Eine Referenzarchitektur für zuverlässige Multiagentensysteme

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der Fakultät für Informatik
von der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

Dissertation

von

Jens Nimis

aus Karlsruhe

Tag der mündlichen Prüfung:	4. Februar 2009
Erster Gutachter:	Prof. Dr.-Ing. Dr. h. c. Peter C. Lockemann
Zweiter Gutachter:	Prof. Dr. rer. nat. Ralf Reussner

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung der Universität Karlsruhe (TH) zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, im Oktober 2008

Jens Nimis.

Danksagung

Die vorliegende Dissertation wurde begonnen am Institut für Programmstrukturen und Datenorganisation (IPD) der Universität Karlsruhe (TH) und fertiggestellt am FZI Forschungszentrum Informatik Karlsruhe im Bereich Information Process Engineering (IPE). Hier wie dort hatte ich das Glück, zahlreiche Menschen kennen zu lernen, die mir viel zugetraut, mich gefördert und immer wieder ermutigt haben. Bei Ihnen allen möchte ich mich an dieser Stelle ganz herzlich bedanken.

Allen voran gilt mein besonderer Dank Professor Lockemann. Sein starkes inhaltliches Interesse an der Arbeit, seine stets konstruktive inhaltliche Kritik und seine auch gegen Ende der Arbeit nie nachlassende Beharrlichkeit in Iterationen weitere Verbesserungen zu erzielen, haben die vorliegende Arbeit maßgeblich geprägt. Er hatte stets ein offenes Ohr für Fragen und Ideen und kommentierte die abgegebenen Zwischenversionen in kürzester Zeit und in größtmöglicher inhaltlicher Tiefe – eine ausführliche Besprechung zwischen Weihnachten und Silvester 2007 im Hause Lockemann ist hierfür beispielhaft. Nicht nur im Hinblick auf diese außergewöhnliche Einsatzbereitschaft wird mir mein Doktorvater weiterhin ein Vorbild sein. Ebenfalls besonderer Dank gebührt Professor Reussner, der ohne zu zögern das Korreferat übernommen hat und dieses unkompliziert aber nicht leichtfertig wahrgenommen hat. Seine inhaltlichen Anmerkungen halfen zielsicher Schwachpunkte zu beseitigen.

Auch wenn eine Promotion eine persönliche Leistung ist, kann sie nur in einem funktionierenden Umfeld erfolgreich durchgeführt werden. Dieses Umfeld bot zu Beginn der Arbeit das IPD, wo die Arbeit im Projekt KRASH ihren Ursprung nahm. Khaled Nagi und Gerd Hillebrand haben als Kollegen in diesem Projekt durch die zahlreichen gemeinsamen Diskussionen geholfen, den Grundstein für die Arbeit zu legen. Bethina Schmidt und Sebastian Pulkowski hatten zuvor als Betreuer meiner Diplomarbeit mein Interesse an eigenständiger wissenschaftlicher Arbeit geweckt und mich so letztlich als wissenschaftlichen Mitarbeiter an das IPD geholt. Neben den bereits genannten Kollegen haben viele andere zu der stets harmonischen und produktiven Arbeitsatmosphäre am IPD beigetragen. Ihnen allen gilt mein Dank – stellvertretend seien Jutta Mülle, Vera Horcic und Erika Götz genannt. Das Projekt KRASH war Teil des DFG Schwerpunktprogramms SPP1083 „Intelligente Agenten und realistische Anwendungsszenarien“, in dem sich durch die enge

Zusammenarbeit unter den wissenschaftlichen Mitarbeitern ein dauerhaftes Netzwerk ausgebildet hat, das neben dem inhaltlichen auch einen persönlichen Erfahrungsaustausch unter „Gleichgesinnten“ förderte. Namentlich aus dem Netzwerk will ich Ingo Timm danken, mit dem die inhaltlichen und persönlichen Diskussionen besonders intensiv waren. Nach meinem Wechsel an das FZI in den rasch wachsenden Bereich IPE war das Umfeld im Vergleich zum IPD zwar ungleich turbulenter, aber weiterhin geprägt von einer produktiven und sehr angenehmen Atmosphäre zu der alle Kollegen ihren Beitrag leisten. Besonderer Dank gilt am IPE dem Bereichsleiter Carsten Holtmann, dem FZI Direktor Professor Stefan Tai und meinen Abteilungsleiter-Kollegen, die angesichts von Mehrfachbelastungen stets großes Verständnis aufgebracht und Freiräume geschaffen haben, die die Fertigstellung der Arbeit ermöglicht haben.

Immer wieder eine gute Erfahrung war die Zusammenarbeit mit Studenten als Diplomanden, Studienarbeitern oder wissenschaftlichen Hilfskräften. Herauszuheben sind Rainer Vogt, Heiko Schepperle, Martin Dinkloh, Christoph Sorge, Artus Krohn-Grimberghe, Michael Kuperberg, Christian Kupper und Johannes Lipsky, von denen einige selbst später zu Kollegen wurden. Ihnen gebührt mein Dank, weil sie inhaltlich zur Arbeit beigetragen haben oder durch ein hohes Maß an Engagement und eigenverantwortlichem Handeln in verschiedenen Projekten eine große Entlastung waren.

Zuletzt gilt mein allergrößter Dank meiner Familie. Einerseits sind das meine Eltern, die mir stets das Gefühl gegeben haben, das Richtige zu tun, mich ermutigt haben, meinen Weg zu gehen, und die sich immer wieder um die nötige Bodenhaftung gesorgt haben. Andererseits sind das meine Frau Inka, meine Tochter Lara und mein Sohn Finn. Mit meiner Frau verbindet mich eine tiefe Liebe, die es ihr erlaubt hat und uns beiden schwer gemacht hat, mit dem langen Verzicht auf ein ausreichendes gemeinsames Familienleben zurecht zu kommen. Die Geduld und Rücksichtnahme, die ich von meiner Frau und meinen Kindern erfahren durfte, lässt sich nicht hoch genug bewerten.

Karlsruhe, im Februar 2009 Jens Nimis

Kurzfassung

Motivation und Zielsetzung

Multiagentensysteme erfreuen sich seit zwei Jahrzehnten einer intensiven Aufmerksamkeit durch die Forschung. In dieser Zeit gelang es vielfach, das große Potenzial der Agententechnologie in komplexen Anwendungsumgebungen nachzuweisen. Dennoch sind nur vergleichsweise wenige voll ausgebildete Multiagentensysteme im industriellen Einsatz bekannt – meist werden dort bestimmte Technologien isoliert, die im Kontext der Agentenforschung entwickelt wurden, und in die vorhandenen herkömmlichen Systeme eingebettet, womit jedoch zwangsläufig nicht mehr das volle Potenzial der Agententechnologie ausgeschöpft werden kann. Ein Grund für die schwache Marktdurchdringung der vollwertigen Agentensysteme sind die erhöhten Zuverlässigkeitsanforderungen in den geeigneten kommerziellen Anwendungsumgebungen bei der gleichzeitig großen Komplexität der Multiagentensysteme selbst, die z.B. in der Verteilung und Autonomie der Agenten begründet ist und die das Erreichen eines hohen Zuverlässigkeitsgrades erschwert.

Diese Erkenntnis führte im Laufe der Jahre zu einer Vielfalt an Mechanismen zur Erhöhung der Zuverlässigkeit von Multiagentensystemen, die jedoch meist dieselbe entscheidende Schwäche aufweisen: Die Zuverlässigkeitsmechanismen wurden im Zusammenhang mit spezifischen anderen Aspekten der Agentenprogrammierung entwickelt, die im eigentlichen Forschungsinteresse lagen, und nicht generisch für beliebige Multiagentensysteme. Damit ist eine Übertragung der Zuverlässigkeitsmechanismen auf andersartige Multiagentensysteme in der Regel mit einem hohen Aufwand zu deren Anpassung verbunden, d.h. die Mechanismen sind nicht orthogonal zu den Programmieraspekten der existierenden Multiagentensysteme – ein Investitionsschutz für diese Systeme ist daher nicht gegeben.

Hier liegt das Ziel der Arbeit: Sie soll einen Beitrag leisten zur Entwicklung von Mechanismen zur Steigerung der Zuverlässigkeit von Multiagentensystemen unter der Randbedingung des Investitionsschutzes. Als wichtigste Aufgabe wird hierbei eine Systematisierung der Programmierartefakte, Störungen und Zuverlässigkeitsmechanismen angesehen. Die Systematisierung erfolgt in Form einer Architekturbetrachtung nach den Prinzipien der Software-Technik, die zu einer **Referenzarchitektur für zuverlässige Multiagentensysteme** führt. Sie unterstützt Entwickler von Multiagentensystemen bei der Auswahl geeigneter existierender Zuverlässig-

keitsmechanismen und/oder bei der Entwicklung eigener orthogonaler Zuverlässigkeitsmechanismen.

Lösungsansatz

Bei der Erstellung der Referenzarchitektur für zuverlässige Multiagentensysteme – in der weiteren Zusammenfassung als Zuverlässigkeitsreferenzarchitektur „ZRA“ bezeichnet – wurde wie folgt vorgegangen:

Zunächst wurde mit einer allgemeingültigen und vollständigen Referenzarchitektur für Multiagentensysteme eine wesentliche Voraussetzung für die ZRA geschaffen. Die geschichtete Referenzarchitektur leistet eine Systematisierung der Agententechnologie, die eine Identifikation aller Aspekte der Agentenprogrammierung und ihres Zusammenspiels ermöglicht. Basis des methodischen Vorgehens bei der Erstellung der Referenzarchitektur sind die anerkannten definierenden Eigenschaften der Multiagentensysteme, etablierte Standards und verwandte Referenzarchitekturen. Der Nachweis der Allgemeingültigkeit und Vollständigkeit erfolgte durch eine Untersuchung ihrer Kompatibilität zu weiteren optionalen Eigenschaften von Multiagentensystemen und ihrer Konformität zu anderen bekannten Agentenarchitekturen.

Die ZRA baut auf der Referenzarchitektur auf und erweitert diese um den Zuverlässigkeitsaspekt. Hierfür wurde ein agentenunspezifisches Fehler- und Fehlerbehandlungsmodell für zuverlässige Komponenten in Schichtenarchitekturen entwickelt und dieses auf die Referenzarchitektur für Multiagentensysteme aufgeprägt. Damit wird eine Verortung spezifischer Störungen nach ihrem Entstehungsort und der zugehörigen Behandlungsoptionen möglich.

Die abstrakte Darstellung der Fehlerbehandlungsoptionen in der ZRA wurde durch die Verortung einer Vielzahl bekannter Fehlerbehandlungsmechanismen zu einer Zuverlässigkeitslandkarte für Multiagentensysteme ausgebaut. Die Landkarte zeigt, wie Störungen entlang von Fehlerbehandlungspfaden durch die Schichten eines Multiagentensystems aufsteigen können und in den Schichten behandelt werden können. Damit weist die Landkarte zum einen die Fähigkeit der ZRA zur Verortung existierender Zuverlässigkeitsmechanismen nach und gibt zum anderen Entwicklern von Multiagentensystemen eine Hilfestellung bei der Auswahl geeigneter Kandidaten. Der Nachweis der Übertragbarkeit der ZRA erfolgte durch eine Abbildung der Fehlerbehandlungspfade auf die bekannten existierenden Agentenarchitekturen. Den Entwicklern

eines Multiagentensystems hilft die Übertragbarkeitsdiskussion bei der Entwicklung eigener Zuverlässigkeitsmechanismen, falls in der Zuverlässigkeitslandkarte kein geeigneter Mechanismus gefunden werden konnte.

Die abschließende Fallstudie gibt ein Beispiel für den Einsatz der ZRA aus Sicht eines Entwicklers, der die Zuverlässigkeit eines existierenden Multiagentensystems zur Produktionsfeinplanung gegen Rechnerstörungen erhöhen will. Die Betrachtung der Mechanismen in der Zuverlässigkeitslandkarte führt zu dem Ergebnis, dass kein existierender Mechanismus das Zuverlässigkeitsproblem ohne tiefe Eingriffe in das existierende System beheben kann. Unter Verwendung der ZRA und der Ergebnisse aus ihrer Übertragbarkeitsdiskussion wird daher ein Transaktions-basierter Zuverlässigkeitsmechanismus entwickelt, der sich orthogonal in das bestehende System einbettet und der sich auch unter Variation der impliziten einschränkenden Annahmen so anpassen lässt, dass wiederum orthogonale Mechanismen unter den geänderten Voraussetzungen resultieren. Damit demonstriert die Fallstudie, wie die ZRA zur Steigerung der Zuverlässigkeit von Multiagentensystemen unter der Randbedingung des Investitionsschutzes beiträgt.

Verallgemeinert beschreibt die ZRA einen Ansatz zur systematischen Behandlung des Qualitätsmerkmals Zuverlässigkeit in komplexen Software-Landschaften. Unabhängig von der im Dissertationsvorhaben betrachteten Systemklasse der Multiagentensysteme lassen sich Fehler- und Fehlerbehandlungsmodell und das angewendete Eskalationsprinzip in Schichtenarchitekturen auf andere Systemklassen übertragen, wie z.B. auf Dienst-orientierte Anwendungssysteme.

Wissenschaftlicher Beitrag

Allgemeingültige und vollständige Referenzarchitektur für Multiagentensysteme

Die Referenzarchitektur systematisiert – auf einer konzeptuellen Abstraktionsstufe – erstmals alle Aspekte der Programmierung von Multiagentensystemen in einem integrierten Modell. Die bekannten verwandten Arbeiten in Form von etablierten Standards und Referenzarchitekturen betrachten durchweg nur gewählte Ausschnitte der Agentenprogrammierung, da sie vor dem Hintergrund spezifischer Zielsetzungen oder Anwendungsgebiete entstanden sind. Damit wurde methodisch ein Rahmen geschaffen, der unabhängig vom Kontext der Zuverlässigkeit eine Reflektion aller Entwicklungsaspekte von Multiagentensystemen erlaubt.

Zuverlässigkeitsreferenzarchitektur für Multiagentensysteme Das Herz der Arbeit ist die Zuverlässigkeitsreferenzarchitektur, die im Vergleich zu den existierenden Arbeiten zur Zuverlässigkeitssteigerung von Multiagentensystemen einen grundlegend neuen Ansatz verfolgt. Die verwandten Arbeiten entwickeln konkrete Zuverlässigkeitsmechanismen, die zwar zum Teil den Anspruch auf Allgemeingültigkeit erheben, denen jedoch zumeist spezifische Systemmodelle oder Annahmen zugrunde liegen, die eine universelle Einsetzbarkeit verhindern. Die ZRA bietet hingegen keinen konkreten Zuverlässigkeitsmechanismus, sondern einen Rahmen, der Entwicklern hilft, geeignete orthogonale Zuverlässigkeitsmechanismen zu identifizieren oder selbst zu entwickeln.

Zuverlässigkeitslandkarte für Multiagentensysteme Die Zuverlässigkeitslandkarte verortet Störungen und die bekannten konkreten Fehlerbehandlungsmechanismen innerhalb der ZRA systematisch entlang von Fehlerbehandlungspfaden und gibt Entwicklern von Multiagentensystemen damit ein Mittel an die Hand, um zielgerichtet existierende Fehlerbehandlungsmechanismen für ein gegebenes Zuverlässigkeitsproblem zu identifizieren. In der Literatur ist dem Autor keine vergleichbare umfassende systematische Vorgehensweise für Zuverlässigkeitsmechanismen für Multiagentensysteme bekannt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Durch die fortwährenden Entwicklungen in der Informationstechnologie (z.B. Vernetzung, Allgegenwärtigkeit,...) und die gleichzeitigen Trends in der Wirtschaft (z.B. Globalisierung, Kostendruck,...) wird die IT-Branche künftig immer mehr gezwungen sein, kommerzielle Anwendungsprobleme von wachsender Komplexität effizient und zuverlässig zu lösen. Als vielversprechender Lösungsansatz für komplexe Anwendungsprobleme gilt – seit geraumer Zeit – die Agententechnologie. Die Vorstellungen, wie die Agententechnologie zu deren Lösung beitragen kann, gehen aber weit auseinander und lassen sich grob in vier Klassen unterteilen:

Agentenforschung als Impulsgeber für neue Technologien Dem jahrelangen umfangreichen Forschungsaufwand für die Agententechnologie stehen bisher vergleichsweise wenige bekannte erfolgreiche Anwendungen gegenüber. Dies hat in der Forschungsgemeinde vor allem seit etwa 1999 immer stärker zur Frage nach einer Anwendung (oder einem ganzen Anwendungsgebiet) geführt, die förmlich nach Agententechnologie schreit (engl. *„Killer Application“*). Auch wenn mittlerweile die Vorstellungen über prädestinierte Anwendungsgebiete weitaus konkreter geworden sind, hat sich unter den Zweiflern eine Strömung etabliert, die die Agentenforschung als reinen Impulsgeber für andere neue informationstechnologische Entwicklungen betrachtet. Nach ihrer Meinung stellt nicht die zu umfassende Agententechnologie selbst die Lösung der Anwendungsprobleme dar. Vielmehr dient aus ihrer Sicht die Agentengemeinde als Schmelztiegel ihrer

beteiligten Wissenschaften (z.B. Informatik, Künstliche Intelligenz, Wirtschaftswissenschaften, Sozialwissenschaften,...) und als Ideenschmiede, aus deren Technologierepertoire sich Teilaspekte als Lösung für Teilprobleme separieren lassen. Beispiele für solche Teilaspekte, die aus der Agentengemeinde befruchtet wurden und sich mittlerweile zu eigenen Forschungsgebieten entwickelt haben, sind u.a. das Semantic Web, das Autonomic Computing, das Ubiquitous Computing, das Peer-to-Peer Computing und das Grid Computing (Luck u. a. 2005).

Agenten als Modellierungsparadigma Die agentenorientierte Modellierung stellt eine Erweiterung der objektorientierten Modellierung dar. In der objektorientierten Modellierung werden die modellierten Artefakte (Objekte) als Ausprägungen von Klassen aufgefasst, denen bestimmte gemeinsame Eigenschaften (Attribute) und Fähigkeiten (Methoden) zugeschrieben werden. Den Agenten wird darüber hinaus unterstellt, dass sie Ziele haben, die ihnen von einem Prinzipal vorgegeben sein können, und die sie unter Einsatz ihrer Fähigkeiten zu verfolgen gedenken. Die agentenorientierte Modellierung führt nach ihren Verfechtern aber nicht zwangsweise auch zu einer entsprechenden Implementierung. Vielmehr soll sie nur dem besseren Verständnis eines noch ungenügend erschlossenen Anwendungsgebiets dienen. Über die Realisierung eines agentenorientierten Entwurfs werden zum Modellierungszeitpunkt aber noch keine Annahmen getroffen. Dies erlaubt es insbesondere, eine deskriptive ziel-orientierte Modellierung in einer „herkömmlichen“ imperativen Implementierung zu realisieren. Es steht zu befürchten, dass bei diesem Schritt ein Großteil der vorteilhaften Eigenschaften verloren gehen könnte.

Agenten als Simulationsparadigma Einen Schritt weiter geht die agentenbasierte Simulation, bei der die agentenorientierte Modellierung eines Systems in ein Simulationsmodell überführt wird. Sie wird häufig eingesetzt, um das Gesamtverhalten in sozialen Systemen, wie z.B. von Autofahrern im Straßenverkehr, analysieren zu können. Ziel der Simulationen ist aber nie die direkte Rückkopplung des Agentensystems an die modellierte reale Umgebung, sondern die Analyse mit dem Ziel eines besseren Domänenverständnisses. Dieses Verständnis kann ähnlich wie bei der agentenorientierten Modellierung in eine beliebige Implementierung einfließen.

Agenten als Software-Paradigma Aus der Sicht eines Verfechters der Agententechnologie erscheinen die drei vorgestellten Ansätze zum Einsatz der Agenten nicht sehr befriedigend. Entweder die Technologie wird nicht in voller Ausprägung eingesetzt oder nicht über den ganzen Entwicklungszyklus hinweg. Letzteres muss aber das Ziel sein, um aus der Agententechnologie auch den erhofften Nutzen zu ziehen, nämlich die wachsende Komplexität der Anwendungsprobleme durch die Flexibilität der Agenten zu beherrschen.

Es stellt sich nun aber die Frage, warum sich die Agententechnologie – trotz der bereits heute vorliegenden Anwendungsproblematik – noch nicht als Technologie für den Massenmarkt etablieren konnte (Hendler 2007). Die Gründe dafür sind vielschichtig. Zunächst einmal beruht die Entscheidung eines Unternehmens, eine neue Technologie einzusetzen, auf stichhaltigen Nachweisen für deren Tauglichkeit, die sich in betriebswirtschaftlichen Größen beziffern lassen muss. Für eine Reihe von konkreten Anwendungen (siehe Belecheanu u. a. 2006; Bussmann u. a. 2004; Dorer und Calisti 2005; Brennan und O 2004) bzw. von umfassenden Anwendungsgebieten (siehe Kirn u. a. 2006b; Parunak 2000; Shehory 2000; Cavalieri u. a. 2000; Frey 2004, u.v.a.m) sind solche Nachweise inzwischen durch einzelne existierende Systeme (engl. *„Success Stories“*) oder durch Simulationen erbracht worden.

Die Ergebnisse der Studien über erfolgreiche Anwendungen und Anwendungsgebiete lassen sich abstrahieren und verallgemeinern (Lockemann und Nimis 2004; O'Malley und DeLoach 2001). Es ergibt sich eine Reihe von Charakteristika für Anwendungsgebiete, die für Agententechnologie besonders geeignet sind – oder umgekehrt, ohne die ein Einsatz von Agenten nicht anzuraten ist. In (Lockemann und Nimis 2004) wird folgende Schlussfolgerung gezogen:

> „Multiagent systems offer an advantage if
>
> - the range of environmental situations (the problem space) is too large to be enumerated and dealt with by conventional means,
>
> - the range of decisions (the solution space) for responding is commensurate in size with the problem space,

- the problem space can be divided into sets of simpler tasks, each requiring specialized competence,
- the simpler tasks can be dealt with autonomously by individual agents,
- the overall situation can only be solved by cooperation among the agents.“

Neben diesen gemeinsam erforderlichen Eigenschaften eines Anwendungsgebiets wird in anderen Arbeiten als zusätzlich notwendiges Kriterium eine große Dynamik, d.h. rasche Veränderlichkeit der Umgebung, genannt. Zusammengefasst beschreiben diese Kriterien Anwendungsumgebungen, die als *komplex* oder auch *schwierig* bezeichnet werden können. Agentensystemen, die solche schwierigen Anwendungsumgebungen beherrschen können, misst man das Merkmal der *Intelligenz* bei.

1.1 Hinführung zum Thema: Problem und Ansatz

Die betriebliche Indikation für den Einsatz der Agententechnologie in Unternehmen aufgrund der Anwendungsumgebung scheint insgesamt recht klar zu sein. Dann muss es aber eine Reihe weiterer in der Technologie selbst begründete Hindernisse geben, die bislang eine stärkere Verbreitung erschwert haben. Dazu zählen z.B. eine mangelnde professionelle Werkzeugunterstützung, das Fehlen durchgängiger Entwicklungsmethoden und eine schlechte Beherrschbarkeit der systeminhärenten Komplexität. Für offene Agentensysteme erweitert sich die Palette der ungelösten Forschungsfragestellungen z.B. noch um Fragen zu Sicherheit, Vertrauen und Reputation.

Darüber hinaus beherbergen die Anwendungsumgebungen, die durch obige Kriterien definiert werden, wie z.B. Produktionsplanung und -steuerung, Güterlogistik, Verkehrssteuerung oder Katastrophen-Management, häufig noch einen weiteren schwerwiegenden Hinderungsgrund in sich. Gerade in diesen Anwendungsumgebungen sind (überlebens-\unternehmens-\...-) *kritische Anwendungen* beheimatet, die sich durch einen erhöhten Bedarf an *Zuverlässigkeit bei Störungen*[1] auszeichnen, um einen zuverlässigen Betrieb auch unter problematischen Umständen garantieren zu können.

[1] Zuverlässigkeit versteht sich hier im Sinne der Definition des Begriffs *Reliability* von Leveson (1986).

In „herkömmlichen“ Softwaresystemen, wie z.B. monolithischen oder Client-\Server-Systemen, wird ein erhöhter Bedarf an Zuverlässigkeit häufig mit dem Einsatz von Datenbanken und speziell deren Transaktionsunterstützung beantwortet. Diesem Vorgehen kommt die meist imperative Programmierung herkömmlicher Softwaresysteme zugute, die einerseits zu einer klar eingrenzbaren Menge von Störungen führt und andererseits den Zuverlässigkeitsmechanismen klare Ansatzpunkte bietet.

Bei Agenten verhält es sich schwieriger: ihre Autonomie und Ziel-Orientiertheit beruht – bei konsequenter Umsetzung des Agentenparadigmas zumindest teilweise – auf einer deskriptiven Programmierung, die erst zur Laufzeit in konkrete Handlungspläne übersetzt wird. Die Koordination in Multiagentensystemen erfolgt nicht nach fest ausprogrammierten Spielregeln; stattdessen findet eine Selbstorganisation nach wirtschaftlichen und/oder sozialen Mechanismen statt. Dieses komplexe Programmiermodell führt zu einem quasi *indeterministischen Verhalten* des Gesamtsystems.

Die Umsetzung des indeterministischen Verhaltens von Multiagentensystemen umfasst eine große *Vielfalt von Programmieraspekten*, die durch *etablierte Methoden, ausgereifte Programmiermodelle und konkrete Systeme* größtenteils gut unterstützt werden. All diese Programmieraspekte bringen ihre eigenen Störungsarten mit sich, bieten aber andererseits häufig auch ihre eigenen Ansatzpunkte für Mechanismen zur Steigerung der Zuverlässigkeit.

Leider werden bisher bei vielen der existierenden Zuverlässigkeitsmechanismen verschiedene *funktionale Programmieraspekte mit dem Zuverlässigkeitsaspekt vermengt*. Dies führt dazu, dass die bewährten Methoden, Programmiermodelle und Systeme bei der Anwendung dieser Zuverlässigkeitsmechanismen in großen Teilen aufgegeben werden müssen. Aus Sicht des *Investitionsschutzes* ist dies inakzeptabel: Um Methoden, Programmiermodelle und konkrete Systeme weiter verwenden zu können, muss auf Implementierungsebene eine möglichst scharfe Trennung zwischen funktionalen Programmieraspekten und dem Zuverlässigkeitsaspekt erfolgen, d.h. *Zuverlässigkeitsmechanismen müssen orthogonal zu den anderen Artefakten der Agentenimplementierung liegen.*

Genau hier liegt das Thema dieser Dissertation: In der vorliegenden Arbeit sollen wesentliche *Beiträge zur Steigerung der Zuverlässigkeit von Multiagentensystemen unter*

der Randbedingung des Investitionsschutzes geleistet werden.

Die zentrale Aufgabe zum Erreichen dieses Ziels ist eine *Systematisierung der Programmierartefakte, Störungen und Zuverlässigkeitsmechanismen*. Generell wird durch diese Systematisierung die Bewusstseinsbildung für Zuverlässigkeitsfragen verbessert und eine höhere Abdeckung der Zuverlässigkeitsbetrachtungen über das gesamte Multiagentensystem ermöglicht. Im Speziellen erlaubt die Systematisierung den Entwicklern eine Verortung von Störungen und eine Auswahl bzw. Eigenentwicklung entsprechender orthogonaler Zuverlässigkeitsmechanismen unter (Weiter-)Verwendung etablierter Methoden, Programmiermodelle und konkreter Systeme. Die Systematisierung erfolgt nach dem Prinzip der *„Separation of Concerns“* aus dem Software Engineering in Form einer Architekturbetrachtung und resultiert in der **Zuverlässigkeitsreferenzarchitektur für Multiagentensysteme**.

1.2 Gliederung

Für die weitere Ausarbeitung ergibt sich folgende Gliederung, die in Abbildung 1.1 auch grafisch dargestellt wird. In Kapitel 2 wird die Aufgabenstellung weiter präzisiert. Dazu wird zunächst ein agententypisches Ressourcenzuteilungsverfahren aus der Produktionsplanung und -steuerung vorgestellt, an dem die Ziele der Arbeit motiviert und anschaulich gemacht werden.

Die Kapitel 3 und 4 bilden den Kern der Arbeit. Zunächst wird in Kapitel 3 als Basis für die „Zuverlässigkeitsreferenzarchitektur“ systematisch eine softwaretechnische Referenzarchitektur für Multiagentensysteme abgeleitet basierend auf den anerkannten definierenden Eigenschaften, etablierten Standards und verwandten Referenzarchitekturen für Multiagentensysteme. Anhand dieser Referenzarchitektur können alle gebräuchlichen Programmieraspekte von Multiagentensystemen systematisiert werden, wie sich durch Konformitätsuntersuchung zu bestehenden Agentenarchitekturen zeigen lässt.

In Kapitel 4 wird die Referenzarchitektur erweitert um den Zuverlässigkeitsaspekt, woraus die Zuverlässigkeitsreferenzarchitektur resultiert. Dazu wird zunächst ein generisches

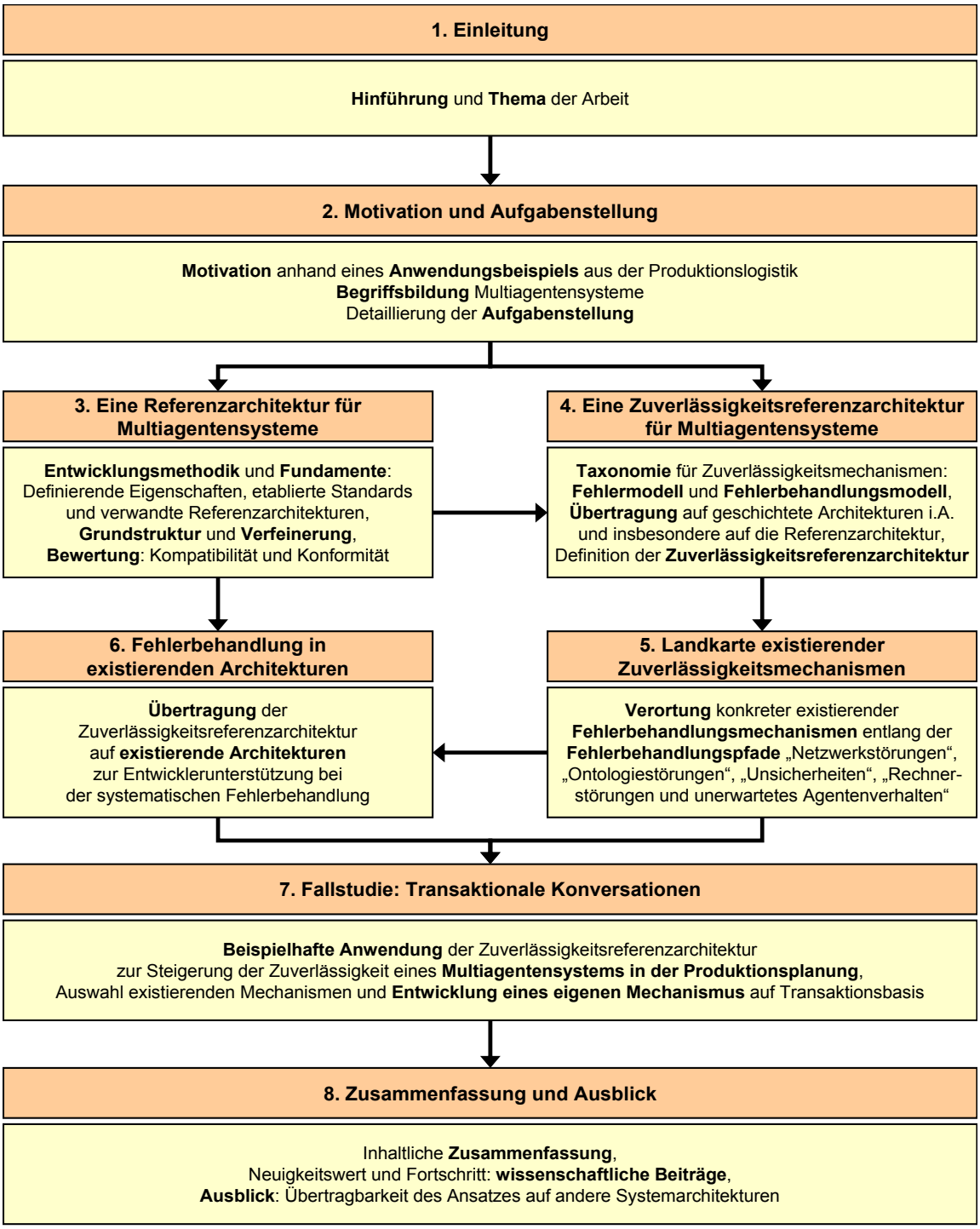


Abbildung 1.1: Gliederung der Ausarbeitung

Fehler- und Fehlerbehandlungsmodell für zuverlässige Softwarekomponenten entwickelt und dieses dann auf die Referenzarchitektur aufgeprägt. So gelingt eine Verortung der Störungen in Multiagentensystemen nach ihrem Entstehungsort und von zugehörigen generellen Optionen zur Fehlerbehandlung.

Die Zuverlässigkeitsreferenzarchitektur aus Kapitel 4 wird in Kapitel 5 durch die Verortung konkreter existierender Zuverlässigkeitsmechanismen entlang von Fehlerbehandlungspfaden konkretisiert zu einer Zuverlässigkeitslandkarte für Multiagentensysteme. Diese Landkarte weist zum einen die Verortungsfähigkeit der Zuverlässigkeitsreferenzarchitektur nach und hilft zum anderen Entwicklern von Multiagentensystemen falls möglich bei der Auswahl geeigneter existierender Zuverlässigkeitsmechanismen. Zugleich stellt die Landkarte die zur vorliegenden Arbeit verwandten Arbeiten dar und systematisiert diese.

Die Übertragbarkeit der Erkenntnisse steht im Mittelpunkt von Kapitel 6, in dem untersucht wird, wie sich die Zuverlässigkeitsreferenzarchitektur auf bekannte Architekturen für Multiagentensysteme anwenden lässt. Dazu wird betrachtet, wie sich die Störungen, Fehlerbehandlungsmechanismen und deren Zusammenhänge aus der Zuverlässigkeitsreferenzarchitektur auf die Komponenten der verschiedenen Architekturen abbilden lassen. Den Entwickler unterstützt diese Übertragung bei der Umsetzung eigener Zuverlässigkeitsmechanismen, falls sich keine existierenden Mechanismen für sein Zuverlässigkeitsproblem finden lassen: Er kann ableiten, wie und wo er in sein vorhandenes System eingreifen muss.

Kapitel 7 präsentiert eine Fallstudie zur Zuverlässigkeitsreferenzarchitektur. Es wird zunächst versucht, für ein existierendes unzuverlässiges System zu einem konkreten Anwendungsszenario in der Zuverlässigkeitslandkarte einen geeigneten existierenden Mechanismus zu finden. Mangels Erfolg wird ein eigener Zuverlässigkeitsmechanismus entwickelt, der eine weitreichende Separierung des Zuverlässigkeitsaspekts von anderen Programmieraspekten aufweist. Als Ansatzpunkt für den Transaktions-basierten Mechanismus, der auf die Zuverlässigkeit des gesamten Multiagentensystems abzielt, lässt sich unter Anwendung der Fehlerbehandlungspfade in der Zuverlässigkeitsreferenzarchitektur die sog. Konversationsüberwachung identifizieren, da sich an den Konversationen die Beziehungen zwischen den einzelnen Agenten manifestieren.

Die Arbeit schließt mit einer Zusammenfassung in Kapitel 8. Für eine bessere Verständlichkeit der eingesetzten Literaturquellen und eine inhaltlichen Abgeschlossenheit der Ausarbeitung sind zusätzlich Anhänge beigefügt, die die Übersetzung der englischsprachigen Fachbegriffe ins Deutsche und die spezifischen Grundlagen der Produktionsplanung und -steuerung und der Transaktionsverwaltung beinhalten.

Kapitel 2

Motivation und Präzisierung der Aufgabenstellung

Multiagentensysteme versprechen durch ihre Flexibilität hoch komplexe und dynamische Anwendungsgebiete zu beherrschen, in denen herkömmliche Systeme an ihre Grenzen stoßen. In vielen dieser Anwendungsgebiete ist für kritische Aufgaben eine besonders hohe Zuverlässigkeit bei Störungen erforderlich. Doch was bedeutet „Zuverlässigkeit“ und „Störung“ überhaupt? Wie sehen konkrete Beispiele dafür aus? Und wie will diese Arbeit dazu beitragen, die Zuverlässigkeit von Multiagentensystemen zu verbessern?

Das vorliegende Kapitel versucht diese Fragen illustriert an Beispielen zu beantworten. Dazu wird zunächst die weit verbreitete Ressourcenallokation mit dem Kontraktnetzprotokoll als Anwendungsbeispiel für Multiagentensysteme eingeführt. An ihm werden exemplarische Störungen diskutiert, von denen sich ein abstrahierter Störungs-, Robustheits- und Zuverlässigkeitsbegriff ableiten lässt. Ausgestattet mit den notwendigen Begriffsdefinitionen werden abschließend die Ziele und Aufgaben der vorliegenden Arbeit präzisiert.

2.1 Illustration der Zuverlässigkeitsproblematik am Anwendungsbeispiel

2.1.1 Ressourcenallokation mit dem Kontraktnetzprotokoll

Das *Kontraktnetzprotokoll (engl. Contract Net Protocol, CNP, auch CNET)*, das in seiner Urform auf Smith (1980) zurück geht, ist ein Koordinationsverfahren und Kommunikationsprotokoll, das den Ablauf und die Nachrichtenformate bei einer Vertragsverhandlung zwischen zwei Parteien beschreibt. Bei dieser Art von (Unter-)Beauftragung werden von einer Partei (dem *Initiator*[1]) Ressourcen bei einer anderen Partei (dem *Participant*) allokiert. Der Ablauf, der in Abbildung 2.1[2] illustriert wird, folgt dem Muster einer Vertragsverhandlung zwischen realen Parteien, also dem Prinzip von Ausschreibung, Angebot und Auftragsvergabe. Beim Einsatz des CNP zur agentenbasierten Maschinenbelegungsplanung wird i.A. die Rolle des Initiators von einem Agenten übernommen, der einen Auftrag verkörpert (`OrderAgent`) und die Rolle der Participants von Agenten, die die verschiedenen Maschinen vertreten (`MachineAgentA` und `MachineAgentB`)[3].

Im fehlerfreien Fall, der in Abbildung 2.1 dargestellt ist, umfasst der Verhandlungsablauf die folgenden Schritte:

Initiator: Ausschreibung Zunächst erstellt der `OrderAgent` eine Ausschreibung für die Aufgabe, die er gerne als Auftrag vergeben möchte. Die Ausschreibung enthält eine Beschreibung der Aufgabe, ggf. Einschränkungen der zulässigen Participants, eine Frist, bis zu deren Ablauf ein Angebot eingegangen sein muss, und die Beschreibung des Nachrichtenformats, in dem das Angebot erstellt werden muss, um eine Vergleichbarkeit sicher zu stellen. Diese Ausschreibung verteilt der `OrderAgent` an alle `MachineAgent`s, die aus seiner Sicht für die Diensterbringung in Betracht kommen (`cfp`-Nachricht, von engl. *Call for Proposal*). Zur Ermittlung der möglichen

[1] Die Beschreibung des Protokolls und die verwendeten Bezeichnungen orientieren sich an der Contract Net Protocol-Spezifikation der FIPA (FIPA 2002k).

[2] Die grafische Darstellung entspricht der AUML-Notation (Agent UML, siehe Bauer u. a. 2001)

[3] Hintergründe zur Rolle der Maschinenbelegungsplanung in der Produktionsplanung und -steuerung werden in Anhang B eingeführt.

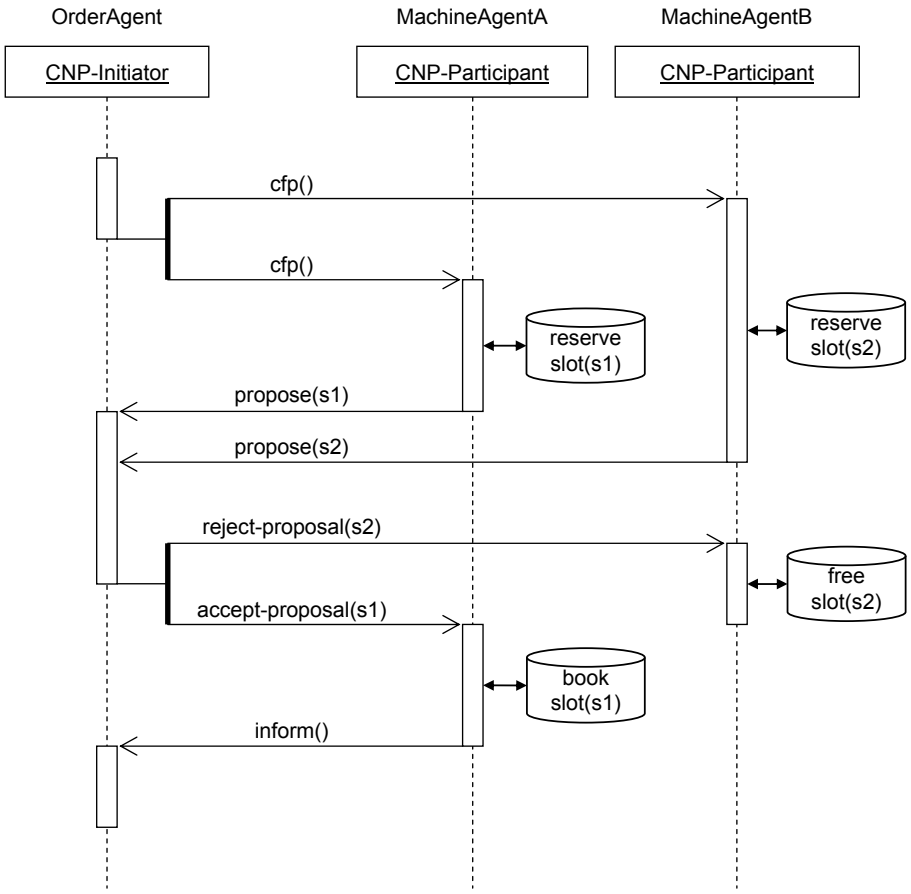


Abbildung 2.1: Ressourcenallokation mit dem Kontraktnetzprotokoll

Participants kann der Initiator entweder ein Diensteverzeichnis durchsuchen oder seine Ausschreibung an alle `MachineAgent`s richten.

Participant: Angebotserstellung Die `MachineAgent`s, an die die Ausschreibung gerichtet wurde, können basierend auf der Aufgabenbeschreibung ein Angebot (engl. *Proposal*) formulieren und an den `OrderAgent` abgeben. (Falls ein `MachineAgent` kein Interesse an der Abgabe eines Angebotes hat, kann er dies dem `OrderAgent` durch eine – in der Abbildung 2.1 nicht dargestellte – `refuse`-Nachricht mitteilen.) Da die Angebote im Kontraktnetzprotokoll bindenden Charakter haben, muss während der Angebotserstellung eine Reservierung der zur Erfüllung des Auftrags notwendigen Ressourcen erfolgen.

Initiator: Angebotsauswahl Der `OrderAgent` bewertet vergleichend die Angebote, die rechtzeitig vor Ablauf der Abgabefrist bei ihm eingegangen sind. Unter den Angeboten kann er eines (oder prinzipiell beliebig viele) auswählen und mit einer `accept-proposal`-Nachricht an den entsprechenden `MachineAgent` annehmen (siehe Abbildung 2.1: `MachineAgentA`). Die restlichen Participants werden durch eine `reject-proposal`-Nachricht von der Ablehnung ihrer Angebote unterrichtet (siehe `MachineAgentB` in Abbildung 2.1).

Participant: Leistungserbringung Der „erfolgreiche“ `MachineAgent`, dessen Angebot angenommen wurde, bucht die zuvor reservierten Ressourcen für die vereinbarten Zeiträume und beginnt zu gegebener Zeit mit der Produktion. Die `MachineAgent`s, deren Angebote abgelehnt wurden, geben ihre Reservierungen frei und können die Ressourcen im Folgenden in anderen Verhandlungen erneut anbieten. Für sie ist das Protokoll damit beendet. Der erfolgreiche Participant überwacht die Ausführung seines Angebots, d.h. im Falle des `MachineAgent`s in der Maschinenbelegungsplanung den Produktionsprozess, und meldet die Fertigstellung an den Initiator (`inform`-Nachricht). In anderen Einsatzszenarien des Kontraktnetzprotokolls, die statt eines Produktionsprozesses z.B. eine Informationsbeschaffung als Vertragsgegenstand haben, kann in der abschließenden `inform`-Nachricht auch direkt das Ergebnis der Leistungserbringung übermittelt werden.

Bisher wurden zur Vereinfachung der Beschreibung des Kontraktnetzprotokolls alle potenziellen Fehlerfälle und ihre Behandlung ausgeblendet. Lediglich das Setzen einer Ab-

gabefrist durch den Initiator könnte man als einen ersten Robustheitsmechanismus gegen verloren gegangene Angebote interpretieren. Im folgenden Abschnitt, der die Aufgabenstellung der Arbeit motiviert, werden u.a. eine Reihe von Fehlerfällen am Beispiel des Kontraktnetzprotokolls eingeführt und ihre Auswirkungen auf die beteiligten Parteien werden beschrieben.

2.1.2 Beispielhafte Fehlerfälle im Kontraktnetzprotokoll

Der Erfolg des Kontraktnetzprotokolls wird auch auf dessen Einfachheit zurückgeführt (Wooldridge 2002, Seite 197). Doch schon in diesem einfachen Agentenszenarium ergeben sich eine Reihe von Fehlerquellen mit weit reichenden Konsequenzen. In Abbildung 2.2 werden einige beispielhafte Fehlerfälle eingeführt.

Fehlerfall 1 Für den ersten Fall sei angenommen, dass `MachineAgentA` das für den `OrderAgent` günstigste Angebot aller Participants machen will. Dazu formuliert er ein Proposal und übermittelt es. Der `OrderAgent` empfängt das Proposal, kann aber nichts damit anfangen, weil er es nicht richtig interpretieren kann. Mögliche Ursachen hierfür sind z.B. Syntaxfehler, unbekannte oder semantisch falsch eingesetzte Bezeichner. Als Auswirkung wird der `OrderAgent` das zweitbeste alternative Angebot auswählen – falls vorhanden – und damit mehr als im fehlerfreien Fall notwendig bezahlen.

Fehlerfall 2 Ist das günstigste Proposal doch durch den `OrderAgent` fehlerfrei interpretierbar, wird eine `accept-proposal`-Nachricht an `MachineAgentA` verschickt. Im zweiten Fehlerfall geht diese Nachricht durch einen Übermittlungsfehler verloren. Ohne einen entsprechenden Robustheitsmechanismus sind beide Vertragspartner von den Konsequenzen aus diesem Fehler betroffen. Der `OrderAgent` wird auf die Fertigstellung seines Produktionsauftrags warten, die aber nie angestoßen wurde, und der `MachineAgentA` wird wegen der fehlenden Rückmeldung die Ressourcen reserviert halten, kann sie damit nicht in weiteren Verhandlungen anbieten und muss sie am Ende ungenutzt lassen.

Fehlerfall 3 Sollte die Rückmeldung beim `MachineAgentA` eintreffen, kann dieser gemäß des Vertrags die Produktion ausführen lassen. Bei einer Störung im Produktions-

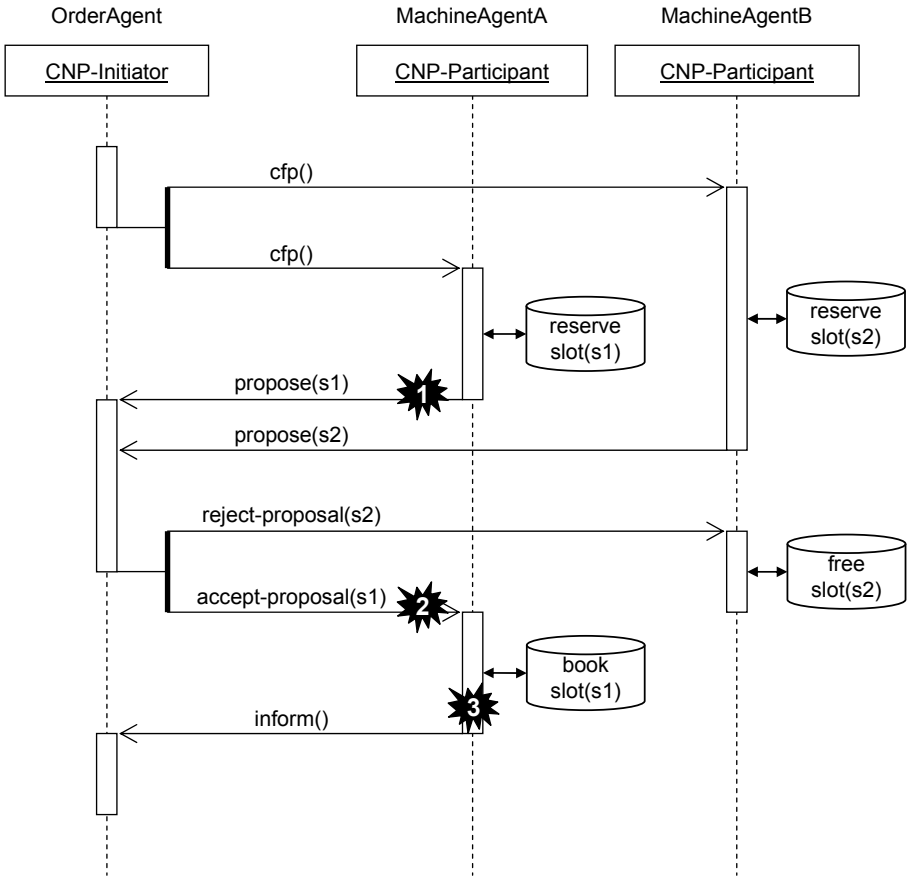


Abbildung 2.2: Beispielhafte Fehlerfälle im Kontraktnetzprotokoll

prozess kann die Fertigstellung des Erzeugnisses verzögert oder ganz verhindert werden. Damit der `OrderAgent` geeignete Gegenmaßnahmen einleiten kann, sind Robustheitsmechanismen erforderlich, die über die eigentlichen Aufgaben des Kontraktnetzprotokolls hinausgehen.

Über die drei geschilderten Fehlerfälle hinaus sind natürlich abhängig vom Einsatzzweck des Kontraktnetzprotokolls noch weitere denkbar. In Knabe, Schillo und Fischer (2002) wird z.B. ein Szenarium beschrieben, bei dem die beauftragten Participants ihrerseits wiederum Teile ihrer Leistung in Unteraufträgen vergeben. Eine fehlerhafte Unterbeauftragung führt dann z.B. ohne weitere Maßnahmen zu einer Kaskadierung der Störung.

2.2 Begriffsbildung

2.2.1 Arbeitsdefinition für Multiagentensysteme

Für eine Konkretisierung der Aufgabenstellung bedarf es zunächst einer (vorläufigen) Arbeitsdefinition für *Multiagentensysteme*. Sie ist in Einklang mit den Eigenschaften, die nach Davidsson und Johansson (2005a, b) von einem Großteil der Forschergemeinde den Multiagentensystemen zugeschrieben werden:

> *Multiagentensysteme* (MAS) sind *verteilte Systeme* aus interagierenden *autonomen Software-Komponenten* (Agenten), die in ihre *Umgebung eingebettet sind*, d.h. sie wahrnehmen und verändern.

Abbildung 2.3 illustriert diese Definition. Darin interagieren Agenten in *organisatorischen Verbänden*. In der vorliegenden Arbeit wird davon ausgegangen, dass diese Interaktion nachrichtenbasiert abläuft. Die Wechselwirkungen innerhalb eines Verbandes können über die nachrichtenbasierte Interaktion hinaus willentlich oder unwillentlich auch über die Handlungen und Wahrnehmungen auf der Umgebung erfolgen. Wechselwirkungen zwischen verschiedenen Verbänden finden nur über Veränderungen in der Umgebung

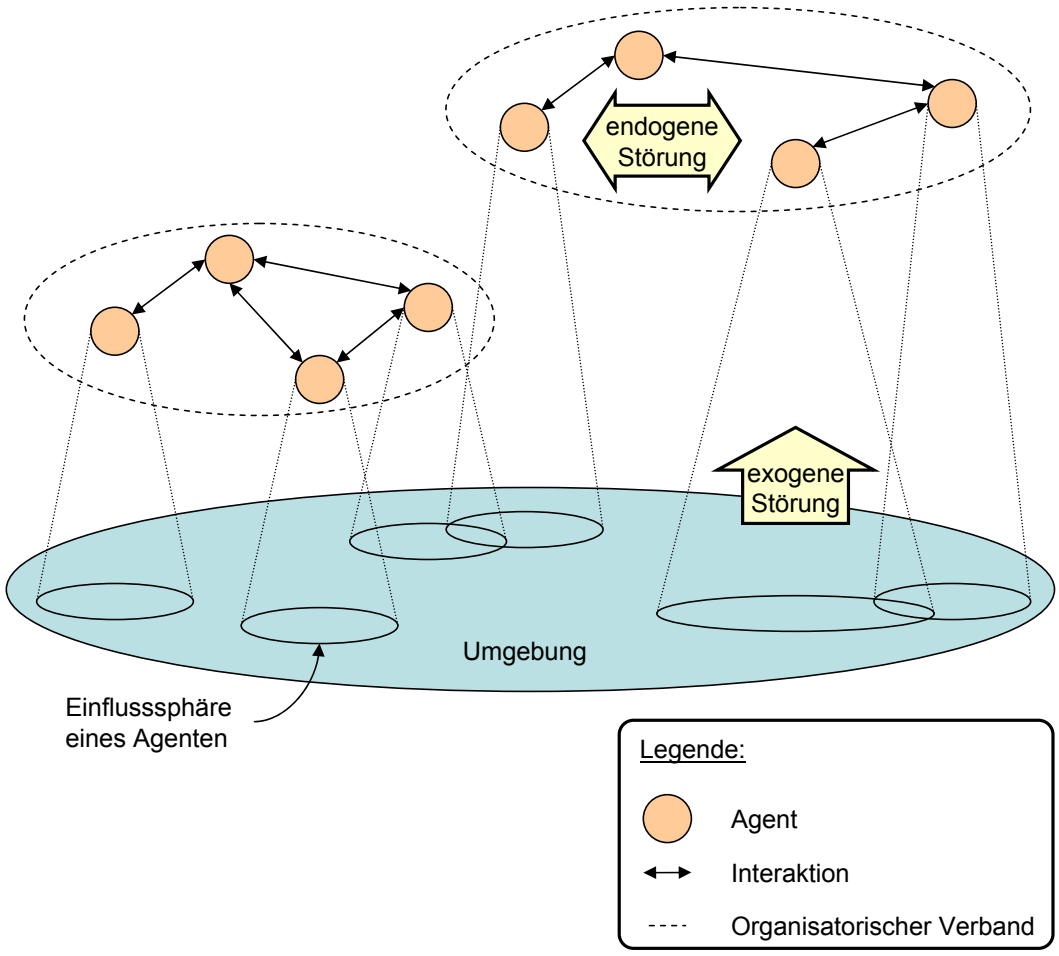


Abbildung 2.3: Vorläufige Arbeitsdefinition für Multiagentensysteme (erweitert nach Jennings 2000)

statt und sind immer unkoordiniert. Trivialerweise ist davon auszugehen, dass ein Multiagentensystem für einen oder mehrere Benutzer gleichzeitig einen nützlichen Dienst gemäß einer Spezifikation erbringt.

2.2.2 Störungen, Zuverlässigkeit und Robustheit

Die Störungen in Multiagentensystemen, wie sie in Abschnitt 2.1.2 beispielhaft eingeführt wurden, lassen sich nach verschiedenen Kriterien kategorisieren. Ein wesentliches Kriterium ist der *Entstehungsort*: Bei *exogenen* Störungen handelt es sich um unvorhergesehene Ereignisse in der Anwendungsumgebung, d.h. um Ereignisse, die beim Entwurf des Systems nicht explizit bedacht wurden und für deren adäquate Handhabe das MAS nicht über die geeigneten Mittel verfügt. *Endogene* Störungen entstammen dem MAS selbst (siehe Abbildung 2.3).

Aus Sicht eines *einzelnen* Agenten bzw. aus Sicht seiner gerade betrachteten Teilkomponente können endogene Störungen entweder externen oder internen Ursprung haben. *Interne* Störungen entstehen im betreffenden Agenten selbst bzw. in der betrachteten Teilkomponente. *Externe* Störungen haben ihren Ursprung außerhalb des betreffenden Agenten (bzw. der betrachteten Komponente), aber innerhalb der restlichen Bestandteile des Multiagentensystems, zu dem der Agent gehört, d.h. insbesondere nicht in der Anwendungsumgebung. Für zugleich *endogene und externe Störungen* bleiben zwei mögliche Störungsquellen:

Infrastrukturstörungen Agenten und ihre Teilkomponenten existieren nicht im luftleeren Raum; sie bauen auf umfangreichen Infrastrukturdiensten auf. Darunter versteht man für Agenten die Rechnerplattform mit Hardware und Betriebssystem, zusätzliche Datenhaltungssysteme, die Kommunikationseinrichtungen und auch die spezialisierten Implementierungsrahmenwerke; für eine bestimmte Teilkomponente kommen zusätzlich alle anderen Teilkomponenten des Agenten hinzu, von denen Dienste in Anspruch genommen werden.

Partnerstörungen Die anderen interagierenden Agenten im organisatorischen Verband sind eine weitere Quelle für Störungen, wenn sie ihre angeforderten Dienste nicht wie vereinbart oder vom betreffenden Agenten erwartet erbringen. Dafür kann

es verschiedene Ursachen geben, wie z.B. eine (vorübergehende) Nichterreichbarkeit oder Überlastung des Partners, was für den Dienstnehmer einem Totalausfall gleichkommt. Aber auch wenn der Partner ein Ergebnis liefert, kann dies immer noch einer Störung gleichkommen, wenn es durch unvorhergesehene Umstände nicht den Erwartungen des betreffenden Agenten entspricht, z.B. wenn bestimmte Dienstgüteeigenschaften nicht erfüllt wurden. (Siehe hierzu die Diskussion von „*unfavorable outcomes*“ in (Pleisch und Schiper 2004)).

Abbildung 2.4 gliedert die Kategorisierung der Störungen nach ihrem Ursprung in Form einer Hierarchie.

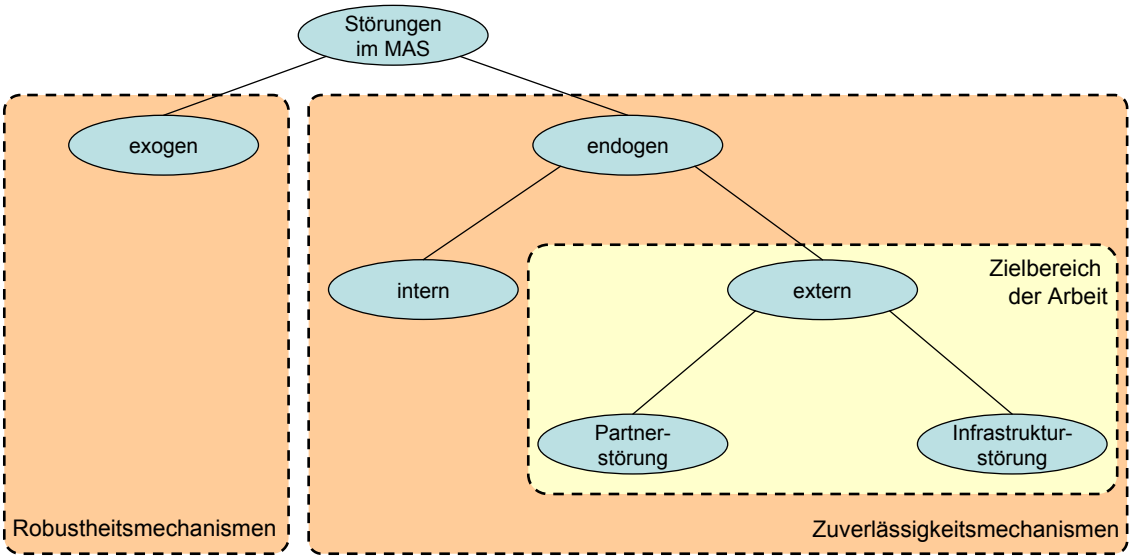


Abbildung 2.4: Hierarchie der Störungen und Zielbereich der vorliegenden Arbeit

Ein MAS wird als *zuverlässig* bezeichnet, wenn es seine Dienste entsprechend ihrer Spezifikation erbringt, unabhängig davon, ob im Verlauf der Diensterbringung endogene Störungen aufgetreten sind oder nicht. Ein MAS ist *robust*, wenn es beim Auftreten exogener Störungen nach deren Behandlung zur spezifikationsgemäßen Diensterbringung zurückkehren kann.[4] Abbildung 2.4 illustriert, gegen welche Arten von Störungen die zu-

[4] In den meisten anderen Definitionen wird zwischen Zuverlässigkeit (engl. *Dependability*) und Robustheit (engl. *Robustness*) nicht unterschieden, sondern für beide Spielarten der Begriff „Robustness“ verwendet (siehe z.B. Schillo u. a. 2001). Weitere englischsprachige Fachbegriffe und die in der vorliegenden Arbeit verwendeten deutschen Übersetzungen sind in Anhang A aufgeführt.

gehörigen Robustheits- bzw. Zuverlässigkeitsmechanismen wirken.

Ein weiteres Kriterium zur Kategorisierung von Störungen ist besonders in offenen Multiagentensystemen von Bedeutung. Dort können Agenten verschiedenen Prinzipalen zugeordnet sein und egoistisch deren Interessen vertreten. In solchen Szenarien kann es aus Sicht einzelner Agenten sinnvoll sein, *absichtlich* Störungen herbeizuführen und insbesondere andere Agenten zu „betrügen“. Absichtliche und unbeabsichtigte Störungen unterscheiden sich grundlegend in ihrer Art und auch in den zugehörigen Gegenmaßnahmen. In der vorliegenden Arbeit werden ausschließlich *unbeabsichtigte Störungen* betrachtet.

2.3 Ziel und Hauptaufgabe der Arbeit

Ziel der Dissertation ist die *Steigerung der Zuverlässigkeit von Multiagentensystemen unter der Randbedingung des Investitionsschutzes*, genauer gesagt die **Wegbereitung für Mechanismen, die die Zuverlässigkeit bei endogenen externen Störungen zu steigern vermögen und die dabei zu den anderen Programmierartefakten weitestgehend orthogonal sind**.

Die Randbedingung des Investitionsschutzes und damit die Forderung nach Orthogonalität zu vorhandenen Programmierartefakten bedeutet, dass die Zuverlässigkeitsmechanismen die etablierten Methoden, ausgereiften Programmiermodelle und konkreten Systeme möglichst unberührt lassen sollen. Dieser Forderung lässt sich am besten nachkommen, indem die endogenen externen Störungen mit generischen technischen Mechanismen weitgehend ohne konkrete Anwendungskenntnisse zur Laufzeit des Systems bekämpft werden.

Zugleich *endogene* und *externe* Störungen, die in dieser Arbeit ausschließlich behandelt werden sollen, sind genau diejenigen, die ihren Ursprung nicht im direkten Einflussbereich des betreffenden Agenten (bzw. der Teilkomponente) haben, d.h. an denen der Agent nicht „selbst schuld“ ist. Im Gegensatz dazu lassen sich die *internen* Störungen besser durch geeignete Entwurfs- und Testmethoden zum Entwicklungszeitpunkt angehen: Treten in vorbetrieblichen Tests interne Störungen auf, muss eine neue Iteration im

Entwicklungszyklus angestoßen werden (Timm u. a. 2006). Die Robustheit von MAS gegen *exogene* Störungen ist ebenfalls eine Frage des Agentenentwurfs, allerdings auf einer anderen Ebene. Durch die Ausstattung der einzelnen Agenten mit mächtigen Handlungsfähigkeiten, einer lernfähigen Verhaltensgenerierung und wandlungsfähigen Koordinationsmechanismen kann eine hohe Flexibilität auch gegen unerwartete exogene Ereignisse erreicht werden – gerade exogene Störungen sind ja auch die Hauptindikation für den Einsatz von Multiagentensystemen.

Die *Herausforderungen* bei der Steigerung der Zuverlässigkeit von Multiagentensystemen haben eine Reihe von Ursachen. Aus informationstechnischer Sicht handelt es sich bei einem Multiagentensystemen um ein Informationssystem mit verteilter Datenhaltung und verteilter Kontrolle. Bei herkömmlichen Systemen liegt in der Regel nur die Verteilung eines dieser Aspekte vor. Die lose Kopplung der autonomen Teilnehmer tut ihr übriges: prinzipiell kann sich ein Agent niemals auf eine bestimmte Wirkung seiner Aktionen in der Umgebung oder auf das Eintreten eines bestimmten Verhaltens bei seinen interagierenden Agenten verlassen, selbst wenn ihm die anderen Agenten wohl gesonnen sind und das Verhalten koordiniert war.

Das Ziel der vorliegenden Arbeit lässt sich abbilden auf eine dominante *Hauptaufgabe*, nämlich die **Zuverlässigkeitssystematisierung**. Die Forschungsbemühungen im Bereich der Zuverlässigkeit von Multiagentensystemen haben bislang zu einer sehr heterogenen Landschaft existierender Zuverlässigkeitsmechanismen geführt, bei denen der Zuverlässigkeitsaspekt stark mit den anderen Aspekten der Agentenimplementierung vermengt ist. Dies liegt vor allem am Facettenreichtum der Agententechnologie selbst und an ihrer fehlenden Systematisierung, die sich in einem Mangel an umfassenden zusammenführenden Referenzmodellen ausdrückt. Daher ist eine erste zentrale Aufgabe dieser Arbeit die *vollständige Systematisierung der Agententechnologie*, d.h. ihre Aufbereitung und Strukturierung. Diese soll in Form einer *Referenzarchitektur* auf einer möglichst fundamentalen Basis erfolgen, die im Sinne der *Allgemeingültigkeit* des Modells ohne ausgrenzende Annahmen auskommt.

In die Referenzarchitektur der Agententechnologie soll dann der Zuverlässigkeitsaspekt eingearbeitet werden mit dem Ziel einer allgemeingültigen Zuverlässigkeitsreferenzarchitektur für Multiagentensysteme. Die Zuverlässigkeitsreferenzarchitektur soll zum einen

die verschiedenen bekannten Ansätze zur Steigerung der Zuverlässigkeit verorten und damit das Forschungsgebiet der agentischen Zuverlässigkeit homogenisieren können. Zum anderen soll die *Zuverlässigkeitsreferenzarchitektur* die Lokalisierung von Störungen und die *Auswahl bzw. Eigenentwicklung entsprechender orthogonaler Zuverlässigkeitsmechanismen ermöglichen*, die für das *gesamte Multiagentensystem* Garantien bei Partner- und Infrastrukturstörungen bieten können.

Unter dem Gesichtspunkt des Investitionsschutzes ist eine möglichst orthogonale Einbettung der Zuverlässigkeitsmechanismen in die Agenten von besonderer Bedeutung, d.h. eine Einbettung, die gemäß des „Separation of Concerns“-Prinzips den Zuverlässigkeitsaspekt möglichst gut von den anderen Aspekten der Agentenimplementierung separiert. Diese Bewahrung vorhandener Strukturen in der Zuverlässigkeitsreferenzarchitektur und den Zuverlässigkeitsmechanismen ermöglicht den Entwicklern von MAS insbesondere, an den gewohnten Methoden, Programmiermodellen und Systemen festzuhalten, ohne einen großen Zusatzaufwand für die Zuverlässigkeit leisten zu müssen.

Kapitel 3

Eine Referenzarchitektur für Multiagentensysteme

Ein gravierender Nachteil vieler existierender Zuverlässigkeitsmechanismen ist ihre enge Vermaschung mit den anderen Aspekten der Agentenprogrammierung. Häufig bringen die Mechanismen ihre eigenen Programmiermodelle mit und zwingen damit den Entwickler zum Verwerfen der gewohnten Methoden, Programmiermodelle und Systeme. Der Fokus der Entwicklung wird somit weggelenkt von der Anwendungsfunktionalität, die eigentlich im Zentrum des Entwicklungsprozesses stehen sollte.

Eine gute Einbettung eines Zuverlässigkeitsmechanismus erlaubt es, bei der Programmierung den Zuverlässigkeitsaspekt weitestgehend getrennt von den anderen Aspekten der Agentenprogrammierung zu behandeln – im folgenden soll in diesem Fall von einer „*orthogonalen Einbettung*“ die Rede sein. Für eine orthogonale Einbettung eines Zuverlässigkeitsmechanismus bedarf es zunächst einer vollständigen Systematisierung der Agententechnologie, die eine Identifikation aller Aspekte der Agentenprogrammierung und ihres Zusammenspiels ermöglicht und darauf aufbauend eine Abgrenzung des Zuverlässigkeitsaspektes erlaubt.

Zur Lösung dieser Aufgabe wird in der vorliegenden Arbeit ein architekturbasierter Ansatz gewählt und eine *Referenzarchitektur für Multiagentensysteme* entwickelt, die die Programmieraspekte und ihre Beziehungen strukturiert, d.h. verortet. Die Eigenentwicklung wird erforderlich, da bisher bekannte Architekturen aus dem Bereich der

Multiagentensysteme jeweils spezifische Standpunkte einnehmen und damit weder vollständig noch allgemeingültig sind.

Die Entwicklung von Referenzarchitekturen ist ein komplexer Prozess und folgt Methoden aus der Software-Technik. Für die besonderen Anforderungen der vorliegenden Arbeit wird zunächst in Abschnitt 3.1 ein eigener Prozess entwickelt. Die Ausgangsbasis dieses Prozesses wird in Abschnitt 3.2 dargestellt und aufbereitet. Die systematische Ableitung der Referenzarchitektur erfolgt in Abschnitt 3.3. Das Kapitel schließt mit der Bewertung der Referenzarchitektur in Abschnitt 3.4, die besonders deren Allgemeingültigkeit und Anwendbarkeit untermauert, indem u.a. bekannte Architekturen für Multiagentensysteme auf die Referenzarchitektur abgebildet werden.

3.1 Entwicklung von Referenzarchitekturen

Referenzarchitekturen sind spezielle Ausprägungen von Software- bzw. Systemarchitekturen[1]. Folglich sind viele Aussagen über Software-Architekturen auch für Referenzarchitekturen bedeutsam, insbesondere über ihre grundlegenden Eigenschaften und ihren Entwicklungsprozess.

Systemarchitekturen beschreiben die wesentlichen Bestandteile eines Systems und ihre Beziehungen untereinander. (Software-)Architekturen umfassen nach Reussner und Hasselbring (2006) i.a. aber noch weitere Sichten:

> Die Architektur eines Software-Systems beschreibt dieses zunächst als Komponenten zusammen mit den Verbindungen, die zwischen den Komponenten bestehen. Dazu gehören zwei weitere Sichten: eine Beschreibung der Dynamik, d.h die Art wie der Kontrollfluss durch die Komponenten fließt, sowie

[1] In der vorliegenden Arbeit wird im Zusammenhang mit der Architektur von Multiagentensystemen bewusst der Begriff „*Systemarchitektur*“ verwendet im Gegensatz zum gebräuchlicheren Begriff „*Software-Architektur*“. Dies geschieht, um den umfassenderen Charakter der Architektur von Multiagentensystemen im Bezug auf die Hardware- und Anwendungsumgebung zu unterstreichen. Die meisten Erkenntnisse aus der Software-Technik gelten jedoch in angepasster Form weiterhin. In der Literatur wird homonym in anderem Zusammenhang der Begriff der Systemarchitektur für Rechnerarchitekturen (z.B. i486-PC, SPARC,...) eingesetzt.

> die Abbildung der Komponenten und Verbindungen auf Ressourcen (Prozesse, virtuelle Maschinen) bzw. auf logische oder physikalische Verbindungen. Eine Software-Architektur beschreibt damit nicht den detaillierten Entwurf, vielmehr geht es darum, die Zusammenhänge zwischen den Anforderungen und dem konstruierten System zu beschreiben, möglichst mit einer Begründung der Entwurfsentscheidungen. Die Wahl einer bestimmten Architektur hat dann einen starken Einfluss auf die nichtfunktionalen (qualitativen), technischen Eigenschaften der resultierenden Systeme.

Referenzarchitekturen unterscheiden sich von gewöhnlichen Software-Architekturen anhand ihres Abstraktionsniveaus und ihres Verwendungszwecks (Riebisch 2006). Eine Referenzarchitektur definiert auf einer hohen Abstraktionsebene prinzipielle Lösungsstrukturen und gibt damit einen Rahmen für eine Familie von Systemen vor. Beneken (2006) charakterisiert den Verwendungszweck von Referenzarchitekturen wie folgt:

> Die Architektur eines Systems ist die Verfeinerung bzw. Anpassung einer oder mehrerer Referenzarchitekturen. Die Komponenten aus dem Architekturüberblick werden konkretisiert oder modifiziert, bereits vorhandene Komponenten und Produkte können an definierten Stellen verwendet werden. Vorgegebene Schnittstellen werden implementiert. So wird die Referenzarchitektur zum Ausgangspunkt der Entwicklung neuer Software-Systeme [...]. Sie dient den Architekten als wiederverwendbarer Ausgangspunkt für Spezifikationen und Entwürfe.

Eine wesentliche Rolle bei der Entwicklung einer System- bzw. Referenzarchitektur spielen also ihre Komponenten oder auch „Bausteine“. Alle Bausteine zeichnen sich durch eine abgrenzbare Kompetenz (bzw. Zuständigkeit) aus, die sie als *Dienst* in die Systemarchitektur einbringen. Die Beziehungen in der Systemarchitektur ergeben sich dann direkt aus den Dienstgeber-/Dienstnehmer-Beziehungen zwischen den Bausteinen. Die Erbringung eines Dienstes durch einen Baustein ist aber nicht nur eingeschränkt auf die *Dienstfunktionalität* zu betrachten. Vielmehr spielen die *Dienstgütemerkmale* (auch *Dienstqualität* oder *QoS*) eine bedeutende Rolle, da sie die Implementierung eines Softwaresystems wesentlich beeinflussen und damit indirekt auch dessen Systemarchitektur

prägen (siehe oben und Prolog zu Lockemann und Dittrich (2004)).

Betrachtet man nun einen Agenten selbst als einen aus Komponenten aufgebauten Diensterbringer, dann folgt, dass sich seine Systemarchitektur aus den ihm zugeschriebenen Eigenschaften ableiten lässt. Über diese Eigenschaften hat sich in der Forschungsgemeinde auf abstrakter Ebene ein gewisser Konsens gebildet (Davidsson und Johansson 2005a, b). In der weiteren Arbeit wird dies ausgenützt, indem die allgemein anerkannten definierenden Eigenschaften ein wesentliches Fundament für die entwickelte Referenzarchitektur bilden und sich damit eine weitreichende Allgemeingültigkeit erreichen lässt.

Die zu entwickelnde Architektur soll aber nicht nur *allgemeingültig* sein, d.h. möglichst wenige ausgrenzende Annahmen verkörpern, sondern sie soll auch möglichst *vollständig* sein, d.h. alle relevanten Aspekte der Agententechnologie umfassen. Die in der Literatur bedeutenden Architekturen sind durchweg nicht vollständig in obigem Sinne, da sie abhängig vom vorliegenden Forschungsschwerpunkt unterschiedliche Blickwinkel auf die Architektur einnehmen. Sie lassen sich außerdem nach der „Betrachtungsnähe“ unterscheiden, aus der die Multiagentensysteme untersucht werden. *Drei Typen von Systemarchitekturen* sind so identifizierbar:

1. auf der *Makroebene* Systemarchitekturen für Multiagentensysteme als Ganzes, die das Zusammenspiel der verschiedenen Agenten im Multiagentensystem festlegen,

2. auf *mittlerer Ebene* Systemarchitekturen für einzelne Agenten, die meist aus Software-technischer Sicht die Bausteine der Agenten beschreiben und

3. auf der *Mikroebene* innere Architekturen, die auf die intelligente Verhaltensgenerierung der einzelnen Agenten beschränkt sind.

Eine vollständige Architektur muss die drei genannten Architekturtypen zusammenführen und die jeweiligen Aspekte bzw. Komponenten zueinander in Beziehung setzen. Da die konkreten Vertreter der drei Architekturtypen spezifische Blickwinkel entsprechend des jeweiligen Forschungsschwerpunkts einnehmen, muss zugleich bei diesem Zusammenführungsschritt auch eine Abstraktion erfolgen zugunsten der Allgemeingültigkeit. Aus diesen Vorüberlegungen resultiert eine strukturierte Vorgehensweise, die im nächsten Abschnitt beschrieben wird.

3.1.1 Vorgehensweise

Die Vorgehensweise bei der Entwicklung von Referenzarchitekturen wird in der Literatur eher mit einem Reife- als einem Entwicklungsprozess verglichen: Referenzarchitekturen werden weniger im Rahmen eines Projektes „von Null an“ entwickelt, sondern sie werden vielmehr aus den Erfahrungen vieler Projekte destilliert und expliziert (Beneken 2006). Abbildung 3.1 beschreibt den Reifeprozess von Referenzarchitekturen.

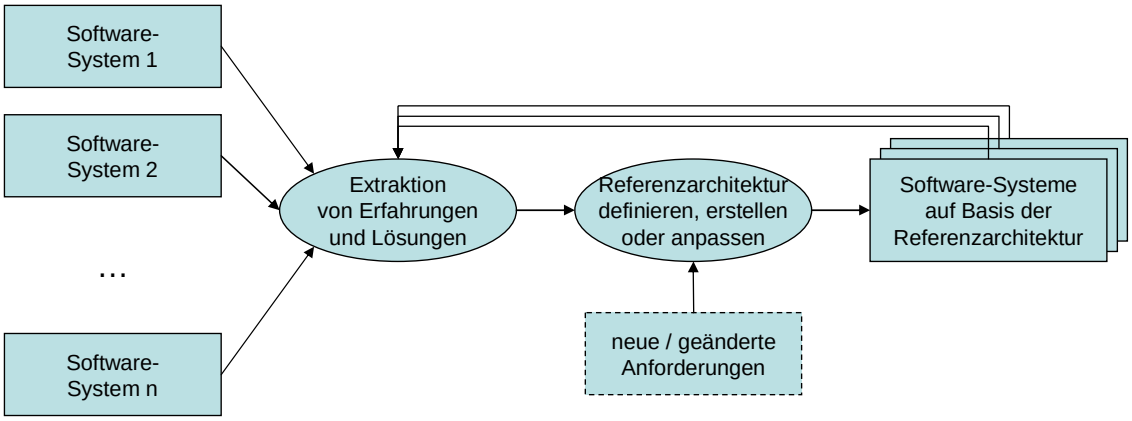


Abbildung 3.1: Entwicklungsprozess von Referenzarchitekturen als Reifeprozess

Zur Erstellung einer ersten Version der Referenzarchitektur werden zunächst existierende Software-Systeme auf gemeinsame erfolgreiche Lösungsbestandteile hin untersucht. Diese werden extrahiert und in einer neuen Referenzarchitektur zusammengeführt. Dieses erste Zwischenergebnis wird abgeglichen mit den Anforderungen, die üblicherweise an ein System der zu beschreibenden Familie von Software-Systemen gestellt werden. Neue Systeme aus der untersuchten Familie werden auf Basis der angepassten Referenzarchitektur spezifiziert und die dabei neu gewonnenen Erkenntnisse über die Referenzarchitektur fließen in einer Rückkopplung wiederum in die zukünftigen Versionen der Referenzarchitektur ein.

Der geschilderte Reifeprozess ist in der vorliegenden Arbeit nicht anwendbar: legt man ausschließlich die eigenen existierenden Systeme zugrunde, dann resultiert eine Referenzarchitektur, die typische Charakteristika des eigenen Programmierstils aufweist. Auf konkrete fremde Systeme besteht jedoch nicht ausreichender Zugriff. Es muss also zur

Wahrung der Allgemeingültigkeit eine breitere Ausgangsbasis gefunden werden. Diese manifestiert sich in den definierenden Eigenschaften der Multiagentensysteme, die ja gerade die Gemeinsamkeiten aller Agentensysteme beschreiben, in den Standards, die als Basis zahlreicher Implementierungen dienen, und in den verwandten Arbeiten auf dem Gebiet der Referenzarchitekturen. Auf dieser Ausgangsbasis wurde der in Abbildung 3.2 illustrierte Entwicklungsprozess für die Referenzarchitektur der Multiagentensysteme entworfen.

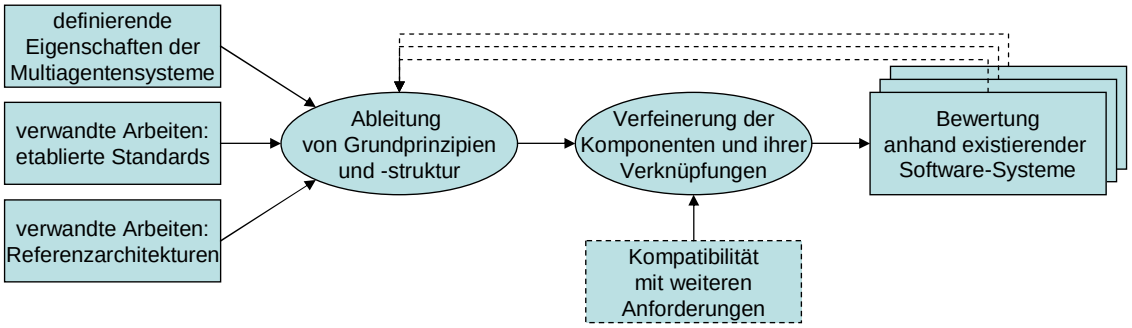


Abbildung 3.2: Entwicklungsprozess für die Referenzarchitektur der Multiagentensysteme

Von der Ausgangsbasis zur vollständigen und allgemeingültigen Referenzarchitektur gelangt man in drei Schritten:

Grundstruktur Bereits aus der alleinigen Betrachtung der grundlegenden Eigenschaften von Agenten und Multiagentensystemen lassen sich die Grundprinzipien und die Grundstruktur der Referenzarchitektur ableiten.

Komponenten Die Verfeinerung der Komponenten der Grundstruktur und ihrer Verknüpfungen folgt im zweiten Schritt durch die Betrachtung der für die Intelligenz maßgeblichen Eigenschaften, der etablierten Standards besonders im Bereich der Agentenkommunikation und der verwandten Referenzarchitekturen.

Bewertung Die Bewertung der Referenzarchitektur erfolgt anhand der Abbildung bekannter gebräuchlicher Multiagentensystem-Architekturen auf die Referenzarchitektur („Konformität“). Diese muss auf möglichst natürliche Weise erfolgen können, um Allgemeingültigkeit und Vollständigkeit – und damit die Anwendbarkeit – der Referenzarchitektur zu untermauern. Die Konformität in obigem Sinne ermöglicht

die Verortung der Programmieraspekte existierender Agentensysteme und erlaubt damit erst, den später zu erwartenden Nutzen der Referenzarchitektur auszuschöpfen. Zusätzliche Anforderungen können durch die erwünschte Umsetzung weiterer möglicher Agenteneigenschaften an die Referenzarchitektur gestellt werden. Diesbezüglich muss die Architektur über eine ausreichende Flexibilität verfügen, die die Umsetzung dieser Agenteneigenschaften erlaubt („Kompatibilität“).

Aus dem beschriebenen Entwicklungsprozess ergibt sich für den Rest des Kapitels der folgende Aufbau: Zunächst werden im folgenden Abschnitt 3.2 die Fundamente der Referenzarchitektur ausführlich beschrieben. Dazu zählen die grundlegenden Eigenschaften (3.2.1), die etablierten Standards (3.2.2) und die verwandten Referenzarchitekturen (3.2.3). Die beiden letzteren stellen zugleich die verwandten Arbeiten und damit den Stand der Forschung für die Referenzarchitektur dieser Arbeit dar. Die Entwicklung der Referenzarchitektur ist in Abschnitt 3.3 dargestellt, in dem zuerst die Grundstruktur entworfen wird (3.3.1) und dann schrittweise die Komponenten und ihre Verbindungen verfeinert werden (3.3.2). Abschließend erfolgt eine Bewertung hinsichtlich der Anwendbarkeit und Flexibilität in Abschnitt 3.4, in dem die Kompatibilität zu optionalen Agenteneigenschaften (3.4.1) und die Konformität bekannter Multiagentensystem-Architekturen (3.4.2) diskutiert wird.

3.2 Fundamente der Referenzarchitektur

3.2.1 Definierende Eigenschaften der Multiagentensysteme

Die Definition eines allgemein anerkannten Agentenbegriffes ist nicht einfach: Im Forschungsgebiet Agententechnologie treffen eine ganze Reihe verschiedener Wissenschaften aufeinander, wie z.B. die Informatik, die Wirtschaftswissenschaften oder auch die Soziologie. Das führt zu einer Vielfalt an uneinheitlichen Agentendefinitionen, die den Blickwinkel und die Schwerpunkte des jeweiligen Faches betonen, maßgebliche Merkmale anderer Gebiete aber vernachlässigen. Eine gute Gegenüberstellung solcher Definitionen, aus der einige herausgegriffen werden sollen, bietet (Franklin und Graesser 1996).

Eine grundlegende, aber nicht sehr restriktive Definition geben Russell und Norvig (2004, Seite 55).

> „Ein *Agent* ist alles, was seine *Umgebung* über *Sensoren* wahrnehmen kann und in dieser Umgebung durch *Aktuatoren* handelt."

Diese Definition sagt nichts über die Funktionsweise der Agenten, ihre Beziehung zum Anwender oder zu anderen Agenten aus. Sie lässt auch den viel zitierten Heizungsthermostat als Agent zu. Ganz im Gegensatz dazu steht die Definition von Software-Agenten aus (Genesereth und Ketchpel 1994):

> „*[S]oftware agents* [are] software components that communicate with their peers by exchanging messages in an expressive *agent communication language*."

Hier steht die Interaktion der Agenten und damit die Agententechnologie als Paradigma für verteiltes Rechnen im Vordergrund und die Umgebungssituiertheit verschwindet völlig. Mit dieser Definition ist die Existenz isolierter Agenten ausgeschlossen, gleichzeitig werden jedoch wiederum keinerlei Einschränkungen bezüglich der Funktionsweise oder der Beziehung zum Agentenanwender getroffen.

Genau diese Beziehung steht im Mittelpunkt einer weiteren Gruppe von Definitionen. Diese sieht die Stellvertreterfunktion als wichtigste definierende Eigenschaft an: Agenten erledigen Aufgaben im Auftrag ihres Benutzers. Nun ist aber von jedem Anwendungsprogramm zu erwarten, dass es eine Funktionalität im Sinne seines Benutzers anbietet (Petrie 1997). Wichtiger scheint zu sein, dass die Agenten diese Funktionalität weitgehend unabhängig von Benutzereingriffen erbringen. Es lässt sich also vermuten, dass sich hinter dieser Definition die Autonomie-Eigenschaft verbirgt.

Autonomie ist die definierende Eigenschaft, über die in der Agentengemeinde der weitestgehende Konsens herrscht (Davidsson und Johansson 2005a, b). Sie findet Eingang in die momentan gebräuchlichste Definition aus (Wooldridge 2002), die in ähnlicher Form auch an anderer Stelle anzutreffen ist, wie z.B. in (Maes 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."

Diese Definition ist die Verdichtung einer Charakterisierung von Agenten anhand ihrer Eigenschaften in (Wooldridge und Jennings 1995), die im Folgenden ausführlich dargestellt und erweitert werden soll. Dabei werden die einzelnen Eigenschaften so weit aufbereitet, dass sie später als Fundamente der Referenzarchitektur für Softwareagenten dienen können (siehe auch Lockemann und Nimis 2005; Lockemann 2006; Lockemann u. a. 2006).

Notwendige Eigenschaften einfacher Agenten

Eigenschaft E1 (Einbettung in Umgebung):
Ein Agent ist ein Rechnersystem, das *in seine Umgebung eingebettet* ist.

Zur Umgebung zählen im Allgemeinen die Anwendungsumgebung, die durch Sensoren wahrgenommen und durch Aktoren verändert wird, im weiteren Sinne aber auch der Agentenanwender und soweit vorhanden die anderen Agenten im Multiagentensystem.

Eigenschaft E2 (Gekapselte Diensterbringung):
Ein Agent erbringt einen *nützlichen Dienst*. Seine Auswirkungen können nur von außen wahrgenommen werden, d.h. die innere Funktionsweise ist *gekapselt* und bleibt damit verborgen.

Die ersten beiden Eigenschaften sind noch nicht besonders einschränkend. Es sollte von jeder herkömmlichen Software zu erwarten sein, dass sie in einer bestimmten Anwendungsumgebung läuft und einen nützlichen Dienst erbringt. Spezifischer sind die weiteren Merkmale, die beschreiben, *wie* ein Agent seine Dienstfunktionalität erbringt (*Dienstmerkmale*).

Eigenschaft E3 (Autonomie):
Ein Agent handelt *autonom* in seiner Umgebung.

Autonomes Handeln bedeutet, dass der Agent selbst die vollständige Kontrolle über seinen inneren Zustand und damit über sein Verhalten hat. Bei seinem Handeln ist er nicht auf Interaktion mit einem Benutzer angewiesen. Dies führt direkt zur nächsten Eigenschaft.

Eigenschaft E4 (Zielorientierung):
Die Autonomie von Agenten ist *zielorientiert*.

Als Verhaltensvorgabe hat ein Agent zunächst eine Beschreibung von Zielen, die er verfolgen soll. Sein Verhalten entsteht dann durch Abwägen zwischen diesen z.T. widersprüchlichen Zielen (engl. *goal deliberation*) und durch die Wahl der besten Aktion zur Verfolgung des momentan höchstpriorisierten Ziels (engl. *practical reasoning*).

Die abstrakte Architektur zustandsbehafteter Agenten in Abbildung 3.3 veranschaulicht die bis hierher beschriebenen Eigenschaften. Mittels der Funktion *see* wird die Umgebung wahrgenommen. Diese Wahrnehmung wird zusammen mit dem Bild, das der Agent vor der Wahrnehmung von der Umgebung hatte (*state*), zu einem neuen Umgebungsbild verarbeitet (Funktion *next*). Basierend auf dem Umgebungsbild und den Zielen des Agenten erfolgt die Wahl der als nächstes auszuführenden Aktion durch die Funktion *act*.

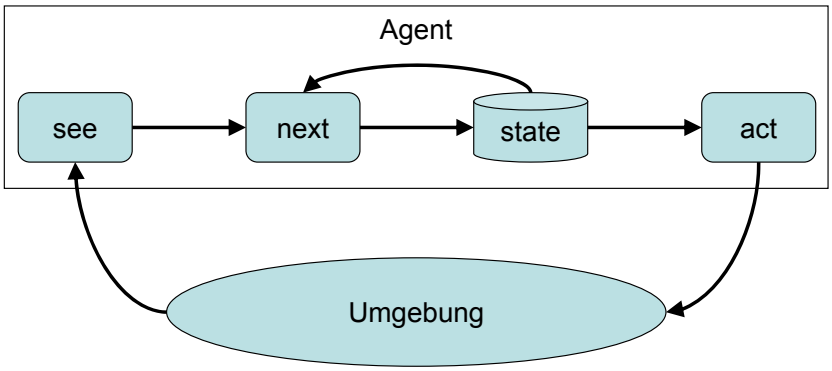


Abbildung 3.3: Abstrakte Architektur zustandsbehafteter Agenten (Wooldridge 2002)

Die Eigenschaften Autonomie und Zielorientierung grenzen Agenten deutlich von anderen Programmen wie z.B. den objektorientierten ab. Objekten können in der Regel keine eigenen Ziele und kein autonomes Handeln unterstellt werden.

Intelligente Agenten und schwierige Umgebungen

Wenn die Rede auf Agenten kommt, sind meist nicht die bis hierher beschriebenen *einfachen Agenten* gemeint, sondern Agenten mit ganz besonderen Fähigkeiten, die für den Beobachter den Anschein intelligenten Verhaltens erwecken und folglich als *intelligente Agenten* bezeichnet werden. Leider ist es nicht einfach, den Begriff der Intelligenz im Allgemeinen zu definieren; glücklicherweise lässt er sich jedoch im Agentenkontext auf einige weitere Agenteneigenschaften abbilden. Hinter diesen Eigenschaften steht in der Literatur häufig die folgende implizite Annahme:

Definition 1 (Intelligenter Agent): *Intelligente Agenten* beherrschen *schwierige Umgebungen*.

Dies wirft die Frage nach einer Klassifikation von Umgebungen auf, die von Russell und Norvig (2004, Seite 66ff.) beantwortet wird. Umgebungen sind nach den folgenden Kriterien unterscheidbar:

Beobachtbarkeit Agenten handeln aufgrund von Beobachtungen ihrer Umgebung. Eine Umgebung, in der dem Agenten alle relevanten Informationen immer zugänglich sind, bezeichnet man als *vollständig beobachtbar*. Bleiben ihm bestimmte relevante Zustände verborgen, bezeichnet man sie als *teilweise beobachtbar*.

Determiniertheit Ist der nächste Zustand einer Umgebung alleine durch ihren momentanen Zustand und durch die Handlung des Agenten festgelegt, so spricht man von einer *deterministischen* Umgebung, andernfalls von einer *stochastischen*. Als Folge ist in einer deterministischen Umgebung die Wirkung einer Handlung des Agenten alleine durch den momentanen Zustand seiner Umgebung vorhersagbar. In einer stochastischen Umgebung kann die Wirkung einer Handlung auf die Umgebung hingegen nicht genau vorausgesagt werden, z.B. weil sich die Umgebung zusätzlich durch externe Einflüsse verändern kann.

Relevanz der Historie Das Leben eines Agenten verläuft in Zyklen („Episoden“) aus Umgebungswahrnehmung und daraus gefolgerter Handlung. Eine Umgebung wird als *episodisch* bezeichnet, wenn die Handlungsauswahl alleine von der Wahrnehmung in dem zugehörigen Zyklus abhängt. Fließen Beobachtungen aus vorange-

gangenen Zyklen in die Auswahl mit ein, handelt es sich um eine *sequenzielle*[2] Umgebung.

Veränderungsgeschwindigkeit Falls sich die Umgebung zwischen der Wahrnehmung durch den Agenten und seiner daraus gefolgerten Handlung derart verändern kann, dass die Wirkung der Handlung bei deren Abschluss bereits nicht mehr adäquat ist, spricht man von einer *dynamischen* Umgebung, andernfalls von einer *statischen*.

Zustandskontinuität Eine Umgebung mit einer endlichen Anzahl von Zuständen und einer endlichen Anzahl von darin ausführbaren Handlungen wird als *diskret* bezeichnet. Umgebungen, in denen beispielsweise kontinuierliche Messwerte beobachtet werden, bezeichnet man als *stetig*.

Agentische Besiedlung Unterschieden werden hier im Wesentlichen Umgebungen, die von *einem* Agenten besiedelt werden (*Einzelagentensystem*) und Umgebungen, in denen sich *mehrere* Agenten befinden (*Multiagentensystem*). Bei letzteren ist darüber hinaus eine Trennung zwischen Umgebungen mit *konkurrierenden* bzw. *kooperativen* Agenten sinnvoll.

Offensichtlich sind die genannten Kriterien nicht orthogonal, sondern stehen in enger Wechselwirkung zueinander: Umgebungen mit konkurrierenden Agenten sind häufig zugleich stochastisch, da sich die Wirkungen der Aktionen der verschiedenen Agenten in der Regel beeinflussen werden. Zugleich kann hier von einer dynamischen Umgebung ausgegangen werden, falls die Aktionen nicht „getaktet“ werden.

Die „schwierigste“ Umgebung liegt vor, wenn diese teilweise beobachtbar, stochastisch, sequenziell, dynamisch, stetig und von mehreren Agenten besiedelt ist. Agenten, die solche Umgebungen beherrschen, als intelligent zu bezeichnen, scheint gerechtfertigt. Im Folgenden wird gezeigt, welche Eigenschaften Agenten dazu befähigen und sie somit zu *intelligenten Agenten* machen.

Eigenschaft E5 (Reaktivität):
Ein intelligenter Agent ist *reaktiv*, d.h. er beobachtet ständig seine Umgebung und reagiert zeitnah auf Änderungen.

[2] *„Sequenziell“* wird hier im Sinne von *„fortlaufend“* verwendet und nicht wie sonst häufig in der Informatik zur Unterscheidung von sequenziellen und nebenläufigen Vorgängen.

Durch seine Reaktivität gelingt es dem Agenten, sich den Veränderungen in seiner Umgebung auch in dynamischen oder Multiagenten-Umgebungen ständig kurzfristig anzupassen.

Eigenschaft E6 (Ausgewogenheit zwischen Reaktivität und Zielorientierung):
Ein intelligenter Agent hält ein sinnvolles *Gleichgewicht zwischen Zielorientierung und reaktivem Verhalten*.

Für reaktive Agenten besteht die Gefahr, dass sie über die ständige Anpassung ihres Verhaltens die längerfristigen Ziele aus den Augen verlieren, d.h. dass ihre Handlungen nicht mehr zielführend sind für die gerade gesteckten Ziele. Intelligenten Agenten gelingt die Balance zwischen Veränderung des eigenen Verhaltens und dem Beharren darauf. Eine gute Zielverfolgung setzt aber noch mehr voraus: die Ziele des Agenten sollten nicht nur auf einen externen Stimulus, beispielsweise in Form einer Beobachtung, überprüft werden; intelligente Agenten sollten vielmehr eigene Initiative bei der Zielverfolgung entwickeln:

Eigenschaft E7 (Proaktivität):
Intelligente Agenten handeln *proaktiv*, d.h. sie übernehmen zur Verfolgung ihrer Ziele die Initiative.

Bei der Verfolgung ihrer Ziele können Agenten auf die Hilfe anderer Agenten angewiesen sein oder mit ihnen in Konkurrenz stehen.

Eigenschaft E8 (Sozialfähigkeit):
Ein intelligenter Agent kann *soziale Fähigkeiten* besitzen, d.h. bei der Verfolgung seiner Ziele kann der Agent sinnvoll mit anderen Agenten interagieren.

Auch wenn die Eigenschaft E8 die Existenz isolierter intelligenter Agenten nicht ausschließen will, so ist doch in den allermeisten Fällen anzunehmen, dass Agenten mit anderen interagieren (Wooldridge 2002, Seite 105). Da in dieser Arbeit Multiagentensysteme betrachtet werden, gilt hier: Die Eigenschaften E1 bis E8 sind notwendige Eigenschaften für intelligente Agenten. Darüber hinaus gibt es noch weitere optionale Merkmale, die im folgenden Abschnitt behandelt werden.

Optionale Eigenschaften von Agenten

Eigenschaft E9 (Lernfähigkeit):
Intelligente Agenten können *lernfähig* sein.

Lernfähigkeit wird häufig als wichtiger Bestandteil der Intelligenz betrachtet. Ein Agent beobachtet dazu die Auswirkungen seiner durchgeführten Handlungen in der Umgebung und bezieht die gewonnene Erfahrung in die künftige Auswahl seiner Aktionen mit ein. Insbesondere sollten Fehler nicht mehrmals begangen werden, d.h. nicht zielführende oder kontraproduktive Handlungsweisen sollten künftig schwächer berücksichtigt werden, erfolgreiche hingegen stärker.

Unabhängig von der Intelligenz gibt es eine weitere, besonders in der Forschung häufig anzutreffende Agenteneigenschaft:

Eigenschaft E10 (Mobilität):
Agenten können *mobil* sein.

Mobile Agenten können ihre physikalische Arbeitsumgebung – bei Softwareagenten „die Plattform“ – wechseln. Denkbare Anwendungsszenarien sind die Informationssuche im Internet, bei der aus Performanzgründen Agenten die Informationsquellen besuchen, oder Arbeitsumgebungen mit zeitweise abgekoppelten Endgeräten, deren Benutzer ihre Agenten während der Abkopplung auf dauerhaft betriebene Plattformen entsenden.

Der „starke“ Agentenbegriff

Das Agentenverständnis, das der obigen Agentendefinition anhand der Eigenschaften E1 bis E8 zugrunde liegt, wird in der Literatur als „schwacher Agentenbegriff“ (engl. *weak notion of agency*) bezeichnet. Ihm gegenübergestellt wird der „starke Agentenbegriff“ (engl. *strong notion of agency*), der seine Wurzeln in der Künstlichen Intelligenz (KI) hat (Wooldridge und Jennings 1995). Dort wird davon ausgegangen, dass Agenten auf Modellierungs- und/oder Implementierungsebene Konzepte einsetzen, die gewöhnlicherweise Menschen zugeschrieben werden. Dazu zählen mentalistische Begriffe wie Wissen

(engl. *knowledge*), Glauben (engl. *beliefs*), Absichten (engl. *intentions*) oder Überzeugungen (engl. *obligations*). Noch weiter gehen Forschungsansätze, die Agenten mit Gefühlen (engl. *emotions*) betrachten. Die Definition des starken Agentenbegriffs beschreibt Agenten nicht mehr nur anhand ihrer nach außen sichtbaren Eigenschaften, vielmehr werden Annahmen über den inneren Aufbau der Agenten gemacht.

Multiagentensysteme

Eigenschaft E8 der Agentendefinition ist die Forderung nach sozialen Fähigkeiten der Agenten, d.h. nach der Fähigkeit, Teil eines so genannten Multiagentensystems (*MAS*) zu sein. Für die vorliegende Arbeit soll diese Eigenschaft als verpflichtend angesehen werden, was im betrachteten Anwendungsszenario und vielen anderen Domänen keine Einschränkung darstellt: meist trifft man auf mehr als einen Akteur in der Anwendungsdomäne, der sich in einem Informationssystem auf natürliche Weise als Agent abbilden lässt. Für diesen Umstand hat sich in der Forschungsgemeinde folgende Redensart etabliert (Wooldridge 2002, Seite 105):

> „There's no such thing as a single agent system."

Die Erweiterung der Betrachtung von Agenten auf Multiagentensysteme eröffnet eine Fülle neuer Dimensionen (Shehory 2000, 1998). Wooldridge (2002, Seite 3) beschreibt diesen Übergang wie folgt:

> „A multiagent system is one that consists of a number of agents, which interact with one another, typically by exchanging messages through some computer network infrastructure. [...] In order to successfully interact, these agents will thus require the ability to cooperate, coordinate, and negotiate with each other [...]."

Auf technischer Seite stellt sich also die Frage nach einer geeigneten Kommunikation, auf konzeptioneller Ebene nach den anzuwendenden Koordinationsmechanismen. Abbildung 3.4 unterscheidet bei letzteren grundlegend zwischen kooperierenden MAS, bei denen ein gemeinsames übergeordnetes Ziel unterstellt werden kann (sog. *geschlossene MAS*), und MAS, bei denen eine Wettbewerbssituation zwischen den einzelnen Agenten vorliegt, die einen Interessensausgleich erfordert (*offene MAS*).

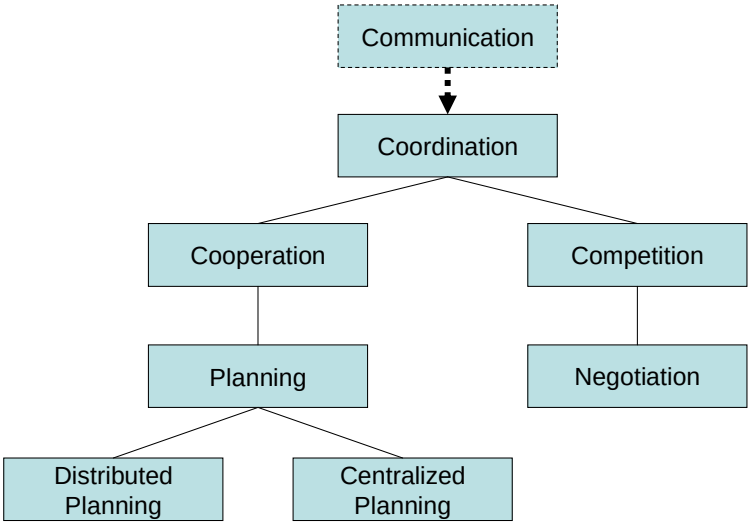


Abbildung 3.4: Taxonomie agentischer Kooperationsmechanismen (nach Huhns und Stephens 1999)

Kooperation zwischen Agenten erfordert die Abstimmung der Aktionen der verschiedenen Agenten zur Erreichung des gemeinsamen Ziels, d.h. es muss ein gemeinsamer Plan entwickelt werden, wobei auch schon dieser Planungsprozess unter den Agenten verteilt stattfinden kann.

In offenen MAS herrscht Wettbewerb, dessen Interessensausgleich nach verschiedenen Spielregeln (siehe *Rules of Encounter*, Rosenschein und Zlotkin 1998) erfolgen kann, die auch als *Mechanismen* bezeichnet werden. Eine häufig anzutreffende und gut erforschte Sonderform sind die verschiedenen Auktionen; fortgeschrittene Verhandlungsformen lösen sich von vorgegebenen Protokollen und setzen symbolische Argumentation ein.

Technische Grundlage für alle Koordinationsmechanismen ist eine entsprechend mächtige Kommunikationsfähigkeit, wie sie z.B. durch gerichtete Nachrichtenübermittlung erbracht werden kann. Vorstellbar ist aber auch Kommunikation über gemeinsam schreib- und lesbare Anschlagbretter (siehe *blackboard systems*, Engelmore und Morgan 1988)), über Spuren, die Agenten in ihrer Umgebung hinterlassen, wie z.B. durch spezielle Koordinationsartefakte (siehe *Coordination Artifacts*, Omicini u. a. 2004) oder bei Agentensystemen, die Insektenschwärmen nachempfunden sind, durch die sog. *Pheromone*

(Bonabeau u. a. 1999).

Im Rahmen dieser Arbeit soll ausschließlich direkte nachrichtenbasierte Kommunikation betrachtet werden, wie sie heute in den allermeisten Systemen anzutreffen ist (s.o.). Ihre Spezifikation liegt auch in Zentrum der etablierten Standards, die im nächsten Abschnitt als weitere Grundlage der zu entwickelnden Referenzarchitektur beschrieben werden.

3.2.2 Verwandte Arbeiten: etablierte Standards

Besonders Multiagentensysteme, die von mehreren Institutionen entwickelt werden, brauchen Übereinkünfte auf verschiedenen Ebenen, um die zur Koordination notwendige Interoperabilität gewährleisten zu können. Im Laufe der Zeit gab es daher einige Gremien, die sich der Standardisierung von MAS angenommen haben. (Virdhagriswaran u. a. 1995; Flores-Mendez 1999; Pichler u. a. 2002) geben Überblicke über deren Aktivitäten. Besonders hervorzuheben sind darunter die *Knowledge Sharing Effort (KSE)* und die *Foundation for Intelligent Physical Agents (FIPA)*.

Knowledge Sharing Effort (KSE)

Die erste wesentliche Standardisierungsbemühung, die bis heute Einfluss auf die Agentenwelt hat, war die Knowledge Sharing Effort (KSE) (Neches u. a. 1991; Patil u. a. 1998). Dabei handelte es sich um einen Zusammenschluss von überwiegend universitären Projekten, die von den verschiedenen US-amerikanischen Fördereinrichtungen (DARPA, NRI, NSF,...) über Drittmittel finanziert wurden. Beginnend in den achtziger Jahren trug die KSE den Entwicklungen in den Bereichen wissensbasierte Systeme und verteilte Systeme Rechnung und entwickelte folgerichtig Standards für interoperable wissensbasierte Systeme, die man als Oberklasse der Agentensysteme ansehen kann.

Die Hauptergebnisse der KSE wurden in zwei Bereichen erzielt. Mit dem Knowledge Interchange Format (KIF) (Genesereth und Fikes 1992) wurde eine Sprache zur Wissensdarstellung entwickelt, mit dem vorrangigen Ziel, damit eine Zwischensprache

anzubieten für die heterogenen Wissensdarstellungen in den zu koppelnden Systemen. Die detailierte Sprachspezifikation bietet eine Erweiterung der Prädikatenlogik erster Ordnung mit einer formal definierten deklarativen Semantik und einer Präfix-Notation, die eine automatische Übersetzung aus und in andere Wissensdarstellungssprachen erlaubt.

Die größte Bedeutung der Ergebnisse der KSE genießt bis heute die Knowledge Query and Manipulation Language (KQML) (Labrou und Finin 1997). Sie definiert zugleich Nachrichtenformate und Übermittlungsprotokolle, um zur Laufzeit Wissen zwischen Systemen austauschen zu können. Zentral ist hierbei die Definition einer Reihe von Performativen, die den Zweck der Nachrichteninhalte bestimmen und diese damit erstmals um eine semi-formale Semantik anreichern.

Foundation for Intelligent Physical Agents (FIPA)

Die Foundation for Intelligent Physical Agents (FIPA)[3] ist die momentan bedeutendste Standardisierungsorganisation im Bereich Agententechnologie. Ihre Arbeiten haben in den zentralen Bereichen einen weitestgehend stabilen Zustand erreicht. Die Mitglieder der FIPA kommen aus dem universitären Bereich und aus Industrieunternehmen aus aller Welt; seit Juni 2005 ist die FIPA offizielles Standardisierungskomitee unter dem Dach der IEEE[4]. Die Bedeutung der FIPA erwächst durch den großen Einfluss auf zahlreiche Plattform-Implementierungen, die aufbauend auf den FIPA-Standards entwickelt wurden und immer noch werden.[5]

FIPA Abstract Architecture Während sich die Spezifikationen der KSE ganz überwiegend auf die Schnittstellen zwischen den Systemen konzentriert haben, ist der Fokus der FIPA breiter und so wurden auch Teile der internen Plattformstruktur von Agentensystemen und beispielhafte Anwendungen spezifiziert. Ausgangspunkt aller Spezifikationen ist dabei die FIPA Abstract Architecture (FIPA 2002a), die durch die FIPA Message

[3] Foundation for Intelligent Physical Agents (FIPA): `http://www.fipa.org`
[4] Institute of Electrical and Electronics Engineers, Inc. (IEEE): `http://www.ieee.org`
[5] Unter `http://www.fipa.org/subgroups/ROFS-SG-docs/History-of-FIPA.htm` stellt die FIPA ihre bewegte Geschichte ausführlich dar.

Transport Specification (FIPA 2004b) und die FIPA Agent Management Specification (FIPA 2004a) instanziiert wird. Auf letztere soll anhand von Abbildung 3.5 eingegangen werden.

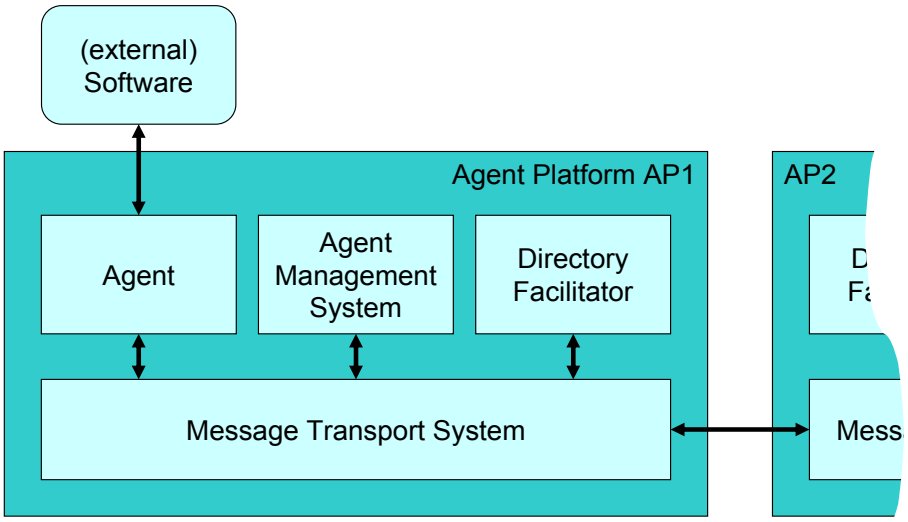


Abbildung 3.5: FIPA Agent Management Reference Model (FIPA 2004a)

Eine Agentenplattform nach FIPA besteht neben den Agenten, die die eigentliche Anwendungsfunktionalität erbringen, aus den Komponenten Message Transport System (MTS), Agent Management System (AMS) und Directory Facilitator (DF). Das MTS übernimmt den Nachrichtentransport innerhalb der Plattform und über Plattformgrenzen hinweg. Das AMS ist für die Verwaltung der Agenten auf einer Plattform verantwortlich. Zu den Aufgaben gehören das so genannte Lebenszyklusmanagement mit dem Erzeugen, Überwachen und Entfernen der Agenten der Plattform und das Mobilitätsmanagement bei Anwendungen, die mobile Agenten einsetzen. Der DF erlaubt die Vermittlung von Dienstangeboten und -gesuchen der Agenten und wird auch als Gelbe-Seiten-Dienst der Agentenplattform bezeichnet. Abbildung 3.5 zeigt außerdem, dass die Einbindung nicht-agentischer Software-Komponenten über FIPA-Agenten erfolgt, die als Adapter fungieren (vgl. *Wrapper-Entwurfsmuster*, Gamma u. a. 1996). Wie solche Agenten genau auszusehen haben, wird in (FIPA 2001b) beschrieben[6].

[6]Eine gute ausführlichere Einführung in die Spezifikationen der FIPA bieten Poslad und Charlton (2001).

Der FIPA-Protokollstapel für nachrichtenbasierte MAS Die Interoperabilität heterogener MAS als Hauptziel der FIPA-Aktivitäten erfordert sehr genaue Übereinkünfte an der Kommunikationsschnittstelle zwischen den beteiligten Agenten. Die entsprechenden Spezifikationen der FIPA können durch weitere Arbeiten ergänzt werden und bilden dann eine vollständige Standardisierung der Kommunikationsschnittstelle. Abbildung 3.6 illustriert das Zusammenspiel der Spezifikationen in einer für Kommunikationssysteme typischen Schichtenanordnung (siehe ISO/OSI-Schichtenmodell, Abeck u. a. 2003) mit nach oben wachsendem Abstraktionsgrad.

(KS8) Mechanismen	Soziale bzw. wirtschaftliche Koordination der Akteure	Einpendeln des Apfelpreises auf einem Wochenmarkt
(KS7) Konversationen	Abfolge von inhaltlich zusammenhängenden Sprechakten	Austausch über das Kaufen und Essen eines Apfels
(KS6) Sprechakte	Zweck des übermittelten Nachrichteninhaltes	Aufforderung an den Empfänger die Handlung "Einen Apfel essen!" auszuführen
(KS5) Ontologien	Beschreibung von Artefakten der Anwendungsumgebung	Bedeutung von "Apfel" und "essen"
(KS4) Inhaltssyntax	Darstellungsformat für Zustände und Handlungen in der Umgebung als Nachrichteninhalt	Zeichenkette "Einen Apfel essen!"
(KS3) Transportsyntax	Darstellungsformat für auszutauschende Gesamtnachrichten	HTML, JPG, SQL,...
(KS2) Transportprotokol	Datenaustauschprotokolle (ISO Schicht 7)	HTTP, IIOP, SMTP,...
(KS1) Transportkanal	Physikalischer Transport und niederschichtige Transportprotokolle (ISO Schichten 1-6)	Glasfaserkabel, TCP-IP,...

Schicht **Beschreibung** **Beispiel**

Abbildung 3.6: Ebenen der Agentenkommunikation (angepasst und erweitert nach Calisti 2002; Helin und Laukkanen 2002)

Transportkanal Zur Übermittlung der Nachrichten ist zunächst ein Transportkanal nötig, der meist durch die Internet-Protokolle bis einschließlich TCP/IP realisiert

wird und als „Black Box“ betrachtet werden kann, die die ISO/OSI-Schichten eins bis sechs umfasst (Abeck u. a. 2003).

Transportprotokoll Auf dem Transportkanal bauen die in ISO/OSI-Schicht sieben anzusiedelnden agentenspezifischen Transportprotokolle auf. In den Spezifikationen der FIPA verwenden diese Transportprotokolle die Transportdienste aus anderen bekannten Anwendungsbereichen: FIPA-HTTP (FIPA 2002h) verwendet das Protokoll des World Wide Web, FIPA-IIOP (FIPA 2002i) das Transportprotokoll der CORBA-Middleware für verteilte Objektsysteme und FIPA-WAP (FIPA 2001a) das Datentransportprotokoll WAP aus dem Mobilfunkbereich. Damit stehen drei Alternativen zur Verfügung, unter denen der Agentensystementwickler abhängig vom Einsatzszenarium seines Systems wählen kann.

Transportsyntax Damit die Nachrichten, die vom Sender mithilfe der Transportprotokolle geschickt werden, vom Empfänger auch wieder korrekt ausgelesen werden können, folgen diese einer vereinbarten Syntax. Bei der FIPA wird an dieser Stelle unterschieden zwischen der Umschlagssyntax, die auf der Nachrichtentransportebene Anwendung findet, und der Agentenkommunikationssprache (*Agent Communication Language (ACL)*), in der die Agenten die Nachrichten zu Gesicht bekommen.

Für verschiedene Einsatzszenarien werden auch hier wieder verschiedene Formate spezifiziert: Für die Umschläge gibt es eine XML[7]-Notation (FIPA 2002g) und eine besonders speichereffiziente Codierung (*FIPA Agent Message Transport Envelope Representation in Bit Efficient Specification*, siehe FIPA 2002f), die bei eingeschränkter Transportkapazität, z.B. im Mobilbereich, Anwendung findet.

Die Agentenkommunikationssprache der FIPA (FIPA-ACL, siehe FIPA 2002e) ist abgeleitet aus dem Syntax-Teil der oben bereits eingeführeten KQML. Die Spezifikation definiert abstrakt die Struktur der Nachrichten, d.h. die einzelnen Nachrichtenfelder und ihre Bedeutung. Drei weitere Spezifikationen definieren darauf aufbauend alternative Syntaxen, wiederum als XML-Dokument (FIPA 2002d), als speichereffiziente Bitfolge (FIPA 2002b) und zusätzlich als Zeichenkette (FIPA 2002c).

[7] Extendable Markup Language (siehe World Wide Web Consortium (W3C) 2006)

Inhaltssyntax Für die eigentlichen Nachrichteninhalte bedarf es wiederum einer vereinbarten Syntax, da Ontologien von dieser abstrahieren. Die verschiedenen alternativen Inhaltssprachen der FIPA orientieren sich an gängigen Sprachen zur Wissensrepräsentation. Von den Sprachen FIPA-RDF (FIPA 2001g), FIPA-CCL (FIPA 2001c), FIPA-KIF (FIPA 2003) und FIPA-SL (FIPA 2002n) sind vor allem die beiden letzteren wegen ihrer vergleichweise weiten Verbreitung erwähnenswert. Diese rührt bei FIPA-KIF aus der frühen Verfügbarkeit von Werkzeugen aus der KSE her. Bei FIPA-SL dürfte die weite Verbreitung auf dem FIPA-internen Einsatz beruhen, z.B. in den Spezifikationen zur Formalisierung der Sprechaktsemantik. Ein neuerer Vertreter der Inhaltssprachen hat seinen Ursprung außerhalb der FIPA in den Forschungsbemühungen im Bereich des Semantic Web. Dort wird für die Beschreibung inhaltlicher Zusammenhänge häufig die Sprache OWL[8], eine Weiterentwicklung von DAML+OIL[9], eingesetzt.

Ontologien Neben den verschiedenen Syntaxen der Nachrichten ist für eine erfolgreiche Koordination von Agenten auch ein gemeinsames semantisches Verständnis der Nachrichteninhalte und ihrer Einbettung in die Anwendungswelt von Nöten. Dies wird im Allgemeinen durch Ontologien zur Wissensrepräsentation innerhalb der Agenten und bei deren Nachrichtenaustausch bewerkstelligt, d.h. durch formal definierte Systeme von Konzepten und Relationen zwischen ihnen. Bei Verwendung unterschiedlicher Ontologien auf Sender- und Empfängerseite können Dienste zur Übersetzung zwischen den Konzepten herangezogen werden. Hierfür wurde der Ontology Service (FIPA 2001f) definiert, der u.a. Funktionen zur Aushandlung einer Nachrichtenontologie, zur Definition von Termen und zur Übersetzung zwischen ihnen anbietet.

Sprechakte Mit einer Nachricht zwischen Agenten ist immer ein Zweck verbunden, d.h. man unterstellt dem Sender der Nachricht eine bestimmte Absicht, die er mit dem Senden verfolgt. Beispiele hierfür sind das Informieren des Konversationspartners über den im Nachrichteninhalt beschriebenen Sachverhalt oder dessen Beauftragung mit der dort beschriebenen Aufgabe. Neben der Syntax und der Semantik muss also eine Kennzeichnung der Nachrichten mit der zugehöri-

[8] Web Ontology Language (siehe World Wide Web Consortium (W3C) 2004)
[9] DARPA Agent Markup Language + Ontology Inference Layer (siehe Horrocks 2002)

gen Pragmatik – dem Zweck – möglich sein. Diese erfolgt mit den so genannten Sprechakten (engl. *Speech Acts* oder in FIPA-Terminologie *Communicative Acts*), wie z.B. `inform` und `request`. Die Bezeichnung „Sprechakt“ deutet darauf hin, dass die Äußerung einer Nachricht als Aktion eines Agenten aufgefasst wird. Die formale Definition der Bedeutung der Sprechakte erfolgt in der Communicative Act Library Specification (FIPA 2002j).

Konversationen Die Nachrichten zwischen den Agenten werden in der Regel nicht alleinstehend betrachtet, sondern sie werden über so genannte Protokolle inhaltlich zusammenhängenden Konversationen zugeordnet. Die FIPA definiert z.B. Protokolle zur marktartigen Vergabe von Aufträgen nach dem Kontraktnetzprotokoll (FIPA 2002k) (s.o.) und für verschiedene Auktionsarten (FIPA 2001e, d). Ein Protokoll beschreibt dabei lediglich auf syntaktischer Ebene zulässige Abfolgen von Nachrichten.

Mechanismen Die zulässigen Nachrichtenabfolgen der Konversationsprotokolle orientieren sich an den so genannten (Koordinations-)Mechanismen. Sie blicken vom Standpunkt der Wirtschafts- und/oder Sozialwissenschaften auf die Konversationen und liegen damit nicht mehr im Fokus der technisch orientierten FIPA. Die Mechanismen beschreiben z.B., unter welchen Randbedingungen Verhandlungen zu einem gemeinsamen Vertrag führen können oder wie die Preisfindung auf einem Markt erfolgt. Typische Mechanismen aus den Wirtschaftswissenschaften betrachten z.B. die bereits erwähnten Auktionen und Kontraktnetze (MacKie-Mason und Wellman 2006). Für die Sozialwissenschaften seien exemplarisch die Mechanismen aus der Organisationstheorie (Kolp u. a. 2006; Giorgini u. a. 2002) und insbesondere die Mechanismen zur Selbstorganisation in (Insekten-)Schwärmen genannt (Bonabeau u. a. 1999).

3.2.3 Verwandte Arbeiten: Referenzarchitekturen

In der einschlägigen Literatur sind nur wenige ernsthafte Forschungsvorhaben auszumachen, die als ein explizites Ziel die Erstellung einer Referenzarchitektur für Multiagentensysteme angeben. Dennoch bilden deren Ergebnisse für die zu entwickelnde Referenz-

architektur der vorliegenden Arbeit zugleich die verwandten Arbeiten und einen Teil des Fundaments.

Agent-Based System Reference Model (ABSRM)

Modi, Mancoridis, Mongan, Regli und Mayk (2006) versuchen seit kürzerer Zeit ein Referenzmodell zu entwickeln, das sich gleichermaßen an Entwickler und Anwender richten soll: das *Agent-Based System Reference Model (ABSRM)*. Mithilfe dieses Modells soll eine Vereinheitlichung von Terminologien und Konzepten für Multiagentensysteme erreicht werden, da in der diesbezüglich vorliegenden Unschärfe ein wesentliches Hindernis gegen den kommerziellen und militärischen Einsatz der Agententechnologie ausgemacht wurde. Insbesondere soll es durch das ABSRM möglich werden, existierende und zukünftige MAS-Rahmenwerke zu vergleichen und gegeneinander abzusetzen, und Gebiete zu identifizieren, die einer weiteren Standardisierung bedürfen.

Das ABSRM resultiert aus einer statischen Code-Analyse einiger existierender MAS-Rahmenwerke (Cougaar (Helsinger und Wright 2005), JADE (Bellifemine u. a. 2007) und RETSINA (Sycara u. a. 2003)) und einer Betrachtung deren Dokumentation. Dabei schwingt die Hoffnung mit, dass die untersuchten Rahmenwerke heterogen genug sind, um eine genügend breite Basis für eine Referenzarchitektur zu bilden. Die bisherigen Ergebnisse sind eine einfache geschichtete Architektur für Agentensysteme, die in Abbildung 3.7 dargestellt ist, und eine textuelle Beschreibung wiederkehrender Funktionseinheiten.

Die Architektur sieht vier Schichten für die Umgebung des Agenten („Agent Environment“) vor: die physische Welt („Physical World“), die Rechner-Schicht („Hosts“), die Plattform-Schicht („Platforms“) und die Schicht der Agenten-Rahmenwerke („Agent Frameworks“). Zwischen diesen Schichten kann jeweils eine n:1-Beziehung bestehen, d.h. es können beispielsweise mehrere Plattformen auf einem Rechner beheimatet sein. Plattformen und Rechner werden zusammen auch als Infrastruktur („Infrastructure“) bezeichnet. Für die Umgebung wird also im wesentlichen die Gestalt des Systems zur Laufzeit beschrieben (engl. *Deployment*). Die Agenten, die mithilfe eines Agenten-Rahmenwerks implementiert werden, bilden eine eigene Schicht. Über die innere Struktur der Agenten

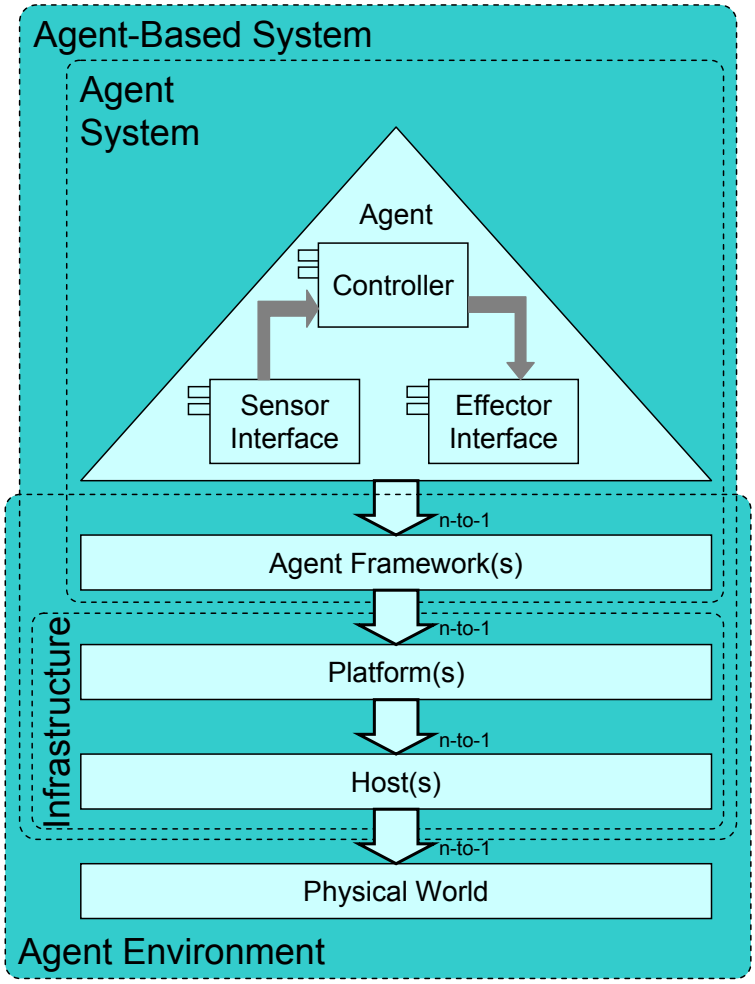


Abbildung 3.7: Geschichtete Architektur für Agentensysteme nach ABSRM

werden bewusst keine Annahmen getroffen, außer dass sie über Sensoren und Effektoren verfügen:

> „Given the vast array of tasks envisioned for agent systems, it is not the role of a reference model to limit or define what an agent is."

Stattdessen wird eine Reihe von Funktionseinheiten (*„Functional Concepts"* in Vermeidung des Begriffes *„components"*) definiert, die aber wiederum bewusst nicht genauer verortet werden, um das Referenzmodell nicht zu restriktriktiv bezüglich der möglichen Implementierungen der Funktionalität werden zu lassen. Identifiziert wurden die Funktionen Administration, Sicherheit („Security"), Konfliktverwaltung zur Koordination der Agenten eines MAS („Conflict Management") und Nachrichtenübermittlung („Messaging"). Die dünnbesetzte Architektur und die knappe textuelle Beschreibung der Funktionalitäten führt dazu, dass das ABSRM insgesamt noch einen recht rudimentären und vorläufigen Eindruck erweckt.

Reference Architecture for Situated Multiagent Systems (RASMAS)

Mit der *Reference Architecture for Situated Multiagent Systems* verfolgen Weyns, Holvoet und Schelfthout (2006) das Ziel, Multiagentensysteme nicht mehr als radikal neuen Ansatz, sondern als eine Option unter vielen in der Software-Entwicklung zu betrachten. Damit soll die Agententechnologie zugänglich werden für die gewohnten Methoden aus der Software-Technik. Die Basis für die Referenzarchitektur bilden eine Reihe von selbstentwickelten Multiagentensystemen, aus denen gemeinsame Funktionalitäten und Strukturen extrahiert und generalisiert wurden. Abbildung 3.8 gibt einen Überblick über die Referenzarchitektur, die als Blaupause für die Erstellung künftiger Systeme dienen soll.

Auffällig an der Reference Architecture for Situated Multiagent Systems, die im folgenden mit *RASMAS* abgekürzt werden soll, ist die Trennung in die beiden Hauptartefakte („Primary Abstractions") Agent und Umgebung, die analog zum ABSRM auch hier als aufeinander aufbauend aufgefasst werden könnten, obwohl dies von den Autoren nicht explizit gemacht wird. Die interne Struktur, die die RASMAS für einen Agenten vorsieht, ähnelt der abstrakten Agentenarchitektur nach Wooldridge (2002) (siehe Abbildung 3.3).

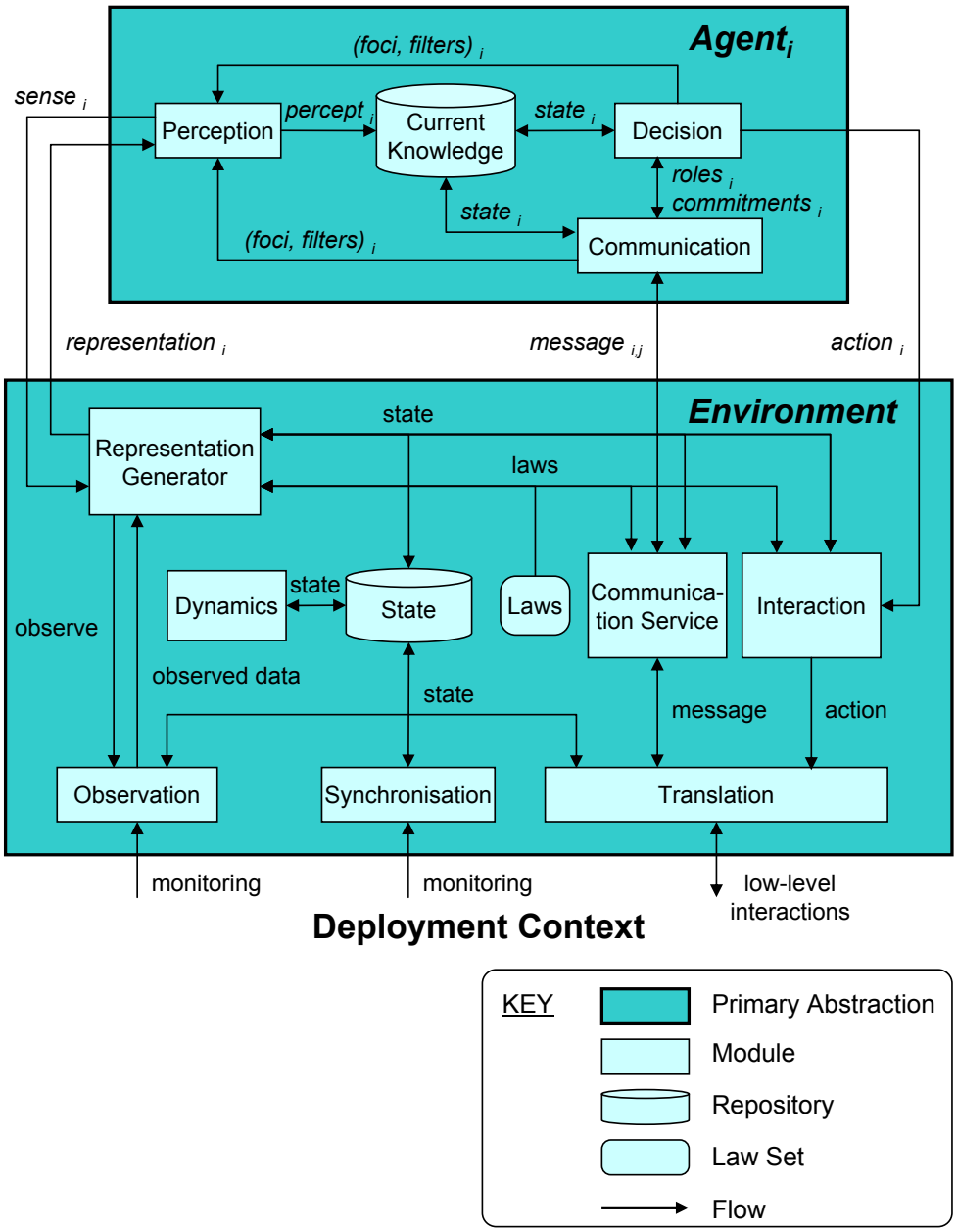


Abbildung 3.8: Überblick über die Reference Architecture for Situated Multiagent Systems (RASMAS)

Darin gibt es ein Modul *Perception*, das für die Wahrnehmung der Umgebung und die Aktualisierung der Wissensbasis („Current Knowledge“) zuständig ist, und ein Modul *Decision*, das die auszuführende Aktion auswählt und anstößt – analog zu den Funktionen *see*, *next* und *act* von Wooldridge. Die Kommunikation nimmt in der RASMAS eine hervorgehobene Rolle unter den Aktionen ein und wird durch das Modul *Communication* gehandhabt. Zu den Einzelmodulen Perception, Decision und Communication sind weitere Veröffentlichungen verfügbar, die jeweils deren konkrete Ausprägungen innerhalb der Referenzarchitektur beschreiben (Weyns u. a. 2004b; Steegmans u. a. 2004; Weyns u. a. 2004a).

Eine Besonderheit der RASMAS ist die detaillierte Ausgestaltung der Umgebung mit zahlreichen Modulen und ihren Beziehungen. (Auch der Umgebung sind eigene konkretisierende Veröffentlichungen gewidmet, wie z.B. (Weyns u. a. 2007).) Zur Umgebung in RASMAS zählt nicht nur der Zustand der Anwendungswelt (abgelegt im „State“-Repository), sondern auch die Schnittstellen zum sog. *Deployment Context*, der die Sensoren und Aktoren zur Interaktion mit der realen Anwendungswelt, die benutzte Hardware und Software und die weiteren externen Ressourcen umfasst. Das Environment von RASMAS bildet also eine virtuelle Zugriffsschicht auf die Umgebung. Die Bezeichnungen sind also etwas irreführend gewählt: das *Environment* von RASMAS würde in der allgemeinen Literatur eher als Infrastruktur bezeichnet werden und der *Deployment Context* als Umgebung.

Insgesamt machen die Arbeiten an der RASMAS einen fortgeschrittenen Eindruck, obwohl beim Studium der Modul-Beschreibungen deutlich wird, dass diese zunächst eigenständig verfolgt wurden und erst dann die Referenzarchitektur als inhaltliche Klammer übergestülpt wurde. Die ungewöhnlich detaillierte Ausarbeitung der Umgebung ist in der wissenschaftlichen Ausrichtung der betreibenden Forschungsgruppe begründet und geht auf Kosten der Ausgestaltung der Agentenstruktur.

3.3 Referenzarchitektur für Multiagentensysteme

3.3.1 Grundstruktur

Die Grundstrukturierung komplexer Software-Systeme folgt häufig sogenannten Architekturmustern (engl. *Architectural Pattern* (nach Buschmann u. a. 1996) oder synonym *Architectural Style* (nach Shaw und Garlan 1996)), in denen erfolgreich eingesetzte, wiederkehrende Organisationsmuster für Software beschrieben werden. Meist zerlegen sie ein komplexes Gesamtsystem in einfachere abstrakte Subsysteme (engl. *Components*), beschreiben deren Aufgaben und geben Richtlinien für die Art und Organisationsstruktur derer Verknüpfungen (engl. *Connectors*) (Posch u. a. 2007).

Tabelle 3.1 zeigt zwei häufig zitierte Listen verbreiteter Architekturmuster[10] und die jeweiligen Kategorisierungen.

Bei den beiden Aufzählungen fallen die unterschiedlichen Kategorisierungen und die geringe Überdeckung zwischen den Mustern ins Auge. Eine Betrachtung der einzelnen Muster zeigt, dass genau diejenigen Muster, die in beiden Sammlungen vorkommen, für die Aufgabe der Grundstrukturierung besonders gut geeignet sind: *(Hierarchical) Layers*, *Pipes and Filters* und *Blackboard*. Dies wird auch von der Kategorisierung „From Mud to Structure“ in (Buschmann u. a. 1996) unterstrichen.

Die anderen Architekturmuster sind zu generisch um nützlich zu sein (z.B. OO Systems) oder sind für die Grundstrukturierung des Gesamtsystems zu spezifisch bzw. feingranular (z.B. Communicating Processes, Databases,...). Solche spezifischen Architekturmuster werden später bei der Ausgestaltung von Subsystemen der Gesamtarchitektur und bei einer möglichen Instanziierung der Referenzarchitektur eine Rolle spielen.

[10] Architekturmuster tragen in der Regel eindeutige englische Bezeichnungen. Soweit es geläufige deutsche Bezeichnungen gibt, werden diese in den ausführlichen Beschreibungen genannt und im weiteren Verlauf verwendet.

Shaw und Garlan (1996)	Buschmann u. a. (1996)
Dataflow Systems	**From Mud to Structure**
Batch Sequential	Layers
Pipes and Filters	Pipes and Filters
Call-and-Return Systems	Blackboard
Main Program and Subroutine	**Distributed Systems**
OO Systems	Broker
Hierarchical Layers	**Interactive Systems**
Independent Components	Model-View-Controller
Communicating Processes	Presentation-Abstraction-Control
Event Systems	**Adaptable Systems**
Virtual Machines	Microkernel
Interpreters	Reflection
Rule-based Systems	
Data-centered Systems (Repositories)	
Databases	
Hypertext Systems	
Blackboards	

Tabelle 3.1: Kategorisierungen von Architekturmustern

Potenziell geeignete Architekturmuster

Im folgenden werden die drei grundlegenden Architekturmuster Layers (Schichtung), Pipes and Filters und Blackboard (Schwarzes Brett) beschrieben, die für eine Grundstrukturierung der Referenzarchitektur in Frage kommen. Neben der Beschreibung der Muster durch ihre Komponenten und Konnektoren werden die Indikationen für ihren Einsatz und die dabei möglichen positiven Auswirkungen auf die resultierenden Systeme besprochen.

Schichtung Bei der Schichtung sind die Komponenten („Schichten“) hierarchisch organisiert. Eine jede Schicht n bietet dabei Dienste auf einem bestimmten Abstraktionsniveau an, die von der nächst abstrakteren Schicht n+1 in Anspruch genommen werden, und sie bedient sich dazu selbst wiederum der Dienste der nächst detaillierteren Schicht n–1. Die Dienstgeber-/Dienstnehmer-Protokolle zwischen den Schichten

können als Konnektoren des Schichtung-Architekturmusters aufgefasst werden (siehe Abbildung 3.9 (a)).

Schichtung gilt als probates Mittel, wenn komplexe Anwendungsfunktionalität auf einfache Grunddienste herunter gebrochen werden soll. Buschmann u. a. (1996) formulieren dies wie folgt:

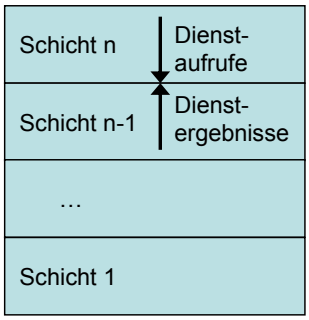
> „The *Layers* architectural pattern helps to structure applications that can be decomposed into groups of subtasks in which each group of subtasks is at a particular level of abstraction."

Die Vorteile der Schichtung liegen also in der Möglichkeit, komplexe Anwendungen in Teilprobleme auf verschiedenen Abstraktionsniveaus zu zerlegen und die Teillösungen unabhängig voneinander zu entwickeln. Da Konnektoren nur zwischen benachbarten Schichten bestehen, hält sich der Anpassungsaufwand bei Änderungen in Grenzen. Durch die Definition abstrakter Konnektoren (siehe Fassade-Entwurfsmuster (Gamma u. a. 1996)) lässt sich auch eine leichte Austauschbarkeit der Implementierung ganzer Schichten erreichen. Diese Vorgehensweise wird häufig bei der Entwicklung von Standard-konformen Implementierungen ausgenutzt, wie z.B. beim ISO/OSI-Standard für Kommunikationssysteme (Abeck u. a. 2003).

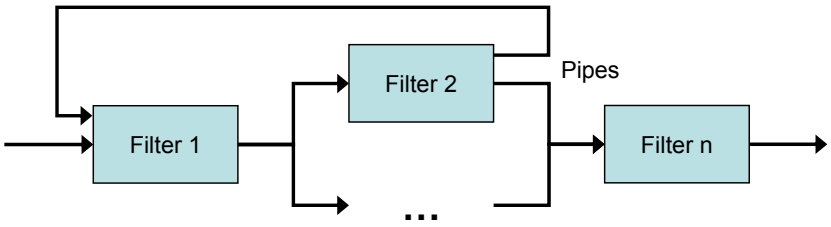
In der geschilderten strengen Form sind bei der Schichtung nur Konnektoren einer dienstnehmenden Schicht n zur nächsttieferen dienstgebenden Schicht n–1 erlaubt. Bei der Sonderform der *Relaxed Layered Systems* sind auch Konnektoren zwischen einer Schicht n und beliebigen Schichten n–m zugelassen. Die dadurch mögliche Steigerung der Performanz geht allerdings auf Kosten der oben genannten Vorzüge der Modularität.

Eine weitere Sonderform der Schichtung kommt zum Einsatz, wenn auf einem bestimmten Abstraktionsniveau – also innerhalb einer Schicht – mehrere verschiedene Dienste angeboten werden, deren Realisierung voneinander unabhängig ist. In diesem Fall zerfallen einzelne Schichten in sogenannte *Säulen*.

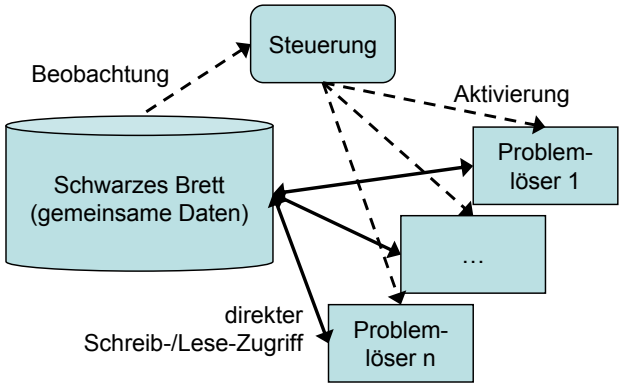
Pipes and Filters Im Pipes und Filters-Architekturmuster verknüpfen sogenannte „Pipes" als Konnektoren die als „Filters" bezeichneten Komponenten, deren Aufgabe die



(a) Schichtung



(b) Pipes and Filters



(c) Schwarzes Brett

Abbildung 3.9: Architekturmuster: Schichtung, Pipes and Filters und Schwarzes Brett

Transformation der von den Konnektoren angelieferten Datenströme ist. Dabei gelten nach Buschmann u. a. (1996) folgenden Eigenschaften:

> „The *Pipes and Filters* architectural pattern provides a structure for systems that process a stream of data. Each processing step is encapsulated in a filter component. Data is passed through pipes between adjacent filters. Recombinig filters allows to build families of related systems."

Das Pipes and Filters-Muster ist also prädestiniert für Systeme, bei denen Eingangsdaten in Form kontinuierlicher Datenströme vorliegen, wie z.B. bei Multimediaanwendungen oder bei kontinuierlicher Beobachtung eines Realwelt-Prozesses durch Messsensoren, und bei denen zugleich die Verarbeitung der Eingangsdaten in unabhängige, aufeinanderfolgende Teilschritte zerlegt werden kann.

Die Motivation für den „Filter"-Begriff kommt aus der Nachrichtentechnik und deutet an, dass in bestimmten Fällen die Transformation in den Komponenten lokal ist, d.h. dass der Beginn der Ausgabe des Ausgangsdatenstroms erfolgen kann, bevor das Ende des Eingangsdatenstroms erreicht wurde. Im Falle von ausschließlich blockierenden Filtern, die das Ende des Eingansdatenstroms vor Beginn der Ausgabe gelesen haben müssen, degeneriert das Pipes and Filters-Muster zum sogenannten Batch Sequential-Muster (dt. *Stapelverarbeitung*).

Wie Abbildung 3.9 (b) zeigt, können die Filter mehrere Eingangsdatenströme haben, die überführt werden können in eine Menge von Ausgangsdatenströmen. Somit können auch Verzweigungen, Zusammenführungen und Zyklen auftreten. Den häufig auftretenden linearen Sonderfall bezeichnet man als Pipeline-Architekturmuster. Beispiele für diesen Sonderfall sind viele Compiler, die Dateiverwaltung der Programmiersprache Java und die Verkettung von Shell-Kommandos in Unix-Betriebssystemen.

Das Pipes and Filter-Muster hat ähnliche positive Eigenschaften wie die Schichtung: Es erlaubt die Zerlegung einer komplexen Aufgabe in Teilaufgaben, allerdings müssen diese nicht unbedingt auf unterschiedlichen Abstraktionsniveaus angesiedelt sein. Da die einzelnen Filter untereinander unabhängig sind, d.h. keinen gemeinsamen Zustand besitzen dürfen, lassen sie sich einfach wiederverwenden und zu neuen Anwendungen arrangieren. Ähnlich lässt sich für die einfache Modifikation und Erweiterung von Pipes and

Filters-Systemen argumentieren. Ein Vorteil, der für die Schichtung nicht ohne weiteres gilt, ist die natürliche Unterstützung der Parallelität durch nebenläufige Ausführung der Filter.

Schwarzes Brett Hintergrund des Schwarzes Brett-Architekturmusters ist die Idee, dass ein Problem, das für einen einzelnen Allrounder zu schwierig ist, evtl. von einem Team von Experten kooperativ gelöst werden könnte. Buschmann u. a. (1996) umschreibt dies so:

> „The *Blackboard* architectural pattern is useful for problems for which no deterministic solution strategies are known. In Blackboard several specialized subsystems assemble their knowledge to build a possibly partial or approximate solution."

Das Schwarzes Brett-Architekturmuster wurde ursprünglich in der Künstlichen Intelligenz entwickelt und findet dort insbesondere in der Muster- und Spracherkennung Anwendung. Es lässt sich am vorteilhaftesten für Anwendungsprobleme einsetzen, bei denen nicht a priori ein deterministischer Lösungsweg bekannt ist, weil z.B. die Anwendungsdomäne nicht ausreichend beherrscht wird, bei denen aber Lösungsverfahren für Teilprobleme verfügbar sind.

Beim Schwarzen Brett, das in Abbildung 3.9 (c) gezeigt wird, lassen sich drei Arten von Komponenten unterscheiden. Zunächst ist die namengebende Datenablage – das *schwarze Brett* (engl. *Blackboard Data Structure*) – zu nennen, auf der zentral die Beschreibung des zu lösenden Problems abgelegt wird.

Auf diese Ablage können die sogenannten *Problemlöser* (engl. *Knowledge Sources*) lesend und schreibend zugreifen, d.h. sie können sich die Beschreibung der Aufgabenstellung abholen, ihren Teil des Lösungsvorgangs durchführen und das neue Zwischenergebnis wieder in die Ablage zurück schreiben. Die Flexibilität des Musters rührt aus der Möglichkeit her, mehrere alternative Zwischenergebnisse abzulegen und die ungeeigneten Hypothesen später zu verwerfen.

Für die Aktivierung der Problemlöser ist die dritte Komponentenart – die zentrale *Steuerung* (engl. *Control*) – zuständig, die permanent die Veränderungen der Zwischenlösun-

gen auf dem schwarzen Brett beobachtet und dann abhängig von den jeweiligen Fähigkeiten die in Frage kommenden Problemlöser anstößt.

Hauptvorteil des Schwarzen Brettes ist seine Fähigkeit, Lösungen zu finden, die mit deterministischen Verfahren nicht erreicht werden könnten. Durch die Isolierung der Teilprozesse in den Problemlösern wird zugleich eine gute Wartbarkeit, Modifizierbarkeit und Wiederverwendungsfähigkeit erreicht, wobei jedoch die Kontrollkomponente entsprechend anzupassen ist.

Auswahl eines geeigneten Architekturmusters

Unter den vorgestellten Architekturmustern Schichtung, Pipes and Filters und Schwarzes Brett, die prinzipiell als Rahmen für die Grundstruktur der Referenzarchitektur geeignet sind, gilt es nun eine begründete Auswahl zu treffen. Dabei sind jeweils drei Sichten zu unterscheiden: die Anwendung der Architekturmuster ist prinzipiell auf die Multiagentensystemarchitektur, auf die Systemarchitektur für einzelne Agenten oder auf die innere Agentenarchitektur möglich. Ein geeignetes Architekturmuster sollte – im Sinne einer homogenen Referenzarchitektur – zumindest die Multiagentensystemarchitektur und die Systemarchitektur für einzelne Agenten gut bedienen können, da diese später die Referenzarchitektur dominieren. Im günstigsten Fall sollte auch die innere Agentenarchitektur gleich gut unterstützt werden.

Schwarzes Brett Das Schwarze Brett eignet sich – genau wie Multiagentensysteme selbst – zur Lösung komplexer Anwendungsprobleme, für die kein deterministischer Lösungsweg vorab bekannt ist. Bei einer Anwendung des Musters als Multiagentensystemarchitektur könnte die zentrale Datenverwaltung durch das Schwarze Brett die örtliche Verteilung der Agenten unterstützen, die dann die Rolle der verschiedenen Problemlöser übernehmen würden. Allerdings ist gerade die Zentralisierung der Hauptnachteil dieses Architekturmusters: die Aktivierung der verschiedenen Agenten würde nämlich durch eine – zumindest konzeptionell – zentrale Steuerung erfolgen. Dieses Vorgehen läuft der Autonomie der Agenten zuwider und unterstellt das Vorhandensein eines einzigen gemeinsamen Ziels aller Agenten, wovon im allgemeinen Fall nicht ausgegangen werden kann.

Die Anwendung des Schwarzen Brettes als Systemarchitektur für einzelne Agenten impliziert, dass ein Agent verschiedene isolierte Teilfunktionalitäten besitzt, die von einer zentralen Komponente angesteuert werden. Z.B. würde die Steuerung für eine auf dem Schwarzen Brett eingegangene Nachricht eines anderen Agenten zunächst eine Parser-Funktion aufrufen, die dann den ausgepackten Inhalt wieder auf das Brett zurück schreibt. Daraufhin würde eine Inhaltsverarbeitungsfunktion angesteuert werden, die versucht, den mitgeteilten Nachrichteninhalt in das Weltmodell zu integrieren. Dieses Vorgehen erscheint zwar auf den ersten Blick nicht sehr elegant, ist aus Softwaretechnischer Sicht aber durchaus denkbar und kann der Erstellung modularer Agenten sogar zuträglich sein.

Innere Agentenarchitekturen sind verantwortlich für die Verhaltensgenerierung, d.h. für die eigentliche Lösung des Anwendungsproblems. Übertrüge man das Schwarze Brett auf die innere Architektur, dann müsste ein Agent verschiedene Problemlösungsstrategien besitzen und die jeweils am besten geeignete Strategie müsste von einer Steuerung mit der Problemlösung beauftragt werden. Obwohl dieser Ansatz für bestimmte Anwendungen sinnvoll sein könnte, widerspricht er der Allgemeingültigkeit, die eine Referenzarchitektur erfordert: das Gesamtverhalten eines Agenten müsste explizit in eine Steuerung und mehrere isolierte Teilverhalten untergliedert werden. Dies ist z.B. bei regelbasierten Agenten nicht der Fall, bei denen man im Allgemeinen das Gesamtverhalten als eine einzelne Regelbasis auffasst und versucht aus den lokal verfügbaren Fakten unter Anwendung einer beliebigen Kombination der Regeln eine Problemlösung abzuleiten. Das Schwarze Brett ist insgesamt also eher ungeeignet, da es lediglich für eine Referenzarchitektur für einzelne Agenten in Frage kommt und die beiden anderen Architekturausprägungen nicht gut unterstützt.

Pipes and Filters Die Betrachtung des Pipes and Filters-Muster als Architekturmuster für einzelne Agenten und für innere Agentenarchitekturen basiert auf derselben Grundüberlegung: Die Einbettung der Agenten in eine Anwendungsumgebung erfolgt auf physikalischer Ebene durch Sensoren und Aktoren. Werden die Sensoren kontinuierlich überwacht, dann ergibt sich zusammen mit den empfangenen Nachrichten ein kontinuierlicher Datenstrom, wie er vom Pipes and Filters-Architekturmuster als Eingabe verwendet werden könnte. Der Eingabedatenstrom würde dann von den verketteten Filtern

in einen kontinuierlichen Ausgabedatenstrom überführt werden.

Ein definierendes Merkmal des Musters ist die Abwesenheit eines gemeinsamen Zustands, d.h. die verschiedenen Filter können – abgesehen vom Datenstrom – keine Informationen austauschen. Die Konsequenzen, die ein bestimmter Messwert im Strom nach sich ziehen kann, können aber für einen einzelnen Agenten sehr vielschichtig sein: die Veränderung der Umgebung kann zur Anpassung des lokalen Weltmodells führen, zur Neuauswahl der verfolgten Ziele, zum Stellen eines Aktors in der Umgebung oder zur Koordination mit anderen Agenten. Wollte man diesen Aufgaben jeweils einen eigenen Filter zuweisen, dann müsste jeder Filter – mangels eines gemeinsam nutzbaren Zustands – seinen eigenen relevanten Ausschnitt des Weltmodells selbst verwalten oder ein umfassendes Weltmodell müsste nach jeder Messung in den Datenstrom eincodiert und durch alle Filter geschleust werden.

Beide Vorgehensweisen würden zu sehr komplizierten und konstruiert anmutenden Agentenarchitekturen führen, sowohl bezogen auf komplette Einzelagenten als auch im Inneren bezogen auf deren Verhaltensgenerierung. Damit ist das Pipes and Filters-Muster nicht auf natürliche Weise für einzelne Agenten oder innere Architekturen geeignet. Und auch als Systemarchitektur für Multiagentensysteme scheidet das Pipes and Filters-Architekturmuster von vorneherein aus, da es keine Autonomie der Filter vorsieht, sondern auf der sklavischen Unterordnung eines jeden Filters gegenüber den jeweils vorgeschalteten Filtern beruht. Eine Umsetzung eines Multiagentensystems als eine Verkettung einzelner Filter, die jeweils einen Agenten verkörpern, steht damit in direktem Widerspruch zur Autonomieeigenschaft E3 und das Pipes and Filters-Architekturmuster scheidet für die Anwendung in der Referenzarchitektur aus.

Schichtung Laut Definition soll das Architekturmuster Schichtung Anwendung finden, wenn ein System auf natürliche Weise in Teilaufgaben auf unterschiedlichen Abstraktionsniveaus zerfällt und sich die Aufgaben auf hohem Abstraktionsniveau abbilden lassen auf konkretere Aufgaben. Dies ist sowohl bei Multiagentensystemen als auch bei einzelnen Agenten und im Agenteninneren mehrfach der Fall:

- aus Sicht des Multiagentensystems müssen die abstrakten globalen Ziele auf agentenlokale konkretere Ziele abgebildet werden,

- die Koordination der abstrakten lokalen Ziele der beteiligten Agenten, erfolgt z.B. durch Umsetzung und Übermittlung mittels ganz konkreter Bit-Ströme in Kommunikationsnetzwerken, und

- innerhalb des einzelnen Agenten erfolgt die Umsetzung lokaler Ziele durch abstrakte Handlungen, die auf konkrete Einstellungen von Aktoren in der Umgebung abgebildet werden.

Nicht überraschend findet daher das Schichtenprinzip in der Agententechnologie häufig Anwendung. So weisen z.B. die verwandten Referenzarchitekturen ABSRM und RASMAS aus Abschnitt 3.2.3 ebenso eine Schichtenarchitektur auf, wie auch einige Systemarchitekturen für einzelne Agenten (z.B. Kendall u. a. 1997; Müller 1996) und innere Agentenarchitekturen (z.B. Brooks 1986, 1985).

Konsequenterweise soll das *Schichtung-Architekturmuster als Basis für die umfassende und allgemeingültige Referenzarchitektur* in dieser Arbeit dienen. Die Instanziierung des Architekturmusters und damit die Definition der Grundstruktur der Referenzarchitektur wird im nächsten Abschnitt durchgeführt.

Instanziierung des Schichtung-Architekturmusters zur Grundstruktur

Nach der Arbeitsdefinition aus Kapitel 2, handelt es sich bei Multiagentensystemen um „*verteilte Systeme* aus interagierenden *autonomen Software-Komponenten* (Agenten), die in ihre *Umgebung eingebettet sind* [...]“. Diese Definition fasst eine Auswahl der definierenden Agenteneigenschaften (E1, E2, E3 und E8) zusammen und soll nun als Ausgangsbasis für die Instanziierung des Schichtung-Architekturmusters dienen, die im folgenden durchgeführt wird.

Die Einbettung der Agenten erfordert die Abbildung und Erschließung aller relevanten Aspekte der Umgebung. Dies erfolgt in der geschichteten Referenzarchitektur durch eine eigene Schicht (S1). Sie ist zugleich die unterste Schicht, da aus Sicht der Agenten keine weiteren sinnvollen konkreteren Schichten vorstellbar sind. Die *Basisdienste*, die von dieser Schicht zum Umgang mit der Umgebung angeboten werden, sind generischer

Natur, d.h. sie sind nicht speziell auf die Diensterbringung für Agenten angepasst, sondern könnten auch als Grundlage für beliebige andere umgebungseingebettete Systeme dienen.

Die Aufbereitung der Dienste für die Agenten erfolgt in Form einer Schicht (S2) *agentenspezifischer Infrastrukturdienste*. Diese abstrahieren von den technischen Details der Schicht S1 und verwenden stattdessen Konzeptualisierungen, die der Agentenwelt zuzuordnen sind.

Für die beiden bis jetzt identifizierten Schichten S1 und S2 gilt eine Besonderheit, die bei der Beschreibung des Schichtung-Architekturmusters angesprochen wurde: sie zerfallen jeweils in Gruppen von Diensten, deren Erbringung unabhängig voneinander erfolgen kann. Nach der Arbeitsdefinition weisen Multiagentensysteme die Charakteristika dreier unabhängiger Systemtypen auf; Multiagentensysteme sind zugleich eingebettete Systeme, Komponenten-Software-Systeme und verteilte Systeme. Da die generischen und agentenspezifischen Dienste für die *Realwelteinbettung* (a), für die *Komponentenverwaltung* (b) und für die *Verteilungsunterstützung* (c) weitgehend unabhängig erbracht werden, bilden sich durch die beiden unteren Schichten hindurch drei entsprechende durchgehende Säulen heraus.

Die Informationen und Dienste der drei Säulen in den Schichten S1 und S2 bilden die Einflussfaktoren für das Weltmodell des Agenten, also für die symbolische Darstellung der Gesamtumgebung, auf der der Agent operiert (ähnlich der Datenbank-Schicht in einer klassischen Zweischichtenarchitektur für Client-Server-Systeme). Durch eine Ontologie-Unterstützung in dieser Schicht gelingt es, die Einflussfaktoren in Schicht S3 auch semantisch zusammen zu führen. Das *Ontologie-basierte Weltmodell* erlaubt somit einen integrierten Blick auf die Anwendungs-, System- und Netzwerkumgebung.

Die Autonomie der Agenten erfordert, dass jeder Agent eine eigene, unabhängige Komponente zur *Verhaltensgenerierung* besitzt. Die Generierung des autonomen Verhaltens, das sich an abstrakten Zielen orientiert (E4), basiert auf dem konkreteren Weltmodell in Schicht S3 und bildet damit die neue Schicht S4.

Das Gesamtverhalten des Multiagentensystems, wie es sich gegenüber den Benutzern

darstellt, folgt übergeordneten globalen Zielen. Die Abbildung auf die lokalen Ziele der einzelnen Agenten und damit die Regelung ihrer *Koordination* erfolgt in der höchsten Schicht S5. Die resultierende Grundstruktur der Referenzarchitektur ist in Abbildung 3.10 dargestellt.

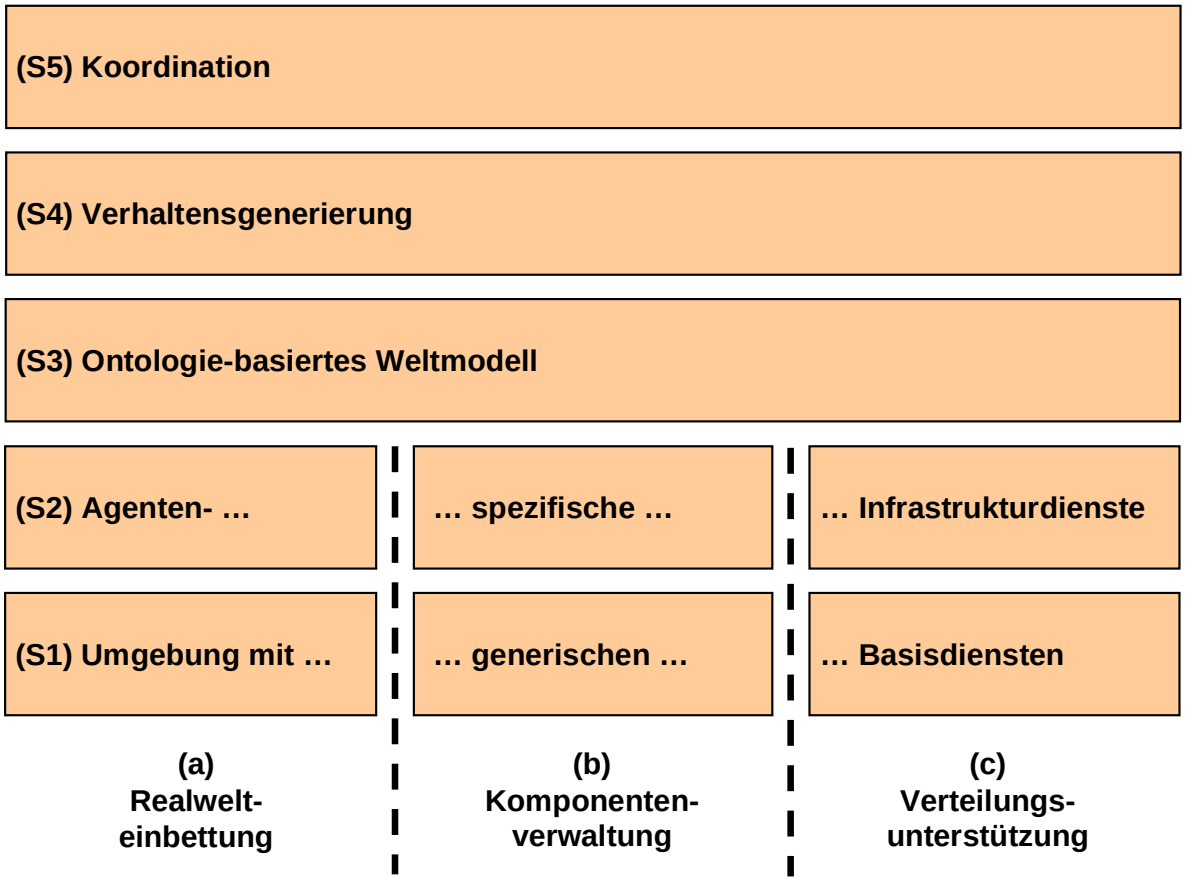


Abbildung 3.10: Resultierende Grundstruktur der Referenzarchitektur: fünf Schichten auf drei Säulen

Eine Gegenüberstellung der Grundstruktur mit dem ABSRM (Abbildung 3.7) zeigt, dass sich die Schichten S1, S2 und S4 im ABSRM wieder finden lassen. Damit kann die Referenzarchitektur aus der vorliegenden Arbeit – insbesondere nach der Verfeinerung im folgenden Abschnitt – als eine Konkretisierung des ABSRM aufgefasst werden.

3.3.2 Verfeinerung

Die Grundstruktur der Referenzarchitektur wurde im letzten Abschnitt auf Basis weniger ausgewählter Agenteneigenschaften skizziert. Ihre Verfeinerung erfolgt zunächst durch die erforderliche detaillierte Diskussion aller definierenden Eigenschaften. Im weiteren Verlauf finden dann auch die in den Abschnitten 3.2.2 und 3.2.3 vorgestellten verwandten Arbeiten Eingang.

Eigenschaft E1: Einbettung in Umgebung Eigenschaft E1 besagt, dass ein Agent ein in eine (Anwendungs-)Umgebung eingebettetes Rechnersystem ist. Diese Einbettung verlangt auf Ebene der Basisdienste *Sensoren und Aktoren*, um den Zustand der Realwelt erfassen und manipulieren zu können (S1a). Bei den häufig anzutreffenden agentenbasierten Simulationen wird die komplette Anwendungsumgebung simuliert, d.h. auch die Zustände und Dynamik der simulierten Realwelt werden dann in S1a durch Software nachgebildet. Die Umgebung („Environment“) der oben eingeführten Referenzarchitektur RASMAS (Abbildung 3.8) bietet hierfür ein Beispiel.

Die Signale der Sensoren werden für den Agenten zu Wahrnehmungen und die Aktoren setzen die Handlungen des Agenten in der Realwelt um. Die Konzepte *Wahrnehmung* und *Handlung* sind im Gegensatz zu den Sensoren und Aktoren nicht mehr generisch, sondern gehören in die nächsthöherliegende Schicht S2 der agentenspezifischen Infrastrukturdienste (S2a).

Um die Wahrnehmungen in einem Kontext interpretieren zu können und aufbauend auf der Interpretation zielführende Handlungen vornehmen zu können, bedarf der Agent eines weitreichenden Verständnisses für sein Anwendungsszenario, d.h. für seine Umgebung und die darin vorkommenden Akteure. Zur Formalisierung der Semantik einer Domäne verwendet man i.A. Ontologien, die die Artefakte der Domäne beschreiben und in Beziehung setzen. Die Säule der Realwelteinbettung wird also über spezifische *Schemata des Anwendungsszenarios* in die Schicht S3 eingebunden.

Der Agent erlangt zur Laufzeit sein Domänenverständnis, indem er sein Weltmodell als Instanz der Ontologien des Anwendungsszenarios betrachtet und dieses entsprechend den Wahrnehmungen fortschreibt. Die Dienste zum Fortschreiben des Weltmodells, wie

z.B. die Instanzenverwaltung und die Weltmodellrevision bilden einen weiteren Teil der Schicht S3.

Eigenschaft E2: Gekapselte Diensterbringung Nach Eigenschaft E2 erbringen Agenten in ihrer Umgebung einen nützlichen Dienst, der lediglich von außen beobachtet werden kann und dabei seine Funktionsweise nicht offenbart. Die Kapselung macht den Einzelagenten zum Bestandteil eines größeren Systems und ist damit auch jene Eigenschaft, die es überhaupt erst erlaubt, den Einzelagenten weitgehend unabhängig von einem Multiagentensystem zu betrachten, in das er ggf. eingebunden werden kann. Aus technischer Sicht handelt es sich bei Agenten um gekapselte Komponenten, die i.A. in einer Laufzeitumgebung ausgeführt werden. Die Laufzeitumgebung für Agenten ist Dienstnehmer einer herkömmlichen *Rechnerinfrastruktur* (S1b) mit *Hardware*, *Betriebssystem* und anderen grundlegenden Diensten, wie z.B. einer persistenten *Datenhaltung* durch Datenbanken.

Die Laufzeitumgebung ist als Dienstnehmer der Basisdienste der Systemumgebung selbst Teil der Schicht S2 in der Komponentenverwaltungssäule (S2b). Bei Multiagentensystemen ist die Laufzeitumgebung typischerweise der zentrale Bestandteil einer so genannten *Agentenplattform*. Diese umfasst auch die weiteren agentenspezifischen und anwendungsgenerischen Infrastrukturdienste, wie z.B. die Lebenszyklusverwaltung oder die Mobilitätsverwaltung.

Eigenschaft E3: Autonomie Die Autonomie der Agenten erweitert die Anforderungen an die Laufzeitumgebung (S2b) und damit indirekt auch an die Rechnerinfrastruktur (S1b): Damit ein Agent die Kontrolle über sein eigenes Verhalten beständig innehaben kann, dürfen keine Abhängigkeiten vom Voranschreiten anderer Agenten oder der Laufzeitumgebung bestehen. Folglich muss die Infrastruktur jedem Agenten einen eigenen Kontrollfaden zur Verfügung stellen.

Eigenschaft E4: Zielorientierung Von der Autonomie des Agenten wurde in Eigenschaft E4 die Ziel-Orientierung des Handelns abgeleitet. Dabei können Agenten zugleich mehrere Ziele verfolgen, die abhängig vom Zustand des Agenten und seiner Umgebung

priorisiert sind. Zum Erreichen eines Ziels stehen dem Agenten möglicherweise wiederum verschiedene Handlungsalternativen zur Verfügung. Dies führt für die Verhaltensgenerierung eines Agenten zu einem zweischrittigen Verfahren aus *strategischer Zielabwägung* (engl. *goal deliberation*) und *taktischer Aktionsauswahl* (engl. *means-end assessment*). Beide Arbeitsschritte greifen auf die Instanzen des Weltmodells zu und gehören als Bestandteile der Verhaltensgenerierung zur Schicht S4.

Häufig werden die Arbeitsschritte Zielabwägung und Aktionsauswahl nicht prozedural realisiert, sondern mittels Inferenz über eine symbolische Darstellung der Umgebung und des Agentenzustandes u.a. mit seinen Zielen und den in Frage kommenden Aktionen. Diese Vorgehensweise erweitert die Anforderungen an Schicht S3, die eine entsprechende symbolische Darstellung anbieten muss.

Eigenschaft E5: Reaktivität Die erste Eigenschaft für *intelligente* Agenten, die es zu betrachten gilt, ist die Reaktivität, d.h. die Eigenschaft eines Agenten, auf eine Veränderung in seiner Umgebung zeitnah reagieren zu können. Dazu erscheint es sinnvoll, die kurzfristige Aktionsauswahl und die längerfristigen Zielabwägung in eigenen Subkomponenten zu isolieren, um zu gewährleisten, dass auf Ereignisse in der Umgebung schnell reagiert werden kann, ohne erst den kompletten Zyklus der Verhaltensgenerierung durchlaufen zu müssen. Zielabwägung und Aktionsauswahl bilden damit Teilkomponenten der Schicht S4, die eigenständig angesprochen werden können.

Darüber hinaus müssen die unteren Schichten über die Möglichkeit verfügen, abhängig von bestimmten Realweltzuständen eine Reaktion in den jeweils übergeordneten Schichten anzustoßen („zu triggern“). Dies betrifft die (Teil-)Schichten S1a, S2a und S3.

Eigenschaft E6: Ausgewogenheit zwischen Reaktivität und Zielorientierung
Die Zweiteilung zwischen Zielabwägung und Aktionsauswahl wird auch von E6 untermauert, die für intelligente Agenten Ausgewogenheit zwischen Reaktivität und Ziel-Orientierung fordert. Die Ausgewogenheit schlägt sich nieder im Zusammenspiel der beiden Subkomponenten und insbesondere im Verhältnis der Häufigkeiten von Aktionsauswahl und Zielrevision in der Verhaltensgenerierung (S4).

Eigenschaft E7: Proaktivität Die Proaktivität stellt an die Referenzarchitektur sowohl technische Anforderungen auf der Ebene der agentenspezifischen Infrastrukturdienste, als auch konzeptionelle Anforderungen an die Verhaltensgenerierung. Proaktivität ist eng verbunden mit der ziel-orientierten Autonomie der Agenten und fügt dem unabhängigen Handeln den eigenen dauerhaften Antrieb zur Verfolgung von Zielen unabhängig von Stimuli aus der Umgebung hinzu. Dies unterstreicht die bereits bei der Autonomie aufgestellte Forderung nach einem eigenen *Kontrollfaden je Agent* an die Agentenplattform in S2b. Außerdem muss die Verhaltensgenerierung (S4) die Proaktivität unterstützen, indem sie bei der Zielabwägung eine dauerhafte Verfolgung übergeordneter Ziele erlaubt, die nicht ohne weiteres zugunsten kurzfristiger Verhaltensweisen verworfen werden sollten.

Eigenschaft E8: Sozialfähigkeit Die Sozialfähigkeit der Agenten bringt eine ganze Reihe zusätzlicher Anforderungen an die Referenzarchitektur mit sich. Die entsprechenden Bausteine wurden bereits bei den verwandten Arbeiten (Abschnitt 3.2.2) zusammen mit der Kommunikationsschichtenarchitektur nach FIPA in Abbildung 3.6 dargestellt. Im folgenden werden diese Bausteine den Abstraktionsniveaus der Referenzarchitektur zugeordnet und dann in deren Schichten eingegliedert. Das Ergebnis – die vollständige und allgemeingültige Referenzarchitektur für intelligente Multiagentensysteme – ist in Abbildung 3.11 dargestellt.

Die *Datenkommunikation* als Bestandteil der Verteilungsunterstützung (c) besteht aus dem Transportkanal (KS1) und den Transportprotokollen (KS2). Beide sind generisch für beliebige verteilte Systeme anwendbar und sie gehören damit zu den generischen Basisdiensten der Umgebung, genauer zur Teilschicht S1c.

Die Dienste für den *Nachrichtentransport* sind im Vergleich zur Datenkommunikation auf einem höheren Abstraktionsniveau. So weisen Transport- und Inhaltssyntax (KS3 und KS4) agentenspezifische Erweiterungen auf, auch wenn als Basis häufig auf generische Formate zurückgegriffen wird. Folglich sind sie der agentenspezifischen Infrastruktur in S2c zuzuordnen. Meist bieten die Implementierungsrahmenwerke, die als einen Bestandteil die Agentenplattformen in S2b mitbringen, auch Parser für diese Syntaxen an, die z.B. den Zusammenbau und das Auslesen der entsprechenden Nachrichten

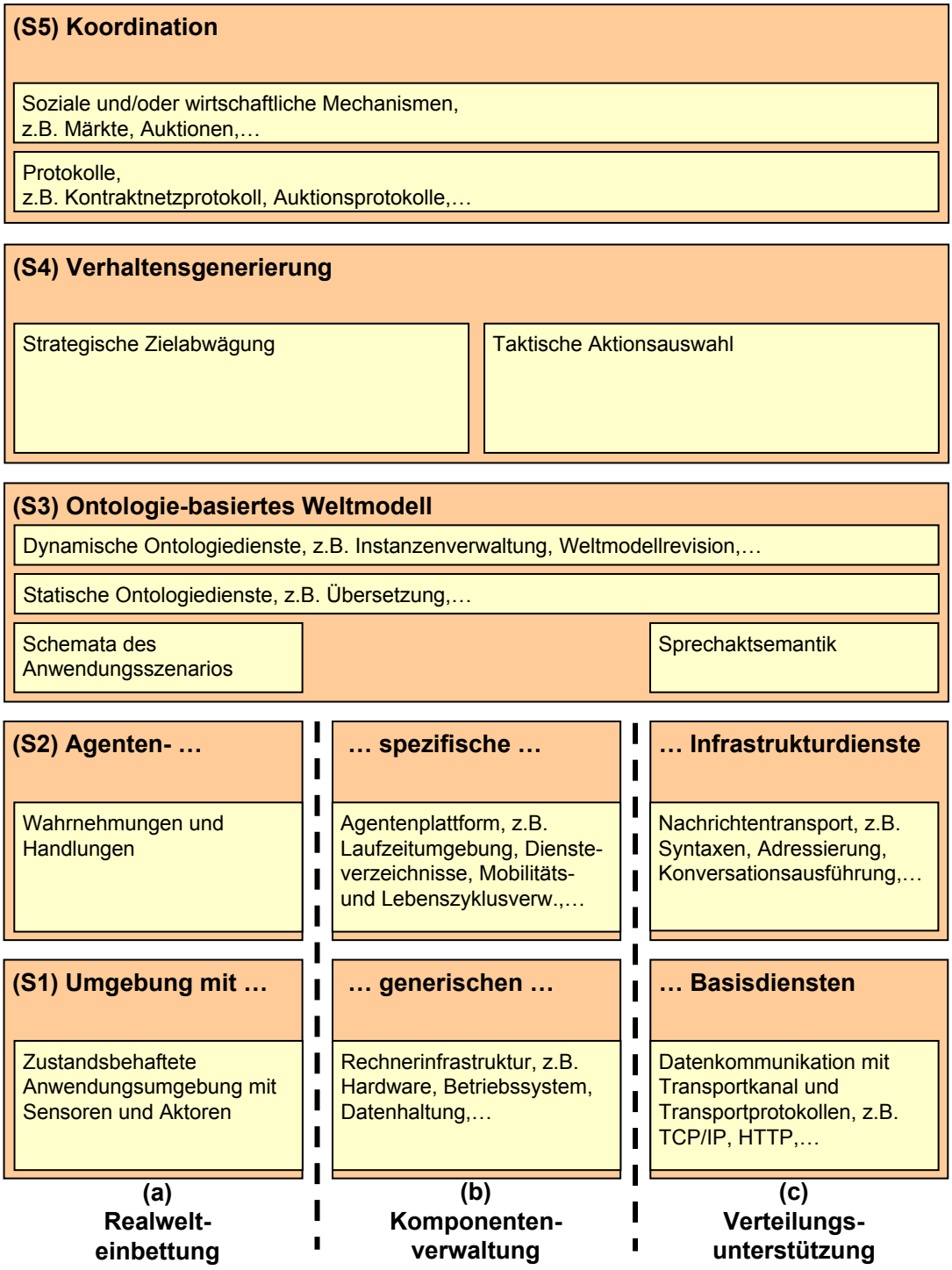


Abbildung 3.11: Referenzarchitektur intelligenter Agenten in Multiagentensystemen (siehe auch Lockemann u. a. 2006)

erleichtern.

Durchsuchbare *Diensteverzeichnisse* zum Auffinden der Dienstangebote anderer Agenten im nachrichtenbasierten MAS ließen sich prinzipiell als Bestandteil der Komponentenverwaltung oder auch der Verteilungsunterstützung in der Schicht S2 der Agentenspezifischen Infrastrukturdienste vorstellen. In den FIPA-Standards sind die Diensteverzeichnisse nicht in der Kommunikationsschichtenarchitektur sondern in der abstrakten Architektur beheimatet, weshalb sie in der Referenzarchitektur den Agentenplattformen in S2b zugeordnet werden sollen.

Die Ontologieschicht KS5 der Kommunikationsschichtenarchitektur hat in der Referenzarchitektur in Schicht S3 bereits eine direkte Entsprechung. Allerdings erweitern sich deren Aufgaben um die kommunikationstypischen Problemstellungen, wie z.B. um einen *statischen Dienst zur Übersetzung* von Instanzen aus der Ontologie eines Senders in semantisch äquivalente Ausdrücke in einer Empfängerontologie.

Die Sprechakte in KS6 geben an, wie mit den Inhalten der Nachrichten in Bezug auf das Weltmodell des empfangenden Agenten umgegangen werden soll. Bei *„einen Apfel essen“* könnte es sich z.B. um eine Beobachtung einer momentan selbst ausgeführten Tätigkeit – also einen Zustand – handeln oder um die Aufforderung einer Marktfrau, von einem Apfel zu kosten – also um einen Auftrag. Die Schichten S3 und S4 sind also noch um Dienste zu ergänzen, die die Integration von Nachrichten in das Weltmodell und ggf. eine Aktionsauswahl entsprechend der *Sprechaktsemantik* erlauben.

Die Konversationen sind in der FIPA-Kommunikationsschichtenarchitektur in Schicht KS7 angesiedelt, da die formale Semantik ihrer Protokolle aus der formalen Semantik der konstituierenden Sprechakte von KS6 abgeleitet wird. Die Protokollsemantik richtet sich nach den *sozialen und/oder wirtschaftlichen Mechanismen* aus KS8. Die globalen Mechanismen werden damit zum Hauptbaustein der bislang noch nicht weiter ausgearbeiteten Koordinationsschicht (S5). Hier findet auch die Abbildung und der Ausgleich zwischen den globalen Zielen des Multiagentensystems und den lokalen Zielen der Verhaltensgenerierung auf S4 der einzelnen Agenten statt.

Aus Sicht der Implementierung – und damit der Referenzarchitektur – kommen auf S5 zur verteilten Abwicklung der Koordination also noch die entsprechenden *Protokolle*

hinzu, also die Ablaufbeschreibungen für die Konversationen. Die Ablaufbeschreibungen auf Schicht S5 führen auf Schicht S2c (agentenspezifische Infrastrukturdienste zur Verteilungsunterstützung) zu einem weiteren Dienst, nämlich zur *Syntax-basierten Konversationsausführung und -überwachung*, die die Abarbeitung der zulässigen Abläufe durch die jeweiligen Kommunikationspartner antreibt und überwacht.

3.4 Bewertung der Referenzarchitektur

Nach dem in Abschnitt 3.1.1 entworfenen Entwicklungsprozess für die Referenzarchitektur erfolgt deren Bewertung durch eine Diskussion der Allgemeingültigkeit und Vollständigkeit. Dazu wird einerseits die Flexibilität der Referenzarchitektur durch die „Kompatibilität“ mit den möglichen Anforderungen durch weitere optionale Agenteneigenschaften untersucht (3.4.1) und andererseits wird ihre Anwendbarkeit überprüft durch den Nachweis der „Konformität“ zu bekannten konkreten Architekturen (3.4.2).

3.4.1 Flexibilität der Referenzarchitektur: Kompatibilität mit weiteren möglichen Agenteneigenschaften

Der Anspruch einer allgemeingültigen und vollständigen Referenzarchitektur erfordert, dass sie flexibel genug ist, weiteren optionalen Eigenschaften, die Agenten zugeschrieben werden können, nicht zuwider zu laufen. Als weitere Eigenschaften für intelligente Agenten wurden in Unterabschnitt 3.2.1 Lernfähigkeit (E9) und Mobilität (E10) sowie der „starke Agentenbegriff“ ausführlicher eingeführt.

Eigenschaft E9: Lernfähigkeit Lernfähige Agenten müssen zunächst die Wirkung ihrer ausgeführten Aktionen in der Umgebung beobachten können (in S1a und S2a), um deren Erfolg oder Misserfolg beurteilen zu können. Die dabei gesammelten Erfahrungen über die Vergangenheit müssen im Weltmodell von Schicht S3 abgelegt werden können. Auf dieser Basis kann in einer zukünftigen Aktionsauswahl (in S4) abgeschätzt werden, wie erfolgversprechend die erneute Ausführung einer bestimmten Aktion ggf. unter

neuen Rahmenbedingungen sein kann. Soll das erlernte Verhalten eines Agenten zeitlich über das Ende seines eigenen Lebenszyklus hinaus von Bestand sein, damit es an neue Inkarnationen seines Agententyps „weitervererbt“ werden kann, so müssen auch entsprechende Erweiterungen zur Persistenz des Agentenzustandes in S1b vorhanden sein.

Eigenschaft E10: Mobilität Die Herausforderungen, die die Mobilität von Agenten mit sich bringt, betreffen vorrangig die Infrastruktur in den unteren Teilschichten S1b und S2b. Z.B. müssen sowohl der Agenten-Programmcode als auch der Agentenzustand serialisierbar sein und mithin alle Fragen mobilen Codes gelöst werden. Insbesondere sind dies Anforderungen an die Systemsicherheit, die sowohl von Seiten des Betriebssystems als auch von den eingesetzten Programmiersprachen und Agentenplattformen erfüllt werden müssen. Alle genannten Anforderungen könnten in den bereits vorhandenen Komponenten umgesetzt werden und bleiben damit innerhalb der Spezifikation der Referenzarchitektur.

Starker Agentenbegriff Die innere Architektur der Agenten, die in der Referenzarchitektur im wesentlichen in der Verhaltensgenerierungsschicht S4 zum Liegen kommt, wurde bislang bewusst nur vergleichsweise grob beschrieben. Lediglich die Unterteilung in strategische Zielabwägung und taktische Aktionsauswahl wurde vorgenommen, nicht jedoch festgelegt, wie diese realisiert werden können. Hauptgrund hierfür ist die Bewahrung der Allgemeingültigkeit: schon eine geringfügige weitere Spezifikation könnte einen Teil der vielfältigen möglichen Realisierungen ausschließen.

Umgekehrt gilt nun: Agenten, die dem starken Agentenbegriff genügen, d.h. die z.B. die mentalistischen Konzepte „Wissen“, „Glauben“, „Absichten“ und „Überzeugungen“ bei ihrer Verhaltensgenerierung einsetzen, konkretisieren die Schicht S4 und ggf. das Weltmodell aus Schicht S3.

3.4.2 Anwendbarkeit der Referenzarchitektur: Konformität zu existierenden Agentenarchitekturen

Ein weiteres Gütemerkmal für eine Referenzarchitektur ist ihre Konformität zu der Klasse der beschriebenen Systeme, die sich im vorliegenden Fall direkt auf deren Anwendbarkeit zur Verortung der Programmieraspekte auswirkt. Beneken (2006) definiert die Konformität:

> „Ein System ist zu einer Referenzarchitektur *konform*, wenn sich die Struktur oder die Strukturen der Referenzarchitektur darin wiederfinden. Es muss Schnittstellen, Schichten oder Komponenten enthalten, die wie in der Referenzarchitektur in Beziehung stehen."

Für die vorliegende Referenzarchitektur wird diese Definition wie folgt konkretisiert: die Vollständigkeit und Allgemeingültigkeit der Referenzarchitektur – und damit deren Anwendbarkeit – wird umso stärker untermauert, je besser ihre Konformität zu den Architekturen gebräuchlicher konkreter Multiagentensysteme ist und je heterogener die Menge der betrachteten konkreten Architekturen ist.

Als Konsequenz wird im folgenden eine Reihe sehr unterschiedlicher Architekturen betrachtet und ihre jeweilige Abbildung auf die Referenzarchitektur wird untersucht. Den Ausgangspunkt bieten die verwandten Arbeiten aus dem Bereich der Standards und der Referenzarchitekturen, die teilweise auch schon in die Definition der eigenen Referenzarchitektur Eingang fanden (3.2). Darüber hinaus wird eine Auswahl verbreiteter konkreter Architekturen mit sehr unterschiedlichem Hintergrund im Vergleich zur Referenzarchitektur diskutiert. (Ein ähnliches Vorgehen wurde auch für eine Auswahl der Architekturen in (Lockemann und Nimis 2005) gewählt.)

FIPA Standard

Ziel der Foundation for Intelligent Physical Agents (FIPA) ist die Spezifikation von Standards für Agentenplattformen, die die Interoperabilität heterogener Agenten als technische Manifestierung der Sozialfähigkeit ermöglichen sollen. Die wichtigsten Übereinkünfte, die bereits in Abschnitt 3.2.2 als verwandte Arbeiten eingeführt wurden, sind

die FIPA Abstract Architecture und der FIPA-Protokollstapel. Abbildung 3.12 zeigt in Ergänzung zu Abbildung 3.5 eine integrierte Darstellung der beiden entsprechenden Standardgruppen und illustriert außerdem eine vorläufige Zuordnung von (Teil-)Schichten der Referenzarchitektur zu FIPA-Komponenten.

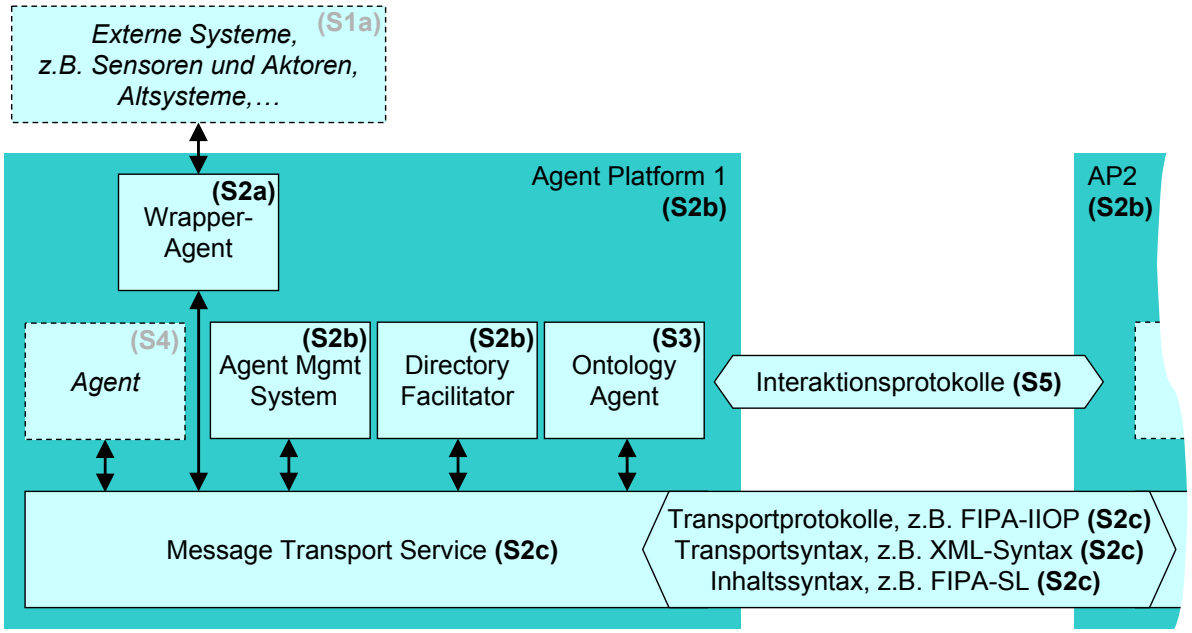


Abbildung 3.12: Zuordnung von Referenzarchitekturschichten zur erweiterten FIPA Abstract Architecture

Da der FIPA-Protokollstapel bereits im Entwicklungsprozess der Referenzarchitektur eine wichtige Rolle gespielt hat, ist es nicht verwunderlich, dass die dort enthaltenen FIPA-Standards leicht Komponenten aus der Referenzarchitektur zugeordnet werden können: Nachrichtentransportdienst, Transportprotokolle, Transport- und Inhaltssyntax finden sich in der Referenzarchitektur bei der Verteilungsunterstützung auf S2c wieder und der Directory Facilitator entspricht dem Diensteverzeichnis in S2b. Der Ontologieagent ist bei der FIPA zuständig für die semantische Vermittlung zwischen kooperierenden Agenten, d.h. er sorgt durch die Übersetzung zwischen Sender- und Empfängerontologie für ein gemeinsames Verständnis des Anwendungsszenarios. Diese Funktionalität lässt sich den statischen Ontologiediensten in Schicht S3 der Referenzarchitektur zuordnen.

Bei den FIPA-Standards rund um die Abstract Architecture steht die Definition einer

Laufzeitumgebung für die Agenten als gekapselte Diensterbringer im Mittelpunkt. Die FIPA-Komponenten der Agentenplattform lassen sich abbilden auf die Teilschicht S2b, wie z.B. der Agent Management Service (AMS) auf die dortige Lebenszyklusverwaltung.

Die Einbettung der Agenten in die Realwelt liegt nicht im Fokus der FIPA, da sie für die Interoperabilität zwischen den Agenten nicht von Bedeutung ist. Dennoch kann man die Spezifikation der Wrapper-Agenten als Versuch sehen, die FIPA-Agenten in eine bestehende Software-Realwelt einzubetten. Bei den daraus resultierenden Agenten handelt es sich in der Regel nicht um vollwertige Agenten im Sinne der Referenzarchitektur, sondern um „entartete“ Agenten, die nur eine Auswahl der Schichten implementieren und den vollwertigen Agenten Teilfunktionalitäten zur Verfügung stellen. So könnte es sich bei den zu integrierenden Fremdsystemen beispielsweise auch um Treiber für Sensoren und Aktoren oder Komponenten eines Altsystems handeln, die in der Referenzarchitektur auf den Schichten S1a anzusiedeln wären. Der Wrapper-Agent wäre dann entsprechend Bestandteil der Schicht S2a.

Genau wie die Einbettung in die Realwelt ist auch der innere Aufbau der Agenten für die Interoperabilität nicht von Bedeutung, sondern lediglich deren nach außen sichtbares Verhalten. Folglich findet die agentische Verhaltensgenerierung (S4) im Allgemeinen in den Standards der FIPA keine direkte Entsprechung. Sie geschieht auf beliebige Weise innerhalb der einzelnen Agenten des Multiagentensystems. Dies gilt in Teilen auch für die Koordination (S5): Sie wird von der FIPA nicht durch die Standardisierung der Mechanismen an sich unterstützt, sondern lediglich durch die Definition einer Auswahl zugehöriger Interaktionsprotokolle, die für die Interoperabilität relevant sind.

Die in Abbildung 3.12 illustrierte Zuordnung von Referenzarchitekturschichten zu Elementen der FIPA-Architektur lässt sich mit der obigen Argumentation auch umkehren: die Elemente der FIPA-Architektur lassen sich genauso umgekehrt auf die Referenzarchitektur abbilden. Das Resultat ist in Abbildung 3.13 dargestellt. Das Prinzip dieser Umkehrung ist für die weitere Arbeit von besonderer Bedeutung: betrachtet man eine Komponente eines bereits existierenden Systems, für das es eine Abbildung auf eine bestimmte Stelle der Referenzarchitektur gibt, dann gelten die über die Stel-

le der Referenzarchitektur getroffenen Aussagen auch für die abgebildete Komponente selbst.

Zusammenfassend lässt sich für die FIPA Abstract Architecture feststellen, dass sich alle Komponenten der (erweiterten) Architektur einfach und auf natürliche Weise auf die Referenzarchitektur abbilden lassen. Dass nicht alle Komponenten der Referenzarchitektur eine Entsprechung auf FIPA-Seite haben, liegt am vergleichsweise engeren Fokus der FIPA auf die Interoperabiltät von Multiagentensystemen.

Agent-Based System Reference Model (ABSRM)

Bereits bei der Begründung der Schichtung als Grundstruktur wurde auf die Ähnlichkeit des ABSRM (Agent-Based System Reference Model, Modi u. a. 2006) zur eigenen Referenzarchitektur hingewiesen. Dabei handelt es sich bei dem in Abbildung 3.7 im Original dargestellten Modell zugleich um eine Abstraktion der Referenzarchitektur, wie auch um eine etwas andere Sicht auf Multiagentensysteme. Während die Referenzarchitektur versucht, alle bei der Agentenprogrammierung relevanten Komponenten zu nennen und zueinander in Beziehung zu setzen, betrachtet das ABSRM exklusiv die Gestalt des Systems zur Laufzeit (engl. *Deployment View*).

Besonders augenfällig ist der Verzicht des ABSRM auf die in der Referenzarchitektur beschriebenen Säulen. Stattdessen wird die Realwelteinbettung in der ABSRM auf das komplette Modell bezogen und die *Physical World* gehört damit als unterste Schicht in S1a, S1b und S1c der Referenzarchitektur. Die nächstabstraktere Schicht wird von den *Hosts* gebildet, die aus Sicht der Referenzarchitektur die Dienste der Rechnerinfrastruktur (in S1b) und der Kommunikationsinfrastruktur (S1c) anbieten.

Die *Platforms* des ABSRM erbringen die Aufgaben der Agentenplattform in S2b und des Nachrichtentransports in S2c der Referenzarchitektur. *Sensor* und *Effector Interface*, die in der ABSRM im Agenten angeordnet sind, komplettieren die Schicht der agentenspezifischen Infrastrukturdienste. Sie lassen sich in der Referenzarchitektur auf die Schicht S2a abbilden, da sie den Wahrnehmungen und Handlungen in der Realwelt Rechnung tragen.

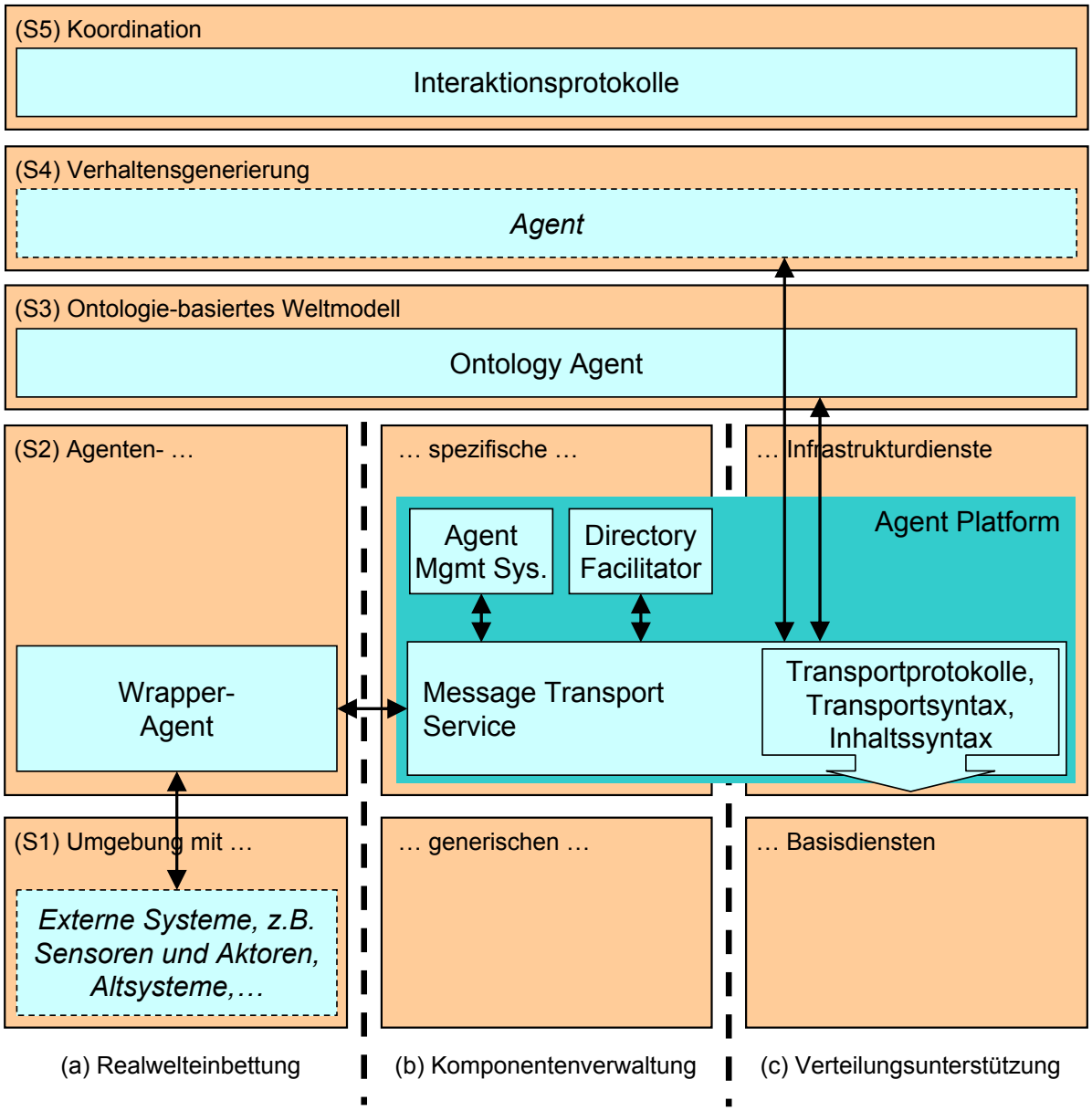


Abbildung 3.13: Abbildung der erweiterten FIPA Abstract Architecture auf die Referenzarchitektur

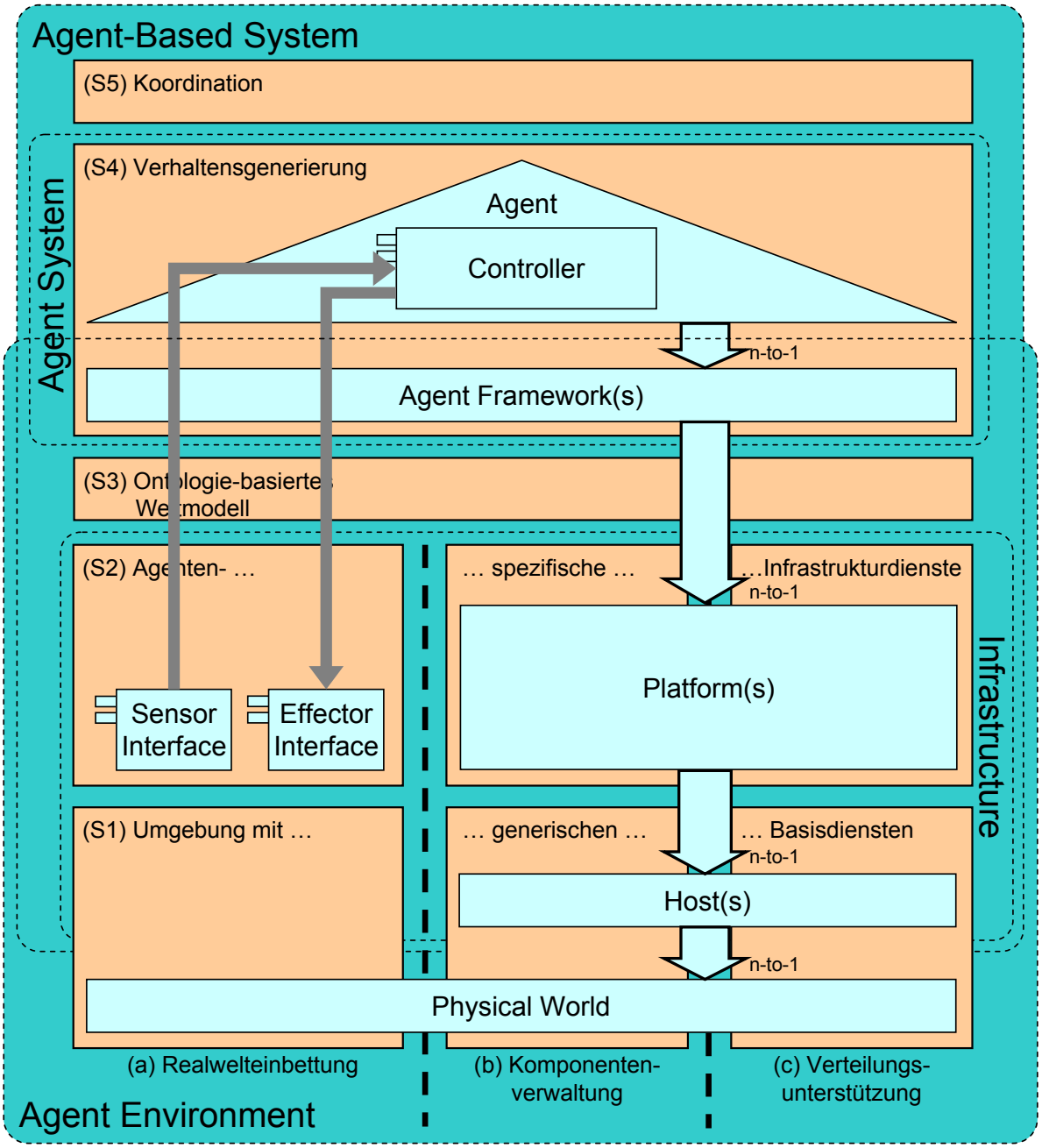


Abbildung 3.14: Zuordnung des ABSRM zur Referenzarchitektur

Die Schichten S3 und S5 der Referenzarchitektur werden im ABSRM nicht explizit aufgegriffen. Sie fallen implizit in das bewusst recht primitiv gehaltene Agentenmodell aus den bereits zugeordneten *Sensor Interface* und *Effector Interface* und aus dem *Controller*, wobei letzterer in die Verhaltensgenerierungsschicht S4 fällt. Der *Controller* des ABSRM greift bei der Verhaltensgenerierung auf ein oder mehrere *Agent Frameworks* zurück, die den Rahmen für die Implementierung der Intelligenz bieten sollen. Sie sind damit als Grundbestandteil der Verhaltensgenerierung ebenfalls auf die Schicht S4 abzubilden.

Die in ABSRM spezifizierten Funktionseinheiten, die dort nicht in der Architektur verankert sind, wie z.B. *Security*, *Conflict Management* und *Messaging*, ließen sich einfach nicht bzw. schwach besetzten Schichten des Referenzmodells zuordnen (S2b, S5 bzw. Verteilungunterstützungssäule (S1/2c)).

Fazit: Wie Abbildung 3.14 zeigt, lassen sich alle Artefakte aus dem ABSRM auf Komponenten der Referenzarchitektur abbilden. Die Differenzen der Beziehungen in den beiden Architekturen sind Folge der unterschiedlichen eingenommenen Blickwinkel und der unterschiedlichen Detailierungsgrade.

Reference Architecture for Situated Multiagent Systems (RASMAS)

Die Reference Architecture for Situated Multiagent Systems (RASMAS), die zusammen mit dem ABSRM bei den verwandten Arbeiten in Abschnitt 3.2.3 vorgestellt wurde und dort auch in Abbildung 3.8 unverändert dargestellt ist, weist eine andere Grundstruktur auf als die Referenzarchitektur der vorliegenden Arbeit. Bei genauerer Betrachtung der Architektur kann man von zwei ineinander geschachtelten Architekturmustern sprechen. Das äußere Muster ist die Strukturierung in die beiden Schichten *Environment* und *Agent* („Primary Abstractions“). Erstere fasst die Schichten S1 und S2 der eigenen Referenzarchitektur zusammen, letztere die Schichten S3, S4 und S5. Die Grobstrukturen lassen sich also leicht einander zuordnen. In Abbildung 3.15 ist RASMAS erneut dargestellt – dieses Mal entsprechend abgebildet auf die Referenzarchitektur.

Innerhalb ihrer beiden Schichten Agent und Environment weist die RASMAS als zweites „Architekturmuster“ eine durch Daten- und Kontrollflüsse eng vermaschte Menge von

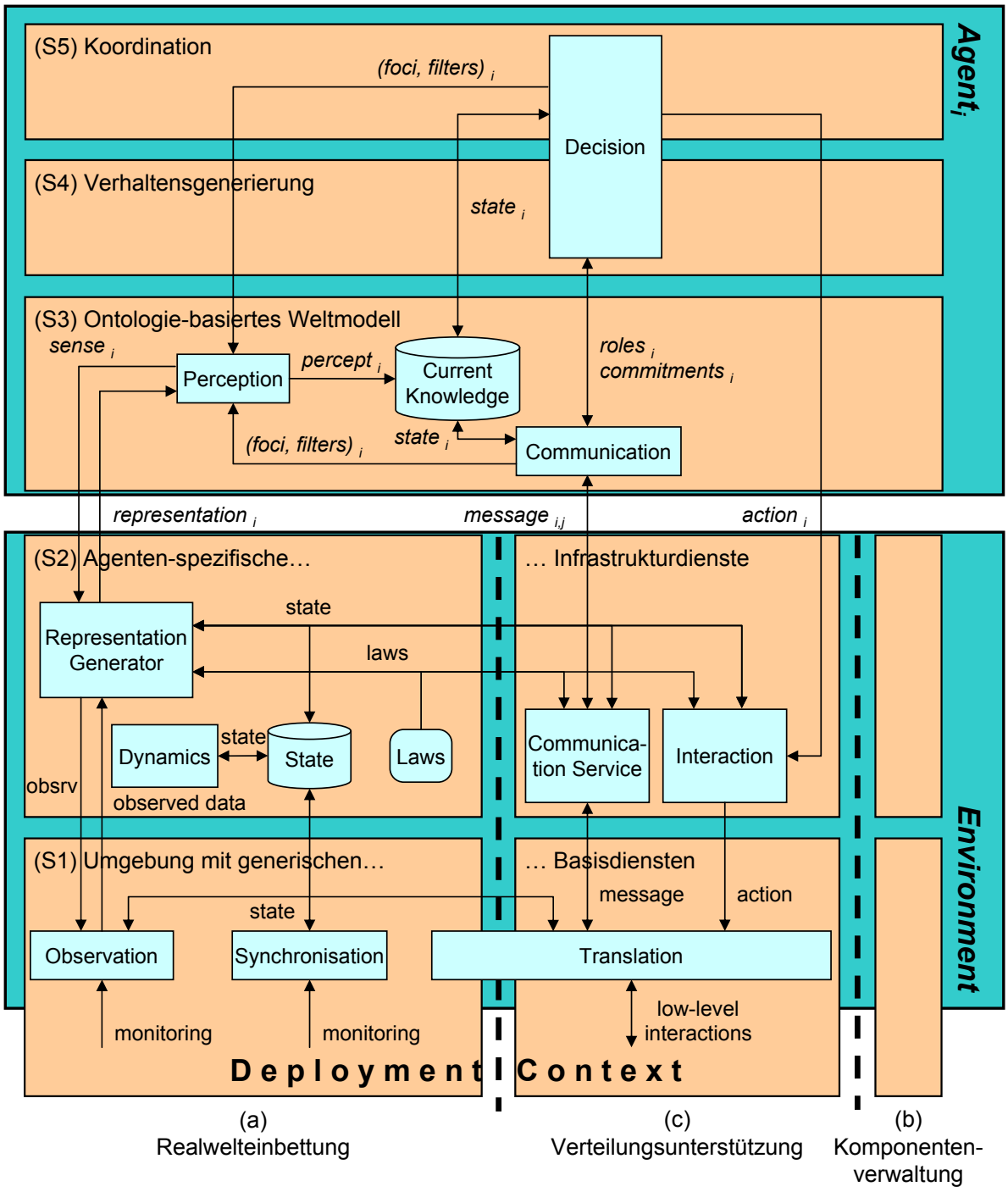


Abbildung 3.15: Reference Architecture for Situated Multiagent Systems (RASMAS) und die Referenzarchitektur

Komponenten auf. Die Komponenten *Perception*, *Current Knowledge* und *Communication* aus der Primary Abstraction Agent sind der Schicht S3 zuzurechnen: Perception und Communication sind verantwortlich für die Einbettung der Umgebungsereignisse bzw. der Nachrichten in das Current Knowledge, das dem Weltmodell entspricht. Die Verhaltensgenerierung (S4) liegt zusammen mit allen Aspekten der Koordinationsmechanismen (S5) in der Agent-Komponente *Decision*.

Bedingt durch den Forschungshintergrund von RASMAS in der Schwarm-Theorie, bei der sich die Agenten der MAS über Spuren in der Umgebung koordinieren, ist die Primary Abstraction *Environment* stärker ausgearbeitet als der Agent selbst. Dabei lassen sich zwei der drei Säulen aus der Referenzarchitektur dieser Arbeit jeweils in einer Gruppe von Komponenten wieder erkennen: Communication Service (S2), Interaction (S2) und Translation (S1) sind der Verteilungsunterstützung in Säule (c) zuzuordnen. Alle anderen Komponenten beziehen sich auf die Realwelteinbettung (a). Der *Representation Generator* sorgt mit Hilfe der *Dynamics*, *State* und *Laws* für die Wahrnehmungen und Handlungen auf Schicht S2a, während *Observation*, *Synchronisation* die Software-Schnittstellen für die physikalische Anwendungsumgebung anbieten (S1a) bzw. für deren Simulation.

Wie in Abbildung 3.15 illustriert, lassen sich in RASMAS also vier der fünf Schichten und zwei der drei Säulen aus der Referenzarchitektur direkt zuordnen. Dabei kann man erkennen, dass in manchen Bereichen RASMAS genauer ausdetailliert ist, in anderen Teilen die eigene Referenzarchitektur, und dass keine Widersprüche zwischen den Architekturen existieren. Die beiden Architekturen lassen sich also zusammenfassend mit Recht als *konform* zueinander bezeichnen und damit ist die Referenzarchitektur auch in diesem Fall anwendbar.

InteRRap

Die Intelligenz der Agenten ist durch die Ausgewogenheit (E6) von zielorientiertem (E4) und reaktivem Verhalten (E5) aus eigenem Antrieb (E7) und durch ihr Sozialverhalten (E8) definiert. Diese Eigenschaften beschreiben also, *wie* ein intelligenter Agent seinen Dienst erbringt. Die entsprechenden Verfahren zur Generierung des Individual-

und Sozialverhaltens werden von den sogenannten inneren Agentenarchitekturen betrachtet, wie z.B. von der BDI-Architektur (Bratman 1987; Bratman u. a. 1988) oder von der InteRRap-Architektur (Müller 1996). Anhand letzterer soll exemplarisch erörtert werden, wie sich innere Agentenarchitekturen auf die Referenzarchitektur abbilden lassen.

Hauptziel von InteRRap ist die Kombination von Reaktivität, Zielorientierung und Kooperation für wissensbasierte Agenten in einem *integrierten Modell* (siehe Abbildung 3.16). Es ist also zu erwarten, dass die zentralen Bestandteile der InteRRap-Architektur besonders die Verhaltensgenerierung (S4) und die Koordinationsschicht (S5) genauer ausgestalten.

Die Hauptbestandteile von InteRRap sind eine Umgebungsschnittstelle, eine dreischichtige Wissensbasis und eine dreischichtige Kontrolleinheit. Die Wissensbasis ist unterteilt in ein Weltmodell, ein mentales Modell und ein soziales Modell. Analog unterteilt sich die Kontrolleinheit in die Schichten *Behavior-based Layer*, *Local Planning Layer* und *Cooperative Planning Layer*. Die beiden ersteren sind verantwortlich für das kurzfristige reaktive Verhalten und das längerfristige zielorientierte Verhalten, während letztere die Sozialfähigkeit realisiert.

Die drei Schichten der Kontrolleinheit sind jeweils weiter spezifiziert durch die miteinander verknüpften Funktionsbausteine SG (*situation recognition and goal activation*) und PS (*planning and scheduling*). Die drei SG-Bausteine stellen aufgrund des jeweiligen Wissensmodells und ggf. durch eine Stimulation durch den PS-Baustein der untergeordneten Schicht fest, wann für ihre eigene Schicht Handlungsbedarf besteht. In diesem Fall wird der eigene SG-Baustein angestoßen, der aus den übergebenen Situations- und Zielbeschreibungen eine neue Vorgehensweise ableitet, die entsprechenden Schritte des Agenten plant und sie zur Ausführung an die unterliegenden Schichten propagiert bzw. selbst wiederum den SG-Baustein der übergeordneten Schicht stimuliert. Die genaue Spezifikation der Funktionsbausteine und ihrer Interaktionen zusammen mit einem ausführlichen Beispiel erfolgt in (Müller 1996).

Die Querbezüge zwischen InteRRap und der Referenzarchitektur sind augenscheinlich. Die Wissensbasis von InteRRap entspricht dem Weltmodell der Referenzarchitektur, d.h. der Schicht S3, wobei zwischen drei konzeptuellen Ebenen unterschieden wird,

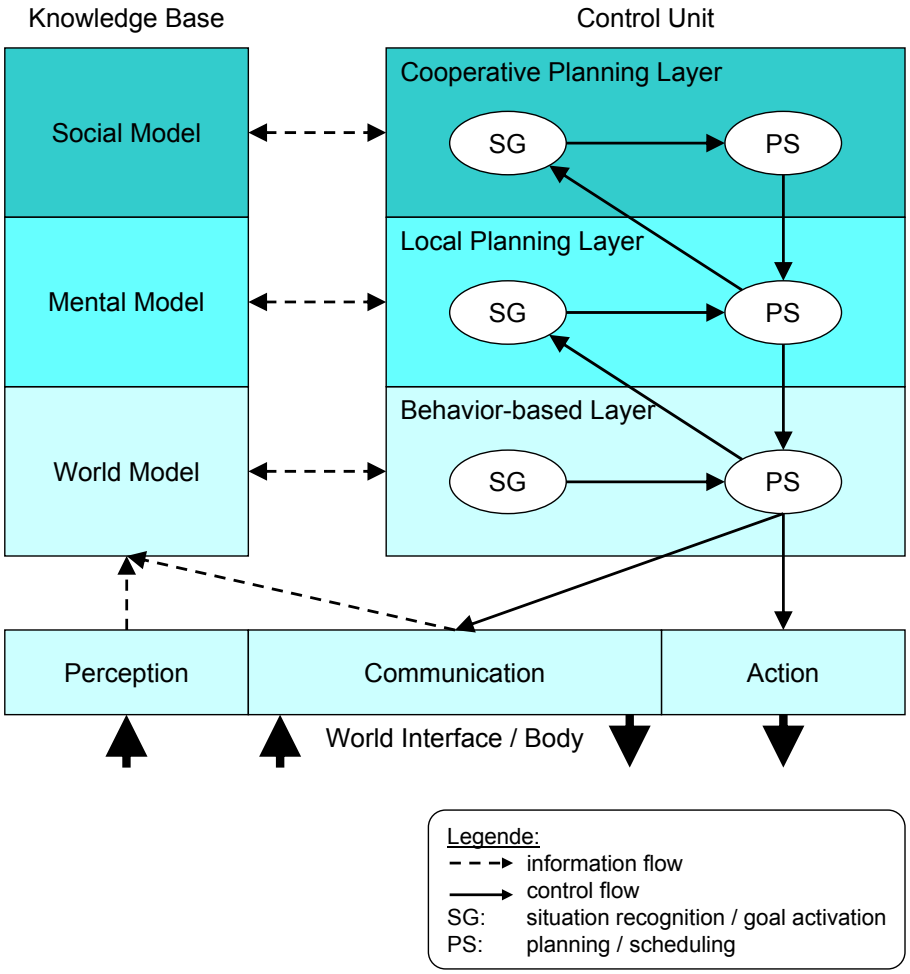


Abbildung 3.16: Die innere Agentenarchitektur InteRRap

die technische Trennung nach statischen und dynamischen Diensten unterbleibt jedoch.

Die Unterteilung der Kontrolleinheit in *Behavior-based Layer*, *Local Planning Layer* und *Cooperative Planning Layer* bildet die Strukturierung der (Teil-)Schichten S4 – mit strategischer Zielabwägung und taktischer Aktionsauswahl – und S5 der Referenzarchitektur ab. Wie zu vermuten war, sind diese Architekturelemente entsprechend dem Hauptziel von InteRRap auf die zugehörigen Eigenschaften deutlich weiter ausgearbeitet als in der Referenzarchitektur, wohingegen die implementierungsorientierten Architekturelemente nur rudimentär durch die Einbettung in die Umgebung analaog S2a und S2c betrachtet werden.

Abbildung 3.17 illustriert die Abbildung der Elemente von InteRRap auf die Referenzarchitektur. Während man RASMAS als eine Konkretisierung der Referenzarchitektur mit Schwerpunkt auf die umgebungsbezogenen Schichten interpretieren konnte, lässt sich InteRRap als eine Spezialisierung der agentenbezogenen Schichten auffassen. Auf beide ist die Referenzarchitektur gut anwendbar.

Goddard Agent Architecture (GAA)

Die Goddard Agent Architecture (Truszkowski 2006; Truszkowski und Rouff 2001) – im folgenden durch „*GAA*“ abgekürzt – wurde am Goddard Space Flight Center der NASA entwickelt, im Rahmen der dortigen Untersuchungen zu den Einsatzmöglichkeiten autonomer Agenten in Raumfahrtmissionen. Die Architektur, die die Erfahrungen aus mehreren entwickelten Multiagentensystemen zusammenfasst, wird als komponentenbasierte modulare Architektur bezeichnet.

Wie die Abbildung 3.18 zeigt, sind die Komponenten durch gerichtete getypte Datenflüsse verbunden. Die Anordnung der Komponenten in der Abbildung folgt dabei nicht der Anordnung in den Original-Publikationen: Für eine vereinfachte Abbildung der GAA-Komponenten auf die Referenzarchitektur wurde zunächst eine Spiegelung an der Horizontalen vorgenommen, da in der GAA unüblicherweise der Abstraktionsgrad der Komponenten nach unten *zunimmt*.

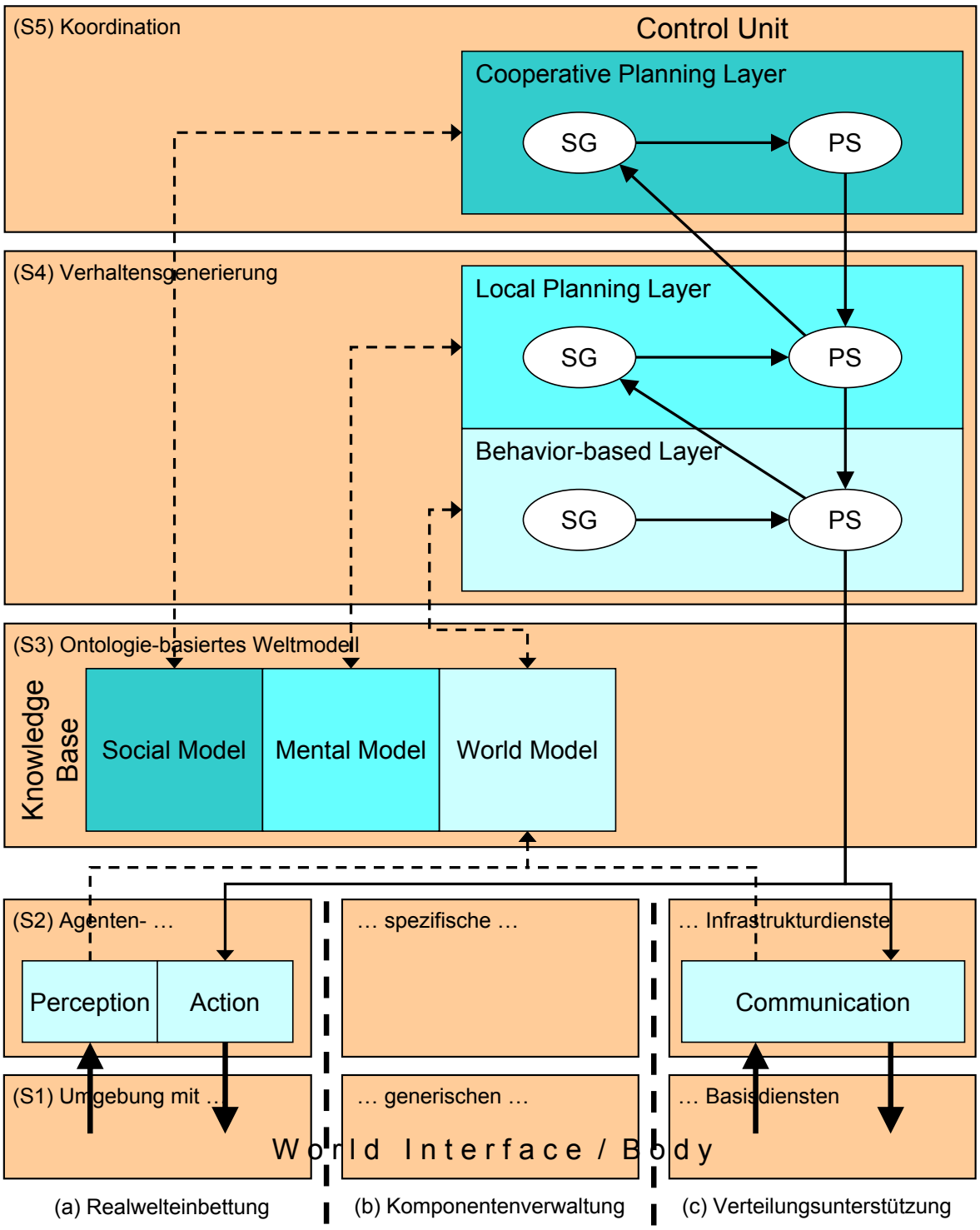


Abbildung 3.17: InteRRap und die Referenzarchitektur

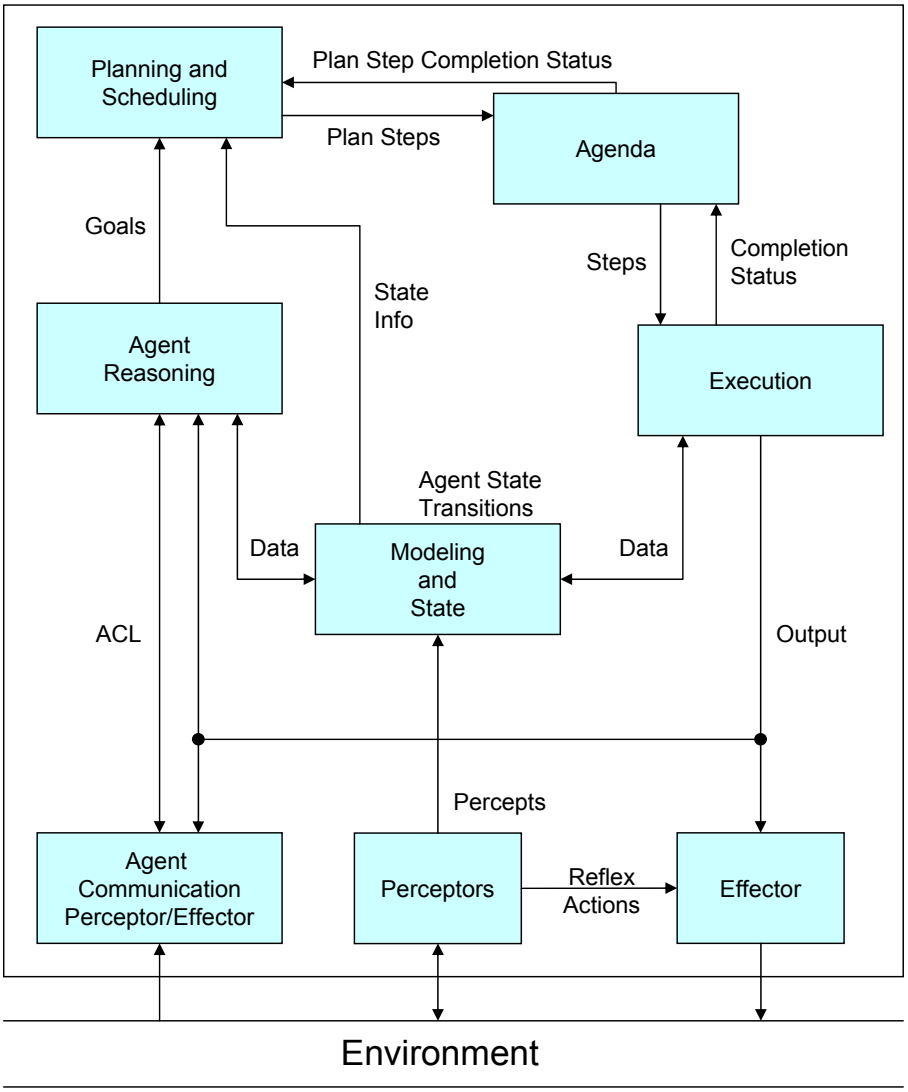


Abbildung 3.18: Die Goddard Agent Architecture (GAA)

Über die Umgebung (*Environment*) auf unterster Ebene macht die GAA keine weiteren Annahmen. Aus den Komponenten, die auf sie zugreifen, lässt sich jedoch ableiten, dass sie sowohl die Anwendungsumgebung (S1a), als auch die anderen Agenten (S1c) umfasst. Rechnerinfrastruktur und Agentenplattform (b) spielen hingegen in der GAA keine Rolle. Die anderen beiden Säulen werden auch auf der Schicht der agentenspezifischen Infrastrukturdienste fortgesetzt: die *Agent Communication*-Komponente verkörpert die Aspekte des Nachrichtentransports (S2c) und die Wahrnehmungen und Handlungen (S2a) werden von den Komponenten *Perceptors* und *Effector* übernommen.

Ein Teil des Ontologie-basierten Weltmodells (S3) wird dem GAA-Agenten durch die *Modeling and State*-Komponente zur Verfügung gestellt. Darüber hinaus können alle Komponenten auch selbst für sie relevante Ausschnitte des Weltmodells verwalten.

Die Schicht S4 zur Verhaltensgenerierung ist in der Referenzarchitektur unterteilt in Zielabwägung und Aktionsauswahl. Diese Unterteilung findet sich auch in der GAA wieder. Das *Agent Reasoning* wählt die aktuell zu verfolgenden Ziele aus, für deren Verfolgung das *Planning and Scheduling* einen Plan erstellt. Die Ausführung der einzelnen Schritte des Plans erfolgt durch die Komponenten *Agenda* und *Execution*. Die Koordinationsschicht S5 findet sich in der GAA nur implizit im *Agent Reasoning* wieder.

Insgesamt lässt sich die GAA auf natürliche Weise auf die Referenzarchitektur abbilden, obwohl von den Entwicklern der GAA dem Schichtungs-Prinzip nicht explizit gefolgt wurde. Abbildung 3.19 zeigt das Resultat. Die zahlreichen Konnektoren, die zum Teil auch durch ganze Schichten hindurchgreifen, sind auf die (redundante) Haltung Komponenten-lokaler und globaler Zustände zurückzuführen.

3.4.3 Zusammenfassung der Bewertung

Ein erster Teil der Hauptaufgabe der vorliegenden Arbeit ist die Systematisierung der Agententechnologie zur Identifikation aller wichtigen Aspekte der Agentenprogrammie-

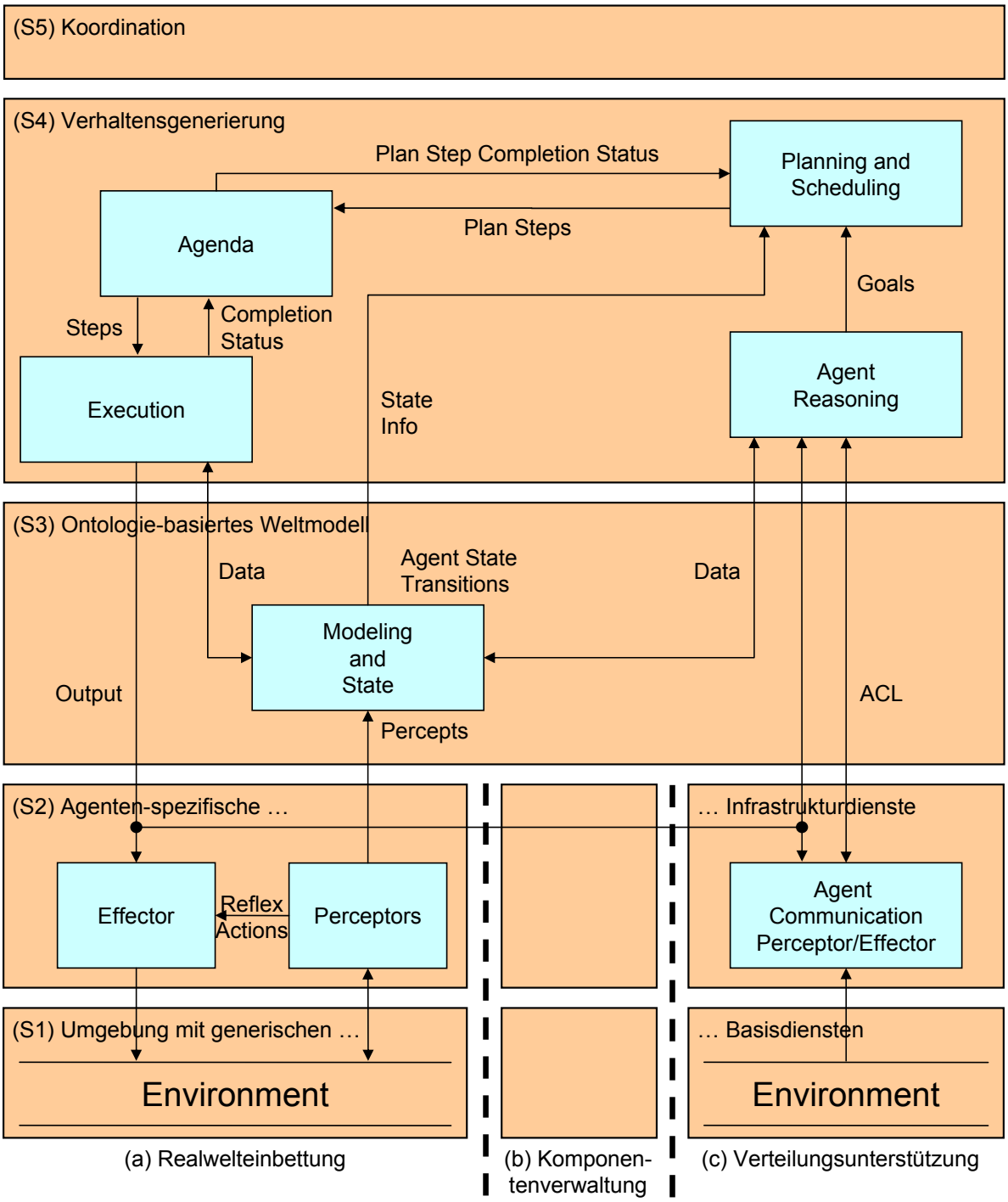


Abbildung 3.19: Goddard Agent Architecture abgebildet auf die Referenzarchitektur

rung und ihres Zusammenspiels. Diese wird mithilfe einer allgemeingültigen und vollständigen Referenzarchitektur für Multiagentensysteme erreicht.

Die Allgemeingültigkeit der Referenzarchitektur liegt in ihrem Hauptfundament, nämlich den definierenden Eigenschaften intelligenter Agenten begründet: keine allgemeinere Grundlage einer Referenzarchitektur als die Definition der Klasse der Systeme, die sie beschreibt, ist vorstellbar. Durch die Diskussion der Kompatibilität der Referenzarchitektur mit weiteren möglichen Eigenschaften der Agenten (Abschnitt 3.4.1) konnte die Allgemeingültigkeit weiter untermauert werden.

Die Untersuchung der Anwendbarkeit der eigenen Referenzarchitektur erfolgt über deren Konformität mit verschiedenen bedeutenden Standardarchitekturen, Referenzarchitekturen und konkreter Architekturen (Abschnitt 3.4.2). Im Bezug auf die betrachteten Architekturen ergab sich die Vollständigkeit der Referenzarchitektur. Alle Elemente der untersuchten Architekturen lassen sich auf natürliche Weise auf Komponenten der Referenzarchitektur abbilden.

Die Schwerpunkte in den untersuchten Architekturen schlagen sich jeweils in Gruppen detailliert ausgearbeiteter Komponenten nieder, die sich in der Referenzarchitektur generischeren Komponenten zuordnen lassen. Umgekehrt werden die in den spezialisierten Architekturen vernachlässigten Komponenten von der Referenzarchitektur ergänzt. Für eine sehr heterogene Auswahl an Architekturen konnte somit die Konformität festgestellt und damit die Anwendbarkeit und Vollständigkeit der Referenzarchitektur belegt werden. Von der breiten Anwendbarkeit der Referenzarchitektur wird im weiteren Verlauf der Arbeit profitiert, weil sie die Übertragbarkeit der an der Referenzarchitektur gewonnenen Erkenntnisse auf konkrete Architekturen ermöglicht.

Die hiermit abgeschlossene Aufbereitung und Strukturierung der Agententechnologie wurde durchgeführt, um eine Grundlage für eine Verortung von Störungen und zugehörigen Zuverlässigkeitsmechanismen in Multiagentensystemen zu schaffen. Im nächsten Kapitel ist nun zu zeigen, dass sich der Zuverlässigkeitsaspekt orthogonal in die Referenzarchitektur einbetten lässt. Im Erfolgsfall erhält man eine *Zuverlässigkeitsreferenzarchitektur*, die eine systematische Verbesserung der Zuverlässigkeit von Multiagentensystemen erlaubt.

Kapitel 4

Eine Zuverlässigkeitsreferenzarchitektur für Multiagentensysteme

Die Referenzarchitektur für intelligente Multiagentensysteme, die in Kapitel 3 entwickelt wurde, systematisiert die Agententechnologie, indem sie die verschiedenen Aspekte der Agentenprogrammierung verortet und zueinander in Beziehung setzt. Die Entwicklung der Architektur erfolgte entlang grundlegender funktionaler Eigenschaften der Agenten und ignorierte folglich den Zuverlässigkeitsaspekt. Im vorliegenden Kapitel soll nun der Zuverlässigkeitsaspekt in die Referenzarchitektur eingearbeitet werden und diese damit zu einer *Zuverlässigkeitsreferenzarchitektur für Multiagentensysteme* ausgebaut werden.

Die Einarbeitung soll dabei derart erfolgen, dass die resultierende Zuverlässigkeitsreferenzarchitektur eine gezielte Verortung von Störungen erlaubt. Daraus soll dann systematisch abgeleitet werden, an welchen Stellen die zugehörigen Zuverlässigkeitsmechanismen einzubringen sind. Diese Vorgehensweise soll das orthogonale Aufprägen der Mechanismen sicherstellen. Ergebnis ist dann eine Zuverlässigkeitsreferenzarchitektur, die aus Sicht des Agenten-Entwicklers eine Steigerung der Zuverlässigkeit bei nur geringfügigen Eingriffen in die eigene Implementierung der Anwendungsfunktionalität zulässt, wie sie bislang nur selten anzutreffen ist. Im Erfolgsfall bildet sie eine *Basis zur Entwicklung eigener neuer orthogonaler Zuverlässigkeitsmechanismen.*

Die Erstellung der Zuverlässigkeitsreferenzarchitektur hat zugleich einen weiteren positiven Nebeneffekt: Für die Zuverlässigkeitsmechanismen für Multiagentensysteme gibt es eine große Auswahl möglicher Ansatzpunkte und den bekannten Mechanismen liegen sehr unterschiedliche Fehlermodelle für die adressierten Störungen zugrunde, wodurch sich die Landschaft der Zuverlässigkeitsmechanismen dem Betrachter bislang sehr heterogen darstellt. Durch die Theoriebildung zur Fehlerbehandlung bei Multiagentensystemen und die exemplarische Verortung bedeutender existierender Zuverlässigkeitsmechanismen, die in der Bewertung der Zuverlässigkeitsreferenzarchitektur in den Kapiteln 5 und 6 erfolgt, wird eine *Homogenisierung der Landschaft der Zuverlässigkeitsmechanismen* erreicht, die dem Agenten-Entwickler die Einordnung auftretender Störungen und – falls vorhanden – die Auswahl eines geeigneten Behandlungsmechanismus erlaubt.

Das vorliegende Kapitel 4, in dessen Rahmen die Zuverlässigkeitsreferenzarchitektur entwickelt wird, gliedert sich wie folgt: Abschnitt 4.1 gibt eine kurze Einführung in die Zuverlässigkeitsbetrachtung von Software-Architekturen, die den theoretischen Hintergrund der Zuverlässigkeitsreferenzarchitektur bildet und diese weiter motiviert. Für eine gemeinsame Behandlung der existierenden heterogenen Zuverlässigkeitsmechanismen im Rahmen der Zuverlässigkeitsreferenzarchitektur ist zunächst die Entwicklung eines abstrahierten Fehlermodells und eines Fehlerbehandlungsmodells erforderlich. Diese erfolgt in Abschnitt 4.2 und liefert eine Taxonomie der Fehlerbehandlung und den Grundstein der orthogonalen Zuverlässigkeitsintegration. Zur Erweiterung der geschichteten Referenzarchitektur bedarf es der Übertragung des Fehler- und des Fehlerbehandlungsmodells auf geschichtete Software-Architekturen in Abschnitt 4.3. Die Umsetzung der allgemeingültigen Zuverlässigkeitsreferenzarchitektur wird in Abschnitt 4.4 ausgeführt durch eine schichtenweise Einarbeitung potenzieller Störungen und durch die Verortung der entsprechenden Fehlerbehandlungsoptionen in die Referenzarchitektur.

4.1 Zuverlässigkeitsbetrachtungen bei Software-Architekturen

Die Entwicklung der Referenzarchitektur im vorangegangenen Kapitel 3 hat sich an den definierenden Eigenschaften der intelligenten Agenten orientiert, die beschreiben, welche (abstrakten) Dienste intelligente Agenten erbringen und wie sie dies bewerkstelligen. Reussner und Hasselbring (2006, Seite 3) unterstreichen die wichtige Rolle, die die qualitativen („nichtfunktionalen“) Anforderungen neben den funktionalen Anforderungen beim Architekturentwurf spielen. Allerdings gilt:

> „Die systematische Behandlung von Qualitätsanforderungen beim Architekturentwurf ist noch Gegenstand der Forschung; ein erster Schritt in diese Richtung stellen Architekturbewertungsverfahren dar. Diese Verfahren schätzen basierend auf der Software-Architektur die nichtfunktionalen Eigenschaften eines Systems ab. Auch wenn zu einem so frühen Zeitpunkt im Software-Entwicklungsprozess wegen fehlender Detailinformation, die erst in der Implementierungsphase zur Verfügung steht, noch keine absoluten Aussagen über die Systemeigenschaften [...] gemacht werden können, so sind diese Verfahren doch hilfreich bei der Bewertung von Architekturalternativen.“

Die Qualitätsanforderung, die in dieser Arbeit im Mittelpunkt des Interesses steht, ist die Zuverlässigkeit bei Störungen. Es gibt eine Reihe von Architekturbewertungsverfahren auf Basis mathematischer Modelle, die sich mit diesem Merkmal beschäftigen, d.h. die eine analytische Abschätzung der Zuverlässigkeit von Software-Architekturen ermöglichen sollen. Gute Einführungen und Überblicke über die Verfahren geben Eusgeld, Freiling und Reussner (2006), Goseva-Popstojanova und Trivedi (2001) und Dimov und Punnekkat (2005).

Sehr viele der Verfahren basieren auf dem Modell von Cheung (1980) und erweitern dieses für spezifische Einsatzzwecke. Die Idee hinter dem Modell von Cheung – und damit hinter vielen anderen Modellen – ist die Abbildung der betrachteten Software-Architektur auf Markov-Ketten. Dabei wird zunächst der Kontrollfluss zwischen den Komponenten einer Architektur modelliert, indem die Komponenten zu Knoten des Graphen werden und gewichtete Kanten die Übergangswahrscheinlichkeiten p_{ij} des Kontrollflusses von

jeweils einer Komponente i auf eine andere Komponente j darstellen. In jedem Kontrollfluss eines terminierenden Programms gibt es einen Eingangsknoten („1“) und einen Endknoten („n“).

Im zweiten Schritt wird jeder Komponente i ein Verlässlichkeitswert R_i zugewiesen und die Menge der Knoten um die beiden Finalzustände C und F ergänzt, die für ein erfolgreiches („correct“) bzw. fehlerhaftes („failed“) Programmende stehen. Die ursprünglichen Kanten werden nun entsprechend angepasst. Die Übergangswahrscheinlichkeit von Komponente i zu Komponente j wird zu $p_{ij}R_i$ modifiziert und unter der Annahme, dass der Fehler einer Komponente zum Gesamtversagen des Systems führt, lässt sich die Übergangswahrscheinlichkeit eines Zustands i in den Finalzustand F berechnen zu $1 - R_i$. Von der Komponente n wird außerdem eine neue Kante mit der Übergangswahrscheinlichkeit R_n zum Finalzustand C eingefügt. Die Zuverlässigkeit eines Systems lässt sich nun berechnen als die Wahrscheinlichkeit, mit der der Zustand C des Graphen erreicht wird. Zur Handhabung dieser Berechnung lässt sich aus dem resultierenden Graphen eine Übergangswahrscheinlichkeitsmatrix (engl. *Transistion Probability Matrix, TPM*) ableiten, mit deren Hilfe sich durch elementare Matrixumformungen einfach die Zuverlässigkeit bestimmen lässt.

Obwohl die mathematischen Grundlagen für die Architekturbewertungsverfahren seit geraumer Zeit sehr ausführlich erforscht werden, gibt es noch recht wenige Arbeiten, die die Ergebnisse direkt für Handlungsanweisungen in den Architekturentwicklungsprozess einbeziehen. Dies ist zurückzuführen auf die vereinfachenden Annahmen der mathematischen Modelle und die wenigen verfügbaren Detailinformationen in der Architekturentwicklung als früher Phase der Software-Entwicklung. Noch schwieriger ist eine Anwendung der Verfahren auf den Prozess der Referenzarchitekturerstellung, wie er im vorherigen Kapitel geschildert wurde: Die Verfahren funktionieren am besten für die vergleichende Bewertung möglichst konkreter Alternativen des Architekturentwurfs, also für einen Vorgang, der eher beim Instanziierungszeitpunkt von Referenzarchitekturen anzusiedeln ist.

Daher lassen sich nur einige wenige grundsätzliche Aussagen über die Referenzarchitektur ableiten. Hilfreich sind hierbei die Arbeiten von Wang, Wu und Chen (1999), die die Zuverlässigkeitseigenschaften verschiedener Architekturmuster auf der Basis von

(Cheung 1980) untersuchen. Dabei zeigt sich, dass das Schichtung-Architekturmuster im Bezug auf die Zuverlässigkeit ähnlich zu bewerten ist wie die anderen in Abschnitt 3.3.1 ausführlich besprochenen Architekturmuster. Außerdem wird eine Reihe intuitiver Annahmen über den Zusammenhang zwischen Architekturen und Zuverlässigkeit bestätigt: redundante Komponenten können die Zuverlässigkeit erhöhen; Parallelausführung nichtredundanter Komponenten kann die Zuverlässigkeit verringern; die Zuverlässigkeit von häufig vom Kontrollfluss durchlaufenen Komponenten (sog. „*Hotspots*“) ist für die Zuverlässigkeit des Gesamtsystems besonders kritisch.

Zusammenfassend ergeben sich für die Zuverlässigkeitsreferenzarchitektur damit drei direkte Konsequenzen:

1. Grundsätzlich sind geschichtete Referenzarchitekturen für die Ableitung zuverlässiger Software-Systeme geeignet. Damit ist es gerechtfertigt, die geschichtete Referenzarchitektur dieser Arbeit zu einer geschichteten Zuverlässigkeitsreferenzarchitektur auszubauen.

2. Zur Steigerung der Zuverlässigkeit eines Gesamtsystems ist die komponentenweise Betrachtung der Zuverlässigkeitsaspekte ein erfolgversprechender und etablierter Ansatz. Dies untermauert insbesondere die Wichtigkeit der „Verortung“ existierender Zuverlässigkeitsmechanismen, also des Festmachens von Störungen und geeigneten Gegenmaßnahmen an den einzelnen Komponenten einer Architektur.

3. Die herausgehobene Rolle der Hotspots für die Zuverlässigkeit eines Gesamtsystems bedeutet übertragen auf die Schichten der Referenzarchitektur, dass der Zuverlässigkeit der unteren Schichten besondere Aufmerksamkeit zukommen sollte, da ein Dienstaufruf auf einer höheren Schicht häufig in mehrere Dienstaufrufe auf tieferen Schichten umgesetzt wird. Als Beispiel sei eine Anfrage nach dem aktuellen Zustand eines Artefakts des Weltmodells genannt, die auf eine Reihe zusammengehöriger Datenbankanfragen umgesetzt werden kann. Folglich sind Zuverlässigkeitsmechanismen, die die Störungen tieferliegender Schichten behandeln – also die sog. Infrastrukturstörungen –, von großer Bedeutung.

Vor diesem Hintergrund wird nun in den folgenden Abschnitten die Referenzarchitektur für intelligente Agentensysteme zur Zuverlässigkeitsreferenzarchitektur ausgebaut,

beginnend mit einer Taxonomie in Form eines abstrakten Fehlermodells und Fehlerbehandlungsmodells.

4.2 Eine Taxonomie für Zuverlässigkeitsmechanismen

In der Motivation der Zuverlässigkeitsmechanismen (Kapitel 2) wurden beispielhafte Fehlerfälle beschrieben und lediglich gesagt, dass diese von verschiedenartigen Störungen hervorgerufen werden können. Die Mechanismen, die im Laufe dieser Arbeit entwickelt werden, wenden sich gegen Partner- und Infrastrukturstörungen. Allerdings wurde das Wesen der Störungen noch nicht näher untersucht; insbesondere ist noch offen, wie sie entstehen und wie die Mechanismen ihnen entgegenwirken. In den folgenden beiden Unterabschnitten wird dazu ein abstraktes Fehler- und Fehlerbehandlungsmodell entwickelt und damit die Taxonomie für die weitere Arbeit eingeführt. Diese Modelle sollen eine einheitliche Handhabung der heterogenen Zuverlässigkeitsmechanismen und deren Anbindung an die Komponenten der Referenzarchitektur erlauben.

4.2.1 Ein abstraktes Fehlermodell

Das abstrakte Fehlermodell basiert auf den folgenden Definitionen von Laprie (1985, 1992).

- Die *Zuverlässigkeit* (engl. *dependability*) eines Systems beschreibt dessen Vertrauenswürdigkeit. Falls es gerechtfertigt erscheint, Vertrauen in die Dienste zu setzen, die das System erbringt, wird ein System als zuverlässig bezeichnet.

- Ein *Dienst* (engl. *service*) ist das nach außen beobachtbare Verhalten eines Systems, für das ein Kontrakt mit dem Dienstnutzer in Form einer *Dienstspezifikation* besteht. Dabei handelt es sich um eine einvernehmliche Beschreibung über die von beiden Seiten im Rahmen der Dienstnutzung zu erbringenden Leistungen. Der bis hierher als System bezeichnete Diensterbringer kann durch seine Beziehung zum Dienstnehmer selbst Teil eines übergeordneten Gesamtsystems sein. In diesem Zusammenhang bezeichnet man den Diensterbringer häufig als *Komponente*. (Diese

Definitionen nach Laprie sind im Einklang mit der obigen Verwendung der Begriffe.)

- Entspricht der erbrachte Dienst nicht der vereinbarten Dienstspezifikation, spricht man von einem *Ausfall* (engl. *failure*) der Komponente bzw. des Dienstes.

Der Ausfall eines Dienstes ist die nach außen beobachtbare Folge eines Ereignisses, das im Inneren des Diensterbringers aufgetreten ist (und das oben vereinfachend als Störung bezeichnet wurde).

- Der Grund eines Ausfalls ist – im phänomenologischen Sinne – eine sog. *Fehlerursache* (engl. *fault*).
- Eine Fehlerursache führt zu einem *latenten Fehlzustand* (engl. *latent error*).
- Ein latenter Fehlzustand wird zu einem *effektiven Fehlzustand* (engl. *effective error*), wenn er im Rahmen der Diensterbringung *aktiviert* wird, was zu einem Ausfall führt.

Zur Illustration: Der Bruch eines Glasfaserkabels ist eine Fehlerursache. Die Folge ist ein latenter Fehlzustand (der Infrastruktur) im Multiagentensystem, nämlich z.B. die Abtrennung eines bestimmten Rechnerknotens vom Netzwerkverbund. Der latente Fehlzustand wird effektiv, wenn ein Agent versucht, einen Dienst eines Agenten auf dem abgetrennten Rechnerknoten in Anspruch zu nehmen.

Im *Vorfeld des Einsatzes* soll sorgsame Softwareentwicklung dafür sorgen, dass sich die Fehlerursachen in Komponenten in Grenzen halten. Die Menge aller Maßnahmen, die zur Reduzierung von Fehlerursachen beitragen, wird als *Fehlervermeidung* (engl. *fault avoidance*) bezeichnet. So könnten beispielweise strukturierte Testprozeduren oder eine verbesserte Programmierausbildung angebrachte Fehlervermeidungsmaßnahmen sein (Timm u. a. 2006). Im *Betrieb* sind latente Fehlzustände nicht erkennbar, weshalb lediglich den effektiven Fehlzuständen eine *Fehlerbehandlung* (engl. *error processing*) zuteil werden kann.

Das abstrakte Fehlermodell muss also Folgendes erfassen: den Diensterbringer selbst, die effektiven Fehlzustände, die im Betrieb auftreten, welche davon wie behandelt wer-

den und wie ggf. ein Ausfall nach außen sichtbar wird (siehe dazu Abbildung 4.1).

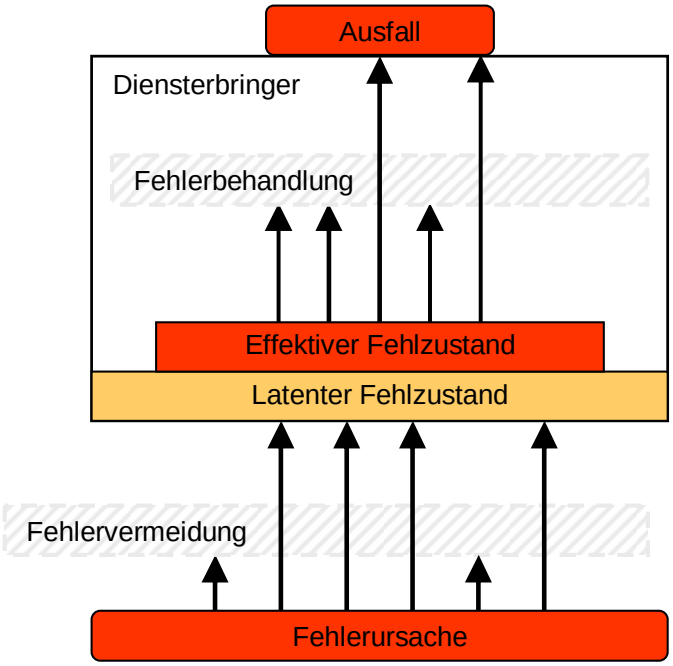


Abbildung 4.1: Abstraktes Fehlermodell: Fehlerursache, Fehlzustand und Ausfall

4.2.2 Fehlerbehandlungsmodell für unbeabsichtigte Fehler

Das obige abstrakte Fehlermodell bezieht sich auf die Fehlerbehandlung für unbeabsichtigte Fehler. Ausgehend von diesem Fehlermodell wird im Folgenden ein Fehler*behandlungs*modell für Diensterbringer abgeleitet.

Wird beim Auftreten eines effektiven Fehlzustandes eine Fehlerbehandlung angestoßen, muss diese nicht zwangsläufig von (umfassendem) Erfolg gekrönt sein. Vielmehr hängt sie von Art und Schwere des effektiven Fehlzustandes und damit auch von der Fehlerursache ab und beeinflusst ihrerseits das Ergebnis der Diensterbringung, das vom Dienstnehmer wahrgenommen wird. Aus seiner Sicht lassen sich je nach Fehlerhaftigkeit der Diensterbringung mehrere Klassen von Ergebniszuständen unterscheiden (Abbildung 4.2).

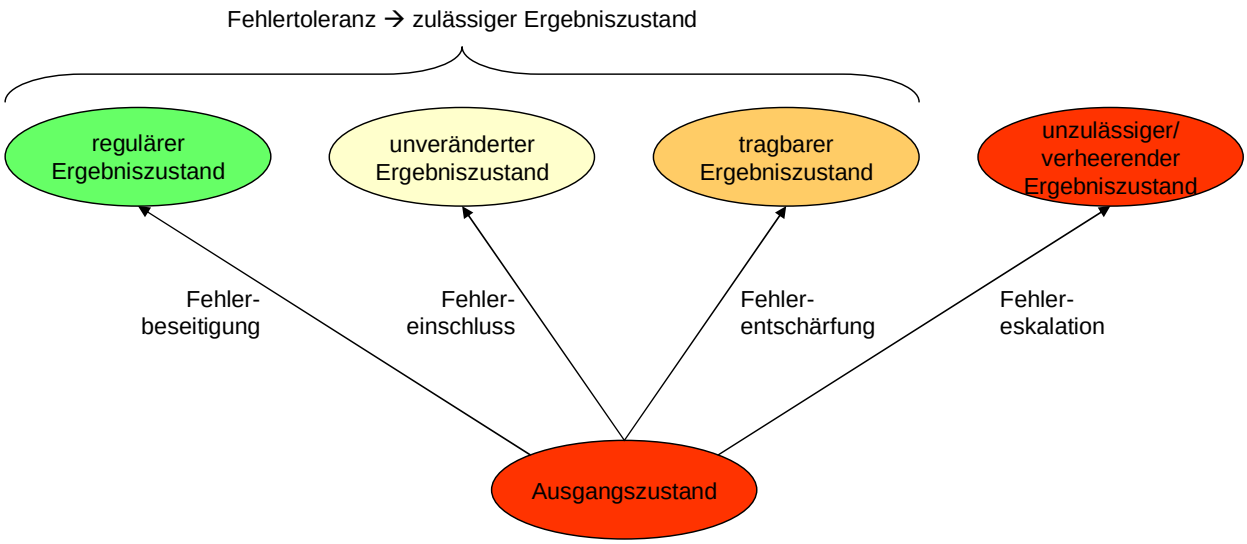


Abbildung 4.2: Ergebniszustand nach der Diensterbringung

Im günstigsten Fall tritt bei der Diensterbringung kein Fehler auf und der *reguläre Ergebniszustand* wird erreicht. Oder es tritt ein Fehler auf, der aber so gut behandelt werden konnte, dass der vom Dienst zurückgelieferte Ergebniszustand mit dem Ergebniszustand der fehlerfreien Diensterbringung identisch ist.

Eine andere mögliche Reaktion des Diensterbringers auf einen effektiven Fehlzustand kann das vollständige Ignorieren der Dienstanforderung sein. Für den Dienstnehmer sind der Ausgangszustand und der *unveränderte Ergebniszustand* (semantisch) äquivalent.

Anstelle des vollständigen Ignorierens kann der Diensterbringer auf einen effektiven Fehler auch reagieren, indem er die bestmögliche Diensterbringung versucht. Das Ergebnis entspricht dann eventuell nicht der Dienstspezifikation, kann dem Dienstnehmer aber möglicherweise dennoch eine sinnvolle Fortführung seiner Arbeit ermöglichen. Für den Dienstnehmer ergibt sich folglich ein *tragbarer Ergebniszustand*.

Die drei Klassen regulärer, unveränderter und tragbarer Ergebniszustand werden zusammengefasst zur Klasse der *zulässigen Ergebniszustände*. Garantiert ein Dienst stets das Erreichen eines zulässigen Ergebniszustands, werden wir ihn in dieser Arbeit als *zuverlässig* bezeichnen.

Im ungünstigsten Fall kann der Diensterbringer keinen der drei zulässigen Ergebniszustände erreichen und gibt einen entsprechend der Dienstspezifikation *unzulässigen Ergebniszustand* zurück oder – noch schlimmer – einen *verheerenden Ergebniszustand*, der sogar den Ausfall des Gesamtsystems verursacht.

Zur Unterscheidung des Verhaltens dienstgebender Komponenten lassen sich den obigen Ergebniszustandsklassen Verhaltensklassen zuordnen:

- *Fehlertoleranz* (engl. *fault tolerance*) bedeutet, dass ein zulässiger Ergebniszustand erreicht wird. Genauer lässt sich unterscheiden zwischen
  - *Fehlerbeseitigung* (engl. *fault resilience*), wenn der reguläre Ergebniszustand erreicht wird,
  - *Fehlereinschluss* (engl. *fault containment*), wenn der unveränderte Ergebniszustand zurück gegeben wird und
  - *Fehlerentschärfung* (engl. *fault mitigation*), wenn der Ergebniszustand tragbar ist.
- Als *Fehlereskalation* (engl. *error escalation*) werden alle anderen Fälle zusammengefasst, d.h. die Fälle, in denen ein effektiver Fehlzustand zu einem unzulässigen oder verheerenden Ergebniszustand führt.
- Die *Fehlerweitergabe* (engl. *error exposure*) bezeichnet als Überbegriff alle Fälle, in denen die dienstgebende Komponente einen anderen als den regulären Ergebniszustand zurück gibt.

4.3 Prinzip der Fehlerbehandlung in geschichteten Architekturen

Das beschriebene Fehlerbehandlungsmodell und das zugrunde liegende Fehlermodell werden nun auf die geschichtete Referenzarchitektur aus Kapitel 3 übertragen.

4.3.1 Fehlerbehandlung in einer Schicht

Zur Übertragung wird in einem ersten Schritt ein einzelner Dienstgeber betrachtet, dessen Dienste von einem Dienstnehmer in Anspruch genommen werden und der zur Diensterbringung selbst wiederum die Dienste anderer Dienstgeber einsetzt. Abbildung 4.3 zeigt den Dienstgeber, in den das Fehlermodell und das Fehlerbehandlungsmodell integriert wurden, und das Zusammenspiel mit dem zugehörigen Dienstnehmer.

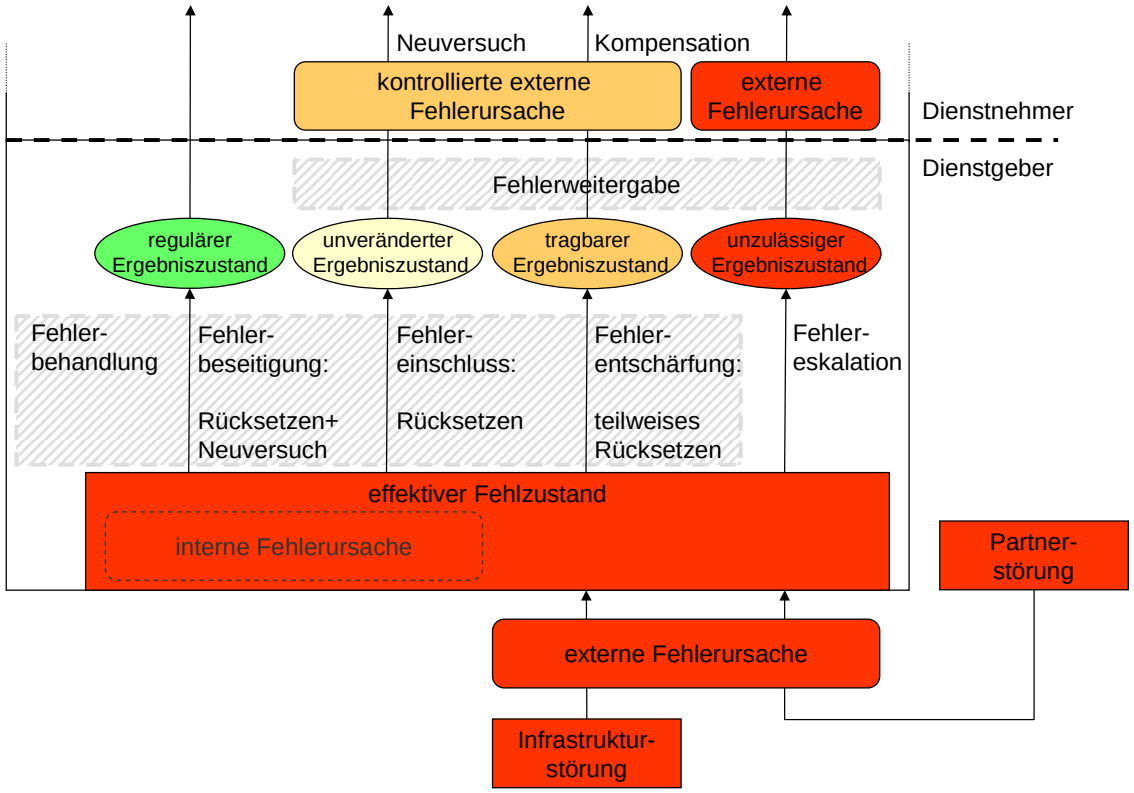


Abbildung 4.3: Fehlerbehandlung eines einzelnen Dienstgebers

Eine vollständige Fehlerbeseitigung fällt ausschließlich in den Bereich des Dienstgebers. Sie beruht häufig auf *Rücksetzen* (engl. *Recovery*) und *Neuversuch* (engl. *Retry*) evtl. unterstützt durch die Replikation von Teilfunktionalitäten. Der Fehlereinschluss beschränkt sich auf das Zurücksetzen im Dienstgeber. Die Fehlerentschärfung braucht auf Dienstgeber-Seite zumindest ein teilweise erfolgreiches Zurücksetzen.

Fehlereinschluss, Fehlerentschärfung und Fehlereskalation führen zu Ergebniszuständen, die auf Seite des Dienstnehmers weitere Maßnahmen erfordern. Im Falle des Fehlereinschlusses kann der Dienstnehmer seine Arbeit wieder aufnehmen und nach in Frage kommenden Ersatzdienstgebern suchen. Bei einer Fehlerentschärfung kann es für den Dienstnehmer erforderlich werden, durch *Kompensation* (engl. *Compensation*) einen Zustand zu erreichen, von dem aus sein Normalbetrieb wieder aufgenommen werden kann. Die Fehlerweitergabe führt beim Dienstnehmer selbst zu einem effektiven Fehlzustand; dieser muss folglich seine eigene Fehlerbehandlung anstoßen.

4.3.2 Fehlerweitergabe in verteilten geschichteten Systemen

Das detaillierte Fehlerbehandlungsmodell aus Abschnitt 4.2.2 bezog sich auf das Zusammenspiel eines Dienstnehmers mit seinem von Fehlzuständen geschädigten Dienstgeber. Es zeigte sich, dass sich die Fehlerbehandlung nicht auf den betroffenen Dienstgeber beschränken lässt, sondern dass in bestimmten Fällen der Dienstnehmer in den Fehlerbehandlungsprozess mit einbezogen werden muss. Dies eröffnet für geschichtete Architekturen zusätzliche Möglichkeiten: Der Dienstnehmer gehört einer semantisch höherstehenden Schicht an und verfügt damit über einen breiteren Kontext und mächtigere Verhaltensweisen als die Dienstgeberschicht. Der nicht regulär erbrachte Dienst kann für die Dienstnehmerschicht beispielsweise nur ein austauschbarer Schritt in einer Abfolge von Aktionen sein, oder die angefragte Information bildet nur einen kleinen Ausschnitt im Datenbestand der dienstnehmenden Schicht.

In verteilten geschichteten Systemen ergeben sich damit die in Abbildung 4.4 gezeigten sechs Fehlerbehandlungsszenarien.

(1) Es findet eine schichteninterne Fehlerbeseitigung statt, d.h. der Ort der Fehlerursache ist zugleich Ort der Fehlerbeseitigung. Nach außen wird nicht sichtbar, dass eine Störung auftrat und erfolgreich behandelt wurde.

(2) Der Dienstgeber liefert bei der Erbringung seines Dienstes einen fehlerbehafteten Zustand zurück, der aber in der nächsthöheren dienstnehmenden Schicht erfolgreich behandelt werden kann. Außer der dienstgebenden und -nehmenden Komponente werden keine weiteren Komponenten involviert.

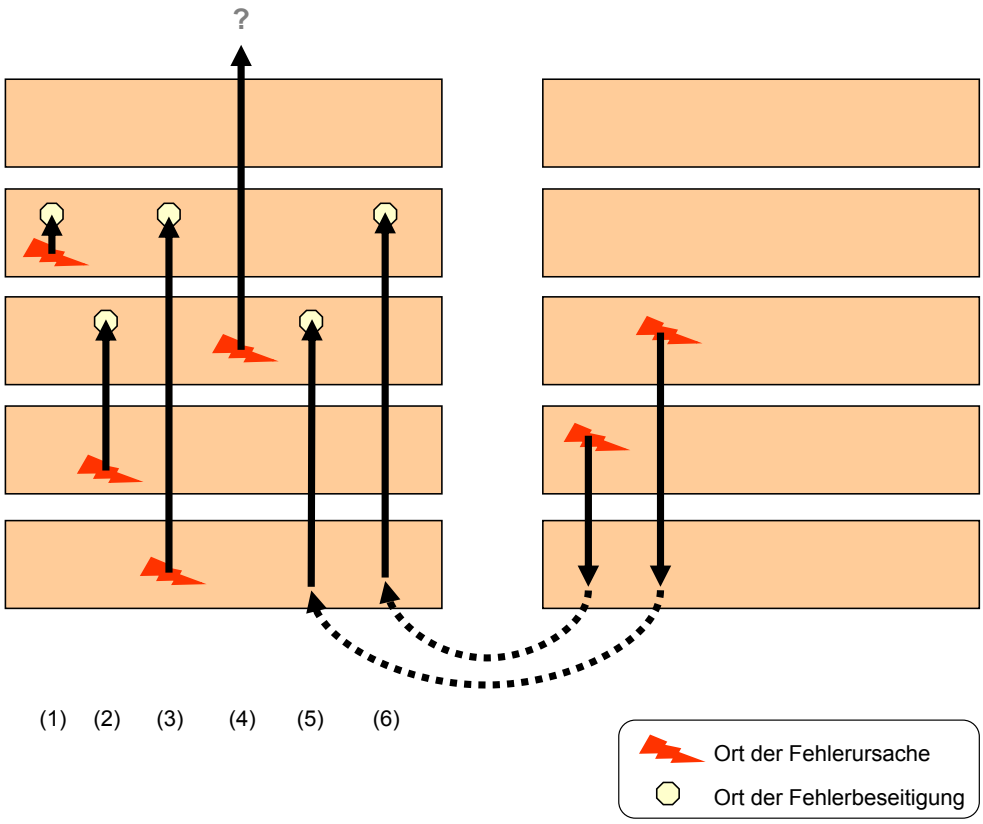


Abbildung 4.4: Fehlerbehandlungsszenarien in einem verteilten geschichteten System

(3) Ein Dienstgeber n erbringt seinen Dienst fehlerhaft und die nächsthöhere Schicht *n+1* behandelt den Fehler nicht erfolgreich. Dabei sind zwei Fälle zu unterscheiden: Die dienstnehmende Schicht *n+1 erkennt* zwar den Fehler, verfügt jedoch nicht über die geeigneten Mittel zur Fehlerbehandlung – es kann aber eine Fehlersignalisierung an den eigenen Dienstnehmer *n+2* erfolgen, oder die dienstnehmende Schicht *n+1* kann den Fehler *nicht erkennen* und erbringt ihren Dienst (scheinbar) regulär. In beiden Fällen wird der Ausfall der Schicht n zur Fehlerursache für die indirekt dienstnehmende Schicht *n+2*. Er durchläuft damit eine Schicht – oder im Wiederholungsfall auch mehrere Schichten, bevor er erfolgreich behandelt werden kann.

(4) Falls in keiner der höheren Schichten eine erfolgreiche Fehlerbeseitigung durchgeführt werden kann, schlägt der Fehler bis zum Anwender durch, der ggf. geeignete Maßnahmen ergreifen muss.

(5) Aus Sicht des Fehlerbehandlungsorts liegen bei den bisher geschilderten Szenarien Infrastrukturstörungen vor. In verteilten geschichteten Systemen kommen noch die Partnerstörungen, also Störungen mit dem Ort der Fehlerursache im Schichtenstapel der dienstgebenden Partnerinstanz, hinzu. Die resultierenden Szenarien sind analog zu denen der Infrastrukturstörungen. Im ersten Fall können die Partnerstörungen auf Seiten der dienstnehmenden Partnerinstanz in der gleichen Schicht beseitigt werden.

(6) Im zweiten Fall werden die Partnerstörungen in einer der höheren Schichten auf der Seite der dienstnehmenden Instanz beseitigt. Da die Partnerstörung auf ihrem Weg zwischen den beteiligten Instanzen ohnehin die unteren Schichten der beiden Protokollstapel traversiert, muss nicht unterschieden werden, ob die Behandlung direkt in der *nächst*höheren Schicht erfolgt oder in einer noch höheren Schicht.

Das breite Spektrum an Fehlerbehandlungsszenarien, das in verteilten geschichteten Systemen denkbar ist, und insbesondere die zusätzlichen Möglichkeiten der semantisch mächtigeren höheren Schichten zur Fehlerbehandlung verstärken die Motivation, nicht nur die Funktionen im störungfreien Betrieb eines Agenten geschichtet zu gliedern, sondern auch die Fehlerbehandlungsmechanismen entlang der entwickelten geschichteten Referenzarchitektur zu verorten.

4.3.3 Vereinfachte Fehlerbehandlung in einer Schicht

Zur Übertragung des Fehlerbehandlungsmodells auf die geschichtete Referenzarchitektur wird nun eine abstrahierte und auf Schichtung angepasste Version des Fehlerbehandlungsmodells für eine einzelne Schicht (Abschnitt 4.3.1) entwickelt. Das vereinfachte Modell, das in Abbildung 4.5 illustriert wird, stellt zugleich eine Anpassung der sog. Ideal Fault Tolerant Component von Anderson und Lee (1981, Seite 298) dar.

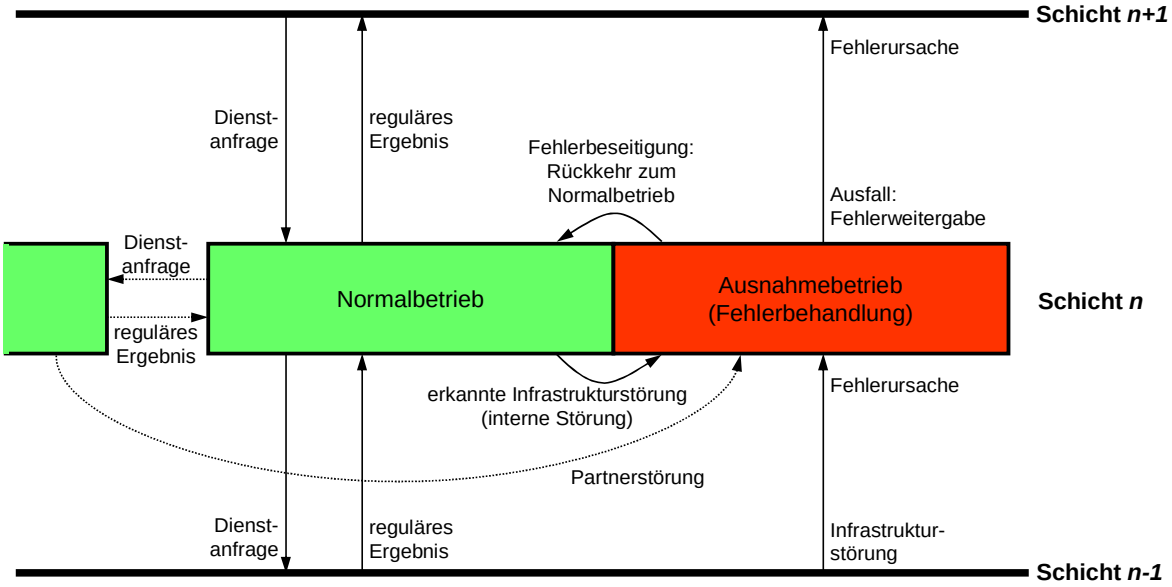


Abbildung 4.5: Abstrahiertes Fehlerbehandlungsmodell für eine einzelne Schicht

Eine Schicht kann sich in einem von zwei Betriebsmodi befinden. Im *Normalbetrieb* nimmt die Schicht Dienstanfragen von höheren Schichten entgegen und versucht sie – ggf. unter Verwendung der Dienste tiefer liegender Schichten – mit einer spezifikationsgemäßen Antwort zu erfüllen. Indirekt, d.h. nach Durchlaufen des unterliegenden Protokollstapels, kann eine Schicht auch Dienstanfragen von anderen Instanzen der eigenen Schicht (*Peers*) bei Partneragenten entgegennehmen oder diese benutzen.

Tritt während einer Diensterbringung im Normalbetrieb ein effektiver Fehlzustand auf, dann wechselt der Betriebsmodus über in den *Ausnahmebetrieb*, in dem versucht wird, den Fehlzustand zu analysieren und zu behandeln. Ziel der Fehlerbehandlung ist die

Fehlerbeseitigung, die eine Rückkehr in den Normalbetrieb und einen erneuten Anlauf zur Diensterbringung ermöglicht. Falls die Fehlerbeseitigung nicht (vollständig) erfolgreich ist, wird der eigene Ausfall an die nächsthöhere dienstnehmende Schicht signalisiert.

Die Signalisierung des Ausfalls an einen Dienstnehmer und auch der Wechsel in den Ausnahmebetrieb innerhalb einer Schicht geschieht typischerweise durch die Erzeugung eines *Ausnahmeereignisses*, die mit der Auslösung einer sogenannten *„Exception“* in der Programmiersprache Java[1] vergleichbar ist. Dieses Modell lässt sich zum Anstoßen des Ausnahmebetriebs für die drei oben eingeführten endogenen Störungsarten einsetzen:

- *Infrastrukturstörungen*: Diese können auf zwei verschiedene Weisen zu einem Wechsel in den Ausnahmebetrieb führen: Fehlzustände, die in niederen Schichten latent – d.h. unbemerkt oder bedeutungslos – geblieben sind, werden von der dienstgebenden Schicht (n-1) als (vermeintlich) reguläres Ergebnis an die dienstnehmende Schicht n übermittelt. Werden sie im Normalbetrieb auf Schicht n erkannt, erfolgt der Wechsel in den Ausnahmebetrieb der Schicht n. Ausserdem können Fehlzustände, die in Schicht (n-1) bereits effektiv wurden, jedoch nicht vollständig behandelt werden konnten, direkt an Schicht n signalisiert werden.

- *Partnerstörungen*: Einen weiteren Anlass, vom Normal- in den Ausnahmebetrieb zu wechseln, stellen Partnerstörungen dar, also Fehlerursachen, die von einer dienstgebenden Instanz der Schicht n herrühren. Wie bei den Infrastrukturstörungen bestehen auch hier die beiden Möglichkeiten, dass diese Fehler erst in der dienstnehmenden Partnerinstanz der Schicht n erkannt werden und dort ein Ausnahmeereignis auslösen oder dass sie bereits in der dienstgebenden Partnerinstanz erkannt und – nach erfolglosem Behandlungsversuch – von dort signalisiert werden. In beiden Fällen erfolgt dann lokal im Ausnahmebetrieb der Versuch einer Fehlerbehandlung.

- *Interne Störungen*: Zur Vollständigkeit sei erwähnt, dass auch bei den – in der Arbeit nicht weiter untersuchten – internen effektiven Fehlzuständen die Schicht n

[1] Siehe `http://java.sun.com`

durch Ausnahmeereignisse vom Normalbetrieb in den Ausnahmebetrieb wechseln kann.

4.4 Die Zuverlässigkeitsreferenzarchitektur

Durch die Einarbeitung des abstrahierten Fehlerbehandlungsmodells aus dem letzten Abschnitt wird die Referenzarchitektur intelligenter Multiagentensysteme nun erweitert um die Aspekte der Fehlerbehandlung in Agentensystemen. Für jede einzelne Schicht wird dabei untersucht, welche effektiven Fehlzustände unabhängig von der Schicht der zugrunde liegenden Fehlerursache auftreten können, welche Optionen zu deren Behandlung die betrachtete Schicht bietet und welche Ausfälle – evtl. abgemildert – an höhere Schichten weitergegeben werden müssen. Die resultierende Zuverlässigkeitsreferenzarchitektur leistet also die separierende Verortung aller Zuverlässigkeitsaspekte.

In den folgenden Unterabschnitten (4.4.1–4.4.6) werden die einzelnen Schichten der Zuverlässigkeitsreferenzarchitektur entwickelt, indem ihre typischen Störungen und Fehlerbehandlungsoptionen zugeordnet werden. Die Schichten werden dafür aus dem Blickwinkel eines Agentensystem-Entwicklers in Reihenfolge wachsender Abstraktion (*„bottom-up“*) besucht. Unterabschnitt 4.4.7 gibt in Abbildung 4.6 einen zusammenfassenden Überblick über die Störungen und die zugehörigen Fehlerbehandlungsoptionen und ihre Zusammenhänge und schließt in Abbildung 4.7 mit der daraus resultierenden vollständigen Zuverlässigkeitsreferenzarchitektur. Für kürzere Darstellungen der Zuverlässigkeitsreferenzarchitektur sei auf (Nimis u. a. 2006; Lockemann und Nimis 2006a, b) verwiesen.

4.4.1 Basisdienste der Systemumgebung (S1)

Schicht S1 umfasst zum einen die typischen generischen Dienste einer Rechnerumgebung, also die Hardware, das Betriebssystem, die Datenhaltung und – bedingt durch die verteilte Natur der Multiagentensysteme – den Datentransport. Zum anderen kommen

durch die Umgebungssituiertheit die Sensoren und Aktoren als Brücke in die Anwendungsumgebung hinzu. Im Sinne der Definition kann es auf Schicht S1 keine Infrastrukturstörungen – Störungen, die ihre Fehlerursache in tiefer liegenden Schichten haben – geben, da hier keine tiefer liegenden Schichten existieren. Es ist also unvermeidlich, auf Schicht S1 neben den Partnerstörungen die – ansonsten nicht zu betrachtenden – internen Störungen zu untersuchen. Diese werden gerade im Falle der Schicht S1 durch Weitergabe zur Fehlerursache für viele in den höheren Schichten zu behandelnde Infrastrukturstörungen.

Rechnerstörungen

Als interne Störungen auf Schicht S1b bedrohen Hardware-Ausfälle, Betriebssystem- und Datenhaltungsfehler die einzelnen Agenten. Gegen diese Störungen bieten sich auf Schicht S1b Fehlerbehandlungsoptionen durch den geschickten Einsatz von Redundanz, insbesondere durch **Hardware-Redundanz**. Die Rechnerinfrastruktur wird nach außen nur dann einen Ausfall zeigen, wenn diese Optionen ins Leere laufen. Im Falle eines nicht-katastrophalen unbehandelten Ausfalls, wie z.B. einer fehlgeschlagenen Speicheroperation, findet eine Propagierung an die nächsthöhere Schicht S2b statt. Im Falle eines katastrophalen Ausfalls, wie z.B. einer Zerstörung der Hardware, die keine digitale Behandlung der Störung erlaubt, kann das komplette System stehen bleiben und erfordert damit möglicherweise einen Benutzereingriff.

Datentransportstörungen

Der Datentransport in Schicht S1c von Multiagentensystemen beruht auf einer Kommunikationsinfrastruktur, wie z.B. dem Internet mit seinen Anwendungsprotokollen HTTP und IIOP, die selbst wiederum auf TCP/IP aufbauen. Störungen des Datentransports von Multiagentensystemen können gleichermaßen als interne Störungen oder als Partnerstörungen auftreten und werden von den typischen Fehlerursachen der klassischen Datenkommunikation hervorgerufen, wie z.B. Paketverlusten, Paketduplizierungen, Reihenfolgevertauschungen oder Netzwerkpartitionierungen. Als Fehlerbehandlungsoption

gegen solcherlei Störungen bietet sich eine geeignete Gestaltung sog. **fehlerkorrigierender Übertragungsprotokolle** an.

Durch die hohe erreichbare Zuverlässigkeit der Übertragungsprotokolle bleibt nur eine begrenzte Auswahl von Fehlzuständen, die ggf. an die höheren Schichten propagiert werden müssen. Beispiele hierfür sind Übertragungsverzögerungen, längerfristige Verbindungsabbrüche, unerreichbare Netzwerkknoten und nicht beseitigbare Übertragungsfehler. Einige dieser Fehlzustände können in die **Aushandlung der Dienstgüte** (engl. *Quality of Service, QoS*) für die Schicht S1c mit einbezogen werden, wie z.B. die zulässige maximale Transportverzögerung. Andere Dienstgütekriterien, die nicht in die Nutzungsvereinbarung einbezogen wurden, müssen von der dienstnehmenden Schicht S2c beobachtet werden und dort müssen auch Ausnahmeereignisse ausgelöst und geeignete Gegenmaßnahmen eingeleitet werden, falls die Güte als unzureichend erachtet wird.

Vom gestörten Agenten selbst kann eine katastrophale Störung – sei es aufgrund eines Maschinenausfalls oder aufgrund einer dauerhaften Netzwerkpartitionierung – nicht erkannt werden. Stattdessen wird der Fehler auf Seite der Partneragenten festgestellt und führt dort zu einem Ausnahmeereignis („*Peer is down!*“), für das eine geeignete Behandlung gefunden werden muss.

Sensor- und Aktor-Störungen

Als dritte Fehlerquelle kommen in Schicht S1 die Sensoren und Aktoren der Realwelteinbettung (S1a) in Betracht, die ungenau arbeiten oder ausfallen können. Auf Schicht S1a selbst kann der *Ausfall* eines Wandlers, d.h. eines Sensors oder Aktors, nur durch direkte Ersatzsensoren/-aktoren behoben werden, also durch **Wandler-Redundanz**. Falls diese nicht vorhanden ist, wird eine Infrastrukturstörung nach oben weitergegeben und kann dort bearbeitet werden: evtl. lassen sich die Wahrnehmungen und Handlungen in S2c auch auf völlig andere Sensoren/Aktoren abbilden oder die Ziele des Agenten in S4 lassen sich durch eine andere Aktionsauswahl verfolgen.

Schwieriger zu handhaben ist eine *Verfälschung* der Sensordaten und Aktionen durch Ungenauigkeiten oder Fehlfunktionen. Da die Wirkung der Aktoren in der Regel durch

Sensoren überwacht wird, werden Aktorstörungen meist erkannt und können evtl. durch wiederholte Ausführung der Aktion oder durch Nachjustierungen der Ansteuerung behoben werden. Sollte dies nicht der Fall sein, oder liegt eine (nicht erkennbare) Sensorstörung vor, kann evtl. dennoch ein erfolgreiches Weiterarbeiten des Systems möglich sein: Da sich Agenten durch die Kooperation und Wechselwirkung mit anderen Agenten der Auswirkungen ihrer Handlungen niemals sicher sein können, sollten Weltmodell (S3) und Verhaltensgenerierung (S4) darauf vorbereitet sein, mit Unsicherheiten umzugehen.

4.4.2 Agentenspezifische Infrastrukturdienste (S2)

In Schicht S2 sind die agentenspezifischen Infrastrukturdienste beheimatet, die generisch gegenüber der konkreten Anwendungsdomäne sind. Die Semantik in der Domäne und auch das anwendungsspezifische zielorientierte Verhalten der Agenten spielen in dieser Schicht also keine Rolle. Üblicherweise werden die Infrastrukturdienste auf Schicht S2 in Agentenplattformen zusammengefasst, wie z.B. in den bekannten Vertretern JADE (Java Agent Development Environment, siehe Bellifemine u. a. 2005, 2007) und LS/TS (Living Systems/Technology Suite, siehe Rimassa u. a. 2005). Agentenplattformen müssen sich mit Partnerstörungen auseinandersetzen, die ihren Ursprung in der eigenen Schicht haben, und mit den nicht beseitigten Infrastrukturstörungen aus S1. Durch die Semantikfreiheit von S2 bleiben die Fehlerbehandlungsansätze der Agentenplattformen begrenzt auf quasi *mechanistische* Verfahren.

Nachrichtentransportstörungen

In Abschnitt 4.4.1 wurde als beispielhafte Störung von S1c ein Paketverlust genannt, der sich bei erfolglosem Behandlungsversuch auf Ebene der Agentenplattform in einer verlorenen Nachricht zwischen zwei Agenten niederschlagen kann. Auf Ebene der Plattformen können solche Störungen durch simple **„Time out"-Verfahren** in der Konversationsausführung und -überwachung adressiert werden, mit denen auf das Ausbleiben einer erwarteten Nachricht eine benutzerdefinierte Fehlerbehandlung angestoßen werden

kann. Wird die verlorene Nachricht als Antwort auf eine selbst gesendete Nachricht erwartet, wäre als ein primitiver Mechanismus z.B. ein wiederholtes Senden der eigenen Nachricht vorstellbar.

Eine solche Vorgehensweise ist jedoch abhängig von der Anwendungs- und Nachrichtensemantik nicht immer generisch realisierbar, da bekannt sein müsste, in welchen Fällen das wiederholte Senden einer Nachricht überhaupt sinnvoll ist oder ob der wiederholte Empfang der Nachricht auf der anderen Seite zu unerwünschten Zuständen führen kann. Es folgt, dass ein so gearteter Mechanismus zwar seinen technischen Ansatzpunkt, d.h. die Auslösung des Ausnahmeereignisses in S2c hat, dass dieses Ereignis aber zu einer erfolgreichen Behandlung an höhere Schichten propagiert werden muss. Dort führt dies zu einer Änderung der Protokolle innerhalb des Verbands und entspricht damit einer entschärften Weitergabe des Ausfalls zur Fehlerbehandlung auf der höheren Schicht S5.

Unerreichbare Partneragenten

Eine weitere Folge einer weitergegebenen Partner- oder Infrastrukturstrukturstörung in der Verteilungsunterstützung auf Schicht S1c ist der dauerhafte Verbindungsabbruch zwischen zwei kooperierenden Agenten. Genau wie ein katastrophaler Rechnerausfall auf der entfernten Seite führt er zu unerreichbaren Partneragenten. Fehlerbehandlungsmechanismen für derartige Störungen beruhen häufig auf der Replikation von Diensten oder – im Falle von Multiagentensystemen spezifischer – auf der **Agentenreplikation**. Falls ein Agent ausfällt, wird auf ein funktionsfähiges Replikat des Agenten zurückgegriffen. Die Fehlerbeseitigung durch Replikation, also das Umschalten auf ein Replikat statt des ursprünglichen Originals, sollte aus Sicht des Dienstnehmers weitestgehend unbemerkt geschehen, d.h. in der agentenspezifischen Infrastruktur von S2b verborgen werden. Sollte es nicht möglich sein, in S2b auf ein Replikat umzuschalten, dann muss versucht werden, die Anwendungsfunktionalität durch eine anderweitige Kooperation zu erbringen, d.h. die Störung wird nach Schicht S5 zur Rekoordinierung eskaliert.

Natürlich führt die Agentenreplikation nur dann zu erfolgreicher Fehlerbeseitigung, wenn die Replikate nicht zur gleichen Zeit – und möglicherweise aus demselben Grund – ausfal-

len wie die Originale. Als Konsequenz schützt Agentenreplikation vor Infrastrukturfehlern, wie z.B. vor Hardware- oder Betriebssystemausfällen, Agentenplattformausfällen und Kommunikationsverbindungsabbrüchen nur dann, wenn die replizierten Agenten auf einem anderen Rechnerknoten laufen und dieser über das Netzwerk verfügbar ist. Replikation wirkt nicht gegen Ausfälle, die auf Fehlerursachen in der Anwendungslogik oder der Programmierung beruhen, oder gegen sog. *ungewollte Anwendungsergebnisse* (engl. *unfavourable outcomes*, siehe Pleisch und Schiper 2004, Seite 222), da diese zusammen mit den Agenten repliziert werden, sondern nur gegen Ausfälle der Schichten S1 oder S2.

Plattformstörungen

Die Agentenreplikation ist geeignet als Fehlerbehandlungsmaßnahme im Falle dauerhafter Verbindungsabbrüche durch Ausfälle in der Verteilungsunterstützung und im Falle von katastrophalen Plattformstörungen durch weitergegebene Rechnerinfrastrukturstörungen auf Seite eines Partneragenten. Auf der Seite der katastrophalen Plattformstörung (S2b) selbst sind zum Zeitpunkt des Störungseintritts keine Maßnahmen mehr möglich. Zur Verbesserung der Zuverlässigkeit können jedoch Vorsorgemaßnahmen getroffen werden, die im Falle eines Wiederanfahrens der Plattform ermöglichen sollen, einen Zustand zu restaurieren, der dem Zustand direkt vor dem Störungseintritt möglichst nahe kommt. Dieser Zustand muss also die Störung überdauern können: Es entstehen sogenannte **persistente Agenten**, bei denen Teile des Zustands eines Agenten ereignisgesteuert auf persistente Medien abgespeichert werden.

Aus Performance-Gründen können einerseits in der Regel nicht alle Zustandsänderungen persistent gemacht werden und folglich können Teile des Zustands verloren gehen und andererseits ändern sich während der Wiederanlaufphase des persistenten Agenten seine Umgebung und die nicht ausgefallenen Agenten. Daher greifen die meisten Persistenzmechanismen auf die Mithilfe der Schichten S3 bis S5 zurück, d.h. sie geben einen Ausfall in abgemilderter Form weiter.

Registrierungsstörungen

Eine in der Praxis häufig anzutreffende Partnerstörung auf Schicht S2b tritt im Zusammenhang mit den Diensteverzeichnissen der Plattformen zu Tage. Die Diensteverzeichnisse implementieren eine „Gelbe Seiten"-Funktionalität, die Agenten hilft, ihre eigenen Dienstleistungen zu publizieren und benötigte Dienstangebote anderer Agenten ausfindig zu machen. Beim Ausfall eines Agenten, der einen Dienst im Verzeichnis offeriert, kann es vorkommen, dass die Offerte im Verzeichnis bestehen bleibt, obwohl der entsprechende Dienst nicht mehr verfügbar ist. Gegen solche veralteten Dienstregistrierungen helfen **aktive Diensteverzeichnisse**, die periodisch überprüfen, ob die publizierten Dienste noch verfügbar sind.

Sollte ein benötigter publizierter Dienst nicht verfügbar sein und sollten im Diensteverzeichnis auch keine alternativen gleichwertigen Dienste zu finden sein, dann muss die Nichtverfügbarkeit des angefragten Dienstes an die höheren Schichten weitergegeben werden. Am vielversprechendsten für eine Fehlerbehandlung ist in diesem Fall die Koordinationsschicht (S5), in der möglicherweise ein neues Dienstnutzungsmuster mit andersartigen verfügbaren Diensten gefunden werden kann.

Mobilitätsstörungen

Als Mobilitätsstörungen sollen Störungen bezeichnet werden, die speziell bei mobilen Agenten in der Phase ihres Umzugs zwischen zwei Plattformen eines MAS auftreten können. Dabei besteht die Gefahr, dass der Ausgang des Umzugs durch eine Infrastrukturstörung auf S1c oder S1b nicht eindeutig ermittelt werden kann, was zum Verlust eines Agenten und damit ggf. zu einem Ausfall auf Ebene des Multiagentensystems führen kann. Um dem Verlust eines Agenten vorzubeugen, verwenden einige **erweiterte Umzugsverfahren** das Klonen des umziehenden Agenten vom Ursprungs- auf den Zielknoten, mit einer Übergabe der Aktivität zum geklonten Agenten nach Abschluss der Duplizierung. Zwar kann theoretisch eine Infrastrukturstörung in dieser Phase dazu führen, dass keine Instanz oder beide Instanzen des Agenten am Ende des Umzugs aktiv sind. Dies lässt sich jedoch durch zusätzliche geeignete Transaktionsprotokolle bei der Aktivitätsübergabe verhindern.

Durch die erweiterten Umzugsverfahren können die mobilitätsgefährdenden Ausfälle der Schichten S1b und S1c vollständig in S2b beseitigt werden, soweit sie nicht auf einer dauerhaften katastrophalen Rechnerstörung oder auf einem dauerhaften Netzwerkproblem beruhen, die einen Umzug nachhaltig verhindern. Für diese Problemkreise existieren jedoch die oben beschriebenen allgemeineren Verfahren, so dass sich die höheren Schichten nicht mehr mit weitergegebenen Mobilitätsstörungen befassen müssen.

4.4.3 Ontologiebasiertes Weltmodell (S3)

In den unteren beiden Schichten S1 und S2 sind Agentensysteme noch völlig frei von aller anwendungsspezifischen Semantik. Das ändert sich ab dem Übergang zur Schicht S3, deren Aufgabe in der Verwaltung des Weltmodells des Agenten liegt, also genau im strukturierten Umgang mit den Ontologien und ihren Instanzen, in denen sich Anwendungssemantik und Umgebungszustand manifestieren. Für die Fehlerbehandlung auf S3 und höher ergibt sich damit die Möglichkeit, Störungen semantisch einzuordnen und anwendungsspezifische Gegenmaßnahmen zu treffen.

Imperfekte Situationseinschätzung

Werden die Sensor-/Aktor-Störungen aus Schicht S1a in Schicht S2a nicht beseitigt, so kann sich dies auf das Weltmodell in S3 auswirken, indem ungenaue Wahrnehmungen aus der Umgebung an das Weltmodell weitergegeben werden oder sogar eine Wahrnehmung eines bestimmten Zustandes aus der Umgebung ganz ausbleibt. Es entsteht eine Unsicherheit bezüglich eines bestimmten Artefaktes aus der Umgebung. Eine ähnliche Situation kann eintreten, wenn bei persistenten Agenten nach dem Wiederanlaufen der Plattform in Schicht S2b Unsicherheit herrscht, ob und wie sich die Umgebung zwischen dem ursprünglichen Abspeichern des wiederhergestellten Zustandes und dem Wiederanlaufen verändert hat.

Die Schicht S3 ist darüber hinaus noch weiteren Spielarten der Imperfektion ausgesetzt, die nicht auf Störungen basieren: Durch unerwünschte Wechselwirkungen mit anderen Agenten kann z.B. die Wirkung einer Aktion in der Umgebung nicht immer garantiert

werden und damit Unsicherheit über ihren Ausgang herrschen. Durch kontinuierliche Veränderungen in der Umgebung können Inkonsistenzen zwischen dem lokal gespeicherten Weltmodell und frischen eigenen Wahrnehmungen entstehen. Ebenso können die mitgeteilten Umgebungszustände von anderen Agenten im organisatorischen Verbund inkonsistent zu dem lokalen Weltmodell des empfangenden Agenten sein, z.B. bedingt durch zeitlich versetzte Beobachtungen.

Da die zuletzt beschriebenen Ereignisse keine Störungen darstellen, muss S3 schon im Normalbetrieb geeignet sein, mit Imperfektion umzugehen – eine Fähigkeit, die Pearl (1993) als **Uncertainty Management** bezeichnet. Dass der Umgang mit Imperfektion für S3 zum Normalbetrieb gehört, ist im Hinblick auf die Fehlerbehandlung der Infrastrukturstörungen hilfreich, da fehlerhafte Sensordaten meist nicht als Störung zu erkennen sind und damit in den unteren Schichten kein Ausnahmeereignis für den Wechsel in den Ausnahmebetrieb erzeugt werden kann.

Im Sinne der Zuverlässigkeitsreferenzarchitektur kann davon ausgegangen werden, dass der Schicht S3 eine Reduzierung der Unsicherheit gelingt, dass sich S4 aber dennoch nicht auf ein scharfes vollständiges Weltmodell verlassen kann – es erfolgt eine Fehlerweitergabe mit tragbarem Ergebniszustand von S3 an S4.

Kommunikationsontologie-Störungen

Neben den Infrastrukturstörungen aus S2a, die zu den imperfekten Wahrnehmungen führen, ist die Schicht S3 auch Partnerstörungen ausgesetzt, die in der ontologiebasierten Kommunikation der Agenten auftreten können. Eine mögliche Fehlerursache ist dort z.B. die Verwendung einer für den Empfänger unbekannten Ontologie durch einen Partneragenten bei der Erstellung einer Nachricht. Damit wird eine semantisch korrekte Interpretation der Nachricht beim Empfänger unmöglich. Selbst wenn Sender und Empfänger einer Nachricht die zugrundeliegende Ontologie teilen, kann es durch inkompatible Codierung/Decodierung beim Übergang von Instanzen des Weltmodells auf serialisierte Nachrichteninhalte zu Fehlern kommen.

Obwohl das Problem der (De-)Codierungs-Fehler in der Praxis häufig auftritt, ist es wissenschaftlich nicht gehaltvoll, da es durch einfache technische Maßnahmen wie z.B. **au-**

tomatische Serialisierer und validierende Parser bewältigt werden kann. Wissenschaftlich interessanter und daher auch intensiver erforscht ist das Problem der Verwendung unterschiedlicher Ontologien auf Sender- und Empfängerseite, das ebenfalls auf S3 durch **Ontologieübersetzung** angegangen werden kann.

Durch den Einsatz dieser Fehlerbehandlungsoptionen ist es in vielen Fällen möglich, Kommunikationsontologie-Störungen in S3 zu beseitigen und diese vor den höheren Schichten zu verbergen. Sollte dies nicht gelingen, besteht in S5 die Möglichkeit, Kommunikationsontologie-Störungen durch Mechanismen zu beheben, die auf eine explizite Aushandlung der Bedeutung von Termen zur Laufzeit abzielen.

Von Schicht S3 werden damit die abgemilderte Unsicherheit im Weltmodell an die Verhaltensgenerierung in Schicht S4 weitergegeben und die unbehandelten Kommunikationsontologie-Störungen an Schicht S5. Außerdem werden die in Schicht S3 nicht behandelbaren Störungen aus S2 durchgereicht, wie z.B. dauerhaft unerreichbare Partneragenten oder in S2 nicht vollständig beseitigte Rechnerausfälle.

4.4.4 Verhaltensgenerierung (S4)

In Schicht S4 ist die Verhaltensgenerierung mit Zielabwägung und Aktionsauswahl beheimatet, d.h. die Art und Weise, wie Agenten ihr Weltmodell aus S3 einsetzen, um ihr Verhalten davon abzuleiten. Dies bringt meist mächtige innere Architekturen mit sich, die gute Ansatzpunkte für Fehlerbehandlungsmechanismen anbieten, da sich neben der Anwendungssemantik hier die Ziele, das Aktionspotenzial und die Aktionshistorie der Agenten manifestieren.

Restunsicherheit im Weltmodell

Die Verhaltensgenerierung von Schicht S4 verwendet das Weltmodell von Schicht S3 als Infrastruktur, auf der es operiert. Wie oben beschrieben existiert dort immer eine gewisse Unsicherheit, die nicht nur von den hier betrachteten Partner- und Infrastrukturstörungen herrührt, sondern auch von den bereits im Normalbetrieb vorhandenen exogenen Störungen. Folglich finden auf S4 in der Regel **innere Agentenarchitekturen mit**

Unsicherheitsbeherrschung Anwendung, die mit der dort operationalisierten Unsicherheit zurecht kommen sollen.

In der Regel ist die Behandlung der Unsicherheit auf S4 als vollständig abgeschlossen zu betrachten und die entsprechenden zugrunde liegenden Fehlzustände – meist Sensor-/Aktor-Störungen – sind aus Sicht der dienstnehmenden Schicht S5 beseitigt. Folglich werden die Fehlerbehandlungsoptionen zur Unsicherheitsbeseitigung auf der Koordinationsschicht – z.B. Nachfrage-Verfahren zur Validierung bzw. Falsifizierung der eigenen Wahrnehmungen – in systematischen wissenschaftlichen Arbeiten kaum betrachtet.

Entschärfte Plattformausfälle

Aufgrund des breiten Spektrums an Aufgaben in S4 und des damit verbundenen Fehlerbehandlungspotenzials sollte das Ziel der in dieser Schicht beheimateten Mechanismen eine möglichst vollständige Fehlerbeseitigung der verbliebenen weitergereichten Infrastrukturstörungen sein. Dies gilt sogar bei einem vorübergehenden Totalausfall der lokalen Agentenplattform. Hier kann im besten Fall auf eine teilweise Entschärfung der Störung aufgesetzt werden, wie sie die persistenten Agenten in S2 anbieten. Dabei verlieren die Agenten vorübergehend die Kontrolle, aber bekommen nach dem Wiederanlaufen der Plattform die Gelegenheit, in einen geregelten Zustand zurückzusetzen. Das Wiederanlaufen in S2 ist jedoch ein rein mechanischer Vorgang: Da in dieser Schicht keine Anwendungssematik vorliegt, kann nur versucht werden, auf Datenebene einen syntaktisch möglichst aktuellen Zustand zu erreichen.

In S4 sind jedoch Anwendungssemantik, Ziele und Aktionen des Agenten bekannt und damit die Gründe, wie und warum es zu einem bestimmten Zustand eines Agenten gekommen ist und welche anderen Zustände nach einem Wiederanlaufen semantisch anstrebenswert sind. Auf dem gewonnenen Wiederaufsetzpunkt von S2 kann der Agent also seine eigenen **erweiterten Wiederanlaufverfahren** durchführen. Diese versuchen S4 in einen Zustand zurückzusetzen, der unter dem erneuten Abwägen der Ziele und/oder einer erneuten Aktionsauswahl eine Rückkehr in den Normalbetrieb erlaubt. Dazu ist es erforderlich, dass der Agent über seine vergangenen Aktionen, deren Effekte nach

dem Wiederanlaufen teilweise verloren gegangen sein können, Buch führt, um diese in geeigneter Form rückgängig machen zu können.

Unerwartetes Verhalten anderer Agenten

Als Partnerstörung in der Verhaltensgenerierung treten unvorhergesehene Aktionen von anderen Agenten auf, die nicht dem erwarteten oder – bei Partneragenten – dem in der Koordinationsschicht S5 vereinbarten Verhalten entsprechen. Falls aufgrund unerwünschter Wechselwirkungen mit den Aktionen anderen Agenten die eigenen ausgewählten Aktionen scheitern und damit die gesetzten Ziele auf dem eingeschlagenen Weg nicht zu erreichen sind, können die oben beschriebenen Rücksetzmechanismen der **erweiterten Wiederanlaufverfahren** greifen. Auch hier kann versucht werden, die Verhaltensgenerierung in einen Zustand zurückzusetzen, in dem der Normalbetrieb alternative zielführende Aktionen bestimmen kann. Allerdings ist dies gerade bei kooperierenden Agenten in der Regel nicht ohne die Inanspruchnahme der Schicht S5 möglich, d.h. nicht ohne eine Fehlerweitergabe im Sinne des Fehlerbehandlungsmodells.

4.4.5 Koordination (S5)

Während in Schicht S4 die Zuverlässigkeit der einzelnen Agenten im Mittelpunkt steht, ist die Koordinationsschicht S5 für eine zuverlässige Funktion des Multiagentensystems als Ganzes zuständig. Da es sich bei S5 um die höchste Schicht der Referenzarchitektur handelt, muss das Ziel einer Fehlerbehandlung die vollständige Beseitigung aller Fehler sein. Andernfalls bleibt nur noch das Anstoßen eines Benutzereingriffs. Beispiele für Infrastrukturstörungen, denen sich die Schicht S5 ausgesetzt sieht, sind längerfristig unerreichbare Partneragenten, was z.B. durch Netzwerkpartitionierung in S2 verursacht werden kann, oder die von S4 weitergegebenen unerwarteten Verhalten anderer Agenten.

Langfristig unerreichbare Partneragenten

Die längerfristige Unerreichbarkeit von Partneragenten kann dazu führen, dass eine vereinbarte Kooperation nicht aufrecht erhalten werden kann. Die wechselseitige Unerreichbarkeit äußert sich zunächst durch das Ausbleiben erwarteter Nachrichten. In der Koordinationsschicht besteht die Möglichkeit, durch **Konversationserweiterungen** der ursprünglichen Konversationsprotokolle festzustellen, ob es sich um eine zeitlich begrenzte Störung handelt, die eine Weiterarbeit mit dem ursprünglichen Kooperationpartner zulässt, oder ob eine permanente Unerreichbarkeit vorliegt. Dies kann beispielsweise eine Restrukturierung der Kooperationsbeziehungen mit einer Neuwahl alternativer Partner mit vergleichbaren Dienstangeboten erforderlich machen. Die erforderliche **Rekoordinierung** gehört zum Standardfunktionsumfang der Schicht S5, da sie einer gewöhnlichen Koordination entspricht mit der Randbedingung, dass die Auswahl bestimmter Partner ausgeschlossen ist

Unerwartetes Verhalten anderer Agenten

In der Koordinationsschicht S5 kann zum Teil nicht scharf unterschieden werden, ob das unerwartete Verhalten absichtlich oder versehentlich zustande gekommen ist, was sich auch in den zugehörigen Fehlerbehandlungsmaßnahmen widerspiegelt: die meisten Ansätze adressieren absichtliches und versehentliches Fehlverhalten gleichermaßen, indem sie nicht nach der Ursache des Fehlverhaltens unterscheiden.

Unerwartetes Verhalten im Allgemeinen Unerwartetes Verhalten im Allgemeinen umfasst absichtliches und versehentliches Fehlverhalten, also auch Fehlverhalten von Partneragenten, das durch ungewollte Wechselwirkungen mit dritten oder durch eine anderweitig begründete Unwirksamkeit der Aktionen bedingt wird. Die Fehlerbehandlung erfolgt in der Koordinationsschicht häufig nicht explizit, sondern wird durch **robuste Koordinationsmechanismen** im Normalbetrieb geleistet. Motiviert werden die robusten Mechanismen aus den Wirtschafts- und Sozialwissenschaften.

Absichtliches Fehlverhalten Neben den Mechanismen, die nicht scharf zwischen versehentlichem und absichtlichem Fehlverhalten trennen, gibt es noch eine Reihe von Ansätzen, die auf absichtliches Fehlverhalten spezialisiert sind. Das absichtliche Fehlverhalten liegt in der kompetitiven Natur speziell bei offenen Multiagentensystemen begründet. Agenten, die kein gemeinsames übergeordnetes Ziel verfolgen, müssen sich auch nicht den Regeln eines gütlichen Zusammenarbeitens unterwerfen. Als Folge entsteht absichtliches Fehlverhalten mit dem Ziel, sich einen Vorteil gegen andere Agenten zu verschaffen – also Konkurrenz unter Einsatz aller Mittel. In diesem Fall ist die Durchsetzung eines übergeordneten Gemeinwohls notwendig. Die entsprechenden Mechanismen beruhen meist auf einer technischen Umsetzung der Konzepte „Vertrauen“ (engl. *Trust*) und „Reputation“ und liegen nicht im Fokus dieser Arbeit, die sich auf unbeabsichtigte Fehler konzentriert. Folglich sollen sie nur erwähnt werden.

Konversationsstörungen

Als Partnerstörung treten auf Schicht S5 Störungen im Koordinationsablauf selbst auf, nämlich die sog. Konversationsstörungen. Sie entstehen, wenn die an der Konversation beteiligten Partneragenten inkompatible Protokolle umsetzen, also abweichende Nachrichtenabfolgen oder Nachrichtentypen annehmen. Beide Fehlerursachen lassen sich von der lokalen Konversationsausführung und -überwachung in S2c zur Laufzeit zwar einfach erkennen, eine Behandlung zur Laufzeit erfordert aber eine Protokollumsetzung unter Berücksichtigung vieler potenzieller Fehlerfälle.

Einfacher zu realisieren sind Maßnahmen während des Agentenentwurfs, der meist den Protokollentwurf als einen dedizierten Teil-Entwurfsprozess zur Spezifikation der Nachrichtenabfolgen und der jeweiligen Nachrichtentypen vorsieht. Am Ende dieses Prozessschrittes kann eine formale oder simulative **Protokollverifikation** erfolgen, bei der die Protokolle z.B. auf Ausführbarkeit, Blockadefreiheit oder vollständigen Informationsfluss untersucht werden können.

Kommunikationsontologie-Störungen

Verwenden Sender und Empfänger einer Nachricht in S3 verschiedene Kommunikationsontologien oder verwenden sie dieselbe, aber messen ihren Termen eine unterschiedliche Bedeutung zu, so kann versucht werden, diese Kommunikationsontologie-Störung in S3 auf einer der Seiten durch eine Übersetzung zwischen der eigenen und der fremden Ontologie zu behandeln. Dabei muss der behandelnde Agent wissen, welche Bedeutung der andere Agent den Termen beimisst, was in der Regel nur bei bekannten – quasi standardisierten oder in der Entwurfsphase vereinbarten – Ontologien vorausgesetzt werden kann.

Fortschrittlichere Ansätze sind in S5 angesiedelt und verfolgen bei einem Scheitern der Versuche auf S3 die dynamische **Bedeutungsaushandlung** für Terme im Rahmen einer vorgeschalteten Konversationsphase zwischen den kommunizierenden Agenten.

4.4.6 Fehlerbehandlung durch den Benutzer

Wenn die Fehlerbeseitigung im Multiagentensystem nicht vollständig geleistet werden kann, zeigt das System nach außen – also gegenüber seinen Benutzern – ein Fehlverhalten. Im Sinne des in diesem Kapitel beschriebenen Fehlerbehandlungsmodells könnte man sagen, dass der Fehler an eine virtuelle Schicht „S6“, nämlich die „Benutzerschicht“, propagiert wird. Es ist einsehbar, dass ein Multiagentensystem nicht jeden erdenklichen Fehler selbstständig behandeln kann und unter Umständen ist dies sogar überhaupt nicht wünschenswert. Wenn ein Fehler im Agentensystem tiefgreifende Spuren in der Umgebung hinterlässt, kann es für die Fehlerbehandlung schwierig werden, eine angebrachte Gegenmaßnahme zu wählen, falls sie überhaupt zum Verhaltensrepertoire der Agenten gehört.

Folglich muss ein weitsichtiger Systementwurf berücksichtigen, welche Fehlerzustände noch im Multiagentensystem behandelt werden sollten und welche bis zur Benutzerschicht eskaliert werden müssen. In letzterem Fall wird dem Benutzer als Ausnahmeereignis durch die Agenten eine möglichst präzise Beschreibung des Fehlerzustandes gegeben und sein korrigierendes Eingreifen angefragt. Das Spektrum für den menschlichen

Eingriff ist breit. Es kann von einer einfachen Bedienung des Multiagentensystems – etwa zur Neuparametrisierung – über Eingriffe an der Rechnerinfrastruktur bis hin zu Handlungen in der realen Anwendungsumgebung reichen.

4.4.7 Zusammenführung der Zuverlässigkeitsreferenzarchitektur

Die Zuverlässigkeitsreferenzarchitektur intelligenter Agenten in Multiagentensystemen ensteht nun, indem die verorteten Störungen und prinzipiellen Fehlerbehandlungsoptionen aus den Unterabschnitten 4.4.1–4.4.5 in die Referenzarchitektur aus Abbildung 3.11 eingeordnet werden und das Ergebnis mit dem (vereinfachten) Fehlerbehandlungsmodell aus Abschnitt 4.3 zusammengeführt wird.

Abbildung 4.6 illustriert den ersten Schritt durch die Verortung der Störungen (in der Abbildung als abgerundete Rechtecke) und der Fehlerbehandlungsoptionen (als Achtecke) und durch deren Verknüpfung. Die durchgezogenen Pfeile zeigen, welche Störungen durch welche Fehlerbehandlungsoption adressiert werden und die gestrichelten Pfeile geben einige ausgewählte Möglichkeiten an, wie nur teilweise behandelte Störungen durch die Schichten eskalieren können. Abbildung 4.6 gibt somit einen Überblick über die Zuverlässigkeitsreferenzarchitektur und insbesondere über das potenzielle Zusammenspiel der Fehlerbehandlungsoptionen auf den verschiedenen Schichten.

Den Überblick aus Abbildung 4.6 führt Abbildung 4.7 mit dem abstrahierten Fehlerbehandlungsmodell für einzelne Schichten aus Abbildung 4.5 zusammen zum Hauptgegenstand der Arbeit: der Zuverlässigkeitsreferenzarchitektur intelligenter Agenten in Multiagentensystemen. Auf Basis dieser Darstellung werden im folgenden Kapitel 5 verschiedene Pfade definiert, entlang derer bestimmte Störungen durch die Schichten der Architektur eskalieren und dabei wird untersucht, wie sich diese Störungen mit existierenden konkreten Fehlerbehandlungsmechanismen in den verschiedenen Schichten behandeln lassen.

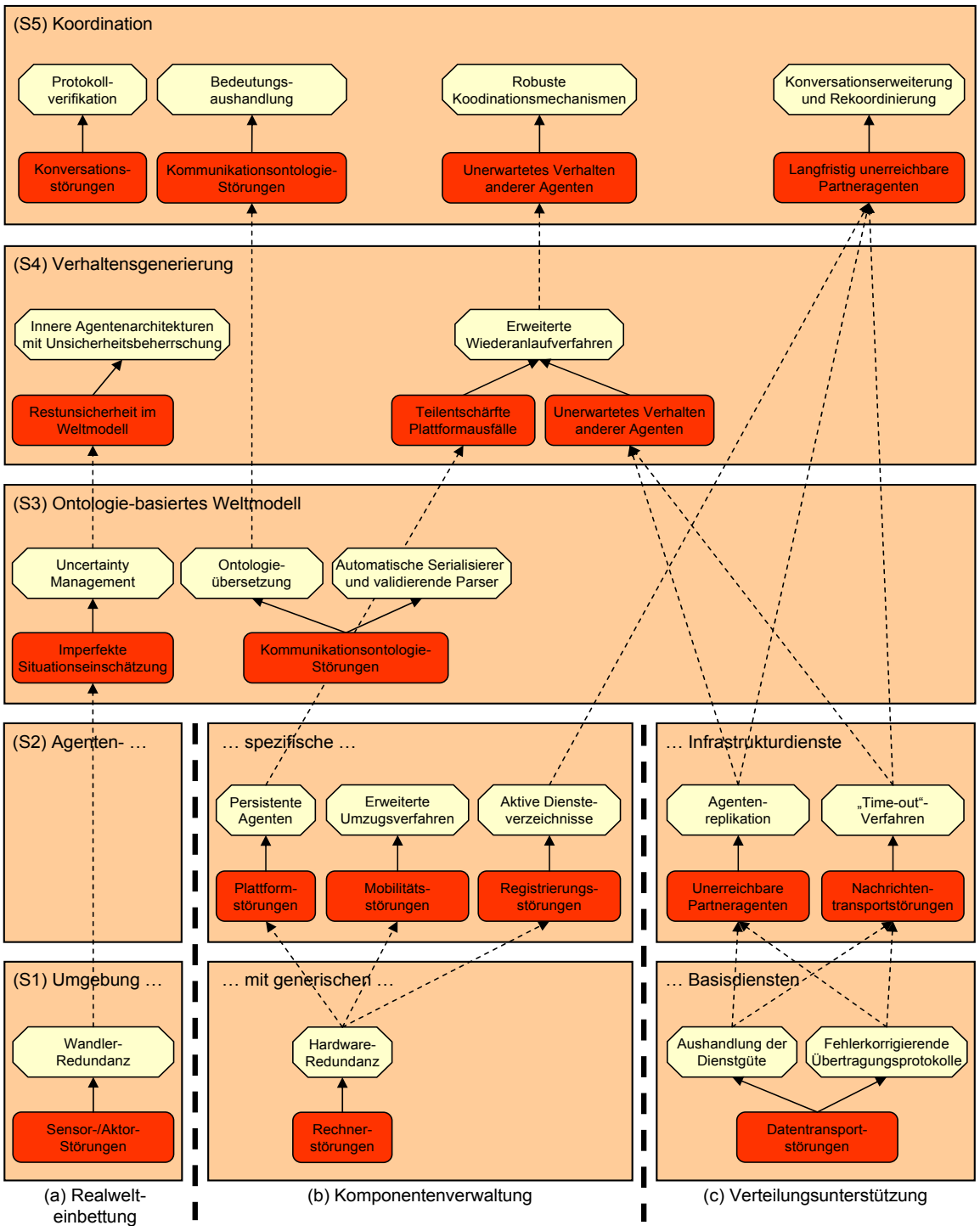


Abbildung 4.6: Die Zuverlässigkeitsreferenzarchitektur im Überblick

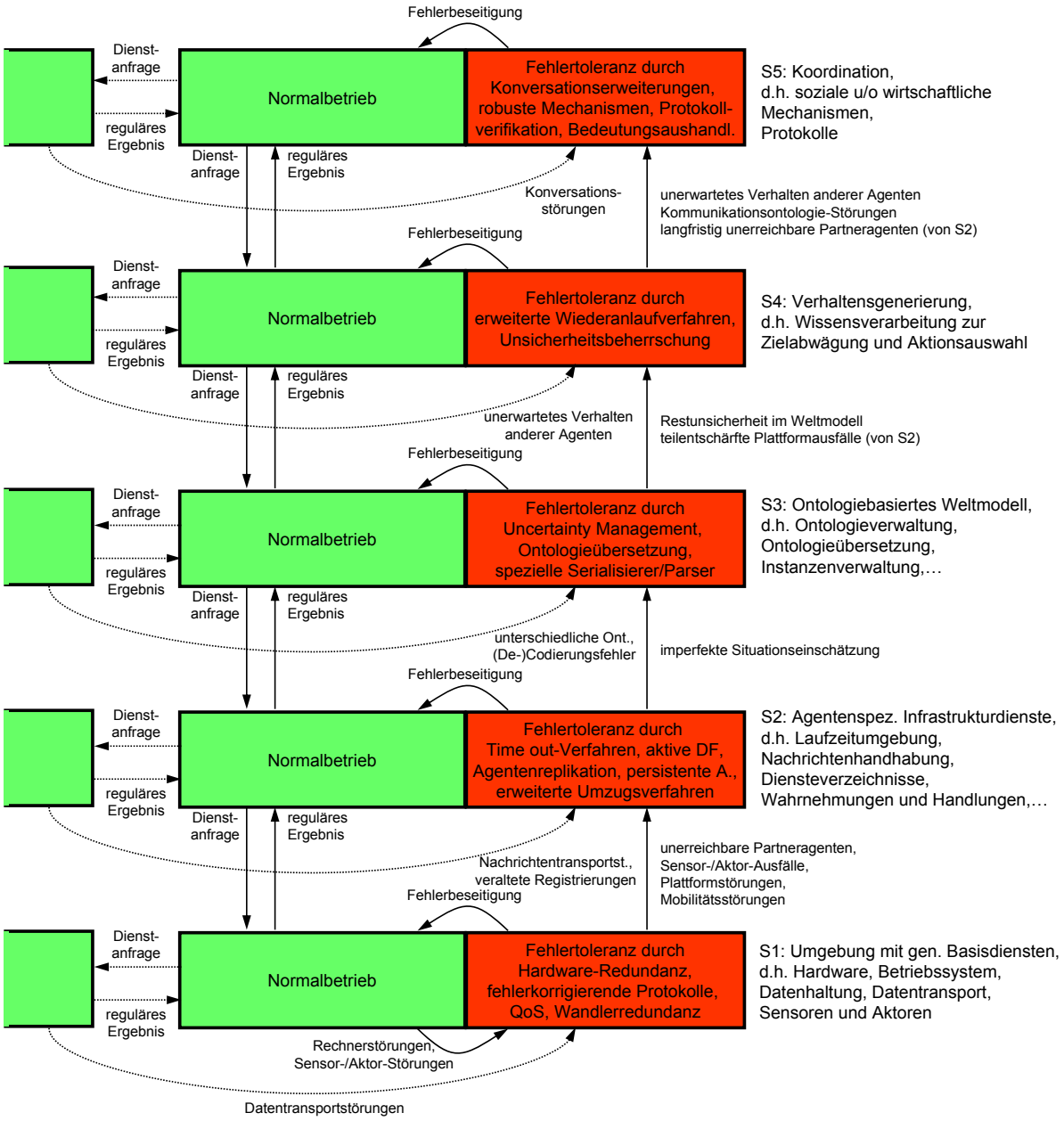


Abbildung 4.7: Die Zuverlässigkeitsreferenzarchitektur intelligenter Agenten in Multiagentensystemen

Kapitel 5

Landkarte existierender Zuverlässigkeitsmechanismen

Im vorangegangenen Kapitel 4 wurde auf Basis der Schichten der allgemeinen Referenzarchitektur für Multiagentensysteme die Zuverlässigkeitsreferenzarchitektur als Hauptgegenstand der vorliegenden Arbeit entwickelt (siehe Abbildung 4.7). Diese Entwicklung erfolgte abstrakt entlang der Aufgaben und Eigenschaften der einzelnen Schichten, aus denen die potenziellen Störungen und Fehlerbehandlungsoptionen abgeleitet wurden. Darüber hinaus wurde gezeigt, wie sich unvollständig behandelte Störungen durch die Schichten fortpflanzen können.

Im vorliegenden Kapitel werden nun konkrete Fehlerbehandlungsmechanismen in der Zuverlässigkeitsreferenzarchitektur verortet. So entsteht eine Landkarte der Zuverlässigkeitsmechanismen für Multiagentensysteme, die zum einen die Verortungsfähigkeit der Zuverlässigkeitsreferenzarchitektur demonstriert und zum anderen die Mechanismen voneinander abgrenzt, ihr Zusammenspiel illustriert und auch deren Schwachstellen aufzeigt. Dem Agentensystem-Entwickler hilft die Landkarte, ein identifiziertes Zuverlässigkeitsproblem einzuordnen und ggf. geeignete existierende Mechanismen auszuwählen.

In der Literatur gibt es eine große Vielfalt von Ansätzen zur Steigerung der Zuverlässigkeit von Multiagentensystemen, deren Mechanismen bzw. adressierte Störungen über alle agentenspezifischen Schichten (S2 bis S5) der Referenzarchitektur verstreut sind. Die

Verortung in diesem Kapitel beschränkt sich auf ausgewählte anspruchsvolle Ansätze, die in der einschlägigen Literatur eine hervorgehobene Stellung einnehmen, und auf archetypische illustrierende Beispiele. Nicht-agentenspezifische Ansätze der Schicht S1, die für die Zuverlässigkeitslandschaft der Multiagentensysteme eine Rolle spielen, werden der Vollständigkeit zuliebe genannt, aber nicht im Detail beschrieben.

Die Gliederung des vorliegenden Kapitels erfolgt nicht wie bei der Entwicklung der Zuverlässigkeitsreferenzarchitektur entlang der Schichten, sondern sie orientiert sich an sog. *Pfaden*. Diese Pfade ergeben sich, wenn in niederen Schichten Störungen auftreten, diese am Ort ihrer Entstehung (unvollständig) behandelt werden und sich durch Eskalation das Muster aus schichtenspezifischen Störungen und Fehlerbehandlungsversuchen dann rekursiv durch die höheren Schichten fortpflanzt. Durch die Darstellung entlang der Pfade lässt sich das Zusammenspiel verschiedener Fehlerbehandlungsmechanismen besonders intuitiv veranschaulichen. Die gezeigten Pfade sind dabei nicht exklusiv zu verstehen, sondern exemplarisch: Nach einer unvollständigen Fehlerbehandlung durch einen bestimmten Mechanismus muss die eskalierte Störung nicht zwangsläufig durch den nächsten Fehlerbehandlungsmechanismus im Pfad erfolgen – häufig kommen auch andere Mechanismen in Frage, die möglicherweise sogar selbst auf anderen Pfaden liegen können.

Konkret werden im Folgenden vier Pfade dargestellt: „Unsicherheiten“ (in Abschnitt 5.1), „Ontologiestörungen“ (5.2), „Rechnerstörungen und unerwartetes Agentenverhalten“ (5.3) und „Netzwerkstörungen“ (5.4). Darüber hinaus werden in Abschnitt 5.5 noch weitere exemplarische Mechanismen vorgestellt, die sich nicht in einem der obigen Pfade befinden.

5.1 Pfad „Unsicherheiten“

Der Pfad „Unsicherheiten“, der in Abbildung 5.1 dargestellt ist, hat seinen Ursprung in den Sensor-/Aktor-Störungen in Schicht S1a, wo versucht wird, diese mit Wandlerredundanz zu beheben. Bei der Eskalation entsteht eine imperfekte Situationseinschätzung in der Weltmodell-Schicht S3, die jedoch mit einem Uncertainty Management explizit auf eine solche Störung vorbereitet ist. Die Restunsicherheit, die nach

dieser Behandlung bleibt, wird an Schicht S4 weitergegeben, wo eine innere Agentenarchitektur mit Unsicherheitsbeherrschung versucht, die Störungen endgültig zu beheben.

Wandlerredundanz gegen Sensor- und Aktor-Störungen

Konkrete Zuverlässigkeitsmechanismen gegen Sensor- und Aktor-Störungen auf Basis der Wandlerredundanz stammen aus den Gebieten Robotik, Eingebettete Systeme und Anlagenbau. Da es sich dabei nicht um agentenspezifische Mechanismen handelt, soll in dieser Arbeit nicht näher auf sie eingegangen werden.

Prinzipiell basieren sie auf der mehrfachen Auslegung der Sensoren/Aktoren und auf Algorithmen, die entweder einen führenden Sensor/Aktor aus den redundanten Wandlern bestimmen oder z.B. durch Mittelwertbildung die Signale aus der Gruppe aggregieren. Im allgemeinen sind solche Mechanismen nicht agentenspezifisch implementiert und damit der Teilschicht S1a zuzurechnen. Vorstellbar wäre aber auch eine Implementierung der genannten Algorithmen innerhalb der agentenspezifischen Infrastruktur in S2a, bei der Umsetzung zwischen Signalen und Wahrnehmungen bzw. Handlungen. Hierzu existieren allerdings keine herauszuhebenden Arbeiten.

Falls mit den beschriebenen Mechanismen von Schicht S1a (bzw. S2a) die Störungen nicht vollständig behoben werden können, kommt es im Weltmodell in Schicht S3 zu einer imperfekten Situationseinschätzung, weil entweder eine unrichtige Wahrnehmung erfolgt ist oder weil eine Handlung nicht wie geplant – und im Weltmodell gespeichert – durchgesetzt werden konnte. Das Weltmodell muss dann Mechanismen bieten, mit der entstandenen Unsicherheit explizit umgehen zu können.

Uncertainty Management gegen imperfekte Wahrnehmungen

Schon im Normalbetrieb ist das Weltmodell in Schicht S3 Unsicherheiten ausgesetzt, die z.B. in der hohen Änderungsgeschwindigkeit der Umgebung oder in Wechselwirkungen mit anderen Agenten begründet sein können. Die auf Störungen beruhende imperfekte Situationseinschätzung ist aus dieser Sicht nur eine weitere Spielart der Imper-

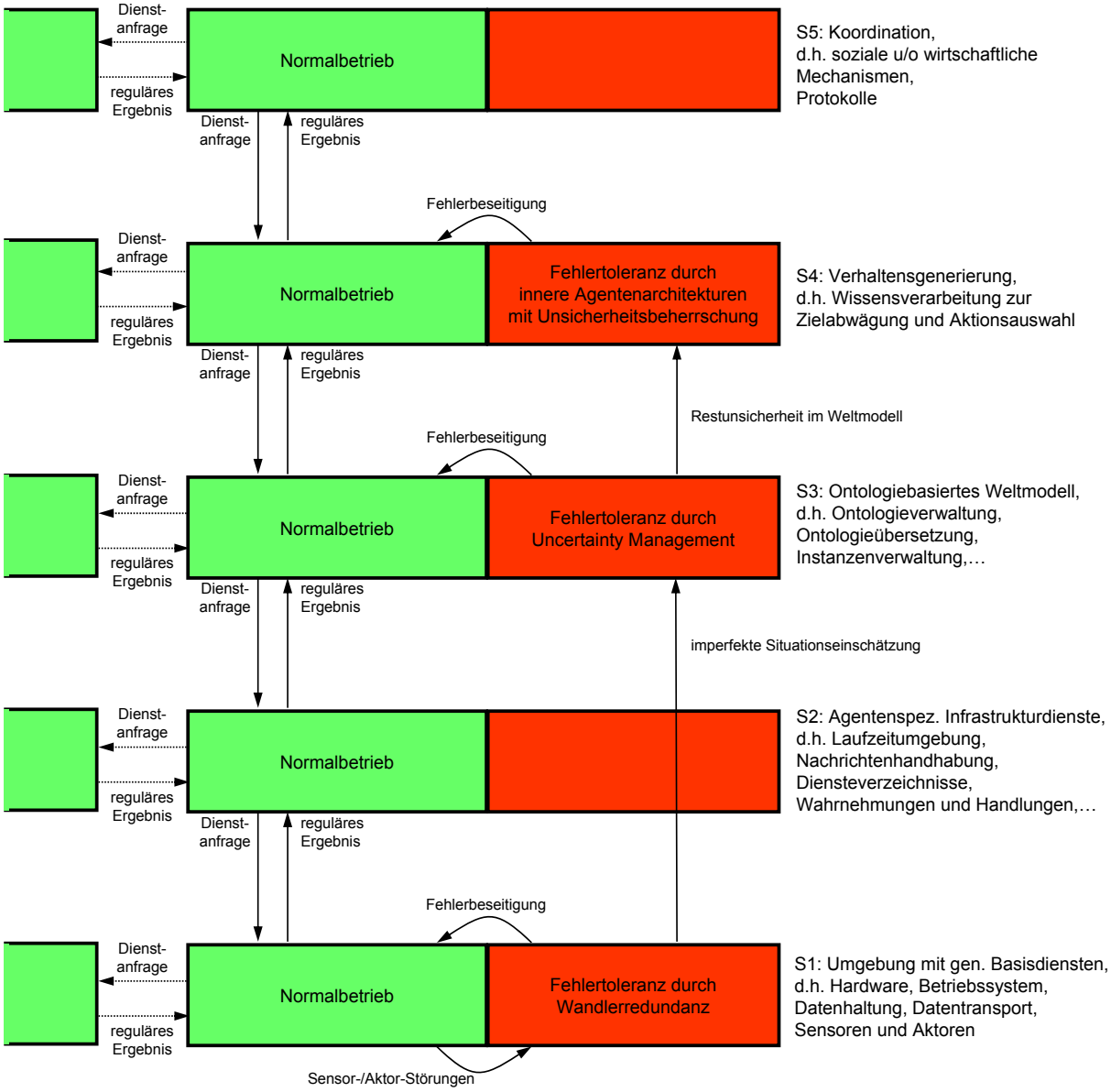


Abbildung 5.1: Pfad „Unsicherheiten“

fektion, mit der S3 durch das sog. Uncertainty Management (Pearl 1993) umzugehen versteht.

Für die Operationalisierung des Uncertainty Management im Agentenkontext, die Lockemann und Witte (2005) aus dem Blickwinkel der verteilten Datenbanken unter dem Begriff *Uncertain Agents* skizzieren, ist zunächst der Begriff der Imperfektion zu schärfen. Sie tritt bei Agenten in den folgenden bekannten Spielarten auf (nach Motro und Smets 1997):

Ungenauigkeiten: Eine Information ist ungenau, wenn sich der tatsächliche Zustand nur auf eine von mehreren möglichen Optionen eingrenzen lässt, aber nicht auf einen genauen Wert.

Inkonsistenzen: Mehrere Einzelinformationen innerhalb des Weltbildes stehen zueinander in Widerspruch, so dass eine schlüssige Folgerung des tatsächlichen Umgebungszustandes nicht direkt möglich ist.

Unsicherheiten: Der Agent kann aus seinem Weltmodell nicht ableiten, ob der angenommene Zustand eines Artefaktes mit dem tatsächlichen Zustand in der Umgebung übereinstimmt oder nicht.

Das lokale Abbild eines Agenten von seiner Umgebung kann also ungenau, widersprüchlich und unsicher sein. So geartetes Wissen eines Agenten von seiner Umgebung wird in der Agententheorie daher als Vorstellung (im Folgenden engl. *Belief*) bezeichnet. Die Operationalisierung der Beliefs wirft nun vor allem die Frage auf, wie der Agent mit neu erworbenem Wissen umgehen soll, das er aus eigenen Beobachtungen oder Mitteilungen anderer Agenten bezogen hat, d.h. wie er aus seinem alten Weltmodell und den neuen Beobachtungen eine neue aktuelle Version seines Weltmodells erstellt. Dabei gilt zunächst die Regel, dass ein Agent bei der Überarbeitung seiner Beliefs (engl. *Belief Revision*) aufgrund einer neuen Beobachtung möglichst viele seiner alten Beliefs beibehalten sollte (Harman 1983). Dazu kann er sich folgender grundlegender Operatoren bedienen (Alchourrón u. a. 1985):

Expansion: Eine neue Beobachtung wird durch diesen Operator dem Weltmodell direkt hinzugefügt. Dabei spielt keine Rolle, ob die hinzugefügte Beobachtung zum bisherigen Weltmodell im Widerspruch steht: Inkonsistenzen werden hingenommen.

Revision: Auch die Revision fügt dem Weltmodell eine neue Beobachtung hinzu. Unverträgliche ältere Beliefs werden dabei jedoch gelöscht.

Kontraktion: Die Kontraktion geht noch einen Schritt weiter als die Revision, indem nicht nur die direkt unverträglichen Beliefs gelöscht werden, sondern auch alle Beliefs, von denen sich Widersprüche ableiten lassen würden.

Basierend auf diesen drei Grundoperatoren sind abhängig von der Darstellung des Weltmodells konkrete Variationen und Erweiterungen der Belief Revision möglich. Beispiele sind die formallogischen Ansätze von Katsuno und Mendelzon (1992) und Perrussel und Thévenin (2004) für einzelne Agenten bzw. für organisatorische Verbünde von Agenten oder der probabilistische Ansatz von Shafer (1976) für kommunizierende Beobachter im Allgemeinen.

Die verschiedenen Verfahren zur Belief Revision können die Unsicherheit im Weltmodell reduzieren, haben dabei aber mit einem Problem zu kämpfen, das in den grundlegenden Operatoren *Revision* und *Kontraktion* des Weltmodells verwurzelt ist. Beide Operatoren sind nicht-deterministisch, da es mehrere Möglichkeiten geben kann, eine Unverträglichkeit im Weltmodell aufzulösen bzw. da dessen Bereinigung kaskadieren kann. Dies führt dazu, dass auf Basis einer abgemilderten Störung im Weltmodell eine Weitergabe des Ausfalls an die Schicht S4 notwendig werden kann.

Innere Agentenarchitektur mit Unsicherheitsbeherrschung gegen Restunsicherheit im Weltmodell

Die Verhaltensgenerierung von Schicht S4 verwendet das Weltmodell von Schicht S3 als Infrastruktur, auf der es operiert, und muss folglich mit den Beliefs – oder anderweitig operationalisierter Unsicherheit – zurecht kommen. Dies ist ein Grund für die weite Verbreitung des BDI-Ansatzes als innere Agentenarchitektur mit Unsicherheitsbeherrschung auf Schicht S4. Beim BDI-Modell werden die als nächstes auszuführenden Handlungsabsichten I (engl. *Intentions*) abhängig gemacht von den Beliefs B eines Agenten und den vorgegebenen Zielen D (engl. *Desires*). Bratman (Bratman 1987; Bratman u. a. 1988) definiert eine formallogische Darstellung von Beliefs, Desires und Intentions und leitet daraus Grundregeln für das Handeln unter Unsicherheit ab. Eine konkrete technische

Umsetzung des konzeptionellen BDI-Modells bietet Jadex (Pokahr u. a. 2005) in Form einer Erweiterung für das Agentenrahmenwerk JADE.

Neben dem BDI-Modell zur Verhaltensgenerierung existiert auf S4 eine große Auswahl weiterer Verhaltensgenerierungsansätze, die aber im Unterschied dazu die Unsicherheit nicht immer explizieren. Bei ihnen ist die Aktionsauswahl implizit so gestaltet, dass sie eine Restunsicherheit im Weltmodell bei der vergleichenden Bewertung der Aktionsalternativen toleriert und dabei zielführende – wenngleich möglicherweise suboptimale – Aktionen findet. Andere Ansätze ignorieren die Möglichkeit der Unsicherheit und verlangen von S3 die Bereitstellung eines konsistenten Weltmodells. Dies kann zwar durch Kontraktion erreicht werden, führt aber andererseits zu einem lückenhaften Weltmodell, das die Bestimmung zielführender Aktionen nicht immer zulässt. Ineffiziente Agenten können die Folge sein.

Wissenschaftlich wird die Behandlung von Unsicherheiten durch die Popularität geeigneter innerer Architekturen meist auf S4 als vollständig abgeschlossen betrachtet. Nachfrageverfahren, die in S5 weitere Agenten zur Auflösung von lokalen Unsicherheiten einbeziehen, sind daher nicht weit verbreitet und der Pfad „Unsicherheiten“ der Zuverlässigkeitslandkarte endet somit in Schicht S4.

5.2 Pfad „Ontologiestörungen“

Der Pfad „Ontologiestörungen“ beginnt, wie in Abbildung 5.2 dargestellt, erst in Schicht S3, in der die Ontologien als Basis des Weltmodells zum ersten Mal vorkommen. Dort treten die Kommunikationsontologiestörungen als Partnerstörungen auf, wenn z.B. Artefakte des Weltmodells nicht richtig (de-)serialisiert werden oder Sender und Empfänger einer Nachricht unterschiedliche Ontologien verwenden. Eine Störungsbehandlung bietet der Pfad lokal in der Schicht S3 oder kooperativ in Schicht S5.

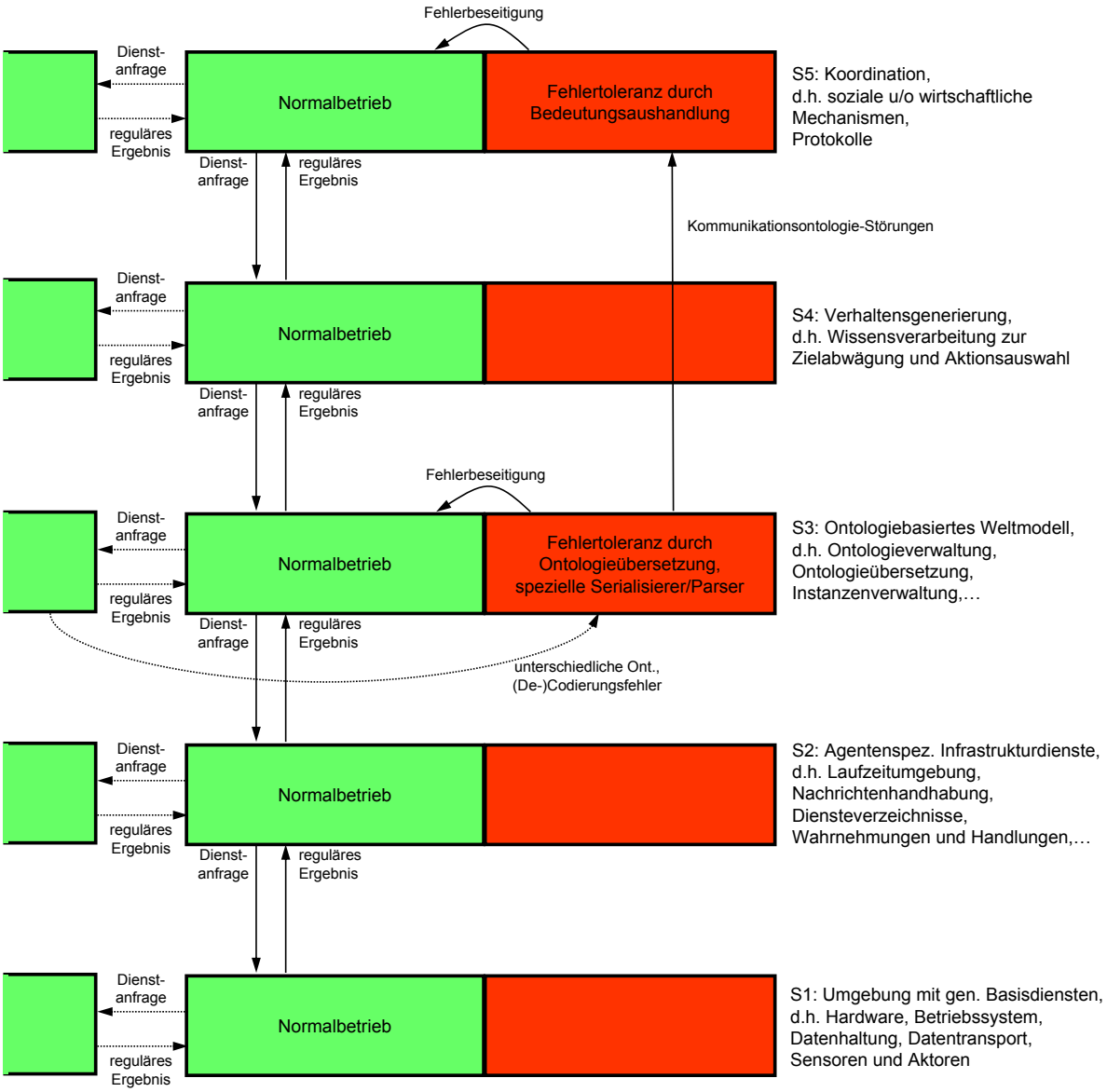


Abbildung 5.2: Pfad „Ontologiestörungen“

Automatische Serialisierer und validierende Parser gegen Kommunikationsontologie-Störungen

Die nicht-automatisierte Überführung von Artefakten eines Ontologie-basierten Weltmodells in eine serialisierte Darstellung zum Zwecke der Nachrichtenübermittlung ist ein fehleranfälliger Prozess. Eine falsche Codierung der Nachricht auf Senderseite bedeutet in der Konsequenz, dass auf der Empfängerseite das übermittelte Artefakt nicht richtig in das Weltmodell eingebaut werden kann.

Als Fehlerbehandlungsmaßnahme werden automatische Serialisierer und validierende Parser als eingebauter Mechanismus von der S3-Unterstützung mancher Agentenrahmenwerke angeboten, wie z.B. von JADE (Bellifemine u. a. 2007, 2005). Sie helfen dem Agentensystem-Entwickler bei der Vermeidung von Fehlern während der Erzeugung bzw. Auswertung von Nachrichteninhalten aus Artefakten des Weltmodells, indem sie ihm ermöglichen, Artefakte aus dem Weltmodell direkt als Nachrichteninhalte zu definieren und ihm damit den Zwischenschritt des Codierens abnehmen, bzw. indem sie beim Empfang den serialisierten Nachrichteninhalt gegen die Kommunikationsontologie validieren und ggf. direkt in ein Artefakt des Weltmodells auf Empfängerseite decodieren.

Ontologieübersetzung gegen Kommunikationsontologie-Störungen

Falls das Weltmodell von Sender- und Empfängeragent auf unterschiedlichen Ontologien basiert, besteht prinzipiell die Möglichkeit, sich auf eine dieser Ontologien zu einigen oder eine dritte Ontologie als Kommunikationsontologie zu definieren. In beiden Fällen muss auf mindestens einer Seite der Konversation eine Übersetzung zwischen zwei Ontologien erfolgen. Die Möglichkeit der statischen Ontologieübersetzung der Instanzen zwischen der Senderontologie und der Empfängerontologie (bzw. der Kommunikationsontologie) beschreibt Scholz u. a. (2006).

Dabei wird mit statischen Regeln eine Abbildung zwischen den Schemata der zu übersetzenden Ontologien definiert und zur Laufzeit werden diese Regeln auf die konkreten Instanzen angewendet. Das beschriebene Verfahren zur Ontologieübersetzung erfordert in der Regel keine zusätzlichen Koordinationsschritte und kann deshalb vollständig in

S3 implementiert werden. Besonders bei spontaner Kooperation zwischen Agenten kann es jedoch vorkommen, dass sich die Regeln zur Ontologieübersetzung nicht lokal finden lassen. Dann muss in Schicht S5 versucht werden, kooperativ zur Laufzeit die weitergereichte Kommunikationsontologie-Störung zu beheben.

Dynamische Bedeutungsaushandlung gegen Kommunikationsontologie-Störungen

Die kooperative Störungsbehandlung gegen Kommunikationsontologie-Störungen auf Schicht S5 erfolgt durch den sehr anspruchsvollen Ansatz der dynamischen Bedeutungsaushandlung. Dabei wird die Bedeutung der Terme einer Ontologie im Rahmen einer gesonderten vorgeschalteten Konversationsphase zwischen den kommunizierenden Agenten ausgehandelt (Mena u. a. 2000). In dieser Phase versuchen die beteiligten Agenten ausgehend von einer gemeinsamen Basis-Ontologie, die Bedeutungen zu erlernen, die sie gegenseitig mit bestimmten Termen verbinden.

Ist auch die dynamische Bedeutungsaushandlung nicht von Erfolg gekrönt, dann kann keine weitere Störungsbehandlung mehr innerhalb des Agentensystems zur Laufzeit erfolgen und ein Benutzer- oder Entwicklereingriff wird notwendig. Der Fehlerbehandlungspfad „Ontologiestörungen“ endet damit hier.

5.3 Pfad „Rechnerstörungen und unerwartetes Agentenverhalten“

Wenn wichtige Teile der Infrastruktur eines Multiagentensystems ausfallen, dann kann zunächst vor Ort in den niederen Schichten versucht werden, die vorliegende Rechnerstörung zu beheben. Wenn dies dort nicht gelingt oder wenn sich Agenten im Verbund nicht wie in der Koordination vereinbart verhalten, dann kann es notwendig werden, Teile des Systems in den höheren Schichten zurückzusetzen. Diese Störungen und die zugehörigen konkreten Fehlerbehandlungsmechanismen bilden den Pfad „Rechnerstörungen und unerwartetes Agentenverhalten“, der in Abbildung 5.3 dargestellt ist.

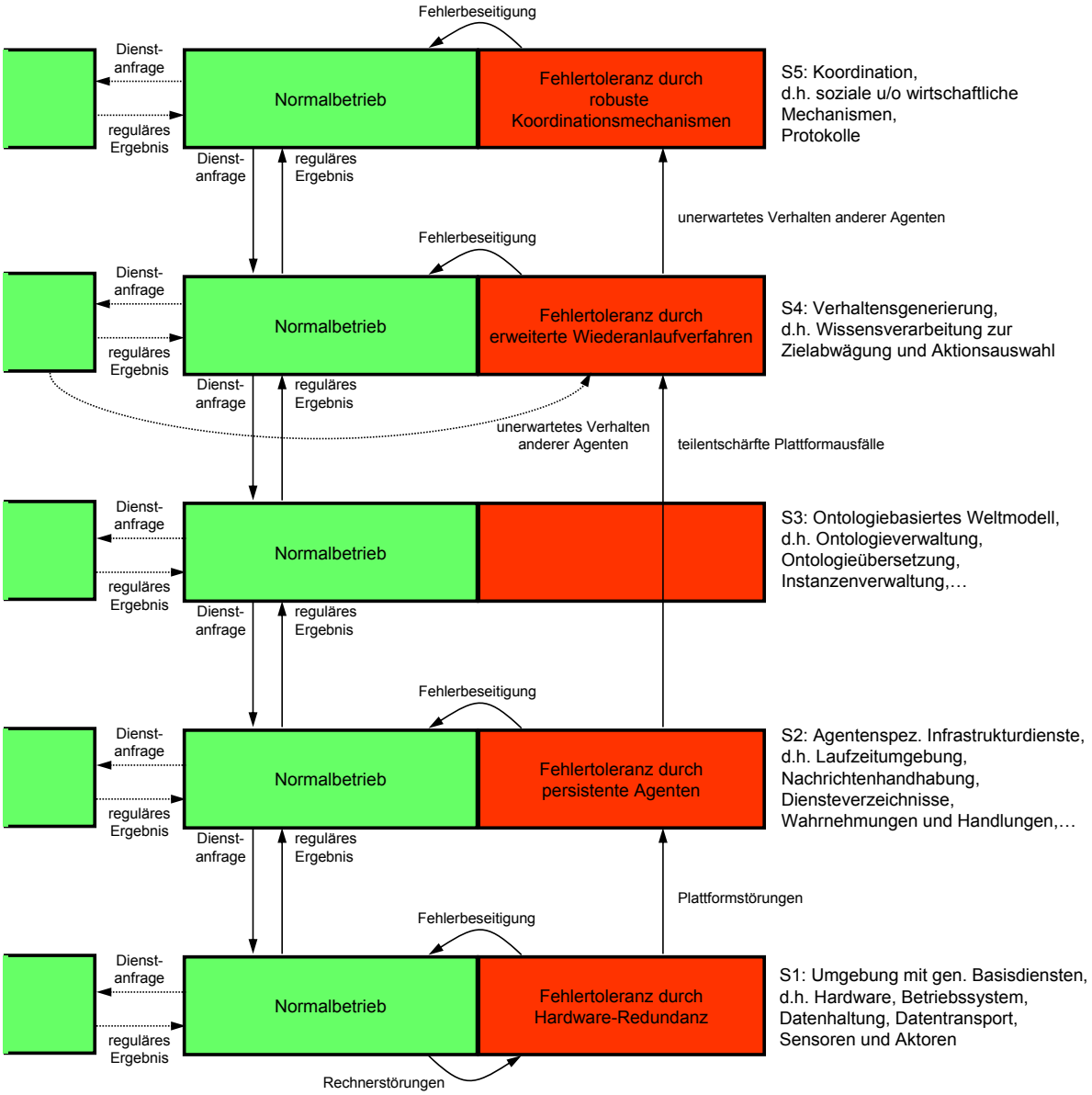


Abbildung 5.3: Pfad „Rechnerstörungen und unerwartetes Agentenverhalten“

Hardware-Redundanz gegen Rechnerstörungen

Gegen die in Schicht S1b beheimateten Rechnerstörungen (Hardware-Ausfälle, Betriebssystem- und Datenhaltungsfehler) bringen moderne Rechnerplattformen ihre eigenen Fehlerbehandlungsverfahren mit, die eine Reihe von effektiven Fehlzuständen häufig auf Basis von Hardware-Redundanz zu beseitigen verstehen. Als Beispiele für diese nicht-agentenspezifischen Mechanismen seien hier Multiprozessoren (Bartlett 1978), Spiegel- und RAID-Festplatten (Patterson u. a. 1988) genannt. Gibt es in S1b keine Redundanz bzw. kann diese nicht erfolgreich eingesetzt werden, eskaliert der Fehler zunächst nach S2b, in der Mechanismen existieren können, die Agenten helfen, eine Rechnerstörung zu überdauern.

Persistente Agenten gegen Plattformstörungen

Die für die Dauerhaftigkeit der Agenten notwendigen Dienste – sofern generisch bezüglich der konkreten Agenten – sind in die Agentenrahmenwerke integrierbar. Beispiele für derartige Persistenzdienste bieten die Rahmenwerke Cougaar (Helsinger und Wright 2005) und JADE (Bellifemine u. a. 2007, 2005). Die Agenten einer Cougaar-Plattform arbeiten gemeinsam an einem Anwendungsproblem, dessen Beschreibung auf einem verteilten Blackboard abgelegt wird, das von den bearbeitenden Agenten verwaltet wird. Falls ein Agent an einem Teil der Problembeschreibung interessiert ist, den er nicht selbst verwaltet, kann er diesen bei dem verwaltenden Agenten „abonnieren“. Wird ein Teil der Problemlösung modifiziert, benachrichtigt der verwaltende Agent alle Abonnenten. Bei dieser Vorgehensweise wird bewusst in Kauf genommen, dass sich abhängig von der Verbreitung der Modifikationen den verschiedenen beteiligten Agenten ein inkonsistentes Bild des Anwendungsproblems präsentiert.

Beim Ausfall eines Cougaar-Agenten geht nicht nur der interne Zustand des Agenten verloren, sondern ein Teil der Problembeschreibung, wodurch die Lösung des Anwendungsproblems im Allgemeinen unmöglich wird. Als Vorsorgemaßnahme werden daher regelmässig Abbilder des verwalteten Blackboard-Fragments, der zugehörigen Abonnement-Beziehungen und des Aktualisierungszustands gespeichert (BBN Technologies 2004b, a). Dies kann abhängig vom Anwendungsproblem und der Plattformstabilität in einem

pessimistischen Modus nach jeder Blackboard-Änderung in Form einer lokalen Transaktion oder in einem optimistischen Modus periodisch erfolgen. Beim pessimistischen Verfahren kann nach dem Wiederanfahren der lokale Blackboard-Zustand restauriert werden und die noch nicht versendeten Aktualisierungsnachrichten werden aus einem „Retransmission“-Log an die Abonnenten übermittelt. Der optimistische Modus erlaubt das Propagieren einer Modifikation zu den Abonnenten, bevor diese lokal persistent festgeschrieben wurde. Im Falle eines Wiederanfahrens wird versucht, aus dem evtl. veralteten lokalen persistenten Zustand und den vormals an die Abonnenten übermittelten Modifikationen einen möglichst aktuellen Zustand zu rekonstruieren. Da der Restaurationsablauf vom Anwendungsproblem und dessen Beschreibung abhängt, muss der Agentenentwickler bei Verwendung des optimistischen Verfahrens „RestartLogicProvider“ definieren, die den Ablauf anwendungsspezifisch übernehmen. In der Zuverlässigkeitsreferenzarchitektur entspricht das Vorgehen im optimistischen Ansatz einer Weitergabe des Ausfalls und einer Fehlerbehandlung auf den speziell erweiterten Schichten S4 und S5.

Auch das Agentenrahmenwerk JADE bietet eine rudimentäre Erweiterung für persistente Agenten (JADE Persistence Add-On, Rimassa 2004). Der Persistenzdienst erlaubt prinzipiell das Speichern des Zustands einzelner Agenten oder einer kompletten Plattform in einer lokalen Transaktion. Dabei werden die generischen Elemente der Schicht S2, wie z.B. der Zustand der Plattform, der Lebenszykluszustand der einzelnen Agenten und deren Nachrichtenwarteschlangen automatisch unterstützt. Um auch den inneren Zustand eines Agenten speichern zu können, muss sich der Agentenentwickler an bestimmte Einschränkungen bezüglich der Gestaltung der Agentenlogik halten. Dies entspricht – wie beim optimistischen Ansatz von Cougaar – einer Ausdehnung der Vorsorgemaßnahmen auf die Schicht S4, d.h. einer engen Vermaschung der verschiedenen Programmieraspekte über die Schichten hinweg. In der aktuell vorliegenden Version des Persistenzdienstes findet weder das Abspeichern des Zustandes automatisiert statt, noch wird die Restauration nach einem Wiederanlaufen einer Plattform transparent unterstützt. Zusammenfassend lässt sich feststellen, dass der Ansatz der persistenten Agenten ohne Einbeziehung der oberen Schichten der Zuverlässigkeitsreferenzarchitektur nur eingeschränkte Garantien bieten kann.

Erweiterte Wiederanlaufverfahren gegen teilentschärfte Plattformausfälle und gegen unerwartetes Verhalten anderer Agenten

Die Wiederherstellung persistenter Agenten in S2 ist – in Abwesenheit jeglicher Anwendungssemantik – ein rein mechanischer Vorgang: Der Zustand des Agenten vor dem Ausfall wird syntaktisch möglichst genau rekonstruiert. Dieser rekonstruierte Zustand bietet dann der Schicht S4 eine Ausgangsbasis für die erweiterten Wiederanlaufverfahren, die einen Zustand anstreben, in dem eine Weiterführung der Arbeiten im Normalbetrieb möglich wird, d.h. in dem eine erneute Zielabwägung und Aktionsauswahl angestoßen werden kann. Die erweiterten Wiederanlaufverfahren der Agenten schreiben hierfür Protokolle ihrer Aktionen und erlauben damit auch die Effekte der Aktionen, die beim Ausfall evtl. nur unvollständig ausgeführt werden konnten, rückgängig zu machen.

Eine intuitive und mächtige Möglichkeit der Buchführung ist die baumartige Darstellung der geplanten und durchgeführten Aktionen und ihrer Ablaufzusammenhänge in Form der offen geschachtelten Transaktionen nach Nagi (2001b). Dabei beschreibt jeder Blattknoten eine Aktion und ihre semantische Kompensation, die im Falle eines Rücksetzens nach Scheitern des Agentenplanes zum Tragen kommt. Die inneren Knoten geben die Ablaufzusammenhänge wieder (z.B. sequenzielle oder parallele Ausführung von Unterbäumen). Im Normalbetrieb wird der Transaktionsbaum in Vorwärtsrichtung durchlaufen und die Aktionen der Blattknoten werden als ACID-Transaktionen (siehe Anhangsabschnitt C.2) ausgeführt. Im Ausnahmebetrieb erfolgt das Rücksetzen durch Ausführung der Kompensationsaktionen in umgekehrter Baumreihenfolge.

Das beschriebene Verfahren nutzt mit seinen Aktionsplänen nur den Teil der Fehlerbehandlungsoptionen auf Schicht S4, der durch die Aktionsauswahl definiert wird. Weitere Spielräume zur Fehlerbehandlung erhält man, indem man auch die Zielabwägung mit einbezieht, wie dies bei der Ziel-basierten semantischen Kompensation von Unruh (Unruh u. a. 2004, 2005; Wang u. a. 2007) der Fall ist. Äußerlich weist der Ansatz eine gewisse Ähnlichkeit mit den offen geschachtelten Transaktionen auf, da auch ihm eine Baum-artige Buchführung der Verhaltensgenerierung zugrunde liegt. Die Bäume stellen jedoch keine Pläne über die Aktionen dar, sondern eine hierarchische Zerlegung der Ziele. Im Falle des Wiederanlaufens wird rückwärts entlang der Zielhierarchie der Agenten

semantisch zurückgesetzt. Falls dabei ein Wiederaufsetzpunkt hergestellt werden kann, der dem Erreichen eines (Unter-)Ziels der Zielhierarchie entspricht, werden die Aktionen, die zum Erreichen des Gesamtziels notwendig sind, umstrukturiert und das Gesamtziel wird erneut angestrebt.

Aridor und andere (in Sher u. a. 2001) verfolgen einen Ansatz, der auf lokalen transaktional geschützten Plänen für mobile Agenten basiert – ähnlich wie Nagi eingeschränkt auf den lokalen Fall. Die lokalen Pläne erlauben eine sehr flexible Transaktionssemantik. Die Allgemeinheit des Ansatzes ist eingeschränkt, weil er einen sehr spezifischen Multiagentensystem-Begriff voraussetzt: Ein Multiagentensystem entsteht, indem sich einzelne Agenten zum Erledigen spezieller Aufgaben aufteilen und nach der Lösung der Teilprobleme wieder vereinen. Die Umsetzung des Mechanismus wäre vor diesem Hintergrund auf den Schichten S4 und S5 einzuordnen, wobei lediglich Störungen in S4 und tiefer adressiert werden.

Einen Ansatz, der verteilte Transaktionen und innere Architektur nach dem BDI-Paradigma (Beliefs, Desires and Intentions) zusammenführt, stellt Busetta (1999) vor. Die Zusammenführung erfolgt nicht über die explizite Verwendung von Transaktionen oder der formalen Semantik der inneren Architektur, sondern über einen spezialisierten BDI-Interpreter („TOMAS“), der die Verhaltensgenerierung in S4 übernimmt (Busetta und Ramamohanarao 1998; Busetta u. a. 2003). Der Ansatz ist also gegenüber der Implementierung der Schicht S4 nicht generisch.

Pitoura präsentiert einen Ansatz auf Schicht S4 zum Schutz mobiler Agenten gegen Plattformausfälle durch Transaktionen. Der Ansatz zielt nicht nur auf das Umziehen der Agenten ab, sondern auch auf deren konfliktfreie Ausführung im Sinne der Isolation (Pitoura 1998; Pitoura und Bhargava 1995). Zur inneragentischen Synchronisation wird eine Konfliktrelation zwischen den Agentenaktionen („methods“) definiert. Für die Synchronisation zwischen verschiedenen Agenten gibt es spezielle Aktionen („breakpoint“ und „delegation“), die bestimmte Kooperationsmuster unterstützen. Überwacht wird die Einhaltung der Konfliktrelation und der Synchronisation durch jeweils einen speziellen Agenten, der der Infrastruktur zuzurechnen ist. Aufbauend auf diesen Primitiven wird ein Transaktionsmodell mit einem angepassten Korrektheitsbegriff für mobile Agenten definiert. Darüber hinaus wird die Systemarchitektur einer Umsetzung vorgestellt, die

einen Scheduler für das Transaktionsmodell realisiert, der die definierte Korrektheit garantiert.

Wie das Verfahren von Pitoura motiviert, kann das Rücksetzen der beschriebenen erweiterten Wiederanlaufverfahren prinzipiell nicht nur gegen teilentschärfte Plattformausfälle eingesetzt werden, sondern auch bei Störungen durch unerwartetes Verhalten anderer Agenten. Im Allgemeinen ist hierbei jedoch das isolierte Zurücksetzen eines einzelnen Agenten nicht immer zielführend, sondern es müssen entweder alle kooperierenden Agenten eines Verbandes auf einen gemeinsamen konsistenten Wiederaufsetzpunkt zurückgesetzt werden oder die eingesetzten Koordinationsmechanismen müssen gegen Inkonsistenzen und unerwartetes Verhalten robust sein.

Erweiterte Wiederanlaufverfahren müssen also entweder selbst die Schicht S5 zum koordinierten Rücksetzen mit einbeziehen oder durch zusätzliche geeignete Koordinationsverfahren ergänzt werden, d.h. es muss im Sinne der Zuverlässigkeitsreferenzarchitektur ein abgemilderter Fehler an S5 weitergegeben werden. Der Ansatz von Nagi bezieht durch eine Erweiterung der offen geschachtelten Transaktionen mittels eines Nachrichten-basierten Synchronisationsmechanismus die Schicht S5 direkt mit ein. Der nächste Abschnitt beschreibt zusätzliche Mechanismen im Pfad, die die erweiterten Wiederanlaufverfahren gegen unerwartetes Verhalten anderer Agenten in S5 unterstützen.

Robuste Koordinationsmechanismen gegen unerwartetes Verhalten anderer Agenten

Robuste Koordinationsmechanismen versuchen unerwartetes Verhalten anderer Agenten in der Koordination zu berücksichtigen und zumindest auf lange Sicht dagegen unanfällig zu werden, d.h. die gewünschten Dienste auch trotz des Fehlverhaltens erbringen zu können. Die im folgenden geschilderten robusten Koordinationsmechanismen sind sozial, ökonomisch oder organisatorisch motiviert und zielen speziell auf nicht böswilliges unerwartetes Verhalten ab.

Kaminka und Tambe führen in die Koordinationsschicht S5 spezielle Monitoring-Agenten ein, die eine sog. *Soziale Diagnose* durchführen und die überwachen, ob sich die kooperie-

renden Agenten an die vereinbarten Team-Vorgaben halten (Kaminka und Tambe 1998). Ähnliche Ansätze gibt es auch von Minsky (*Law-governed Interaction*, Minsky und Murata 2003) und von Hägg (*Sentinels*, Hägg 1997). Ein Kritikpunkt an den robusten sozialen Mechanismen ist die Überwachung der Anwendungsagenten durch andere übergeordnete Agenten oder Komponenten, die konzeptionell deren Autonomie einschränken. Dies wird jedoch zur Durchsetzung von gemeinsamen globalen Zielen bewusst in Kauf genommen.

Ökonomisch motiviert ist das sog. *Leveled-Commitment Contracting* nach Sandholm (Sandholm und Lesser 2001, 2002). Dabei wird Agenten die Möglichkeit gegeben, von einmal geschlossenen Verträgen zurückzutreten (engl. *Decommitment*). Im Gegenzug muss der zurückgetretene Diensterbringer für die nicht erbrachte Leistung eine vorher vereinbarte Strafe an den dienstnehmenden Agenten bezahlen. Es erfolgt also eine finanzielle Kompensation (siehe auch Sorge 2004).

Der Grund für den Rücktritt von einem Dienstleistungsvertrag durch den Diensterbringer kann in der Unfähigkeit liegen, den Dienst vereinbarungsgemäß zu erbringen – ein versehentliches Fehlverhalten. Es könnte sich aber auch nach Abschluss des Vertrages eine Möglichkeit aufgetan haben, die Dienstleistung teurer an einen anderen Dienstnehmer zu verkaufen und zwar so teuer, dass nach Abzug der Vertragsstrafe mehr eingenommen werden kann, als durch die ursprüngliche Entlohnung. Es liegt also ein absichtliches aber zugleich wirtschaftlich vernünftiges Fehlverhalten vor. Um die Wahrscheinlichkeit der Vertragseinhaltung zu erhöhen („*Pacta sunt servanda.*“), sind Erweiterungen der Konversationsprotokolle durch zusätzliche Verhandlungsphasen möglich. Aknine u. a. (2004) schlagen eine Erweiterung des Kontraktnetzprotokolls um eine Vorbuchungsphase vor. Durch Simulation lassen sich eine bessere Ressourcenausnutzung und geringere Fehleranfälligkeit gegen Vertragsrücktritte nachweisen.

Neben den geschilderten sozialen und ökonomischen Ansätzen zur Laufzeit gibt es auf der Koordinationsschicht auch Ansätze organisatorischer Natur. Ein bekannter und umfassend evaluierter Ansatz ist das *Knowledge-based Exception Handling* von Klein und Dellarocas. Dabei werden die Fehler und das Ausnahmeverhalten explizit modelliert und im Fehlerfall in das Agentenverhalten dynamisch eingebaut (Klein und Dellarocas 2000; Klein u. a. 2001; Klein 2003; Dellarocas u. a. 2000; Dellarocas und Klein 2000).

Das typische Problem eines derartigen Ansatzes, dass alle potenziellen Fehler bereits im Vorfeld bekannt sein müssen, wird dadurch adressiert, dass die Fehlerfälle und ihre Fehlerbehandlungsmaßnahmen dauerhaft über Anwendungs- und Projektgrenzen hinweg in einer globalen, zentralisierten Knowledge-Base abgelegt werden. Zentralisierung wird jedoch in der Agentenwelt üblicherweise gerne vermieden, weshalb dieser Ansatz nicht unumstritten ist. Die beschriebenen Fehler und Ausnahmeverhalten sind überwiegend der Schicht S5 zuzuordnen (in Teilen auch S4), wobei man die Knowledge-Base selbst als Unterkomponente der agentenspezifischen Infrastruktur in Teilschicht S2b auffassen kann.

Der Pfad „Rechnerstörungen“, der an dieser Stelle endet, hat gezeigt, wie sich eine Störung bei ihrer Eskalation durch die Zuverlässigkeitsreferenzarchitektur von Schicht zu Schicht verändern kann – von Rechner- zu teilentschärften Plattformstörungen – und wie weitere Partnerstörungen – hier das unerwartete Verhalten anderer Agenten – im Verlaufe eines Pfades durch die steigende Mächtigkeit der Mechanismen in diesen Pfad münden können.

5.4 Pfad „Netzwerkstörungen“

Kurze und längerfristige Ausfälle im Daten- oder Nachrichtentransport und ihre Wirkungen in den höheren Schichten sind die Störungen, die im Pfad „Netzwerkstörungen“ (siehe Abbildung 5.4) durch Mechanismen der Infrastrukturschichten und der Koodinationsschicht behandelt werden.

Fehlerkorrigierende Übertragungsprotokolle und Aushandlung der Dienstgüte gegen Datentransportstörungen

In der (agentenunspezifischen) Datenkommunikation der Schicht S1c können, z.B. durch Medien- oder Übertragungskomponentenfehler, Störungen im Datentransport auftreten, die von Übertragungsverzögerungen bis hin zu Verbindungsabbrüchen reichen. Viele dieser Störungen können selbstständig von fehlerkorrigierenden Übertragungsprotokollen (siehe Iren u. a. 1999) beseitigt werden. Diese beruhen z.B. auf Paketnummerierungen,

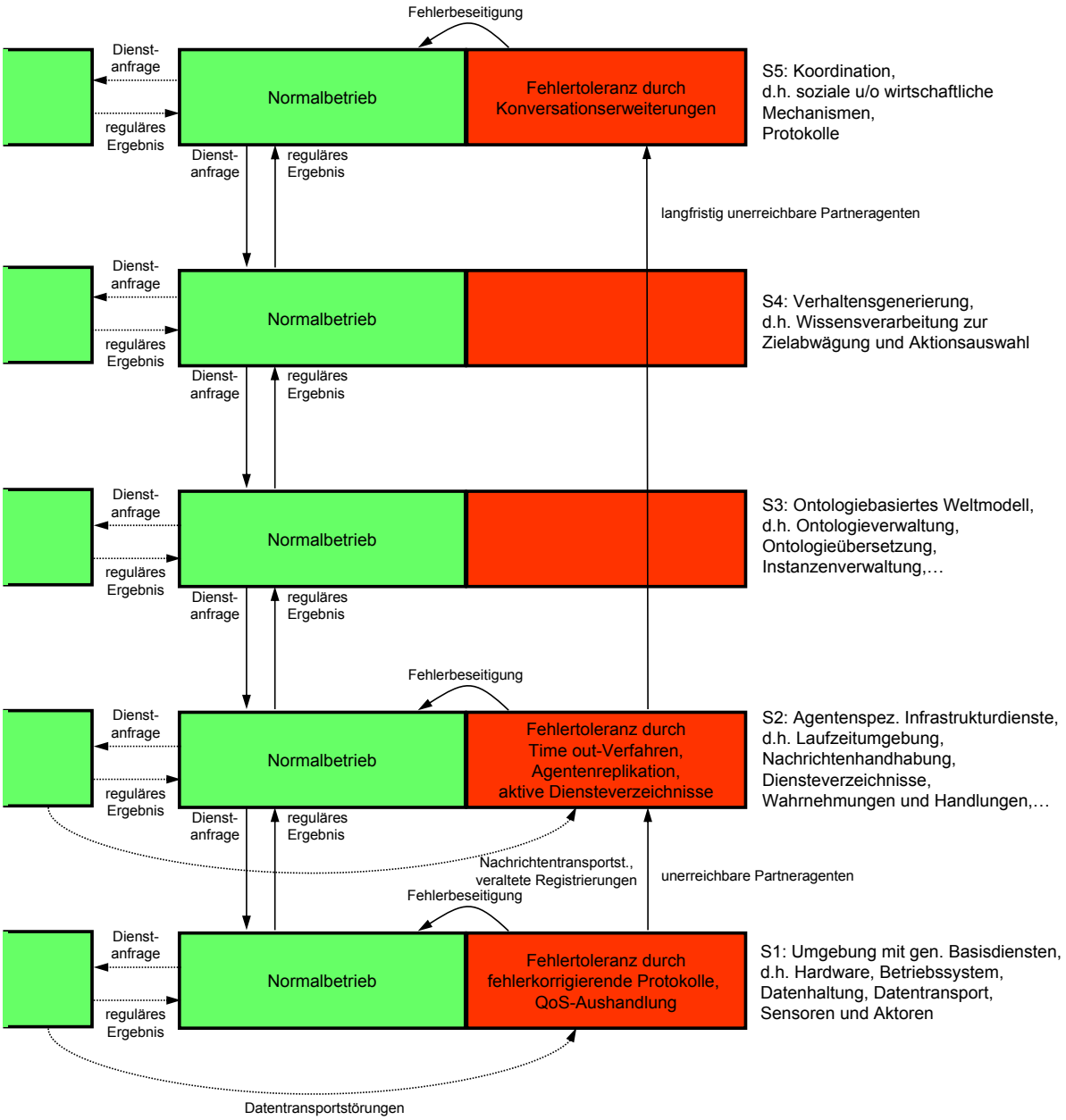


Abbildung 5.4: Pfad „Netzwerkstörungen“

zusätzlich übertragenen Prüfsummenzeichen und Quittungen. Für eine höhere erreichbare Qualität des Datentransports muss natürlich ein höherer Aufwand betrieben werden, der über eine Dienstnutzungsvereinbarung reguliert werden kann. Für die verbleibenden Störungen besteht die Möglichkeit einer Dienstgüteaushandlung, z.B. in Bezug auf Paketverluste, Paketduplizierungen oder Reihenfolgevertauschungen, für die verschiedene Klassen von Garantien vereinbart werden können: at-least-once, at-most-once, exactly-once, best effort bzw. order-preserving (siehe Halsall 1998).

„Time out"-Verfahren gegen Nachrichtentransportstörungen

Die Datentransportstörungen aus Schicht S1c können bei erfolgloser Behandlung in S2 zu verlorenen Nachrichten zwischen zwei Agenten führen. Dort bieten einige Plattformen für solche Fälle einfache „Time out"-Verfahren an. So kann man z.B. in der Agentenplattform JADE (Bellifemine u. a. 2007, 2005) mit jeder zu versendenden Nachricht angeben, bis zu welchem Zeitpunkt man eine Antwort vom Empfänger erwartet. Trifft die Antwort nicht vor Ablauf der Frist ein, z.B. durch einen Datentransportfehler auf dem Hin- oder Rückweg oder auch durch eine übermäßig lange Bearbeitungszeit beim Empfänger, dann wird ein Ausnahmeereignis ausgelöst und eine spezifische Fehlerbehandlungsroutine aufgerufen, die vom Entwickler vorab definiert werden muss. Das Verhalten, das in dieser Routine spezifiziert wird, ist keiner Einschränkung unterworfen: Es kann von einem einfachen Versuch, die Nachricht erneut zu versenden, bis zu einer kompletten Umorientierung in der Problemlösungsstrategie reichen. Letzteres entspräche einer Weitergabe des Fehlers zur Fehlerbehandlung an eine höhere Schicht (S4 bzw. S5).

Der Fehlerfall 2 im Kontraktnetz-Beispiel aus Kapitel 2 kann mit einem einfachen „Time out"-Verfahren kaum sinnvoll bekämpft werden. Der Grund dafür liegt in der langen Zeitspanne, die zwischen der `accept-proposal`-Nachricht zur Auftragsvergabe vom `OrderAgent` und der `inform`-Antwort zur Produktionsbestätigung durch den erfolgreichen `MachineAgentA` üblicherweise verstreicht. Eine Frist auf die `accept-proposal`-Nachricht, die angibt, bis wann die `inform`-Nachricht empfangen werden muss, müsste also später als der geplante Fertigstellungstermin gesetzt werden. Besonders bei frühzeitiger Planung, d.h. bei langer Wartezeit bis zum Produktionsbeginn, kann es dann für eine erneute Durchführung einer CNP-Planung bei Fristablauf bereits zu spät sein. Dennoch

wäre durch die Einführung zusätzlicher Bestätigungsnachrichten ein wesentlich früheres Anstoßen einer Fehlerbehandlungsroutine möglich, weil damit nicht auf das Verstreichen der Wartezeit bis zum Produktionsbeginn und der Produktionsdauer gewartet werden müsste und die Zeitdauern, die auf den Nachrichtenversand und die Erstellung und Auswertung der Angebote entfallen i.A. gegenüber den Realweltzeitspannen vernachlässigbar sind.

Eine andere Variante des „Time out“-Verfahrens könnte eine bessere Fehlerbehandlung realisieren, bedarf aber einer Abstimmung des Ausnahmeverhaltens zwischen den Order- und MachineAgents. Bei einer Frist auf die `propose`-Nachricht durch die MachineAgents würde nach Ausbleiben einer Antwort in Form einer `refuse-proposal-` oder `accept-proposal`-Nachricht auf Seiten des `MachineAgentA` die Fehlerbehandlungsroutine angestoßen. Wenn die Routine auf das Ausbleiben der Antwort durch ein erneutes Senden reagiert, dann sollten zwei identische Angebote beim `OrderAgent` eingehen, zuerst das Originalangebot, auf das eigentlich eine Zustimmung erfolgen sollte, und dann das Wiederholungsangebot. Aus der wiederholten Nachricht könnte der `OrderAgent` schließen, dass seine eigene Nachricht verloren gegangen sein muss und seinerseits eine wiederholte Übertragung der `accept-proposal`-Nachricht initiieren.

Wie in Abschnitt 4.4.2 beschrieben ist ein so mächtiges aber auch komplexes „Time out“-Verfahren in der Regel nicht generisch für beliebige Konversationen alleine auf Schicht S2c zu realisieren. Die im letzten Absatz angesprochene notwendige Abstimmung zwischen den Order- und MachineAgents entspricht einem Eingriff in die Verhaltens- und Mechanismengestaltung und geschieht damit in den höheren Schichten des „Netzwerkstörungen“-Pfades.

Agentenreplikation gegen unerreichbare Partneragenten

Unerreichbare Partneragenten können – wie in diesem Pfad – als Folge von Datentransportstörungen auftreten oder aus Rechnerstörungen entfernter Knoten resultieren. Eine tiefschichtige Möglichkeit zur Störungsbehandlung unerreichbarer Partneragenten bieten die agentenspezifischen Infrastrukturdienste in S2, in denen versucht werden kann, für jeden Agenten mehrere gleichwertige Replikate zu halten und dann im Störungsfall transparent zwischen den verfügbaren Replikaten umzuschalten.

Beispiele für derartige Ansätze sind die „redundant broker teams“ von Kumar und Cohen (Kumar und Cohen 2000), bei denen die Dienstvermittleragenten repliziert vorliegen, oder die allgemeineren Ansätze, die auf der transparenten Replikation beliebiger Agenten beruhen (z.B. Guessoum u. a. 2002; Fedoruk und Deters 2002).

Bei idealer Realisierung ist das Risiko eines Dienstausfalles umgekehrt proportional zur Anzahl der Replikate des Dienstes. Es resultiert ein Optimierungsproblem zwischen den Kosten für die Replikation und den Kosten eines möglichen Dienstausfalls. Ein universelles spieltheoretisches Verfahren zur Berechnung der Zuverlässigkeit bei Replikationsmechanismen stellen Rudnianski und Bestougeff (2004) vor. Krempels (2001) verknüpft seinen Fehlerbehandlungsansatz durch Replikation bei mobilen Agenten mit einem spezifischen Berechnungsmodell unter Verwendung von Markov-Ketten. Exemplarisch für die Klasse der Zuverlässigkeitsmechanismen auf Basis der Agentenreplikation soll dieser Ansatz im folgenden ausführlicher betrachtet werden.

Dem Ansatz liegt ein Cluster von Agentenplattformen auf verschiedenen Rechnerknoten zugrunde, bei dem jeder Agent auf allen Knoten des Clusters repliziert wird. Im Fehlerfall ersetzt ein replizierter Agent auf einem anderen Knoten die Dienste des ausgefallenen Agenten. Einer der Knoten im Cluster übernimmt eine Führungsrolle („Leader“) und bestimmt, welches Replikat aktiv werden soll. Damit der Leader nicht selbst zum kritischen Element („single point of failure“) im Cluster wird, sollte er dynamisch ausgewählt werden. Dazu wird ein angepasster Auswahlalgorithmus (nach Garcia-Molina 1982) zur Führerbestimmung eingesetzt, der fehlertolerant gegen Verbindungsfehler ist.

Ein Agent, der einen Dienst im Cluster erbringen will, muss hierfür ein Token mit einer vorgegebenen Verfallszeit („Lease-time“) vom Leader erwerben und ggf. vor Ablauf der Frist erneuern. Die Fehlerbeseitigung erfolgt in zwei Phasen: In der ersten Phase überwacht der Leader, ob die Verfallszeit eines Agenten abläuft, ohne dass eine Erneuerung des Token erfolgt ist. In diesem Fall geht er von einem Ausfall aus, startet ein Replikat des betroffenen Agenten auf einem anderen Knoten. Danach beginnt die zweite Phase. Der Agent, der ursprünglich den Dienst erbracht hat, wird – falls er noch läuft – feststellen, dass er das Token nicht erneuern kann, z.B. weil keine Verbindung zum Leader möglich ist oder dieser die Erneuerung ablehnt. Als Reaktion hierauf kann der

ursprüngliche Dienstleister während eines vordefinierten Zeitfensters versuchen, die erreichten Zwischenergebnisse an seinen Ersatzagenten zu übertragen. Unabhängig vom Erfolg dieser Zustandsübertragung nimmt der Nachfolger nach Ablauf des potenziellen Übertragungszeitfensters den Dienst auf und nimmt dazu die Identität seines Vorgängers an.

Das Berechnungsmodell zu dem beschriebenen Ansatz erlaubt die Bestimmung einer unteren und oberen Schranke für die erforderliche Anzahl und Verfügbarkeit der Einzelplattformen, um eine gewünschte Verfügbarkeit und Wiederanlaufzeit des Gesamtsystems garantieren zu können. So lässt sich auch die Anzahl der notwendigen Knoten im Cluster bestimmen. Neben der erforderlichen Hardware für den Unterhalt der verschiedenen Rechnerknoten verursacht der Fehlerbeseitigungsansatz auch Kosten durch erhöhten Netzwerkverkehr z.B. durch das Auswahlverfahren für den Leader, durch die zyklische Erneuerung der Token und durch die Replikation der Agenten. Folglich wird der Ansatz nur für Netzwerke mit vergleichweise „günstigem“ Verbindungsaufbau empfohlen, wie z.B. für typische stationäre Internetanwendungen oder Ad hoc-Netze auf WLAN-Basis. Von einem Einsatz in Mobilfunknetzen mit ihren relativ teuren Verbindungen wird abgeraten.

Zur Illustration des Ansatzes betrachte man den Fehlerfall 3 im motivierenden Beispiel aus Kapitel 2, in dem der erfolgreiche `MachineAgent A` nicht innerhalb der zugesagten Frist liefern kann. Abweichend vom Beispiel sei angenommen, dass es sich bei dem herzustellenden „Produkt“ um eine Information handelt und nicht um einen physikalischen Gegenstand. Ein replizierter `MachineAgent` könnte hier in der Lage sein, die Information rechtzeitig abzuliefern, falls er nicht durch dieselbe Fehlerursache gestört wird, die auch schon den Ausfall des ursprünglichen diensterbringenden Agenten verursachte.

Konversationserweiterungen und Rekoordinierung gegen langfristig unerreichbare Partneragenten

Eine langfristige Unerreichbarkeit eines Partneragenten äußert sich zunächst auf Schicht S2 durch das Ausbleiben einer einzelnen Nachricht und eskaliert bei erfolg-

loser Behandlung im Verlauf auf Schicht S5 durch einen Abbruch bzw. ein Nicht-Zustandekommen einer Kooperationsbeziehung.

Im Falle einer bereits bestehenden Kooperationsbeziehung kann bei einer Eskalation auf die Koordinationsschicht durch Konversationserweiterungen zunächst festgestellt werden, ob es sich um eine temporäre oder permanente Unerreichbarkeit des Partneragenten handelt. Zur Illustration einer entsprechenden Konversationserweiterung betrachte man erneut das Beispiel aus der Motivation: In den Fehlerfällen 2 und 3 wird der `OrderAgent` nach Ablauf einer gewissen Zeit das Ausbleiben der Produktionsbestätigung feststellen. Er wird folgende mögliche Störungen in Betracht ziehen: Seine Beauftragung an den `MachineAgentA` könnte verloren gegangen sein (Fehlerfall 2) oder der Produktionsablauf könnte vorübergehend oder dauerhaft gestört sein (Fehlerfall 3) oder es ist lediglich die Benachrichtigung über die erfolgreiche Produktion selbst verloren gegangen. Wie bereits bei den Erörterungen zu den Time out-Verfahren beschrieben, könnte hier eine zusätzliche Nachfrage beim `MachineAgentA` durch den `OrderAgent` Klarheit schaffen. Die dafür erforderliche Erweiterung der Protokolle gehört ebenso zu S5, wie im Falle einer festgestellten permanenten Störung die erforderliche Aufstellung eines alternativen Kooperationsplans, d.h. die Rekoordinierung.

Da die Auswahl eines passenden Dienstgebers für ein Dienstgesuch meist von Informationen abhängt, die nicht in der Infrastruktur enthalten sind, wie z.B. von der Reputation des Dienstgebers, kann die Auswahl eines alternativen Dienstanbieters in der Regel nicht durch die Agentenplattform automatisch erfolgen, sondern der Ausfall muss bis Schicht S5 weitergegeben werden, in der ein alternativer Anbieter ermittelt werden kann oder möglicherweise ein völlig neues Dienstnutzungsmuster mit andersartigen verfügbaren Diensten gefunden werden kann. Mit der beschriebenen Rekoordinierung, die in Schicht S5 in der Regel im Normalbetrieb erfolgen kann, endet der Pfad der Netzwerkstörungen.

5.5 Weitere konkrete Mechanismen

Neben den konkreten anspruchsvollen Verfahren und den illustrierenden Beispielen, die in den obigen Pfaden vorgestellt wurden, gibt es noch weitere prominente Ansätze, die

zwar keinem der Pfade direkt zugeordnet werden können, die zur Vervollständigung der Zuverlässigkeitslandkarte aber dennoch in diesem Abschnitt vorgestellt werden sollen.

Aktive Diensteverzeichnisse gegen Registrierungsstörungen

Registrierungsstörungen treten auf, wenn versucht wird, auf einen Agenten zuzugreifen, dessen Dienste zwar noch in einem Verzeichnis publiziert sind, der aber durch eine Rechner- oder Netzwerkstörung nicht mehr erreichbar ist. In den Spezifikationen der FIPA und den konformen Plattformen (wie z.B. JADE) erfüllt der Directory Facilitator (siehe oben in Abb. 3.12) die Funktion des Diensteverzeichnisses.

Gegen veraltete Dienstregistrierungen helfen aktive Directory Facilitators, die periodisch überprüfen, ob die publizierten Dienste noch verfügbar sind. Sie wurden beispielsweise in den Systemen *Agent.Enterprise* und *Agent.Hospital* des SPP1083 eingesetzt (siehe Woelk u. a. 2006; Nimis und Stockheim 2004; Kirn u. a. 2006a). In diesem Fall wurde die Störungsbehandlung durch aktive Diensteverzeichnisse teilweise zusätzlich mit der Agentenreplikation kombiniert. Im Falle eines Dienstangebotes durch einen zwischenzeitlich unerreichbaren Agenten wird das Dienstangebot nicht nur einfach aus dem Verzeichnis gelöscht, sondern ein Dienstgesuch wird direkt auf eines der anderen registrierten Replikate umgeleitet, die denselben Dienst anbieten.

Erweiterte Umzugsverfahren gegen Mobilitätsstörungen

Mobile Agenten können während ihres Lebenszyklusses von einer Agentenplattform zu einer anderen umziehen, um z.B. kürzere Zugriffszeiten auf eine beobachtete Umgebung zu erlangen. Treten während der Umzugsphase Störungen auf, die den Erfolg des Umzugs gefährden, spricht man von Mobilitätsstörungen. Als Zuverlässigkeitsmechanismen gegen Mobilitätsstörungen werden in Schicht S2 erweiterte Umzugsverfahren eingesetzt, die den Ausfall oder die Verdoppelung eines Agenten verhindern sollen.

Ein erweitertes Umzugsverfahrens durch Klonen stellt Pleisch (2002) vor, der die gesamte Lebensdauer eines Agenten unter den Schutz einer lokalen geschachtelten Transakti-

on stellt und damit eine verlustfreie „exactly-once“-Semantik selbst beim Umzug mobiler Agenten garantiert. Vogler entwirft eine Agentenplattform für mobile Agenten, die durch verteilte Transaktionen die „exactly-once“ Semantik realisiert (Vogler 1999; Vogler und Buchmann 1998; Vogler u. a. 1997). Die Funktionsweise ist dem Ansatz von Pleisch ähnlich, kommt aber ohne eine explizite Replikation der Agenten aus.

Protokollverifikation gegen Konversationsstörungen

Bei den Fehlerbehandlungsmechanismen auf Schicht S5 lassen sich drei grundsätzliche Klassen unterscheiden: Mechanismen, die zur Laufzeit zum Tragen kommen, Mechanismen für die Entwurfsphase und organisatorische Maßnahmen. Gegen die als Partnerstörungen auftretenden Konversationsstörungen, bei denen die beteiligten Agenten von unterschiedlichen Nachrichtenprotokollen und/oder -inhalten ausgehen, sind besonders die Mechanismen für die Entwurfsphase erfolgversprechend.

Eine große Anzahl der existierenden Zuverlässigkeitsmechanismen gegen Konversationsstörungen sind daher Erweiterungen des Konversations-Entwurfprozesses, d.h. sie verifizieren die Nachrichtenabfolgen und -inhalte. Paurobally u. a. (2003), Van Eijk u. a. (2003), Nodine und Unruh (2000), Galan und Baker (1999) und Hannebauer (2002) präsentieren Ansätze, die alle auf einer Anreicherung der Protokollspezifikationen durch ein formales Modell beruhen. Je nach Ansatz ermöglicht dies eine konstruktive oder analytische Protokollverifikation.

Während die bisher genannten Verfahren zur Protokollverifikation auf rein syntaktischer Ebene arbeiten, gibt es auch Ansätze, die versuchen, möglichst robuste Protokolle für einen konkreten Anwendungskontext zu entwerfen und deren Robustheit gegen bestimmte Fehlerfälle im Vorfeld des Einsatzes durch Simulation zu verifizieren. Ein Beispiel hierfür sind die Arbeiten von Schepperle und Yue (Yue 2007), die im Bereich der automatischen Steuerung von KFZ-Verkehrsflüssen robuste Protokolle („fail-safe“) entwickeln, indem sie die oben genannten bekannten Techniken aus der Telematik (Sequenznummern, Quittungen, Timer,...) auf die Ebene der Agentenprotokolle übertragen.

Bei den Verfahren zur Protokollverifikation wird vorausgesetzt, dass alle an der betreffenden Konversation beteiligten Agenten – oder zumindest deren Protokollautomaten – gemeinsam entwickelt werden.

Fehlerbehandlung durch die Benutzer

Trotz der breiten Palette an existierenden Mechanismen kann es Störungen geben, die nicht innerhalb des Multiagentensystems behandelt werden können und folglich für die Benutzer offenbar werden. Er bildet dann die „virtuelle Schicht S6“ in der Fehlerbehandlung, die sehr divergent ist und von einem direkten Eingriff in das Agentensystem bis zu einer völlig systemunabhängigen Handlung in der Umgebung oder ganz ohne Umgebungsbezug reichen kann.

Als ein extremes Beispiel einer Fehlerbehandlung durch den Benutzer könnte man sich vorstellen, dass sich das Fehlverhalten eines Multiagentensystems in der Realwelt in einem finanziellen Schaden niederschlägt, für den Schadensersatzansprüche z.B. gegen andere Agenten bzw. deren Prinzipale geltend gemacht werden können. Die rechtlichen Fragen, die im Zusammenhang mit Agenten durch deren Autonomie hervorgerufen werden, diskutiert Nitschke (2006).

5.6 Zusammenfassung

Wie die letzten Abschnitte gezeigt haben, lassen sich die meisten existierenden Zuverlässigkeitsmechanismen für Multiagentensysteme in der Zuverlässigkeitsreferenzarchitektur auf natürliche Weise verorten, obwohl die Entwicklung dieser Mechanismen nicht im Zusammenhang mit der Architektur erfolgte. Besonders das Eskalieren der Störungen durch die Schichten, die Störungsbehandlungsmächtigkeit der einzelnen Mechanismen, deren Abgrenzung und Zusammenspiel konnte entlang der Pfade „Unsicherheiten“, „Ontologiestörungen“, „Rechnerstörungen und unerwartetes Agentenverhalten“ und „Netzwerkstörungen“ illustriert werden. Entstanden ist dabei eine systematische Landkarte der Zuverlässigkeitsmechanismen für Multiagentensysteme, die Entwicklern solcher Systeme

beim Auffinden geeigneter Mechanismen für ein bestimmtes Zuverlässigkeitsproblem helfen kann. Die Verortungsfähigkeit im Bezug auf die existierenden Mechanismen konnte somit nachgewiesen werden.

Bei der Diskussion der Mechanismen zeigte sich auch, dass viele davon bestimmte Anforderungen an die Konzeption und Implementierung des Multiagentensystems stellen. Meist sind sie nicht allgemeingültig, sondern gehen z.B. von konkreten Agentenarchitekturen oder anderen Randbedingungen aus. Folglich kann es passieren, dass ein Entwickler für sein Zuverlässigkeitsproblem zwar einen konzeptionell geeigneten Mechanismus identifizieren kann, dass dieser sich aber nicht einsetzen lässt, weil beispielsweise bereits eine bestimmte Systemarchitektur in seiner eigenen Entwicklung zum Einsatz kommt. Der Investitionsschutz wäre damit nicht gewährleistet.

Das nächste Kapitel betrachtet genau diesen Fall und versucht am Beispiel der oben bereits eingeführten Architekturen dem Entwickler eine Hilfestellung zu geben, wo er in seinem System mit einem eigenen Mechanismus innerhalb der eingesetzten Architektur gegensteuern kann.

Kapitel 6

Fehlerbehandlung in existierenden Architekturen

Während im vorangegangenen Kapitel 5 eine Landkarte existierender Zuverlässigkeitsmechanismen entworfen und damit deren Verortungsfähigkeit untermauert wurde, steht im aktuellen Kapitel die Übertragbarkeit der Zuverlässigkeitsreferenzarchitektur auf existierende Architekturen im Mittelpunkt. Dazu werden die im Verlaufe der Arbeit (in Abschnitt 3.4.2) bereits betrachteten Architekturen für Multiagentensysteme erneut untersucht und zur Zuverlässigkeitsreferenzarchitektur in Bezug gesetzt.

Die Übertragbarkeit erlaubt es dem Entwickler eines Agentensystems, das einer dieser Architekturen folgt, Querschlüsse zu ziehen und dadurch die evtl. vorhandenen Zuverlässigkeitsprobleme systematisch anzugehen: Die Zuverlässigkeitsreferenzarchitektur hilft, Störungen bestimmten Systemkomponenten zuzuordnen und gibt Hinweise darauf, wo existierende Mechanismen greifen könnten bzw. wo im System ein eigener Zuverlässigkeitsmechanismus angesiedelt werden muss, falls kein existierender Mechanismus eingesetzt werden kann. Dafür werden in den folgenden Abschnitten für die Architekturen ABSRM (in Abschnitt 6.1), RASMAS (6.2), InteRRap (6.3), Goddard Agent Architecture (6.4) und FIPA (6.5) die Pfade „Unsicherheiten“, „Ontologiestörungen“, „Rechnerstörungen und unerwartetes Agentenverhalten“ und „Netzwerkstörungen“ und die weiteren Mechanismen traversiert und dabei untersucht, welche Störungen und Störungsbehand-

lungsoptionen aus der Zuverlässigkeitsreferenzarchitektur sich auf welche Architekturkomponenten abbilden lassen.

6.1 Agent-Based System Reference Model (ABSRM)

Das Agent-Based System Reference Model (ABSRM), das in Abbildung 3.14 im Kontext der Referenzarchitektur dargestellt ist, versteht sich – genau wie die Referenzarchitektur selbst – nicht als konkrete Architektur, in der ein System direkt implementiert werden könnte, sondern eher als Meta-Modell, das eine gemeinsame Struktur und Begriffswelt für eine Klasse konkreter Agentensysteme definiert (Modi u. a. 2006). Bei ABSRM steht dabei besonders eine generische Laufzeitumgebung im Mittelpunkt, die noch um die Funktionseinheiten Administration, Security, Conflict Management und Messaging angereichert wird.

In der Referenzarchitektur werden damit von ABSRM vor allem die Schichten S1 und S2 (in der b- und c-Säule) und mit Einschränkungen die Schicht S4 und die a-Säule abgedeckt. Die Schichten S3 und S5 werden hingegen nicht explizit behandelt, sondern fallen implizit in den Controller. Für die Übertragbarkeit der Zuverlässigkeitsreferenzarchitektur bedeutet dies, dass im ABSRM vor allem die Anfänge der Pfade „Netzwerkstörungen“ und „Rechnerstörungen und unerwartetes Agentenverhalten“ interessant sind.

Die Funktionseinheit Messaging, die als Bestandteil der Platforms die Aufgaben Message Construction, Naming and Addressing, Transmission und Receiving übernimmt, kann insbesondere von Nachrichtentransportstörungen, Registrierungsstörungen und von unerreichbaren Partneragenten betroffen sein. Damit bietet sie sich auch für die Umsetzung von aktiven Diensteverzeichnissen, „Time out“- oder Agentenreplikationsverfahren an. Ob ein Weiterreichen von langfristig unerreichbaren Partneragenten an die Koordinationsschicht S5 Sinn macht, hängt von deren genauer Umsetzung im Controller ab: ABSRM sieht hierfür lediglich das sehr spezifische Conflict Management vor, auf das die Konversationserweiterungen und die Rekoordinierung abgebildet werden müssen.

Von dem Pfad „Rechnerstörungen und unerwartetes Agentenverhalten“ können in ABSRM vor allem Rechner- und Plattformstörungen auftreten, die auf der Ebene der Plattformen mit persistenten Agenten zu beheben sind. Dieser Ansatz passt gut zu den Aufgaben der Funktionseinheit Administration (Agent Creation, Agent Management, Resource Control und Agent Termination) und kann dort folglich gut implementiert werden. Eine Eskalation der Störung als teilentschärfter Plattformausfall an die Schicht S4 hängt wieder von der konkreten Implementierung des Controllers von ABSRM ab und muss daher fallweise abgewogen werden.

6.2 Reference Architecture for Situated Multiagent Systems (RASMAS)

Die beiden Hauptabstraktionen der Reference Architecture for Situated Multiagent Systems (RASMAS) (Weyns u. a. 2006) Environment und Agent umfassen die Schichten S1 und S2 bzw. S3, S4 und S5, und decken damit die komplette Referenzarchitektur ab (siehe Abbildung 3.15). Als Konsequenz aus dem Entstehungshintergrund der RASMAS in der Schwarmtheorie, in der die Informationsübermittlung über die Umgebung und die Weltmodellierung eine wichtige Rolle spielt, sind dabei in Schicht S1 und S2 die Säulen a und c und die Schicht S3 besonders stark ausgeprägt. Die Komponentenverwaltung (Säule b) und die Schichten zur Verhaltensgenerierung und Koordination sind hingegen von untergeordnetem Interesse und nicht detailliert ausgearbeitet.

Durch den starken Umgebungsbezug der RASMAS ist besonders die Betrachtung des Pfades „Unsicherheiten“ der Zuverlässigkeitsreferenzarchitektur relevant. Dieser Pfad beginnt mit einer Sensor-/Aktor-Störung in S1a. Die Sensor-Störung tritt in RASMAS in der Observation-Komponente auf. Als Fehlerbehandlung wird im Pfad die Ausnutzung redundanter Wandler vorgeschlagen, die direkt in S1a oder auch in S2a implementiert werden kann. In RASMAS korrespondiert dies mit dem Zyklus aus `obsrv` und `observed data` zwischen den Komponenten Observation und Representation Generator: Für eine bestimmte gewünschte Wahrnehmung eines Umgebungszustandes können mehrere Sensoren in der Umgebung abgefragt werden. Für Aktor-Störungen müssen

zusätzlich die Komponenten Interaction und Translation mit in den Zyklus einbezogen werden[1], wodurch sich eine Beauftragung redundanter Aktoren verkomplizieren wird.

Im Falle einer nicht vollständigen Fehlerbehandlung durch Wandlerredundanz würde entlang des Pfades eine imperfekte Situationseinschätzung an S3 weitergegeben und dort durch ein Uncertainty Management behandelt werden. Weyns, Steegmans und Holvoet (2004b) präsentieren für diese Aufgabe ein bestens präpariertes formal fundiertes Weltmodell, das explizit mit Unschärfe umgehen kann[2]. So besteht z.B. die Möglichkeit, Reizschwellen für Wahrnehmungen oder Constraints auf zulässige Weltmodellzustände zu definieren. Restunsicherheiten im Weltmodell, die evtl. noch an die Verhaltensgenerierung in S4 weitergegeben werden müssen, lassen sich dort mittels sog. Filter ausblenden. Die Decision-Komponente bietet also eine innere Agentenarchitektur mit Unsicherheitsbeherrschung, die den Pfad „Unsicherheiten“ abschließt.

Das Kommunikationsmodell von RASMAS (Weyns u. a. 2004a) verwendet einfache Ontologie-basierte Nachrichten und formuliert daraus Protokolle. Folglich sind die Pfade „Ontologiestörungen“, „Netzwerkstörungen“ und die Protokollverifikation der Zuverlässigkeitsarchitektur im Kontext der RASMAS interpretierbar. Kommunikationsontologiestörungen lassen sich durch automatische Serialisierer und validierende Parser in der Communication-Komponente auf S3 behandeln und bei unvollständiger Störungsbehebung kann eine Bedeutungsaushandlung in der Decision-Komponente erfolgen, die u.a. die Koordination (S5) abdeckt.

Auch der Pfad „Netzwerkstörungen“ lässt sich gemäß der Verortung in Abbildung 3.15 weitestgehend auf Komponenten von RASMAS (Translation, Interaction, Communication Service, Communication und Decision) abbilden. Ebenso lässt sich die Protokollverifikation in der Entwurfsphase gegen Konversationsstörungen in RASMAS gut unterstüt-

[1] In Abbildung 3.15 sind diese Komponenten der Verteilungsunterstützung in Säule c zugeordnet und nicht der Realwelteinbettung. Dies rührt daher, dass diese Komponenten sowohl für die Kommunikation mit anderen Agenten als auch für die Interaktion mit der Umgebung verantwortlich sind. Für das aus der Schwarmtheorie stammende RASMAS hat dies seine Berechtigung: Kommunikation wird hier oft mit dem Hinterlassen von Botschaften in der Umgebung gleichgesetzt (vergleichbar der Kommunikation von Ameisen über Pheromonspuren).

[2] Das gut ausgearbeitete Weltmodell ist erneut durch den Hintergrund Schwarmtheorie begründet, da bei der Wahrnehmung von „Duftspuren“ in der Umgebung stets von einer gewissen Unschärfe ausgegangen wird.

zen, da bereits ein formales Modell für die Nachrichten, deren Inhalte und die Protokolle existiert und dieses leicht um die notwendigen Mechanismen zu erweitern ist (Weyns u. a. 2004a). Eine Ausnahme im Pfad „Netzwerkstörungen“ ist die Agentenreplikation, für die es keine Entsprechungen gibt, da RASMAS alle Aspekte der Komponentenverwaltung ausblendet. Dies führt auch dazu, dass die Störungen und Mechanismen aus dem Pfad „Rechnerstörungen und unerwartetes Agentenverhalten“ keine Anwendung in RASMAS finden können.

6.3 InteRRap

Bei InteRRap (Müller 1996) steht die Konzeption einer inneren Agentenarchitektur im Fokus, die die definierenden Agenteneigenschaften Reaktivität, Zielorientierung und Kooperation in Einklang bringt. Dabei entsteht eine (im wesentlichen) dreischichtige Architektur mit den Schichten Behavior-based Layer, Local Planning Layer und Cooperative Layer, die sich auf ein gemeinsames Weltmodell („Knowledge Base“) abstützen. Diese Komponenten definieren in der Referenzarchitektur die Schichten S3 bis S5, wohingegen die Infrastrukturschichten S1 und S2 bei InteRRap durch die einfachen Komponenten Perception, Action und Communication nur rudimentär besetzt sind (siehe Abbildung 3.17).[3]

Für die Zuverlässigkeitsreferenzarchitektur ergibt sich aus der unterschiedlichen Detaillierung der Schichten, dass nur eine Betrachtung der Mechanismen der Schichten S3 bis S5 nützlich sein kann. In der Ontologie-basierten Knowledge Base ist Unsicherheit zunächst nicht explizit modelliert und es existieren auch keine Operatoren für Uncertainty Management. Im Falle von weitergereichten Sensor-/Aktor-Störungen, die im Pfad „Unsicherheiten“ in S3 zu imperfekten Situationseinschätzungen führen, müssen diese Eigenschaften in der Knowledge Base ergänzt werden. Die Unsicherheitsbeherrschung in der inneren Agentenarchitektur wird hingegen nativ unterstützt, da ihre Konzeption auf dem BDI-Modell basiert, das mit den Beliefs die Unsicherheit expliziert.

[3] Für die Implementierung realer Anwendungssysteme würde man die Infrastruktur von InteRRap sinnvollerweise durch eine vollständige Agentenplattform ersetzen und nur die innere Agentenarchitektur übernehmen.

Der Pfad „Ontologiestörungen“ lässt sich in InteRRap komplett abbilden und seine Betrachtung ist von besonderem Interesse, weil die Nachrichtenübermittlung keine strenge Typisierung vorsieht und damit fehleranfällig gegen Kommunikationsontologiestörungen erscheint. In Ermangelung der Typisierung ist eine Realisierung automatischer Serialisierer und validierender Parser jedoch ohne eine Erweiterung der Infrastrukturschicht nicht möglich. Eine Behandlung unterschiedlicher Ontologien bei Sender und Empfänger ist durch Ontologieübersetzung in der Knowledge Base bzw. durch Bedeutungsaushandlung in der Cooperative Planning Layer realisierbar.

Nur schwer innerhalb der InteRRap-Architektur interpretierbar ist der Pfad der „Rechnerstörungen und unerwartetes Agentenverhalten“, da die Aspekte der Komponentenverwaltung hier ausgeklammert sind. Obwohl der Zustand in der Knowledge Base ähnlich wie in Datenbanken behandelt wird und damit prinzipiell den transaktionsbasierten erweiterten Wiederanlaufverfahren zugänglich ist, ist eine glatte konzeptuelle Einbettung in den festgelegten Zyklus aus Situation Recognition/Goal Activation und Planning/Scheduling in den S4 zugeordneten InteRRap-Schichten nicht trivial. Die Entwicklung robuster Koordinationsmechanismen passt hingegen gut zur Konzeption der Cooperative Planning Layer und damit ist zumindest das unerwartete Verhalten anderer Agenten im vorliegenden Pfad leicht zu behandeln.

Übrig zur Betrachtung bleiben aus den höheren Schichten der Zuverlässigkeitslandkarte die Protokollverifikation gegen Konversationsstörungen und aus dem Pfad „Netzwerkstörungen“ die Konversationserweiterung und Rekoordinierung als Schicht S5-Mechanismus. Obwohl Konversationsstörungen in InteRRap auftreten können, sind sie hier nur schwer durch Protkollverifikation zu adressieren, weil der Protokollbegriff in InteRRap konzeptionell nicht vorhanden ist. Vor einer formalen Anreicherung von Protokollen zur Verifikation muss also zunächst eine explizite Handhabung von Protokollen in die Formalismen von InteRRap eingebettet werden. Die Abwesenheit eines Protokollbegriffs erschwert auch eine Behandlung langfristig unerreichbarer Partneragenten durch Konversationserweiterung, da diese implizit Protokolle als zu erwartende Nachrichtenfolgen voraussetzt – eine Rekoordinierung kann im Normalbetrieb der Cooperative Planning Layer erfolgen.

6.4 Goddard Agent Architecture (GAA)

Im Fokus der Goddard Agent Architecture (GAA) (Truszkowski 2006; Truszkowski und Rouff 2001) steht die modulare Entwicklung von Agentensystemen durch eine Definition von Komponenten und getypten Datenflüssen zwischen diesen. Erneut werden dabei alle Aspekte der Komponentenverwaltung ausgeblendet. Insgesamt werden durch die GAA damit die Schichten S1 und S2 (mit Ausnahme der Säule b) und die Schichten S3 und S4 abgedeckt. Die Koordination wird nicht explizit behandelt, fällt aber funktional implizit mit in die Komponenten der Verhaltensgenerierung (siehe Abbildung 3.19).

Durch den geplanten Einsatz der Goddard Agenten in der Raumfahrt der NASA ist die Architektur so ausgelegt, dass auch unter relativer Unsicherheit (z.B. auch durch gestörte Aktoren oder Sensoren) in kürzester Zeit Entscheidungen getroffen und Aktionen durchgesetzt werden können. Dafür wird der Komponente Perceptors eine eigene Intelligenz beigemessen, die z.B. Sensorsignale aggregieren und bis zu einem bestimmten Niveau auswerten kann. Bei dringend erforderlichen Reaktionen können die Perceptors daher auch sog. Reflex Actions anstoßen, die nicht durch die komplette Verhaltensgenerierung des Systems laufen müssen, sondern direkt von den Effectors ausgeführt werden. Mit dieser kurzen Kopplung von intelligenten Sensoren und Aktoren in S2a lassen sich einfach auch aufwändige Mechanismen der Wandlerredundanz gegen Sensor-/Aktor-Störungen im Pfad „Unsicherheiten“ implementieren. Durch diese Mächtigkeit wird in der GAA davon ausgegangen, dass keine Unsicherheiten mehr an die höheren Schichten weitergegeben werden müssen und folglich muss hier auch keine Behandlung mehr stattfinden. Damit endet der Pfad „Unsicherheiten“ in der GAA bereits in Schicht S2.

Die Kommunikation in der GAA wird über die Komponente Agent Communication Perceptor/Effector abgewickelt, die Nachrichten nach dem entsprechenden FIPA-Standard einsetzt und diese über eine externe (nicht FIPA-konforme) Kommunikationsinfrastruktur sendet und empfängt. Störungen in dieser externen Infrastruktur führen zu Nachrichtenstörungen in der GAA-Komponente und eröffnen damit den Pfad „Netzwerkstörungen“. Allerdings werden die Nachrichten in der GAA-Komponente nicht weiter bearbeitet – insbesondere findet keine Konversationsüberwachung statt, die für Time

out-Verfahren benötigt werden würde. Vielmehr werden die Nachrichten (und damit ggf. auch die Störungen) direkt an das Agent Reasoning auf Schicht S4 weitergeleitet, was im Störungsfall in der Zuverlässigkeitsreferenzarchitektur einer bedingungslosen Fehlerweitergabe entspricht. Im Reasoner muss dann auf Basis des Weltmodells entschieden werden, wie mit einer (fehlenden) Nachricht weiter verfahren werden soll. Hier lassen sich also auch Reaktionen definieren, die einer Konversationserweiterung oder Rekoordinierung gegen langfristig unerreichbare Partneragenten entsprechen. Der Aufwand hierfür ist aber relativ hoch, weil zuerst ein Protokollbegriff in der Architektur verankert werden muss. Die weiteren Störungen und Mechanismen des Pfades „Netzwerkstörungen“ können in der GAA mangels einer Betrachtung aller Verwaltungsaspekte nicht sinnvoll interpretiert werden. Dies gilt wegen des fehlenden Protokollbegriffs ebenso für die Protokollverifikation und durch die Nichtbeachtung mobiler Agenten auch für die erweiterten Umzugsverfahren.

Die Ontologie-basierte Modeling and State-Komponente der GAA entspricht der Schicht S3 der Referenzarchitektur und setzt sowohl deren statische als auch dynamische Dienste auf der Schema-Ebene um. Die dynamischen Dienste sollen es ermöglichen, dass die Agenten mittelfristig neue Schemata für das Weltmodell erlernen und sich damit besser an diese anpassen können. Auf dieser Funktionalität lassen sich einfach Verfahren zur Ontologieübersetzung gegen Kommunikationsontologiestörungen auf Schicht S3 aufbauen und unter Einbezug des Agent Reasoning und der Kommunikation auch die höherschichtigen Verfahren zur Bedeutungsaushandlung. Der Pfad „Ontologiestörungen“ lässt sich also sehr gut in GAA unterstützen.

Für den noch verbleibenden Pfad „Rechnerstörungen und unerwartetes Agentenverhalten“ gelten die Aussagen zu InteRRap analog auch für GAA: durch die fehlende Komponentenverwaltung ist der Pfad auch hier schwer interpretierbar. Mechanismen mit persistentem transaktional geschütztem Weltmodell müssen in der Modeling and State-Komponente verankert werden und die erweiterten Wiederanlaufverfahren dann im Agent Reasoning abgebildet werden. Dort sind auch die robusten Koordinationsmechanismen zu implementieren, die mangels expliziter Koordinationsunterstützung jedoch vollständig auf lokale Ziele und Aktionen abzubilden sind.

6.5 FIPA Standard

Die Standards der Foundation for Intelligent Physical Agents (FIPA) (FIPA 2002a, 2004b, a, und andere) haben die Spezifikation von Plattformen für Multiagentensysteme zum Zwecke der Interoperabilität zum Ziel. Wie Abbildung 3.13 zeigt, fallen die meisten Komponenten der FIPA-Architektur damit in die verschiedenen Säulen der Schicht S2, in der die Agenten-spezifischen Infrastrukturdienste beheimatet sind. Die nicht-agentenspezifische Schicht S1 und die Schichten S3 bis S5 werden nur soweit spezifiziert, wie dies der Interoperabilität zuträglich ist. Da insgesamt für die Interoperabilität von Agentenplattformen vor allem die Verteilungsunterstützung und die Komponentenverwaltung von Bedeutung sind, sollen zunächst die Pfade „Netzwerkstörungen“ und „Rechnerstörungen und unerwartetes Agentenverhalten“ betrachtet werden. Hier ist eine besonders gute Abdeckung zu erwarten.

Datentransportstörungen treten außerhalb der FIPA-Plattformen auf, weil für den Datentransport nur generische Internetprotokolle benutzt werden. Falls deren eigene Mechanismen nicht greifen, kommt es im Message Transport Service zu Nachrichtentransportstörungen, für die z.B. in der FIPA-konformen Plattform JADE bereits einfache Time out-Verfahren implementiert sind, die auf die Konversationsüberwachung der FIPA-Protokolle zurückgreifen. Agentenreplikation gegen unerreichbare Partneragenten wird nicht durch native Funktionen der FIPA-Plattformen direkt unterstützt, lässt sich aber auf Basis der Dienste des Agent Management Systems und des Directory Facilitators implementieren. Aus dem Pfad „Netzwerkstörungen“ bleiben noch die langfristig unerreichbaren Partneragenten, die ggf. von den Mechanismen der Plattform weitergereicht werden müssen an die Schicht S5, wo sie durch Konversationserweiterungen und Rekoordinierung zu behandeln wären. Die dort von der FIPA definierten Protokolle erlauben eine zuverlässigkeitserhöhende Erweiterung z.B. durch das Einfügen zusätzlicher Bestätigungsnachrichten.

Der Pfad „Rechnerstörungen und unerwartetes Agentenverhalten“ beginnt innerhalb der FIPA-Plattform in S2b, nachdem aus S1b Rechnerstörungen als Plattformstörungen weitergeleitet wurden. Die Spezifikationen der FIPA-Plattform sehen Persistenz weder für die Infrastrukturkomponenten noch für die Agenten selbst nativ vor. Eine entsprechende

Erweiterung ist über einen Eingriff in die Lebenszyklusverwaltung des Agent Management Systems realisierbar (FIPA 2004a). Bei einer weiteren Eskalation als teilenschärfter Plattformausfall oder bei unerwartetem Agentenverhalten können die erweiterten Wiederanlaufverfahren ohne Einschränkungen aber auch ohne spezifische Unterstützung seitens der FIPA-Standards in der Verhaltensgenerierung von S4 implementiert werden. Dies gilt auch für die robusten Koordinationsverfahren in S5.

Gegen Kommunikationsontologiestörungen wird aus dem Pfad „Ontologiestörungen“ die Ontologieübersetzung direkt durch den Ontology Service (FIPA 2001f) umgesetzt und automatische Serialisierer/validierende Parser werden von einigen FIPA-implementierenden Plattformen als integraler Dienst angeboten. Die Bedeutungsaushandlung als S5-Koordinationsmechanismus gegen weitergereichte Kommunikationsontologiestörungen liegt außerhalb der von der FIPA spezifizierten Komponenten.

Aus dem Pfad „Unsicherheiten“ können Sensor-/Aktor-Störungen in FIPA gut adressiert werden. Sensoren und Aktoren von S1a können in S2a über die einfachen Wrapper angebunden werden, die eine generische Schnittstelle zwischen Plattform-externen Komponenten und der Agenten-Plattform definieren (FIPA 2001b). Technisch gesehen werden die Sensoren und Aktoren dann jeweils gegenüber der Plattform in einem speziellen Agenten gekapselt. Zwischen der Agentenschnittstelle und der Wandlerschnittstelle lassen sich konzeptionell beliebige Programme verbergen, so dass hier auch transparent ein Wandlerredundanz-Mechanismus implementiert werden kann. Falls dennoch eine Störung entlang des „Unsicherheiten“-Pfades weitergereicht werden muss und damit in den höheren Schichten imperfekte Situationseinschätzungen oder Restunsicherheiten entstehen, werden diese in Komponenten behandelt, die außerhalb der FIPA-Spezifikation liegen, d.h. die die FIPA-Komponenten ihrer Plattform lediglich benutzen.

Neben den beschriebenen Ausschnitten aus den Pfaden lassen sich auch weitere Störungen und deren Fehlerbehandlungsmechanismen, die nicht in die Pfade aufgenommen wurden, innerhalb von FIPA verorten. Die Spezifikationsweise der FIPA-Protokolle macht diese z.B. leicht zugänglich für formale Erweiterungen, die eine Grundlage für die Protokollverifikationsverfahren gegen Konversationsstörungen darstellen. Falls Rechner-

störungen während einer Umzugsphase auftreten können, lassen sich auch die einfachen Umzugsverfahren aus dem entsprechenden FIPA-Standard zur Mobilitätsunterstützung (FIPA 2002l) gegen Mobilitätsstörungen erweitern und aktive Diensteverzeichnisse gegen Registrierungsstörungen sind im Rahmen der FIPA-Spezifikation des Directory Facilitators umsetzbar.

6.6 Zusammenfassung

In den vorangegangenen Abschnitten 6.1 bis 6.5 erfolgte eine Zuordnung von Störungen und zugehörigen Störungsbehandlungsoptionen auf eine Auswahl prominenter Architekturen entlang der definierten Behandlungspfade.

Für die Plattform-bezogenen Architekturen FIPA und ABSRM ließen sich die Anfänge aller Pfade und die kompletten Pfade „Netzwerkstörungen“ und „Rechnerstörungen und unerwartetes Agentenverhalten“ gut zuordnen. InteRRap, das seinen Schwerpunkt auf die Ausprägung der Komponenten der inneren Agentenarchitektur legt, profitierte hingegen stärker von den höherschichtigen Mechanismen, d.h. von den Enden der Pfade. In den Architekturen GAA und RASMAS, die durch ihren Einsatzhintergrund einen starken Umgebungsbezug haben, ließen sich wiederum die Störungen und Mechanismen der Pfade „Unsicherheiten“ und „Ontologiestörungen“ besonders gut verorten.

Zusammenfassend konnte damit gezeigt werden, dass sich die Erkenntnisse aus der Zuverlässigkeitsreferenzarchitektur erfolgreich auf sehr verschiedene Architekturen übertragen lassen. Wie zu erwarten konnte auch beobachtet werden, dass für die jeweils am detailliertesten ausgearbeiteten Bereiche der Architekturen eine besonders gute Abbildung von Störungen und Behandlungen möglich ist. Dies bedeutet jedoch keine Einschränkung der Übertragbarkeit, da der Entwickler in den Bereichen, die von seiner eingesetzten Architektur nicht definiert werden, alle Freiheiten der Ausgestaltung genießt, insbesondere die Freiheit zur unbeschränkten Gestaltung von Fehlerbehandlungsmechanismen gemäß der Zuverlässigkeitsreferenzarchitektur.

Kapitel 7

Eine Fallstudie: Transaktionale Konversationen

In den vorangegangenen beiden Kapiteln wurden mit der Zuverlässigkeitslandkarte die Verortungsfähigkeit der Zuverlässigkeitsreferenzarchitektur (in Kapitel 5) und deren Übertragbarkeit auf andere Architekturen (in Kapitel 6) nachgewiesen. Entwickler von Multiagentensystemen profitieren davon, indem sie entweder direkt aus der Zuverlässigkeitslandkarte existierende Fehlerbehandlungsmechanismen gegen ein vorliegendes Zuverlässigkeitsproblem auswählen können oder indem sie – falls eine Auswahl durch Einschränkungen der Mechanismen nicht möglich ist – Hinweise bekommen, wie und an welcher Stelle sie mit eigenen Fehlerbehandlungsmechanismen in ihre Systeme eingreifen können.

Die Fallstudie im vorliegenden Kapitel gibt ein Beispiel für den Einsatz der Zuverlässigkeitsreferenzarchitektur durch einen Entwickler zunächst beim Versuch, einen existierenden Mechanismus für ein konkretes Zuverlässigkeitsproblem eines vorhandenen Systems zu finden und danach bei der Eigenentwicklung einer geeigneten Fehlerbehandlung. Die Fallstudie illustriert damit zugleich, wie die Zuverlässigkeitsreferenzarchitektur zur Entwicklung orthogonaler Mechanismen zur Steigerung der Zuverlässigkeit von Multiagentensystemen gegen endogene externe Störungen beiträgt.

7.1 Ausgangssituation

Im vorliegenden Abschnitt soll zunächst die Ausgangssituation der Fallstudie geschildert werden. Dazu gehören eine Beschreibung der Anwendungsfunktionalität des betrachteten Multiagentensystems, eine Beschreibung seiner technischen Umsetzung und eine Beschreibung der aufgetretenen Störungen.

7.1.1 Ein Multiagentensystem zur Produktionsfeinplanung

Das Multiagentensystem, das in der Fallstudie betrachtet wird, realisiert genau die Anwendungsfunktionalität, die im motivierenden Beispiel in Kapitel 2 bereits eingeführt wurde und deren Hintergrund in Anhang B dargestellt ist. Es handelt sich also um ein MAS, das eine Teilaufgabe der Produktionsplanung und -steuerung realisiert, nämlich die Feinterminierung von Produktionsschritten in der Auftragsfreigabe.

Das System wurde gemeinsam am Institut für Prozessrechentechnik, Automation und Robotik (IPR, Lehrstuhl Prof. Wörn) und am Institut für Programmstrukturen und Datenorganisation (IPD, Lehrstuhl Prof. Lockemann) im Teilprojekt KRASH (Karlsruhe Robust Agent SHell) des DFG-Schwerpunktprogramms SPP1083 „Intelligent Agents in Real-World Business Applications“ entwickelt. Dem Teilprojekt KRASH kam im Verbund des SPP1083 die Rolle zu, ein robustes und zuverlässiges MAS zur intraorganisationalen Produktionsfeinplanung zu entwickeln, das sich in das gemeinsame Multi-Multiagentensystem (MMAS) *Agent.Enterprise* des SPP1083 (Woelk u. a. 2006) zur organisationsübergreifenden Lieferkettenverwaltung (engl. *Supply Chain Management, SCM*) einbetten lassen sollte. (Zur technischen Integration siehe (Nimis und Stockheim 2004; Stockheim u. a. 2004; Krempels u. a. 2003; Scholz u. a. 2005).)

Die Feinplanung erfolgte, wie in Abschnitt 2.1.1 beschrieben, nach dem Kontraktnetzprotokoll, d.h. mit einem Markt-basierten Koordinationsmechanismus mittels organisationsinternen Ausschreibungen durch sog. OrderAgents und Bearbeitungsangeboten

durch die MachineAgents der geeigneten Produktionsressourcen. Die betrieblichen Rahmenbedingungen für eine gute produktionstechnische Robustheit gegen exogene Störungen im Kontext des KRASH-Projektes beschreiben Frey, Nimis, Wörn und Lockemann (2003a, b) und im allgemeineren Fall in seiner Dissertation Frey (2004). Die notwendigen Schritte zur Erlangung einer hohen technischen Zuverlässigkeit gegen endogene externe Störungen sind Gegenstand der weiteren Fallstudie.

7.1.2 Umsetzung des Produktionsfeinplanungs-MAS

Durch die vorgesehene Einbettung des Produktionsfeinplanungs-MAS in das MMAS *Agent.Enterprise* wurde bei der Entwicklung Wert auf die Interoperabilität gelegt, d.h. es wurde insbesondere eine Standard-konforme Implementierung angestrebt. Die genaue Umsetzung des MAS nach dem dominierenden FIPA-Standard spielt für den weiteren Verlauf der Fallstudie eine wichtige Rolle und wird daher im folgenden intensiv betrachtet. Dabei wird die bewährte Vorgehensweise entlang der Schichten der Referenzarchitektur in wachsender Abstraktion („bottom-up“) verfolgt.

Als Basis der Implementierung wurde das FIPA-konforme Agentenrahmenwerk JADE (Bellifemine u. a. 2007, 2005) gewählt, wodurch sowohl für die Schicht S2 als auch für S1 bereits viele Systemeigenschaften festgelegt sind. JADE selbst ist in der Programmiersprache Java[1] entwickelt und stellt dem Entwickler Java-Schnittstellen (engl. *Application Programming Interface, API*) für die MAS-Implementierung zur Verfügung. Damit kommen in Schicht S1b alle Rechnerinfrastrukturen in Frage, die die Java-Plattform unterstützen. Im Projektverlauf wurden Windows[2]-Rechner auf Basis von PC-Hardware eingesetzt.

Neben anderen Netzwerken (z.B. aus dem Mobilfunkbereich) unterstützt JADE die Internet-Protokollfamilie TCP/IP, die in S1c zusammen mit dem höherschichtigen HTTP-Protokoll des World Wide Web für die Nachrichten-basierte Kommunikation Verwendung findet.

[1] Siehe `http://java.sun.com`
[2] Siehe `http://www.microsoft.com/windows/`

Die Einbettung in die „Realwelt“ in S1a wurde im Rahmen des SPP1083 lediglich simuliert. Sie erfolgt durch die Anbindung des MAS an das professionelle Simulationssystem für Produktionsanalagen *eM-Plant*[3] (ehemals *SiMPLE++*) über eine selbstentwickelte Datenbank- und Ereignis-basierte Schnittstelle (Sedivy 2003).

Die Ereignisse aus der Produktionssimulation werden in S2a direkt in Wahrnehmungen umgesetzt, z.B. in gerade eintreffende Fertigungsaufträge, die die Entstehung eines neuen OrderAgents initiieren, oder in Maschinenausfälle, die die aktuellen Produktionsabläufe eines MachineAgents verzögern können. Die Handlungen der Agenten, z.B. in Form neu eingeplanter Maschinenbelegungen werden über dieselbe Schnittstelle zur Auswertung wieder in die Simulationsumgebung zurück gespielt.

In S2b werden alle durch den FIPA-Standard definierten Dienste der JADE-Plattform in Anspruch genommen. Dazu gehören z.B. die Laufzeitumgebung mit den entsprechenden Verwaltungswerkzeugen, der Lebenszyklusdienst und auch das Diensteverzeichnis, das wie in Abschnitt 5.5 beschrieben als aktives Diensteverzeichnis ausgelegt ist.

Oben wurde bereits erwähnt, dass der Nachrichtentransport in S1c via HTTP erfolgt. Hierfür muss in der JADE-Plattform in Schicht S2c eine entsprechende Programmbibliothek selektiert werden, das sog. HTTP-MTP (für engl. *Message Transport Protocol*). Die Nachrichten, die als Nutzlast der HTTP-Verbindungen transportiert werden, folgen der XML-Serialisierung der FIPA und werden durch automatische Serialisierer und validierende Parser geschrieben bzw. ausgepackt. Für das Kontraktnetzprotokoll, das in S2c ausgeführt und überwacht wird, wird nicht die einfache von JADE bereitgestellte Implementierung verwendet, sondern eine Implementierung auf Basis JADEs generischer Automatenknoten (*FSMBehaviour* für engl. *Finite State Machine Behaviour*).

Unter Anwendung einer Erweiterung der in (Dinkloh und Nimis 2004) vorgestellten werkzeuggestützten Methodik zur Protokollgenerierung aus AUML-Modellen (siehe Bauer u. a. 2001) werden hierfür aus der Protokollspezifikation in Form eines Sequenzdiagramms (ähnlich Abbildung 2.1) zwei korrespondierende Protokollautomaten generiert. Abbildung 7.1 zeigt den resultierenden Protokollautomaten des Kontraktnetz-

[3] Siehe `http://www.emplant.com`

Initiators (OrderAgent), Abbildung 7.2 den zugehörigen Kontraktnetz-Participant (MachineAgent). Jeder Knoten der Automaten in diesen beiden Abbildungen erbt von FSMBehaviour das generische Verhalten eines Automatenknotens und erweitert ihn um das spezifische Verhalten im jeweiligen Zustand.

Das Ontologie-basierte Weltmodell in S3 des MAS umfasst im Wesentlichen die Fertigungspläne für die Aufträge (engl. *Bill of Material, BoM*), die Fähigkeiten der Maschinen und deren Belegungspläne. Alle Daten werden in einer Datenbank abgelegt, so dass man das Datenbankschema in diesem Fall als die Ontologie des MAS bezeichnen kann und die Lese- und Schreiboperationen auf der Datenbank als die dynamischen Ontologiedienste. Als Kommunikationsontologie kommt eine direkte Umsetzung des Datenbank-Schemas zum Einsatz, die mit dem Werkzeug Beangenerator[4] generiert wurde.

In der Aktionsauswahl der Verhaltensgenerierung von S4 ist die Belegungstaktik der Auftrags- und MachineAgents beheimatet. Für die MachineAgents bestimmt sie, welche konkreten Zeitfenster dem OrderAgent für einen nachgefragten Fertigungsauftrag zu welchem Preis angeboten werden. Umgekehrt wird bei den OrderAgents in der Aktionsauswahl festgelegt, für welchen größeren Zeitraum eine Ausschreibung eines Fertigungsschritts erfolgt und welches der konkurrierenden Angebote der MachineAgents ggf. ausgewählt werden soll.

Bei beiden Agentenarten ist der Aktionsauswahl eine Zielabwägung ebenfalls in S4 vorangeschaltet. Bei den OrderAgents erfolgt diese Abwägung zwischen einer Durchlaufdauerminimierung ab dem Produktionsbeginn, einer frühestmöglichen Fertigstellung unabhängig von der Durchlaufdauer und einer Fertigungskostenoptimierung. Bei den MachineAgents wird zwischen einer Minimierung der Rüstzeiten und einer Maximierung der Auslastung gewählt. Implementiert sind Aktionsauswahl und Zielabwägung als Java-Methoden, die ihre Eingaben aus dem Weltmodell von S3 beziehen und innerhalb der Konversationsausführung von S2c angestoßen werden.

In Schicht S5 des Produktionsfeinplanungs-MAS sind die Agenten-übergreifenden Koordinationsmechanismen und die Protokolle beheimatet. Wie bereits bei der Konversationsausführung und -überwachung von Schicht S2c beschrieben, tauschen sich die

[4]Siehe `http://protege.cim3.net/cgi-bin/wiki.pl?OntologyBeanGenerator`

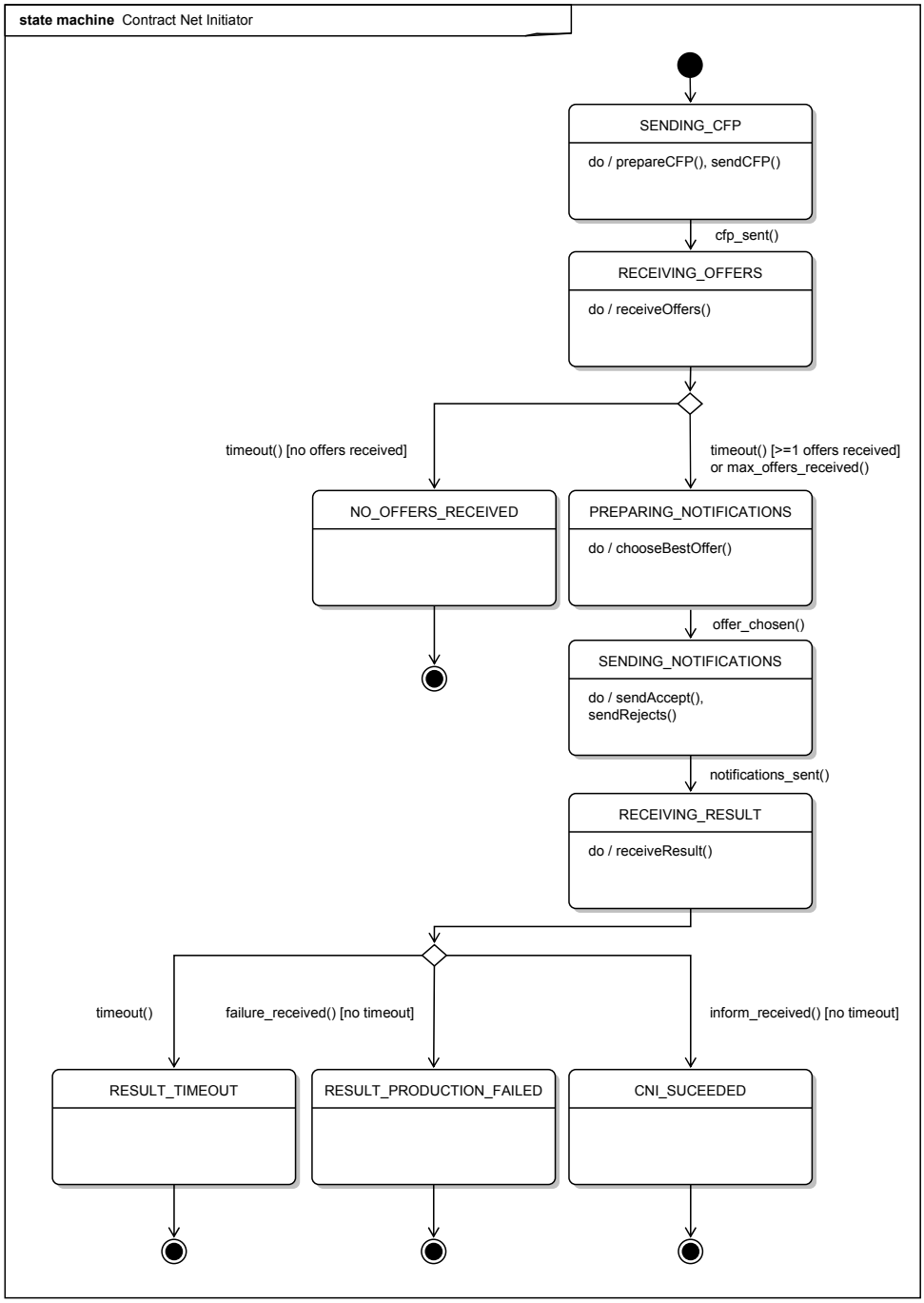


Abbildung 7.1: Protokollautomat des Kontraktnetzprotokolls zur Produktionsfeinplanung: Initiator-Seite

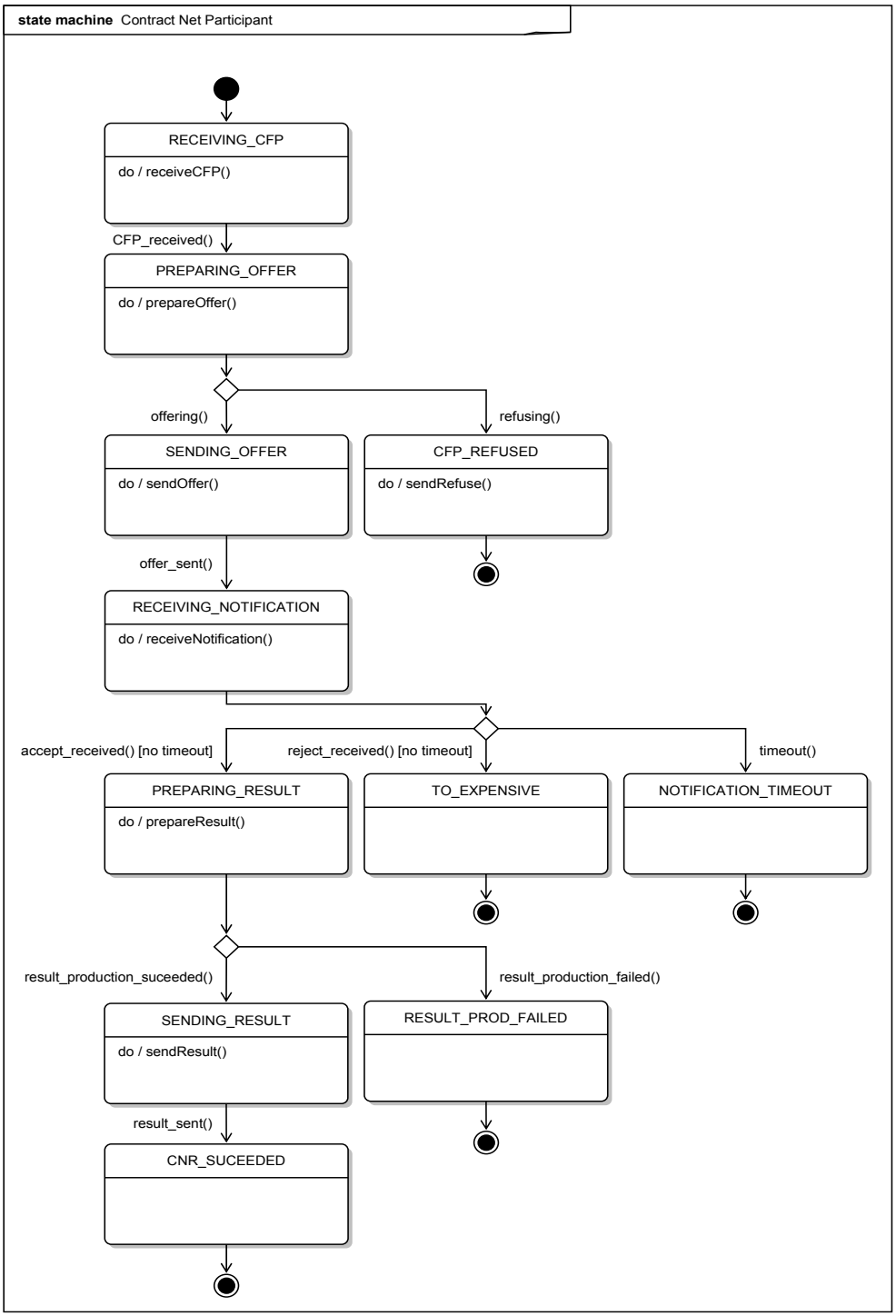


Abbildung 7.2: Protokollautomat des Kontraktnetzprotokolls zur Produktionsfeinplanung: Participant-Seite

Auftrags- und MachineAgents nach dem Kontraktnetzprotokoll aus. Die übergreifenden Ziele der Koordinationsmechanismen werden als Randbedingungen auf die lokalen Ziele der Verhaltensgenerierung in S4 abgebildet und wie dort beschrieben während der Konversationsausführung angestoßen. Zu diesen Zielen gehört, dass die Gesamtauslastung aller Maschinen bezogen auf einen bestimmten Fertigungsschritt nicht größer als 80% sein darf, weil sonst die Robustheit gegen betriebliche Störungen mangels verfügbarer Puffer signifikant abfällt (siehe Frey 2004). Da das Feinplanungs-MAS im beschriebenen Anwendungsszenarium innerhalb einer Organisation eingesetzt wird, ist die Minimierung der Gesamtkosten aller Fertigungsaufträge ein weiteres übergeordnetes Ziel, das sich in S4 z.B. auf die Minimierung der Rüstzeiten als lokales Ziel der MachineAgents durchschlägt und bei den OrderAgents auf das lokale Ziel der Kostenminimierung.

7.1.3 Zuverlässigkeitsprobleme

Das beschriebene MAS zur Produktionsfeinplanung zeigte im Testbetrieb keine größeren technischen Probleme und auch zu Beginn des Dauerbetriebs im Rahmen des MMAS *Agent.Enterprise* war es zunächst zuverlässig und zeigte aus betrieblicher Sicht ebenfalls die erwarteten Ergebnisse. Mit zunehmender Betriebsdauer ließ sich jedoch eine signifikante Verschlechterung der Einplanungsergebnisse beobachten: Aufträge aus der Lieferkette mussten abgelehnt werden, weil sie nicht eingeplant werden konnten, obwohl die geeigneten Maschinen gleichzeitig nicht vollständig ausgelastet waren.

Eine Analyse des Systemverhaltens aus betrieblicher Sicht brachte bei einem Blick in die Belegungspläne der Maschinen einen effektiven Fehlzustand zu Tage: viele als reserviert gekennzeichnete Zeiträume auf den Maschinen wurden nie zu festen Buchungen für einen bestimmten Auftrag bzw. wurden nicht freigegeben. Bei einer näheren Betrachtung des zugrundeliegenden Kontraktnetzes wird klar, dass die zu der veralteten Reservierung gehörige Konversation vor der Buchung bzw. Freigabe abbricht. Abbildung 7.2 zeigt, dass dies im im Zustand `RECEIVING_NOTIFICATION` des Kontraktnetzprotokolls geschehen muss.

Der Abbruch des Kontraktnetzprotokolls könnte zwei mögliche Fehlerursachen haben:

1. Wie in der Motivation in Kapitel 2 beschrieben geht eine der Nachrichten der entsprechenden Konversation beim Transport im Netzwerk verloren. Die Fehlerfälle 1 und 2 in Abbildung 2.2 führen dann genau zu dem beschriebenen Fehlzustand.

2. Einer der an der Konversation beteiligten Agenten oder seine zugrundeliegende Infrastruktur haben eine Störung und der betroffene Agent kann die Konversation nicht bis zu ihrem Ende durchführen.

7.2 Potenzielle existierende Zuverlässigkeitsmechanismen

Bevor die aufwändige Eigenentwicklung eines Zuverlässigkeitsmechanismus angegangen wird, soll versucht werden, einen geeigneten existierenden Mechanismus zu finden. Dazu muss der Entwickler zunächst die Störung in der Zuverlässigkeitsreferenzarchitektur aus Kapitel 4 verorten und kann dann mit der Verortung in die Landkarte der Zuverlässigkeitsmechanismen aus Kapitel 5 gehen, um dort einen geeigneten Mechanismus zu identifizieren.

7.2.1 Verortung der Störungen

Eine genauere Untersuchung der Störung zeigt, dass die hochgradig verteilte Plattform im MMAS *Agent.Enterprise* nicht durchgehend stabil läuft, sondern dass gelegentlich Knoten außerhalb des Einflussbereichs der Entwickler ausfallen, auf denen sich u.a. MachineAgents des Feinplanungs-MAS befinden. Die gestörten Plattformen laufen automatisch sofort wieder an und starten dabei auch die MachineAgents wieder. Dabei kann der Zustand der MachineAgents jedoch nur unvollständig wiederhergestellt werden: Das Weltmodell der MachineAgents ist in Teilen persistent, weil es direkt in einer Datenbank

abgelegt wird, nicht jedoch der innere Zustand des Agenten. So wird z.B. der Ausführungszustand der Konversationen, an denen der Agent beteiligt war, nicht vor dem Ausfall der Plattform geschützt und geht verloren. Selbst wenn nach dem Wiederanlaufen noch weitere Nachrichten aus der entsprechenden Konversation eintreffen würden, könnte der MachineAgent mangels Kontext aus der Konversation diese Nachrichten nicht mehr sinnvoll verarbeiten.

Eine Betrachtung der Zuverlässigkeitsreferenzarchitektur, deren Ergebnis in Abbildung 7.3 dargestellt ist, zeigt, dass die Fehlerursache eine Rechnerstörung in S1b ist, in deren Folge es in S2b zu einer Plattformstörung kommt. Die Plattformstörung wird durch eine einfache Form des Ansatzes der persistenten Agenten, die schon im Ursprungssystem eingebaut ist, bereits abgemildert: zwar führen die persistenten Reservierungen zu dem Anwendungsproblem, viel wichtiger ist aber beispielsweise, dass die verbindlichen Buchungen nicht verloren gehen. Bei der gegebenen Situation handelt es sich also aus der Sicht der Zuverlässigkeitsreferenzarchitektur um eine entschärfte Fehlerweitergabe aus S2b an die höheren Schichten, die in Schicht S3 zu einem inkonsistenten Weltmodell führt.

7.2.2 Identifikation potenzieller Mechanismen

Mit dem Wissen aus der Verortung in der Zuverlässigkeitsreferenzarchitektur kann man nun die Landkarte der existierenden Zuverlässigkeitsmechanismen aus Kapitel 5 betrachten und schaut sich dort die in Frage kommenden Mechanismen genauer an. Durch die als Fehlerursache festgestellte Rechnerstörung ist zunächst eine Betrachtung des Pfades Rechnerstörungen und unerwartetes Agentenverhalten sinnvoll. Es wurde bereits festgestellt, dass die angetroffene Situation auf einem teilentschärften Plattformausfall durch persistente Agenten beruht und man steigt den Pfad daher entsprechend höher und kommt so zu den teilentschärften Plattformausfällen, gegen die die Landkarte erweiterte Wiederanlaufverfahren in S4 vorsieht.

An dieser Stelle sei noch einmal angemerkt, dass eine geeignete Fehlerbehandlung nicht ausschließlich durch die Mechanismen auf dem entsprechenden weiteren Pfad erfolgen muss, da die Pfade als naheliegende exemplarische Möglichkeit zu verstehen sind, wie

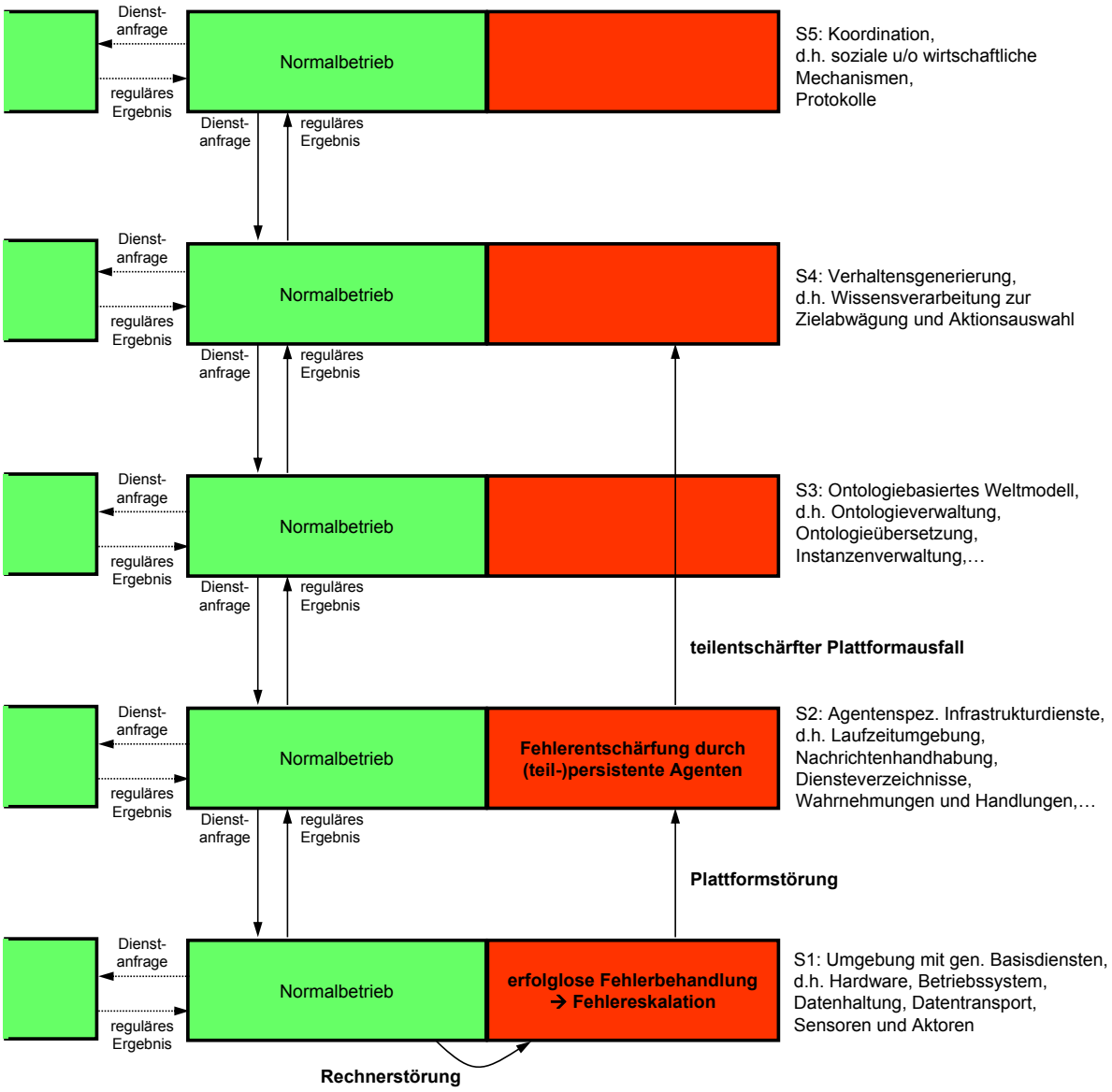


Abbildung 7.3: Verortung der Störungen

Störungen durch die Zuverlässigkeitsreferenzarchitektur eskalieren und in ihr behandelt werden können. Dies schließt nicht aus, dass es noch weitere sinnvolle Übergänge geben kann, die in den vorhandenen Pfaden nicht abgebildet sind und dass dabei sogar zwischen den vorhandenen Pfaden gewechselt werden kann. Im Beispiel wäre es also möglich, dass noch weitere Mechanismen in den höheren Schichten zur Fehlerbehandlung in Frage kommen, die nicht auf dem Pfad Rechnerstörungen liegen. Da dies im vorliegenden Fall nicht zutrifft, sollen die Mechanismen außerhalb des Pfades Rechnerstörungen hier aber nicht weiter diskutiert werden.

Entsprechend der Stelle auf dem Pfad Rechnerstörungen, an der sich die Störung manifestiert, sind alle erweiterten Wiederanlaufverfahren gegen teilentschärfte Plattformausfälle auf ihre Anwendbarkeit im beschriebenen Störungsfall zu analysieren. In der Zuverlässigkeitslandkarte werden an dieser Stelle eine Reihe von Mechanismen aufgeführt, die z.T. jedoch sehr spezielle Annahmen über die Systeme treffen, für die sie geeignet sind, bzw. die sich nicht orthogonal in das vorliegende Zielsystem einbetten lassen.

Die Verfahren der Gruppe um Aridor (in Sher, Aridor und Etzion 2001) und von Pitoura (1998) zielen beispielsweise auf mobile Agenten ab, wobei sich insbesondere Pitoura jedoch auch für stationäre Agenten einsetzen ließe. In diesem Fall ist der Ansatz jedoch eher für die Nebenläufigkeitskontrolle (im Sinne der Isolation) als für die Wiederherstellung im Störungsfall geeignet. Der Ansatz von Busetta (1999) ist hingegen spezialisiert auf das Zurücksetzen von BDI-Agenten und verwendet dafür einen besonderen Interpreter für die Verhaltensgenerierung.

Der Mechanismus von Unruh u. a. (2004) baut in der Zielabwägung von S4 eine deskriptive Zielhierarchie auf, entlang derer im Störungsfall semantisch zurückgesetzt (d.h. kompensiert) wird. Der Ansatz zielt also auf Systeme mit einer stark ausgeprägten Zielabwägung mit vielen – möglicherweise auch widersprüchlichen – Zielen. Der prinzipielle Ansatz des Rücksetzens wäre für das Feinplanungs-MAS geeignet, jedoch ist die Zielabwägungs-Teilschicht dort eher schwach ausgelegt und die Komplexität der Agenten steckt stärker in der Aktionsauswahl. Genau hier ist der Ansatz von Nagi (2001b) beheimatet, der das Rücksetzen nicht entlang einer Zielhierarchie sondern entlang einer Aktionshierarchie organisiert. Daher soll dieser Ansatz im folgenden intensiver auf seine Anwendbarkeit

im Feinplanungs-MAS untersucht werden.

7.2.3 Diskussion eines potenziellen Mechanismus

Beim Mechanismus von Nagi (2001b) wird über die Aktionen der Agenten in einem hierarchischen Agentenplan nach dem Transaktionsmodell der geschachtelten Transaktionen[5] Buch geführt. Mit den offen geschachtelten Transaktionen kann ein Agentenverhalten modelliert werden, bei dem eine Aktion rekursiv aus mehreren zusammengehörigen Unteraktionen bestehen kann. Aus Transaktionssicht werden diese analog abgebildet auf Transaktionen und Subtransaktionen. Eine grafische Darstellung eines beispielhaften Transaktionsbaums zeigt Abbildung 7.4 (vgl. Nagi 2000).

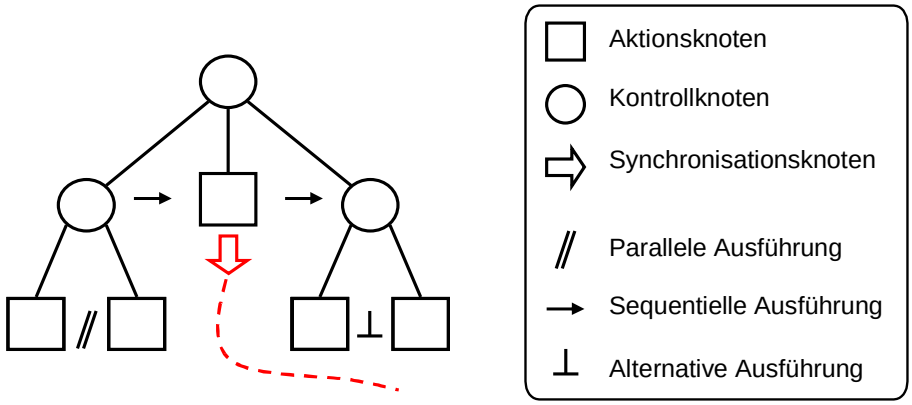


Abbildung 7.4: Agentenpläne als Transaktionsbäume

Das abgebildete Transaktionsmodell ist zugleich eine Einschränkung und Erweiterung der gewöhnlichen offen geschachtelten Transaktionen: Aktionen können hier ausschließlich in den Blattknoten („Aktionsknoten“) ausgeführt werden. Außerdem enthält jeder Aktionskonten noch eine Kompensationsaktion, die die Wirkung der ursprünglichen Aktion semantisch rückgängig macht. In den Zwischenknoten („Kontrollknoten“) wird hingegen ein Kontrollfluss (parallele, sequentielle oder alternative Ausführung) für die untergeordneten Teilbäume festgelegt. Bei der ungestörten Ausführung des Agentenplans im Normalbetrieb wird der Transaktionsbaum nun gemäß dem definierten Kontrollfluss

[5]Für die hier erforderlichen Grundlagen der Transaktionsverwaltung sei auf Anhang C verwiesen.

durchlaufen und die Aktionen des Agenten werden mit ACID-Transaktionssemantik ausgeführt.

Im Allgemeinen kann die Isolationseigenschaft einer Transaktion zu einem Blockieren anderer zeitgleich laufender Transaktionen führen. Dies widerspricht jedoch der Autonomie von Agenten: eine laufende Transaktion eines Agenten hätte Einfluss auf das Fortschreiten anderer Agenten. Der Ansatz umgeht dies durch den Einsatz von *offen* geschachtelten Transaktionen. Bei diesen wird eine Transaktion festgeschrieben, sobald alle ihre Kindtransaktionen erfolgreich festgeschrieben wurden bzw. im alternativen Ausführungsmodus sobald eine ihrer Kindtransaktionen erfolgreich festgeschrieben wurde. Beim vorliegenden Transaktionsmodell werden dann die Ergebnisse für alle anderen Transaktionen sofort sichtbar, d.h. die Isolation wird aufgeweicht.

Im Falle eines Fehlers bei der Ausführung eines Kindknotens gibt es für den Elternknoten mehrere mögliche Reaktionen: einerseits den Abbruch der eigenen Transaktion gefolgt von einem Rücksetzen (engl. *backward recovery*) und ggf. von einem Wiederholungsversuch (engl. *retry*) oder andererseits den Versuch eines Vorwärtssetzens (engl. *forward recovery*) durch die Ausführung einer sogenannten Kompensationstransaktion. Die Kompensation des Vorwärtssetzens wird besonders dann wichtig, wenn einzelne Subtransaktionen bereits festgeschrieben wurden und ihre Ergebnisse damit für andere Transaktionen sichtbar geworden sind.

Mit dem bis hierher beschriebenen Modell lassen sich praktisch beliebige Pläne *einzelner* Agenten darstellen (Nagi 2000). Darüber hinaus wird ein Mechanismus benötigt, der die Abbildung der Kooperation ermöglicht. Um die Autonomie der einzelnen Agenten zu erhalten, wird auf die Einführung einer zentralen Koordinationsinstanz in der Infrastruktur für die verteilten Transaktionen verzichtet. Stattdessen wird die Synchronisation direkt in die Agentenpläne nach dem offen geschachtelten Transaktionsmodell eingebaut (Nagi u. a. 2001; Nagi 2001a).

Dazu werden in die individuellen Transaktionsbäume spezielle Synchronisationsknoten eingebaut. Abbildung 7.5 zeigt eine Synchronisation zwischen zwei Agenten – genauer gesagt zwischen den Transaktionsbäumen, die ihre lokalen Pläne darstellen. Die Synchronisationsknoten sind durch die in der Legende zu Abbildung 7.4 eingeführten breiten Pfeile gekennzeichnet. Die dargestellte Synchronisation stellt eine Unterbeauftragung eines

Agenten („Slave“) durch einen anderen („Master“) dar, also eine Situation, die aus einer Verhandlung nach dem Kontraktnetzprotokoll resultieren könnte.

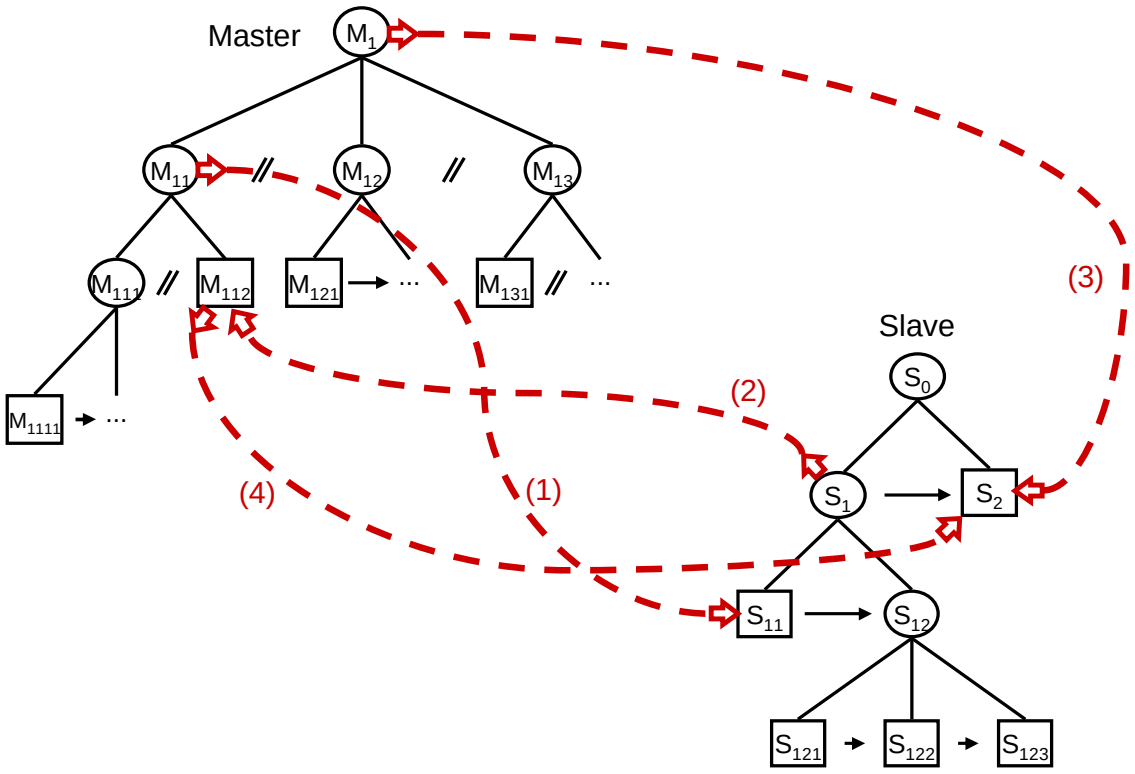


Abbildung 7.5: Synchronisierte Transaktionsbäume zweier Agenten bei einer Unterbeauftragung

Der Knoten M_{11} des Masters startet nicht nur die Subtransaktionen M_{111} und M_{112}, sondern aktiviert durch seine Synchronisationsnachricht (1) auch die Subtransaktion S_{11} des empfangenden Slave-Agenten. In umgekehrter Richtung kann Knoten M_{112} nicht fortfahren, bis die Synchronisatiosnachricht (2) eintrifft, d.h. erst wenn S_{11} beendet wurde (und damit auch S_{111}, S_{112} und S_{113}) und S_1 die Kontrolle zurückbekommen hat. Um eine vollständige Kompensation zu ermöglichen, bedarf es noch zwei weiterer Paare von Synchronisationsknoten mit den zugehörigen Nachrichten. Nachricht (3) verhindert das Beenden der Transaktion auf Slave-Seite vor der Beendigung des Master-Transaktionsbaums und Nachricht (4) sorgt für eine Kompensation des Slave-Unterbaums S_1 falls M_{11} fehlschlägt.

Insgesamt sind die verteilten offen geschachtelten Transaktionen von Nagi (2001b) ein

mächtiger Ansatz, da das Kompensationsverhalten zusammen mit dem regulären Verhalten gehandhabt werden kann und selbst während der Ausführung der Transaktion noch Kompensationsverhalten hinzugefügt werden kann. Grundsätzlich liegen sie auch als erweitertes Wiederanlaufverfahren gegen teilentschärfte Plattformausfälle aus Sicht der Zuverlässigkeitsreferenzarchitektur und der Zuverlässigkeitslandkarte auf einer geeigneten Schicht und auf einem geeigneten Pfad. Weil darüberhinaus der Ansatzpunkt der Aktionspläne gut geeignet für das Produktionsfeinplanungs-MAS erscheint, wurde in (Nimis 2001) ausführlich untersucht, wie sich das Verfahren, das in seiner ursprünglichen Implementierung nicht auf einer standardisierten Plattform aufbaut, auf eine FIPA-Plattform übertragen lässt, mit dem Ergebnis, dass technisch eine Abbildung auf die Schichten S1 und S2 einer FIPA-Plattform möglich wäre.

Die Mächtigkeit des Verfahrens kommt jedoch zu dem Preis, dass die Trennung zwischen Verhaltensgenerierung in S4 und Koordination in S5 weitestgehend aufgegeben werden muss. Damit sind Änderungen am lokalen Verhalten nur schwierig ohne Anpassungen an der Koordination durchführbar und umgekehrt. Seine volle Wirkung kann der Ansatz nur dann entfalten, wenn der MAS-Entwickler für möglichst viele Aktionen und die dabei potenziell auftretenden Fehler vorab Kompensationsaktionen vorsieht und sich auch für das reguläre Verhalten dem Programmiermodell der verteilten offen geschachtelten Transaktionen unterwirft. Da das Feinplanungs-MAS jedoch unabhängig von diesem Programmiermodell entwickelt wurde und bereits ausimplementiert war, bevor das Verfahren eingesetzt werden sollte, würde diese Umstellung im Programmiermodell eine komplette Neuentwicklung der Schichten S4 und S5 bedeuten. Das Verfahren ist also nicht orthogonal zu der bereits existierenden Implementierung und gewährleistet damit nicht den erforderlichen Investitionsschutz – es wird hier folglich verworfen.

7.3 Entwicklung eines eigenen Zuverlässigkeitsmechanismus

Im vorangegangenen Abschnitt 7.2 hat sich gezeigt, dass keiner der existierenden Mechanismen das vorliegende Zuverlässigkeitsproblem ohne einen invasiven Eingriff in das ursprüngliche Feinplanungs-MAS lösen kann. Folglich muss unter Zuhilfenahme der Zuver-

lässigkeitsreferenzarchitektur ein eigener Mechanismus entwickelt werden, der sich orthogonal in das MAS einfügt (siehe auch Nimis und Lockemann 2004).

7.3.1 Transaktionale Konversationen: Vorüberlegungen

Zu Beginn der Eigenentwicklung eines Zuverlässigkeitsmechanismus sind zwei grundlegende Entscheidungen zu treffen: an welchen Systemkomponenten soll der Mechanismus verortet werden und wie ist seine Wirkungsweise.

Eingriffsstelle in das existierende System

Die erste der beiden anstehenden Entscheidungen wird durch die Zuverlässigkeitsreferenzarchitektur und insbesondere durch ihre Übertragung auf die existierenden Architekturen in Kapitel 6 unterstützt. Im vorliegenden Fall basiert das MAS auf einer FIPA-konformen Plattform, weshalb speziell die Zuordnung in Abschnitt 6.5 hilfreich ist.

Wie oben bereits diskutiert, sind Rechnerstörungen die Fehlerursache für das vorliegende Zuverlässigkeitsproblem, weshalb die Mechanismen auf dem Pfad Rechnerstörungen und unerwartetes Agentenverhalten der Zuverlässigkeitslandkarte auf Anwendbarkeit im Feinplanungs-MAS untersucht wurden. Die untersuchten Mechanismen richten sich gegen teilentschärfte Störungen, die von persistenten Agenten nach oben weitergegeben werden. Obwohl die höheren Schichten i.A. mehr Möglichkeiten zur Fehlerbehebung anbieten, wird gerne versucht, einen Ansatz auf einer möglichst tiefen Schicht zu verorten, um damit zu verhindern, dass die Störung überhaupt erst weit durch die Schichten eskalieren kann.

Denkbar wäre es also, den Persistenzansatz aus S2b zu erweitern, um die Störung schon hier so zu verarbeiten, dass die oberen Schichten darauf geregelt weiterarbeiten können. Statt eines tragbaren Ergebniszustands wie bisher soll also mindestens ein unveränderter Ergebniszustand zurückgeliefert werden. Eine mögliche Eingriffsstelle, die sich aus dieser Diskussion ergibt, wäre damit in der Agentenplattform von S2b.

Ebenfalls oben bereits erwähnt wurde, dass sich dasselbe Störungsbild aus fehlerhaften Reservierungen ergibt, wenn im Kontraktnetzprotokoll durch Netzwerkstörungen bestimmte Nachrichten verloren gehen. Aus Sicht eines OrderAgents kann sich der Ausfall eines MachineAgents, mit dem er im Kontraktnetz kommuniziert, als Netzwerkstörung zeigen: die Bestätigung der Erledigung des Auftrags kann ausbleiben und ein Time out-Verfahren könnte dagegen eine Fehlerbehandlung anstoßen. Folglich wäre es auch denkbar, den Mechanismus in S2c zu verankern.

Als Eingriffsstellen kommen also S2b und S2c in Frage. Genauer werden in der Übertragbarkeitsdiskussion von Abschnitt 6.5 die persistenten Agenten an der Lebenszyklusverwaltung in S2b verortet und die Time out-Verfahren an der Konversationsausführung und -überwachung in S2c.

Motivation des transaktionsbasierten Ansatzes

Die Transaktionsverwaltung übernimmt in Datenbankverwaltungssystemen und Anwendungen auf transparente Weise zwei wichtige Aufgaben: sie sorgt für einen Schutz gegen unerwünschte Effekte bei Nebenläufigkeit und für die Erstellung eines geregelten Wiederaufsetzpunktes im Störungsfall[6]. Erreicht wird dies, indem die Transaktionsverwaltung versucht, für eine Transaktion genannte Abfolge von Operationen die sog. ACID-Eigenschaften Atomizität (engl. *Atomicity*), Konsistenz (engl. *Consistency*), Isolation und Dauerhaftigkeit (engl. *Durability*) durchzusetzen.

Generell eignet sich die Transaktionsverwaltung durch die obigen Eigenschaften als Technik zur Entwicklung von orthogonalen Zuverlässigkeitsmechanismen für beliebige Systeme, was auch in Abbildung 4.3 beim Fehlerbehandlungsmodell für allgemeine Dienstgeber deutlich wird, in der die Atomizität – also die „Alles oder Nichts"-Eigenschaft der Transaktionen – und das Wiederherstellen vergangener Systemzustände eine herausgehobene Rolle spielen. Auch der oben ausführlich diskutierte Ansatz von Nagi beruht auf der Transaktionstechnik und unterstreicht durch seine Mächtigkeit und prinzipielle

[6] Für eine kurze Einführung in die spezifischen Grundlagen der Transaktionsverwaltung im Kontext dieser Arbeit sei noch einmal auf Anhang C verwiesen, für eine allgemeine und ausführliche Einführung auf (Weikum und Vossen 2002).

Eignung die Bedeutung der Transaktionsverwaltung auch im Bereich der Multiagentensysteme.

Zusammenfassend bleibt als Ergebnis der Vorüberlegungen festzuhalten, dass der selbstentwickelte Zuverlässigkeitsmechanismus an der Lebenszyklusverwaltung und/oder an der Konversationsausführung und -überwachung festgemacht werden soll und auf Transaktionstechnik beruht.

7.3.2 Transaktionale Konversationen: Konzept und Umsetzung

Grundidee des hier zu entwickelnden Ansatzes ist es, den durch das Speichern des Weltmodells in einer Datenbank bereits im System vorhandenen rudimentären Ansatz der persistenten Agenten so zu erweitern, dass aus globaler Sicht des Feinplanungs-MAS nur noch konsistente Zustände dauerhaft werden. Die veralteten Reservierungen, in denen sich das Zuverlässigkeitsproblem des Feinplanungs-MAS manifestiert haben, stellen hingegen eine globale Inkonsistenz dar, wenn für den ausgeschriebenen Auftrag bereits eine feste Buchung bei einem anderen MachineAgent vorliegt. Aus Transaktionssicht sind es genau die Konversationen, die einen global konsistenten Zustand in einen neuen global konsistenten Zustand überführen. Folglich soll für diese bei Verwendung des Ansatzes das „Alles oder Nichts“-Prinzip gelten: die Konversationen des MAS werden mit Transaktionen identifiziert und werden damit zu den sog. Transaktionalen Konversationen.

Eine Transaktionale Konversation führt dazu, dass alle an ihr teilnehmenden Agenten lokale Transaktionen starten, die durch den Agenten koordiniert werden, der die Konversation initiiert hat. Jeder beteiligte Agent muss dazu einen Ressourcen-Manager (engl. *resource manager*) haben, der lokal die ACID-Eigenschaften erzwingt und den Zugriff auf die Datenobjekte verwaltet, und einen Transaktions-Manager (engl. *transaction manager*), der am Zwei-Phasen-Commit-Protokoll (engl. *two-phase-commit-protocol, 2PC*) teilnimmt und damit die Atomizität der verteilten Transaktion garantiert.

Bezogen auf das Feinplanungs-MAS bedeutet das, dass wie bisher der OrderAgent mit den MachineAgents verhandelt, um eine Maschinenbelegung zu finden. Während der

Konversation nimmt der MachineAgent Änderungen an seinem Weltmodell und damit an der zugrundeliegenden lokalen Datenbank vor, indem er z.B. Reservierungen in seinen Maschinenbelegungsplan einträgt und ggf. auch wieder löscht, wenn er den Zuschlag vom OrderAgent nicht bekommt. Analog dazu ändert der OrderAgent die Datenbank mit seinem Weltmodell sobald eine feste Buchung einer Maschine erfolgt ist. Am Ende der Konversation entscheidet der Transaktions-Manager des initiierenden Agenten auf Basis einer Abstimmung durch die beteiligten Agenten, ob die verteilte Transaktion erfolgreich abgeschlossen wurde oder nicht, und leitet entsprechend ein globales Commit oder Rollback ein, um die Änderungen festzuschreiben bzw. rückgängig zu machen.

Betrachte man nun z.B. die Rechnerstörung, die indirekt durch die veralteten Reservierungen zum Zuverlässigkeitsproblem für das Feinplanungs-MAS wird. Fällt nun unter Verwendung der Transaktionalen Konversationen ein MachineAgent aus, dann wird der Transaktions-Manager in der Abstimmungsphase nicht innerhalb des festgelegten Zeitlimits von allen beteiligten Agenten die sog. `VOTE-COMMIT`-Nachricht erhalten und folglich auf Abbruch der verteilten Transaktion entscheiden. In der Folge werden alle Änderungen an den lokalen Datenbanken rückgängig gemacht, darunter auch die veraltete Reservierung, nachdem der entsprechende Rechnerknoten nach einem Neustart wieder verfügbar ist.

Sollte zum Zeitpunkt der Wiederverfügbarkeit des Knotens die Transaktion noch nicht beendet worden sein – d.h. nach dem Voting abgebrochen – dann kann von der Lebenszyklusverwaltung der wiederanlaufenden Plattform, die über die Agenten und die Transaktionen, an denen diese beteiligt sind, Buch führt, von den jeweiligen Transaktions-Managern mittels einer `VOTE-ABORT`-Nachricht einen sofortigen Abbruch der Transaktion explizit anfordern. Hier greift also die Verortung des Mechanismus bei der Lebenszyklusverwaltung.

Die zusätzliche Verortung bei der Konversationsausführung und -überwachung kommt z.B. zum Tragen bei den Netzwerkstörungen aus den Fehlerfällen (1) und (2) des motivierenden Beispiels aus Kapitel 2. Hier kann man bei den beteiligten Agenten Time outs für den Erhalt der erwarteten Nachrichten setzen, bei deren Ablauf wiederum durch eine `VOTE-ABORT`-Nachricht ein Abbruch der Transaktion angefordert werden

kann.

Abbildung 7.6 zeigt schematisch für das Feinplanungs-MAS, wie die Transaktionalen Konversationen in das eingesetzte FIPA-Agentenrahmenwerk JADE integriert werden können. Als Ressourcen-Manager kommt das Oracle 10g Datenbankverwaltungssystem[7] zum Einsatz und als Transaktion-Manager wird Atomikos TransactionEssentials[8] verwendet. (Für eine ausführliche technische Beschreibung – allerdings einer Vorläuferimplementierung auf Basis von FIPA-OS (Poslad u. a. 2000), Oracle 9i und ORBacus OTS[9] – sei auf (Vogt 2001) verwiesen.)

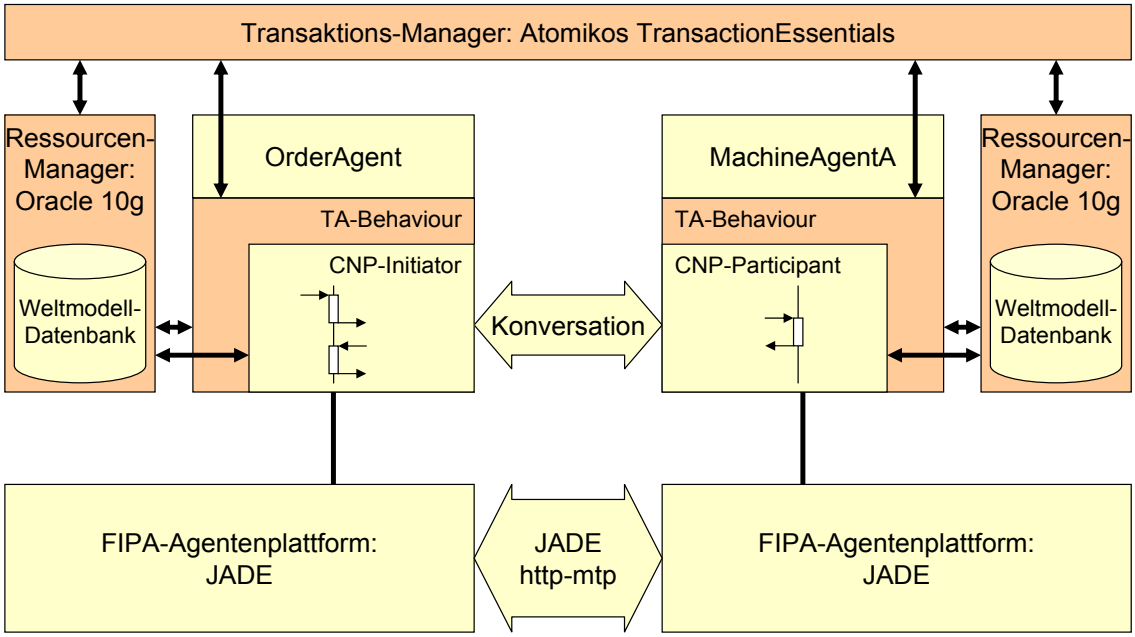


Abbildung 7.6: Transaktionale Konversationen in JADE: Systemarchitektur

Bei der JADE-Agentenplattform werden die verschiedenen Funktionalitäten der Agenten in sog. Behaviours gekapselt. Wie in Abschnitt 7.1.2 bereits beschrieben sind Protokollautomaten des Kontraktnetzes im Feinplanungs-MAS beispielsweise aus Behaviours aufgebaut, die vom sog. FSMBehaviour erben und es spezialisieren. Zur Durchführung einer Konversation werden die resultierenden Protokollautomaten an die Konversationsausführung übergeben, die die Automaten schrittweise abarbeitet. Außerdem kann

[7] Oracle Corporation: Oracle Databases: http://www.oracle.com/database
[8] Atomikos: Atomikos TransactionEssentials: http://www.atomikos.com/Main/AtomikosCommunity
[9] IONA Technologies: ORBacus Object Request Broker: http://www.orbacus.com

jedes Behaviour von JADE eine beliebige Anzahl untergeordneter Behaviours (sog. Kind-Behaviours) haben, wodurch eine hierarchische Organisation des Agentenverhaltens unterstützt wird. Die übergeordneten Behaviours (sog. Eltern-Behaviours) stoßen die Ausführung der untergeordneten Behaviours an, können ihr Fortschreiten beobachten und werden über ihr Ausführungsende informiert. Von der Möglichkeit der hierarchischen Anordnung wird bei der Umsetzung der Transaktionalen Konversationen Gebrauch gemacht.

Dazu wurde ein spezielles Behaviour namens TA-Behaviour entwickelt, das nahezu alle Transaktions-bezogenen Aufgaben übernimmt. Insbesondere leistet es die Interaktion mit dem Transaktions- und dem Ressourcen-Manager. Diese Funktionalität kann transparent beliebigen Protokollautomaten auf Basis von FSMBehaviours zur Verfügung gestellt werden, indem das TA-Behaviour als Eltern-Behaviour für alle Behaviours des kompletten Protkollautomaten definiert wird. Wie in der Abbildung angedeutet umhüllt damit das TA-Behaviour quasi die komplette Konversation in allen Belangen der Transaktionalen Konversation.

Zur Illustration der dynamischen Funktionsweise der Transaktionalen Konversationen wird in Abbildung 7.7 der Konversationablauf aus dem motivierenden Beispiel in Abbildung 2.1 als verteilte Transaktion nach dem Modell der Transaktionalen Konversationen dargestellt. Dabei wird der Übersichtlichkeit zuliebe nur ein MachineAgent – der erfolgreiche – gezeigt.

Zu Beginn der Initialisierungsphase will der OrderAgent ein Kontraktnetz als Transaktionale Konversation ausführen. Dazu erzeugt er ein TA-Behaviour und übergibt ihm als Parameter einen Identifikator, der angibt, welchen Protokollautomat er geschützt ausführen will. Das TA-Behaviour erzeugt dann einen Transaktionskontext und übermittelt diesen an die geeigneten MachineAgents zusammen mit einem Identifikator für den Protokollautomaten, der auf Empfängerseite ausgeführt werden soll. Bei den empfangenden MachineAgents wird nun ebenfalls jeweils ein TA-Behaviour erzeugt, an das auch der empfangene Identifikator übergeben wird. (Zur besseren Verständlichkeit ist die Interaktion zwischen den Agenten und ihren jeweiligen TA-Behaviours in den Abbildungen nicht dargestellt.) Auf beiden Seiten wird dann der eingebettete Protokollautomat (bzw. präziser sein Startknoten) als Kind-Behaviour an die Konversationsausführung

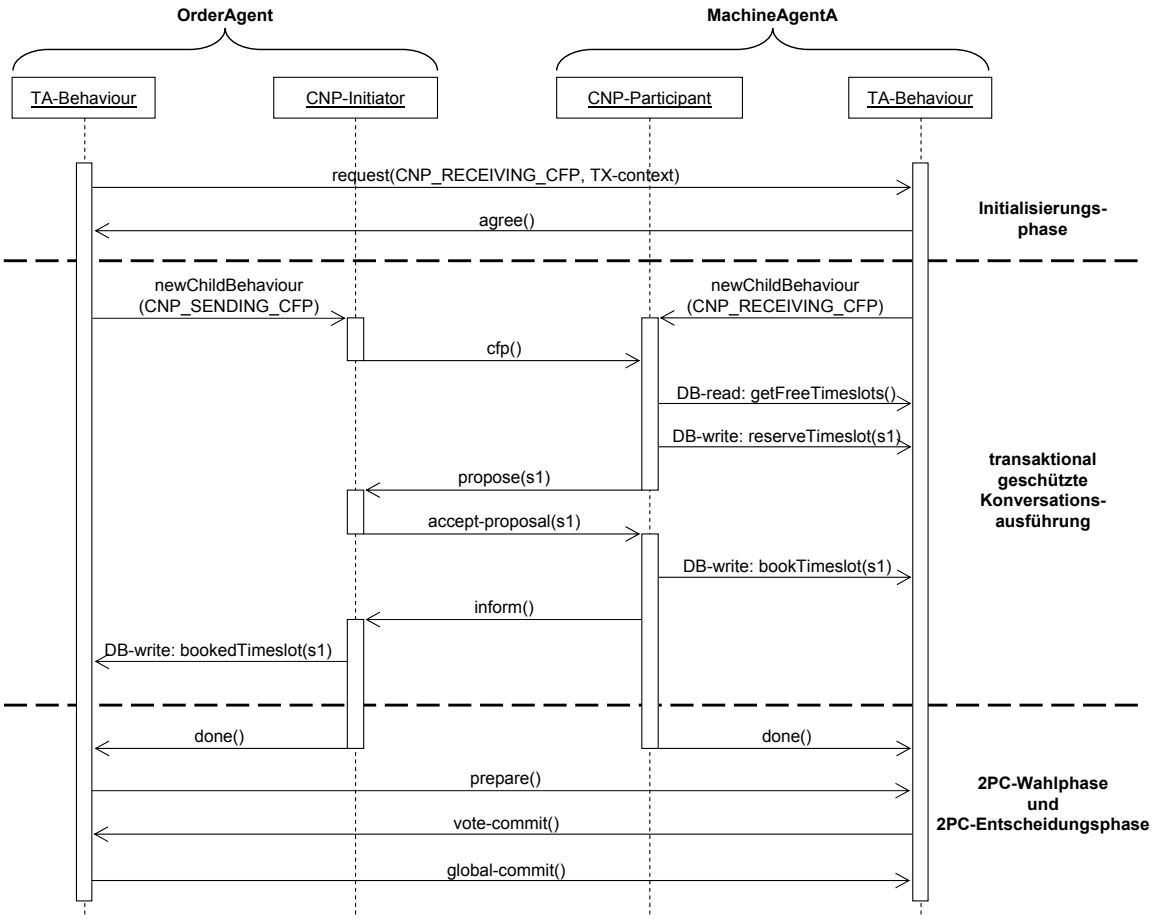


Abbildung 7.7: Beispielhafter Ablauf einer Transaktionalen Konversation

übergeben.

Damit beginnt die zweite Phase der transaktional geschützten Konversationsausführung, in der die beteiligten Seiten auf herkömmliche Weise die Protokollautomaten ausführen. Dabei beobachten die TA-Behaviours die Ausführung und insbesondere die Operationen auf dem Weltmodell. Wenn die Kindknoten innerhalb der verteilten Transaktion auf das Weltmodell zugreifen wollen, dann tun sie das nicht mehr direkt wie zuvor, sondern indirekt über das TA-Behaviour. Im Fall eines solchen indirekten Zugriffes übernimmt dann das TA-Behaviour ggf. die Etablierung einer Verbindung zum Weltmodell über den Ressourcen-Manager und stellt diese Verbindung im richtigen Transaktionskontext unter die Kontrolle des Transaktions-Managers.

Die Indirektion des Zugriffs gibt dem Entwickler einen zusätzlichen Freiheitsgrad. Indem er auswählt, ob ein Zugriff auf das Weltmodell indirekt über das TA-Behaviour oder unmittelbar über eine selbstverwaltete Verbindung zur Ressource erfolgt, legt er fest, ob der Zugriff im Kontext der Transaktion erfolgt oder nicht. Mit den direkten Zugriffen ist es also möglich, Änderungen am Weltmodell außerhalb der Transaktion vorzunehmen. Diese Änderungen sind ggf. auch noch nach einem Rücksetzen der Transaktionalen Konversation verfügbar.

Die Entscheidung, ob am Ende einer Transaktionalen Konversation ein Festschreiben oder Rücksetzen erfolgt, fällt in der dritten Phase, die die Wahl- und Entscheidungsphase des 2PC-Protokolls umfasst. Die Änderungen an den Weltmodellen der beteiligten Agenten werden in dieser Phase über den Erfolg bzw. Misserfolg der gesamten verteilten Transaktion synchronisiert. Dazu stimmen die TA-Behaviours der beteiligten Agenten nach Aufforderung des TA-Behaviours des koordinierenden Agenten ab, der Koordinator trifft eine Entscheidung auf Basis der Rückmeldung aller Beteiligter und verbreitet seine Entscheidung.

7.3.3 Transaktionale Konversationen: Orthogonalität, Einschränkungen und Gegenmittel

Der beschriebene Zuverlässigkeitsmechanismus bringt als Ergebnis die global konsistente Atomizität der Operationen auf dem Weltmodell. Damit ist das in Abschnitt 7.1.3

beschriebene Zuverlässigkeitsproblem gelöst, denn es kann keine dauerhaften veralteten Reservierungen mehr geben: entweder das Kontraktnetzprotokoll lief erfolgreich bis zum Ende und es wurde bei einem MachineAgent eine verbindliche Buchung durchgeführt und die anderen Reservierungen wurden gelöscht oder das Kontraktnetzprotokoll wurde – aus welchem Grund auch immer – abgebrochen und alle Kontraktnetzbezogenen Änderungen am Weltmodell wurden rückgängig gemacht. Wie oben beschrieben, spielt es im Falle eines Abbruches keine Rolle, ob ihm eine Rechnerstörung oder eine Netzwerkstörung zugrundeliegt: das Verfahren, das ursprünglich gegen teilentschärfte Plattformstörungen entwickelt wurde, deckt also einen viel größeren Störungsbereich ab.

Aus Sicht des beschriebenen Feinplanungs-MAS kann der Zuverlässigkeitsmechanismus als weitestgehend orthogonal und damit als investitionsschützend bezeichnet werden, weil die Eingriffe, die in das existierende System notwendig sind, um den Zuverlässigkeitsmechanismus zu benutzen, minimal sind. Die beteiligten Agenten müssen lediglich die Protokolle indirekt gekapselt durch das TA-Behaviour initiieren und die Behaviours, die das eigentliche Kontraktnetzprotokoll implementieren, müssen so abgeändert werden, dass die Zugriffe auf das Weltmodell, die im Kontext der verteilten Transaktion geschützt werden sollen, nicht mehr direkt, sondern indirekt über das TA-Behaviour erfolgen. Von allen weiteren Änderungen, die am System zur Abwicklung der Transaktionalen Konversationen notwendig sind, bleibt der Entwickler unbehelligt. Sie sind hinter den Eingriffspunkten Lebenszyklusverwaltung und Konversationsüberwachung und im TA-Behaviour verborgen. Insbesondere sind die Änderungen der ursprünglichen Implementierung des Feinplanungs-MAS damit auf die Schicht S2 begrenzt und die Implementierung von Weltmodell (S3), Verhaltensgenerierung (S4) und Koordination (S5) bleibt unberührt.

Die Übertragbarkeit des Zuverlässigkeitsmechanismus wird dennoch durch zwei Annahmen eingeschränkt, die den Transaktionalen Konversationen zugrunde liegen und die nicht für beliebige Multiagentensysteme zutreffen:

1. Die Agenten im intraorganisationellen Feinplanungs-MAS sind einem gemeinsamen übergeordneten Ziel stark verpflichtet. Der konsistente Systemzustand kann daher höher bewertet werden als die Autonomie der einzelnen Agenten. Die bereits

diskutierte Autonomieeinschränkung durch die verteilten Transaktionen fällt hier also nicht ins Gewicht, kann aber in anderen stärker kompetitiven Multiagentensystemen zum Hemmschuh werden.

2. Das Weltmodell im Feinplanungs-MAS ist als generische Datenbank realisiert und arbeitet damit auf der Basis von „harten Fakten“. Im Allgemeinen können Agenten jedoch Weltmodelle mit einer komplexeren formalen Semantik aufweisen, auf der explizit gearbeitet wird. Die Vorteile, die sich hierfür bei der Fehlerbehandlung ergeben können, lassen sich unter Verwendung der Transaktionalen Konversationen nicht ausnutzen.

Im Folgenden werden zwei Variationen des TK-Ansatzes andiskutiert, die auf diese Annahmen verzichten und für die die beschriebenen Einschränkungen daher nicht gelten.

Transaktionale Nachrichtenwarteschlangen

Die globale Konsistenz und Atomizität der verteilten Transaktionen, die bei den Transaktionalen Konversationen zum Einsatz kommen, wird um den Preis einer Autonomie-Einschränkung der beteiligten Agenten erkauft. Nicht mehr die individuellen Agenten bestimmen über das Festschreiben ihres Zustandes, sondern die Gemeinschaft der Agenten bestimmt mittels des Transaktions-Managers, ob festgeschrieben werden kann. Insbesondere kann ein einzelner Agent den Abbruch der verteilten Transaktion erzwingen und damit alle anderen beteiligten Agenten zum Zurücksetzen zwingen. Diese Agenten sind dann nicht mehr autonom im ursprünglichen Sinn, was den Bruch einer der definierenden Eigenschaften der Agenten darstellt und besonders in offenen MAS, die auf einen starken Wettbewerb zwischen den Agenten ausgelegt sind, untragbar sein kann.

Es muss also ein Weg gefunden werden, wie man die enge Kopplung des MAS durch die verteilten Transaktionen lockern kann, um die Autonomie wieder herzustellen. Dies geht aber nur auf Kosten des koordinierten Rücksetzens, indem man z.B. die Agenten lokal entscheiden lässt, ob sie zurücksetzen wollen. Die Konversation als Ganzes ist dann jedoch nicht mehr geschützt, d.h. die Konsistenz über das gesamte MAS kann

nicht mehr garantiert werden. Wenn es jedoch gelingt, dass man jeden Agenten einzeln zurücksetzen und wiederherstellen kann und sich darüber hinaus der Ablauf des jeweils lokalen Anteils des Nachrichtenaustauschs innerhalb der Konversation wiederherstellen lässt, dann kann man zumindest feststellen, ob sich der individuelle Anteil der Konversation im Ausgangszustand oder im aktuellen Zustand befindet. Diese Art der Fehlerentschärfung in der Konversationsausführung und -überwachung in S2c ließe sich als Basis für dynamische Fehlerbehandlungsverfahren in der Koordinationsschicht S5 nutzen.

Realisieren lässt sich ein derartiger Mechanismus zur Fehlerentschärfung mit Hilfe von sog. *transaktionalen Nachrichtenwarteschlangen* (engl. *queued transactions*) (siehe auch C.6.2 und Abbildung C.1). Diese bestehen aus drei einzelnen Transaktionen. Die beiden Transaktionen bei den beteiligten Agenten umfassen deren Aktionen und auch das Senden und Empfangen der Nachrichten. In der Terminologie der Nachrichtenwarteschlangen spricht man beim Senden bzw. Empfangen vom Einstellen von Nachrichten in die Warteschlange (engl. *queuing*) bzw. vom Entnehmen von Nachrichten aus der Warteschlange (engl. *dequeuing*). Die dritte Transaktion betrifft die Nachrichtenwarteschlangenverwaltung (engl. *queue manager*) der persistenten und wiederanlaufbaren Nachrichtenwarteschlange und schützt diese insbesondere vor einem Verlust von Nachrichten. Mit dieser Kombination aus Transaktionen lässt sich beim Wiederanlaufen Gewissheit erlangen, ob sich ein Agent bezüglich einer Konversation im Zustand vor Konversationsbeginn oder im aktuellen Zustand der Konversation befindet.

Curry (Curry u. a. 2003; Curry 2003) beschreibt eine mögliche Realisierung transaktionaler Nachrichtenwarteschlangen für das Agentenrahmenwerk JADE. Die dortige Realisierung behandelt die transaktionalen Warteschlangen als eine neue Option zur Nachrichtenübermittlung, die sich neben den bereits existierenden Übermittlungskanälen wie z.B. HTTP-MTP alternativ einsetzen lässt. Sie ist damit als Ansatz in Schicht S2c verortet. Die Verwendung des Ansatzes erfordert vom Entwickler lediglich kleine Eingriffe in die Plattform-Konfiguration und eine Umstellung der Nachrichten-Adressierung auf das Format der Nachrichtenwarteschlangen. Da keine weiteren Artefakte der MAS-Programmierung anzupassen sind, ist der Mechanismus als orthogonal und investitionsschützend einzustufen. Allerdings gilt zu beachten, dass z.B. das Zuverlässigkeitsproblem des Feinplanungs-MAS noch nicht vollständig durch die transaktionalen Nachrichten-

warteschlangen gelöst ist, sondern dass lediglich eine Fehlerentschärfung gegen Rechner- und Netzwerkstörungen stattfindet, die als Ausgangspunkt für einen zusätzlichen höherschichtigen Fehlerbehandlungsmechanismus dienen kann.

Semantik-bewusste Transaktionale Konversationen

Die transaktionalen Nachrichtenwarteschlangen haben gezeigt, dass die Verwendung eines Transaktions-basierten Ansatzes zur Fehlerbehandlung nicht wie bei den Transaktionalen Konversationen zwangsläufig zu einer Einschränkung der Autonomie führen muss. Die zweite Einschränkung der Transaktionalen Konversationen beruht auf der Annahme, dass das Weltmodell der Agenten einer generischen Datenbank entspricht. Im Allgemeinen können Agenten jedoch komplexere Weltmodelle mit einer expliziten formalen Semantik aufweisen, die nicht nur auf Fakten beruhen, sondern die z.B. Abstufungen über die Sicherheit des Wissens zulassen oder das Weltmodell nach logischen Kategorien unterteilen, wie das beispielsweise beim BDI-Modell (Bratman 1987; Bratman u. a. 1988) der Fall ist.

Die komplexere Struktur dieses Weltmodells lässt sich bei einer Transaktions-basierten Fehlerbehandlung ausnutzen, indem das Rücksetzen einer Änderung am Weltmodell nicht mehr automatisch gleichgesetzt werden muss mit dem „spurlosen Vergessen“ der Änderung wie bei den Transaktionalen Konversationen. In der Sprache der Transaktionsverwaltung handelt es sich bei den Datenobjekten, die dem Transaktionsmodell zugrunde liegen, also nicht mehr um quasi Semantik-freie Speicherseiten, wie beim sog. Seitenmodell, sondern es werden Datenobjekte mit einer abweichenden komplexeren Semantik als Grundlage eines neuen Transaktionsmodells eingeführt[10].

Will man die Semantik des Weltmodells und der Nachrichten in einem Fehlerbehandlungsmechanismus ausnutzen, der an die Konversationsausführung und -überwachung gebunden ist, dann muss man bei der Fehlerbehandlung in die Nachrichten „hineinschauen“, um deren Semantik zu analysieren. Grundsätzlich lässt sich für die Semantik der Nachrichten und auch der Artefakte des Weltmodells unterscheiden zwischen einer domänenabhängigen Semantik und einer generischen Semantik. Die domänenabhängige

[10] Zum Hintergrund siehe Anhangsabschnitt C.3

Semantik soll hier nicht weiter betrachtet werden, da sie nur zu domänenspezifischen Zuverlässigkeitsmechanismen beitragen kann. Die generische Semantik von Nachrichten und Weltmodell kann hingegen für alle MAS hilfreich sein, die auf demselben semantischen Modell aufbauen.

Als Beispiel enthält im Feinplanungs-MAS die Ausschreibung `cfp()` durch den OrderAgent domänenspezifische Informationen, wie etwa technische Details zu den gesuchten Produktionsschritten, und das Angebot `propose()` eines MachineAgent kann Zeiträume mit Rüstzeiten und Bearbeitungszeiten enthalten. Die Verwendung dieser domänenabhängigen Semantik bei der Fehlerbehandlung führt naturgemäß zu sehr (domänen-)spezifischen – wenngleich auch mächtigen – Fehlerbehandlungsmechanismen.

Die beiden genannten Nachrichten haben aber auch noch eine generische Semantik, die sich bei der Fehlerbehandlung ausnutzen lässt: beim `cfp()` handelt es sich immer um eine Aufforderung an den Empfänger, ein Angebot abzugeben und beim `propose()` um eben dieses. Wie in Abschnitt 7.1.3 diskutiert wurde, kann der Verlust – aus welchem Grund auch immer – einer Angebots-Nachricht durch den verbindlichen Charakter des Angebots und die damit einhergehende Reservierung zu einem ernsthaften Zuverlässigkeitsproblem führen, während aus dem Verlust der Aufforderung zur Angebotsabgabe in der Regel kein Zuverlässigkeitsproblem resultiert.

Das Beispiel macht klar, dass die generische Semantik der Nachrichten an deren jeweiligem Sprechakt festgemacht wird. Im Gegensatz zum Anschein im vereinfacht dargestellten Beispiel ist diese Semantik keinesfalls beliebig oder unscharf, sondern sie ist z.B. für die FIPA-Sprechakte (FIPA 2002j) und für das BDI-Modell (Bratman 1987; Bratman u. a. 1988) mittels modaler Logik formal spezifiziert. Auf Basis dieser formalen Spezifikation der Semantik von Weltmodell und Sprechakten lassen sich auch formale Semantiken für Transaktionsmodelle entwickeln, wie verschiedene Arbeiten über die Zusammenführung von Konversationen, Weltmodell und Transaktionen zeigen (Esterline 2002; Rovatsos 2007).

Mit dem Ansatz der sog. Semantik-bewussten Transaktionalen Konversationen lassen sich unter Ausnutzung der generischen Semantik und der Konversationen mächtige

Fehlerbehandlungsmechanismen auf Transaktions-Basis gegen Rechner- und Netzwerkstörungen entwickeln. Diese sind in der Referenzarchitektur bei der Konversationsausführung in S2c und bei der Auswertung der Sprechaktsemantik und den dynamischen Ontologie-Diensten von S3 verortet. Im Vergleich zum Ansatz der Transaktionalen Konversationen wird also zusätzlich die Schicht S3 mit in den Mechanismus einbezogen, womit prinzipiell die Gefahr besteht, dass der Mechanismus nicht orthogonal in existierende Systeme eingebettet werden kann, weil er verschiedene Aspekte der MAS-Programmierung eng miteinander verknüpft. Tatsächlich ist jedoch in Multiagentensystemen, bei denen die Annahme der expliziten Handhabung einer formalen Semantik im Weltmodell erfüllt ist, gerade diese enge Verknüpfung a priori gegeben. Für ein Zielsystem, das die geänderten Annahmen erfüllt, ist die Orthogonalität also wiederum gewährleistet.

7.4 Zusammenfassung

Die Fallstudie Transaktionale Konversationen im vorliegenden Kapitel ist ein Beispiel für die Verwendung der Zuverlässigkeitsreferenzarchitektur bei der Auswahl von existierenden und der Erstellung von neuen Zuverlässigkeitsmechanimen. Ausgehend von einer Beschreibung des ursprünglichen mit Zuverlässigkeitsproblemen behafteten Multiagentensystems zur Produktionsfeinplanung wurde zunächst anhand der Symptome eine Analyse der Störung vorgenommen. Unterstützt durch die Zuverlässigkeitsreferenzarchitektur wurden in deren Verlauf der effektive Fehlzustand und die Fehlerursache identifiziert und an den verantwortlichen Komponenten des MAS festgemacht.

Mit der identifizierten Fehlerursache – einer Rechnerstörung, die zu einer teilentschärften Plattformstörung abgemildert wird – wurde dann die Zuverlässigkeitslandkarte der existierenden Zuverlässigkeitsmechanismen zu Rate gezogen und dort besonders der Pfad Rechnerstörungen und unerwartetes Agentenverhalten betrachtet. Eine Untersuchung der erweiterten Wiederanlaufverfahren gegen teilentschärfte Plattformausfälle führte zu dem Ergebnis, dass durch spezielle Annahmen und einen Mangel an Orthogonalität der existierenden Mechanismen keiner von ihnen ohne erhebliche Eingriffe im Feinplanungs-MAS anwendbar ist.

Während der nun erforderlichen Eigenentwicklung eines Zuverlässigkeitsmechanismus helfen die Überlegungen zur Übertragbarkeit der Zuverlässigkeitsreferenzarchitektur insbesondere bei der Identifikation eines geeigneten Eingriffspunktes. Es zeigte sich, dass sich das Zuverlässigkeitsproblem abbilden lässt auf ein Problem der globalen Konsistenzerhaltung der Agentenzustände innerhalb des Kontraktnetzprotokolls, das dem Feinplanungs-MAS zugrunde liegt. Der entwickelte Ansatz der Transaktionalen Konversationen beruht auf einer Abbildung der Konversationen auf verteilte Transaktionen und ist an den Komponenten Lebenszyklusverwaltung und Konversationsüberwachung des FIPA-basierten Multiagentensystems festgemacht. Durch die scharf abgegrenzte Verortung des Ansatzes ist dieser aus Sicht des Zielsystems orthogonal und kann mit nur minimalen Anpassungen vom Feinplanungs-MAS eingesetzt werden.

Die Transaktionalen Konversationen beruhen unter anderem auf zwei Annahmen, die die Übertragbarkeit auf andere Multiagentensysteme einschränken, nämlich auf der Annahme, dass die geschützten Agenten für den Zuverlässigkeitsgewinn einen Teil ihrer Autonomie aufzugeben bereit sind, und auf der Annahme, dass ein vergleichsweise einfaches Weltmodell in Form einer Datenbank zum Einsatz kommt. Für den Fall, dass diese Annahmen nicht gelten, werden zwei Varianten des Ansatzes skizziert und hinsichtlich ihrer Mächtigkeit und Orthogonalität diskutiert.

Es zeigt sich, dass auch die Ansätze der transaktionalen Nachrichtenwarteschlangen bei Aufrechterhaltung der Autonomieforderung und die Semantik-bewussten Transaktionalen Konversationen für Systeme mit expliziter Handhabung einer formalen Semantik des Weltmodells orthogonale Zuverlässigkeitsmechanismen aus Sicht des jeweiligen Zielsystems sind. Mit Hilfe der Zuverlässigkeitsreferenzarchitektur gelingt es in den Beispielen der Fallstudie also, unter den jeweils geltenden Annahmen des Zielsystems orthogonale Zuverlässigkeitsmechanismen zu entwickeln.

Kapitel 8

Zusammenfassung und Ausblick

8.1 Inhaltliche Zusammenfassung

Multiagentensysteme erfreuen sich seit zwei Jahrzehnten einer intensiven Aufmerksamkeit durch die Forschung. In dieser Zeit gelang es vielfach, das große Potenzial der Agententechnologie in komplexen Anwendungsumgebungen nachzuweisen. Dennoch sind nur vergleichsweise wenige voll ausgebildete Multiagentensysteme im industriellen Einsatz bekannt – meist werden dort bestimmte Technologien isoliert, die im Kontext der Agentenforschung entwickelt wurden, und in die vorhandenen herkömmlichen Systeme eingebettet, womit jedoch zwangsläufig nicht mehr das volle Potenzial der Agententechnologie ausgeschöpft werden kann. Ein Grund für die schwache Marktdurchdringung der vollwertigen Agentensysteme sind die erhöhten Zuverlässigkeitsanforderungen in den geeigneten kommerziellen Anwendungsumgebungen bei der gleichzeitig großen Komplexität der Multiagentensysteme selbst, die eine Erreichung eines hohen Zuverlässigkeitsgrades erschwert.

Diese Erkenntnis führte im Laufe der Jahre zu einer Vielfalt an Mechanismen zur Erhöhung der Zuverlässigkeit von Multiagentensystemen, die jedoch meist dieselbe entscheidende Schwäche aufweisen: Die Zuverlässigkeitsmechanismen wurden im Zusammenhang mit spezifischen anderen Aspekten der Agentenprogrammierung entwickelt, die im eigentlichen Forschungsinteresse lagen, und nicht generisch für beliebige Multiagentensysteme. Damit ist eine Übertragung der Zuverlässigkeitsmechanismen auf andersartige

Multiagentensysteme in der Regel mit einem hohen Aufwand zu deren Anpassung verbunden, d.h. die Mechanismen sind nicht orthogonal zu den Programmieraspekten der existierenden Multiagentensysteme – ein Investitionsschutz für diese Systeme ist daher nicht gegeben.

Hier liegt das Ziel der Arbeit: Sie soll einen Beitrag leisten zur Entwicklung von Mechanismen zur Steigerung der Zuverlässigkeit von Multiagentensystemen unter der Randbedingung des Investitionsschutzes. Als wichtigste Aufgabe wird hierbei eine Systematisierung der Programmierartefakte, Störungen und Zuverlässigkeitsmechanismen angesehen. Die Systematisierung erfolgt in Form einer Architekturbetrachtung, die zu einer **Zuverlässigkeitsreferenzarchitektur für Multiagentensysteme** führt. Sie unterstützt den Entwickler eines Multiagentensystems bei der Auswahl geeigneter existierender Zuverlässigkeitsmechanismen und/oder bei der Entwicklung seines eigenen orthogonalen Zuverlässigkeitsmechanismus.

Bei der Erstellung der Zuverlässigkeitsreferenzarchitektur wurde wie folgt vorgegangen: Nach Motivation und Präzisierung der Aufgabenstellung anhand eines Beispiels aus der Fertigungslogistik, das auch die abschließende Fallstudie wieder aufgreift, wurde zunächst eine wesentliche Voraussetzung für die Zuverlässigkeitsreferenzarchitektur entwickelt, nämlich eine allgemeingültige und vollständige Referenzarchitektur für Multiagentensysteme. Diese leistet eine Systematisierung der Agententechnologie, die eine Identifikation aller Aspekte der Agentenprogrammierung und ihres Zusammenspiels ermöglicht. Basis des methodischen Vorgehens bei der Erstellung der Referenzarchitektur sind die anerkannten definierenden Eigenschaften der Multiagentensysteme, etablierte Standards und verwandte Referenzarchitekturen. Der Nachweis der Allgemeingültigkeit und Vollständigkeit erfolgte durch eine Untersuchung ihrer Kompatibilität zu weiteren optionalen Eigenschaften von Multiagentensystemen und ihrer Konformität zu anderen bekannten Agentenarchitekturen.

Die Zuverlässigkeitsreferenzarchitektur baut auf der Referenzarchitektur auf und erweitert diese um den Zuverlässigkeitsaspekt. Hierfür wurde ein agentenunspezifisches Fehler- und Fehlerbehandlungsmodell für zuverlässige Komponenten in Schichtenarchitekturen entwickelt und dieses auf die Referenzarchitektur für Multiagentensysteme aufgeprägt. Damit wird eine Verortung spezifischer Störungen nach ihrem Entstehungsort und der

zugehörigen Behandlungsoptionen möglich.

Die abstrakte Darstellung der Fehlerbehandlungsoptionen in der Zuverlässigkeitsreferenzarchitektur wurde durch die Verortung einer Vielzahl bekannter Fehlerbehandlungsmechanismen zu einer Zuverlässigkeitslandkarte für Multiagentensysteme ausgebaut. Die Landkarte zeigt, wie Störungen durch die Schichten eines Multiagentensystems aufsteigen können und in den Schichten behandelt werden können. Damit weist die Landkarte zum einen die Verortungsfähigkeit der Zuverlässigkeitsreferenzarchitektur nach und gibt zum anderen Entwicklern von Multiagentensystemen eine Hilfestellung bei der Auswahl geeigneter Zuverlässigkeitsmechanismen. Der Nachweis der Übertragbarkeit der Zuverlässigkeitsreferenzarchitektur erfolgte durch eine Abbildung der Fehlerbehandlungspfade auf die bekannten existierenden Agentenarchitekturen. Den Entwicklern eines Multiagentensystems hilft die Übertragbarkeitsdiskussion bei der Entwicklung eigener Zuverlässigkeitsmechanismen, falls in der Zuverlässigkeitslandkarte kein geeigneter Mechanismus gefunden werden konnte.

Die abschließende Fallstudie gibt ein Beispiel für den Einsatz der Zuverlässigkeitsreferenzarchitektur aus Sicht eines Entwicklers, der die Zuverlässigkeit eines existierenden Multiagentensystems zur Produktionsfeinplanung gegen Rechnerstörungen erhöhen will. Die Betrachtung der Mechanismen in der Zuverlässigkeitslandkarte führt zu dem Ergebnis, dass kein existierender Mechanismus das Zuverlässigkeitsproblem ohne tiefe Eingriffe in das existierende System beheben kann. Unter Verwendung der Zuverlässigkeitsreferenzarchitektur und der Ergebnisse aus ihrer Übertragbarkeitsdiskussion wird daher ein Transaktions-basierter Zuverlässigkeitsmechanismus entwickelt, der sich orthogonal in das bestehende System einbettet und der sich auch unter Variation der impliziten einschränkenden Annahmen so anpassen lässt, dass wiederum orthogonale Mechanismen unter den geänderten Voraussetzungen resultieren.

Die Fallstudie demonstriert damit, wie die Zuverlässigkeitsreferenzarchitektur zur Steigerung der Zuverlässigkeit von Multiagentensystemen unter der Randbedingung des Investitionsschutzes beiträgt. Bewusst ausgeklammert wurde die Frage, wie die Erhöhung der Zuverlässigkeit mit anderen nicht-funktionalen Eigenschaften der Multiagentensysteme korreliert, z.B. mit deren Leistung. Im Allgemeinen konnte beobachtet werden, dass der Zusatzaufwand beim Einsatz konkreter Zuverlässigkeitsmechanismen im

Rahmen der Zuverlässigkeitsreferenzarchitektur zu einer vertretbaren Leistungseinbuße geführt hat. Dennoch muss hier der Entwickler im Vorfeld zwischen den Optimierungen für die verschiedenen nicht-funktionalen Eigenschaften abwägen. Hier wäre vorstellbar, statt der Zuverlässigkeitseigenschaft andere nicht-funktionale Eigenschaften auf die Referenzarchitektur aufzuprägen und damit den Grundansatz der Referenzarchitektur für die Optimierung beliebiger Eigenschaften zu verallgemeinern (siehe auch Abschnitt 8.3).

8.2 Wissenschaftlicher Beitrag

Als wissenschaftlicher Beitrag der vorliegenden Arbeit lassen sich drei wesentliche Ergebnisse ausmachen, die diese vom Stand der Wissenschaft abheben.

Allgemeingültige und vollständige Referenzarchitektur für Multiagentensysteme Die Referenzarchitektur systematisiert auf einer konzeptuellen Abstraktionsstufe erstmals alle Aspekte der Programmierung von Multiagentensystemen in einem integrierten Modell. Die bekannten verwandten Arbeiten in Form von etablierten Standards (Patil u. a. 1998; FIPA 2002a) und Referenzarchitekturen (z.B. Modi u. a. 2006; Weyns u. a. 2006; Müller 1996; Truszkowski 2006) betrachten durchweg nur gewählte Ausschnitte der Agentenprogrammierung, da sie vor dem Hintergrund spezifischer Zielsetzungen oder Anwendungsgebiete entstanden sind. Damit wurde methodisch ein Rahmen geschaffen, der unabhängig vom Kontext der Zuverlässigkeit eine Reflektion aller Entwicklungsaspekte von Multiagentensystemen erlaubt.

Zuverlässigkeitsreferenzarchitektur für Multiagentensysteme Das Herz der Arbeit ist die Zuverlässigkeitsreferenzarchitektur, die im Vergleich zu den existierenden Arbeiten zur Zuverlässigkeitssteigerung von Multiagentensystemen (z.B. Nagi 2001b; Sandholm und Lesser 2002; Klein 2003; Rimassa 2004, u.v.a.m.) einen grundlegend neuen Ansatz verfolgt. Die verwandten Arbeiten entwickeln konkrete Zuverlässigkeitsmechanismen, die zwar zum Teil den Anspruch auf Allgemeingültigkeit erheben, denen jedoch zumeist spezifische Systemmodelle oder Annahmen

zugrunde liegen, die eine universelle Einsetzbarkeit verhindern. Die Zuverlässigkeitsreferenzarchitektur bietet hingegen keinen konkreten Zuverlässigkeitsmechanismus, sondern einen Rahmen, der Entwicklern hilft, geeignete orthogonale Zuverlässigkeitsmechanismen zu identifizieren oder selbst zu entwickeln.

Zuverlässigkeitslandkarte für Multiagentensysteme Die Zuverlässigkeitslandkarte verortet Störungen und die bekannten konkreten Fehlerbehandlungsmechanismen innerhalb der Zuverlässigkeitsreferenzarchitektur systematisch entlang von Fehlerbehandlungspfaden und gibt Entwicklern von Multiagentensystemen damit ein Mittel an die Hand, um zielgerichtet existierende Fehlerbehandlungsmechanismen für ein gegebenes Zuverlässigkeitsproblem zu identifizieren. In der Literatur ist dem Autor keine vergleichbare umfassende systematische Vorgehensweise für Zuverlässigkeitsmechanismen für Multiagentensysteme bekannt.

8.3 Ausblick

Basierend auf den Ergebnissen der Arbeit sind grundsätzlich drei Optionen zur Weiterführung denkbar: eine direkte Anwendung, eine Fortführung von Teilaspekten und eine Übertragung der Herangehensweise auf andere Systemklassen und Eigenschaften.

Direkte Anwendung der Ergebnisse In der abschließenden Fallstudie wurde mit den Transaktionalen Konversationen bereits eine erste voll ausgearbeitete Anwendung der Zuverlässigkeitsreferenzarchitektur vorgestellt. Die Bewertung des Ansatzes hat gezeigt, dass dabei recht wenige einschränkende Annahmen gemacht wurden und sich der entstandene orthogonale Zuverlässigkeitsmechanismus nicht nur im Feinplanungs-MAS, sondern in einer Vielzahl ähnlich aufgebauter Multiagentensysteme anwenden lässt. Mit den Ansätzen der transaktionalen Nachrichtenwarteschlangen und der Semantik-bewussten Transaktionalen Konversationen wurden außerdem zwei Möglichkeiten angerissen, wie sich auch die einschränkenden Annahmen überwinden lassen. Eine vollständige Ausarbeitung der beiden Ansätze würde zu einer Familie von Zuverlässigkeitsmechanismen führen, mit der sich eine große Bandbreite von Multiagentensystemen gegen Rechner-

und Netzwerkstörungen schützen ließe. Es könnte daher der Verbreitung von Multiagentensystemen zuträglich sein, diese Ausarbeitung durchzuführen und die Mechanismen einer breiteren Öffentlichkeit zugänglich zu machen. Unter wissenschaftlichen Gesichtspunkten könnte besonders der Ansatz der Semantik-bewussten Transaktionalen Konversationen von Interesse sein, da hinter der Abbildung zwischen der formal-logischen Semantik des Agentenmodells und der Semantik des Transaktionsmodells spannende und ergiebige Forschungsfragestellungen zu vermuten sind.

Fortführung von Teilaspekten der Arbeit Bei der Veröffentlichung und wissenschaftlichen Präsentation der Ergebnisse stieß neben der Zuverlässigkeitsreferenzarchitektur insbesondere die Referenzarchitektur für Multiagentensysteme unabhängig vom Zuverlässigkeitsaspekt auf Interesse. Die Referenzarchitektur könnte eigenständig weiterverfolgt werden und durch formalere Modelle und Spezifikationen der einzelnen Komponenten und ihrer Schnittstellen weiter geschärft werden, um sie dann z.B. in den wiederaufgenommenen Standardisierungsprozess der FIPA unter dem Dach der IEEE[1] zurückzuspielen. Aus Sicht der FIPA ließe sich über eine Standardisierung der Schnittstellen auch auf den höheren Schichten eine Austauschbarkeit der vielfältigen verfügbaren Software-Komponenten erreichen, die bislang nicht gegeben ist. Für die Hersteller von Software-Komponenten für Agenten ergäbe sich auf diese Weise eine größere potenzielle Benutzergemeinde und für die Anwender ein verbesserter Investitionsschutz.

Übertragung der Arbeit auf andere Systemklassen und Eigenschaften Die grundsätzliche Herangehensweise der Arbeit, ein generisches Fehler- und Fehlerbehandlungsmodells auf eine geschichtete Referenzarchitektur für Multiagentensysteme aufzuprägen, lässt sich auf zwei Arten verallgemeinern und übertragen.

Zum einen könnten, wie oben bereits angesprochen, generische Modelle für andere nichtfunktionale Eigenschaften auf die geschichtete Referenzarchitektur für Multiagentensysteme aufgeprägt werden. Resultieren würden spezialisierte Referenzarchitekturen, die

[1] Siehe http://www.fipa.org/index.html

den Entwicklern als Unterstützung bei der investitionsschützenden Optimierung ihrer Multiagentensysteme dienen könnten, z.B. für die Aspekte Leistung oder Sicherheit.

Zum anderen ließe sich der Zuverlässigkeitsaspekt auf andere Systemklassen mit geschichteten Referenzarchitekturen übertragen. Die dabei entstehenden Zuverlässigkeitsreferenzarchitekturen sollten auch für diese Systemklassen Hilfestellung bei der Auswahl und Entwicklung orthogonaler Zuverlässigkeitsmechanismen geben können. Vorstellbar wäre beispielsweise eine Übertragung auf die folgenden Systemklassen:

- Service-Orientierte Architekturen (Papazoglou 2008)
- Grid-Computing-Systeme (Foster u. a. 2001)
- Dreistufige Dienstnehmer/Dienstgeber-Architekturen (engl. *three-tier architectures*) (Eckerson 1995)
- Datenbasisverwaltungssysteme (Lockemann und Dittrich 2004)
- Datenkommunikationssysteme nach ISO/OSI (Halsall 1998)

Besonders in den neueren Gebieten des Grid-Computing und der Service-basierten Systeme könnten auf diese Weise grundlegende Beiträge zur Systematisierung der Zuverlässigkeit erfolgen.

Anhang A

Fachbegriffe und Übersetzungen

In der folgenden Tabelle A.1 werden zur Verbesserung der Zuordnung in der verwendeten englischsprachigen Literatur einige der in der Arbeit verwendeten Fachbegriffe zusammen mit ihrer englischen Übersetzung angegeben.

Ausfall – failure
Ausgangszustand – initial state
Dienst – service
Ergebniszustand, regulärer – regular state
Ergebniszustand, tragbarer – sustainable state
Ergebniszustand, unveränderter – old/optional state
Ergebniszustand, verheerender/unzulässiger – incorrect state
Ergebniszustand, zulässiger – acceptable state
Fehlerbehandlung – error processing
Fehlerbeseitigung – fault resilience
Fehlereinschluss – fault containment
Fehlerentschärfung – fault mitigation
Fehlereskalation, Fehlerpropagierung – error exposure
Fehlertoleranz – fault tolerance
Fehlerursache – fault
Fehlervermeidung – fault avoidance
Fehlzustand, effektiver – effective error

Fehlzustand, latenter	latent error
Rücksetzen	recovery, backward recovery
Zuverlässigkeit	dependability

Tabelle A.1: Fachbegriffe und Übersetzungen

Anhang B

Das Anwendungsszenario Produktionsplanung und -steuerung

In diesem Anhang wird das Anwendungsszenario *„Produktionsplanung und -steuerung“* kurz illustriert, das in der vorliegenden Arbeit den Hintergrund zum motivierenden Beispiel in Kapitel 2 und zur Fallstudie in Kapitel 7 liefert.

Die Produktionsplanung und -steuerung (PPS) ist eines der Anwendungsgebiete, das die in Kapitel 1 genannten Charakteristika aufweist, die Anwendungsgebiete für einen Einsatz der Agententechnologie prädestiniert erscheinen lässt. Nicht verwunderlich also, dass in der Literatur eine Vielzahl von Ansätzen zur agentenbasierten Produktionsplanung und -steuerung beschrieben werden, die jeweils bestimmte Teilfunktionalitäten der PPS abdecken (Shen und Norrie 1999; Parunak 2000; Cetnarowicz und Kozlak 2002; Pechoucek u. a. 2004).

Die PPS unterteilt sich in die Produktionsplanung, die die Abläufe auf der Produktionsebene eines Unternehmens plant, und die Produktionssteuerung, die für die korrekte Umsetzung der Planung verantwortlich zeichnet (Glaser u. a. 1992). In der betriebswirtschaftlichen Literatur sind verschiedene gleichbedeutende funktionale Gliederungen der PPS bekannt (vgl. Kurbel 2005; Zäpfel 1996; Hansmann 1994; Hoitsch 1993; Glaser u. a. 1992, u.v.a.m.). Verallgemeinert lassen sich folgende Teilfunktionalitäten für die Produktionsplanung erkennen:

Produktionsprogrammplanung Die Produktionsprogrammplanung bestimmt, welche Produkte in welchem Umfang in einem Unternehmen gefertigt werden sollen.

Mengenplanung Bei der Mengenplanung wird basierend auf dem Produktionsprogramm, den Stücklisten und Arbeitsplänen und den Lagerbeständen der Bedarf an Zwischenprodukten und Rohstoffen ermittelt.

Produktionsprozessplanung Wichtigster Aspekt der Produktionsprozessplanung sind Termin- und Kapazitätsplanung, bei denen für jeden (Teil-)Auftrag der früheste und späteste Arbeitsbeginn geplant wird und kontrolliert wird, ob für diese Grobplanung ausreichende Kapazitäten aller Produktionsfaktoren verfügbar sind.

Die Produktionssteuerung umfasst folgende Funktionen:

Auftragsfreigabe Aus der Produktionsprozessplanung wird ein grober Plan an die Produktionssteuerung weiter gegeben. Diese Grobplanung wird in der Auftragsfreigabe in eine Feinplanung überführt. In diesem Schritt findet auch die zeitliche Zuordnung von Arbeitsschritten zu Maschinen, also die sog. Maschinenbelegungsplanung, statt.

Auftragsüberwachung Durch unzureichend berücksichtigte Einflussfaktoren oder Störungen kann auf der Produktionsebene der Ist-Zustand zum Ausführungszeitpunkt vom Soll-Zustand aus der Feinplanung abweichen. Die Auftragsüberwachung verarbeitet die entsprechenden Statusmeldungen und stößt geeignete Gegenmaßnahmen an, wie z.B. eine Reallokation der Ressourcen, d.h. sie löst u.a. neue Iterationen der Maschinenbelegungsplanung aus.

Von den beschriebenen Aufgaben der Produktionsplanung und -steuerung scheint vor allem die *Feinterminierung in der Auftragsfreigabe*, d.h. die Allokation produktionstechnischer Ressourcen für einen Auftrag besonders gut auf die Agententechnologie zugeschnitten, was sich in der großen Anzahl entsprechender Veröffentlichungen nieder schlägt (Bussmann und Schild 2000; Dang und Frankovic 2002, u.v.a.m.).

Ein Verfahren zur Ressourcenallokation mittels Agenten, das besonders häufig angewandt wird und das auch in der vorliegenden Arbeit im motivierenden Beispiel (siehe Abschnitt 2.1) und in der Fallstudie (siehe Abschnitt 7.1) herangezogen wurde, ist die

Auftragsvergabe nach dem sog. *Contract Net Protocol* (Smith 1980; Parunak 1987; FIPA 2002k) oder nach einer seiner Varianten (Fischer u. a. 1995; Shen und Norrie 1998; Knabe u. a. 2002; FIPA 2002m; Aknine u. a. 2004).

Anhang C

Spezifische Grundlagen der Transaktionsverwaltung

In der Fallstudie in Kapitel 7 wird ein Zuverlässigkeitsmechanismus auf Basis von Techniken der Transaktionsverwaltung entwickelt. Bei der dortigen Beschreibung des Mechanismus werden auch die nötigsten Grundlagen in aller Kürze dargestellt. Da eine weiter reichende Kenntnis der Transaktionsverwaltung nicht bei allen Lesern vorausgesetzt werden kann, diese aber dem tieferen Verständnis des Zuverlässigkeitsmechanismus zuträglich ist, erfolgt in diesem Anhang eine Einführung in die spezifischen Grundlagen der Transaktionsverwaltung. Für eine umfassende Darstellung sei auf die Lehrwerke von Weikum und Vossen (2002) und Bernstein und Newcomer (1997) verwiesen.

C.1 Ziele der Transaktionsverwaltung

Die Transaktionsverwaltung (TAV) hat ihren Ursprung im Bereich der Datenbasisverwaltungssysteme (DBMS), die ihren Benutzern den geteilten Zugriff auf einen gemeinsamen, besonders geschützten Datenbestand ermöglichen sollen. Im Kontext der DBMS übernimmt die Transaktionsverwaltung eine wesentliche Rolle, indem sie folgende beide Aufgaben als Teil der Datenbasisverwaltung für den Anwendungsprogrammierer transparent übernimmt:

Nebenläufigkeitskontrolle (engl. *Concurrency Control*) Der nebenläufige Zugriff mehrerer Benutzer auf denselben Datenbestand kann ohne spezielle Maßnahmen zu unerwünschten Wechselwirkungen führen, z.B. indem sich die Datenbankoperationen der Benutzer gegenseitig stören und so zu (semantisch) fehlerhaften Ausgängen der Programme (Operationsfolgen) der einzelnen Benutzer führen. Daher müssen von der Nebenläufigkeitskontrolle Synchronisationsprotokolle realisiert werden, die einen korrekten und effizienten Zugriff auf die Datenbank erlauben, dem Benutzer also in beiden Beziehungen eine exklusive Nutzung vorspielen.

Wiederherstellung (engl. *Recovery*) Fehler in Datenbanken können nicht nur durch Wechselwirkungen zwischen nebenläufigen Anwendungen entstehen, sondern auch durch Fehler in den Anwendungsprogrammen selbst oder in ihrer Infrastruktur. In diesem Fall gilt es zu verhindern, dass Spuren von unvollständig ausgeführten Anwendungen in der Datenbank überdauern können und damit die Ergebnisse künftiger Anwendungsläufe verfälschen.

Der Anwendungsprogrammierer muss sich bei DBMS, die obige Funktionalitäten realisieren, keine weiteren Gedanken über Nebenläufigkeit und Fehlerbehandlungsmaßnahmen machen. Die Anwendungsprogrammierung kann nun erfolgen, wie in einer exklusiv genutzten fehlerfreien Umgebung; lediglich die zu schützenden Teile der Anwendung müssen entsprechend markiert werden.

C.2 Das Transaktionskonzept

Nebenläufigkeitskontrolle und Wiederherstellung sind eng verknüpft mit dem *Transaktionskonzept* als zentraler Abstraktion. Eine Transaktion ist eine Folge von Datenbankoperationen innerhalb eines Anwendungsprogramms, die als unteilbare Einheit aufgefasst werden soll. Dazu müssen innerhalb des Anwendungsprogramms lediglich die Transaktionsgrenzen durch die entsprechenden Primitive der Datenbankschnittstelle gekennzeichnet werden. I.d.R. werden hierfür die Primitive `begin` für den Transaktionsbeginn, `commit` für ein erfolgreiches Transaktionsende und `abort` für den Transaktionsabbruch verwendet.

Untrennbar mit dem Transaktionkonzept sind die sog. *ACID-Eigenschaften* verbunden. Unter diesem Acronym werden die Eigenschaften subsumiert, die üblicherweise von einem DBMS für eine Transaktion garantiert werden:

Atomizität (engl. *Atomicity*) Transaktionen werden *atomar*, d.h. vollständig oder gar nicht und nicht teilweise ausgeführt. Atomizität wird auch als das „Alles-oder-Nichts"-Prinzip der Transaktionen bezeichnet. Die Änderungen, die eine Transaktion an der Datenbasis vornimmt, werden nur durch das erfolgreiche Beenden der Transaktion mit einem `commit` nach außen sichtbar. Im Falle eines Transaktionsabbruchs durch ein `abort` aus dem Anwendungsprogramm oder eines Fehlers beliebigen Ursprungs, der die vollständige Ausführung verhindert, hinterlässt die Transaktion keine Spuren in der Datenbasis. Ggf. müssen dazu alle bereits vorgenommenen Änderungen am Datenbestand rückgängig gemacht werden. Die Datenbasis befindet sich dann in einem Zustand, der gleich (oder äquivalent) dem Ausgangszustand vor Transaktionsbeginn ist.

Konsistenzerhaltung (engl. *Consistency*) Der *Erhalt der Konsistenz* einer Datenbasis, also die Wahrung der (impliziten und expliziten) Integritätsbedingungen ist eine Eigenschaft, die ein DBMS nicht automatisch und auf sich selbst gestellt gewährleisten kann. Vielmehr garantieren die anderen Eigenschaften, dass eine Datenbasis von einem konsistenten Zustand wieder in einen anderen konsistenten Zustand überführt wird, solange die ausgeführte Transaktion die Konsistenzbedingungen wahrt. Während der Ausführung kann die Transaktion inkonsistente Zwischenzustände erzeugen, die aber nach außen nicht sichtbar werden.

Dass die Konsistenzbedingungen am Ende einer Transaktion eingehalten werden, fällt in den Verantwortungsbereich des Anwendungsentwicklers. Das DBMS kann den Entwickler bei der Erhaltung der Konsistenz aber dahingehend unterstützen, dass die Integritätsbedingungen in der Datenbank explizit gemacht und vor einem erfolgreichen Abschluss überprüft werden. Schlägt die Überprüfung fehl, kann das DBMS den Abbruch der Transaktion einleiten. Dies ist jedoch nicht direkt Gegenstand der Transaktionsverwaltung.

Isolation (engl. *Isolation*) Transaktionen sind gegeneinander *isoliert*, d.h. jede Transaktion läuft so ab, als ob sie die einzige wäre und keine nebenläufigen Transak-

tionen vorhanden wären. Dies bedeutet insbesondere, dass kein Zwischenzustand einer Transaktion von einer anderen „gelesen“ werden kann, und dass Anwendungen nur konsistente Datenbasiszustände zu Gesicht bekommen können.

Eine hinreichende Bedingung für Isolation ist die *Serialisierbarkeit*, d.h. die Möglichkeit zu einer Ausführung der Transaktionen eine – bezüglich eines noch zu bestimmenden Kriteriums – äquivalente überlappungsfreie Aneinanderreihung der einzelnen Transaktionen zu finden. Eine weiter führende Diskussion des Serialisierbarkeitsbegriff folgt in Abschnitt C.4.

Dauerhaftigkeit (engl. *Durability*) Dauerhaftigkeit (auch „Persistenz“) beschreibt die Transaktionseigenschaft, dass das Ergebnis einer einmal erfolgreich abgeschlossenen Transaktion nicht mehr verloren geht, unabhängig davon, ob im Nachhinein beliebig ernsthafte Fehler auftreten. Die einzige Möglichkeit, ein festgeschriebenes Ergebnis in der Datenbasis zu ändern, besteht in der erfolgreichen Durchführung einer anderen Transaktion.

Die ACID-Eigenschaften werden von DBMS *üblicherweise* für sog. *flache Transaktionen* garantiert, d.h. für nicht weiter strukturierte Transaktionen, wie sie in vielen Anwendungen Einsatz finden. Für manche Anwendungsgebiete gibt es aber spezielle Anforderungen, die den Einsatz von ACID-Transaktionen unpraktikabel werden lässt. Als Beispiel sei die schwierig aufrecht zu erhaltende Isolation bei langlaufenden Transaktionen genannt. Hier finden spezialisierte Transaktionsmodelle Anwendung, von denen einige ausgewählte weiter unten beschrieben werden.

C.3 Entwicklung von Transaktionsmodellen

Obwohl in der Literatur schon eine reichhaltige Auswahl an gut untersuchten Transaktionsmodellen beschrieben wird, bedürfen besondere Anforderungen an die Nebenläufigkeitskontrolle, die Wiederherstellung, die Effizienz oder die sonstige Systemumgebung der Neuentwicklung von Transaktionsmodellen. Eine vereinheitlichte Vorgehensweise für den Entwicklungsprozess stellt Vossen (1995) vor. In (Weikum und Vossen 2002, Seite 42ff) wird die Vorgehensweise weiter verfeinert:

Datenobjekte und Elementaroperationen Zunächst werden die Datenobjekte und ihre zugehörigen Elementaroperationen definiert. Aus Sicht der Transaktionstechnik sind diese Operationen einzeln betrachtet unteilbar, d.h. sie sind ohne weiteres Zutun atomar und isoliert gegenüber anderen Operationen.

Transaktionen Ausgehend von den Elementaroperationen lässt sich ein Transaktionsbegriff definieren, indem die Ausführung von Transaktionsprogrammen als Folge (oder partielle Ordnung, s.u.) von Elementaroperationen interpretiert wird, mit der impliziten Annahme, dass die Transaktionen als zu schützende Einheit betrachtet werden.

Schedules Die nebenläufige Ausführung mehrerer Transaktionen führt zu einer vermischten Folge von Elementaroperationen aus den beteiligten Transaktionen, den sog. Schedules (oder Historien). In diesem Schritt wird definiert, welche Schedules (bzw. Historien) *syntaktisch* als zulässig betrachtet werden.

Korrektheitsbegriff Ausgehend von der Grundmenge der syntaktisch zulässigen Schedules ist die Menge der – im Sinne der ACID-Eigenschaften – *inhaltlich „korrekten“* Schedules zu definieren. Die Definition des Korrektheitsbegriffs legt zum einen fest, welche Wechselwirkungen zwischen nebenläufigen Transaktionen zulässig sind und welche nicht (Synchronisationskorrektheit) und zum anderen, welche Garantien eine korrekte Wiederherstellung im Fehlerfall liefern muss.

Algorithmen und Protokolle Für die vollständige Beschreibung des Transaktionsmodells, die aus den obigen vier Schritten resultiert, bedarf es zum Einsatz noch entsprechender Algorithmen und Protokolle, die die kontinuierlich eintreffenden Operationen aus den Anwendungsprogrammen dynamisch so anwenden, dass ausschließlich korrekte Schedules entstehen und gegen die Datenbasis ausgeführt werden.

Die fünf beschriebenen Schritte werden (implizit oder explizit) bei der Entwicklung praktisch jedes Transaktionsmodells auf verschieden abstraktem und formalem Niveau durchlaufen. Mit einer formalen Spezifikation aus den ersten vier Schritten lässt sich teilweise sogar die Korrektheit der Algorithmen aus Schritt fünf beweisen oder wenigstens untermauern. Das wohl besterforschte und bewährteste Modell, für das eine weitreichende

Formalisierung vorliegt, ist das Seitenmodell, das als (konzeptionelle) Basis vieler Transaktionsmodelle dient und im folgenden Abschnitt vorgestellt wird.

C.4 Ein exemplarisches Transaktionsmodell

Das *Seitenmodell* (auch bekannt als *Read/Write-Modell*, siehe z.B. Weikum und Vossen 2002) beruht auf der Beobachtung, dass praktisch jede Anwendungsoperation, die Daten involviert, auf einer technischen Ebene auf Lese- und Schreibzugriffe auf Seiten eines Speichermediums abgebildet wird.

Datenobjekte und Elementaroperationen Folglich werden als *Datenobjekte* die Elemente einer Menge von Speicherseiten verwaltet, auf denen die unteilbaren *Operationen* lesen r (von *read*) und schreiben w (von *write*) definiert sind. Die Operationen zur Begrenzung der Transaktionen werden bezeichnet mit b (für *begin*), c (für *commit*) und a (für *abort*).

Transaktionen Eine Transaktion im Seitenmodell ist definiert als eine partiell geordnete Abfolge von Lese- und Schreiboperationen auf die Seiten, in der keine Operation mehrfach vorgenommen werden darf und in der kein Datenobjekt (erneut) gelesen werden darf, nachdem es in der Transaktion einmal geschrieben wurde.

Schedules *Schedules* und *Historien* bilden den Übergang von einer einzelnen Transaktion zu einer nebenläufig ausgeführten Menge von Transaktionen. Eine Historie umfasst alle Operationen der beteiligten Transaktionen. Die Ordnungen innerhalb der Transaktionen bleiben erhalten und werden um eine Ordnung der unverträglichen Operationen ergänzt. Zwei Operationen werden als unverträglich bezeichnet, wenn beide auf die selbe Seite zugreifen und mindestens eine der Operationen ein Schreibzugriff ist. Ein Schedule ist ein Präfix einer Historie, der keine aktiven Transaktionen enthält.

Korrektheitsbegriff Der Korrektheitsbegriff bezüglich der Nebenläufigkeit für das Seitenmodell – und auch für andere Modelle – ist eng mit dem Serialisierbarkeitsbegriff verbunden. Eine Historie H heißt serialisierbar, falls es eine serielle Historie gibt,

die nach einem noch zu definierenden Kriterium äquivalent zur Menge der abgeschlossenen Transaktionen in H ist. In einer seriellen Historie sind die Operationen aus den Transaktionen unverzahnt, d.h. die Transaktionen laufen nacheinander ab. Unter den verschiedenen existierenden Äquivalenzbegriffen ist besonders die Konflikt-Äquivalenz von praktischer Bedeutung, da für sie Verfahren existieren, die in polynomialer Zeit ermitteln können, ob eine gegebene Historie Konfliktserialisierbar ist. Man bezeichnet zwei Historien als Konflikt-äquivalent, wenn sie die gleichen Operationen umfassen und in ihnen die unverträglichen Operationen aus erfolgreich abgeschlossenen Transaktionen in der gleichen Reihenfolge auftreten.

Analog zur Serialisierbarkeit als Korrektheitskriterium für die Nebenläufigkeit lässt sich auch die Wiederherstellbarkeit (engl. *Recoverability*) als Korrektheitskriterium für den Fehlerfall über Eigenschaften der Historien definieren. Im Mittelpunkt steht dabei die Vermeidung einer Materialisierung von sog. *"dirty-reads"*. Eine Transaktion kann erst dann ihre Änderungen selbst festschreiben, wenn alle Transaktionen, von denen gelesen wurde, ebenfalls festgeschrieben sind. Darüber hinaus wäre wünschenswert, dass der Abbruch einer Transaktion nicht den Abbruch weiterer Transaktionen nach sich zieht. Auch diese Eigenschaft lässt sich über die Historien definieren. Eine Historie H vermeidet kaskadierendes Rücksetzen (engl. *avoids cascading aborts, ACA*), wenn für alle Transaktionen, die ein Datenobjekt lesen, das zuvor von einer anderen Transaktion geschrieben wurde, gilt, dass die schreibende Transaktion vor der entsprechenden Leseoperation abgeschlossen wurde.

Die Definition der ACA-Historien ist eine echte Verschärfung der Definition der Wiederherstellbarkeit. Theoretisch genügen die aufgestellten Forderungen nun, um trotz Nebenläufigkeit und Fehlern semantische Korrektheit zu erreichen. Aus praktischen Gründen ist es dennoch sinnvoll, die Klasse ACA weiter einzuschränken. Hintergrund der Einschränkung ist, dass ACA-Historien Situationen zulassen, in denen durch gegenseitiges Überschreiben auf vergleichsweise alte Werte eines Datenobjekts zurückgesetzt werden muss, d.h. insbesondere nicht auf den Wert, der vor der abgebrochenen Transaktion vorlag. Dies führt zur Definition der Striktheit, bei der eine Transaktion ein Datenobjekt erst schreiben oder lesen darf, nachdem die Transaktion, die es zuvor geschrieben hat, abgebrochen oder erfolgreich been-

det wurde. In der Praxis werden meist Konflikt-serialisierbare strikte Historien als korrekt erachtet.

Algorithmen und Protokolle Mit der Definition der Konflikt-Serialisierbarkeit und der Striktheit liegt eine formale Spezifikation für die Algorithmen und Protokolle vor, die ein entsprechendes Transaktionsverwaltungssystem implementieren muss. Die Architektur eines Transaktionsverwaltungssystems enthält u.a. die Komponenten *Scheduler* und *Recovery Manager*. Der Scheduler sortiert die Operationen, die aus den nebenläufigen Anwendungsprogrammen eintreffen, gemäß eines Protokolls mit dem Ziel, einen korrekten Schedule zu erzeugen. Die Operationen des korrekten Schedules werden dann sequenziell dem Recovery Manager übergeben, der eine Protokolldatei der übergebenen Schreiboperationen führt und für das Abbrechen bzw. Festschreiben der Transaktionen und das Wiederanlaufen im Fehlerfall verantwortlich ist.

Ein Scheduling-Protokoll, das den oben definierten Korrektheitsbegriff durchsetzt, ist das sog. *strikte Zwei-Phasen-Sperrprotokoll (S2PL)*. Es gehorcht den folgenden Regeln:

- Eine Transaktion muss vor der Ausführung einer Leseoperation auf ein Datenobjekt eine sog. Shared-Sperre anfordern bzw. für jede Schreiboperation eine sog. Exclusive-Sperre erwerben, sofern sie diese nicht bereits besitzt.
- Die erworbenen Sperren werden erst beim Abbruch oder beim Festschreiben der Transaktion wieder freigegeben.
- Datenobjekte dürfen nicht mit unverträglichen Sperren belegt werden. Eine Transaktion, die eine benötigte Sperre nicht erwerben kann, muss ggf. warten.

Hauptaufgabe des Recovery Managers ist die Wiederherstellung des letzten festgeschriebenen Wertes für alle Datenobjekte im Falle eines Systemausfalls oder eines Transaktionsabbruchs. Dies lässt sich erreichen durch das Führen einer Protokolldatei („Log“), die alle Schreibzugriffe und Transaktionsbegrenzungsoperationen enthält. Rücksetzen (engl. *backward recovery*) der Datenbank erfolgt durch das Entfernen aller bereits durchgeführten Änderungen, für die kein Commit im Log vorhanden ist. Ggf. können nach

einem Systemausfall die aus dem Cache noch nicht in den persistenten Speicher übertragenen Änderungen an den Datenobjekten verloren gehen. Das Restaurieren (engl. *forward recovery*) der verlorenen Datenobjekte erfolgt durch Wiederholung der im Log als festgeschrieben gekennzeichneten Transaktionen.

C.5 Fortgeschrittene Transaktionsmodelle

Das oben vorgestellte Seitenmodell macht nahezu keine Annahmen über die Anwendungsprogramme. Es ist mechanistisch und vielseitig anwendbar, passt aber nicht optimal auf alle Anwendungsfälle, wie z.B. auf langlaufende Transaktionen. Je mehr jedoch über die Art der Anwendungsprogramme bekannt ist und einschränkend angenommen werden kann, desto spezialisiertere und für den konkreten Anwendungsfall bessere Transaktionsmodelle lassen sich entwickeln. Aus der vorgestellten Vorgehensweise folgt, dass als Stellschrauben das zugrundeliegende Datenobjekt- und Operationsmodell, der Transaktions- und Schedulebegriff und der Korrektheitsbegriff in Frage kommen. Im Ergebnis führt dies zu den sog. fortgeschrittenen Transaktionsmodellen (engl. *Advanced Transaction Models*), denen eine geänderte operationale Abstraktion zugrunde liegt und/oder die eine Aufweichung der ACID-Eigenschaften realisieren.

C.5.1 Geschachtelte Transaktionen

Für Anwendungsprogramme, deren Funktionalität hierarchisch in delegierbare Teilaufgaben zerfällt, bedarf es eines Transaktionsmodells, das diese Hierarchie abbilden kann. Dem tragen die sog. *geschachtelten Transaktionen* (engl. *nested transactions*) (Moss 1981) dadurch Rechnung, dass Transaktionen („Elterntransaktionen“) rekursiv aus Unteraktionen („Kindtransaktionen“) aufgebaut werden können. Die Blattknoten im resultierenden Transaktionsbaum sind flache ACID-Transaktionen. Die Kindtransaktionen einer gemeinsamen Elterntransaktion laufen sequenziell ab. Schlägt eine Kindtransaktion fehl, bleibt es der Elterntransaktion überlassen, wie sie reagiert. Sie kann an einem beliebigen Punkt innerhalb der eigenen Transaktion wieder aufsetzen und damit insbesonde-

re eine Wiederholung der gescheiterten Kindtransaktion anstoßen oder deren Scheitern ignorieren. Tritt innerhalb der Elterntransaktion selbst ein Fehler auf oder entscheidet sie sich nach dem Scheitern einer der Kindtransaktionen selbst zu einem Abbruch, werden alle bereits ausgeführten Kindtransaktionen ebenfalls zurückgesetzt, was einer lokalen Abschwächung der Dauerhaftigkeitseigenschaft gleichkommt.

In der Variante der *geschlossen geschachtelten Transaktionen* (engl. *closed nested transactions*) wird die Wirkung einer abgeschlossenen Kindtransaktion nur für die nachfolgenden Kindtransaktionen derselben Elterntransaktion und die Elterntransaktion selbst sichtbar, nicht aber außerhalb. Eine Elterntransaktion und insbesondere die Anwendungstransaktion wird dann festgeschrieben, wenn alle Kindtransaktionen festgeschrieben werden konnten. Damit lassen sich für die Anwendungstransaktion die ACID-Eigenschaften garantieren.

In den sog. *offen geschachtelten Transaktionen* (engl. *open nested transactions*) verzichtet man auf die Isolation der Untertransaktionen und macht damit Zwischenergebnisse der Anwendungstransaktion für andere Transaktionen sichtbar. Im Falle des Abbruchs der Anwendungstransaktion ist ein einfaches Rücksetzen damit nicht mehr möglich, sondern es muss eine (semantische) Kompensation erfolgen (Korth u. a. 1990).

C.5.2 Sagas

Sagas, die einen Spezialfall der offen geschachtelten Transaktionen darstellen, eignen sich besonders für langlaufende sequenziell gliederbare Transaktionen (Garcia-Molina und Salem 1987). Sie haben ein vergleichsweise starres Transaktionsmodell, das die Kompensation explizit mit einbezieht. Eine Saga wird aufgebaut aus zwei Folgen flacher ACID-Untertransaktionen. Die erste Menge $\{t_1, \ldots, t_n\}$ definiert das erwünschte Verhalten im Erfolgsfall (durch Aneinanderreihung der Untertransaktionen in Reihenfolge aufsteigender Indizes). Die Isolation wird aufgegeben, so dass andere Transaktionen die Zwischenergebnisse teilweise ausgeführter Sagas sehen können. Die zweite Menge $\{ct_1, \ldots, ct_n\}$ definiert das zugehörige Kompensationsverhalten: jedes ct_i macht semantisch die Effekte von t_i rückgängig. Eine Saga wird erfolgreich festgeschrieben, wenn

alle ihre Untertransaktionen erfolgreich festschreiben konnten, womit als ausgeführte Untertransaktionsfolge $(t_1, \ldots, t_n)$ resultiert und sie wird abgebrochen, wenn eine Untertransaktion t_k fehlschlägt. In diesem Fall werden die Kompensationsuntertransaktionen der bereits festgeschriebenen Untertransaktionen in umgekehrter Reihenfolge ausgeführt, was zur insgesamt ausgeführten Untertransaktionsfolge $(t_1, \ldots, t_{k-1}, ct_{k-1}, \ldots, ct_1)$ führt.

C.5.3 Das ConTract-Modell

Auch das ConTract-Modell (Wächter und Reuter 1990, 1992) widmet sich den speziellen Anforderungen langlaufender Anwedungsprogramme. Solche Programme weisen oft komplexere Strukturen auf, als den von den Sagas unterstützten sequenziellen Aufbau.

> A ConTract is a consistent and fault tolerant execution of an arbitrary sequence of predefined actions (called *steps*) according to an explicitly specified control flow description (called *script*).

Die Kontrollflussspezifikation ist das Herzstück eines Programmiermodells und eines entsprechenden Kontrollmechanismus (sog. ConTract Manager) auf einer eigenen Schicht über den zugrundeliegenden ACID-Transaktionen. Damit sind ConTracts kein Transaktionsmodell im traditionellen Sinn, sondern ein Programmiermodell mit expliziter Einbeziehung von Dauerhaftigkeit, Konsistenz, Wiederherstellung, Synchronisation und Kooperation.

C.6 Verteilte Transaktionen

Wenn Dienstnehmer und -geber nicht zentral sondern verteilt über mehrere Rechnerknoten vorliegen, bedarf es zusätzlicher Maßnahmen, um auf globaler Ebene transaktionale Eigenschaften garantieren zu können. Bei autonomen Knoten kann nicht unbedingt erwartet werden, dass sie sich einem globalen Scheduler unterordnen und ihren Fortschritt von ihm abhängig machen. Damit wird die Isolationsforderung auf globaler

Ebene i.d.R. aufgegeben und Atomizität wird zum Hauptziel der verteilten Transaktionsverwaltung. Drei zusätzliche Aufgaben sind dafür im verteilten (aber unreplizierten) Fall vom Transaktionsverwaltungssystem zu lösen:

1. Vergabe und Zuordnung global eindeutiger Transaktionskennungen
2. Globale Koordination der commit-Zeitpunkte
3. Durchsetzung des einheitlichen Transaktionsausgangs (`commit` bzw. `abort`)

Diese wird kooperativ von den Komponenten *Koordinator* (technisch auch „Transaktionsverwalter“) und *Teilnehmer* (auch „Ressourcenverwalter“) realisiert. Während der Koordinator für die globalen Aufgaben zuständig ist, müssen die Teilnehmer die lokalen ACID-Garantien gewährleisten und kooperativ mit dem Koordinator den Ausgang der verteilten Transaktionen bestimmen.

Zu Beginn einer verteilten Transaktion erzeugt der Koordinator einen global eindeutigen Transaktionsidentifikator (TID), der bei jeder Operation an die beteiligten Ressourcenverwalter übergeben wird. Damit ist die Zuordnung zwischen globaler Transaktion und lokalen Veränderungen gewährleistet.

Ohne eine globale Koordination der commit-Zeitpunkte sind Situationen denkbar, in denen zwar lokal betrachtet alle Transaktionen serialisierbar sind, die dazugehörigen globalen Transaktionen jedoch nicht. Außerdem darf kein lokaler Teil der verteilten Transaktion festschreiben, solange noch ein lokaler Teil der Transaktion auf einem anderen Knoten scheitern könnte. Ein Protokoll, das häufig zur erforderlichen Koordination der lokalen Transaktionen eingesetzt wird, ist das Zwei-Phasen-Commit-Protokoll.

C.6.1 Zwei-Phasen-Commit-Protokoll

Beim *Zwei-Phasen-Commit-Protokoll* (engl. *two-phase-commit-protocol, 2PC*) erfolgt die Koordination in den beiden Phasen „Wahlphase“ und „Entscheidungsphase“. In der Wahlphase benachrichtigt der Koordinator die Teilnehmer, dass ein globales Festschreiben erfolgen soll (sog. `PREPARE`-Nachricht) und erwartet eine Rückmeldung, ob

die jeweilige lokale Transaktion festgeschrieben werden kann oder abgebrochen werden muss.

Die Teilnehmer stellen fest, ob ein lokales Festschreiben möglich wäre und geben dem Koordinator bei möglichem Erfolg eine positive Rückmeldung (`VOTE-COMMIT`). Falls die lokale Transaktion nicht erfolgreich zu Ende gebracht werden kann, gibt der Teilnehmer einen negativen Bescheid an den Koordinator (`VOTE-ABORT`).

In der *Entscheidungsphase* sammelt der Koordinator zunächst alle Antworten der Teilnehmer. Wenn ein Festschreiben auf allen beteiligten Knoten möglich ist, dann steht auch der erfolgreichen Beendigung der globalen Transaktion nichts mehr im Wege: der Koordinator versendet `GLOBAL-COMMIT` an die Teilnehmer. Falls mindestens eine lokale Transaktion fehlschlägt, wird `GLOBAL-ABORT` verschickt. Die Teilnehmer, die mit `VOTE-COMMIT` geantwortet haben, reagieren auf die globale Entscheidung und schreiben ihre lokalen Transaktionen fest bzw. setzen sie zurück. Der Erhalt der globalen Entscheidung wird dem Koordinator quittiert.

Das 2PC-Protokoll hat den Nachteil, dass verlorene Nachrichten in bestimmten Fällen zum Blockieren einzelner Teilnehmer führen können. Dies gilt insbesondere für die sog. Unsicherheitsphase: ein Teilnehmer hat mit `VOTE-COMMIT` abgestimmt, die Nachricht mit der globalen Entscheidung an ihn geht aber verloren. Die lokale Transaktion kann nun weder festgeschrieben noch abgebrochen werden, da das globale Ergebnis nicht bestimmbar ist und ein beliebiges lokales Verhalten zur globalen Entscheidung inkonsistent sein könnte. Im Falle eines `VOTE-ABORT` könnte die lokale Transaktion sofort abgebrochen werden, da durch das eigene Abstimmungsverhalten auch das `GLOBAL-ABORT` als Entscheidung für die globale Transaktion vorausgesagt werden kann.

Einige problematische Situationen im 2PC-Protokoll, wie z.B. der Ausfall einzelner Knoten, lassen sich durch sog. Terminierungsprotokolle und Time out-Mechanismen zwar beheben, trotzdem kann keine erfolgreiche Fehlerbeseitigung garantiert werden und Blockaden sind möglich.

C.6.2 Transaktionale Nachrichtenwarteschlangen

Einer der Hauptnachteile der verteilten Transaktionen nach dem Zwei-Phasen-Commit-Protokoll ist die enge Kopplung ihrer Teilnehmer über den Koordinator, die deren Autonomie einschränkt. Eine losere Kopplung führt aber andererseits zu einer Aufweichung der global gewährbaren Garantien. Wenn die Applikation die genannte Aufweichung zulässt, dann lassen sich in Systemen mit asynchronem Nachrichtenaustausch sog. *transaktionale Nachrichtenwarteschlangen* (engl. *queued transactions*) einsetzen (siehe Weikum und Vossen 2002, Seite 625ff.).

Wie Abbildung C.1 zeigt, besteht eine derart geschützte Nachrichtenwarteschlange aus drei völlig entkoppelten Transaktionen. Die erste Transaktion läuft lokal auf Senderseite (oder in der Taxonomie der Warteschlangen beim „Erzeuger“) und umfasst dessen lokales Verhalten und das zugehörige Einreihen der zu sendenden Nachricht in die Warteschlange. Entsprechendes gilt auf der Empfänger-Seite (beim „Verbraucher“). Die dritte Transaktion betrifft die Nachrichtenwarteschlange selbst und verhilft ihr zur Dauerhaftigkeit und Wiederherstellbarkeit, indem sie insbesondere gegen Nachrichtenverluste geschützt wird.

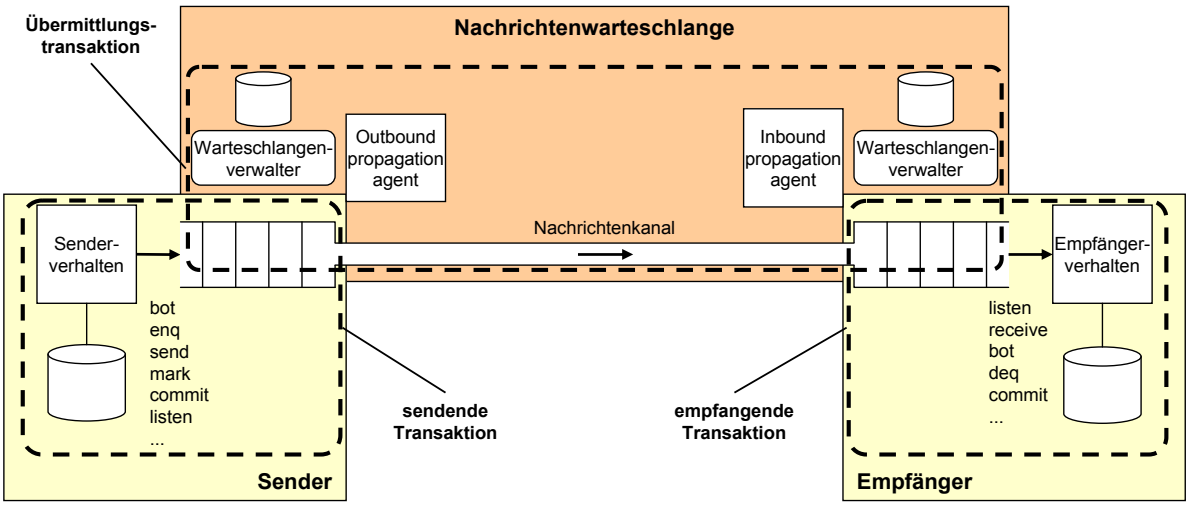


Abbildung C.1: Transaktionale Nachrichtenwarteschlangen (nach Lockemann und Dittrich 2004)

Literaturverzeichnis

Abeck u. a. 2003 Abeck, Sebastian ; Lockemann, Peter C. ; Schiller, Jochen ; Seitz, Jochen: *Verteilte Informationssysteme – Integration von Datenübertragungstechnik und Datenbanktechnik*. dpunkt.verlag, 2003

Aknine u. a. 2004 Aknine, Samir ; Pinson, Suzanne ; Shakun, Melvin F.: An Extended Multi-Agent Negotiation Protocol. In: *Autonomous Agents and Multi-Agent Systems* 8 (2004), Nr. 1, S. 5–45

Alchourrón u. a. 1985 Alchourrón, Carlos E. ; Gärdenfors, Peter ; Makinson, David: On the Logic of Theory Change. In: *Journal of Symbolic Logic* 50 (1985), Nr. 2, S. 510–530

Anderson und Lee 1981 Anderson, Thomas ; Lee, Peter A.: *Fault Tolerance: Principles and Practice*. Prentice/Hall International, 1981

Bartlett 1978 Bartlett, Joel F.: A 'Non-stop' Operating System. In: *Proceedings of the 11th Hawaii International Conference on System Sciences (11th HICSS'78)* Bd. 3, IEEE Computer Society Press, 1978, S. 103–117

Bauer u. a. 2001 Bauer, Bernhard ; Müller, Jörg P. ; Odell, James: Agent UML : A Formalism for Specifying Multiagent Interaction. In: *Agent-Oriented Software Engineering (AOSE): First International Workshop*. Limerick, Ireland : Springer Verlag, 2001, S. 91–103

BBN Technologies 2004a BBN Technologies: Cougaar Architecture Document / BBN Technologies. 2004. – Systemdokumentation

BBN Technologies 2004b BBN Technologies: Cougaar Developers' Guide / BBN Technologies. 2004. – Systemdokumentation

Belecheanu u. a. 2006 Belecheanu, Roxana A. ; Munroe, Steve ; Luck, Michael ; Payne, Terry: Commercial Applications of Agents: Lessons, Experiences and Challenges. In: Weiss, Gerhard (Hrsg.) ; Stone, Peter (Hrsg.): *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '06) – Industry Track*, ACM Press, 2006, S. 1549–1555

Bellifemine u. a. 2005 Bellifemine, Fabio ; Bergenti, Federico ; Caire, Giovanni ; Poggi, Agostino: JADE – A Java Agent Development Framework. In: Bordini, Rafael H. (Hrsg.) ; Dastani, Mehdi (Hrsg.) ; Dix, Jürgen (Hrsg.) ; El Fallah Seghrouchni, Amal (Hrsg.): *Mutli-Agent Programming*. Kluwer, 2005, S. 125–148

Bellifemine u. a. 2007 Bellifemine, Fabio ; Caire, Giovanni ; Greenwood, Dominic: *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, 2007

Beneken 2006 Beneken, Gerd: Referenzarchitekturen. In: Reussner, Ralf H. (Hrsg.) ; Hasselbring, Wilhelm (Hrsg.): *Handbuch der Software-Architektur*. dpunkt.verlag, 2006, S. 357–370

Bernstein und Newcomer 1997 Bernstein, Philip A. ; Newcomer, Eric: *Principles of Transaction Processing*. Morgan Kaufmann Publishers, 1997

Bonabeau u. a. 1999 Bonabeau, Eric ; Dorigo, Marco ; Theraulaz, Guy: *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999

Bratman u. a. 1988 Bratman, Michael ; Israel, David J. ; Pollack, Martha E.: Plans and Resource-Bounded Practical Reasoning / SRI International. 1988 (AI Center Technical Note SRI-AI 425R). – Forschungsbericht

Bratman 1987 Bratman, Michael E.: *Intentions, Plans, and Practical Reason*. Harvard University Press, 1987

Brennan und O 2004 Brennan, Robert W. ; O, William: Performance Analysis of a Multi-agent Scheduling and Control System under Manufacturing Disturbances. In: *Production Planning and Control* 15 (2004), Nr. 2, S. 225–235

Brooks 1985 Brooks, Rodney A.: A Robust Layered Control System for a Mobile Robot / MIT AI-Lab. 1985 (A.I. Memo 864). – Technischer Bericht

Brooks 1986 Brooks, Rodney A.: A Robust Layered Control System for a Mobile Robot. In: *IEEE Journal of Robotics and Automation* 2 (1986), Nr. 1, S. 14–23

Buschmann u. a. 1996 Buschmann, Frank ; Meunier, Regine ; Rohnert, Hans ; Sommerlad, Peter ; Stal, Michael: *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, 1996

Busetta 1999 Busetta, Paolo: *A Transaction Based Multi-Agent Architecture*, University of Melbourne, Department of Computer Science, Dissertation, 1999

Busetta u. a. 2003 Busetta, Paolo ; Bailey, James ; Ramamohanarao, Kotagiri: A Reliable Computational Model for BDI Agents. In: *Proceedings of the First International Workshop on Safety and Security in Multi Agent Systems*, 2003

Busetta und Ramamohanarao 1998 Busetta, Paolo ; Ramamohanarao, Kotagiri: An Architecture for Mobile BDI Agents. In: *Proceedings of the 1998 ACM Symposium on Applied Computing*, 1998, S. 445–452

Bussmann u. a. 2004 Bussmann, Stefan ; Jennings, Nicolas R. ; Wooldridge, Michael: *Multiagent Systems for Manufacturing Control – A Design Methodology*. Springer Verlag, 2004

Bussmann und Schild 2000 Bussmann, Stefan ; Schild, Klaus: Self-Organizing Manufacturing Control: And Industrial Application of Agent Technology. In: *Proceedings of the 4th International Conference on Multi-agent Systems (ICMAS'2000)*, 2000, S. 87–94

Calisti 2002 Calisti, Monique: Abstracting Communication in Distributed Agent-Based Systems. In: *ECOOP2002 Workshop on Concrete Communication Abstractions of the Next 701 Distributed Object Systems*, 2002

Cavalieri u. a. 2000 Cavalieri, Sergio ; Macchi, Marco ; Taisch, Marco ; Garetti, Marco: An Experimental Benchmarking of Two Multiagent Systems for Production Scheduling and Control. In: *Computers in Industry* 43 (2000), S. 139–152

Cetnarowicz und Kozlak 2002 Cetnarowicz, Krzysztof ; Kozlak, Jaroslaw: Multi-agent System for Flexible Manufacturing Systems Management. In: Dunin-Kplicz, Barbara (Hrsg.) ; Nawarecki, Edward (Hrsg.): *From Theory to Practice in Multi-Agent Systems: Second International Workshop of Central and Eastern Europe on Multi-Agent Systems (CEEMAS 2001)*, Springer Verlag, 2002, S. 73–82

Cheung 1980 Cheung, Roger C.: A User-Oriented Software Reliability Model. In: *IEEE Transactions on Software Engineering* 6 (1980), Nr. 2, S. 118–125

Curry 2003 Curry, Edward: How to use the Java Messaging Service (JMS) MTP with JADE / ECRG, NUI, Galway, Irland. 2003. – Systemdokumentation

Curry u. a. 2003 Curry, Edward ; Chambers, Desmond ; Lyons, Gerard: A JMS Message Transport Protocol for the JADE Platform. In: Zhong, Ning (Hrsg.): *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology (IAT2003)*, IEEE Computer Society Press, 2003, S. 596–600

Dang und Frankovic 2002 Dang, T.-Tung ; Frankovic, Baltazar: Agent based Scheduling in Production Systems. In: *International Journal of Production Research* 40 (2002), Nr. 15, S. 3669–3679

Davidsson und Johansson 2005a Davidsson, Paul ; Johansson, Stefan J.: On the Metaphysics of Agents. In: *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '05)*, ACM Press, 2005, S. 1299–1300

Davidsson und Johansson 2005b Davidsson, Paul ; Johansson, Stefan J.: On the Metaphysics of Agents (Extended Version) / Blekinge Tekniska Högskola. 2005. – Forschungsbericht

Dellarocas und Klein 2000 Dellarocas, Chrysanthos ; Klein, Mark: An Experimental Evaluation of Domain-Independent Fault Handling Services in Open Multi-Agent Systems. In: *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-2000)*, 2000, S. 95–102

Dellarocas u. a. 2000 Dellarocas, Chrysanthos ; Klein, Mark ; Rodriguez-Aguilar, Juan A.: An Exception-Handling Architecture for Open Electronic Marketplaces of Contract Net Software Agents. In: *Proceedings of the 2nd ACM Conference on Electronic Commerce*, ACM Press, 2000, S. 225–232

Dimov und Punnekkat 2005 Dimov, Aleksandar ; Punnekkat, Sasikumar: On the Estimation of Software Reliability of Component-Based Dependable Distributed Systems. In: Reussner, Ralf (Hrsg.) ; Mayer, Johannes (Hrsg.) ; Stafford, Judith A. (Hrsg.) ; Overhage, Sven (Hrsg.) ; Becker, Steffen (Hrsg.) ; Schroeder, Patrick J. (Hrsg.): *Proceedings of the First International Conference on the Quality of Software Architectures (QoSA '05) and of the Second International Workshop on Software Quality (SOQUA '05)*, Springer Verlag, 2005, S. 171–187

Dinkloh und Nimis 2004 Dinkloh, Martin ; Nimis, Jens: A Tool for Integrated Design and Implementation of Conversations in Multiagent Systems. In: Dastani, Mehdi (Hrsg.) ; Dix, Jürgen (Hrsg.) ; Fallah-Seghrouchni, Amal E. (Hrsg.): *Proceedings of the 1st International Workshop on Programming Multiagent Systems: Languages, Frameworks, Techniques and Tools (ProMAS 2003)*, Springer Verlag, 2004, S. 187–200

Dorer und Calisti 2005 Dorer, Klaus ; Calisti, Monique: An Adaptive Solution to Dynamic Transportation Optimization. In: *AAMAS '05: Proceedings of the 4th international Joint Conference on Autonomous Agents and Multiagent Systems*, ACM Press, 2005, S. 45–51

Eckerson 1995 Eckerson, Wayne W.: Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications. In: *Open Information Systems* 10 (1995), Nr. 1

Engelmore und Morgan 1988 Engelmore, Robert S. ; Morgan, Anthony J.: *Blackboard Systems*. Addison-Wesley, 1988

Esterline 2002 Esterline, Albert C.: Using Statecharts and Modal Logics to Model Multiagent Plans and Transactions. In: Hinchey, Michael G. (Hrsg.) ; Rash, James L. (Hrsg.) ; Truszkowski, Walt (Hrsg.) ; Rouff, Christopher (Hrsg.) ; Gordon-Spears, Diana F. (Hrsg.): *Proceedings of the Second*

International Workshop on Formal Approaches to Agent-Based Systems (FAABS '02), Springer Verlag, 2002, S. 146–161

Eusgeld u. a. 2006 Eusgeld, Irene ; Freiling, Felix ; Reussner, Ralf: *Dependability Metrics*. Springer Verlag, 2006

Fedoruk und Deters 2002 Fedoruk, Alan ; Deters, Ralph: Using Agent Replication to Enhance Reliability and Availability of Multi-agent Systems. In: *Proceedings of the 15th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, Springer Verlag, 2002, S. 237–251

FIPA 2001a FIPA: Agent Message Transport Protocol for WAP Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2001. – Spezifikation

FIPA 2001b FIPA: Agent Software Integration Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2001. – Spezifikation

FIPA 2001c FIPA: CCL Content Language Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2001. – Spezifikation

FIPA 2001d FIPA: Dutch Auction Interaction Protocol Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2001. – Spezifikation

FIPA 2001e FIPA: English Auction Interaction Protocol / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2001. – Spezifikation

FIPA 2001f FIPA: Ontology Service Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2001. – Spezifikation

FIPA 2001g FIPA: RDF Content Language Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2001. – Spezifikation

FIPA 2002a FIPA: Abstract Architecture Specification. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2002. – Spezifikation

FIPA 2002b FIPA: ACL Message Representation in Bit-Efficient Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2002. – Spezifikation

FIPA 2002c FIPA: ACL Message Representation in String Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2002. – Spezifikation

FIPA 2002d FIPA: ACL Message Representation in XML Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2002. – Spezifikation

FIPA 2002e FIPA: ACL Message Structure Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2002. – Spezifikation

FIPA 2002f FIPA: Agent Message Transport Envelope Representation in Bit Efficient Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2002. – Spezifikation

FIPA 2002g FIPA: Agent Message Transport Envelope Representation in XML Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2002. – Spezifikation

FIPA 2002h FIPA: Agent Message Transport Protocol for HTTP Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2002. – Spezifikation

FIPA 2002i FIPA: Agent Message Transport Protocol for IIOP Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2002. – Spezifikation

FIPA 2002j FIPA: Communicative Act Library Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2002. – Spezifikation

FIPA 2002k FIPA: Contract Net Interaction Protocol Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2002. – Spezifikation

FIPA 2002l FIPA: FIPA Agent Management Support for Mobility Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2002. – Spezifikation

FIPA 2002m FIPA: Iterated Contract Net Interaction Protocol Specification / Foundation for Intelligent Physical Agents. Foundation for Intelligent Physical Agents, 2002. – Spezifikation

FIPA 2002n FIPA: SL Content Language Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2002. – Spezifikation

FIPA 2003 FIPA: KIF Content Language Specification / Foundation for Intelligent Physical Agents. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2003. – Spezifikation

FIPA 2004a FIPA: Agent Management Specification. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2004. – Spezifikation

FIPA 2004b FIPA: Agent Message Transport Service Specification. Alameda, CA, USA : Foundation for Intelligent Physical Agents, 2004. – Spezifikation

Fischer u. a. 1995 Fischer, Klaus ; Müller, Jörg P. ; Pischel, Markus ; Schier, Darius: A Model for Cooperative Transportation Scheduling. In: *Proceedings of the First International Conference on Multiagent Systems (ICMAS 1995)*, AAAI Press / MIT Press, 1995, S. 109–116

Flores-Mendez 1999 Flores-Mendez, Roberto A.: Towards a Standardization of Multi-Agent System Frameworks. In: *Crossroads* 5 (1999), Nr. 4, S. 18–24

Foster u. a. 2001 Foster, Ian ; Kesselman, Carl ; Tuecke, Steven: The Anatomy of the Grid: Enabling Scalable Virtual Organization. In: *The International Journal of High Performance Computing Applications* 15 (2001), Nr. 3, S. 200–222

Franklin und Graesser 1996 Franklin, Stan ; Graesser, Art: Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents. In: *Proceedings of the 3rd International Workshop on Agent Theories, Architectures, and Language (ATAL'96)*. Springer Verlag, 1996, S. 21–35

Frey 2004 Frey, Daniel: *Simulationsgestützte Entwicklung eines Kanban-Systems in der Montage*, Universität Karlsruhe (TH), Fakultät für Informatik, Dissertation, 2004

Frey u. a. 2003a Frey, Daniel ; Nimis, Jens ; Wörn, Heinz ; Lockemann, Peter C.: Benchmarking and Robust Multi-agent-based Production Planning and Control. In: *Engineering Applications of Artificial Intelligence - The International Journal of Intelligent Real-Time Automation (EAAI)* 16 (2003), Nr. 4, S. 307–320

Frey u. a. 2003b Frey, Daniel ; Nimis, Jens ; Wörn, Heinz ; Lockemann, Peter C.: Benchmarking and Robust Multi-agent-based Production Planning and Control. In: *Proceedings of the 7th IFAC Workshop on Intelligent Manufacturing Systems (IMS2003)*, Elsevier, 2003, S. 229–234

Galan und Baker 1999 Galan, Alan K. ; Baker, Albert D.: Multi-agent Communications in JAFMAS. In: *Proceedings of the 1999 Workshop on Specifying and Implementing Conversation Policies*, 1999, S. 67–70

Gamma u. a. 1996 Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, 1996

Garcia-Molina 1982 Garcia-Molina, Hector: Elections in a Distributed Computing System. In: *IEEE Transactions on Computers* C-31 (1982), Nr. 1, S. 48–59

Garcia-Molina und Salem 1987 Garcia-Molina, Hector ; Salem, Kenneth: SAGAS. In: Dayal, Umeshwar (Hrsg.) ; Traiger, Irving L. (Hrsg.): *Proceedings of the SIGMod 1987 Annual Conference*, ACM Press, 1987, S. 249–259

Genesereth und Fikes 1992 Genesereth, Michael R. ; Fikes, Richard E.: Knowledge Interchange Format, Version 3.0 Reference Manual / Computer Science Department, Stanford University. 1992 (Logic-92-1). – Forschungsbericht

Genesereth und Ketchpel 1994 Genesereth, Michael R. ; Ketchpel, Steven P.: Software Agents. In: *Communications of the ACM* 37 (1994), Nr. 7, S. 48–53

Giorgini u. a. 2002 Giorgini, Paolo ; Kolp, Manuel ; Mylopoulos, John: Multi-agent and Software Architectures: A Comparative Case Study. In: Giunchiglia, Fausto (Hrsg.) ; Odell, James (Hrsg.) ; Weiss, Gerhard (Hrsg.): *Third International Workshop on Agent-Oriented Software Engineering (AOSE '02)* Bd. 2585, Springer Verlag, 2002, S. 101–112

Glaser u. a. 1992 Glaser, Horst ; Geiger, Werner ; Rohde, Volker: *Produktionsplanung und -steuerung.* 2., überarbeitete Auflage. Gabler, 1992

Goseva-Popstojanova und Trivedi 2001 Goseva-Popstojanova, Katerina ; Trivedi, Kishor S.: Architecture Based Approach to Reliability Assessment of Software Systems. In: *Performance Evaluation* 45 (2001), Nr. 2–3, S. 179–204

Guessoum u. a. 2002 Guessoum, Zahia ; Briot, Jean-Pierre ; Charpentier, Sébastien ; Marin, Olivier ; Sens, Pierre: A Fault-Tolerant Multi-agent Framework. In: Gini, Maria (Hrsg.) ; Ishida, Toru (Hrsg.) ; Castelfranchi, Cristiano (Hrsg.) ; Johnson, W. L. (Hrsg.): *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, ACM Press, 2002, S. 672–673

Halsall 1998 Halsall, Fred: *Data Communications, Computer Networks and Open Systems.* Addison-Wesley, 1998

Hannebauer 2002 Hannebauer, Markus: Modeling and Verifying Agent Interaction Protocols with Algebraic Petri Nets. In: *Proceedings of the 6th*

International Conference on Integrated Design and Process Technology (IDPT-2002), 2002

Hansmann 1994 HANSMANN, Karl-Werner: *Industrielles Management*. Oldenbourg, 1994

Harman 1983 HARMAN, Gilbert H.: *Change in View: Principles of Reasoning*. MIT Press, 1983

Helin und Laukkanen 2002 HELIN, Heikki ; LAUKKANEN, Mikko: Performance Analysis of Software Agent Communication in Slow Wireless Networks. In: *Proceedings of the Eleventh International Conference on Computer Communications and Networks*, 2002, S. 354–361

Helsinger und Wright 2005 HELSINGER, Aaron ; WRIGHT, Todd: Cougaar: A Robust Configurable Multi Agent Platform. In: *Proceedings of the 2005 IEEE Conference on Aerospace (IEEEAC '05)*. Los Alamitos, CA, USA : IEEE Computer Society, 2005, S. 1–10

Hendler 2007 HENDLER, James: Where Are All the Intelligent Agents? In: *IEEE Intelligent Systems* 22 (2007), Nr. 3, S. 2–3

Hägg 1997 HÄGG, Staffan: A Sentinel Approach to Fault Handling in Multi-Agent Systems. In: ZHANG, Chengqi (Hrsg.) ; LUKOSE, Dickson (Hrsg.): *Multi-Agent Systems: Methodologies and applications: Selected Papers of the 2nd Australian Workshop on Distributed Artificial Intelligence (DAI-96)*, Springer Verlag, 1997, S. 181–195

Hoitsch 1993 HOITSCH, Hans-Jörg: *Produktionswirtschaft*. Vahlen, 1993

Horrocks 2002 HORROCKS, Ian: DAML+OIL: a Reason-able Web Ontology Language. In: JENSEN, Christian S. (Hrsg.) ; JEFFERY, Keith G. (Hrsg.) ; POKORNÝ, Jaroslav (Hrsg.) ; SALTENIS, Simonas (Hrsg.) ; BERTINO, Elisa (Hrsg.) ; BÖHM, Klemens (Hrsg.) ; JARKE, Matthias (Hrsg.): *Proceedings of the Advances in Database Technology - EDBT 2002, 8th International Conference on Extending Database Technology*, Springer Verlag, 2002, S. 2–13

Huhns und Stephens 1999 Huhns, Michael N. ; Stephens, Larry M.: Multiagent Systems and Societies of Agents. In: Weiss, Gerhard (Hrsg.): *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1999, S. 79–120

Iren u. a. 1999 Iren, Sami ; Amer, Paul D. ; Conrad, Phillip T.: The Transport Layer: Tutorial and Survey. In: *ACM Computing Surveys* 31 (1999), Nr. 4, S. 360–404

Jennings 2000 Jennings, Nicholas R.: On Agent-based Software Engineering. In: *Artificial Intelligence* 177 (2000), Nr. 2, S. 277–296

Kaminka und Tambe 1998 Kaminka, Gal A. ; Tambe, Milind: What's Wrong With Us? Improving Robustness through Social Diagnosis. In: *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998, S. 97–104

Katsuno und Mendelzon 1992 Katsuno, Hirofumi ; Mendelzon, Alberto O.: On the Difference between Updating a Knowledge Base and Revising it. In: Gärdenfors, Peter (Hrsg.): *Belief Revision*. Cambridge University Press, 1992 (Cambridge Tracts in Theoretical Computer Science 29), S. 183–203

Kendall u. a. 1997 Kendall, Elizabeth A. ; Pathak, Chirag V. ; Krishna, P. V. M. ; Suresh, C. B.: The Layered Agent Pattern Language. In: *Proceedings of the 1997 Conference on Pattern Languages of Programs (PLoP'97)*, 1997

Kirn u. a. 2006a Kirn, Stefan ; Anhalt, Christian ; Krcmar, Helmut ; Schweiger, Andreas: Agent.Enterprise in a Nutshell. In: Kirn, Stefan (Hrsg.) ; Herzog, Otthein (Hrsg.) ; Lockemann, Peter C. (Hrsg.) ; Spaniol, Otto (Hrsg.): *Multiagent Engineering – Theory and Applications in Enterprises*. Springer Verlag, 2006, S. 199–220

Kirn u. a. 2006b Kirn, Stefan (Hrsg.) ; Herzog, Otthein (Hrsg.) ; Lockemann, Peter C. (Hrsg.) ; Spaniol, Otto (Hrsg.): *Multiagent Engineering – Theory and Applications in Enterprises*. Springer Verlag, 2006

Klein 2003 Klein, Mark: A Knowledge-Based Methodology for Designing Reliable Multi-agent Systems. In: *Agent-Oriented Software Engineering IV*, 2003, S. 85–95

Klein und Dellarocas 2000 Klein, Mark ; Dellarocas, Chrysanthos: Domain-Independent Exception Handling Services That Increase Robustness in Open Multi-Agent Systems. 2000. – Forschungsbericht

Klein u. a. 2001 Klein, Mark ; Dellarocas, Chrysanthos ; Rodriguez-Aguilar, Juan A.: A Knowledge-Based Methodology for Designing Robust Electronic Markets / Massachusetts Institute of Technology. 2001. – Forschungsbericht

Knabe u. a. 2002 Knabe, Tore ; Schillo, Michael ; Fischer, Klaus: Improvements to the FIPA Contract Net Protocol for Performance Increase and Cascading Applications. In: Timm, Ingo (Hrsg.): *Proceedings of the 1st International Workshop on Multi-Agent Interoperability (MAI'02)*, 2002

Kolp u. a. 2006 Kolp, Manuel ; Giorgini, Paolo ; Mylopoulos, John: Multi-Agent Architectures as Organizational Structures. In: *Autonomous Agents and Multi-Agent Systems* 13 (2006), Nr. 1, S. 3–25

Korth u. a. 1990 Korth, Henry F. ; Levy, Eliezer ; Silberschatz, Abraham: A Formal Approach to Recovery by Compensating Transactions. In: McLeod, Dennis (Hrsg.) ; Sacks-Davis, Ron (Hrsg.) ; Schek, Hans-Jörg (Hrsg.): *Proceedings of the 16th International Conference on Very Large Data Bases*, 1990, S. 95–106

Krempels 2001 Krempels, Karl-Heimz: *Fehlertoleranz von Mobilen Agenten.* Aachen, Lehrstuhl für Informatik 4, RWTH Aachen, Diplomarbeit, 2001

Krempels u. a. 2003 Krempels, Karl-Heinz ; Nimis, Jens ; Braubach, Lars ; Pokahr, Alexander ; Herrler, Rainer ; Scholz, Thorsten: Entwicklung intelligenter Multi-Multiagentensysteme – Werkzeugunterstützung, Lösungen und offene Fragen. In: Dittrich, Klaus (Hrsg.) ; König, Wolfgang (Hrsg.) ; Oberweis, Andreas (Hrsg.) ; Rannenberg, Kai (Hrsg.) ; Wahlster, Wolfgang (Hrsg.): *Informatik 2003 - Innovative Informatikanwendungen*, 2003, S. 31–46

Kumar und Cohen 2000 Kumar, Sanjeev ; Cohen, Philip: Towards a Fault-Tolerant Multi-Agent System Architecture. In: *Proceedings of The Fourth International Conference on Autonomous Agents (Agents 2000)*, 2000, S. 459–466

Kurbel 2005 Kurbel, Karl: *Produktionsplanung und -steuerung im Enterprise Resource Planning und Supply Chain Management*. Oldenbourg, 2005

Labrou und Finin 1997 Labrou, Yannis ; Finin, Tim: A Proposal for a New KQML Specification / Computer Science and Electrical Engineering Department, University of Maryland Baltimore County. 1997 (TR CS-97-03). – Forschungsbericht

Laprie 1985 Laprie, Jean-Claude: Dependable Computing and Fault Tolerance: Concepts and Terminology. In: *Proceedings of the 15th IEEE Symposium on Fault Tolerant Computing Systems (FTCS-15)*, 1985, S. 2–11

Laprie 1992 Laprie, Jean-Claude (Hrsg.): *Dependable computing and fault tolerant systems*. Bd. 5: *Dependability: Basic Concepts and Terminology : in English, French, German, Italian and Japanese*. Springer Verlag, 1992

Leveson 1986 Leveson, Nancy G.: Software Safety: Why, What, and How. In: *ACM Computing Surveys* 18 (1986), Nr. 2, S. 125–163

Lockemann 2006 Lockemann, Peter C.: Agents. In: Kirn, Stefan (Hrsg.) ; Herzog, Otthein (Hrsg.) ; Lockemann, Peter C. (Hrsg.) ; Spaniol, Otto (Hrsg.): *Multiagent Engineering – Theory and Applications in Enterprises*. Springer Verlag, 2006, S. 17–34

Lockemann und Dittrich 2004 Lockemann, Peter C. ; Dittrich, Klaus R.: *Architektur von Datenbanksystemen*. dpunkt.verlag, 2004

Lockemann und Nimis 2004 Lockemann, Peter C. ; Nimis, Jens: Flexibility through Multiagent Systems: Advancement or Disappointment? In: Van Emde Boas, Peter (Hrsg.) ; Pokorný, Jaroslav (Hrsg.) ; Bieliková, Mária (Hrsg.): *30th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2004)*, Springer Verlag, 2004, S. 41–56

Lockemann und Nimis 2005 Lockemann, Peter C. ; Nimis, Jens: Softwareagenten: Von den Eigenschaften zur Architektur. In: *it – Information Technology* 47 (2005), Nr. 1, S. 28–35

Lockemann und Nimis 2006a Lockemann, Peter C. ; Nimis, Jens: Agent Dependability as an Architectural Issue. In: Barley, Mike (Hrsg.) ; Mouratidis,

Haralambos (Hrsg.) ; Spears, Diana F. (Hrsg.) ; Unruh, Amy (Hrsg.): *Proceedings of the Third International Workshop on Safety and Security in Multiagent Systems (SaSeMAS '06)*, 2006, S. 20–36

Lockemann und Nimis 2006b Lockemann, Peter C. ; Nimis, Jens: Agent Dependability as an Architectural Issue. In: Weiss, Gerhard (Hrsg.) ; Stone, Peter (Hrsg.): *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '06)*, ACM Press, 2006, S. 1101–1103

Lockemann u. a. 2006 Lockemann, Peter C. ; Nimis, Jens ; Braubach, Lars ; Pokahr, Alexander ; Lamersdorf, Winfried: Architectural Design. In: Kirn, Stefan (Hrsg.) ; Herzog, Otthein (Hrsg.) ; Lockemann, Peter C. (Hrsg.) ; Spaniol, Otto (Hrsg.): *Multiagent Engineering – Theory and Applications in Enterprises*. Springer Verlag, 2006, S. 405–430

Lockemann und Witte 2005 Lockemann, Peter C. ; Witte, René: Agents and Databases: Friends or Foes? In: *9th International Database Applications and Engineering Symposium (IDEAS05)*, IEEE Computer Society Press, 2005, S. 137–147

Luck u. a. 2005 Luck, Michael ; McBurney, Peter ; Shehory, Onn ; Willmott, Steve: Agent Technology Roadmap—A Roadmap for Agent Based Computing / AgentLink III. 2005. – Forschungsbericht

MacKie-Mason und Wellman 2006 MacKie-Mason, Jeffrey K. ; Wellman, Michael P.: Automated Markets and Trading Agents. In: Tesfatsion, Leigh (Hrsg.) ; Judd, Kenneth L. (Hrsg.): *Handbook of Computational Economics, Vol. 2: Agent-Based Computational Economics*. North-Holland, 2006, S. 1381–1432

Maes 1995 Maes, Pattie: Artificial life meets entertainment: lifelike autonomous agents. In: *Communications of the ACM* 38 (1995), Nr. 11, S. 108–114

Mena u. a. 2000 Mena, Eduardo ; Illarramendi, Arantza ; Goni, Alfredo: Automatic Ontology Construction for a Multiagent-Based Software Gathering Service. In: Klusch, Matthias (Hrsg.) ; Kerschberg, Larry (Hrsg.): *Proceedings of the 4th International Workshop on Cooperative Information Agents*, Springer Verlag, 2000, S. 232–243

Minsky und Murata 2003 Minsky, Naftaly H. ; Murata, Takahiro: On Manageability and Robustness of Open Multi-agent Systems. In: Lucena, Carlos (Hrsg.) ; Garcia, Alessandro (Hrsg.) ; Romanovsky, Alexander (Hrsg.) ; Castro, Jaelson (Hrsg.) ; Alencar, Paulo S. C. (Hrsg.): *Software Engineering for Multi-Agent Systems II: Research Issues and Practical Applications*, 2003, S. 189–206

Müller 1996 Müller, Jörg P.: *The Design of Intelligent Agents: a Layered Approach*. Heidelberg, Deutschland : Springer Verlag, 1996. – ISBN 3-540-62003-6

Modi u. a. 2006 Modi, Pragnesh J. ; Mancoridis, Spiros ; Mongan, William M. ; Regli, William ; Mayk, Israel: Towards a Reference Model for AgentBased Systems. In: Weiss, Gerhard (Hrsg.) ; Stone, Peter (Hrsg.): *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '06) – Industry Track*. Hakodate, Japan : ACM Press, 2006, S. 1475–1482

Moss 1981 Moss, J. Eliot B.: *Nested Transactions: an Approach to Reliable Distributed Computing*, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science, Dissertation, 1981

Motro und Smets 1997 Motro, Amihai (Hrsg.) ; Smets, Philippe (Hrsg.): *Uncertainty Management in Information Systems: From Needs to Solutions*. Kluwer Academic Publishers, 1997

Nagi 2000 Nagi, Khaled: Scalability of a Transactional Infrastructure for Multi-agent Systems. In: Wagner, Tom (Hrsg.) ; Rana, Omer F. (Hrsg.): *Proceedings of the 1st International Workshop on Infrastucture for Scalable Multi-Agent Systems*, Springer Verlag, 2000, S. 266–272

Nagi 2001a Nagi, Khaled: Modeling and Simulaion of cooperative Multi-Agents in Transactional Database Enviroments. In: Wagner, Tom (Hrsg.) ; Rana, Omer F. (Hrsg.): *Proceedings of the 2nd International Workshop on Infrastructure for Scalable Multi-Agent Systems*, 2001

Nagi 2001b Nagi, Khaled: *Transactional Agents : Towards a Robust Multi-Agent System*. Heidelberg, Universität Karlsruhe (TH), Fakultät für Informatik, Dissertation, 2001

Nagi u. a. 2001 Nagi, Khaled ; Nimis, Jens ; Lockemann, Peter C.: Transactional Support for Cooperation in Multiagent-based Information Systems. In: *Verbundtagung Verteilte Informationssysteme auf der Grundlage von Objekten, Komponenten und Agenten (vertIS 2001)*, 2001

Neches u. a. 1991 Neches, Robert ; Fikes, Richard ; Finin, Tim ; Gruber, Tom ; Patil, Ramesh ; Senator, Ted ; Swartout, William R.: Enabling technology for knowledge sharing. In: *AI Magazin* 12 (1991), Nr. 3, S. 36–56

Nimis 2001 Nimis, Jens: Einbettung eines transaktionsgestützten Robustheitsdienstes in die FIPA-Plattform / Universität Karlsruhe (TH), Fakultät für Informatik. 2001. – Forschungsbericht

Nimis und Lockemann 2004 Nimis, Jens ; Lockemann, Peter C.: Robust Multi-Agent Systems: The Transactional Conversation Approach. In: *Proceedings of the 1st International Workshop on Safety and Security in Multiagent Systems (SASEMAS04)*, 2004

Nimis u. a. 2006 Nimis, Jens ; Lockemann, Peter C. ; Krempels, Karl-Heinz ; Buchmann, Erik ; Böhm, Klemens: Towards Dependable Agent Systems. In: Kirn, Stefan (Hrsg.) ; Herzog, Otthein (Hrsg.) ; Lockemann, Peter C. (Hrsg.) ; Spaniol, Otto (Hrsg.): *Multiagent Engineering – Theory and Applications in Enterprises*. Springer Verlag, 2006, S. 465–502

Nimis und Stockheim 2004 Nimis, Jens ; Stockheim, Tim: The Agent.Enterprise Multi-Multi-agent System. In: Bichler, Martin (Hrsg.) ; Holtmann, Carsten (Hrsg.) ; Kirn, Stefan (Hrsg.) ; Müller, Jörg P. (Hrsg.) ; Weinhardt, Christof (Hrsg.): *Proceedings of the Conference on Agent Technology in Business Applications (ATeBA 2004), Teilkonferenz der Multikonferenz Wirtschaftsinformatik (MKWI 2004)*. Berlin : GITO-Verlag, 2004, S. 364–371

Nitschke 2006 Nitschke, Tanja: Legal Consequences of Agent Deployment. In: Kirn, Stefan (Hrsg.) ; Herzog, Otthein (Hrsg.) ; Lockemann, Peter C. (Hrsg.) ; Spaniol, Otto (Hrsg.): *Multiagent Engineering – Theory and Applications in Enterprises*. Springer Verlag, 2006, S. 597–618

Nodine und Unruh 2000 Nodine, Marian ; Unruh, Amy: Constructing Robust Conversation Policies in Dynamic Agent Communities. In: Dignum, Frank (Hrsg.) ; Greaves, Mark (Hrsg.): *Issues in Agent Communication*, Springer Verlag, 2000, S. 205–219

O'Malley und DeLoach 2001 O'Malley, Scott A. ; DeLoach, Scott A.: Determining When to Use an Agent-Oriented Software Engineering Paradigm. In: *Proceedings of the Second International Workshop On Agent-Oriented Software Engineering (AOSE-2001)*, Springer Verlag, 2001, S. 188–205

Omicini u. a. 2004 Omicini, Andrea ; Ricci, Alessandro ; Viroli, Mirko ; Castelfranchi, Cristiano ; Tummolini, Luca: Coordination Artifacts: Environment-Based Coordination for Intelligent Agents. In: *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, IEEE Computer Society Press, 2004, S. 286–293

Papazoglou 2008 Papazoglou, Michael P.: *Web Services: Principles and Technology*. Pearson Education, 2008

Parunak 1987 Parunak, H. Van Dyke: *Manufacturing Experience with the Contract Net*. S. 285–310. In: Huhns, Michael N. (Hrsg.): *Distributed Artificial Intelligence*. London, UK : Pitman, 1987

Parunak 2000 Parunak, H. Van Dyke: A Practitioners' Review of Industrial Agent Applications. In: *Proceedings of the 2000 Conference on Autonomous Agents and Multi-Agent Systems* 3 (2000), Nr. 4, S. 389–407

Patil u. a. 1998 Patil, Ramesh S. ; Fikes, Richard E. ; Patel-Schneider, Peter F. ; McKay, Don ; Finin, Tim ; Gruber, Thomas ; Neches, Robert: The DARPA Knowledge Sharing Effort: Progress Report. In: *Readings in Agents*. Morgan Kaufmann Publishers, 1998, S. 243–254

Patterson u. a. 1988 Patterson, David ; Gibson, Garth ; Katz, Randy: A case for redundant arrays of inexpensive disks (RAID). In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, ACM Press, 1988, S. 109–116

Paurobally u. a. 2003 Paurobally, Shamimabi ; Cunningham, Jim ; Jennings, Nicholas R.: Developing agent Interaction Protocols Using Graphical and Logical Methodologies. In: *Proceedings of the AAMAS 2003 PROMAS Workshop on Programming Multi-Agent Systems*, 2003, S. 1–10

Pearl 1993 Pearl, Judea: Belief Networks Revisited. In: *Artificial Intelligence* 59 (1993), Nr. 1–2, S. 49–56

Pechoucek u. a. 2004 Pechoucek, Michal ; Vokrinek, Jiri ; Hodik, Jiri ; Becvar, Petr ; Pospisil, Jiri: ExPlanTech: Multi-agent Framework for Production Planning, Simulation and Supply Chain Management. In: Lindemann, Gabriela (Hrsg.): *Proceedings of the 2nd German Conference on Multi-agent system TEchnologieS (MATES2004)*, 2004, S. 287–300

Perrussel und Thévenin 2004 Perrussel, Laurent ; Thévenin, Jean-Marc: A Logical Approach for Describing (Dis)Belief Change and Message Processing. In: Jennings, Nicholas R. (Hrsg.) ; Sierra, Carlos (Hrsg.) ; Sonenberg, Liz (Hrsg.) ; Tambe, Milind (Hrsg.): *Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, ACM Press, 2004, S. 614–621

Petrie 1997 Petrie, Charles J.: What's an Agent...and What's so Intelligent About It? In: *IEEE Internet Computing* 1 (1997), Nr. 4, S. 4–5

Pichler u. a. 2002 Pichler, Josef ; Plösch, Reinhold ; Weinreich, Rainer: MASIF und FIPA: Standards für Agenten. In: *Informatik Spektrum* 25 (2002), Nr. 2, S. 91–100

Pitoura 1998 Pitoura, Evaggelia: Transaction-Based Coordination of Software Agents. In: *Proceedings of the 9th International Conference on Database and Expert Systems Applications (DEXA'98)*, Springer Verlag, 1998, S. 460–471

Pitoura und Bhargava 1995 Pitoura, Evaggelia ; Bhargava, B.: A Framework for Providing Consistent and Recoverable Agent-based Access to Heterogeneous Mobile Databases. In: *SIGMOD Record (ACM Special Interest Group on Management of Data)* 24 (1995), Nr. 3, S. 44–49

Pleisch 2002 Pleisch, Stefan: *Fault-Tolerant and Transactional Mobile Agent Execution*, EPFL, Lausanne, Schweiz, Dissertation, Oktober 2002

Pleisch und Schiper 2004 Pleisch, Stefan ; Schiper, André: Approaches to Fault-Tolerant and Transactional Mobile Agent Execution – An Algorithmic View. In: *ACM Computing Surveys* 36 (2004), Nr. 3, S. 219–262

Pokahr u. a. 2005 Pokahr, Alexander ; Braubach, Lars ; Lamersdorf, Winfried: Jadex: A BDI Reasoning Engine. In: Bordini, Rafael H. (Hrsg.) ; Dastani, Mehdi (Hrsg.) ; Dix, Jürgen (Hrsg.) ; El Fallah Seghrouchni, Amal (Hrsg.): *Multi-Agent Programming*. Kluwer Academic Publishers, 2005, S. 149–174

Posch u. a. 2007 Posch, Torsten ; Birken, Klaus ; Gerdom, Michael: *Basiswissen Softwarearchitektur*. dpunkt.verlag, 2007

Poslad u. a. 2000 Poslad, Stefan ; Buckle, Phil ; Hadingham, Rob: The FIPA-OS Agent Platform : Open Source for Open Standards. In: Bradshaw, Jeffrey (Hrsg.) ; Arnold, Geoff (Hrsg.): *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000)*. Manchester, UK : The Practical Application Company Ltd., 2000, S. 355–368

Poslad und Charlton 2001 Poslad, Stefan ; Charlton, Patricia: Standardizing Agent Interoperability: The FIPA Approach. In: Carbonell, Jaime G. (Hrsg.) ; Siekmann, Jörg (Hrsg.): *Tutorial Collection of the 2001 Summer School on Multi-Agent Systems and Their Applications*. Springer Verlag, 2001, S. 98–117

Reussner und Hasselbring 2006 Reussner, Ralf H. (Hrsg.) ; Hasselbring, Wilhelm (Hrsg.): *Handbuch der Software-Architektur*. dpunkt.verlag, 2006

Riebisch 2006 Riebisch, Matthias: Prozess der Architektur und Komponentenentwicklung. In: Reussner, Ralf H. (Hrsg.) ; Hasselbring, Wilhelm (Hrsg.): *Handbuch der Software-Architektur*. dpunkt.verlag, 2006, S. 65–88

Rimassa 2004 Rimassa, Giovanni: JADE Persistence Add-On / TILAB S.p.A. 2004. – Systemdokumentation

Rimassa u. a. 2005 Rimassa, Giovanni ; Calisti, Monique ; Kernland, Martin E.: Living Systems/Technology Suite / Whitestein Technologies AG. Zürich, Schweiz : Whitestein Technologies AG, 2005. – Forschungsbericht

Rosenschein und Zlotkin 1998 Rosenschein, Jeffrey S. ; Zlotkin, Gilad: *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. MIT Press, 1998

Rovatsos 2007 Rovatsos, Michael: Dynamic Semantics for Agent Communication Languages. In: Durfee, Edmund H. (Hrsg.) ; Yokoo, Makoto (Hrsg.): *Proceedings of the Sixth Intl. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2007)*, 2007, S. 100–107

Rudnianski und Bestougeff 2004 Rudnianski, Michel ; Bestougeff, Helene: Multi-agent Systems Reliability, Fuzziness, and Deterrence. In: Hinchey, Michael G. (Hrsg.) ; Rash, James L. (Hrsg.) ; Truszkowski, Walt (Hrsg.) ; Rouff, Christopher (Hrsg.): *Proceedings of the 3rd International Workshop on Formal Approaches to Agent-Based Systems (FAABS '04)*, Springer Verlag, 2004, S. 41–56

Russell und Norvig 2004 Russell, Stuart ; Norvig, Peter: *Künstliche Intelligenz: ein moderner Ansatz*. Bd. 2. Pearson Studium, 2004

Sandholm und Lesser 2001 Sandholm, Tuomas W. ; Lesser, Victor R.: Leveled Commitment Contracts and Strategic Breach. In: *Games and Economic Behavior* 35 (2001), S. 212–270

Sandholm und Lesser 2002 Sandholm, Tuomas W. ; Lesser, Victor R.: Leveled-Commitment Contracting: A Backtracking Instrument for Multiagent Systems. In: *The AI Magazine* 23 (2002), Nr. 3, S. 89–100

Schillo u. a. 2001 Schillo, Michael ; Bürckert, Hand-Jürgen ; Fischer, Klaus ; Klusch, Matthias: Towards a Definition of Robustness for Market-style Open Multi-agent Systems. In: Müller, Jörg P. (Hrsg.) ; Andre, Elisabeth (Hrsg.) ; Sen, Sandip (Hrsg.) ; Frasson, Claude (Hrsg.): *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, ACM Press, 2001, S. 75–76

Scholz u. a. 2005 Scholz, Thorsten ; Krempels, Karl-Heinz ; Nimis, Jens ; Schiemann, Bernhard ; Woelk, Peer-Oliver ; Braubach, Lars ; Pokahr, Alexander: www.AgentEnterprise.net – a MMAS-based Web-Portal for Planning and Control of Complex Manufacturing Supply Chains. In: Willmot, Steven (Hrsg.): *Proceedings of NETDEMO 2005: openNet Networked Agents Demonstration at AAMAS'05*, 2005

Scholz u. a. 2006 Scholz, Thorsten ; Timm, Ingo J. ; Herzog, Otthein ; Görz, Günter ; Schiemann, Bernhard: Semantics for Agents. In: Kirn, Stefan (Hrsg.) ; Herzog, Otthein (Hrsg.) ; Lockemann, Peter C. (Hrsg.) ; Spaniol, Otto (Hrsg.): *Multiagent Engineering – Theory and Applications in Enterprises*. Springer Verlag, 2006, S. 431–464

Sedivy 2003 Sedivy, Miroslav: *Eine Simulationsplattform zur Bewertung von Multiagentensystemen in der Fertigungsplanung und -steuerung*, Universität Karlsruhe (TH), Fakultät für Informatik, Diplomarbeit, 2003

Shafer 1976 Shafer, Glenn: *A Mathematical Theory of Evidence*. Princeton, NJ, USA : Princeton University Press, 1976

Shaw und Garlan 1996 Shaw, Mary ; Garlan, David: *Software Architecture: Perspective on an Emerging Discipline*. Prentice-Hall, 1996

Shehory 1998 Shehory, Onn: Architectural Properties of Multi-Agent Systems / Carnegie Mellon University. 1998 (CMU-RI-TR-9828). – Forschungsbericht

Shehory 2000 Shehory, Onn: Software Architecture Attributes of Multi-agent Systems. In: Ciancarini, Paolo (Hrsg.) ; Wooldridge, Michael (Hrsg.): *1st International Workshop on Agent-oriented Software Engineering (AOSE '00)*, Springer Verlag, 2000, S. 77–90

Shen und Norrie 1998 Shen, Weiming ; Norrie, Douglas H.: Combining Mediation and Bidding Mechanisms for Agent-based Manufacturing Scheduling (Poster). In: *Proceedings of the 2nd International Conference on Autonomous Agents (Agents '98)*, ACM Press, 1998, S. 469–470

Shen und Norrie 1999 Shen, Weiming ; Norrie, Douglas H.: Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey. In: *Knowledge and Information Systems, an International Journal* 1 (1999), Nr. 2, S. 129 – 156

Sher u. a. 2001 Sher, Ron ; Aridor, Yariv ; Etzion, Opher: Mobile Transactional Agents. In: *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS 2001)*, IEEE Computer Society Press, 2001, S. 73–80

Smith 1980 Smith, Reid G.: The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. In: *IEEE Transactions on Computers* (1980), Nr. 12, S. 1104–1113

Sorge 2004 Sorge, Christoph: *Softwareagenten : Vertragsschluss, Vertragsstrafe, Reugeld*. Karlsruhe, Universität Karlsruhe (TH), Diplomarbeit, 2004

Steegmans u. a. 2004 Steegmans, Elke ; Weyns, Danny ; Holvoet, Tom ; Berbers, Yolande: A Design Process for Adaptive Behavior of Situated Agents. In: Odell, James (Hrsg.) ; Giorgini, Paolo (Hrsg.) ; Müller, Jörg P. (Hrsg.): *Fifth International Workshop on Agent-Oriented Software Engineering (AOSE'04)* Bd. 3382, Springer Verlag, 2004, S. 109–125

Stockheim u. a. 2004 Stockheim, Tim ; Nimis, Jens ; Scholz, Thorsten ; Stehli, Marcel: How to Build Multi-Multi-Agent Systems: the Agent.Enterprise Approach. In: *Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS 2004)*, 2004

Sycara u. a. 2003 Sycara, Katia ; Paolucci, Massimo ; Velsen, Martin V. ; Giampapa, Joseph: The RETSINA MAS Infrastructure. In: *Autonomous Agents and Multi-Agent Systems* 7 (2003), Nr. 1–2, S. 29–48

Timm u. a. 2006 Timm, Ingo J. ; Scholz, Thorsten ; Fürstenau, Hendrik: From Testing to Theorem Proving. In: Kirn, Stefan (Hrsg.) ; Herzog, Otthein (Hrsg.) ; Lockemann, Peter C. (Hrsg.) ; Spaniol, Otto (Hrsg.): *Multiagent Engineering – Theory and Applications in Enterprises*. Springer Verlag, 2006, S. 531–554

Truszkowski 2006 Truszkowski, Walter F.: What is an Agent? And What is an Agent Community? In: Rouff, C.A. (Hrsg.) ; Hinchey, M. (Hrsg.) ; Rash, J. (Hrsg.) ; Truszkowski, W. (Hrsg.) ; Gordon-Spears, D. (Hrsg.): *Agent Technology from a Formal Perspective*. Heidelberg, Deutschland : Springer Verlag, 2006 (NASA Monographs in Systems and Software Engineering), S. 3–24

Truszkowski und Rouff 2001 Truszkowski, Walter F. ; Rouff, Christopher: *New Agent Architecture for Evaluation in Goddard's Agent Concepts Testbed*. Online veröffentlicht unter: `http://agents.gsfc.nasa.gov/papers/pdf/aap41.pdf` (Zuletzt besucht am 14. September 2008). 2001

Unruh u. a. 2004 Unruh, Amy ; Bailey, James ; Ramamohanarao, Kotagiri: A Framework for Goal-Based Semantic Compensation in Agent Systems. In: *Proceedings of the Second International Workshop on Safety and Security in Multi Agent Systems*, 2004, S. 125–140

Unruh u. a. 2005 Unruh, Amy ; Harjadi, Henry ; Bailey, James ; Ramamohanarao, Kotagiri: Compensation-based Recovery Management in Multi-Agent Systems. In: *Proceedings of the Second IEEE Symposium on Multi-Agent Security and Survivability (IEEE MASS)*, 2005, S. 85–94

Van Eijk u. a. 2003 Van Eijk, Rogier M. ; De Boer, Frank S. ; Van Der Hoek, Wiebe ; Meyer, John-Jules C.: A Verification Framework for Agent Communication. In: *Autonomous Agents and Multi-Agent Systems* 6 (2003), Nr. 2, S. 185–219

Virdhagriswaran u. a. 1995 Virdhagriswaran, Sankar ; Osisek, Damian ; O'Connor, Pat: Standardizing Agent Technology. In: *StandardView* 3 (1995), Nr. 3, S. 96–101

Vogler 1999 Vogler, Hartmut: *Eine Infrastruktur für den Einsatz autonomer mobiler Agenten in Electronic Commerce*, TU Darmstadt, Dissertation, 1999

Vogler und Buchmann 1998 Vogler, Hartmut ; Buchmann, Alejandro: Using Multiple Mobile Agents for Distributed Transactions. In: *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems*, IEEE Computer Society Press, 1998, S. 114–121

Vogler u. a. 1997 VOGLER, Hartmut ; KUNKELMANN, Thomas ; MOSCHGATH, Marie-Luise: Distributed Transaction Processing as a Reliability Concept for Mobile Agents. In: *Proceedings of the 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'97)*, IEEE Computer Society Press, 1997, S. 59–64

Vogt 2001 VOGT, Rainer: *Einbettung eines transaktionsgestützten Robustheitsdienstes in das FIPA-Agentenrahmenwerk*, Universität Karlsruhe (TH), Diplomarbeit, 2001

Vossen 1995 VOSSEN, Gottfried: Database Transaction Models. In: VAN LEEUWEN, Jan (Hrsg.): *Computer Science Today – Recent Trends and Developments*. Springer Verlag, 1995, S. 560–574

Wang u. a. 2007 WANG, Mingzhong ; UNRUH, Amy ; RAMAMOHANARAO, Kotagiri: ARTS: Agent-oriented Robust Transactional System. In: DURFEE, Edmund H. (Hrsg.) ; YOKOO, Makoto (Hrsg.): *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007)*, 2007, S. 529–531

Wang u. a. 1999 WANG, Wen-Li ; WU, Ye ; CHEN, Mei-Hwa: An Architecture-Based Software Reliability Model. In: *Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing (PRDC '99)*, IEEE Computer Society Press, 1999, S. 143–152

Wächter und Reuter 1990 WÄCHTER, Helmut ; REUTER, Andreas: Grundkonzepte und Realisierungsstrategien des ConTract-Modells. In: HÄRDER, T. (Hrsg.) ; WEDEKIND, H. (Hrsg.) ; ZIMMERMANN, G. (Hrsg.): *Entwurf und Betrieb verteilter Systeme*, 1990, S. 150–171

Wächter und Reuter 1992 WÄCHTER, Helmut ; REUTER, Andreas: The ConTract Model. In: ELMAGARMID, Ahmed K. (Hrsg.): *New Transaction Models for Advanced Applications*. Morgan Kaufmann Publishers, 1992, S. 219–264

Weikum und Vossen 2002 WEIKUM, Gerhard ; VOSSEN, Gottfried: *Transactional Information Systems: Theory, Algorithms and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann Publishers, 2002

Weyns u. a. 2006 Weyns, Danny ; Holvoet, Tom ; Schelfthout, Kurt: Multiagent Systems as Software Architecture – Another Perspective on Software Engineering with Multiagent Systems. In: Weiss, Gerhard (Hrsg.) ; Stone, Peter (Hrsg.): *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '06) – Industry Track*. Hakodate, Japan : ACM Press, 2006, S. 1314–1316

Weyns u. a. 2007 Weyns, Danny ; Omicini, Andrea ; Odell, James: Environment, First-Order Abstraction in Multiagent Systems. In: *International Journal on Autonomous Agents and Mutli-Agent Systems* 14 (2007), Nr. 1, S. 5–30

Weyns u. a. 2004a Weyns, Danny ; Steegmans, Elke ; Holvoet, Tom: Protocol-Based Communication for Situated Multi-Agent Systems. In: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*. Washington, DC, USA : IEEE Computer Society, 2004, S. 118–125

Weyns u. a. 2004b Weyns, Danny ; Steegmans, Elke ; Holvoet, Tom: Towards Active Perception in Situated Multi-Agent Systems. In: *Applied Artificial Intelligence* 18 (2004), Nr. 9-10, S. 867–883

Woelk u. a. 2006 Woelk, Peer-Oliver ; Rudzio, Holger ; Zimmermann, Roland ; Nimis, Jens: Agent.Enterprise in a Nutshell. In: Kirn, Stefan (Hrsg.) ; Herzog, Otthein (Hrsg.) ; Lockemann, Peter C. (Hrsg.) ; Spaniol, Otto (Hrsg.): *Multiagent Engineering – Theory and Applications in Enterprises*. Springer Verlag, 2006, S. 73–90

Wooldridge und Jennings 1995 Wooldridge, Michael ; Jennings, Nicholas R.: Intelligent Agents: Theory and Practice. In: *The Knowledge Engineering Review* 10 (1995), Nr. 2, S. 115–152

Wooldridge 2002 Wooldridge, Michael J.: *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2002

World Wide Web Consortium (W3C) 2004 World Wide Web Consortium (W3C): *Web Ontology Language (OWL) – W3C Recommendation*.

Online veröffentlicht unter: `http://www.w3.org/2004/OWL/` (Zuletzt besucht am 14. September 2008). 2004

World Wide Web Consortium (W3C) 2006 World Wide Web Consortium (W3C): *Extensible Markup Language (XML) 1.0 (Fourth Edition) – W3C Recommendation*. Online veröffentlicht unter: `http://www.w3.org/TR/xml/` (Zuletzt besucht am 14. September 2008). 2006

Yue 2007 Yue, Jing Z.: *Robuste Verhandlungen zwischen verteilten, agentenbasierten Fahrerassistenzsystemen*, Universität Karlsruhe (TH), Fakultät für Informatik, Diplomarbeit, 2007

Zäpfel 1996 Zäpfel, Günther: *Grundzüge des Produktions- und Logistikmanagement*. de Gruyter, 1996