekten als Kernparadigma besitzt. Neuere Ansätze beziehen auch die Verteiltheit von Komponenten über Netzwerke mit ein und gestalten den Zugriff auf Komponenten und deren Interaktionen untereinander offener und dynamischer. Aus dieser Perspektive heraus lassen sich auch Multiagentensysteme als eine Fortsetzung des Komponentenansatzes auffassen, da auch Agenten bestimmte Funktionalitäten kapseln und Agentensysteme diese in eine Gesamtfunktionalität integrieren.

Gemeinsam ist allen Komponentenansätzen die funktionale Kapselung und die Bereitstellung der Funktionalität einer Komponente und deren Nutzung durch andere Komponenten über eine wohldefinierte Schnittstelle, die die Interoperabilität zwischen den Komponenten durch eine Konvention oder Standardisierung gewährleistet. Diese Schnittstellen können spezifisch durch die einzelnen Komponenten selbst definiert sein oder generisch für alle Komponenten gelten, wobei in der Regel eine Kombination aus spezifischen und generischen Anteilen benötigt wird, um einerseits eine gemeinsame Basis für Komponenteninteraktionen zu besitzen und andererseits spezifische Funktionalitäten durch die einzelnen Komponenten anbieten zu können. Nachfolgend werden einige Komponententechnologien kurz vorgestellt.

Sun JavaBeans

Die JavaBeans-Spezifikation [Hamilton 1997] definiert ein einfaches Komponentenmodell, das die bereits in Java enthaltenen objektorientierten Mechanismen um einige Konventionen erweitert, die das Zusammenfügen von den – im Einklang mit der Java-typischen Kaffee-Terminologie als Beans bezeichneten – Komponenten erleichtern sollen. Ziel ist es, aus diesen Beans durch graphische Editoren ohne konventionelle Programmierung neue Anwendungen erstellen zu können, indem deren standardisierte Schnittstellen miteinander verbunden werden.

Eine Bean ist eine Instanz einer Klasse, die über eine Menge von Eigenschaften (Properties) konfigurierbar ist und Veränderungen über Ereignisse (Events) anderen Komponenten mitteilt. Für eine Eigenschaft wird eine Methode zum Abruf von deren Wert und optional eine Methode zum Setzen eines neuen Werts deklariert. Ein Ereignis besteht aus der Instanz einer eigens dafür definierten Klasse und wird über einen Methodenaufruf an einem zugehörigen Interface mitgeteilt. Komponenten, die dieses Interface implementieren, können über vorgegebene Methoden sich für das Ereignis an- und wieder abmelden. Alle diese Methoden lassen sich per Namenskonvention erkennen. Zudem kann an einer Komponente auch jede übrige öffentliche Methode aufgerufen werden.

Sun Enterprise JavaBeans

Die Spezifikation für Enterprise JavaBeans (EJB) [DeMichiel et al. 2000] beschreibt ein Komponentenmodell, das speziell auf Geschäftsprozesse in Unternehmen ausgerichtet ist.

Kernstück sind die Container, die auf der Server-Seite einzelne EJB erzeugen und verwalten. Dabei wird unterschieden zwischen Session Beans, die nur für die Dauer einer Client / Server-Session existieren, und Entity Beans, die persistent sind. Ein EJB-Container bietet zahlreiche Infrastrukturfunktionalitäten, wozu eine automatische Transaktionskontrolle, die Steuerung der Lastverteilung und die Absicherung der Komponenten untereinander wie gegenüber unzulässigen Zugriffen gehört.

Ein Server kann mehrere EJB-Container umfassen. Ein Client kann über einen Verzeichnisdienst eine dort registrierte EJB finden und über den zugehörigen Container eine Instanz erzeugen lassen. Daraufhin erhält der Client ein Remote Interface zu einem Stellvertreter-Objekt (EJB Object) der EJB, über das er via Remote Method Invocation (RMI) auf diese zugreifen kann. Das Stellvertreter-Objekt wird verwendet, um den Zugriff des Client auf die EJB durch den Container kontrollieren zu können.

Microsoft COM / DCOM

Das Komponentensystem Component Object Model (COM) [COM] von Microsoft erlaubt beliebige Implementierungssprachen für Komponenten. Lediglich eine

Schnittstellenspezifikation vermittels der Microsoft Interface Description Language (MIDL) unter Verwendung üblicher Basistypen ist notwendig. Eine Komponente besitzt mehrere Schnittstellen, von denen eine obligatorisch ist, über die eine Liste der übrigen angebotenen Schnittstellen abgerufen werden kann.

Identifiziert werden Komponenten wie Schnittstellen über eine global eindeutige ID (CLSID), über die sie an einer zentralen Stelle – der Registrierung eines Microsoft Windows-Betriebssystems – angemeldet werden. Andere Komponenten können über diese CLSID nach einer Komponente oder Schnittstelle suchen und über Methodenaufrufe gemäß der Schnittstelle auf deren Funktionalität zugreifen. Das System leitet einen derartigen Aufruf transparent an die Komponente weiter und übermittelt schließlich dessen Ergebnis zurück.

Das Distributed Component Object Model (DCOM) erweitert COM um Verteiltheit, so daß die angesprochene Komponente nicht auf demselben System verfügbar zu sein braucht, sondern auch über ein Netzwerk angesprochen werden kann.

OMG CORBA

Die Common Object Request Broker Architecture (CORBA) [CORBA] ist eine Spezifikation der Object Management Group (OMG) zur Interoperabilität von Objekten unabhängig von Ausführungsort und Implementierung. Die Interaktion zwischen Objekten beruht auf Methodenaufrufen, bei denen der Aufruf und die Übergabe von Parametern und Ergebniswert transparent durch CORBA weitergeleitet werden.

Aufrufende und ausführende Objekte können wie bei lokalen Methodenaufrufen implementiert werden, da diese direkt mit einem lokalen Stellvertreterobjekt (Stub bzw. Skeleton) interagieren, dessen Schnittstelle in der programmiersprachenunabhängigen Interface Definition Language (IDL) definiert wird. Die Verbindung zwischen Stub auf der Client-Seite und Skeleton auf der Server-Seite geschieht dann über den Object Request Broker (ORB). Dieser lokalisiert das Server-Objekt über ein zentrales Interface Repository, bei dem die Server-Schnittstellen registriert werden, und überträgt die Daten. Neben zur Designzeit festgelegten Methodenaufrufen sind auch dynamische Zugriffe zur Laufzeit und asynchrone Aufrufe über Nachrichten möglich.

In der aktuellen Spezifikation CORBA3 wird das CORBA Component Model (CCM) eingeführt. Dieses Modell orientiert sich an der EJB-Spezifikation. Die Übereinstimmung geht so weit, daß in Java implementierte CORBA-Komponenten als EJB verwendet werden können und umgekehrt.

2.3. Der Einzelagent

Die Ebene des einzelnen Agenten befaßt sich mit der autonomen Steuerung von dessen Verhalten. Da Agenten selbständig komplexe Aufgaben bearbeiten sollen, basiert ihre Verhaltenssteuerung in der Regel auf allgemeinen Kontrollprinzipien, die durch Architekturen realisiert sind (vgl. Kapitel 2.2.1). Dies erlaubt einerseits die Spezifikation des Verhaltens auf einer hohen Abstraktionsebene zur Designzeit und andererseits eine autonome und flexible Kontrolle zur Laufzeit.

Mit dem Aufbau von natürlichen und künstlichen Systemen mit einem autonomen, komplexen Verhalten beschäftigen sich verschiedene Wissenschaftszweige, die gemeinsam auch als Kognitionswissenschaften [Posner 1989] bezeichnet werden. Das verbindende Konzept dieser Forschungsbereiche ist das der Intelligenz, auch wenn für dieses keine einheitliche, allgemein akzeptierte exakte Definition existiert. Weit verbreitet sind Auffassungen, nach denen Intelligenz in erfolgreichem Verhalten besteht gemäß einem jeweils im Einzelfall genauer zu bestimmenden (und meist intuitiv verbleibenden) Rationalitätskriterium. Den kognitiven Apparat von Menschen und anderen Lebewesen untersuchen Psychologen und Gehirnforscher. Normative Aspekte der Intelligenz sind Gegenstand verschiedener Teilbereiche der Philosophie wie der Logik und der Erkenntnistheorie sowie der Ökonomie. Mit der Erschaffung künstlicher intelligenter Systeme beschäftigen sich die Kybernetik und vor allem der als Künstliche Intelligenz (KI) bezeichnete Teilbereich der Informatik.

Ansätze zur Künstlichen Intelligenz

In der KI existieren zwei unterschiedliche Grundrichtungen: die an formaler Logik ausgerichtete symbolische KI und der von Gehirnmodellen ausgehende Konnektionismus. Auch wenn beide Ansätze nicht prinzipiell miteinander unvereinbar sind, werden sie doch weitgehend unabhängig voneinander erforscht und angewendet. Nur wenige Systeme kombinieren symbolische und konnektionistische Verfahren, wobei diese jedoch meistens nur in getrennten Subsystemen ohne konzeptionelle Integration eingesetzt werden.

Die Grundidee der symbolischen KI ist die Symbolmanipulation gemäß einer formalen Logik. Symbole stehen dabei für reale und abstrakte Gegenstände und Beziehungen, indem sie jeweils durch ein beliebiges eindeutiges Zeichen innerhalb eines Zeichensystems repräsentiert werden. Aus den Symbolen werden nach einer vorgegebenen Syntax zusammengesetzte Ausdrücke gebildet und diese dadurch in Bezug zueinander gesetzt. Die Bedeutung der Ausdrücke ist über eine formale Semantik definiert, nach der sie aus der Struktur eines Ausdrucks und der Bedeutung der einzelnen Symbole abgeleitet wird. Die Symbolmanipulation besteht nun in rein syntaktischen Vergleichen und Umformungen solcher symbolischen Ausdrücke, die der in der Semantik enthaltenen Bedeutung gerecht werden. Der Computer kann auf diese Weise auf einer bedeutungshaltigen Ebene operieren. Einige Vertreter

dieser Richtung fassen dies bereits als die Essenz von Intelligenz auf [Fodor 1975, Newell & Simon 1976]. Kritiker wenden ein, daß die Bedeutung der Symbole rein beim menschlichen Betrachter liegt [Dreyfus 1972, Winograd & Flores 1986, Searle 1993]. Die Frage, ob es sich bei Symbolen um eine Repräsentation als Wissen des Computers oder eine Repräsentation von Wissen des Menschen handelt, ist jedoch für die Funktionsweise derartiger Systeme letztlich unerheblich.

Die dem Gehirn nachempfundenen neuronalen Netze bestehen aus vielen gleichartigen, als Neuronen bezeichneten Einheiten. Anders als bei symbolischen Systemen beinhalten sie keine klar abgegrenzten Repräsentationen, sondern diese sind verteilt und einander überlagernd in den Verbindungen zwischen den Neuronen enthalten. Informationsverarbeitung besteht in der Propagierung von Aktivität durch das Netz, wobei einzelne Neuronen Aktivität über die gewichteten Verbindungen erhalten und bei Überschreiten eines Schwellwerts ihrerseits weiterreichen. Die Programmierung erfolgt nicht direkt, sondern durch das Lernen der Wichtungen anhand von Beispielen.

Für die Spezifikation und Kontrolle des Verhaltens autonomer Agenten überwiegen die Vorteile des Symbolansatzes im Vergleich zum Konnektionismus. Insbesondere bei umfassenden und verteilten Systemen ist es wichtig, das Verhalten möglichst klar und explizit spezifizieren zu können, um deren Komplexität beherrschbar zu machen. Dazu ist die Abstraktionsebene der formalen Logik des Symbolansatzes besonders geeignet. Deshalb werden im folgenden nur Techniken der Symbolmanipulation näher betrachtet.

Techniken der Symbolmanipulation

Die symbolische KI umfaßt unterschiedliche Gebiete, die verschiedene Aspekte der Manipulation von Symbolen nach formallogischen Prinzipien abdecken. Als Grundlage der Symbolmanipulation dient das durch symbolische Ausdrücke repräsentierte Wissen. Der Bereich der Wissensrepräsentation beschäftigt sich deshalb mit Formalismen zur logischen Darstellung von Wissen, die für bestimmte Gegenstandsbereiche adäquat sind und für die berechenbare und wenn möglich effiziente Verfahren für grundlegende logische Operationen wie Unifikation und Inferenz benötigt werden. Um dieses Wissen für eine bestimmte Aufgabe zu verwenden, bauen darauf Strategien zur Verhaltenssteuerung auf, die unter Nutzung des logischen Gehalts aus dem repräsentierten Wissen zweckgerichtete Handlungen ableiten. Durch Lernen können das Wissen und die Fähigkeiten aufgrund von Erfahrung erweitert und optimiert werden.

Ein weiterer Aspekt ist die Schnittstelle zwischen der formallogischen Ebene des Systems und dessen Umgebung. Ist dies ein menschlicher Benutzer, so ist das Interaktionsmedium eine graphische Oberfläche oder die natürliche Sprache, bestenfalls Experten können direkt auf der formallogischen Ebene mit dem System interagieren. Bei einem autonomen System wie einem Roboter wandelt die Wahrnehmung sensorische Daten in Symbole um und die Aktorik setzt Handlungsabsich-

ten in motorische Bewegungen um. Da diese Schnittstellen stark von der jeweiligen Umgebung abhängen, gibt es in diesem Bereich nur wenige allgemeingültige Ansätze. Eine Ausnahme stellen Softwareagenten dar, die über Kommunikation direkt miteinander interagieren können, ohne die Ebene ihrer Wissensrepräsentation zu verlassen (s. Kapitel 2.4.2). Integriert zu einem Gesamtsystem werden diese unterschiedlichen Bereiche durch eine Kontrollarchitektur, die dessen Struktur und die internen Abläufe festlegt (s. Kapitel 2.2.1).

2.3.1. Wissensrepräsentation und Inferenz

Der Bereich der Wissensrepräsentation und Inferenz beschäftigt sich mit der Darstellung von formalem Wissen und dessen Manipulation durch logische Operationen. Er hat eine lange Tradition in der KI, ist gut erforscht und wird in zahlreichen Anwendungssystemen umgesetzt [Bibel 1993].

Grundlage einer Wissensrepräsentation bildet eine formale Logik, die die Bedeutung des repräsentierten Wissens und damit die zulässigen logischen Operationen festlegt. Dabei gibt es verschiedene Arten von formalen Logiken, die sich hinsichtlich ihrer Ausdrucksstärke und ihres Anwendungsbereichs unterscheiden. Neben der Objektebene spielt auch die terminologische Ebene eine Rolle, die das mögliche Wissen vorstrukturiert. Schließlich sind bei der Verwaltung von Wissen in einer Wissensbasis auch nicht-logische Aspekte zu berücksichtigen.

Formale Logik

Eine formale Logik besteht aus einer Syntax und einer Semantik. Die Syntax legt ausgehend von einem vorgegebenen Zeichensatz fest, welche Zeichenfolgen zulässige Ausdrücke sind. An Zeichen gibt es dabei einerseits die der Logik, die Verknüpfungen ausdrücken oder der Gliederung von Aussagen dienen, sowie die der Signatur, deren Bedeutung nicht durch die Semantik, sondern durch eine sogenannte Interpretation angegeben wird.

Die Semantik bestimmt die Bedeutung sämtlicher dieser zulässigen Ausdrücke. Dabei wird in der Regel zwischen Termen und Aussagen unterschieden. Gemäß der Modelltheorie besteht die Bedeutung eines Terms in der Referenzierung auf ein (konkretes oder abstraktes) Objekt des Gegenstandsbereichs, wobei die Gesamtheit der Referenzen als Modell bezeichnet wird. Während einzelne Symbole direkt für ein Objekt stehen, ergibt sich das referenzierte Objekt eines zusammengesetzten Ausdrucks aus der Semantik. In einer typisierten Logik sind Objekte zudem in Wertebereiche eingeteilt.

Aussagen hingegen besitzen einen Wahrheitswert, der das Gegebensein des durch die Aussage beschriebenen Zustands in einem bestimmten Modell beinhaltet. Eine Aussage ist allgemeingültig, wenn sie in allen möglichen Modellen einen bestimmten ausgezeichneten Wahrheitswert besitzt, und widersprüchlich, wenn sie

in keinem Modell diesen Wahrheitswert besitzen kann. Zwei Aussagen sind äquivalent, wenn sie in jedem möglichen Modell denselben Wahrheitswert besitzen.

Syntaktische Operationen

Der eigentliche Sinn von formalen Logiken liegt in der Möglichkeit, auf einer rein syntaktischen Ebene mit Aussagen auf eine Weise zu operieren, die der Semantik der Logik gerecht wird. So kann nach rein formalen und damit bedeutungsblinden Verfahren in einer korrekten Weise mit bedeutungshaltigen Ausdrücken umgegangen werden.

Die wichtigste Operation ist das logische Schließen – auch als Inferenz bezeichnet –, bei dem nach vorgegebenen Regeln aus bekanntem Wissen neues abgeleitet wird. Ein Beweiskalkül ist ein Verfahren zum Durchführen von Inferenzen. Es ist korrekt, wenn dabei die Semantik nicht verletzt wird, und vollständig, wenn damit sämtliche möglichen Schlußfolgerungen vollziehbar sind. Nicht für alle Logiken braucht ein korrekter Kalkül zu existieren, der auch vollständig ist. Zudem kann das Erstellen eines Beweises sehr aufwendig sein.

Als Deduktion werden Schlußfolgerungsverfahren bezeichnet, die garantiert korrekt sind, so daß aus richtigen Annahmen keine falschen Schlüsse gezogen werden können. Die Induktion ist eine Verallgemeinerung, bei der aus einer Menge von positiven Einzelfällen und dem Fehlen negativer Beispiele eine allgemeine Regel abgeleitet wird. Bei der Abduktion wird eine konkrete Hypothese aufgestellt, die eine mögliche Erklärung für einen bestehenden Fall gibt, so daß dieser aus der Hypothese hergeleitet werden kann. Es gibt jedoch keine Verfahren zur Induktion oder Abduktion, die ausschließlich korrekte Schlüsse erlauben.

Neben diesen allgemeinen Schlußfolgerungen gibt es auch solche, die Wissen über einen spezifischen Bereich enthalten und als allgemeine Aussagen formuliert werden. Definitionen legen Wahrheit per Konvention fest. Empirische Regeln werden als allgemeingültig angesehen, beruhen aber auf unsicheren Schlußmethoden. Heuristiken sind Regeln, die zwar in der Mehrzahl aller Fälle zutreffen, von denen aber Ausnahmen bekannt sind.

Neben den verschiedenen Arten der Inferenz ist auch die Unifikation eine wichtige syntaktische Operation. Dabei wird die logische Äquivalenz zweier Ausdrücke überprüft.

Verschiedene formale Logiken

Eine der einfachsten Logiken ist die Aussagenlogik, in der die denotierenden Symbole bereits für ganze Aussagen stehen, die entweder wahr oder falsch sein können. Der Gegenstandsbereich ist also die Menge der Wahrheitswerte. An Verknüpfungen gibt es die Negation, die den Wahrheitswert invertiert, und die Konjunktion, die eine Und-Verknüpfung beinhaltet. Andere Verknüpfungen wie die Disjunktion und die Implikation lassen sich über diese definieren.

In der Prädikatenlogik referenzieren die Symbole der Signatur Objekte, Funktionen und Relationen des Gegenstandsbereichs. Zusätzlich zu den Verknüpfungen der Aussagenlogik gibt es noch Quantoren, die allgemeine Aussagen erlauben, in denen statt denotierender Symbole Variablen als Platzhalter enthalten sind. In der Prädikatenlogik erster Stufe sind Quantoren und Relationen auf Objekte beschränkt, in höherstufigen Varianten können diese sich auch auf Funktionen und Relationen beziehen. Die folgenden Logiken stellen jeweils unterschiedliche Erweiterungen der Prädikatenlogik erster Stufe dar, ähnlich wie diese die Aussagenlogik erweitert.

In einer Modallogik lassen sich Modaloperatoren auf eine Aussage anwenden, um eine Einstellung gegenüber dieser auszudrücken. Die Semantik dieser Operatoren wird über Erreichbarkeitsrelationen zwischen möglichen Welten, die jeweils eine Interpretation darstellen, definiert [Kripke 1963]. Eine Aussage innerhalb eines Modaloperators ist dabei in einer bestimmten Welt wahr, wenn sie in einer bzw. in allen von dieser Welt aus gemäß dem Operator erreichbaren Welten wahr ist. Die erste Modallogik unterschied notwendige und mögliche Aussagen, wobei sämtliche Welten untereinander erreichbar sind, so daß eine Aussage notwendig wahr ist, wenn sie in jeder Welt wahr ist, und möglich ist, wenn sie in mindestens einer Welt wahr ist. Die epistemische Logik verwendet Operatoren für Wissen und Glauben, um zwischen objektivem Wissen und subjektivem Glauben zu unterscheiden. Bei der deontischen Logik werden ethische Normen über die Modalitäten „vorgeschrieben“ und „erlaubt“ beschrieben. Die temporale Logik ordnet einer Aussage einen zeitlichen Geltungsbereich zu, wozu sie „für immer“ und „irgendwann einmal“ als Modaloperatoren einführt. Die BDI-Logik (Belief, Desire, Intention) baut auf der temporalen Logik auf, um das Verhalten von Agenten zu beschreiben [Bratman 1987, Cohen & Levesque 1990]. Dazu werden zusätzliche Modaloperatoren für die Annahmen (Belief), Zielsetzungen (Desire) und Handlungsabsichten (Intention) eines Agenten eingeführt. Um Wissen über andere Agenten zu repräsentieren, werden diese Modaloperatoren auch jeweils einzelnen Agenten zugeordnet.

Die bisher genannten Logiken sind alle monoton, d.h. zusätzliches Wissen hat keinen Einfluß auf den Wahrheitswert einmal vorgenommener deduktiver Schlüsse. Um vorläufige, hypothetische Annahmen zu berücksichtigen, erlaubt die Default-Logik [Reiter 1980] allgemeine Regeln, deren Anwendung durch konkretes Wissen verhindert werden kann, wodurch sich u.a. Regeln mit Ausnahmen modellieren lassen. Dadurch wird die Logik nicht-monoton, da neu hinzukommendes Wissen die Anwendung einer derartigen Regel im nachhinein unzulässig machen kann, so daß alle darauf aufbauenden Schlüsse zurückgenommen werden müssen.

Mehrwertige Logiken verwenden neben den klassischen Wahrheitswerten *wahr* und *falsch* noch weitere Wahrheitswerte [Urquhart 1986]. Neben generellen Ansätzen mit einer beliebigen Anzahl von Wahrheitswerten gibt es dabei auch Logiken, die zusätzliche Wahrheitswerte mit einer bestimmten Bedeutung einführen, beispielsweise für undefinierte oder widersprüchliche Ausdrücke.

Ein Sonderfall der mehrwertigen Logiken sind die unendlichwertigen, bei denen zumeist das Intervall der realen Zahlen von Null bis Eins als Menge der Wahrheitswerte verwendet wird. Ein Beispiel hierfür ist die Fuzzy-Logik [Zadeh 1965, 1979], mit der unscharfes Wissen ausgedrückt wird, bei dem es neben eindeutig wahren und falschen Fällen auch graduelle Übergangsfälle gibt. Unsicheres Wissen läßt sich mit Hilfe der Wahrscheinlichkeitstheorie ausdrücken, indem die Gewißheit als Wahrscheinlichkeit der Übereinstimmung zwischen Annahmen und Tatsachen aufgefaßt wird. Die Wahrscheinlichkeitstheorie kann dabei als eine Form von unendlichwertiger Logik angesehen werden.

Ontologien

Formale Logiken sind insofern domänenunabhängig, als daß sie keinen expliziten Bezug zu einem bestimmten Gegenstandsbereich besitzen. Allerdings setzen sie einige implizite Annahmen voraus, was besonders bei den Modaloperatoren deutlich wird, aber auch die Einteilung der Prädikatenlogik in klar voneinander trennbare Objekte, Funktionen und Relationen ist nicht in jedem Fall gegeben.

Zur Bereichsmodellierung werden Ontologien verwendet, mit denen das Wissen einer Domäne vorstrukturiert wird. Dazu definieren diese ein Vokabular an Symbolen mit einer vorgegebenen Bedeutung sowie Beziehungen zwischen diesen.

Objekte werden dabei in Klassen mit einer gemeinsamen Struktur – auch als Kategorien oder Konzepte bezeichnet – eingeteilt, relativ zu denen mögliche Beziehungen und Eigenschaften vorgegeben werden. Die Klassen stehen ihrerseits untereinander in Beziehung, beispielsweise der Teil/Ganzes-Relation oder als Ober- bzw. Untermenge, wodurch sich gegebenenfalls die Klassenzugehörigkeiten und die damit verbundenen Definitionen vererben.

Die in Ontologien definierten Klassen und Beziehungen lassen sich logisch modellieren, so daß auch über die Terminologie als solche Schlußfolgerungen möglich sind. Dies gilt auch für vom Ursprung her nicht-formallogische Ansätze zur Kategorisierung von Gegenstandsbereichen wie Frames [Minsky 1975] und semantische Netze [Quillian 1968], insoweit diese keine prozeduralen oder zeitbezogenen Elemente beinhalten. Eine derartige terminologische Logik hat als Gegenstandsbereich die Kategorisierung eines anderen Gegenstandsbereichs.

Wissensbasis

Die technische Umsetzung der Wissensrepräsentation geschieht durch die Verwaltung von Wissen in einer Wissensbasis. Dazu werden effiziente Methoden zur Speicherung von Wissen und für den Zugriff auf dieses benötigt. Außerdem sollte das enthaltene Wissen immer konsistent sein sowie möglichst korrekt und so vollständig wie nötig den tatsächlichen Zustand der Welt widerspiegeln.

Da die Inferenzmechanismen bei sämtlichen Logiken eine statische Welt voraussetzen, müssen bei Änderungen der Annahmen über die Welt – bei einer nicht-

monotonen Logik sogar bei bloß hinzukommenden Annahmen – im Prinzip sämtliche Schlußfolgerungen daraufhin überprüft werden, ob sie weiterhin gültig sind[5]. Zudem kann neues Wissen im direkten Widerspruch zu den bestehenden Annahmen stehen, oder Inkonsistenzen ergeben sich indirekt relativ zu vorhandenen Schlußregeln. Deshalb muß das Wissen nicht nur gespeichert, sondern aktiv gewartet werden, um Inkonsistenzen und ungültige Schlüsse zu erkennen und gegebenenfalls zu beheben.

Eine einfache Methode zur Beseitigung von direkten Widersprüchen ist beispielsweise die Faustregel, daß aktuellere Informationen verläßlicher sind, da sich die Welt mit der Zeit ändert. Andere Inkonsistenzen lassen sich auflösen, indem Annahmen eine Gewißheit zugeordnet ist, so daß weniger sichere Annahmen bei einem Widerspruch aufgegeben werden. In diese Gewißheit können viele verschiedene Faktoren mit einbezogen werden wie die Aktualität, die Rechtfertigung durch andere Annahmen oder auch die Beständigkeit von Tatsachen eines bestimmten Typs zusammen mit der Zeit der letzten Bestätigung.

Mit einem Truth Maintenance System werden die (bekannten) logischen Abhängigkeiten zwischen verschiedenen Annahmen explizit verwaltet, so daß diese bei Änderungen leichter zu berücksichtigen sind. Zudem dienen sie als Rechtfertigung und können somit als Erklärung von Annahmen dienen.

2.3.2. Verhaltenssteuerung

Die Steuerung des Verhaltens eines Agenten baut nun auf dem repräsentierten Wissen auf. Zum einen lassen sich direkt aus dem Wissen Handlungen ableiten, wobei der Agent auf die jeweilige Situation reagiert. Derartiges Verhalten wird durch regelbasierte Systeme realisiert. Zum anderen kann das Verhalten zusätzlich auf bestimmte Aufgaben hin ausgerichtet sein, so daß der Agent zielgerichtet handelt. Dazu werden Probleme gelöst oder Handlungsabläufe geplant. In jedem Fall müssen Entscheidungen getroffen werden, sobald mehrere Alternativen zur Auswahl stehen. Bei Agenten werden oft Ansätze zur Verhaltenssteuerung verwendet, die auf der BDI-Logik (s.o.) basieren.

Regelbasierte Systeme

Ein regelbasiertes System verwendet logische Schlußfolgerungen zur Verhaltenskontrolle. Es besteht im wesentlichen aus einer Wissensbasis, einer Regelmenge und einem Inferenzmechanismus. Der Inferenzmechanismus ändert dabei die Wissens-

[5] Eine Ausnahme bilden Logiken, die Zeit explizit repräsentieren, beispielsweise indem jeder Aussage der Zeitpunkt oder ein Intervall der Gültigkeit zugeordnet ist. Bei diesen ergibt sich jedoch ein ähnliches Problem, da auch hier das Fortbestehen von Annahmen und Schlußfolgerungen explizit berücksichtigt werden muß.

basis gemäß den Regeln, deren Anwendbarkeit sich ihrerseits aus der Wissensbasis ergibt.

Streng logisch gehen dabei Theorembeweiser vor, die als Inferenzregeln nur die der Deduktion verwenden, so daß sie ausschließlich logisch korrekte Ableitungen vornehmen. Bereichsspezifische Regeln sind als allgemeine Aussagen Teil der Wissensbasis. Theorembeweiser verwenden einen logischen Kalkül, um vorgegebene Behauptungen aus den initialen Axiomen abzuleiten oder wenn möglich zu widerlegen. Je nach der zugrundeliegenden Logik und der Komplexität der Annahmen und der Behauptung kann die Durchführung eines Beweises jedoch sehr aufwendig werden. Bei nicht-entscheidbaren Logiken kann zudem nicht garantiert werden, daß nach endlicher Zeit die Behauptung bewiesen oder widerlegt ist.

Logische Programmiersprachen wie Prolog [Colmerauer 1985] verwenden deshalb eine vereinfachte Logik, um effiziente Beweisverfahren realisieren zu können. Zusätzlich integrieren sie nicht-logische Aspekte, die eine einfachere Programmierung und Ablaufkontrolle ermöglichen.

Produktionensysteme wie OPS-5 unterscheiden getrennte Speicher für Fakten und Regeln. Die als Produktionen bezeichneten Regeln bestehen aus einer Bedingung und einer Aktion. Der Inferenzmechanismus ist ein wiederholter Zyklus, bei dem zunächst bestimmt wird, bei welchen Regeln die Bedingung gemäß dem Faktenwissen erfüllt ist. Aus diesen aktiven Regeln wird im Zuge der Konfliktauflösung eine Regel ausgewählt und deren Aktionsteil ausgeführt. Der Aktionsteil beschreibt Änderungen der Fakten, so daß der Zyklus mit einer geänderten Faktenmenge erneut durchgeführt wird. Entscheidend für das Verhalten eines Produktionensystems ist daher die Strategie zur Auswahl einer Regel bei der Konfliktauflösung. Kriterien hierfür sind beispielsweise die Vermeidung von Wiederholungen, die Spezifität der Regeln oder statische Prioritäten.

Problemlöser

Computergestützte Problemlöser werden vor allem für klar spezifizierte Probleme verwendet, für die eine aufzählbare Menge an Lösungen existiert. In diesem Problemraum sucht der Problemlöser nach einer Lösung. Das erste derartige System war der General Problem Solver von Newell und Simon [1961], das als Simulation des menschlichen Denkens gedacht war.

Der erste Schritt zur Problemlösung ist die Repräsentation des Problems, die durch den Benutzer geschieht. Dieser legt den Problemraum fest, der aus verschiedenen Zuständen des Problembereichs besteht, die durch mögliche Lösungsschritte verbunden sind. Die Lösung selbst ist dann eine Folge solcher Schritte, mit denen beginnend beim Ausgangszustand ein Zielzustand erreicht wird.

Da dieser Problemraum sehr umfangreich sein kann, benötigt man Suchstrategien, die möglichst schnell eine oder die beste Lösung finden. Einfache Strategien sind die Breitensuche, die jeweils alle möglichen Lösungswege parallel weiterentwickelt,

und die Tiefensuche, die einen einzigen Lösungsweg verfolgt und bei einer Sackgasse an einer früheren Stelle eine andere Alternative versucht. Da die Breitensuche sehr ressourcenaufwendig ist und die Tiefensuche weder vollständig, noch optimal ist, werden beide miteinander kombiniert, wobei die Tiefensuche auf eine bestimmte Tiefe beschränkt wird, die sukzessive erhöht wird, so daß wie bei der Breitensuche alle möglichen Lösungswege ausprobiert werden.

Eine völlig blinde Suche wie diese ist jedoch sehr ineffektiv, da u.U. die Zahl der Zustände mit der Suchtiefe exponentiell wächst. Der Problemraum kann zusätzlich eingeschränkt oder vorstrukturiert werden durch allgemeine und bereichsspezifische Strategien. Beispielsweise lassen sich bei der Suche Redundanzen vermeiden, indem von einem bereits auf anderem Weg erreichten Zustand aus nicht erneut gesucht wird.

Gibt es eine Bewertungsfunktion einzelner Lösungsschritte, so kann jeweils der günstigste Schritt zuerst versucht werden. Gibt es eine Bewertung der Nähe einzelner Zustände zur Lösung, so kann aus den möglichen Lösungsschritten jeweils der zu dem der Lösung nächsten Zustand gewählt werden. Angewandt auf den gesamten Suchpfad kann auf diese Weise eine optimale Lösung gefunden werden, sofern alle möglichen Lösungsschritte positive Kosten besitzen. Oft sind die Bewertungsfunktionen jedoch nur Heuristiken, die eine ungefähre Abschätzung erlauben, so daß auch die Effizienz der Suche und die Optimalität der Lösung nur bedingt gelten.

Planen

Das Planen von Handlungen ist ein Sonderfall des Problemlösens, bei dem die Zustände und Zustandsübergänge des Problemraums formallogisch beschrieben werden. Im einfachsten Fall ist ein Zustand eine Situation und ein Übergang eine Situationsänderung. Die logische Repräsentation erlaubt eine Zerlegung von Problemen in Teilprobleme, die mehr oder weniger unabhängig voneinander behandelt werden können, was den Suchaufwand erheblich verringern kann. Zudem lassen sich die einzelnen Lösungsschritte sehr allgemein über Bedingung und Effekt beschreiben, die angeben, in welchen Situationen ein Schritt anwendbar ist und welche Veränderung er herbeiführt. Das erste derartige Planungssystem war STRIPS [Fikes & Nilsson 1971], das am SRI zur Steuerung des Roboters Shakey entwickelt wurde.

Im Situationen-Kalkül werden Situationen und Handlungen so modelliert, daß Zustandsübergänge durch logische Inferenz ausgedrückt werden. Planung läßt sich somit durch einen Theorembeweiser umsetzen, der den Zielzustand ausgehend von der Anfangssituation und den Handlungen als Axiomen herleitet. Die Beweiskette läßt sich dann auf einen Plan abbilden gemäß der Verwendung der Handlungsaxiome. Der Ereignis-Kalkül basiert statt auf zeitlich punktuellen Situationen auf Ereignissen mit einer Dauer.

Derartige logische Kalküle bieten eine gute formale Grundlage zur Analyse der Ausdrucksstärke eines Problembeschreibungsformalismus, sind aber in der praktischen Umsetzung nicht effizient. Zur Planung werden daher analog zum Problemlösen Suchalgorithmen verwendet, bei denen ein Plan als logische Verkettung von Handlungen erstellt wird, dessen erfolgreiche Ausführung den Ausgangszustand in den Zielzustand überführt. Auch hier dienen Heuristiken der Effizienzsteigerung. Beispielsweise verringert eine Rückwärtssuche beginnend beim Ziel i.a. die Anzahl der zu betrachtenden Alternativen und damit den Suchaufwand, da das Ziel in der Regel spezifischer ist als die Ausgangssituation.

In ihrer klassischen Form sind Planungsalgorithmen nur in bestimmten Domänen verwendbar, da sie auf recht rigiden Annahmen basieren. So ist ein erstellter Plan nur dann sicher ausführbar, wenn sich die Umgebung ausschließlich aufgrund der Handlungen des Planers verändert. Durch verschiedene Erweiterungen des Planformalismus wird versucht, derartige Beschränkungen zu verringern.

Durch eine hierarchische Planung (ABSTRIPS [Sacerdoti 1974]) läßt sich die Komplexität verringern, indem der Problemraum in verschiedene Abstraktionsebenen unterteilt wird. So kann ein zunächst grober Plan sukzessive verfeinert werden, wobei jeweils nur ein begrenzter Problemraum zu betrachten ist. Zudem erlaubt die hierarchische Planung eine gewisse Anpassung an die Dynamik der Umgebung, wenn nur der jeweils nächste abstrakte Schritt verfeinert wird, so daß der zeitliche Abstand zwischen Planung und Umsetzung auf der detaillierten Ebene gering gehalten wird. Verändert sich die Umgebung während der Planausführung auf eine vom Plan nicht vorgesehen Weise – sei es durch äußere Einflüsse oder eine fehlgeschlagene Aktion –, so muß der Plan gegebenenfalls an die neue Situation angepaßt werden, da notwendige Bedingungen nicht mehr erfüllt oder einzelne Aktionen überflüssig geworden sind.

Entscheiden

Unabhängig von der Art der Verhaltenskontrolle ergeben sich verschiedene Entscheidungspunkte, an denen eine Auswahl aus Alternativen zu treffen ist. Erschwert werden diese Entscheidungen dadurch, daß das Wissen über den gegenwärtigen Zustand der Welt und deren künftige Entwicklung unvollständig und nur bedingt verläßlich ist.

Die Wahrscheinlichkeitstheorie ermöglicht die Modellierung von unsicherem Wissen, indem Annahmen auf eine bestimmte Wahrscheinlichkeit des Zutreffens relativiert werden. Diese Wahrscheinlichkeit kann subjektiv als Grad der Gewißheit oder auch objektiv durch einen realen Indeterminismus gegeben sein. Die Wahrscheinlichkeitswerte einzelner Aussagen lassen sich logisch miteinander verknüpfen, anders als bei einer formalen Logik müssen dabei jedoch gegebenenfalls empirische Abhängigkeiten zwischen diesen berücksichtigt werden. Somit kann die Wahrscheinlichkeit ermittelt werden, daß bestimmte Ereignisse eintreten oder bestimmte

Aktionen ausführbar bzw. erfolgreich sind, um dies bei der Entscheidungsfindung zu berücksichtigen.

Die Entscheidungstheorie [Von Neumann & Morgenstern 1944] basiert auf dem Prinzip der Nutzenoptimierung, nach dem bei mehreren Alternativen diejenige mit dem höchsten zu erwartenden Nutzen ausgewählt wird. Gegebenenfalls wird dieser Nutzen relativiert auf die Wahrscheinlichkeit der jeweiligen Alternative. Der Nutzen gibt eine numerische Präferenz einzelner Alternativen an, die auf diese Weise in Relation zueinander gesetzt werden können. Nutzen wird dabei als willkürlich und nicht objektivierbar angesehen, so daß dieser explizit vorgegeben werden muß. Andernfalls muß er berechnet oder abgeschätzt werden relativ zu sämtlichen Zuständen mit einem bekanntem Nutzen, denen eine Alternative dienlich oder hinderlich ist. Die Spieltheorie befaßt sich mit Entscheidungen, in die mehrere nutzenoptimierende Akteure mit gemeinsamen oder widersprechenden Zielen involviert sind.

BDI

Ein verbreiteter Ansatz zur Modellierung eines Einzelagenten ist die BDI-Theorie. BDI steht dabei für die Einstellungen Überzeugung (Belief), Motivation (Desire) und Absicht (Intention), mit denen der Zustand eines Agenten beschrieben wird. Die Attraktivität dieses Ansatzes stammt in erster Linie aus der Formalisierung des (normativen) Zusammenhangs zwischen diesen Einstellungen (vgl. Kapitel 2.3.1), mit deren Hilfe sich die Eigenschaften von Agenten definieren und analysieren lassen. Die verwendete Logik dient dabei der Spezifikation, der Implementierung wie auch der Verifikation des Verhaltens von Agenten.

Im Vergleich zu klassischen Ansätzen von Wissen und Zielen in der KI geschieht die Abgrenzung dahingehend, daß die Überzeugungen als unvollständig und unsicher aufgefaßt werden und bei den Motivationen vielfältige und auch inkompatible Zielsetzungen zulässig sind [Rao & Georgeff 1995]. Die zusätzliche Repräsentation von Intentionen erlaubt die explizite Berücksichtigung von Entscheidungen, gegenüber denen unterschiedliche Grade der Verpflichtung bestehen bezüglich der Bedingungen, unter denen vorhandene Absichten aufrechterhalten oder aufgegeben werden. Auf diese Weise wird eine Balance zwischen der Kontinuität der Entscheidungen und der Reaktivität angesichts geänderter Umstände hergestellt.

Allerdings orientieren sich die meisten konkreten Umsetzungen der BDI-Theorie vor allem terminologisch an dieser und lassen sich inhaltlich ziemlich direkt auf die klassischen KI-Begriffe Wissen, Ziel und Plan zurückführen, so daß sie sich oft nur unwesentlich von klassischen KI-Ansätzen unterscheiden, wie sie bereits oben beschrieben wurden.

2.3.3. Lernen und Adaptivität

Um das Verhalten eines Agenten mit der Zeit zu verbessern, können zudem Lernverfahren eingesetzt werden. Dazu gehört zunächst die Fähigkeit, Wissen über die Umgebung aufzunehmen und gegebenenfalls auch diese aktiv zu erkunden. Meist ist diese Wissensaufnahme allerdings auf reines Faktenwissen über den Zustand der Umgebung innerhalb eines fest vorgegebenen Rahmens beschränkt. Weitergehende Lernverfahren bezüglich der Umgebung betreffen das Ableiten von Regelmäßigkeiten aus einzelnen Beispielen durch induktives Bilden von Hypothesen.

Eine andere Lerngelegenheit ist die Optimierung der eigenen Verhaltenssteuerung. Dies ist jedoch stark von der internen Struktur des Agenten und seiner Mechanismen zur Verhaltenskontrolle abhängig. Beispiele hierfür sind die Wiederverwendung und Modifikation von Plänen, das Ableiten und Ändern von Regeln oder die Optimierung von Suchverfahren.

Insbesondere bei Agenten, die als Stellvertreter eines menschlichen Benutzers fungieren, ist eine Adaption an dessen persönliche Präferenzen, Interessen und Gewohnheiten wünschenswert. Eine derartige Benutzermodellierung kann sowohl weitgehend automatisiert durch Beobachtung und Protokollierung des Benutzerverhaltens geschehen, als auch in der Interaktion mit dem Benutzer durch Fragen und Gelegenheiten zur Bewertung der Aktionen des Agenten.

2.4. Die Agentengesellschaft

Während bei der Gestaltung eines einzelnen Agenten vorwiegend auf bekannte Techniken der Informatik und der KI zurückgegriffen werden kann, orientiert man sich bei der Frage der Organisation der Agenten innerhalb eines Multiagentensystems mehr an sozialwissenschaftlichen Theorien. Diese umfassen die Aspekte Kommunikation, Koordination und Kooperation, auf die im folgenden näher eingegangen werden soll. Konstituiert wird eine Agentengesellschaft dabei über eine Infrastruktur, die mehrere Einzelagenten zu einem System verbindet. Weiterhin gibt es gerade in diesem Bereich agentenspezifische Standardisierungsbemühungen zur Interoperabilität innerhalb von und zwischen unterschiedlichen Agentensystemen.

2.4.1. Standardisierungen

Sobald innerhalb einer Agentengesellschaft mehrere Agenten miteinander interagieren, muß eine hinreichende Interoperabilität zwischen diesen gewährleistet sein. Dies gilt insbesondere für heterogene Agentengesellschaften, die aus Agenten verschiedener Hersteller und sogar verschiedener Agentensysteme bestehen.

Interoperabilität läßt sich in technischen Systemen am ehesten durch Standardisierungen erreichen, die einen einheitlichen Rahmen zur Kompatibilität verschiedener Systeme definieren.

Systemübergreifende Standards

Im Bereich der Agententechnologie haben vor allem die folgenden Standardisierungsbemühungen Beachtung gefunden:

- *Knowledge Sharing Effort*[6] (KSE): Ziel des KSE ist es, den Austausch von Wissen zwischen heterogenen Systemen mit unterschiedlichen Wissensrepräsentationsformalismen zu ermöglichen. Dazu existieren standardisierte Sprachformate auf den drei Ebenen der Kommunikation (KQML), der Wissensrepräsentation als Inhalt der Kommunikation (KIF) sowie der Domänenmodellierung für zu repräsentierendes Wissen (Ontolingua).
- *Mobile Agent System Interoperability Facilities*[7] (MASIF): MASIF spezifiziert eine Infrastruktur für mobile Agenten. Dies umfaßt zum einen die Verwaltung von Agenten und Aufenthaltsumgebungen für diese, zum anderen den Transport von Agenten zwischen verschiedenen Aufenthaltsumgebungen.
- *Foundation for Intelligent Physical Agents*[8] (FIPA): Das umfassendste Standardisierungsvorhaben ist das der FIPA, das verschiedene Aspekte aus Bereichen wie Agentenmanagement, Kommunikation und mobile Agenten abdeckt und beständig erweitert wird. Nur ein Teil dieser Spezifikationen wird jedoch als obligatorisch angesehen, vieles ist entweder optional oder rein deskriptiv.

Einige relevante Inhalte der genannten Standardisierungen werden in den nachfolgenden Unterkapiteln jeweils noch im thematischen Zusammenhang vorgestellt.

Bislang hat sich aber keiner dieser Standards durchsetzen können. Zwar nutzen viele Agentensysteme zumindest einzelne ihrer Spezifikationen, jedoch weniger zugunsten der Kompatibilität zu anderen Agentensystemen, sondern eher als geeigneten Lösungsansatz für ein spezifisches Problem, der dann entsprechend frei ausgelegt und angepaßt wird.

Dies hat verschiedene Gründe. Zunächst ist das Gebiet der Agententechnologie sehr umfassend und uneinheitlich. Dadurch kann eine Standardisierung meist nur gewisse Aspekte eines bestimmten Agentenbegriffs abdecken, so daß sie zu spezifisch ist, um weithin anwendbar zu sein und akzeptiert zu werden, zumal wenn die Kompatibilität zu Standardisierungen für andere Bereiche fehlt. Andererseits bleibt eine umfassende Standardisierung wie die der FIPA leicht unvollständig und an

---

6 www-ksl.stanford.edu/knowledge-sharing, www.cs.umbc.edu/kse

7 www.fokus.gmd.de/research/cc/ecco/masif; MASIF wurde als Vorschlag eingereicht bei OMG (Object Management Group, www.omg.org; s.a. OMG Agent Working Group, www.objs.com/isig/agents.html)

8 www.fipa.org

vielen Stellen zu unpräzise, um noch eine ausreichende Interoperabilität gewährleisten zu können.

Das eigentliche Problem ist aber die enorme Komplexität und der Forschungscharakter der Agententechnologie. Eine Standardisierung beinhaltet i.a. eine Festlegung auf eine von mehreren hinreichend bekannten Alternativen oder eine Synthese aus diesen. Das Spektrum der verfolgten Ansätze im Agentenbereich reicht von technischen, praxisnahen bis zu allgemeinen, theoretischen und ist noch zu weit gestreut und uneinheitlich, um bereits auf einen gemeinsamen, den unterschiedlichen Anforderungen und Ansichten gerecht werdenden Standard reduzierbar zu sein. Eine frühzeitige Festlegung auf noch unausgereifte und nicht hinreichend erforschte Technologien könnte sich entweder als Standard nicht durchsetzen oder könnte zu einer unnötigen Einengung des Forschungsspielraums führen. Somit werden die Standardisierungsbemühungen zunächst nur ein Beitrag unter anderen zu den vielfältigen Forschungsarbeiten auf dem Gebiet der Agenten bleiben.

Systeminterne Standards

Dies gilt jedoch nur für systemübergreifende Standardisierungen, die eine Interoperabilität zwischen Agenten verschiedener Systeme ermöglichen sollen. Die Interoperabilität zwischen Agenten eines Systems wird in der Regel durch einheitliche Mechanismen und Datenformate zur Interaktion gewährleistet, die systemintern die Aufgabe von Standards übernehmen.

Dabei sind zwei Standardisierungsebenen zu unterscheiden. Auf einer Systemebene muß zunächst die allgemeine Möglichkeit zu Interaktionen zwischen Agenten geschaffen werden. Da Agentensysteme offene Systeme sind, reichen diese Standardisierungen alleine noch nicht aus, sondern sie müssen eine weitere Ebene unterstützen, auf der anwendungsspezifisch für die Inhalte der Interaktionen jeweils dynamisch neue Konventionen etabliert werden können.

2.4.2. Kommunikation

Kommunikation ist das zentrale Interaktionsmedium für Multiagentensysteme, bei Softwareagenten sogar oft die einzige Möglichkeit zur Interaktion. Die Agentenkommunikation unterscheidet sich dabei vom herkömmlichen Datenaustausch durch die Verwendung einer Agentenkommunikationssprache, die von spezifischen Datenformaten abstrahiert und eine Kommunikation verschiedenster Agenten ermöglichen soll. Als theoretischer Ausgangspunkt für die Agentenkommunikation dient meistens die Sprechakttheorie. Und auch für die Inhalte der Kommunikation werden einheitliche Formalismen und Begriffswelten benötigt.

Sprechakttheorie

Die Sprechakttheorie [Searle 1969] hebt den Doppelcharakter der menschlichen Kommunikation hervor. Zum einen werden in der Sprache Informationen übermittelt, zum anderen sind einzelne sprachliche Äußerungen konkrete Handlungen, mit denen eine bestimmte Absicht verfolgt wird. Entsprechend wird ein Sprechakt in die drei folgenden Bedeutungsebenen unterteilt [Austin 1962]:

- *lokutionäre Ebene*: Die physische Handlung des Sprechens beinhaltet Aspekte wie die Artikulation und den Aufbau eines Sprechakts.
- *illokutionäre Ebene*: Die kommunikative Funktion betrifft den Informationsgehalt eines Sprechakts. Die illokutionäre Rolle gibt dabei mit der Art der Handlung an, wie der Inhalt aufgefaßt werden soll, beispielsweise als Feststellung oder als Frage. Der propositionale Gehalt ist die übermittelte Bedeutung.
- *perlokutionäre Ebene*: Als Handlung hat ein Sprechakt sowohl eine intendierte Wirkung des Aussprechenden, als auch eine eingetretene Wirkung auf den Angesprochenen.

Die beiden umfassendsten und am häufigsten verwendeten Umsetzungen der Sprechakttheorie im Bereich der Agentenkommunikation entstammen den beiden bereits erwähnten Standardisierungen KSE und FIPA und werden im folgenden kurz vorgestellt.

Agentenkommunikationssprachen

Eine Kommunikationssprache definiert ein Format zur Übertragung von Sprechakten, wobei das Format des zu übertragenden Inhalts noch möglichst offen gelassen wird.

Das Sprachformat Knowledge Query and Manipulation Language (KQML) wurde im Rahmen des KSE entwickelt [DARPA 1993, Labrou & Finin 1997]. Syntaktisch ist ein KQML-Sprechakt eine Menge von Parametern, denen jeweils ein Wert zugeordnet wird. Die Menge der möglichen Parameter ist offen, typische Sprechaktparameter sind jedoch bereits vorgegeben. Dazu gehören die Adressierung über Absender (sender) und Empfänger (receiver) und die Zuordnung von Antworten (reply-with und in-reply-to). Die zu übermittelnde Information wird über ein Performative (das ohne Parameterbezeichnung immer am Anfang steht) und den Inhalt (content) angegeben. Um das Format für den Inhalt offen zu halten, wird dieses über die Parameter der Inhaltssprache (language) und der Ontologie (ontology) angegeben.

KQML stellt eine vordefinierte Menge an Performatives zur Verfügung, deren Semantik über Operationen auf einer sogenannten Virtuellen Wissensbasis (VKB) definiert ist. Jeder Agent verhält sich nach außen hin, als ob sein Kommunikationsverhalten eine interne Wissensbasis aus gespeicherten Fakten und Zielen betrifft, die beispielsweise mitgeteilt, abgerufen, überprüft oder abgeändert werden. So ent-

spricht eine Feststellung (tell, untell, deny) dem Mitteilen von Inhalten der VKB und eine Frage (ask-if, ask-one, ask-all) dem Abruf von deren Inhalten. Andere vorgegebene Performatives sind definiert als Veränderungen der VKB oder der Umgebung des Agenten. Hinzu kommen noch Performatives für spezifische Aspekte der Agentenkommunikation wie das Agentenmanagement und die Kommunikationsinfrastruktur. Es existiert auch ein formaler Ansatz für eine Semantik basierend auf Wissen und Zielen von Agenten [Labrou & Finin 1994].

Einen ähnlichen Ansatz verfolgt die Agent Communication Language (ACL) der FIPA-Spezifikation [FIPA 2000: PC00037F, XC00061D]. Die Syntax ist identisch, verwendet aber zum Teil andere Bezeichnungen für die vordefinierten Parameter und Communicative Acts (der FIPA-Entsprechung der Performatives in KQML), auch wenn die Funktionalität sehr ähnlich ist. Auch hier sind die Mengen der Parameter und Communicative Acts offen. Die Semantik der vorgegebenen Communicative Acts ist über eine Modallogik namens Semantic Language (SL) definiert, die Agenten als handelnde Entitäten mit sicherem und unsicherem Wissen sowie mit Zielen beschreibt.

Auch wenn eine formale Semantik für eine exakte Festlegung der Bedeutung von Sprechakten unerläßlich ist, so ist sie doch nur bedingt verwendbar zur Verifizierung, ob das Kommunikationsverhalten eines Agenten dieser Semantik gerecht wird oder nicht [Wooldridge 1998]. Dies liegt zum einen an der Ausdrucksmächtigkeit der verwendeten Logiken, die i.a. nicht entscheidbar sind, zum anderen an der bereits erwähnten Unterbestimmtheit der Zuschreibung mentaler Zustände, die die Grundlage dieser Logiken bilden, zu Agenten.

Inhaltssprachen

KQML und ACL lassen bewußt das Format für den propositionalen Inhalt der Sprechakte offen. KSE und FIPA schlagen jedoch jeweils eine zugehörige Inhaltssprache vor, die der Semantik der Performatives der Kommunikationssprache besonders gerecht wird. Da die Inhaltssprache der Kommunikation nicht mit der internen Wissensrepräsentationssprache eines Agenten übereinzustimmen braucht, müssen diese gegebenenfalls ineinander übersetzbar sein, weshalb eine Inhaltssprache eine Obermenge üblicher Wissensrepräsentationsformalismen bereitstellen sollte.

Im Falle von KSE ist dies das Knowledge Interchange Format (KIF) [Genesereth & Fikes 1992], das ausschließlich zur Kommunikation zwischen Agenten gedacht ist und somit nicht als interner Repräsentationsformalismus oder zur Interaktion mit dem Benutzer verwendet werden sollte. Dafür ist KIF möglichst ausdrucksstark und umfassend, so daß sich viele Wissensrepräsentationen nach KIF übersetzen lassen.

Die Syntax von KIF entspricht der von LISP, die Semantik orientiert sich an der Prädikatenlogik. Es werden vier Typen von Ausdrücken unterschieden: Terme, Sätze, Regeln und Definitionen. Terme beschreiben Objekte der Welt und sind

zusammengesetzt aus Konstanten, Variablen, Funktionen, Relationen usw. Sätze sind logische Formeln und drücken Fakten über die Welt aus. Regeln beschreiben erlaubte Ableitungsregeln, die auf Sätze angewandt werden können. Definitionen legen die Bedeutung von Konstanten fest, wobei auch partielle Definitionen erlaubt sind, die die mögliche Bedeutung lediglich einschränken.

Als Inhaltssprache für FIPA-Sprechakte wird die zur Definition der Semantik der FIPA-ACL benutzte Modallogik SL mit einer an LISP angelehnten Syntax vorgeschlagen [FIPA 2000: XC00008D]. Vorgeschrieben für Sprechakte, die für durch FIPA spezifizierte Kommunikationshandlungen verwendet werden, ist i.a. SL0, ein auf einen entscheidbaren Teilbereich von SL beschränkter Formalismus.

Die FIPA-Spezifikation beinhaltet noch drei alternative Inhaltssprachen, für die jedoch kein expliziter semantischer Bezug zu den vorgegebenen Communicative Acts besteht. Dies ist zum einen das bereits erwähnte KIF [FIPA 2000: XC00010A], zum anderen das Resource Description Framework (RDF) [FIPA 2000: XC00011A], ein vom World Wide Web Consortium definiertes Schema zur formalen Beschreibung des Inhalts von Internet-Seiten, sowie die von FIPA definierte Constraint Choice Language (CCL) [FIPA 2000: XC00009A], einer Sprache zur Formulierung von Contraint-Lösungsproblemen.

Gemeinsame Ontologien

Die bisher dargestellten Sprachen legen lediglich die Form, aber nicht die möglichen Inhalte der Kommunikation fest. Um neben einer einheitlichen Sprachstruktur auch ein gemeinsames Vokabular bereitzustellen, werden gemeinsame Ontologien verwendet. Wie im Fall der Wissensrepräsentation für Einzelsysteme (Kapitel 2.3.1) beinhaltet dies ein domänenspezifisches Vokabular und gegebenenfalls auch zugehörige Kategoriestrukturen und Definitionen. Zusätzlich muß bei einer gemeinsamen Ontologie gewährleistet sein, daß sie allen Kommunikationspartnern bekannt ist und diese sie einheitlich interpretieren. Wie bei den Inhaltssprachen brauchen die in der Kommunikation verwendeten Ontologien nicht mit den internen Repräsentationen übereinzustimmen, so daß gegebenenfalls eine Übersetzung notwendig ist.

Ontolingua [Gruber 1992] ist eine im Rahmen von KSE entstandene Sprache, in der aufbauend auf KIF Ontologien in einem einheitlichen Format definiert werden können. Sie umfaßt Definitionsschemata für Klassen, Relationen, Funktionen und Objektkonstanten. Außerdem existieren Übersetzungsprogramme für einige verbreitete Wissensrepräsentationsformalismen wie KEE und Loom.

Die FIPA-Spezifikation zur Verwendung von Ontologien in Agentensystemen [FIPA 2000: XC00086C] definiert als Zwischenrepräsentationssprache für Ontologien eine Metaontologie in Anlehnung an das OKBC-Format (Open Knowledge Base Connectivity [Chaudhri et al. 1998]), das auf Kategorieschemata beschränkt ist. Weiterhin fordert sie einen eigenen Agenten, den Ontologie-Agenten (OA), der

unter Zugriff auf Programme für die Verwaltung und Übersetzung von Ontologien folgende Aufgaben erfüllt:

- *Auswahl von Ontologien*: Agenten müssen für die Kommunikation übereinstimmende oder übersetzbare Ontologien auswählen können.
- *Zugriff auf Ontologien*: Ein Agent kann Ontologien neu erzeugen und speichern sowie bestehende aktualisieren und laden. Außerdem kann er einzelne Definitionen und Ausdrücke einer Ontologie abrufen oder verändern.
- *Übersetzen von Ausdrücken*: Der Ontologie-Agent kann Ausdrücke aus einer Ontologie in eine andere übersetzen, sofern die Ontologien ineinander übersetzbar sind.

Zur Verwendung unterschiedlicher Ontologien und Inhaltssprachen ist anzumerken, daß Übersetzungen sehr aufwendig sein können und weder eindeutig, noch vollständig möglich zu sein brauchen. Die vorherige Festlegung einer einheitlichen Inhaltssprache und gemeinsamer Ontologien ist deshalb immer ratsam, sofern bereits zur Designzeit eine Interaktion zwischen bestimmten Agenten oder Agententypen absehbar ist.

2.4.3. Koordination

Da das Verhalten der Agenten innerhalb einer Gesellschaft in der Regel gegenseitige Abhängigkeiten aufweist, müssen die einzelnen Aktivitäten der Agenten gegebenenfalls untereinander koordiniert werden. Koordination dient dabei einerseits der Steigerung der Effektivität, andererseits dem Umgang mit Konflikten. Koordinationsmechanismen werden insbesondere zur Verteilung und gemeinsamen Durchführung von Aufgaben sowie der Verteilung und gemeinsamen Nutzung von Ressourcen eingesetzt.

Sonderfälle der Koordination sind die Zusammenarbeit und die Kooperation. Durch Zusammenarbeit können Agenten Aufgaben durchführen, die sie alleine nicht oder nicht so effektiv bewältigen könnten, indem sie auf die Fähigkeiten anderer Agenten zurückgreifen oder ihre Kräfte vereint einsetzen[9]. Kooperation bedeutet die Koordinierung des Verhaltens mehrerer Agenten auf eine gemeinsame Aufgabe oder einen gegenseitigen Nutzen hin. Ein Beispiel für Kooperation sind Produktionsprozesse, in denen jeder Agent auf einen Arbeitsschritt spezialisiert ist, so daß er zur Erfüllung seiner Aufgabe von den jeweils vorhergehenden Schritten abhängt. Im folgenden werden einige Ansätze zur Koordination, Zusammenarbeit und Kooperation in Multiagentensystemen kurz vorgestellt.

---

[9] Eine Summierung von Kräften ist bei reinen Softwareagenten selten sinnvoll oder möglich. Ein typisches Beispiel sind Roboter, die zusammen einen Gegenstand bewegen, den jeder alleine aufgrund des Gewichts nicht bewegen könnte.

Konversationsprotokolle

Die Verwendung von Sprechakten geschieht gewöhnlich nicht isoliert, sondern diese sind Teil von strukturierten Konversationen. Typische Konversationsmuster lassen sich durch Protokolle beschreiben, die abstrakt eine Menge möglicher Konversationsverläufe festlegen. Die Kommunikationspartner werden dabei durch Rollen repräsentiert, die sie bei der Durchführung einer konkreten Konversation einnehmen. Ein Protokoll ist ein gerichteter Graph, in dem die Knoten eine Rolle zu einem bestimmten Zeitpunkt darstellen und die Kanten die an dieser Stelle zulässigen Sprechakte, wobei der Ausgangspunkt einer Kante die Rolle des Senders und der Endpunkt die Rolle des Empfängers ist. Mit dem Einsatz von Protokollen lassen sich die kommunikativen Handlungen von Agenten im voraus koordinieren, da für alle Beteiligten an jedem Punkt einer Konversation die Menge der möglichen Aktionen durch die eingenommene Rolle fest vorgegeben ist.

Zuweisung von Aufgaben und Ressourcen

Bei der Zuweisung von Aufgaben oder Ressourcen gibt es einerseits Agenten, die eine Problemlösungskompetenz oder Ressource anbieten, und andererseits Agenten, die dieses Angebot zur Durchführung ihrer Aufgaben benötigen. Diese beiden Rollen werden als Anbieter und Nutzer bezeichnet, wobei ein Agent auch beide Rollen zugleich oder hinsichtlich verschiedener Zuweisungen einnehmen kann.

Ferber [1999] unterscheidet dabei die folgenden vier Arten von Zuweisungen:

- *a priori*: Es wird im voraus festgelegt, welche Aufgabe ein Nutzer an welchen Anbieter delegiert. Dies ist zwar effizient, aber sehr starr, da alle Beziehungen zwischen Agenten fest vorgegeben sind, was dem Grundgedanken einer veränderlichen Agentengesellschaft widerspricht.
- *über einen Vermittler*: Die Zuweisung geschieht durch eine zusätzliche Rolle, den Vermittler. Dieser verwaltet zentral die Fähigkeiten aller Anbieter und vermittelt diese auf Anfrage an die Nutzer. Dieser Ansatz wird der Dynamik von Agentengesellschaften gerecht, indem Anbieter ihre Fähigkeiten an- und abmelden können. Die zentrale Verwaltung stellt jedoch vor allem bei Gesellschaften mit vielen Agenten eine Engstelle dar, weil sämtliche Zuweisungen über den Vermittler vorgenommen werden.
- *gegenseitige Bekanntheit*: Jeder Agent kennt im voraus eine begrenzte Menge anderer Agenten und deren Fähigkeiten. Im Bedarfsfall wählt ein Nutzer aus den ihm bekannten Agenten einen geeigneten Anbieter. Ist kein Anbieter bekannt, so kann entlang der Bekanntheitsbeziehungen zwischen den Agenten nach einem Anbieter gesucht werden. Dies ist eine sehr verteilte Lösung, weshalb sowohl die Suche nach Anbietern, als auch die Anpassung an Veränderungen der Agentengesellschaft sehr aufwendig werden können.

- *Ausschreibungen*: Der Nutzer sendet eine Ausschreibung an alle oder an einige ausgewählte Agenten. Diese antworten, ob sie die ausgeschriebene Leistung erbringen können. Aus den positiven Antworten wählt der Nutzer einen geeigneten Anbieter. Dieses Verfahren wird der Verteiltheit und Dynamik von Agentensystemen gerecht, ist jedoch bei vielen Agenten ohne geeignete Beschränkung der möglichen Anbieter bei einer Ausschreibung ebenfalls sehr aufwendig.

Um die jeweiligen Nachteile auszugleichen, können auch Kombinationen dieser Ansätze verwendet werden. Beispielsweise kann eine Vorauswahl der möglichen Anbieter bei einer Ausschreibung durch einen zentralen Vermittler oder die Verteilung von Ausschreibungen auf der Basis gegenseitiger Bekanntheit geschehen. Sobald eine Auswahl aus mehreren Anbietern möglich ist, läßt sich die Systemeffizienz steigern, indem die Auswahl anhand geeigneter Optimalitätskriterien geschieht. Dafür werden meist Marktmechanismen eingesetzt, wie sie nachfolgend beschrieben werden.

Marktmechanismen

Die Verteilung von Aufgaben und Ressourcen mit Hilfe von Marktmechanismen basiert auf dem Prinzip von Angebot und Nachfrage [Fischer et al. 1998, Ygge 1998]. Dabei werden Verfahren wie Ausschreibungen und Auktionen als Konversationsprotokolle modelliert, in denen jeweils ein Agent eine Aufgabe oder Ressource zur Verfügung stellt und dann unter mehreren Bewerbern den mit den günstigsten Konditionen ermittelt.

Beim Contract-Net-Protokoll [Smith 1980] sendet ein Agent eine Ausschreibung für eine Aufgabe an mögliche Interessenten. Diejenigen, die zur Übernahme der Aufgabe bereit sind, teilen dies dem ausschreibenden Agenten unter Angabe der diesbezüglichen Bedingungen mit. Unter diesen wählt der erste Agent einen aus, dem er die Aufgabe überträgt. Im Anschluß daran ist weitere Kommunikation zwischen Auftraggeber und Auftagnehmer zur Umsetzung der Aufgabe möglich, bis schließlich das Mitteilen des Ergebnisses das Protokoll beendet. Durch das Contract-Net-Protokoll lassen sich so dynamisch Aufgaben zu den jeweils momentan besten Konditionen zuteilen.

Andere Verteilungsverfahren orientieren sich an geläufigen Auktionsmechanismen. Bei der Englischen Auktion machen einzelne Bieter ausgehend von einem Mindestgebot jeweils so lange höhere Angebote, bis niemand ein höheres Gebot abgibt. Bei der Holländischen Auktion verringert umgekehrt der Anbieter seine anfängliche Forderung, bis ein Käufer den Preis akzeptiert. In beiden Fällen wird iterativ ein Höchstpreis anvisiert, den einer der Kunden bereit ist zu zahlen. Bei der Umsetzung in Multiagentensystemen können auch andere Kriterien als nur der Preis zur Erteilung des Zuschlags herangezogen werden.

Multiagentenplanung

Um die Handlungen mehrerer Agenten zu koordinieren, kann ein gemeinsamer Plan für diese Agenten erstellt werden [Martial 1992]. Dies empfiehlt sich vor allem bei Agenten, deren Verhalten ohnehin durch Pläne gesteuert wird und deren Aktionen nicht teilbare Ressourcen betreffen. Dazu kann teilweise auf Planungsverfahren für Einzelagenten zurückgegriffen werden, da auch in diesen bereits unterschiedliche Handlungen und Handlungssequenzen miteinander koordiniert werden müssen (vgl. Kapitel 2.3.2). Der Planungsvorgang und die Kontrolle der Planausführung kann auf unterschiedliche Arten erfolgen [Ferber 1999]:

- *zentral*: Ein ausgezeichneter Agent erstellt den Plan für alle beteiligten Agenten und überwacht dessen Ausführung.
- *moderiert*: Ein ausgezeichneter Agent verwaltet den Plan und den Ausführungszustand, aber alle Agenten sind an der Planerstellung beteiligt.
- *verteilt*: Jeder Agent erstellt zunächst einen eigenen Plan, den er dann so lange mit den anderen Agenten koordiniert, bis keine Konflikte zwischen den Einzelplänen mehr bestehen. Gegebenenfalls muß auch die Ausführung koordiniert werden, wenn ein Agent auf die Beendigung eines Planschrittes eines anderen Agenten warten muß.

Jeder dieser drei Ansätze hat eigene Vor- und Nachteile. Insgesamt ist der lokale Planungsaufwand und die Fehleranfälligkeit um so geringer, je verteilter die Planung vorgenommen wird, dafür erhöhen sich die Kommunikationskosten und das Konfliktpotential zwischen den Agenten entsprechend. Verteiltes Planen erlaubt eine dynamischere Form der Planung, bei der lokal auf Änderungen während der Ausführung reagiert werden kann, während zentrales Planen eine bessere Kontrolle des Planungsvorgangs erlaubt. Welcher dieser Ansätze besser geeignet ist, hängt von der konkreten Anwendungssituation ab.

Koordination ohne Kommunikation

Während Marktmechanismen und Multiagentenplanung eine Koordinierung durch Austausch von Informationen und Verhandlungen zur Laufzeit vornehmen, verzichten andere Koordinationsmechanismen auf Kommunikation, indem sie auf im voraus etablierte Regelungen und Annahmen zurückgreifen. Dies verringert zwar zumeist den Aufwand zur Koordination, ist dafür aber auch weniger flexibel. In den meisten Fällen empfiehlt sich eine Kombination von apriorischer und kommunikativer Koordination, um diese sowohl effizient, als auch flexibel zu gestalten.

Eine Variante der Koordination ohne Kommunikation sind Konventionen. Für vorhersehbare Konfliktfälle wird bereits im voraus festgelegt, wie sie zu beheben sind. Dazu müssen allerdings sämtliche beteiligten Agenten einen bestehenden Konflikt gleich auffassen, um ihn in einer kohärenten Weise zu behandeln. Konventionen können explizite Regeln und Gesetze sein, sie lassen sich aber auch

tionen können explizite Regeln und Gesetze sein, sie lassen sich aber auch analog zu Kräfteverhältnissen modellieren, bei denen jeweils der „Stärkere“ gewinnt.

Die Nutzung des Wissens über andere Agenten und die gemeinsame Umgebung kann ebenfalls den Bedarf an Kommunikation zur Koordination verringern oder auch ganz beseitigen. Zusammen mit Annahmen über die Verhaltensweisen eines Agenten erlaubt dieses Wissen Hypothesen über dessen Verhalten in einer gegebenen Situation, die dann in der Auswahl des eigenen Verhaltens berücksichtigt werden können. Dies gilt für kooperative wie für kompetitive Agentengesellschaften. Ein Beispiel hierfür ist die Spieltheorie [Axelrod 1984], die für bestimmte Spielszenarien optimale Strategien ableitet ausgehend von der Annahme der Rationalität im Sinne der Nutzenmaximierung des Gegenübers.

Organisationsstrukturen

Agentengesellschaften lassen sich in Gruppierungen und Hierarchien von Agenten organisieren, um Interaktions- und Koordinierungsstrukturen zu optimieren. Auf diese Weise lassen sich vor allem Systeme mit vielen Agenten zugunsten eines effektiveren Gesamtverhaltens strukturieren. Durch Organisationsstrukturen lassen sich Kompetenzen bündeln oder verteilen sowie Interaktionsmuster und Koordinierungsverfahren festlegen.

Ein Beispiel für Gruppierungen sind Teams, deren Mitglieder bevorzugt oder ausschließlich untereinander interagieren. Dadurch wird der Koordinierungsaufwand auf die Mitglieder eines Teams und auf die Teams untereinander beschränkt und somit bei geeigneter Zusammensetzung der Teams reduziert. In Hierarchien übernehmen übergeordnete Ebenen vorwiegend Aufgaben der Koordinierung und Entscheidung für die darunterliegenden. Dies verringert den Koordinierungsaufwand, indem die Koordinierung auf verschiedene Abstraktionsebenen aufgeteilt wird.

2.4.4. Infrastruktur

Zu einer Gesellschaft verbunden werden Agenten durch eine gemeinsam genutzte Infrastruktur, die die Grundlage für Interaktionen bereitstellt[10]. Im einfachsten Fall kann dies eine reine Kommunikationsinfrastruktur sein wie beispielsweise eine Netzwerkinfrastruktur, die auch unabhängig vom Agentensystem existieren kann. Normalerweise bietet eine Agenteninfrastruktur aber zusätzliche Funktionalitäten zur Verwaltung der Agenten einer Gesellschaft. Um Interaktionen auch über die Kommunikation hinaus zu unterstützen, sammelt und offeriert sie Informationen über verfügbare Agenten und die Möglichkeiten, mit diesen zu interagieren. Darüber hinaus kann eine Agenteninfrastruktur auch allgemeine Hilfsfunktionalitäten

[10] Im Falle physischer Agenten kann bereits die gemeinsame Umgebung als Infrastruktur ausreichen.

umfassen, beispielsweise für Management und Absicherung des Agentensystems und seiner Agenten. Bei Systemen mit mobilen Agenten (vgl. Kapitel 2.1.4) wird der Transport dieser Agenten üblicherweise ebenfalls über die Infrastruktur realisiert.

FIPA Agentenmanagement

Die FIPA-Management-Spezifikation [FIPA 2000: XC00023G] legt eine Infrastruktur für Agentensysteme fest. Agenten besitzen dabei eine Identifikation (Agent Identifier, AID) mit einem unveränderlichen, global eindeutigen Namen und sind zu einem Zeitpunkt genau einer Agentenplattform (Agent Platform, AP) zugeordnet. Jede Agentenplattform stellt zumindest die folgenden drei Infrastrukturdienste bereit:

- *Agent Management System* (AMS): Das AMS verwaltet die Agenten einer AP. Zum einen kontrolliert es den Lebenszyklus der Agenten, wozu das Erzeugen, Beenden und Migrieren gehören. Zum anderen gibt es Auskunft über die bei ihm angemeldeten Agenten und ihre Kommunikationsadressen.
- *Message Transport System* (MTS): Das MTS stellt einen Kommunikationskanal zwischen Agenten zur Verfügung, über den Sprechakte anhand der AID versendet werden. Dies geschieht innerhalb einer AP direkt durch den MTS, zwischen zwei APs durch Weiterleiten an den MTS der anderen AP über den sogenannten ACC (Agent Communication Channel).
- *Directory Facilitator* (DF): Der DF verwaltet die Dienste[11] der Agenten einer AP. Agenten melden die Dienste an, die sie anbieten, und können nach angebotenen Diensten suchen.

Untereinander verbunden sind verschiedene APs über ihre MTS sowie über die gegenseitige Registrierung der Infrastrukturdienste beim DF anderer APs.

2.5. Anwendungsfelder für Multiagentensysteme

Die Agententechnologie und insbesondere Multiagentensysteme stellen neue Programmierparadigmen dar, die anders als beispielsweise die objektorientierte Programmierung keine generellen Ansätze sind, sondern auf bestimmte Anwendungsfelder und Einsatzbereiche hin ausgerichtet sind [Jennings & Wooldridge 1998]. Aufgrund ihrer Eigenschaften eignen sich Multiagentensysteme für Anwendungen, die komplex und verteilt sind [Jennings 2001] und in denen wegen ihrer Offenheit und Dynamik herkömmliche Ansätze an ihre Grenzen stoßen. Die typischen Anwendungsbereiche lassen sich unterteilen in Koordinierungsaufgaben,

[11] Die Dienstbeschreibung besteht lediglich aus einem Namen, einem Typ, einer Ontologie und je einer Liste an festen und verhandelbaren Eigenschaften.

kooperative Problemlösungen, elektronische Handelsplattformen und simulierte Gesellschaften.

2.5.1. Koordinierung in verteilten Systemen

Ein primärer Einsatzbereich von Multiagentensystemen ist die Koordinierung in verteilten Systemen. Die Verteiltheit kann in einer räumlichen oder funktionalen Verteilung oder auch in beidem bestehen. Abgebildet wird die Verteiltheit auf einzelne Agenten, die jeweils für eine räumliche oder funktionale Einheit stehen. Die Agenten bzw. die von ihnen vertretenen Entitäten können dabei kooperativ an einer gemeinsamen Zielsetzung arbeiten oder in Wettbewerb zueinander stehen. In beiden Fällen bestehen Abhängigkeiten zwischen den Agenten, die sich in der Regel durch Konflikte, Redundanzen und verteilte Kompetenzen beschreiben lassen. Die durch das Agentensystem zu erreichende Koordinierung besteht nun in der Vermeidung und Auflösung von Konflikten, der Nutzung vorhandener bzw. der Vermeidung ungewollter Redundanzen und der Verteilung von Aufgaben und Ressourcen. Allgemeiner formuliert bedeutet die Koordinierung die Optimierung des Gesamtverhaltens und einzelner Abläufe hinsichtlich des Gesamtsystems und/oder der Einzelinteressen der Agenten.

Für derartige Koordinierungsaufgaben gibt es zahlreiche Anwendungsgebiete. Beim Supply Chain Management (z.B. ISCM [Fox at al. 1993], SURGE [Ivezic & Potok 1999][12]) werden die Zulieferketten von den Rohstoffen über verschiedene Zwischenprodukte bis hin zum Endprodukt durch Agenten modelliert. Jeder Lieferant und jeder Einkäufer wird durch jeweils einen Agenten repräsentiert, wobei die Hersteller der Zwischenprodukte gleichzeitig als Einkäufer der von ihnen benötigten Ausgangsprodukte und als Lieferant für die ihre Produkte weiterverarbeitenden Hersteller auftreten, woraus gerade die Zulieferketten entstehen. Die Optimierungs- bzw. Koordinierungsaufgabe besteht nun darin, ausgehend vom Bedarf der Endkunden den Warenfluß so zu steuern, daß sämtliche Ausgangsprodukte jeweils genau dann bei einem Hersteller verfügbar sind, wenn sie benötigt werden. Ein Teilproblem hiervon ist das Transport Chain Management (z.B. TeleTruck [Funk et al. 1998]), das auf die Organisation der Transportketten beschränkt ist.

Ähnliche Koordinierungsprobleme beinhalten die Produktionssteuerung (z.B. MABES [Ivezic et al. 1999]), das Flottenmanagement (z.B. Mars [Fischer et al. 1996], TeleTruck [Bürckert et al. 1997]) und das Netzwerkmanagement (z.B. HYBRID [Somers 1996], Tele-MACS [Hayzelden 1998]). In der Produktionssteuerung werden die zu bearbeitenden Werkstücke zwischen verschiedenen Maschinen so verteilt, daß diese möglichst vollständig ausgelastet sind, um die Produktivität zu maximie-

[12] Die im folgenden zu den einzelnen Anwendungsarten genannten Systeme und Referenzen beziehen sich auf Agentensysteme aus dem jeweiligen Bereich.

ren und Ausfälle dynamisch auszugleichen. Beim Flottenmanagement geht es um eine optimale Auslastung einer Menge von Transportfahrzeugen, indem die zu transportierenden Güter so verteilt werden, daß Leerfahrten vermieden und Fahrstrecken minimiert werden. Das Netzwerkmanagement besteht in einer möglichst gleichmäßigen Verteilung der Netzauslastung, um die Übertragungszeiten von Daten über das Netz zu minimieren. In all diesen Bereichen wird jeweils eine aktive Einheit – Werkmaschine, Transportfahrzeug bzw. Netzknoten – durch einen eigenen Agenten repräsentiert, die gemeinsam den Durchsatz an Material – Werkstück, Güter bzw. Daten – über ein verteiltes System optimieren. Gegebenenfalls werden zusätzliche Agenten für Kontrollhierarchien oder andere Organisationsstrukturen verwendet.

Ebenfalls der Koordinierung der Aktivitäten verschiedener Personen dienen Systeme zur Terminplanung. Hierbei verwaltet jeweils ein Agent den Terminplan einer Person, und die Agenten handeln untereinander Termine so aus, daß es nicht zu Terminkonflikten kommt. Derartige Systeme werden sowohl zur Planung privater und beruflicher Termine (Personal Assistant) (z.B. [FIPA 1997: Part 5], [Garrido & Sycara 1996]), als auch zur Koordinierung der Untersuchungstermine von Patienten eines Krankenhauses konzipiert (z.B. [Decker & Li 1998]).

Bei der verteilten Interpretation wird eine Vielzahl verteilt vorliegender Rohdaten zu einem kohärenten höherstufigen Modell integriert. Jeder Agent verfügt dabei nur über einen begrenzten Ausschnitt der Ausgangsinformationen, die er gemäß seinem lokalen Wissen analysiert und in Bezug zu den Interpretationen anderer Agenten setzt. Durch die Koordinierung von partiellen Informationen zwischen den Agenten entsteht so ein kohärentes Gesamtmodell, das zentral oder auch verteilt in Form von Teilmodellen repräsentiert wird. Die bekannteste konkrete Anwendung hierfür ist die mit dem Partial Global Planning (PGP) [Durfee & Lesser 1991] umgesetzte Verkehrsüberwachung Distributed Vehicle Monitoring Testbed (DVMT) [Lesser & Corkill 1983], bei der die Routen von Fahrzeugen anhand ihrer akustischen Signale verfolgt werden.

2.5.2. Kooperative Problemlösung

Während bei der Koordinierung die Verteiltheit des Problems im Vordergrund steht, ist beim kooperativen Problemlösen vor allem die Kompetenz oder Erfahrung zur Bearbeitung von Aufgaben verteilt. Auch hier kann die Kooperation durch ein gemeinsames Ziel oder den gegenseitigen Nutzen bei der Verfolgung eigener Interessen motiviert sein. In jedem Fall verkörpert der einzelne Agent eine bestimmte Expertise oder ein Wissen, auf das andere zur Bewältigung ihrer Aufgaben zurückgreifen. Typische Anwendungsbereiche sind verteilte Planung und Kontrolle, kooperierende Expertensysteme sowie die Unterstützung menschlicher Kooperation [Durfee et al. 1989].

Bei der verteilten Planung wird im einfachsten Fall das Problem zentral in Teilprobleme zerlegt und auf mehrere Agenten verteilt, die diese isoliert oder kooperativ bearbeiten und deren Ergebnisse schließlich zur Gesamtlösung vereint werden. Ein anderer Ansatz ist die Blackboard-Architektur [Hayes-Roth 1985], bei der spezialisierte Agenten auf dem in einem gemeinsamen Speicher enthaltenen Problem arbeiten und dort jeweils ihren Beitrag zu dessen Lösung einfügen. Bei Ansätzen ohne eine zentrale Koordinierungsstelle ist neben der Kompetenz auch das Problem selbst verteilt oder besteht in Einzelproblemen verschiedener Agenten. Ein Spezialfall sind kooperierende Expertensysteme, in denen mehrere Expertensysteme zusammenarbeiten, die jeweils auf einen bestimmten Teilbereich spezialisiert sind. Anwendungsbeispiele sind die Spracherkennung (Hearsay II [Erman et al. 1980]), Air Traffic Control [Findler & Lo 1986] und Robotersteuerung [Arkin et al. 1987].

Systeme für Computer Supported Cooperative Work (CSCW) und Workflow Management (z.B. ADEPT [Jennings et al. 2000], MACIV [Fonseca et al. 1996]) organisieren die Zusammenarbeit zwischen Personen, Arbeitsabteilungen und/oder Datenverarbeitungssystemen. Während beim CSCW eine weitgehend freie Interaktion zwischen den Beteiligten unterstützt wird, schreibt das Workflow Management feste Bearbeitungsabläufe vor. Hierbei ist jeweils einem Beteiligten und/oder einem Arbeitsvorgang ein Agent zugeordnet, der die Koordinierung mit anderen übernimmt oder unterstützt.

2.5.3. Elektronischer Handel

Auch im Bereich des elektronischen Handels (neudeutsch auch als E-Commerce bezeichnet) werden Multiagentensysteme eingesetzt. Dabei steht jedoch keine globale Koordinierung oder Kooperation im Vordergrund, sondern die Handelspartner – Käufer und Verkäufer – besitzen jeweils einen oder mehrere Agenten, die als deren Stellvertreter fungieren und Handel treiben oder unterstützen. Da Käufer und Verkäufer untereinander wie gegenseitig im Wettbewerb zueinander stehen, besteht die Koordinierung nicht in einer Optimierung, sondern ergibt sich anhand von Marktmechanismen aus den Einzelinteressen der Beteiligten. Motivation für den Einsatz von Multiagentensystemen im elektronischen Handel ist einerseits die Autonomie der Agenten, die als Stellvertreter den Käufer entlasten und den Verkauf automatisieren sollen, sowie die Offenheit und Dynamik der Agentengesellschaften, die flexible Veränderungen an Anbietern, Angeboten und Preisen erlauben.

Einfachere Agentensysteme im Bereich des E-Commerce beschränken sich auf Kaufberatung durch die Suche nach und den Vergleich von Angeboten, so daß die Entscheidung weiterhin dem Benutzer selbst überlassen wird. Modelliert werden zudem Verhandlungen und Auktionen, die die Agenten weitgehend autonom im Auftrag ihrer Besitzer durchführen (z.B. Tête à Tête [Guttman & Maes 1998], Sardine [Morris et al. 2000]). Um wirklich in deren Sinne zu handeln, müssen dabei die

Agenten über eine hinreichende Entscheidungskompetenz verfügen und über Personalisierung an die unterschiedlichen Bedürfnisse und Interessen anpaßbar sein.

Bei komplexeren Multiagentensystemen für elektronischen Handel sollen Agenten neben einzelnen Handelsaktionen auch aufwendigere Aufgaben erledigen, die die Kombination mehrerer Einzelleistungen zu einer Gesamtlösung beinhalten (Mehrwertdienste). Ein Beispiel hierfür ist die Reiseplanung (z.B. [FIPA 1997: Part 4], [Albayrak & Kirsch 1998]), bei der der Agent selbständig Hotels und Verkehrsmittel für eine komplette Reise auch über mehrere Aufenthaltsorte buchen soll. Ein anderes Beispiel ist die Verkehrstelematik (z.B. [Albayrak et al. 1998]), in der unterschiedliche fahrtbezogene Dienste integriert einen Autofahrer unterstützten sollen, wozu u.a. Routenplanung, Verkehrshinweise und Wetterbedingungen gehören.

2.5.4. Gesellschaftssimulationen

Schließlich werden Multiagentensysteme auch zur Simulation von künstlichen und natürlichen Gesellschaften verwendet. Dabei wird das Verhalten einzelner Individuen auf einer bestimmten Abstraktionsebene durch Agenten modelliert und dann das sich daraus ergebende Gesamtverhalten von Gesellschaften dieser Individuen beobachtet. Simuliert werden sowohl Gesellschaften mit relativ einfachen, gleichartigen Agenten – oft in Analogie zu Insektenstaaten – (z.B. Swarm [Minar et al. 1996]), als auch solche mit komplexeren, individuell verschiedenen Agenten (z.B. Echo [Hraber et al. 1997]). Während andere Multiagentenansätze vor allem in der VKI beheimatet sind, findet Forschung auf dem Gebiet der Gesellschaftssimulationen vor allem im Bereich des Artificial Life statt.

Gesellschaften aus Agenten mit möglichst realistischen Charakteren werden vor allem für den Unterhaltungsbereich konzipiert, in denen Agenten untereinander oder auch mit Menschen auf einer realitätsnahen Ebene interagieren. Im Spielebereich ist hier Creatures [Grand & Cliff 1998] zu nennen, ein Spiel, in dem künstliche Ökosysteme erzeugt und simuliert werden können. Agenten als glaubhafte Charaktere werden in dem System OZ [Bates et al. 1992] für interaktives Theater verwendet, wobei ein Agent jeweils eine schauspielerische Rolle übernimmt.

2.6. Agentensysteme

Im folgenden werden einige existierende Agentensysteme beschrieben. Mit PRS, INTERRAP, ZEUS und AgentBuilder werden Systeme vorgestellt, deren Schwerpunkt wie bei CASA auf einer Architektur zur Verhaltenssteuerung mit integrierter Unterstützung von Interaktionen innerhalb von Agentengesellschaften liegt. An-

schließend werden noch einige aktuelle Agentensysteme mit einer anderen thematischen Ausrichtung vorgestellt.

2.6.1. Procedural Reasoning System

Das Procedural Reasoning System (PRS) [Georgeff & Ingrand 1989] ist eine Kontrollarchitektur zur Steuerung von autonomen Agenten unter Echtzeitbedingungen in dynamisch veränderlichen Umgebungen. Es kombiniert im Rahmen eines BDI-Ansatzes (vgl. Kapitel 2.3.2) Reaktivität für die Einbettung in die Umgebung mit Zielgerichtetheit zum Verfolgen längerfristiger Aufgaben. Für den Einsatz in Multiagentensystemen existiert lediglich ein rudimentärer Interaktionsmechanismus zum Austausch von Nachrichten zwischen Agenten, die parallel und ansonsten unabhängig voneinander agieren.

Nachfolgend werden zunächst die Kontrollarchitektur des ursprünglichen PRS beschrieben und anschließend kurz einige spätere Implementierungen und deren jeweilige Besonderheiten vorgestellt. Das PRS ist insofern eine der erfolgreichsten Agentenarchitekturen, als mit ihm und den aus ihm hervorgegangenen Systemen eine Vielzahl von Anwendungen entwickelt und eingesetzt wurden.

Architektur

PRS orientiert sich konzeptionell an dem BDI-Modell, ohne daß die logischen Formalismen der BDI-Theorie explizit verwendet werden oder das Verhalten der Agenten in jedem Fall den Eigenschaften der BDI-Logik entspricht. Dem BDI-Ansatz gemäß ist die Einteilung des Wissens eines Agenten in Fakten, Ziele, Pläne und Intentionen sowie die Verwendung dieses Wissens durch den Interpreter, der anhand von Fakten und Zielen Pläne instantiiert und dann als Intentionen aufstellt. Die Architektur von PRS (Abbildung 3) besteht aus dem zentralen Interpreter und

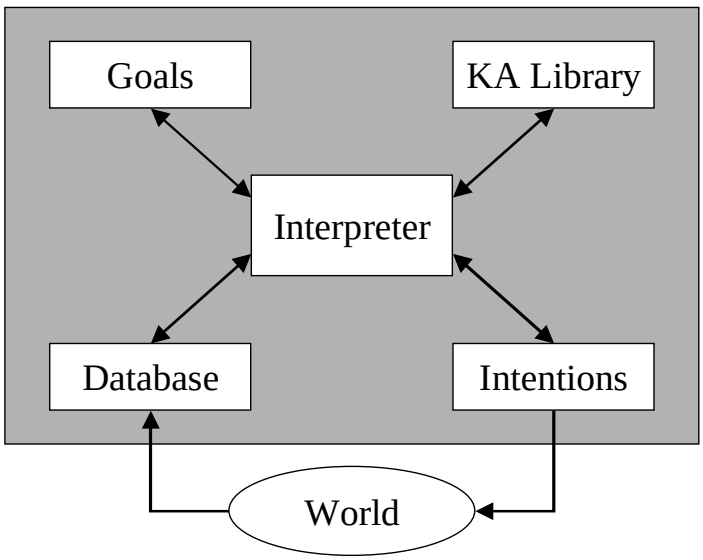

Abbildung 3: Kontrollarchitektur von PRS (nach [SRI 1997])

Komponenten für die verschiedenen Wissenstypen. Die Interaktion mit der Umwelt besteht in der Veränderung der Welt durch Ausführen von Intentionen und der Aufnahme von Informationen über die Welt in Form neuer Fakten.

Die Datenbank von PRS enthält formallogische Ausdrücke, die das Wissen des Agenten über unveränderliche Eigenschaften der Welt und über deren gegenwärtigen Zustand repräsentieren. Zudem kann sie Metawissen über die jeweiligen Fakten, Ziele und Intentionen des Agenten beinhalten. Quelle des Wissens sind neben Beobachtungen der Umgebung auch Schlußfolgerungen und direkte Veränderungen durch die Ausführung von Intentionen. Die Ziele drücken eine Einstellung gegenüber einer logischen Aussage aus, beispielsweise daß der damit beschriebene Weltzustand erreicht oder erhalten werden soll.

Die Pläne, die als Knowledge Areas (KA) bezeichnet werden, bestehen aus einem Ausführungsteil und aus einem Bedingungsteil, der die Umstände der Ausführung beschreibt. Die Bedingungen geben u.a. an, beim Vorliegen welcher Fakten und / oder Ziele eine Ausführung nötig ist sowie in welchen Situationen die Ausführung auch möglich ist. Der Ausführungsteil gibt entweder eine primitive, direkt ausführbare Handlung oder ein abstraktes, prozedurales Handlungsschema an. Ein derartiges Schema ist ein Graph mit einem Startknoten und mehreren möglichen Endknoten, wobei eine Kante jeweils ein Teilziel beinhaltet und Verzweigungen Alternativen darstellen, zwischen denen zur Ausführungszeit gewählt werden kann. Für eine flexible Kontrolle des Interpreters gibt es Metapläne, die an den weiter unten beschriebenen Entscheidungspunkten des Interpreters Anwendung finden. Die zur Ausführung bestimmten Planinstanzen sind die Intentionen des Agenten. Diese werden in einer partiell geordneten Intentionenstruktur verwalten, wobei die Intentionen für ein Teilziel für eine vorherige Ausführung vor der zugehörigen Intention angeordnet sind, während die Intentionen im übrigen parallel nebeneinander stehen.

Der Interpreter arbeitet nun auf diesen Wissensstrukturen und vereint in sich den Kontrollzyklus und die Entscheidungsstrategie der Kontrollarchitektur von PRS. Zugunsten der Reaktivität ist der Interpreterzyklus sehr einfach gehalten. Ausgehend von den aktuellen Änderungen bei Fakten und Zielen bestimmt der Interpreter zunächst die Menge der anwendbaren Pläne durch direktes Matchen mit deren Aktivierungsbedingungen. Aus diesen aktiven Plänen werden nun einige zur Ausführung ausgewählt und zu den Intentionen hinzugefügt. Schließlich wird noch eine Intention ausgewählt und ausgeführt. Diese Ausführung besteht entweder in einer direkt ausführbaren primitiven Handlung, im Aufstellen eines neuen Teilziels gemäß dem momentanen Planschritt oder einer Modifikation des Wissens des Agenten.

Der Interpreter besitzt eine sehr einfache und effiziente Entscheidungsstrategie für die Auswahl zwischen verschiedenen aktiven Plänen für ein Ziel, mehreren ausführbaren Intentionen sowie alternativen Teilzielen innerhalb eines Plans. Für eine flexiblere Verhaltenskontrolle wird an diesen Entscheidungspunkten jedoch

zunächst überprüft, ob es Metapläne gibt, die eine situationsspezifische Entscheidung ermöglichen. Die Aktivierung und Auswahl der Metapläne geschieht durch den normalen Interpreterzyklus, so daß einerseits die Reaktivität dadurch nicht beeinträchtigt wird und andererseits auch hier die Auswahl wiederum durch höherstufige Metapläne gesteuert werden kann.

Systeme

Die ursprüngliche Implementierung mit dem Namen PRS [Georgeff & Ingrand 1989] erfolgte in LISP. Sie war vor allem als Kontrollarchitektur für Einzelagenten gedacht, wurde aber auch in einer Anwendung mit mehreren Agenten eingesetzt, die durch Nachrichtenaustausch interagierten [Ingrand et al. 1992].

UM-PRS [Huber et al. 1995] ist eine geringfügig abweichende Implementierung in C++, die an der Universität von Michigan realisiert wurde. Sie enthielt neben einigen Vereinfachungen die Möglichkeit zur Angabe statischer Prioritäten für Ziele und Pläne, die der Interpreter in seiner Entscheidungsstrategie berücksichtigte, die so auch ohne Metapläne zu einem gewissen Grad kontrollierbar wurde.

Die von SRI International in Common-LISP vorgenommene Implementierung PRS-CL [SRI 1997] verwendet eine erweiterte Repräsentation für Pläne namens ACT, die intern auf übliche KAs abgebildet werden, zusätzlich aber eine Plangenerierung durch Operatorverkettung erlauben. Außerdem besitzt sie eine explizite Unterstützung der Kommunikation mit anderen Agenten oder externen Systemen über Nachrichtenaustausch. Das Senden einer Nachricht geschieht bei statischer Adressierung durch eine primitive Handlung, während der Inhalt einer empfangenen Nachricht einfach als neuer Fakt in die Datenbasis aufgenommen wird. Dieser Fakt kann entweder direkt eine logische Aussage sein oder eine Nachricht bestehend aus einem Nachrichtentyp, dem Absender und dem eigentlichen Inhalt. Die Verarbeitung von Nachrichten erfolgt wie die aller Fakten durch KAs. Der Einsatz mehrerer Agenten wird dabei vorwiegend als Form der Modularisierung aufgefaßt, weshalb die Beziehungen zwischen den Agenten sehr statisch sind und keine höherstufigen Kommunikations- oder Koordinierungsmechanismen bereitgestellt werden.

Auch bei dMARS (distributed Multi-Agent Reasoning System) [d'Inverno et al. 1997], dem in C++ implementierten Nachfolger des ursprünglichen PRS, ist die Kommunikation auf statischen Nachrichtenaustausch beschränkt. Dieses System besitzt einen etwas anderen als den oben beschriebenen Interpreterzyklus, bei dem aus den Änderungen des Wissens des Agenten zunächst Ereignisse abgeleitet werden, die in einer Schlange gepuffert werden und als alleinige Grundlage der Planaktivierung dienen.

Die bislang jüngste Implementierung JAM [Huber 1999] erfolgte in Java und besitzt eine abweichende Struktur der Kontrollarchitektur. Zum einen wurden die Komponenten für Ziele und Intentionen in eine zusammengefaßt, wobei Intentionen dem jeweiligen Ziel zugeordnet sind. Zum anderen gibt es zusätzlich eine optionale,

als Observer bezeichnete Komponente, die einmal pro Interpreterzyklus die Kontrolle erhält und von der BDI-Steuerung unabhängige Aktionen ausführen kann. Die Plansprache wurde um prozedurale Kontrollbefehle erweitert und Ziele und Pläne um eine Nutzenberechnungsfunktion, über die der Interpreter im Sinne einer Nutzenmaximierung an den Entscheidungspunkten die Auswahl vornimmt, falls kein Metawissen anwendbar ist. Basierend auf der Objektserialisierung in Java bietet JAM einen Persistenzmechanismus, über den sich Agenten speichern, auf einen anderen Rechner übertragen und klonen lassen. Die Agentenmobilität kann über ein Handlungsprimitiv genutzt werden. Schließlich gibt es noch eine optionale Erweiterung für die Verwendung von Sprechakten im Format der FIPA-ACL zur Kommunikation.

Bewertung

Die große Stärke aller Versionen von PRS liegt in der Kombination von effizienter prozeduraler Verhaltenssteuerung und einem einfachen Interpreterzyklus mit einer flexiblen Kontrolle durch Metawissen. Dadurch läßt sich eine ausgewogene Balance zwischen Reaktivität und Zielgerichtetheit herstellen, wie sie für den primären Einsatzbereich dieses Systems als autonome Kontrolleinheiten unter Echtzeitbedingungen benötigt werden. Das BDI-Modell bietet zudem eine geeignete Abstraktionsebene zur Verhaltensbeschreibung und zur Formulierung von Metawissen.

Die Unterstützung von Agentengesellschaften ist jedoch stark begrenzt auf statische Beziehungen zwischen Agenten und einen simplen Nachrichtenmechanismus. Es gibt weder höherstufige Interaktionsmechanismen oder eine echte Integration der Kommunikation in die Architektur, noch Infrastrukturmechanismen zur Organisation von dynamischen Agentengesellschaften mit veränderlichen Interaktionsbeziehungen.

2.6.2. INTERRAP

Die Agentenarchitektur INTERRAP (**Inte**gration of **R**eactive Behaviour and **Ra**tional **P**lanning) [Fischer et al. 1994, Müller 1996, Jung & Fischer 1998] ist ein konzeptionelles Modell für den Aufbau von Agenten, das reaktives, deliberatives und interaktives Verhalten kombiniert. Dazu ist sie funktional gegliedert in drei hierarchische Ebenen, die alle einen gleichartigen strukturellen Aufbau besitzen, der jeweils entsprechend der Funktion der Ebene instantiiert ist. Diese Hierarchie von Ebenen dient der Abstraktion hinsichtlich des Wissens wie auch der Kontrolle, so daß höhere Ebenen dank dieser Abstraktion komplexere Funktionen erfüllen können, während niederere Ebenen aufgrund ihrer Einfachheit zeitkritisch arbeiten und reagieren können. Die Parallelität von einfachen, effizienten Ebenen hin zu komplexen, leistungsstarken eröffnet so ein weites Verhaltensspektrum mit einer klaren hierarchischen Kontrollstruktur.

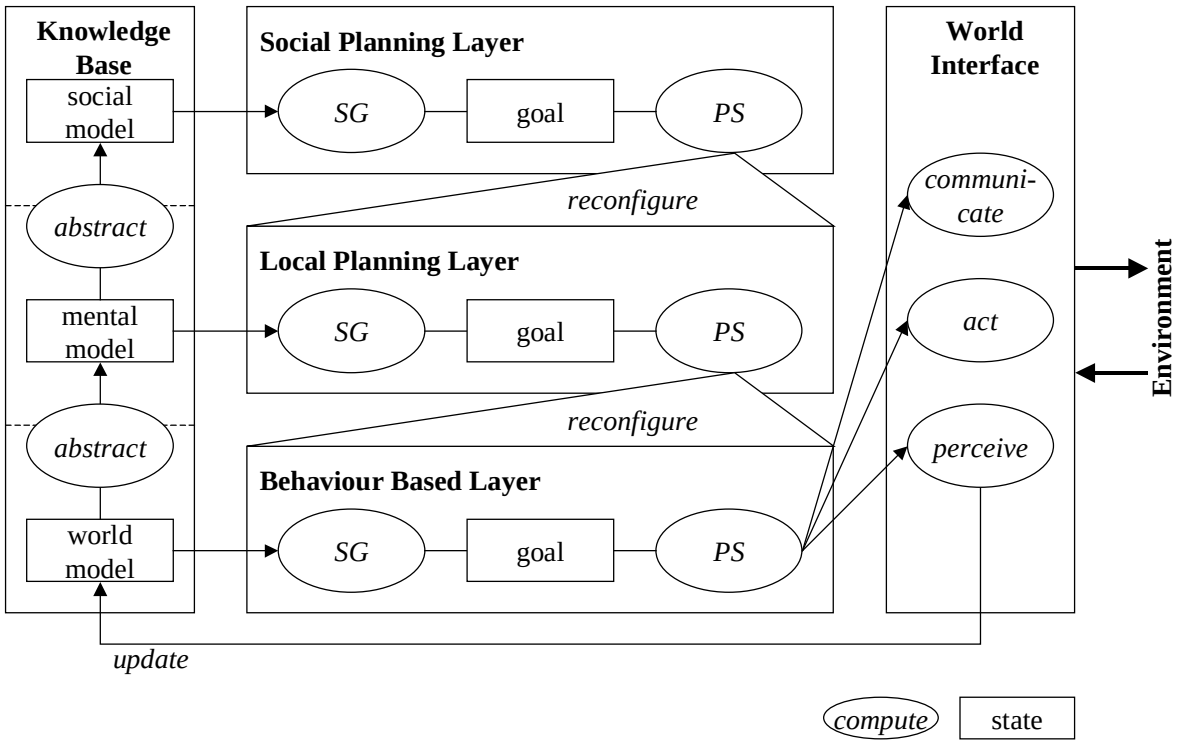

Abbildung 4: Kontrollarchitektur von INTERRAP (aus [Jung & Fischer 1998])

Architektur

Die Kontrollarchitektur von INTERRAP (Abbildung 4) ist zunächst gegliedert in die drei funktionalen Module Wissensbasis, Kontrolleinheit und Weltschnittstelle. Die Weltschnittstelle bildet die Verbindung zur Umgebung durch Wahrnehmen, Handeln und Kommunizieren. Wissensbasis und Kontrolleinheit sind jeweils in die drei genannten Abstraktionsebenen unterteilt.

Angewandt auf die Wissensbasis ergeben die Ebenen eine Unterteilung in ein Weltmodell, das faktisches Wissen über die Welt und Fähigkeiten zur Änderung der Welt enthält, ein mentales Modell, das Selbstwissen in Form der Ziele und Intentionen des Agenten beinhaltet, und ein soziales Modell, das Wissen über andere Agenten und über Interaktionen mit diesen umfaßt. Bezogen auf die Kontrolleinheit sind die verhaltensbasierte Ebene für reaktives und schematisches Verhalten, die lokale Planungsebene für Einzelagentenplanung und die soziale Planungsebene für Multiagentenplanung definiert. Eine Ebene der Kontrolleinheit operiert dabei nur auf dem Inhalt der korrespondierenden Ebene der Wissensbasis und den darunterliegenden, was die Informationsmenge und damit die Komplexität der unteren Ebenen verringert.

Die Ebenen der Kontrolleinheit besitzen eine gemeinsame Struktur, die sich aus dem BDI-Ansatz (vgl. Kapitel 2.3.2) herleitet. Anhand des aktuellen Wissens wird dabei zunächst festgestellt, welche Ziele der Agent in der gegebenen Situation verfolgen soll (SG: Situation recognition / Goal activation). Anschließend wird

entschieden, auf welche Weise die momentanen Ziele verfolgt werden sollen, und die dabei aufgestellten Intentionen werden durch Handlungen umgesetzt (PS: Planning / Scheduling).

Die Interaktion zwischen den Kontrollebenen gehorcht dem hierarchischen Prinzip und erfolgt anhand der Kompetenz der Ebenen. Eine Aufgabe wird von unten nach oben weitergeleitet, wenn eine Ebene ein Ziel besitzt, dessen Erreichen außerhalb ihrer Möglichkeiten steht. Da höhere Ebenen aufgrund der zunehmenden Abstraktion komplexere Aufgaben bewältigen können, wird so ein Ziel jeweils durch die unterste dafür kompetente Ebene bearbeitet. Die Ausführung hingegen wird von oben nach unten durchgereicht, indem jede Ebene zum Erreichen ihrer Ziele auf die operationalen Primitive der darunterliegenden Ebene zurückgreift. Zudem kontrolliert jede Ebene die darunterliegende und kann diese umkonfigurieren, so daß nicht nur einzelne Handlungen, sondern auch Handlungsweisen und Strategien der unteren Ebene beeinflußt werden können.

Nur die unterste, reaktive Ebene besitzt direkt ausführbare Handlungsprimitive, die sie über die Weltschnittstelle in konkrete Handlungen umsetzen kann. Umgekehrt aktualisiert die Wahrnehmung nur die unterste Ebene der Wissensbasis, das Weltmodell. Die direkte Interaktion mit der Umgebung über die Weltschnittstelle ist somit der untersten Ebene vorbehalten, während die übrigen Ebenen nur indirekt über die darunterliegenden Ebenen in Kontakt mit der Umgebung stehen.

Bewertung

Eine Bewertung von INTERRAP ist insofern schwierig, als es sich um ein abstraktes Modell handelt und über die verschiedenen vorgenommenen Implementierungen nur wenige Details veröffentlicht sind. Die hierarchische Aufteilung in eine reaktive, lokale und soziale Ebene erlaubt eine Spezialisierung auf den jeweiligen Aufgabenbereich, bringt aber auch das Problem der Koordinierung der Ebenen mit sich. Aufgrund der Ebenenhierarchie muß jede Ebene in der Lage sein, die Aktivitäten der direkt darunterliegenden auf eine kohärente Weise zu kontrollieren, ohne deren Aufgabe selbst zu übernehmen.

Für die Gesamtfunktionalität entscheidend ist die konkrete Realisierung der Kontrollmechanismen der einzelnen Ebenen. Die einheitliche abstrakte Kontrollstruktur der Ebenen erleichtert die Analyse des Systemverhaltens und erlaubt einheitliche Koordinierungsmechanismen zwischen den Ebenen, läßt aber notwendigerweise viel Spielraum zur Umsetzung relativ zu den Aufgaben der einzelnen Ebenen. Für die reaktive Ebene werden dabei wie bei PRS vordefinierte Handlungsabläufe und ein einfacher Interpreter verwendet. Die Planungsebenen verwenden Planungsmechanismen zur zielgerichteten Verhaltenskontrolle.

Eine konkrete Infrastruktur für Multiagentengesellschaften ist nicht Teil von INTERRAP. Unterstützt wird die Interaktion zwischen Agenten durch die soziale

Planungsebene in Form von Multiagentenplanung und der Durchführung von Protokollen.

2.6.3. ZEUS

Das ZEUS Agent Building Toolkit [Nwana et al. 1999] ist ein integrierendes System zur Erstellung und Ausführung von Multiagentensystemen. Es besteht aus einer Agentenarchitektur (Agent Component Library), einer Entwicklungsumgebung (Agent Building Software) inklusive eigener Entwicklungsmethodologie und einer Laufzeitumgebung (Utility Agents). Implementierungssprache der Architektur, der Werkzeuge und spezifischer Agentenfunktionalitäten ist Java.

Agent Component Library

ZEUS-Agenten besitzen eine einheitliche Kontrollarchitektur (Abbildung 5), die funktional in Komponenten gegliedert ist. Diese realisiert den anwendungsunabhängigen Teil eines Agenten mit generellen Mechanismen zur Steuerung seines autonomen und interaktiven Handelns. Das Kontrollschema orientiert sich an planbasierten, deliberativen Ansätzen (vgl. Kapitel 2.3.2).

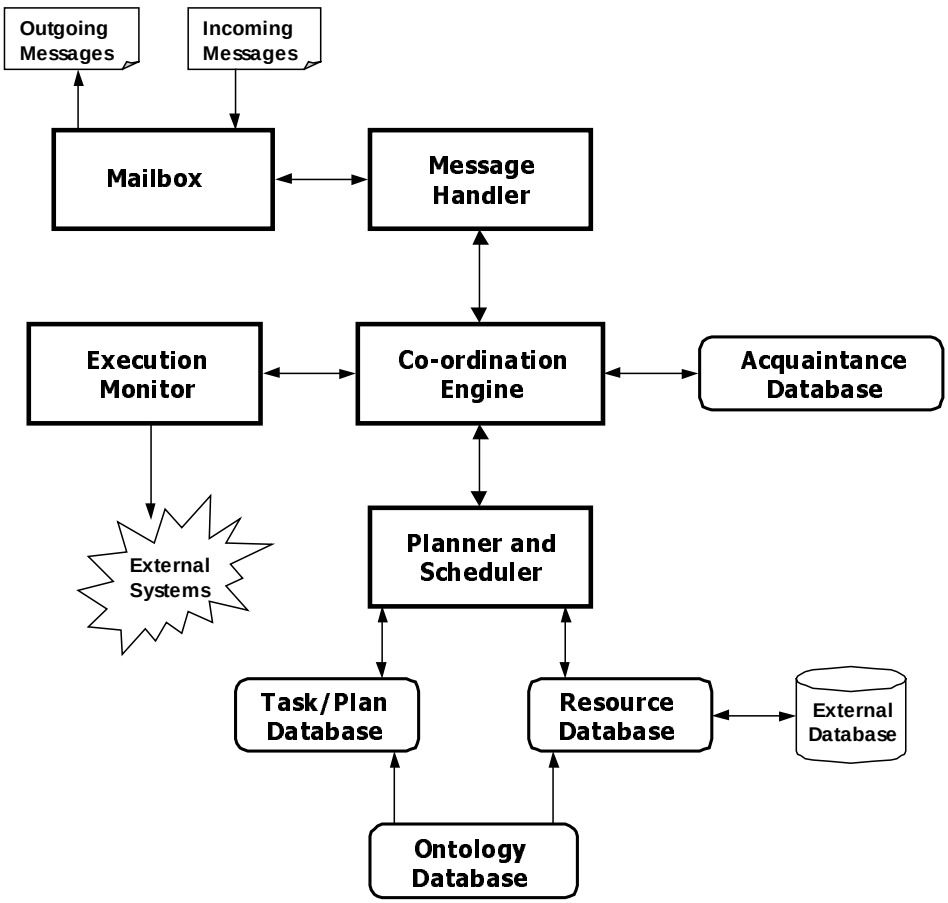

Abbildung 5: Kontrollarchitektur von ZEUS (aus [Nwana et al. 1999])

Die Basis dieses Handelns bildet das Wissen des Agenten, das deklarative, prozedurale und interaktive Aspekte umfaßt. Grundlage allen Wissens bilden domänenspezifische Terminologien, die über Begriffsschemata in Ontologien definiert sind. Die Fakten (auch als Ressourcen bezeichnet) werden in einer Frame-basierten Wissensrepräsentationssprache als Instanzen dieser Schemata formuliert. Die Fähigkeiten eines Agenten werden logisch über Vorbedingungen und Effekte sowie über Kosten und Dauer der Ausführung definiert. Unterschieden wird dabei zwischen primitiven, direkt ausführbaren Handlungen und Plänen aus abstrakten Planschritten. Schließlich gibt es zur Steuerung der Interaktionen noch Koordinierungsprotokolle und Organisationsbeziehungen zwischen Agenten. Enthalten ist das Wissen in der *Ontology Database*, der *Resource Database* und der *Task/Plan Database*. Zudem gibt es die *Acquaintance Database* mit Wissen über die Fähigkeiten anderer Agenten und die Beziehungen zu diesen.

Die Kommunikation zwischen den Agenten verwendet eine direkte, asynchrone Nachrichtenzustellung über die *Mailbox*, die in KQML[13] ausgedrückte Sprechakte empfängt und versendet. Der *Message Handler* interpretiert eingehende Nachrichten aus der Mailbox und leitet sie an die jeweils für die Verarbeitung zuständige Komponente weiter. Aufgabe der *Co-ordination Engine* ist einerseits die Organisation der Interaktion mit anderen Agenten anhand von Koordinierungsprotokollen und Interaktionsstrategien sowie andererseits die Entscheidung, welche der bestehenden Ziele wann verfolgt bzw. aufgegeben werden. Für die ausgewählten Ziele erstellt der *Planner and Scheduler* Pläne aus den verfügbaren Aufgabenspezifikationen und koordiniert diese unter Berücksichtigung von Zeit, Kosten und Ressourcen. Ausgeführt werden die Pläne zeitgesteuert durch den *Execution Monitor*, der auch die Ausführung überwacht und schließlich deren Ergebnis zurückmeldet. Die eigentliche Ausführung geschieht durch externe Komponenten.

Agent Building Software

Die Erstellung von Agentensystemen mit ZEUS wird durch ein Vorgehensmodell und durch Werkzeuge unterstützt. Das Vorgehensmodell beginnt mit einer *Domänenanalyse*, in der mögliche Agenten identifiziert und die grundlegenden Kategorien der Domäne in Ontologien spezifiziert werden. Die *Agentendefinition* legt die Aufgaben der Agenten und die ihnen initial zur Verfügung stehenden Ressourcen fest, während die *Aufgabendefinition* diese Aufgaben genauer spezifiziert. In der *Organisationsstruktur* sind die Bekanntschaftsverhältnisse zwischen den Agenten festgelegt und im Zuge der *Agentenkoordination* werden Interaktionsprotokolle zur Durchführung von Interaktionen ausgewählt. Schließlich erfolgt die *Kodegenerierung und Implementierung*, bei der aus den Spezifikationen Kode generiert wird und die primitiven Handlungen implementiert werden.

---

[13] Gemäß dem ZEUS Technical Manual von 1999 wird nunmehr FIPA-ACL verwendet.

Dieser Softwareentwicklungsprozeß wird im Rahmen des ZEUS-Toolkits durch visuelle Editoren unterstützt, mit denen die Spezifikationen von Ontologien, Fakten, Agentendefinitionen, Aufgabenbeschreibungen, Organisationsstrukturen und Koordinierungsprotokollen erstellt werden. Diese Editoren bilden zusammen mit einem Kodegenerator zur Übersetzung in ausführbaren Programmkode die Entwicklungsumgebung.

Utility Agents

Die Laufzeitumgebung wird durch ZEUS-Agenten realisiert, die Infrastruktur- und Überwachungsfunktionalitäten bereitstellen. Die Infrastruktur bietet Dienste zur Unterstützung der Interaktion zwischen Agenten. Sie besteht aus dem *Name Server*, der Namen und Kommunikationsadressen der Agenten verwaltet, und dem *Facilitator*, der Informationen über die Dienste von Agenten vermittelt.

Der Analyse und Kontrolle von laufenden Agentensystemen dient der *Visualiser Agent*. Er sammelt Laufzeitinformationen anderer Agenten, die durch verschiedene Werkzeuge analysiert werden können. Diese umfassen die Darstellung von Agentengesellschaften und deren Interaktionen sowie die Bearbeitung und Verteilung von Aufgaben. Zudem können die internen Zustände einzelner Agenten angezeigt und verfolgt sowie die Agenten zur Laufzeit umkonfiguriert und auch beendet werden.

Bewertung

Das ZEUS-System bietet einen sehr pragmatischen Ansatz zur Realisierung von Multiagentenanwendungen. Es deckt sämtliche Aspekte von Multiagentensystemen ab beginnend bei einem werkzeuggestützten Vorgehensmodell zur Agentenentwicklung, Sprachen zur Spezifikation des Wissens und des Verhaltens von Agenten, eine generische Kontrollarchitektur für zielgerichtetes und interaktives Verhalten, eine Infrastruktur für dynamische Agentengesellschaften und schließlich die Kontrolle und Überwachung von laufenden Systemen. Alle diese Bereiche bilden ein integriertes Komplettsystem mit umfassender Funktionalität.

Die Kontrollarchitektur ist beschränkt auf traditionelle generative Planung, wobei das einzige reaktive Element die Verwendung abstrakter Pläne darstellt. Dafür sind Kommunikation und Interaktion ein integraler Bestandteil der Kontrollarchitektur und werden durch die Dienste von Infrastrukturagenten unterstützt. Ontologien und eine wissensbasierte Verhaltenssteuerung bieten eine geeignete Abstraktionsebene sowohl zur Agentenspezifikation, als auch zur Interaktion zwischen Agenten.

2.6.4. AgentBuilder

Der Ansatz von AgentBuilder [Reticular Systems 1999] nimmt basierend auf Agent-0 [Shoham 1993] und dessen Weiterentwicklung PLACA (PLAnning Communicating

Agents) [Thomas 1994] eine strikte Trennung vor zwischen einer abstrakten Spezifikation eines Agenten auf einer „mentalen“ Ebene und einem Interpreter, der auf den Inhalten dieser Ebene operiert. Entsprechend besteht das in Java implementierte System aus einer Menge von Entwicklungswerkzeugen zur Agentenspezifikation und aus einer Laufzeitumgebung zur Ausführung von Agenten, dessen Kernstück der Interpreter ist. Im folgenden wird zunächst die Architektur bestehend aus der Agentenspezifikation und dem Interpreter und anschließend die werkzeuggestützte Entwicklungsmethodologie von AgentBuilder vorgestellt.

Architektur

Ein Agent wird im AgentBuilder-System auf einer „mentalen“ Ebene bestehend aus Faktenwissen (Belief), Fähigkeiten (Capability), Verpflichtungen (Commitment), Intentionen (Intention) und Verhaltensregeln (Behavior Rule) beschrieben. Die Fakten beinhalten das Wissen des Agenten über den gegenwärtigen Stand der Welt, wozu neben der Umgebung auch das Wissen anderer Agenten und des Agenten selbst sowie laufende Interaktionen mit anderen Agenten zählen. Die Fähigkeiten beschreiben die Handlungsmöglichkeiten eines Agenten jeweils bestehend aus einer Handlung und deren Vorbedingungen, die zur Ausführung der Handlung erfüllt sein müssen. Es wird unterschieden zwischen privaten Handlungen, die ein Agent alleine ausführt und die in der Regel der Veränderung seiner Umgebung dienen, und kommunikativen Handlungen, bei denen eine Nachricht an einen anderen Agenten geschickt wird.

Gesteuert wird das Verhalten eines Agenten über reaktive Regeln bestehend aus einem Ereignis, einer Bedingung und einer Aktion. Die Aktion wird ausgeführt, sobald das Ereignis eintritt und falls die Bedingung erfüllt ist. Das Ereignis beschreibt Änderungen in der Umgebung inklusive dem Empfang von Nachrichten, während die Bedingung Voraussetzungen im Faktenwissen des Agenten beinhaltet. Als Aktion ist neben den Fähigkeiten des Agenten auch eine Wissensänderung möglich. Die Zielsetzungen eines Agenten werden über Verpflichtungen und Intentionen ausgedrückt, die auch gegenüber einem anderen Agenten bestehen können. Eine Verpflichtung drückt eine Absicht aus, eine bestimmte Handlung zu einem festgelegten Zeitpunkt auszuführen. Eine Intention hingegen ist eine Absicht, einen bestimmten Weltzustand herbeizuführen[14].

Formuliert werden alle diese Wissensarten in der Reticular Agent Definition Language (RADL), die logische Ausdrücke mit einer typisierten objektorientierten Repräsentation von Fakten kombiniert. Objektschemata werden in Ontologien

[14] Die Terminologie von AgentBuilder weicht hier von der bei den meisten anderen Architekturen inklusive CASA verwendeten BDI-Terminologie ab. Eine Intention bei AgentBuilder entspricht einem Ziel gemäß BDI, während eine Intention gemäß BDI einer Verpflichtung bei AgentBuilder entspricht. Der Unterschied besteht jedoch nur in der Wortwahl, zumal der Aspekt der Verpflichtung bei Intentionen in der BDI-Theorie eine wichtige Rolle spielt.

definiert. Als Format für die Nachrichten zwischen Agenten wird KQML verwendet. Konversationsprotokolle werden nur implizit unterstützt.

Der als Agent Engine bezeichnete Interpreter realisiert nun das Verhalten eines Agenten ausgehend von dem jeweils vorhandenen Wissen nach einem festen Kontrollzyklus. Zunächst werden neu eingetroffene Nachrichten verarbeitet, wozu der Sender identifiziert und dessen Autorisierung überprüft wird. Ist der Sender autorisiert, so wird die Nachricht zu dem Faktenwissen des Agenten hinzugefügt. Es folgt die regelbasierte Steuerung, bei der die Menge der erfüllten Regeln bestimmt wird und dann deren Aktionen ausgeführt werden. Zudem werden zeitgesteuert Verpflichtungen ausgeführt, sofern die Bedingung der enthaltenen Fähigkeit erfüllt ist. Anschließend wird das Wissen aufgrund der ausgeführten Handlungen aktualisiert. Optional kann ein Planungsmechanismus in die Agent Engine eingebunden werden, der Planung zum Erreichen von Intentionen durchführt.

Entwicklungsprozeß und -werkzeuge

Die Entwicklung eines Agentensystems mit AgentBuilder umfaßt zwei Ebenen, die Multiagentenebene und die Einzelagentenebene. Auf der Multiagentenebene werden Rollen und Funktionen der verschiedenen Agenten identifiziert, während auf der Einzelagentenebene das Verhalten der einzelnen Agenten über deren initiales Wissen festgelegt wird.

Wie der Name AgentBuilder bereits nahelegt, ist der Entwicklungsprozeß ein integraler Bestandteil dieses Systems, für den ein durch eine ganze Werkzeugfamilie unterstütztes Vorgehensmodell vorgegeben ist. Die Organisation der für eine Anwendung spezifizierten Ontologien, Wissenselemente und Agenten geschieht durch einen Projektmanager. Dieser verwaltet mehrere Projekte in einer gemeinsamen Bibliothek, um die Wiederverwendung der Spezifikationen für andere Anwendungen zu erleichtern.

Der Entwicklungsprozeß beginnt mit der Bereichsanalyse, in der die Struktur der Anwendungsdomäne auf ein Objektmodell aus Objekten und deren Operationen abgebildet wird. Unterstützt wird dies durch den Ontologiemanager, mit dem sich Begriffe und ihre Relationen untereinander visuell spezifizieren und nach RADL übersetzen lassen. Der nächste Schritt ist die Definition der Agentengesellschaft, wozu die Bereichsanalyse auf Agenten und ihre Funktionalitäten übertragen wird und die Abhängigkeiten der Agenten untereinander festgelegt werden. Für die Interaktionen zwischen Agenten lassen sich mit dem Protokollmanager Protokolle relativ zu Rollen definieren, die dann durch Verhaltensregeln zu realisieren sind, die den Protokollen gemäß Nachrichten verschicken und empfangene Nachrichten verarbeiten. Es folgt die Spezifikation der einzelnen Agenten mit Hilfe des Agentenmanagers. Dazu wird einerseits unter Verwendung von RADL das initiale Wissen der einzelnen Agenten in den unterschiedlichen Wissensarten spezifiziert, andererseits die Umsetzung von privaten Handlungen durch Methodenaufrufe in

Java festgelegt. Dabei erfolgt auch die Anbindung an externe Systeme und Benutzeroberflächen über Java-Schnittstellen.

Schließlich werden die Agenten erzeugt, indem pro Agent dessen initiales Wissen in eine Instanz der Agent Engine geladen und diese gestartet wird. Dies wird durch den Agency-Manager unterstützt, mit dem sämtliche zu einer Anwendung gehörenden Agenten gestartet werden. Außerdem stellt der Agency-Manager Funktionalitäten zur Visualisierung und zur Fehlersuche bereit, wozu der Inhalt kommunizierter Nachrichten und das Wissen der einzelnen Agenten angezeigt und die Ausführung der Agenten kontrolliert werden kann.

Bewertung

Wie ZEUS bildet auch das AgentBuilder-System ein praxisorientiertes Komplettsystem zur Erstellung von Agentensystemen, indem ein werkzeuggestützter Entwicklungsprozeß, eine Spezifikationssprache, eine Kontrollarchitektur und eine Laufzeitumgebung bereitgestellt werden.

Die Kontrollarchitektur des AgentBuilder ist regelbasiert und damit eher reaktiv ausgerichtet, eine zielgerichtete generative Planung ist optional. Die Interaktionen zwischen Agenten basieren auf dem Konzept der Nutzung der Fähigkeiten anderer Agenten, wozu auf Anfrage eine Verpflichtung zur Durchführung einer Handlung eingegangen wird. Protokolle dienen lediglich der Strukturierung von Interaktionen während der Implementierung, explizites Befolgen von Protokollen oder eine architekturgestützte Erstellung und Verarbeitung von Nachrichten gibt es nicht. Infrastrukturfunktionalitäten für dynamische Interaktionen zwischen Agenten fehlen.

2.6.5. Weitere Systeme

Im folgenden werden einige Beispiele für Agentensysteme beschrieben, die keine umfassende Kontrollstruktur für einzelne Agenten vorgeben. JADE bietet eine FIPA-konforme Infrastruktur für Multiagentensysteme, während die von Grasshopper bereitgestellte Infrastruktur vor allem der Verwendung mobiler Agenten dient. Bond integriert ein Middleware-System mit Agenten.

JADE

Der Schwerpunkt von JADE (Java Agent Development Environment) [Bellifemine et al. 1999] liegt bei der FIPA-Konformität[15]. JADE stellt umfassende Infrastrukturfunktionalitäten für Agentengesellschaften bereit, besitzt aber nur rudimentäre Kontrollmechanismen für einzelne Agenten und deren Interaktionen.

[15] In [Bellifemine et al. 1999] ist es noch die FIPA-Spezifikation von 1997, mittlerweile ist JADE kompatibel zu FIPA 2000

Entsprechend der FIPA-Management-Spezifikation (vgl. Kapitel 2.4.4) ist die Infrastruktur in Agentenplattformen aufgeteilt, die jeweils die Agenten für AMS, DF und ACC besitzen. Eine Plattform kann verteilt sein, wobei das Agent Platform Front-End die genannten Infrastrukturagenten beherbergt, während zusätzliche Agent Container über einen Kommunikationskanal mit diesem verbunden sind. Die Kommunikation zwischen Agenten verwendet FIPA-ACL-Sprechakte, die die Agenten als Java-Objekte verschicken und empfangen. Zur Überwachung und Verwaltung von Plattformen und ihren Agenten gibt es einen besonderen Remote Monitoring Agent.

Um einen Agenten zu implementieren, wird die Java-Klasse Agent abgeleitet, die dessen Einbindung in die Plattform inklusive dem Senden und Empfangen von Sprechakten realisiert. Jedem Agenten ist ein Java-Thread zugeordnet, der über ein einfaches internes Scheduling mehrere Behaviours eines Agenten ausführt. Ein Behaviour ist eine Java-Klasse mit einer action-Methode, die wiederholt ausgeführt wird, bis das Behaviour beendet ist. Einige Behaviours beispielsweise zum Senden und Empfangen von Sprechakten sind vorgegeben.

Grasshopper

Die Ausrichtung von Grasshopper [Bäumer et al. 1999] konzentriert sich auf mobile Agenten. Das System ist konform zu den Standards OMG-MASIF und FIPA'97.

Die Topologie einer sogenannten Distributed Agent Environment (DAE) von Grasshopper umfaßt Regionen, Agenturen und Plätze. Eine Region bildet den Rahmen für mehrere Agenturen und unterstützt Managementdienste zur Verwaltung von Agenturen und Plätzen. Eine Agentur stellt die Laufzeitumgebung für Agenten bereit und beinhaltet die Kernagentur und einen oder mehrere Plätze. Ein Platz ist eine funktionale Gruppierung von Agenten innerhalb einer Agentur.

Die Kernagentur realisiert die Infrastrukturfunktionalitäten der Agentur durch eine Menge von Diensten. Die *Kommunikation* ermöglicht Interaktionen zwischen Grasshopper-Komponenten, wozu neben den Agenten auch die unterschiedlichen Infrastrukturkomponenten zählen. Dieser Dienst umfaßt eine ortsunabhängige Kommunikation, den Transport mobiler Agenten zwischen zwei Agenturen und die Lokalisierung von Agenten. Bei der *Registratur* werden sämtliche Agenten und Plätze einer Agentur angemeldet. *Sicherheitsdienste* schützen die Kommunikation und regeln den Zugriff der Agenten untereinander. Die *Persistenz* erlaubt das Speichern und erneute Starten von Agenten und Plätzen. Zur *Administration* des Systems wird die Überwachung und Kontrolle der Agenten und Plätze sowie der Agentur selbst ermöglicht.

Die Unterstützung der FIPA-Konformität ist optional und durch eine reine Erweiterung realisiert. Diese umfaßt einerseits stationäre Agenten für AMS, DF und ACC auf einem eigenen Platz, deren Funktionalität durch die vorhandenen Verwaltungs-, Registratur- und Kommunikationsdienste von Grasshopper umgesetzt wird.

Andererseits gibt es mit FIPA-ACL, FIPA-SL1 und XML Unterstützung für Sprachen zur Sprechaktkommunikation.

Bond

Die Bond-Architektur [Bölöni & Marinescu 2000] besteht aus einem Middleware-System für verteilte Objekte und einer Erweiterung um Agenten als Objekte mit besonderem Aufbau und Kontrollmechanismus.

Bond-Middleware

Die Grundeinheit des Middleware-Systems von Bond sind Objekte im üblichen objektorientierten Sinn, also eine Menge von Feldern und Methoden. Zusätzlich besitzt jedes Objekt eine Menge an dynamischen Eigenschaften in Form von Attribut/Wert-Paaren und unterstützt Persistenz. Unterschieden werden passive und aktive Objekte, wobei letztere einen eigenen Ausführungs-Thread besitzen. Schließlich können Objekte über Nachrichten kommunizieren, wozu sie eine ID als Adresse besitzen.

Damit die Middleware-Infrastruktur Objekte wie Agenten gleichermaßen unterstützt, geschieht sämtliche Kommunikation auch zwischen Objekten über die Agentenkommunikationssprache KQML. Um die Verarbeitbarkeit empfangener Nachrichten durch Objekte sicherzustellen, besitzt ein Objekt jeweils eine Menge an sogenannten Unterprotokollen, die dessen Kommunikationsfähigkeiten festlegen. Ein Unterprotokoll ist eine feste Menge an Nachrichten, die ein das Unterprotokoll realisierendes Objekt verarbeiten können muß. Eine Kommunikation zwischen Objekten ist nur auf der Basis eines gemeinsamen Unterprotokolls möglich.

Ein Objekt befindet sich immer in einer sogenannten Residenz, die die Laufzeitumgebung für eine Menge von Objekten und Threads darstellt. Eine Residenz stellt den Nachrichtentransport (auch zu anderen Residenzen) zur Verfügung und ein lokales Verzeichnis, in dem alle (dauerhaften) Objekte registriert sind. Als globale Infrastruktur gibt es Server für ein globales Verzeichnis, zur persistenten Speicherung von Objekten, als Systemmonitor zur Überwachung des Systems und andere zentrale Dienste, beispielsweise für Sicherheitsfunktionalitäten.

Bond-Agenten

Ein Agent in Bond ist nun ein spezielles Objekt bzw. eine Sammlung von Objekten, so daß die vorhandenen Kommunikationsmittel genutzt werden können. Ein Agent setzt sich aus einem Weltmodell, einer Agenda, einem endlichen Automaten und einer Menge von Strategien zusammen. Das *Weltmodell* ist ein unstrukturierter Behälter mit Informationen des Agenten über seine Umgebung in einem beliebigen Format.

Das Verhalten eines Agenten wird durch einen *endlichen Automaten* bestimmt, dessen jeweiliger Zustand die zu verfolgende Strategie festlegt. Eine *Strategie* besteht aus einzelnen Aktionen, von denen nacheinander jeweils eine ausgeführt wird, solange sich der Agent in dem assoziierten Zustand befindet. Die Zustandsübergänge des Automaten erfolgen entweder intern durch die Strategien gesteuert oder extern durch den Empfang einer Nachricht. Das Ziel eines Agenten ist über dessen *Agenda* definiert in Form einer Erfülltheitsfunktion, die das Erreichen des Ziels bestimmt, und einer Distanzfunktion, die den Abstand zwischen Ziel und Weltmodell beurteilt. Ein Agent wird terminiert, sobald sein Ziel erreicht ist, außer es handelt sich um ein kontinuierliches Ziel.

Bewertung

JADE, Grasshopper und Bond besitzen keine oder nur rudimentäre Ansätze zur Verhaltenssteuerung der einzelnen Agenten. Entsprechend bieten sie für Interaktionen zwischen Agenten lediglich eine Infrastruktur, aber keine Integration von Interaktionsmechanismen in die Verhaltenskontrolle. Alle drei Systeme bieten umfassende Infrastrukturfunktionalitäten, deren Nutzung jedoch weitgehend durch konventionelle Programmierung erfolgt. So gesehen fallen diese Systeme eher in den Bereich der Middleware als in den der Multiagentensysteme.

3. Die Komponentenarchitektur

In diesem Kapitel wird die Grundarchitektur von CASA-Agenten beschrieben, die das Zusammensetzen eines Agenten aus einzelnen Komponenten ermöglicht. Dies beinhaltet zum einen die gemeinsame Basisschnittstelle aller Komponenten, über die sie sich in einen Agenten integrieren lassen, und zum anderen einige besondere Komponenten, die den Agenten und seine Komponenten verwalten und dessen interne Infrastruktur bereitstellen. Zuvor wird die Grundkonzeption der Architektur vorgestellt, um die Motivation für deren Gestaltung herzuleiten sowie um die gegenseitigen Abhängigkeiten zwischen Infrastrukturkomponenten und der generischen Komponentenfunktionalität darzulegen.

3.1. Konzeption

Um einen hohen Grad an Modularität im Aufbau der Agenten zu erreichen, verfolgt die CASA-Architektur einen Komponentenansatz, der ein freies Zusammensetzen eines Agenten aus einzelnen Komponenten je nach Bedarf und Aufgabe ermöglicht. Diese Offenheit in der Struktur eines Agenten ist nicht nur in der Entwurfsphase, sondern auch noch zur Laufzeit gewährleistet, indem Komponenten auch bei einem laufenden Agenten praktisch übergangslos neu hinzugefügt, entfernt oder ausgetauscht werden können.

Zur Verwaltung der Komponentenstruktur eines Agenten gibt es eine eigene Komponente, als Agentenkern bezeichnet, die neben der Zusammensetzung des Agenten auch dessen internen Zustand und Konfiguration kontrolliert. Außerdem dient der Agentenkern als Schnittstelle für den direkten Zugriff auf den Agenten, beispielsweise zu Managementzwecken.

Wegen der Austauschbarkeit von Komponenten müssen diese hinreichend voneinander entkoppelt sein, weshalb deren Abhängigkeiten untereinander durch Rollen beschrieben werden, die die Komponenten innerhalb eines Agenten einnehmen. Eine Rolle spezifiziert dabei eine Schnittstelle zur Interaktion und Konfiguration für eine Komponente. Darüber hinaus besitzen alle Komponenten eine einheitliche Schnittstelle zur Interaktion mit den Infrastrukturkomponenten.

Die Interaktionen der Komponenten untereinander basieren auf Nachrichten, die vermittels einer Infrastrukturkomponente, der Nachrichtenzustellung, ausgetauscht werden können. Die Adressierung der Nachrichten geschieht anhand der Rollen, so daß an eine Komponente Informationen übermittelt und Aufträge erteilt werden können, ohne daß mehr als die jeweilige Rolle bekannt zu sein braucht. Nachrichten besitzen dabei einen Typ, der ihren Inhalt und die vorgesehene Art der Verarbeitung festlegt.

Jede Komponente speichert die empfangenen Nachrichten in einer Warteschlange. Auch für die Verarbeitung der wartenden Nachrichten stellt die Architektur die nötigen Kontrollmechanismen bereit. Dazu gehört eine Komponente, die einen Kontrollzyklus realisiert, der die Verarbeitungsressourcen zur Nachrichtenverarbeitung für mehrere Komponenten bereitstellt.

Die drei Komponenten Agentenkern, Nachrichtenzustellung und Kontrollzyklus bilden zusammen die Agentenhülle, über die die agenteninterne Infrastruktur zur Verwaltung von Komponenten, für Interaktionen zwischen diesen und zur Ablaufkontrolle von deren Aktivitäten realisiert ist. Sie sind die einzigen Komponenten, die jeder CASA-Agent besitzen muß. Abbildung 6 veranschaulicht das Baukastenprinzip, nach dem sich ein Agent vermittels der Agentenhülle und den Basisschnittstellen aus Komponenten zusammensetzen läßt. Bevor die Mechanismen der Komponenten der Agentenhülle im folgenden genauer beschrieben werden, werden noch das Konzept der Komponentenrolle sowie die gemeinsame Basisschnittstelle aller Komponenten dargestellt, auf denen diese Mechanismen aufbauen.

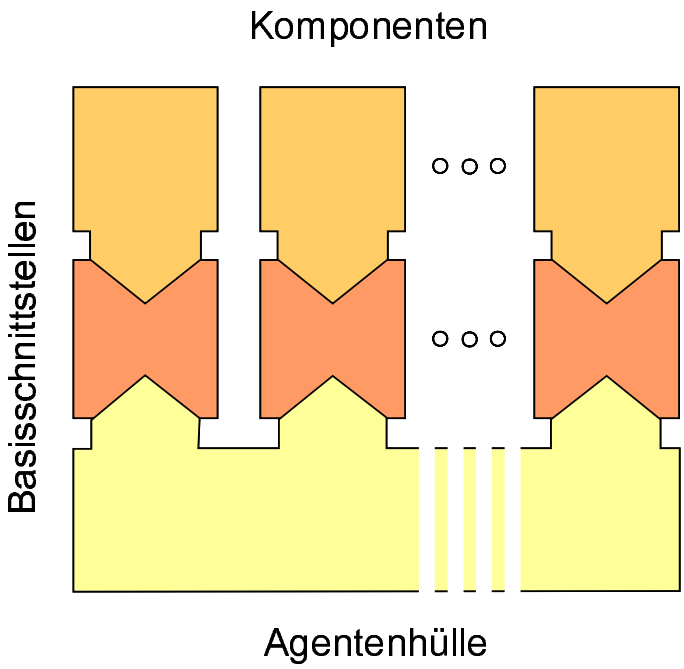

Abbildung 6: Aufbau eines Agenten aus Komponenten

3.2. Komponentenrollen

Komponentenrollen dienen der Identifikation der Komponenten innerhalb des Agenten und beschreiben jeweils deren spezifische Funktionalität und Schnittstelle. Über Rollengruppen werden verschiedene Rollen mit übereinstimmender Schnittstelle und Funktionalität zusammengefaßt. Jede Komponente besitzt eine feste Menge an Rollen, die sie innerhalb des Agenten wahrnimmt und über die sie in Bezug auf andere Komponenten charakterisiert ist. Da Komponenten sich untereinander ausschließlich anhand dieser Rollen identifizieren können, darf in einem Agenten eine bestimmte Rolle jeweils nur von höchstens einer Komponente eingenommen werden.

In der CASA-Architektur erfüllen Komponentenrollen eine ähnliche Funktion wie Schnittstellenspezifikationen anderer Komponentenansätze (vgl. Kapitel 2.2.3). Ebenso wie eine Komponentenrolle abstrakt ein Interaktionsverhalten vorgibt, das eine die jeweilige Rolle ausfüllende Komponente zu realisieren hat, beschreibt beispielsweise ein Interface in Java abstrakt eine Schnittstelle, die eine das jeweilige Interface implementierende Klasse konkret umzusetzen hat. Eine über die reine Deklaration hinausgehende Beschreibung der spezifischen Funktionalität einer Schnittstelle verbleibt in beiden Fällen informal.

Schnittstellenspezifikation

Die Schnittstellenspezifikation[16] einer Rolle besteht aus der für die Konfiguration durch die Agentenhülle, der zur Interaktion über Nachrichten und schließlich aus der für den Komponentenaustausch.

Für die Konfiguration wird eine Menge von Eigenschaften angegeben, über die der Zustand der Komponente abgefragt und gegebenenfalls auch verändert werden kann. Zu jeder Eigenschaft werden der Typ, der Wertebereich und der Standardwert spezifiziert. Über diese Eigenschaften kann dann eine die Rolle ausfüllende Komponente initial wie zur Laufzeit konfiguriert und ihr aktueller Zustand abgefragt werden.

Für die Interaktion über Nachrichten gibt eine Rolle an, welche Nachrichtentypen sie empfangen und verarbeiten kann. Jedem Nachrichtentyp ist dabei eine Sendeberechtigung zugeordnet, die bestimmt, welche Rollen und Rollengruppen Nachrichten des jeweiligen Typs an diese Rolle senden dürfen. Diese Informationen werden bei der Nachrichtenzustellung berücksichtigt. Die Spezifikation der Schnittstelle für die Interaktion zwischen Komponenten ist dabei auf den Nachrichtenempfang beschränkt. Die Angabe der zur Wahrnehmung einer Rolle versendeten Nachrich-

[16] Bei der Umsetzung von Komponentenrollen in einer Implementierungssprache lassen sich nicht unbedingt alle verlangten Spezifikationen zufriedenstellend ausdrücken, so daß diese gegebenenfalls nur informal gegeben werden können.

tentypen inklusive der jeweiligen Empfängerrollen ist auf der Ebene der Komponentenrolle nicht möglich, da dies mit der jeweiligen Umsetzung der Rolle durch eine Komponente variieren kann. Bei einer Komponente ist die Angabe der Empfängerrollen jedoch unerläßlich, da die vollständige Erfüllung einer Rolle durch eine Komponente i.a. nur dann gewährleistet ist, wenn der Agent auch Komponenten zur Verarbeitung dieser Nachrichtentypen enthält.

Damit bei einem Austausch eine neue Komponente den Zustand der ersetzten Komponente mit derselben Rolle übernehmen kann, benötigt sie Zugriff auf deren Zustandsdaten. Hierzu können die Konfigurationseigenschaften gegebenenfalls noch um weitere Eigenschaften erweitert werden, die in einer analogen Weise spezifiziert werden.

Rollengruppen

Jede Rolle wird einer oder mehreren Rollengruppen zugeordnet. Eine Rollengruppe besitzt eine analoge Spezifikation zu einer Rolle, außer daß keine Komponente direkt dieser Spezifikation genügt, sondern nur über eine entsprechende Rolle, die zu dieser Gruppe gehört. Die Spezifikation der Gruppe vererbt sich dabei auf die zu ihr gehörenden Rollen, falls sie nicht durch diese explizit überschrieben wird. Bei mehreren Gruppen werden die Spezifikationen miteinander vereinigt[17]. Abgesehen von der Vererbung der Spezifikation dienen Rollengruppen der Erteilung gemeinsamer Sendeberechtigungen für Rollen mit gleichartiger Funktionalität. Rollengruppen können selbst wiederum anderen Gruppen angehören, so daß sich Hierarchien von Gruppen mit entsprechenden Vererbungsbeziehungen definieren lassen.

Alle Rollengruppen besitzen eine gemeinsame oberste Rollengruppe, die die allen Rollen gemeinsame Schnittstelle beschreibt. Dazu gehören vor allem die Eigenschaften, die durch die im nachfolgenden Kapitel beschriebene Basisschnittstelle aller Komponenten umgesetzt werden. Diese oberste Rollengruppe kann zudem als Sendeberechtigung angegeben werden, um beliebigen Komponentenrollen das Senden eines Nachrichtentyps an eine Rolle zu erlauben.

Spezifikation von Rollen und Komponenten

Definition 1: Komponentenrollen

Eine Rolle r ist ein 4-Tupel [n, G, S, E]. n ist der Name der Rolle, über den die Rolle eindeutig bezeichnet werden kann. $\emptyset \neq G = \{g_1, \ldots, g_m\}$ ist die nicht-leere Menge der Rollengruppen, zu denen die Rolle gehört. Die Menge $S = \{s_1, \ldots, s_n\}$ gibt die Sendeberechtigungen an, wobei jedes $s_i = [t_i, R_i]$ einem Nachtichtentyp t_i

[17] Wie bei der objektorientierten Vererbung lassen sich hier zueinander inkompatible Spezifikationen nicht völlig ausschließen (z.B. gleiche Eigenschaftsnamen mit unterschiedlichem Typ). Im Falle der Nachrichten besteht die Vereinigung sowohl in der Menge der Nachrichtentypen, als auch bei übereinstimmenden Typen in der Menge der Sendeberechtigungen.

eine Menge $R_i = \{r_1, \ldots, r_o\}$ an sendeberechtigten Rollen und Rollengruppen zuordnet. Die Menge $E = \{e_1, \ldots, e_p\}$ enthält die konfigurierbaren Eigenschaften der Rolle, wobei jede Eigenschaft $e_i = [n_i, t_i, W_i, s_i, k_i]$ aus dem Namen n_i, dem Typ t_i, dem Wertebereich W_i und dem Standardwert s_i besteht; der Wert k_i gibt an, ob die Eigenschaft abfragbar ist (r für read), initial gesetzt wird (i für init) und/oder jederzeit zur Konfiguration geändert werden kann (w für write).

Im folgenden bezeichnet Gruppen(r) die Menge aller Rollengruppen einer Rolle r und Gruppen(R) die Menge aller Rollengruppen einer Menge von Rollen R:

- $\text{Gruppen}(r) = G \cup \text{Gruppen}(G)$ für $r = [n, G, S, E]$
- $\text{Gruppen}(R) = \cup_{i=1\ldots n} \text{Gruppen}(r_i)$ für $R = \{r_1, \ldots, r_n\}$

Eine Rollengruppe besitzt dieselbe Definition, außer daß sie die oberste Rollengruppe `Main` direkt ableitet, d.h. es gilt $\texttt{Main} \in G$. Einzig diese oberste Rollengruppe gehört keiner Gruppe an, d.h. bei ihr ist $G = \emptyset$. Bei der Definition einer Rollengruppe g sind Zyklen nicht erlaubt: $g \notin \text{Gruppen}(g)$. Mehrfachvererbung ist aufgrund der Mengenbildung unproblematisch.

Definition 2: Komponentenspezifikationen

Eine Komponentenspezifikation s ist ein Tripel [n, R, E]. n ist der eindeutige Name der Komponente. $\emptyset \neq R = \{r_1, \ldots, r_n\}$ ist die nicht-leere Menge der Rollen der Komponente; als r_i sind dabei keine Rollengruppen zulässig. E ist analog zu der Definition für Rollen eine Menge konfigurierbarer Eigenschaften.

Definition 3: Komponenten

Eine Komponente k wird innerhalb eines Agenten repräsentiert über ein Tupel [b, s]. b ist dabei die Basisschnittstelle der jeweiligen Komponenteninstanz, über die sämtliche Interaktionen zwischen Agentenhülle und Komponente stattfinden. s ist die Komponentenspezifikation.

Im folgenden bezeichnet Rollen(k) die Menge R der Rollen einer Komponente k und Rollen(K) die Menge aller Rollen einer Menge von Komponenten K:

- $\text{Rollen}(k) = R$ für $k = [b, [n, R, E]]$
- $\text{Rollen}(K) = \cup_{i=1\ldots n} \text{Rollen}(k_i)$ für $K = \{k_1, \ldots, k_n\}$

3.3. Basisschnittstelle einer Komponente

Um als Teil eines Agenten verwaltet und kontrolliert werden zu können sowie um Nachrichten versenden, empfangen und verarbeiten zu können, benötigen alle Komponenten eine einheitliche Basisschnittstelle zur Agentenhülle. Diese Schnittstelle ist anders als die Komponentenrolle keine reine Spezifikation, sondern beinhaltet auch Funktionalitäten zu deren Umsetzung. Die Basisschnittstelle und ihre Grund-

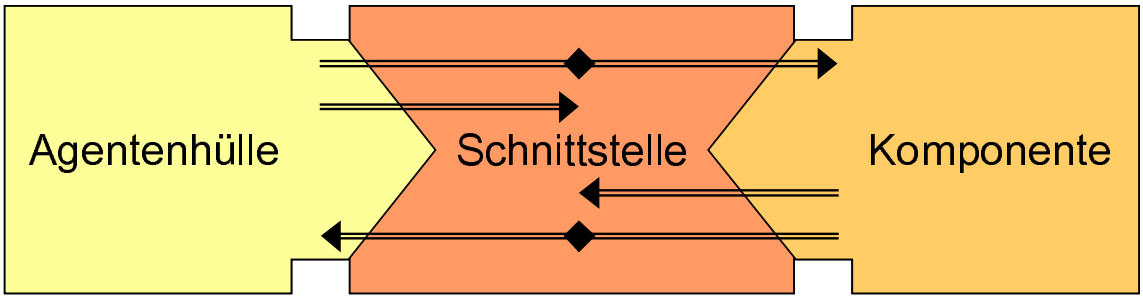

Abbildung 7: Basisschnittstelle zwischen Agentenhülle und Komponente

funktionalität sind es letztlich, die zusammen mit der Komponentenrolle ein Stück Programmkode zu einer CASA-Komponente machen.

Eine Ausnahme bilden die Infrastrukturkomponenten für Nachrichtenzustellung und Kontrollzyklus, die nur eine eingeschränkte Version der Basisschnittstelle besitzen[18]. Da sie selbst keine eigenen Nachrichten verwenden oder unabhängige Aktivitäten ausführen, benötigen sie die dafür bereitgestellten Funktionalitäten nicht. Aus demselben Grund brauchen sie auch keine Komponentenrolle, sondern lediglich eine Konfigurationsschnittstelle als spezifische Schnittstellenbeschreibung.

Die Basisschnittstelle umfaßt Schnittstellen zu allen drei Komponenten der Agentenhülle, dem Agentenkern, der Nachrichtenzustellung und dem Kontrollzyklus. Dabei ermöglicht sie Interaktionen zwischen diesen und der Komponente auf vier Arten (Abbildung 7). Zum einen bietet sie der Agentenhülle Zugriff auf die Komponentenschnittstelle selbst, über die sie die Komponente kontrollieren und ihren Zustand verändern kann. Des weiteren erlaubt sie einer Komponente die individuelle Ausgestaltung von Funktionen, über die die Agentenhülle der Komponente Zustandsänderungen und Ereignisse mitteilen kann und dieser so Gelegenheit zur Verarbeitung von diesen gibt. Schließlich kann noch in umgekehrter Richtung die Komponente die durch die Schnittstelle sowie die durch die Agentenhülle bereitgestellten Funktionalitäten nutzen.

3.3.1. Schnittstelle zum Agentenkern

Entsprechend der Aufgabe des Agentenkerns dient die Schnittstelle zu diesem der Verwaltung, Konfiguration und Zustandskontrolle der Komponente.

Komponentenverwaltung

Zur Verwaltung der Komponente gibt es Schnittstellen und Funktionalitäten für die beim Hinzufügen, Austauschen und Entfernen einer Komponente notwendigen

[18] Im folgenden bezieht sich Komponente i.a. nur auf solche Komponenten, die die vollständige Basisschnittstelle realisieren. Dies sind alle außer Nachrichtenzustellung und Kontrollzyklus, die meist explizit als Infrastrukturkomponenten bezeichnet werden.

Vorgänge. Beim Hinzufügen wird die Komponente initialisiert, wodurch sie zum Teil eines Agenten wird und auf dessen Infrastrukturfunktionalitäten zugreifen kann, bis sie wieder entfernt wird. Eine Komponente kann also die Funktionalität der Basisschnittstelle nur nutzen, während sie zu einem Agenten gehört, da nur dann der Zugriff auf die Komponenten der Agentenhülle möglich ist.

Auch wenn der Austausch von Komponenten letztlich im Entfernen der alten und anschließenden Hinzufügen der neuen Komponenten besteht, benötigt er dennoch eine gesonderte Unterstützung durch die Basisschnittstelle, da für einen übergangslosen Austausch übereinstimmende Rollen permanent belegt bleiben müssen (vgl. Kapitel 3.4.3). Außerdem wird der neuen Komponente Gelegenheit zur Übernahme des Zustands der alten gemäß der rollenspezifischen Schnittstelle gegeben.

Konfiguration

Die Agentenhülle konfiguriert Komponenten anhand ihrer und der für deren Rollen spezifizierten Eigenschaften. Die Basisschnittstelle ermöglicht zusätzlich die Konfiguration ihrer eigenen Funktionalitäten. Die initiale Konfiguration geschieht jeweils beim Hinzufügen der Komponente. Jede Änderung der Konfiguration wird der Komponente mitgeteilt, um sich an diese anzupassen.

Zustandskontrolle

Der Agent und seine Komponenten unterliegen einem vorgegebenen Lebenszyklus (s. Kapitel 3.4.1), der vom Agentenkern kontrolliert wird. Der Lebenszykluszustand bestimmt für eine Komponente, in welchem Konfigurationszustand sie sich befinden soll und welche Arten von Prozessen sie ausführen darf. Die Basisschnittstelle speichert diesen Zustand und erlaubt dem Agentenkern dessen Veränderung. Jede Veränderung des Lebenszykluszustands teilt die Schnittstelle der Komponente mit, damit diese sich entsprechend verhalten kann. Die Komponente teilt ihrerseits mit, ob die Zustandsänderung erfolgreich war.

Überwachung

Optional kann die Basisschnittstelle auch weitere Funktionalitäten zur Kontrolle und Überwachung der Komponente inklusive von Eigenschaften zur Konfiguration dieser Funktionalitäten bereitstellen.

Dazu zählen beispielsweise Mechanismen für das Monitoring und Debugging, über die Abläufe innerhalb des Agenten und seiner Komponenten protokolliert und nachvollziehbar gemacht werden. Die Basisschnittstelle dient dabei vor allem dem Sammeln und Zusammenführen der Laufzeitinformationen der einzelnen Komponenten, die dann über den Agentenkern weitergeleitet oder durch besondere Managementkomponenten ausgewertet werden können.

3.3.2. Schnittstelle zur Nachrichtenzustellung

Die Schnittstelle zur Nachrichtenzustellung dient dem Senden und Empfangen von Nachrichten, wozu sie eine Nachrichtenschlange für empfangene Nachrichten bereitstellt.

Nachrichten senden

Das Versenden einer Nachricht besteht einfach in der Übergabe dieser Nachricht an die Komponente für die Nachrichtenzustellung. Die Nachricht enthält alle zur Zustellung erforderlichen Angaben inklusive Absender und Empfänger. Die Nachrichtenzustellung teilt mit, ob die Zustellung erfolgen konnte oder nicht. Eine Zustellung ist nur möglich, wenn eine Komponente für die Empfängerrolle existiert und der Absender die entsprechende Sendeberechtigung besitzt. Eine Zustellung garantiert jedoch noch nicht, daß die Nachricht auch verarbeitet wird.

Nachrichten empfangen

Zum Empfang einer Nachricht fügt die Nachrichtenzustellung diese lediglich in die Nachrichtenschlange der Komponente ein. Dabei muß außer der Reihenfolge des Eintreffens auch die Priorität, mit der die Nachricht verarbeitet werden soll, berücksichtigt werden, so daß die Anordnung der Nachrichten in der Warteschlange immer der vorgesehenen Verarbeitungsreihenfolge entspricht.

Nachrichtenschlange

Die Nachrichtenschlange wird durch die Basisschnittstelle realisiert und dient als Zwischenspeicher für empfangene Nachrichten[19]. Es handelt sich dabei um einen nach Prioritäten untergliederten FIFO-Stapel, in den neue Nachrichten jeweils nach den bereits wartenden Nachrichten gleicher Priorität eingefügt werden. Die vorderste Nachricht ist somit immer die älteste von denen mit der höchsten Priorität und damit diejenige, die als erste zu verarbeiten ist.

3.3.3. Schnittstelle zum Kontrollzyklus

Die Ausführungskontrolle umfaßt die Ausführung einzelner Schritte und die Deklaration des Beschäftigungszustands.

---

[19] Im Prinzip genügt eine Nachrichtenschlange pro Komponente, die Basisschnittstelle kann aber auch für jede Rolle einer Komponente eine eigene Schlange verwenden. In diesem Fall muß der Kontrollzyklus entsprechend Rollen statt Komponenten verwalten. Dies vereinfacht die Übergabe von wartenden Nachrichten beim Austausch von Komponenten.

Schrittausführung

Für eine kontrollierte Ausführung müssen alle Aktivitäten einer Komponente in Einzelschritte untergliedert sein (vgl. Kapitel 3.6.2). Die Ausführung der Schritte wird über die Basisschnittstelle kontrolliert. Dies besteht zum einen in der Verarbeitung einer Nachricht, wozu jeweils die oberste Nachricht aus der Warteschlange entfernt und dann der Komponente zur Ausführung übergeben wird. Zum anderen wird der Komponente bei Bedarf die Gelegenheit zur Ausführung individueller Schritte gegeben. Diesen Bedarf zeigt die Komponente über die Basisschnittstelle dem Kontrollzyklus an.

Beschäftigungszustand

Der Beschäftigungszustand (vgl. Kapitel 3.6.3) gibt an, ob die Komponente gerade mit irgendwelchen Aktivitäten beschäftigt ist, und wird von der Basisschnittstelle verwaltet. Bei der gerade beschriebenen Schrittausführung über die Basisschnittstelle registriert diese selbst die entsprechenden Aktivitäten. Zusätzlich kann die Komponente unabhängig davon ausgeführte Aktivitäten deklarieren, indem sie deren Beginn und Ende mitteilt, wobei ein Beginn jedoch nur dann zulässig ist, wenn die Komponente sich in einem Lebenszyklusstand befindet, der dies erlaubt.

Die Agentenhülle und die Komponente selbst können den Beschäftigungszustand an der Basisschnittstelle abfragen. So kann beispielsweise verhindert werden, daß eine Komponente entfernt wird, während sie noch mit nicht unterbrechbaren Aktivitäten beschäftigt ist.

Komponente	Schnittstelle	Richtung
Agentenkern	Anmelden beim Hinzufügen	H → S
	Abmelden beim Entfernen	H → S
	Komponentenaustausch (generisch)	H → S
	Komponentenaustausch (komponentenspezifisch)	H → S → K
	Konfiguration (generisch)	H → S
	Konfiguration (komponentenspezifisch)	H → S → K
	Lebenszykluszustand verwalten	H → S
	Lebenszykluszustand umsetzen	H → S → K
	Abfrage des Lebenszykluszustands	H → S ← K
	Monitoring und Debugging	S ← K
Nachrichten-zustellung	Nachrichten versenden	H ← S ← K
	Nachrichten empfangen	H → S → K
Kontrolle	Schritt ausführen (Nachrichtenverarbeitung)	H → S → K
	Schritt ausführen (komponentenspezifisch)	H → S → K
	Anmelden von komponentenspezifischen Schritten	H ← S ← K
	Deklaration des Beschäftigungszustands	S ← K
	Abfrage des Beschäftigungszustands	H → S ← K

Abbildung 8: Schnittstelle zwischen Agentenhülle und Komponente

3.3.4. Zusammenfassung

Abbildung 8 faßt die einzelnen Schnittstellen gruppiert nach der zugehörigen Komponente der Agentenhülle zusammen, wobei zusätzlich die Richtung des Zugriffs auf die Schnittstelle angegeben ist. (Richtung und Anzahl der Pfeile entsprechen Abbildung 7; H steht für Agentenhülle, S für Basisschnittstelle und K für Komponente.)

Spezifikation der eingeschränkten Basisschnittstelle der Infrastrukturkomponenten

Die Basisschnittstelle verwaltet intern den Lebenszykluszustand der jeweiligen Komponente:

- Lebenszykluszustand l ∈ LifeCycleStates = {”undefined“, ”stopped“, ”initiated“, ”active“, ”stepping“, ”suspended“, ”persistent“, ”transit“, ”defective-undefined”, ”defective-stopped“, ”defective-initiated“, ”defective-active“, ”defective-stepping“, ”defective-suspended“, ”defective-persistent“, ”defective-transit“}; Der Lebenszykluszustand ist eine Eigenschaft mit dem Bezeichner State, dem Typ string und dem angegebenen Wertebereich LifeCycleState. Abgefragt und gesetzt wird er über die unten für Eigenschaften deklarierten Methoden getValue und setValue.

Die Schnittstelle zum Agentenkern besteht aus folgenden Methoden:

Methode	boolean initComponent()
Beschreibung	initialisiert die Basisschnittstelle, wenn die Komponente einem Agenten hinzugefügt wird
Rückgabewert	true, falls Initialisierung erfolgreich, sonst false

Methode	void resetComponent()
Beschreibung	setzt die Basisschnittstelle zurück, wenn die Komponente aus einem Agenten entfernt wird

Methode	Value getValue(Attribute)
Beschreibung	ruft den Wert einer Eigenschaft der Komponente ab
Parameter	Attribute: Bezeichner für die Eigenschaft
Rückgabewert	der Wert der Eigenschaft

Methode	boolean setValue(Attribute, Value)
Beschreibung	verändert den Wert einer Eigenschaft
Parameter	Attribute: Bezeichner für die Eigenschaft Value: der neue Wert der Eigenschaft
Rückgabewert	true, wenn der neue Wert gesetzt wurde, sonst false

Spezifikation der Basisschnittstelle

Die erweiterte Basisschnittstelle für Komponenten, die über Nachrichten interagieren, besitzt zusätzlich die folgenden Instanzvariablen und Methoden. Sie verwaltet die folgende Daten:

- der Beschäftigungszustand boolean b; b ist wahr, während die Komponente einen Schritt ausführt, sonst falsch (Der Beschäftigungszustand ist eine Eigenschaft mit dem Bezeichner Busy und dem Typ boolean, die intern verwaltet wird und nur abgefragt werden kann.)
- die Nachrichtenschlange $N = [n_1, \ldots, n_m]$, die die wartenden empfangenen Nachrichten der Komponente enthält; für die Reihenfolge der Nachrichten gilt:
  $i < j \rightarrow (p(n_i) < p(n_j)) \vee (p(n_i) = p(n_j) \wedge t(n_i) > t(n_j))$
  mit p(n) Priorität der Nachricht, t(n) Zeitpunkt des Eintreffens

Die Schnittstelle zum Agentenkern umfaßt zusätzlich folgende Methoden:

Methode	void exchangeComponent(Component)
Beschreibung	gibt der Basisschnittstelle beim Austausch von Komponenten Gelegenheit zur Übernahme interner Daten; insbesondere werden Nachrichten der Warteschlange gemäß der übereinstimmenden Rollen übernommen
Parameter	Component: die zu ersetzende Komponente

Methode	void setMessageServer(MessageServer)
Beschreibung	tauscht die Nachrichtenzustellung aus
Parameter	MessageServer: die neue Komponente zur Nachrichtenzustellung

Methode	void setControlCycle(ControlCycle)
Beschreibung	tauscht den Kontrollzyklus aus
Parameter	ControlCycle: die neue Komponente für den Kontrollzyklus

Die Schnittstelle zur Nachrichtenzustellung besitzt nur eine Methode:

Methode	void receive(Message)
Beschreibung	fügt zum Empfang einer Nachricht diese in die Nachrichtenschlange ein; falls die Warteschlange leer war, wird der Kontrollzyklus über newMessage über die neue Nachricht informiert
Parameter	Message: die empfangene Nachricht n
Änderung	$N' = [n_1, \ldots, n_i, n, n_j, \ldots, n_m]$ mit $p(n_i) < p(n) \wedge p(n) \leq p(n_j)$ (außer bei direkter Priorität)

Die Schnittstelle zum Kontrollzyklus umfaßt:

Methode	boolean stepComponent()
Beschreibung	initiiert die Ausführung eines komponentenspezifischen Schritts
Rückgabewert	true, falls weitere Schritte auszuführen sind, sonst false; bei false wird stepComponent erst dann wieder aufgerufen, wenn über newStep dem Kontrollzyklus das Vorliegen neuer Schritte mitgeteilt wird

Methode	boolean stepMessage()
Beschreibung	initiiert die Abarbeitung einer Nachricht, die dazu aus der Nachrichtenschlange entfernt und an die Methode execute der Komponente übergeben wird
Rückgabewert	true, falls weitere Nachrichten in der Warteschlange enthalten sind, sonst false; bei false wird stepMessage erst dann wieder aufgerufen, wenn über newMessage dem Kontrollzyklus das Eintreffen neuer Nachrichten mitgeteilt wird
Änderung	N' = $[n_1, \ldots, n_{m-1}]$, falls m > 0

Die Basisschnittstelle stellt der Komponente folgende Methoden zur Nutzung bereit:

Methode	boolean send(Message)
Beschreibung	übergibt eine Nachricht zum Versenden an die Nachrichtenzustellung
Parameter	Message: die zu versendende Nachricht
Rückgabewert	true, falls die Nachricht zugestellt werden konnte, sonst false

Methode	boolean beginBusy()
Beschreibung	deklariert den beabsichtigten Beginn einer Aktivität
Rückgabewert	true, falls die Aktivität begonnen werden darf, sonst false; nur bei true muß über endBusy das Ende der Aktivität angezeigt werden

Methode	void endBusy()
Beschreibung	deklariert das Ende einer begonnenen Aktivität

Methode	void newStep()
Beschreibung	teilt dem Kontrollzyklus mit, daß neue komponentenspezifische Schritte auf die Ausführung warten

Spezifikation der Schnittstelle von Komponenten

Jede Komponente muß für die Basisschnittstelle die folgenden Methoden bereitstellen:

Methode	boolean changeValue(Attribute, Value)
Beschreibung	teilt der Komponente die Veränderung des Werts einer Eigenschaft mit, damit diese sich entsprechend verhalten kann
Parameter	Attribute: Bezeichner für die Eigenschaft Value: der neue Wert der Eigenschaft
Rückgabewert	true, falls die Eigenschaft geändert werden kann, sonst false

Methode	boolean step()
Beschreibung	führt einen komponentenspezifischen Schritt aus
Rückgabewert	true, wenn ein weiterer Schritt ausgeführt werden soll, sonst false

Methode	void execute(Message)
Beschreibung	bearbeitet eine Nachricht
Parameter	Message: die zu bearbeitende Nachricht

Methode	void exchange(Component)
Beschreibung	ermöglicht die Übernahme komponentenspezifischer Daten beim Austausch einer Komponente
Parameter	Component: die zu ersetzende Komponente

Methode	string recoverState()
Beschreibung	stellt eine Komponente wieder her, falls sie sich in einem fehlerhaften Lebenszykluszustand befindet
Rückgabewert	der Lebenszykluszustand, dem die Komponente nach der (versuchten) Wiederherstellung gerecht wird

3.4. Agentenkern

Der Agentenkern verwaltet den Agenten, dessen Komponentenstruktur sowie die einzelnen Komponenten. Damit fungiert er als zentraler Kern, der die Komponenten eines Agenten zu einer Einheit zusammenfügt und so den Agenten konstituiert.

Der Agentenkern ist eine Komponente wie alle anderen, d.h. er realisiert die Basisschnittstelle und besitzt eine eigene Rolle, so daß er über Nachrichten Informationen mit anderen Komponenten austauschen kann. Eine Besonderheit dabei ist jedoch, daß die Interaktion mit anderen Komponenten nicht wie sonst auf den Nachrichtenaustausch beschränkt ist, sondern der Agentenkern – wie die beiden anderen Komponenten der Agentenhülle – einen direkten Zugriff auf alle Komponenten hat, um diese direkt kontrollieren und verwalten zu können.

Die folgenden Aufgaben erfüllt der Agentenkern innerhalb eines Agenten: Er kontrolliert den Lebenszykluszustand sowie die Konfiguration des Agenten und seiner Komponenten. Er verwaltet die Komponentenstruktur, indem er das Hinzufügen, Entfernen und Austauschen von Komponenten ermöglicht. Und schließlich dient er noch als Schnittstelle für den Zugriff auf den Agenten von außen.

3.4.1. Lebenszykluszustand

Ein Agent besitzt einen vorgegebenen Lebenszyklus aus Zuständen, in denen er sich befinden kann. Dabei nimmt er jeweils genau einen der folgenden Zustände ein:

- *undefined*: undefiniert (Ausgangszustand und Endzustand)
- *stopped*: existent, aber noch nicht initiiert
- *initiated*: initiiert, aber arbeitet noch nicht
- *active*: arbeitet kontinuierlich
- *stepping*: arbeitet in vorgegebenen Einzelschritten
- *suspended*: Arbeit unterbrochen

- *persistent*: vorbereitet zur Persistenz
- *transit*: vorbereitet zur Migration
- *defective*: zu jedem der Zustände gibt es einen korrespondierenden Fehler-Zustand

Zustandsübergänge

Da einige dieser Zustände andere voraussetzen, sind nur bestimmte Zustandsübergänge erlaubt (Abbildung 9). Soll ein Agent in einen Zustand versetzt werden, zu dem kein direkter Übergang möglich ist, so werden zuvor die minimal nötigen Zwischenzustände durchlaufen.

Da – wie aus Abbildung 9 ersichtlich – jeder Zustand mehrfach während der Lebensspanne eines Agenten eingenommen werden kann, muß der Agent bei Rückkehr in einen Lebenszykluszustand sich auch intern jeweils wieder in einem entsprechenden Zustand befinden.

Die Lebenszykluszustände

Der initiale Zustand eines Agenten ist immer *undefined*, d.h. er befindet sich noch nicht in einem festgelegten Agentenzustand. Von diesem Zustand aus kann der Agent nur in den Zustand *stopped* überführt werden, in dem lediglich vorausgesetzt

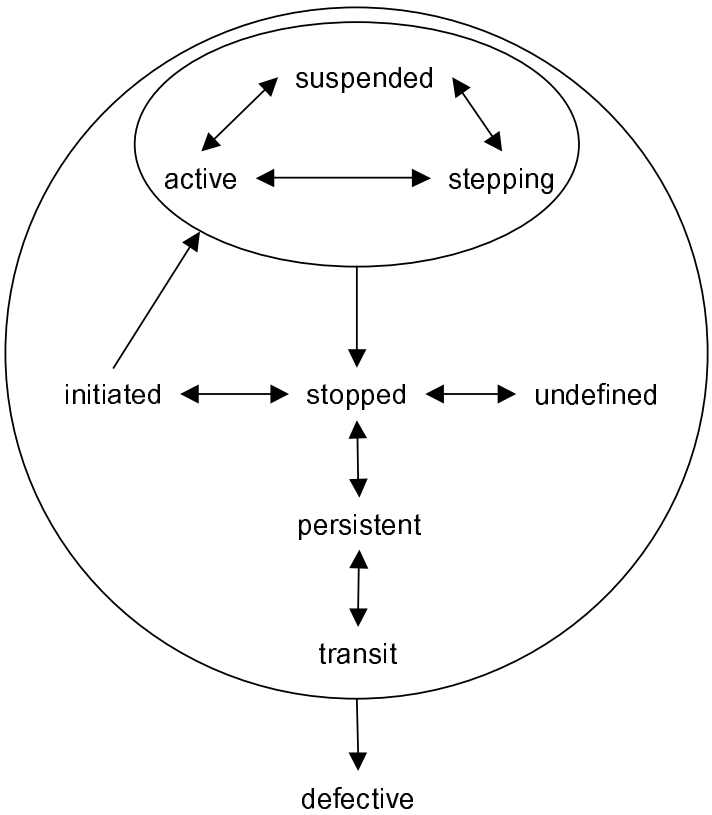

Abbildung 9: Übergänge zwischen Lebenszykluszuständen

wird, daß der Agent bereits als Agent existiert, also die interne Infrastruktur vollständig aufgebaut ist. Da dieser Übergang zu Beginn und üblicherweise nur einmal während der Lebensspanne eines Agenten stattfindet, ist er der geeignete Punkt für einmalige Initialisierungen. Zurück in den Zustand *undefined* wird der Agent erst wieder zum Beenden versetzt.

Da für die drei Arbeitszustände *active*, *stepping* und *suspended*, zwischen denen beliebig gewechselt werden kann, eine eigene Initialisierung notwendig sein kann, muß in jedem Fall zuvor der Zustand *initiated* eingenommen worden sein. Der Übergang von *stopped* nach *initiated* ist somit der vorgesehene Punkt für Initialisierungen, die nur für die Arbeitszustände benötigt werden und bei der Rückkehr nach *stopped* gegebenenfalls wieder rückgängig gemacht werden.

Nur im Zustand *active* dürfen der Agent und damit seine Komponenten eigenständig Aktivitäten ausführen. Im Zustand *stepping* befindet sich der Agent im Schrittmodus (vgl. Kapitel 3.6.2) und sämtliche Aktivitäten werden einem von außen vorgegebenen Takt unterworfen. Um die Aktivitäten zeitweilig zu unterbrechen, ohne die Arbeitsbereitschaft zu beenden, kann der Agent in den Zustand *suspended* versetzt werden.

Um den Gesamtzustand eines Agenten dauerhaft speichern zu können, sind u.U. Vorbereitungen notwendig. Um diese vorzunehmen, wird der Agent in den Zustand *persistent* überführt, in dem er alle Voraussetzungen zur Persistenz erfüllt. Zuvor muß der Agent sämtliche Aktivitäten beenden, also im Zustand *stopped* sein. Die Persistenz ist eine Voraussetzung für die Migration, weshalb der Zustand *transit* nur von *persistent* aus erreichbar ist. Beim Übergang in den Zustand *transit* sollte der Agent zusätzlich auf die Durchführung einer Migration vorbereitet werden.

Im Falle eines Fehlers wird der Agent in einen Fehlerzustand versetzt. Damit die Information über den Zustand beim Auftreten des Fehlers nicht verloren geht, gibt es zu jedem Zustand einen korrespondierenden Fehlerzustand beginnend mit *defective*. Der Übergang zu einem Fehlerzustand und zurück ist jeweils nur für den korrespondierenden Zustand möglich.

Komponentenzustände

Jede Komponente eines Agenten befindet sich ebenfalls immer in genau einem der genannten Lebenszykluszustände. Der aktuelle Zustand ist in der Basisschnittstelle der Komponente gespeichert und wird vom Agentenkern kontrolliert. Im Normalfall ist dies derselbe wie der des Agenten, d.h. bei jeder Zustandsänderung des Agenten versetzt der Agentenkern alle vorhandenen Komponenten in den entsprechenden Zustand. Diese Zustandsänderung wird der Komponente jeweils über die Basisschnittstelle mitgeteilt, so daß sie sich auf den neuen Zustand einstellen kann. Falls die Komponente den neuen Zustand nicht übernehmen kann, so wird sie in einen Fehlerzustand versetzt.

Eine Komponente kann nur aus zwei Gründen einen anderen Lebenszykluszustand einnehmen als der Agent. Zum einen kann eine Komponente nicht mehr funktionsfähig sein und deshalb in einen Fehlerzustand überführt worden sein. Dies ist jeweils der korrespondierende Fehlerzustand zu dem Zustand, in dem der Fehler aufgetreten ist. Einmal in einem Fehlerzustand kann an einer Komponente keine Zustandsänderung mehr vorgenommen werden, bis der Fehler behoben ist.

Zum anderen besitzt eine Komponente beim Hinzufügen zur Laufzeit zunächst den Zustand *undefined* und wird erst nach dem Hinzufügen in den Lebenszykluszustand des Agenten versetzt, damit sie während der Zustandsübergänge bereits die Funktionalität der Basisschnittstelle nutzen kann. Umgekehrt wird auch eine Komponente beim Entfernen erst in den Zustand *undefined* versetzt, bevor sie dann tatsächlich entfernt werden kann.

3.4.2. Konfiguration

Der Agentenkern konfiguriert sowohl den Agenten als ganzes, als auch dessen Komponenten. Er übernimmt zudem auch das Erzeugen und die Initialisierung des Agenten.

Erzeugen eines Agenten

Der Agentenkern erzeugt einen Agenten anhand einer Spezifikation[20], die auch die Komponenten und deren initiale Konfiguration umfaßt. Als erstes werden dazu die Infrastrukturkomponenten für Nachrichtenzustellung und Kontrollzyklus erzeugt und konfiguriert, da diese bereits vor dem Hinzufügen von Komponenten, die deren Funktionalität über ihre Basisschnittstelle nutzen können sollen, existieren müssen. Anschließend fügt sich der Agentenkern selbst als Komponente dem Agenten hinzu und danach für jede in der Spezifikation angegebene Komponente eine neu erzeugte Instanz. Beim Hinzufügen werden wie nachfolgend beschrieben die Komponenten gemäß der Spezifikation initialisiert.

Zum Abschluß konfiguriert der Agentenkern den Agenten und versetzt ihn in den spezifizierten Lebenszykluszustand. Ist dies der Zustand *active*, so kann der Agent seine Aktivitäten beginnen.

Konfiguration eines Agenten

Der Agentenkern beherbergt sämtliche Konfigurationsdaten, die den Agenten als ganzes betreffen. Dies umfaßt zum einen den Lebenszykluszustand des Agenten und seine Komponentenstruktur, zum anderen die Zugriffsrechte auf den Agenten

[20] Je nach Implementierungssprache ist entweder eine rein textuelle Spezifikation möglich, die der Agentenkern interpretiert, oder diese muß unter Zugriff auf die weiter unten beschriebene externe Schnittstelle des Agentenkerns jeweils einzeln programmiert werden.

von außen für die unten beschriebene Schnittstelle, über die sich der Agent und seine Komponenten auch noch zur Laufzeit konfigurieren lassen.

Konfiguration einer Komponente

Die Konfiguration einer Komponente geschieht anhand der Basisschnittstelle und der für die Komponente und ihre Rollen spezifizierten Eigenschaften. Der Agentenkern braucht nur den Namen der Komponente bzw. Rolle, die Eigenschaft und den Wert, um an der richtigen Komponente über die Basisschnittstelle eine Eigenschaft zu ändern.

Die Agentenspezifikation für die initiale Konfiguration muß dementsprechend lediglich jeweils einem Paar aus Name der Komponente bzw. Rolle und Name der Eigenschaft den initialen Wert zuordnen. Ist kein Wert explizit angegeben, so wird der Standardwert verwendet.

3.4.3. Komponentenstruktur

Die Verwaltung der Komponentenstruktur durch den Agentenkern besteht im Hinzufügen, Entfernen und Austauschen von Komponenten. Dies geschieht so, daß der momentane Lebenszykluszustand des Agenten dabei unerheblich ist, d.h. für das initiale Zusammensetzen des Agenten wie für Änderungen der Komponentenstruktur zur Laufzeit werden dieselben Mechanismen verwendet. Lediglich die Agentenhülle muß bestehen, bevor die Komponentenstruktur verändert werden kann.

Hinzufügen einer Komponente

Bevor eine Komponente zu einem Agenten hinzugefügt wird, überprüft der Agentenkern, daß keine ihrer Rollen bereits durch eine andere Komponente des Agenten belegt ist, da wegen der Identifikation der Komponenten anhand ihrer Rollen eine eindeutige Zuordnung von Rolle zu Komponente gewährleistet sein muß. Anschließend nimmt der Agentenkern die initiale Konfiguration der Komponente wie oben beschrieben vor. Da Komponenten auch zur Laufzeit hinzugefügt werden können, muß auch die Spezifikation des Agenten, die zur Konfiguration verwendet wird, noch zur Laufzeit erweiterbar sein um die Eigenschaften neuer Komponenten. Diese Erweiterung sollte jeweils vor dem Hinzufügen stattfinden.

Das eigentliche Hinzufügen der Komponente besteht in der Anmeldung der Komponente bei der Agentenhülle und umgekehrt in der Initialisierung der Basisschnittstelle der Komponente mit den Komponenten der Agentenhülle. Bei der Nachrichtenzustellung werden die Rollen der Komponente angemeldet, damit diese die als Adressen von Nachrichten verwendeten Rollen der jeweiligen Komponente zuordnen kann. Je nach Konfiguration wird die Komponente in den Kontrollzyklus eingegliedert oder erhält eigene Kontrollressourcen zur Nachrichtenverarbeitung

(vgl. Kapitel 3.6.2). Zum Schluß wird die Komponente in den Lebenszykluszustand des Agenten versetzt und kann dann gegebenenfalls mit ihren Aktivitäten beginnen.

Da sich die Komponente während des Hinzufügens im Zustand *undefined* befindet, kann dabei kein inkonsistenter Zustand entstehen. Daß die Komponente Teil eines Agenten ist, erfährt sie erst durch den Übergang von *undefined* nach *stopped*, weshalb sie vorher auch nicht auf die Infrastrukturfunktionalitäten des Agenten zugreifen darf. Und umgekehrt werden von der Agentenhülle kontrollierte Aktivitäten der Komponente nur in den Zuständen *active* und *stepping* ausgeführt, was ebenfalls erst nach Ende des Hinzufügens eintreten kann.

Entfernen einer Komponente

Das Entfernen einer Komponente besteht in dem umgekehrten Prozeß wie das Hinzufügen. Als erstes wird die Komponente in den Zustand *undefined* versetzt. Bevor die Komponente entfernt werden kann, muß noch abgewartet werden, bis sie ihre laufenden Aktivitäten beendet hat, was über den von der Basisschnittstelle verwalteten Beschäftigungszustand festgestellt werden kann. Der bereits eingenommene Zustand *undefined* verhindert dabei die Aufnahme neuer Aktivitäten.

Anschließend folgt die gegenseitige Abmeldung von Komponente und Agentenhülle, von wo ab die zugehörigen Rollen der Komponente nicht mehr innerhalb des Agenten verfügbar sind. Für die zum Zeitpunkt der Abmeldung der Komponente bei der Nachrichtenzustellung in der Warteschlange enthaltenen Nachrichten wird deren Absendern über eine Nachricht mitgeteilt, daß diese nicht verarbeitet werden konnten. Damit kann keine Nachricht verloren gehen, da alle an die zu entfernende Komponente gerichteten und noch nicht verarbeiteten Nachrichten entweder noch zugestellt und dann wie beschrieben beantwortet werden, oder nicht mehr zugestellt werden können, was der Absender dann direkt beim Versenden erfährt.

Da sich die Komponente nunmehr wieder im Zustand *undefined* befindet, kann sie prinzipiell wieder demselben oder einem anderen Agenten hinzugefügt werden, auch wenn dies im Regelfall nicht geschieht. Deshalb sollte beim Übergang von *stopped* nach *undefined* der interne Zustand der Komponente soweit zurückgesetzt werden, daß ein erneutes Hinzufügen problemlos möglich ist.

Austausch von Komponenten

Der Austausch von Komponenten besteht prinzipiell im Entfernen der alten und im Hinzufügen der neuen Komponenten wie bereits beschrieben. Allerdings müssen diese Vorgänge so miteinander verschränkt werden, daß der Austausch insofern übergangslos geschieht, als daß keine der ausgetauschten Rollen auch nur kurzfristig nicht belegt ist und keine zugehörigen Informationen verloren gehen. Eine mit einer Rolle interagierende Komponente darf durch deren Austausch abgesehen von unvermeidlichen geringfügigen zeitlichen Verzögerungen in ihrer Interaktion nicht beeinträchtigt werden. Da Komponenten auch mehrere Rollen einnehmen können,

kann es notwendig sein, mehrere Komponenten gleichzeitig auszutauschen, damit sämtliche Rollen zugleich durch neue Komponenten belegt werden. Beispielsweise kann eine Komponente mit zwei Rollen so ersetzt werden, daß zwei unterschiedliche neue Komponenten jeweils eine der Rollen übernehmen.

Für den Komponentenaustausch wird deshalb jeweils eine Menge von hinzuzufügenden Komponenten angegeben. Die Menge der zu ersetzenden ergibt sich dann von selbst anhand der übereinstimmenden Rollen vorhandener Komponenten zu den neuen. Es werden also alle Komponenten entfernt, die mindestens eine Rolle besitzen, die eine der neuen Komponenten für sich beansprucht, so daß auch nach dem Austausch die eindeutige Zuordnung von Rolle zu Komponente gewährleistet ist. In der Menge der neuen Komponenten darf es eben deshalb auch keine mehrfach vorkommenden Rollen geben. Da nicht zwangsläufig alle Rollen der so ermittelten zu entfernenden Komponenten von neuen Komponenten übernommen werden, muß jeweils angegeben werden, ob der Austausch auch in diesem Fall stattfinden soll oder nicht.

Der eigentliche Austausch findet nun in den folgenden Schritten statt. Als erstes wird überprüft, daß bei den neuen Komponenten keine Rolle doppelt vorkommt, und deren initiale Konfiguration vorgenommen. Anschließend wird die Menge der zu entfernenden Komponenten bestimmt und gegebenenfalls überprüft, ob diese keine nicht ersetzten Rollen besitzen. Die zu entfernenden Komponenten werden in den Zustand *undefined* versetzt und auf das Ende ihrer Aktivitäten gewartet. Dann werden sie bei der Agentenhülle abgemeldet, wobei für übereinstimmende Rollen eine direkten Ummeldung auf die neue Komponente vorgenommen wird, so daß insbesondere bei der Nachrichtenzustellung keine der ersetzten Rollen auch nur für einen Moment unbelegt bleibt. Noch nicht verarbeitete Nachrichten werden von der ersetzenden Komponente der jeweiligen Empfängerrolle übernommen, wobei die Reihenfolge der Nachrichten relativ zur Rolle gewahrt bleibt. Nur falls es keine ersetzende Komponente für eine Rolle gibt, wird wie auch sonst beim Entfernen einer Rolle über eine Nachricht dem Absender mitgeteilt, daß sie nicht verarbeitet werden kann. Dies garantiert, daß keine Nachricht verloren gehen kann. Gegebenenfalls werden noch zusätzliche, neu hinzukommende Rollen der hinzugefügten Komponenten angemeldet. Nun erhalten die neuen Komponenten die Gelegenheit, alle zur Übernahme der jeweiligen Rolle notwendigen Daten über die rollenspezifische Schnittstelle der entsprechenden alten Komponente abzurufen und sich so selbst zu konfigurieren, damit sie auch deren laufende Aufgaben fortsetzen können. Zum Abschluß werden dann die neuen Komponenten in den Lebenszykluszustand des Agenten versetzt und sind somit vollständig in den Agenten integriert und arbeitsfähig.

Abbildung 10 faßt das Hinzufügen (links, grün) und Entfernen (rechts, rot) von Komponenten sowie deren Verknüpfung beim übergangslosen Austausch (Mitte, blau) zusammen.

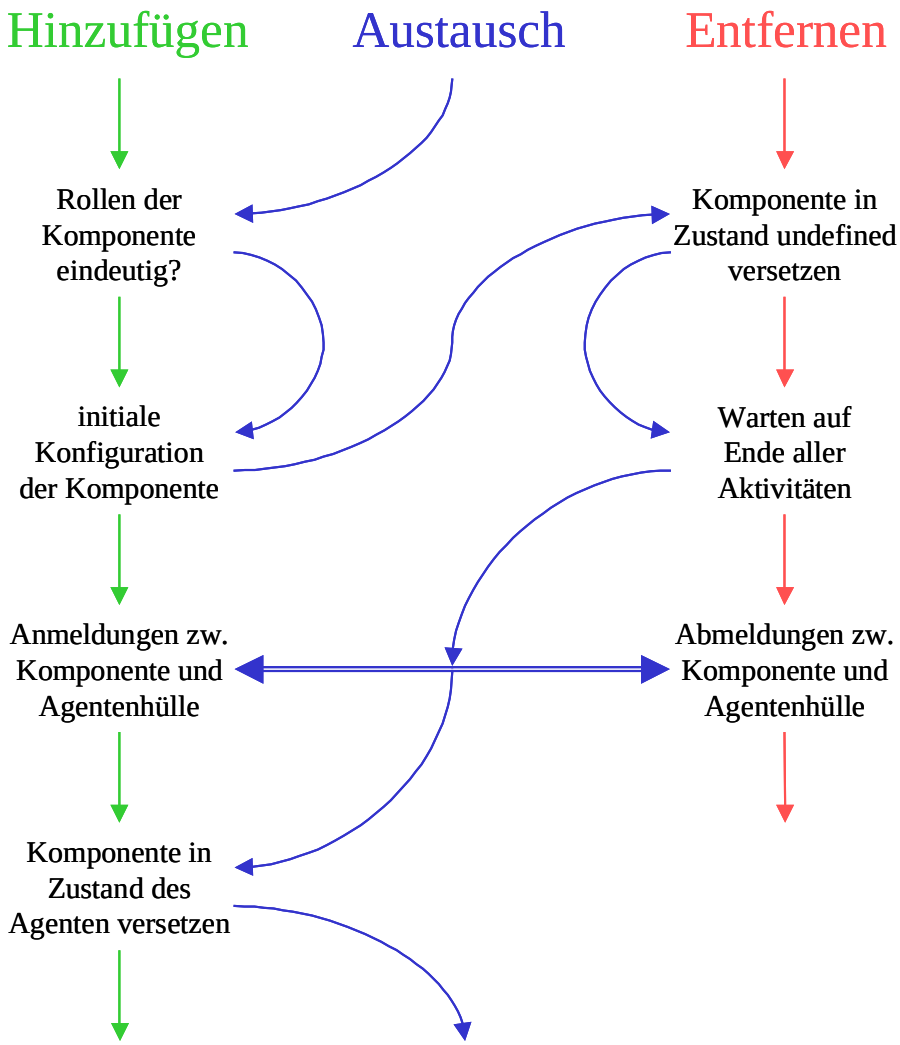

Abbildung 10: Hinzufügen, Austauschen und Entfernen von Komponenten

Austausch von Komponenten der Agentenhülle

Prinzipiell sind auch die Komponenten der Agentenhülle austauschbar, allerdings sind dabei einige Besonderheiten zu beachten. Ein reines Hinzufügen oder Entfernen ist nicht möglich, da sie in jedem Agenten permanent genau ein Mal vorhanden sein müssen.

Der Agentenkern ist eine Komponente mit der üblichen Basisschnittstelle und eigener Rolle und kann somit normal ausgetauscht werden. Dabei ist jedoch darauf zu achten, daß der Austausch so implementiert ist, daß die Übergabe der internen Daten den Austausch richtig widerspiegelt, da diese beim Agentenkern gerade die Agentenkonfiguration inklusive Komponentenstruktur beinhalten. Außerdem müssen externe Referenzen für den Zugriff auf den Agenten gewahrt bleiben, was beispielsweise durch das Zwischenschieben eines eigenen Schnittstellenmoduls gewährleistet werden kann, auf das sämtliche Referenzen von außen verweisen und

an dem seinerseits dessen Referenz auf den Agentenkern ausgewechselt werden kann.

Kontrollzyklus und Nachrichtenzustellung besitzen keine Komponentenrolle und benötigen eine eigene Form des Austauschs. Da dieser alle Komponenten betrifft, müssen während des Austauschs Zugriffe auf die Infrastrukturkomponenten über die Basisschnittstelle unterbunden werden. Falls dies nicht bereits auf Implementierungsebene (z.B. über Thread-Synchronisation) möglich ist, muß der gesamte Agent gegebenenfalls in einen Zustand versetzt werden, in dem keine neuen Aktivitäten mehr begonnen werden – dies sind alle außer *active* und *stepping* – und auf das Ende aller laufenden Aktivitäten gewartet werden. Dann wird die jeweilige Komponente innerhalb der Agentenhülle wie auch an den Basisschnittstellen aller Komponenten ausgetauscht, bevor der Agent seine Tätigkeit wieder aufnehmen kann.

3.4.4. Schnittstelle nach außen

Neben der internen Verwaltung kapselt der Agentenkern auch den Agenten nach außen hin ab. Dabei kann er über eine eigene Schnittstelle selektiv oder vollständig eine externe Kontrolle des Agenten erlauben. Eine Verwaltung von außen kann zum einen durch besondere Komponenten des Agenten (Managementkomponenten, vgl. Kapitel 4.6.4) oder durch Teile der übergeordneten Agenteninfrastruktur (AMS, vgl. Kapitel 5.2) geschehen. Für beide Arten des Zugriffs müssen separat als Teil der Agentenspezifikation die Zugriffsrechte festgelegt werden, die bestimmen, in wie weit der Agentenkern jeweils eine Steuerung von außen zuläßt. Auch der Erzeuger eines Agenten – gewöhnlich das Agent Management System (AMS) einer Agentenplattform (AP) – erhält dabei lediglich eine Referenz auf diese Schnittstelle.

Die Kontrollschnittstelle des Agentenkerns umfaßt zunächst dessen interne Verwaltungsfunktionalität, wozu der Lebenszykluszustand des Agenten, seine Komponentenstruktur, die Konfiguration von Agent und Komponenten inklusive Agentenspezifikation und schließlich der vollständige Zugriff auf alle Komponenten gehören. Zusätzlich unterstützt sie Funktionalitäten zur Umsetzung der Persistenz und Migration von Agenten sowie optional zur Überwachung des Agenten in Form von Monitoring und Debugging. Für den Schrittmodus ermöglicht die Schnittstelle des Agentenkerns die externe Vorgabe des Ausführungstakts, den sie dann an den Kontrollzyklus weiterleitet.

Spezifikation der Schnittstelle

Der Agentenkern besitzt die folgenden Instanzvariablen:

- der Lebenszykluszustand a des Agenten. Der Lebenszykluszustand des Agenten ist eine Eigenschaft mit dem Bezeichner AgentState, dem Typ string und dem Wertebereich LifeCycleState.
- die Menge $K = \{k_1, \ldots, k_n\}$ der Komponenten, aus denen der Agent besteht.

Folgende Methoden dienen der Verwaltung der Komponentenstruktur:

Methode	static AgentInterface create(Specification)
Beschreibung	erzeugt einen neuen Agenten anhand einer Spezifikation
Parameter	Specification: die Spezifikation für den Agenten
Rückgabewert	die Schnittstelle zum Zugriff auf den neu erzeugten Agenten

Methode	boolean addComponent(Component)
Beschreibung	fügt dem Agenten eine neue Komponente hinzu
Parameter	Component: die hinzuzufügende Komponente k
Rückgabewert	true, falls die Komponente hinzugefügt wurde, sonst false
Änderung	$K' = K \cup \{k\}$, falls $Rollen(K) \cap Rollen(k) = \emptyset$

Methode	boolean removeComponent(Component)
Beschreibung	entfernt eine Komponente aus dem Agenten
Parameter	Component: die zu entfernende Komponente k
Rückgabewert	true, falls die Komponente entfernt wurde, sonst false
Änderung	$K' = K \setminus \{k\}$, falls $k \in K$

Methode	boolean exchangeComponent(Component{}, boolean)
Beschreibung	tauscht Komponenten aus
Parameter	Component{}: eine Menge K^+ hinzuzufügender Komponenten boolean: Angabe b, ob sämtliche Rollen der zu entfernenden Komponenten neu belegt werden müssen (true) oder nicht (false)
Rückgabewert	true, falls der Austausch durchgeführt wurde, sonst false
Änderung	$K' = K \cup K^+ \setminus K^-$ mit $K^- = \{k \in K \mid \exists k' \in K^+, \exists r \in Rollen(k'): r \in Rollen(k)\}$ falls b = false oder $Rollen(K^-) \subseteq Rollen(K^+)$

Methode	Component getComponent(Role)
Beschreibung	ermöglicht den Zugriff auf eine Komponente, die innerhalb des Agenten eine bestimmte Rolle einnimmt
Parameter	Role: die gesuchte Rolle r
Rückgabewert	die Basisschnittstelle der Komponente $k \in K$ mit $r \in Rollen(k)$, falls $r \in Rollen(K)$

Methode	MessageServer getMessageServer()
Beschreibung	ermöglicht den Zugriff auf die Nachrichtenzustellung des Agenten
Rückgabewert	die Komponente zur Nachrichtenzustellung

Methode	void exchangeMessageServer(MessageServer)
Beschreibung	tauscht die Nachrichtenzustellung des Agenten aus
Parameter	MessageServer: die neue Komponente zur Nachrichtenzustellung

Methode	ControlCycle getControlCycle()
Beschreibung	ermöglicht den Zugriff auf den Kontrollzyklus des Agenten
Rückgabewert	die Komponente des Kontrollzyklus

Methode	void exchangeControlCycle(ControlCycle)
Beschreibung	tauscht den Kontrollzyklus des Agenten aus
Parameter	ControlCycle: die neue Komponente des Kontrollzyklus

Über die folgenden Methoden lassen sich die Komponenten des Agenten konfigurieren:

Methode	Value getValue(Component, Attribute)
Beschreibung	ruft den Wert einer Eigenschaft einer Komponente ab
Parameter	Component: die Komponente oder Rolle Attribute: Bezeichner für die Eigenschaft
Rückgabewert	der Wert der Eigenschaft

Methode	boolean setValue(Component, Attribute, Value)
Beschreibung	verändert den Wert einer Eigenschaft einer Komponente
Parameter	Component: die Komponente oder Rolle Attribute: Bezeichner für die Eigenschaft Value: der neue Wert der Eigenschaft
Rückgabewert	true, wenn der neue Wert gesetzt wurde, sonst false

Folgende Methoden unterstützen Persistenz und Migration:

Methode	Agent persistentAgent(boolean)
Beschreibung	erzeugt eine persistente Version des Agenten
Parameter	boolean: gibt an, ob der Agent danach beendet werden soll oder nicht
Rückgabewert	eine persistente Version des Agenten

Methode	Agent transportableAgent(boolean)
Beschreibung	erzeugt eine transportierbare Version des Agenten
Parameter	boolean: gibt an, ob der Agent danach beendet werden soll oder nicht
Rückgabewert	eine transportierbare Version des Agenten

Methode	static AgentInterface restartAgent(Agent)
Beschreibung	stellt einen Agenten aus einer persistenten oder transportierbaren Form wieder her
Parameter	Agent: der Agent in persistenter oder transportierbarer Form
Rückgabewert	die Schnittstelle zum Zugriff auf den wiederhergestellten Agenten

Schließlich gibt es noch folgende Methoden für den Schrittmodus:

Methode	boolean agentStep()
Beschreibung	führt über den Kontrollzyklus einen Schritt im Einzelschrittbetrieb aus
Rückgabewert	true, falls ein Schritt ausgeführt wurde, sonst false

Methode	void agentWait(time)
Beschreibung	simuliert das Vergehen von Zeit im Schrittmodus
Parameter	time: die vergangene Zeitspanne

3.5. Interaktionen zwischen Komponenten

Aufgrund der offenen internen Struktur im Aufbau von CASA-Agenten und insbesondere wegen der Austauschbarkeit der Komponenten zur Laufzeit unterliegen die Mechanismen für Interaktionen zwischen den Komponenten besonderen Anforderungen. Nach einer Erörterung dieser Anforderung werden die beiden Ansätze zu deren Erfüllung vorgestellt, zum einen eine indirekte Interaktion über Nachrichten, zum anderen eine direkte Interaktion vermittels Zwischenmodulen.

3.5.1. Anforderungen

Die zentrale Anforderung an die internen Infrastrukturmechanismen der CASA-Architektur ist eine Entkoppelung der Komponenten untereinander, um die Menge der in einem Agenten enthaltenen Komponenten möglichst offen und frei veränderbar zu halten und dennoch gerichtete Interaktionen zwischen diesen zu ermöglichen. Dazu müssen die Interaktionspartner anonymisiert werden, damit keine Abhängigkeiten zwischen konkreten Komponenteninstanzen entstehen können. Statt dessen wird eine abstrakte Spezifikation des Interaktionspartners verwendet, die jeweils von unterschiedlichen Komponenteninstanzen, die sogar noch zur Laufzeit wechseln können, erfüllt werden kann.

Eine andere Form der Entkoppelung ist die zeitliche. Die von verschiedenen Komponenten ausgeführten Aktivitäten sollten zeitlich unabhängig voneinander gehalten werden, so daß sie prinzipiell nebenläufig zueinander ausführbar sind. Für die Interaktionen bedeutet dies, daß diese lediglich im Austausch von Informationen bestehen, wobei der Zeitpunkt zu deren Verarbeitung jeweils von den die Informationen erhaltenden Komponenten selbst bestimmt wird. Entsprechend darf eine Komponente nicht blockiert werden, wenn sie während einer Interaktion auf Antworten oder Ergebnisse wartet, sondern muß dabei andere Aktivitäten ausführen können.

Allerdings ist die zeitliche Entkoppelung der Interaktionen eine nur bedingt notwendige Anforderung, die zeitweilig verletzt werden kann. Zur Gewährleistung der Austauschbarkeit genügt es im Prinzip, daß jede Komponente hinreichend oft in keiner direkten zeitlichen Abhängigkeit zu sämtlichen anderen steht und so genügend Gelegenheiten zum Austausch der Komponente ohne Verletzung dieser Abhängigkeiten gegeben sind. Zeitlich nicht entkoppelte Interaktionen besitzen dafür den Vorteil, daß sie sofort stattfinden und ihr Ergebnis direkt und schnellstmöglich verfügbar und verarbeitbar ist.

Die folgenden Interaktionsmuster zwischen Komponenten sollen unterstützt werden:

- *Benachrichtigung*: Eine Komponente teilt einer anderen bestimmte Informationen mit, woraufhin die Interaktion bereits wieder beendet ist.

- *Auftrag*: Eine Komponente beauftragt eine andere mit der Durchführung einer Aufgabe. Diese teilt zunächst mit, ob sie den Auftrag annimmt oder nicht. Nach der Ausführung teilt sie den Erfolg und gegebenenfalls zugleich das Ergebnis mit. Der Auftraggeber kann einen Auftrag zurücknehmen, falls er noch nicht ausgeführt wird.
- *Registrierung von Informationsbedarf*: Eine Komponente meldet sich bei einer anderen an, über bestimmte Ereignisse informiert zu werden. Dies kombiniert die beiden ersten Interaktionsarten zu einem Auftrag für Benachrichtigungen über bestimmte Informationen. Zur Abmeldung wird der Auftrag zurückgenommen.

Bei Interaktionsmustern aus mehreren Interaktionen muß dabei vor allem bei zeitlicher Entkoppelung eine Zuordnung der einzelnen Teilinteraktionen zueinander gewährleistet sein.

3.5.2. Interaktion über Nachrichten

Zeitlich entkoppelte Interaktionen zwischen Komponenten geschehen über den Austausch von Nachrichten. Die Interaktionspartner werden anonymisiert, indem deren Komponentenrollen zur Adressierung verwendet werden. Die Nachrichtenzustellung übernimmt die gleichnamige Infrastrukturkomponente. Empfangene Nachrichten werden in der Warteschlange der Basisschnittstelle zwischengespeichert, so daß die Komponente sie nicht sofort zu verarbeiten braucht, wodurch die zeitliche Entkoppelung gewährleistet ist. Ein Austausch von Nachrichten zwischen Komponenten ist in jedem Lebenszykluszustand möglich, sobald eine Komponente vollständig einem Agenten hinzugefügt ist, verarbeitet werden Nachrichten aber nur in den Zuständen *active* und *stepping*.

Die drei oben genannten Interaktionsmuster werden wie folgt umgesetzt:

- *Benachrichtigung*: Die benachrichtigende Komponente sendet eine Nachricht an die zu benachrichtigende, die die Nachricht lediglich zur Kenntnis nimmt und verarbeitet (Abbildung 11).
- *Auftrag*: Die beauftragende Komponente sendet eine Nachricht an die beauftragte, die zunächst eine Zusage oder Ablehnung und schließlich auch das Ergebnis als Antwort sendet[21] (Abbildung 12). Gegebenenfalls kann die beauftragende Komponente eine Nachricht zum Abbruch des Auftrags senden, worauf die beauftragte wiederum entweder eine Zusage oder Ablehnung sendet, je nachdem, ob der Auftrag noch zurückgezogen werden konnte oder nicht (Abbildung 13).

[21] Im Falle eines sofort gesendeten Ergebnisses kann auf die explizite Annahme des Auftrags durch eine eigene Nachricht auch verzichtet werden. Diese ergibt sich dann implizit aus dem Eintreffen des Ergebnisses.

- *Registrierung von Informationsbedarf*: Die sich registrierende Komponente sendet eine Nachricht an den Informationsanbieter. Dieser teilt mit, ob er den Auftrag annimmt oder ablehnt. Gegebenenfalls benachrichtigt der Informationsanbieter die registrierten Komponenten jeweils über neue Ereignisse. Um sich abzumelden, kann eine registrierte Komponente den Auftrag zurückziehen, was wiederum über eine Nachricht bestätigt wird (Abbildung 14).

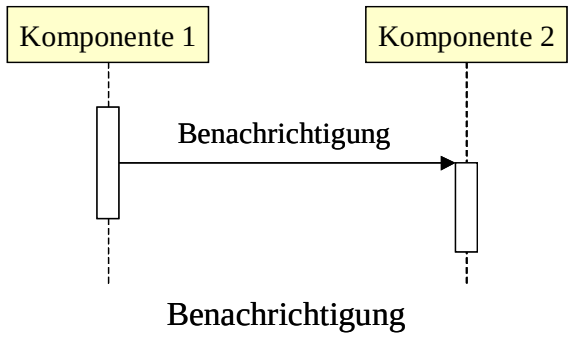

Abbildung 11: Nachrichtenschema für eine Benachrichtigung

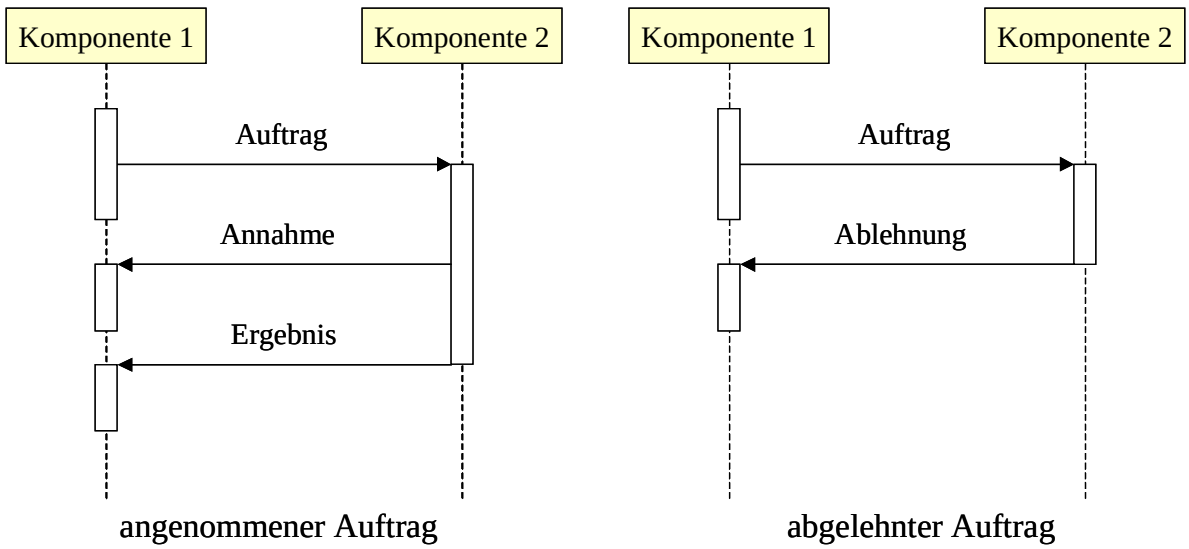

Abbildung 12: Nachrichtenschema für einen Auftrag

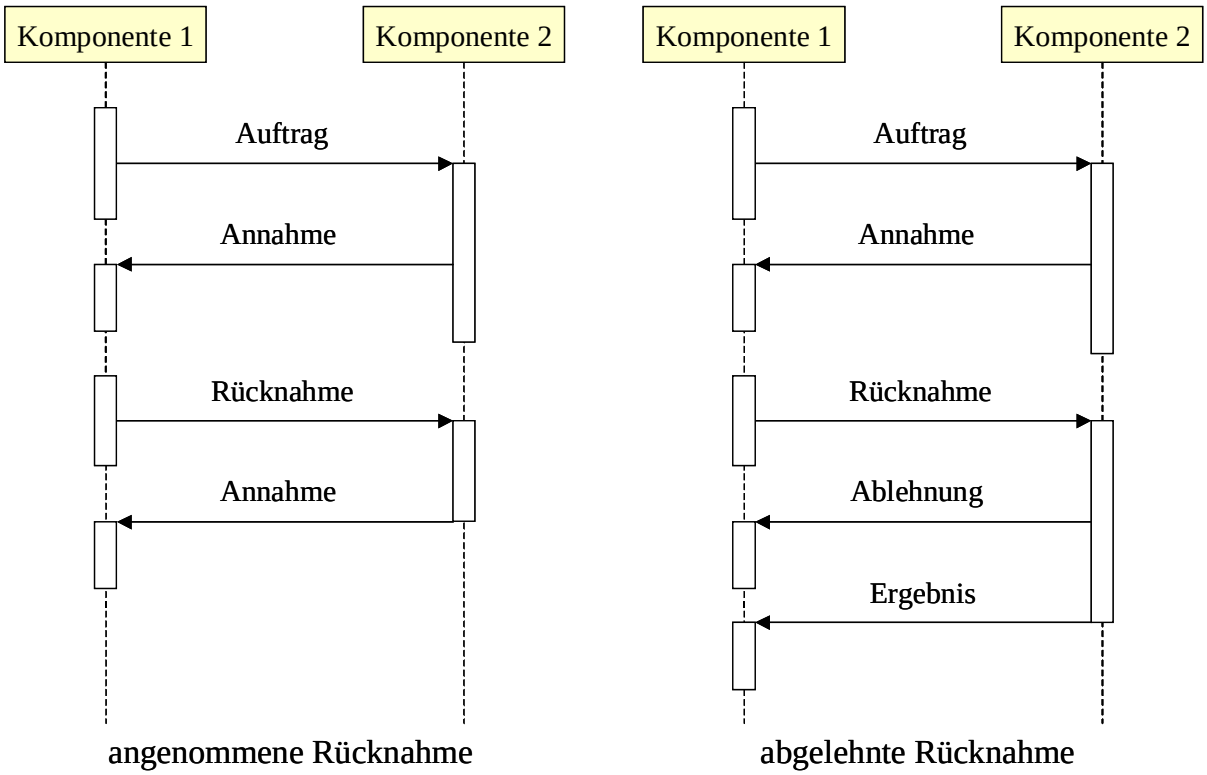

Abbildung 13: Nachrichtenschema zur Rücknahme eines Auftrags

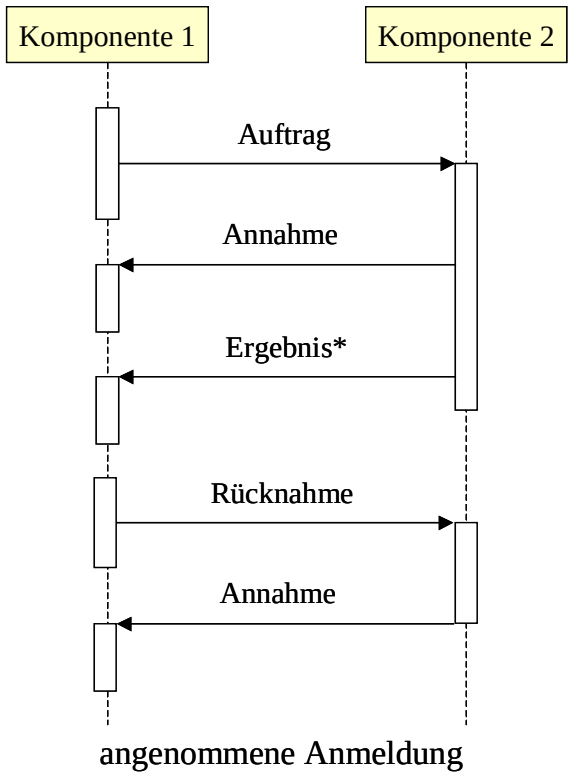

Abbildung 14: Nachrichtenschema zur Anmeldung für Benachrichtigungen

Nachrichten

Eine Nachricht enthält alle zum Versenden und Verarbeiten benötigten Informationen. Dies sind zunächst Absender und Empfänger der Nachricht in Form der jeweiligen Komponentenrolle. Diese werden nicht nur zur Zustellung der Nachricht benötigt, sondern der Empfänger verwendet sie auch zur Zuordnung zu einer seiner Komponentenrollen und zur Adressierung von Antworten. Weiterhin kann über die Priorität der Nachricht angegeben werden, wie sie zugestellt werden soll. Da üblicherweise die Warteschlange einer Komponente die Reihenfolge des Eintreffens der Nachrichten widerspiegeln soll, werden lediglich drei Prioritätsstufen unterschieden. Außer einfachen Nachrichten gibt es noch priorisierte, die bevorzugt verarbeitet werden, und direkte Nachrichten, die die Nachrichtenschlange umgehen[22]. Auf diese Weise kann zeitkritischen Vorgängen der Vorrang gegeben werden vor weniger dringlichen oder sogar vor allen anderen.

Für die Zuordnung von Antworten besitzen diese eine Referenz auf die Ausgangsnachricht, so daß der Empfänger einer Antwort den nötigen Kontext zu deren Verarbeitung ableiten kann. Da Antworten nicht unbedingt dieselbe Dringlichkeit zu besitzen brauchen wie die Ausgangsnachricht, kann eine eigene Antwortpriorität angegeben werden.

Nachrichten besitzen einen Typ, der deren Inhalt und die Art ihrer Verarbeitung festlegt. Der Inhalt umfaßt die zu übertragenden Informationen sowie gegebenenfalls zusätzliche Angaben zur Verarbeitung der Nachricht. Für die Verarbeitung ist jedem Nachrichtentyp eine Empfängerschnittstelle zugeordnet, die der Empfänger umzusetzen hat und über die er schließlich die Nachricht zur Verarbeitung von der Basisschnittstelle erhält. Außerdem geschieht die Vergabe von Sendeberechtigungen in den Komponentenrollen relativ zu diesem Nachrichtentyp.

Nachrichtenzustellung

Die Infrastrukturkomponente Nachrichtenzustellung übernimmt die Zustellung von Nachrichten zwischen Komponenten. Dazu besitzt sie eine Schnittstelle, an die die Komponenten zu versendende Nachrichten über ihre Basisschnittstelle übergeben können, und verwendet ihrerseits die Basisschnittstelle, um zu empfangende Nachrichten bei einer Komponente abzuliefern. Sämtliche vorhandenen Komponenten eines Agenten sind mit ihren Rollen bei der Nachrichtenzustellung registriert, so daß diese eine Zuordnung zwischen einer adressierten Rolle und der implementierenden Komponenteninstanz vornehmen kann. Dazu bietet die Nachrichtenzustellung eine Schnittstelle zur Registrierung von Komponenten an, die vom Agentenkern genutzt wird.

---

[22] Direkte Nachrichten werden insbesondere für Interaktionen zwischen Komponenten außerhalb der Aktivitätsphasen benötigt, beispielsweise während der Übergänge zwischen verschiedenen Lebenszykluszuständen.

Erhält die Nachrichtenzustellung eine Nachricht, so überprüft sie zunächst die Identität des Absenders, indem sie die von der Komponente angegebene Rolle mit den bei ihr für diese registrierten Komponentenrollen vergleicht. Auf diese Weise wird gewährleistet, daß keine falschen Absender möglich sind und somit Antwortnachrichten immer richtig adressiert werden können. Weiterhin wird überprüft, ob für die angegebene Empfängerrolle eine Komponente existiert, ob sie Nachrichten dieses Typs empfangen und verarbeiten kann sowie ob der Absender eine entsprechende Berechtigung besitzt[23]. Nur wenn all dies erfüllt ist, wird die Nachricht an den Empfänger zugestellt, andernfalls wird dem Absender mitgeteilt, daß sie nicht zustellbar ist. Die Zustellung der Nachricht geschieht durch das Einfügen der Nachricht in die Warteschlange der Komponente unter Berücksichtigung der Priorität. Nur bei einer direkten Nachricht wird diese sofort verarbeitet.

Zur Adressierung von Nachrichten kann nicht nur eine Rolle, sondern auch eine Rollengruppe inklusive der obersten Rollengruppe als Empfänger verwendet werden. In diesem Fall erhalten alle im Agenten vorhandenen Rollen, die zu dieser Gruppe gehören, eine Kopie der Nachricht, in der diese selbst als Empfänger angegeben ist. Die Kopie der Nachricht ist notwendig, damit in jedem Fall der Adressat eindeutig ist, auch wenn eine Komponente mehrere Rollen einnimmt. Entsprechend kann eine Komponente dabei auch mehrere Kopien der Nachrichten erhalten, nämlich jeweils genau eine für jede ihrer Rollen, die zur Rollengruppe gehört.

Spezifikation von Nachrichten

Definition 4: Nachrichten

> Eine Nachricht n ist ein 7-Tupel $[r_s, r_e, p, p_a, n_r, t, i]$. r_s und r_e sind die Rollen von Absender und Empfänger; als Empfänger sind auch Rollengruppen zulässig. p ist die Priorität zur Zustellung der Nachricht, p_a die für Antworten zu verwendende Priorität. Die Referenznachricht n_r ermöglicht den Bezug auf eine andere Nachricht. Der Nachrichtentyp t legt das Format für den Inhalt i fest.

Eine Antwort n_a auf eine Nachricht n hat die Form $[r_e, r_s, p_a, p', n, t', i']$. Die Rollen von Absender und Empfänger werden vertauscht und als Priorität die Antwortpriorität der Ausgangsnachricht n verwendet, die auch die Referenznachricht ist. Nur die neue Antwortpriorität p', der Nachrichtentyp t' und der Inhalt i' einer Antwortnachricht ergeben sich nicht direkt aus der Ausgangsnachricht.

Spezifikation der Nachrichtenzustellung

Die Nachrichtenzustellung verwaltet die Adressen der Komponenten:

---

[23] Aus Effizienzgründen kann auf die Überprüfung von Senderidentität und Sendeberechtigung verzichtet werden, da diese in einem korrekt implementierten Agenten nie verletzt werden.

- Das Adreßverzeichnis $A = \{a_1, \ldots, a_n\}$ ist eine Menge von Adreßeinträgen. Jeder Eintrag a_i ist ein Paar $[r_i, k_i]$, das der Rolle r_i die Komponente k_i zuordnet. Die Rollen sind paarweise verschieden ($\forall i, j \in \{1, \ldots, n\}: i \neq j \rightarrow r_i \neq r_j$), für die Komponenten braucht dies nicht zu gelten.

Als Schnittstelle zum Agentenkern sind folgende Methoden gegeben:

Methode	boolean register(Component)
Beschreibung	meldet eine Komponente bei der Nachrichtenzustellung an
Parameter	Component: die anzumeldende Komponente k = [b, [n, R, E]]
Rückgabewert	true, falls die Komponente angemeldet wurde, sonst false
Änderung	$A = A \cup \{[r, k] \mid r \in R\}$, falls $\forall [r', k'] \in A: r' \notin R$

Methode	boolean unregister(Component)
Beschreibung	meldet eine Komponente bei der Nachrichtenzustellung ab
Parameter	Component: die abzumeldende Komponente k = [b, [n, R, E]]
Rückgabewert	true, falls die Komponente abgemeldet wurde, sonst false
Änderung	$A = A \setminus \{[r, k] \mid r \in R\}$, falls $\exists r \in R: [r, k] \in A$

Die Schnittstelle zu den Basisschnittstellen der Komponenten besteht aus genau einer Methode:

Methode	boolean send(Message)
Beschreibung	versendet eine Nachricht
Parameter	Message: die zu versendende Nachricht $n = [r_s, r_e, p, p_a, n_r, t, i]$
Rückgabewert	true, falls die Nachricht zugestellt wurde, sonst false
Änderung (Rolle als Empfänger)	n ist zustellbar, wenn gilt: $\exists k_s$ mit $[r_s, k_s] \in A$ und $\exists k_e$ mit $[r_e, k_e] \in A$ und $\exists [t, R] \in$ Sendeberechtigungen(r_e): $(\{r_s\} \cup$ Gruppen$(r_s)) \cap R \neq \emptyset$
Änderung (Rollengruppe als Empfänger)	n ist zustellbar, wenn gilt: $\exists k_s$ mit $[r_s, k_s] \in A$ eine Nachricht $n' = [r_s, r, p, p_a, n_r, t, i]$ erhalten dann alle Rollen r mit: $\exists k$ mit $[r, k] \in A \wedge r_e \in$ Gruppen(r) und $\exists [t, R] \in$ Sendeberechtigungen(r): $(\{r_s\} \cup$ Gruppen$(r_s)) \cap R \neq \emptyset$

3.5.3. Direkte Interaktion

Auch wenn die Interaktion über Nachrichten grundsätzlich vorzuziehen ist, kann es dennoch Fälle geben, in denen eine zeitlich nicht entkoppelte Interaktion von Vorteil ist. Dazu zählen Prozesse wie der Zugriff auf eine globale Speicherkomponente des Agenten, die einerseits häufig auftreten und schnell veränderliche Informationen betreffen, andererseits aber nicht allzu aufwendig und langwierig sind. Bei der direkten Interaktion wartet die initiierende Komponente auf das Ergebnis, wodurch zwar die Komponente zeitweilig blockiert wird, sie dafür aber an derselben Stelle in

der Ausführung fortfahren kann, ohne jeweils den relevanten Kontext für eine Antwortnachricht ermitteln zu müssen.

Um die Austauschbarkeit der Komponente, auf die ein direkter Zugriff ermöglicht werden soll, dennoch zu gewährleisten, wird dazu ein Zwischenmodul mit einer entsprechenden Schnittstelle zwischengeschoben, über die der Zugriff erfolgt. Sämtliche Komponenten erhalten dann über ihre Basisschnittstelle eine Referenz auf das Zwischenmodul, das seinerseits eine Referenz auf die Komponente besitzt und diese so anonymisiert. Dadurch ist von einem Austausch der Komponente nur das Zwischenmodul betroffen, an dem lediglich die Referenz auf diese ausgetauscht zu werden braucht. Allerdings darf während des Austauschs keine Interaktion über das Zwischenmodul stattfinden, um eine konsistente Übernahme des internen Zustands der Komponente beim Austausch zu gewährleisten, so daß gegebenenfalls der gesamte Agent zuvor in einen nichtaktiven Lebenszykluszustand überführt werden muß.

Analog zur Nachrichtenzustellung kann eine zentrale Komponente die Verwaltung derartiger Mittlermodule übernehmen, so daß deren Menge offen und dynamisch veränderbar bleibt. Bei einer expliziten Beschränkung auf einige wenige Zwischenmodule kann auch eine statischere Lösung gewählt werden, bei der die Basisschnittstelle jedes dieser Module eigens unterstützt. In jedem Fall sind derartig zeitlich eng gekoppelte Abhängigkeiten zwischen Komponenten zu vermeiden, da deren Kombination die internen Abläufe sehr komplex werden lassen kann und damit die nachfolgend aufgestellte Forderung der Einteilung aller Vorgänge in hinreichend kleine Einzelschritte beeinträchtigt.

3.6. Ausführungskontrolle

Zunächst werden im folgenden einige Anforderungen erörtert, die sich aus dem Aufbau eines Agenten aus Komponenten und dem nachrichtenbasierten Interaktionsmechanismus für die Kontrolle der Aktivitäten der einzelnen Komponenten und des Agenten als ganzes ergeben. Danach wird dargestellt, wie die Architektur diesen Anforderungen gerecht wird, indem sie zentrale Kontrollmechanismen und eine Überwachung des Beschäftigungszustands bereitstellt. Zum Abschluß wird dann noch das Zusammenspiel der Komponenten und Schnittstellen der internen Agenteninfrastruktur bei der Zustellung und Verarbeitung von Nachrichten zusammengefaßt.

3.6.1. Anforderungen

Da die Interaktionen zwischen den Komponenten auf dem Austausch von Nachrichten basieren, ist auch die zentrale Aktivität der einzelnen Komponenten die sukzes-

sive Verarbeitung der Nachrichten, die in ihrer Warteschlange enthalten sind. Darüber hinaus darf jede Komponente aber auch parallel weitere Aktivitäten ausführen, die gegebenenfalls untereinander und mit der Nachrichtenverarbeitung zu synchronisieren sind. Der Kontrollmechanismus soll also einerseits die Verarbeitung von Nachrichten unterstützen, andererseits zusätzliche komponentenspezifische Prozesse zulassen sowie gegebenenfalls beide miteinander koordinieren.

Die wichtigste Anforderung an die Aktivitäten der Komponenten ist deren kontrollierte Unterbrechbarkeit. Sowohl zur Persistenz oder Migration des Agenten, als auch beim Entfernen oder Austauschen von Komponenten müssen diese einen stabilen Zustand einnehmen, der sich nicht mehr durch laufende Prozesse verändern kann, damit keine Inkonsistenzen durch Nebenläufigkeiten entstehen können. Dazu ist es notwendig, daß alle Aktivitäten in hinreichend kleine Schritte untergliedert sind, zwischen denen eine Unterbrechung und Wiederaufnahme problemlos möglich ist. Die Ausführung der einzelnen Schritte muß zentral kontrolliert werden können.

Die einzelnen Verarbeitungsschritte verschiedener Komponenten sind sowohl sequentiell nacheinander, als auch parallel zueinander ausführbar. Der Vorteil einer sequentiellen Verarbeitung besteht vor allem in der zentralen Steuerung, die eine genauere Kontrolle und Ressourcenzuteilung ermöglicht. Die Vorgänge werden dadurch nachvollziehbarer und vorhersehbarer. Dafür besteht die Gefahr, daß eine Komponente alle übrigen blockiert, falls ein Schritt beispielsweise aufgrund eines Fehlers nicht beendet wird. Die parallele Ausführung vermeidet derartige Beeinträchtigungen von Komponenten durch andere, dies wird jedoch auf Kosten der Kontrollierbarkeit der Prozesse erreicht. Die Art der Ausführung und die Zuteilung von Verarbeitungsressourcen sollte für jede Komponente möglichst frei wählbar sein, um für den Agenten als ganzes je nach Zusammensetzung jeweils eine möglichst optimale Kontrollstrategie realisieren zu können. Dazu ist eine zentrale Steuerung und Konfiguration notwendig.

Für zeitlich nicht entkoppelte Prozesse zwischen Komponenten – direkte Interaktionen und direkte Nachrichten – ist keine zentral gesteuerte Ausführung möglich, da diese ausschließlich durch die initiierende Komponente kontrolliert werden. Auch aus diesem Grund sollten zeitlich nicht entkoppelte Interaktionen nur unter den genannten Umständen verwendet werden.

3.6.2. Kontrollmechanismen

Zur Realisierung der zentral gesteuerten Kontrollmechanismen gibt es eine Infrastrukturkomponente, den Kontrollzyklus, die auf den durch die Basisschnittstellen der Komponenten bereitgestellten Funktionalitäten aufbaut. Sie kontrolliert die Ausführung einzelner Verarbeitungsschritte, vor allem zur Nachrichtenverarbeitung.

Untergliederung in Einzelschritte

Die Gliederung sämtlicher Prozesse in einzelne, hinreichend kleine Schritte läßt sich praktisch nicht durch die Architektur erzwingen[24]. Zum einen lassen sich laufende Prozesse nicht beliebig unterbrechen und fortsetzen, zum anderen läßt sich weder die im Einzelfall gegebene, noch die optimale Schrittgröße absolut bestimmen, zumal bereits die Wahl eines objektiven Maßes nicht leicht fällt. Die Ausführungszeit – unter dem Gesichtspunkt der Austauschbarkeit der Komponenten ein naheliegendes Kriterium – hängt beispielsweise von vielen Faktoren ab, die sowohl von der Ausführungsumgebung, als auch von den jeweiligen Ausführungsparametern bestimmt sein können.

Somit ist die Gliederung in sinnvolle Einzelschritte vor allem eine Programmierkonvention, deren Einhaltung und Durchführung von der Architektur unterstützt wird. Dies geschieht zum einen über die Kontrollmechanismen, über die einzelne Schritte der Komponenten ausgeführt werden, und zum anderen über den Beschäftigungszustand, der angibt, ob sich eine Komponente gerade in der Ausführung eines Schritts befindet oder nicht.

Für die Nachrichtenverarbeitung wird die Verarbeitung einer einzelnen Nachricht als Schrittgröße angenommen. Da der Aufwand zur Verarbeitung je nach Nachrichtentyp, Inhalt und verarbeitender Komponente stark variieren kann, sollte die zentral gesteuerte Nachrichtenverarbeitung lediglich als erster Schritt angesehen werden, dem bei aufwendiger Ausführung die Komponente unter eigener Kontrolle weitere Schritte folgen läßt, um für eine fein genuge Granularität der Schritte zu sorgen.

Kontrollzyklus

Der Kontrollzyklus stellt die Verarbeitungsressourcen[25] für alle zentral kontrollierten Aktivitäten bereit. Zur Nachrichtenverarbeitung ist jede Komponente entweder in den Kontrollzyklus eingegliedert oder erhält eigene Verarbeitungsressourcen, was auch noch zur Laufzeit gewechselt werden kann. Sämtliche vom Kontrollzyklus kontrollierten Ressourcen führen an den zugeordneten Komponenten über deren Basisschnittstelle sukzessive einzelne Schritte aus, jedoch nur dann, wenn sich der Agent bzw. die jeweilige Komponente im Lebenszykluszustand *active* befindet.

---

[24] Implementierungssprachen für sequentielle Rechnerarchitekturen besitzen zwangsläufig eine Unterteilung in einzelne, nacheinander ausgeführte Operationen. Diese haben jedoch eine zu feine Granularität und lassen sich i.a. nur schwer von der Ebene der Implementierungssprache selbst aus nutzen, so daß sie zur Umsetzung der hier benötigten Unterteilung in Einzelschritte nicht geeignet sind.

[25] Den Verarbeitungsressourcen entsprechen auf der Implementierungsebene aktive Prozesse (Threads). Je nach Implementierungssprache läßt es sich nicht verhindern, daß Komponenten eigene Prozesse starten. Dies ist unproblematisch, solange diese sich an die Untergliederung in Einzelschritte und die Deklaration des Beschäftigungszustands halten.

Der Kontrollzyklus selbst vergibt seine Ressourcen nacheinander jeweils an eine der in ihm enthaltenen Komponenten zur Ausführung genau eines Schritts, der entweder in der Verarbeitung einer Nachricht oder einer komponentenspezifischen Aktivität besteht. Dabei werden nur die Komponenten berücksichtigt, die gerade Nachrichten in ihrer Warteschlange enthalten oder über die Basisschnittstelle den Bedarf zur Ausführung individueller Schritte angemeldet haben. Parallel dazu führen die übrigen Komponenten jeweils mit ihren eigenen Ressourcen Einzelschritte aus, sofern Bedarf dazu besteht. Auf diese Weise kann für jede Komponente eine sequentielle oder parallele Ausführungsart innerhalb des Agenten gewählt werden.

Kontrollzyklusstrategien

Für die Zuteilung der Verarbeitungsressourcen durch den Kontrollzyklus an die ihm zugeordneten Komponenten können verschiedene Strategien gewählt werden. Die wichtigste Anforderung an jede Strategie ist dabei Fairneß, d.h. es muß gewährleistet sein, daß jede Komponente, die einen Schritt auszuführen hat, dazu nach endlicher Zeit auch Gelegenheit erhält – vorausgesetzt, daß der Agent lange genug aktiv ist. Eine völlige Gleichbehandlung aller Komponenten ist jedoch nicht zwingend notwendig, da je nach Funktion oder Beanspruchung einer Komponente eine bevorzugte Behandlung sinnvoll sein kann. Diese darf aber nicht dazu führen, daß andere Komponenten völlig von der Ressourcenzuteilung ausgeschlossen werden können, solange die bevorzugten Komponenten Ressourcen beanspruchen.

Die einfachste Garantie für eine faire Zuteilung ist die Gleichbehandlung aller Komponenten. Dazu erhalten sämtliche Komponenten des Kontrollzyklus reihum und damit gleich oft die Gelegenheit zur Ausführung genau eines Schritts. Diese Strategie ist jedoch insofern unausgewogen, als sie nicht den unterschiedlichen Bedarf der einzelnen Komponenten berücksichtigt. Dieser Bedarf kann in die Strategie mit einbezogen werden, indem den Komponenten a priori ein fester Anteil an den Verarbeitungsressourcen zugeteilt wird, so daß jede Komponente je nach Anteil pro Durchlauf durch den Zyklus einen oder mehrere Schritte verarbeiten darf[26]. Bei diesem Anteil kann auch explizit für jede Komponente zwischen den Schritten zur Nachrichtenverarbeitung und für individuelle Aktivitäten unterschieden werden.

Für die Nachrichtenverarbeitung ist auch eine dynamische Ressourcenzuteilung je nach tatsächlichem Bedarf anhand der in den Warteschlangen der einzelnen Komponenten enthaltenen Nachrichten möglich. Dabei können verschiedene Faktoren wie die Anzahl der wartenden Nachrichten der einzelnen Komponenten, die Priorität der wartenden Nachrichten sowie die globale Reihenfolge, in der die Nachrichte versendet wurden, berücksichtigt werden. Durch eine anteilige Ressourcenzuteilung abhängig von der Anzahl der wartenden Nachrichten wird erreicht,

[26] Es läuft auf dasselbe hinaus, wenn Komponenten mit geringerem Anteil an den Ressourcen jeweils eine bestimmte Zahl an Durchläufen keinen Schritt ausführen dürfen.

daß diese Anzahl bei allen Komponenten gleichmäßig gegen Null tendiert. Die Bevorzugung von Komponenten mit wartenden priorisierten Nachrichten weitet deren Priorisierung auf sämtliche Komponenten des Kontrollzyklus aus. Die Einhaltung der ursprünglichen Reihenfolge unter Berücksichtigung der Priorität garantiert ein hohes Maß an Fairneß, verlangt dafür aber eine globale Verwaltung aller Nachrichten oder einen globalen Zeitstempel für Nachrichten. In den beiden anderen Fällen hingegen muß die Fairneß durch zusätzliche Maßnahmen gewährleistet werden, beispielsweise indem eine maximale Anzahl an Schritten vorgegeben wird, die ausgeführt werden darf, ohne daß jede Komponente mindestens einmal die Gelegenheit zur Schrittausführung erhalten hat.

Neben dem Bedarf kann der Kontrollzyklus auch den Ressourcenverbrauch der Komponenten berücksichtigen, der jedoch – wie bereits im Zusammenhang mit der Schrittgröße erwähnt – nur schwer sicher zu bestimmen ist. Auf diese Weise läßt sich eine anteilige Zuordnung der Ressourcen relativ zum Verbrauch und nicht nur relativ zur Schrittzahl realisieren.

Der Vorteil der statischen Kontrollzyklusstrategien ist ihre Einfachheit und Fairneß. Die dynamischen Strategien sind dafür ausgewogener und orientierten sich mehr an dem tatsächlichem Bedarf. Eine in jedem Fall optimale Strategie gibt es jedoch nicht, zumal sich einige der Strategien auch kombinieren lassen. Im Sinne der Offenheit der CASA-Architekur lassen sich für den Kontrollzyklus Komponenten mit unterschiedlichen Strategien realisieren, so daß jeweils nach Bedarf eine Komponente mit einer geeigneten Strategie ausgewählt werden kann.

Schrittmodus

Im Lebenszykluszustand *stepping* befindet sich der Agent in einem Schrittmodus, bei dem die Ausführung der einzelnen Schritte des Agenten von außen über die Schnittstelle des Agentenkerns kontrolliert wird. Dazu werden sämtliche Komponenten – auch solche, denen eigene Verarbeitungsressourcen zugeteilt sind – in den Kontrollzyklus eingegliedert. Der Kontrollzyklus ist dabei nicht aus sich heraus aktiv, sondern wartet auf die über den Agentenkern weitergeleitete Vorgabe von außen, einen Schritt auszuführen. Die Auswahl der Komponente und der Art des auszuführenden Schritts geschieht nach derselben Strategie wie im aktiven Zustand. Das Ende der Schrittausführung wird wieder nach außen hin mitgeteilt und erst dann ist der nächste Schritt möglich.

Spezifikation des Kontrollzyklus

Der Kontrollzyklus verwaltet zwei Mengen an Komponenten:

- die Menge $K^K = \{k_1, \ldots, k_n\}$ der Komponenten innerhalb des Kontrollzyklus
- die Menge $K^R = \{k'_1, \ldots, k'_m\}$ der Komponenten mit eigenen Ressourcen

Die folgenden Methoden bilden die Schnittstelle zum Agentenkern:

Methode	void addComponent(Component, Mode)
Beschreibung	teilt einer Komponente Verarbeitungsressourcen zu
Parameter	Component: die hinzuzufügende Komponente k Mode: die Angabe, ob die Komponente eigene Ressourcen erhält (Thread) oder in den Kontrollzyklus eingegliedert ist (ControlCycle)
Änderung	Mode = Thread: $K^R' = K^R \cup \{k\}$ Mode = ControlCycle: $K^K' = K^K \cup \{k\}$

Methode	boolean removeComponent(Component)
Beschreibung	entzieht einer Komponente ihre Verarbeitungsressourcen
Parameter	Component: die zu entfernende Komponente k
Rückgabewert	true, falls der Komponente Ressourcen zugeteilt waren, sonst false
Änderung	$K^R' = K^R \setminus \{k\}$, falls $k \in K^R$ $K^K' = K^K \setminus \{k\}$, falls $k \in K^K$

Methode	Mode getMode(Component)
Beschreibung	gibt Auskunft über die Art der Verarbeitungsressourcen einer Komponente
Parameter	Component: die betreffende Komponente k
Rückgabewert	Thread, falls $k \in K^R$, ControlCycle, falls $k \in K^K$, None sonst

Methode	boolean setMode(Component, Mode)
Beschreibung	setzt die Art der Verarbeitungsressourcen
Parameter	Component: die zu ändernde Komponente k Mode: Thread für eigene Ressourcen oder ControlCycle für Kontrollzyklus
Rückgabewert	true, falls eine Änderung stattfindet, sonst false
Änderung	Mode = Thread: $K^R' = K^R \cup \{k\}$ und $K^K' = K^K \setminus \{k\}$, falls $k \in K^K$ Mode = ControlCycle: $K^K' = K^K \cup \{k\}$ und $K^R' = K^R \setminus \{k\}$, falls $k \in K^R$

Methode	boolean step()
Beschreibung	führt einen Schritt im Einzelschrittbetrieb aus
Rückgabewert	true, falls ein Schritt ausgeführt wurde, sonst false

Die Schnittstelle zu den Basisschnittstellen der Komponenten umfaßt folgende Methoden:

Methode	void newMessage(Component)
Beschreibung	meldet neue wartende Nachrichten einer Komponente an
Parameter	Component: die anzumeldende Komponente

Methode	void newStep(Component)
Beschreibung	meldet neue komponentenspezifische Schritte einer Komponente an
Parameter	Component: die anzumeldende Komponente

3.6.3. Beschäftigungszustand

Über ihre Basisschnittstelle verwaltet jede Komponente einen Beschäftigungszustand, der angibt, ob sie gerade Aktivitäten ausführt oder nicht. Beginn und Ende der über die Basisschnittstelle ausgeführten Schritte werden dabei automatisch registriert. Außerdem sollte jede Komponente davon unabhängig ausgeführte Aktivitäten an- und wieder abmelden, damit auch diese beim Beschäftigungszustand berücksichtigt werden. Die Aufnahme einer neuen Aktivität und dementsprechend auch deren Anmeldung ist nur zulässig und möglich, wenn die Komponente sich in einem der Lebenszykluszustände *active* oder *stepping* befindet. Im Schrittmodus müssen dabei alle Aktivitäten ausschließlich im Rahmen der Schrittausführung geschehen.

Der Beschäftigungszustand des Agenten ergibt sich aus dem seiner Komponenten. Er gilt als beschäftigt, sobald mindestens eine seiner Komponenten beschäftigt ist.

Von Bedeutung ist der Beschäftigungszustand immer dann, wenn der Agent oder eine Komponente von einem der beiden Zustände mit Aktivitäten in einen ohne überwechselt. Dieser Zustandsübergang ist erst dann vollständig vollzogen, wenn keine Beschäftigung mehr vorliegt und somit keine internen Zustandsänderungen mehr möglich sind. Erst dann ist der Agent bzw. die Komponente in einem stabilen Zustand, der Persistenz oder Migration bzw. Entfernen oder Austausch der Komponente erlaubt.

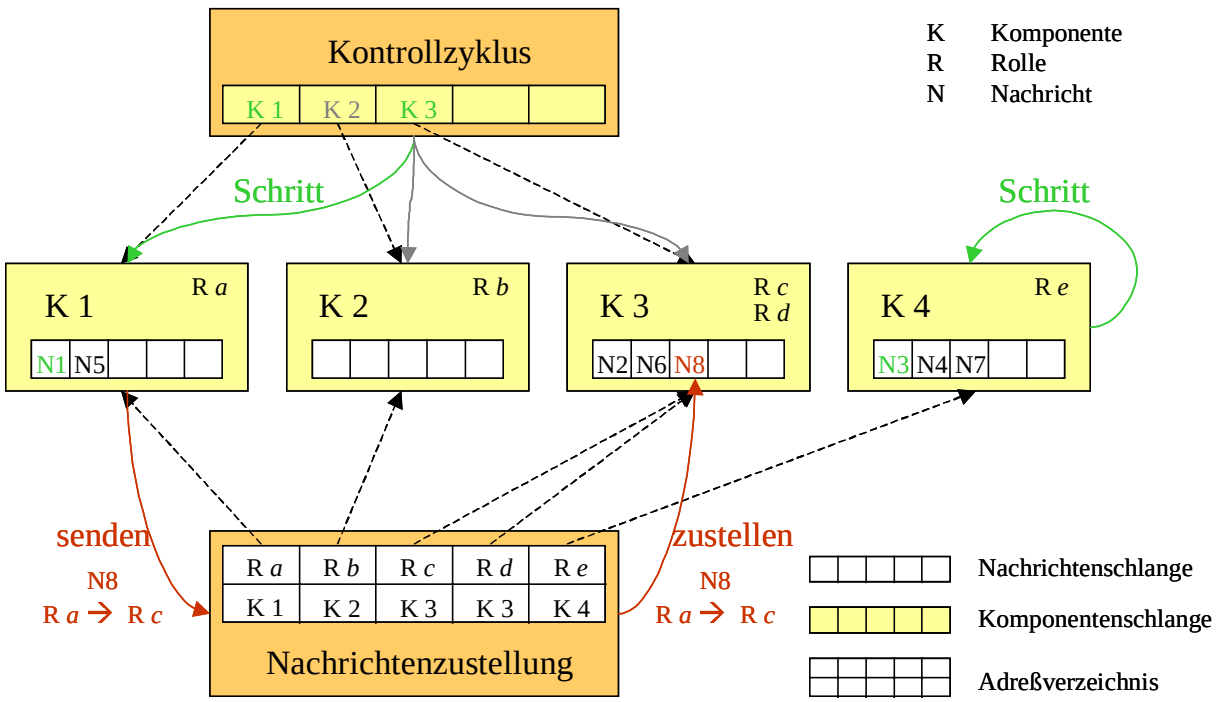

Abbildung 15: Nachrichtenaustausch und -verarbeitung

3.6.4. Nachrichtenaustausch und -verarbeitung

Abbildung 15 gibt einen zusammenfassenden Überblick über die Mechanismen für den Austausch und die Verarbeitung von Nachrichten. Der Austausch von Nachrichten geschieht über die Nachrichtenzustellung, ihre Verarbeitung wird entweder durch die Komponente selbst oder durch den Kontrollzyklus kontrolliert.

Um eine Nachricht zu versenden, übergibt eine Komponente diese an die Nachrichtenzustellung. Diese ermittelt anhand der Zuordnung von Komponenten zu Rollen die Empfängerkomponente und fügt die Nachricht in deren Warteschlange ein. Dieser Vorgang ist in Abbildung 15 mit der Nachricht N8 von Rolle *a* nach *c* unten in rot eingezeichnet, die oberen, grünen Pfeile stellen die Nachrichtenverarbeitung dar. Diese geschieht entweder über den Kontrollzyklus (links) oder parallel zu diesem durch eigene Verarbeitungsressourcen der Komponente (rechts). In beiden Fällen wird über die Basisschnittstelle die Schrittausführung angestoßen, woraufhin diese die erste Nachricht aus der Warteschlange entnimmt und sie über die zu deren Typ gehörige Empfängerschnittstelle der Komponente zur Verarbeitung übergibt.

3.7. Zusammenfassung

Der strukturelle Aufbau einzelner Agenten ist aufgrund der Modularität der Komponentenarchitektur offen und skalierbar. Dadurch lassen sich je nach Bedarf und Aufgabe verschiedene Agententypen realisieren. Gemäß dem Baukastenprinzip wird ein Agent aus streng voneinander gekapselten Komponenten zusammengesetzt, die anhand ihrer deklarierten Eigenschaften an den jeweiligen Agenten angepaßt werden können. Einmal erstellte Komponenten lassen sich auf diese Weise wiederverwenden, so daß Bibliotheken mit generischen Komponenten bereitgestellt werden können, um die Entwicklungszeit zu verringern. Und auch noch in laufenden Systemen können Agenten ohne Beeinträchtigung umkonfiguriert und erweitert werden, indem die Zusammensetzung des Agenten und die Eigenschaften einzelner Komponenten geändert werden.

Erreicht wird diese Offenheit und Skalierbarkeit durch die Entkoppelung der Komponenten untereinander anhand von Rollen und Nachrichten. Eine Rolle dient als Schnittstellenspezifikation sowohl für die Konfiguration über Eigenschaften, als auch für die Interaktion zwischen den Komponenten über die verarbeitbaren Nachrichtentypen. Dabei anonymisieren die Rollen die ihre Funktionalität realisierenden Komponenten, so daß Abhängigkeiten zwischen den Komponenten auf deren Rollen beschränkt sind, die zur Adressierung von Nachrichten verwendet werden. Durch den Austausch von Nachrichten geschehen die Interaktionen asynchron, um auch zeitliche Abhängigkeiten zu reduzieren. Nachrichtentypen legen den Inhalt und die Bedeutung von Nachrichten fest und werden nach vorgegebenen

Schemata ausgetauscht. Schließlich eignet sich die Ebene von Rollen und Nachrichtentypen auch zur Definition abstrakter Kontrollarchitekturen, die auf funktionale Abhängigkeiten beschränkt sind und durch konkrete Komponenten umgesetzt werden, die diese Rollen ausfüllen. Rollengruppen ermöglichen dabei eine zusätzliche Strukturierung.

Die Agentenhülle stellt die interne Infrastruktur eines Agenten bereit, die von den Komponenten über eine einheitliche Basisschnittstelle genutzt werden kann. Sie bildet die eigentliche Komponentenarchitektur und besteht ihrerseits aus austauschbaren Komponenten. Die Verwaltung der Komponentenstruktur und der einzelnen Komponenten obliegt dem Agentenkern. Dieser erzeugt und repräsentiert den Agenten in seiner Gesamtheit, wozu auch die Kontrolle über den Lebenszyklus und die Konfiguration des Agenten und seiner Komponenten gehört. Die Nachrichtenzustellung ermöglicht Interaktionen zwischen Komponenten über den Austausch von Nachrichten, wobei sie die Zuordnung von Rollen zu Komponenten durchführt. Die Ressourcen zur Verarbeitung von Nachrichten und für komponentenspezifische Aktivitäten verwaltet der Kontrollzyklus. Er kontrolliert den Ablauf aller Prozesse durch Ausführung einzelner Schritte, was die regelmäßige Unterbrechbarkeit der Aktivitäten des Agenten sicherstellt.

Insgesamt bildet die Komponentenarchitektur von CASA einen flexiblen Rahmen zur Gestaltung von Agenten, die wiederverwendbare Funktionalitäten und Grundstrukturen anwendungsspezifisch erweitern und auch noch zur Laufzeit ohne Beeinträchtigung des laufenden Systems umgestaltet werden können.

4. Der Einzelagent

Auf der Ebene des Einzelagenten ist ein Agent vor allem charakterisiert durch die Mechanismen zur autonomen Kontrolle seines Verhaltens. Im folgenden wird hierfür zunächst ein Ansatz zur Verhaltenssteuerung basierend auf explizit repräsentierten Wissensstrukturen (vgl. Kapitel 2.3.1) vorgestellt, über den sich reaktive, deliberative und interaktive Verhaltensweisen beschreiben lassen.

Darauf aufbauend wird eine Standardarchitektur für CASA-Agenten vorgeschlagen, die als Grundgerüst für wissensbasierte Agenten verwendet werden kann. Gemäß dem Komponentenansatz definiert sich die Standardarchitektur über eine Menge von Komponentenrollen und deren Interaktionsmöglichkeiten untereinander. Die Beschreibung der einzelnen Rollen umfaßt entsprechend neben deren Funktionalität vor allem die verarbeitbaren Nachrichtentypen und deren Inhalt.

4.1. Wissensbasierte Verhaltenssteuerung

Ein wissensbasierter Agent ist definiert zum einen über die unterschiedlichen Arten von Wissen, die er repräsentieren kann, zum anderen über die Aktivitäten, die er aufgrund der Inhalte dieser Wissensstrukturen ausführen kann. Der Begriff „Wissen“ bezeichnet dabei Datenstrukturen, die aufgrund ihrer formalisierten Repräsentationsweise eine automatisierte Verarbeitung erlauben, die der vom menschlichen Benutzer intendierten Bedeutung gerecht wird (vgl. Kapitel 2.3). Im folgenden wird zunächst ein Schema zur wissensbasierten Verhaltenssteuerung für Agenten dargestellt, bevor die dabei verwendeten einzelnen Wissenstypen und Kontrollfunktionen näher erläutert werden.

4.1.1. Schema zur Verhaltenssteuerung

Abbildung 16 zeigt das der CASA-Standardarchitektur zugrundeliegende Schema zur wissensbasierten Steuerung von reaktivem, deliberativem und interaktivem Verhalten. Der Agent besitzt Wissen über die Welt in Form von Fakten, die den Weltzustand in einer durch Ontologien vorgegebenen Terminologie repräsentieren.

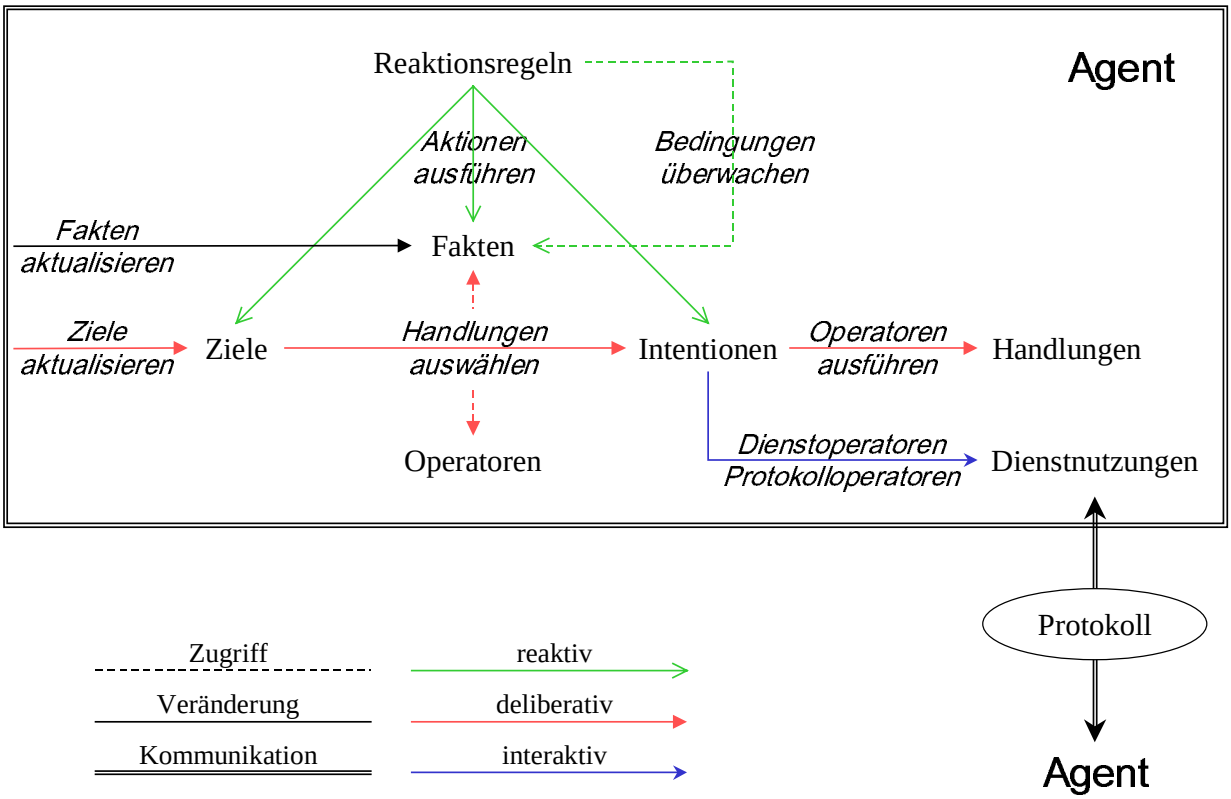

Abbildung 16: Schema zur wissensbasierten Verhaltenssteuerung

Weiterhin besitzt er Ziele zur Veränderung dieser Welt und daraus abgeleitete Intentionen, die sein momentanes Handeln bestimmen, sowie Reaktionsregeln und Operatoren, die seine Handlungsdispositionen und -möglichkeiten charakterisieren.

Der Agent aktualisiert beständig sein Faktenwissen und seine Ziele und reagiert auf die aktuelle Situation. Für seine Ziele wählt er Handlungen aus, indem er aus den vorhandenen Operatoren Intentionen ableitet. Diese werden koordiniert und schließlich ausgeführt. Ist die auszuführende Handlung eine Dienstnutzung, so kommuniziert er geleitet durch Protokolle mit anderen Agenten.

4.1.2. Wissenstypen

Dieser Ansatz zur Verhaltenssteuerung nutzt die folgenden Arten von Wissen[27]:

- *Ontologien* geben eine Terminologie zur Beschreibung von Weltzuständen vor.
- *Fakten* beinhalten Annahmen über den aktuellen Weltzustand.
- *Ziele* stellen intendierte Weltzustände dar.

[27] Auch wenn im folgenden aufgrund der naheliegenden Analogien zum menschlichen Denken für die einzelnen Wissenstypen eine mentalistische Terminologie verwendet wird, darf diese nicht zu wörtlich genommen werden, um irreführende Anthropomorphismen und überzogene Erwartungen an das Verhalten eines derartigen Systems zu vermeiden.

- *Reaktionsregeln* sind Schemata für Verhaltensänderungen, die als Reaktion auf Faktenänderungen vorgenommen werden.
- *Operatoren* beschreiben mögliche Handlungen.
  - *Pläne* bestehen aus mehreren, logisch miteinander verknüpften Operatoren.
  - *Skripte* sind aus Operatoren zusammengesetzte Handlungsschemata, deren Ablauf über prozedurale *Skriptanweisungen* kontrolliert wird.
  - *Dienstoperatoren* beschreiben mögliche Interaktionen mit anderer Agenten.
  - *Protokolle* regeln den Verlauf von Interaktionen mit anderen Agenten.
  - *Protokolloperatoren* füllen eine Rolle innerhalb eines Protokolls aus.
  - *Sprechaktoperatoren* führen kommunikative Handlungen durch.
- *Intentionen* sind beabsichtigte Handlungen.

Fakten und Ziele stellen dabei den deklarativen Anteil des Wissens dar, der die Welt als Gesamtheit einer Menge von Zuständen beschreibt, die mit der Zeit veränderlich sind. Das prozedurale Wissen repräsentiert in Form von Reaktionsregeln und Operatoren die Möglichkeiten des Agenten zur Veränderung dieser Zustände. Ontologien und Intentionen verbinden deklaratives und prozedurales Wissen. Ontologien bilden dabei das gemeinsame Vokabular zur Formulierung von deklarativem und prozeduralem Wissen, während Intentionen die Anwendung der Operatoren auf die Welt beinhalten.

Im folgenden wird von einer spezifischen Wissensrepräsentationssprache zunächst noch weitgehend abstrahiert, um den allgemeinen Charakter des Ansatzes nicht einzuschränken. Von dem Formalismus zur Wissensrepräsentation wird lediglich angenommen, daß sich in ihm zumindest Aussagen ausdrücken lassen, die aus primitiven Fakten und deren Negation sowie aus Konjunktionen positiver und negativer Fakten bestehen. Fakten beziehen sich dabei auf Objekte und deren Eigenschaften, die jeweils auch durch Variablen repräsentiert werden können, die falls nicht anders deklariert als existenz-quantifiziert aufgefaßt werden. Ein Beispiel für eine derartige Sprache ist die Prädikatenlogik erster Stufe. Kapitel 6 stellt eine diesen Anforderungen genügende Wissensrepräsentationssprache für CASA-Agenten vor, die sämtliche hier verwendeten Wissenstypen umfaßt.

Der zeitliche Aspekt spielt in dieser Wissensrepräsentation keine explizite Rolle. Fakten, Regeln und Operatoren beziehen sich jeweils auf die Gegenwart bzw. die direkte Vergangenheit, Ziele und Intentionen auf die nahe Zukunft.

Ontologien

Eine Ontologie stellt ein Vokabular und ein Schema zur Repräsentation einer bestimmten Domäne bereit und gibt damit die Form des Wissens über diesen Bereich der Welt vor. Ontologien sind somit kein inhaltliches Wissen, sondern sie strukturieren das mögliche Wissen eines Agenten im voraus.

Eine besondere Rolle spielen Ontologien bei der Kommunikation zwischen Agenten. Um die Verarbeitung des Inhalts von Sprechakten zu gewährleisten, muß nicht nur eine gemeinsame Sprache, sondern auch ein gemeinsames Vokabular verwendet werden, das durch gemeinsame Ontologien bereitgestellt wird.

Fakten

Fakten repräsentieren die Annahmen eines Agenten über den gegenwärtigen Zustand der Welt. Das Faktenwissen besteht aus einer konsistenten Menge einzelner Fakten, die jeweils das Bestehen oder Nicht-Bestehen eines bestimmten Zustands beinhalten. Die Form der Fakten wird durch Ontologien vorgegeben, die mögliche Eigenschaften von Objekten anhand von Kategorien festlegen (vgl. Kapitel 2.3.1). Ein einzelner Zustand besteht jeweils in einer konkreten Eigenschaft eines Objekts.

Das Faktenwissen eines Agenten kann weder als vollständig, noch als korrekt angenommen werden. Es spiegelt nur den beschränkten Ausschnitt des gesamten Weltzustands wider, über den der Agent zu diesem Zeitpunkt Annahmen besitzt, alles übrige ist ihm unbekannt. Außerdem braucht sich die Welt nicht in dem Zustand zu befinden, den der Agent annimmt.

Ziele

Ein Ziel beschreibt einen möglichen Zustand, dessen Bestehen der Agent beabsichtigt. Dieser Zustand wird als eine Aussage in der Wissensrepräsentationssprache formuliert und gilt als erreicht, sobald diese Aussage relativ zum Faktenwissen des Agenten erfüllt ist. Neben diesen Zustandszielen gibt es noch Interaktionsziele, deren Inhalt die Durchführung einer Interaktion mit einem anderen Agenten ist.

Das Bestehen eines Ziels schließt nicht aus, daß der Zielzustand bereits eingetreten ist, ohne daß der Agent dies weiß. Es besagt auch nichts darüber, ob der Agent die Fähigkeit besitzt, diesen Zustand alleine oder mit Hilfe anderer Agenten herbeizuführen oder nicht. Die Gesamtheit der Ziele eines Agenten braucht nicht konsistent zu sein, da einander widersprechende Ziele zwar nicht gleichzeitig, dafür aber nacheinander erreicht werden können.

Reaktionsregeln

Das reaktive Wissen eines Agenten wird über Reaktionsregeln beschrieben. Eine Regel besteht dabei aus zwei Teilen, einer Bedingung und einer Aktion. Die Bedingung ist eine Aussage, die eine Situation beschreibt, deren Eintreten eine Reaktion des Agenten verlangt.

Die durch die Erfüllung der Bedingung durch das Faktenwissen auszulösende Aktion bewirkt eine Veränderung im Wissen des Agenten auf der Ebene von Fakten, Zielen oder Intentionen. Auf diese Weise kann ein Agent sein Faktenwissen und

seine Ziele an die jeweilige Situation anpassen oder durch Handlungen auf diese reagieren.

Operatoren

Die Handlungsmöglichkeiten eines Agenten werden dargestellt über Operatoren. Ein Operator besteht neben einem Ausführungsteil, der zur Ausführung interpretiert wird, aus einer formalen Beschreibung der Handlung. Diese umfaßt in Form von Aussagen über die jeweiligen Zustände die Vorbedingung, die Rahmenbedingung und die möglichen Effekte der Handlung.

Vorbedingung und Rahmenbedingung beschreiben die Umstände, unter denen eine Handlung ausgeführt werden kann. Beide Bedingungen müssen vor, die Rahmenbedingung auch während der gesamten Ausführung erfüllt sein. Die Effekte beschreiben die möglichen Zustände, die nach einer erfolgreichen Handlungsausführung eintreten können. Eine Handlung ist deterministisch, wenn sie nur genau einen Effekt besitzt, was den Normalfall darstellt.

Die formale Beschreibung der Handlungen ermöglicht eine automatische Auswahl von Operatoren zum Erreichen von Zielen sowie deren Verknüpfung zu Plänen.

Pläne

Ein Plan ist eine Menge von Operatoren, deren Ausführungsreihenfolge über logische Abhängigkeiten der Operatoren untereinander festgelegt ist. Aus der Struktur eines Plans lassen sich Vorbedingung, Rahmenbedingung und Effekte des Plans als ganzes ableiten, so daß sich auch ein Plan wiederum als Operator auffassen und verwenden läßt.

Skripte und Skriptanweisungen

Ein Skript ist ein Operator, der einen komplexen Handlungsverlauf festlegt. Dazu werden Operatoren durch Skriptanweisungen miteinander verknüpft, die den Skriptablauf steuern. Die Menge der Skriptanweisungen und die sich daraus ergebende Struktur von Skripten wird durch eine Skriptsprache vorgegeben.

Dienstoperatoren

Während normale Operatoren diejenigen Handlungen beschreiben, die ein Agent alleine ausführen kann, beschreiben Dienstoperatoren solche Handlungen, die ein Agent zur Ausführung für einen anderen anbietet.

Die formale Beschreibung geschieht dabei aus der Sicht des Nutzers, so daß dieser Dienstoperatoren analog zu anderen Operatoren verwenden kann. Der einzige Unterschied besteht dann in der Ausführung, bei der der Anbieter des Dienstes den Effekt des Operators herbeiführt, nachdem der Nutzer ihn dazu beauftragt hat.

Protokolle

Die zur Durchführung von Dienstnutzungen notwendigen Interaktionen zwischen Agenten werden durch Protokolle gesteuert. Ein Protokoll beschreibt ein Interaktionsmuster relativ zu einer Menge von Rollen, die von den beteiligten Agenten eingenommen werden. Für jede Rolle schreibt das Protokoll vor, welche Interaktionen abhängig vom bisherigen Protokollverlauf durchgeführt werden dürfen oder müssen, d.h. welche Rolle wann bestimmte Arten von Sprechakten an wen versendet sowie umgekehrt entsprechende Sprechakte empfängt.

Im Rahmen von Dienstnutzungen besitzt ein Protokoll genau zwei Rollen, den Anbieter und den Nutzer, die zur Dienstnutzung jeweils von genau einem Agenten eingenommen werden, während zur Dienstauswahl auch mehrere Agenten als (potentielle) Anbieter auftreten können.

Protokolloperatoren

Um eine Rolle innerhalb eines Protokolls auszufüllen, führt ein Agent einen Protokolloperator für diese Rolle aus. Ein Protokolloperator ist dabei definiert über das Protokoll, die darin eingenommene Rolle und den Dienst, zu dessen Durchführung das Protokoll dient. Protokolloperatoren sind i.a. Skripte, die die Struktur der jeweiligen Rolle innerhalb eines Protokolls widerspiegeln.

Sprechaktoperatoren

Die eigentlichen kommunikativen Handlungen im Rahmen von Protokollen bestehen im Austausch von Sprechakten. Diese werden repräsentiert durch Sprechaktoperatoren, die innerhalb von (zusammengesetzten) Protokolloperatoren vorkommen und das Versenden und den Empfang von Sprechakten ermöglichen.

Intentionen

Intentionen sind zur Ausführung ausgewählte Handlungen. Sie bestehen aus einem Operator oder einem Plan aus mehreren Operatoren sowie gegebenenfalls aus der zugehörigen Instantiierung zu einer konkreten Handlung.

Für jede Intention gilt, daß zum Zeitpunkt ihres Aufstellens die Möglichkeit zur Ausführung und damit zum Erreichen des Effekts als gegeben angesehen wird, d.h. die Bedingungen des Operators sind erfüllt. Die Gesamtheit der Intentionen ist konsistent, wenn keine Bedingung durch den Effekt einer zuvor auszuführenden Intention zunichte gemacht wird. Eine explizite Ausführungsreihenfolge für Intentionen wird auf dieser Ebene jedoch nicht festgelegt.

4.1.3. Kontrollfunktionen

Aufbauend auf diesen Wissenstypen steuern die folgenden Kontrollfunktionen das Verhalten des Agenten:

- *Aktualisieren* von Fakten und Zielen
- *Reagieren* auf Änderungen des Faktenwissens
- *Entscheiden*: Auswählen und Planen von Handlungen für Ziele
- *Agieren*: Koordinieren und Ausführen von Handlungen
- *Interagieren*: Dienstnutzung und Kommunikation mit anderen Agenten

Das reaktive Verhalten besteht in den Reaktionen, die durch das Aktualisieren des Faktenwissens ausgelöst werden. Die aktuellen Ziele steuern das deliberative Verhalten, das durch Entscheidungen zu bestimmten Handlungen und deren Ausführung umgesetzt wird. Das interaktive Verhalten resultiert aus denjenigen Teilen des reaktiven und deliberativen Verhaltens, die in Form von Dienstnutzungen zu Interaktionen mit anderen Agenten führen.

Fakten und Ziele aktualisieren

Um den dynamischen Veränderungen der Welt gerecht zu werden, muß ein Agent sein Wissen aktuell halten. Einerseits verlangt dies eine permanente Aktualisierung des Faktenwissens, damit dieses möglichst korrekt und so vollständig wie möglich und nötig den tatsächlichen Zustand der Welt widerspiegelt. Dazu werden neu bekannt gewordene Fakten in das Faktenwissen aufgenommen und gegebenenfalls dadurch auftretende Inkonsistenzen beseitigt.

Andererseits entstehen neue Ziele und bestehende werden erreicht oder hinfällig. Ein Zielzustand kann sowohl durch das erfolgreiche Ausführen einer dafür aufgestellten Intention, als auch durch zufällige äußere Umstände eintreten. Da ein Ziel lediglich das Erreichen des Zielzustands beinhaltet, wird es auch nur so lange beabsichtigt, bis dieser erstmals eintritt. Wird ein Ziel anders als durch eine dafür ausgeführte Handlung erreicht, so müssen gegebenenfalls durch das Ziel initiierte Aktivitäten wieder beendet werden, da diese nicht mehr benötigt werden.

Woher ein Agent neue Fakten und Ziele erhält, wird an dieser Stelle nicht festgelegt. Eine Quelle sind zumindest die Aktionsteile der Reaktionsregeln, zusätzlich kann ein Agent Möglichkeiten zur Wahrnehmung seiner Umgebung und zur Ausrichtung seines Verhaltens auf bestimmte Ziele hin besitzen.

Reagieren

Das Faktenwissen des Agenten wird beständig daraufhin überwacht, ob die durch die Bedingungen der Reaktionsregeln beschriebenen Situationen eintreten. Jedesmal, wenn sich die Erfülltheit der Aussage einer Bedingung von nicht erfüllt zu erfüllt ändert, wird der zugehörige Aktionsteil einmal ausgeführt und entsprechend auf die

eingetretene Situation reagiert, indem Fakten, Ziele oder Intentionen abgeändert werden.

Entscheiden

Um seine Ziele aktiv zu verfolgen, stellt ein Agent Intentionen auf, die bei erfolgreicher Umsetzung das zugehörige Ziel erfüllen. Dazu muß er entscheiden, welche Ziele als nächstes verfolgt werden sollen und auf welche Weise sie erreicht werden sollen.

Zum Aufstellen einer Intention werden die möglichen Handlungsalternativen – Operatoren, deren Effekt ausgehend vom momentanen Faktenwissen den Zielzustand erfüllt – zum Erreichen eines Ziels betrachtet. Gegebenenfalls können auch neue Pläne für ein Ziel erstellt werden. Aus den Alternativen wird die gemäß einem Bewertungskriterium beste ausführbare Handlung – mit durch das Faktenwissen erfüllter Vor- und Rahmenbedingung – ausgewählt und als neue Intention aufgestellt. Dabei können auch andere Ziele und bestehende Intentionen berücksichtigt werden, um Redundanzen auszunutzen und Konflikte zu vermeiden.

Wird keine ausführbare Handlung für ein Ziel gefunden, so muß entschieden werden, ob das Ziel aufgegeben werden soll oder nur zurückgestellt, bis es aufgrund geänderter Umstände doch noch verfolgt werden kann. Die geänderten Umstände können das Eintreten eines Weltzustands sein, den der Agent nicht selbst herbeiführen konnte, der aber die Anwendung eines vorhandenen Operators ermöglicht, oder eine Erweiterung der verfügbaren Operatoren, insbesondere auch in Form von Diensten anderer Agenten, die neue Dienste anbieten oder gänzlich neu zu einer Agentengesellschaft hinzugefügt werden.

Agieren

Zur Umsetzung der so aufgestellten Intentionen in Handlungen werden diese zunächst untereinander anhand ihrer formalen Beschreibung koordiniert, um deren unbeeinträchtigte Ausführbarkeit zu gewährleisten. Es folgt die Auswahl der als nächstes auszuführenden Intentionen sowie schließlich deren Ausführung.

Die Ausführung eines Operators hängt von dessen Typ ab (vgl. Kapitel 4.5.9). Für die meisten Arten von Operatoren besteht diese in einem einzelnen Schritt, Pläne und Skripte werden interpretiert und gemäß ihrer Struktur in Einzeloperatoren zerlegt. Mit dem Ende der Ausführung ist auch die Intention beendet. Deren Ergebnis bestimmt Erfolg und Resultat der Intention und damit das Erreichen des Ziels, für das die Intention aufgestellt wurde.

Interagieren

Die Ausführung eines Dienstoperators geschieht durch eine Dienstnutzung bei einem Anbieter dieses Dienstes. Die zur Dienstnutzung notwendige Kommunikation

wird über ein Protokoll gesteuert, wozu beide Interaktionspartner jeweils einen passenden Protokolloperator ausführen.

Bei der Ausführung eines Sprechaktoperators wird ein Sprechakt erzeugt und an den Kommunikationspartner geschickt bzw. es wird auf das Eintreffen eines entsprechenden Sprechakts gewartet.

Die Details der Durchführung von Interaktionen werden in Kapitel 5.1 beschrieben.

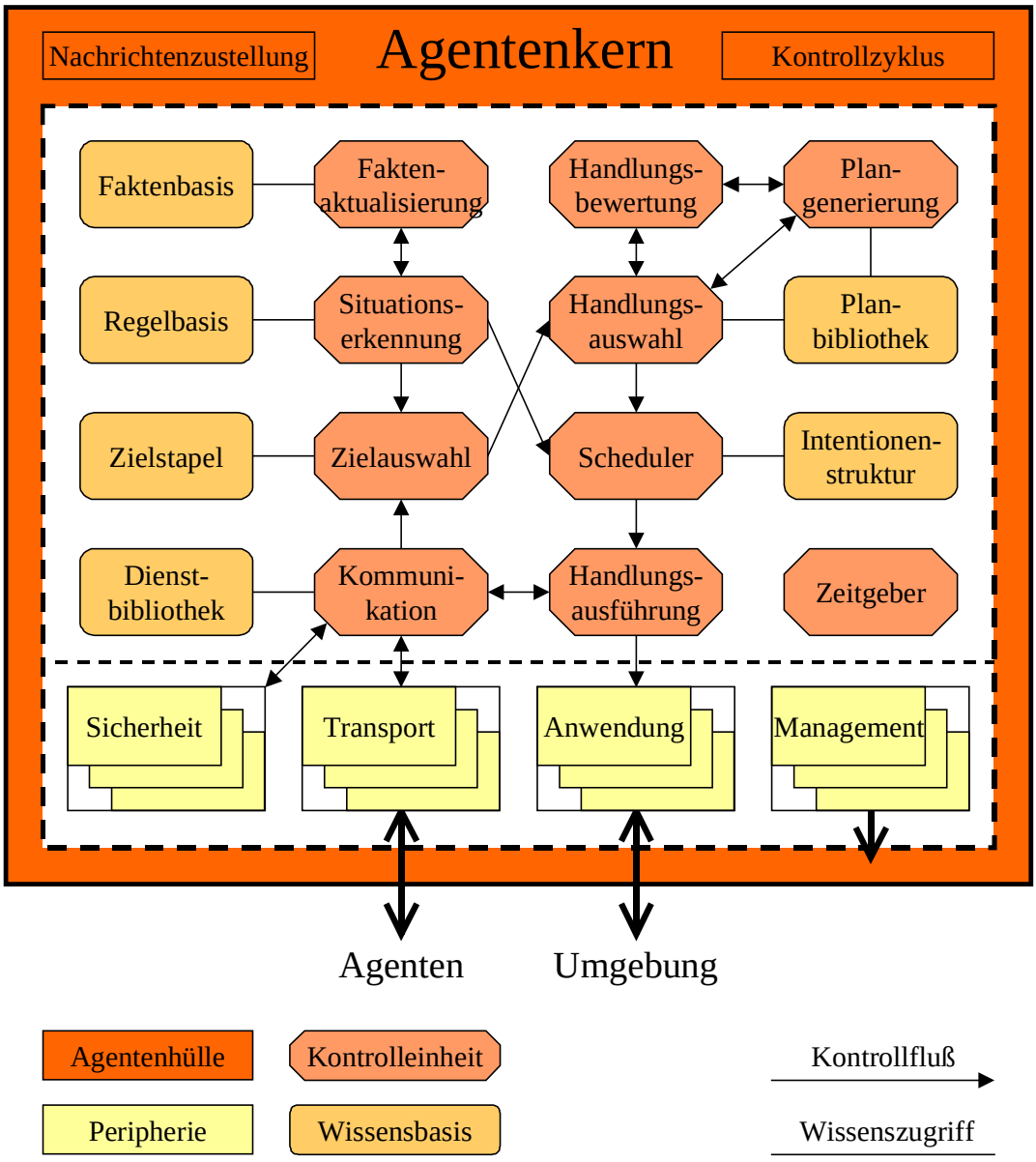

Abbildung 17: Komponentenrollen der CASA-Standardarchitektur

4.2. Die Standardarchitektur

Aus der Wissensbasiertheit folgt für die Implementierung einer Agentenarchitektur eine klare Trennung zwischen einem formalen, beschreibenden und einem ausführenden, interpretierenden Anteil eines Agenten, seinem Wissen und der auf diesem operierenden Kontrolle. Der beschreibende Teil besteht aus einer Sprache zur Formulierung von Wissensinhalten und Aussagen sowie aus Datenstrukturen zur Repräsentation der verschiedenen Wissenstypen. Ein Formalismus für eine derartige Sprache bildet den Inhalt von Kapitel 6. Der operationale Teil besteht in Komponenten zur Speicherung, Interpretation und Verarbeitung des Wissens sowie zur Durchführung von Handlungen aufgrund dieses Wissens gemäß den dargestellten Kontrollfunktionen. Hierfür definiert die Standardarchitektur für CASA-Agenten eine Menge von Komponentenrollen sowie Nachrichtentypen zur Interaktionen zwischen diesen.

Die CASA-Standardarchitektur (Abbildung 17) modelliert das Funktionsprinzip eines wissensbasierten Agenten durch entsprechende Komponenten zur Speicherung, Verarbeitung und Nutzung von Wissen. Aufgrund der Austauschbarkeit von Komponenten werden diese über Rollen charakterisiert, die ihre Funktion innerhalb eines Agenten beschreiben. Die Standardarchitektur besteht aus folgenden Arten von Komponentenrollen:

- *Agentenhülle*: Die Agentenhülle bildet unabhängig von der Standardarchitektur das Gerüst eines jeden CASA-Agenten, das den Aufbau aus Komponenten ermöglicht. Sie besteht aus den im vorhergehenden Kapitel vorgestellten Infrastrukturkomponenten und dem Agentenkern.
- *Kern*: Der Kern der Standardarchitektur realisiert die wissensbasierte Verhaltenssteuerung und ist unterteilt in eine Wissensbasis und eine Kontrolleinheit:
  - *Wissensbasis*: Die Komponenten der Wissensbasis dienen der Speicherung des Wissens eines Agenten. Die Wissensbasis ist unterteilt nach den verschiedenen Wissenstypen, für die jeweils eine eigene Komponentenrolle vorgesehen ist.
  - *Kontrolleinheit*: Die Umsetzung der Kontrollfunktionen zur Verhaltenssteuerung geschieht durch die Komponenten der Kontrolleinheit. Auch hier entspricht einer Funktionalität jeweils eine Rolle.
- *Peripherie*: In der Peripherie sind diejenigen Arten von Komponenten angesiedelt, die sich nicht direkt aus dem wissensbasierten Schema ergeben. Im einzelnen sind dies:
  - *Anwendungskomponenten*: Diese Komponenten realisieren anwendungsspezifische Aspekte eines Agenten. Dazu gehören insbesondere die Interaktion mit der Umgebung durch Sensorik und Aktorik, wodurch neue Fakten entstehen und Handlungen ausgeführt werden können, sowie die Kontrolle des zielgerichteten Verhaltens über neue Ziele.

- *Transportkomponenten*: Den physischen Aspekt der Kommunikation übernehmen die Transportkomponenten. Sie verschicken Sprechakte an andere Agenten und ermöglichen den Empfang von Sprechakten.
- *Sicherheitskomponenten*: Zur Absicherung der Kommunikation können Sicherheitskomponenten eingesetzt werden, die Aufgaben wie die Authentifikation und Autorisierung von Agenten und die Verschlüsselung der Sprechakte übernehmen.
- *Managementkomponenten*: Diese Komponenten besitzen einen direkten Zugriff auf den Agentenkern und können so den Agenten als ganzes kontrollieren. Auf diese Weise lassen sich Managementfunktionalitäten realisieren.

Somit ergibt sich eine konzeptionelle Dreiteilung der Standardarchitektur in die obligatorische Agentenhülle, den wissensbasierten Kern und die Peripherie. Abbildung 18 bietet eine kurze Übersicht über die Aufgaben der Rollen des Kerns im Rahmen der Verhaltenssteuerung innerhalb eines Agenten. Wie auch aus Abbildung 17 ersichtlich, ist jeder Rolle der Wissensbasis jeweils eine Rolle der Kontrolleinheit zugeordnet, die deren Inhalt verwaltet.

Die Standardarchitektur ist jedoch keineswegs verbindlich für den Aufbau eines CASA-Agenten, lediglich die Agentenhülle und die Konformität der übrigen verwendeten Komponenten zur Basisschnittstelle für die interne Interaktion konstituieren den Agenten. Der Sinn der Standardarchitektur ist es, ein Grundgerüst bereitzustellen und so einen Rahmen für eine Menge an vorgegebenen Komponenten zu liefern, aus denen ein Agent nach Bedarf zusammengesetzt werden kann, so daß jeweils nur noch für die anwendungsspezifischen Aspekte neue Komponenten implementiert zu werden brauchen. Dabei können auch nur Teile der Standardarchi-

Rolle	Name	Funktion
Faktenbasis	FactBase	Speicher für Fakten
Faktenaktualisierung	FactMaintenance	Aktualisierung von Faktenwissen
Zeitgeber	Timer	Zeitreferenz des Agenten
Regelbasis	RuleBase	Speicher für Regeln
Situationserkennung	SituationAssessment	Überwachung und Ausführung von Regeln
Zielstapel	GoalStack	Speicher für Ziele
Zielauswahl	GoalSelection	Aktualisierung und Auswahl von Zielen
Planbibliothek	PlanLibrary	Speicher für Operatoren
Handlungsauswahl	ActSelection	Auswahl von Handlungen für Ziele
Plangenerierung	PlanGeneration	Erzeugen von Plänen
Handlungsbewertung	ActEvaluation	Bewertung von Handlungsalternativen
Intentionenstruktur	IntentionStructure	Speicher für Intentionen
Scheduler	Scheduler	Koordinierung von Intentionen
Handlungsausführung	ActExecution	Ausführung von Intentionen
Dienstbibliothek	ServiceLibrary	Speicher für angebotene Dienste
Kommunikation	Communication	Durchführen von Interaktionen

Abbildung 18: Rollen des Kerns der Standardarchitektur

tektur verwendet werden, beispielsweise falls ein Agent keine reaktiven oder interaktiven Fähigkeiten benötigt.

Da die Standardarchitektur nur Rollen vorgibt, lassen sich diese durch verschiedenartige Komponenten ausfüllen, die bestimmten besonderen Anforderungen wie Effizienz oder Leistungsfähigkeit gerecht werden. So bleibt eine hinreichende Offenheit und Skalierbarkeit der Architektur gewährleistet. Entsprechend werden für die einzelnen Rollen im folgenden lediglich Mindestanforderungen spezifiziert sowie weitergehende Möglichkeiten zu deren Umsetzung aufgezeigt. Die Spezifikation umfaßt neben den in Definition 1 (Seite 80) geforderten Angaben – Name, Rollengruppen, Sendeberechtigungen und Eigenschaften – auch den internen Zustand einer Komponente und die zur Verarbeitung der empfangbaren Nachrichtentypen vorgesehenen Änderungen dieses Zustands. Sämtliche nachfolgend im Zusammenhang mit den einzelnen Komponentenrollen vorgenommenen Definitionen stellen jeweils nur Mindestanforderungen dar, die bei Bedarf zu erweitern sind.

Zuvor werden noch einige generische Nachrichtentypen zur Umsetzung der in Kapitel 3.5.2 vorgestellten Interaktionsschemata zwischen Komponenten sowie die Rollengruppen für die Standardarchitektur eingeführt.

4.2.1. Generische Nachrichtentypen

Bei allen drei für die Interaktion zwischen CASA-Komponenten eingeführten Nachrichtenschemata besitzt die initiale Nachricht immer einen spezifischen Nachrichtentyp, da über diese die rollenspezifischen Sendeberechtigungen festgelegt werden. Da die Benachrichtigung nur aus der initialen Nachricht besteht und die Registrierung einen Sonderfall des Auftrags darstellt, werden lediglich zur Umsetzung von Beauftragungen generische Nachrichtentypen benötigt.

Das Nachrichtenschema für einen Auftrag (Abbildung 12, Seite 102) sieht zum einen die Annahme oder Ablehnung, zum anderen ein Ergebnis zum Abschluß des Auftrags vor. Weiterhin erlaubt es die Rücknahme eines Auftrags (Abbildung 13, Seite 103), die ihrerseits angenommen und abgelehnt werden kann. Daraus ergeben sich die folgenden generischen Nachrichtentypen:

Name	Accept
Typ	Bestätigung
Beschreibung	Bestätigung eines Auftrag bzw. der Rücknahme eines Auftrags
Inhalt	ein boolean-Wert, der Annahme (true) bzw. Ablehnung (false) angibt
Referenz	die betreffende Nachricht für den Auftrag bzw. die Rücknahme

Name	Result
Typ	Ergebnis
Beschreibung	Mitteilung zum Abschluß eines angenommenen Auftrags
Inhalt	ein boolean-Wert, der den Erfolg der Ausführung angibt (Falls das Ergebnis zusätzliche Parameter enthalten kann, so müssen eigene Ergebnisnachrichtentypen mit einem entsprechend erweiterten Inhalt definiert und verwenden werden.)
Referenz	die beauftragende Nachricht

Name	Retract
Typ	Rücknahme
Beschreibung	Rücknahme eines Auftrags
Referenz	die Nachricht mit dem zurückzunehmenden Auftrag

Die Nachrichtentypen Accept und Retract sind für das Nachrichtenschema zur Beauftragung fest vorgeschrieben, Result wegen der möglichen zusätzlichen Inhalte nicht. Accept und Result sind Antwortnachrichten, d.h. Absender und Empfänger sind genau umgekehrt wie bei der beantworteten Nachricht, während diese bei Retract mit der zurückgenommenen Nachricht übereinstimmen.

Die Sendeberechtigungen für diese drei Nachrichtentypen werden in der obersten Rollengruppe jeweils für die oberste Rollengruppe erteilt, so daß jede Rolle diese Nachrichten an jede Rolle senden darf. Umgekehrt muß jede Rolle sie verarbeiten können, wobei durch die Verwendung der beauftragenden Nachricht als Referenz bei allen drei Nachrichtentypen gewährleistet ist, daß keine falsche Zuordnung möglich ist. Eine Komponente kann so leicht überprüfen, daß sich ein Accept oder Result auf einen selbst erteilten Auftrag bzw. ein Retract auf einen erhaltenen Auftrag bezieht.

Aufgrund der Pufferung der empfangenen Nachrichten in der Warteschlange der empfangenden Komponente kann die Nachrichtenzustellung Retract-Nachrichten gesondert behandeln, indem sie die zurückzunehmende Nachricht wieder aus der Warteschlange entfernt, falls sie noch in dieser enthalten ist. So wird bereits die Verarbeitung der zurückzunehmenden Nachricht verhindert, so daß auch die Retract-Nachricht nicht mehr zugestellt zu werden braucht. Dies ist nicht zuletzt auch deshalb ratsam, weil andernfalls eine Retract-Nachricht mit einer höheren Priorität als die zurückzunehmende Nachricht diese in der Warteschlange überholen kann, so daß der Empfänger sie nicht berücksichtigen kann, da die zugehörige Auftragsnachricht noch gar nicht verarbeitet wurde.

4.2.2. Rollengruppen der Standardarchitektur

Die Rollengruppen für die Standardarchitektur entsprechen den oben genannten Arten von Komponentenrollen:

- *Kernel*: Rollen der Agentenhülle
- *KnowledgeBase*: Rollen der Wissensbasis
- *ControlUnit*: Rollen der Kontrolleinheit
- *Application*: Rollen für anwendungsspezifische Komponenten
- *Transport*: Rollen zum Transport von Sprechakten zwischen Agenten
- *Security*: Rollen zur Absicherung der Kommunikation des Agenten
- *Management*: Rollen zur Verwaltung und Kontrolle des Agenten

Die Spezifikation der Rollengruppen erfolgt jeweils im Zusammenhang mit den zu der Gruppe gehörenden Rollen.

Die oberste Rollengruppe

```
[CASA.Main,
 {},
 {[Accept, {CASA.Main}],
  [Result, {CASA.Main}],
  [Retract, {CASA.Main}],
  [InformRole, {CASA.Agent}],
  [InformFact, {CASA.FactBase, CASA.SituationAssessment}],
  [InformRule, {CASA.RuleBase}],
  [InformGoal, {CASA.GoalStack}],
  [InformOperator, {CASA.PlanLibrary, CASA.ServiceLibrary}],
  [InformIntention, {CASA.IntentionStructure}]},
  [InformTime, {CASA.Timer}]},
 {[State, string, LifeCycleStates, ”undefined“, riw],
  [Busy, boolean, boolean, false, r],
  [QueueSize, int, {i ∈ int | i ≥ 0}, 0, r]}]
```

Der Name der obersten Rollengruppe ist CASA.Main und sie gehört selbst keiner Rollengruppe an, was sie gerade zur obersten Rollengruppe macht (vgl. Kapitel 3.2). Für die drei generischen Nachrichtentypen Accept, Result und Retract erteilt sie Sendeberechtigungen für alle Komponentenrollen, indem die oberste Rollengruppe selbst als Berechtigter angegeben wird. Weiterhin werden für einige spezifische Ergebnisnachrichtentypen von nachfolgend beschriebenen Rollen der Standardarchitektur Sendeberechtigungen an die jeweiligen Rollen vergeben. An Eigenschaften sind generische Komponenteneigenschaften definiert (vgl. Kapitel 3.3.4). Dies sind der Lebenszykluszustand, der Beschäftigungszustand und die Anzahl der in der Nachrichtenschlange wartenden Nachrichten.

4.3. Agentenhülle

Die Agentenhülle besteht aus dem Agentenkern und den Infrastrukturkomponenten Nachrichtenzustellung und Kontrollzyklus, die zusammen den Aufbau eines Agenten aus Komponenten ermöglichen. Nur der Agentenkern ist dabei selbst eine Komponente, die eine Rolle besitzt und damit Nachrichten versenden, empfangen und verarbeiten kann. Die Funktionalität dieser Komponenten wurde bereits in den Kapiteln 3.4, 3.5.2 und 3.6.2 beschrieben, da sie unabhängig von der Standardarchitektur sind.

Rollengruppe

```
[CASA.Kernel,
 {CASA.Main},
 {},
 {}]
```

Die Rollengruppe mit dem Namen CASA.Kernel gehört lediglich zur obersten Rollengruppe. Spezifische, allen Rollen dieser Gruppe gemeinsame Sendeberechtigungen und Eigenschaften gibt es nicht.

4.3.1. Agentenkern

Zwar besteht über die Basisschnittstelle die Möglichkeit zu direkten Interaktionen zwischen Agentenkern und sämtlichen Komponenten in beiden Richtungen, dennoch sind beim Agentenkern zusätzlich auch entkoppelte Interaktionen über Nachrichten sinnvoll. Dafür benötigt dieser eine eigene Rolle, die mit der Aufgabe der Verwaltung der Komponentenstruktur verknüpft ist. Diese Rolle ist die einzige, die der Rollengruppe für die Agentenhülle zugeordnet ist.

Interner Zustand

Formal ist der Agentenkern eine Menge $K = \{k_1, \ldots, k_n\}$ an Komponenten (vgl. Definition 3, Seite 81), die er verwaltet. Diese Menge und der Lebenszykluszustand des Agenten machen den in jedem Fall vorhandenen internen Zustand einer Komponente zur Umsetzung der Rolle des Agentenkerns aus. Die Zustandsänderungen im Rahmen der Funktionalität des Agentenkerns wurden bereits in Kapitel 3.4.4 beschrieben.

Nachrichtentypen

Eine über Nachrichten realisierte Funktion des Agentenkerns ist das Mitteilen der aktuellen Rollenstruktur des Agenten sowie ihrer Veränderungen. Hierfür werden Nachrichten verwendet, um neben einmaligen Anfragen auch eine Registrierung für

Veränderungen gemäß dem entsprechenden Nachrichtenschema zu ermöglichen. Die verwendeten Nachrichtentypen sind:

Name	FindRole
Typ	Auftrag
Beschreibung	Suche nach durch Komponenten belegte Rollen eines bestimmten Typs
Inhalt	eine Rolle oder Rollengruppe r
Ergebnis	eine InformRole-Nachricht mit der Menge R der gesuchten Rollen ▪ $R = \{r' \mid \exists k \in K: r' \in Rollen(k) \wedge r = r'\}$ für Rolle r ▪ $R = \{r' \mid \exists k \in K: r' \in Rollen(k) \wedge r \in Gruppen(r')\}$ für Rollengruppe r

Name	MonitorRole
Typ	Registrierung
Beschreibung	Anmeldung für Mitteilungen über Änderungen der Menge der durch Komponenten belegten Rollen eines bestimmten Typs
Inhalt	eine Rolle oder Rollengruppe r
Ergebnis	eine InformRole-Nachricht mit neu hinzugefügten bzw. entfernten Rollen R; bei einem Austausch von Komponenten werden nicht die ersetzten, sondern nur gegebenenfalls zusätzlich hinzugekommene und entfernte Rollen mitgeteilt Sei K die alte, K' die neue Menge der Komponenten mit $K \neq K'$; $R^+ = Rollen(K') \setminus Rollen(K)$ und $R^- = Rollen(K) \setminus Rollen(K')$ sind dann die Mengen der hinzugefügten bzw. entfernten Rollen. Gilt nun $R \neq \emptyset$ für $R' = R^+$ bzw. $R' = R^-$ mit ▪ $R = \{r' \in R' \mid r = r'\}$ für Rolle r ▪ $R = \{r' \in R' \mid r \in Gruppen(r')\}$ für Rollengruppe r so wird jeweils eine entsprechende InformRole-Nachricht als Antwort gesendet

Name	InformRole
Typ	Ergebnis
Beschreibung	Mitteilung, ob im Agenten eine Menge von Rollen durch Komponenten belegt ist
Inhalt	eine Menge von Rollen und ein boolean-Wert, der angibt, ob diese Rollen belegt sind oder nicht
Referenz	die beauftragende Nachricht

Mit Hilfe der Nachrichtentypen FindRole und MonitorRole kann eine Komponente sich so über das Vorhandensein möglicher bzw. benötigter Interaktionspartner innerhalb des Agenten informieren.

Rolle

```
[CASA.Agent,
 {CASA.Kernel},
 {[FindRole, {CASA.Main}],
  [MonitorRole, {CASA.Main}]},
 {[AgentState, string, LifeCycleStates, ”undefined“, riw],
  [AgentBusy, boolean, boolean, false, r],
  [Components, Component{}, Component{}, ∅, ri]}]
```

Der Name der Rolle ist CASA.Agent und sie gehört zur Rollengruppe CASA.Kernel. Sendeberechtigungen gibt es für FindRole und MonitorRole für alle Rollen. Die Eigenschaften sind der Lebenszykluszustand und der Beschäftigungszustand des Agenten sowie die Menge der im Agenten vorhandenen Komponenten.

4.3.2. Infrastrukturkomponenten

Die beiden Infrastrukturkomponenten Nachrichtenzustellung und Kontrollzyklus bilden die agenteninterne Infrastruktur zur Interaktion von Komponenten über Nachrichten. Dazu besitzen sie eine direkte Interaktionsmöglichkeit mit allen Komponenten über deren Basisschnittstelle, eine zusätzliche Interaktion über Nachrichten wird nicht benötigt. Entsprechend besitzen sie keine Rolle, weshalb sie im folgenden nur über einen Bezeichner und eine Menge von Eigenschaften spezifiziert werden.

Nachrichtenzustellung

```
[CASA.MessageServer,
 {[Addresses, Address{}, Address{}, ∅, r]}]
```

Die Nachrichtenzustellung – bezeichnet als CASA.MessageServer – verwaltet intern eine Menge von Adressen (vgl. Kapitel 3.5.2), die ihre einzige Eigenschaft darstellt, die zur Übergabe der Adressen beim Komponentenaustausch benötigt wird.

Kontrollzyklus

```
[CASA.ControlCycle,
 {[CycleComponents, Component{}, Component{}, ∅, r],
  [ThreadComponents, Component{}, Component{}, ∅, r]}]
```

CASA.ControlCycle bezeichnet den Kontrollzyklus. Die Eigenschaften CycleComponents und ThreadComponents dienen beim Austausch dem Abruf derjeniger Komponenten, die in den Kontrollzyklus eingegliedert sind bzw. denen eigene Verarbeitungsressourcen zugeordnet sind (vgl. Kapitel 3.6.2).

4.4. Wissensbasis

Die Wissensbasis enthält das aktuelle Wissen eines Agenten. Sie ist unterteilt in eigene Komponentenrollen für die verschiedenen Wissenstypen, einerseits damit für diese unterschiedliche Repräsentationsformalismen und Organisationsstrukturen verwendet werden können, andererseits weil sie jeweils von einer anderen Rolle der Kontrolleinheit verwaltet werden. Im einzelnen sind dies:

- *Faktenbasis*: eine Menge F von Fakten
- *Regelbasis*: eine Menge R von Reaktionsregeln
- *Zielstapel*: eine geordnete Liste G von Zielen
- *Planbibliothek*: eine Menge O von Operatoren (inklusive der nutzbaren Dienste)
- *Intentionenstruktur*: eine strukturierte Menge I von Intentionen
- *Dienstbibliothek*: eine Menge S von Dienst-Operatoren (angebotene Dienste)

Die Wissensbasis und damit die Gesamtheit des (explizit repräsentierten) Wissens eines Agenten ergibt sich damit als ein 6-Tupel KB = [F, R, G, O, I, S] der Inhalte der Wissensbasiskomponenten für die einzelnen Wissenstypen[28].

Unabhängig vom jeweiligen Wissenstyp besitzen alle Komponenten der Wissensbasis eine übereinstimmende Funktionalität zur grundlegenden Verwaltung der in ihnen gespeicherten Inhalte:

- *Zugriff* auf Wissen: der Abruf und die Suche nach bestimmten Inhalten der jeweiligen Wissensbasis
- *Verändern* von Wissen: Hinzufügen, Entfernen und Ändern von Inhalten der jeweiligen Wissensbasis
- *Überwachen* von Wissensänderungen: Mitteilen von bestimmten Änderungen der Inhalte der jeweiligen Wissensbasis an Komponenten, die sich für eine derartige Überwachung angemeldet haben

Die Nutzung dieser Funktionalitäten geschieht über Nachrichten vermittels der Schemata für Aufträge und Registrierungen. Entsprechend gibt es zu jedem Wissenstyp zugehörige Nachrichtentypen für einen Auftrag zum Zugriff auf die Wissensbasis, zum Hinzufügen und Entfernen von Inhalten, für eine Registrierung zur Überwachung von Änderungen sowie für Ergebnisnachrichten mit dem jeweiligen Wissenstyp als Inhalt.

---

[28] Für Ontologien und Protokolle sind keine Wissensbasiskomponenten vorgesehen, da bei diesen eine explizite Repräsentation nur kaum nutzbringend verwendet werden kann. Beide legen a priori Formen zur Repräsentation von Wissen bzw. zur Durchführung von Konversationen fest, an die der Agent sich unbedingt halten muß und die zwischen verschiedenen Agenten übereinstimmen müssen, um Interaktionen zu ermöglichen. Implizit sind sie im Wissen bzw. in den Operatoren für Interaktionen enthalten. Eine explizite Verwendung oder gar Veränderung übersteigt jedoch in der Regel die Fähigkeiten von Agenten.

Die direkte Veränderung des Inhalts einer Wissensbasiskomponente kann auf die jeweils zugeordnete Kontrollkomponente beschränkt sein, indem nur diese die entsprechenden Sendeberechtigungen erhält. Andere Komponenten müssen Wissensänderungen dann über diese Kontrollkomponente vornehmen, damit diese das betreffende Wissen konsistent halten kann. Dies gilt vor allem für diejenigen Wissenstypen, für die sich das repräsentierte Wissen zur Laufzeit des Agenten stark ändern kann – namentlich Fakten, Ziele und Intentionen.

Rollengruppe

```
[CASA.KnowledgeBase,
 {CASA.Main},
 {},
 {}]
```

Die Rollengruppe für die Komponentenrollen der Wissensbasis heißt CASA.KnowledgeBase und besitzt weder Sendeberechtigungen, noch Eigenschaften, da diese bei den einzelnen Rollen trotz der einheitlichen Funktionalität jeweils andere Wissenstypen betreffen.

4.4.1. Faktenbasis

Die Faktenbasis beinhaltet das Wissen eines Agenten über den gegenwärtigen Zustand der Welt in Form einer Menge von Fakten. Dieses dient als Grundlage für das reaktive und deliberative Verhalten zur Veränderung der Welt beim Überprüfen der Erfülltheit von Regeln und bei der Auswahl von Handlungen.

Das Wissen eines Agenten über die Welt besteht nicht alleine in den explizit in der Faktenbasis repräsentierten Fakten, sondern darüber hinaus auch in sämtlichen Aussagen, die er daraus ableiten kann. Dazu muß die Menge der vorhandenen Fakten konsistent sein und ein korrektes Beweisverfahren verwendet werden. Der Beweiskalkül braucht jedoch nicht unbedingt vollständig zu sein, sondern kann auf einen hinreichend effizient entscheidbaren Teilbereich der Logik beschränkt sein[29]. Somit kann es prinzipiell Aussagen geben, die zwar gemäß der zugrundeliegenden formalen Logik relativ zum Faktenwissen wahr sind, für deren Beweis der Agent aber nicht die nötigen Fähigkeiten besitzt.

Neben dem für alle Komponenten der Wissensbasis vorgesehenen Zugriff auf einzelne Inhalte unterstützt die Faktenbasis somit auch die Evaluierung komplexer Aussagen über den aktuellen Weltzustand. Die Wahrheit dieser durch logische

[29] Bereits die Prädikatenlogik erster Stufe ist nur semi-entscheidbar, d.h. es läßt sich nicht immer beweisen, daß für eine Aussage kein Beweis existiert. Beschränkt man sich wie im Fall der Faktenbasis auf eine endliche Menge an Grundaussagen als Axiomenmenge, so ist zwar die Entscheidbarkeit gegeben, aber selbst dann kann die Komplexität ein effizientes Beweisverfahren verhindern. Andere Logiken sind nicht entscheidbar, d.h. es gibt für sie keine vollständigen Beweisverfahren.

Formeln ausgedrückten Aussagen stellt der verwendete Beweiskalkül relativ zur Faktenmenge fest. Enthalten diese Formeln Variablen, so können die Formeln als Fragen aufgefaßt werden, auf die eine Antwort in Form einer Variablenbelegung gefunden werden soll. Da die Variablen dabei implizit existenzquantifiziert sind, können sich auch mehrere Ergebnisse für dieselbe Anfrage ergeben.

Inhalt

Die Faktenbasis enthält die Menge $F = \{f_1, \ldots, f_n\}$ der repräsentierten Fakten. Ein Fakt ist dabei eine positive oder negative Grundaussage bezüglich der Eigenschaft eines Objekts. Prinzipiell können auch beliebige andere Formeln zur Formulierung von Faktenwissen verwendet werden, was jedoch zu wesentlich mehr Aufwand für das Beweisverfahren sowie bei der Verwaltung der Fakten führt, da Inkonsistenzen nicht mehr trivial zu erkennen sind.

Definition 5: Fakten

> Ein Fakt f ist ein 4-Tupel [a, o, v, t]. a bezeichnet ein Attribut und o ein Objekt. v ist der Wert, der dem Objekt für das Attribut zukommt. t ist der Wahrheitswert des Fakts (wahr ($\top$) oder falsch ($\bot$)).

Die Menge der für ein Objekt möglichen Attribute und deren Typ ergeben sich anhand der Kategorie, der das Objekt zugeordnet ist. Kategorien und ihre Attribute werden in Ontologien deklariert. Die Kategoriezugehörigkeit selbst kann als ein unveränderliches Attribut des Objekts aufgefaßt werden, das immer bekannt sein muß.

$F^- = \{f \mid \exists f' \in F: f' \rightarrow \neg f\} = \{[a,o,v,\neg t] \mid [a,o,v,t] \in F\} \cup \{[a,o,v',\top] \mid [a,o,v,\top] \in F \wedge v' \neq v\}$ ist die Menge der direkten Widersprüche zu F. Sämtliche möglichen, nicht in F oder F^- enthaltenen Fakten gelten als unbekannt.

Die Menge F muß konsistent sein bezüglich der in ihr enthaltenen Fakten, d.h. für keine Eigenschaft eines Objekts existieren sich widersprechende Annahmen. Dies ist der Fall, wenn $F \cap F^- = \varnothing$ gilt. Dies schließt eine Inkonsistenz bezüglich zusätzlichem Schlußfolgerungswissen jedoch nicht aus.

Nachrichtentypen

Für die Faktenbasis sind die folgenden Nachrichtentypen definiert:

Name	AddFact
Typ	Auftrag
Beschreibung	fügt ein Fakt zur Faktenbasis hinzu und entfernt widersprechende Fakten
Inhalt	ein Fakt f
Ergebnis	Result-Nachricht, immer true
Änderung	$F' = F \cup \{f\} \setminus \{f' \in F \mid f \rightarrow \neg f'\}$

Name	RemoveFact
Typ	Auftrag
Beschreibung	entfernt ein Fakt aus der Faktenbasis
Inhalt	ein Fakt f
Ergebnis	Result-Nachricht, true falls $f \in F$, false sonst
Änderung	$F' = F \setminus \{f\}$

Name	FindFact
Typ	Auftrag
Beschreibung	stellt die Erfülltheit einer Aussage relativ zur Faktenbasis fest
Inhalt	eine Aussage $a(x_i)$, ggf. mit freien Variablen x_i
Ergebnis	InformFact-Nachricht mit allen Aussagen $a_j = a(x_i)[x_i / v_{j,i}]$, für die $a(x_i)$ relativ zur Faktenbasis mit der Variablenbelegung $[x_i / v_{j,i}]$ erfüllt ist[30]

Name	MonitorFact
Typ	Registrierung
Beschreibung	Anmeldung zur Überwachung der Faktenbasis auf Änderungen; optional kann dabei ein zu überwachendes Muster angegeben werden
Inhalt	ein Fakt $f(x_i)$, ggf. mit freien Variablen x_i
Ergebnis	InformFact-Nachricht für jede (dem Muster $f(x_i)$ entsprechende) Änderung

Name	InformFact
Typ	Antwort-Benachrichtigung
Beschreibung	informiert über das Bestehen von Fakten
Inhalt	eine Menge von Aussagen sowie deren Wahrheitswert

Über AddFact und RemoveFact läßt sich der Inhalt der Faktenbasis verändern. FindFact und MonitorFact dienen dem Zugriff bzw. der Überwachung der Faktenbasis und teilen ihre Ergebnisse vermittels InformFact mit. Die Überwachung ist dabei beschränkt auf einzelne Faktenmuster, für komplexe Aussagen ist die Situationserkennung zuständig.

Rolle

```
[CASA.FactBase,
 {CASA.KnowledgeBase},
 {[AddFact, {CASA.FactMaintenance}],
  [RemoveFact, {CASA.FactMaintenance}],
  [FindFact, {CASA.Main}],
  [MonitorFact, {CASA.Main}]},
 {[Facts, Fact{}, Fact{}, ∅, ri]}]
```

[30] Eine Variablenbelegung $[x_i / v_i]$ ordnet jeder Variablen x_i aus einer Menge $\{x_1, \ldots, x_n\}$ jeweils genau einen Wert v_i zu. Bei einer Variablenersetzung $a[x_i / v_i]$ wird in dem Term oder der Formel a jedes Vorkommen einer Variablen x_i durch den in der Variablenbelegung $[x_i / v_i]$ zugeordneten Wert v_i ersetzt.

Die Rolle für die Faktenbasis heißt CASA.FactBase und gehört zur Rollengruppe Wissensbasis. Das Verändern des Inhalts der Faktenbasis über die Nachrichtentypen AddFact und RemoveFact ist der Rolle der Faktenaktualisierung vorbehalten, der Zugriff über FindFact und MonitorFact ist allen Rollen erlaubt. Die Eigenschaft Facts erlaubt den Zugriff auf die Gesamtheit aller Fakten und das initiale Setzen des Inhalts der Faktenbasis.

4.4.2. Regelbasis

Die Regelbasis enthält das reaktive Wissen eines Agenten. Sie besteht aus einer Menge an Regeln, die Situationen beschreiben, auf deren Eintreten eine Reaktion notwendig ist. Verwendet werden diese Regeln von der Situationserkennung, die deren Erfülltheit relativ zum Faktenwissen überwacht und gegebenenfalls den Aktionsteil der Regeln ausführt.

Inhalt

Die Regelbasis ist eine Menge $R = \{r_1, \ldots, r_n\}$ an Reaktionsregeln.

Definition 6: Reaktionsregeln

Eine Regel r ist ein Tupel [pre, $exec^{act}$] oder ein Tripel [pre, $exec^{act}$, $exec^{deact}$]. pre ist eine Formel und beschreibt die Aktivierungsbedingung. $exec^{act}$ ist die bei der Aktivierung, $exec^{deact}$ die bei der Deaktivierung auszuführende Aktion.

Der Ausführungsteil einer Regel beinhaltet als mögliche Aktionen das Hinzufügen und Entfernen von Fakten, Zielen und Intentionen. Alternativ ist auch über eine InformFact-Nachricht eine Benachrichtigung an diejenige Komponentenrolle möglich, die die Regel aufgestellt hat, so daß diese jeweils individuell auf die eingetretene Situation reagieren kann.

Nachrichtentypen

Die Regelbasis verwendet die folgenden Nachrichtentypen:

Name	AddRule
Typ	Auftrag
Beschreibung	fügt eine Regel zur Regelbasis hinzu
Inhalt	eine Regel r
Ergebnis	Result-Nachricht, true falls $r \notin R$, false sonst
Änderung	$R' = R \cup \{r\}$

Name	RemoveRule
Typ	Auftrag
Beschreibung	entfernt eine Regel aus der Regelbasis
Inhalt	eine Regel r
Ergebnis	Result-Nachricht, true falls $r \in R$, false sonst
Änderung	$R' = R \setminus \{r\}$

Name	FindRule
Typ	Auftrag
Beschreibung	sucht eine Regel anhand der Bedingung
Inhalt	eine Formel $f(x_i)$, ggf. mit freien Variablen x_i
Ergebnis	InformRule-Nachricht mit allen Regeln, deren Bedingung die gleiche oder eine spezifischere Situation ausdrückt als f; dies sind die Regeln, die bei Erfülltheit von f mit einer passenden Variablenbelegung ebenfalls erfüllt sind, d.h. es gibt eine Variablenbelegung $[x_i / v_i]$, so daß $f(x_i)[x_i / v_i] \rightarrow$ pre

Name	MonitorRule
Typ	Registrierung
Beschreibung	Anmeldung zur Überwachung der Regelbasis auf Änderungen; optional kann dabei ein zu überwachendes Muster angegeben werden
Inhalt	eine Formel $f(x_i)$, ggf. mit freien Variablen x_i
Ergebnis	InformRule-Nachricht für jede (gegebenenfalls analog zu FindRule dem Muster $f(x_i)$ entsprechende) Änderung

Name	InformRule
Typ	Antwort-Benachrichtigung
Beschreibung	informiert über vorhandene Regeln
Inhalt	eine Menge von Regeln sowie deren Vorhandensein

Die Menge der Reaktionsregeln des Agenten läßt sich über Nachrichten der Typen AddRule und RemoveRule ändern. Mit FindRule werden Regeln gesucht und mit MonitorRule Veränderungen der Regelmenge überwacht. Zur Mitteilung von aufgefundenen und geänderten Regeln wird InformRule verwendet.

Rolle

```
[CASA.RuleBase,
 {CASA.KnowledgeBase},
 {[AddRule, {CASA.Main}],
  [RemoveRule, {CASA.Main}],
  [FindRule, {CASA.Main}],
  [MonitorRule, {CASA.Main}]},
 {[Rules, Rule{}, Rule{}, ∅, ri]}]
```

CASA.RuleBase bezeichnet die Komponentenrolle für die Regelbasis, die zur Rollengruppe der Wissensbasis gehört. Die Nachrichten AddRule, RemoveRule, FindRule und MonitorRule dürfen von allen Rollen an die Regelbasis gesendet

werden. Als Eigenschaft gibt es Rules, über die die Menge der vorhandenen Regeln abgerufen und initial gesetzt werden kann.

4.4.3. Zielstapel

Der Zielstapel ist eine geordnete Liste der aktuellen Ziele, die die Reihenfolge des angestrebten Erreichens widerspiegelt. Ein Ziel ist dabei eine Aussage über einen intendierten Weltzustand, die wie eine Aussage über den aktuellen Weltzustand durch eine logische Formel ausgedrückt wird. Auch hier werden freie Variablen existenzquantifiziert interpretiert, d.h. der Zielzustand muß für eine beliebige Variablenbelegung erreicht werden. Die Reihenfolge der Ziele im Zielstapel ergibt sich aus der zeitlichen Reihenfolge des Hinzufügens und der Priorität der Ziele.

Für die Koordinierung der Ziele werden Redundanzen und Konflikte zwischen diesen festgestellt. Ein Konflikt besteht, wenn zwei Ziele miteinander unvereinbare Zielzustände beschreiben und somit nicht gleichzeitig erreichbar sind. Ziele sind redundant, wenn das Erreichen eines Ziels zugleich ein anderes vollständig oder teilweise erfüllt, so daß der übereinstimmende Teil auch nur einmal herbeigeführt zu werden braucht.

Inhalt

Der Zielstapel ist eine Liste $G = [g_1, \ldots, g_n]$ an Zielen. Die Ziele in dieser Liste sind sortiert anhand ihrer Priorität und der Reihenfolge des Hinzufügens:

$\forall g_i, g_j \in G: i < j \rightarrow Pri(g_i) < Pri(g_j) \vee (Pri(g_i) = Pri(g_j) \wedge Zeit(g_i) > Zeit(g_j))$
($Zeit(g_i) > Zeit(g_j)$ heißt, daß g_i später als g_j zum Zielstapel hinzugefügt wurde)

Definition 7: Ziele

Ein Ziel g ist ein 5-Tupel [goal, pri, C, R, state]. Bei einem Zustandsziel ist goal eine Formel, die den zu erreichenden Zielzustand beschreibt, und bei einem Interaktionsziel ein Tripel [service, prot, role] bestehend aus einem Dienst, einem Protokoll und einer Rolle. pri ist eine numerische Priorität, die angibt, wie wichtig das Ziel ist. Die übrigen Angaben werden zur Verarbeitung des Ziels benötigt. Die Abhängigkeiten zu anderen Zielen im Zielstapel werden über die Konfliktmenge C und die Redundanzmenge R ausgedrückt. state gibt den aktuellen Bearbeitungsstatus des Ziels an.

Neben diesen Angaben zu einem Ziel sind auch noch zusätzliche Parameter möglich, die die Bewertung oder Verarbeitung des Ziels betreffen. Beispielsweise kann die Rolle des Urhebers des Ziels dessen Priorität beeinflussen. Auch können für ein Ziel Situationen angegeben werden, unter denen es auch unerreicht aufgegeben wird, oder solche, in denen es trotzt erstmaligen Erreichens weiter aufrecht erhalten wird, so daß es erneut aktiv verfolgt wird, falls der Zielzustand nicht mehr gegeben ist.

Der Status eines Ziels bestimmt dessen weitere Bearbeitung. Unterschieden werden (mindestens) die folgenden Zustände:

- *unreached*: Der Zielzustand ist nicht erfüllt.
- *intended*: Das Ziel wird aktiv verfolgt.
- *reached*: Das Ziel wurde erreicht.
- *failed*: Das Ziel wurde nicht erreicht.
- *unreachable*: Das Ziel ist nicht erreichbar.

Für ein Ziel g ist die Konfliktmenge $C = \{g' \in G \mid g' \rightarrow \neg g\}$ die Menge aller Ziele im Zielstapel, die nicht zugleich mit g erreicht werden können. Die Redundanzmenge $R = \{g' \in G \mid g' \rightarrow g\}$ umfaßt diejenigen Ziele, deren Erreichen (gegebenenfalls unter einer geeigneten Variablenbelegung) auch g erfüllt.

Nachrichtentypen

Die Interaktion mit dem Zielstapel geschieht durch die nachfolgenden Nachrichtentypen:

Name	AddGoal
Typ	Auftrag
Beschreibung	fügt ein Ziel anhand der Priorität in den Zielstapel ein; Konflikt- und Redundanzmenge des Ziels werden berechnet; gegebenenfalls wird das Ziel in die Konflikt- und Redundanzmengen anderer Ziele aufgenommen
Inhalt	ein Ziel g
Ergebnis	Result-Nachricht, true falls $g \notin G$, false sonst
Änderung	$G' = [g_1, \ldots, g_i, g, g_j, \ldots, g_n]$ für i, j mit Priorität(g_i) < Priorität(g) und Priorität(g) ≤ Priorität(g_j)

Name	RemoveGoal
Typ	Auftrag
Beschreibung	entfernt ein Ziel und gegebenenfalls dazu redundante Ziele aus dem Zielstapel; die zu entfernenden Ziele werden aus den Konflikt- und Redundanzmengen der verbleibenden Ziele entfernt
Inhalt	ein Ziel g und die Angabe, ob auch redundante Ziele entfernt werden sollen (falls das Ziel erreicht wurde)
Ergebnis	Result-Nachricht, true falls $g \in G$, false sonst
Änderung	$G' = G \setminus (\{g\} \cup \{g' \in G \mid g \in$ Redundanzmenge(g')$\})$, inkl. Redundanzen $G' = G \setminus \{g\}$, sonst

Name	SelectGoal
Typ	Auftrag
Beschreibung	ruft das als nächstes zu bearbeitende Ziel g aus dem Zielstapel ab; dies ist das rechteste Ziel im Zustand unreached, das keine Konflikte oder Redundanzen besitzt zu weiter rechts stehenden Zielen sowie zu Zielen im Zustand intended; der Zustand dieses Ziels wird zu intended geändert
Ergebnis	InformGoal-Nachricht mit dem als nächstes zu bearbeitenden Ziel

Name	FindGoal
Typ	Auftrag
Beschreibung	sucht Ziele im Zielstapel
Inhalt	eine Formel $f(x_i)$, ggf. mit freien Variablen x_i
Ergebnis	InformGoal-Nachricht mit allen Zielen, deren Zielformel die gleiche oder eine spezifischere Situation ausdrückt als f; dies sind die Ziele, die bei Erfülltheit von f mit einer passenden Variablenbelegung ebenfalls erfüllt sind, d.h. es gibt eine Variablenbelegung $[x_i / v_i]$, so daß $f(x_i)[x_i / v_i] \rightarrow$ goal

Name	MonitorGoal
Typ	Registrierung
Beschreibung	Anmeldung zur Überwachung des Zielstapels auf Änderungen; optional kann dabei ein zu überwachendes Muster angegeben werden
Inhalt	eine Formel $f(x_i)$, ggf. mit freien Variablen x_i
Ergebnis	InformGoal-Nachricht für jede (gegebenenfalls analog zu FindGoal dem Muster $f(x_i)$ entsprechende) Änderung

Name	InformGoal
Typ	Antwort-Benachrichtigung
Beschreibung	informiert über das Bestehen von Zielen
Inhalt	eine Menge von Zielen sowie deren Vorhandensein

Zur Verwaltung des Zielstapels gibt es die Nachrichtentypen AddGoal und RemoveGoal, über die Ziele gemäß der Priorität und der Reihenfolge des Aufstellens eingefügt werden sowie unter Berücksichtigung von Redundanzen entfernt werden. SelectGoal wählt das als nächstes zu bearbeitende Ziel aus. FindGoal und MonitorGoal ermöglichen die Suche nach Zielen und die Überwachung von Änderungen im Zielstapel. InformGoal wird als Antwort für Mitteilungen über Ziele verwendet.

Rolle

```
[CASA.GoalStack,
 {CASA.KnowledgeBase},
 {[AddGoal, {CASA.GoalSelection}],
  [RemoveGoal, {CASA.GoalSelection}],
  [SelectGoal, {CASA.GoalSelection}],
  [FindGoal, {CASA.Main}],
  [MonitorGoal, {CASA.Main}]},
 {[Goals, Goal[], Goal[], ∅, r]}]
```

Die Rolle für den Zielstapel heißt CASA.GoalStack und besitzt als Rollengruppe die der Wissensbasis. Vermittels der Nachrichtentypen AddGoal, RemoveGoal und SelectGoal darf lediglich die Komponente für die Zielauswahl den Zielstapel verändern, während alle Rollen mit FindGoal und MonitorGoal auf die aktuellen Ziele zugreifen dürfen. Für den Zugriff auf sämtliche Ziele gibt es die Eigenschaft Goals der Liste der aktuellen Ziele.

4.4.4. Planbibliothek

Die Planbibliothek enthält alle Operatoren, die der Agent ausführen kann. Dazu gehören auch diejenigen Dienste, die andere Agenten anbieten und die der Agent nutzen kann.

Bei den Operatoren ist zu unterscheiden zwischen Operatoren, die direkt ausgeführt werden können, und solchen, die nur zur Planung oder als Teil eines Skripts verwendet werden können. Eine besondere Behandlung verlangen auch die Protokolloperatoren, da diese nicht zum Erreichen von Zielen, sondern zur Umsetzung von Protokollen im Rahmen von Dienstnutzungen verwendet werden.

Inhalt

Die Planbibliothek ist eine Menge $O = \{o_1, \ldots, o_n\}$ an Operatoren, die der Agent zur Handlungsauswahl und Planung verwenden kann.

Definition 8: Operatoren

> Ein Operator o ist ein 4-Tupel [pre, cond, eff, exec]. Die Vorbedingung pre und die Rahmenbedingung cond sind Formeln. Der Effekt eff ist eine nicht-leere Menge möglicher Effekte in Form von Formeln. Dient der Operator der Umsetzung einer Rolle eines Protokolls für einen Dienst, so ist der Effekt ein Tripel [service, prot, role]. Der Ausführungsteil exec beschreibt die Ausführung des Operators.

Die Formeln der Bedingungen und Effekte sind normalerweise auf Konjunktionen beschränkt, so daß sie eindeutige Aussagen ausdrücken.

Der Ausführungsteil von Operatoren umfaßt zumindest Handlungsprimitive, deren Umsetzung durch Anwendungskomponenten geschieht, und Dienstoperatoren, die Interaktionen zwischen Agenten beschreiben und in Form von Dienstnutzungen umgesetzt werden. Neben diesen individuellen und interaktiven Handlungen sind weitere Operatortypen möglich. Abstrakte Operatoren beschreiben mögliche Handlungen ohne eine konkrete Realisierung, so daß zur Ausführung eine erneute Handlungsauswahl nötig ist. Bei Schlußfolgerungen ergibt sich der Effekt logisch aus den Vorbedingungen, weshalb nur ein Effekt möglich ist. Skripte und Pläne beschreiben zusammengesetzte Handlungsabläufe, wobei Pläne eine rein logische Verknüpfung der einzelnen Handlungen besitzen, während diese in

Skripten durch Skriptanweisungen gesteuert werden. In Protokolloperatoren können Operatoren zum Senden und Empfangen von Sprechakten enthalten sein.

Operatoren können zusätzlich Auswahlkriterien zugeordnet sein. Dies kann eine statische Priorität oder auch eine von der jeweiligen Instantiierung abhängige Bewertungsfunktion sein.

Nachrichtentypen

Für die Planbibliothek werden die folgenden Nachrichtentypen verwendet:

Name	AddOperator
Typ	Auftrag
Beschreibung	fügt einen Operator zur Planbibliothek hinzu
Inhalt	ein Operator o
Ergebnis	Result-Nachricht, true falls $o \notin O$, false sonst
Änderung	$O' = O \cup \{o\}$

Name	RemoveOperator
Typ	Auftrag
Beschreibung	entfernt einen Operator aus der Planbibliothek
Inhalt	ein Operator o
Ergebnis	Result-Nachricht, true falls $o \in O$, false sonst
Änderung	$O' = O \setminus \{o\}$

Name	FindOperator
Typ	Auftrag
Beschreibung	sucht einen Operator anhand des Effekts
Inhalt	eine Formel f
Ergebnis	InformOperator-Nachricht mit allen Operatoren, deren Effekt die gleiche oder eine allgemeinere Situation ausdrückt als f; dies sind die Operatoren, deren erfolgreiche Ausführung f bei einer passenden Variablenbelegung erfüllt, d.h. es gibt eine Variablenbelegung $[x_i / v_i]$, so daß $eff(x_i)[x_i / v_i] \rightarrow f$

Name	MonitorOperator
Typ	Registrierung
Beschreibung	Anmeldung zur Überwachung der Planbibliothek auf Änderungen; optional kann dabei ein zu überwachendes Muster angegeben werden
Inhalt	eine Formel f
Ergebnis	InformOperator-Nachricht für jede (gegebenenfalls analog zu FindOperator dem Muster f entsprechende) Änderung

Name	FindProtocol
Typ	Auftrag
Beschreibung	sucht einen Protokolloperator
Inhalt	der Name des Dienstes, der Name des Protokolls und die Rolle
Ergebnis	InformOperator-Nachricht mit allen Protokolloperatoren gemäß angegebenen Dienstnamen, Protokollnamen und Rolle

Name	MonitorProtocol
Typ	Registrierung
Beschreibung	Anmeldung zur Überwachung der Planbibliothek auf Änderungen bezüglich von Protokolloperatoren; optional kann dabei ein zu überwachendes Muster angegeben werden
Inhalt	der Name des Dienstes, der Name des Protokolls und die Rolle
Ergebnis	InformOperator-Nachricht für jede (gegebenenfalls analog zu FindProtocol zutreffende) Änderung

Name	InformOperator
Typ	Antwort-Benachrichtigung
Beschreibung	informiert über vorhandene Operatoren
Inhalt	eine Menge von Operatoren sowie deren Vorhandensein

Vermittels der Nachrichtentypen AddOperator und RemoveOperator kann die Operatorenmenge eines Agenten verändert werden, indem Operatoren hinzugefügt oder entfernt werden. FindOperator und MonitorOperator dienen dem Zugriff auf die Handlungsoperatoren zum Erreichen von Zielen, FindProtocol und MonitorProtocol dem Zugriff auf die Interaktionsoperatoren zur Umsetzung von Protokollen. Mit InformOperator werden die Ergebnisse des Zugriffs mitgeteilt.

Rolle

```
[CASA.PlanLibrary,
 {CASA.KnowledgeBase},
 {[AddOperator, {CASA.Main}],
  [RemoveOperator, {CASA.Main}],
  [FindOperator, {CASA.Main}],
  [MonitorOperator, {CASA.Main}],
  [FindProtocol, {CASA.Main}],
  [MonitorProtocol, {CASA.Main}]},
 {[Operators, Operator{}, Operator{}, ∅, ri]}]
```

Die Rolle der Planbibliothek heißt CASA.PlanLibrary und gehört zur Rollengruppe der Wissensbasis. Sämtlich Nachrichtentypen für Zugriffe und Veränderungen der Planbibliothek können von allen Rollen genutzt werden. Über die Eigenschaft Operators erhält man die Menge aller in der Planbibliothek enthaltenen Operatoren und kann man die initiale Menge an Operatoren festlegen.

4.4.5. Intentionenstruktur

In der Intentionenstruktur befinden sich die auszuführenden Operatorinstanzen geordnet nach der Reihenfolge der anstehenden Ausführung.

Wie bei Zielen sind die wichtigsten Ordnungskriterien die zeitliche Reihenfolge des Aufstellens und die Priorität des Ziels, für das eine Intention aufgestellt wurde. Zusätzlich sollten die Intentionen anhand von Redundanzen und Konflikten koordiniert sein, um eine möglichst problemlose Ausführung zu gewährleisten. Weitere Kriterien wie ein expliziter Start- oder Endpunkt sind bei Bedarf ebenfalls zu berücksichtigen.

Inhalt

Die Intentionenstruktur ist eine geordnete Liste $I = [i_1, \ldots, i_n]$ an Intentionen. Die Reihenfolge entspricht dem vorgesehenen Ausführungsbeginn von rechts nach links. Gemäß der nachstehenden Definition stellt die Intentionenstruktur einen Plan dar und kann als Gesamtplan des Agenten für seine nächsten Aktivitäten aufgefaßt werden.

Definition 9: Intentionen

Eine Intention i ist ein 6-Tupel [op, var, Pre, C^{pre}, C^{cond}, state]. Der auszuführende Operator op zusammen mit der Instantiierung var = $[x_i / v_i]$ der freien Variablen aus dessen formaler Beschreibung stellen die eigentliche Intention dar. Die übrigen Parameter werden zur Bearbeitung benötigt. Pre ist die Menge der Voraussetzungen, C^{pre} und C^{cond} sind die Konfliktmengen für die Vorbedingungen bzw. Rahmenbedingungen. state beinhaltet den jeweiligen Bearbeitungszustand.

Die Voraussetzungen für eine Intention sind diejenigen Intentionen in der Intentionenstruktur, die vorher ausgeführt werden müssen, weil deren Effekt die Vorbedingungen der Intention (mit) herbeiführt. Die Konfliktmengen einer Intention enthalten diejenigen Intentionen, die nicht vorher bzw. nicht vorher oder zugleich ausgeführt werden dürfen, da deren Effekt die Vorbedingung bzw. Rahmenbedingung der Intention verletzt:

$C^{pre}(i) = \{i' \in I \mid \neg(pre(i) \wedge eff(i'))\}$
$C^{cond}(i) = \{i' \in I \mid \neg(cond(i) \wedge eff(i'))\}$
$C(i) = C^{pre}(i) \cup C^{cond}(i)$

Für Intentionen werden die folgenden Bearbeitungszustände unterschieden:

- *unexecuted*: Die Intention wurde noch nicht ausgeführt.
- *executing*: Die Intention befindet sich in der Ausführung.
- *executed*: Die Ausführung wurde erfolgreich beendet.
- *failed*: Die Ausführung ist fehlgeschlagen.

Eine besondere Form von Intentionen stellen Pläne dar, die mehrere Intentionen bereits zu einer komplexen Handlung verknüpfen.

Definition 10: Pläne

Ein Plan P ist eine nach der Ausführungsreihenfolge geordnete Liste $[i_1, \ldots, i_n]$ von Intentionen, die anhand der Voraussetzungen und Konfliktmengen logisch untereinander verknüpft sind.

Die Liste stellt dabei eine vollständige Ordnung, die Voraussetzungen und Konfliktmengen eine partielle Ordnung zur Ausführung dar. Die einzelnen Instantiierungen der Operatoren können auch Variablen anderer Operatoren enthalten.

Ein Plan P ist konsistent, wenn für alle Teilintentionen i_k gilt:

- Die Voraussetzungen und die Elemente der Konfliktmengen von i_k entstammen dem Plan, enthalten aber nicht i_k selbst:
  $Pre(i_k) \cup C^{pre}(i_k) \cup C^{cond}(i_k) \subseteq P \setminus \{i_k\}$
- Die Voraussetzungen i_l von i_k müssen vor i_k selbst ausgeführt werden und stehen deshalb weiter rechts:
  $\forall i_l \in Pre(i_k): k < l$
- Keine vor i_k ausgeführte Teilintention i_m ($k < m$) gehört zu einer der Konfliktmengen von i_k, außer wenn es eine oder mehrere zwischen i_m und i_k ausgeführte Intentionen aus $Pre(i_k)$ gibt, die die durch i_m bedrohte Bedingung wiederherstellen.

Insgesamt bedeutet dies, daß die Bedingungen von i_k erfüllt sind, wenn die vorhergehenden Intentionen erfolgreich ausgeführt wurden.

Nachrichtentypen

Zur Interaktion mit der Intentionenstruktur gibt es die folgenden Nachrichtentypen:

Name	AddIntention
Typ	Auftrag
Beschreibung	fügt eine Intention in die Intentionenstruktur ein; Pläne werden dabei in ihre Einzelschritte aufgelöst; Abhängigkeiten zwischen Intentionen (Voraussetzungen und Konflikte) werden neu berechnet; gegebenenfalls lassen sich Konflikte analog zur Plangenerierung durch Erweitern der Voraussetzungen oder zusätzliche Intentionen auflösen
Inhalt	eine Intention i
Ergebnis	Result-Nachricht, true falls $i \notin I$ und sich i in den Gesamtplan konfliktfrei integrieren läßt, false sonst

Name	RemoveIntention
Typ	Auftrag
Beschreibung	entfernt eine Intention und gegebenenfalls davon abhängige Intentionen aus der Intentionenstruktur; die zu entfernenden Intentionen werden aus den Voraussetzungen und Konfliktmengen der verbleibenden Intentionen entfernt
Inhalt	eine Intention i und die Angabe, ob auch abhängige Intentionen entfernt werden sollen (falls die Intention nicht erfolgreich ausgeführt wurde)
Ergebnis	Result-Nachricht, true falls $i \in I$, false sonst

Name	SelectIntention
Typ	Auftrag
Beschreibung	ruft die als nächste auszuführende Intention i aus der Intentionenstruktur ab; dies ist die rechteste Intention im Zustand unexecuted, für die alle Voraussetzungen im Zustand executed sind und kein Konflikt besteht; der Zustand dieser Intention wird zu executing geändert
Ergebnis	InformIntention-Nachricht mit der als nächstes auszuführenden Intention

Name	FindIntention
Typ	Auftrag
Beschreibung	sucht Intentionen in der Intentionenstruktur
Inhalt	ein Operator o
Ergebnis	InformIntention-Nachricht mit allen Intentionen, deren Operator der angegebene Operator o ist

Name	MonitorIntention
Typ	Registrierung
Beschreibung	Anmeldung zur Überwachung der Intentionenstruktur auf Änderungen; optional kann dabei ein zu überwachender Operator angegeben werden
Inhalt	ein Operator o
Ergebnis	InformIntention-Nachricht für jede (gegebenenfalls den Operator o betreffende) Änderung

Name	InformIntention
Typ	Antwort-Benachrichtigung
Beschreibung	informiert über das Bestehen von Intentionen
Inhalt	eine Menge von Intentionen sowie deren Vorhandensein

Über die Nachrichtentypen AddIntention und RemoveIntention wird eine Intention in die Intentionenstruktur eingefügt bzw. aus dieser gegebenenfalls inklusive ihrer Abhängigkeiten entfernt. SelectIntention wählt die nächste auszuführende Intention aus. Mit FindIntention wird nach Intentionen gesucht und mit MonitorIntention Änderungen in der Intentionenstruktur überwacht. Nachrichten vom Typ InformIntention übermitteln Auskünfte über den Inhalt der Intentionenstruktur.

Rolle

```
[CASA.IntentionStructure,
 {CASA.KnowledgeBase},
 {[AddIntention, {CASA.Scheduler}],
  [RemoveIntention, {CASA.Scheduler}],
  [SelectIntention, {CASA.Scheduler}],
  [FindIntention, {CASA.Main}],
  [MonitorIntention, {CASA.Main}]},
 {[Intentions, Intention[], Intention[], ∅, r]}]
```

CASA.IntentionStructure bezeichnet die Rolle der Intentionenstruktur, die zur Wissensbasis-Rollengruppe gehört. Eine Sendeberechtigung für die Nachrichtentypen zur Veränderung der Intentionenstruktur über AddIntention, RemoveIntention und SelectIntention erhält nur die Rolle für den Scheduler, der Zugriff über FindIntention und MonitorIntention ist allen Rollen erlaubt. Die Eigenschaft Intentions hat als Wert den jeweils aktuellen Inhalt der Intentionenstruktur.

4.4.6. Dienstbibliothek

In der Dienstbibliothek befinden sich alle diejenigen Dienste, die der Agent nach außen hin zur Nutzung durch andere Agenten anbietet. Ihr Inhalt wird von der Kommunikationskomponente bei Dienstanfragen verwendet und über den Namen des Dienstes referenziert.

Inhalt

Die Dienstbibliothek besteht aus einer Menge $S = \{s_1, \ldots, s_n\}$ an Dienstoperatoren. Ein Dienstoperator ist ein Operator, dessen Ausführungsteil eine Dienstbeschreibung enthält, die alle zur Durchführung einer Dienstnutzung notwendigen Informationen umfaßt.

Definition 11: Dienstbeschreibungen

> Eine Dienstbeschreibung s ist ein Tupel [name, protocols]. name gibt den Namen des Dienstes an, der (global) eindeutig der Identifikation des Dienstes dient. protocols = $\{p_1, \ldots, p_n\}$ ist eine Menge von Protokollen, über die der Dienst genutzt werden kann.

Eine Dienstbeschreibung kann noch um beliebige weitere generische Parameter wie den Preis zur Dienstnutzung oder Sicherheitsanforderungen ergänzt werden, die dann bei der Auswahl und Durchführung von Diensten entsprechend berücksichtigt werden können.

Nachrichtentypen

Die Interaktion mit der Dienstbibliothek geschieht unter Verwendung der folgenden Nachrichtentypen:

Name	AddService
Typ	Auftrag
Beschreibung	fügt einen Dienst zur Dienstbibliothek hinzu
Inhalt	ein Dienst s
Ergebnis	Result-Nachricht, true falls es noch keinen Dienst mit demselben Namen in der Dienstbibliothek gibt, false sonst
Änderung	S' = S ∪ {s}

Name	RemoveService
Typ	Auftrag
Beschreibung	entfernt einen Dienst aus der Dienstbibliothek
Inhalt	ein Dienst s
Ergebnis	Result-Nachricht, true falls s ∈ S, false sonst
Änderung	S' = S \ {s}

Name	FindService
Typ	Auftrag
Beschreibung	sucht einen Dienst anhand des Dienstnamens
Inhalt	ein Dienstname n
Ergebnis	InformOperator-Nachricht mit dem Dienstoperator des angegebenen Namens n, falls vorhanden, sonst mit leerer Menge

Name	MonitorService
Typ	Registrierung
Beschreibung	Anmeldung zur Überwachung der Dienstbibliothek auf Änderungen; optional kann dabei ein zu überwachender Name angegeben werden
Inhalt	ein Dienstname n
Ergebnis	InformOperator-Nachricht für jede (gegebenenfalls den Namen n betreffende) Änderung

Analog zu den übrigen Rollen der Wissensbasis gibt es mit AddService und RemoveService Nachrichtentypen zum Hinzufügen und Entfernen von Diensten sowie mit FindService und MonitorService Nachrichtentypen zur Suche und Überwachung. Da Dienste besondere Operatoren sind, werden Ergebnisse mit dem bereits bei der Planbibliothek definierten Nachrichtentyp InformOperator mitgeteilt.

Rolle

```
[CASA.ServiceLibrary,
 {CASA.KnowledgeBase},
 {[AddService, {CASA.Main}],
  [RemoveService, {CASA.Main}],
  [FindService, {CASA.Main}],
  [MonitorService, {CASA.Main}]},
 {[Services, Operator{}, Operator{}, ∅, ri]}]
```

Die Rolle der Dienstbibliothek trägt die Bezeichnung CASA.ServiceLibrary und ist Teil der Rollengruppe der Wissensbasis. Für die verarbeitbaren Nachrichtentypen AddService, RemoveService, FindService und MonitorService erhalten alle Rollen die Sendeberechtigung. Über die Eigenschaft Services kann die Menge aller vorhandenen Dienste initiiert und abgerufen werden.

4.5. Kontrolleinheit

Die Kontrolleinheit steuert das Verhalten eines Agenten unter Nutzung des in der Wissensbasis gespeicherten Wissens. Ihre Komponentenrollen lassen sich gemäß ihrer Aufgabe in vier Gruppen für informative, reaktive, deliberative und interaktive Rollen unterteilen.

Die informativen Rollen liefern die zur Verhaltenssteuerung benötigten Informationen über die gegenwärtige Situation. Dazu gehören die folgenden Rollen:

- *Faktenaktualisierung*: Die Faktenaktualisierung verwaltet das in der Faktenbasis enthaltene Wissen über den aktuellen Weltzustand.
- *Zeitgeber*: Der Zeitgeber stellt eine interne Uhr für zeitgesteuerte Vorgänge zur Verfügung.

Die Umsetzung der durch die in der Regelbasis enthaltenen Reaktionsregeln ausgedrückten reaktiven Dispositionen obliegt der nachfolgenden Rolle:

- *Situationserkennung*: Die Situationserkennung überwacht die Faktenbasis auf das Eintreten der in den Reaktionsregeln angegebenen Situationen und führt die diesen zugeordneten Aktionen aus.

Das zielgerichtete Verhalten eines Agenten ergibt sich aus dem Zusammenwirken der deliberativen Rollen der Kontrolleinheit:

- *Zielauswahl*: Die Zielauswahl verwaltet den Zielstapel und wählt aus diesem die zu verfolgenden Ziele aus. Um ein Ziel aktiv zu verfolgen, initiiert sie für dieses die Auswahl von Handlungen.
- *Handlungsauswahl*: Die Handlungsauswahl wählt anhand der Planbibliothek Handlungen zum Erreichen der jeweils verfolgten Ziele aus und stellt diese als Intentionen auf. Gegebenenfalls kann dazu auch die Generierung von Plänen angestoßen werden.
- *Plangenerierung*: Die Plangenerierung erzeugt neue Pläne durch die logische Verkettung von Operatoren zur Erfüllung eines Zielzustands.
- *Handlungsbewertung*: Die Handlungsbewertung ermöglicht Entscheidungen zwischen verschiedenen Handlungsalternativen, indem sie instantiierte Operatoren und Pläne bewertet.

- *Scheduler*: Der Scheduler verwaltet und koordiniert die Intentionen eines Agenten. Er wählt die auszuführenden Intentionen aus und initiiert deren Ausführung.
- *Handlungsausführung*: Die Handlungsausführung interpretiert die auszuführenden Intentionen und setzt die in ihnen ausgedrückten Handlungen um. Gegebenenfalls wird mit der eigentlichen Ausführung eine dafür vorgesehene Komponentenrolle beauftragt.

Das interaktive Verhalten eines Agenten stellt einen Sonderfall des deliberativen Verhaltens dar, in dessen Umsetzung auch andere Agenten involviert sind. Indirekte Interaktionen zwischen Agenten sind über Veränderungen der gemeinsamen Umgebung möglich. Interaktionen auf einer Informationsebene werden von einer eigenen Komponentenrolle unterstützt:

- *Kommunikation*: Die Kommunikationskomponente organisiert und koordiniert die kommunikativen Handlungen eines Agenten. Sie ist für das Empfangen und Versenden von Sprechakten innerhalb von Konversationen sowie für die Durchführung von Dienstnutzungen zuständig.

Die Komponentenrollen der Kontrolleinheit besitzen aufgrund ihrer unterschiedlichen Aufgaben anders als die Wissensbasiskomponenten keine einheitlichen Grundfunktionalitäten oder gleichartigen Nachrichtentypen.

Rollengruppe

```
[CASA.ControlUnit,
 {CASA.Main},
 {},
 {}]
```

Die Rollengruppe für die Komponentenrollen der Kontrolleinheit heißt CASA.ControlUnit. Sie definiert keine allen Rollen gemeinsamen Sendeberechtigungen oder Eigenschaften.

4.5.1. Faktenaktualisierung

Die Faktenaktualisierung ist verantwortlich für die Verwaltung des in der Faktenbasis gespeicherten Faktenwissens. Dies umfaßt zumindest das Anpassen der Faktenbasis an Änderungen des aktuellen Weltzustands, die andere Komponenten der Faktenaktualisierung mitteilen, zusätzlich kann der Faktenbestand auch aktiv gepflegt werden. In jedem Fall ist es die zentrale Aufgabe der Faktenaktualisierung, das Faktenwissen aktuell und konsistent zu halten.

Deshalb darf nur sie den Inhalt der Faktenbasis direkt verändern, alle anderen Komponenten müssen Änderungen des Faktenwissens über die Faktenaktualisierung vornehmen, damit diese eine konsistente Aktualisierung unter Berücksichtigung der bereits vorhandenen Fakten vornehmen kann. Dabei muß die Faktenaktua-

lisierung wie die Faktenbasis außer mit einzelnen Fakten auch mit Aussagen in Form von Formeln umgehen können und die Faktenbasis entsprechend dem Gehalt einer behaupteten Aussage aktualisieren, sofern dies ohne Widersprüche oder Mehrdeutigkeiten möglich ist.

Für eine aktive Wartung des Faktenbestands müssen zusätzliche Angaben über die einzelnen Fakten bezüglich ihrer Verläßlichkeit und ihrer Abhängigkeiten untereinander gesammelt, gespeichert und verwaltet werden. Dazu können Angaben gehören wie der Zeitpunkt der letzten Bestätigung, die Quellen und deren Bewertung der Gewißheit sowie gegebenenfalls stützende Fakten als Rechtfertigung. So kann veraltetes, unzuverlässiges oder nicht mehr gerechtfertigtes Wissen entfernt werden oder bei der Evaluierung von Formeln entsprechend berücksichtigt werden, so daß eine Aussage nur mit einem bestimmten Grad an Gewißheit von der Faktenbasis gestützt wird. Weiterhin können Axiome verwendet werden, über die sich versteckte Inkonsistenzen aufdecken lassen, die dann anhand der Verläßlichkeit oder durch aktives Überprüfen der widersprechenden Fakten aufgelöst werden.

Funktion

Die Faktenaktualisierung leitet aus dem bestehenden Inhalt der Faktenbasis und neu gewonnenem Faktenwissen den neuen Inhalt der Faktenbasis ab.

Definition 12: Faktenaktualisierung

> Die Faktenaktualisierung ist eine Abbildung FM: $F(t) \times F^{+}(t) \times F^{-}(t) \rightarrow F(t+1)$. Die Menge der neuen Fakten F(t+1) ergibt sich jeweils aus der Menge der alten Fakten F(t) und der aktuellen Änderungen an hinzuzufügenden Fakten $F^{+}(t)$ und zu entfernenden Fakten $F^{-}(t)$.

Die Menge der alten und neuen Fakten ist jeweils der Inhalt der Faktenbasis zu einem bestimmten Zeitpunkt. Die vorzunehmenden Änderungen ergeben sich zum einen aus den Mitteilungen anderer Komponenten über Änderungen des Weltzustands, zum anderen aus der aktiven Wartung des Faktenbestands durch die Faktenaktualisierung selbst.

Nachrichtentypen

Veränderungen im Wissen über den Weltzustand werden der Faktenaktualisierung mit Nachrichten vom Typ ChangeFacts mitgeteilt:

Name	ChangeFacts
Typ	Auftrag
Beschreibung	teilt die Wahrheit, Unbekanntheit oder Unwahrheit einer Aussage mit; die Faktenbasis wird so angepaßt, daß sie den angegebenen Wahrheitsgehalt der behaupteten Aussage widerspiegelt, falls dies widerspruchsfrei möglich ist
Inhalt	eine Aussage a und ihr Wahrheitswert t
Annahme	Accept-Nachricht, true falls die Behauptung eine eindeutige Änderung erlaubt, false sonst[31]
Ergebnis	Result-Nachricht, true falls die Faktenbasis gemäß der Behauptung geändert wurde, false sonst

Die Rücknahme von Änderungen der Faktenbasis ist durch Retract-Nachrichten gemäß dem Interaktionsschema für Aufträge nur möglich, solange die eigentliche Änderung noch nicht vorgenommen wurde. Andernfalls wird eine Retract-Nachricht nicht angenommen. Eine Rücknahme durch Aufstellen einer gegenteiligen Aussage stellt i.a. nicht den ursprünglichen Zustand wieder her, da dabei u.U. auch ursprünglich redundante Änderungen mit zurückgenommen werden.

Rolle

```
[CASA.FactMaintenance,
 {CASA.ControlUnit},
 {[ChangeFacts, {CASA.Main}]},
 {}]
```

Die Rolle der Faktenaktualisierung ist CASA.FactMaintenance und gehört zur Rollengruppe der Kontrolleinheit. Die einzige Sendeberechtigung geht für Change-Facts an alle Rollen, Eigenschaften gibt es keine.

4.5.2. Zeitgeber

Der Zeitgeber dient dem Agenten und seinen Komponenten als interne Uhr und gemeinsame Zeitbasis. Er liefert die Referenzzeit für alle zeitabhängigen Prozesse. Im Normalbetrieb des Agenten sollte dies die „echte“ Zeit sein, also i.d.R. die aktuelle Systemzeit des Betriebssystems. Befindet sich der Agent in einem besonderen Zustand wie dem Schrittmodus, in dem der Ablauf des Agenten nicht in Realzeit, sondern unter besonders kontrollierten Bedingungen stattfindet, wird eine entsprechend angepaßte Zeit zugrundegelegt.

[31] Beispielsweise erlaubt eine wahre disjunktive Aussage in der Regel keine eindeutige Aktualisierung des Faktenwissens, da verschiedene Faktenmengen eine Disjunktion wahr machen können. Der Umfang an akzeptierten Nachrichten braucht nicht für jede Implementierung einer Komponente zur Faktenaktualisierung übereinstimmen, da die Eindeutigkeit einer Aussage nicht trivial festzustellen ist. So ist eine Disjunktion einer eindeutigen Aussage mit einer komplexen Kontradiktion eindeutig, wozu aber erst die Kontradiktion erkannt werden muß.

Die Aufgabe des Zeitgebers besteht in Auskünften über die aktuelle interne Zeit sowie in Alarmfunktionalitäten, bei denen Komponenten nach einer Anmeldung über das Erreichen eines vorgegebenen Ereignisses benachrichtigt werden. Dieses Ereignis kann im Eintreten eines Zeitpunkts, dem Ablaufen eines Zeitintervalls oder der Vorgabe eines gleichmäßigen periodischen Takts bestehen.

Funktion

Der Zeitgeber stellt die aktuelle Zeit T des Agenten zur Verfügung.

Definition 13: Zeitangabe

Eine Zeitangabe t ist ein 8-Tupel [year, month, day, hour, min, sec, milli, type]. Das Datum wird durch Jahr, Monat und Tag, die Uhrzeit durch Stunde, Minute, Sekunde und Millisekunde im üblichen Format[32] angegeben. Der Typ gibt an, ob es sich um eine absolute, relative oder periodische Zeitangabe handelt.

Eine absolute Zeitangabe gibt einen bestimmten Zeitpunkt an, eine relative eine Zeitspanne gemessen ab der aktuellen Zeit. Eine periodische Zeitangabe ist eine sich wiederholende relative Zeitspanne.

Definition 14: Zeitgeber

Der Zeitgeber ist eine absolute Zeitangabe T, die die aktuelle Zeitannahme des Agenten beinhaltet.

Die Zeitangabe T des Zeitgebers entspricht i.a. der aktuellen Systemzeit. Im Schrittmodus kann sie aufgrund der verlangsamten Ausführung von dieser abweichen.

Nachrichtentypen

Der Zugriff auf die aktuelle interne Zeit des Agenten geschieht über die beiden folgenden Nachrichten:

Name	SetTimer
Typ	Auftrag (absolute / relative Zeit) bzw. Registrierung (periodische Zeit)
Beschreibung	Anmeldung für das Eintreten bestimmter Zeitpunkte
Inhalt	eine Zeitangabe t
Annahme	Accept-Nachricht, true falls die Zeitangabe nicht in der Vergangenheit liegt, false sonst
Ergebnis	InformTime-Nachricht, sobald bzw. jedesmal wenn der angegebene Zeitpunkt t eintritt

[32] Absolute Datumsangaben werden beginnend bei 1, relative Datumsangaben und Zeitangaben beginnend bei 0 gerechnet. Bei absoluten Datumsangaben gelten die kalendarischen Einschränkungen des Wertebereichs.

Name	InformTime
Typ	Antwort-Benachrichtigung
Beschreibung	informiert über die aktuelle Zeit
Inhalt	eine absolute Zeitangabe t

Mit Nachrichten des Typs SetTimer wird eine zu überwachende Zeitangabe angemeldet, deren Eintreten über InformTime mitgeteilt wird. Die aktuelle Zeit erhält man bei einer relativen Zeitangabe 0. Damit durch das Verweilen in der Warteschlange keine Zeitverzögerungen entstehen, sollte bei SetTimer als Antwortpriorität und gegebenenfalls auch als Priorität immer *direkt* verwendet werden.

Rolle

```
[CASA.Timer,
 {CASA.ControlUnit},
 {[SetTimer, {CASA.Main}]},
 {[CurrentTime, Time, Time, 0, riw]}]
```

CASA.Timer ist die Rolle für den Zeitgeber, die zur Rollengruppe der Kontrolleinheit gehört. Für den Nachrichtentyp SetTimer erhalten alle Rollen eine Sendeberechtigung. Über die Eigenschaft CurrentTime kann die aktuelle Zeit abgefragt und eingestellt werden.

4.5.3. Situationserkennung

Die Situationserkennung überwacht das Faktenwissen des Agenten auf das Eintreten der Situationen hin, die in den Reaktionsregeln der Regelbasis beschrieben sind. Dazu meldet sie sich bei der Faktenbasis mit einer MonitorFact-Nachricht an, um über alle Faktenänderungen informiert zu werden.

Ändert sich mit der Änderung von Fakten eine mögliche Erfülltheit der Bedingung einer Regel, so wird deren Aktionsteil ausgeführt. Je nach Art des Aktionsteils wird eine Nachricht zu dessen Umsetzung an eine entsprechende Komponente gesendet.

Die Situationserkennung benötigt effiziente Algorithmen für eine Zuordnung von Faktenänderungen zu möglicherweise davon betroffenen Reaktionsregeln, um den Aufwand zur Evaluierung der Regeln möglichst gering zu halten. Auch durch eine Beschränkung der Ausdrucksmächtigkeit der Formeln zur Situationsbeschreibung kann die Effizienz der Situationserkennung gesteigert werden.

Da effiziente Algorithmen zur Situationserkennung (z.B. Rete [Forgy 1982]) i.a. eine prozeduralisierte Form der Regeln verwenden, ist eine Umsetzung der Rollen von Regelbasis und Situationserkennung in zwei getrennten Komponenten nur selten sinnvoll.

Funktion

Die Situationserkennung bildet die aktuellen Fakten und Reaktionsregeln auf eine Menge aktiver Regelinstanzen ab. Bei Änderung der Aktivierung einer Regel wird die entsprechende Aktion ausgeführt.

Definition 15: Regelinstanz

Eine Regelinstanz ist ein Tupel [r, var]. r ist eine Regel und var = $[x_i / v_i]$ eine Variablenbelegung für die in der Bedingung pre von r enthaltenen Variablen x_i.

Definition 16: Situationserkennung

Die Situationserkennung ist eine Abbildung SA: $F(t) \times R(t) \rightarrow R_A(t+1)$. F(t) sind die aktuellen Fakten und R(t) die vorhandenen Regeln. $R_A(t+1)$ ist die Menge aller erfüllten Regelinstanzen mit $R_A(t+1) = \{[r, var] \mid r \in R(t) \wedge F(t) \rightarrow pre[x_i / v_i]\}$.

$R^+(t+1) = R_A(t+1) \setminus R_A(t)$ ist die Menge der aktivierten, $R^-(t+1) = R_A(t) \setminus R_A(t+1)$ die der deaktivierten Regelinstanzen. Für die Elemente von $R^+(t+1)$ und $R^-(t+1)$ wird zum Zeitpunkt t+1 jeweils die Aktivierungsaktion bzw. falls vorhanden die Deaktivierungsaktion ausgeführt.

Nachrichtentypen

Die Situationserkennung benötigt keine eigenen Nachrichtentypen. Zur Benachrichtigung der Komponente, die die Regel aufgestellt hat, wird an diese eine InformFact-Nachricht mit der eingetretenen Situation als Inhalt gesendet. Zum Ändern von Fakten, Zielen oder Intentionen ist dies eine Nachricht vom Typ ChangeFacts, NewGoal bzw. NewIntention, die an die Komponente mit der Rolle FactMaintenance, GoalSelection bzw. Scheduler geschickt wird.

Rolle

```
[CASA.SituationAssessment,
 {CASA.ControlUnit},
 {},
 {}]
```

Die Rolle der Situationserkennung trägt den Namen CASA.SituationAssessment, ihre Rollengruppe ist die Kontrolleinheit. Sie besitzt keine Sendeberechtigungen oder Eigenschaften.

4.5.4. Zielauswahl

Die Zielauswahl verwaltet den Zielstapel. Wie bei der Faktenbasis dürfen andere Komponenten nicht direkt den Zielstapel verändern, sondern sie müssen das Hinzufügen oder Entfernen von Zielen der Zielauswahl mitteilen, die daraufhin entsprechend den Zielstapel aktualisiert. Weiterhin ist die Zielauswahl für die

Koordinierung der Ziele verantwortlich und für die Entscheidung, wann welches Ziel zu erreichen versucht werden soll.

Für jedes neu aufgestellte Ziel führt die Zielauswahl zunächst eine Bewertung durch, anhand der die Priorität bestimmt wird, gemäß der es in den Zielstapel eingeordnet wird. Im einfachsten Fall ist dies die von der aufstellenden Komponente vorgegebene Priorität, es können aber auch weitere Kriterien berücksichtigt werden wie Prioritäten bezüglich der aufstellenden Komponenten oder Abhängigkeiten zu anderen Zielen. Zuvor wird anhand der Faktenbasis festgestellt, ob das Ziel nicht bereits erfüllt ist und deshalb nicht verfolgt zu werden braucht.

Solange das Ziel im Zielstapel enthalten ist, kann die Situationserkennung verwendet werden, um zu überwachen, ob der Zielzustand bereits unabhängig von eigenen Aktivitäten zum Erreichen des Ziels eintritt. In diesem Fall braucht das Ziel nicht weiter aktiv verfolgt zu werden. Falls für ein Ziel Situationen definiert sind, unter denen es vorzeitig zurückgenommen oder trotz Erreichen aufrecht erhalten werden soll, so können auch diese mit Hilfe der Situationserkennung überwacht werden.

Aus dem Zielstapel wählt die Zielauswahl jeweils die Ziele aus, die als nächstes bearbeitet werden soll. Neben der Anordnung der Ziele im Zielstapel sind vor allem Konflikte und Redundanzen relevante Auswahlkriterien. Da ein Agent mehrere Ziele gleichzeitig aktiv verfolgen kann, kann die Anzahl der parallel auszuführenden Ziele beschränkt werden, um eine Überlastung des Agenten zu verhindern. Wird zum Erreichen eines Ziels ein oder mehrere Unterziele aufgestellt, so wird das obere Ziel suspendiert, bis alle Unterziele erreicht sind, und zählt solange auch nicht zu den aktiven Zielen, da andernfalls eine Blockade durch die Obergrenze der aktiven Ziele entstehen könnte, da diese u.U. das Verfolgen der Unterziele verhindern würde.

Ein ausgewähltes Ziel wird an die Handlungsauswahl übergeben, damit diese geeignete Handlungen zum Erreichen des Zielzustands auswählt. Wird ein Ziel erreicht – sei es aufgrund der Ausführung der dafür ausgewählten Handlungen oder durch andere Umstände – oder kann das Ziel nicht erreicht werden – weil keine Handlung gefunden wird, die Ausführung fehlschlägt oder es zurückgenommen wird –, so wird die Komponente, die das Ziel aufgestellt hat, entsprechend benachrichtigt und das Ziel aus dem Zielstapel entfernt. Wird ein bereits ausgewähltes Ziel entfernt, so wird gegebenenfalls zudem der Auftrag zur Handlungsauswahl widerrufen, so daß alle Aktivitäten zum Erreichen des Ziels eingestellt werden.

Funktion

Die Zielauswahl verwaltet den Inhalt des Zielstapels und wählt aus diesem die als nächstes zu verfolgenden Ziele aus.

Definition 17: Zielauswahl

Die Zielauswahl ist eine Abbildung GS: $G(t) \times G^{+}(t) \times G^{-}(t) \rightarrow G(t+1) \times G_A(t+1)$. Anhand der bestehenden Ziele G(t) und der hinzuzufügenden Ziele $G^{+}(t)$ und der zu entfernenden $G^{-}(t)$ wird zum einen die Menge der neuen Ziele G(t+1) sowie die Menge $G_A(t+1)$ der als nächstes zu verfolgenden Ziele abgeleitet.

Die hinzuzufügenden Ziele ergeben sich aus den Zielen, die andere Komponenten aufstellen und die nicht bereits erfüllt sind. Entfernt werden Ziele, wenn sie zurückgenommen werden sowie sobald sie erreicht wurden oder nicht erreicht werden können. Für die ausgewählten zu verfolgenden Ziele wird über eine Nachricht vom Typ ExecuteGoal die Handlungsauswahl und damit das aktive Verfolgen des Ziels initiiert.

Nachrichtentypen

Mit dem folgenden Nachrichtentyp NewGoal werden neue Ziele aufgestellt:

Name	NewGoal
Typ	Auftrag
Beschreibung	stellt ein neues Ziel auf
Inhalt	ein Ziel g
Annahme	Accept-Nachricht, true falls das Ziel in den Zielstapel eingefügt wurde, false sonst
Ergebnis	Result-Nachricht, true falls das Ziel erreicht wurde, false falls das Ziel nicht erreicht wurde

Rolle

```
[CASA.GoalSelection,
 {CASA.ControlUnit},
 {[NewGoal, {CASA.Main}]},
 {[ActiveGoals, int, {i ∈ int | i ≥ 0}, 0, r],
  [MaxActiveGoals, int, {i ∈ int | i > 0}, MAX_INT, riw]}]
```

Die Rolle der Zielauswahl namens CASA.GoalSelection ist der Rollengruppe der Kontrolleinheit zugeordnet. Für den Nachrichtentyp NewGoal wird allen Rollen eine Sendeberechtigung erteilt. Die Eigenschaft ActiveGoals gibt die Anzahl der aktiv verfolgten Ziele an, für die mit der Eigenschaft MaxActiveGoals eine Obergrenze angegeben werden kann.

4.5.5. Handlungsauswahl

Die Handlungsauswahl wählt Handlungen in Form von instantiierten Operatoren zum Erreichen von Zielen aus. Die Ziele hierfür erhält sie ausschließlich von der Zielauswahl.

Ein Operator wird über Bedingungen und Effekt beschrieben, also über den Weltzustand, der vor der Ausführung gegeben sein muß, und den, der durch die Ausführung herbeigeführt wird. Die Handlungsauswahl sucht nun in der Planbibliothek nach Operatoren, deren Effekt das Ziel erfüllt und deren Bedingungen gemäß der Faktenbasis erfüllt sind, womit eine Ausführung möglich ist und deren erfolgreicher Abschluß den Zielzustand herbeiführt. Eine Ausnahme hiervon bilden Interaktionsziele, die von der Kommunikation zur Durchführung von Protokollen aufgestellt werden. Hierbei müssen der Dienst, das zu verwendende Protokoll und die jeweilige Rolle des Agenten innerhalb des Protokolls als Auswahlkriterium verwendet werden.

Existieren mehrere mögliche Operatorinstanzen zum Erreichen eines Ziels, so wählt die Handlungsauswahl die geeignetste aus, wobei falls vorhanden entsprechende durch den Operator vorgegebene Entscheidungskriterien verwendet werden. Zudem besteht dann die Möglichkeit, im Falle eines Scheiterns der Ausführung eine alternative Handlung zu versuchen, anstatt das Ziel als nicht erreicht zu beenden.

Eine ausgewählte Handlung wird als Intention an den Scheduler übergeben, der schließlich das Ergebnis der Ausführung mitteilt. Bei einem Fehlschlag kann gegebenenfalls eine andere Handlung ausgewählt werden, andernfalls wird das Ergebnis an die Zielauswahl weitergeleitet.

Für die beiden primären Aufgaben der Handlungsauswahl – das Auffinden und das Bewerten von Handlungen – existieren mit der Plangenerierung und der Handlungsbewertung jeweils noch eigene Komponentenrollen, deren Fähigkeiten die Handlungsauswahl in Anspruch nehmen kann. Dies dient der Modularität, wobei die Rolle der Handlungsauswahl lediglich den zur Umsetzung der deliberativen Verhaltenssteuerung unverzichtbaren Anteil übernimmt, während die beiden anderen Rollen domänen- und aufgabenspezifische Spezialisierungen erlauben.

Funktion

Die Handlungsauswahl stellt Intentionen für die von der Zielauswahl ausgewählten Ziele auf.

Definition 18: Handlungsauswahl

> Die Handlungsauswahl ist eine Abbildung AS: $F(t) \times O(t) \times G_A(t) \rightarrow I_A(t+1)$. Aus den vorhandenen Operatoren O(t) werden aufgrund der aktuellen Fakten F(t) für die neu verfolgten Ziele $G_A(t)$ neue Intentionen $I_A(t+1)$ erstellt.

Die neu erstellten Intentionen werden über Nachrichten vom Typ NewIntention an den Scheduler übergeben. Ist die Bearbeitung eines Ziels beendet – sei es weil keine Handlung gefunden wurde oder weil die Ausführung der erstellten Intention abgeschlossen ist – so wird der Zielauswahl das Ergebnis über eine entsprechende Result-Nachricht mitgeteilt.

Vermittels einer Nachricht vom Typ CreatePlan kann gegebenenfalls die Plangenerierung zum Erstellen eines Plans beauftragt werden. Um die Entscheidung

zwischen Handlungsalternativen der Handlungsbewertung zu überlassen, können dieser die möglichen Intentionen über eine EvaluateIntentions-Nachricht zur Auswahl übergeben werden.

Nachrichtentypen

Über Nachrichten vom Typ ExecuteGoal teilt die Zielauswahl die von ihr ausgewählten zu verfolgenden Ziele mit:

Name	ExecuteGoal
Typ	Auftrag
Beschreibung	initiiert die Handlungsauswahl und -ausführung für ein Ziel
Inhalt	ein Ziel g
Ergebnis	Result-Nachricht, true falls das Ziel erreicht wurde, false falls keine Handlung gefunden oder das Ziel nicht erreicht wurde

Rolle

```
[CASA.ActSelection,
 {CASA.ControlUnit},
 {[ExecuteGoal, {CASA.GoalSelection}],
  [InformIntention, {CASA.PlanGeneration, CASA.ActEvaluation}]},
 {}]
```

Die Rolle der Handlungsauswahl heißt CASA.ActSelection und gehört zur Rollengruppe der Kontrolleinheit. Für Nachrichten vom Typ ExecuteGoal erhält die Zielauswahl eine Sendeberechtigung. Bei InformIntention sind zusätzlich Plangenerierung und Handlungsbewertung sendeberechtigt, um ihre Ergebnisse mitzuteilen. Es gibt keine rollenspezifischen Eigenschaften.

4.5.6. Plangenerierung

Die Plangenerierung erzeugt aus den in der Planbibliothek enthaltenen Operatoren neue Pläne zum Erreichen eines Ziels. Aufträge zur Planung erhält sie ausschließlich von der Handlungsauswahl, der auch das Ergebnis übergeben wird.

Zur Generierung eines Plans werden abhängig von dem Ziel und dem aktuellen Weltzustand Operatoren aufgrund ihrer Bedingungen und Effekte so untereinander verkettet, daß die Ausführung des resultierenden Plans, der die Abhängigkeiten der verwendeten Operatoren untereinander beschreibt, den Zielzustand herbeiführt.

Da der Suchraum zur Generierung eines Plans i.a. sehr groß ist, werden effiziente Such- und Optimierungsstrategien benötigt, um in akzeptabler Zeit einen möglichst guten Plan zu finden. Für die Planoptimierung kann auf die Handlungsbewertung zurückgegriffen werden.

Wird ein Plan erzeugt, so kann es sinnvoll sein, den Plan zu verallgemeinern und in die Planbibliothek aufzunehmen, um bei ähnlichen Zielen auf den bereits vorhandenen Plan zurückzugreifen und eine erneute aufwendige Plangenerierung zu vermeiden.

Funktion

Die Plangenerierung erstellt Pläne zum Erreichen von Zielen. Sie ist eine Abbildung PG: F(t) x O(t) x $G_P(t) \rightarrow I_P(t+1)$, die der Abbildung AS der Handlungsauswahl entspricht, außer daß sie nur auf der Teilmenge $G_P(t)$ derjeniger Ziele operiert, die ihr von der Handlungsauswahl zur Planung übergeben wurden.

Nachrichtentypen

Den Auftrag zur Plangenerierung erteilt die Handlungsauswahl vermittels Nachrichten vom Typ CreatePlan:

Name	CreatePlan
Typ	Auftrag
Beschreibung	erstellt einen Plan für ein Ziel
Inhalt	ein Ziel g
Ergebnis	InformIntention-Nachricht mit dem erstellten Plan bzw. der Angabe, daß kein Plan erstellt werden konnte

Rolle

```
[CASA.PlanGeneration,
 {CASA.ControlUnit},
 {[CreatePlan, {CASA.ActSelection}]},
 {}]
```

Die Rolle der Plangenerierung heißt PlanGeneration und ist Teil der Rollengruppe der Kontrolleinheit. Eine Sendeberechtigung für Nachrichten vom Typ CreatePlan erhält nur die Handlungsauswahl. Spezifische Eigenschaften existieren nicht.

4.5.7. Handlungsbewertung

Die Handlungsbewertung dient der Bewertung von instantiierten Operatoren als Entscheidungsgrundlage zur Auswahl aus verschiedenen Handlungsalternativen sowohl bei der Handlungsauswahl, als auch bei der Plangenerierung. Eine eigene Komponente zur Handlungsbewertung ist vor allem deshalb sinnvoll, weil diese anders als die anderen Kontrollkomponenten sehr stark von Domänenwissen und individuellen Präferenzen abhängig sein kann. Dank der Modularität der CASA-Architektur können so für jeden Agenten nach Bedarf eigene Komponenten zur Handlungsbewertung verwendet werden.

Die Handlungsbewertung berechnet für die zu bewertenden Operatoren eine Präferenz, anhand derer diese untereinander verglichen werden können. Einige mögliche Kriterien sind:

- *Typ* des Operators: I.a. sind eigene Handlungen den Diensten anderer Agenten vorzuziehen, ebenso sind Handlungsprimitive meist zusammengesetzten Handlungen vorzuziehen.
- *Nebeneffekte*: Effekte, die nicht für das Ziel benötigt werden, können vermieden werden, sie können aber auch günstige Nebeneffekte für andere Ziele sein.
- *Kosten* der Ausführung: Dazu zählen zeitlicher Aufwand und Ressourcenbelegung / -verbrauch. Die Kosten lassen sich oft nur grob abschätzen.
- *Erfolgswahrscheinlichkeit* der Ausführung: Abhängig vom gegebenen Kontext können manche Handlungen bessere Erfolgsaussichten haben als andere. Zudem kann die Erfolgswahrscheinlichkeit anhand der Resultate von vorgenommenen Operatorausführungen statistisch abgeschätzt werden.

Weitere, insbesondere auch domänenspezifische Kriterien können zur Bewertung verwendet werden und auch die Gewichtung der einzelnen Kriterien kann domänenabhängig variieren.

Funktion

Die Handlungsbewertung wählt aus einer Menge von Intentionen die gemäß einer Evaluierungsfunktion value: Intention → Wert beste Intention aus.

Nachrichtentypen

Zur Bewertung von Intentionen verarbeitet die Handlungsbewertung Nachrichten vom Typ EvaluateIntentions:

Name	EvaluateIntentions
Typ	Auftrag
Beschreibung	wählt die beste Intention aus
Inhalt	eine Menge von Intention I
Ergebnis	InformIntention-Nachricht mit der besten Intention i mit: $\forall i' \in I: value(i') \leq value(i)$

Rolle

```
[CASA.ActEvaluation,
 {CASA.ControlUnit},
 {[EvaluateIntentions, {CASA.ActSelection, CASA.PlanGeneration}]},
 {}]
```

Die Rolle der Handlungsbewertung trägt die Bezeichnung CASA.ActEvaluation und ist Teil der Rollengruppe der Kontrolleinheit. Nachrichten vom Typ EvaluateIntenti-

ons werden von der Handlungsauswahl und der Plangenerierung als Absender akzeptiert. Eigenschaften sind keine definiert.

4.5.8. Scheduler

Der Scheduler koordiniert die durch die Handlungsauswahl und die Situationserkennung aufgestellten Intentionen und wählt aus diesen die als nächstes auszuführenden aus. Dazu berücksichtigt er wiederum die Reihenfolge des Aufstellens und die Priorisierung der Intentionen sowie Abhängigkeiten zwischen diesen. Enthält eine Intention einen Plan, so wird dieser in seine Einzelschritte aufgelöst.

Zur Koordinierung muß festgestellt werden, welche Intentionen unabhängig voneinander ausgeführt werden können und welche nur durch eine bestimmte Ausführungsreihenfolge miteinander vereinbar sind. Dazu wird die logische Beschreibung der Operatoren aufgrund ihrer Bedingungen und Effekte ausgenutzt, um Synergien und Konflikte zu entdecken und entsprechend zu behandeln. Synergien sind Handlungen mit überlappenden Effekten, die gegebenenfalls nur durch eine der Intentionen herbeigeführt zu werden brauchen. An Konflikten gibt es zunächst einander ausschließende Effekte sowie Effekte, die Vorbedingungen späterer Operatoren zerstören. Weiterhin dürfen keine Operatoren zeitgleich ausgeführt werden, wenn der Effekt des einen die Randbedingung des anderen verletzt. Durch eine geeignete Koordinierung kann bereits vor der Ausführung die Konsistenz der jeweils auszuführenden Operatoren gewährleistet und somit Konflikte während der Ausführung vermieden werden.

Falls für eine Intention Angaben bezüglich des Ausführungszeitpunkts wie eine vorgeschriebene feste, früheste oder späteste Anfangszeit oder Endzeit existiert, ist der Scheduler für deren Einhaltung durch eine fristgerechte Ausführung verantwortlich.

Funktion

Der Scheduler verwaltet die Intentionenstruktur und wählt aus dieser die auszuführenden Intentionen aus.

Definition 19: Scheduler

Der Scheduler ist eine Abbildung Sch: $I(t) \times I^{+}(t) \times I^{-}(t) \rightarrow I(t+1) \times I_A(t+1)$. Ausgehend von den bestehenden Intentionen I(t) und den hinzuzufügenden Intentionen $I^{+}(t)$ sowie den zu entfernenden $I^{-}(t)$ werden die neuen Intentionen I(t+1) und die als nächstes auszuführenden Intentionen $I_A(t+1)$ bestimmt.

Die hinzuzufügenden Intentionen sind diejenigen, die dem Scheduler über eine NewIntention-Nachricht mitgeteilt werden und sich konsistent in die Intentionenstruktur einfügen lassen. Zu entfernen ist eine Intention, falls sie zurückgenommen wird oder ihre Ausführung beendet ist. Mit dem Ende der Ausführung ist auch der zugehörige Auftrag für die Intention beendet, so daß das Ergebnis über eine Result-

Nachricht der beauftragenden Rolle mitgeteilt wird. Für die zur Ausführung ausgewählten Intentionen wird jeweils eine ExecuteIntention-Nachricht an die Handlungsausführung gesendet.

Nachrichtentypen

Eine neue Intention wird über eine Nachricht vom Typ NewIntention aufgestellt:

Name	NewIntention
Typ	Auftrag
Beschreibung	stellt eine neue Intention auf
Inhalt	eine Intention i
Annahme	Accept-Nachricht, true falls die Intention in die Intentionenstruktur eingefügt wurde, false sonst
Ergebnis	Result-Nachricht, true falls die Intention erfolgreich ausgeführt wurde, false falls die Ausführung fehlgeschlagen ist

Rolle

```
[CASA.Scheduler,
 {CASA.ControlUnit},
 {[NewIntention,
    {CASA.ActSelection, CASA.SituationAssessment, CASA.ActExecution}]},
 {}]
```

Der Name der Rolle des Schedulers, die zur Rollengruppe der Kontrolleinheit gehört, ist CASA.Scheduler. Sendeberechtigungen für Nachrichten vom Typ NewIntention werden der Handlungsauswahl, der Situationserkennung und der Handlungsausführung erteilt. Rollenspezifische Eigenschaften sind nicht definiert.

4.5.9. Handlungsausführung

Die Handlungsausführung kontrolliert und koordiniert die Ausführung der Intentionen, die der Scheduler zur Ausführung an diese übergeben hat.

Jede Intention enthält einen Operator zur Ausführung. Die Art der Ausführung hängt dabei vom Typ des Operators ab. Jeder Operator enthält neben seiner logischen Beschreibung einen Ausführungsteil, der durch die Handlungsausführung interpretiert wird. Abhängig von dessen Typ vollzieht die Handlungsausführung entweder selbst die Ausführung, oder bereitet diese vor und übergibt sie dann vermittels einer Nachricht an die zur Ausführung vorgesehene Komponente. Zusammengesetzte Operatoren wie Pläne oder Skripte werden gemäß den in ihnen enthaltenen Steuerungsanweisungen zerlegt und ihre Bestandteile entsprechend ausgeführt.

Vor der Ausführung eines Operators berücksichtigt die Handlungsausführung dynamische Änderungen des Weltzustands seit dem Erstellen der Intention. Sind

die Bedingungen nicht mehr erfüllt, so kann der Operator nicht erfolgreich ausgeführt werden und die Intention schlägt fehl. Ist der Effekt bereits eingetreten, so ist die Intention erreicht, auch ohne daß der Operator ausgeführt zu werden braucht.

Schließlich überwacht die Handlungsausführung noch den Erfolg der Ausführung. Dazu wartet sie auf deren Ende und die zugehörige Erfolgsmeldung, die gegebenenfalls auch das Resultat enthält. Zusätzlich kann sie das Ergebnis noch anhand des Effekts und des eingetretenen Zustands der Faktenbasis verifizieren bzw. anhand des Effekts die Faktenbasis aktualisieren. Anschließend gibt sie das Ergebnis an den Scheduler zurück bzw. verwendet es innerhalb von zusammengesetzten Operatoren zu deren Fortführung.

Operatortypen

Die Umsetzung der in Kapitel 4.4.4 erwähnten und in Kapitel 6.7.2 genauer spezifizierten Operatortypen durch die Handlungsausführung geschieht wie folgt:

- *Handlungsprimitiv*: Einem Handlungsprimitiv ist jeweils eine Anwendungskomponente zugeordnet, der die Ausführung übertragen wird.
- *abstrakte Handlung*: Da für eine abstrakte Handlung keine konkrete Umsetzung angegeben ist, muß ein neues Ziel aufgestellt werden, um den angegebenen Effekt zu erreichen.
- *Schlußfolgerung*: Aus der Erfülltheit der Vorbedingung wird direkt der Effekt als ebenfalls erfüllt abgeleitet, so daß die Faktenbasis entsprechend aktualisiert werden kann.
- *Skript*: Zur Ausführung eines Skripts interpretiert die Handlungsausführung deren Steueranweisungen und führt die enthaltenen Handlungen demgemäß aus. Eine Skriptsprache für die CASA-Standardarchitektur wird in Kapitel 6.7.3 definiert.
- *Plan*: Die Zerlegung von Plänen in einzelne Operatoren und deren Koordinierung mit den übrigen Intentionen ist Aufgabe des Schedulers. Dennoch kann die Handlungsausführung einen kompletten Plan zur Ausführung erhalten, beispielsweise innerhalb eines Skripts. In diesem Fall übergibt sie den Plan zur Zerlegung an den Scheduler.
- *Dienstoperator*: Mit der Ausführung eines Dienstoperators wird die Komponente zur Kommunikation beauftragt, die daraufhin eine entsprechende Dienstnutzung initiiert.
- *Protokolloperator*: Ein Protokolloperator wird gemäß dem Typ des enthaltenen Operators umgesetzt, wobei zusätzlich die Informationen bezüglich der Dienstnutzung, für die das Protokoll ausgeführt wird, verfügbar sind. Diese benötigen vor allem die Sprechaktoperatoren, damit kommunikative Handlungen einer bestimmten Dienstnutzung zugeordnet werden können.

- *Sprechaktoperator*: Da die Kommunikationskomponente verantwortlich für Interaktionen mit anderen Agenten ist, obliegt ihr auch die Umsetzung der Sprechaktoperatoren.

Die letzten Operatortypen betreffen die Realisierung des interaktiven Verhaltens eines Agenten und werden in den nachfolgenden Kapiteln 5.1.2 und 5.1.3 noch näher erläutert.

Funktion

Die Handlungsausführung interpretiert die in den auszuführenden Intentionen enthaltenen Operatoren und sorgt für deren Umsetzung. Die Ausführung geschieht entweder durch die Komponente selbst, oder sie wird an eine dazu befähigte Rolle vermittels einer ExecuteIntention-Nachricht delegiert. Zusammengesetzte Operatoren werden in ihre Einzelschritte zerlegt.

Definition 20: Handlungsausführung

Die Handlungsausführung ist eine Abbildung AE: $I_E(t) \times I_+(t) \times I_-(t) \rightarrow I_E(t+1)$. Die neue Menge der in Ausführung befindlicher Intentionen $I_E(t+1)$ ergibt sich aus der vorherigen Menge $I_E(t)$ sowie den neu auszuführenden Intentionen $I_+(t)$ und den abgeschlossenen $I_-(t)$.

Die neu auszuführenden Intentionen teilt der Scheduler der Handlungsausführung über Nachrichten vom Typ ExecuteIntention mit. Abgeschlossen ist eine Intention, wenn sie zurückgenommen wird oder ihre Ausführung beendet ist. Für eine beendete Ausführung wird deren Ergebnis dem Scheduler über eine Result-Nachricht mitgeteilt.

Nachrichtentypen

Der Auftrag zur Ausführung einer Intention wird über eine Nachricht vom Typ ExecuteIntention erteilt:

Name	ExecuteIntention
Typ	Auftrag
Beschreibung	initiiert die Ausführung einer Intention
Inhalt	eine Intention i
Annahme	Accept-Nachricht, true falls der Typ des Operators der Intention ausgeführt werden kann, false sonst
Ergebnis	Result-Nachricht, true falls die Intention erfolgreich ausgeführt wurde, false falls die Ausführung fehlgeschlagen ist

Rolle

```
[CASA.ActExecution,
 {CASA.ControlUnit},
 {[ExecuteIntention, {CASA.Scheduler}]},
 {}]
```

Die Rolle namens ActExecution beschreibt die Handlungsausführung und gehört zur Rollengruppe für die Kontrolleinheit. Nachrichten vom Typ ExecuteIntention werden vom Scheduler akzeptiert. Eigenschaften gibt es keine.

4.5.10. Kommunikation

Die Kommunikation ist zuständig für die Organisation von Konversationen mit anderen Agenten im Rahmen der Nutzung von Diensten sowie der damit verbundenen kommunikativen Handlungen in Form des Austauschs von Sprechakten. Weiterhin ist sie für die Einhaltung von Sicherheitsanforderungen zuständig, sofern diese für einen Dienst spezifiziert sind.

Im einzelnen führt die Kommunikationskomponente folgende Aufgaben durch:

- Dienstnutzungen durchführen
  - Dienstoperatoren ausführen
  - Dienstaufträge entgegennehmen
  - Protokollabläufe initiieren
- Kommunikationshandlungen durchführen
  - Sprechaktoperatoren umsetzen
  - Sprechakte empfangen
- Interaktionen absichern
  - Autorisierung für Dienstnutzungen überprüfen
  - Authenzität von Kommunikationspartnern überprüfen
  - Kommunikationsinhalte schützen

Die Ausführung eines Dienstoperators durch den Dienstnutzer führt zu einem Dienstauftrag, den der Anbieter über einen Sprechakt erhält. Den Rahmen jeder Dienstnutzung bildet dabei ein einheitliches Protokoll, das den gemeinsamen Anteil aller Dienstinteraktionen regelt. Innerhalb dieses Metaprotokolls werden über Ziele dienstspezifische Protokolle initiiert, die Operatoren zum Versenden und Empfangen von Sprechakten für die jeweilige Dienstnutzung enthalten können. Die Funktionalitäten und Abläufe zur Interaktion zwischen Agenten basierend auf Dienstnutzungen und die zugehörigen Aufgaben der Kommunikationskomponente werden im nachfolgenden Kapitel 5.1 noch eingehender beschrieben.

Zur Einhaltung von Sicherheitsanforderungen greift die Kommunikation auf die Funktionalität entsprechender Sicherheitskomponenten (vgl. Kapitel 4.6.3) zurück,

sofern diese vorhanden sind. Die Autorisierung legt fest, welche Agenten einen bestimmten angebotenen Dienst nutzen dürfen. Vermittels der Authentifikation wird die Identität eines Kommunikationspartners überprüft. Schließlich sind noch die Inhalte der Kommunikation während des Transports vor unbefugter Einsichtnahme und Veränderung zu schützen.

Funktion

Die Kommunikationskomponente führt Intentionen mit interaktiven Operatoren aus. Dazu kann sie Operatoren zur Initiierung von Dienstnutzungen sowie zum Empfang und zum Versenden von Sprechakten verarbeiten. Außerdem nimmt sie Anfragen für angebotene Dienste des Agenten entgegen.

Nachrichtentypen

Zum Empfang von Sprechakten dienen Nachrichten des Typs ReceivedLetter:

Name	ReceivedLetter
Typ	Benachrichtigung
Beschreibung	teilt den Empfang eines Sprechakts mit
Inhalt	ein Brief mit einem Sprechakt als Inhalt

Da Fehler in Bezug auf empfangene Sprechakte nicht durch die Transportkomponenten selbst, sondern nur auf Sprechaktebene durch die Kommunikation behandelt werden, ist ReceivedLetter kein Auftrag, nach dessen Verarbeitung das Ergebnis mitgeteilt wird, sondern eine reine Benachrichtigung.

Außerdem verarbeitet die Kommunikationskomponente Nachrichten vom Typ ExecuteIntention analog zur Handlungsausführung. Dabei werden nur Intentionen akzeptiert, die Operatoren zur Durchführung einer Dienstnutzung oder zum Senden und Empfangen von Sprechakten enthalten.

Rolle

```
[CASA.Communication,
 {CASA.ControlUnit},
 {[ExecuteIntention, {CASA.ActExecution}],
  [ReceivedLetter, {CASA.Transport}]},
 {}]
```

Die Rolle der Kommunikation heißt CASA.Communication und gehört zur Rollengruppe für die Kontrolleinheit. Sie erteilt eine Sendeberechtigung für Nachrichten vom Typ ExecuteIntention an die Handlungsauswahl und eine für Nachrichten vom Typ ReceivedLetter an die Rollengruppe der Transportkomponenten. Eigenschaften sind nicht definiert.

4.6. Peripherie

Die Komponentenrollen der Peripherie spezifizieren Agentenfunktionalitäten außerhalb der Ebene der wissensbasierten Verhaltenssteuerung. Dazu gehören zum einen anwendungsspezifische Funktionen inklusive der Interaktion mit der Umgebung sowie der Transportanteil der Kommunikation mit anderen Agenten. Zum anderen beinhalten sie eine Kontrolle des Agenten auf einer Meta-Ebene sowie die Absicherung des Agenten und seiner Interaktionen nach außen hin.

Jeder dieser vier Aufgabenbereiche kann innerhalb eines Agenten durch mehrere verschiedene Rollen umgesetzt werden, die jeweils eine übereinstimmende Grundfunktionalität besitzen. Deshalb werden sie im folgenden über Rollengruppen definiert.

4.6.1. Anwendung

Anwendungskomponenten übernehmen den anwendungsspezifischen operationalen Anteil eines CASA-Agenten. Ihre Aufgabe ist dabei das Bereitstellen einer Schnittstelle zwischen dem wissensbasierten Teil des Agenten und seiner Umgebung sowie die Umsetzung der Ausführung von Handlungsprimitiven durch Programmkode.

Die Umgebung des Agenten macht dabei alles das aus, mit dem für den Agenten eine Interaktion zusätzlich zur Agentenkommunikation möglich ist. Dies sind typischerweise andere Programme oder Datenbanken sowie menschliche Benutzer oder auch über Sensoren und Aktoren erfahrbare Teile der physischen Welt. In jedem Fall ist eine Interaktion in beide Richtungen möglich. Der Agent kann auf Einwirkungen von außen durch Änderungen seines Faktenwissens oder über das Aufstellen von Zielen reagieren. Umgekehrt kann im Rahmen der Ausführung von Handlungsprimitiven auf die Umgebung eingewirkt werden. Besteht die Umgebung in anderer Software, so kann der Agent deren Funktionalität über eine Softwareschnittstelle entweder selbst nutzen oder der Agentengesellschaft in Form von Diensten zugänglich machen. Für die Interaktion mit einem menschlichen Benutzer werden vor allem graphische Oberflächen verwendet, andere Benutzerschnittstellen sind prinzipiell aber auch möglich.

Zur Ausführung von Handlungsprimitiven verarbeitet eine Anwendungskomponente die entsprechenden Nachrichten, die die Handlungsausführung ihr als Auftrag zusendet. Diese enthalten als Intention das auszuführende Planelement und dessen Instantiierung, wodurch die auszuführende Handlung bestimmt ist. Die Umsetzung der Ausführung obliegt der Anwendungskomponente, die deren Ergebnis schließlich wieder der Handlungsauswahl mitteilt.

Funktion

Anwendungskomponenten realisieren die Umsetzung von Intentionen mit Operatoren für Handlungsprimitive. Außerdem können sie wie jede andere Komponente das Verhalten des Agenten beeinflussen, indem sie Fakten, Ziele und Regeln aufstellen.

Nachrichtentypen

Die Aufträge zur Ausführung eines Handlungsprimitivs erhält eine Anwendungskomponente über eine Nachricht vom Typ ExecuteIntention von der Handlungsausführung. Sie akzeptiert sie nur, wenn die enthaltene Intention einen Operator betrifft, den sie umsetzen kann. Nach Ende der Ausführung sendet sie deren Ergebnis in einer Result-Nachricht an die Handlungsausführung.

Rollengruppe

```
[CASA.Application,
 {CASA.Main},
 {[ExecuteIntention, {CASA.ActExecution}]},
 {}]
```

Die Rollengruppe für Anwendungskomponenten wird als CASA.Application bezeichnet. Sie erteilt eine Sendeberechtigung für Nachrichten vom Typ ExecuteIntention an die Handlungsausführung. Gemeinsame Eigenschaften sind nicht definiert.

4.6.2. Transport

Die Komponenten für den Transport übernehmen den physikalischen Anteil der Kommunikation mit anderen Agenten. Dazu können sie zum einen jederzeit Mitteilungen anderer Agenten empfangen, zum anderen selbst Mitteilungen an die korrespondierende Transportkomponente anderer Agenten zustellen. Jede Transportkomponente legt dabei genau eine Adresse des Agenten fest, über die er für andere Agenten erreichbar ist und die deshalb innerhalb des Agenten wie auch der Agentengesellschaft bekannt gemacht und bei Bedarf aktualisiert werden muß.

Eine Transportkomponente muß – sofern sie aktiv und funktionsfähig ist – permanent bereit sein, an ihre Adresse gerichtete Mitteilungen von anderen Agenten zu empfangen, auch wenn diese zeitlich überlappend eintreffen[33]. Dazu brauchen nicht unbedingt mehrere Mitteilungen parallel empfangen werden zu können, sondern ein sequentieller Empfang genügt, sofern das Transportprotokoll ein Warten auf

[33] Im Falle einer Überlastung durch eine Überschwemmung mit Mitteilungen kann allerdings die Funktionsfähigkeit durch Ressourcenknappheit eingeschränkt werden, so daß ein implizites oder auch explizites Limit der Empfangskapazität bestehen kann.

erneute Empfangsbereitschaft unterstützt. Die eingegangenen Mitteilungen werden gegebenenfalls aus ihrer Transportform zurückverwandelt und soweit nötig auf ihre Korrektheit überprüft, daraufhin werden sie zur weiteren Verarbeitung an die Kommunikationskomponente weitergegeben.

Von dieser erhält eine Transportkomponente auch die zu versendenden Mitteilungen, die den Agentennamen und eine Adresse enthalten, über die der Kommunikationspartner erreichbar ist. Diese Adresse besteht aus einem Transportprotokoll, das die betreffende Transportkomponente unterstützt, sowie aus einer protokollspezifischen Adressierung des Kommunikationspartners. Die Mitteilung wird nun gegebenenfalls noch in eine transportierbare Form gebracht und übermittelt. Der Erfolg oder Fehlschlag der Zustellung einer Mitteilung wird der Kommunikationskomponente mitgeteilt. Dabei bedeutet eine erfolgreiche Zustellung allerdings nicht, daß der Empfänger die Mitteilung auch beachtet sowie korrekt verarbeitet und gegebenenfalls beantwortet.

Funktion

Transportkomponenten empfangen und versenden Sprechakte, die zum Transport in einem Brief mit den nötigen Zustellungsparametern enthalten sind. Dazu besitzen sie jeweils eine eindeutige Kommunikationsadresse (URL), über die sie Sprechakte empfangen und versenden können.

Zu versendende Sprechakte erhält eine Transportkomponente von der Kommunikation über eine SendLetter-Nachricht. Empfangene Nachrichten werden dieser über Nachrichten vom Typ ReceivedLetter mitgeteilt.

Nachrichtentypen

Zum Versenden eines Sprechakts wird dieser vermittels einer Nachricht vom Typ SendLetter einer Transportkomponente übergeben:

Name	SendLetter
Typ	Auftrag
Beschreibung	versendet einen Sprechakt
Inhalt	ein Brief mit einem Sprechakt als Inhalt

Rollengruppe

```
[CASA.Transport,
 {CASA.Main},
 {[SendLetter, {CASA.Communication}]},
 {[Address, Address, Address, NULL, r]}]
```

Sämtliche Transportkomponenten gehören der Rollengruppe namens CASA.Transport an. Diese erteilt der Kommunikationskomponente eine Sendebe-

rechtigung für Nachrichten vom Typ SendLetter. Die Eigenschaft Address beinhaltet die Kommunikationsadresse der Transportkomponente.

4.6.3. Sicherheit

Die Sicherheitskomponenten dienen der Absicherung des Agenten gegenüber unbefugten Eingriffen von außen, insbesondere bezüglich der Interaktion mit anderen Agenten, da vor allem hier Möglichkeiten einer Einflußnahme bestehen.

Die Kommunikation benötigt zur Einhaltung der möglichen Sicherheitsanforderungen von Diensten zum einen eine Komponente zur Überprüfung von Dienstnutzungsberechtigungen zur Beurteilung von Dienstanfragen, zum anderen Komponenten zur Absicherung der einzelnen kommunikativen Handlungen. Letztere können einerseits Transportkomponenten sein, die einen sicheren Transport gewährleisten, und andererseits eine eigene Komponente zur Kommunikationsabsicherung, die beispielsweise den Inhalt von Sprechakten verschlüsseln bzw. nach dem Empfang entschlüsseln und die Authenzität von Kommunikationspartnern verifizieren kann.

Funktion

Sicherheitskomponenten sichern Interaktionen zwischen Agenten ab. Dies betrifft Dienstnutzungen als ganzes sowie einzelne Sprechakte.

Nachrichtentypen

Es gibt keine gemeinsamen vordefinierten Nachrichtentypen für Sicherheitskomponenten, da diese verschiedenartige Aufgaben erfüllen können.

Rollengruppe

```
[CASA.Security,
 {CASA.Main},
 {},
 {}]
```

Die Rollengruppe für Sicherheitskomponenten heißt CASA.Security. Sie besitzt keine gemeinsamen Sendeberechtigungen oder Eigenschaften für alle zugehörigen Rollen.

4.6.4. Management

Managementkomponenten ermöglichen eine weitreichende Kontrolle über den Agenten. Sie überwachen vermittels Introspektion den Zustand des Agenten und können diesen über Manipulation verändern. Da diese Komponenten einen direkten Zugriff auf den Agentenkern erhalten, können sie den gesamten Agenten und alle seine Komponenten im selben Umfang wie der Agentenkern verwalten.

Im Agentenkern und der allen Komponenten gemeinsamen Basisschnittstelle sind dabei die Möglichkeiten zur Introspektion und Manipulation angelegt, die dann von Managementkomponenten voll genutzt werden können. Zudem können die Monitor-Nachrichten der Wissensbasiskomponenten für eine Introspektion auf Wissensebene verwendet werden. Wichtige Managementaspekte sind zum Beispiel die Konfiguration des Agenten und seiner Komponente sowie eine Unterstützung bei der Fehlererkennung und -behebung.

Funktion

Managementkomponenten verwalten einen Agenten und seine Komponenten. Dazu besitzen sie Zugriff auf dessen Schnittstelle nach außen (vgl. Kapitel 3.4.4), die als besondere Eigenschaft durch den Agentenkern gesetzt wird.

Nachrichtentypen

Managementkomponenten besitzen keine einheitliche Funktionalität, für die Nachrichtentypen vordefiniert werden könnten. Zudem besitzen sie über den Agentenkern direkten Zugriff auf alle Komponenten.

Rollengruppe

```
[CASA.Management,
 {CASA.Main},
 {},
 {[Agent, AgentInterface, AgentInterface, NULL, ri]}]
```

Die Rollengruppe CASA.Management umfaßt alle Managementkomponenten. Sie erteilt keine Sendeberechtigungen. Als Eigenschaft besitzt sie Agent, über die der Zugriff auf die Agentenschnittstelle ermöglicht wird.

4.7. Zusammenfassung

Die Standardarchitektur von CASA definiert eine Kontrollarchitektur zur Steuerung des Verhaltens einzelner Agenten auf der Grundlage eines allgemeinen wissensbasierten Kontrollschemas, das reaktives, deliberatives und interaktives Verhalten integriert. Die explizite Repräsentation von symbolischem Wissen bietet dabei einen einheitlichen und ausdrucksstarken Formalismus, der zur Designzeit wie zur Laufzeit von Vorteil ist. Beim Erstellen eines Agenten erlaubt sie die Spezifikation von dessen Verhalten auf einer abstrakten Beschreibungsebene, auf der dieser dann auch zur Laufzeit operiert. Die syntaktische Verarbeitung des Wissens unter Wahrung von dessen formallogischer Semantik ermöglicht dem Agenten eine flexible und zugleich kohärente Steuerung seines autonomen Verhaltens. Und schließlich

stellt sie Agenten auch die benötigte gemeinsame Syntax und Semantik zur Verfügung für den Austausch von Informationen bei deren Interaktionen.

Das Kontrollschema der Standardarchitektur trennt strikt die Aspekte der Repräsentation und der Verarbeitung von Wissen. Bei der Repräsentation werden verschiedene Wissenstypen anhand ihrer funktionalen Rolle unterschieden. Fakten beinhalten Wissen über den gegenwärtigen Zustand der Umgebung des Agenten. Ziele repräsentieren angestrebte Weltzustände und Intentionen beabsichtigte Handlungen zum Erreichen dieser Zustände. Die Handlungsmöglichkeiten des Agenten werden über Operatoren beschrieben, und Regeln geben Reaktionen auf eingetretene Situationen an. Alle diese Formen von Wissen verwenden dabei eine gemeinsame Terminologie, die über Ontologien definiert wird.

Auf diesem Wissen operieren nun entsprechende Kontrollfunktionen, die die eigentliche Verhaltenssteuerung ausmachen. Zum einen hält der Agent sein Wissen aktuell, indem er seinen Faktenbestand an Änderungen der Umgebung anpaßt und gegebenenfalls auch seine Ziele ändert. Auf neu eingetretene Situationen wird dabei gemäß den Reaktionsregeln reagiert. Die Ziele steuern das proaktive Verhalten, wozu geeignete Handlungen als Intentionen aufgestellt werden, die dann koordiniert und schließlich ausgeführt werden. Beinhaltet eine Handlung eine Dienstnutzung, so besteht deren Umsetzung in der Interaktion mit einem anderen Agenten. Somit ergibt sich das Verhalten eines Agenten aus dem beständigen Wechselspiel zwischen der Überführung des momentanen Wissens in Handlungen und Interaktionen und der Anpassung des Wissens an die sich verändernden Umstände inklusive der Resultate dieser Handlungen.

Die Standardarchitektur überträgt dieses Schema auf eine Kontrollstruktur unter Nutzung der im vorangegangenen Kapitel definierten Komponentenarchitektur. Aufgrund von deren Offenheit und Skalierbarkeit sind dabei auch Erweiterungen oder Beschränkungen auf bestimmte Funktionalitäten möglich. Die Spezifikation der Kontrollstruktur geschieht über eine Menge von Komponentenrollen und Nachrichtentypen, die die Funktionalität der einzelnen Komponenten und deren Abhängigkeiten und Interaktionsmöglichkeiten untereinander festlegen.

Die Kontrollarchitektur gliedert sich in sieben Arten von Komponenten, die jeweils zu einer Rollengruppe zusammengefaßt sind. Zunächst gibt es die Agentenhülle, die die Komponentenarchitektur realisiert. Die wissensbasierte Verhaltenssteuerung zerfällt in die Wissensbasis zur Speicherung des Wissens und die Kontrolleinheit zur Verwendung des Wissens. Innerhalb der Wissensbasis gibt es jeweils eine Rolle für die obengenannten Wissenstypen, die hierfür den Zugriff, die Änderung und die Überwachung des Wissens ermöglicht. Bei der Kontrolleinheit übernimmt ebenfalls jeweils eine Rolle eine bestimmte Kontrollfunktion zur Verwaltung und Nutzung des in der Wissensbasis enthaltenen Wissens.

Die übrigen vier Rollengruppen sind der Peripherie zugeordnet und dienen als Schnittstelle zu nicht-wissensbasierten Funktionalitäten. Anwendungskomponenten realisieren den nicht-generischen Anteil der spezifischen Aufgaben eines Agenten

inklusive der Schnittstelle zur Umgebung des Agenten. Der Transport von Sprechakten zwischen Agenten wird von Transportkomponenten übernommen. Aufgabe der Sicherheitskomponenten ist die Absicherung des Agenten und seiner Kommunikation. Schließlich erlauben Managementkomponenten die Überwachung und Kontrolle des Agenten und seiner Komponenten zur Laufzeit.

Die Standardarchitektur definiert somit eine abstrakte Kontrollstruktur zur wissensbasierten Verhaltenssteuerung aufbauend auf der Komponentenarchitektur von CASA. Sie erlaubt eine flexible Umsetzung des auf einer formalen Wissensebene spezifizierbaren Verhaltens eines Agenten.

5. Die Agentengesellschaft

Agenten mit der Möglichkeit, untereinander zu interagieren, bilden zusammen eine Agentengesellschaft. Die Interaktionen bestehen im Austausch von Informationen und Leistungen zwischen den Agenten. Ermöglicht werden sie zum einen durch Mechanismen, die die Interaktionen nach einem einheitlichen Schema regeln, und zum anderen durch eine Infrastruktur, die die Agenten einer Agentengesellschaft miteinander verbindet. Anders gesagt, die Agenten benötigen die Fähigkeit und die Gelegenheit zur Interaktion. Nachfolgend wird beschrieben, wie diesen beiden Aspekten einer Agentengesellschaft in der CASA-Architektur Rechnung getragen wird.

5.1. Interaktionen

Um verläßliche Interaktionen gewährleisten zu können, muß deren Verlauf und Inhalt hinreichend einheitlich strukturiert sein, damit die an einer Interaktion beteiligten Agenten die Aktionen der anderen richtig interpretieren und verarbeiten können. Andererseits muß genügend Offenheit gewahrt bleiben, um dabei den Raum an möglichen Interaktionen nicht allzusehr einzuschränken. Wie bereits in Kapitel 2.4.1 geschildert, wird eine flexible Interoperabilität in Agentensystemen üblicherweise durch die Verwendung von Standards erreicht, die entweder ein festes Format vorschreiben oder ein Schema zur Definition domänenspezifischer Standards bereitstellen.

Gemäß dem Paradigma der Interaktion durch Kommunikation betrifft die Standardisierung die folgenden Ebenen (vgl. Kapitel 2.4.2 und 2.4.3):

- *Vokabular*: Agenten benötigen ein gemeinsames Vokabular, mit dem sie die zu kommunizierenden Inhalte ausdrücken können. Dazu stellen Ontologiebeschreibungssprachen Schemata zur Formulierung von domänenspezifischen Terminologien bereit. Diese werden entweder gemeinsam genutzt oder müssen gegebenenfalls ineinander übersetzt werden.

- *Inhaltssprache*: Unter Verwendung des in Ontologien definierten Vokabulars werden die Inhalte der Kommunikation in einer formal-logischen Sprache mit einer vorgegebenen Syntax und Semantik ausgedrückt. Derartige Sprachen sind weitgehend domänenunabhängig, weshalb bei der Wahl einer Sprache vor allem ein Kompromiß zwischen der Ausdrucksstärke und effizienten Formalismen zur Verarbeitung gefunden werden muß.
- *Kommunikationssprache*: Eine Kommunikationssprache ermöglicht die Durchführung von kommunikativen Handlungen im Sinne der Sprechakttheorie mit Hilfe einer vorgegebenen Syntax und Semantik.
- *Interaktionsverläufe*: Mehrere zusammengehörige kommunikative Handlungen bilden eine Konversation. Die Menge der möglichen Interaktionsverläufe läßt sich dabei durch Konversationsprotokolle einschränken. Ein derartiges Protokoll repräsentiert die Interaktionspartner abstrakt als Rollen und legt fest, welche kommunikativen Handlungen ein eine Rolle einnehmender Agent abhängig vom bisherigen Konversationsverlauf ausführen darf.

Diese Standardisierungen stellen ein hohes Maß an Interoperabilität sicher, indem das Erstellen und Verarbeiten von kommunikativen Handlungen durch alle Kommunikationspartner nach einer gemeinsamen Syntax und Semantik geschieht. Derartige Formalismen nutzt auch die CASA-Architektur, um zugleich verläßliche und flexible Interaktionsmechanismen bereitzustellen.

In CASA werden zusätzlich sämtliche Interaktionen einem einheitlichen Schema unterworfen, indem sie als Teile von Dienstnutzungen aufgefaßt werden. Das Dienst-Konzept bildet eine weitere, übergeordnete Ebene der Standardisierung von Interaktionen, indem es einen einheitlichen Rahmen zur Beschreibung und Durchführung von Interaktionen bereitstellt. Auf diese Weise lassen sich Interaktionen besser in die übrigen Kontrollmechanismen integrieren und durch vorgegebene Architekturmechanismen unterstützen.

Da zur Realisierung der Agentenkommunikation in CASA auf etablierte Methoden zurückgegriffen wird, wird auf diese im folgenden nur kurz eingegangen. Es folgt eine detaillierte Vorstellung des Dienst-Konzepts sowie dessen Umsetzung im Rahmen der Standardarchitektur.

5.1.1. Kommunikation

Die Agentenkommunikation in CASA orientiert sich an der Spezifikation der FIPA-ACL [FIPA 2000: PC00037F, XC00061D]. Diese schreibt Syntax und Semantik für eine Agentenkommunikationssprache vor, die offen ist bezüglich der Verwendung unterschiedlicher Inhaltssprachen und Ontologien. Die weiter unten beschriebenen Architekturmechanismen zur Umsetzung von Interaktionen in CASA verwenden für den Inhalt der Kommunikation dieselbe Sprache und dieselben Ontologien wie zur internen Wissensrepräsentation, um zusätzliche Übersetzungen oder Interpretatio-

nen zu vermeiden, die andernfalls durch eigene Komponenten realisiert werden müssen. In Kapitel 6 werden Sprachen zur Formulierung von Wissen und Ontologien spezifiziert sowie die genaue Umsetzung der FIPA-ACL in CASA inklusive der verwendeten Sprechakttypen vorgestellt.

Gemäß der Syntax der FIPA-ACL besteht ein Sprechakt aus einem Typ und einer Menge von Parametern (vgl. Kapitel 2.4.2 und 6.8). Zu den Parametern gehören i.a. der Absender und der Empfänger zur Adressierung, die Zuordnung zu einem Protokoll und einer Konversation, sowie der eigentliche Inhalt inklusive der Sprache und der Ontologien, mit denen dieser formuliert wird. Die Semantik wird relativ zu den Sprechakttypen definiert und legt die Bedeutung der vermittels eines Sprechakts ausgeführten Handlung fest.

Die Semantik der FIPA-ACL beschreibt für einen Sprechakttyp die Voraussetzungen beim Sender und den zu erwartenden Effekt beim Empfänger über eine Modallogik, mit der sich sicheres und unsicheres Wissen sowie Absichten ausdrükken lassen. Diese Art der formalen Semantik birgt jedoch einige Schwierigkeiten. Zum einen läßt es sich nicht nachprüfen, ob das Kommunikationsverhalten eines Agenten der Semantik entspricht, weil die verwendete Modallogik nicht entscheidbar ist und die Zuschreibung der verwendeten Modalitäten unterdeterminiert ist [Wooldridge 1998]. Weiterhin setzt sie Wissen und Absichten eines Agenten in Bezug auf Wissen und Absichten anderer Agenten voraus, was nicht zwangsläufig für beliebige kommunizierende Agenten angenommenen werden kann[34]. Zudem benötigt man hierfür entsprechend ausdrucksstarke Wissensrepräsentationsformalismen, die ihrerseits aufwendige Wissensverarbeitungsmechanismen nach sich ziehen. Schließlich ist die Semantik auch zu rigide, da sie keine Kontextabhängigkeiten zur jeweiligen Konversation berücksichtigt.

Eine strikte Einhaltung dieser Semantik ist also weder in jedem Fall möglich, noch unter allen Umständen wünschenswert. Mehr Spielraum bietet die Verwendung von Konversationsprotokollen, die typische Interaktionsmuster beschreiben. Einerseits kann die Bedeutung jedes Sprechakts eines Protokolls gesondert und abhängig vom jeweiligen Kontext definiert werden. Andererseits kann die Semantik auf die Funktion des Sprechakts innerhalb des Protokolls beschränkt werden. Die CASA-Architektur setzt die Semantik der Sprechakttypen der FIPA-ACL deshalb nur insoweit um, als sie in Protokollen entsprechend verwendet werden.

5.1.2. Dienste

Das Konzept eines Dienstes in CASA erlaubt die Beschreibung von Interaktionen in einer Weise, die eine flexible und automatisierte Verarbeitung durch die Agenten ermöglicht. Ein Dienst wird dabei als eine Handlung aufgefaßt, die ein Agent, der

[34] Die in Kapitel 6 vorgeschlagene Wissensrepräsentationssprache verzichtet aus Gründen der Pragmatik bewußt auf die Repräsentation des Wissens anderer Agenten.

Anbieter, für einen anderen, den Nutzer, ausführt. Da ein Dienst eine Handlung ist, kann er seitens des Dienstnutzers analog zu anderen Handlungen behandelt werden, außer daß die Ausführung an einen anderen Agenten delegiert wird.

Erreicht wird dies zum einen durch die Repräsentation von Diensten als Operatoren, zum anderen durch die Verwendung eines generischen Rahmenprotokolls zur Durchführung von Dienstnutzungen. Da Interaktionen über Dienstoperatoren beschrieben werden, können sie wie andere Handlungen zur Verhaltenssteuerung eingesetzt werden und somit dynamisch ausgewählt und in Handlungssequenzen integriert werden. Die Durchführung der Interaktionen geschieht nach dem durch das Dienst-Metaprotokoll vorgegebenen einheitlichen Schema, das die allen Dienstnutzungen gemeinsamen Abläufe regelt und gleichzeitig offen ist für zusätzliche dienstspezifische Kommunikation.

Im folgenden werden Dienstoperatoren und das Dienst-Metaprotokoll auf einer konzeptuellen Ebene vorgestellt, während der nachfolgende Abschnitt deren Integration in die Standardarchitektur beschreibt.

Das Dienst-Metaprotokoll

Um die allen Dienstnutzungen gemeinsamen Parameter und Abläufe fest in die Kontrollarchitektur integrieren zu können, werden diese durch ein einheitliches Protokoll geregelt, das Dienst-Metaprotokoll. Es wird als Metaprotokoll bezeichnet, da es ein allgemeines Protokoll zur Durchführung von Konversationen ist, in das wiederum spezifische Protokolle eingebettet werden.

Da jedes Konversationsprotokoll in CASA der Durchführung von Dienstnutzungen dient, besitzt es jeweils genau zwei Rollen, die des Anbieters und die des Nutzers des Dienstes[35]. Die Rolle des Nutzers übernimmt dabei immer genau ein Agent, während als Anbieter u.U. auch mehrere Agenten möglich sind.

Das Dienst-Metaprotokoll (Abbildung 19) besteht aus den folgenden Phasen:

1. *Initiierung*: Nach der Dienstanfrage werden zunächst Parameter ausgehandelt, die allen Diensten gemeinsam sind. Diese Phase endet, indem der Anbieter seine Bereitschaft zur Diensterbringung zusichert oder verweigert.
2. *Auswahl*: Anschließend finden Verhandlungen über dienstspezifische Parameter statt. Zum Ende dieser Phase entscheidet nunmehr der Nutzer, welcher der Anbieter den Dienst erbringen soll.
3. *Diensterbringung*: Die nun folgende Diensterbringung endet, indem der Anbieter deren Ergebnis übermittelt.

---

[35] Interaktionsmuster zwischen mehr als zwei Rollen lassen sich prinzipiell auf eine Menge von Zweierinteraktionen zurückführen und können somit immer als Dienstnutzungen mit den Rollen des Anbieters und des Nutzers modelliert werden.

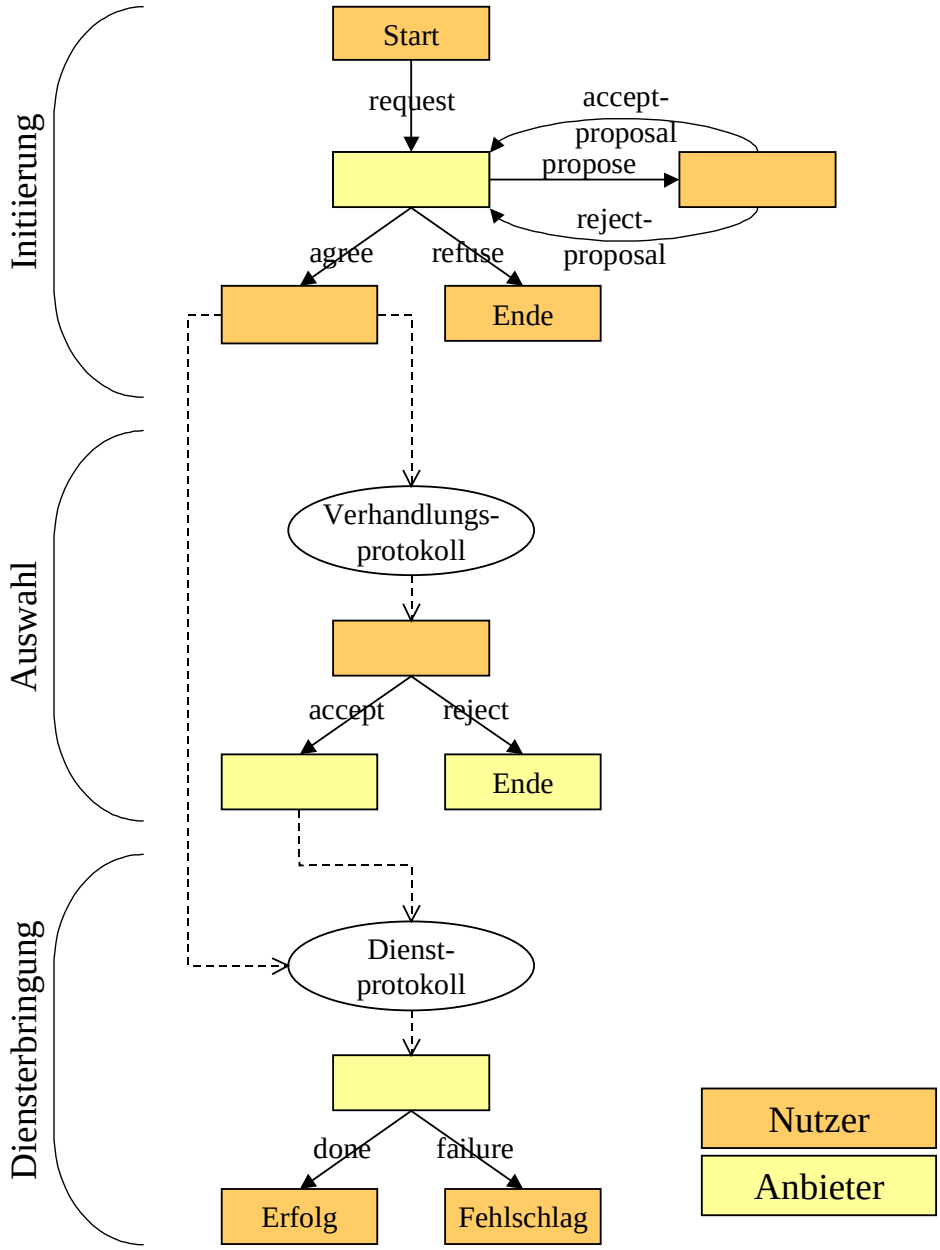

Abbildung 19: Graph-Darstellung des Dienst-Metaprotokolls

Die Initiierungsphase dient dazu festzustellen, ob eine Dienstnutzung prinzipiell möglich ist und ein Anbieter zur Diensterbringung bereit ist, indem über allgemeine Dienstparameter verhandelt wird. Mit der Auswahlphase wird aufgrund der Aushandlung dienstspezifischer Parameter derjenige Anbieter ausgewählt, der die Dienstleistung tatsächlich erbringen soll, was dann in der abschließenden Diensterbringungsphase geschieht.

Zusätzlich kann diesen drei Phasen eine optionale Sicherheitsaushandlung mit einem eigenen Konversationsprotokoll vorangestellt sein, in der gegebenenfalls Parameter zur Absicherung der Dienstnutzung ausgehandelt werden. Diese Sicherheitsparameter werden dann von sämtlichen Kommunikationshandlungen im Rahmen des Metaprotokolls berücksichtigt.

Schließlich können beide Rollen zu jeder Zeit des Metaprotokolls auch Sprechakte zur Fehlerbehandlung verschicken. Zum einen muß ein empfangener Sprechakt, der nicht verarbeitet werden kann, mit einem Sprechakt vom Typ `not-understood` beantwortet werden. Zum anderen kann ein Agent eine Konversation vorzeitig beenden (Sprechakttyp `cancel`), beispielsweise weil er selbst beendet wird oder weil ein Zeitlimit zum Warten auf Antworten überschritten wird. Prinzipiell kann auch eine Terminierung erfolgen, indem ein Konversationspartner gar nicht oder nicht mehr antwortet, sei es absichtlich oder wegen eines Defektes.

Der Protokollverlauf

Initiiert wird das Dienst-Metaprotokoll immer von der Rolle des Nutzers. Dazu wird eine Dienstanfrage (`request`) an einen oder mehrere Anbieter des betreffenden Dienstes gesendet. Diese Anfrage enthält bereits die initialen generischen Parameter, die der Nutzer auf diese Weise vorschlägt. Der Anbieter kann nun sukzessive andere Vorschläge unterbreiten (`propose`), die der Nutzer jeweils annimmt (`accept-proposal`) oder ablehnt (`reject-proposal`). Eine Ablehnung muß einen Gegenvorschlag enthalten, der als akzeptiert gilt, falls für denselben Parameter kein erneuter Vorschlag des Anbieters gemacht wird. Ohne Gegenvorschlag bedeutet eine Ablehnung, daß seitens des Nutzers keine Verhandlungsoptionen für diesen Parameter mehr bestehen. Auf diese Weise kann nacheinander über die einzelnen generischen Parameter verhandelt werden. Wird dabei keine Übereinkunft gefunden oder lehnt der Anbieter die Anfrage aus einem anderen Grund ab, so beendet er das Protokoll mit einem Sprechakt vom Typ `refuse`. Andernfalls nimmt er die Anfrage an (`agree`) und erklärt so seine Bereitschaft zur Erbringung des Dienstes.

Für die nun folgende Aushandlung der dienstspezifischen Parameter wird ein eingebettetes Protokoll verwendet, um offen zu sein für verschiedene Arten von Parametern und Aushandlungsverfahren. Zum Abschluß erteilt der Nutzer einem der Anbieter den Dienstauftrag (`accept`), alle anderen erhalten eine Absage (`reject`). Prinzipiell kann der Nutzer auch sämtliche Anbieter ablehnen und das Protokoll beenden. Außerdem kann er mit den Absagen warten, bis der ausgewählte Anbieter den Dienst erfolgreich erbracht hat, damit er im Fehlerfall noch einen anderen Anbieter beauftragen kann. Schließlich kann auf die Auswahlphase auch ganz verzichtet werden, falls nur an einen Anbieter eine Anfrage gestellt wurde und keine spezifischen Parameter ausgehandelt zu werden brauchen.

Mit dem ausgewählten Anbieter beginnt nun die Dienstnutzung. Falls hierzu noch weitere Kommunikation nötig ist, wird deren Verlauf wiederum durch ein frei bestimmbares eingebettetes Protokoll geregelt. Der Anbieter beendet schließlich die Dienstnutzung, indem er die erfolgreiche Ausführung (`done`) inklusive des Ergebnisses oder einen Fehlschlag (`failure`) mitteilt.

Eingebettete Protokolle

Welche eingebetteten Protokolle zur Aushandlung und Diensterbringung für einen bestimmten Dienst eingesetzt werden können, wird über den Dienstoperator angegeben. Die Festlegung auf eines der möglichen Protokolle geschieht dann jeweils im Zuge der Parameteraushandlung, d.h. die zu verwendenden eingebetteten Protokolle sind Teil der generischen Parameter. Ein Protokoll ist dabei jedoch nicht zwangsläufig an einen bestimmten Dienst gebunden, sondern kann für unterschiedliche Dienste mit übereinstimmender Verhandlungs- oder Diensterbringungsphase verwendet werden. Im einfachsten Fall kann auf eingebettete Protokolle sogar ganz verzichtet werden, falls das Dienst-Metaprotokoll bereits sämtliche benötigte Kommunikation abdeckt.

Der Verlauf eingebetteter Protokolle kann frei bestimmt werden, lediglich die Rollen sind mit Anbieter und Nutzer vorgegeben. Die einzelnen kommunikativen Handlungen innerhalb eines Protokolls werden jeweils über ein Sprechaktmuster definiert, das zumindest den Sprechakttyp und die Form des Inhalts umfaßt. Zur Protokollbeschreibung gehört weiterhin ein eindeutiger Name zur Identifikation sowie die zur Kommunikation zu verwendende Inhaltssprache und die benötigten Ontologien. Schließlich muß noch die Art des Protokolls angegeben werden, d.h. ob es zur Diensterbringung oder zur Verhandlung dient, sowie im letzteren Fall, ob es der Anbieterauswahl, der Parameteraushandlung oder beidem dient.

Eingebettete Protokolle können ihrerseits Unterprotokolle besitzen, die typische Aspekte der Verhandlung oder Diensterbringung allgemein behandeln. Ein Beispiel hierfür ist die Rechnungserstellung, die am Ende von gebührenpflichtigen Diensten nach einem einheitlichen Schema vorgenommen werden kann.

Dienstoperatoren

Wie der Name bereits besagt, ist ein Dienstoperator ein Operator und wird entsprechend spezifiziert und verwendet (vgl. Kapitel 4.1, 4.4.4 und 4.5). Die formale Beschreibung der Handlung über Vorbedingung, Rahmenbedingung und Effekte geschieht dabei aus der Sicht des Dienstnutzers, da dieser den Operator im Rahmen seiner Verhaltenssteuerung verwendet und dann lediglich die Ausführung einem Anbieter des zugehörigen Dienstes überläßt.

Der Ausführungsteil eines Dienstoperators besteht in der Dienstbeschreibung. Neben einem eindeutigen Namen zur Identifikation umfaßt diese einige feste Parameter sowie Vorgaben für verhandelbare generische Parameter. Zu den festen Parametern gehören die Wissensrepräsentationssprache und die Ontologien, mit denen Bedingungen und Effekte formuliert sind. Die verhandelbaren umfassen zumindest die Mengen der verwendbaren Protokolle zur Verhandlung und

Diensterbringung, aus denen jeweils eins ausgewählt werden muß[36]. Die Dienstbeschreibung ist jedoch offen für zusätzliche Parameter[37] wie Sicherheitsanforderungen oder Abrechnungs- und Bezahlungsmodalitäten.

Dienstoperatoren und die in ihnen enthaltenen Dienstbeschreibungen sind prinzipiell unabhängig von einem bestimmten Anbieter oder Nutzer. Die Zuordnung von Diensten zu den jeweiligen Anbietern ist Aufgabe der Agenteninfrastruktur (s. Kapitel 5.2.2).

Ausführbarkeit von Dienstoperatoren

Auch wenn sämtliche Arten von Operatoren bis auf bei der Ausführung durch die Kontrollarchitektur prinzipiell gleich behandelt werden können, gibt es bei Dienstoperatoren eine Besonderheit zu beachten. Wenn ein Agent die Ausführung eines Operators selbst übernimmt, so reichen das Vorhandensein des Operators in der Planbibliothek und die Erfülltheit der Vorbedingungen im Grunde aus, dessen Ausführbarkeit zu gewährleisten. Andernfalls ist der Agent nicht korrekt konfiguriert. Bei Dienstoperatoren ist jedoch i.a. ein anderer Agent als Ausführender involviert, so daß jeweils zusätzlich ein Anbieter gefunden werden muß, der bereit ist, den Dienst zu erbringen.

Aufgrund der Dynamik von Agentengesellschaften impliziert die Existenz eines Dienstoperators nicht, daß es auch Agenten gibt, die den Dienst anbieten. Insbesondere kann sich das verfügbare Dienstangebot jederzeit ändern, weil Agenten einen Dienst nicht mehr anbieten, nicht erreichbar sind oder gar nicht mehr existieren. Und selbst wenn es einen Anbieter gibt, so kann dieser eine Dienstanfrage immer noch ablehnen. Ein Dienstoperator besitzt somit als zusätzliche implizite Vorbedingung, daß mindestens ein Anbieter bereit ist, den Dienst zu erbringen.

Dies ist vor allem dann von Relevanz, wenn Dienstoperatoren in einem Skript oder Plan enthalten sind, denn in diesem Fall kann sich die ungewisse Ausführbarkeit eines Operators auf die der gesamten Handlungssequenz auswirken. Deshalb sollte vor der Ausführung einer Handlungssequenz die Ausführbarkeit aller Einzelschritte gewährleistet sein, zumal die Ausführung von Operatoren mit Aufwand verbunden ist, wozu insbesondere bei Dienstoperatoren auch finanzielle Kosten zählen können.

Bezogen auf Dienstoperatoren bedeutet dies, daß vor der Ausführung einer Handlungssequenz für alle enthaltenen Dienstoperatoren eine Zusicherung der Bereitschaft zur Erbringung des Dienstes von jeweils mindestens einem Anbieter eingeholt werden sollte. Realisieren läßt sich das, indem für sämtliche Dienstoperatoren die Initiierungsphase des Metaprotokolls vorgezogen wird und die Ausfüh-

---

[36] Daß ein eingebettetes Protokoll nicht benötigt wird, entspricht einem leeren Protokoll ohne kommunikative Handlungen und wird so explizit als Protokoll repräsentiert.

[37] Das Schema zur Dienstbeschreibung wird über eine Kategorie in einer Ontologie definiert, so daß zusätzliche Parameter in einer abgeleiteten Kategorie deklariert werden können.

rung des Skripts oder Plans nur dann begonnen wird, wenn zu jeder Dienstanfrage mindestens eine Zusage empfangen wurde, so daß für alle Einzelschritte die prinzipielle Ausführbarkeit gegeben ist. Da auch die Plangenerierung mit nicht unerheblichem Aufwand verbunden sein kann und die Kosten zur Ausführung eines Dienstoperators mit vom jeweiligen Anbieter und den ausgehandelten Parametern abhängen können, läßt sich hierbei die Initiierungsphase noch weiter vorziehen, um bei der Planung nur sicher ausführbare Dienstoperatoren mit bekannten Kosten zu verwenden.

5.1.3. Umsetzung von Interaktionen

Das Schema zur wissensbasierten Verhaltenssteuerung und deren Umsetzung durch die Standardarchitektur von CASA (vgl. Kapitel 4) beinhalten bereits Wissenstypen und Komponentenrollen zur Realisierung von Interaktionen gemäß dem oben vorgestellten Dienstkonzept. Diese wird im folgenden eingehender betrachtet.

Dazu werden drei Ebenen unterschieden:

- *Dienst-Metaprotokoll*: Jeder Agent muß zur Interaktion das Dienst-Metaprotokoll beherrschen, wofür die Kommunikationsrolle zuständig ist. Dies kann auf eine der beiden Rollen beschränkt sein, falls ein Agent ausschließlich als Anbieter bzw. als Nutzer von Diensten auftritt.
- *eingebettete Protokolle*: Da eingebettete Protokolle frei definierbar sind, werden sie jeweils durch eigene Protokolloperatoren umgesetzt. Ein Protokolloperator ist ein Operator, der einem Dienst und einer Rolle in einem Protokoll zugeordnet ist und kommunikative Handlungen enthalten kann.
- *kommunikative Handlungen*: Die kommunikativen Handlungen innerhalb von Protokolloperatoren werden durch spezielle Sprechaktoperatoren zum Versenden und Empfangen von Sprechakten beschrieben. Die Kommunikation für das Metaprotokoll übernimmt die Kommunikationsrolle direkt.

Innerhalb der Standardarchitektur übernimmt dabei die Kommunikationsrolle die rein interaktionsspezifischen Aspekte, indem sie das Metaprotokoll und die Kommunikation durchführt. Gemäß der Offenheit des CASA-Ansatzes ist die nachfolgend beschriebene Umsetzung jedoch nur eine nicht verbindliche Variante zur Kontrolle von interaktiven Handlungen. Für über Dienste interagierende Agenten verbindlich ist lediglich die Einhaltung des Dienst-Metaprotokolls und die Beschreibung von Diensten über Dienstoperatoren.

Dienst-Metaprotokoll

Initiiert wird das Metaprotokoll auf der Nutzerseite durch das Ausführen eines Dienstoperators und auf der Anbieterseite durch den Empfang einer Dienstanfrage. Aufgabe der Kommunikationsrolle ist dann die Durchführung der kommunikativen Handlungen und das Initiieren der eingebetteten Protokolle gemäß dem Metaproto-

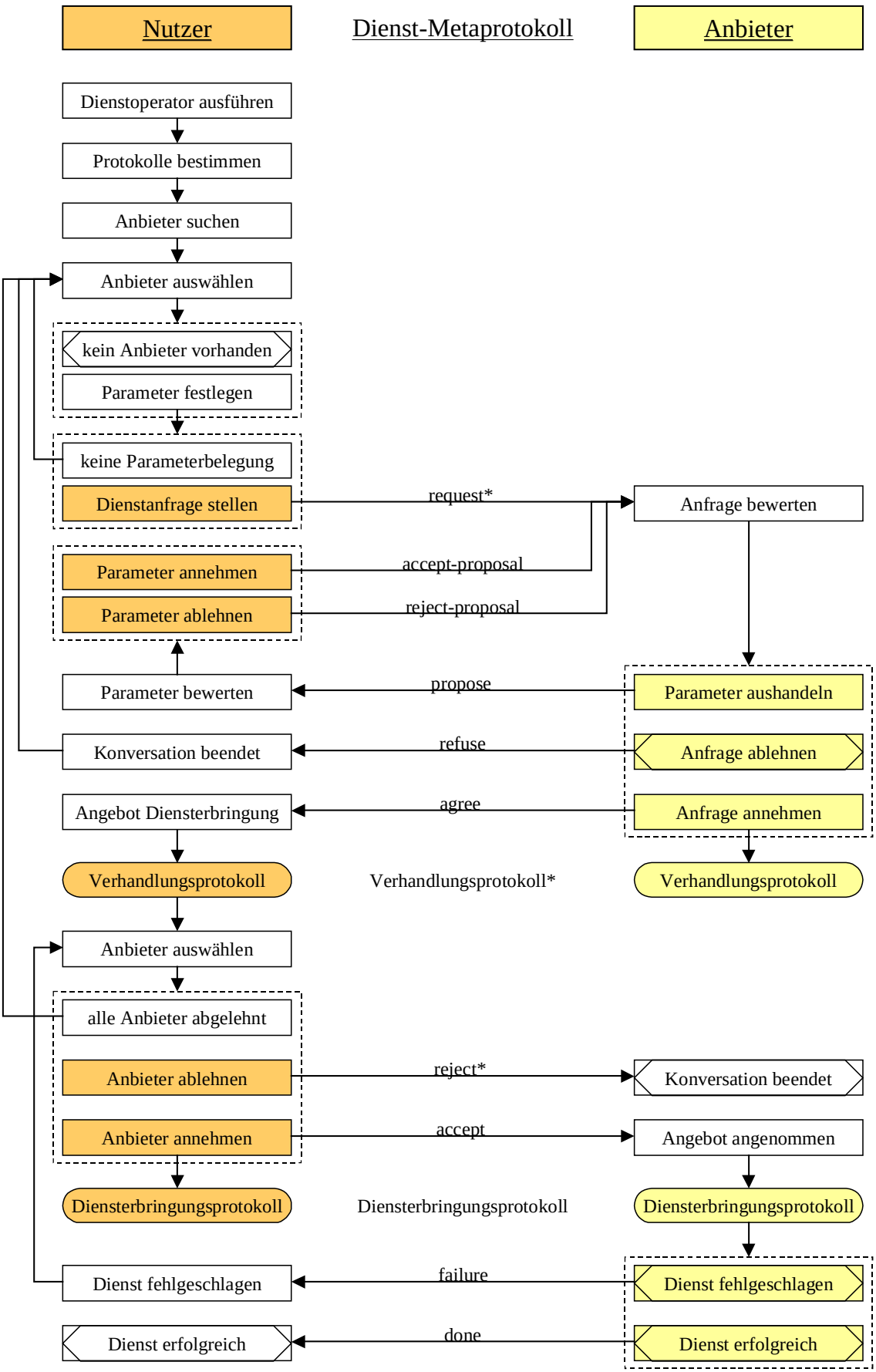

Abbildung 20: Umsetzung des Dienst-Metaprotokolls durch die Kommunikationsrolle

koll und der Dienstbeschreibung. Einer Dienstnutzung entspricht jeweils genau eine Instanz des Metaprotokolls, wobei sämtliche zugehörigen kommunikativen Handlungen einer Konversation zugeordnet werden. Die Repräsentation des Metaprotokolls verbleibt dabei rein operational.

Abbildung 20 faßt die nachfolgend beschriebenen einzelnen Schritte der Kommunikationsrolle zur Umsetzung des Metaprotokolls relativ zur Rolle des Nutzers bzw. Anbieters zusammen. (Mehrere gestrichelt umrandete Schritte stellen Alternativen dar. Mit einem Stern gekennzeichnete kommunikative Handlungen können mit mehreren Agenten gleichzeitig stattfinden. Kästchen mit abgerundeten Ecken beinhalten mehrere Schritte und Kästchen mit markierten Ecken sind Endpunkte der Ausführung.) Der Übersichtlichkeit halber ist der Fall ohne Verhandlungsprotokoll nicht dargestellt, bei dem statt dem Verhandlungsprotokoll sofort das Dienstnutzungsprotokoll gestartet wird.

Dienstoperatoren ausführen

Die Ausführung eines Dienstoperators bedeutet die Nutzung eines Dienstes. Dazu wird die entsprechende Intention von der Handlungsausführung an die Kommunikation weitergeleitet. Letztere stellt zunächst fest, für welche der für den Dienst vorgesehenen eingebetteten Protokolle der Agent Protokolloperatoren in der Nutzerrolle besitzt, da ohne diese eine Durchführung des Metaprotokolls und damit eine Dienstnutzung nicht möglich ist. Zur Verhandlung wie zur Diensterbringung muß mindestens ein Protokolloperator existieren, wobei das leere Protokoll ohne Kommunikation immer als vorhanden angesehen werden kann.

Anschließend sucht die Kommunikationsrolle nach Anbietern des Dienstes, wozu sie einerseits auf Wissen in der Faktenbasis und andererseits auf Infrastrukturdienste zur Verwaltung von Diensten (s. Kapitel 5.2.2) zurückgreifen kann. Gibt es keinen Anbieter, so schlägt die Ausführung fehl. Andernfalls muß entschieden werden, ob eine Dienstanfrage an einen einzigen oder an mehrere Anbieter gestellt werden soll. Letzteres ist nur möglich, wenn der Dienst ein Verhandlungsprotokoll mit Anbieterauswahl erlaubt. Entsprechend wird aus den bekannten Anbietern einer oder mehrere ausgewählt und die initialen Parameter für die Anfrage festgelegt. Als Werte für die eingebetteten Protokolle können dabei wie später in der generischen Aushandlungsphase nur Protokolle verwendet werden, für die ein Protokolloperator existiert und die der Festlegung auf einen oder mehrere Anbieter gerecht werden.

Stehen die initialen Parameter fest, so wird an die ausgewählten Anbieter jeweils eine gleichlautende Dienstanfrage gestellt. Zu den darin übertragenen Parametern gehört auch die Variablenbelegung für die Intention zur Dienstnutzung, da diese die Parametrisierung des Dienstoperators beinhaltet. Falls die Sicherheitsanforderungen des Dienstes dies verlangen, wird zuvor noch die Sicherheitsaushandlung durchgeführt.

Die Anbieter haben nun die Gelegenheit, über die generischen Parameter zu verhandeln. Bei mehreren Anbietern kann dabei das eingebettete Protokoll für die Auswahlphase nicht Gegenstand der generischen Aushandlung sein, da sämtliche Anbieter hierfür dasselbe Protokoll befolgen müssen. Die Vorschläge der einzelnen Anbieter muß der Nutzer jeweils entweder annehmen oder ablehnen. Im letzteren Fall kann er zugleich einen Gegenvorschlag übermitteln. Schließlich muß jeder Anbieter die Dienstanfrage annehmen oder ablehnen. Erfolgt durch keinen Anbieter eine Zusage zur Diensterbringung, so beginnt der gesamte Vorgang wieder bei der Auswahl von Anbietern unter Ausschluß der bereits befragten.

Wird eine spezifische Verhandlung oder eine Auswahl benötigt, so beginnt das zugehörige eingebettete Protokoll mit denjenigen Agenten, die ihre Bereitschaft zur Diensterbringung zugesagt haben. Die Kommunikationsrolle stellt dazu jeweils ein Interaktionsziel pro Anbieter zur Durchführung dieses Protokolls für den betreffenden Dienst in der Rolle des Nutzers auf. Das Resultat dieser Ziele legt fest, welcher der Anbieter den Auftrag zur Diensterbringung erhält und welche eine Ablehnung. Werden sämtliche Anbieter abgelehnt, muß wiederum zur ersten Anbieterauswahl zurückgekehrt werden.

Ob mit oder ohne Verhandlung verbleibt zu Beginn des Diensterbringungsprotokolls genau ein Anbieter, der den Dienst zu erbringen hat. Auch dieses eingebettete Protokoll wird über ein entsprechendes Interaktionsziel initiiert. Zum Abschluß der Diensterbringung teilt der Anbieter dessen Ergebnis mit. Ist dieses negativ oder ist das Interaktionsziel fehlgeschlagen, so muß erneut eine Anbieterauswahl vorgenommen werden, wobei gegebenenfalls zunächst auf noch nicht in der Auswahlphase des Metaprotokolls abgelehnte Anbieter zurückgegriffen werden kann. Nur wenn das Resultat des Anbieters wie auch des Interaktionsziels positiv sind, ist die Dienstnutzung erfolgreich abgeschlossen. Falls noch Anbieter keine Zusage oder Ablehnung in der Auswahlphase erhalten haben, so werden diese nunmehr abgelehnt.

Die Ausführung eines Dienstoperators endet somit entweder erfolgreich, sobald ein Anbieter den geforderten Dienst erbracht hat, oder wenn an keinen Anbieter mehr eine Anfrage gerichtet werden kann. In jedem Fall wird das Ergebnis der Handlungsausführung mitgeteilt, die wie bei jedem anderen Operatortyp die zugehörige Intention entsprechend beendet.

Dienst-Aufträge entgegennehmen

Seitens des Anbieters beginnt das Metaprotokoll mit dem Empfang einer Dienstanfrage. Jeder Agent, der Dienste anbietet, muß deshalb Dienstanfragen verarbeiten können. Der initiale Sprechakt des Metaprotokolls ist der einzige Sprechakt, den die Kommunikationsrolle außerhalb von laufenden Konversationen bearbeiten können muß, weil diese jeweils eine neue Konversation etablieren. Alle übrigen Sprechakte sowie Dienstanfragen zu nicht angebotenen Diensten können als nicht verarbeitbar

beantwortet werden, wenn sie keiner bestehenden Konversation zugeordnet werden können.

Geht eine Anfrage zu einem von dem betreffenden Agenten angebotenen Dienst ein, so wird diese zunächst dahingehend bewertet, ob sie angenommen werden soll oder nicht oder ob zuvor noch über generische Parameter verhandelt werden soll. Insbesondere kann eine Anfrage nur dann angenommen werden, wenn der Agent für die zu verwendenden eingebetteten Protokolle auch über entsprechende Protokolloperatoren in der Rolle des Anbieters verfügt. Gegebenenfalls lassen sich auch Zugriffsbeschränkungen über eine dafür vorgesehene Sicherheitskomponente überprüfen, so daß nur bestimmte Agenten einen Dienst nutzen dürfen.

Zur Aushandlung eines generischen Parameters sendet der Anbieter einen alternativen Vorschlag, den der Nutzer entweder annimmt oder ablehnt und u.U. einen Gegenvorschlag unterbreitet. In Anbetracht der Parameteränderung wird die Anfrage jeweils erneut bewertet, bis keine weitere Verhandlung mehr nötig oder möglich ist. Falls keine Übereinkunft gefunden wird oder der Dienst aus einem anderen Grund nicht erbracht werden kann oder soll, lehnt der Anbieter die Anfrage ab und beendet damit die Konversation. Andernfalls erklärt der Anbieter seine Bereitschaft zur Erbringung des Dienstes.

Daraufhin beginnt falls benötigt die Auswahlphase. Wie der Nutzer stellt auch der Anbieter hierbei ein Interaktionsziel zur Durchführung des zugehörigen Protokolls auf, allerdings für die Rolle des Anbieters. Wird das Ziel nicht erreicht oder erhält der Anbieter nicht den Auftrag, so beendet er die Konversation.

Andernfalls ist der Anbieter mit der Diensterbringung beauftragt, die er mit dem Aufstellen eines Interaktionsziels für das entsprechende Protokoll beginnt. Das Resultat dieses Ziels ist zugleich das Resultat der Diensterbringung, das dem Nutzer mitgeteilt wird, woraufhin seitens des Anbieters die Konversation beendet wird.

Fehlerbehandlung

Unabhängig von der Rolle innerhalb des Metaprotokolls muß die Kommunikationsrolle zudem noch jederzeit auf Fehler wie nicht verarbeitbare Sprechakte, überschrittene Zeitlimits und den Abbruch der Konversation durch den Kommunikationspartner reagieren können. Falls keine explizite Fehlerbehebung möglich ist, was aufgrund der Vielzahl unterschiedlicher Fehlerquellen der Regelfall ist, so wird die Konversation mit dem betreffenden Gegenüber abgebrochen. Besteht zu diesem Zeitpunkt ein Interaktionsziel für ein eingebettetes Protokoll, so wird dieses zurückgenommen. Im Falle der Anbieterrolle endet damit das Metaprotokoll, während der Nutzer wie in den übrigen Fällen, in denen der Dienst noch nicht erbracht ist, das Metaprotokoll noch mit anderen Anbietern fortführen kann.

Repräsentation von Protokollen

Für das Dienst-Metaprotokoll gibt es innerhalb eines Agenten keine explizite Repräsentation des Protokolls als solchem. Statt dessen wird für jede der beiden Rollen eine eigene Operationalisierung verwendet, die jedoch insofern ohnehin jeweils beide Rollen widerspiegelt, als ein versendeter Sprechakt der einen Rolle gerade einem empfangenen Sprechakt der anderen Rolle entspricht und umgekehrt. Und auch die Umsetzung eingebetteter Protokolle durch Protokolloperatoren stellt eine Operationalisierung relativ zur Rolle ohne explizite Repräsentation dar.

Der Grund hierfür wird offensichtlich, wenn man das Dienst-Metaprotokoll (Abbildung 19) und seine Operationalisierung betrachtet (Abbildung 20). Die Operationalisierung ergibt sich dabei nicht alleine aus dem Protokoll, sondern enthält zusätzliche, frei bestimmbare Elemente zur Ablaufkontrolle und zur Bestimmung der Inhalte der Kommunikation, für die das Protokoll lediglich einen Rahmen vorgibt. Entsprechend ist auch eine explizite Repräsentation des Protokolls nicht hinreichend zu dessen Durchführung. Andererseits ist in der Operationalisierung jeder Rolle das Protokoll jeweils vollständig enthalten, so daß eine zusätzliche explizite Repräsentation keinen zusätzlichen Nutzen bringt und auf diese somit verzichtet werden kann.

Dennoch ist es zur Definition von Protokollen und zur Erstellung von Protokolloperatoren zumeist unverzichtbar, von einer expliziten Repräsentation beispielsweise in Form eines Zustandsgraphen auszugehen, um die Konformität einer Operationalisierung zur jeweiligen Rolle innerhalb des Protokolls gewährleisten und überprüfen zu können, auch wenn diese von den Agenten selbst nicht genutzt werden kann.

Eingebettete Protokolle

Die Operationalisierung von eingebetteten Protokollen geschieht durch besondere Operatoren, die Protokolloperatoren. Dadurch lassen sich die durch Operatoren ausdrückbaren Handlungstypen und Kontrollmöglichkeiten voll zur Umsetzung der Protokolle nutzen. Als zusätzlicher Handlungstyp kommt dabei noch die Kommunikation hinzu, in der Sprechakte versendet oder empfangen werden. Der Protokolloperator bildet dabei die Kontrollstruktur und erstellt und verarbeitet die Inhalte der Kommunikation, während die Verwendung der kommunikativen Handlungen gerade der Rolle innerhalb des entsprechenden Protokolls Rechnung tragen muß.

Da die eingebetteten Protokolle Teil der generischen Verhandlung sind und beide Rollen die entsprechenden Parameter nur akzeptieren, falls sie einen geeigneten Protokolloperator besitzen, ist mit einer Übereinkunft bei der generischen Aushandlung auch die Durchführbarkeit der ausgehandelten eingebetteten Protokolle sichergestellt.

Protokolloperatoren

Ein Protokolloperator ist ein gewöhnlicher Operator, der einem Dienst, einem Protokoll und einer Rolle in diesem zugeordnet ist. Die Zuordnung zu einem Dienst beinhaltet dabei einen Zugriff auf dessen Beschreibung durch einen Dienstoperator inklusive der dort deklarierten Variablen. Ein Protokolloperator kann somit als ein eingebetteter Teiloperator des Dienstoperators aufgefaßt werden. Zudem kann ein Protokolloperator anders als andere Operatoren Sprechaktoperatoren zur Durchführung kommunikativer Handlungen enthalten. Dies ist allerdings nur bei zusammengesetzten Operatortypen möglich, weshalb Protokolloperatoren üblicherweise Skripte oder vordefinierte Pläne sind. Ebenso wie Protokolle Unterprotokolle besitzen können, können auch Protokolloperatoren wiederum Protokolloperatoren zur Umsetzung dieser Unterprotokolle enthalten.

Die Ausführung eines Protokolloperators geschieht infolge eines Interaktionsziels, das durch die Kommunikationsrolle im Rahmen der Durchführung des Dienst-Metaprotokolls aufgestellt wird. Es enthält neben Dienst, Protokoll und Rolle, die zur Auswahl eines Protokolloperators verwendet werden, auch den benötigten Konversationskontext zur Instantiierung des Protokolloperators inklusive der Variablenbelegung für den Dienstoperator. Die aus der Handlungsauswahl resultierende Intention wird von der Handlungsausführung verarbeitet. Die Ausführung entspricht der des zugehörigen normalen Operators. Hinzu kommt lediglich der Zugriff auf die Parametrisierung des Dienstes sowie auf diejenigen Konversationsparameter, die zur Durchführung kommunikativer Handlungen benötigt werden.

Eine Besonderheit stellt noch die Umsetzung des leeren Protokolls ohne Kommunikation dar, das in der Auswahlphase wie in der Diensterbringungsphase vorkommen kann. In jedem Fall muß dennoch ein Interaktionsziel aufgestellt und somit ein Protokolloperator ausgeführt werden, da das Ergebnis dieses Ziels die Entscheidung über die Fortsetzung des Metaprotokolls beinhalt. Dies betrifft jedoch nur diejenige Rolle, die in der jeweiligen Phase diese Entscheidung zu treffen hat. In der Auswahlphase ist dies der Nutzer, bei der Diensterbringung der Anbieter. Die jeweils andere Rolle stellt kein Interaktionsziel auf und führt entsprechend keinen Protokolloperator aus.

Kommunikative Handlungen

Die Durchführung von kommunikativen Handlungen besteht im Senden und Empfangen von Sprechakten, die für den Transport gemäß dem FIPA-Ansatz zusätzlich in einen sogenannten Brief verpackt werden (vgl. Kapitel 6.8). Jeder Sprechakt gehört in der Regel zu einer Dienstnutzung und ist somit einer Konversation und einem Protokoll zugeordnet, die beide über eigene Sprechaktparameter angegeben werden. An Protokollen sind dann nur das Dienst-Metaprotokoll oder ein eingebettetes Protokoll einer laufenden Dienstnutzung zulässig.

Um auch Konversationsprotokolle durchführen zu können, die nicht dem CASA-Dienst-Schema entsprechen, können eigene Komponentenrollen zu deren Umsetzung eingeführt werden. Die Kommunikationsrolle übernimmt dann lediglich die Zustellung und Weiterleitung der Sprechakte in Abhängigkeit des jeweiligen Protokolls, während die für ein bestimmtes Protokoll zuständige Rolle die Erzeugung und Verarbeitung der Sprechakte übernimmt. Beispiele für derartige Protokolle sind das Protokoll zur Sicherheitsaushandlung oder die Protokolle zur Umsetzung von Infrastrukturdiensten gemäß dem FIPA-Standard (vgl. Kapitel 5.2.2).

Senden von Sprechakten

Die Quelle eines zu versendenden Sprechakts kann ein Sprechaktoperator, die Kommunikationsrolle selbst oder eine Rolle zur Umsetzung von weiteren Protokollen sein. Im ersten Fall delegiert die Handlungsausführung eine Intention mit dem Sprechaktoperator und seiner Instantiierung an die Kommunikationsrolle, die daraus den Sprechakt erzeugt. Im zweiten Fall ist der Sprechakt Teil des Metaprotokolls und wird von der Kommunikationsrolle direkt erzeugt. Im letzten Fall schließlich wird bereits der fertige Sprechakt übergeben.

Zum Versenden wird nun für den Sprechakt ein Brief erstellt, der neben dem Sprechakt die Namen und Adressen von Absender und Empfänger enthält. Die Adressen des Absenders ergeben sich aus den vorhandenen Transportkomponenten, die des Empfängers entstammen in der Regel der Faktenbasis. Um die jeweils aktuellen Adressen eines Empfängers zu erhalten, kann ein dafür vorgesehener Infrastrukturdienst genutzt werden (vgl. Kapitel 5.2.2).

Nun wird ein geeigneter Transportweg gewählt, für den beide Kommunikationspartner eine Adresse besitzen, so daß sie auf diesem Weg Sprechakte sowohl senden als auch empfangen können. Gegebenenfalls müssen dabei noch Sicherheitsanforderungen berücksichtigt werden. Schließlich wird der Brief gemäß der gewählten Adressierung an die entsprechende Transportkomponente übergeben, die mitteilt, ob sie diesen zustellen konnte oder nicht.

Wurde der Brief zugestellt, wird dies der jeweiligen initiierenden Rolle des Sprechakts mitgeteilt. Andernfalls können andere mögliche Transportwege versucht werden und gegebenenfalls die Adressen des Empfängers aktualisiert werden. Wenn auch dies nicht zum Erfolg führt, so wird dies ebenfalls der initiierenden Rolle mitgeteilt.

Empfang von Sprechakten

Der erste Schritt beim Empfang eines Briefs ist die Überprüfung des Namens des Empfängers, um falsch adressierte Briefe abzulehnen. Bei korrekter Adressierung erfolgt dann die Zuordnung des Sprechakts anhand der angegebenen Parameter für Konversation und Protokoll. Da für jede Konversation eine eindeutige Bezeichnung verwendet werden muß, ist diese Zuordnung immer eindeutig.

Bezeichnet die Konversation eine laufende Dienstnutzung, so muß das Protokoll entweder das Metaprotokoll oder eines der ausgehandelten eingebetteten Protokolle sein. Im ersten Fall geschieht die Verarbeitung gemäß dem gegenwärtigen Stand der Dienstnutzung. Im zweiten Fall muß sich zudem das Metaprotokoll in der entsprechenden Phase befinden, d.h. ein Interaktionsziel für das eingebettete Protokoll muß bereits aufgestellt und noch nicht beendet sein. In diesem Fall wird der Sprechakt einem passenden in Ausführung befindlichen Operator zum Sprechaktempfang für diese Konversation zugeteilt, so daß dessen Ausführung dadurch erfolgreich beendet wird. Da die Ausführung des Empfangsoperators und der tatsächliche Empfang des Sprechakts zeitlich voneinander unabhängig sind, kann die Zuordnung erst dann stattfinden, wenn beide Ereignisse eingetreten sind, worauf entsprechend gewartet werden muß. Die Wartezeit kann durch ein Zeitlimit begrenzt werden, nach dessen Ablauf ein Fehler angenommen wird.

Bezeichnet die Konversation keine laufende Dienstnutzung, so sind als Protokoll nur das Metaprotokoll oder eines der Protokolle mit eigener zugeordneter Komponente zulässig. Im ersten Fall initiiert eine Dienstanfrage für einen angebotenen Dienst eine Dienstnutzung, andere Sprechakte werden nicht akzeptiert. Im zweiten Fall wird der Sprechakt an die betreffende Komponente weitergeleitet.

Alle übrigen Sprechakte können nicht zugeordnet werden und sind deshalb nicht verarbeitbar. Aber auch ein erfolgreich zugeordneter Sprechakt braucht nicht verarbeitbar zu sein, beispielsweise weil der Inhalt nicht interpretierbar ist oder eine Protokollverletzung vorliegt. Jeden nicht verarbeitbaren Brief beantwortet ein Agent mit einem Sprechakt vom Typ `not-understood`, um dem Absender den Fehler mitzuteilen. Lediglich nicht verarbeitbare Sprechakte dieses Fehlertyps werden nicht wiederum beantwortet, um einen endlosen Austausch gegenseitiger Fehlermeldungen zu vermeiden.

Sprechaktoperatoren

Das Senden und Empfangen von Sprechakten aus Protokolloperatoren heraus geschieht über Sprechaktoperatoren. Diese besitzen anders als andere Operatoren keine Bedingungen und Effekte, da sich diese bei kommunikativen Handlungen nur über formale Sprachen korrekt beschreiben lassen, die hinreichend ausdrucksstark zur Formulierung von Wissen über andere Agenten sind, weshalb sie für eine effiziente Verwendung i.d.R. zu komplex sind. Sie werden aber auch ohnehin nicht benötigt, da Sprechaktoperatoren aufgrund der Strukturierung von Konversationen durch Protokolle nicht zur Handlungsauswahl verwendet werden können, sondern nur innerhalb von Protokolloperatoren vorkommen, deren Struktur durch das zugehörige Protokoll vorgegeben ist, das damit implizit, wenn auch nicht formal Bedingungen und Effekte des Sprechakts festlegt.

Im Falle eines Sprechaktoperators zum Versenden besteht der Ausführungsteil aus einem Sprechaktmuster, das zur Ausführung durch den Protokolloperator über

enthaltene Variablen instantiiert wird. Die Ausführung besteht im Versenden des Sprechakts durch die Kommunikationsrolle.

Der Ausführungsteil eines Sprechaktoperators für den Empfang beinhaltet eine Menge an Sprechaktmustern, um den Empfang verschiedener alternativer Sprechakte zu ermöglichen. Das Eintreten einer dieser Alternativen wird analog zum Eintreten eines Effekts bei Operatoren mit mehreren Effekten behandelt. Zur Ausführung wird auf das Eintreffen eines Sprechakts gewartet, der dem übergeordneten Protokolloperator zugeordnet ist und auf eines der vorgegebenen Sprechaktmuster paßt. Die in dem Muster verwendeten Variablen werden dem empfangenen Sprechakt entsprechend mit Werten belegt, so daß der Inhalt des Sprechakts durch nachfolgende Schritte des Protokolloperators verarbeitet werden kann.

5.2. Infrastruktur

Konstituiert wird eine Agentengesellschaft als Gesellschaft durch eine Infrastruktur, die die zugehörigen Agenten verwaltet. Diese unterstützt Interaktionen zwischen den Agenten, indem sie Informationen über die Mitglieder einer Agentengesellschaft sammelt und diesen wiederum zur Verfügung stellt. Sie übernimmt organisatorische und vermittlerische Aufgaben, die unabhängig von der jeweiligen Anwendung in einer einheitlichen Weise innerhalb einer Agentengesellschaft benötigt werden.

Da der Schwerpunkt der vorliegenden Arbeit auf der Ebene der Agentenarchitektur und nicht auf der Infrastrukturebene liegt, beschränkt sich die folgende Darstellung einer Infrastruktur für Gesellschaften aus CASA-Agenten auf die Anwendung bestehender Ansätze auf die Architekturkonzepte in CASA. Das beschriebene Infrastrukturkonzept orientiert sich dabei vorwiegend an der FIPA-Spezifikation zum Agentenmanagement [FIPA 2000: XC00023G, XC00067B], ohne diese als bindenden Standard anzusehen, nicht zuletzt wegen der in Kapitel 2.4.1 diskutierten Problematik der Konformität zu Standards im Agentenbereich. Eine FIPA-konforme Implementierung ist auf dieser Grundlage möglich, aber nicht zwingend.

Zur Verwaltung werden Agenten einer Agentenplattform zugeordnet, auf der ausgezeichnete Agenten die Infrastruktur über eine Menge von Diensten realisieren. Zu den Aufgaben der Infrastruktur zählt zum einen die Unterstützung der Kommunikation zwischen Agenten, indem Informationen über die existierenden Agenten und ihre Kommunikationsmöglichkeiten bereitgestellt werden. Zusätzliche Dienste können auch den Transport von Sprechakten ganz übernehmen. Zum anderen übernimmt die Infrastruktur die Vermittlung von Diensten, wozu sie die Gesamtheit der angebotenen Dienste sowie deren Anbieter verwaltet. Prinzipiell sind neben diesen beiden notwendigen Infrastrukturfunktionalitäten noch weitere möglich. Beispiele hierfür sind die Agentenmobilität, die Agenten einen Wechsel ihres Ausführungsortes zur Laufzeit ermöglicht, erweitertes Agentenmanagement

beispielsweise zur Konfiguration sowie Sicherheitsfunktionalitäten. Im folgenden werden zunächst Agentenplattformen und die auf diesen angesiedelten Infrastrukturdienste beschrieben und anschließend die Nutzung dieser Infrastruktur durch Komponenten der Standardarchitektur.

5.2.1. Agentenplattformen

Eine Agentenplattform (Agent Platform, AP) stellt die Basiseinheit einer Agenteninfrastruktur da. Eine einzelne AP ist dabei gemäß dem FIPA-Managementmodell darüber definiert, daß sie eine minimale Menge an Infrastrukturdiensten bereitstellt. Im einzelnen sind dies der Sprechakttransport zwischen Agenten (Message Transport System, MTS), die Verwaltung von Agenten und ihren Adressen (Agent Management Service, AMS) sowie die Verwaltung von Diensten und ihren Anbietern (Directory Facilitator, DF). Die Funktionalität dieser Dienste wird im nachfolgenden Teilkapitel beschrieben.

Jeder Agent ist einer AP zugeordnet, die ihn momentan verwaltet. Zudem besitzt jeder Agent eine Heimatplattform (Home Agent Platform, HAP), auf der er erzeugt wurde. Zur Identifikation besitzt jeder Agent einen global eindeutigen Namen, der Teil seines Agent Identifier (AID) ist. Diesen Namen erhält ein Agent, wenn er erzeugt wird, und behält ihn bei, bis er beendet wird. Bei einem mobilen Agenten braucht die AP, der er momentan zugeordnet ist, nicht mit seiner HAP übereinzustimmen.

Eine Agenteninfrastruktur besteht nicht nur aus einer einzige AP, sondern aus mehreren APs, die untereinander verbunden sind, indem ihre Infrastrukturdienste bei den DFs der anderen APs angemeldet sind. Die Kommunikation zwischen den Plattformen ist über deren MTS gewährleistet, jeder Agent kann darüber hinaus aber auch weitere Kommunikationsadressen zur direkten Kommunikation besitzen.

Da der Zugriff auf die Infrastrukturdienste als Agentenkommunikation vermittels Sprechakten über vorgegebene Protokolle definiert ist, sind deren Anbieter zwangsläufig selbst Agenten. Die Vermittlung dieser Dienste kann jedoch nicht auf den Infrastrukturdiensten selbst beruhen, sondern muß zumindest teilweise fest vorgegeben sein. Die Namen und Adressen der Infrastrukturagenten sind deshalb nach dem FIPA-Ansatz abgesehen von dem HAP-Anteil standardisiert. Die Namen lauten ams*@hap* bzw. df*@hap* für die HAP *hap*, und als Adresse ist die des MTS der HAP vorgeschrieben. Die Namen der Dienste dieser Agenten und die dafür zu verwendenden Konversationsprotokolle sind ebenfalls statisch vorgegeben. Eine Ausnahme bildet der MTS, der kein Agent zu sein braucht, weshalb der Zugriff auf dessen Funktionalität nicht über Dienste und Sprechakte geschieht, sondern auf eine durch die AP zu realisierende und durch FIPA nicht spezifizierte Weise.

In CASA kann für den MTS einfach eine ausgezeichnete Transportkomponente verwendet werden, während die anderen Infrastrukturdienste durch einen oder mehrere Agenten bereitgestellt werden. Zwecks Konformität zu FIPA können die

Infrastrukturdienste statt nach dem Dienst-Konzept von CASA auch gemäß den von FIPA vorgegebenen Protokollen realisiert werden, wozu dann eine eigene Komponente zur Durchführung dieser Protokolle benötigt wird (vgl. Kapitel 5.1.3). Alternativ kann auch eine direkte Anbindung der Infrastrukturfunktionalität über die Agentenschnittstelle vorgenommen werden.

5.2.2. Infrastrukturdienste

Im folgenden werden zum einen die minimal notwendigen Infrastrukturdienste zur Agentenverwaltung, Kommunikationsunterstützung und Dienstverwaltung vorgestellt. Dies beinhaltet deren Funktionalität gemäß dem FIPA-Managementmodell sowie einige Besonderheiten, die bei einer Umsetzung im Rahmen der CASA-Architektur zu berücksichtigen sind. Zum anderen wird ein Ansatz zur Realisierung der Persistenz von Agenten basierend auf der Komponentenarchitektur von CASA umrissen.

Agentenverwaltung

Das AMS übernimmt die zentralen Aufgaben hinsichtlich der Verwaltung der Agenten, die sich auf einer AP befinden. Dies betrifft deren Existenz und Ausführung sowie die Registrierung von Agenten und ihren Adressen. Zudem bietet es Dienste an, mit denen Agenten sich über die Agenten einer AP informieren können.

Über das AMS lassen sich Agenten auf einer AP erzeugen sowie vorhandene Agenten beenden. Zum Beenden müssen alle Agenten einen Dienst anbieten, über den das AMS sie zur Terminierung auffordern kann, so daß sich der Agent selbst regulär beenden kann. Um auch fehlerhafte oder schädliche Agenten beenden zu können, kann das AMS eine Terminierung von außen erzwingen, falls ein Agent der Terminierungsaufforderung nicht nachkommt. Da die Migration das Beenden eines Agenten auf einer AP und dessen erneute Erzeugung auf einer anderen AP beinhaltet, übernimmt das AMS auch die Migration, falls diese für eine AP angeboten wird. Das AMS besitzt zudem die Ausführungskontrolle über Agenten, indem es die Ausführung eines Agenten suspendieren und wiederaufnehmen kann.

Ein Agent ist auf einer AP vorhanden, solange er beim AMS registriert ist. Agenten, die das AMS selbst erzeugt oder beendet oder deren Migration es durchführt, werden dabei automatisch an- bzw. abgemeldet. Optional kann ein AMS auch eine sogenannte dynamische Registrierung von Agenten zulassen, bei der sich Agenten auf einer AP anmelden können, die das AMS nicht selbst erzeugt hat. Die Registrierung eines Agenten besteht zumindest aus dem Agentennamen, dessen Eindeutigkeit das AMS dabei gewährleisten muß, dem Ausführungszustand, sowie den Kommunikationsadressen, über die der Agent erreichbar ist. Änderungen des Ausführungszustands oder der Adressen zur Laufzeit muß der Agent dem AMS mitteilen.

Dienste des AMS

Nicht alle Funktionalitäten und Informationen des AMS müssen anderen Agenten in Form von Diensten zur Verfügen stehen. Notwendig sind nur diejenigen, die die Agenten zur Interaktion untereinander oder mit dem AMS benötigen, während die übrigen dem Administrator der AP vorbehalten sein können.

Zu den notwendigen Diensten des AMS gehört die Modifikation der registrierten Informationen über den Agenten, damit Änderungen in diesen zur Laufzeit dem AMS mitgeteilt werden können und dieses den aktuellen Zustand der Agenten der AP widerspiegelt. Unterstützt das AMS dynamische Registrierung, so muß es zusätzlich Dienste zur An- und Abmeldung externer Agenten anbieten. Auch die Migration zu einer anderen AP ist ein optionaler Dienst des AMS.

An Auskunftsmöglichkeiten bietet das AMS Informationen über diejenigen registrierten Eigenschaften, die die Agenten zur Interaktion untereinander benötigen. Dazu gehört die Existenz und der Aufenthaltsort eines Agenten, d.h. ob und auf welcher AP er registriert ist, sowie die Kommunikationsadressen eines Agenten. Für eine Anfrage muß dabei jeweils die AID des gesuchten Agenten angegeben werden. Die Suche ist nicht auf eine AP beschränkt, sondern wird für unbekannte Agenten auf andere APs ausgeweitet, indem die entsprechenden Dienste von bekannten AMS genutzt werden.

AMS für Gesellschaften aus CASA-Agenten

In CASA wird das AMS über einen gewöhnlichen Agenten realisiert, dessen anwendungsspezifische Funktionalität gerade die des AMS ausmacht und der die hierfür benötigten Dienste anbietet.

Als Erzeuger hat der AMS-Agent in CASA Zugriff auf die erzeugten Agenten über deren Schnittstelle nach außen (vgl. Kapitel 3.4.4), über die sich somit grundlegende Verwaltungsfunktionalitäten realisieren lassen. Die Ausführungskontrolle inklusive der erzwungenen Terminierung ist dabei über den Lebenszykluszustand des Agenten möglich[38]. Dies gilt jedoch nicht für dynamisch registrierte, externe Agenten.

[38] Die Kontrolle des Agenten über dessen Schnittstelle und insbesondere über dessen Lebenszyklus durch das AMS setzen eine konforme Implementierung des Agentenkerns und eine Beachtung der Konventionen zur zentralen Verwaltung aller Verarbeitungsressourcen vermittels des Kontrollzyklus bei allen Agenten voraus. Andernfalls bewirkt eine erzwungene Terminierung lediglich die Abmeldung bei der AP. Um dem AMS die vollständige Kontrolle über die Existenz und Ausführung der Agenten zu garantieren, muß das AMS sämtliche Verarbeitungsressourcen aller Agenten direkt verwalten können. Dies kann je nach Implementierungssprache recht aufwendig oder sogar unmöglich sein.

Kommunikation zwischen Agenten

Auch wenn die direkte Kommunikation zwischen Agenten von der FIPA-Spezifikation nicht ausgeschlossen ist, fordert sie mit dem MTS einen Standardmechanismus zum Weiterleiten von Sprechakten innerhalb einer AP wie zwischen den APs. Zur Nutzung des MTS braucht nur die AID bekannt zu sein, die nicht notwendigerweise bereits die Kommunikationsadressen oder den Aufenthaltsort des Adressaten zu enthalten braucht, da das MTS auch Anfragen beim AMS und Routing von Sprechakten über mehrere MTS hinweg unterstützen soll. Andere optionale Funktionalitäten des MTS sind Broadcast-Mechanismen und das Zwischenspeichern von Sprechakten für temporär nicht erreichbare Agenten.

MTS für CASA-APs

Nach dem CASA-Ansatz kann jeder Agent eine beliebige Anzahl von Transportkomponenten zur Kommunikation besitzen. Diese können, brauchen aber nicht auf die Kommunikation mit Agenten derselben AP beschränkt zu sein. Somit besteht keine zwingende Notwendigkeit zur Verwendung eines MTS, zumal dieser eine Engstelle darstellen kann, wenn er für sämtliche Kommunikation zwischen Agenten in Anspruch genommen wird. Andererseits schont ein zentraler MTS beschränkte Ressourcen wie IP-Adressen, die sich bei der Beschränkung auf einen einzigen Agenten zur AP-übergreifenden Kommunikation einsparen lassen.

Dienstvermittlung

Der DF dient der Vermittlung von Diensten. Bei ihm registrieren Agenten diejenigen Dienste, die sie zur Nutzung durch andere Agenten anbieten. Der DF selbst stellt Dienste zur Verfügung, um Dienste an- und wieder abzumelden, die Dienstbeschreibung zu ändern sowie um nach Diensten und ihren Anbietern zu suchen.

Zur Registrierung von Diensten wird durch die FIPA-Spezifikation ein einheitliches Format zur Dienstbeschreibung vorgegeben. Diese besteht neben einem Namen zur Identifikation, einem Diensttyp zur Kategorisierung und der Dienstontologie aus einer Mengen an frei definierbaren festen und verhandelbaren Eigenschaften. Verwaltet werden die registrierten Dienste ausschließlich relativ zu den registrierten Agenten, d.h. die Registrierung erfolgt jeweils für einen Agenten als eine Liste von Diensten.

Auch die Anfragen sind deshalb beschränkt auf die Daten, die für einen Agenten registriert sind. Der Dienst zur Suche nach angebotenen Diensten verlangt als Anfrageparameter Angaben zu einem Agenten und liefert als Antwort eine Menge von passenden Agentenbeschreibungen, die jeweils die Beschreibung der von diesen angebotenen Diensten enthalten. Die Suche kann auf einen DF beschränkt sein oder andere DFs mit einbeziehen.

Die FIPA-Spezifikation des DF besitzt zwei Schwachstellen. Zum einen ist die Repräsentation der Dienstbeschreibungen, wie sie in den Diensten zur Nutzung des DF verwendet wird, rein agentenzentriert. Dies ist zwar keine Einschränkung in dem Sinne, daß dadurch irgendwelche dienstbezogenen Informationen nicht beim DF registriert oder abgerufen werden können, es ist jedoch ineffizient, da hierbei jeweils die Agentenbeschreibung als minimale Informationseinheit dient. Beispielsweise liefert die Suche nach einem Dienst als Ergebnis eine Liste der vollständigen Agentenbeschreibung aller Agenten, die diesen Dienst anbieten, jeweils inklusive sämtlicher übrigen nicht gesuchten Dienste.

Dies läßt sich leicht beheben, indem der DF zusätzliche Dienste für eine analoge Funktionalität mit einem dienstzentrierten Datenformat anbietet. Da gemäß der Standardarchitektur von CASA die Dienstbeschreibung bereits zur Handlungsauswahl, die Anbieter des Dienstes aber erst zur Durchführung der Dienstnutzung benötigt werden, empfiehlt es sich, getrennte DF-Dienste zur Suche nach angebotenen Diensten und zur Suche nach den Anbietern eines Dienstes zu realisieren, so daß die jeweilige Information nur dann abgefragt zu werden braucht, wenn sie auch wirklich benötigt wird.

Die andere, gravierendere Schwachstelle ist die fehlende Semantik der Dienstbeschreibung gemäß der FIPA-Spezifikation. Die einzigen obligatorischen Parameter der Dienstbeschreibung sind – die von FIPA vordefinierten Bezeichnungen für die Infrastrukturdienste ausgenommen – beliebige Bezeichner ohne festgelegte Semantik. Somit kann ein Agent nur nach Diensten suchen und nur Dienste nutzen, die er bereits im voraus kennt, d.h. er kann letztlich nur nach den Anbietern von ihm bekannten Diensten suchen.

Das Dienst-Konzept von CASA geht hier weiter, da hier die Dienstbeschreibung immer die Bedingungen und Effekte einer Dienstnutzung aus der Sicht des Nutzers umfaßt, deren Semantik relativ zu verwendeten Sprache und Ontologien klar definiert ist. Dies ermöglicht CASA-Agenten, beim DF auch nach im voraus nicht bekannten Diensten zu suchen, deren Effekt ein Ziel des Agenten erfüllt. Auf diese Weise wird ein höherer Grad an Flexibilisierung und Automation in der Interaktion zwischen Agenten erreicht.

Da die Dienstbeschreibungssprache von FIPA offen ist für beliebige, frei definierbare Parameter, läßt sie sich durch vorgegebene Parameter erweitern, die die Dienstbeschreibung gemäß dem Dienstkonzept von CASA widerspiegeln, so daß eine Realisierung des DF konform zur FIPA-Spezikation möglich ist. Die Semantik dieser Parameter ist dann allerdings nur CASA-Agenten bekannt.

Persistenz

Persistenz beinhaltet das Speichern des Ausführungszustands eines Agenten in einer Form, die eine Wiederherstellung des Agenten ermöglicht, so daß dieser die Aus-

führung von dem alten Zustand ausgehend weiter fortsetzen kann. Die Komponentenarchitektur von CASA unterstützt die Persistenz von Agenten als eigenen Zustand des Lebenszyklus sowie durch die Ausführungskontrolle seitens des Kontrollzyklus und durch die Konfiguration anhand von Eigenschaften (vgl. Kapitel 3). Zur Persistenz wird ein Agent in den entsprechenden Lebenszykluszustand versetzt, woraufhin sämtliche Aktivitäten eingestellt werden, so daß sich der Agent in einem konstanten Zustand befindet. Dann werden über die Konfigurationsschnittstellen die momentanen Eigenschaften des Agenten und seiner Komponenten abgerufen und gespeichert. Anschließend kann der Agent beendet oder reaktiviert werden.

Um einen Agenten aus dem persistenten Zustand heraus erneut zu starten, wird ein Agent mit der gleichen Komponentenstruktur erzeugt und anhand der gespeicherten Eigenschaften reinitialisiert, bevor er wieder in den alten Lebenszykluszustand versetzt wird. Ein Agent unterstützt Persistenz also nur dann, wenn sämtliche seiner Komponenten sich anhand der konfigurierbaren Eigenschaften vollständig rekonstruieren lassen. Da dies auch eine Voraussetzung für den Austausch von Komponenten zur Laufzeit ist und diese auf denselben Architekturmechanismen wie die Persistenz beruht, läßt sich die Persistenz somit als eine Form des Komponentenaustauschs auffassen, bei der sämtliche Komponenten zugleich durch neue Instanzen ersetzt werden.

Auf den Mechanismen zur Persistenz bauen die Migration und das Klonen auf[39]. Bei der Migration wird ein Agent von einem Ausführungsort – i.a. eine AP – an einen anderen transferiert, wo er seine Tätigkeit fortsetzt. Das Klonen beinhaltet das Erzeugen und Starten von Kopien eines Agenten. In beiden Fällen wird zunächst die persistente Information des ursprünglichen Agenten extrahiert. Zur Migration wird diese gegebenenfalls zusammen mit dem Programmkode an einen anderen Ausführungsort übertragen, wo eine neue Instanz des Agenten erstellt wird, während die alte beendet wird. Beim Klonen werden anhand der persistenten Informationen neue Instanzen erstellt, die jeweils einen eigenen neuen Namen erhalten, wobei der ursprüngliche Agent erhalten bleibt.

Da das AMS für das Erzeugen und Beenden von Agenten verantwortlich ist, liegt es nahe, hier auch Funktionalitäten im Zusammenhang mit der Persistenz anzusiedeln. Gemäß der FIPA-Spezifikation ist die Migration als optionaler Dienst dem AMS zugeordnet, während einfache Persistenz und Klonen von dieser nicht explizit berücksichtigt werden.

---

[39] Es gibt auch Formen von Migration und Klonen, die nicht auf der Persistenz beruhen. Dabei werden lediglich funktional äquivalente Instanzen des ursprünglichen Agenten ohne Berücksichtigung des Ausführungszustands erzeugt.

5.2.3. Nutzung der Infrastrukturdienste

Relativ zu der in Kapitel 4 vorgeschlagenen Standardarchitektur für CASA-Agenten lassen sich diejenigen Komponentenrollen identifizieren, in deren Funktionalität die Nutzung der Infrastrukturdienste zu integrieren ist. Dies sind die Rollen, zu deren Aufgaben entweder die Verwaltung des Agenten und seiner Komponenten oder die Interaktion mit anderen Agenten über Dienste gehört. In letzterem Fall läßt sich zusätzlich zwischen dem Anbieten von Diensten, der Planung der Dienstnutzung sowie der Durchführung von Interaktionen unterscheiden. Da die meisten Infrastrukturfunktionalitäten über Dienste realisiert sind, lassen sich diese jedoch prinzipiell auch von Komponenten mit anderen Rollen nutzen, indem sie ein entsprechendes Ziel aufstellen, das zur Nutzung eines dieser Dienste führt.

Agentenverwaltung

Die Verwaltung des Agenten und seiner Komponenten geschieht durch den Agentenkern. Zudem ist dies die einzige Rolle, die in jedem Agenten permanent belegt sein muß, und damit die einzige, die die Identität des Agenten als ganzes repräsentieren kann. Die Nutzung der Funktionalitäten des AMS fällt somit in den Aufgabenbereich der Komponente mit der Rolle des Agentenkerns.

Üblicherweise übernimmt die Registrierung des Agenten beim AMS das AMS selbst, da es den betreffenden Agenten selbst erzeugt oder beendet, wovon die Migration nur ein Spezialfall ist, bei dem der Agent auf einer AP beendet und auf einer anderen im gleichen Ausführungszustand wieder erzeugt wird. Andernfalls muß der Agentenkern die Ankunft oder das Verlassen bei der jeweiligen AP selbst an- bzw. abmelden. Bei welchem AMS der Agent registriert werden soll und wie mit diesem kommuniziert werden kann, muß dabei dem Agentenkern anhand von dessen Konfiguration mitgeteilt werden.

Die Kommunikationsadressen des Agenten ergeben sich aus den Transportkomponenten, die er besitzt. Dabei wird davon ausgegangen, daß eine derartige Komponente die zugeordnete Adresse beim Übergang in den Lebenszykluszustand *initiated* festlegt und mindestens so lange beibehält, bis sie wieder in den Zustand *stopped* versetzt wird. Der Agentenkern kann somit die gültigen Adressen des Agenten als Eigenschaft der jeweiligen Transportkomponente abrufen (vgl. Kapitel 4.6.2), solange diese sich in einem der initialisierten Lebenszykluszustände befindet. Die Registrierung der Adressen obliegt dem Agentenkern, weil nur dieser die vollständige Registrierungsinformation über den Agenten als ganzes besitzt und dieser auch das Zusammensetzen des Agenten aus Komponenten vornimmt.

Anbieten von Diensten

Um einen Dienst anzubieten, muß dessen Beschreibung in der Dienstbibliothek des Agenten enthalten sein. Entsprechend ist auch die Komponente, die die Rolle der

Dienstbibliothek einnimmt, verantwortlich für die Registrierung der angebotenen Dienste beim DF.

Beim Start des Agenten werden zunächst sämtliche initial in der Dienstbibliothek enthaltenen Dienste beim DF angemeldet. Zur Laufzeit vorgenommene Änderungen an der Dienstbibliothek werden jeweils sofort dem DF mitgeteilt. Dabei sollte ein Dienst immer erst in die Dienstbibliothek eingefügt und dann beim DF angemeldet sowie umgekehrt erst beim DF abgemeldet und dann aus der Dienstbibliothek entfernt werden, so daß keine Dienste beim DF registriert sind, die der Agent noch nicht oder nicht mehr anbietet.

Ein Problem ergibt sich jedoch mit der Abmeldung der Dienste beim Beenden des Agenten. Da die Interaktion mit dem DF ausschließlich auf Diensten beruht, muß der Agent in einem aktiven Zustand sein, um sich per Dienstnutzung abmelden zu können. Andererseits besteht das Beenden des Agenten darin, daß er in den Lebenszykluszustand *undefined* zurückversetzt wird, so daß die Rolle der Dienstbibliothek die Abmeldung zu diesem Zeitpunkt nicht mehr vornehmen kann.

Dieses Problem läßt sich auf zwei Arten beheben. Zum einen kann die Terminierung des Agenten auf der Handlungsebene geschehen, deren letzter Schritt dann der Wechsel des Lebenszykluszustands ist, so daß zuvor noch eine Abmeldung beim DF per Dienstnutzung möglich ist. Dazu muß sich der Agent vor dem Beenden allerdings in einem aktiven Zustand befinden. Wird der Agent durch das AMS beendet, kann alternativ auch das AMS die Abmeldung beim DF vornehmen, da zur Abmeldung sämtlicher über einen Agenten im DF vorhandener Informationen nur der Agentenname benötigt wird.

Planen von Dienstnutzungen

Auf der Seite des Nutzers geschieht die Entscheidung über die Nutzung eines Dienstes im Verlauf der Handlungsauswahl. Dabei können jedoch nur diejenigen Dienste berücksichtigt werden, für die der Agent eine Dienstbeschreibung in Form eines Dienstoperators besitzt. Informationen über die verfügbaren Dienste erhält ein Agent üblicherweise über den DF, sie können ihm aber auch initial als Teil der Konfiguration mitgegeben werden.

Grundlage der Handlungsauswahl bilden die in der Planbibliothek enthaltenen Operatoren. Durchgeführt wird die Auswahl von Handlungen und damit die Entscheidung über die Nutzung von Diensten von den Rollen zur Handlungsauswahl und zur Plangenerierung. Entsprechend obliegt der Planbibliothek die Verwaltung der Dienstoperatoren, während die beiden anderen Rollen den Bedarf an nutzbaren Diensten in Abhängigkeit der gerade verfolgten Ziele feststellen.

Planbibliothek

Über die Konfiguration der Planbibliothek läßt sich deren Inhalt an Operatoren direkt verändern, wozu auch Dienstoperatoren gehören. Auf diese Weise kann der

Programmierer oder der Benutzer dem Agenten nutzbare Dienste mitteilen, sofern dieser über die Dienstbeschreibung verfügt. Um dynamisch auf das jeweils verfügbare Dienstangebot zuzugreifen, ist jedoch ein Rückgriff auf den Inhalt des DF vorzuziehen.

Dies braucht nicht ausschließlich anhand des aktuellen Bedarfs zu geschehen, sondern Dienste können auch antizipierend relativ zu bestimmten Domänen, in denen ein Agent Aufgaben zu erfüllen hat, beim DF erfragt werden, wenn er initialisiert wird oder seinen Aufenthaltsort wechselt. Eine Domäne wird dabei charakterisiert über Kategorien und Schlüsselworte, die in der Dienstbeschreibung als zusätzliche Parameter angegeben werden, oder auch anhand der Ontologien und Protokolle, die der Agent beherrscht. Auf diese Weise läßt sich die Zahl der Zugriffe auf den DF verringern und der Agent kennt im voraus besser seine Handlungsmöglichkeiten, wodurch sich aber auch der Aufwand zur Handlungsauswahl entsprechend erhöht.

Da sich das Dienstangebot jederzeit ändern kann, bedeutet das Vorhandensein eines Dienstoperators in der Planbibliothek noch nicht, daß es auch einen Anbieter für diesen Dienst gibt. Um den Bestand an Dienstoperatoren in der Planbibliothek aktuell zu halten, gibt es drei Möglichkeiten. Zum einen kann der DF über einen Dienst eine Überwachungsfunktionalität anbieten, bei der er dafür angemeldeten Agenten mitteilt, wenn keine Anbieter für einen Dienst, den sie von ihm erfahren haben, mehr registriert sind. Gerade bei Gesellschaften mit vielen Agenten bedeutet dies jedoch einen sehr hohen Verwaltungsaufwand, da der DF zentral sämtliche von ihm erteilten Auskünfte speichern und überwachen muß. Umgekehrt kann auch ein Agent in regelmäßigen Abständen beim DF nachfragen, ob für die in seiner Planbibliothek enthaltenen Dienste noch Anbieter existieren, was jedoch ein hohes Kommunikationsaufkommen mit sich bringt. Der einfachste Weg ist schließlich, Dienstoperatoren jeweils dann aus der Planbibliothek zu entfernen, wenn bei einer versuchten Dienstnutzung kein Anbieter mehr vorhanden war.

Handlungsauswahl und Plangenerierung

Die Komponenten zur Handlungsauswahl und Plangenerierung verwenden die in der Planbibliothek enthaltenen Operatoren, um Handlungen auszuwählen, deren Ausführung ein vorgegebenes Ziel erreicht. Falls keine Handlung gefunden wird oder noch nach besseren Handlungsalternativen gesucht werden soll, kann das Ziel für eine Anfrage an den DF verwendet werden, um Dienste zum Erreichen dieses Ziels zu erfahren. Gegebenenfalls wird dann nach den üblichen Mechanismen zur Handlungsauswahl der geeignetste Dienst aus den gefundenen Dienstoperatoren ausgewählt.

Im Verlauf der Plangenerierung treten i.a. wiederholt Situationen auf, in denen mit den vorhandenen Operatoren keine oder keine optimale Lösung gefunden werden kann oder der eingeschlagene Lösungsweg nicht fortgesetzt werden kann, sondern an früherer Stelle anders fortgeführt werden muß (Backtracking). Ist ein

Agent Teil einer Agentengesellschaft, so verfügt er jedoch abgesehen von den ihm explizit bekannten Handlungsoptionen zusätzlich über die Gesamtheit aller Dienste, die andere Agenten zur Nutzung anbieten. Es ist aber in den meisten Fällen weder praktikabel, im voraus sämtliche vorhandenen Dienste aus dem DF abzurufen, noch in jedem Planschritt dort jeweils nach geeigneten Diensten zu suchen.

Diese beiden Möglichkeiten zur Erweiterung der Handlungsoptionen anhand des Dienstangebots müssen jeweils in einer Weise eingeschränkt werden, die den Aufwand zur Planung wie zur Interaktion mit dem DF möglichst gering hält. Da der Suchraum mit der Zahl der Operatoren wächst, sollten vor der Planung nur Dienste für solche Bereiche aus dem DF abgerufen werden, die für das jeweilige Ziel relevant sind und für die noch keine Anfrage an den DF gestellt wurde. Während der Planerzeugung läßt sich die Menge der Anfragen an den DF reduzieren, indem nur dann nach weiteren Diensten gesucht wird, wenn eine erfolgreiche Planung anders nicht möglich ist, und nur nach solchen Diensten, die außerhalb des bereits im voraus abgefragten Bereichs liegen.

Die einmal vom DF bezogenen Dienstbeschreibungen sollten in der Planbibliothek abgelegt werden, so daß sie erneut zur Handlungsauswahl verwendet werden können.

Suche nach Diensten

Der Zugriff auf das durch den DF verwaltete Dienstangebot geschieht durch die Dienste, die der DF selbst anbietet. Diese Dienste müssen dem Agenten im voraus bekannt sein, da sie die Voraussetzung für den Zugriff auf den DF darstellen und deshalb nicht selbst über den DF abgerufen werden können.

Die Suche nach Diensten kann entweder anhand eines bestimmten Ziels oder für eine gesamte Domäne geschehen. Im ersten Fall sucht der DF nach verfügbaren Dienstoperatoren, deren Effekt das Ziel erfüllen kann. Dies geschieht im Prinzip auf dieselbe Weise wie bei der Handlungsauswahl für ein Ziel (vgl. Kapitel 4.5.5), außer daß der DF die Vorbedingungen nicht prüfen kann, da er nur das Ziel, aber nicht das Faktenwissen des anfragenden Agenten kennt. Zur domänenspezifischen Suche können die Dienste in ein Kategoriensystem eingeordnet oder durch Schlüsselworte charakterisiert werden. Alternativ lassen sich hierfür auch die in der Dienstspezifikation angegebenen Ontologien als Suchkriterium verwenden, da eine Ontologie jeweils ein Wissensrepräsentationsschema für einen bestimmten Bereich vorgibt, zu dem in der Regel gerade die Dienste gehören, die die entsprechende Ontologie in ihrer Beschreibung verwenden. Die Verwendung der Ontologien zur Angabe des Suchbereichs hat zudem den Vorteil, daß die Wissensrepräsentation der Agenten auf den Ontologien beruht und diese so eine Zuordnung von Wissen zu einer Domäne selbst vornehmen können.

Um die Suche beim DF auf Dienste zu beschränken, die der anfragende Agent auch nutzen kann, kann dieser zusätzlich die von ihm verarbeitbaren Inhaltsspra-

chen und Ontologien angeben sowie die Protokolle, die er in der Rolle des Nutzers durchführen kann. Weiterhin besteht die Möglichkeit, eine Dienstsuche nicht nur auf einen DF zu beschränken, sondern auch bei DFs auf anderen APs eine Anfrage zu stellen. Dies kann direkt durch den Agenten selbst oder indirekt durch den DF geschehen, dem dazu die Zahl oder Liste an anderen APs angegeben werden muß.

Durchführung von Interaktionen

Die zentrale steuernde Komponente zur Durchführung von Interaktionen ist die in der Rolle der Kommunikation. Sie nutzt die Infrastrukturdienste zur Suche nach Interaktionspartnern beim DF sowie zur Kommunikationsunterstützung durch das AMS und den ACC.

Anbietersuche

Bei der oben beschriebenen Planung von Dienstnutzungen findet üblicherweise noch keine Festlegung auf einen bestimmten Anbieter des ausgewählten Dienstes statt. Dies ist – wie in Kapitel 5.1.3 beschrieben – Aufgabe der Kommunikationsrolle und geschieht vor der Dienstnutzung oder in der Auswahlphase des Dienst-Metaprotokolls.

In jedem Fall muß die Kommunikation zu Beginn einer Dienstnutzung einen oder mehrere Anbieter auswählen, an die eine Anfrage zur Diensterbringung gestellt werden soll. Welche Agenten einen zu nutzenden Dienst anbieten, erfährt die Kommunikation durch eine entsprechende Dienstnutzung beim DF. Wie die Suche nach Diensten kann auch die Anbietersuche dabei auf eine oder einige bestimmte APs beschränkt werden. Für weitere Dienstnutzungen lassen sich diese Informationen zur Wiederverwendung in der Faktenbasis ablegen und nur bei Bedarf über eine erneute Anfrage an den DF aktualisieren.

Aktualisierung von Adressen

Zur Kommunikation benötigt ein Agent die Kommunikationsadressen seines Kommunikationspartners. Diese kann er vom AMS seiner momentanen AP über eine Dienstnutzung erfahren, wobei das AMS gegebenenfalls auf das AMS der HAP des anderen Agenten zurückgreift, falls es selbst keine entsprechenden Informationen besitzt.

Um nicht für jeden einzelnen Sprechakt eine Dienstnutzung beim AMS durchzuführen, werden die Adressen in der Faktenbasis gespeichert. Eine Anfrage an das AMS ist nur dann nötig, wenn die Adressen nicht bekannt sind oder mit den gespeicherten Adressen keine Kommunikation möglich ist, so daß davon ausgegangen werden kann, daß sich diese geändert haben.

Die Adresse des AMS selbst ist unveränderlich und ergibt sich per Konvention aus der Adresse der jeweiligen AP, so daß diese Kommunikationsadresse nicht über eine Anfrage beim AMS in Erfahrung gebracht zu werden braucht.

Nutzung des ACC

Unter Umständen ist keine direkte Kommunikation zwischen zwei Agenten möglich. Dies ist der Fall, wenn unter den Adressen des Kommunikationspartners keine mit einem Protokoll ist, für das der Agent selbst eine Transportkomponente besitzt, oder wenn die gemeinsamen Kommunikationsprotokolle nur innerhalb einer AP verwendet werden können und sich die Agenten auf verschiedenen APs befinden.

Falls auch nach einer Aktualisierung der Adressen keine direkte Kommunikationsmöglichkeit besteht, ist eine Kommunikation immer noch über den ACC möglich. Dazu nutzt der sendende Agent den Dienst zum Weiterleiten von Sprechakten des ACC und übergibt dabei den zu versendenden Sprechakt, dessen Zustellung dann der ACC übernimmt.

5.3. Zusammenfassung

Innerhalb einer Agentengesellschaft erhalten die Agenten die Möglichkeit zu dynamischen Interaktionen untereinander. Konstituiert wird die Agentengesellschaft durch eine Infrastruktur, die als Bindeglied zwischen den Agenten dient und die Gelegenheiten zur Interaktion vermittelt. Sie organisiert und verwaltet die Agentengesellschaft und gewährleistet deren Offenheit und Skalierbarkeit. Um diese Möglichkeiten der Agentengesellschaft nutzen zu können, benötigen Agenten die Fähigkeit zur Interaktion. Dies umfaßt zum einen einheitliche Verfahren zur Interaktion, um die Interoperabilität zwischen den Agenten sicherzustellen. Zum anderen sind die Durchführung von Interaktionen und die Nutzung der Infrastrukturfunktionalitäten so in die vorgeschlagene Kontrollarchitektur integriert, daß sie sich nahtlos in das Gesamtschema zur Verhaltenssteuerung einfügen.

Zentrale Voraussetzung für Interaktionen in technischen Systemen ist die Interoperabilität der beteiligten Teilsysteme, ohne die keine gemeinsame Grundlage für eine gegenseitige Kompatibilität gegeben ist. Erreicht wird dies in CASA durch Standardisierungen auf verschiedenen Ebenen, die entweder ein festes Schema oder den Rahmen zur Definition anwendungsspezifischer Standards vorgeben. Bei Agenten bestehen Interaktionen vorwiegend in der Kommunikation, in der formales Wissen ausgetauscht wird. Dieses Wissen wird in einer einheitlichen Inhaltssprache ausgedrückt und über eine gemeinsame Kommunikationssprache ausgetauscht, wobei das verwendete Vokabular über bereichsspezifische Ontologien definiert wird. Interaktionsverläufe werden durch Protokolle vorstrukturiert, die den Raum

der möglichen Kommunikation einschränken. Zusammen sorgen diese Konventionen zu einer übereinstimmenden Syntax und Semantik bei der Erstellung und Verarbeitung von kommunikativen Handlungen bei verschiedenen, voneinander unabhängig spezifizierten Agenten. Dabei orientiert sich CASA an etablierten Techniken und bestehenden Standards aus diesem Bereich.

Zusätzlich ist in CASA auch die Beschreibung und Durchführung von Interaktionen vereinheitlicht. Diese werden dazu als Dienste aufgefaßt, bei denen ein Agent eine Handlung für einen anderen ausführt. Der Nutzer des Dienstes delegiert sein Ziel an den Anbieter, der dieses für ihn zu erreichen sucht. Den Rahmen aller Interaktionen bildet dabei ein Metaprotokoll, das allgemeine Aspekte von Dienstnutzungen abdeckt, während eingebettete Protokolle die spezifische Parameteraushandlung und Diensterbringung übernehmen.

Interaktionen sind in die Mechanismen der Kontrollarchitektur integriert, indem Dienste und die Umsetzung von Protokollen für eine Rolle jeweils durch Operatoren repräsentiert werden. Dienstoperatoren beschreiben die Parameter eines Dienstes sowie die Bedingungen und Effekte aus der Sicht des Nutzers. Dadurch kann ein Agent Dienste auf dieselbe Weise wie andere Handlungen auswählen und kombinieren, während lediglich die Ausführung von einem anderen Agenten übernommen wird. Protokolloperatoren nutzen die Möglichkeiten zur Ablaufkontrolle von Plänen und Skripten, die noch um Sprechaktoperatoren für die Kommunikation erweitert werden.

Gemäß dem Dienstkonzept berücksichtigt das Dienst-Metaprotokoll genau zwei Rollen, den Anbieter und den Nutzer des Dienstes. Es besteht aus den drei Phasen Initiierung, Auswahl und Diensterbringung. Der Nutzer initiiert das Metaprotokoll bei der Ausführung eines Dienstoperators mit einer Anfrage an einen oder mehrere Anbieter. Anhand der Aushandlung generischer Parameter entscheidet der Anbieter dann über die Annahme dieser Anfrage. In der Auswahlphase handelt der Nutzer mit den Anbietern, die sich zur Diensterbringung bereiterklärt haben, die spezifischen Konditionen aus. Auf dieser Grundlage wählt er einen Anbieter aus und erteilt ihm einen entsprechenden Auftrag. Es folgt die eigentliche Erbringung der Dienstleistung durch den Anbieter, die mit der Übermittlung des Ergebnisses an den Nutzer endet. Dies ist dann zugleich das Ergebnis der Ausführung des Dienstoperators. Für die beiden letzten Phasen können eingebettete Protokolle eingesetzt werden, um Aushandlung und Diensterbringung an den jeweiligen Dienst anzupassen.

Die vorgenommene Skizze einer Infrastruktur für Gesellschaften von CASA-Agenten orientiert sich an der Management-Spezifikation der FIPA. Zum einen bietet sie ein umfassendes und ausgereiftes Modell, zum anderen erlaubt dies eine Kompatibilität auch über Systemgrenzen hinweg. Das Kernkonzept der Spezifikation ist die Agentenplattform (AP), auf der Agenten angesiedelt sind und die diesen ihre Infrastrukturfunktionalität zur Verfügung stellt. Realisiert wird eine AP durch eine Menge von ausgezeichneten Agenten und deren Diensten. Mehrere APs können

über eine gegenseitige Anmeldung ihrer Infrastrukturdienste miteinander verbunden sein.

Dem Agent Management System (AMS) obliegt die Verwaltung der AP. Mit ihm lassen sich Agenten neu erzeugen oder terminieren, ihre Ausführung suspendieren und wiederaufnehmen sowie zusätzliche verwandte Funktionalitäten wie Persistenz und Migration nutzen. Da das AMS für die Existenz von Agenten zuständig ist, informiert es auch über den Aufenthaltsort von Agenten und über deren Kommunikationsadressen. Die Durchführung der Kommunikation wird durch das Message Transport System (MTS) unterstützt. Neben dem reinen Transport übernimmt es auch die Weiterleitung oder Aufbewahrung, falls der Adressat seinen Aufenthaltsort gewechselt hat oder nicht erreichbar ist. Die Vermittlung von Diensten ist die Aufgabe des Directory Facilitator (DF). Es führt ein Verzeichnis mit den verfügbaren Diensten und ihren Anbietern, bei dem sich die Anbieter registrieren und in dem die Nutzer suchen können. Zur Integration in die Standardarchitektur wurden diejenigen Komponentenrollen identifiziert, deren Funktionalität von der Nutzung einzelner Infrastrukturdienste abhängt oder profitiert.

Die Grundlage aller Interaktionen zwischen Agenten bildet das Dienstkonzept, das Verhaltenskontrolle, Kommunikation und Infrastruktur miteinander verbindet. Spezifiziert sind die Dienste über Operatoren aus der Sicht des Nutzers, umgesetzt werden sie über Kommunikation nach flexiblen Protokollen, und die Infrastruktur bringt Anbieter und Nutzer von Diensten zusammen. Diese Integration von Interaktionen in die Verhaltenssteuerung der Agenten erlaubt eine flexible Auswahl und Kombination von Diensten. Wie bei der Komponentenarchitektur sind auch auf der Multiagentenebene die Interaktionspartner möglichst weitgehend voneinander entkoppelt. Agenten nutzen dynamisch das vorhandene Dienstangebot, das sich beständig ändern kann, wenn Agenten hinzugefügt, entfernt oder ausgetauscht werden oder ihre Angebotspalette ändern. Statische Abhängigkeiten bestehen nur zu den Infrastrukturdiensten als die Grundlage der Offenheit und Flexibilität der Agentengesellschaft. Die allgemeinen Fähigkeiten, Dienste anzubieten oder zu nutzen, sind ausreichend, um einen spezifischen Dienst zu nutzen, sofern beide Agenten über ein entsprechendes gemeinsames Domänenwissen verfügen.

6. Wissensrepräsentation

Eine wissensbasierte Verhaltenssteuerung von Agenten (vgl. Kapitel 4.1) basiert auf einer expliziten Repräsentation formalen Wissens als Grundlage zur Steuerung des Verhaltens eines Agenten. Nachfolgend wird die Wissensrepräsentationssprache CASA Agent Language (CAL) definiert, die agentenspezifische Aspekte berücksichtigt und insbesondere an den Aufbau der in Kapitel 4 beschriebenen Standardarchitektur angepaßt ist. Neben der Verhaltenssteuerung umfaßt diese Sprache auch die Inhalte der Kommunikation zwischen Agenten und dient als logische Beschreibungsebene sowohl zur Programmierung, als auch zur Analyse von Agenten und deren Verhalten.

Die Wissensrepräsentationssprache umfaßt verschiedene Sprachebenen und Inhaltstypen, die ineinander übergreifen und aufeinander aufbauen, so daß die unterschiedlichen Teile kompatibel sind und ein kohärentes Ganzes bilden. Die gemeinsame Grundlage bildet eine formale Logik, die die Prädikatenlogik erster Stufe um einen dritten Wahrheitswert erweitert, um die subjektive Sicht eines Agenten und die Unvollständigkeit seines Wissens zu reflektieren. Eine einheitliche Terminologie zur Repräsentation und Kommunikation von Wissen definieren Ontologien in Form von Kategorien und ihren Attributen sowie über Funktionen und Vergleiche. Das deklarative Wissen zur Beschreibung von Weltzuständen und mentalen Zuständen umfaßt Fakten, Ziele und Intentionen. Der Repräsentationsformalismus für faktisches Wissen kombiniert dabei logische und objektorientierte Ansätze. Reaktives Wissen besteht in Reaktionsregeln, die Handlungsdispositionen für bestimmte Situationen ausdrücken. Das prozedurale Wissen beschreibt die Handlungsmöglichkeiten eines Agenten zur Veränderung des Weltzustands. Es besteht aus einer abstrakten, deklarativen Beschreibung der Handlung und ihrer logischen oder prozeduralen Umsetzung. Schließlich umfaßt das kommunikative Wissen neben dem kommunizierten deklarativen Inhalt einerseits Informationen zur intendierten Verarbeitung durch den Empfänger und andererseits Informationen für den Transport zum Empfänger und die Durchführung von Konversationen.

Gemäß diesen verschiedenen Sprachaspekten gliedert sich das nachfolgende Kapitel. Es beginnt mit der Semantik der ternären Logik, wobei nur auf die Besonderheiten der Dreiwertigkeit eingegangen wird, da sowohl die Prädikatenlogik, als auch andere in der Wissensrepräsentationssprache verwendete Erweiterungen wie die Typisierung von Termen hinreichend bekannt sind. Anschließend wird die

Grammatik von CAL definiert, beginnend bei dem teminologischen und logischen Anteil, gefolgt von dem deklarativen, prozeduralen und kommunikativen Teil. Zuvor wird noch kurz die zur Definition der Grammatik verwendete Notation eingeführt.

Notation

Die Notation der Grammatik von CAL erfolgt in Anlehnung an BNF mit folgenden Konventionen:

- Zeichen:
  - `normal` : terminales Zeichen
  - *`kursiv`* : definierter Ausdruck
  - **`fett`** : Element eines Typs
  - ***`fett&kursiv`*** : Symbol
  - `unterstrichen` : optional
- Kennzeichnung von Wiederholungen:
  - …* : beliebige Anzahl, auch 0
  - …+ : beliebige Anzahl, mindestens 1
  - …++ : beliebige Anzahl, mindestens 2

6.1. Ternäre Logik

Die klassische Prädikatenlogik ist zweiwertig, d.h. jede Aussage ist entweder wahr oder falsch. Dies liegt nicht zuletzt an ihrer ursprünglichen Bestimmung zur Formalisierung der Mathematik, in der keine Ungenauigkeit oder Unbestimmtheit vorkommen sollte und somit jede Aussage nur entweder wahr oder falsch sein durfte. Andererseits ist bereits die Metalogik der Prädikatenlogik dreiwertig, da jede Aussage relativ zu einer Menge an Axiomen als wahr oder falsch beweisbar sowie nicht entscheidbar sein kann. Mit der Übertragung auf andere Bereiche wie dem menschlichen Denken oder der Entwicklung von „künstlich intelligenten“ Computerprogrammen zeigt sich jedoch die Beschränkung dieser und anderer Voraussetzungen der Prädikatenlogik, sobald unvollständiges, ungenaues oder auch teilweise inkonsistentes Wissen behandelt werden muß. In der Philosophie wie in der KI sind deshalb viele andere formale Logiken[40] entwickelt worden für Bereiche, in denen die Exaktheit, Geschlossenheit und Unveränderlichkeit der Mathematik nicht gegeben

[40] Einen Überblick über nicht-klassische Logiken aus philosophischer Sicht gibt [Gabbay & Guenthner 1986].

ist. Aufgrund der gewonnenen Ausdrucksstärke sind einige dieser Logiken jedoch schwer zu handhaben, zumal falls keine vollständigen Entscheidungsverfahren existieren[41].

Das zentrale Motiv für die Verwendung formal repräsentierter Wissensstrukturen in CASA ist nicht alleine die dadurch gewonnene Autonomie und Flexibilität der Agenten, sondern vor allem die Automatisierung von Interaktionen durch deren formale Spezifikation. Deshalb wurde einerseits versucht, die Semantik der Wissensrepräsentationssprache bei hinreichender Ausdrucksstärke möglichst einfach zu halten, um eine effiziente Verarbeitung zu ermöglichen, andererseits aber dennoch agentenspezifische Anforderungen zu berücksichtigen. Die zugrundeliegende Semantik ist dabei eine Erweiterung der Prädikatenlogik erster Stufe, in der die subjektive Sicht eines Agenten und dessen begrenztes Wissen über die Welt berücksichtigt wird.

Dazu wird neben den klassischen Wahrheitswerten *wahr* und *falsch* zusätzlich der Wahrheitswert *unbekannt* eingeführt. Diese Wahrheitswerte werden nicht mehr als absolut und objektiv, sondern als revidierbare, subjektive Annahmen des jeweiligen Agenten aufgefaßt. Die klassischen Verknüpfungen werden für den hinzugekommenen Wahrheitswert seiner Bedeutung gemäß als nicht Vorhandensein einer konkreten Annahme interpretiert und entsprechend erweitert, während sie für die klassischen Wahrheitswerte unverändert bleiben. In gewisser Weise ist *unbekannt* also kein Wahrheitswert, sondern das Fehlen eines Wahrheitswerts und besitzt einen nur vorläufigen Status.

Da ein unbekannter Wert bei Bekanntwerden sowohl wahr als auch falsch werden kann, werden die Verknüpfungen so erweitert, daß eine derartige Änderung keine bereits bestehenden Annahmen ändern, sondern nur zu neuen Annahmen führen kann. Die Semantik erhält somit eine gewisse Monotonie aufrecht hinsichtlich der Änderung von Unbekanntem, da dies nur zu einer Revision von solchen Aussagen führen kann, für die noch keine Annahmen bestanden, aber nicht von solchen, für die bereits eine Annahme vorhanden war. Diese Interpretation geht also in gewisser Weise davon aus, daß Unwissenheit häufiger ist als Irrtum und deshalb eine entsprechende Änderung des Wissens mit geringerem Aufwand zu bewältigen sein muß.

6.1.1. Wissen und Welt

Das Verhältnis zwischen dem Wissen eines Agenten und der ihn umgebenden Welt läßt sich als das zwischen zwei Interpretationen oder formalen Modellen derselben Annahmen auffassen. Die Welt wird dabei in Analogie zu einem „Agenten“ gese-

[41] Bereits die Prädikatenlogik ist nur semi-entscheidbar, d.h. es existiert ein Verfahren, daß sämtliche beweisbaren Aussagen beweisen kann, daß aber nicht in jedem Fall die Unbeweisbarkeit einer Aussage feststellen kann.

Agent \ Welt	wahr	falsch
als wahr angenommen	Wissen	Irrtum
keine Annahme	Unwissen	Unwissen
als falsch angenommen	Irrtum	Wissen

Abbildung 21: Beziehung zwischen Wissen und Welt

hen, der dieselbe Terminologie und Logik verwendet wie der betreffende Agent, dabei „allwissend“ und „unfehlbar“, aber „unkommunikativ“ ist. Die Allwissenheit beinhaltet vollständiges Wissen über alle ausdrückbaren Sachverhalte und impliziert somit die Zweiwertigkeit der Welt, in der der Wahrheitswert unbekannt nicht vorkommt. Die Unfehlbarkeit macht die Welt zu einem Referenzpunkt für Wissen und Irrtum für den Agenten, indem bei einer Abweichung zwischen den Annahmen des Agenten und der Welt immer letztere im Recht ist. Schließlich ist die Welt unkommunikativ in dem Sinne, daß ein Agent die von dieser „gewußten“ Wahrheiten nicht direkt erfahren kann, sondern nur partiell und indirekt auf diese schließen kann aufgrund seines Wahrnehmens und Handelns in der Welt.

Aus dieser objektiven Perspektive, die Welt und Agenten von außen betrachtet, beinhaltet der Begriff Wissen nun eine Art Korrespondenz zwischen der Welt und den Annahmen des Agenten. Somit beinhaltet die Übereinstimmung von Annahme und Welt Wissen, deren Verschiedenheit Irrtum und das Fehlen einer Annahme Unwissen (Abbildung 21). Es sei an dieser Stelle noch darauf hingewiesen, daß wenn an anderen Stellen in diesem Text von Wissen im Zusammenhang mit Agenten die Rede ist, die subjektive Sicht des Agenten gemeint ist, die keinen uneingeschränkten Wahrheitsanspruch beinhaltet. Gemäß einer gelegentlich zur expliziten Abgrenzung verwendeten Diktion handelt es sich dann also nur um Glaube oder Meinung, die von Wissen als „wahrer, gerechtfertigter Meinung“ unterschieden wird.

Dieses Verhältnis zwischen den Annahmen des Agenten und den Tatsachen der Welt besteht nur von der Außenperspektive her, sie ist nicht in der Logik selbst enthalten, die auf die subjektive Sicht des Agenten beschränkt ist. Der Agent handelt gemäß seiner Annahmen und ihrer logischen Folgerungen und ändert seine Annahmen gemäß seiner Wahrnehmung, aber er macht dies ohne expliziten logischen Bezug auf die Welt, die seine Annahmen zu Wissen oder Irrtum macht. Die Welt als etwas von den Annahmen des Agenten Unterschiedenes und die Bezugnahme der Annahmen auf diese Welt kommen in der Logik selbst nicht vor, sondern nur in deren Interpretation. Anschaulicher ausgedrückt beinhaltet dies, daß eine Aussage wie beispielsweise „Ich halte X für wahr, es könnte in Wirklichkeit aber auch falsch sein.“ außerhalb des logischen Horizonts des Agenten liegt.

6.1.2. Syntax und Semantik

Für dreiwertige (und andere mehrwertige) Logiken gibt es verschiedene Ansätze zur Definition der logischen Verknüpfungen (vgl. [Urquhart 1986]), die jeweils aus der intendierten Bedeutung der Wahrheitswerte heraus motiviert sind. Die meisten dieser Logiken sind Erweiterungen der klassischen zweiwertigen Logik, so daß die Verknüpfungen eingeschränkt auf die Wahrheitswerte *wahr* und *falsch* der klassischen Definition entsprechen.

Für eine sehr ähnliche Interpretation des dritten Wahrheitswerts, wie sie hier für das unvollständige Wissen von Agenten vorgeschlagen wird, hat Kleene [1938, 1952] zwei dreiwertige Logiken definiert, in denen der zusätzliche Wahrheitswert als *undefiniert* aufgefaßt wird. Er unterscheidet dabei eine schwache und eine starke Variante. In der schwachen Variante führt jedes Vorkommen von *undefiniert* in einer Aussage dazu, daß die gesamte Aussage *undefiniert* ist. Die starke Variante erweitert die logischen Verknüpfungen so, daß die Wahrheitswerte *wahr* und *falsch* dann zugewiesen werden, wenn eine Änderung enthaltener undefinierter Teilaussagen darauf keinen Einfluß hat: „*u* means only the absence of information that Q(x) is *t* or *f*.“ [Kleene 1952] (*u*, *t*, *f* sind die Wahrheitswerte *undefined*, *true*, *false* und Q(x) steht für eine beliebige Aussage). Die Motivation für diese Interpretation ist die Formalisierung von Aussagen über partiell definierte rekursive Funktionen. Kleene diskutiert dabei aber auch die alternative Interpretation des dritten Wahrheitswerts als *unbekannt* (*unknown*) sowie die Relativierung aller drei Wahrheitswerte auf deren Bekanntheit, wobei *wahr* zu *als wahr bekannt* (*known to be true*), *falsch* zu *als falsch bekannt* (*known to be false*) und *unbekannt* zu *unbekannt ob wahr oder falsch* (*unknown whether true or false*) wird. Letzteres entspricht gerade der Anwendung dieser Logik auf vorläufiges und unvollständiges Wissen. Die nachfolgend definierte Semantik ist deshalb die einer starken, dreiwertigen Logik nach Kleene.

Syntax

Da die dreiwertige Logik eine reine Erweiterung der Prädikatenlogik erster Stufe darstellt, kann von dieser die Syntax übernommen werden.

Definition 21: Zeichen der ternären Logik

Die Menge $Z = W \cup V_1 \cup V_2 \cup V_q \cup H \cup V$ der Zeichen der ternären Logik besteht aus der Vereinigung der untereinander disjunkten Mengen $W = \{ \mathtt{T}\ \mathtt{0}\ \bot \}$ der Wahrheitswertsymbole, der Mengen $V_1 = \{ \neg \sim \uparrow \downarrow \}$, $V_2 = \{ \wedge \vee \rightarrow \equiv \}$ und $V_q = \{ \forall\ \exists \}$ der einstelligen und zweistelligen Verknüpfungssymbole sowie der Quantorsymbole, der Menge $H = \{ (\ ) , \}$ der Hilfssymbole sowie einer unendlichen Menge V an Variablensymbolen.

Bei den Wahrheitswertsymbolen steht $\mathtt{T}$ für *wahr*, $\mathtt{0}$ für *unbekannt* und $\bot$ für *falsch*. Die Bedeutung der Verknüpfungssymbole wird in der nachfolgenden Semantik

definiert. Die Hilfssymbole sind öffnende und schließende Klammern sowie das Komma.

Definition 22: Signatur für eine ternäre Logik

Eine Menge $S = \cup_{n \in N} (F_{n-1} \cup R_n)$ ist eine Signatur einer ternären Logik, wobei sämtliche der möglicherweise leeren Mengen aus F_i und R_j untereinander disjunkt sind sowie $Z \cap S = \varnothing$. F_n ist jeweils eine Menge an n-stelligen Funktionssymbolen und R_n eine Menge an n-stelligen Relationssymbolen.

Eine Signatur definiert Symbole für denotierende Elemente der Logik, denen im Sinne einer modelltheoretischen Interpretation über eine Abbildung auf einen Gegenstandsbereich eine Bedeutung zugewiesen wird. Die Elemente von F_0 sind Funktionen ohne Parameter und werden als Konstanten bezeichnet.

Definition 23: Terme der ternären Logik

Ein Ausdruck t zu einer Signatur S ist ein Term, wenn er durch endliche Anwendungen folgender Regeln gebildet werden kann:

- Variable: $t \in V$
- Konstante: $t \in F_0$
- Funktionsausdruck: $t = f(t_1, \ldots, t_n)$ mit $n > 0$, $f \in F_n$; $t_1, \ldots, t_n$ Terme $\neq t$

Definition 24: Formeln der ternären Logik

Ein Ausdruck φ zu einer Signatur S ist eine Formel, wenn er durch endliche Anwendungen folgender Regeln gebildet werden kann:

- Relationsausdruck: $\varphi = r(t_1, \ldots, t_n)$ mit $n > 0$, $r \in R_n$; $t_1, \ldots, t_n$ Terme
- einstellige Verknüpfung: $\varphi = \times\varphi'$ mit $\times \in V_1$, φ' Formel $\neq \varphi$
- zweistellige Verknüpfung: $\varphi = \varphi_1 \times \varphi_2$ mit $\times \in V_2$; φ_1, φ_2 Formeln $\neq \varphi$
- Quantor: $\varphi = \times x\, \varphi'$ mit $\times \in V_q$, $x \in V$, φ' Formel $\neq \varphi$

Eine atomare Formel ist eine nicht zerlegbare Formel, deren Definition nicht auf einer anderen Formel beruht. Gemäß Definition 24 sind dies nur die Relationsausdrücke.

Semantik

Für die ternäre Logik ist die Semantik der Verknüpfungen durch die untenstehenden Wahrheitstabellen (Definition 25) und die der Quantoren durch die untenstehende Definition (Definition 26) gegeben. Im folgenden werden für Formeln als abkürzende Bezeichnung die Buchstaben A, B und C verwendet, für Formeln mit einer Variablen x wird A(x) verwendet.

Definition 25: Wahrheitstabellen für logische Verknüpfungen der ternären Logik

$\neg$	
$\bot$	T
0	0
T	$\bot$

~	
$\bot$	T
0	$\bot$
T	T

$\wedge$	$\bot$	0	T
$\bot$	$\bot$	$\bot$	$\bot$
0	$\bot$	0	0
T	$\bot$	0	T

$\vee$	$\bot$	0	T
$\bot$	$\bot$	0	T
0	0	0	T
T	T	T	T

$\uparrow$	
$\bot$	$\bot$
0	$\bot$
T	T

$\downarrow$	
$\bot$	T
0	$\bot$
T	$\bot$

$\rightarrow$	$\bot$	0	T
$\bot$	T	T	T
0	0	0	T
T	$\bot$	0	T

$\equiv$	$\bot$	0	T
$\bot$	T	0	$\bot$
0	0	0	0
T	$\bot$	0	T

Definition 26: Quantoren der ternären Logik

	$\forall$x A(x)	$\exists$x A(x)
T	für alle x ist A(x) = T	es gibt ein x mit A(x) = T
$\bot$	es gibt ein x mit A(x) = $\bot$	für alle x ist A(x) = $\bot$
0	sonst	sonst

Zur Definition der Semantik sind die Verknüpfungen *nicht* ($\neg$), *bekannt* (~) und *und* ($\wedge$) sowie der Quantor *alle* ($\forall$) ausreichend. Die übrigen lassen sich aus diesen wie folgt definieren:

- *bekannt-wahr*: $\uparrow A \cong (\sim A \wedge A)$
- *bekannt-falsch*: $\downarrow A \cong (\sim A \wedge \neg A)$
- *oder*: $A \vee B \cong \neg(\neg A \wedge \neg B)$
- *impliziert*: $A \rightarrow B \cong \neg A \vee B$
- *äquivalent*: $A \equiv B \cong (A \wedge B) \vee (\neg A \wedge \neg B)$
- *existiert*: $\exists x\, A(x) \cong \neg(\forall x\, \neg A(x))$

Die Quantoren lassen sich wie folgt auf die Annahmen des Agenten beschränken:

- *bekannt-alle*: $\tilde{\forall} x\, A(x) \cong \forall x\, (\sim A(x) \rightarrow A(x))$
- *bekannt-existiert*: $\tilde{\exists} x\, A(x) \cong \exists x\, (\sim A(x) \wedge A(x))$

Wie aus den Definitionen leicht ersichtlich ist, stellt diese Semantik eine echte Erweiterung der Prädikatenlogik erster Stufe dar. Beschränkt auf die Wahrheitswerte *wahr* und *falsch* stimmen sämtliche Definitionen mit den klassischen überein. Die einstelligen Verknüpfungen *bekannt*, *bekannt-wahr* und *bekannt-falsch* kommen zu denen der klassischen Logik hinzu und erlauben als einzige einen sicheren Übergang zur Zweiwertigkeit, da sie nie den Wert *unbekannt* ergeben. Die Menge der hier definierten Verknüpfungen ist nicht funktional vollständig, da es keine Möglichkeit gibt, durch Verknüpfung bekannter Wahrheitswerte den Wahrheitswert *unbekannt* zu erhalten. Aufgrund der intendierten Interpretation ist dies auch nicht sehr sinnvoll, ließe sich jedoch leicht durch eine zusätzliche einstellige Verknüpfung beheben, die in jedem Fall den Wert *unbekannt* zuweist.

Wie für die Prädikatenlogik läßt sich auch für die ternäre Logik eine modelltheoretische Interpretation von Formeln geben. Eine Interpretation im Sinne der Modelltheorie ist eine Abbildung der Symbole der Signatur auf einen Gegenstandsbereich.

Ein Modell für eine Formel ist eine Interpretation, die der Formel den Wahrheitswert *wahr* zuordnet.

Definition 27: Erfüllbarkeit von Formeln in der ternären Logik

- Eine Formel φ ist allgemeingültig, wenn sie unter jeder Interpretation *wahr* ist.
- Eine Formel φ ist erfüllbar, wenn sie unter einer Interpretation *wahr* ist.
- Eine Formel φ ist nicht erfüllbar, wenn sie unter keiner Interpretation *wahr* ist.
- Eine Formel φ ist eine Folgerung aus einer Formelmenge Φ, wenn sie unter jeder Interpretation *wahr* ist, in der alle Formeln aus Φ *wahr* sind.

Allgemeingültige Formeln werden als Tautologien bezeichnet. Ist φ eine Tautologie, so schreibt man: $\models \varphi$. Folgt φ aus Φ, so schreibt man: $\Phi \models \varphi$. Besitzen zwei Formeln φ_1 und φ_2 denselben Wahrheitswert, so schreibt man: $\varphi_1 \Leftrightarrow \varphi_2$.

Aufgrund der gegebenen dreiwertigen Semantik läßt sich anders als in der Prädikatenlogik nicht die Folgerung auf der Interpretationsebene durch die Implikation auf der Formelebene ersetzen und ebenso nicht die Gleichheit des Wahrheitswerts durch die Äquivalenz. Folgende Aussagen sind deshalb in der ternären Logik nicht allgemeingültig:

- wenn $\{A\} \models B$, dann auch $\models A \rightarrow B$
- wenn $A \Leftrightarrow B$, dann auch $\models A \equiv B$

6.1.3. Eigenschaften

Für die definierte ternäre Logik lassen sich einige Eigenschaften ableiten, die mit der klassischen Prädikatenlogik übereinstimmen, die sich aus der *bekannt*-Verknüpfung ergeben sowie die Unterschiede zur Prädikatenlogik darstellen. Zunächst gilt wie aus den Wahrheitstabellen ersichtlich wie in der Prädikatenlogik:

- *doppelte Negation*: $\neg(\neg A) \Leftrightarrow A$
- *Idempotenzgesetz*: $A \wedge A \Leftrightarrow A$
  $A \vee A \Leftrightarrow A$
- *Kommutativgesetz*: $A \wedge B \Leftrightarrow B \wedge A$
  $A \vee B \Leftrightarrow B \vee A$
- *Assoziativgesetz*: $A \wedge (B \wedge C) \Leftrightarrow (A \wedge B) \wedge C$
  $A \vee (B \vee C) \Leftrightarrow (A \vee B) \vee C$
- *Distributivgesetz*: $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$
  $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$

Für die *bekannt*-Verknüpfung und die aus ihr abgeleiteten Verknüpfungen und Quantoren lassen sich folgende Eigenschaften herleiten:

$\sim(\neg A) \Leftrightarrow \sim A$

bekannt unterscheidet bekannt und unbekannt, aber nicht wahr und falsch.

$\sim(A \wedge B) \Leftrightarrow (\sim A \wedge \neg A) \vee (\sim B \wedge \neg B) \vee (\sim A \wedge \sim B) \Leftrightarrow \downarrow A \vee \downarrow B \vee (\sim A \wedge \sim B)$
Eine Konjunktion aus A und B ist bekannt, wenn A oder B als falsch bekannt ist oder A und B beide bekannt sind.

$\sim(A \vee B) \Leftrightarrow (\sim A \wedge A) \vee (\sim B \wedge B) \vee (\sim A \wedge \sim B) \Leftrightarrow \uparrow A \vee \uparrow B \vee (\sim A \wedge \sim B)$
Eine Disjunktion aus A und B ist bekannt, wenn A oder B als wahr bekannt ist oder A und B beide bekannt sind.

$\sim(\forall x\ A(x)) \Leftrightarrow (\exists x\ (\sim A(x) \wedge \neg A(x))) \vee (\forall x \sim A(x)) \Leftrightarrow (\exists x \downarrow A(x)) \vee (\forall x \sim A(x))$
Eine Allquantifikation ist bekannt, wenn A(x) für ein x als falsch bekannt ist oder für alle x A(x) bekannt ist.

$\sim(\exists x\ A(x)) \Leftrightarrow (\exists x\ (\sim A(x) \wedge A(x))) \vee (\forall x \sim A(x)) \Leftrightarrow (\exists x \uparrow A(x)) \vee (\forall x \sim A(x))$
Eine Existenzquantifikation ist bekannt, wenn A(x) für ein x als wahr bekannt ist oder für alle x A(x) bekannt ist.

$\neg(\sim A) \Leftrightarrow \neg\uparrow A \wedge \neg\downarrow A$
Eine Aussage ist unbekannt, wenn sie weder als wahr, noch als falsch bekannt ist.

$\neg\bar{\forall} x\ A(x) \Leftrightarrow \neg\forall x\ (\sim A(x) \rightarrow A(x)) \Leftrightarrow \exists x\ \neg(\neg\sim A(x) \vee A(x)) \Leftrightarrow \exists x\ (\sim\neg A(x) \wedge \neg A(x)) \Leftrightarrow \bar{\exists} x\ \neg A(x)$ und analog $\neg\bar{\exists} x\ A(x) \Leftrightarrow \bar{\forall} x\ \neg A(x)$
Die Negation läßt sich bei den bekannt-Quantoren genauso behandeln wie bei den klassischen Quantoren.

$\sim(\sim A) \Leftrightarrow \top$
Es ist immer bekannt, ob etwas bekannt ist oder nicht.

Aufgrund des zusätzlichen Wahrheitswerts gelten die Tautologien der Prädikatenlogik prinzipiell nicht. Ohne die *bekannt*-Verknüpfung (oder die von ihr abgeleiteten Verknüpfungen), die in der ursprünglichen Logik von Kleene nicht definiert ist, kann es in der ternären Logik keine Tautologien geben, da nur diese aus Unbekanntem etwas Bekanntes machen können. So ist beispielsweise der Satz vom ausgeschlossenen Dritten trivialerweise nicht allgemeingültig: $\not\models A \vee \neg A$.

Sämtliche Tautologien der Prädikatenlogik gelten jedoch, sobald man die Bekanntheit aller in ihnen enthaltenen atomaren Formeln voraussetzt und somit zur klassischen Zweiwertigkeit übergeht. Folglich gilt: $\sim A \models A \vee \neg A$, d.h. wenn eine Annahme betreffs A besteht, so ist A entweder wahr oder falsch. Ebenso läßt sich ein Satz des ausgeschlossenen Vierten formulieren: $\models A \vee \neg\sim A \vee \neg A$.

6.2. CASA Agent Language

Die CASA Agent Language (CAL) umfaßt sämtliche Aspekte der Beschreibung und Spezifikation eines Agenten auf der Wissensebene bestehend aus einem terminologischen, einem deklarativen, einem prozeduralen und einem kommunikativen Anteil.

Der deklarative Teil ermöglicht die Repräsentation von Weltzuständen basierend auf einer in Ontologien definierten Terminologie. Dazu werden formallogische und objektorientierte Ansätze kombiniert, um die durch die Logik erreichte Ausdrucksstärke und Exaktheit in der Formulierung von Aussagen um konzeptuelle Schemata mit einer bereichsspezifischen Struktur zu erweitern. Prozedurale Fähigkeiten eines Agenten werden in Form von Planoperatoren dargestellt, die Bedingungen und Effekte von Handlungen auf einer logischen Ebene beschreiben und so deren Auswahl und Verknüpfung durch formallogische Methoden ermöglichen. Ergänzt wird dies durch eine Skriptsprache für eine prozedurale Kontrolle von Handlungsabläufen. Auf der kommunikativen Ebene erhalten die in der deklarativen Sprache ausgedrückten Inhalte innerhalb von Sprechakten eine aktorische Bedeutung, die die mit dem Sprechakt vollzogene Handlung ausdrückt. Eingebettet sind die kommunikativen Handlungen in Protokolle, die mögliche Verlaufsmuster für Konversationen festlegen. Diese einzelnen Aspekte von CAL stehen untereinander in Beziehung und bauen auf bestehenden Ansätzen in den jeweiligen Bereichen auf, wie sie in den Kapiteln 2.3.1, 2.3.2, 2.4.2 und 2.4.3 vorgestellt wurden.

Wie die in Kapitel 4 vorgenommene Spezifikation der Komponenten der Standardarchitektur stellt auch die nachfolgende Definition von CAL im wesentlichen eine Minimalanforderung dar, die nach Bedarf Erweiterungen zuläßt. Für eine effiziente Implementierung sind gegebenenfalls auch Einschränkungen der Ausdrucksstärke möglich.

Beispielsweise verzichtet CAL auf die Repräsentation von Wissen über die mentalen Zustände anderer Agenten, da für die dienstbasierten Interaktionen gemäß dem CASA-Ansatz das in den Dienstoperatoren ausgedrückte Wissen über die Interaktionsmöglichkeiten mit anderen Agenten ausreicht. Eine zusätzliche Repräsentation der Annahmen, Ziele und Intentionen anderer Agenten läßt sich leicht durch eine Erweiterung der Logik um Modaloperatoren (vgl. Kapitel 2.3.1) und der Faktenbasis um entsprechende Fakten erreichen. Der Vorteil hiervon liegt in der Möglichkeit, das Verhalten anderer Agenten in der eigenen Verhaltenssteuerung berücksichtigen zu können und so die Handlungen mehrerer Agenten explizit planen und koordinieren zu können. Andererseits ist die logisch korrekte Verwendung dieses Wissens mit einem hohen Aufwand verbunden und nicht trivial, da Annahmen über die Verhaltensweisen des anderen Agenten gemacht werden müssen und die zugrundegelegte Modallogik nicht entscheidbar ist. Weiterhin ergibt sich das Problem, wie ein Agent korrekte Informationen über die mentalen Zustände anderer Agenten erlangt und aktuell hält, da diese nicht direkt wahrnehmbar, dafür aber recht veränderlich sind. Das Ableiten aus Beobachtungen ist spekulativ und Kommunikation nur verläßlich, wenn man Ehrlichkeit und Auskunftsbereitschaft voraussetzt. Und schließlich müssen noch der Umfang und der Detaillierungsgrad der Modellierung anderer Agenten auf die relevanten Informationen beschränkt werden. All diese Schwierigkeiten vermeidet das Dienstschema von CASA, da es auf einfache Weise flexible und dynamische Interaktionen zwi-

schen Agenten ermöglicht, ohne über die Dienstbeschreibungen hinausgehende Informationen über andere Agenten zu benötigen.

Kombination von Logik und Objektorientiertheit

Da die Kombination von Logik und Objektorientiertheit die terminologische und die deklarative Ebene betrifft, werden deren Grundzüge bereits an dieser Stelle vorab zusammengefaßt. Sie beinhaltet eine Typisierung von Werten, die Verwendung von Kategorien als Objektschemata, operationalisierte Funktionen und Relationen sowie Metawissen für zusätzliche Informationen.

Durch die Typisierung werden Terme in gleichförmige Wertebereiche eingeteilt. Indem für die Parameter von Funktionen und Relationen jeweils ein Typ fest vorgegeben wird, lassen sich diese auf einen bestimmten Gegenstandsbereich beschränken, auf den sie sinnvoll angewendet werden können. Beispielsweise sind arithmetische Funktionen nur für Zahlen definiert, während nur gegenständliche, sichtbare Objekte eine Farbe besitzen können. Dabei wird eine Menge von Basistypen vorgegeben, während Kategorien als zusätzliche Typen frei definierbar sind. Weiterhin kann zu jedem Typ ein Listen- und ein Mengentyp mit Elementen dieses Typs gebildet werden.

Eine Kategorie ist ein Repräsentationsschema für gleichartige Objekte, die sich mit einer gemeinsamen Menge an Attributen beschreiben lassen. Wie in objektorientierten Sprachen gibt es Vererbungsrelationen zwischen Kategorien und sind die Attribute typisiert sowie auf eine bestimmte Kategorie beschränkt. Formal stellt die Kategoriezugehörigkeit eine Relation dar, die zwischen einem Objekt und dessen Kategorie sowie sämtlichen Oberkategorien dieser Kategorie besteht. Attribute hingegen stellen formal gesehen jeweils Funktionen des deklarierten Typs dar mit genau einem Parameter vom Typ der zugeordneten Kategorie. Angewandt auf ein Objekt dieser Kategorie ist der Wert einer derartigen Funktion gerade der Attributwert.

Die Relation der Kategoriezugehörigkeit sowie die Attributfunktionen sind die einzigen, deren Interpretation durch eine explizite Aufzählung – in der Regel in der Faktenbasis – gegeben ist. Andere Funktionen und Relationen werden in Ontologien deklariert und werden operational durch einen zu implementierenden Algorithmus definiert.

Schließlich erlaubt die Wissensrepräsentationssprache noch Metawissen, das einem bestimmten Wissensinhalt zugeordnet ist und anders als dieses kein Wissen auf der Objektebene, sondern Wissen über dieses Wissen ausdrückt. Das Metawissen besitzt keine feste formale Semantik, sondern wird von den wissensverarbeitenden Komponenten als zusätzliche Information über das Wissen selbst verwendet. Ein Beispiel hierfür ist der Zeitpunkt, zu dem ein Fakt bekannt wurde bzw. zu dem er zuletzt bestätigt wurde. Diese Information kann dann zur Beurteilung der Verläßlichkeit des Fakts herangezogen werden, indem aktuelleres Wissen als eine bessere

Grundlage von Entscheidungen angesehen wird, falls man die Wahrscheinlichkeit der Änderung von Fakten als im Verlauf der Zeit zunehmend ansieht.

6.3. Ontologien

Die terminologische Ebene der Wissensrepräsentationssprache CAL entspricht der Signatur einer formalen Logik (vgl. Definition 22), da beide jeweils die verwendeten denotierenden Zeichen festlegen. Ontologien deklarieren jedoch nicht nur die Symbole, sondern auch Kategorien als Repräsentationsschemata für Objekte sowie Typen für die Parameter von Funktionen und Relationen.

Nachfolgend werden bereits die Definitionen für Typen, Terme und Formeln in CAL vorausgesetzt, die erst in den beiden nachfolgenden Unterkapiteln angegeben werden.

Metawissen

```
KeywordDecl        = (key Keyword KeyType)
KeyType            = keyword
                   | term
                   | formula
Keyword            = Keyword
                   | (Keyword Term)
                   | (Keyword Formula)
MetaAttributeDecl  = (meta MetaAttName Type Keyword*)
MetaDecl           = MetaAttributeDecl
                   | KeywordDecl
MetaOnto           = (ont meta MetaDecl*)
```

Metawissen stellt Informationen außerhalb der formallogischen Ebene bereit, deren Semantik operational durch die das Wissen verarbeitenden Komponenten festgelegt wird. Deshalb werden die Ausdrücke zur Repräsentation von Metawissen in einer Metaontologie für ein System als ganzes definiert.

Unterschieden werden zwei Arten von Metawissen. Über Schlüsselworte läßt sich ein Attribut oder Metaattribut zusätzlich charakterisieren, wozu optional ein Term oder eine Formel verwendet werden kann. Beispielsweise läßt sich für ein Attribut über ein Schlüsselwort festlegen, daß der Wert unveränderlich ist, es kann über einen Term ein Default-Wert festgelegt werden oder eine Formel als Rahmenbedingung angegeben werden, die den Bereich zulässiger Werte einschränkt.

Metaattribute sind Eigenschaften, die einem Wissenselement als solchem zukommen. Mögliche Beispiele hierfür sind die Gewißheit oder die Rechtfertigung von Fakten, der Zeitpunkt, seit dem ein Fakt, ein Ziel oder eine Intention besteht, oder die Prioritäten von Zielen oder Planelementen. Zur Deklaration eines Metaattributs wird neben dem Namen der Typ und eine Menge von Schlüsselworten angegeben.

Ein Schlüsselwort besteht je nach Deklaration aus dem Namen oder zusätzlich aus einem Term oder einer Formel, in denen per Konvention die Variable ?this zur Referenzierung auf den Attributwert selbst vorkommen kann.

Die Wissensrepräsentationssprache CAL ist offen für beliebige Deklarationen an Metawissen. Ob und wie damit formuliertes Metawissen erzeugt und verwendet wird, hängt jedoch ausschließlich von dem jeweiligen implementierten System ab und ist nicht Teil der Spezifikation von CAL.

Kategorien

```
CategoryDecl      = (cat CatName (ext CatName+) AttributeDecl*)
AttributeDecl     = (AttName Type Keyword*)
```

Die Deklaration einer Kategorie umfaßt deren Namen sowie optional eine Menge von abgeleiteten Kategorien, deren Deklaration geerbt wird, indem die für diese gültigen Attribute auch für die neu deklarierte Kategorie gültig sind. Es folgt eine Menge von Attributdeklarationen, die jeweils aus einem Attributnamen und dem Typ sowie optional einer Menge von Schlüsselworten besteht.

Die Namen von Kategorien und Attributen müssen auch über verschiedene Ontologien hin eindeutig sein. Dazu wird dem Namen einer Kategorie der Name der Ontologie und dem eines Attributs der so eindeutig gemachte Name der Kategorie vorangestellt. Als Trennzeichen wird ein Punkt verwendet. Somit brauchen Kategorienamen nur relativ zur jeweiligen Ontologie und Attributnamen nur relativ zur jeweiligen Kategorie eindeutig zu sein, während sich die Eindeutigkeit des vollständigen Namens dann aus der Eindeutigkeit der Ontologienamen ergibt.

Anders als in vielen objektorientierten Sprachen ist die Vererbung auch mehrerer Klassen und über mehrere Ebenen hin in CAL völlig problemlos. Die Eindeutigkeit sämtlicher Attributnamen verhindert sowohl Konflikte durch mehrfache Deklaration desselben Attributs in unterschiedlichen abgeleiteten Kategorien, als auch die Verdopplung von Deklarationen durch das mehrfache Ableiten derselben Kategorie, da in Einklang mit der formallogischen Interpretation von Attributen als Funktionen in diesem Fall lediglich dieselbe Funktion auf dieselbe Weise definiert wird.

Funktionen und Vergleiche

```
FunctionDecl      = (fun Type FunName Type*)
ComparisonDecl    = (comp CompName Type+)
```

Funktionen und Vergleiche sind logische Funktionen und Relationen, für die eine operationale Semantik angegeben wird. Für eine Funktion wird deren Typ, der Name und die Typen der Parameter angegeben. Konstante Funktionen ohne Parameter sind zulässig. Die Deklaration eines Vergleichs beinhaltet dessen Namen sowie die Typen der Parameter. Da ein konstanter Vergleich gerade einem der Wahrheitswerte entspricht, wird für Vergleiche mindestens ein Parameter gefordert.

Auch die Namen von Funktionen und Vergleichen brauchen nur relativ zu der jeweiligen Ontologie eindeutig zu sein und werden ebenfalls durch Voranstellen des Ontologienamens global eindeutig gemacht. Die Zuordnung des Namens einer Funktion oder eines Vergleichs zu der operationalen Interpretation wird an dieser Stelle nicht festgelegt.

Ontologien

```
Ontology        = (ont OntoName (incl OntoName+) OntoDecl*)
OntoDecl        = CategoryDecl
                | FunctionDecl
                | ComparisonDecl
```

Die Deklaration einer Ontologie beinhaltet deren Namen sowie optional eine Menge von einzuschließenden Ontologien. Es folgt eine Menge von Deklarationen für Kategorien, Funktionen und Vergleiche, in denen nur solche Symbole verwendet werden dürfen, die in derselben oder einer eingeschlossenen Ontologie deklariert sind.

Aufgrund der beschriebenen Eindeutigkeit sämtlicher Symbole eines bestimmten Typs – Namen für verschiedene Arten von Symbolen können aufgrund der Syntax anhand der Stelle ihres Vorkommens unterschieden werden – können auch durch das Einschließen von Ontologien keine Mehrdeutigkeiten, sondern nur übereinstimmende Doppelnennungen derselben Deklaration entstehen. Lediglich die Eindeutigkeit der Namen von Ontologien muß anderweitig gewährleistet werden.

6.4. Werte und Terme

Terme sind – wie bereits oben im Zusammenhang mit der ternären Logik beschrieben – die denotierenden Elemente einer formallogischen Repräsentation, die auf konkrete Objekte[42] des modellierten Bereichs Bezug nehmen. Terme besitzen einen Typ, der jeweils eine Menge gleichartiger Objekte zusammenfaßt, für die sich gemeinsam anwendbare Funktionen und Relationen definieren lassen. Werte sind Terme, die genau einem Objekt des Gegenstandsbereichs entsprechen, das sie direkt bezeichnen. Variablen sind Platzhalter für beliebige Terme eines bestimmten Typs.

[42] Der Begriff Objekt wird sowohl zur Bezeichnung der repräsentierten Gegenstände, als auch der repräsentierenden Strukturen verwendet. Diese doppelte Bedeutung entstammt der Begrifflichkeit der objektorientierten Programmierung. Falls nicht explizit unterschieden, ist im folgenden immer die letztere Bedeutung gemeint.

Typen

```
BaseType        = int
                | real
                | bool
                | string
Type            = BaseType
                | CatName
                | Type[]
                | Type{}
                | abstract
```

An Basistypen sind ganze Zahlen (`int`), reelle Zahlen (`real`), logische Werte (`bool`) und Zeichenfolgen (`string`) definiert. Die Menge dieser Typen läßt sich nach Bedarf erweitern, beispielsweise um Typen einer Implementierungssprache. Weiterhin ist jede in einer Ontologie deklarierte Kategorie ein Typ, der jeweils durch den Namen der Kategorie bezeichnet wird. Schließlich existiert zu jedem Typ noch ein Listentyp (eckige Klammern) und ein Mengentyp (geschweifte Klammern) für Listen bzw. Mengen des angegebenen Typs. Der Typ *abstract* steht für einen beliebigen Typ.

Werte

```
Constant        = int
                | real
                | bool
                | string
Object          = ObjName:CatName
Value           = Constant
                | Object
                | [Value*]
                | {Value*}
                | ?:Type
```

Die Elemente der Basistypen werden als Konstanten bezeichnet. Werte sind außerdem die Objekte, die jeweils durch einen eindeutigen Namen bezeichnet werden, und als Typ die angegebene Kategorie sowie sämtlich ihrer Oberkategorien besitzen[43]. Schließlich gibt es noch Listen (eckige Klammern) und Mengen (geschweifte Klammern) von Werten. Da jeder Wert einen bestimmten Typ hat, müssen sämtliche Elemente einer Liste oder Menge denselben Typ besitzen, so daß die Liste bzw. Menge den zugeordneten Listen- bzw. Mengentyp besitzt. Das Fragezeichen steht als Platzhalter für einen beliebigen Wert des angegebenen Typs, beispielsweise zur Definition von Mustern.

---

[43] Alternativ zu der expliziten Angabe der Kategorie bei jeder Verwendung eines Objekts kann auch eine einmalige Deklaration der Typen von Objekten geschehen.

Variablen

```
Variable          = ?VarName
VarDecl           = Variable:Type=Term
VariableDecl      = (var VarDecl+)
```

Eine Variable hat einen beliebigen Namen und wird durch ein vorangestelltes Fragezeichen als Variable gekennzeichnet. Der Typ einer Variablen wird explizit deklariert, wobei dieser optional auch bereits eine Bindung an einen bestimmten Term zugewiesen werden kann. Eine Variablendeklaration umfaßt die Typdeklaration für mehrere Variablen.

Terme

```
Attribution       = (att AttName Term)
Function          = (fun FunName Term*)
Cast              = (cast Term Type)
Collection        = [Term*]
                  | {Term*}
Term              = Value
                  | Variable
                  | Attribution
                  | Function
                  | Cast
                  | Collection
```

Die Gesamtheit der Terme umfaßt zunächst sämtliche Werte und Variablen. Hinzu kommen Funktionsausdrücke, die entweder den Wert eines Attributs eines Objekts oder eine operationalisierte Funktion beinhalten. In beiden Fällen ergeben sich der Typ des Terms und der Parameter sowie in letzterem Fall die Anzahl der Parameter aus der entsprechenden Deklaration in einer Ontologie. Mit der Cast-Funktion, deren Interpretation die identische Abbildung ist, kann eine Typumwandlung durchgeführt werden. Diese ist nur zulässig, falls der Term auch den angegeben Typ besitzt, und wird benötigt wegen des Typs *abstract*, und da aufgrund der Vererbungsrelation zwischen Kategorien der Typ eines Objekts, für den ein Term steht, spezieller sein kann als der Typ des Terms selbst. Wie bei den Werten lassen sich auch über Terme Listen und Mengen bilden, deren Elemente ebenfalls alle vom selben Typ sein müssen.

6.5. Formeln

Formeln repräsentieren Aussagen über Objekte des modellierten Bereichs. Im folgenden werden im wesentlichen die bereits in Kapitel 6.1.2 definierten Formeln mit einer anderen Syntax als Teil von CAL eingeführt.

Atomare Formeln

```
TruthValue        = true
                  | unknown
                  | false
Equation          = (equal Term Term)
Comparison        = (comp CompName Term+)
TypeTest          = (type Term Type)
AtomicFormula     = TruthValue
                  | Equation
                  | Comparison
                  | TypeTest
```

Zu den atomaren Formeln zählen die Wahrheitswerte sowie Relationen, bei denen drei Arten unterschieden werden. Die Gleichheitsrelation beinhaltet die Gleichheit zweier Terme. Sie ist wahr, wenn beide Terme denselben Wert besitzen, und falsch, wenn sie unterschiedliche Werte besitzen. Andere Vergleichsrelationen werden in Ontologien deklariert und besitzen jeweils eine operationale Interpretation. Der Typtest schließlich ist wie die Cast-Funktion notwendig wegen der Typisierung von Termen, insbesondere wegen dem allgemeinen Typ *abstract* und der Vererbungsrelation zwischen Kategorien. Er ist wahr, wenn der Term den angegeben Typ besitzt, und falsch, wenn dieser einen anderen Typ besitzt. Der Typtest ist nur insofern eine Relation, als hier anders als sonst die Typbezeichnungen ihrerseits als Werte aufgefaßt werden.

Verknüpfungen

```
Negation          = (not Formula)
Known             = (known Formula)
                  | (ktrue Formula)
                  | (kfalse Formula)
Conjunction       = (and Formula++)
Disjunction       = (or Formula++)
Implication       = (imp Formula Formula)
Equivalence       = (equ Formula Formula)
UnaryJunction     = Negation
                  | Known
BinaryJunction    = Conjunction
                  | Disjunction
                  | Implication
                  | Equivalence
```

Die einstelligen Verknüpfungen sind die Negation sowie die drei Bekanntheitsoperatoren. Die zweistelligen Operatoren umfassen Konjunktion, Disjunktion, Implikation und Äquivalenz. Infolge des Assoziativgesetzes lassen sich Konjunktion und Disjunktion dabei auf eine beliebige Anzahl an Formeln ausweiten.

Quantoren

```
Quantor           = forall | exists
Quantification    = (Quantor VariableDecl Formula)
Iteration         = (Quantor VarDecl Collection Formula)
```

Die All- und die Existenzquantifikation werden auf mehrere Variablen ausgeweitet. Um eine Konjunktion oder Disjunktion über die Elemente einer Menge oder Liste auszudrücken, wird die Iteration verwendet, wobei in der Formel die angegebene Variable für diese Elemente steht.

Formeln

```
Formula           = AtomicFormula
                  | UnaryJunction
                  | BinaryJunction
                  | Iteration
                  | Quantification
```

Die Gesamtheit aller Formeln ergibt sich nun aus atomaren Formeln, einstelligen und zweistelligen Verknüpfungen sowie Quantifikation und Iteration.

6.6. Deklaratives Wissen

Das deklarative Wissen umfaßt Fakten, Ziele und Intentionen. Zuvor wird noch die Darstellung von Metaattributen angegeben.

Metaattribute

```
MetaAtts          = (meta (MetaAttName Value)+)
```

Die einem Wissenselement zugeordneten Metaattribute bestehen aus einer Menge an Attribut/Wert-Paaren. Die Metaattribute müssen in der Metaontologie deklariert sein, woraus sich auch der Typ des Werts ergibt.

Fakten

```
Fact              = (FactType AttName Object Value):MetaAtts
FactType          = pos
                  | neg
```

Fakten sind positive oder negative Annahmen über den Wert eines Attributs eines Objekts. Zudem können Fakten Metaattribute besitzen.

Diese Ausdrücke für Fakten entsprechen folgenden Formeln:

- (pos *AttName Object Value*) ⇔ (equal (att *AttName Object*) *Value*)
- (neg *AttName Object Value*) ⇔ (not (equal (att *AttName Object*) *Value*))

Prinzipiell lassen sich auch beliebige Formeln als Fakten verwenden, was jedoch einen erheblich höheren Aufwand bei der Evaluierung von Formeln und der Verwaltung des Faktenbestandes nach sich zieht.

Ziele

```
StateGoal          = (goal Formula):MetaAtts
ServiceGoal        = (goal ProtocolDescr ServiceInst Term*):MetaAtts
ProtocolDescr      = (prot Protocol Role Service)
ServiceInst        = (serv ServiceInst)
Role               = user | provider
Goal               = StateGoal
                   | ServiceGoal
```

Unterschieden werden zwei Arten von Zielen. Zustandsziele beschreiben einen zu erreichenden Zielzustand mit Hilfe einer Formel. Ein Interaktionsziel besitzt eine Protokollbeschreibung, eine Dienstinstanz und eine Menge von Parametern mit der Instantiierung des Dienstes. Die Protokollbeschreibung charakterisiert die durchzuführende Interaktion und besteht aus dem Namen des Protokolls, der darin eingenommenen Rolle und dem Namen des Dienstes. Die Dienstinstanz erlaubt die Zuordnung zu einer konkreten Dienstnutzung, wozu sie zumindest deren Konversation identifizieren muß. Anzahl und Typen der übergebenen Parameter ergeben sich aus der Variablendeklaration des Dienstoperators (s.u.).

Intentionen

```
Intention          = (intent OperatorName ServiceInst Term*)
```

Eine Intention enthält den Namen des zur Ausführung bestimmten Operators (s.u.) und eine Parameterliste zur Instantiierung dieses Operators. Falls die Intention für ein Interaktionsziel aufgestellt wurde, enthält sie zudem dessen Dienstinstanz, um eine Zuordnung von Kommunikationshandlungen zu Konversationen zu ermöglichen.

6.7. Prozedurales Wissen

Das prozedurale Wissen stellt die Fähigkeiten eines Agenten dar und umfaßt Reaktionsregeln, Operatoren und Handlungssequenzen. Um in Einklang zu dem Faktenwissen und den Zielen des Agenten verwendet zu werden, besitzen diese jeweils einen deklarativen Anteil, der die Umstände der Ausführung abstrakt beschreibt.

6.7.1. Reaktionsregeln

```
Rule            = (rule Condition Actions Actions):MetaAtts
Condition       = Formula
Actions         = Action
                | (Action++)
Action          = (update Formula)
                | (Change Knowledge)
                | (inform ComponentRole)
Change          = add
                | remove
Knowledge       = Fact
                | Goal
                | Intention
                | Rule
```

Eine Reaktionsregel besitzt eine Formel als Bedingung sowie eine Menge von Aktionen, die bei Eintreten der Erfülltheit der Bedingung ausgeführt werden. Optional können noch Aktionen zur Ausführung bei Ende der Erfülltheit sowie eine Menge von Metaattributen angegeben werden. Eine Aktion besteht in der Aktualisierung der Fakten anhand einer Formel, der Änderung des Wissens durch Hinzufügen oder Entfernen von Fakten, Zielen, Intentionen oder Regeln oder in der Benachrichtigung einer Komponentenrolle über die geänderte Situation.

6.7.2. Operatoren

Ein Operator beschreibt eine intentional ausführbare Fähigkeit eines Agenten. Eine Handlung ist eine Fähigkeit zur Änderung des Weltzustands und beinhaltet eine formallogische Beschreibung der Bedingungen und Effekte ihrer Ausführung. Ein Sprechakt beschreibt eine kommunikative Handlung, die jedoch keine formallogische Beschreibung besitzen, da in CAL kein Wissen über andere Agenten explizit repräsentiert wird. Bedingungen repräsentieren eine Auswahl aus mehreren Alternativen. Sämtliche Operatoren besitzen einen eindeutigen Namen, über den sie referenziert werden.

Handlungen

```
Act                = (act ActName VariableDecl
                        (pre Formula) (cond Formula) Effect
                        Execution):MetaAtts
Effect             = (eff Formula+)
                   | ProtocolDescr
Execution          = PrimitiveExec
                   | inference
                   | abstract
                   | ServiceExec
                   | PlanExec
                   | ScriptExec
PrimitiveExec      = (call ComponentRole)
ServiceExec        = (service Service)
PlanExec           = (plan VariableDecl PlanStep+)
ScriptExec         = (script VariableDecl Script)
```

Eine Handlung besitzt eine formale Beschreibung in Form von Vorbedingung, Rahmenbedingung und Effekten, die jeweils aus einer Formel bestehen. Diese können freie Variablen – Variablen die nicht durch einen Quantor gebunden sind – enthalten, die dann in einer Variablendeklaration aufgeführt werden und die Parametrisierbarkeit der Handlung ausdrücken. Eine Handlung mit mehreren Effekten ist indeterministisch, d.h. jeder der angegebenen Effekte kann eintreten. Falls eine Handlung eine Rolle in einem Protokoll umsetzt, ist der Effekt eine entsprechende Protokollbeschreibung, aus der auch die Variablendeklaration des darin enthaltenen Dienstoperators übernommen wird. Schließlich gibt es noch verschiedene Arten der Handlungsausführung sowie optional eine Menge von Metaattributen.

An Operatortypen gibt es Handlungsprimitive, Inferenzen, abstrakte Handlungen, Dienste, Pläne und Skripte. Für ein Handlungsprimitiv wird die Rolle einer Anwendungskomponente angegeben, die dieses umsetzt. Inferenz und abstrakte Handlung bestehen nur aus einem Schlüsselwort, da die daraus abzuleitende Handlung – Änderung der Faktenbasis bzw. Aufstellen eines Ziels – sich aus dem Effekt ergibt. Für einen Dienst wird der Name des Dienstes angegeben, wobei davon ausgegangen wird, daß anhand des Dienstnamens die eigentlichen Informationen über diesen Dienst als Fakten in der Faktenbasis gespeichert werden. Entsprechendes gilt auch für die Protokolle, die in Protokollbeschreibungen enthalten sind und dort ebenfalls nur über einen Namen referenziert werden. Schließlich gibt es als Handlungssequenzen noch Pläne und Skripte, die jeweils optional eine Variablendeklaration für lokale Variablen besitzen. Ein Plan besteht aus einer Liste von Planschritten und ein Skript aus einem initialen Skriptschritt. Planschritte und Skriptschritte werden weiter unten in Kapitel 6.7.3 beschrieben.

Sprechakte

```
SpeechAct       = Send
                | Receive
Send            = (send SpeechActName VariableDecl Params Message)
Receive         = (receive SpeechActName VariableDecl Params Message+)
Message         = Performative
                | (Performative Content+)
Performative    = request | request_when | agree | refuse
                | done | failure | cancel
                | inform | data | query | query_if
                | cfp | propose | accept_proposal | reject_proposal
                | not_understood
Content         = (act Service)
                | (par Term+)
                | (cond Formula)
                | (inf Formula)
                | (data Term+)
                | (reason Term)
Params          = (par Param+)
Param           = (with Term)
                | (in Term)
                | (by Term)
```

Das Versenden und Empfangen von Sprechakten geschieht aus Protokolloperatoren heraus. Dazu wird ein Sprechaktoperator verwendet, der ein Muster für den zu sendenden oder alternative Muster für zu empfangende Sprechakte vorgibt. Optional kann dabei eine Variablendeklaration zur Parametrisierung des Inhalts eines Sprechakts verwendet werden. Ebenfalls optional ist die Verwendung zusätzlicher Sprechaktparameter. Der Sprechakt selbst besteht entweder nur aus einem Performative oder aus einem Performative und einer Menge von Inhalten. Die Bedeutung von Performatives, Inhaltstypen und Sprechaktparametern wird zusammen mit der Transportform von Sprechakten in Kapitel 6.8 beschrieben.

Bedingungen

```
Cond            = (cond VariableDecl CondName Formula+)
```

Eine Bedingung beschreibt eine Menge von Alternativen, die jeweils durch eine eigene Formel ausgedrückt werden. Verwendet werden Bedingungen in Verzweigungen in Skripten, wobei die erste erfüllte Formel als die zu wählende Alternative gilt. Zur Parametrisierung können zuvor deklarierte Variablen verwendet werden.

Operatoren

```
Operator        = Act
                | Cond
                | SpeechAct
```

Die Gesamtheit der Operatoren besteht nun aus Handlungen, Bedingungen und Sprechaktoperatoren.

6.7.3. Handlungsabläufe

Handlungsabläufe sind Pläne und Skripte, die jeweils aus einzelnen Schritten zusammengesetzt sind. Sie referieren auf Operatoren in Form von Operatorinstanzen, die ähnlich wie Intentionen die Instantiierung eines Operators beschreiben.

Operatorinstanzen

```
OperatorInstance   = OperatorID
                   | (OperatorID Term+)
OperatorID         = OperatorName:int
UncondOperator     ≅ unbedingte Operatorinstanz
CondOperator       ≅ bedingte Operatorinstanz
```

Eine Operatorinstanz besteht aus einer Referenz auf den Operator und gegebenenfalls der Instantiierung durch Angabe von Werten für die deklarierten Variablen. Um verschiedene Instanzen desselben Operators zu unterscheiden, wird an dessen Namen durch einen Doppelpunkt getrennt eine numerische ID angehängt. Gelegentlich wird zwischen Instanzen von konditionalen und unbedingten Operatoren unterschieden, wobei die bedingten Operatoren gerade die Bedingungen, den Sprechaktempfang mit mehreren Mustern sowie indeterministische Handlungen ausmachen.

Planschritte

```
PlanStep           = (Predecessor (order OperatorID+) Successor)
Predecessor        = OperatorID
                   | (Connective Predecessor++)
Connective         = and | or
Successor          = OperatorInstance
                   | (alt OperatorInstance++)
```

Ein Planschritt beschreibt die Abhängigkeiten zwischen den einzelnen Operatorinstanzen, aus denen ein Plan besteht. Für eine einzelne oder eine Menge alternativer Operatorinstanzen wird jeweils die Menge der notwendigen Vorgänger angegeben, die zusammen oder alternativ zuvor ausgeführt werden müssen. Dies sind diejenigen Operatorinstanzen desselben Plans, deren Effekte die Bedingungen des gerade beschriebenen Operators erfüllen. Für diese können optional noch Ordnungsbedingungen festgelegt werden, um eine totale oder partielle Reihenfolge zur Vermeidung von Konflikten zwischen den Operatoren vorzugeben.

Start- und Endpunkt eines Plans sind Pseudooperatoren mit den Namen *start* und *end*, deren Variablendeklaration der des Operators für den Gesamtplan entspricht. *start* besitzt immer die Bedingungen des Gesamtplans als Effekt, während *end* dessen Effekt als Vorbedingung besitzt.

Skriptschritte

```
Script            = ScriptControl
                  | ScriptKnowledge
                  | UncondOperator
ScriptControl     = ScriptOrder
                  | ScriptIter
                  | ScriptCond
                  | ScriptPrim
ScriptOrder       = (OrderType Script++)
ScriptIter        = (OrderType VarDecl Collection Script)
ScriptCond        = (CondType CondOperator Script+)
OrderType         = seq | par | alt
CondType          = branch | loop
ScriptPrim        = continue
                  | break
                  | end
                  | fail
ScriptKnowledge   = (bind Variable Term)
                  | (unbind Variable)
                  | (add Fact)
                  | (remove Fact)
                  | (eval Formula)
                  | (update Formula)
```

Die Skriptsprache umfaßt Kontrollelemente in Form von Steueranweisungen, Wissensveränderungen und Operatorausführungen. Letztere bestehen in der direkten Angabe einer Operatorinstanz.

Zu den Steueranweisungen gehört die Ausführungsreihenfolge, die in einer sequentiellen, parallelen oder alternativen – d.h. nacheinander bis zur erster erfolgreichen Alternative – Ausführung einer Menge von Kontrollelementen oder eines Kontrollelements iteriert über die Elemente einer Liste oder Menge besteht. Weiterhin zählen hierzu Verzweigungen und Schleifen, in denen anhand der Evaluierung eines konditionalen Operators eine Alternative zur Ausführung ausgewählt wird. Bei einer Schleife wird nach einer Ausführung erneut die Bedingung ausgewertet. Schließlich gibt es noch Steueranweisungen zum Fortführen oder Verlassen einer Schleife sowie zum Beenden des Skripts mit einem Erfolg oder einem Fehlschlag.

Zur Veränderung von Wissen gibt es drei Arten von Kontrollelementen. Zunächst kann für eine Variable ein Wert zugewiesen oder die Wertzuweisung aufgehoben werden. Weiterhin lassen sich Fakten aufstellen und zurücknehmen. Schließlich kann auf der Aussageebene auf das Faktenwissen zugegriffen werden, indem eine Formel evaluiert oder zur Aktualisierung der Fakten verwendet wird.

6.8. Kommunikatives Wissen

Der Inhalt der Kommunikation zwischen Agenten besteht in Sprechakten, die den darin ausgedrückten Gehalt und die damit ausgeführte Handlung in sich vereinen (vgl. Kapitel 2.4.2 und 5.1.1). Die Repräsentation von Sprechakten in CASA geschieht gemäß dem FIPA-Standard ACL (Agent Communication Language) [FIPA 2000: XC00061D, PC00037F, XC00070F]. Für Briefe mit zusätzlicher Transportinformation wird das Format der FIPA-98-Spezifikation (FIPA 1998, Part 1) verwendet, da dies in der neueren Spezifikation fehlt.

Sprechaktparameter

```
ACLSpeechAct      = (PerformativeName Parameter*)
Parameter         = :ParameterName SExpression
SExpression       = Symbol
                  | (SExpression*)
```

In ACL besteht ein Sprechakt aus einem Sprechakttyp (Performative) und einer Liste von Parametern, die jeweils aus Paaren aus einem durch einen vorangestellten Doppelpunkt gekennzeichneten Namen und einem Wert bestehen. Der Wert ist ein sogenannter S-Expression aus beliebig geschachtelten vollständig geklammerten Ausdrücken. Da die vorangegangene Syntax von CAL ausschließlich S-Expressions verwendet, können in ihr formulierte Ausdrücke als Parameterwerte von Sprechakten verwendet werden. Abbildung 22 faßt die von FIPA vorgegebenen Sprechaktparameter zusammen. Obligatorisch ist dabei lediglich der Sprechakttyp, die übrigen vorgegebenen Parameter sind optional und weitere Parameter können nach Bedarf frei definiert werden.

Parameter	Beschreibung
performative	Typ der ausgeführten Handlung
:sender	Agentenname des Absenders
:receiver	Agentennamen der Empfänger
:reply-to	Agentenname des Empfängers von Antworten
:protocol	Liste der verwendeten Konversationsprotokolle
:conversation-id	Liste von Identifikatoren für die Konversation
:reply-with	Identifikator zur Verwendung in Antworten
:in-reply-to	Bezugnahme auf den Identifikator eines Sprechakts
:reply-by	Zeitlimit für eine Antwort
:content	der Inhalt
:encoding	Format des Inhalts
:language	Sprache zur Formulierung des Inhalts
:ontology	Ontologie zur Formulierung des Inhalts

Abbildung 22: Sprechakt-Parameter

Parameter	Beschreibung
`:name`	global eindeutiger Name des Agenten
`:addresses`	Liste von Agentenadressen
`:resolvers`	Liste von Agenten zur Namensauflösung

Abbildung 23: AID-Parameter

Adressierung

Zur Adressierung gibt es Parameter für den Absender, für eine Menge von Empfängern, die jeweils ein Exemplar des Sprechakts erhalten sollen, sowie für einen Empfänger von Antworten, falls diese nicht an den Absender geschickt werden sollen. Als Wert besitzen alle diese Parameter AIDs. Da in CASA jeder Sprechakt Teil einer Dienstnutzung ist, ergibt sich aus dieser automatisch der Absender und die Empfänger und werden durch die Kommunikationskomponente gesetzt.

Das Format eines AID entspricht dem von ACL-Sprechakten, außer daß es statt mit einem Sprechakttyp mit dem Wort `agent-identifier` beginnt und andere Parameter vordefiniert sind (Abbildung 23). Über den Namen muß eine global, d.h. über sämtliche APs hin eindeutige Identifizierung des Agenten möglich sein, weshalb dieser von der HAP vergeben wird. Optional sind die Kommunikationsadressen des Agenten in Form von URLs sowie eine Menge von Agenten in Form von AIDs, über die anhand des Namens dieses Agenten seine aktuellen Adressen erfahrbar sind.

Konversationsparameter

Weiterhin gibt es Parameter zur Einbettung eines Sprechakts in eine Konversation. Dazu kann einer Konversation eine Identifikation zugeordnet und das verwendete Protokoll bezeichnet werden. Da Konversationen wie auch Protokolle ineinander verschachtelt auftreten können, werden für diese Parameter Listen verwendet, in denen vorne jeweils der aktuelle und nachfolgend die jeweils übergeordneten Kontexte angegeben werden. Auch diese beiden Parameter werden in CASA durch die Kommunikationskomponente bestimmt, da diese sämtliche Konversationen kontrolliert.

Auch die Zuordnung von Antworten über einen Identifikator und die Festlegung eines Zeitlimits für Antworten betreffen die Durchführung von Konversationen, lassen sich aber für jeden Sprechakt explizit im Rahmen der Parameter eines Sprechaktoperators festlegen.

	Bedeutung	Typ	Beschreibung
:act	action	Symbol	Dienstname
:par	parameter	Variablenbelegung	Instantiierung eines Dienstoperators
:cond	condition	Formel	Bedingung
:inf	inform	Formel	Aussage
:data	data	Liste von Termen	Daten
:reason	reason	Symbol	Begründung
:sact	speech act	Sprechakt	Sprechakt

Abbildung 24: Inhaltstypen von Sprechakten

Typ	Sprechakttyp	C	F	Inhalt	Beschreibung
Action	**request**	x	x	AP	Anfrage zur Nutzung von Dienst A mit Parametern P
	request-when	x	x	APC	Anfrage zur Nutzung von Dienst A mit Parametern P, sobald C erfüllt ist
	request-whenever	–	x		
	agree	x	x	AC	Annahme des Dienstes A unter Bedingung C
	refuse	x	x	AR	Ablehnung des Dienstes A mit Begründung R
	cancel	x	x	AR	Abbrechen des Dienstes A mit Begründung R
	done	x	–	AP	Dienst beendet mit Parametern P [inform (and (Done(A), result(A, P)))]
	propagate	–	x		
	proxy	–	x		
Info	**inform**	x	x	F	Information mitteilen
	inform-if	–	x		
	inform-ref	–	x		
	confirm	–	x		
	disconfirm	–	x		
	data	x	–	D	beliebige Daten übertragen
Query	**query**	x	–	F	Anfrage stellen
	query-if	x	x	F	ja / nein-Frage stellen
	query-ref	–	x		
	subscribe	–	x		
Negotiation	**cfp**	x	x	AC	Aufruf, ein Angebot für Dienst A zu machen mit Anforderungen C
	propose	x	x	AC	Angebot C für Dienst A
	accept-proposal	x	x	AC	Annahme von Angebot C für Dienst A
	reject-proposal	x	x	ACR	Ablehnung von Angebot C für Dienst A wegen R
Error	**failure**	x	x	AR	Fehler R bei Ausführung von Dienst A
	not-understood	x	x	SR	Sprechakt S nicht verarbeitbar wegen R

Abbildung 25: Sprechakttypen von CASA und FIPA

Inhalt von Sprechakten

Schließlich ist auch der Inhalt eines Sprechakts ein Parameter. Zur Interpretation des Inhalts beinhalten weitere Parameter dessen Format sowie die zur Formulierung verwendeten Sprache und Ontologien. In CASA ist die Inhaltssprache in der Regel CAL.

Da je nach Sprechakttyp ein Sprechakt auch mehrere Inhalte besitzen kann (s.u.), besteht bei der Verwendung von CAL als Inhaltssprache der Inhalt immer aus einer Liste von Inhalten. Dabei wird dieselbe Syntax wie bei ACL-Sprechakten und AIDs zugrundegelegt, jedoch beginnend mit dem Wort `content` und den in Abbildung 24 zusammengefaßten Parametern.

Sprechakttypen

Wie bei den Parametern gibt FIPA auch eine Menge von Sprechakttypen vor, die weder zwingend noch ausschließlich zu verwenden sind. Da deren Semantik auf einer Modallogik mit einer höheren Ausdrucksstärke als der CAL zugrundeliegenden ternären Logik beruht, weicht die für CASA vorgeschlagene Menge an Sprechakttypen (Abbildung 25) teilweise von den in FIPA vordefinierten ab, um nicht von der originalen semantischen Interpretation abzuweichen. Andererseits ergibt sich aufgrund der Einbettung sämtlicher Sprechakte in Protokolle in CASA die Semantik eines Sprechakts letztlich aus dem Protokoll, das die intendierte Verarbeitung des Sprechakts festlegt. Entsprechend ist die in Abbildung 25 angegebene Zuordnung von Sprechakttypen zu benötigtem Inhaltstyp eher informativ aufzufassen. Prinzipiell ist auch in CASA die Menge der Sprechakttypen und der Sprechaktparameter nach Bedarf erweiterbar.

Briefe

Letter = (letter :envelope (*Parameter**) :message *ACLSpeechAct*)

Ein Brief dient dem Versand von Sprechakten und enthält zusätzliche Informationen, die nur für den Versand, nicht aber für die Verarbeitung des Sprechakts benötigt werden. Er besteht aus einem Umschlag und einer Nachricht. Letztere ist gerade ein Sprechakt. Der Umschlag ist eine offene Liste an Parametern, vorgeschrieben sind Informationen über den Absender und den Empfänger (Abbildung 26).

Parameter	Beschreibung
`:destination`	AID des Empfängers
`:sender-details`	AID des Absenders

Abbildung 26: Brief-Parameter

6.9. Zusammenfassung

Die CASA Agent Language (CAL) ist eine Sprache zur Wissensrepräsentation, die sämtliche Arten von Wissen umfaßt, die für die in Kapitel 4 spezifizierte Standardarchitektur zur Verhaltenssteuerung von Agenten benötigt werden. Sie dient als logische Beschreibungsebene zur Spezifikation, Programmierung und Analyse von Agenten und kann in einem geeigneten Format direkt von den Agenten zur Wissensrepräsentation verwendet werden.

Die semantische Grundlage von CAL bildet eine dreiwertige Logik, die die klassische Prädikatenlogik um die explizite Berücksichtigung von unvollständigem Wissen durch den zusätzlichen Wahrheitswert *unbekannt* erweitert. Die bezeichnenden Symbole werden in Ontologien deklariert. Darauf aufbauend läßt sich in CAL deklaratives Wissen in Form von Fakten, Zielen und Intentionen, reaktives Wissen durch Regeln, prozedurales Wissen mit Handlungen und Diensten sowie kommunikatives Wissen als Sprechakte und Protokolle ausdrücken.

Darüber hinaus bietet CAL dank der Typisierung von Termen Typsicherheit bei Attributen, Funktionen und Relationen gemäß ihrer Deklaration in Ontologien. Der formallogische Ansatz wird mit Objektorientierung kombiniert, indem Kategorien als Schemata zur Repräsentation von Objekten verwendet werden. Funktionen und Relationen sind über eine operationale Semantik zu definieren. Schließlich erlaubt Metawissen die Ergänzung von Informationen über bestimmte Wissensinhalte auf einer nicht-logischen Ebene, das bei der Speicherung und Verarbeitung genutzt werden kann.

Auf der prozeduralen Ebene erlauben Operatoren eine flexible Handlungskontrolle aufgrund der formalen Beschreibung ihrer Bedingungen und Effekte und der Vielzahl verschiedener Arten von Operatoren. Schlußfolgerungen und abstrakte Operatoren sind Aktionen auf der logischen Ebene. Handlungsprimitive und Dienste sind grundlegende Aktivitäten, wobei letztere durch Interaktionen unter Rückgriff auf Operatoren für Protokolle und Sprechakte umgesetzt werden. Komplexe Handlungsabläufe lassen sich aus Operatoren zusammensetzen, entweder durch prozedurale Verknüpfung in Skripten oder durch logische Verkettung in Plänen.

CAL ist somit eine umfassende Sprache, die alle wesentliche Aspekte zur Spezifikation von und zur Verwendung durch wissensbasierte Agenten abdeckt. Ontologien stellen gemeinsam nutzbare Terminologien zur Verfügung, was zur Interoperabilität von Agenten unverzichtbar ist. Das deklarative Wissen der Agenten ist in einem ausdrucksstarken Formalismus ausgedrückt und das prozedurale Wissen erlaubt eine flexible und autonome Verhaltenskontrolle.

7. Schluß

Abschließend wird nun eine Zusammenfassung der beschriebenen Agentenarchitektur CASA gegeben, wobei nochmals deren Besonderheiten herausgestellt werden. Es folgt ein kurzer Überblick über das am DAI-Labor implementierte Agenten-Toolkit JIAC IV, dessen Architekturanteil auf den Konzepten von CASA basiert. Nach einer kurzen Gegenüberstellung von CASA und JIAC IV mit anderen Agentensystemen endet diese Arbeit mit einem Ausblick auf deren weitere Entwicklung.

7.1. Zusammenfassung

CASA ist eine Architektur für Multiagentensysteme, die ausgerichtet ist auf den Einsatz in offenen, dynamischen, verteilten Anwendungsdomänen. Um den besonderen Anforderungen derartiger Bereiche gerecht zu werden, basiert CASA auf den Kernkonzepten Modularität, Flexibilität und Interoperabilität. Erreicht werden diese durch einen komponentenbasierten Aufbau, eine wissensbasierte Kontrolle und durch dienstbasierte Interaktionen.

Bestandteile von CASA

Die Architektur von CASA umfaßt die drei Ebenen Komponenten, Agenten und Gesellschaften (Abbildung 27). Agenten werden gemäß einer Kontrollarchitektur aus Komponenten zusammengesetzt. Die Abhängigkeiten zwischen den Komponenten werden über Rollen beschrieben und die Interaktion zwischen den Komponenten geschieht über den Austausch von Nachrichten. Gesellschaften setzen sich unter Verwendung der Infrastruktur aus Agenten zusammen. Die Interaktionsmöglichkeiten zwischen den Agenten werden über Dienste beschrieben und die Interaktionen über den Austausch von Sprechakten durchgeführt. Die Wissensrepräsentationssprache ist angepaßt an das Kontrollschema der Standardarchitektur und bildet den Inhalt von Nachrichten und Sprechakten wie auch der Steuerungsprozesse auf allen Ebenen.

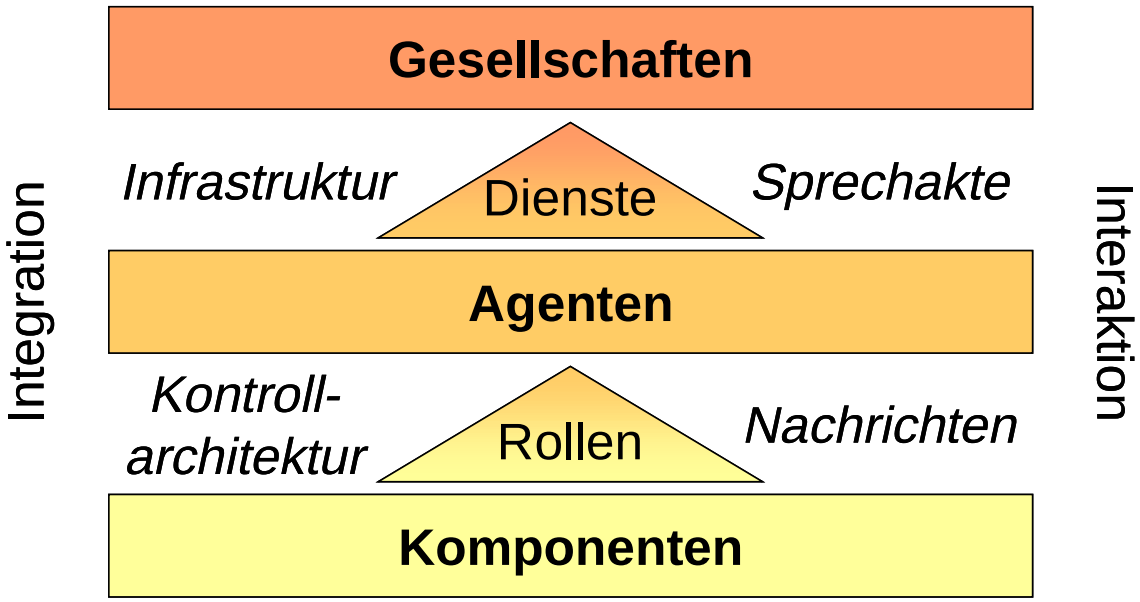

Abbildung 27: Designebenen von CASA

Die Komponentenarchitektur von CASA erlaubt einen modularen Aufbau von Agenten. Diese werden zusammengesetzt aus wiederverwendbaren Komponenten, die jeweils eine bestimmte Funktionalität in sich kapseln. Als Schnittstellen der Komponenten untereinander dienen Rollen, die deren Eigenschaften und Interaktionsmöglichkeiten spezifizieren. Auf diese Weise werden die Komponenten anonymisiert und so voneinander entkoppelt. Die Interaktionen zwischen den Komponenten geschehen über einen Nachrichtenmechanismus relativ zu deren Rollen, so daß Abhängigkeiten weder zeitlich noch bezüglich der konkreten Instantiierung einer Rolle entstehen. Darüber hinaus dienen Rollen der Definition von abstrakten Kontrollarchitekturen, da sich über sie deren Struktur und interne Abläufe implementierungsunabhängig festlegen lassen. Realisiert ist die Komponentenarchitektur durch besondere Infrastrukturkomponenten, die die Komponenten verwalten, Interaktionen über Nachrichten ermöglichen und die Aktivitäten der Komponenten kontrollieren.

Zur Steuerung des Verhaltens von Agenten wird ein wissensbasiertes Kontrollschema vorgeschlagen, das reaktives, deliberatives und interaktives Verhalten miteinander verbindet und eine autonome, flexible Kontrolle erlaubt. Dieses Schema wird über eine Menge von Rollen und Nachrichtentypen konkretisiert durch die Standardarchitektur von CASA, die in mehrere Rollengruppen unterteilt ist. Die Agentenhülle stellt dabei die Infrastruktur der Komponentenarchitektur bereit. Auf der Wissensebene dient die Wissensbasis der Speicherung von Wissen und die Kontrolleinheit der Nutzung dieses Wissens zur Verhaltenssteuerung. In der Peripherie sind Zusatzfunktionalitäten und der Übergang zur nicht-logischen Ebene angesiedelt.

Agenten lassen sich in CASA zu Gesellschaften zusammenfügen, die eine flexible und dynamische Zusammenarbeit der Agenten untereinander ermöglichen. Dabei

bietet das Dienstschema eine einheitliche Grundlage zur Planung und Durchführung von Interaktionen. Die Kommunikation verwendet offene und feste Standards, um eine flexible Interoperabilität zu gewährleisten. Das Metaprotokoll erfaßt generische Anteile von Dienstnutzungen, während eingebettete Protokolle die Flexibilität für spezifische Aspekte wahren. Dienstoperatoren beschreiben die Interaktionsmöglichkeiten und Protokolloperatoren deren Umsetzung durch kommunikative Handlungen. Konstituiert werden Agentengesellschaften durch eine Infrastruktur, die Agenten und Dienste verwaltet und eine Kommunikationsunterstützung bietet.

Mit der Wissensrepräsentationssprache CAL lassen sich Agenten über ihr Wissen und Verhalten auf einer logischen Ebene beschreiben. Zugleich kann sie von Agenten in einem geeigneten Format zur Wissensrepräsentation, Verhaltenssteuerung und Kommunikation verwendet werden. CAL besteht aus mehreren Sprachebenen für Terminologien, für Aussagen in einer ternären Prädikatenlogik und für die Kommunikation. In ihr lassen sich deklaratives, reaktives, prozedurales und interaktives Wissen angepaßt an die Mechanismen der Standardarchitektur ausdrücken. Durch die Kombination von logischen Formalismen und Techniken von Programmiersprachen wie Typisierung, Objektorientierung und Skripte bietet CAL eine einheitliche und ausdrucksstarke Sprache, die sowohl die Grundlage von Verhaltenssteuerung und Interaktionen bildet, als auch die Verbindung zwischen diesen darstellt.

Konzepte von CASA

CASA beruht auf der Kombination und Integration verschiedener Konzepte, um die Offenheit, Verteiltheit und Heterogenität bestimmter Anwendungsfelder beherrschbar zu machen. Zentral sind dabei die Modularität des Aufbaus, die Interoperabilität verschiedener Bestandteile und die Flexibilität der ablaufenden Prozesse. Diese und andere Konzepte finden sich in der Regel auf allen Ebenen von CASA – der Agentenstruktur, dem Agentenverhalten und der Agentengesellschaft – wieder.

Die Modularität erlaubt das variable Zusammensetzen verschiedener wiederverwendbarer und anpaßbarer Bestandteile je nach Bedarf und Anforderungen. Erreicht wird sie durch den Komponentenaufbau der Agenten einerseits und die dynamische Agenteninfrastruktur andererseits, die beide bei der Erstellung wie zur Laufzeit eine frei wählbare und veränderbare Konfiguration ermöglichen. Das System ist somit skalierbar, da sich die Größe und der Funktionsumfang von Agententypen und -gesellschaften an der jeweiligen Aufgabe ausrichten läßt. Zudem sind Anwendungen offen, da sie keine a priori festgelegten Agenten und Funktionalitäten zu umfassen brauchen, sondern sich noch zur Laufzeit erweitern und umstrukturieren lassen. Schließlich bildet auch der formallogische Aufbau des Wissens eine Form von Modularität aufgrund des Bedeutungsatomismus, nach dem sich die Bedeutung zusammengesetzter Ausdrücke alleine aus der Struktur eines Ausdrucks und der Bedeutung der enthaltenen Bestandteile ableiten läßt.

Die Interoperabilität gewährleistet die Kompatibilität der so zusammengesetzten Bestandteile und damit das Funktionieren des Gesamtsystems, ohne direkte, starre Abhängigkeiten zwischen Teilsystemen zu fordern. Sie ist somit eine unabdingbare Voraussetzung für die Modularität. Das formale Wissen dient dabei als einheitliches Format für den Informationsaustausch zwischen Komponenten wie zwischen Agenten, wobei lediglich die Ontologien einen variablen Anteil darstellen, während die verschiedenen Sprachebenen über eine gemeinsam zugrundeliegende Logik verbunden sind. Die Schnittstellen zwischen den Komponenten werden über Rollen spezifiziert, die auf die Funktionalität und das Interaktionsverhalten beschränkt sind. Zwischen den Agenten übernimmt diese Aufgabe der Dienst als abstrakte Beschreibung möglicher Interaktionen. Die Durchführung von Dienstnutzungen verwendet standardisierte Kommunikationssprachen, und ihr Verlauf wird durch Protokolle geleitet.

Interoperabilität und Modularität von Komponentenansatz und Agenteninfrastruktur unterstützen zusammen dynamische, verteilte Anwendungen. Die Verteiltheit kann dabei sowohl funktional in Form von Kompetenzen und Fähigkeiten, als auch bei Agentengesellschaften räumlich über ein Netzwerk bestehen. Die Dynamik ergibt sich vor allem aus der Nutzung der Offenheit von Agentengesellschaften vermittels der Infrastrukturfunktionalitäten, über die zur Laufzeit Dienste und Interaktionspartner nach Bedarf aus dem vorhandenen Angebot ausgewählt werden können.

Die Flexibilität ermöglicht es nun den Agenten, diese Offenheit und Dynamik zur Erfüllung ihrer Aufgaben zu nutzen. Das in der Standardarchitektur umgesetzte wissensbasierte Verhaltensschema erlaubt dabei eine autonome Kontrolle, die sich an die jeweiligen Gegebenheiten anpaßt. Veränderte Situationen können aufgrund der Reaktivität sofort erkannt und behandelt werden. Das zielgerichtete Verhalten kann durch die logische Verkettung von Operatoren jeweils geeignete Handlungsabläufe neu erstellen und dabei das gerade verfügbare Dienstangebot berücksichtigen. Bei Interaktionen erlaubt die Auswahlphase des Metaprotokolls eine dynamische Anbieterauswahl über Verhandlungen.

Gegenüberstellung von Komponenten und Agenten

Konzeptuell wie strukturell gibt es in CASA viele Gemeinsamkeiten zwischen dem Aufbau einzelner Agenten und Agentengesellschaften (vgl. Abbildung 28). Beide sind zusammengesetzt aus gekapselten Modulen, von denen einige die Infrastruktur für die übrigen bereitstellen. Die Interaktion zwischen den Modulen geschieht durch den Austausch von strukturierten Informationen und die Interoperabilität wird durch die Verwendung formaler Sprachen gewährleistet. Auf einer detaillierteren Betrachtungsebene werden jedoch die Unterschiede deutlich, aufgrund derer Agenten mehr Dynamik und Flexibilität erlauben, während Komponenten eine mehr klassische Programmierung mit expliziten Kontrollstrukturen unterstützen.

Modul	Komponente	Agent
Rollenspezifikation	Nachrichentypen	Dienste
Rollenabhängigkeiten	statisch	dynamisch
Interaktion	Nachrichten	Dienstnutzungen
Interaktionsverlauf	Nachrichtenschemata	Protokolle
Verhältnis	Aufruf / Ergebnis	Anbieter / Nutzer
Ausführung	kontrolliert	autonom

Infrastuktur	Agentenhülle	Agentenplattformen
Verwaltung der Module	Agentenkern	Agent Management System
Interaktionsdurchführung	Nachrichtenzustellung	Message Transport System, Transportkomponent
Interaktionsvermittlung	–	Directory Facilitator
Ausführungskontrolle	Kontrollzyklus	–

System	Agent	Agentengesellschaft
Verhaltensgrundlage	Wissen	Dienste
Grobstruktur	Wissensbasis, Kontrolleinheit, Peripherie	Anbieter, Nutzer, Vermittler
Infrastruktur	zentral	verteilt
Kontrolle	zentral	verteilt

Abbildung 28: Gegenüberstellung von Komponenten und Agenten

Die gekapselte Funktionalität von Komponenten wie von Agenten wird über Rollen beschrieben, die die jeweiligen Interaktionsmöglichkeiten spezifizieren. Bei Komponenten sind die Abhängigkeiten der Rollen untereinander jedoch statisch und schließen Redundanzen aus, bei Agenten können sich die Abhängigkeiten dynamisch ändern und erlauben eine Auswahl aus Alternativen. Die Interaktionen der Komponenten über Nachrichten beruhen auf einfachen, statischen Nachrichtenschemata. Agenten hingegen interagieren über Dienste, die über flexible und beliebig komplexe Protokolle genutzt werden. Entsprechend ist das Verhältnis zwischen Komponenten analog zu Methodenaufrufen und -ergebnissen, das der Agenten eines zwischen Anbieter und Nutzer. Während die Ausführung von Komponenten zentral kontrolliert wird, sind Agenten in ihrer Ausführung autonom.

Die Infrastruktur für das Komponentensystem wird in CASA durch die Agentenhülle realisiert, bei der Agentengesellschaft besteht sie aus einer oder mehreren Agentenplattformen. Die Infrastrukturfunktionalität erbringen jeweils ausgezeichnete Infrastrukturmodule, wobei Agentenkern und Agent Management System mit der Verwaltung sowie Nachrichtenzustellung und Message Transport System mit der Interaktionsdurchführung jeweils vergleichbare Aufgaben wahrnehmen. Hingegen benötigt nur die Agentengesellschaft einen Directory Facilitator zur Unterstützung dynamischer Interaktionen und nur der Agent einen Kontrollzyklus für die Ausführungskontrolle.

Auf der Systemebene bildet das Wissen der Agenten deren Verhaltensgrundlage, während in der Agentengesellschaft alleine die Dienste diese Rolle übernehmen. Demgemäß gliedert sich die Grobstruktur eines Agenten gemäß der Standardarchitektur in Wissensbasis und Kontrolleinheit zur Aufbewahrung und Verarbeitung von Wissen sowie die Peripherie als Schnittstelle, die Grobstruktur einer Agentengesellschaft in Anbieter und Nutzer sowie Vermittler zwischen diesen. Weiterhin sind Infrastruktur und Kontrolle eines Agenten zentral und in einer Agentengesellschaft auf mehrere Plattformen und Agenten verteilt.

Die Komponentenarchitektur von CASA erweitert somit den objektorientierten Ansatz um die Austauschbarkeit der Module zur Laufzeit, wozu eine stärkere Entkoppelung nötig wird. Die direkten funktionalen Abhängigkeiten zwischen den Komponentenrollen und die zentrale Kontrolle bleiben jedoch erhalten. Agenten hingegen besitzenen einen höheren Autonomiegrad und keine statischen, direkten Abhängigkeiten untereinander, wodurch eine entsprechende Dynamik und Flexiblität im Verhalten des Gesamtsystems erreicht wird.

CASA als Basis für Dienstplattformen

Eine zentrale Motivation von CASA war die Entwicklung eines Systems, das sich als Grundlage zur Realisierung von elektronischen Dienstplattformen beispielsweise im Internet eignet. Wie bereits beschrieben ist CASA aufgrund der Modularität und Interoperabilität besonders geeignet für Anwendungen in offenen, verteilten, heterogenen Umgebungen, wie sie charakteristisch sind für derartige Dienstplattformen. Darüber hinaus genügt CASA den in der Einleitung hergeleiteten Anforderungen der verschiedenen Teilnehmerrollen des elektronischen Handels.

Aus der Sicht der Anbieter von Diensten ermöglichen die Modularität und die sich daraus ergebende Konfigurierbarkeit eine sehr flexible Pflege und Umgestaltung ihres Angebots zur Laufzeit, ohne den laufenden Betrieb zu beeinträchtigen. Zusatzfunktionalitäten lassen sich leicht integrieren und neue Funktionalitäten durch die Kombination bestehender Dienste schaffen. Überhaupt verringert die Wiederverwendbarkeit den Entwicklungsaufwand durch den Rückgriff auf bestehende Komponenten und die einfache Aufrüstung laufender Systeme mit neu entwickelten Komponenten. Die Dienstvermittlung kann auf die einheitliche formale Beschreibung der Dienste zurückgreifen und so eine effektivere Suche nach Angeboten ermöglichen.

Das Dienstschema erlaubt dem Nutzer eine flexible und dynamische Auswahl und Kombination von Diensten. Der Agent ist dabei nicht einfach nur der Zugang zu den Diensten, sondern kann den Nutzer darüber hinaus unterstützen, indem er selbständig als dessen Stellvertreter agiert. Die Flexibilität und Autonomie der Agenten bei der Verhaltenssteuerung und den Interaktionen ermöglicht allen der genannten Rollen einen höheren Automatisierungsgrad, da die Agenten auch komplexere Aufgaben eigenständig bearbeiten können. Schließlich bietet sich auch der Dienst als Interaktionseinheit zwischen den Agenten an als Grundlage der

geschäftlichen Transaktionen, die sich über das Dienstkonzept protokollieren und den jeweiligen Akteuren genau zuordnen lassen.

Besonderheiten von CASA

Ein besonderes Merkmal von CASA ist das durchgängige Baukastenprinzip. Auf allen Ebenen erlauben Modularität und Interoperabilität ein freies Zusammensetzen und vielfältiges Kombinieren einzelner Bausteine, so daß sich unter Ausnutzung von deren Wiederverwendbarkeit Anwendungen paßgenau und mit geringem Aufwand erstellen lassen. Dieses Prinzip kommt nicht nur beim Design zum Tragen, sondern auch noch zur Laufzeit. Dank der Komponentenarchitektur und ihren Schnittstellenspezifikationen über Rollen können Agenten und deren Komponenten ohne Unterbrechung ihres Einsatzes in einer Anwendung umkonfiguriert und so an geänderte Umstände und Aufgaben angepaßt werden. Fähigkeiten und Komponenten lassen sich nach Bedarf hinzufügen, entfernen und übergangslos austauschen.

Auf der Multiagentenebene bietet das Dienstkonzept ein einheitliches Schema für Interaktionen zwischen Agenten, indem es eine zusätzliche Standardisierungsebene oberhalb der Kommunikation bereitstellt. Die Beschreibung von Diensten mit Hilfe von Planoperatoren berücksichtigt den deklarativen wie den prozeduralen Aspekt auf eine bewährte formale Weise. Sie erlaubt einerseits die Verwaltung von Diensten durch die Agenteninfrastruktur, andererseits die nahtlose Integration der Nutzung und Durchführung von Diensten in eine wissensbasierte Kontrollstruktur, wie sie durch die Standardarchitektur umgesetzt wird. Dabei wird ein Dienst analog zu anderen Handlungen aufgefaßt, so daß dieselben Mechanismen zur Auswahl und Verknüpfung wie bei allen Operatoren verwendet werden können, während die Ausführung an einen anderen Agenten delegiert wird. Die generischen Anteile der Durchführung von Interaktionen behandelt das Metaprotokoll für Dienstnutzungen, in das spezifische Unterprotokolle eingebettet werden. Das Dienstkonzept integriert auf diese Weise Kommunikation, Verhaltenskontrolle und Agenteninfrastruktur zu einem Interaktionsschema, das die Balance wahrt zwischen der Flexibilität für eine hinreichende Offenheit einerseits und der Standardisierung zur nötigen Interoperabilität andererseits.

Schließlich bietet CAL eine umfassende und strukturierte Wissensrepräsentationssprache für das Design und die Programmierung von Agenten. Die zugrundeliegende ternäre Logik dient dabei als einfaches Mittel zur Berücksichtigung des unvollständigen Wissens von Agenten.

7.2. Das Agenten-Toolkit JIAC IV

Am DAI-Labor der TU Berlin wurde in mehreren Projekten im Auftrag der T-Nova Deutsche Telekom Innovationsgesellschaft mbH das Agenten-Toolkit JIAC IV (Java Intelligent Agent Componentware, Version 4) [Fricke et al. 2001] mit der Implementierungssprache Java [Java] entwickelt. Der Architekturanteil dieses Systems basiert auf den Konzepten von CASA.

Insgesamt war es die Zielsetzung der Entwicklung von JIAC IV, eine vollständige Umgebung für die Entwicklung und den Einsatz von Multiagentensystemen bereitzustellen. Neben der Agentenarchitektur umfaßt das Toolkit deshalb auch eine Methodologie und Werkzeuge zur Entwicklung von Agentenanwendungen sowie eine Agenteninfrastruktur als Laufzeitumgebung und Schnittstellen zur Nutzung und Administration laufender Anwendungen (Abbildung 29).

Architektur

Das Grundgerüst von Agenten ist in JIAC IV gemäß der in Kapitel 3 beschriebenen Komponentenarchitektur realisiert. Die Komponenten Agentenkern, Kontrollzyklus und Nachrichtenzustellung bilden zusammen die interne Infrastruktur des Agenten, vermittels derer die Interaktion über Nachrichten zwischen Komponenten relativ zu ihren Rollen möglich ist. Weiterhin gibt es für den Zugriff auf diese Infrastruktur und zur Verwaltung durch den Agentenkern eine abstrakte Komponente, auf deren Funktionalität die konkreten Komponenten aufbauen.

Als Basis für wissensbasierte Kontrollarchitekturen ebenso wie als Inhalt für die Kommunikation ist die Wissensrepräsentationssprache aus Kapitel 6 (mit geringfügigen Änderungen) durch Datenstrukturen in Java realisiert, aus denen sich Inhalte der verschiedenen Wissenstypen zusammensetzen lassen. Die Umsetzung der Standardarchitektur aus Kapitel 4 erfolgt durch Klassen für die entsprechenden

JIAC IV Agenten-Toolkit		
Architektur	*Entwicklung*	*Einsatz*
Komponentenarchitektur ▪ Komponenten ▪ Rollen ▪ Nachrichten	*Methodologie* ▪ Analyse ▪ Design ▪ Implementierung ▪ Evaluierung	*Infrastruktur* ▪ Marktplatz ▪ Agentenregister ▪ Dienstregister ▪ Migration ▪ Sicherheit
Kontrollarchitektur ▪ Wissenssprachen ▪ Reaktivität ▪ Zielgerichtetheit ▪ Interaktionen	*Werkzeuge* ▪ Sprachkompiler ▪ Konfiguration ▪ Debugger	*Administration* ▪ Endnutzerzugang ▪ Monitor ▪ Konfiguration

Abbildung 29: Überblick über das Agenten-Toolkit JIAC IV

Rollengruppen, Rollen, und Nachrichtentypen. Für die spezifizierten Rollen der Wissensbasis und der Kontrolleinheit sind Komponenten implementiert, wobei in der Regel eine Komponente der Kontrolleinheit jeweils auch die Rolle der zugeordneten Wissensbasis übernimmt, so daß ein effizienter direkter Zugriff zur Verwaltung des Wissens möglich ist.

Auch aus dem Bereich der Peripherie werden Komponenten mit Standardfunktionalitäten bereitgestellt. An Protokollen für den Transport von Sprechakten werden momentan TCP/IP, das abgesicherte SSL und eine direkte Übertragung innerhalb einer JVM durch jeweils eigene Komponenten unterstützt. Managementkomponenten nutzen die Überwachung von Wissensänderungen zur Protokollierung interner Vorgänge und erfassen darauf aufbauend Daten zur Verrechnung kommerzieller Dienste. Die Vertraulichkeit und Authenzität der Kommunikation sowie die Zugriffskontrolle auf Dienste gewährleisten bei Bedarf Sicherheitskomponenten über Verschlüsselung, Zertifikate und Zugangsberechtigungen.

Entwicklung

Die Entwicklungsmethodologie für JIAC IV umfaßt die vier klassischen Phasen Analyse, Design, Implementierung und Evaluierung und orientiert sich an bestehenden Ansätzen [Burmeister 1996, Wooldridge et al. 2000]. Die Analyse ist noch weitgehend architekturunabhängig, da sie auf dem allgemeinen Konzept des Multiagentensystems beruht. Dabei werden zunächst Agententypen anhand ihrer Rollen innerhalb der Agentengesellschaft definiert, wozu jeweils deren Aufgaben sowie Interaktionsbedarf und -angebot für eine Rolle identifiziert werden. Anschließend werden diese Agententypen zueinander in Bezug gesetzt, indem in Form von Interaktionsmustern die Organisationsstruktur und die Abläufe zur Umsetzung einzelner Aufgaben festgelegt werden. Dazu gehört auch die Verteiltheit des Systems über mehrere Agentenplattformen sowie die Migration zwischen diesen.

Das Design beinhaltet die Spezifikation der Agenten auf der Wissensebene, wozu die Wissensrepräsentationssprache verwendet wird. Der erste Schritt ist dabei die Domänenbeschreibung über Ontologien, um ein einheitliches Vokabular zur Wissensrepräsentation und für Interaktionen zu erhalten. Gemäß der Organisationsstruktur und der Agententypen der Analyse werden dann zunächst die Interaktionen über Dienste und Protokolle als Operatoren beschrieben. Schließlich werden aus den Agententypen konkrete Agenteninstanzen abgeleitet und deren Aufbau aus Komponenten festgelegt sowie deren Wissen und Fähigkeiten als Fakten, Regeln und Operatoren bestimmt.

Die Implementierung beginnt mit der Kompilierung des Wissens, wodurch die Design-Spezifikation auf die Programmiersprachenebene übertragen wird. Hierfür stehen spezielle Kompiler zur Verfügung. Diejenigen Komponenten, für die keine vorgefertigten Standardversionen verwendet werden sollen oder können, werden als Java-Klassen programmiert. Dies sind vor allem Anwendungskomponenten zur Umsetzung spezifischer Handlungsprimitive und zur Gewinnung aktuellen Wissens

über die Umgebung. Es folgt das Zusammensetzen der einzelnen Bestandteile zu einem Multiagentensystem, indem die Konfiguration der Agenten und der Plattformen (d.h. der zugehörigen Infrastrukturagenten) aufgestellt wird. Dies wird durch ein graphisches Werkzeug unterstützt, mit dem sich unter Nutzung der Modularität und Wiederverwendbarkeit Komponenten und Wissen aus Bibliotheken den Agenten zuordnen lassen.

Abschließend wird zur Evaluierung aus den einzelnen Konfigurationsangaben heraus das System gestartet und getestet. Gegebenenfalls sind auftretende Fehler zu analysieren und ihre Ursachen durch Änderung von Design oder Implementierung zu beheben. Neben Standardverfahren der zugrundeliegenden Programmiersprache Java können dazu auch spezifische Analysewerkzeuge von JIAC IV verwendet werden, die besondere Aspekte wie Verteiltheit und Wissensbasiertheit unterstützen. Zum einen gibt es die weiter unten beschriebenen Administrationswerkzeuge, die eine Überwachung und Kontrolle des laufenden Systems auch bereits zu Testzwecken erlauben. Zum anderen gibt es eine Debugger-Komponente, die per Konfiguration oder zur Laufzeit in einzelne Agenten eingefügt werden kann, um dessen Verhalten zu überwachen und zu kontrollieren. Sie bietet den Abruf und die Änderung von Konfiguration und Lebenszykluszustand und ermöglicht die Anzeige des internen Nachrichtenaustauschs, des Inhalts der Wissensbasis und des Ausführungszustands von Operatoren. Zudem kann der Agent im Einzelschrittbetrieb gesteuert werden.

Die beschriebene Methodologie ist bislang nur ein erster Ansatz, der momentan weiter ausgearbeitet wird. Ziel ist eine detaillierte und vollständig durch Werkzeuge unterstützte Vorgehensweise, die eine effiziente und strukturierte Entwicklung von Anwendungen mit JIAC IV erlaubt.

Einsatz

Als Laufzeitumgebung dienen in JIAC IV sogenannte Marktplätze, die durch einen ausgezeichneten Manager-Agenten konstituiert werden. Dieser verwaltet jeweils eine Agentenplattform mit den bei ihm registrierten Agenten. Zusammen mit anderen Agenten realisiert der Manager über eine Menge von Diensten die Infrastruktur für eine Agentengesellschaft. Diese Agenten sind letztlich normale Agenten mit der besonderen Anwendungsdomäne Agenteninfrastruktur. Wie die in Kapitel 5.2 beschriebene Infrastruktur orientiert sich auch die von JIAC IV an der FIPA-Spezifikation, eine volle FIPA-Konformität ist als optionaler Zusatz vorgesehen. Dementsprechend unterstützt die Infrastruktur von JIAC IV die Verwaltung von Agenten und Diensten sowie einen Standardweg für den Transport und die Aufbewahrung von Sprechakten. Zudem bietet der Manager-Agent einen Dienst zur Migration auf einen anderen Marktplatz an.

Zusätzlich zur Agenteninfrastruktur besitzt JIAC IV noch eine Sicherheitsinfrastruktur. Die Authenzität von Agenten läßt sich über Zertifikate prüfen, wobei die dazu benötigten Referenzzertifikate über Agenten zur Zertifikatsverteilung besorgt

werden können, falls sie der Agent nicht selber kennt. Das Ausstellen von Zertifikaten übernimmt ein Agent als Zertifikatsautorität, der in der Regel auf einem eigenen, besonders gesicherten Marktplatz residiert. Aufgrund von Vertrauensverhältnissen zwischen Marktplätzen entscheidet der jeweils andere Manager-Agent, ob bei einer Migration der Ursprungs- bzw. Zielmarktplatz als sicher anzusehen ist. Für Vertragsabschlüsse gibt es einen Agenten als Trusted Third Party, über den das Bestehen von Verträgen sicher nachweisbar ist.

Für die Interaktion zwischen Agentensystem und menschlichem Benutzer stellt JIAC IV ein generisches Schema bereit, über den ein Zugriff auf Agentenfunktionalitäten auch über ein Netzwerk und von unterschiedlichen Endgerätetypen aus möglich ist. Dabei präsentiert ein sogenannter Alter Ego-Agent, der den Dienstanbieter vertritt, jeweils einen oder mehrere kombinierte Dienste über eine graphische Oberfläche und übersetzt die von ihm durchgeführte Dienstnutzung beim Anbieteragenten in eine Interaktion mit dem Benutzer. Die Benutzerschnittstelle wird durch einen speziellen Navigatoragenten auf einem beliebigen Marktplatz angezeigt, über den auch die Suche und Anforderung von auf diese Weise zugänglichen Diensten vorgenommen wird. Alternativ ist auch eine Nutzung über das Internet in verschiedenen Formaten wie HTML und WML möglich.

Die Administration des laufenden Systems geschieht über besondere Dienste mit Alter Ego-Schnittstelle, wobei die Zugriffsberechtigung auf den Systemadministrator oder den Besitzer eines Agenten beschränkt werden kann. Sie ist nur möglich, wenn die betreffenden Agenten diese Dienste anbieten und sie auch erbringen können. Über die Administrationsdienste kann die Konfiguration und der Zustand von Agenten und Marktplätzen abgerufen und verändert werden. Über den Manager-Agenten lassen sich auf dessen Marktplatz Agenten erzeugen oder beenden sowie per Migration auf einen anderen Marktplatz transferieren. Zudem können Agenten in einen anderen Lebenszykluszustand versetzt und so beispielsweise suspendiert oder aktiviert werden. Die direkte Administration einzelner Agenten beinhaltet das Hinzufügen, Entfernen und Austauschen von Komponenten und Wissen. Auf diese Weise lassen sich die Fähigkeiten und Dienste von Agenten zur Laufzeit ändern.

Zur Visualisierung von Marktplätzen gibt es einen Monitor, über den ein Manager-Agent den ihm zugeordneten Marktplatz und die Vorgänge auf diesem darstellt. Er zeigt an, welche Agenten sich momentan auf dem Marktplatz befinden, ob es sich um den Manager, einen stationären oder einen mobilen Agenten handelt und in welchem Lebenszykluszustand sich ein Agent befindet. Zudem wird der Sprechaktverkehr der Agenten innerhalb des Marktplatzes wie auch zu anderen Marktplätzen durch Verbindungslinien zwischen den kommunizierenden Agenten nachvollziehbar gemacht.

Realisierte Anwendungen

Mit JIAC IV wurden bislang vor allem kleinere Test- und Beispielanwendungen realisiert, die nur isolierte oder einige wenige Eigenschaften des Systems nutzen und

deren prinzipielle Funktionstauglichkeit zeigen. Diese Anwendungen sind jedoch nur bedingt aussagekräftig zur Beurteilung von JIAC IV und dem der Architektur zugrundeliegenden CASA-Konzept, da deren Design auf umfassende, dynamische, verteilte Anwendungen hin abzielt und erst dann die Offenheit, Skalierbarkeit und Flexibilität des Ansatzes zum Tragen kommt.

Die einzige größere Anwendung ist bislang ein Verkehrstelematikszenario basierend auf Albayrak et al. [1998]. Dabei handelt es sich um eine Portierung einer mit einer anderen Agentenarchitektur entwickelten Anwendung, weshalb viele der besonderen Eigenschaften von JIAC IV nicht berücksichtigt wurden. Da JIAC IV in mehreren laufenden Projekten am DAI-Labor eingesetzt wird, wird es bald mehr Erfahrung mit der Realisierung von Anwendungen geben, so daß dann eine Beurteilung des Systems anhand des Einsatzes in der Praxis möglich sein wird.

7.3. Gegenüberstellung mit anderen Architekturen

Ein tiefergehender Vergleich zwischen CASA und den in Kapitel 2.6 vorgestellten Architekturen ist an dieser Stelle sicherlich nicht möglich, da ein solcher neben der konzeptionellen Ebene auch die praktische Erfahrung im Umgang mit den implementierten Systemen bei der Programmierung und im Einsatz in Betracht ziehen müßte. Die folgende Gegenüberstellung beschränkt sich deshalb auf die zentralen Konzepte und Besonderheiten der Architekturen, wobei die abstrakte Architektur CASA und das darauf aufbauend implementierte System JIAC IV gemeinsam betrachtet werden. Abbildung 30 gibt einen zusammenfassenden Überblick über Eigenschaften und Konzepte von CASA / JIAC IV, PRS, INTERRAP, ZEUS und AgentBuilder.

Sämtlichen dieser Architekturen gemeinsam sind die Verwendung symbolischer Methoden zur Wissensrepräsentation und zur Verhaltenskontrolle sowie die Kombination von reaktivem, deliberativem und interaktivem Verhalten, wenn auch mit unterschiedlicher Gewichtung. Nur JIAC IV, ZEUS und Agentbuilder bieten dabei vollständige Toolkits, die das gesamte Spektrum von Architektur, Anwendungsentwicklung und Einsatz abdecken, während PRS und INTERRAP sich vorwiegend auf die Kontrollarchitektur konzentrieren.

Ein wichtiger Vorteil von CASA ist der offene, modulare Aufbau, der sich über alle Ebenen erstreckt und vielfältige, flexible Gestaltungsmöglichkeiten bei der Anwendungserstellung wie auch im Einsatz bietet. Das Dienstschema vereinheitlicht Interaktionen zwischen Agenten hinsichtlich ihrer Beschreibung, Planung, Koordinierung und Durchführung. Auf diese Weise ermöglicht es ein hohes Maß an Interoperabilität und eine enge Integration in die Verhaltenssteuerung, ohne dabei die Offenheit und Flexibilität der Agentengesellschaft einzuschränken. JIAC IV besticht zudem auch durch seinen Umfang an Zusatzfunktionen wie der Agenten-

mobilität, den Entwicklungswerkzeugen, dem Sicherheitskonzept, den Administrationsmöglichkeiten und dem Endnutzerzugang. Eine Schwäche ist gegenwärtig noch das zu rudimentäre Kontrollmodell der Verhaltenssteuerung, das zwar ein weites Spektrum abdeckt, dabei jedoch zu unspezifisch verbleibt (vgl. auch den Ausblick in Kapitel 7.4).

Die Stärke von PRS ist die effiziente Kombination von Reaktivität und Zielgerichtetheit, die sich aus dem einfachen, aber effektiven Kontrollmodell ergibt. Die Metakontrolle erlaubt dabei eine flexible Steuerung, ohne ein aufwendiges Kontrollschema vorzugeben, das dafür jedoch für jede Anwendung neu entwickelt oder angepaßt werden muß. Nachteilig erscheint vor allem der monolithische Aufbau um einen zentralen Interpreter sowie die mangelnde Unterstützung von Interaktionen, die auf einen statischen Austausch von Nachrichten beschränkt ist.

Die Modularisierung von INTERRAP durch die Unterteilung in Ebenen wie auch innerhalb der Ebenen ermöglicht die Parallelität und damit die Unabhängigkeit von Prozessen, die auf bestimmte Funktionalitäten spezialisiert sind. Interaktionen werden dabei durch eine eigene Ebene für Multiagentenplanung und -koordinierung unterstützt. Die Verhaltenskontrolle besitzt eine detailliert ausgearbeitete formallogische Grundlage [Jung 1999]. Es fehlt jedoch eine Einbettung der Kontrollarchitektur in eine konkrete Infrastruktur für Agentengesellschaften.

ZEUS besitzt eine stark planungszentrierte Verhaltenssteuerung, in die Kommunikation und Interaktion fest integriert sind. Die Agenteninfrastruktur erlaubt dynamische Gesellschaften. Die Entwicklung und der Einsatz von Anwendungen wird durch eine Reihe von Werkzeugen unterstützt.

Beim AgentBuilder ist vor allem die Erstellung von Agenten aus einer formalen Spezifikation heraus erwähnenswert, die auf einem allgemeinen Modell der Verhaltenssteuerung basiert und vollständig über eigene Werkzeuge erfolgt. Der Interpreter zur Umsetzung dieser Spezifikation ist jedoch recht statisch und vor allem auf reaktives Verhalten nach festen Regeln hin ausgerichtet. Die vorhandenen Interaktionsmechanismen sind nur lose in die Verhaltenskontrolle integriert und erlauben aufgrund der fehlenden Agenteninfrastruktur nur statische Interaktionen.

	CASA / JIAC IV	PRS	INTERRAP	Zeus	AgentBuilder
Methodologie	Analyse Design Implementierung Evaluierung	–	–	Domänenanalyse Agentendefinition Aufgabendefinition Organisationsstruktur Agentenkoordination Implementierung	Bereichsanalyse Agentengesellschaft Agentenspezifikation Agentengenerierung
Entwicklungswerkzeuge	Sprachkompiler Agentenerstellung	–	–	Editoren Kode-Generator	Editoren Projektmanager
Laufzeitwerkzeuge	Visualisierung Debugging Konfiguration	–	–	Visualiser Agent Analysewerkzeuge Konfiguration	Agency-Manager Visualisierung Debugging
Endnutzerzugang	Alter-Ego Agenten Navigator-Agent	–	–	–	–
Wissensrepräsentation	Ontologien ternäre Logik	BDI	BDI	Ontologien	Ontologien Agent-0
Kontrollarchitektur	modular, offen	zentral	hierarchisch	modular, fest	zentral
Verhaltensspektrum	reaktiv deliberativ interaktiv	reaktiv deliberativ	reaktiv deliberativ interaktiv	deliberativ interaktiv	reaktiv deliberativ interaktiv
Verhaltenskontrolle	prozedural Planung	prozedural Metakontrolle	prozedural Planung	Planung	regelbasiert Planung (optional)

	CASA / JIAC IV	PRS	INTERRAP	Zeus	AgentBuilder
Kommunikation	Sprechakte FIPA-ACL	Nachrichten	Sprechakte	Sprechakte KQML, FIPA-ACL	Sprechakte KQML
Adressierung	direkt, indirekt	direkt	–	direkt	direkt
Transport	mehrere Protokolle	–	–	TCP-IP	–
Interaktionspartner	dynamisch	statisch	dynamisch	dynamisch	statisch
Interaktion	Protokolle Verhandlungen Dienste	–	Protokolle Verhandlungen Multiagentenplanung	Protokolle Organisationsschemata	Protokolle
Agenteninfrastruktur	Agentenregister (AMS) Dienstregister (DF) Kommunikation (MTS)	–	–	Name Server (Agenten) Facilitator (Dienste)	–
Mobilität	Agentenmigration	–	–	–	–
kommerzielle Dienste	Gebührenerfassung Gebührenkontrolle	–	–	–	–
Sicherheit	SSL-Kommunikation Verschlüsselung Zertifikate Autorisierung Vertrauensverhältnisse	–	–	–	Autorisierung

Abbildung 30: Gegenüberstellung von Agentenarchitekturen

7.4. Ausblick

Den Schluß bildet nun ein Ausblick auf Möglichkeiten zur Weiterentwicklung und Erweiterung von CASA. Ein zentraler Punkt dabei ist eine detaillierte Ausarbeitung der Komponenten der Standardarchitektur, um eine exakte und aufgabengerechte Spezifikation des Verhaltens von autonomen Agenten zu ermöglichen. Auf der Multiagentenebene ist eine weitere Flexibilisierung des Dienstschemas denkbar, ohne daß durch die zusätzliche Offenheit die gewonnene Interoperabilität beeinträchtigt werden darf. Weiterhin kann die Standardarchitektur um weitere Rollen beispielsweise für Lernen erweitert werden, alternative Kontrollarchitekturen lassen sich im Rahmen der Komponentenarchitektur realisieren, und die Voraussetzungen für eine Interoperabilität auch über Systemgrenzen hin weg sollten erkundet werden.

Verhaltenskontrolle

Bei der Konkretisierung der Verhaltenssteuerung sind die besonderen Anforderungen zu berücksichtigen, die sich aus dem zugrundegelegten Agentenbegriff als autonome, interagierende Spezialisten ergeben. In die flexible und eigenständige Kontrolle des Verhaltens muß dabei der Multiagentenaspekt innerhalb einer Gesellschaft interagierender Agenten und die Stellvertreterfunktion bei der Erfüllung von Aufgaben für einen menschlichen Benutzer integriert werden.

Da der Agent autonom handelt, benötigt er eine entsprechende Entscheidungskompetenz, um sein Verhalten auf seine Aufgabe hin auszurichten und diese möglichst optimal zu erfüllen. Dabei spielen objektive Aspekte im Sinne einer rationalistischen Nutzenoptimierung eine Rolle, aber auch subjektive Anforderungen und Erwartungen des jeweiligen Benutzers. Insgesamt muß das Verhalten verläßlich, konsistent und effektiv sein, um eine Übertragung selbständig zu bearbeitender Aufgaben an Agenten zu rechtfertigen. Das Entscheidungsverhalten eines Agenten spiegelt sich dabei vorwiegend in der Auswahl von Zielen und Handlungen sowie in deren Koordinierung wider. Da zu den Handlungen auch die Interaktionen gehören, fallen hierunter auch die Auswahl von Diensten und ihren Anbietern sowie Entscheidungen bei der Durchführung von Diensten im Rahmen von Protokollen.

Für eine differenziertere Handlungsauswahl lassen sich die Operatoren um zusätzliche Parameter erweitern, die eine bessere Beurteilung von Alternativen erlauben. Dazu zählen zunächst die aufzuwendenden Kosten bei der Umsetzung einer Handlung wie der Verbrauch von Ressourcen oder die Ausführungsdauer, von denen jedoch nur wenige völlig domänenunabhängig und objektiv zu ermitteln

sind. Zudem kann die Erfolgswahrscheinlichkeit berücksichtigt werden, mit der eine Handlung ihr Ziel auch erreicht. Diese und andere Parameter können dabei fest vorgegeben sein oder kontextabhängig errechnet werden. Die Nutzung dieser Informationen geschieht entweder durch die jeweiligen Komponenten direkt, oder diese verwenden im Sinne einer Metakontrolle die vorhandenen Mechanismen zur Verhaltenssteuerung auch zur Entscheidungsfindung.

Eine flexiblere Planung und Durchführung von Interaktionen kann durch erweiterte Planungsmechanismen erreicht werden. Zur Koordinierung mehrerer Agenten lassen sich durch Multiagentenplanung Pläne erstellen, die die Handlungen mehrerer Agenten zugleich festlegen. Explizite, abstrakte Protokollbeschreibungen erlauben mehr Freiheit bei der Umsetzung einer Rolle innerhalb eines Protokolls, wenn diese als Gerüst zur Planung verwendet wird. Dafür wird jedoch eine Repräsentation von Sprechaktoperatoren benötigt, die deren Bedingungen und Effekte innerhalb des Protokolls ausdrücken können. Auf diese Weise lassen sich auch Verhandlungen und Verhandlungsstrategien flexibler gestalten.

Besonders hinsichtlich des Einsatzes als Stellvertreter des Kunden beim elektronischen Handel spielt die Anpassung des Agentenverhaltens an die individuellen Interessen, Präferenzen und Absichten des jeweiligen Auftraggebers eine wichtige Rolle. Nur durch eine derartige Personalisierung wird der Benutzer durch seinen Agenten auch adäquat vertreten und besitzt einen entsprechenden Einfluß auf dessen Entscheidungen. Dazu werden die Eigenheiten des Benutzers über ein Profil modelliert, das der Agent dann bei seinen Entscheidungen berücksichtigt. Umgekehrt muß der Agent auch im Nachhinein gegebenenfalls unter Bezugnahme auf das Benutzerprofil autonom getroffenen Entscheidungen erklären und rechtfertigen können, um so seine Handlungen für den Benutzer nachvollziehbar zu machen. Nur so kann das nötige Maß an Vertrauen in die Entscheidungen des Agenten und die Zuverlässigkeit ihrer Umsetzung erreicht werden, das die Überantwortung auch komplexer Aufgaben an eigenständig handelnde Agenten erlaubt.

Dienstschema

Bislang behandelt das Dienstkonzept Interaktionen streng atomar und damit isoliert voneinander. Deshalb wird die Koordinierung von mehreren voneinander abhängigen Dienstnutzungen nur unzureichend unterstützt, beispielsweise bei Interaktionen mit mehr als zwei Beteiligten. Um Abhängigkeiten zwischen Diensten explizit zu machen, können diese einerseits in einer geeigneten Weise mit in die Dienstbeschreibung aufgenommen werden, andererseits können Protokolloperatoren um explizite Koordinierungsmechanismen erweitert werden.

Um Dienstnutzungen besser planbar zu machen, können wie bei anderen Planoperatoren zusätzliche Parameter aufgenommen werden, die die Umstände der Ausführung genauer beschreiben. Dazu zählen vor allem auch Parameter realer Dienste wie die Kosten, Bezahlungsmodalitäten, Qualitätsanforderungen, Garantieansprüche und rechtliche Grundlagen sowie die spezifischen Aspekte des elektroni-

schen Handels wie die Absicherung. Bei Diensten können diese Parameter auch Gegenstand von Verhandlungen sein und in Vertragsabschlüssen festgehalten werden.

Bei der Kommunikation geben Sprechaktoperatoren innerhalb von Protokollen ein festes Muster für die auszutauschenden Sprechakte vor. Dies ist sehr effizient und fehlersicher, aber oft zu unflexibel. Alternativ werden Ansätze benötigt, die eine mehr inhaltlich orientierte Verarbeitung durch logische Interpretation erlauben und so auch dem Sprechaktgedanken besser gerecht werden.

Andere Erweiterungen

In vielen Anwendungsbereichen ist eine vollständige Vorgabe der benötigten Fähigkeiten eines Agenten nicht möglich, insbesondere wenn sich wie bei Multiagentengesellschaften das Umfeld dynamisch ändern kann. Durch Lernen läßt sich das Agentenverhalten an veränderte Gegebenheiten anpassen und angesichts von Erfahrung optimieren. Beispielsweise können die Werte für viele Parameter wie die Ausführungsdauer oder die Erfolgswahrscheinlichkeit, die als Entscheidungsgrundlage bei der Planung dienen, am ehesten statistisch erfaßt werden, da sie stark mit dem jeweiligen Umfeld variieren können. Einmal erstellte Pläne lassen sich wiederverwenden und gegebenenfalls verallgemeinern, so daß sie in ähnlichen Situationen nicht erneut erstellt zu werden brauchen. Andere Lerngelegenheiten sind die aktive Suche und Aktualisierung von Wissen über den gegenwärtigen Zustand der Umgebung und eine adaptive Benutzermodellierung, bei der das Benutzerprofil und damit das Agentenverhalten anhand des beobachteten Verhaltens des Benutzers oder expliziter Rückmeldungen angepaßt wird.

Die vorgeschlagene Standardarchitektur beschreibt nur eine Möglichkeit, aufbauend auf der Komponentenarchitektur ein Verhaltensschema für autonome, interagierende Agenten zu realisieren. Alternative Kontrollarchitekturen können unter teilweiser Verwendung der dort definierten Rollen oder mit völlig eigener Modularisierung definiert werden. Beispiele für alternative Kontrollschemata geben u.a. die in Kapitel 2.6 diskutierten Architekturen. Durch ein einheitliches Grundgerüst zur Realisierung verschiedener Architekturen ließen sich auch aussagekräftigere Vergleiche zwischen diesen durchführen und die Kompatibilität unterschiedlicher Ansätze ermitteln.

Ein letzter Punkt ist die Interoperabilität von Agenten über Systemgrenzen hinaus, wie sie von der FIPA angestrebt wird (vgl. Kapitel 2.4.1). Auch wenn hierfür erst noch die Voraussetzungen geschaffen werden müssen, die über standardisierte Sprachen für den Informationsaustausch hinaus Interoperabilität auf allen relevanten Ebenen – insbesondere der inhaltlichen – erlauben, ist dies ein wichtiges längerfristiges Ziel, damit Agenten sich unter freien Markt- und Wettbewerbsbedingungen durchsetzen und behaupten können.

Literatur

[Albayrak 1998] Sahin Albayrak (ed.); Intelligent Agents in Telecommunications Applications – Basics, Tools, Languages and Applications. Volume 36 of Frontiers in Articial Intelligence and Applications. IOS Press, January 1998.

[Albayrak & Kirsch 1998] Sahin Albayrak, Dietrich Kirsch; Agent-based Travel Information Service. in: [Albayrak 1998]

[Albayrak et al. 1998] Sahin Albayrak, Karsten Bsufka, Andreas M. Kirchwitz, Hans Schlenker, Kai Seidler, Ralf Sesseler; Agent-based Traffic Telematics Services. in: [Albayrak 1998]

[Arkin et al. 1987] Ronald C. Arkin, Edward M. Riseman, and Allen R. Hanson; ArRA: An architecture for vision-based robot navigation. in: Proceedings of the DARPA Image Understanding Workshop, Los Angeles, February 1987.

[Austin 1962] J. L. Austin; How to do Things with Words. Oxford University Press, 1962.

[Axelrod 1984] R. Axelrod; The Evolution of Cooperation. Basic Books, 1984.

[Bates et al. 1992] Joseph Bates, A. Bryan Loyall, W. Scott Reilly; Integrating Reactivity, Goals, and Emotion in a Broad Agent. Techical Report CMU-CS-92-142, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. May 1992.

[Bäumer et al. 1999] C. Bäumer, M. Breugst, S. Choy, T. Magedanz; Grasshopper – A universal agent platform based on OMG MASIF and FIPA standards. in: Proceedings MATA 99.

[Bellifemine et al. 1999] Fabio Bellifemine, Agostino Poggi, Giovanni Rimassa; JADE – A FIPA-compliant agent framework. in: Proceedings of PAAM 99.

[Bibel 1993] W. Bibel; Wissensrepräsentation und Inferenz. Vieweg, Braunschwieg, 1993.

[Bölöni & Marinescu 2000] Ladislau Bölöni, Dan C. Marinescu; An Object-Oriented Framework for Building Collaborative Network Agents. in: A. Kandel, K. Hoffmann, D. Mlynek, N. H. Teodorescu (eds), Intelligent Systems and Interfaces, Kluewer Publishing, 2000.

[Bratman 1987] Michael E. Bratman; Intentions, Plans, and Practical Reason. Cambridge, MA: Harvard U. Press, 1987.

[Brooks 1986] Rodney A. Brooks; A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation, 2(1):14-23, 1986.

[Bürckert et al. 1997] Hans-Jürgen Bürckert, Klaus Fischer, Gero Vierke; TeleTruck: A Holonic Fleet Management System. DFKI TM-97-03, Saarbrücken, 1997.

[Burmeister 1996] Birgit Burmeister; Models and Methodology for Agent-Oriented Analysis and Design. in: K. Fischer (Hrsg.) Working Notes of the KI'98 Workshop on Agent-Oriented Programming and Distributed Systems. DFKI Document D-96-06, Saarbrücken, 1996.

[Chaudhri et al. 1998] Vinay K. Chaudhri, Adam Farquhar, Richard Fikes, Peter D. Karp & James P. Rice; Open Knowledge Base Connectivity 2.0.3. (http://www.ai.sri.com/~okbc).

[Cohen & Levesque 1990] P. R. Cohen, H. J. Levesque; Intention is Choice with Commitment. Artificial Intelligence 42, 213-261, 1990.

[Colmerauer 1985] Alain Colmerauer; Prolog in 10 Figures. Communications of the ACM, 28(12), 1985.

[COM] Microsoft Com Technologie; http://www.microsoft.com/com

[CORBA] OMG Corba Standard; http://www.corba.org

[DARPA 1993] The DARPA Knowledge Sharing Initiative; Specification of the KQML Agent-Communication Language, 1993.

[Decker & Li 1998] Keith Decker, Jinjiang Li; Coordinated Hospital Patient Scheduling. Proceedings of the Third International Conference on Muli-Agent Systems (ICMAS'98), 1998.

[DeMichiel et al. 2000] Linda G. DeMichiel, L. Ümit Yalçinalp, Sanjeev Krishnan; Enterprise JavaBeans Specification, Version 2.0. Sun Microsystems, http://java.sun.com/products/ejb

[d'Inverno et al. 1997] Mark d'Inverno, David Kinny, Michael Luck, Michael Wooldridge; A Formal Specification of dMARS. Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages, ATAL'97, 1997.

[Dreyfus 1972] Hubert L. Dreyfus; What Computers Can't Do. Harper and Row, New York, 1972.

[Durfee & Lesser 1991] E. H. Durfee, V. R. Lesser; Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation. IEEE Transactions on Systems, Man, and Cybernetics, 21(5):1167-1183, September/October 1991.

[Durfee et al. 1989] E. H. Durfee, V. R. Lesser, D. D. Corkill; Trends in Cooperative Distributed Problem Solving. IEEE Knowledge and Data engineering, 1(1): 63-83, 1989.

[Erman et al. 1980] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, D. Raj Reddy. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. Computing Surveys, 12(2):213-253, June 1980.

[Ferber 1999] Jacques Ferber; Multi-Agent Systems – An Introduction to Distributed Artificial Intelligence. Addison-Wesley, 1999.

[Ferguson 1992] I. A. Ferguson; TouringMachines: An architecture for dynamic, rational, mobile agents. PhD, Universität University of Cambridge, 1992.

[Fikes & Nilsson 1971] R. E. Fikes, N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 5(2), 1971.

[Findler & Lo 1986] Nicholas V. Findler and Ron Lo; An examination of distributed planning in the world of air traffic control. Journal of Parallel and Distributed Computing, 3:411-431, 1986.

[FIPA 1997] FIPA 97 Specification; http://www.fipa.org

[FIPA 1998] FIPA 98 Specification; http://www.fipa.org

[FIPA 2000] FIPA 2000 Specification; http://www.fipa.org

[Fischer et al. 1994] Klaus Fischer, Jörg P. Müller, Markus Pischel; Unifying Control in a Layered Agent Architecture; DFKI Technical Memo TM-94-05, Saarbrücken, 1994.

[Fischer et al. 1996] Klaus Fischer, Jörg P. Müller, Markus Pischel; Cooperative Transportation Scheduling: an Application Domain for DAI. Journal of Applied Artificial Intelligence, 10(1), 1996.

[Fischer et al. 1998] Klaus Fischer, Christian Ruß, Gero Vierke; Decision Theory and Coordination in Multiagent Systems; DFKI RR-98-02, Saarbrücken, 1998.

[Fodor 1975] Jerry A. Fodor: The Language of Thought; Crowell, New York, 1975.

[Fonseca et al. 1996] J. M. Fonseca, E. Oliveira, A. Steiger-Garção; MACIV - A DAI Based Resource Management System. Proceedings of PAAM'96, pages 263-277, London, UK, April 1996.

[Forgy 1982] Charles Forgy; Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem. Artificial Intelligence 19(1), S. 17-37, 1982.

[Fox at al. 1993] M. S. Fox, J. F. Chionglo, M. Barbuceanu; The Integrated Supply Chain Management System. Internal Report, Dept. of Industrial Engineering, University of Toronto, 1993.

[Franklin & Graesser 1996] Stan Franklin, Art Graesser; Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.

[Fricke et al. 2001] Stefan Fricke, Karsten Bsufka, Jan Keiser, Torge Schmidt, Ralf Sesseler, Sahin Albayrak; Agent-based Telematic Services and Telecom Applications. in: Communications of the ACM, April 2001.

[Funk et al. 1998] Petra Funk, Gero Vierke, Hans-Jürgen Bürckert; A Multi-Agent Perspective on Intermodal Transport Chains. DFKI Technisches Memo DFKI-TM-98-06, Saarbrücken, 1998.

[Gabbay & Guenthner 1986] Dov M. Gabbay, Franz Guenthner; Handbook of Philosophical Logic, Volume III: Alternatives to Classical Logic; D. Reidel Publishing, 1986.

[Garrido & Sycara 1996] Leonardo Garrido, Katia Sycara; Multi-Agent Meeting Scheduling: Preliminary Experimental Results. International Conference on Multi-Agent Systems, 1996.

[Genesereth & Fikes 1992] Michael R. Genesereth, Richard E. Fikes; Knowledge Interchange Format, Version 3.0 Reference Manual, 1992.

[Genesereth & Ketchpel 1994] Michael R. Genesereth, Stephen P. Ketchpel; Software Agents; Communications of the ACM, 37(7), pages 48-53, 1994.

[Genesereth & Nilsson 1987] Michael R. Genesereth and Nils J. Nilsson; Logical Foundations of Artificial Intelligence. Morgan Kaufmann Publishers, San Mateo, CA, 1987.

[Georgeff & Ingrand 1989] Michael P. Georgeff, Francois F. Ingrand; Decision-Making in an Embedded Reasoning System. in: Proceedings of the Eleventh International Jopint Conference on Artificial Intelligence (IJCAI-89), 1989.

[Gilbert 1996] D. Gilbert; Intelligent Agents White Paper; IBM Intelligent Agent Center of Competency, 1996.

[Grand & Cliff 1998] S. Grand, D. Cliff; Creatures: Entertainment software agents with artificial life. Autonomous Agents and Multi-Agent Systems, 1(2), 1998.

[Gruber 1992] Thomas R. Gurber; Ontolingua: A Mechanism to Support Portable Ontologies. Stanford University, Knowledge Systems Laboratory, Technical Report KSL-91-66, 1992.

[Guttman & Maes 1998] Robert H. Guttman, Pattie Maes; Agent-mediated Integrative Negotiation for Retail Electronic Commerce. Proceedings of the Workshop on Agent Mediated Electronic Trading 1998 (AMET'98).

[Hamilton 1997] Graham Hamilton (Editor); JavaBeans – Specification 1.01. Sun Microsystems, http://java.sun.com/products/javabeans

[Harrison et al. 1995] Colin G. Harrison, David M. Chess, Aaron Kershenbaum; Mobile Agents: Are they a good idea? IBM Research Report, 1995.

[Hayes-Roth 1985] Barbara Hayes-Roth; A blackboard architecture for control. Artificial Intelligence, 26, 1985.

[Hayzelden 1998] Alex L. G. Hayzelden; Telecommunications Multi-Agent Control System (Tele-MACS). in: ECAI '98, Brighton, U.K, August 1998.

[Hraber et al. 1997] Peter T. Hraber, Terry Jones, Stephanie Forrest; The Ecology of Echo. Artificial Life 3(3):165-190, 1997.

[Huber 1999] Marcus J. Huber; JAM: A BDI-theoretic Mobile Agent Architecture. Proceedings of the Third International Conference on Autonomous Agents (Agents'99), Seattle, WA, May, 1999.

[Huber et al. 1995] Marcus J. Huber, Jaeho Lee, Patrick Kenny, Edmund H. Durfee; UM-PRS V3.0 Programmer and User Guide. http://ai.eecs.umich.edu/people/durfee/UMPRS.html

[Ingrand et al. 1992] Francois F. Ingrand, Michael P. Georgeff, Anand S. Rao; An Architecture for Real-Time Reasoning and System Control. IEEE Expert, 7(6), Dezember 1992.

[Ivezic & Potok 1999] N. Ivezic, T.E. Potok; An Agent-Based Approach For Supply Chain Management. Proceedings of the 5th International Conference on Information Systems Analysis and Synthesis, Vol. 3, 570-573, 1999.

[Ivezic et al. 1999] N. Ivezic, T. E. Potok, L. C. Pouchard; Multiagent Framework for Lean Manufacturing. IEEE Internet Computing, Vol 3., No. 5, 58-59, 1999.

[Java] Sun Java Technologie; http://java.sun.com

[Jennings 2001] Nicholas R. Jennings; Building Complex Software Systems: The Case for an Agent-Based Approach. in: Communications of the ACM, April 2001.

[Jennings & Wooldridge 1998] Nicholas R. Jennings, Michael J. Wooldridge; Applications of Intelligent Agents. in: N. R. Jennings and M. Wooldridge (eds.); Agent Technology: Foundations, Applications, and Markets; Springer-Verlag, 1998.

[Jennings et al. 1998] Nicholas R. Jennings, Katia Sycara, Michael Wooldridge; A Roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent Systems 1, 275-306, 1998.

[Jennings et al. 2000] Nicholas R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, B. Odgers; Autonomous Agents for Business Process Management; in: Int. Journal of Applied Artificial Intelligence 14, p.145-189, 2000.

[Jung 1999] Christoph G. Jung; Theory and Practice of Hybrid Agents; Dissertation, Saarbrücken, 1999.

[Jung & Fischer 1998] Christoph G. Jung & Klaus Fischer; Methodological Comparison of Agent Models; DFKI Research Report RR-98-01, Saarbrücken, 1998.

[Kleene 1938] Stephen Cole Kleene; On a Notation for Ordinal Numbers; in: Journal of Symbolic Logic 3, 1938.

[Kleene 1952] Stephen Cole Kleene; Introduction to Metamathematics; Wolters-Noordhoff Publishing and North-Holland Publishing Company 1971.

[Kripke 1963] Saul A. Kripke; Semantical Analysis of Modal Logic. in: Zeitschrift für mathematische Logik und Grundlagen der Mathematik, 9/1963.

[Labrou & Finin 1994] Yannis Labrou, Tim Finin; A semantics approach for KQML – a general purpose communication language for software agents. Third International Conference on Knowledge and Information Management (CIKM-94), Silver Spring, 1994.

[Labrou & Finin 1997] Yannis Labrou, Tim Finin; A Proposal for a new KQML Specification. TR CS-97-03, Computer Science and Electrical Engineering Department, UMBC, Baltimore, 1997.

[Lesser & Corkill 1983] Victor R. Lesser, Daniel D. Corkill; The Distributed Vehicle Monitoring Testbed: A tool for investigating distributed problem solving networks. AI Magazine, 4(3):15-33, Fall 1983.

[Maes 1994] Pattie Maes; Agents that Reduce Work and Information Overload; Communications of the ACM, Vol. 37, No. 7, July 1994.

[Martial 1992] Frank von Martial; Coordinating Plans of Autonomous Agents; Lecture Notes in Artificial Intelligence 610, Springer Verlag, 1992.

[Minar et al. 1996] Nelson Minar, Roger Burkhart, Chris Langton, Manor Askenazi; The Swarm Simulation System: A Toolkit for Building Multi-agent Simulations. http://www.swarm.org

[Minsky 1975] Marvin L. Minsky; A Framework for Representing Knowledge. in: Patrick H. Winston (ed); The Psychology of Computer Vision. McGraw-Hill, New York, 1975.

[Morris et al. 2000] Joan Morris, Peter Ree, Pattie Maes; Sardine: Dynamic Seller Strategies in an Auction Marketplace. Proceedings of the Conference on Electronic Commerce (EC '00), Minneapolis, MN, October 17-20, 2000.

[Müller 1996] Jörg P. Müller, The Design of Intelligent Agents: A Layered Approach; volume 1177 of Lecture Notes in Articial Intelligence, Springer-Verlag, 1996.

[Newell & Simon 1961] Allen Newell, Herbert A. Simon; GPS, a Program that Simulates Human Thought. in: H. Billing; Lernende Automaten. Oldenbourg, München, 1961.

[Newell & Simon 1976] Allen Newell, Herbert A. Simon; Computer Science as Empirical Inquiry: Symbols and Search. CACM 19(3): 113-126, 1976.

[Nwana et al. 1999] Hyacinth S. Nwana, Divine T. Ndumu, Lyndon C. Lee, Jaron C. Collis; ZEUS: A Toolkit for Building Distributed Multi-Agent Systems. Applied Artifical Intelligence Journal, Vol 13 (1) , p129-186, 1999.

[Posner 1989] Michael I. Posner (ed): Foundations of Cognitive Science; Cambridge, MA: MIT Press, 1989.

[Quillian 1968] M. R. Quillian; Semantic Memory. in: Marvin L. Minsky (ed); Semantic Information Processing. MIT Press, Cambridge MA, 1968.

[Rao & Georgeff 1995] Anand S. Rao, Michael P. Georgeff; BDI Agents: From Theory to Practice. Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, USA, 1995.

[Reiter 1980] Raymond Reiter; A Logic for Default Reasoning. in: Artificial Intelligence, 13(1-2), 1980.

[Reticular Systems 1999] Reticular Systems, Inc.; AgentBuilder – An Integrated Toolkit for Constructing Intelligent Software Agents. http://www.agentbuilder.com

[Russell & Norwig 1995] Stuart Russel, Peter Norwig; Artificial Intelligence, A Modern Approach. Prentice-Hall, 1995.

[Russell & Subramanian 1995] S. J. Russell and D. Subramanian; Provably Bounded Optimal Agents. Journal of Artificial Intelligence Research, 2, 1995.

[Sacerdoti 1974] E. D. Sacerdoti; Planning in a Hierarchy of Abstraction Spaces. Artificial Intelligence, 5(2), 1974.

[Searle 1969] John R. Searle; Speech Acts. Cambridge University Press, 1969.

[Searle 1993] John R. Searle; Die Wiederentdeckung des Geistes. Artemis und Winkler, München, 1993.

[Sesseler 2001] Ralf Sesseler; Building Agents for Service Provisioning out of Components. in: AGENTS ’01, Proceedings of the Fifth International Conference on Autonomous Agents, 2001.

[Sesseler & Albayrak 2001a] Ralf Sesseler, Sahin Albayrak; Agent-based Marketplaces for Electronic Commerce. in: Proceedings of the 2001 International Conference on Artificial Intelligence, IC-AI 2001 Vol. III, 2001.

[Sesseler & Albayrak 2001b] Ralf Sesseler, Sahin Albayrak; Service-ware Framework for Developing 3G Mobile Services. in: The Sixteenth International Symposium on Computer and Information Sciences, ISCIS XVI, 2001.

[Sesseler & Albayrak 2002] Ralf Sesseler, Sahin Albayrak; JIAC IV – An Open, Scalable Agent Architecture for Telecommunications Applications. in: First International NAISO Congress on Autonomous Intelligent Systems, ICAIS 2002.

[Shoham 1993] Yoav Shoham: Agent-Oriented Programming. Artificial Intelligence, Vol 60, pages 51-92, 1993.

[Smith 1980] R. G. Smith; The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. IEEE Transactions on Computers C-29, 1104-1113, 1980.

[Somers 1996] F. Somers; HYBRID: Intelligent Agents for Distributed ATM Network Management. in: IATA Workshop at ECAI’96, Budapest, Hungary, 1996.

[SRI 1997] SRI International; Procedural Reasoning System – User’s Guide. http://www.ai.sri.com/~prs/

[Thomas 1994] S. Rebecca Thomas; The PLACA Agent Programming Language. in: ECAI Workshop on Agent Theories, Architectures, and Languages, 1994.

[Urquhart 1986] Alasdair Urquhart; Many-Valued Logic. in: [Gabbay & Guenthner 1986]

[Von Neumann & Morgenstern 1944] John von Neumann, Oskar Morgenstern; Theory of Games and Economic Behavior. Princeton University Press, 1944.

[Winograd & Flores 1986] Terry Winograd, Fernando Flores; Understanding Computers and Cognition. Ablex Publishing, Norwood, New Jersey, 1986.

[Wooldridge 1998] Michael J. Wooldridge; Verifiable Semantics for Agent Communications; Proceedings ICMAS '98, S. 349-356, 1998.

[Wooldridge 1999] Michael J. Wooldridge; Intelligent Agents. Kapitel 1 in G. Weiss (ed): Multiagent Systems, MIT Press, 1999.

[Wooldridge & Jennings 1995] Michael J. Wooldrige, Nicholas R. Jennigs; Intelligent Agents: Theory and Practice. Knowledge Engineering Review, 10, 1995.

[Wooldridge & Jennings 1998] Michael J. Wooldridge, Nicholas R. Jennings; Pitfalls of Agent-Oriented Development; Proc 2nd Int. Conf. on Autonomous Agents (Agents-98), Minneapolis, USA, 385-391, 1998.

[Wooldridge et al. 2000] Michael J. Wooldridge, Nicholas R. Jennings, David Kinny; The Gaia Methodology for Agent-Oriented Analysis and Design. in: Journal of Autonomous Agents and Multi-Agent Systems 3 (3), S. 285-312, 2000.

[Ygge 1998] Fredrik Ygge; Market-Oriented Programming and its Application to Power Load Management. Dissertation, Lund University, 1998.

[Zadeh 1965] Lofti A. Zadeh; Fuzzy Sets. in: Information and Control, 8 / 1965.

[Zadeh 1979] Lofti A. Zadeh; A Theory of Approximate Reasoning. in: Machine Intelligence, 9 / 1979.

Lebenslauf

Name		Ralf Sesseler
Geburt	18. Oktober 1969	Ludwigsburg
Schule	1975 - 1979 1979 - 1988	Grundschule Homburg / Schwarzenbach Gymnasium Johanneum, Homburg Abschluß: Abitur (2,0)
Studium	Oktober 1988 - Dezember 1989, April 1991 - September 1997	Informatik, Mathematik und Philosophie an der Universität des Saarlandes Abschluß: Diplom (1,4)
Zivildienst	Januar 1990 - März 1991	Universitätsklinik Homburg
Arbeit	seit Oktober 1997	Wissenschaftlicher Mitarbeiter am DAI-Labor der TU Berlin