



Thema:

**Effiziente Trainingsverfahren  
für die Multi-Agenten-Systementwicklung**

**Diplomarbeit**

Arbeitsgruppe Softwaretechnik

Themensteller: Prof. Dr.-Ing. habil. Reiner Dumke  
Betreuer: Dipl.-Inf. Cornelius Wille

vorgelegt von: Thomas Wesche  
Weidenstr. 3  
39114 Magdeburg  
(0179) 1309851  
E-Mail: [ThomasWesche@web.de](mailto:ThomasWesche@web.de)

Abgabetermin: 31. Dezember 2004

## Inhaltsverzeichnis

Inhaltsverzeichnis .....	II
Verzeichnis der Abkürzungen und Akronyme .....	IV
Abbildungsverzeichnis .....	VII
Tabellenverzeichnis.....	IX
1 Einleitung .....	1
1.1 Problemstellung .....	2
1.2 Struktur der Arbeit .....	3
2 Intelligente Agenten .....	4
2.1 Begriffsbestimmung und Abgrenzung.....	4
2.2 Klassifizierung .....	7
2.3 Architekturen .....	8
2.3.1 Kognitive Agenten .....	9
2.3.2 Reaktive Agenten .....	11
2.3.3 Hybride Agenten .....	12
3 Multi-Agenten-Systeme.....	14
3.1 Begriffsbestimmung.....	14
3.2 Klassifizierung.....	15
3.3 Architekturen .....	16
3.4 Interaktion und Agenten-Koordination.....	17
3.4.1 Kooperationsprotokolle .....	18
3.4.2 Protokolle für den Wettbewerb .....	20
3.5 Agenten-Kommunikation .....	21
3.6 Multi-Agenten-Organisation.....	25
3.7 Intelligenz .....	27
4 Agentenorientiertes Software-Engineering.....	28
4.1 Begriffsbestimmung.....	28
4.2 Der Entwicklungsprozess von MAS .....	28
4.2.1 Analyse, Spezifikation und Design .....	29
4.2.2 Implementierung .....	30
4.2.3 Training .....	32
4.3 Standards.....	33
5 Softwaremessung von Agentenbasierten Systemen.....	35
5.1 Allgemeine Ansätze für die Agentenmessung.....	35
5.2 Effizienz und deren Anwendung auf Lern- und Trainingsverfahren.....	38
5.3 Zusammenfassung .....	43
6 Effiziente Trainingsmethoden in Multi-Agenten-Systemen.....	45
6.1 Theoretische Grundlagen.....	45
6.1.1 Ausgewählte Techniken der KI.....	48
6.1.2 Besonderheiten des Trainings in MAS.....	53

6.1.3	Evolution der Trainingsverfahren.....	55
6.2	Training eines einzelnen Agenten.....	57
6.2.1	Aktionsselektion (Agent Operation).....	57
6.2.2	Agentenwissen (Agent Knowledge).....	61
6.3	Training von Multi-Agenten-Systemen .....	64
6.3.1	Agenten-Koordination (Agent Coordination) .....	64
6.3.2	Agenten-Kommunikation (Agent Communication).....	76
7	Effiziente Technologien für das Agenten-Training .....	84
7.1	Agent Building and Learning Environment (ABLE).....	84
7.2	JAVa DEvelopment Framework (JADE) .....	87
7.3	JACK™ Intelligent Agents.....	89
7.4	Soar .....	91
7.5	Agent Academy .....	94
8	Bewertung und Auswahl effizienter Trainingsverfahren.....	97
9	Schlussbetrachtung und Ausblick .....	102
A	Einsatzdomänen von MAS.....	103
	Literaturverzeichnis.....	105

## Verzeichnis der Abkürzungen und Akronyme

AARIA	Autonomous Agents for Rock Island Arsenal
ABLE	Agent Building and Learning Environment
ACE	Action Estimation Algorithm
ACL	Agenten Communication Language
ADAPT	Adaptive Multi Agent Process Planning & Coordination of Clinical Trials
ADEPT	Agent-based Business Management
AF	Agent Factory
AGE	Action Group Estimation Algorithm
AiC	Agent-in-Charge
AiCoop	Agent-in-Cooperation
AiS	Agent-in-ST
AOP	Agentenorientierte Programmierung
AOSE	Agentenorientiertes Software-Engineering
ARCHON	ARchitecture for Cooperative Heterogeneous ON-line systems
ARL	ABLE RuleSet Language
ATM	Agent Training Module
Aufl.	Auflage
AUR	Agent Use Repository
BBA	Bucket Brigade Algorithm
BDI	Belief-Desire-Intention
CASE	Computer Aided Software Engineering
CIRL	Concurrent Isolated Reinforcement Learning
CM	Coordination Method
CoMoMAS	Conceptual Modelling of Multi Agent Systems
CORBA	Common Object Request Broker Architecture
COTS	Commercial-Off-The-Shelf
CPU	Central Processing Unit
CT	Cooperative Task
DAI	Distributed Artificial Intelligence
DAML	DARPA Agent Markup Language
DARPA	Defense Advanced Research Project Agency
DASMA	Deutschsprachige Anwendergruppe für Software-Metrik und Aufwandschätzung e.V.
DM	Data Mining
DMM	Data Mining Module
e. g.	zum Beispiel
EBL	Explanation-based Learning
et al.	und andere
EU	Europäische Union

FIPA	Foundation for Intelligent Physical Agents
Gaia	Generic Architecture for Information Availability
GQM	Goal-Question-Metric
HTTP	Hypertext Transfer Protocol
ILP	Inductive Logic Programming
IBM	International Business Machines
IIOP	Internet Inter-ORB Protocol
inkl.	inklusive
Int.	international
INTERRAP	INTEgration of Reactive behavior and RAational Planning
ISO	International Organization for Standardization
JACOB	Jack Objects
JADE	JAva DEvelopment Framework
JAL	Jack Agent Language
JESS	Java Expert System Shell
KI	Künstliche Intelligenz
KIF	Knowledge Interchange Format
KQML	Knowledge Query and Manipulation Language
KSE	Knowledge Sharing Effort
LEAP	Lightweight Extensible Agent Platform
LISP	List Processor
LPGL	GNU Library or Lesser General Public License
LTM	Long Term Memory
MAL	Multi Agent Learning
MALINA	Multi-Agent Local Integrated Network Associations
MAS	Multi-Agenten-System
MASA	Multiagent Systems, Artificial Societies, and Simulated Organizations
MASIF	Mobile Agent System Interoperability Facilities
MASSIVE	MultiAgent SystemS Iterative View Engineering
MIA	Mobile Interface Agents
MIT	Massachusetts Institute of Technology
ML	Machine Learning
MOA	Mobile Objects and Agents
NASA	National Aeronautics and Space Administration
NN	Neuronal Network
No.	Nummer
o. g.	oben genannte
OMG	Object Management Group
OO	Objektorientiert
OOP	Objektorientierte Programmierung

OOSE	Objektorientiertes Software-Engineering
ORB	Object Request Broker
OvG	Otto-von-Guericke
OWL	Web Ontology Language
PC	Personal Computer
PDA	Personal Digital Assistant
PLACA	PLAnning Communicating Agents
PR	Production Rule
Prolog	PROgramming in LOGic
PSP	Profit Sharing Plan
RADL	Runtime Agent Development Language
RAPPID	Responsible Agents for Product-Process Integrated Design
RDF	Resource Description Framework
RL	Reinforcement Learning
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SAL	Single Agent Learning
SL	Semantic Language
SQL	Structured Query Language
SS	Sommersemester
ST	Specific Task
Tab.	Tabelle
Tcl	Tool Command Language
TCP/IP	Transmission Control Protocol/Internet Protocol
UA	Untrainierter Agent
UML	Unified Modeling Language
UPC	Utility-Probability-Cost
VIENA	Virtual Environments and Agents
VKI	Verteilte Künstliche Intelligenz
Vol.	Jahrgang
WBI	Web Browser Intelligence
WM	Working Memory
WS	Wintersemester
WWW	World Wide Web
XML	Extensible Markup Language
z. B.	zum Beispiel

## Abbildungsverzeichnis

<b>Abb. 2-1:</b> Einflussgebiete auf die Agenten-Technologie .....	4
<b>Abb. 2-2:</b> A Classification of Software Agents .....	8
<b>Abb. 2-3:</b> Architektur eines kognitiven Agenten.....	10
<b>Abb. 2-4:</b> Architektur reaktiver Agenten.....	11
<b>Abb. 2-5:</b> Konzeptuelles INTERRAP-Agentenmodell.....	13
<b>Abb. 3-1:</b> Komponenten eines Multi-Agenten-Systems.....	16
<b>Abb. 3-2:</b> Eine Taxonomie möglicher Koordinationsformen.....	17
<b>Abb. 3-3:</b> Die Schritte im Kontraktnetzprotokoll.....	19
<b>Abb. 3-4:</b> Struktur eines Blackboard-Systems.....	19
<b>Abb. 3-5:</b> Konfliktbehandlung zwischen Software-Agenten.....	20
<b>Abb. 3-6:</b> Mikro-Makro-Beziehungen in Multi-Agenten-Systemen .....	25
<b>Abb. 4-1:</b> Bestandteile AOSE.....	28
<b>Abb. 4-2:</b> MAS-Entwicklungsphasen.....	29
<b>Abb. 4-3:</b> Sprachen in MAS .....	30
<b>Abb. 4-4:</b> Strategische Ansätze für das Agenten-Training.....	32
<b>Abb. 4-5:</b> FIPA-Referenzarchitektur .....	34
<b>Abb. 5-1:</b> Fischgrätendiagramm .....	36
<b>Abb. 5-2:</b> Phasen Goal-Question-Metric .....	36
<b>Abb. 5-3:</b> Unterschiedliche Geschwindigkeiten beim Konvergieren .....	42
<b>Abb. 5-4:</b> Lernkurve eines Entscheidungsbaums und eines Perzeptrons .....	43
<b>Abb. 6-1:</b> Abstrakte Darstellung des Lernprozesses.....	46
<b>Abb. 6-2:</b> Übersicht Fallbasiertes Schließen.....	51
<b>Abb. 6-3:</b> Schema eines Lernprozesses, der Hintergrundwissen nutzt und ansammelt.....	52
<b>Abb. 6-4:</b> Entwicklungsformen des Lernens in MAS .....	56
<b>Abb. 6-5:</b> Auswertung Experiment 1.....	59
<b>Abb. 6-6:</b> Gitterwelt.....	60
<b>Abb. 6-7:</b> Auswertung Experiment 2.....	60
<b>Abb. 6-8:</b> Testumgebung für Beispiel 3 .....	63
<b>Abb. 6-9:</b> Auswertung Experiment 3.....	64
<b>Abb. 6-10:</b> Übersicht Lernmethoden für das Koordinationsproblem .....	65
<b>Abb. 6-11:</b> Testumgebung für Beispiel 4 .....	65
<b>Abb. 6-12:</b> Auswertung Experiment 4.....	66
<b>Abb. 6-13:</b> Testumgebung Beispiel 5 .....	67
<b>Abb. 6-14:</b> Auswertung Experiment 5.....	69
<b>Abb. 6-15:</b> Auswertung Experiment 6.....	71

<b>Abb. 6-16:</b> Auswertung Experiment 7.....	73
<b>Abb. 6-17:</b> Gitterwelt Beispiel 8.....	74
<b>Abb. 6-18:</b> Auswertung Experiment 8.....	75
<b>Abb. 6-19:</b> Gitterwelt Beispiel 10.....	80
<b>Abb. 6-20:</b> Auswertung III Beispiel 10 .....	82
<b>Abb. 7-1:</b> Able Agent Editor .....	87
<b>Abb. 7-2:</b> Architektur eines JADE-Agenten.....	89
<b>Abb. 7-3:</b> Architektur eines Soar-Agenten.....	93
<b>Abb. 7-4:</b> Beschreibung von Umweltzuständen mittels gelabelter Graphen.....	93
<b>Abb. 7-5:</b> Aufbau Agent Academy .....	95
<b>Abb. 7-6:</b> Schematische Darstellung des Trainings in der Agent Academy .....	96
<b>Abb. A-1:</b> Eine Klassifikation der verschiedenen Anwendungstypen für Multi-Agenten- Systeme.....	103

## Tabellenverzeichnis

<b>Tab. 2-1:</b> Eigenschaften von Software-Agenten .....	7
<b>Tab. 3-1:</b> Eigenschaften von MAS .....	15
<b>Tab. 4-1:</b> Vergleich von OOP und AOP.....	31
<b>Tab. 5-1:</b> Produkt-/Performancemetriken.....	39
<b>Tab. 5-2:</b> Prozessmetriken.....	40
<b>Tab. 5-3:</b> Ressourcenmetriken.....	41
<b>Tab. 6-1:</b> Aufstellung Roboter und lieferbares Gericht.....	77
<b>Tab. 6-2:</b> Auswertung I Beispiel 10 .....	81
<b>Tab. 6-3:</b> Auswertung II Beispiel 10 .....	82
<b>Tab. 7-1:</b> Überblick ABLE.....	85
<b>Tab. 7-2:</b> Überblick JADE.....	88
<b>Tab. 7-3:</b> Überblick JACK .....	89
<b>Tab. 7-4:</b> Überblick Soar .....	92
<b>Tab. 7-5:</b> Überblick Agent Academy .....	94
<b>Tab. 8-1:</b> Effiziente Trainingsmethoden I .....	97
<b>Tab. 8-2:</b> Effiziente Trainingsmethoden II.....	98
<b>Tab. 8-3:</b> Effiziente Trainingsmethoden III.....	99
<b>Tab. 8-4:</b> Effiziente Trainingstechnologien.....	101
<b>Tab. A-1:</b> Einsatzgebiete von Agenten.....	104

## 1 Einleitung

Der Begriff „Intelligenter Agent“ gehörte in den 90er Jahren zu den Schlagwörtern in der Informatik. Software-Agenten, oder kurz Agenten<sup>1</sup>, wurden phantastische Fähigkeiten zugesprochen. Inzwischen hat sich die Euphorie gelegt und man ist zur „Tagesordnung“ übergegangen. Agentenbasierte Systeme sind heute, wenn auch nicht immer offensichtlich, in vielen Bereichen anzutreffen. Das Einsatzspektrum reicht dabei von der Unterstützung bei der Informationssuche und -aufbereitung im World Wide Web (WWW) über persönliche Assistenten, mobile Anwendungen und den Electronic Commerce bis hin zur Steuerung autonomer Roboter. Diese Aufzählung ist bei weitem nicht vollständig<sup>2</sup>.

Obwohl man sich in der Fachwelt bis dato auf keine gemeinsame Definition eines „Intelligenten Agenten“ einigen konnte, gibt es mehrere charakteristische Merkmale, die ihn auszeichnen (vgl. Weiß (2001)):

- *Flexibilität*: Ein Agent ist sowohl zu reaktivem (Umweltveränderungen in angemessener Zeit berücksichtigendem) als auch proaktivem (im Hinblick auf Zwischen- und Endziele vorausschauendem) Verhalten fähig.
- *Autonomie*: Ein Agent kann bei Bedarf selbständig und unabhängig Entscheidungen über von ihm auszuführende zielrelevante Handlungen treffen. In diesem Sinne besitzt er Kontrolle über seinen internen Zustand und sein Verhalten.
- *Interaktivität*: Mehrere Agenten können auf hohem Niveau in Wechselwirkung treten, z. B. bei automatisierten Verhandlungen, Vertragsabschließungen oder verteilten Planungs- und Problemlösungsprozessen.

Das Konzept, Aufgaben an selbständig handelnde Einheiten zu delegieren, stammt aus der Robotertechnik sowie der Künstlichen Intelligenz (KI). Es wurde durch die Verteilte Künstliche Intelligenz (VKI) aufgegriffen und weiterentwickelt. Gegenstand der Forschung der VKI sind u. a. Multi-Agenten-Systeme (MAS), die sich aus interagierenden autonomen Agenten zusammensetzen. Vorbilder für derartige Systeme sind zum einen soziale Organisationen (Teamarbeit beim Menschen) und zum anderen biologische Verbände (Insektenstaaten). MAS stellen einen neuen Ansatz in der Informatik dar, der den folgenden Punkten Rechnung trägt (vgl. Ferber (2001)):

- Probleme sind von ihrer Natur her physisch verteilt: z. B. Transportnetzwerke, Kontrollsysteme in industriellen Großbetrieben, Produktionsprozesse usw.

---

<sup>1</sup> Intelligenter Agent, Software-Agent und Agent werden als Synonyme gebraucht. Eine genauere Unterscheidung wird in Kapitel 2.1 vorgenommen.

<sup>2</sup> Eine Auflistung von Einsatzdomänen nebst Beispielen findet sich im Anhang A (siehe Tab. A-1).

- Probleme sind in funktionaler Hinsicht verteilt und heterogen: Der Entwurf komplexer industrieller Produkte erfordert z. B. die Mitarbeit von Spezialisten verschiedener Fachrichtungen - eine einzelne Person wäre überfordert.
- Zunehmende Vernetzung von Systemen: Im Zeitalter globaler Netzwerke sind die Daten und Rechnerkapazitäten über eine große Anzahl von Netzknoten verteilt. MAS sind für offene, verteilte, heterogene und flexible Architekturen ausgelegt, ohne irgendwelche Strukturen a priori vorauszusetzen.
- Zunehmende Komplexität von Problemen: Wenn Aufgabenstellungen zu umfangreich werden (Beispiel Luftverkehrsüberwachung), eignen sich auf lokalen (verteilten) Perspektiven gründende Lösungsansätze oft besser, um ein Ziel rasch und in guter Qualität zu erreichen.
- Adaptive Systeme: Zukünftige Software muss in der Lage sein, sich an Veränderungen der Aufgabenstruktur und der Umwelt anpassen zu können. Ständig steigende Anforderungen führen zu Erweiterungen bzw. Änderungen der Funktionalität sowie dem Aufbau von Computersystemen. Für solche Herausforderungen eignen sich MAS besonders, weil sie verteilt sind, durch das Prinzip der Lokalität logische Strukturen erzwingen und die Integration, Aufnahme und Ausgliederung von Agenten sogar zur Laufzeit unterstützen.
- Software-Engineering: In der Softwareentwicklung werden Architekturen bevorzugt, die auf autonomen, interagierenden Einheiten basieren (im Gegensatz zum monolithischen Ansatz).

## 1.1 Problemstellung

Die Komplexität und/oder Struktur von Problemen übersteigt oft die Fähigkeiten eines einzelnen Agenten, z. B. die Überwachung und Steuerung von Netzwerken oder Produktionsprozessen. Dem versucht man durch den Einsatz von MAS zu begegnen. Die Umwelt, in der sie typischerweise wirken, lässt sich als komplex, offen, dynamisch und unvorhersehbar beschreiben (vgl. Weiss (1999a)). Es ist daher extrem schwierig oder sogar unmöglich, diese Systeme a priori korrekt und komplett zu spezifizieren. Deshalb müssen intelligente Agenten die Fähigkeit besitzen, sowohl sich selbst, als auch das MAS im Ganzen anzupassen bzw. weiterzuentwickeln.

Die damit verbundenen Problemstellungen sind ein aktuelles Forschungsgebiet verschiedener Fachrichtungen. Ziel dieser Diplomarbeit ist es, einen strukturierten Überblick über Lern- und Trainingsmethoden in MAS zu geben. Des Weiteren werden momentan verfügbare Software-Werkzeuge für die Agentenentwicklung im Hinblick auf ihre Unterstützung bei der Implementierung von Lernprozessen untersucht. Darüber hinaus sollen erste Bewertungsformen der Effizienz von Trainingsverfahren angewandt und diskutiert werden.

## 1.2 Struktur der Arbeit

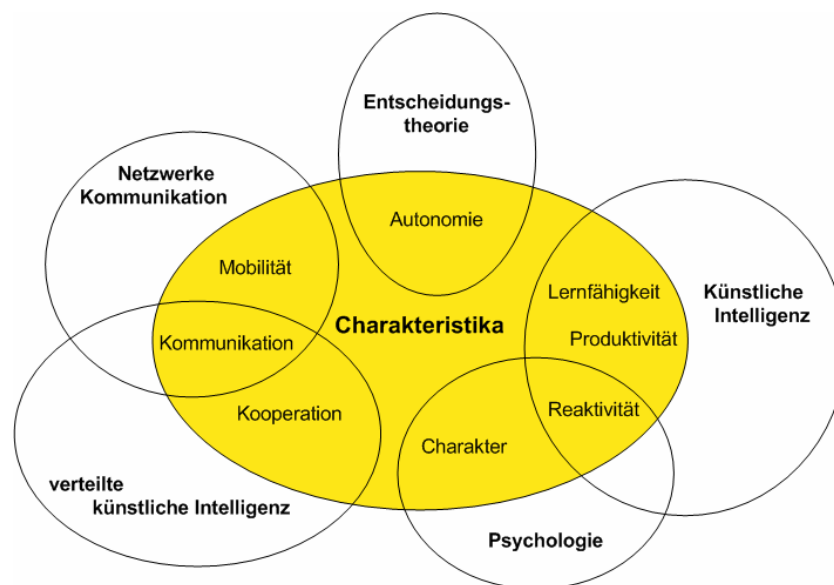
Nach der Einführung in das Thema sowie der Erläuterung der Aufgabenstellung beschäftigen sich die nächsten drei Kapitel mit den *wichtigsten* theoretischen Grundlagen zu „Intelligenten Agenten“, „Multi-Agenten-Systemen“ sowie dem „Agentenorientierten Software-Engineering“. *Wesentliche* Punkte, die für die weiteren Ausführungen von Bedeutung sind, werden genauer besprochen. Das Kapitel 5 befasst sich mit der Softwaremessung von agentenbasierten Systemen. Außerdem werden die Effizienz und deren Anwendung auf das Agenten-Training betrachtet. Im Anschluss daran folgt eine strukturierte Übersicht zu Lern- und Trainingsverfahren in MAS. Anhand von Beispielfällen wird aufgezeigt, wie das Training die Performance von MAS verbessern kann. Toolgestützte Formen des Lernens bilden den Schwerpunkt von Kapitel 7. Erste Ansätze zur Bewertung der Effizienz von Trainingsmethoden und –technologien sind Gegenstand des letzten Kapitels. Obwohl die beiden letztgenannten Aspekte für die praktische Umsetzung des Trainings in MAS bedeutsam sind, wurden ihnen bis dato kaum Aufmerksamkeit in der Literatur geschenkt. Die Schlussbetrachtung fasst wesentliche Aussagen der Diplomarbeit noch einmal zusammen und gibt einen Ausblick über mögliche zukünftige Entwicklungen.

## 2 Intelligente Agenten

### 2.1 Begriffsbestimmung und Abgrenzung

Wer in der Literatur nach einer einheitlichen Definition für die Begriffe „Agent“ oder „Intelligenter Agent“ sucht, wird erstaunlicherweise nicht fündig. (Wooldridge (1995)) zitieren in ihrem Artikel Carl Hewitt, der einmal gesagt hat: „... the question *what is an agent?* is embarrassing for the agent-based computing community in just the same way that the question *what is intelligence?* is embarrassing for the mainstream AI community.“ Sie führen weiter aus: „The problem is that although the term is widely used, by many people working in closely related areas, it defies attempts to produce a single universally accepted definition. This need not necessarily be a problem: after all, if many people are successfully developing interesting and useful applications, then it hardly matters that they do not agree on potentially trivial terminological details. However, there is also the danger that unless the issue is discussed, ‘agent’ might become a ‘noise’ term, subject to both abuse and misuse, to the potential confusion of the research community.“

Diese Aussage soll die Abb. 2-1 illustrieren, indem sie den Charakteristika eines Agenten die jeweiligen Einflussgebiete zuordnet.



Quelle: Brenner (1998)

**Abb. 2-1:** Einflussgebiete auf die Agenten-Technologie

Die folgende Auflistung enthält Definitionen von verschiedenen Autoren, die im Anschluss diskutiert werden.

- „An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.“ (Russel (1995))
- "Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed.“ (Maes (1995))
- „... a hardware or (more usually) software-based computer system that enjoys the following properties (Wooldridge (1995a)):
  - *autonomy*: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;
  - *social ability*: agents interact with other agents (and possibly humans) via some kind of *agent-communication language*;
  - *reactivity*: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;
  - *pro-activeness*: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour *by taking the initiative*."
- “An autonomous Agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what senses in the future.” ( Franklin (1996))

Die Definition von Russel und Norvig ist sehr allgemein gehalten. In Abhängigkeit von der Auslegung der Begriffe „Umwelt“, „Wahrnehmung“ und „Aktion“ könnte man beispielsweise auch einen Thermostat oder ein „gewöhnliches“ Computerprogramm als Agenten ansehen<sup>3</sup>. Trotzdem lässt sich festhalten, dass Agenten ein Bestandteil ihrer Umwelt sind, diese (teilweise) wahrnehmen und (bis zu einem gewissen Grad) beeinflussen können. Konkreter wird Maes. Sie fordert zusätzlich autonomes, zielgerichtetes Handeln und schränkt die Umgebung auf komplexe, dynamische Systeme<sup>4</sup> ein. Wooldridge und Jennings geben die umfassendste Erklärung. Sie beschreiben einen Agenten durch die bereits genannten Charakteristika sowie seine sozialen Fähigkeiten – einer sehr entscheidenden Eigenschaft, wie später noch deutlich wird. Die Interaktivität von Agenten bildet die Grundlage von MAS. Franklin und Graesser

---

<sup>3</sup> Abgrenzungen von Agenten zu „gewöhnlichen“ Computerprogrammen, Software-Objekten und Expertensystemen werden auf Seite 6 f. vorgenommen.

<sup>4</sup> Eine Klassifikation von Umgebungen, in denen Agenten wirken können, geben (Russel (1995)).

weisen auf einen weiteren wichtigen Punkt auf. Software-Agenten sind Programme, die kontinuierlich in einem eigenen Thread laufen.

Weiterhin kann man den o. g. Definitionen entnehmen, dass Agenten sowohl Hardware (autonome Roboter), als auch spezielle Software sein können. Diese Arbeit beschränkt sich auf Software-Agenten. Bei Ausnahmen wird darauf hingewiesen.

Als Arbeitsgrundlage für die weiteren Ausführungen soll (Dumke (2003)) dienen: „Ein *Software-Agent* (software agent) ist ein Software-System, welches innerhalb einer definierten Umgebung, dessen Bestandteil er auch ist, in einer bestimmten Zeit, mit einer ihm immanenten Verhaltensweise und einer vorgegebenen Zielstellung wirkt bzw. agiert.“ Die Definition impliziert, dass Agenten zum (relativ) autonomen Handeln in der Lage sind (können eine eigene Strategie verfolgen) und zumindest über einen gewissen Grad an Intelligenz verfügen (um die ihnen gestellten Aufgaben zu bewältigen). Des Weiteren können sie mobil sein, daher im System „umherwandern“. Das muss aber zumindest konzeptuell geplant und durch die Implementierung (Plattform) vorgesehen sein, da ein Agent stets Element seiner Umwelt bleibt.

In dieser Arbeit werden die Begriffe „Agent“ und „Intelligenter Agent“ synonym verwendet. Intelligente Agenten zeichnen sich aber streng genommen gegenüber Agenten dadurch aus (vgl. Wooldridge (1999)), dass sie zu flexiblen Aktionen in der Lage sind, um ihre Ziele zu erreichen. (Darunter wird reaktives, proaktives und soziales Verhalten verstanden.)

### **Was unterscheidet Agenten von Objekten?**

Die wichtigsten Unterschiede sieht (vgl. Wooldridge (1999)) in nachstehenden Punkten:

- Agenten besitzen (zumindest einen gewissen Grad an) Autonomie, Objekte nicht. Diese haben zwar die Kontrolle über ihren inneren Zustand (Attribute), nicht aber über Verhalten. Methoden von Objekten werden von außerhalb aufgerufen und dann ausgeführt. Agenten hingegen können selbst darüber entscheiden, ob und wann sie aktiv werden.
- Agenten können flexibel handeln, d. h. sie haben die Fähigkeit zur Reaktivität, Proaktivität und sozialem Verhalten. Das objektorientierte Modell sagt über diesen Typ von Verhalten nichts aus.
- MAS sind multithreadfähig - jeder Agent verfügt über wenigstens einen eigenen Thread.

### **Was unterscheidet Agenten von „gewöhnlichen“ Computerprogrammen?**

Dazu führen (Franklin (1996)) aus: „What about ordinary programs? A payroll program in a real world environment could be said to sense the world via its input and act on it via its output, but is not an agent because its output would not normally effect what it senses later. A payroll program also fails the "over time" test of temporal continuity. It runs once and then goes into a

coma, waiting to be called again. Most ordinary programs are ruled out by one or both of these conditions, regardless of how we stretch to define a suitable environment. All software agents are programs, but not all programs are agents.”

### Was unterscheidet Agenten von Expertensystemen?

Expertensysteme lösen Probleme, indem sie Ratschläge erteilen. Sie führen aber keine Aktionen aus. Sie interagieren auch nicht mit ihrer Umwelt. Informationen bekommen derartige Systeme nicht über Sensoren, sondern über Eingaben des Benutzers. Des Weiteren können Expertensysteme nicht wirklich mit anderen Agenten (Expertensystemen) kooperieren (vgl. Wooldridge (1999)).

## 2.2 Klassifizierung

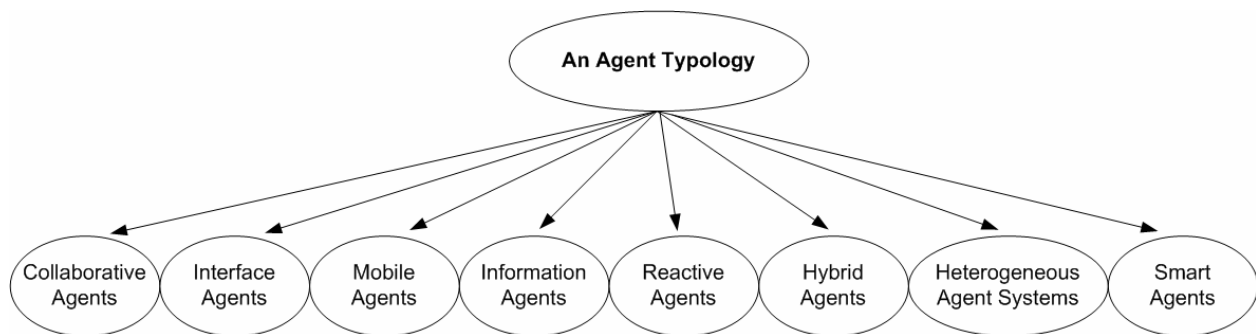
Agenten lassen sich auf verschiedene Art und Weise einteilen, z. B. anhand ihrer Fähigkeiten. Die Tab. 2-1 listet mögliche Attribute von Agenten auf, ohne jedoch den Anspruch auf Vollständigkeit zu erheben. Im Allgemeinen haben Agenten mehrere der aufgeführten Eigenschaften. Mobile lernende Agenten bilden etwa eine Subklasse der mobilen Software-Agenten.

**Tab. 2-1:** Eigenschaften von Software-Agenten

Eigenschaft	Beschreibung
reaktive	reagieren auf Umweltveränderungen in angemessener Zeit
autonome	haben Kontrolle über ihren inneren Zustand sowie ihr Verhalten
zielorientierte	verfolgen bestimmte (vorgegebene) Ziele
zeitlich fortlaufende	sind in ununterbrochener Nutzung
kommunikative	kommunizieren mit anderen Agenten, Nutzern
lernende (adaptive)	ermöglichen erfahrungsbasierte Verhaltensänderungen
mobile	können sich über mehrere Rechner hinweg bewegen
flexible	Variabilität bzgl. der auszuführenden Aktionen, kein festgelegter Ablauf
charakterbezogene	Glaubwürdigkeit und emotionaler Status

Quelle: vgl. Franklin (1996)

Eine weitere gängige Untergliederung stellt (Nwana (1996)) in ihrer Arbeit vor. Sie fasst jeweils mehrere Eigenschaftsdimensionen zu einer Klasse zusammen und schließt in ihre Betrachtung potentielle Anwendungsgebiete derselben mit ein (siehe Abb. 2-2).



Quelle: Nwana (1996)

**Abb. 2-2:** A Classification of Software Agents

### 2.3 Architekturen

Es gibt zwei grundsätzliche Herangehensweisen bei der Konstruktion von Agenten. Die „kognitive“ Schule sieht Agenten als intelligente Individuen, die über das notwendige Wissen verfügen, das sie zur Lösung ihrer Aufgaben und zur Interaktion mit anderen Agenten befähigt (siehe Abschnitt 2.3.1). Kognitive Agenten verfolgen Pläne, um ihre Ziele zu erreichen. Oft kooperieren sie in kleinen Gruppen, ähnlich der menschlichen Teamarbeit (soziologischer Aspekt). Die Befürworter der „reaktiven“ Perspektive vertreten dagegen die Meinung, dass Agenten nicht notwendigerweise intelligent sein müssen, damit das Gesamtsystem ein intelligentes Verhalten zeigt. Dieses entsteht „emergent“ durch eine ständige Interaktion von vielen relativ einfach gebauten Einheiten mit ihrer Umwelt (siehe Abschnitt 2.3.2). Vorbilder sind z. B. Insektenstaaten (biologischer Aspekt). Schließlich wird auch versucht, die Vorteile beider Sichtweisen zu vereinigen. Derartige Agenten werden als hybrid bezeichnet (siehe Abschnitt 2.3.3). Für jede Gruppe von Architekturen wird ein repräsentativer Vertreter vorgestellt<sup>5</sup>. In der Literatur sind weitere, von der eingeführten Systematik mehr oder weniger abweichende, Einteilungen zu finden, beispielsweise in (vgl. Wooldridge (1999), Müller (1996)).

<sup>5</sup> Die Übergänge sind in der Praxis oft fließend.

### 2.3.1 Kognitive Agenten

Kognitive Agenten firmieren oft auch unter der Bezeichnung „Deliberative Agenten“<sup>6</sup>. Die BDI-Architektur (Belief-Desire-Intention) wird im Folgenden genauer betrachtet. Sie wurde von (Rao (1991, 1995)) entwickelt (siehe Abb. 2-3).

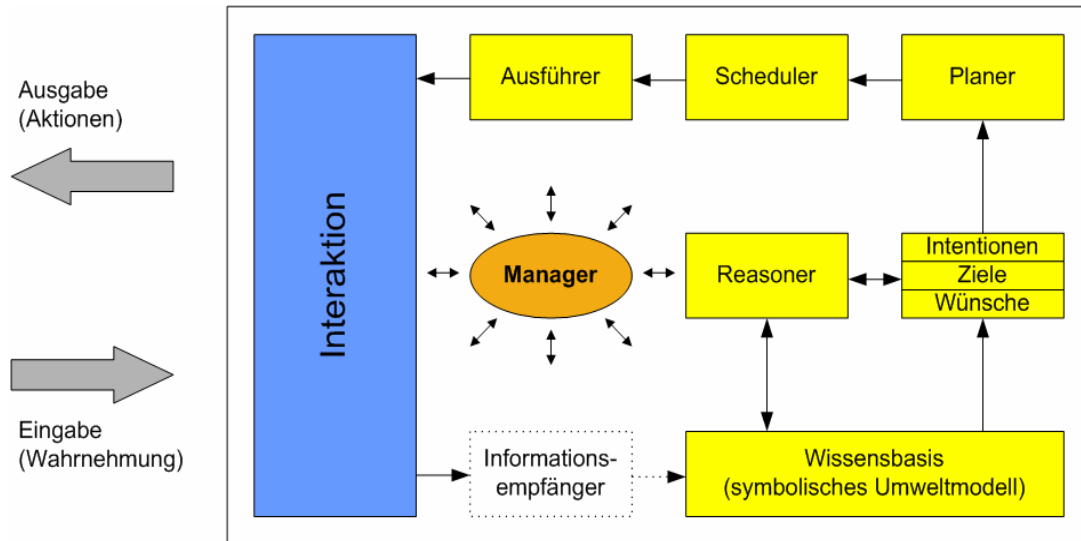
Die Grundidee von BDI-Agenten entspringt der philosophischen Tradition des Verständnisses für „Praktisches Schlussfolgern“. Darunter ist der inkrementelle Prozess der Auswahl von Aktionen zu verstehen, die dem Erreichen der eigenen Ziele dienen. Er setzt sich aus zwei wesentlichen Bestandteilen zusammen, der Entscheidung, *welche* Ziele zu verfolgen und *wie* diese zu erreichen sind. Unbewusst wenden Menschen diese Art zu Denken täglich an (vgl. Wooldridge (1999)).

Kennzeichnende Merkmale von BDI-Agenten sind das interne symbolische Weltmodell und die Fähigkeit zum logischen Schlussfolgern. Die Modellierung der Umwelt geschieht in der Regel vorab und stellt den Kern der Wissensbasis dar. Der Schlussfolgerungsprozess verwendet das vorhandene Wissen, um den mentalen Zustand des Agenten zu verändern (modifizieren). Dieser setzt sich aus fünf Komponenten zusammen:

- *Überzeugungen (Beliefs)*: stellen die grundlegenden Ansichten eines Agenten bezüglich seiner Umwelt dar. Mit ihrer Hilfe werden Erwartungen über mögliche zukünftige Umweltzustände beschrieben.
- *Wünsche (Desires)*: folgen direkt aus den Überzeugungen und enthalten subjektive Beurteilungen zukünftiger Situationen. Ein Agent könnte z. B. den Wunsch besitzen, dass ein bestimmter Zustand eintritt und ein anderer wiederum nicht. Es werden aber noch keine Aussagen darüber getroffen, inwieweit die Wünsche realistisch oder widerspruchsfrei sind.
- *Ziele (Goals)*: sind die Wünsche eines Agenten, welche er potentiell umsetzen kann. Sie sollten nicht in Konflikt zueinander stehen.
- *Absichten (Intentions)*: sind eine Untermenge aller Ziele. Beschließt ein Agent ein bestimmtes Ziel zu verfolgen, spricht man von einer (festen) Absicht. (Es können aufgrund der beschränkten Ressourcen eines Agenten nicht alle Ziele gleichzeitig bearbeitet werden.)
- *Pläne (Plans)*: Die zur Realisierung der Intentionen notwendigen Schritte werden in Plänen zusammengefasst.

---

<sup>6</sup> Genau genommen gibt es konzeptuelle Unterschiede zwischen BDI- und deliberativen Agenten, auf welche an dieser Stelle aber nicht eingegangen werden soll.



Quelle: vgl. Brenner (1998)

**Abb. 2-3:** Architektur eines kognitiven Agenten

Funktionsweise des BDI-Agenten: Der Agent leitet auf der Basis seines aktuellen Wissens über die Umwelt eine Menge von Beliefs, Goals und Intentions ab. Diese Arbeit übernimmt der Reasoner (Für BDI-Agenten wurde eine spezielle Art von formaler Logik entwickelt.). Dem Planer kommt die Aufgabe zu, einen konsistenten Gesamtplan zusammenzustellen. Der Scheduler entscheidet dann zur Laufzeit, welche konkrete Aktion durch die Ausführungskomponente realisiert wird.

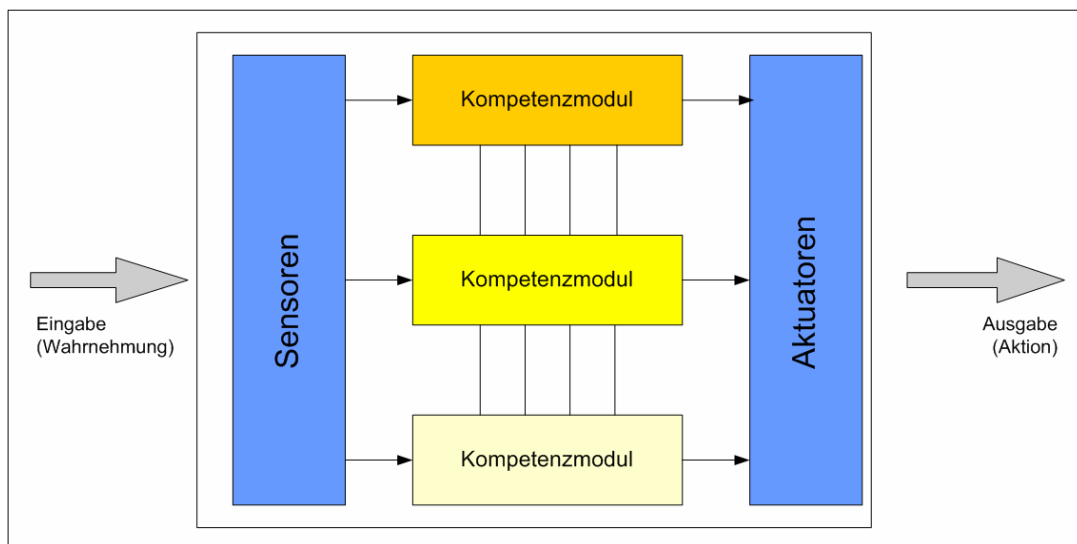
Die Interaktion mit anderen Agenten ist vor allem auf Kommunikation und Kooperation ausgerichtet und kann auf einem sehr hohen abstrakten Niveau, z. B. bei automatisierten Verhandlungen oder einem Wissensaustausch, durchgeführt werden. Die Wahrnehmung der Umwelt dagegen beeinflusst den Agenten in wesentlich geringerem Maße. Sensordaten werden nur für kurzfristige Entscheidungen herangezogen und verändern selten das „Langzeitgedächtnis“ (Regeln, Wissensbasis; siehe auch Abschnitte 6.1.1 und 6.2.2).

Obwohl kognitive Agenten über einige unbestrittene Vorteile verfügen (Lösen komplexer Aufgabenstellungen, vielfältige Kooperationsformen auf hohem Level), ziehen sie jedoch aufgrund ihrer Komplexität einige Probleme nach sich. In dynamischen Umgebungen sollten Entscheidungen auf Grundlage aktueller Informationen getroffen werden. Das ist aber nur eingeschränkt möglich, da die Wissensbasis des Agenten in der Vergangenheit entworfen und zur Laufzeit nur marginal angepasst wird. Des Weiteren sind die Vorgänge zum Entwickeln und Ausführen von Plänen sehr rechenintensiv (zeitintensiv). Oft haben sich die Rahmenbedingungen bereits wieder geändert, bevor eine geplante Aktion überhaupt zur Ausführung kommt. Zudem besitzt ein Agent nur selten alle zum optimalen Planen notwendigen Informationen (vgl. Brenner (1998)).

### 2.3.2 Reaktive Agenten

Einen anderen Ansatz verfolgen reaktive Agenten. Ihre Architektur ist im Vergleich zu der von kognitiven Agenten einfacher gestaltet. Im Wesentlichen handelt es sich um Stimulus-Response-Systeme. Wahrgenommene Reize aus der Umgebung werden mehr oder weniger direkt auf Aktionen abgebildet, z. B. durch einfache Regeln<sup>7</sup>. Eine interne symbolische Darstellung der Umwelt (Modell) fehlt diesen Agenten genau so wie die Fähigkeit zu komplexen Schlussfolgerungen. Trotzdem kann ein durch sie gebildetes Gesamtsystem intelligentes Verhalten hervorbringen, man denke beispielsweise an biologische Verbände („Schwarmintelligenz“). Ein bekannter Vertreter der reaktiven Architektur ist die „Subsumtion Architecture“ (vgl. Brooks (1986, 1990, 1991)).

Ein Agent (autonomer mobiler Roboter) nach Brooks besteht aus einer Reihe von *hierarchisch* angeordneten Kompetenzmodulen (siehe Abb. 2-4). Jedes von ihnen ist für genau eine Verhaltensweise verantwortlich. Es wird also eine Dekomposition nach Aktivitäten vorgenommen, z. B. in Zusammenstöße vermeiden, Umwelt erkunden oder Bodenproben sammeln<sup>8</sup>. Dabei nimmt die Komplexität des Verhaltens von unten nach oben zu. Die in der Hierarchie weiter oben angesiedelten Module können die unter ihnen liegenden durch Unterdrückung des jeweiligen Ein- oder Ausgangssignals kontrollieren.



Quelle: vgl. Brenner (1998), in Anlehnung an Brooks (1986)

**Abb. 2-4:** Architektur reaktiver Agenten

<sup>7</sup> Ein allgemeines Beispiel: Wenn Situation x eintritt, dann wähle Aktion y – ansonsten Aktion z.

<sup>8</sup> Bei kognitiven Agenten geht man nach funktionalen Gesichtspunkten vor (Wissensbasis, Reasoner, Planer, usw.).

Funktionsweise des reaktiven Agenten: Die Reize aus der Umwelt werden über eine Vielzahl von Sensoren aufgenommen und an die entsprechenden Kompetenzmodule weitergegeben, wo eine parallele Verarbeitung der Daten (im Rohformat) stattfindet. Über Aktuatoren kann der Agent auf seine Umgebung einwirken.

Die Interaktion zwischen reaktiven Agenten basiert auf einfachen Mechanismen. Eine Agenten-Kommunikationssprache (ACL), wie bei kognitiven Agenten üblich, wird dabei nicht verwendet (siehe auch Abschnitt 3.4).

Die relativ einfache Bauweise hat einige Vorteile. Das Fehlen zentraler Strukturen macht den Agenten beispielsweise robuster. Der Ausfall einer Komponente bedeutet dann nicht zwangsläufig den Ausfall des gesamten Systems.<sup>9</sup> Weiterhin ermöglichen der intensive Kontakt mit der Umgebung sowie die schnelle Informationsverarbeitung dem Agenten eine rasche Anpassung an Umweltveränderungen (vgl. Brenner (1998)). Daneben gibt es natürlich auch Nachteile (vgl. Wooldridge (1999)). Reaktive Agenten benötigen auf Grund des fehlenden Weltmodells die „richtigen“ Informationen zur „richtigen“ Zeit. Entscheidungen, die momentan (auf Basis derzeitiger „lokaler“ Informationen) sinnvoll erscheinen, erweisen sich eventuell langfristig als wenig hilfreich (keine Planung). Die Fähigkeit zum Lernen lässt sich in *rein* reaktive Agenten kaum integrieren, da sie lediglich über ein „Kurzzeitgedächtnis“ verfügen. Des Weiteren wird ihr Verhalten durch eine Vielzahl von Bedingungen (Interaktionen, Umwelt) beeinflusst. Dadurch wird es kompliziert, ein reaktives System für eine spezifische Aufgabe zu entwerfen (bzw. dessen Reaktionen vorauszusagen). Komplexe Verhaltensweisen sind aufgrund der vielfältigen Abhängigkeiten zwischen den Kompetenzmodulen schwierig nachzubilden.

### 2.3.3 Hybride Agenten

Hybride Agenten weisen üblicherweise eine Schichtenarchitektur (layered architecture) auf. Dabei implementieren die unteren Schichten grundlegende (reaktive) Verhaltensweisen, während weiter höher liegende für komplexere (kognitive) zuständig sind. Auf welche dabei der Schwerpunkt gelegt wird, liegt im Ermessen des Entwicklers. Je nachdem, wie der Informations- und Kontrollfluss durch die Komponenten erfolgt, spricht man von horizontaler oder vertikaler Schichtung. Die INTERRAP-Architektur (INTEgration of Reactive behavior and RAtional Planning) von (Müller (1996), siehe Abb. 2-5) besitzt die letztere Struktur.

Die drei vertikalen Layer (verhaltensbasierte [reaktive] Schicht, lokale [kognitive] Planungsschicht, kooperative [für Interaktionen in einem MAS zuständige] Planungsschicht)

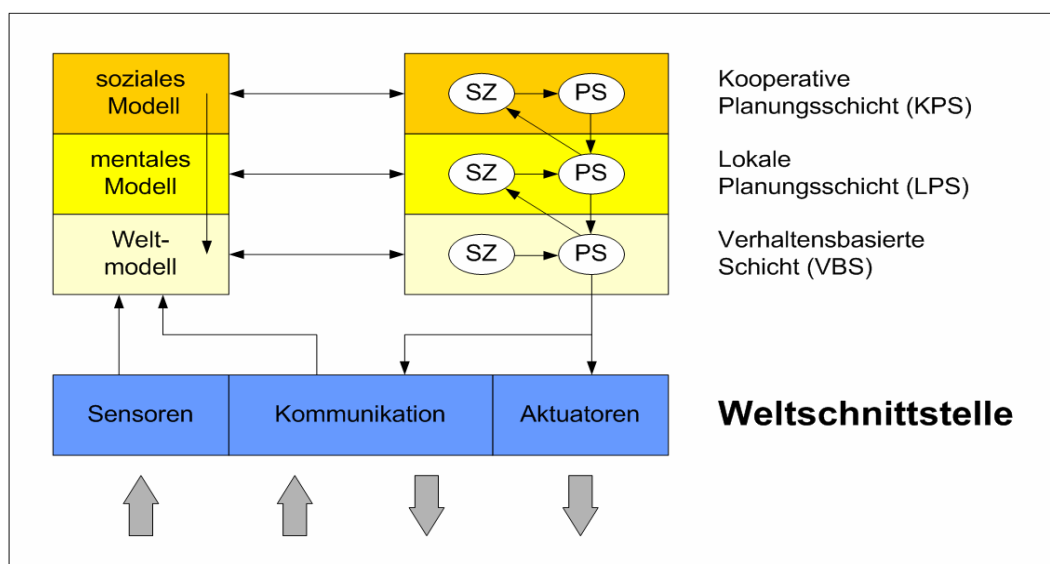
---

<sup>9</sup> Bei reaktiven MAS wird dieser Vorteil noch augenscheinlicher. Durch die große Anzahl von Agenten sind oft Redundanzen hinsichtlich der Funktionalität vorhanden. Der Ausfall einer Einheit fällt dann gewöhnlich nicht sehr ins Gewicht.

bestehen aus jeweils einer Kontrolleinheit und einer Wissensbasis. Der INTERRAP-Agent baut auf dem BDI-Agenten auf.

Das Wissen des Agenten wird in drei Modelle untergliedert. Das Weltmodell enthält seine Überzeugungen in Bezug auf die Umwelt, während das mentale den inneren Zustand des Agenten widerspiegelt und das soziale für die Beschreibung anderer Agenten im MAS zuständig ist. Aus seinen Überzeugungen leitet der Agent so genannte Situationen her. Sie stellen diejenigen Umweltzustände dar, welche für ihn von konkretem Interesse sind. Man unterscheidet in Routine- bzw. Notfallsituationen, lokale oder kooperative Situationen (in Anlehnung an die Bezeichnung der Schichten; gleiches gilt für die Ziele). Das Eintreten einer bestimmten Situation aktiviert im Allgemeinen ein oder mehrere Ziele (Optionen). Mittels operationaler Primitive<sup>10</sup>, welche je nach Schicht unterschiedlich ausgestaltet sind<sup>11</sup>, wählt der Agent zwischen den Optionen aus und legt seine Intentionen fest.

Funktionsweise eines INTERRAP-Agenten: Das Situations-/Zielerkennungsmodul der verhaltensbasierten Schicht erstellt auf Basis der verfügbaren Informationen (Weltschnittstelle, Weltmodell) Optionen, sofern es mit dem eingetretenen Umweltzustand umgehen kann. Danach werden mit Hilfe des Planungs-/Schedulingmoduls Intentionen abgeleitet und das Scheduling ausgeführt. Im Fall der „Überforderung“ der verhaltensbasierten Schicht wird die Kontrolle an die darüber liegende lokale Planungsschicht abgegeben, welche über mehr Wissen verfügt. Sollte auch diese nicht angemessen reagieren können, muss die kooperative Planungsschicht die Situation auflösen (vgl. Brenner (1998)).



Quelle: vgl. Brenner (1998)

**Abb. 2-5:** Konzeptuelles INTERRAP-Agentenmodell

<sup>10</sup> Operationale Primitive entsprechen in etwa einem Schlussfolgerungsprozess.

<sup>11</sup> Reaktives Verhalten (schnelles Reagieren) wird z. B. durch fest hinterlegte Verhaltensmuster (Regeln) erreicht.

### 3 Multi-Agenten-Systeme

Zu den Gründen, warum man sich mit MAS befasst, wurden bereits Ausführungen in der Einleitung gemacht. (Vgl. Weiss (1999a)) führt folgende Punkte als Motivation an:

- Technologische und anwendungsbezogene Bedürfnisse/Anforderungen: MAS bieten einen viel versprechenden und innovativen Weg, um verteilte, komplexe, dynamische, offene und heterogene Computer- und Informationssysteme zu verstehen, zu managen und zu nutzen. (Als Beispiel wird das Internet herangezogen.)
- Natürliche Sicht auf „Intelligente Systeme“: MAS bieten einen natürlichen Weg, um „Intelligente Systeme“ zu beschreiben. Intelligenz und Interaktion sind eng (im realen Leben [Beispiel Mensch]) miteinander verbunden. MAS bauen auf diesem Prinzip auf.

#### 3.1 Begriffsbestimmung

Für diese Arbeit wird eine Definition von (Ferber (2001)) verwendet. „Der Begriff Multiagentensystem (MAS) bezeichnet ein System, das aus folgenden Elementen besteht:

- Eine Umwelt  $E$ .  $E$  ist ein Raum, der im Allgemeinen ein Volumen hat.
- Eine Menge von Objekten,  $O$ . Diese sind situiert, das bedeutet, dass zu einem beliebigen Zeitpunkt jedem Objekt eine Position in  $E$  zugewiesen werden kann. Objekte können von Agenten wahrgenommen, erzeugt, modifiziert und gelöscht werden.
- Eine Menge von Agenten,  $A$ . Diese repräsentieren die aktiven Objekte ( $A \subseteq O$ ) des Systems.
- Eine Menge von Beziehungen,  $R$ , die Objekte miteinander verbinden.
- Eine Menge von Operationen,  $Op$ , damit Agenten Objekte empfangen, erzeugen, konsumieren, verändern und löschen können.
- Operatoren mit der Aufgabe, die Anwendung dieser Operationen und die Reaktion der Umwelt auf die entsprechende Veränderungsversuche darzustellen. Wir bezeichnen diese Operatoren als die Gesetze des Universums.“

Die folgenden Eigenschaften ordnen (vgl. Jennings (1998)) einem MAS zu:

- Jeder Agent besitzt nur unvollständige Informationen oder Fähigkeiten zur Problemlösung. Deshalb hat jeder Agent lediglich eine eingeschränkte Sicht auf das MAS.
- Es gibt keine zentrale Kontrollkomponente (Steuereinheit).
- Die Datenhaltung erfolgt dezentral, d. h. jeder Agent verfügt über seine eigene Datenbasis.
- Die Berechnungen der Agenten erfolgen asynchron. (Jeder Agent verfügt über einen eigenen Thread. Dadurch wird eine parallele Ausführung möglich.)

### 3.2 Klassifizierung

Zur Klassifizierung von MAS können, wie bei den Intelligenten Agenten, deren Eigenschaften herangezogen werden. Die Tab. 3-1 gibt einen Überblick über die wichtigsten Charakteristika von MAS.

**Tab. 3-1:** Eigenschaften von MAS

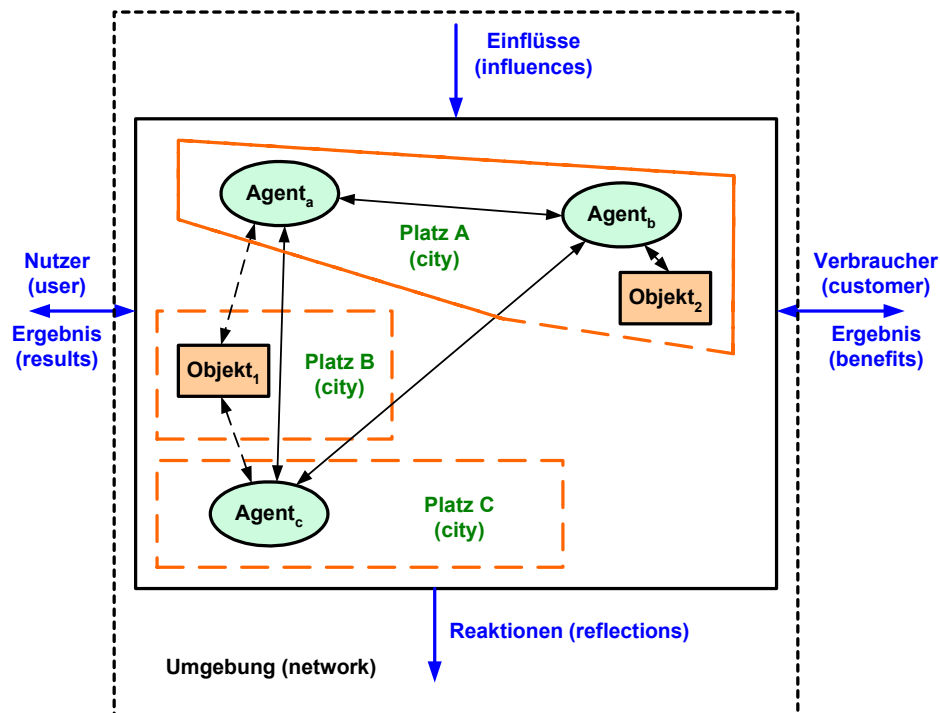
Eigenschaft	Ausprägung
Anzahl der Agenten	viele vs. wenig / fest vs. dynamisch
Granularität	Gibt es wenige komplexe oder viele einfache Agenten?
Homogenität	Sind die Agenten gleichartig oder verschieden?
Organisation	ohne Hierarchie, gleichberechtigt, hierarchisch / fest vs. dynamisch
Kooperationsbereitschaft	Haben eigene Ziele oder die der anderen Agenten den Vorrang?
Zielorientierung der Agenten	Verfolgen alle Agenten ein Ziel? / fest vs. dynamisch
Kommunikationslevel	niedrig vs. hoch (bezogen auf das Abstraktionsniveau)
Grad der Kopplung	lose (keine Zusammenarbeit) vs. eng (Problemlösen im Team)

Quelle: vgl. Klügl (2004)

Daneben gibt es wiederum andere Möglichkeiten, MAS zu unterteilen. (Doran (1997)) ziehen beispielsweise als Kriterium die Kooperationsform heran. Die Unterscheidung in „Verteilte Problemlöser“ (Alle Agenten verfolgen ein Ziel.) und MAS im engeren Sinne (Es ist keine Zielorientierung für die Agenten vorgeschrieben.) wird nicht mehr verwendet.

### 3.3 Architekturen

Aufgrund fehlender Standards existiert ein breites Spektrum an MAS-Architekturen<sup>12</sup>. In letzter Zeit bemühen sich verstärkt einige Organisationen, z. B. die Foundation of Intelligent Physical Agents (FIPA, <http://www.fipa.org/>) oder die Object Management Group (OMG, <http://www.omg.org/>), diesen Misstand zu beheben (siehe Abschnitt 4.3). Deshalb wird an dieser Stelle ein allgemeines Modell eines MAS präsentiert (siehe Abb. 3-1).



Quelle: Dumke (2003)

**Abb. 3-1:** Komponenten eines Multi-Agenten-Systems

Das abgebildete MAS arbeitet in einer Netzumgebung wie beispielsweise dem WWW. Ein Knoten (Host) wird als Platz oder City bezeichnet und kann eine (beliebige) Anzahl von Agenten aufnehmen. In City B sind in diesem Beispiel nur Objekte O (passive Verarbeitungseinheiten) vorhanden, in City C dagegen nur Agenten A (aktive Objekte).

Die folgenden Abschnitte des Kapitels behandeln Schlüsselkonzepte von MAS.

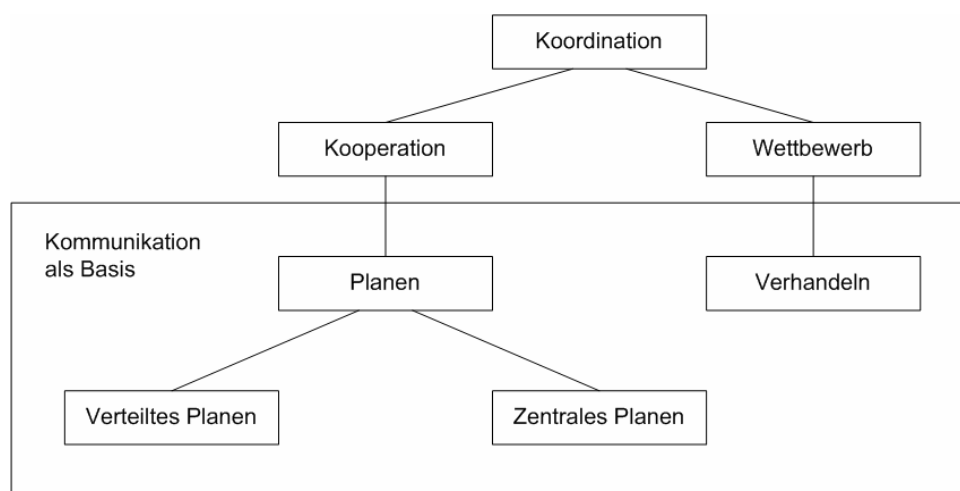
<sup>12</sup> (Wille (2002a)) geben einen guten Überblick.

### 3.4 Interaktion und Agenten-Koordination

Die Interaktion ist ein zentrales Element in MAS. „Eine Interaktion findet statt, wenn zwei oder mehrere Agenten durch eine Anzahl wechselseitiger Aktionen in eine dynamische Beziehung zueinander treten. Interaktionen entstehen also aus einer Reihe von Aktionen, deren Konsequenzen wiederum Einfluss auf das zukünftige Verhalten der Agenten haben.“ (Ferber 2001)).

Es gibt verschiedene Interaktionsarten (vgl. Ferber (2001)). Agenten interagieren beispielsweise beim Wissensaustausch, dem Zugriff auf gemeinsame (knappe) Ressourcen oder bei der Kooperation in Gruppen. Die *Koordination* dient dabei als Mechanismus, um Interaktionssituationen zu bewältigen. Dazu (Sen (1998)): „In a world inhabited by multiple agents, coordination is a key to group as well as individual success. We need to coordinate our actions whenever we are sharing goals, resources, or expertise. By coordination we mean choosing one’s own action based on the expectation of others’ actions. Coordination is essential for cooperative, indifferent, and even adversarial agents.“ (Jennings (2003)) führen aus: „Effective coordination is essential if autonomous agents are to achieve their goals in a multiagent system (MAS). Such coordination is required to manage the various forms of dependency that naturally occur when agents have inter-linked objectives, when they share a common environment or when there are shared resources.“.

Die Literatur hält ein breites Spektrum von Ansätzen parat, wie das Zusammenwirken von Agenten in MAS organisiert werden kann (siehe Abb. 3-2).



Quelle: Huhns (1999)

**Abb. 3-2:** Eine Taxonomie möglicher Koordinationsformen

Man unterscheidet in zwei große Gruppen von Strategien - Kooperation und Wettbewerb. Erstere kommt zum Tragen, wenn die Agenten ein gemeinsames Ziel verfolgen (z. B. beim

verteilten Lösen von Problemen). Haben Agenten dagegen inkompatible Ziele, treten sie in Wettbewerb zueinander (z. B. bei knappen Ressourcen) und benötigen Konfliktlösungsstrategien. Für eine effiziente Koordination werden Protokolle benötigt, die den Rahmen für das Verhalten der Agenten festlegen. (Einige behandeln die Abschnitte 3.4.1 und 3.4.2. Diese werden durch die in Kapitel 6 beschriebenen Beispiele für das Lernen und Training in MAS verwendet.)

Die den Agenten zur Verfügung stehenden Koordinationsmittel sind abhängig von ihrer Architektur. *Rein reaktive* Agenten haben aufgrund der geringeren kommunikativen Fähigkeiten und der nicht vorhandenen internen Repräsentation anderer Agenten (Stichworte: symbolisches Weltmodell, Schlussfolgern, Planen) nur eingeschränkte Möglichkeiten. Sie nutzen z. B. Potentialfelder und Markierungen, um eine Aktionskoordination zu erreichen (vgl. Ferber (1999)).

Kennzeichnend für die Koordination in MAS ist das Fehlen einer zentralen Steuer- oder Kontrollkomponente (siehe auch Abschnitt 3.1). „Distributed control means that agents have degree of autonomy of generating new actions and deciding which goals to pursue next, disadvantage is that system’s overall state is dispersed throughout the system and each agent has only a partial and imprecise perspective.“ (Huhns (1999)).

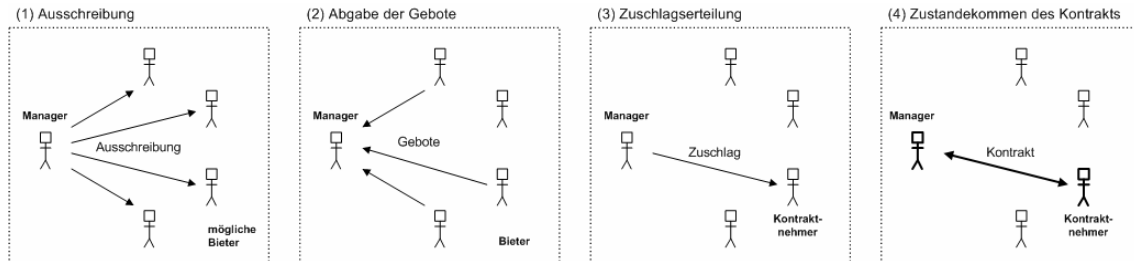
### **3.4.1 Kooperationsprotokolle**

Kooperationsprotokolle verfolgen oft die Strategie, eine Aufgabe in mehrere kleinere zu zerlegen und dann zu verteilen. Die Vorteile eines solchen „Herrsche-und-Teile“-Ansatzes liegen in einer Verringerung der Komplexität der Teilprobleme. Damit sinken auch die Ansprüche an die beteiligten Agenten und deren Ressourcen. Im Gegenzug führt diese Herangehensweise zu einem erhöhten Koordinationsaufwand. Die Dekomposition von Aufgaben kann durch den Entwickler oder die Agenten vorgenommen werden bzw. ist ein Bestandteil der Repräsentation des Problems selbst (vgl. Huhns (1999)).

#### **Kontraktnetz-Systeme**

Kontraktnetze bestehen aus einer gewissen Anzahl von Knoten, die den beteiligten Agenten entsprechen. Das Verfahren nutzt marktähnliche Mechanismen (siehe Abb. 3-3). In einem öffentlichen Ausschreibungsverfahren suchen Agenten („Manager“) andere Agenten („Bieter“), die für sie ein Teilproblem bearbeiten. Diese potentiellen „Vertragspartner“ können sich um die ausgeschriebenen Kontrakte bewerben. Der Manager wählt dann denjenigen mit den (seiner Meinung nach) „besten“ Fähigkeiten aus und übersendet diesem die Problemspezifikation. Nach Erledigung der Aufgabe werden die Ergebnisse zurück an den Manager geschickt. Die Rollenverteilung in einem Kontraktnetz ist dynamisch, d. h. Agenten können sowohl Manager

als auch Vertragspartner sein. Stellt ein Agent z. B. nachträglich fest, dass er die ihm übertragene Aufgabe nicht lösen kann, steht es ihm frei, ebenfalls eine Ausschreibung vorzunehmen.



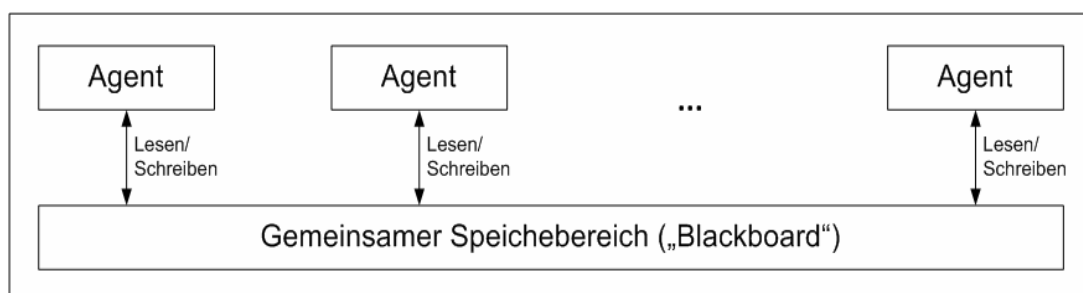
Quelle: vgl. Ferber (2001)

**Abb. 3-3:** Die Schritte im Kontraktnetzprotokoll

Grundlage des Kontraktnetz-Systems ist ein gemeinsames Nachrichtenformat (Common Internode Language, Kontraktnetz-Protokoll), welches von allen Agenten beherrscht werden muss. Die Vorteile des Systems liegen in seiner leichten Verständlichkeit und Umsetzung. Jedoch belastet die große Anzahl an ausgetauschten Nachrichten sowie die Rechenlast für die Prüfung von Ausschreibungen und Angeboten die Infrastruktur des MAS stark (vgl. Brenner (1998)).

### Blackboard-Systeme

Derartige Protokolle nutzen einen gemeinsamen Speicherbereich, das Blackboard. Über dieses kann jegliche Art von Informationen ausgetauscht werden (siehe Abb. 3-4). Allerdings ist zur korrekten Interpretation eine gemeinsame Sprache/Ontologie notwendig. Das Blackboard informiert die angemeldeten Agenten über neue Einträge bzw. Updates. Die neueren Versionen verfügen mittlerweile über Moderatoren, welche die Organisation des Problemlösungsprozesses und die Qualitätsprüfungen von Beiträgen übernehmen.



Quelle: vgl. Brenner (1998)

**Abb. 3-4:** Struktur eines Blackboard-Systems

Das Verfahren ist flexibel und stellt nur geringe Anforderungen an die beteiligten Agenten. Die Nachteile liegen in der zentralen Struktur von Blackboards begründet: Das Lesen und Schreiben an zentraler Stelle führt zu einer hohen Rechner- und Netzwerkbelastung (vgl. Brenner (1998)).

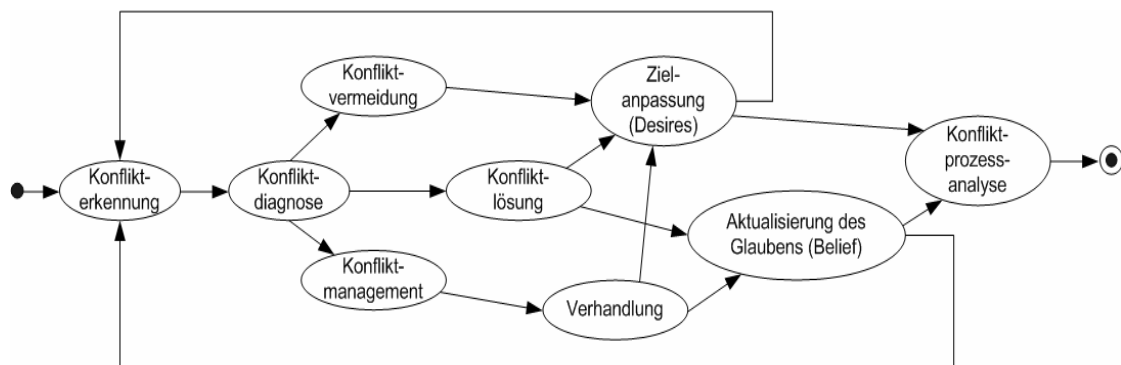
### 3.4.2 Protokolle für den Wettbewerb

Bedingt durch unterschiedliche Ziele der Agenten können Interessenskonflikte (z. B. bei Nutzung gemeinsamer Ressourcen) entstehen. Um ein möglichst effektives Zusammenwirken innerhalb eines MAS zu gewährleisten, müssen diese aufgelöst werden. Die Abb. 3-5 zeigt ein Beispiel für die Modellierung einer Konfliktlösung anhand eines Zustandsdiagramms auf Grundlage des BDI-Modells.

#### Verhandlungen

Verhandlungen werden oft in MAS genutzt, um Konflikte zu bewältigen. Voraussetzungen sind:

- Eine gemeinsame Sprache der Verhandlungspartner
- Ein Verhandlungsprotokoll
- Ein Entscheidungsprozess für jeden Agenten, damit dieser seine Position, Zugeständnisse und Vereinbarungen festlegen kann.



Quelle: vgl. Knapik (1998)

**Abb. 3-5:** Konfliktbehandlung zwischen Software-Agenten

Verhandlungen verlaufen meistens in mehreren Runden. Zu Beginn teilen die Agenten sich ihre Standpunkte mit. Durch (gegenseitige) Zugeständnisse und die Einbeziehung von möglichen Handlungsalternativen wird versucht, zu einer Übereinkunft zu gelangen (vgl. Huhns (1999)).

## Märkte

Wenn die Anzahl der Agenten größer wird oder unbekannt ist, lässt sich eine direkte Kommunikation zwischen ihnen, wie beispielsweise bei Verhandlungen, nur schwer realisieren. Auf virtuellen Märkten müssen Agenten nur sehr wenige Informationen *direkt* austauschen. Alle für die Koordination relevanten Faktoren (materielle und immaterielle Güter) werden mit einem Preis belegt, der über Angebot und Nachfrage beeinflusst wird. Man unterstellt den Agenten rationales Handeln, beispielsweise die Maximierung ihres eigenen Profits oder Nutzens. Über die gängigen Marktmechanismen wird das Problem der Ressourcenallokation gelöst. Es gibt zwei Typen von Agenten, Konsumenten (tauschen Güter) und Produzenten (verfügen über die Technologie der Transformation eines Gutes in ein anderes). Folgende Punkte müssen a priori spezifiziert werden:

- die zu handelnden Güter
- die Konsumenten, welche die Güter handeln
- die Produzenten, die die gewünschte Transformationen vornehmen können
- Verhaltensprotokolle für das Bieten und Handeln

(vgl. Huhns (1999), Wellmann (1995)).

### 3.5 Agenten-Kommunikation

Die Kommunikation stellt die Basis für Interaktionen und soziale Organisationen von Agenten dar. Sie ist das verbindende Element in MAS. Voraussetzungen für eine erfolgreiche Kommunikation sind (Klügl (2004)):

- Technische Voraussetzungen an Übertragungskanälen und Übertragungsprotokollen
- Eine Kommunikationssprache in Syntax und Semantik
- Eine Ontologie (gemeinsame Begriffsmenge, gemeinsame Semantik für diese Begriffe)

Die Infrastruktur für die Agenten-Kommunikation wird in den meisten Fällen durch die Agenten-Plattform bereitgestellt (siehe Abschnitt 4.3)<sup>13</sup>. Durch die Verwendung einer ACL legt man Format, Syntax und Semantik des Kommunikationsprotokolls (domänenunabhängig) fest. Sie dient gewissermaßen als Wrapper für den tatsächlichen Inhalt der Nachricht. Dieser wird in einer Wissensrepräsentations- bzw. -austauschsprache verschlüsselt. Des Weiteren benötigt man eine gemeinsame Ontologie, damit die Agenten den Begriffen ihrer Domäne auch eine

---

<sup>13</sup> Für den physischen Transport von Nachrichten nutzt man Protokolle wie HTTP (Hypertext Transfer Protocol), TCP/IP (Transmission Control Protocol/Internet Protocol) oder IIOP (Internet Inter ORB Protocol).

übereinstimmende Bedeutung zumessen (domänenabhängig). Neben den nachrichtenbasierten Verfahren werden auch die in Abschnitt 3.4.1 vorgestellten Blackboard-Systeme eingesetzt.<sup>14</sup> Im Folgenden wird auf die o. g. Voraussetzungen (außer Infrastruktur/Transportprotokolle) näher eingegangen.

## KQML

Die Knowledge Query and Manipulation Language (KQML) ist eine ACL und definiert sowohl ein Nachrichtenformat als auch ein Nachrichtenübermittlungssystem (Kommunikationsprotokoll). Sie basiert auf der Sprechakt-Theorie von (Austin (1962)). Diese hat die menschliche Sprache als Vorbild. Eine übermittelte Nachricht enthält demzufolge nicht nur einfache Aussagen, sondern auch implizit die Aufforderung zu einem bestimmten Handeln (Sprechakt)<sup>15</sup>. KQML-Performative<sup>16</sup> versuchen das nachzubilden. Entwickelt wurde die Sprache im Rahmen des amerikanischen Knowledge Sharing Efforts (KSE) an der Maryland-Universität (vgl. Brenner (1998)).

Eine KQML-Nachricht besteht aus drei wesentlichen Bestandteilen:

- Kommunikationsebene: Parameter wie Sender, Empfänger und Nachrichten-Id
- Nachrichtenebene: legt den Sprechakt-Typ fest (z. B. ask-one, tell)
- Inhaltsebene: spezifiziert den Inhalt der Nachricht (content, language, ontology)

```
(KQML-performative
  :sender          <word>
  :receiver       <word>
  :language       <word>
  :ontology       <word>
  :content        <expression>
  ... )
```

Das Performativ entspricht einem Sprechakttyp (z. B. ask-one: Agent A möchte eine Antwort von Agent B auf eine bestimmte Frage). Sender und Empfänger beinhalten die Kennungen der involvierten Agenten. Die Nachricht selbst findet sich im Feld „content“ wieder. Die zur Darstellung des Inhalts verwendete Sprache wird in „language“ spezifiziert. Welche Ontologie zur Anwendung kommt, bestimmt „ontology“.

---

<sup>14</sup> Der einfachste Fall der Agenten-Kommunikation, der Methoden-Aufruf eines Agenten (lokal oder via RPC (Remote Procedure Call) wird nicht weiter betrachtet, da er komplexe Kommunikation nicht zulässt (vgl. Brenner (1998)).

<sup>15</sup> Beispiel: Kannst Du mir das Salz geben?

<sup>16</sup> In (Finin (1994)) findet sich eine Aufstellung wichtiger KQML-Performative.

Ein selbsterklärendes Beispiel (vgl. Dumke (2003)):

ask

```
:sender Sammelagent
:receiver Suchagent
:content (Produktname, Beschreibung, Preis)
:reply-with Anfrage 1
:language XML
:ontology Produktinformationen
```

tell

```
:sender Suchagent
:receiver Sammelagent
:content (Produktname, Beschreibung, Preis)
:in-reply-to Anfrage 1
:language XML
:ontology Produktinformationen
```

Mittels KQML werden nicht nur Nachrichten ausgetauscht, sondern auch Dialoge geführt (Folgen von Nachrichten). Die Anzahl der Teilnehmer ist dabei nicht begrenzt, d. h. sie reicht von zwei bis n (broadcast) Agenten. Die Kommunikation kann sowohl synchron als asynchron erfolgen (vgl. Huhns (1999)).

Aufgrund von Kritikpunkten (keine Begrenzung der Anzahl der Performative, keine exakte Spezifikation von KQML [viele Dialekte]) wurde auf KQML aufbauend FIPA-ACL als (nicht verbindlicher) Standard entwickelt. Beide Sprachen sind heute weit verbreitet. Ein Nachteil bei deren Verwendung besteht in den hohen Anforderungen an die Fähigkeiten der involvierten Agenten, z. B. das Interpretieren der Nachrichten und das Schlussfolgern über deren Inhalt, wodurch einige von vorneherein ausgeschlossen werden.

## **KIF**

KIF steht für Knowledge Interchange Format und entstand ebenfalls im Zuge des KSE-Projekts. Vorrangiges Ziel war die Schaffung einer Sprache zur Codierung des Inhalts von Nachrichten. Sie dient jedoch auch zur Übersetzung zwischen unterschiedlichen Wissensrepräsentationsformen (intermediäres Format). Da KIF logikbasiert ist, können nicht nur Informationen (Fakten) dargestellt werden, sondern ebenfalls komplexe Zusammenhänge wie Regeln, Bedingungen oder Meta-Wissen. Es ist aufgrund der Mächtigkeit der Sprache sogar möglich, kleinere Programme (Skripte) zu schreiben.

Einige einfache Beispiele:

Daten-Tupel:

```
(Gehalt Mitarbeiter-ID Abteilung Betrag)
```

logischer Ausdruck:

```
(>(* (width chip1) (length chip1))
   (* (width chip2) (length chip2)))
```

Programm:

```
(progn (fresh-line t)
       (print „Hello!“)
       (fresh-line t))
```

Da KQML nicht zwingend KIF zur Darstellung des Nachrichteninhalts vorschreibt, können auch andere „Inhaltssprachen“ zur Anwendung kommen, z. B. SL (Semantic Language) oder RDF (Resource Description Framework) (vgl. Huhns (1999)).

## Ontologien

Ontologien kann man sich als fachspezifisches Nachschlagewerk vorstellen. Beschrieben werden die Bedeutung (Semantik) von Begriffen und die zwischen ihnen bestehenden Beziehungen. Dazu finden u. a. logische Ausdrücke Verwendung. Andere Darstellungsweisen nutzen Klassen und Konstrukte wie „ist eine Instanz von“ oder „ist eine Sub-Klasse von“, um Konzepte der Wissensdomäne abzubilden. Ontologien ähneln damit Datenbanken-Schemata.

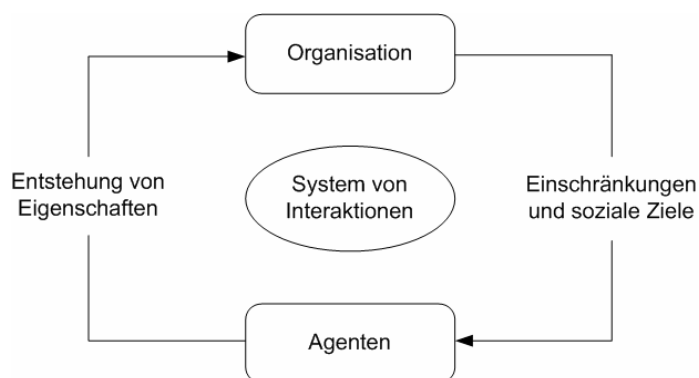
Wiederum ein simples Beispiel (Block Welt, vgl. Huhns (1999)):

```
(class Block)
(class PhysicalObject)
(subclassOf Block PhysicalObject)
 $\forall x, y, z (instanceOf\ x\ y) \wedge (subclassOf\ y\ z) \Rightarrow (instanceOf\ x\ z)$ 
```

Beispiele für Sprachen, in denen Ontologien verfasst werden, sind KIF, RDF, DAML (DARPA Agent Markup Language) oder OWL (Web Ontology Language).

### 3.6 Multi-Agenten-Organisation

Durch die vielfältigen Beziehungen zwischen Agenten in einem MAS, wie die Delegation von Aufgaben, das Eingehen von Verpflichtungen, das Synchronisieren von Aktionen oder der Datenaustausch, bilden sich organisationale Strukturen heraus. „Jede Organisation ist dort das Ergebnis der Interaktion zwischen Agenten und das Verhalten der Agenten wird durch die vorhandenen Organisationsstrukturen begrenzt.“ (Ferber (2001)) Die Abb. 3-6 veranschaulicht den Dualismus von Agent und Organisation.



Quelle: Ferber (2001)

**Abb. 3-6:** Mikro-Makro-Beziehungen in Multi-Agenten-Systemen

Eine Organisation definiert (Ferber (2001)) in Anlehnung an (Morin (1977)) folgendermaßen: „... definiert eine Menge von Beziehungen zwischen Objekten oder Individuen, die eine Einheit oder ein System bilden, welches dann Eigenschaften besitzt, die auf der Ebene der Individuen nicht zur Verfügung stehen. Die Organisation bringt verschiedene Elemente, Ereignisse oder Individuen in eine Wechselbeziehung, die dadurch zum Teil eines Ganzen werden. Das gewährleistet ein relativ hohes Maß an Vernetzung und Stabilität, was es einer Organisation ermöglicht, selbst dann auch über längere Zeitspannen hinweg zu existieren, wenn grundlegende Veränderungen eintreten.“

Obwohl Organisationen dynamischer Natur sind, enthalten sie jedoch auch stabile Elemente – die Organisationsstruktur. Diese stellt das Gerüst jeder konkreten Organisationsform dar und lässt sich durch zwei Merkmale beschreiben:

- Eine Menge von Klassen von Agenten, denen bestimmte Rollen zugeordnet sind.
- Eine Menge von abstrakten Beziehungen, die zwischen diesen Rollen bestehen.

Organisationsstrukturen bilden sich dynamisch und/oder werden vom Entwickler a priori festgelegt. Dazu kann ein Top-Down- oder Bottom-Up-Ansatz gewählt werden. Im ersten Fall geht man von einem zu lösenden Problem aus und entwirft dann das MAS (ingenieurmäßiges Vorgehen). Im zweiten wird versucht, bestimmte Systemeigenschaften durch „Emergenz“ zu erzeugen (experimentelles Vorgehen in Forschungsprojekten).

Organisationen können von verschiedenen Stufen aus betrachtet werden. (Ferber (2001)) nimmt die Unterscheidung in die mikrosoziale Ebene, die Gruppenebene sowie die der globalen Gesellschaften (Populationen) vor, welche hier jedoch nicht weiter aufgegriffen wird. Als elementare Einheiten bezeichnet man nicht weiter zerlegbare primitive Agenten, als allgemeines MAS diejenigen Komponenten, welche nicht Bestandteil einer noch höheren Organisation sind<sup>17</sup>.

Die Analyse von Multi-Agenten-Organisationen lässt sich aus drei unterschiedlichen Perspektiven vornehmen:

- Die funktionale Betrachtung: beantwortet die Frage, welche Funktionen die Komponenten eines MAS erfüllen. An dieser Stelle kommt das Rollenkonzept (Rolle als Menge von Eigenschaften, die einem Agenten zugeschrieben wird, der diese innehat) zum Einsatz.
- Die strukturelle Betrachtung: untersucht die Organisationsstrukturen (z. B. gleichberechtigt vs. hierarchisch, dynamisch vs. statisch) und ihre Parameter.
- Die Konkretisierungsparameter: steuern die Überführung einer abstrakten Organisationsform in eine konkrete. Es werden Probleme der effektiven Umsetzung der Multi-Agenten-Organisation behandelt.

Eine Gruppe von Agenten kann, wie bereits erwähnt, eine Gesellschaft formen, in der die Agenten verschiedene Rollen ausfüllen, z. B. Käufer und Verkäufer in einem elektronischen Markt. Der Beitritt zu solch einer Gemeinschaft ist freiwillig. Danach jedoch ist man durch die sozialen Regeln gebunden (social commitments), welche für diese Rolle definiert sind (für die Dauer der Übernahme). Soziale Bindungen von Agenten können durch das Konzept der sozialen Abhängigkeit (social dependency) charakterisiert werden. Ein Beispiel: Agent x verfolgt das Ziel, den Zustand p herzustellen. Dazu muss allerdings erst Zustand a eintreten. Um das zu bewerkstelligen, besitzt er aber nicht die erforderlichen Fähigkeiten. Deshalb bittet er Agent y, der über diese verfügt, ihm zu helfen. Stimmt der zu, ist Agent x solange von Agent y abhängig, bis dieser seine (freiwillige) Verpflichtung erfüllt hat (vgl. Ferber (2001), Huhns (1999)).

---

<sup>17</sup> Das „Matroschkasystem“: Ein Agent kann sich aus anderen Agenten zusammensetzen.

### 3.7 Intelligenz

„Zur Ausführung seiner Aufgaben benötigt ein Agent immer einen gewissen Grad an Intelligenz. ... Erst die Intelligenz ermöglicht es jedoch dem Agenten, seine Aufgaben weitestgehend autonom zu bearbeiten und nur bei wichtigen Entscheidungen Eingriffe des Benutzers zu erfordern.“ (Brenner (1998)). Über die Frage, wie Intelligenz definiert werden kann, gibt es keine einheitliche Auffassung. (Turing (1950)) schlug den nach ihm benannten Turing-Test<sup>18</sup> vor. Intelligenz bei Agenten äußert sich nach (Wooldridge (1999)) durch die Fähigkeit zur Ausführung flexibler Aktionen (siehe Abschnitt 2.1).

Um Agenten mit Intelligenz auszustatten, werden Technologien diverser Fachrichtungen (KI, VKI, Sprachverarbeitung, Spieltheorie, Psychologie usw.) genutzt. Einige von ihnen stellt der Abschnitt 6.1.1 vor. Dort wird auch auf die Rolle des Wissens und seine Darstellung eingegangen.

Zu intelligenten Leistungen sind sowohl kognitive als auch reaktive Agenten fähig, allerdings mit unterschiedlichen Mitteln (siehe Abschnitt 2.3). „Kognitive Agenten können aufgrund ihrer Fähigkeit zur Repräsentation der Welt und zum Schlussfolgern über der Repräsentation relativ unabhängig operieren. Sie können individuell relativ komplexe Aufgaben ausführen und sie können ihr Verhalten ausdrücken. Reaktive Agenten zeigen wegen ihrer simplen Struktur und ihrer geringen Ressourcen nur einfache Verhaltensweisen. Sie können aber leicht Gruppen oder Populationen bilden, die sich an unterschiedliche Umgebungen anpassen können. Sie haben wenig oder gar keine Individualität, aber da sie große Gruppen bilden können, sind sie darin wegen der Redundanz in der Lage, komplexe Probleme zu lösen.“<sup>19</sup> (Dilger (2003), Ferber (2001)).

---

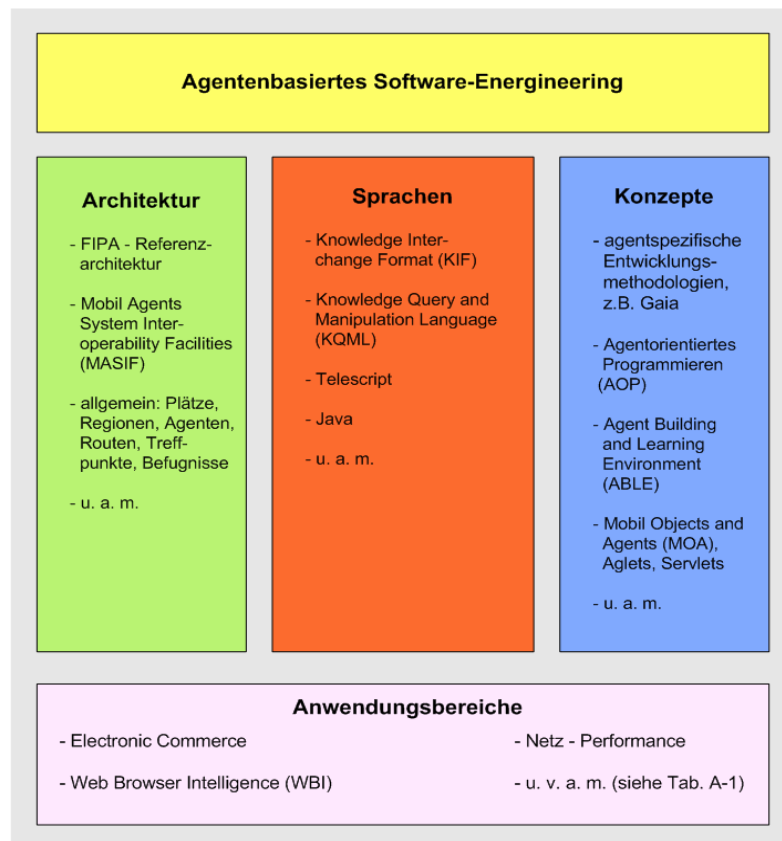
<sup>18</sup> Beim Turing-Test kommuniziert eine Person ohne Sicht- und Hörkontakt über Tastatur und Bildschirm mit zwei Gesprächspartnern. Der eine ist ein Mensch, der andere ein Computerprogramm. Kann die Person nicht zwischen Beiden unterscheiden, hat die Maschine den Test bestanden.

<sup>19</sup> Die Bedeutung der Interaktion nimmt beim Vollbringen intelligenter Leistungen zu, je reaktiver das Gesamtsystem ist.

## 4 Agentenorientiertes Software-Engineering

### 4.1 Begriffsbestimmung

Die grundlegenden Bestandteile des Agentenorientierten Software-Engineerings (AOSE) zeigt die folgende Abb. 4-1:



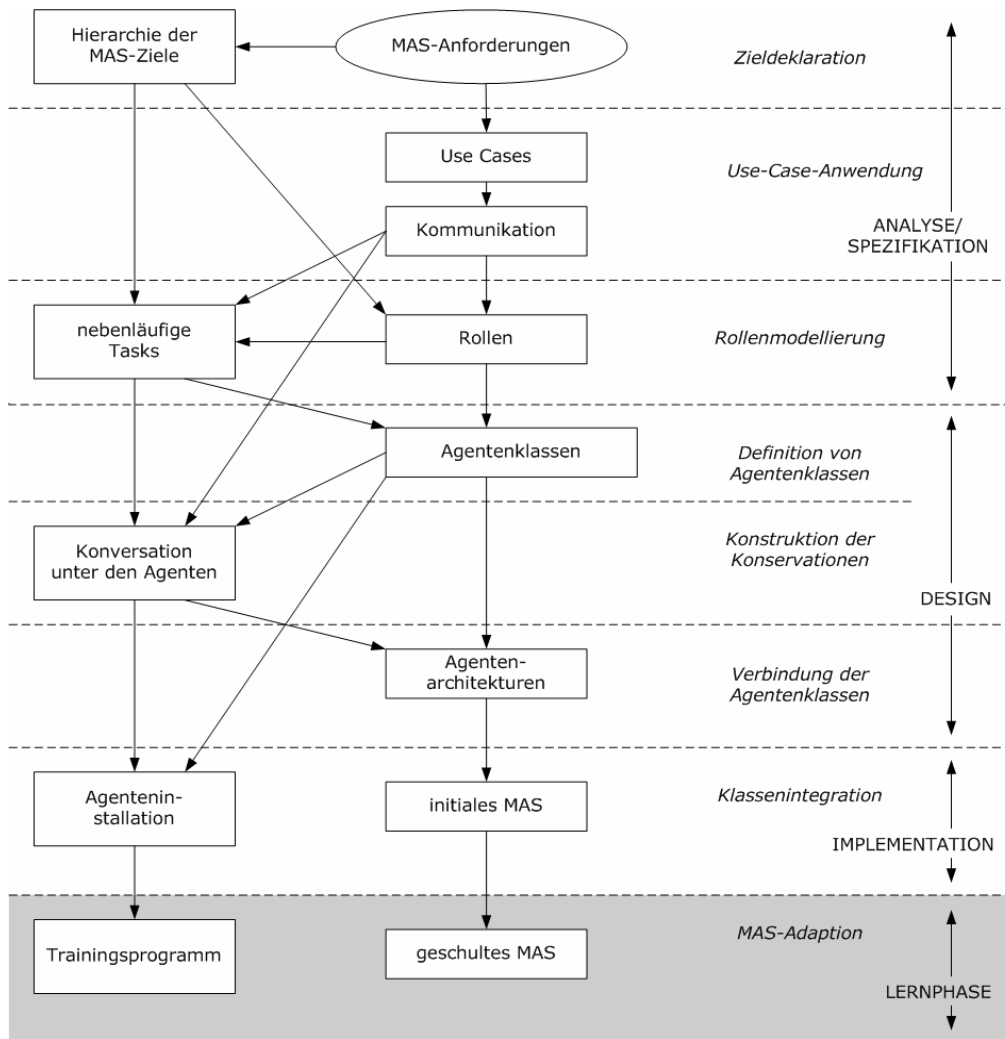
Quelle: vgl. Dumke (2003a)

**Abb. 4-1:** Bestandteile AOSE

„Gegenstand von AOSE sind Vorgehensweisen, Methoden, Techniken und Tools für die Erstellung und Handhabung von agentenorientierter Software“ (Weiss (2001)). Die Entwicklung des AOSE steht erst am Anfang. Inzwischen wurde allerdings deren Bedeutung erkannt. Dazu (Wooldridge (2000)): „If agents are to realise their potential as a software engineering paradigm, then it is necessary to develop software engineering techniques that are specifically tailored to them.“

### 4.2 Der Entwicklungsprozess von MAS

Bevor auf einzelne Komponenten des AOSE eingegangen wird, soll ein allgemeines Phasenmodell der MAS-Entwicklung eingeführt werden (vgl. Wood (2001), Wille (2002)).



Quelle: Dumke (2003)

**Abb. 4-2:** MAS-Entwicklungsphasen

Zusätzlich zu den aus OO-Modellen bekannten Schritten gibt es eine explizite Lernphase.

#### 4.2.1 Analyse, Spezifikation und Design

Die Anwendung des agentenorientierten Paradigmas im Software-Engineering zieht ein Denken auf einer höheren Abstraktionsebene nach sich, als es bei einer OO-Sichtweise der Fall wäre. Der Grund dafür liegt in den konzeptuellen Unterschieden zwischen Software-Agenten und Objekten (siehe Abschnitt 2.1). Der Agenten-Ansatz schließt aber andere nicht aus: „... ermöglichen das Agenten- und das Objekt-konzept auf qualitativ verschiedenen Abstraktionsebenen gleichermaßen relevante System- und Entwicklungssichten, die sich sinnvoll ergänzen statt gegenseitig ausschließen. Die Agentensicht impliziert damit aber auch quer über alle Phasen der Softwareentwicklung zahlreiche neue, von der Objektsicht nicht abgedeckte Anforderungen und Fragestellungen.“ (Weiss (2001)).

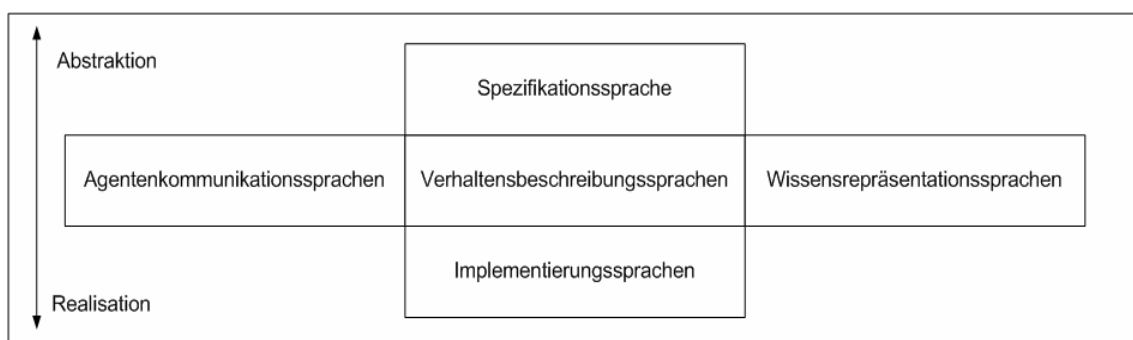
Die Analyse- und Designmethoden für MAS lassen sich in drei Gruppen unterteilen:

- **Agenten-Technologie:** Der Schwerpunkt liegt bei diesen Methoden auf den für die agentenorientierte Sichtweise typischen Abstraktionen (Gruppe, Organisation, Rolle, usw.) und Verfahren (z. B. zur wissensgestützten Koordination). Vertreter: Generic Architecture for Information Availability (Gaia, Wooldridge (2000))
- **Objektorientierung:** Bestehende OO-Ansätze werden um agentenspezifische Konzepte erweitert. Vertreter: MultiAgent SystemS Iterative View Engineering (MASSIVE, Lind (2001))
- **Knowledge Engineering:** Diese Verfahren identifizieren und modellieren Wissen, welches die einzelnen Software-Agenten besitzen. Vertreter: Conceptual Modelling of Multi Agent Systems (CoMoMAS, Glaser (2002))

Zur Spezifikation können „traditionelle“ und logikbasierte Formalismen eingesetzt werden. Zur ersten Kategorie zählen UML (Unified Modeling Language) oder Petrinetze, in welche die neuen agentenspezifischen Konstrukte integriert wurden<sup>20</sup>. Zur zweiten gehören temporale Logiken und Sprachen, welche auf solchen Logiken aufsetzen<sup>21</sup> (vgl. Weiss (2001)).

#### 4.2.2 Implementierung

Zur Entwicklung und Implementierung von MAS kommen fünf Klassen von Sprachen zur Anwendung (siehe Abb. 4-3).



Quelle: Ferber (2001)

**Abb. 4-3:** Sprachen in MAS

#### Implementierungssprachen

<sup>20</sup> z. B. Agent-UML (<http://www.auml.org>)

<sup>21</sup> beispielsweise Concurrent METATEM (Fisher (1995))

Der Einsatz einer Implementierungssprache richtet sich im Allgemeinen nach der zugrunde liegenden Agenten-Plattform. Die Programmiersprache Java hat dabei eine weite Verbreitung gefunden. Sie bietet im Hinblick auf die Agenten-Technologie folgende Vorzüge: Objektorientierung, Plattformunabhängigkeit, Kommunikationsfähigkeit, Sicherheit, Code-Manipulation, Multitasking, Persistenz u. a. Java unterstützt allerdings keine agentenspezifischen Konzepte. Diese Besonderheit weisen Sprachen auf, die speziell für die Agentenorientierte Programmierung (AOP) entworfen wurden. Dazu zählen die JACK Agent Language (JAL, <http://www.agent-software.com/>) oder die Runtime Agent Definition Language (RADL, <http://www.agentbuilder.com/>), welche auf Agent0 (Shoham (1993)) und PLACA (Thomas (1993)) basiert<sup>22</sup>. Die Tab. 4-1 listet die unterschiedlichen Merkmale von OOP (Objektorientierter Programmierung) und AOP auf.

**Tab. 4-1:** Vergleich von OOP und AOP

Charakteristik	OOP	AOP
Strukturelle Elemente	abstrakte Klasse	generische Rollen
	Klasse	domainspezifische Rollen
	Klassenvariablen	Wissen, Glaube
	Methoden	Fähigkeiten
Relationen	Kollaboration	Verhandlungsfähigkeit
	Komposition	Agentengruppierungen
	Vererbung	Rollenvervielfachung
	Instantiierung	domainspezifische Rolle + individuelles Wissen
	Polymorphismus	Service-Kombinationen

Quelle: Lind (2001)

## Plattformen

In der Praxis finden sich viele verschiedene Agenten-Plattformen. Die Spannweite reicht dabei von prototypischen Anwendungen aus dem universitären Umfeld, über Open-Source-Projekte bis hin zu kommerziell genutzten Systemen. Es sollen nur Verweise zu Internetseiten genannt werden, die Übersichten von verfügbaren Plattformen geben: <http://agents.umbc.edu/>, <http://aose.ift.ulaval.ca/index.php> oder <http://www.agencities.org/index.jsp>.

<sup>22</sup> Daneben gibt es noch weitere Sprachen wie Tcl/Tk (Tool Command Language) oder Telescript.

### 4.2.3 Training

Um ihre volle Leistungsfähigkeit zu entfalten, benötigen adaptive Agenten eine zusätzliche Trainingsphase.<sup>23</sup> Diese kann online (während des Einsatzes), offline (vor dem Einsatz) bzw. kombiniert realisiert werden. Die folgende Abb. 4-4 zeigt mögliche Ansätze für ein Training. Dargestellt ist ein MAS mit zwei generischen Agenten. Die mit Rot hervorgehobenen Teile der Grafik kennzeichnen potentiell zu trainierende Komponenten. Eine ausführliche Darstellung des Agenten-Trainings erfolgt in Kapitel 6.

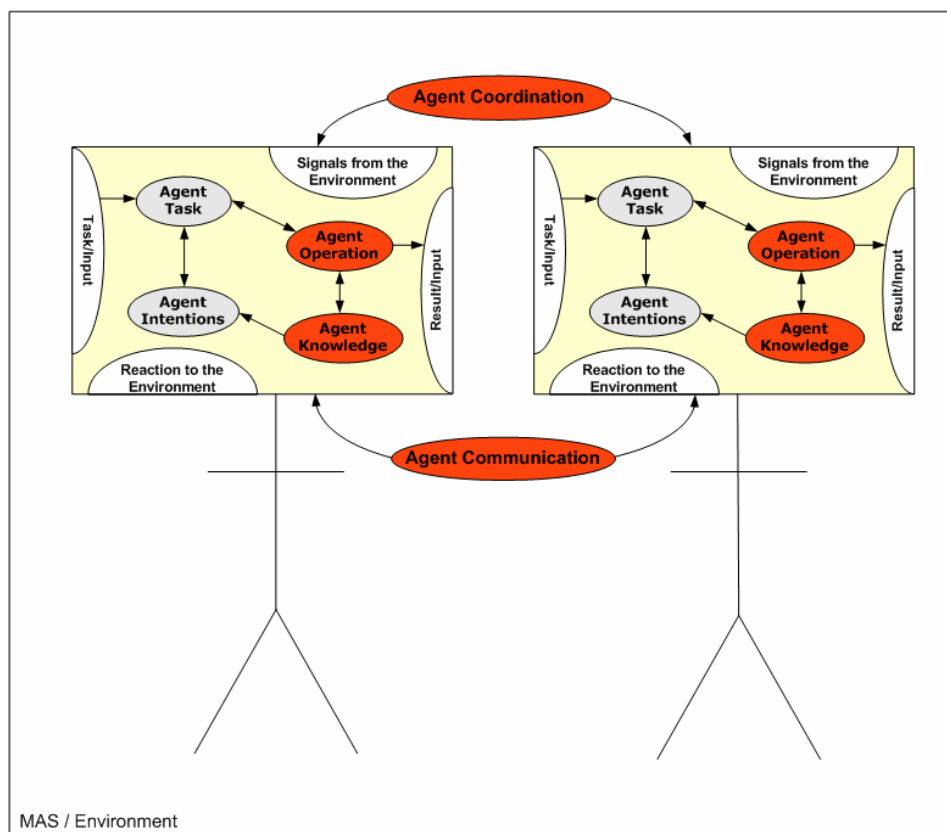


Abb. 4-4: Strategische Ansätze für das Agenten-Training

#### Aktionsselektion (Agent Operation)

Agenten stehen oft vor der Entscheidung, aus einem Set von Aktion die „richtige“ auszuwählen. Dazu bedarf es einer Bewertungsfunktion, die der Agent kennen oder erlernen muss (siehe Abschnitt 6.2.1).

<sup>23</sup> Agenten, die in komplexen, dynamischen, offenen und unvorhersehbaren Umgebungen ohne Training stets optimal handeln, sind – wenn überhaupt – sehr schwierig umzusetzen.

### **Agenten-Wissen (Agent Knowledge)**

Aus der Wissensbasis (dem „Gedächtnis“) leitet der Agent seine Überzeugungen in Bezug auf die Umwelt ab. Sie bildet die Grundlage für alle Entscheidungsprozesse<sup>24</sup>. Eine Aktualisierung (Anpassung) und/oder Erweiterung derselben durch den Agenten stellt eine Form des Lernens dar (siehe Abschnitte 6.2.2 und 6.3.2).

### **Agenten-Koordination (Agent Coordination)**

Die Koordination von Agenten ist eine Schlüsseleigenschaft von MAS. Sie bewirkt allgemein eine Leistungssteigerung des Systems. Um dies zu erreichen, werden üblicherweise Protokolle eingesetzt. Andere Möglichkeiten bestehen darin, Agenten das erforderliche Koordinationswissen lernen zu lassen oder beide Ansätze miteinander zu verbinden (siehe Abschnitte 3.4 und 6.3.1).

### **Agenten-Kommunikation (Agent Communication)**

Agenten verfügen meistens nur über „lokales“ Wissen (siehe Punkt 6.1.1), was zu suboptimalen oder Fehlentscheidungen führen kann. Mittels Kommunikation können Agenten versuchen, fehlendes Wissen zu akquirieren. Diese verringert aber die Performance des MAS (zeitintensiv, Fehleranfälligkeit steigt, usw.) und muss mit Bedacht eingesetzt werden. Deshalb erscheint es sinnvoll Agenten lernen zu lassen, mit welchem Partner sich eine Kommunikation „lohnt“ (siehe Abschnitt 6.3.2).

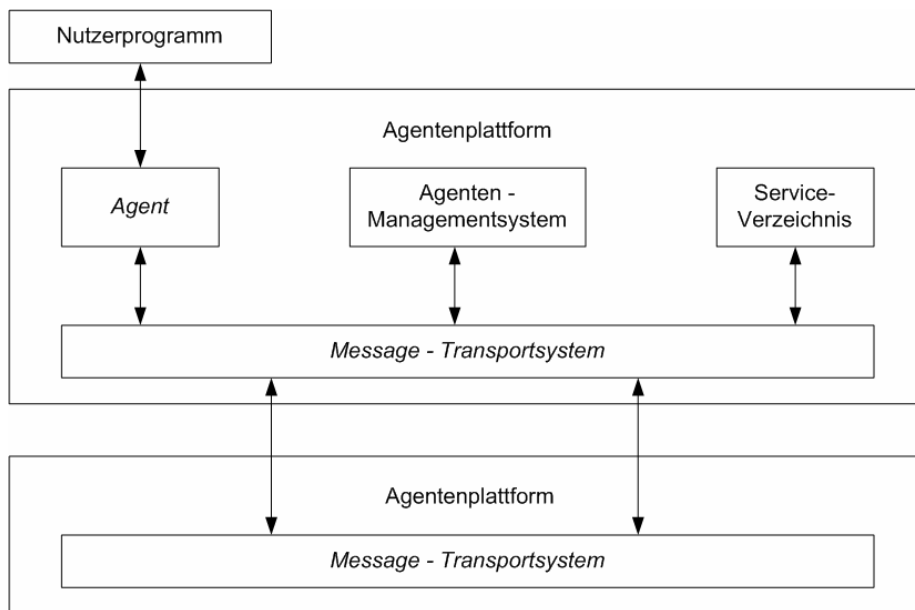
Viele Verfahren für das Training von MAS stammen aus der KI. Aber nicht jeder Programmierer ist ein Experte auf diesem Gebiet. Deshalb versuchen Tools in verschiedener Art und Weise die Entwicklung lernfähiger Agenten zu unterstützen (siehe Kapitel 7). Des Weiteren stehen für ein Problem oft mehrere Trainingsansätze zur Verfügung. Deren Effizienzbewertung ist für eine Auswahl von Bedeutung (siehe Kapitel 5 und 8).

## **4.3 Standards**

Die Vielzahl an Plattformen, Architekturen oder ACL ist für eine weite Verbreitung und kommerzielle Nutzung der Agenten-Technologie kontraproduktiv. Deshalb gibt es Anstrengungen seitens einiger Organisationen, notwendige Standardisierungen einzuführen.

---

<sup>24</sup> Art und Umfang von Schlussfolgerungen hängen von der Architektur des Agenten ab (siehe Punkt 2.3).



Quelle: Dumke (2003)

**Abb. 4-5:** FIPA-Referenzarchitektur

Diese zeigen erste Erfolge, wie z. B. die Akzeptanz der durch die FIPA vorgeschlagenen Referenzarchitektur für Agenten-Plattformen (siehe Abb. 4-5). „FIPA-Konformität gilt zunehmend als Voraussetzung für industrielle und kommerzielle agentenorientierte Software, weshalb dieser Standard auch von nahezu allen neueren Tools und Plattformen unterstützt wird.“ (Weiß (2005)). Die FIPA beschäftigt sich außerdem mit Agent-UML (agentenbasierte Erweiterung von UML), FIPA-ACL (Standard für ACL, Weiterentwicklung von KQML) und einem SPEM-basierten Meta-Modell für den Softwareentwicklungsprozess von agentenorientierter Software. Weitere Bemühungen sind durch die OMG zu verzeichnen. Diese arbeitet an der Integration von agentenspezifischen Konzepten in Standard-UML sowie der Förderung der Interoperabilität zwischen mobilen Agenten durch den Standard Mobile Agent System Interoperability Facilities (MASIF) (vgl. Weiß (2005)).

## 5 Softwaremessung von Agentenbasierten Systemen

### 5.1 Allgemeine Ansätze für die Agentenmessung

Ebenso wie sich das AOSE erst am Anfang seiner Entwicklung befindet, steckt auch die Softwaremessung von MAS noch „in den Kinderschuhen“. Ihre Bedeutung wird weithin unterschätzt, dabei bildet sie doch eine Grundlage für:

- die Beurteilung der Softwarequalität von agentenbasierten Systemen (Evaluierung), was wiederum die Voraussetzung für eine breite Akzeptanz und den verstärkten Einsatz solcher Systeme in der Praxis ist;
- das Benchmarking von MAS, Entwicklungsmethodologien und Tools;
- die Datengewinnung für den Softwareentwicklungsprozess (z. B. zur Planungssicherheit [Aufwandsschätzungen, Skalierung von MAS]).

Ein Beleg für den großen Nachholbedarf auf diesem Gebiet ist die Tatsache, dass die Überprüfung der Softwarequalität sowie Aspekte der Softwaremessung gegenwärtig nicht *integraler* Bestandteil (mehr als nur Testen) von Entwicklungsmethodologien<sup>25</sup> für MAS sind, wie es z. B. bei OO-Softwareentwicklungsverfahren der Fall ist (vgl. Wille (2004), Weiß (2005); Hinweis: Interpretation der Quellen durch den Autor).

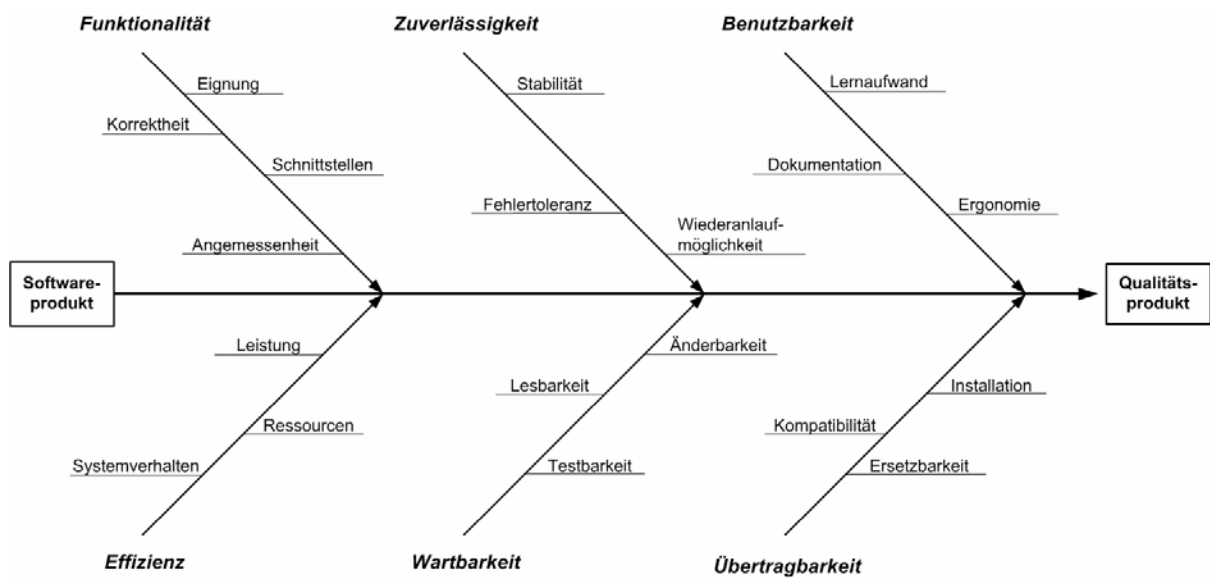
Bis dato gibt es keine Standards und nur wenige Ansätze zur Softwaremessung von Agenten. Die Konsequenz sind fehlende quantifizierbare Erfahrungen im Bereich der Software-Agenten und MAS, von einigen Arbeiten zu speziellen Themen abgesehen (vgl. Wille (2004)). Messansätze kann man (vgl. Wille (2002b)) in Verfahren für das Produkt (also auf den Software-Agenten oder das MAS selbst bezogen), den Prozess (als Entwicklungstechnologie von MAS) sowie die (in der Entwicklung und Anwendung) benötigten Ressourcen unterteilen<sup>26</sup>.

Für die Evaluierung von Softwareprodukten (und nichts anderes sind MAS) empfiehlt sich der Einsatz eines Qualitätsmodells. Als wichtige Norm in diesem Zusammenhang ist der internationale Standard ISO 9126 (<http://www.iso.org>) zu nennen. Er definiert qualitative Anforderungen (Merkmale) an Software und gibt Richtlinien für deren Bestimmung bzw. Messung vor (siehe Abb. 5-1).

---

<sup>25</sup> Es gibt einige wenige Ausnahmen, z. B. MASSIVE (Lind (2001)), MALINA (Multi-Agent Local Integrated Network Associations, Stojanov (1996)).

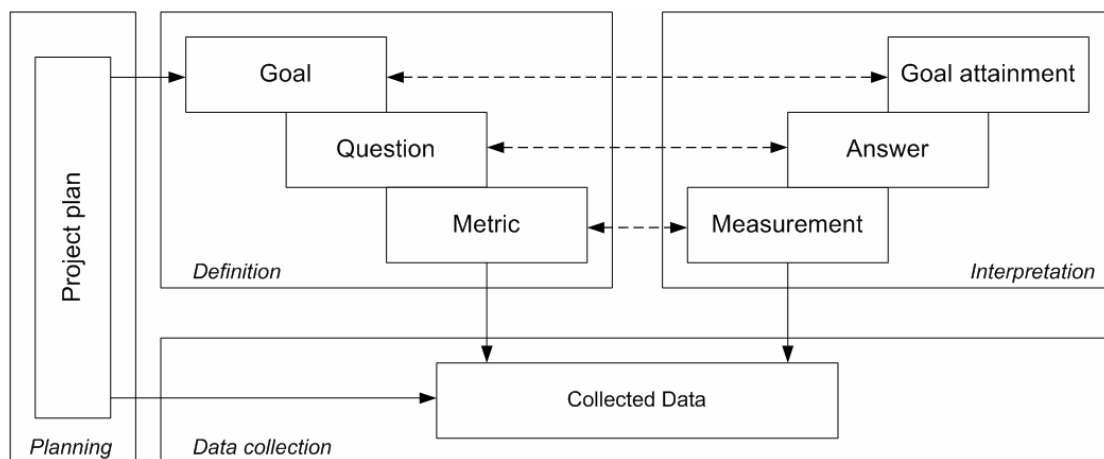
<sup>26</sup> Beispiele finden sich in (Wille (2002b)).



Quelle: Dumke (2003)

**Abb. 5-1:** Fischgrätendiagramm

Goal-Question-Metric (GQM) ist ein Konzept für die Umsetzung des eigentlichen Messvorgangs, das seine Praxistauglichkeit bewiesen hat<sup>27</sup>. Die GQM-Methode ist auf *zielerorientiertes* Messen ausgerichtet (Ein Punkt, der für die Messung von MAS von Bedeutung ist, wie später noch gezeigt wird.).



Quelle: vgl. Wille (2004)

**Abb. 5-2:** Phasen Goal-Question-Metric

<sup>27</sup> Ein Beispiel für die Anwendung im Agenten-Umfeld findet sich in (Wille (2004)).

GQM umfasst vier Phasen (vgl. Abb. 5-2, Punter (2001)):

- Planung: Entwurf eines Projektplans für die Messung
- Definition: Das Messprogramm wird definiert und dokumentiert (Messziele, Fragen [Teilziele], Metriken und Hypothesen).
- Software-Messung: das Sammeln der erforderlichen Daten
- Interpretation: Die Daten werden ausgewertet und interpretiert. Die gewonnenen Ergebnisse beantworten die zuvor definierten Fragen. Danach ist man in der Lage, den Grad der Zielerfüllung zu bestimmen.

Grundsätzlich sind die aus dem OOSE bekannten klassischen Mess- und Bewertungsformen einsetzbar<sup>28</sup>. Folgende Punkte sollten allerdings berücksichtigt werden (vgl. Wille (2002b)):

- Es sind zusätzliche Messansätze zu konzipieren und implementieren, die
  - den besonderen *Eigenschaften* von Software-Agenten und MAS gerecht werden, z. B. im Hinblick auf die besonderen Kommunikations- oder Koordinationsformen. (Messungen müssen zur *Laufzeit* erfolgen.)
  - den Eigenheiten des *Entwicklungsprozesses* von MAS<sup>29</sup> Rechnung tragen, vor allem der zusätzlichen Lern- und Trainingsphase. Speziell bei Aufwandsschätzverfahren sind die potentiellen, impliziten und generischen Funktionalitäten zu beachten.
  - bei der *Ressourcenbewertung* die unterschiedlichen Architekturen von MAS berücksichtigen.
- Software-Agenten können selbst in Form *systembezogener Messagenten*<sup>30</sup> Parameter von MAS (oder Teilbereichen) kontrollieren.
- Softwaremessungen und -bewertungen zur Laufzeit stellen stets einen *Overhead* dar. Deshalb ist die Messaufwandseffizienz zu überprüfen. Erforderlich ist auf jeden Fall ein *zweck- und zielgerichtetes* Messen.

(Dumke (2000)) gibt zahlreiche Beispiele für *agentenorientierte* Metriken, deren empirische Bedeutung von ihm ebenfalls diskutiert wird.

---

<sup>28</sup> Agenten (MAS) werden oft in OO-Sprachen implementiert.

<sup>29</sup> siehe auch (Knapik (1998), Wood (2001) und (Wooldridge (2001))

<sup>30</sup> Die erwähnte Selbstmessung durch Software-Agenten wurde beispielhaft an der OvG-Universität Magdeburg für Java-Aglets umgesetzt (siehe <http://ivs.cs.uni-magdeburg.de/sw-eng/agruppe/>).

Metriken haben eine unterschiedlich starke Aussagekraft, abhängig von der Grundlage, auf der sie ermittelt wurden (Auszug; vgl. Dumke (2000)):

- Trendanalyse: die schwächste Form, anwendbar, wenn keine besseren Informationen zur Verfügung stehen
- Empirisch gestützte Schätzung: basiert auf Erfahrungen und unterstützt die Evaluation von Software-Merkmalen durch Formeln
- Evaluation: betrachtet empirische Aspekte auf Basis statistischer Verfahren
- Direkte Messung: Aussagen werden aufgrund konkreter Messungen gewonnen (keine [modelbasierten] Abstraktionen oder Transformationen).

## 5.2 Effizienz und deren Anwendung auf Lern- und Trainingsverfahren

Das Wort „effizient“ stammt aus dem Lateinischen und bedeutet „besonders wirksam, wirtschaftlich“ (diverse Lexika). Der ISO Standard 9000:2000 definiert „Effizienz“ folgendermaßen: „... ist das Verhältnis zwischen dem erzielten Ergebnis und den eingesetzten Mitteln.“ (Im Gegensatz dazu bezeichnet der ISO Standard 9000:0000 „Effektivität“ als das „Ausmaß, in dem geplante Tätigkeiten verwirklicht und geplante Ergebnisse erreicht werden.“<sup>31</sup>) Demzufolge müssen Kriterien ausgewählt werden, anhand derer eine Beurteilung der Effizienz erfolgen soll. Diese sind spezifisch für jeden Anwendungsfall zu ermitteln. Anhaltspunkte gibt der bereits erwähnte internationale Standard ISO 9126. In Anlehnung an diesen werden, geordnet nach den untenstehenden Kriterien, Metriken für die Effizienzbestimmung vorgestellt (siehe auch Abb. 5-1, vgl. Wille (2001)). Als Betrachtungsebene kann der einzelne Software-Agent oder das MAS als Ganzes bestimmt werden.

- Produkt/Performance: Hierunter fallen beispielsweise Antwort- und Verarbeitungszeiten oder auch Durchsatzraten (Bestimmung zur Laufzeit).
- Prozess: Es erfolgt eine Bewertung des MAS-Softwareentwicklungsprozesses (siehe auch Abb. 4-2), der MAS-Entwicklungsmethodologie und eingesetzter CASE-Tools (Computer Aided Software Engineering).
- Ressourcen: Zu diesem Punkt zählen das Entwicklungspersonal, die angewandte Software (CASE-Tools als auch die Komponenten des MAS selbst) sowie die benötigte Hardware zum Betreiben des Systems.

---

<sup>31</sup> Effektivität ("Die richtigen Dinge tun") und Effizienz ("Die Dinge richtig tun") sind zwei korrespondierende Größen. Allerdings muss zuerst ein effektiver Weg zur Lösung eines Problems gefunden werden, bevor dieser effizienter gestaltet werden kann.

Tab. 5-1: Produkt-/Performancemetriken

PRODUKT-/PERFORMANCEMETRIKEN	
Software-Agent	Agentensystem
Agentendesignniveau:	Systemdesignniveau:
<b>Größe:</b> die Größe kann die Effizienz hinsichtlich der Bewegung des Agenten beeinflussen (als <i>implizite Effizienz</i> )	<b>Größe:</b> die Größe des Systems kann durch einen Overhead die Leistung beeinflussen (als <i>potentielle Effizienz</i> )
<b>Komponentenstruktur:</b> Sie beeinflusst die Effizienz bzgl. einer modularen Aktionsunterteilung (als <i>Struktureffizienz</i> )	<b>Komponentenstruktur:</b> hierbei wird entsprechend der Agentenhierarchien, dem Grad der Parallelität, und der Arten der Funktionalität die Effizienz beeinflusst (als <i>Architektureffizienz</i> )
<b>Komplexität:</b> sie hat ggf. eine direkte Beeinflussung zur Effizienz eines Agenten (als <i>immanente Effizienz</i> )	<b>Komplexität:</b> hierbei bezieht man sich auf die Form der organisatorischen Dimensionen (sozial, Verhältnis, physikalisch, Umgebung, persönlich) (als <i>Entropieeffizienz</i> )
<b>Funktionalität:</b> hierbei ist das Leistungsverhalten pro Funktion zu betrachten (als <i>Aktionseffizienz</i> )	<b>Funktionalität:</b> die Systemfunktionalität impliziert die <i>Modelleffizienz</i>
Agentenbeschreibungsniveau:	Systembeschreibungsniveau:
<b>Entwicklungsbeschreibung:</b> hierbei wird die Effizienz hinsichtlich der Unterstützung für die Wartung betrachtet (als <i>Änderungseffizienz</i> )	<b>Entwicklungsbeschreibung:</b> bewertet wird die Unterstützung als <i>Wartungseffizienz</i>
<b>Anwendungsbeschreibung:</b> hierbei geht es um die Unterstützung der <i>Anwendungseffizienz</i>	<b>Anwendungsbeschreibung:</b> dabei bewertet man die <i>Nutzungseffizienz</i>
<b>Veröffentlichungen:</b> als Bewertung der <i>Verbreitungseffizienz</i>	<b>Veröffentlichungen:</b> hinsichtlich der Bewertung der <i>Marketingeffizienz</i>
Agentenoperationsniveau:	Systemoperationsniveau:
<b>Kommunikation:</b> betrachtet das Kommunikationsaufkommen und das Kommunikationsniveau bzgl. der <i>Kommunikationseffizienz</i>	<b>Kommunikation:</b> z. B. Anzahl der Agenten-Kommunikationssprachen zwischen den verschiedenen Arten der Software-Agenten und der verschiedenen Rollen und Aktionen (als <i>Verständigungseffizienz</i> )
<b>Interaktion:</b> diese Metrik bezieht sich auf den Agentenkontext bzw. die Umgebung (als <i>Interaktionseffizienz</i> )	<b>Interaktion:</b> betrachtet die Art der Interaktion bezogen auf den Agenten und seiner Rolle in der Umgebung (als <i>Teameffizienz</i> )
<b>Lernen:</b> bezieht sich auf die Erfahrungen, Absichten und Aktionen (als <i>Lerneffizienz</i> )	<b>Wissen:</b> misst die Resultate des Lernens des Agenten für das Agentensystem (als <i>Wissenseffizienz</i> )
<b>Adaption:</b> die Fähigkeit der Änderung eines Agenten (als <i>Anpassungseffizienz</i> )	<b>Systemlebensdauer:</b> basierend auf der Adaption der Agenten im Gesamtsystems (als <i>Adaptionseffizienz</i> )
<b>Verhandlung:</b> diese Metrik bewertet eine erfolgreiche Verhandlungstätigkeit (als <i>Verhandlungseffizienz</i> )	<b>Konfliktmanagement:</b> der Systemerfolg basiert auf der Verhandlungsfähigkeit der Agenten (als <i>Konfliktlösungseffizienz</i> )
<b>Zusammenarbeit:</b> dabei orientiert man sich an den Möglichkeiten der Zusammenarbeit mit anderen Agenten (als <i>Kollaborationseffizienz</i> )	<b>Gemeinschaft:</b> betrachtet das Niveau der verschiedenen Agentengemeinschaften (als <i>Community-Effizienz</i> )
<b>Koordination:</b> bewertet die Formen des Management im Rahmen einer Aufgabenerfüllung (als <i>Koordinationseffizienz</i> )	<b>Management:</b> basiert auf dem Koordinationsniveau im gesamten System (als <i>Managementeffizienz</i> )
<b>Kooperation:</b> betrachtet Umfang und Effizienz der gemeinsamen Aufgabenrealisierung (als <i>Kooperationseffizienz</i> )	<b>Anwendung:</b> setzt die Anwendungsgebiete und die unterschiedlichen Rollen der Agenten in Beziehung (als <i>Applikationseffizienz</i> )

<b>Selbstreproduktion:</b> der Aufwand der Reparaturen im Agenten wird abgeschätzt (als <i>Reproduktionseffizienz</i> )	<b>Stabilität:</b> basiert auf der Selbstreproduktion der Agenten (als <i>Stabilitätseffizienz</i> )
<b>Leistung:</b> diese Metrik betrachtet die Aufgabenerfüllung in Relation zur Leistung des Agenten (als <i>Operationseffizienz</i> )	<b>Leistung:</b> als Handhabung mit Objekten, um eine spezielle Aufgabe zu realisieren (als <i>Verarbeitungseffizienz</i> )
<b>Mobilität:</b> diese Metrik betrachtet die Bewegung eines Agenten über verschiedene Plätze bzw. Cities hinweg (als <i>Mobilitätseffizienz</i> )	<b>Flexibilität:</b> das Mobilitätsverhalten aller Agenten im System ist Ausdruck einer flexiblen Aufgabenerfüllung (als <i>Flexibilitätseffizienz</i> )
<b>Spezialisierung:</b> diese Metrik betrachtet den Grad der Spezialisierung und den Grad der Redundanz der Fähigkeiten eines Agenten (als <i>Eignungseffizienz</i> )	<b>Organisation:</b> hinsichtlich der verschiedenen Aufgaben eines Agenten (als Archivar, Verbraucher, Vermittler, Planer, Entscheider, Kommunikator usw.) (als <i>Organisationseffizienz</i> )

Tab. 5-2: Prozessmetriken

PROZESSMETRIKEN	
Software-Agent	Agentensystem
<b>Agentenlebenszyklusniveau:</b>	<b>Systemlebenszyklusniveau:</b>
<b>Phasen:</b> diese Metrik bezieht sich auf die Charakteristika in den verschiedenen Entwicklungsphasen (als <i>Entwicklungsphaseneffizienz</i> )	<b>Phasen:</b> betrachtet die Systemmaße Größe, Struktur und Komplexität während der Systementwicklung (als <i>Gestaltungseffizienz</i> )
<b>Meilensteine:</b> definiert die Phasenerfüllung zu einem speziellen Zeitpunkt (als <i>Erfüllungseffizienz</i> )	<b>Meilensteine:</b> Realisierung der jeweiligen Phase zu einem speziellen Zeitpunkt (als <i>Realisierungseffizienz</i> )
<b>Verlauf der Anforderungsumsetzung:</b> beschreibt den Teil der jeweils implementierten Anforderungen (als <i>Gewährleistungseffizienz</i> )	<b>Verlauf der Anforderungsumsetzung:</b> beschreibt den Umsetzungsanteil während der Entwicklungsphasen des Systems (als <i>Umsetzungseffizienz</i> )
<b>Entwicklungsniveau:</b>	<b>Entwicklungsniveau:</b>
<b>Methodik:</b> Niveau der genutzten Entwicklungsmethode wird quantifiziert (als <i>Methoden-Effizienz</i> )	<b>Methodik:</b> das Niveau der genutzten Entwicklungsmethode wird quantifiziert (als <i>Methodologie-Effizienz</i> )
<b>Paradigma:</b> bewertet wird die Eignung des gewählten Entwicklungsparadigmas (als <i>Paradigmeneffizienz</i> )	<b>Paradigma:</b> das Paradigma zur Systementwicklung wird bewertet (als <i>Technologieeffizienz</i> )
<b>CASE:</b> dieses Maß quantifiziert die Werkzeugunterstützung (als <i>Tooleffizienz</i> )	<b>CASE:</b> Werkzeugunterstützung für die Systementwicklung wird bewertet (als <i>CASE-Effizienz</i> )
<b>Entwicklungsmanagementniveau:</b>	<b>Entwicklungsmanagementniveau:</b>
<b>Projektmanagement:</b> diese Metriken beziehen sich auf die Managementebene (als <i>Koordinierungseffizienz</i> )	<b>Projektmanagement:</b> Metriken beziehen sich auf die Managementebene (als <i>Prozesseffizienz</i> )
<b>Konfigurationsmanagement:</b> bewertet wird eine erfolgreiche Versionskontrolle für den Agenten (als <i>Versionseffizienz</i> )	<b>Konfigurationsmanagement:</b> bewertet wird die Systemkonfiguration inklusive dynamischer Aspekte (als <i>Konfigurationseffizienz</i> )
<b>Qualitätsmanagement:</b> bewertet die qualitätssichernden Techniken für einen Agenten (als <i>Qualitätssicherungseffizienz</i> )	<b>Qualitätsmanagement:</b> bewertet die qualitätssichernden Techniken für das Agentengesamtsystem (als <i>Systemgüteeffizienz</i> )

Tab. 5-3: Ressourcenmetriken

RESSOURCENMETRIKEN	
Software-Agent	Agentensystem
<b>Entwicklerniveau</b>	<b>Entwicklerniveau</b>
<b>Entwicklererfahrungen:</b> die Bewertung betrifft die Entwicklererfahrungen (als <i>Skilleffizienz</i> )	<b>Entwicklererfahrungen:</b> erweitert um ggf. dynamische Systemcharakteristika (als <i>Erfahrungsumsetzungseffizienz</i> )
<b>Entwicklerkommunikation:</b> als Fähigkeit, seine Arbeit durch Kooperation zu verbessern (als <i>Teamentwicklereffizienz</i> )	<b>Entwicklerkommunikation:</b> als Fähigkeit, seine Entwicklungsarbeit am Agentensystem durch Kooperation zu verbessern (als <i>Entwicklungsteameffizienz</i> )
<b>Entwicklerproduktivität:</b> misst die Quantität der Arbeitsleistung (als <i>Entwicklereffizienz</i> )	<b>Entwicklerproduktivität:</b> misst die Quantität der Arbeitsleistung für das System (als <i>Entwicklungsteameffizienz</i> )
<b>Softwareressourcenniveau</b>	<b>Softwareressourcenniveau</b>
<b>Paradigma:</b> schätzt die Zweckmäßigkeit der benutzten Komponenten ab (als <i>Komponenteneffizienz</i> )	<b>Paradigma:</b> bewertet die Eignung verwendeter COTS für die Systementwicklung (als <i>COTSEffizienz</i> )
<b>Effizienz:</b> ist bezogen auf die Effektivität der Komponenten selbst (als <i>Agentenbasiseffizienz</i> )	<b>Effizienz:</b> betrifft die Entwicklung der Effizienz der beteiligten Softwarebasis (als <i>Agentensystembasiseffizienz</i> )
<b>Ersetzung:</b> dieses Maß betrifft den Aufwand zur Anpassung an neue Versionen der Basissoftware (als <i>Migrationseffizienz</i> )	<b>Ersetzung:</b> betrifft die Adaption auf die neuen Versionen und Arten der Basissoftware (als <i>Umstellungseffizienz</i> )
<b>Hardwareressourcenniveau</b>	<b>Hardwareressourcenniveau</b>
<b>Zuverlässigkeit:</b> betrifft die Zuverlässigkeit der erforderlichen Hardware (als <i>Zuverlässigkeitseffizienz</i> )	<b>Zuverlässigkeit:</b> bewertet die erforderliche Hardware, auf der das System läuft (als <i>Systemzuverlässigkeitseffizienz</i> )
<b>Hardware-Effizienz:</b> betrifft die Leistungsfähigkeit der Plattform, die der Agent benutzt (als <i>Hardwareeffizienz</i> )	<b>Hardware-Effizienz:</b> betrifft die Leistungsfähigkeit der Plattform, die das Agentensystem benutzt (als <i>Plattformeffizienz</i> )
<b>Hardwareverfügbarkeit:</b> bewertet die durchschnittliche Verfügbarkeit der verschiedenen Plattformen, die ein mobiler Agent benutzt (als <i>Verfügbarkeitseffizienz</i> )	<b>Hardwareverfügbarkeit:</b> bewertet die durchschnittliche Verfügbarkeit der verschiedenen Plattformen, die ein mobiles Agentensystem benutzt (als <i>Gewährleistungseffizienz</i> )

Die Tabellen zeigen die vielfältigen Möglichkeiten einer leistungsbezogenen Bewertung von Agentensystemen und implizieren unmittelbar Fragestellungen zum Verhältnis bzw. zur Kausalität der jeweiligen Metriken untereinander bzw. miteinander, welche an dieser Stelle nicht weiter erörtert werden.

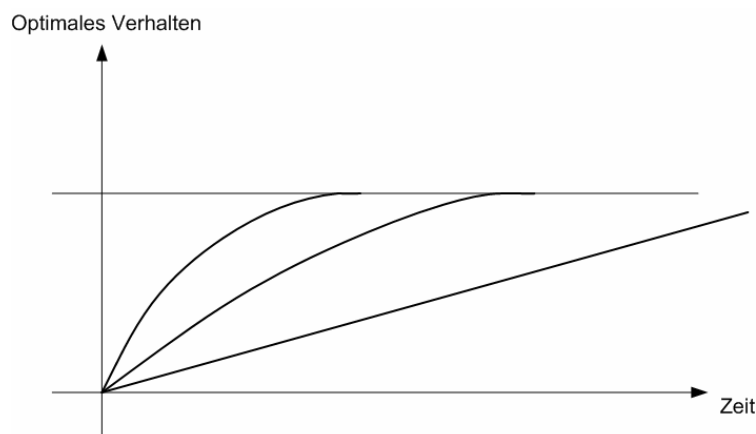
Durch das Agenten-Training sollen ausgewählte Leistungsparameter<sup>32</sup> von MAS positiv beeinflusst werden. *Als effizient wird in dieser Diplomarbeit diejenige Lern- oder Trainingsform bezeichnet, welche im Vergleich zum Ausgangszustand zu effizienteren Agenten oder MAS führt.*

<sup>32</sup> durch die Entwickler für jedes Projekt separat festzulegen

In der Praxis dient als „Messlatte“ in diesem Zusammenhang meistens die *Produkteffizienz bzw. Performance*. Zu deren Bestimmung wird üblicherweise, so auch bei den im Kapitel 6 vorgestellten Beispielen, wie folgt verfahren<sup>33</sup>: Vor jedem Experiment werden Zielgrößen (können auch Aufwand beschreiben, z. B. Kommunikationslast) definiert, die durch den Lernprozess verbessert werden sollen. Des Weiteren benötigt man Metriken, um eventuelle Veränderungen quantifizierbar zu machen. Zur Laufzeit erfolgen dann die erforderlichen Messungen der (zur Berechnung der Metriken) notwendigen Messgrößen. In der Auswertungs- und Analysephase werden zu Vergleichszwecken bzw. zur Bewertung der erreichten Performancesteigerungen bestimmte Basiswerte herangezogen, z. B. das Leistungsverhalten des MAS beim Einsatz von untrainierten Agenten. Die Betrachtung des verursachten *Mehraufwands* unterscheidet sich von Fall zu Fall. Erfolgt das Training beispielsweise automatisiert offline (Der ausgebildete Agent kommt zum Einsatz.), spielen derartige Überlegungen eine untergeordnete Rolle. Innerhalb von hoch dynamischen Umgebungen dagegen muss ein Agent gegebenenfalls online lernen. Dann ist ein erhöhter Aufwand (z. B. mehr CPU-Zeit, Arbeitsspeicher), sofern er sich nicht in den Zielgrößen niederschlägt, von Bedeutung und muss berücksichtigt (gemessen) werden.

Zur Visualisierung von Ergebnissen findet man in der Literatur oft so genannte Lernkurven (bzw. Abwandlungen davon). Dargestellt werden:

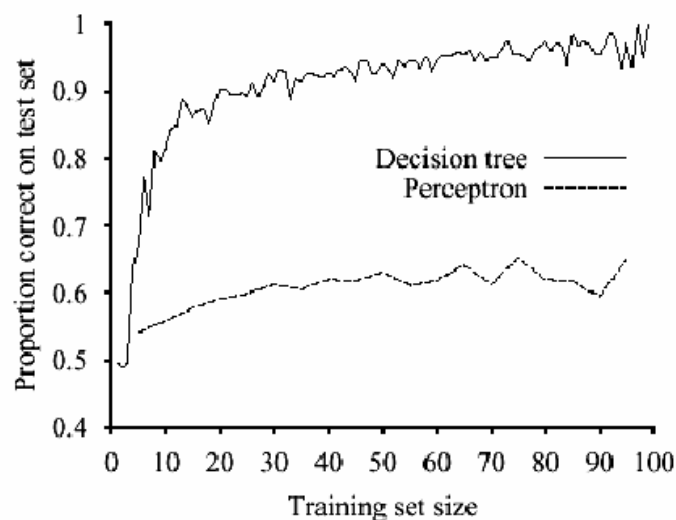
- Die Konvergenzgrenze (siehe Abb. 5-3): Wie weit nähert man sich der optimalen Lösung (dem optimalen Verhalten)?
- Die Geschwindigkeit der Konvergenz (siehe Abb. 5-3): Wie schnell wird der angestrebte Lerneffekt erreicht?
- Die Höhe des Regrets (ohne Abb.): Welche Einbußen in der Bewertung gibt es, die durch die Notwendigkeit zum Lernen entstehen?



**Abb. 5-3:** Unterschiedliche Geschwindigkeiten beim Konvergieren

<sup>33</sup> vgl. GQM-Methode in Abschnitt 5.1

Die Aussagekraft von Effizienzbetrachtungen muss genau geprüft werden, insbesondere beim Vergleich von Lern- und Trainingsmethoden. Anhand eines Experiments getroffene Aussagen können nicht ohne weiteres verallgemeinert werden. Ein einfaches Beispiel zur Bewertung der Effizienz von Lernverfahren zeigt die Abb. 5-4. Dargestellt sind die Lernkurven eines Entscheidungsbaums und eines sehr einfachen Neuronalen Netzes (NN) (Restaurantbeispiel; Russel (1995)). Effizienz ist hier gleichbedeutend mit dem Verhältnis der Anzahl von korrekt klassifizierten Fällen zur Gesamtanzahl aller klassifizierten Fälle. Im vorliegenden Szenario schneidet der Entscheidungsbaum klar besser ab. Die Behauptung, dass dem immer so ist, wäre dagegen falsch. (Die Ergebnisse sind abhängig von der Art der Attribute, der Zusammensetzung der Trainingsmenge, dem Aufbau des NN u. v. a. m.) Eine wahre Aussage könnte lauten: Die Vorhersagequalität steigt mit wachsender Trainingsmenge.



Quelle: Russel (1995)

Abb. 5-4: Lernkurve eines Entscheidungsbaums und eines Perzeptrons

### 5.3 Zusammenfassung

Die Softwaremessung von agentenbasierten Systemen steht erst am Anfang ihrer Entwicklung. In Ermangelung von anderen Standards greift man u. a. auf die ISO-Norm 9126 zurück, welche ein Qualitätsmodell für Software-Produkte definiert und innerhalb dessen Richtlinien zur Bestimmung von Qualitätsmerkmalen gibt. Zur Durchführung von zweck- und zielgerichteten Messungen empfiehlt sich die Verwendung der GQM-Methode. Neben neu zu entwickelnden Mess- und Bewertungsformen können grundsätzlich die aus dem OOSE bekannten eingesetzt werden, jedoch sind agentenspezifische Charakteristika zu beachten.

Für Effizienzmessungen von MAS wurden agentenorientierte Metriken vorgestellt. Diese können u. a. dazu genutzt werden, die durch das Training von MAS hervorgerufenen Effizienzsteigerungen zu quantifizieren. Als effiziente Trainings- bzw. Lernform wurde diejenige definiert, welche im Vergleich zu einem Basiswert<sup>34</sup> effizientere Agenten bzw. MAS „ausbildet“.

Als Probleme für eine *generellere* Bewertung der Effizienz von Trainingsverfahren wurden die folgenden Punkte identifiziert:

- Es gibt keine verbindlichen Standards im Bereich der Agenten-Messung.
- Effizienzmessungen erfolg(t)en in der Praxis immer unter einem speziellen Blickwinkel für einen speziellen Einsatzfall.
- Sie besitzen damit keine allgemeine Aussagekraft.

---

<sup>34</sup> beispielsweise untrainierte Agenten

## 6 Effiziente Trainingsmethoden in Multi-Agenten-Systemen

Gründe zur Motivation von Lernen und Training in MAS wurden bereits in der Einleitung genannt. Zum einen können sich adaptive Systeme besser an veränderte Umweltbedingungen anpassen, zum anderen führt der Lernprozess (in der Regel) zu einer Performancesteigerung der Agenten (und damit des MAS).

Der Abschnitt 6.1 gibt zunächst einige theoretische Grundlagen zum Training in MAS, einschließlich einer *kurzen*<sup>35</sup> Vorstellung von Verfahren der KI (und VKI), die für das Agenten-Training von Bedeutung sind. Das darauf folgende Unterkapitel beschäftigt sich mit den Lern- und Trainingsformen einzelner Agenten, welche die Basis für das im Anschluss betrachtete Multi-Agenten-Lernen bilden (siehe Punkte 6.2 und 6.3). Es wird dabei wie folgt verfahren. Zwischen den strukturellen Elementen eines Agenten bzw. MAS sowie den Trainingsmöglichkeiten zur Verbesserung der Leistungsfähigkeit derselben werden Verbindungen gezogen (siehe auch Punkt 4.2.3). Beispiele aus der Praxis unterlegen die getroffenen Aussagen. Aufgrund des Umfangs des Themenkomplexes kann allerdings nur ein Überblick gegeben werden (Einstieg in die Thematik).

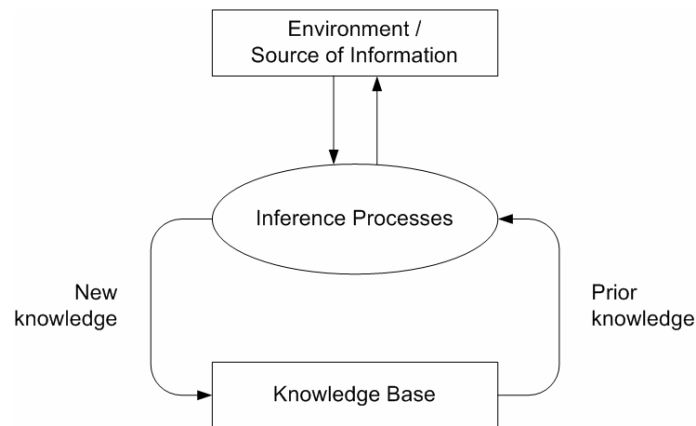
### 6.1 Theoretische Grundlagen

Eine informelle Definition des Lernens, die dem ML entlehnt ist, geben (Weis (1999a)): „The acquisition of new knowledge and motor and cognitive skills and the incorporation of the acquired knowledge and skills in future system activities, provided that this acquisition and incorporation is conducted by the system itself and leads to an improvement in its performance.“. (Mitchell (1997)) definiert Lernen folgendermaßen: „A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at T, as measured by P, improves with E.“.

Das Lernen stellt im Allgemeinen einen permanenten Prozess dar. Ein Agent nimmt über seine Sensoren Daten (z. B. Rohdaten (Messdaten) oder Kommunikation) aus der Umgebung auf (environment/source of information), die er intelligent (inference processes) verarbeitet. Das daraus neu extrahierte Wissen (new knowledge) wird Bestandteil seiner Wissensbasis (knowledge base), die wiederum sein zukünftiges Verhalten (ausgedrückt durch seine Aktionen) beeinflusst (prior knowledge). Durch die Beobachtung der Umwelt bzw. direktes Feedback kann der Agent Rückschlüsse über die Auswirkungen seiner Aktionen ziehen, womit sich der Kreis schließt.

---

<sup>35</sup> Die Ausführungen stellen keine Einführung in die KI oder VKI dar. Grundlegende Kenntnisse werden vorausgesetzt. Für ausführliche Informationen wird auf die entsprechende Fachliteratur verwiesen.



**Abb. 6-1:** Abstrakte Darstellung des Lernprozesses

(Weiss (1999a)) führen den Begriff „Lernprozess“ wie folgt ein: „The term learning process refers to all activities (e.g., planning, inference or decision steps) that are executed with the intention to achieve a particular learning goal.“. Sie nutzen ihn zur Charakterisierung zweier grundlegender Prinzipien des Lernens in MAS. Beim *zentralen Lernen* werden alle Aktivitäten des Lernprozesses durch einen einzelnen Agenten ausgeführt. (Er verhält sich so, als wäre er allein. Interaktionen zu Lernzwecken finden zwischen den Agenten nicht statt.) Unter *dezentralem Lernen* wird der entgegengesetzte Fall verstanden. Agenten, die in denselben Lernprozess involviert sind, übernehmen nur für einzelne Aktivitäten die Verantwortung. (Das Vorhandensein mehrerer interagierender Agenten ist somit zwingend erforderlich.) In einem MAS können beide Prinzipien mehrfach und parallel nebeneinander auftreten. Beispielsweise kann ein Agent zur gleichen Zeit in unterschiedliche zentrale und dezentrale Lernprozesse eingebunden sein, wobei die Lernziele übereinstimmen oder differieren können. (vgl. Weiss (1999a), siehe auch Abschnitt 6.1.2).

Folgende allgemeine Ziele werden durch das Lernen verfolgt (vgl. Wanner (2004)):

- Erwerb von deklarativem Wissen
- Entwicklung von motorischen und kognitiven Fähigkeiten durch Unterweisung und Übung
- Transformation von Wissen in eine effektivere Form
- Entdecken neuer Fakten durch Beobachten und Experimentieren

Um diese zu erreichen, wurden hauptsächlich durch die KI eine Vielzahl von Lern- und Trainingsmethoden entwickelt. Eine Kategorisierung derselben kann man auf verschiedene Art und Weise (vgl. Weiss (1999a)) vornehmen, z. B.:

- Auswendiglernen: direktes (hartes) Eincodieren von Wissen und Fähigkeiten ohne die Notwendigkeit eines späteren Schlussfolgerns oder Transformierens durch den Lernenden
- Lernen durch Anweisung oder Ratschlag: Transformation von neuem Wissen in die eigene interne Repräsentationsform und Integration mit vorhandenem Wissen und Fähigkeiten; die Anweisungen bzw. Ratschläge (neues Wissen) sind ansonsten nicht direkt zugänglich für den Lernenden
- Lernen anhand von Beispielen und aus Erfahrungen: Extraktion und Verfeinerung<sup>36</sup> von Wissen (und Fähigkeiten), wie einem generellen Modell (Konzept) oder Muster, anhand von positiven und negativen Beispielen bzw. gesammelten Erfahrungen
- Lernen durch Analogie: Sicherung von erworbenem Wissen und Fähigkeiten aus früheren Problemstellungen sowie dessen Transformation (Anwendung) auf ähnliche, ungelöste Situationen
- Lernen durch Entdeckung: Sammeln neuen Wissens und Ausprägung von neuen Fähigkeiten durch Beobachtungen und Experimente (Aufstellen und Überprüfen von Hypothesen)

Ein grundlegender Unterschied zwischen den aufgeführten Ansätzen liegt im zu betreibenden Aufwand. Dieser steigt von oben nach unten an.

Ein weiteres, oft gebrauchtes Unterscheidungsmerkmal ist das Feedback aus der Umwelt auf die Aktionen des Agenten. Aufgrund der Bedeutung des Kriteriums werden diese Lernformen ebenfalls kurz charakterisiert.

- Überwachtes Lernen (supervised learning): Dem Agenten werden konstruierte oder historische Trainingsdaten vorgelegt. *Feedback* wird in Form eines *vorgegebenen Ausgabewertes (Aktion) bei bestimmten Eingabewerten* erteilt. Das entspricht einer mathematischen Funktion, deren interne Repräsentation mit jedem Trainingsbeispiel schrittweise angepasst wird. Das Lernziel besteht in einer möglichst guten Approximation der wahren Funktion.
- Unüberwachtes Lernen (unsupervised learning): Es gibt *kein explizites Feedback* aus der Umgebung. Das Lernziel ist das Finden von gewünschten und nützlichen Aktionen durch ein „trial-and-error“-Prinzip oder einen selbstorganisierenden Prozess.
- Verstärkendes Lernen (reinforcement learning): Nach der Ausführung einer Aktion (oder mehrerer) erhält der Agent *ein explizites Feedback* (positiv/negativ/neutral) durch die Umgebung. Lernziel ist die Maximierung der erwarteten Reinforcement-Signale durch die zukünftigen Aktionen.

---

<sup>36</sup> Verbreiterung der Wissensbasis durch neues Wissen

### 6.1.1 Ausgewählte Techniken der KI

Nachfolgend werden im Überblick einige Verfahren aus der KI (und VKI) besprochen, die in MAS Anwendung finden.

#### Schlussfolgerungssysteme / Rule-based Learning

Die Verarbeitung von *Symbolen* stellt den Kern der „Klassischen KI“ dar. Das Paradigma der Symbolverarbeitung besagt, dass nur ein physikalisches Symbolsystem über die erforderlichen und ausreichenden Mittel für allgemeines intelligentes Handeln verfügt (vgl. Newell (1980)). Ausgehend von Fakten und Regeln in einer Wissensbasis leitet eine Inferenzmaschine neues (folgerbares) Wissen ab. Sie nutzt dazu formale Logiken, wie beispielsweise die Prädikaten-, Fuzzy- oder BDI-Logik<sup>37</sup> (siehe auch weiter unten im Abschnitt „Wissen(srepräsentation)“). Derartige Agenten finden also neue Regeln und modifizieren bestehende auf Basis des vorhandenen Wissens und der gesammelten Erfahrungen. Darauf aufbauend entscheiden sie, welche Aktionen auszuführen sind, um ihre Ziele zu erreichen (vgl. Williams (2003)). „Symbolverarbeitende Verfahren stehen im kognitiven Prozess auf einer relativ hohen Stufe ... entspricht die Symbolverarbeitung dem bewussten Denken mit einer expliziten Wissensdarstellung, wobei das Wissen selbst untersucht und bearbeitet werden kann.“ (Bigus (2001)).

#### Neuronale Netze

NN verfolgen einen *nicht symbolischen* Ansatz und versuchen die Vorgänge im menschlichen Gehirn nachzubilden. Ein NN besteht aus einer Menge von Knoten, die durch gewichtete Kanten miteinander verbunden sind. Es findet eine parallele Informationsverarbeitung statt. Dazu bedarf es keiner Manipulation von Symbolen, sondern einer Anpassung der Verbindungsgewichte<sup>38</sup> entsprechend den zwischen den Daten wahrgenommenen Beziehungen (vgl. Bigus (2001)). (Rummelhart (1986)) führten einen noch heute populären Lernalgorithmus für NN ein - das „Backpropagation“. NN eignen sich vor allem zur Klassifikation, Mustererkennung oder Funktionsapproximation. „Im Vergleich zu symbolverarbeitenden Systemen leisten neuronale Netzwerke kognitive Funktionen auf einem relativ niedrigen Niveau. Das Wissen, das sie sich durch den Lernprozess aneignen, wird in den Verbindungsgewichten gespeichert und kann nicht einfach untersucht oder verändert werden ... verkörpern neuronale Netzwerke eher die unbewussten Vorgänge im menschlichen Gehirn ...“ (Bigus (2001)).

---

<sup>37</sup> Ein Beispiel dafür sind die bekannten Expertensysteme.

<sup>38</sup> Die Struktur des NN dagegen ändert sich nicht.

## Planen

Die Planung umfasst mehrere Schritte. Erstens erfolgt eine Unterteilung des Problems in mehrere Teilprobleme, die leichter zu lösen sind. Zweitens sind Einschränkungen hinsichtlich der Reihenfolge der Bearbeitung zu treffen, um Teilergebnisse nicht zu entwerten. Drittens muss der Zustand der Welt im Auge behalten werden, während der Plan ausgeführt wird (Russel (1995)). Planen ist nicht mit der Suche<sup>39</sup> zu verwechseln, da die einzelnen Schritte (Aktionen) auf der Grundlage eines *internen Weltmodells*<sup>40</sup> *berechnet*<sup>41</sup> werden. Es gibt sowohl Algorithmen für das „Zentrale Planen mit verteilter Ausführung“, „Verteiltes Planen für einen zentralen Plan“ sowie das „Verteilte Planen für einen verteilten Plan“.

Agenten haben beim Planen das Problem der Behandlung unvollständiger und inkorrektur Informationen. Dazu verfügen sie über zwei Möglichkeiten (vgl. Russel (1995)):

- **Bedingtes Planen:** Ein bedingter Plan berücksichtigt alle möglichen Situationen. Mittels seiner Sensoren überprüft der Agent, welchen Teil des Plans er ausführen kann (Bedingungsüberprüfung).
- **Ausführungsüberwachung:** Der Agent beobachtet während der Ausführung des Plans die Umwelt. Scheitert dieser, versucht er ein Nachplanen auf Basis der neuen Situation.

## Verstärkendes Lernen / Reinforcement Learning (RL)

Aufgrund der relativ häufigen Nutzung des RL in den noch folgenden Beispielen zum Agenten-Training bzw. dessen allgemeiner Bedeutung für die Thematik wird ausführlicher darauf eingegangen.

Formal handelt es sich beim RL um Markov'sche Entscheidungsprozesse, die durch ein Tupel der Art  $\langle S, A, P, r \rangle$  beschrieben werden können. (S: Menge von Zuständen, A: Menge von Aktionen, P: die Wahrscheinlichkeit, sich durch Aktion  $a$  vom Zustand  $s_1$  zum Zustand  $s_2$  zu bewegen,  $r$ : Reward-Funktion). Die Umgebung erteilt in Abhängigkeit vom aktuellen Zustand  $s$  und der gewählten Aktion  $a(s)$  Reinforcement-Signale  $r(s, a(s))$  (Belohnung/Bestrafung). Der Agent versucht nun, die Summe der zukünftigen Belohnungen zu maximieren. Um dieses Ziel zu erreichen, kann man beispielsweise das Q-Learning anwenden. Gesucht wird eine Policy  $\pi^*$ , welche  $V_\gamma^\pi(s)$  für alle  $s \in S$  maximiert.

$$V_\gamma^{\pi^*}(s) = \max_{a \in A} Q_\gamma^{\pi^*}(s, a) \quad \text{für alle } s \in S$$

<sup>39</sup> Weg von einem Start zu einem Zielzustand über Entscheidungsknoten finden (Suchbäume)

<sup>40</sup> symbolische Darstellung der Umwelt

<sup>41</sup> Planen eignet sich daher auch für große Zustandsräume, in denen eine Suche aufgrund der Vielzahl von Entscheidungsknoten ineffizient ist.

$Q(s, a)$  ist eine Bewertungsfunktion, die für jedes Situation/Aktion-Paar<sup>42</sup> die diskontierte<sup>43</sup> Summe der zu erwartenden Reinforcement-Signale schätzt. Es wird die Aktion mit dem höchsten Q-Wert gewählt. Dieser wird nach jeder ausgeführten Aktion aktualisiert (Die Aktion  $a$  im Zustand  $s$  produziert den Reward  $R$  und führt zu Zustand  $s'$ ):

$$Q(s, a) \leftarrow (1 - \beta)Q(s, a) + \beta(R + \gamma \max_{a' \in A} Q(s', a'))$$

Die Lernrate  $\beta$  (im Intervall [0-1]) bringt zum Ausdruck, inwieweit der Agent seinen Erfahrungen (seinem Wissen) vertraut. Für  $\beta$  gleich 0 lernt der Agent nichts Neues, da der alte Q-Wert unverändert übernommen wird. Für  $\beta$  gleich 1 orientiert er sich nur am aktuellen Reward, sodass gesammelte Erfahrungen keine Rolle spielen. Beide Extremfälle sind unerwünscht. Optimal wäre ein am Anfang des Lernprozesses relativ hoher Wert, welcher sich, je intelligenter (erfahrener) der Agent wird, immer weiter abschwächt (Der Agent verfügt über ausreichendes spezifisches Wissen.). Um exploratives Verhalten *während der Trainingsphase* zu erreichen, werden nicht immer die „besten“ Aktionen ausgewählt, sondern in „zufälligen“ Abständen auch vermeintlich schlechtere Alternativen.

Eine spezielle Ausprägung des RL sind Classifier-Systeme, welche eine gewisse Anzahl von Regeln (Classifier)<sup>44</sup> besitzen. Jeder von ihnen ist eine bestimmte Stärke (Gewicht), im einfachsten Fall eine Zahl, zugeordnet. In Abhängigkeit vom aktuellen Umweltzustand werden alle Regeln mit passender Vorbedingung aktiviert. Diejenige mit dem höchsten Gewicht darf feuern. Nachdem die Umgebung ein Feedback (Reward) gegeben hat, werden die Werte für die Stärke der Regeln aktualisiert. Insoweit gleicht der Vorgang dem RL. Die Besonderheit liegt in der Verteilung der „Belohnung/Bestrafung“.

- Verschiedene Formen der Verrechnung des Credits:
  - Nur auf den Gewinner
  - Auf Regeln, welche die aktuelle Situation herbeigeführt haben (bucket brigade algorithm (BBA))
  - Alle Regeln, die im letzten Zyklus (Zeitspanne muss festgelegt werden) gefeuert haben (z. B. bei zeitlich verzögertem Feedback, profit sharing plan (PSP))
- Jede Regel, die “mit bietet” (aktiviert wird), zahlt eine kleine Gebotssteuer → selten gewählte Regeln, die aber eine allgemeine Vorbedingung besitzen, werden abgeschwächt
- Alle Regeln entrichten Lebenssteuer → Regeln, die nicht mehr mit bieten „veralten“.

<sup>42</sup> Situationen, zugehörige Aktionen nebst erwartetem Reward (Belohnung) werden in einer Matrix (Q-Matrix) abgespeichert.

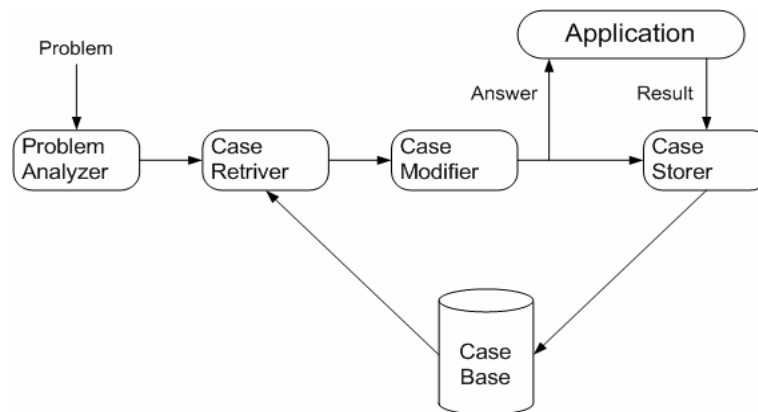
<sup>43</sup> Der Dämpfungsfaktor  $\gamma$  erlaubt es, zeitlich weiter entfernt liegendes Feedback geringer zu bewerten (je kleiner, desto eher werden schneller erreichbare Belohnungen bevorzugt).

<sup>44</sup> Beispiel: Wenn Situation  $s$  eintritt, wähle Aktion  $a$ .

In bestimmten Intervallen werden schlecht bewertete Regeln mittels genetischer Operatoren durch neue ersetzt (vgl. Weiss (1999), Klügl (2004)).

### Fallbasiertes Schließen / Case-based Reasoning

Bei dieser Methode versucht man Probleme auf der Basis von Fällen aus der Vergangenheit zu lösen (siehe Abb. 6-2). Ein Beispiel besteht dabei immer aus einer Situation (Problem), die durch verschiedene Attribute beschrieben wird, und der auszuführenden Aktion (Lösung). Tritt ein neuer Zustand ein, sucht der Agent im Speicher (seiner Wissensbasis) nach ähnlichen Fällen. Das Ähnlichkeitsmaß kann dabei frei gewählt werden (siehe auch Beispiele 1 und 9). Die abgelegten Beispiele sind unabhängig voneinander, sodass ein problemloses Hinzufügen oder Löschen sichergestellt ist. Das „Vergessen“ von alten oder selten genutzten Regeln dient der Anpassung des Agenten an veränderte Umweltbedingungen (Exploration neuen Verhaltens).



Quelle: Ohko (1996)

**Abb. 6-2:** Übersicht Fallbasiertes Schließen

### Wissen(srepräsentation)

Intelligente Agenten benötigen Wissen<sup>45</sup>, welches a priori durch den Entwickler vorgegeben oder aus Erfahrungen gewonnen (gelernt) werden muss. *Art und Umfang unterscheiden sich in Abhängigkeit von der Agenten-Architektur.* Der Aufbau von Wissensbasen ist ein nicht trivialer Vorgang, der mit verschiedenen Problemen behaftet ist. Deshalb beschäftigt sich das „Knowledge Engineering“ u. a. mit folgenden Fragen:

- Wie identifiziert und gewinnt man relevantes Wissen?
- Wie überführt man relevantes Wissen effizient in eine für den Computer bearbeitbare Darstellung?

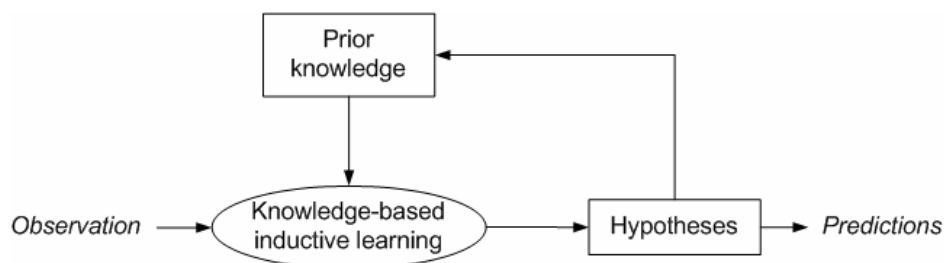
<sup>45</sup> in unterschiedlichem Umfang, abhängig von der Art der Architektur (siehe Kapitel 2.3)

- Wie tauscht man Wissen aus und modifiziert die vorhandene Wissensbasis?

Für die Verarbeitung von Wissen wäre es am einfachsten, wenn der Computer die natürliche Sprache verstehen würde. Eine Schwierigkeit dabei ist jedoch deren Mehrdeutigkeit. Aufgrund dessen entwickelte man formale Logiken, die dieses Problem vermeiden. Darauf aufbauende Programmiersprachen (Lisp, Prolog, u. a.) können in dieser Form codiertes Wissen auswerten und modifizieren. Für die interne Darstellung von Wissen wurden verschiedene Möglichkeiten gefunden (vgl. Bigus (2001))<sup>46</sup>.

- **Prozedurale Darstellung:** Bei dieser Repräsentationsart werden Tatsachen (Konstanten, Variablen) festgelegt sowie die Abläufe von Operationen zu deren Nutzung und Veränderung<sup>47</sup> definiert. Schwäche: hart programmierte Logik, Wissen und seine Bearbeitung sind untrennbar verbunden; Diesen Nachteil vermeiden die folgenden *deklarativen Ansätze*<sup>48</sup>.
- **Relationale Darstellung:** Das Wissen wird in relationalen Datenbanken als Tupel abgelegt. Diese Form ist sehr flexibel, eignet sich aber nicht zur Abbildung von komplexen Beziehungen zwischen Objekten, wie sie in der realen Welt auftreten.
- **Hierarchische Darstellung:** Man legt den Schwerpunkt auf die Beschreibung von Objektarten und deren Beziehungen zueinander (siehe auch Punkt 3.5/Ontologien). Objektorientierte Programmiersprachen und Datenbanken nutzen Mechanismen wie Klassen oder Vererbungshierarchien, um Wissen kompakt darzustellen.

Die Idee der Verwendung von Hintergrundwissen beim Lernen veranschaulicht Abb. 6-3. Auf der Basis von bereits vorhandenem Wissen und der Beobachtung der Umwelt stellt der Agent Hypothesen auf, die er testet. Die wahrgenommenen Ergebnisse (Veränderungen in der Umwelt) fließen über die Wissensbasis wiederum in den Lernprozess ein.



Quelle: Russel (1995)

**Abb. 6-3:** Schema eines Lernprozesses, der Hintergrundwissen nutzt und ansammelt

<sup>46</sup> Auf nicht symbolische Ansätze wird nicht eingegangen, dazu siehe beispielsweise NN (Punkt 6.1).

<sup>47</sup> z. B. ein in einer Skriptsprache geschriebenes Computerprogramm

<sup>48</sup> Der Benutzer definiert nur noch Fakten, Regeln und Beziehungen.

Zwei Formen des wissensbasierten Lernens sind das Explanation-based Learning (EBL) und die Inductive Logic Programming (ILP). Das EBL versucht *durch* Hintergrundwissen zu erklären, warum einzelne Beispiele (Aktionen) erfolgreich waren oder nicht. Durch Verallgemeinerung dieser Aussagen gelangt man zu neuen Hypothesen. Dagegen versucht die ILP auf der Basis einer Menge von Trainingsbeispielen *zusammen mit* dem Hintergrundwissen neue Hypothesen abzuleiten. Der Erfolg derartiger Ansätze ist stark abhängig von der Qualität (Vollständigkeit, Aussagekraft, Widerspruchsfreiheit) der Wissensbasis.

Man unterscheidet weiterhin zwischen *lokalem*<sup>49</sup> und *globalem* Wissen. Während Ersteres von Agent zu Agent unterschiedlich sein und erlernt werden kann, muss Letzteres bei Nichtvorhandensein durch Kommunikation in Erfahrung gebracht werden. Die Nutzung von Wissen kann den Lernvorgang beschleunigen, da falsche Hypothesen aussortiert und einfachere (weniger komplexe) zur Erklärung der Umwelt gefunden werden können.

### 6.1.2 Besonderheiten des Trainings in MAS

In MAS können Agenten miteinander kommunizieren, ihre Aktionen koordinieren oder künstliche Organisationen bilden, innerhalb derer sie soziale Rollen ausfüllen. Das bietet neue Wege für das Lernen. Im Folgenden werden Besonderheiten des Trainings in MAS beschrieben. Diese stellen auch gleichzeitig Unterschiede zwischen dem „single-agent learning“ und „multi-agent learning“ dar.

#### Multiplikationseffekt

Führen alle Agenten eines MAS einen (nicht notwendigerweise gleichen) Lernalgorithmus aus, lernt das System ebenfalls. Vom „multi-plied learning“ spricht man, wenn die Agenten bei diesem Prozess unabhängig voneinander sind. Jeder von ihnen verfolgt dabei sein eigenes Lernziel. Gewöhnlich ergänzen sich diese; dabei handelt es sich aber um einen auftretenden Nebeneffekt. Der Agent handelt als „Generalist“, d. h. er besitzt die Fähigkeit, alle notwendigen Aktivitäten selbst auszuführen. Wenn Kommunikation zwischen Agenten auftritt, beschränkt sich diese auf den Austausch des Inputs und/oder Outputs eines Lernvorgangs und ist nicht Bestandteil desselben.

Von Vorteil ist dies zum Beispiel bei einer Gruppe von identischen Software-Agenten (Robotern). Der Ausfall einer Einheit fällt dann nicht so sehr ins Gewicht (Robustheit durch Redundanz). Ein System von verschiedenen Agenten (unterschiedliches Wissen/Fertigkeiten) kann dagegen eine größere Palette von Aufgaben bearbeiten als ein einzelner Agent. Besonders im Bereich von MAS, die unterhalb des „knowledge levels“ arbeiten, sind die

---

<sup>49</sup> gilt nur an einem spezifischen Ort und zu einer spezifischen Zeit

Multiplikationseffekte gut zu beobachten. Beispielsweise steigt die Leistungsfähigkeit eines Neuronalen Netzes mit der Anzahl der Neuronen (obwohl ein Neuron kaum als Agent bezeichnet werden kann).

### **Divisionseffekt**

Im Fall des „divided learning“ wird eine Aufgabe unter mehreren Agenten aufgeteilt (ein gemeinsames Lernziel). Dies kann in Abhängigkeit von funktionalen Aspekten oder der Verteilung der Daten geschehen. Das Splitten nimmt typischerweise der Programmierer vor und ist nicht Bestandteil des Lernprozesses. Der Agent handelt gewissermaßen als „Spezialist“. Kommunikation wird genutzt, um Teilergebnisse zusammenzuführen. Ebenso wie beim „multi-plied learning“ entsteht sie nicht als Folge des Lernens, sondern wird vielmehr a priori und im Detail durch den Entwickler festgelegt.

Die Aufteilung der Arbeit bringt verschiedene Vorteile mit sich. Zum einen wird der Entwurf von Agenten einfacher<sup>50</sup>, zum anderen bietet dieses Vorgehen mehr Flexibilität. So können Agenten z. B. auf verschiedenste Weise miteinander kombiniert werden (Das System bildet neue Fähigkeiten aus.).

### **Interaktionseffekt**

Im Gegensatz zum Multiplikations- und Divisionseffekt stellt die Kommunikation zwischen den Agenten beim Interaktionseffekt eine dynamische Aktivität dar und findet auf hohem abstraktem Niveau statt. Andere Agenten, die in denselben Lernprozess involviert sind, werden explizit modelliert. Das Lernen erfolgt schrittweise, im Sinne einer gemeinsamen, auf „Verhandlungen“ basierenden Suche nach einer Lösung für die gemeinsame Aufgabe. Der Agent handelt nicht als „Generalist“ oder „Spezialist“, sondern vielmehr als „Regulator“ und „Integrator“. Er beeinflusst den Weg des Lernens und versucht gleichzeitig die verschiedenen Ansichten<sup>51</sup> zusammenzubringen. Durch die intensive Interaktion untereinander (basierend auf einer komplexen Kommunikation) haben Agenten die Möglichkeit, ihr unterschiedliches Wissen zur Lösung des gestellten Problems einzusetzen. Im Unterschied zum „divided learning“ ist ein Agent nicht für eine spezifische Teilaufgabe zuständig, sondern arbeitet in einer Gruppe von Agenten an der Bewältigung der Gesamtaufgabe.

Der Interaktionseffekt ermöglicht eine flexiblere, aber auch anspruchsvollere Koordination der Aktivitäten von Agenten. Des Weiteren ist ein *Wissenstransfer* zwischen den Agenten möglich.

---

<sup>50</sup> Die Dekomposition führt zu „schlankeren“ Agenten, monolithische Software wird vermieden.

<sup>51</sup> der anderen beteiligten Agenten

## Psychologische Effekte

Die aufgeführten Effekte können natürlich auch gemeinsam (gleichzeitig) in einem MAS (Lernprozess) auftreten. Vergleichbar wäre das mit der Zusammenarbeit von Menschen in einer Arbeitsgruppe (human-human collaborative learning). Bei dieser Form des Lernens stehen kognitive Prozesse im Vordergrund. Der Mehrwert entsteht („emerges“) durch Vorgänge wie dem gemeinsamen Lösen von Problemen, dem Auflösen von Konflikten, der gegenseitigen Beeinflussung, Erklärungen, Beweisen usw. Auf diesem Gebiet wurde bis dato nur wenig geforscht, es rückt erst jetzt in den Blickpunkt des Interesses (vgl. Weiß (1999b)).

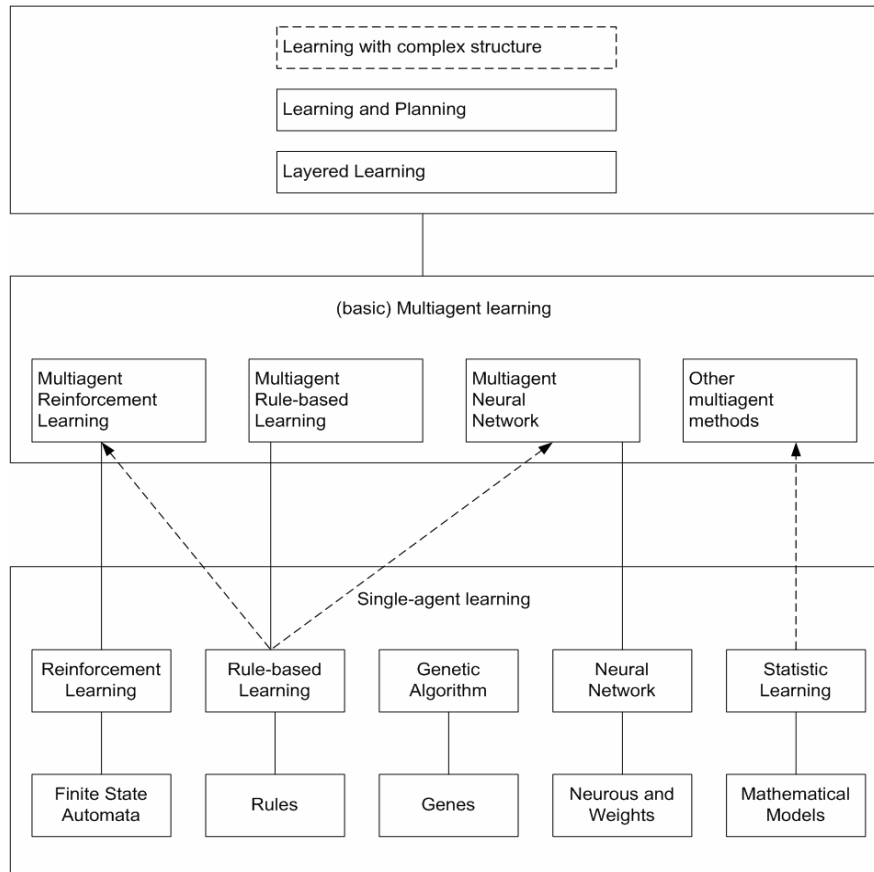
### 6.1.3 Evolution der Trainingsverfahren

(Williams (2003)) unterteilen das Lernen und Training in MAS in:

- pure single agent learning (SAL): Es handelt sich um Verfahren des ML. Der Agent interagiert „lediglich“ mit der Umwelt (nicht mit anderen Agenten), um die für das Lernen notwendigen Informationen zu beziehen.
- multiagent learning (MAL) in a weak sense: Der Agent lernt durch das Beobachten anderer Agenten, ohne jedoch direkten Kontakt aufzunehmen.
- multiagent learning in a strong sense: Die Interaktion mit der Umwelt und anderen Agenten stellt das wesentliche Element des Lernprozesses dar.

Aufbauend auf dieser Einteilung diskutieren sie zwei Evolutionsrichtungen: die Entwicklung vom SAL zum MAL sowie von einfachen zu komplexen Lernverfahren. Die Abb. 6-4 fasst die Aussagen zusammen. (Durchgezogene Linien repräsentieren die aktuelle Forschung, gestrichelte mögliche zukünftige Arbeiten.)

Das SAL bildet die Grundlage für das MAL. Der Unterschied zwischen beiden Formen besteht in der Existenz weiterer Agenten und den sich daraus ergebenden neuen Möglichkeiten für das Lernen (siehe auch Abschnitt 6.1.2.). Auf der Ebene der Algorithmen ähneln sie sich dagegen sehr.



Quelle: Williams (2003)

**Abb. 6-4:** Entwicklungsformen des Lernens in MAS

Komplexe Probleme allerdings lassen sich mit den „einfachen“ Lernverfahren des MAL nicht effizient lösen. Deshalb werden „fortgeschrittene“ Ansätze, wie das „Layered Learning“<sup>52</sup>, die Verbindung von Planen und Lernen oder gänzlich neu zu entwickelnde Methoden (z. B. in Anlehnung an das in Abschnitt 6.1.2 erwähnte human-human collaborative learning) benötigt.

Generell lassen sich gegenwärtig drei Evolutionsstrategien feststellen:

- MAS, in denen jeder Agent seinen eigenen Lernalgorithmus besitzt und von Koordination, Kooperation oder Wettbewerb profitiert (Basistyp eines lernenden MAS); Bisher wurden vor allem homogene Systeme untersucht, nicht so sehr heterogene.
- Einzelner Agent mit verschiedenen Lernmodulen; Der Agent entscheidet selbständig, welches Verfahren zum Einsatz kommt oder probiert sie der Reihe nach aus.
- Anwendung bzw. Entwicklung komplexer Lernmethoden

<sup>52</sup> Layered Learning: Hierarchischer Ansatz, siehe z. B. (Stone (2000)); Auf Basis einer Aufgabenerlegung finden unterschiedliche, an die jeweiligen Erfordernisse der Ebene angepasste, Lernverfahren Anwendung. Die Ergebnisse einer Stufe werden von der nächsten als Basis ihrer Arbeit weiterverwendet (von unten nach oben).

## 6.2 Training eines einzelnen Agenten

### 6.2.1 Aktionsselektion (Agent Operation)

Um seine (vorgegebenen) Ziele zu erreichen, stehen einem Agenten normalerweise mehrere Handlungsalternativen mit unterschiedlicher Güte zur Auswahl. Damit steht ein Agent vor der entscheidenden Frage, welche Aktion(en) er als nächstes ausführen soll. Unterstellt man ihm rationales Handeln, wird er bestrebt sein, die „beste“<sup>53</sup> Möglichkeit zu finden. Für die Beurteilung der Konsequenzen von Aktionen benötigt der Agent daher eine Bewertungsfunktion. Entweder hat er diese als Domänenwissen durch den Entwickler mitbekommen oder er lernt sie aus gesammelten Erfahrungen<sup>54</sup>. Besitzt ein Agent a priori alle Informationen, braucht er nicht zu Lernen, sondern wählt einfach die korrekte Aktion aus. In den meisten Fällen ist das aber illusorisch.

Rahmenbedingung für Aktionen von Agenten (Lettmann (2000)):

- incompatibility constraint: Aktionen können unverträglich sein, d.h. die Ausführung einer Aktion führt zu einer Veränderung der Umgebung, sodass die Ausführung anderer Aktionen beeinträchtigt bzw. sogar verhindert wird.
- local information constraint: Ein Agent hat nur lokale Informationen über die Umgebung. Diese können unvollständig und von Agent zu Agent verschieden sein.

Auf die Unterschiede bei der Aktionsselektion zwischen den verschiedenen Agenten-Architekturen sei noch einmal verwiesen (siehe Abschnitt 2.3).

#### Beispiel 1 – Persönliche Assistenten / Email-Filter:

*Collaborative Interface Agents (Lashkar (1994))*

Der Email-Filter MAXIMS gehört zur Gruppe der Interface-Agenten und soll den User beim Management seiner Emails unterstützen. Verfügbar ist das Programm für den Apple Macintosh als Erweiterung der Email-Software „Eudora“. MAXIMS beobachtet permanent die Handhabung von Emails durch den Nutzer. Dabei speichert das Programm Situation/Aktion-Paare. Eine Situation wird beschrieben durch Features wie Sender, Empfänger, Betreff usw. Unter Aktionen sind z. B. Lesen, Antworten, Weiterleiten oder Löschen von Nachrichten zu verstehen.

---

<sup>53</sup> Für das Attribut „Beste“ müssen von Fall zu Fall spezifische Kriterien definiert werden. Es braucht sich dabei nicht immer um das Optimum zu handeln, welches oft auch gar nicht durch den Agenten ermittelbar ist.

<sup>54</sup> Die Aktionsauswahl per Zufall ist keine wirkliche Alternative.

### Training:

Mit Hilfe von Beispielen aus der Vergangenheit soll vorausgesagt werden, wie neu eingetroffene Emails im Sinne des Nutzers zu bearbeiten sind (Muster in den Entscheidungen des Users erkennen). Das Training wird umgesetzt durch fallbasiertes Schließen in Verbindung mit RL.

*Algorithmus:* Beim Eingang einer neuen Email versucht MAXIMS aufgrund seines „Gedächtnisses“, die wahrscheinliche Aktion des Users vorherzusehen und sie gegebenenfalls selbständig auszuführen. Dazu sucht das Programm zuerst im Speicher nach ähnlichen Situationen. Als Distanzmaß (Ähnlichkeitsmaß) zwischen dem neu zu klassifizierenden Fall und den vorhandenen dient eine gewichtete Summe der Abstände der jeweils korrespondierenden Features. Zusätzlich fließen Korrekturen<sup>55</sup> für spezifische Werte in Feldern (bestimmte Situationen) mit in die Berechnung ein. Diese können a priori festgelegt<sup>56</sup> oder im Laufe der Zeit durch das RL bestimmt werden (anhand von Feedback des Nutzers bei falsch klassifizierten Beispielen). Eine exakte Beschreibung des Vorgehens findet sich in (Kozierok (1993)). Die am zutreffendsten Situationen (hier fünf) bilden schließlich die Grundlage für die Voraussage. Des Weiteren bestimmt MAXIMS einen Konfidenzlevel für seinen Vorschlag. In Abhängigkeit davon wird die Aktion selbständig ausgeführt (do-it; >0,6), eine Bestätigung des Nutzers eingeholt (tell-me; >0,1) oder gar nichts getan (<=0,1).

### Ergebnisse:

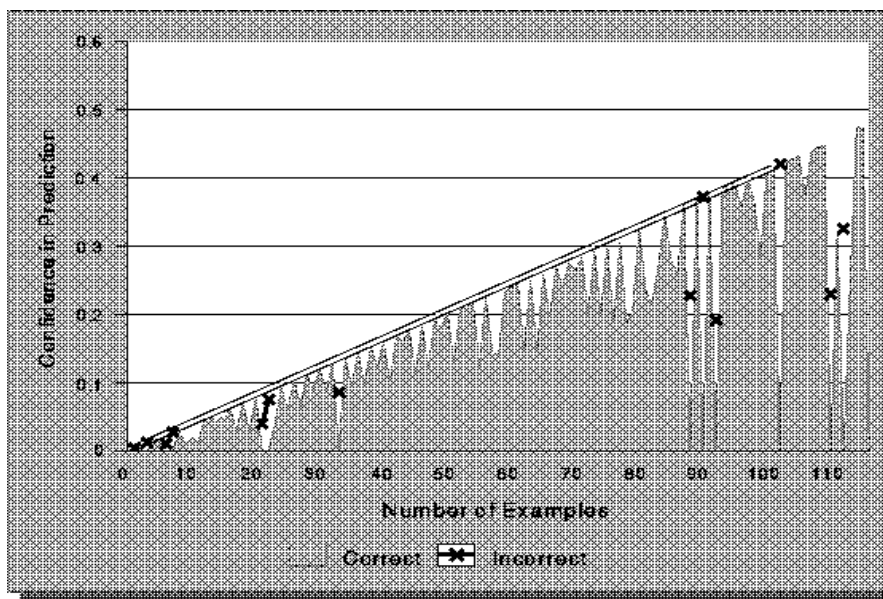
Getestet wurde MAXIMS durch zwei Studenten der Arbeitsgruppe von Patty Maes über einen Zeitraum von 3 Tagen. Im Durchschnitt erhielten beide 100 Nachrichten pro Tag. Einer der beiden Studenten (Hobbes) verfügte über einen bereits trainierten Agenten, während der des anderen (Calvin) gar keine Erfahrungen (Wissensbasis, Beispielfälle) besaß.

Die Performance der Lernmethode ist stark abhängig von der vorhandenen Wissensbasis (# Situation/Aktion-Paare) und sicherlich in gewissem Maße vom User selbst (Gibt es überhaupt Muster?). Die Abb. 6-5 zeigt die Ergebnisse für Calvins Agenten. Erst bei 30-40 Trainingsbeispielen wird die „tell-me“ Grenze überschritten. Der Agent lernt stetig mit zunehmender Anzahl von Fällen. Die Einschnitte zu Beginn des Versuchs entstehen aufgrund fehlender Informationen, später weisen sie auf eine Änderung des Nutzerverhaltens hin. Eine Verbesserung lässt sich erreichen, wenn auf der Basis von anzugebenden Präferenzen des Users ein Standardprofil geladen werden kann. Eine weitere Möglichkeit wäre die Zusammenarbeit von Hobbes und Calvins Agenten. (Die Agenten versuchen sich gegenseitig in neuen oder schwer einzuschätzenden Situationen durch Ratschläge [Informationsaustausch] zu helfen.)

---

<sup>55</sup> kleiner, konstanter Betrag wird mit einem positiven oder negativen „Modifier“ multipliziert, um die Gewichte eines Features zu korrigieren

<sup>56</sup> z. B. eine hohe Priorität für den Absender eines Vorgesetzten



Quelle: Lashkari (1994)

Abb. 6-5: Auswertung Experiment 1

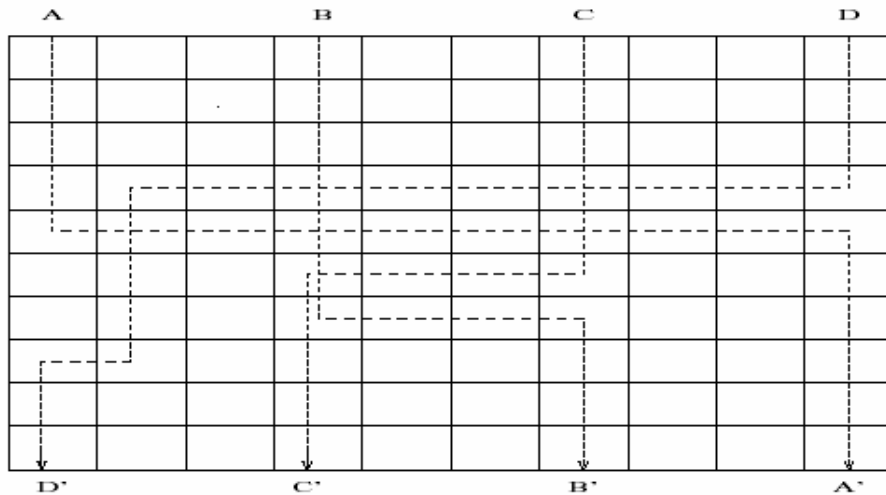
## Beispiel 2 – Roboter-Navigation:

*Individual learning of coordination knowledge (Sen (1998))*

Die Aufgabe besteht für vier simulierte Roboter (A, B, C, D) darin, von ihren Startpositionen aus schnellstmöglich zum Ziel zu gelangen, ohne dabei mit anderen zu kollidieren. Die Abb. 6-6 zeigt die Gitterwelt der Agenten und potentielle Pfade (mit Kreuzungspunkten), die während des Trainings „abgelaufen“ werden könnten. Nach jedem Schritt (jeweils 1 Einheit, vorwärts, rückwärts, links, rechts) erhalten die Roboter ein Feedback (ausgedrückt als ganze Zahl: +1 bei Bewegung zum Ziel hin, -1 im entgegengesetzten Fall, 0 bei gleich bleibender Entfernung zum Ziel, Kollisionen werden mit -10 bis -5 bestraft [je nach Art des Zusammenstoßes]).

### Training:

Wie weiter oben erwähnt, sollen die Roboter den optimalen Weg zu ihrem eigenen Zielpunkt finden, ohne dabei zusammenzustößen. Die Agenten werden immer wieder mit der gleichen Aufgabe konfrontiert. Dabei lernen sie aus ihren gemachten Erfahrungen mittels RL (Q-Learning) sowie in einem zweiten Versuch durch ein Classifier-System (mit BBA-Algorithmus).



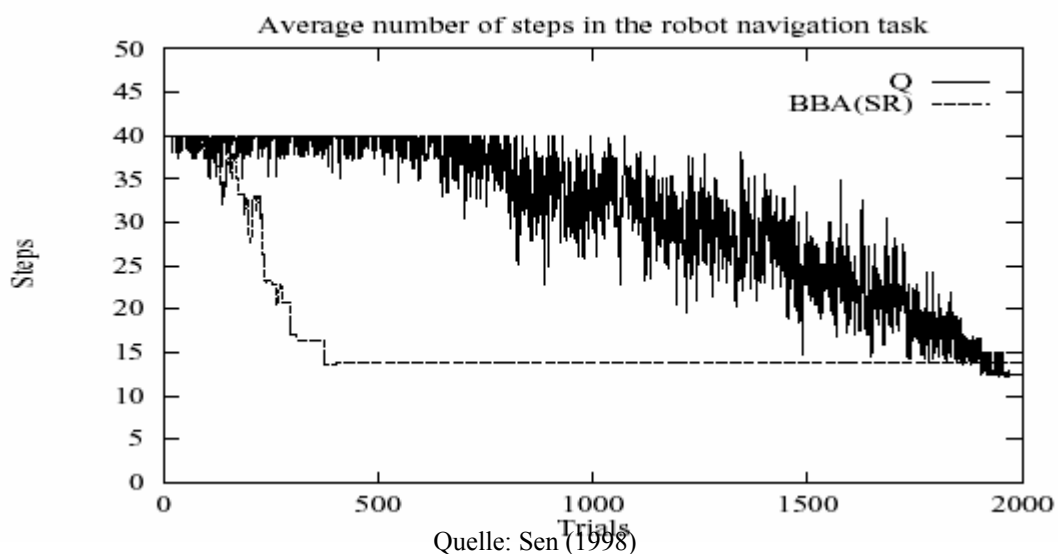
Quelle: Sen (1998)

**Abb. 6-6:** Gitterwelt

*Algorithmus:* Die Algorithmen unterscheiden sich nicht von den in Punkt 6.1.1 vorgestellten. Ein Nachtrag: Die Formel für das Update der Gewichte der Regeln des BBA-Algorithmus lautet:  $S_{t+1}(c_i, a_i) = (1 - \alpha) * S_t(c_i, a_i) + \alpha * (R + S_{t+1}(c_j, a_j))$ . ( $\alpha$ : Lernrate des Systems, c: Bedingungen für das Feuern einer Regel, a: gewählte Aktion, i: aktueller Classifier, j: nächster Classifier)

#### Ergebnisse:

Im Experiment wurden verschiedene Parameter verwendet. Für die in der Abb. 6-7 dargestellten Ergebnisse wurden  $\beta = 0,5 / \gamma = 0,8$  (Q-Learning) sowie  $\alpha = 0,1$  (Classifier-System) gewählt.



Quelle: Sen (1998)

**Abb. 6-7:** Auswertung Experiment 2

Der Graph ist spezifisch für *dieses* Experiment<sup>57</sup>. Das Classifier-System schneidet eindeutig besser ab (konvergiert schneller gegen eine fast optimale Lösung). Auffällig ist die relativ große Anzahl von Versuchen, bis die Roboter eine zufrieden stellende Performance erzielen. (Anmerkung: Bei  $\alpha = 0,5$  sind beide Verfahren gleich performant, bei  $\beta = 0,1$  verschlechterte sich das Q-Learning extrem.)

## 6.2.2 Agentenwissen (Agent Knowledge)

In den Abschnitten 2.3 und 6.1.1 wurde auf die Nutzung und Bedeutung von Wissen, vor allem für kognitive Agenten, hingewiesen. In der traditionellen KI besteht Lernen typischerweise darin, Schlussfolgerungen aus dem vorhandenen Wissen zu ziehen und für zielstrebiges Handeln zu nutzen. Zum Beispiel kann ein einmal erfolgreich umgesetzter Plan für eine spätere Wiederverwendung abgespeichert werden. Nach (Maes (1994)) lernen kognitive Agenten allerdings sehr selten induktiv neue Informationen bzw. korrigieren ihren Wissensspeicher auf Basis von Experimenten oder dem Feedback aus der Umgebung. (Vielmehr werden aktuelle Sensordaten „nur“ zur Überprüfung von Bedingungen genutzt.) Gründe dafür sind (Auffassung des Autors):

- Schwierigkeiten bei der Auswahl relevanter Informationen und deren Einordnung in einen Kontext
- Ein allgemeines Problem von regelbasierten (wissensbasierten) Systemen besteht im Update der Regeln bei Erhalt von neuen Informationen (Fakten). Da die Wissensbasis konsistent sein muss, dürfen sich „neue“ und „alte“ Regeln nicht widersprechen. Stellen sich einige der vorhandenen als unwahr heraus, müssen sie korrigiert oder gelöscht werden und es bedarf zusätzlich einer Überprüfung von abhängigen Fakten (nicht monotones Schlussfolgern) (vgl. Bigus (2001)).
- Ein Agent verfügt oft nur über lokales Wissen.
- Die schlechte Skalierung (Performance) des Agenten bei großen Regelmengen

Mit der Nutzung einer expliziten Wissensbasis und dem Anwenden von Schlussfolgerungstechniken sind aber auch Vorteile beim Lernen verbunden. Das Wissen kann in einem allgemeinen Kontext verwendet werden. Einmal abgeleitete Regeln können auf ähnliche Situationen (Zielstellungen) übertragen werden. Durch den Austausch oder die Erweiterung der Wissensbasis wird der Agent schnell in anderen Domänen einsetzbar. Des Weiteren lassen sich komplexe Zusammenhänge nicht ohne weiteres automatisch erlernen.

---

<sup>57</sup> Bei Experimenten in einer anderen Domain („Resource Sharing“) erzielten beide Verfahren eine ähnliche (fast identische) Performance.

Wissen kann, wie schon mehrfach erwähnt, einem Agenten in Form einer Wissensbasis mitgegeben oder aus Erfahrungen erlernt werden. Weiterhin ist es natürlich möglich, durch Data Mining aus Rohdaten (Sensorinformationen) neues Wissen zu generieren oder mit anderen Agenten Wissen auszutauschen (im MAS). Inwiefern sich all diese Methoden zur Verwendung anbieten, hängt von verschiedenen Faktoren ab. Dazu zählen z. B. das Einsatzgebiet, der Typ der Umgebung oder die geforderten Eigenschaften eines Agenten.

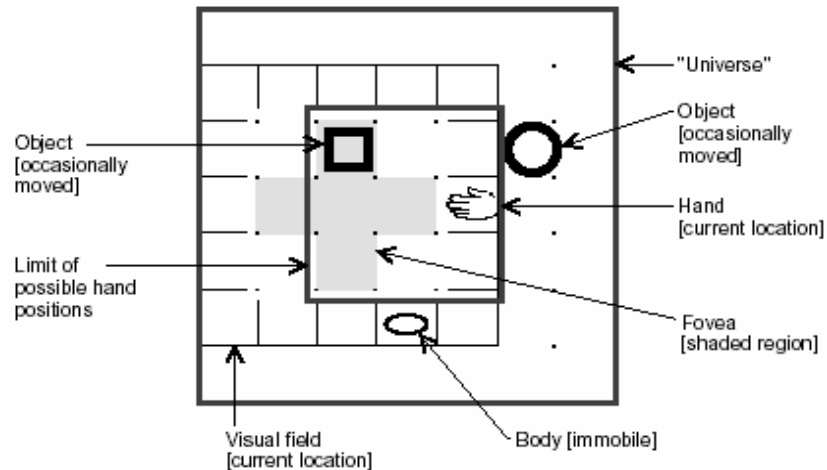
### **Beispiel 3: Statistische Modelle lernen**

*Paying Attention to What's Important: Using Focus of Attention to Improve Unsupervised Learning (Foner (1994))*

Adaptive autonome Agenten müssen in der Lage sein, die Konsequenzen ihrer Aktionen zu beurteilen. Das Wissen um die Zusammenhänge zwischen seinem Handeln und den Reaktionen der Umgebung versetzt den Agenten in die Lage, sein Verhalten den Umweltbedingungen entsprechend anzupassen und dadurch seine Leistungsfähigkeit zu verbessern. Analysiert man allerdings Aktionen im Zusammenhang mit den Veränderungen der Sensordaten, beträgt die Komplexität  $O(n^2)$  und ist somit für nicht-triviale Anwendungen<sup>58</sup> sehr schlecht handhabbar. Deshalb sollte sich der Agent darauf konzentrieren, nur Relevantes zu lernen. Im o. g. Paper bewegt sich der Software-Agent (ähnlich einem Kleinkind) in der nachstehend abgebildeten Welt (siehe Abb. 6-8). Dabei speichert er Situationsbeschreibungen (auf der Grundlage von Sensordaten), ausgeführte Aktion und deren Auswirkung auf die Umgebung (wiederum durch Sensorinformationen ermittelt) in so genannten Schemata ab. Ein Beispiel könnte lauten: Wenn meine Hand die Position (3,4) „meldet“ und ich bewege sie eine Position zurück, „fühle“ ich eine Berührung von Hand und Mund. Zusätzlich werden Statistiken (Eintrittswahrscheinlichkeiten) ermittelt. Der Agent kann in diesem Experiment nicht das gesamte „Universum“ überblicken, sondern immer nur einen begrenzten Ausschnitt (begrenzte Reichweite der Sensoren, diese übermitteln nur bestimmte Informationen).

---

<sup>58</sup> MAS werden oft in komplexen, dynamischen Umgebungen eingesetzt, wo schnelle Reaktionen auf veränderte Bedingungen erforderlich sind.



Quelle: Foner (1994)

**Abb. 6-8:** Testumgebung für Beispiel 3

### Training:

„Der Agent besitzt ein probabilistisches Modell über die Auswirkungen einer bestimmten Aktion zu einem bestimmten Zeitpunkt. Im Rahmen des Lernprozesses beobachtet der Agent Änderungen innerhalb der Umwelt und bildet Korrelationen<sup>59</sup> zwischen einzelnen Situation/Aktion-Paaren und bestimmten Ergebnissen. Mit deren Hilfe werden die Ergebnisse, samt Wahrscheinlichkeiten, aller in Frage kommenden Schemata aktualisiert.“ (Brenner (1998)). Es handelt sich um unüberwachtes (selbständiges) Lernen. Die Ausführung der Aktionsselektion ist vom eigentlichen Lernvorgang getrennt.

*Kern des Algorithmus (vereinfacht):* Um nicht alle Sensordaten mit allen Schemata verknüpfen zu müssen, wird ein „Pruning“ durchgeführt. Dazu entwickelten Foner und Maes die „perceptual selectivity“ und „cognitive selectivity“.

- *perceptual selectivity:* Es werden beim Update von Statistiken nur Sensordaten berücksichtigt, die sich in den letzten 2 Zeiteinheiten signifikant verändert haben (Ziel ist die Begrenzung des Sensorinputs auf Relevantes). Außerdem werden nur Statistiken modifiziert, die auch Voraussagen über die entsprechenden Größen machen.
- *cognitive selectivity:* Für die Entscheidung, ob ein neues Schema abgespeichert wird oder nicht, werden nur Sensordaten berücksichtigt, die sich in der letzten Zeiteinheit signifikant verändert haben. Ähnliches gilt für die Schemata. Deren Statistiken müssen sich ebenfalls in dieser Zeitspanne geändert haben. Stellt man fest, dass neue zusätzliche Informationen (z. B. ein neues Objekt im Sichtfeld) hinzugekommen sind, legt man ein weiteres Schema an (Ziel ist die Begrenzung der Schemata).

<sup>59</sup> Die statistischen Mechanismen werden in (Drescher (1991)) beschrieben.

### Ergebnisse:

Es wurden verschiedene Lernalgorithmen (Pruningverfahren) getestet. Die Resultate zeigt die Abb. 6-9. Die erste Zeile der Tabelle steht für den ursprünglichen Algorithmus von (Drescher (1991)), der keinerlei Begrenzungen des Sensorinputs oder der anzulegenden Schemata vornimmt, die letzte Zeile für das oben beschriebene Verfahren. Es ist offensichtlich, dass Letzteres weniger Schemata produziert. Des Weiteren ist der Anteil derer, welche statistisch zuverlässige Aussagen (der Grenzwert ist zu spezifizieren) enthalten (Spalte T/R), höher.

Algorithm				Learning			Work required			Facts per work unit		
Spinoff selectors		Statistic selectors		Schemas			Inner loops ( $\times 10^6$ )			Reliable schemas over		
Items	Schemas	Items	Schemas	Total	Rel	T/R	Spin	Stat	Both	Spin	Stats	Both
AIN	ASN	AIN	ASN	1756	993	1.77	533	533	1066	1.9	1.9	0.9
AIN	ASN	CINIH	ABSPSDUCI	1135	403	2.82	398	12	410	1.0	33.6	1.0
AIN	ASN	AIN	ABSPSDUCI	1110	518	2.14	391	55	446	1.3	9.4	1.2
CIN	ASN	AIN	ASN	1693	948	1.79	44	524	568	21.5	1.8	1.7
CIN	SWRUS	AIN	ASN	1395	791	1.76	2	463	466	316.4	1.7	1.7
CIN	ABSPSDUCI	AIN	ASN	1622	924	1.76	15	510	525	61.6	1.8	1.8
CIN	ASN	AIN	ABSPSDUCI	1110	506	2.19	33	54	87	15.3	9.4	5.8
CIN	ABSPSDUCI	AIN	ABSPSDUCI	1110	506	2.19	10	54	64	50.6	9.4	7.9
CIN	ABSPSDUCI	CINIH	ASN	1366	643	2.12	13	64	77	49.5	10.0	8.4
CIN	ASN	CINIH	ABSPSDUCI	1136	399	2.85	34	12	46	11.7	33.3	8.7
CIN	SWRUS	AIN	ABSPSDUCI	1102	498	2.21	1	53	54	415.0	9.4	9.2
CIN	SWRUS	CINIH	ASN	1353	688	1.97	2	64	66	275.2	10.8	10.3
CIN	ABSPSDUCI	CINIH	ABSPSDUCI	1136	399	2.85	10	12	22	40.7	33.3	18.3
CIN	SWRUS	CINIH	ABSPSDUCI	1134	398	2.85	1	12	13	331.7	33.2	30.2

Figure 4. Various selectors versus number of schemas and total computation, for 5000 iterations.

Quelle: Foner (1994)

Abb. 6-9: Auswertung Experiment 3

## 6.3 Training von Multi-Agenten-Systemen

### 6.3.1 Agenten-Koordination (Agent Coordination)

Grundsätzlich können Agenten in einem MAS ihre Aktionen genauso auswählen, als wären sie alleine (siehe Punkt 6.2.1). Wie aber bereits in Abschnitt 3.4 ausgeführt, bringt die Koordination von Aktionen Vorteile mit sich. Sie bildet ein wesentliches Merkmal von MAS, das es diesem erst ermöglicht, seine volle Leistungsfähigkeit zu entfalten. In MAS gibt es keine zentrale Kontroll- oder Steuereinheit. So bedient man sich diverser Protokolle und/oder Lernmechanismen, um diese Aufgabe zu bewältigen. Aufgrund der zentralen Bedeutung der Problematik wurden viele verschiedene Lösungsansätze entwickelt. Die nachstehende Abbildung gibt eine generelle Strukturierung von Lernformen in MAS vor (vgl. Sen (1998)). Diese eignet sich ebenfalls dazu, Trainingsverfahren für das Koordinationsproblem zu gliedern.

	Cooperative relationships		Non-cooperative relationships
	Individualistic learning	Shared learning	
Communicating agents			
Non-communicating agents			

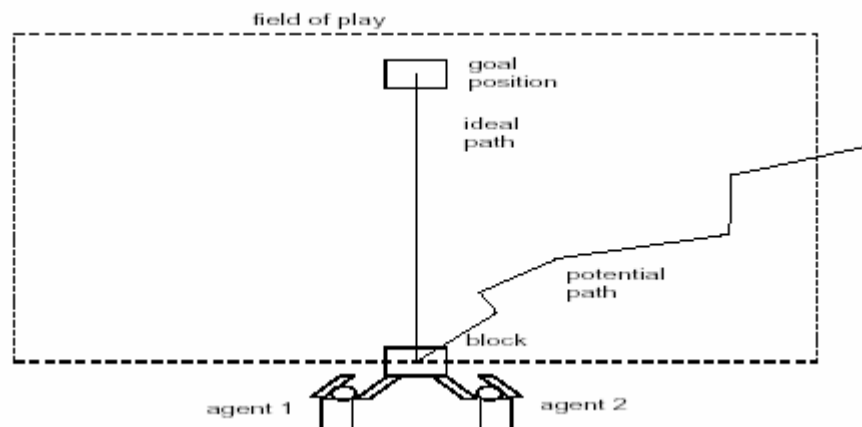
Quelle: Sen (1998)

**Abb. 6-10:** Übersicht Lernmethoden für das Koordinationsproblem

#### Beispiel 4 – Koordination ohne Kommunikation

##### *Individual learning of coordination knowledge (Sen (1998))*

Beim „Block pushing problem“ versuchen zwei oder mehrere Software-Agenten von einer Startposition aus, einen virtuellen Block zu einem Zielpunkt zu bewegen. Die Agenten nehmen sich gegenseitig nicht direkt wahr, kommunizieren nicht und kennen keine physikalischen Gesetzmäßigkeiten<sup>60</sup> (siehe Abb. 6-11). Die Agenten (im Experiment zwei) können selbständig entscheiden, mit welcher Kraft und in welchem Winkel (0-180 Grad) sie den Block „pushen“. Die Aktionen werden gleichzeitig vollzogen. Nach jedem Stoß wird ihnen in Abhängigkeit vom Abstand zum idealen Pfad ein Feedback (Reward) gegeben. Wenn  $(x,y)$  die aktuelle Position des Blocks ist und  $P_x(y)$  die x-Koordinate des Pfades P der selben y-Koordinate, dann ist  $\Delta x = |x - P_x(y)|$  der Abstand des Blocks entlang der x-Achse zum Pfad. Das Feedback wird wie folgt berechnet:  $K * \alpha^{-\Delta x}$  (K: Konstante;  $K = 50$ ,  $\alpha = 1,15$ ). Ein trail wird mit dem Erreichen der Zielposition, dem Verlassen des Spielfeldes oder einer festgelegten Anzahl von Stößen beendet. Ein run (Experiment) hingegen endet nach 10 aufeinander folgenden, erfolgreichen Versuchen den Block zum Ziel zu bewegen oder 1500 trails.



Quelle: Sen (1998)

**Abb. 6-11:** Testumgebung für Beispiel 4

<sup>60</sup> Sie stellen nur die Wirkung ihrer eigenen Aktionen und der anderer Agenten auf den Block fest.

### Training:

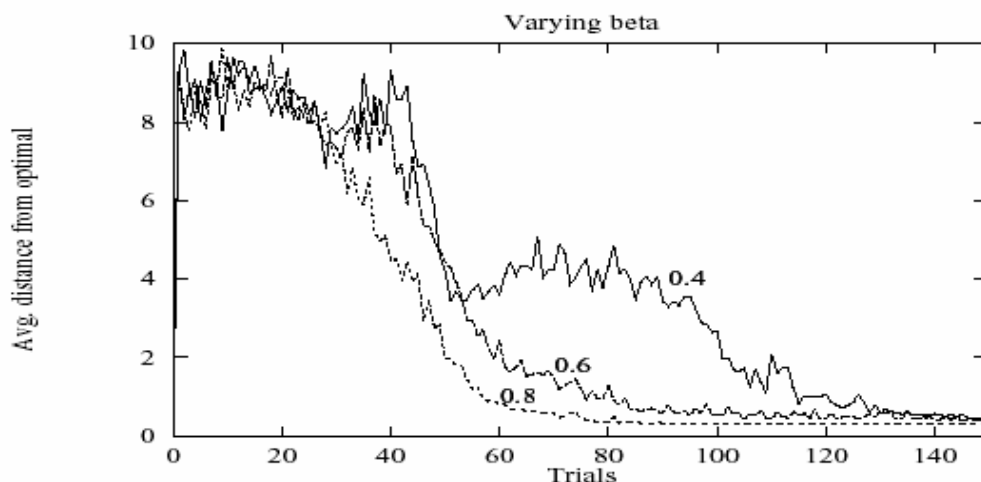
Das Ziel der Agenten besteht darin, den Block auf dem idealen Pfad zum Ziel zu bewegen (Maximierung des Rewards). Sie lösen die Aufgabe immer wieder, um durch RL aus ihren Erfahrungen schrittweise zu lernen. Das verwendete Verfahren wird in der Literatur auch als CIRL (Concurrent Isolated Reinforcement Learning) bezeichnet.

*Algorithmus:* Dieser unterscheidet sich nicht von dem in Punkt 6.1.1 (Q-Learning) besprochenen. Die Parameter betragen, soweit nichts Anderes erwähnt wird, für  $\beta = 0,2$  und  $\gamma = 0,9$ .

### Ergebnisse:

a) Cooperative domain (Agenten haben die gleiche Zielposition)

a1) Veränderung der Lernrate  $\beta$ : Es wurden verschiedene Lernraten getestet und die Anzahl der trials bis zum Erreichen der Konvergenzgrenze miteinander verglichen. Alle weiteren Parameter blieben konstant. Die Ergebnisse zeigt die Abb. 6-12. In diesem Fall ist es so, dass je größer  $\beta$  gewählt wird, das System schneller gegen einen „steady state“ Zustand strebt.



Quelle: Sen (1998)

**Abb. 6-12:** Auswertung Experiment 4

a2) Veränderung der Fähigkeiten der Agenten: Es wurden drei Experimente durchgeführt und in jedem die Fähigkeiten der Agenten variiert. Im ersten Versuch spielte ein Agent den „Dummy“. Er konnte keine Kraft auf den Block ausüben und somit praktisch nichts tun. Ein anderer durfte lediglich den Winkel seines Stoßes wählen (konstante Kraft – Vereinfachung der Aufgabe). Im zweiten Test herrschten ähnliche Voraussetzungen, nur hatte der Letztgenannte zusätzlich die Möglichkeit, auch den Betrag der Kraft zu ändern. Das letzte Experiment entsprach dem eingangs beschriebenen Szenario. Die durchschnittliche Anzahl von trials bis zum Erreichen des

„steady state“ Zustands betragen 55, 431 und 115. Das Interessante daran ist, dass die Agenten zusammen effektiver lernten ( $115 < 431$ ) als auf sich allein gestellt.

a3) Transfer des Wissens auf ähnliche Lernaufgaben: Die Startposition wurde beibehalten, aber das Ziel variiert. Die im Ausgangsexperiment erlernten Q-Matrizen dienen zur Initialisierung des Lernvorgangs. Im Ergebnis führte dieser Ansatz zur durchschnittlichen Einsparung von 10% der trials bis zum Erreichen der Konvergenzgrenze. Es wurde durch statistische Tests bestätigt. Des Weiteren stellte man fest, dass die Ersparnis umso geringer ausfiel, je weiter die neue Zielposition von der alten entfernt lag. Die Übertragung des Gelernten auf neue Aufgaben ist also nur in begrenztem Umfang mit dem Q-Learning möglich.

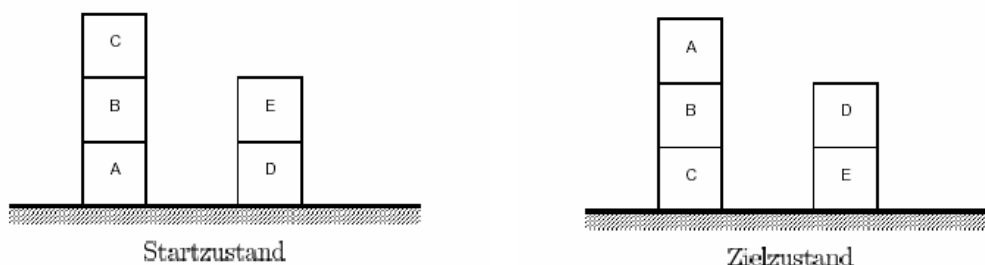
b) Non-Cooperative domain (Agenten haben unterschiedliche Zielpositionen)

Beide Agenten hatten bei diesem Experiment unterschiedliche Zielpositionen. Weiterhin wurden der Betrag der Kraft sowie die Möglichkeit zur Abstufung derselben bei einem Agenten eingeschränkt. Das Ergebnis fiel erwartungsgemäß aus: Der „Stärkere“ setzte sich gegen den „Schwächeren“ durch. Je geringer der Unterschied in den Fähigkeiten ausfällt, desto mehr trials benötigt der „stärkere“ Agent zum Erreichen seiner Zielkoordinaten.

### Beispiel 5 – Koordination mit Kommunikation

*Learning to coordinate actions in multi-agent systems (Weiss (1993))*

Das Paper behandelt das Problem der Aktionskoordination einer Gruppe von Agenten beim Lösen einer gemeinsamen Aufgabe (kooperierende Agenten). Als Testumgebung dient wiederum die Block-Welt (siehe Abb. 6-13). Die Agenten müssen Blöcke umstapeln und dabei vorgegebene Bedingungen einhalten.<sup>61</sup>



Quelle: Lettmann (2000)

**Abb. 6-13:** Testumgebung Beispiel 5

<sup>61</sup> In Anlehnung an das Spiel „Türme von Hanoi“.

Sie verfügen jeweils über spezielle Fähigkeiten. Beispielsweise kann Agent  $A_1$  die Aktionen  $\text{put}(A,B)$ <sup>62</sup> und  $\text{put}(A, -)$  ausführen, während Agent  $A_2$   $\text{put}(B,C)$  und  $\text{put}(B,D)$  beherrscht. Jeder Agent erkennt nur die für ihn relevanten Zustände (local information constraint). Durch gezielte Kommunikation lernen sie kollektiv, „die wahrscheinliche Zielrelevanz ihrer Aktionen vorherzusagen und auf diese Schätzwerte gestützt, ihre Aktionen zu koordinieren und geeignete Aktionssequenzen zu generieren.“ (Lettmann (2000)).

### Training:

Da kein Agent die Aufgabe allein lösen kann, muss durch geeignete Mechanismen eine Aktionskoordination erreicht werden. Dazu findet ein modifizierter BBA-Algorithmus Anwendung, das ACE-Verfahren (Action Estimation). Darauf gestützt lernen die Agenten durch wiederholtes Bearbeiten des gleichen Problems schrittweise aus ihren Erfahrungen.

### *Algorithmus:*

- Jeder Agent  $A_i$  der Gruppe bestimmt entsprechend seines Wissens und in Abhängigkeit vom aktuellen Zustand  $S_i$  die Menge von Aktionen  $A_i(S)$ , welche er augenblicklich ausführen kann. Danach schätzt er die Zielrelevanz  $E_i^j(S)$  jeder einzelnen Aktion  $A_i^j \in A_i(S)$  ein. Liegt dieser Wert über einer festzulegenden Grenze, wird ein Gebot für das Recht zur Ausführung kalkuliert:  $B_i^j(S) = (\alpha + \beta)E_i^j(S)$  ( $\alpha$ : kleiner, konstanter Risikofaktor;  $\beta$ : Rauschen, um konvergieren gegen lokale Minima zu vermeiden).
- Die Aktion mit dem höchsten Gebot aller Agenten kommt zur Ausführung, inkompatible Aktionen werden zurückgezogen. Dieser Vorgang wiederholt sich, bis alle Aktionen mit Geboten ausgeführt oder eliminiert wurden. Die realisierten Aktionen bilden einen „activity context“,  $A$ .
- Erhält die Gruppe eine externe Belohnung  $R$ , wird sie zu gleichen Teilen auf die ausgeführten Aktionen verteilt. Weiterhin reduzieren die Gewinner der Runde den Schätzwert  $E_i^j(S)$  ihrer Aktionen (Recht zur Ausführung). Der neue ermittelt sich wie folgt:  $E_i^j(S) \leftarrow E_i^j(S) - B_i^j(S) + \frac{R}{|A|}$ .
- Die Summe  $\sum_{A_i^j \in A} B_i^j(S)$  wird gleichmäßig auf die Aktionen des vorherigen „activity context“ verteilt (Belohnung für Schaffung des Zustandes).

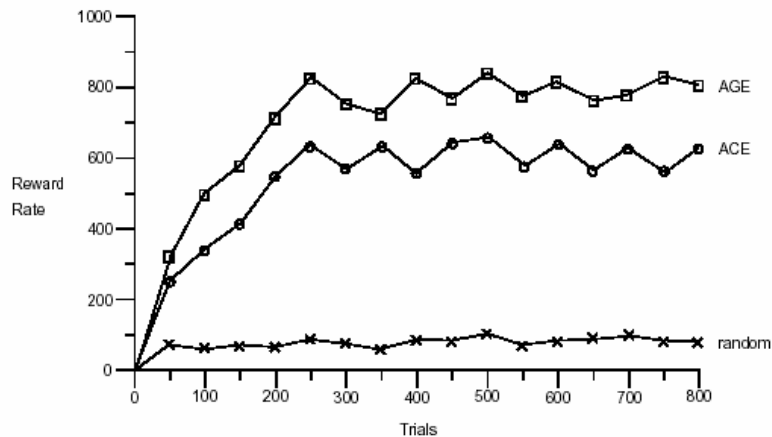
---

<sup>62</sup> Lege Block A auf B.

### Ergebnisse:

Im Experiment bildeten fünf Agenten eine Gruppe. Sie hatten vier Zyklen (siehe Beschreibung Algorithmus) Zeit, den Zielzustand herzustellen. In Abb. 6-14 sind die Ergebnisse dargestellt. Folgende Parameter wurden gewählt:

$$E^{init}=1000; \alpha=0,1; \beta \in [-\alpha/5; +\alpha/5] \text{ random}; \Theta = 0,7E^{init}$$



Quelle: Lettmann (2000)

**Abb. 6-14:** Auswertung Experiment 5

Der AGE-Algorithmus ist eine weiterentwickelte Form des ACE-Verfahrens. Die möglichen Aktionen werden nicht nur aus der begrenzten Perspektive eines Agenten bewertet, sondern zusätzlich im Zusammenhang mit dem aktuellen „activity context“. Die Werte im Diagramm zeigen, dass beide Algorithmen eine deutlich bessere Performance erzielen als eine zufällige Aktionsauswahl.

### **Beispiel 6 – Automatisches Design**

*Learning Organizational Roles in a Heterogeneous Multi-agent System (Lesser (1995))*

Die gemeinsame Aufgabe besteht im Zusammensetzen eines (vereinfachten) Dampfkondensators. Dabei können die technischen Anforderungen variieren. Das MAS setzt sich aus sieben Agenten zusammen, wobei jeder für den Einbau eines spezifischen Teils (z. B. Heizstab, Motor oder Pumpe) verantwortlich ist. Die Agenten haben ihre eigene lokale Sicht auf den Designprozess, eine Datenbank mit technischen Informationen über die entsprechende Komponente (inkl. Bedingungen für deren Einbau) sowie eine Liste mit potentiell ausführbaren Aktionen. Als Kommunikationsmedium dient ein zentrales Blackboard. Um zu einer Gesamtlösung zu gelangen, arbeiten die Agenten entsprechend ihrer Fähigkeiten an

Teillösungen, die in sich konsistent sein sowie der globalen Problemspezifikation gerecht werden müssen. Deshalb ist es notwendig, Konfliktsituationen aufzuspüren, zu vermeiden oder gegebenenfalls durch Informationsaustausch zu beheben. Die Agenten können drei organisatorische Rollen übernehmen: eine Lösung (ein (Teil-) Design) vorschlagen, eine bestehende erweitern oder sie kritisieren. Es ist weiterhin möglich, gleichzeitig an mehreren Teillösungen zu arbeiten und verschiedene Rollen auszufüllen.

#### Training:

Das Ziel des Trainings liegt im Finden und Übernehmen einer passenden Rolle(n) im Problemlösungsprozess, die in einer gegebenen Situation wahrscheinlich zu einer besseren Lösung mit möglichst geringen Kosten führt. Die Agenten lernen mittels RL inkrementell immer bessere Werte für den Nutzen (Utility), die Wahrscheinlichkeit des Eintretens (Probability) und die Kosten (Cost) einer Rolle in einer bestimmten Situation zu schätzen.

*Algorithmus (vereinfacht):* Beim o. g. beschriebenen Problem handelt es sich um eine komplexe Suche in verteilten Suchräumen. Um diese Suche effizient durchzuführen, werden nur Aktionen weiterverfolgt, die Erfolg versprechend sind. Dazu werden für jede Situation, beschrieben durch ein Tupel der Art  $\langle state, op, finalstate \rangle$ , UPC-Werte geschätzt (op: Rolle). Wenn  $S_k$  und  $R_k$  Mengen von Situationsvektoren und Rollen eines Agenten  $k$  sind, müssen bis zu  $|S_k| * |R_k|$  UPC-Vektoren und Potential-Werte<sup>63</sup> ermittelt werden. Während des Lernvorgangs beträgt die Wahrscheinlichkeit eine Rolle  $r$  in einer Situation  $s$  zu

wählen 
$$\Pr(r) = \frac{f(U_{rs}, P_{rs}, C_{rs}, Potential_{rs})}{\sum_{j \in R_k} f(U_{js}, P_{js}, C_{js}, Potential_{js})}$$
, wobei  $f$  eine objektive Funktion darstellt,

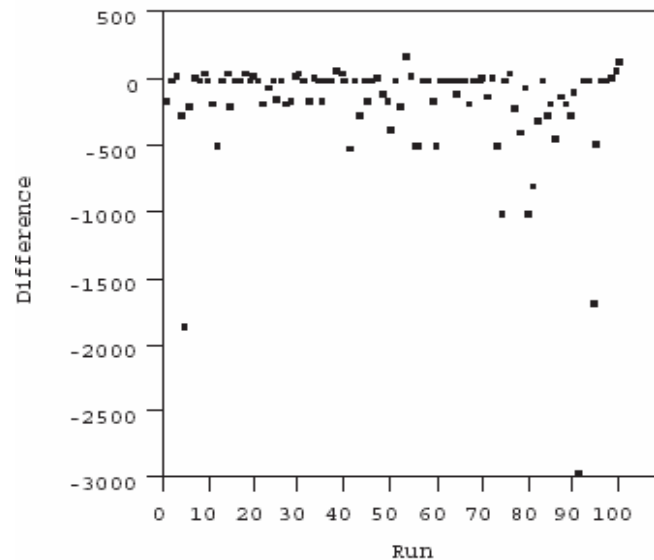
welche Rollen durch eine Kombination der bereits erwähnten Komponenten bewertet. Das Update der UPC-Werte (Gewichte) erfolgt wie beim RL üblich. Ein positiver Reward wird nur bei einer erfolgreichen Gesamtlösung gegeben. Alle Entscheidungsknoten (gewählten Rollen) entlang des Suchpfades, ausgehend von einem initialen Zustand  $s$  und der zugehörigen Rolle  $r$  bis hin zum Ziel, werden aktualisiert. Nach dem Ende des Trainings stehen die Rollen fest, da immer diejenige mit dem höchsten UPC-Wert gewählt wird.

#### Ergebnisse:

Im Experiment traten zwei Teams, ein trainiertes (Name: L-Team) und ein untrainiertes (Name: Team), gegeneinander an. Das L-Team lernte vorher an 150 zufällig generierten Trainingsbeispielen. Zur Bewertung der Rollen legte man  $f(U, P, C, potential) = U * P + potential$  fest. Die Lernrate betrug  $\beta=0,1$  für die Utility- und Probability-Komponente sowie  $\beta=0,01$  für die Potential-Komponente.

<sup>63</sup> Potentialwerte werden nicht weiter betrachtet. Sie stehen im Zusammenhang mit Wegen im Suchraum, die zur Gesamtlösung führen und Konfliktlösungssituationen beinhalten.

Danach wurden 100 andere Aufgaben durch beide Gruppen bearbeitet. Die Abb. 6-15 zeigt die Differenz der geringsten Kosten des Designprozesses des Teams und des L-Teams, die bei jeweils dem gleichen run (Problem) verursacht wurden. Die durchschnittlichen Kosten des L-Teams lagen danach um 3,2% tiefer (5587,6 vs. 5770,6), was bei einer Massenproduktion deutlich ins Gewicht fallen würde. Das Ergebnis wurde mittels statistischer Tests bestätigt.



Quelle: Lesser (1995)

**Abb. 6-15:** Auswertung Experiment 6

### **Beispiel 7 - Verteilte Terminplanung:**

*Learning other agents preferences in multi-agent negotiation using the Bayesian Classifier (Venkatesh (1999))*

In diesem Beispiel verwalten Agenten die Terminkalender ihrer User. Sie sind dafür verantwortlich, Termine von Besprechungen zu koordinieren. Das geschieht durch Verhandlungen (via Austausch von KQML-Nachrichten) der Agenten aller beteiligten Personen. Anfragen enthalten typischerweise Angaben über das Zeitfenster, in dem das Treffen stattfinden soll, die Dauer desselben sowie die Namen der jeweiligen Teilnehmer. Das Absagen von bereits bestehenden Vereinbarungen ist mit im Vorfeld festgelegten Kosten verbunden. Die Agenten haben ein nur unvollständiges Wissen in Bezug auf geeignete Zeitpunkte für anzuberaumende Treffen anderer Agenten, d. h. keinen Zugriff auf fremde Terminkalender um die Privatsphäre aller Benutzer zu schützen.

### Training:

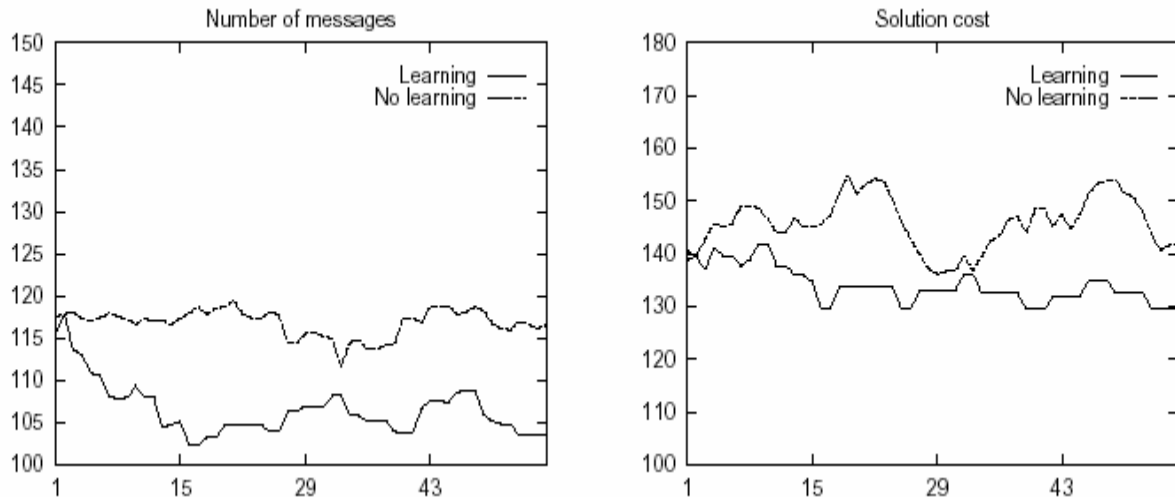
Das Ziel des Trainings besteht im Voraussagen der Präferenzen anderer Agenten. Somit wäre man in der Lage, gezielter zu kommunizieren und schneller zu einer Übereinstimmung zu gelangen (sich besser zu koordinieren). Das Lernen soll während der Ausführung der „normalen“ Aktivitäten der Agenten erfolgen – also online. Um Schätzungen vorzunehmen, bedient man sich der (naiven) Bayes-Klassifizierung. Das Lernen erfolgt bei diesem Verfahren schrittweise.

*Algorithmus:* Zuerst werden domainabhängige Features (z. B. Tag, Monat, Jahr) festgelegt, über welche die Präferenzen identifiziert werden sollen. Danach erfolgt eine Einteilung jeder Dimension in diskrete Intervalle. Somit wird es möglich, die jeweils einem Agenten zugeordneten Datensätze in Klassen (Set von spezifischen Intervallen) einzuteilen und die Wahrscheinlichkeit für jede Klasse gegeben die beschreibenden Merkmale zu ermitteln (Bayes-Klassifizierung). Bei Bekannt werden neuer Beispielfälle erfolgt eine Aktualisierung der Heuristiken. So gewinnt man schrittweise Präferenzfunktion der einzelnen Agenten. Ausgehend von diesen bildet man auf die Gruppe ab. Somit können neue Terminvorschläge gezielter abgegeben werden.

### Ergebnisse:

Im Experiment mussten die Agenten ein Meeting pro Arbeitstag (8.00 – 16.30 Uhr) mit der Dauer von einer Stunde vereinbaren. Jedem Agenten wurden Präferenzen für Terminvereinbarungen mitgegeben. Außerdem fügte man den Gewohnheiten Rauschen hinzu. Dieses kann als Einwilligung in nicht periodische Treffen bzw. das Absagen von turnusmäßigen Veranstaltungen interpretiert werden. Der Versuch wurde mit Gruppen von zwei und drei Agenten durchgeführt. Ermittelt wurden für jedes Agreement die Anzahl der ausgetauschten Nachrichten sowie die Kosten der Übereinkunft. Die Abb. 6-16 zeigt die Ergebnisse für drei beteiligte Agenten. (Auf der x-Achse sind die Arbeitstage abgetragen.) Als Vergleichswert dient die Performance einer gleichen Anzahl von nicht lernfähigen Agenten, die ebenfalls mit der Aufgabe betraut wurden.

In beiden Versuchen schnitten die lernenden Agenten besser ab. Deutlich wurde, dass der Unterschied umso größer ausfällt, je mehr Agenten involviert sind. (Die Komplexität der Verhandlungen steigt mit der Anzahl der beteiligten Agenten.)



Quelle: Venkatesh (1999)

**Abb. 6-16:** Auswertung Experiment 7

Es gibt in der Literatur viele weitere Beispiele zur Thematik von Lernen und Aktionskoordination. Neuere Ansätze versuchen, mehrere Verfahren zur Koordinationsverbesserung zu kombinieren, so z. B. (Garland (2004)). Das im Folgenden vorgestellte Paper verfolgt ebenfalls dieses Ziel. Die Agenten haben die Möglichkeit zu entscheiden, wann und wie sie mit anderen Agenten zusammenarbeiten wollen.

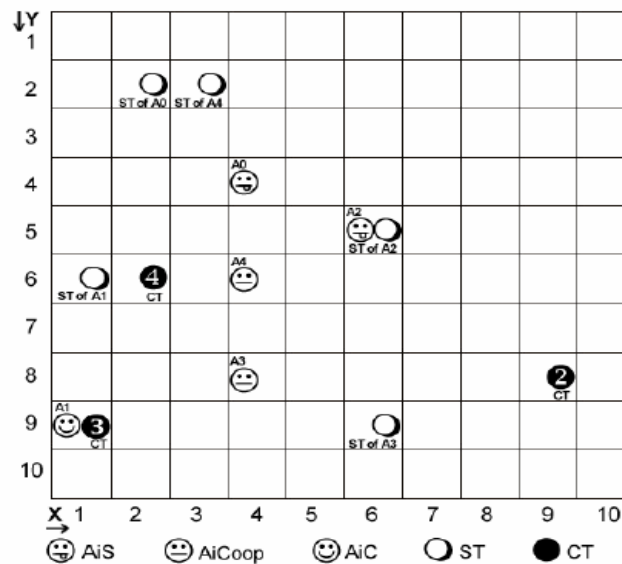
### **Beispiel 8 – Entscheiden, wann und wie man zusammenarbeitet:**

*Learning When and How to Coordinate (Jennings (2003))*

Die Agenten bewegen sich in einer Gitterwelt (vorwärts, rückwärts, links, rechts) von 10\*10 Feldern und führen Aufgaben aus. Das können entweder persönliche (specific task - ST) oder kooperative (cooperative task - CT) sein (siehe auch Abb. 6-17). Zu jedem Zeitpunkt kann ein Agent nur eine CT verfolgen. Eine ST gilt als erledigt, wenn eine vorgegebene Position auf dem Gitter erreicht wird. Danach erscheint zufällig im Spielfeld eine neue. Ähnlich verhält es sich mit CT, doch muss im Unterschied zur ST eine vorgegebene Anzahl von Agenten das spezifizierte Feld erreichen. Nach erfolgreicher Ausführung einer Aufgabe erhält der Agent eine Belohnung.

*Koordinationsprozess bei CT's:* Es kommt eine Variante des Kontrakt-Netz Protokolls zum Einsatz. Derjenige Agent, welcher ein CT entdeckt, entscheidet, ob er diese Aufgabe in Angriff nehmen möchte oder nicht. Im erstgenannten Fall wird er zum Agent-in-Charge (AiC) und initiiert den Koordinationsprozess. Ihm obliegt die Entscheidung, welche Koordinationsmethode (coordination method [CM]) angewandt wird. Diese sind von Agent zu Agent verschieden. Jede CM hat zwei Schlüsselattribute - Kosten (Zeitschritte bis zum Set Up) und die Chance auf Erfolg (Auszahlung des versprochenen Rewards mit der Wahrscheinlichkeit

p). *Nach* der Wahl der CM erfolgt ein broadcast mit der Aufforderung zur Abgabe von Geboten und der Position des CT an die anderen Agenten. Die Antworten gehen nach einer gewissen Zeitspanne (Verzögerung entspricht den Kosten der CM) beim AiC ein. Sie enthalten u. a. die Beträge, welche die Agenten am Reward fordern sowie ihre augenblicklichen Entfernungen vom CT. Akzeptiert der AiC ein Angebot, wird der Bietende zum Agent-in-Cooperation (AiCoop). Agenten, die ein ST verfolgen, sind weiterhin Agent-in-ST (AiS) (siehe Abb. 6-17).



Quelle: Jennings (2003)

Abb. 6-17: Gitterwelt Beispiel 8

### Training:

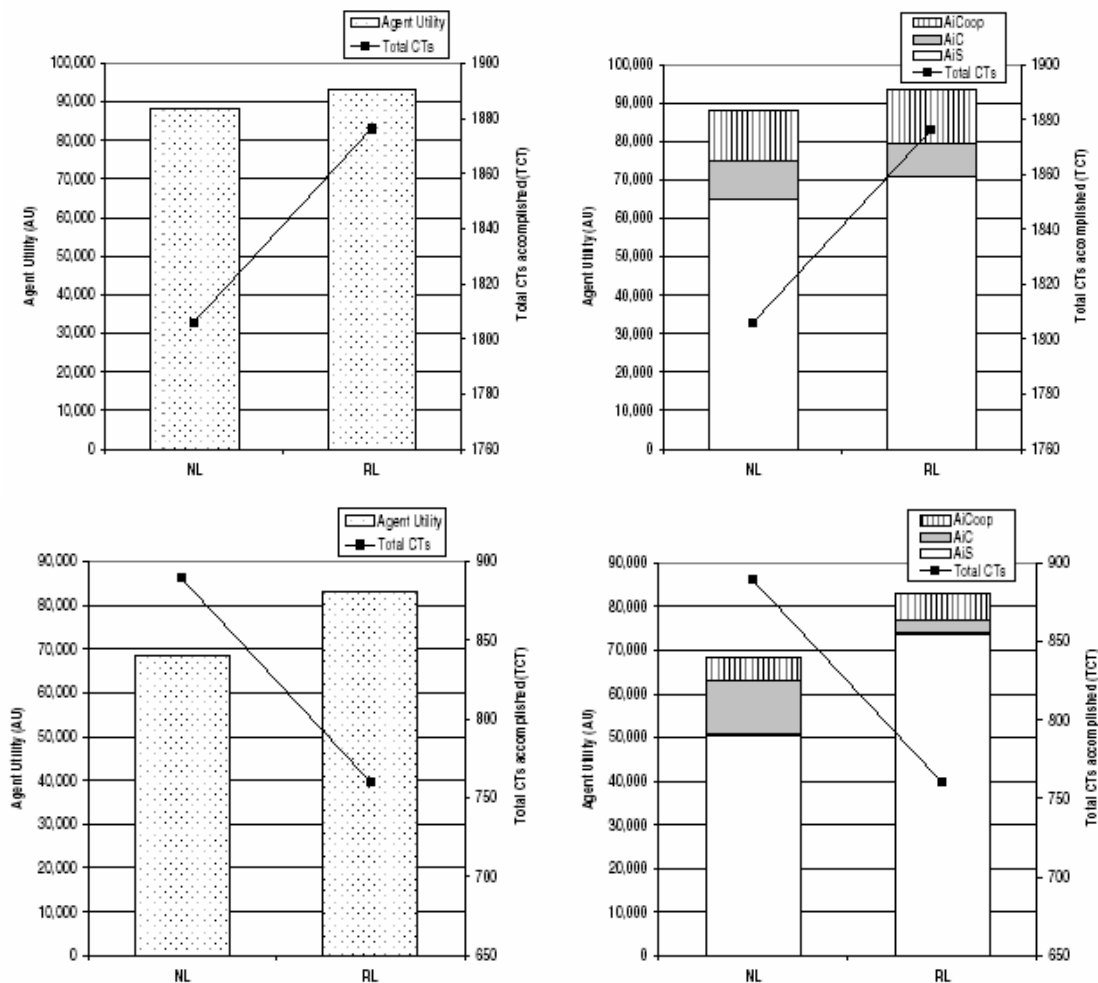
Das Ziel der Agenten besteht in der Maximierung ihres *persönlichen* Nutzens (der eingesammelten Belohnungen). In diesem Fallbeispiel gibt es mehrere Möglichkeiten zu lernen, z. B. die Entscheidung für bzw. gegen eine Kooperation oder die Wahl der „richtigen“ CM. Mit Letztgenanntem beschäftigt sich das Paper. Das RL (Q-Learning) kam wiederum zur Anwendung.

Der *Algorithmus* weist keine Besonderheiten auf (siehe Abschnitt 6.1.1).

### Ergebnisse:

Im Experiment verglich man anhand mehrerer Parameter die Performance einer Gruppe von Agenten mit der Fähigkeit zum Lernen (Name: RL) und einer ohne (Name: NL). Dazu wurden zwei Szenarios konstruiert. Im „statischen“ Fall (siehe „1. Zeile“ Abb. 6-18) folgten alle Agenten vorgegebenen Regeln (Formeln<sup>64</sup>) beim Bietvorgang für ein CT. Dadurch konnte der AiC auf einer „sicheren“ Basis bei der Auswahl der CM entscheiden.

<sup>64</sup> siehe (Jennings (2003))



Quelle: Jennings (2003)

**Abb. 6-18:** Auswertung Experiment 8

Im „dynamischen“ Fall (siehe „2. Zeile“ Abb. 6-18) dagegen wurde Rauschen hinzugefügt. Der AiC traf somit seine Wahl unter Unsicherheit. Der Versuch dauerte jeweils 500.000 Zeiteinheiten. Jede Gruppe bestand aus fünf Agenten. Es waren immer 3 CT's in der Gitterwelt. Die Agenten erhielten einen Reward von 1 für ein ST und 20 für ein CT. Maximal wurden drei Agenten zur Erfüllung einer CT benötigt. Die mit Linien verbundenen Punkte in den Balken zeigen die Anzahl der erfüllten CT's, die Balken selbst (Höhe) stehen für den erzielten Gesamtnutzen aller Agenten.

Aus den Abbildungen ist zu erkennen, dass das Lernen nur im „dynamischen“ Fall zu einer wirklichen Verbesserung führte<sup>65</sup>. Unter „statischen“ Bedingungen dagegen wurden lediglich mehr CT's verfolgt, was sich aber nicht wesentlich im erzielten Nutzen niederschlug. Die Ergebnisse stützen die These, dass die Bedeutung des Lernvorgangs in statischen Umgebungen geringer einzuschätzen ist als in dynamischen (vgl. Brenner (1998)).

<sup>65</sup> Einsatzumgebungen von MAS zeichnen sich allerdings u. a. durch das Treffen von Entscheidungen unter Unsicherheit aus.

### 6.3.2 Agenten-Kommunikation (Agent Communication)

Die Fähigkeit der Agenten miteinander zu kommunizieren, stellt eine der Schlüsseleigenschaften von MAS dar. Demzufolge werden nun exemplarisch Zusammenhänge zwischen dem Training von MAS und Agenten-Kommunikation untersucht. Folgende Punkte sind dabei näher zu betrachten:

- Lernen, um die Kommunikation zu reduzieren (Kostensparnis)
- Durch Kommunikation das Lernen verbessern (Informations- und Wissensaustausch)

#### Lernen die Kommunikation zu reduzieren

Die Kommunikation zwischen Agenten verursacht Kosten. Sie beansprucht Zeit und belastet das Netzwerk. Das schlägt sich vor allem in kommunikationsintensiven Domänen negativ auf die Performance des Systems nieder. Ein weiteres Problem besteht darin, dass die neuen Informationen bewertet (verifiziert) und in die eigene Wissensbasis integriert werden müssen. Ausgetauschte Nachrichten könnten außerdem von nicht berechtigten Dritten gelesen oder manipuliert werden. Deshalb sucht man nach Wegen, Agenten-Kommunikation auf das notwendige Maß zu reduzieren<sup>66</sup>. Eine Möglichkeit besteht darin, Agenten Informationen über die Fähigkeiten anderer lernen zu lassen. Dadurch können diese dann gezielter kommunizieren (siehe Beispiel 7). Diesen Ansatz verfolgt auch das nächste Beispiel.

#### Beispiel 9 – Roboter Koordination

*Addressee Learning and Message Interception for Communication Load Reduction in Multiple Robot Environments (Ohko (1996))*

Simulierte Roboter (Lemminge) bedienen in einem virtuellen Lokal Gäste. Jeder von ihnen kann lediglich ein Gericht liefern (siehe Tab. 6-1). Bestellt ein Gast eine Speise, die der Lemming nicht vorrätig hat, muss er einen anderen Roboter um Hilfe bitten. Als Kommunikations-/Koordinationsmechanismus wird das Kontrakt-Netz Protokoll genutzt. Um die Anzahl der Auftragsausschreibungen via broadcast zu verringern (diese ziehen den größten Teil der Kommunikationslast nach sich), versuchen die Lemminge aus ähnlichen Fällen in der Vergangenheit zu lernen und nur mit „geeigneten“ Partnern zu verhandeln. (Der Aspekt der „Message Interception“ wird nicht betrachtet.)

---

<sup>66</sup> Einige Koordinationsverfahren versuchen beispielsweise ganz ohne Kommunikation auszukommen (siehe Beispiele 2 und 4). Das ist aber nur in begrenztem Umfang möglich.

**Tab. 6-1:** Aufstellung Roboter und lieferbares Gericht

Robot	Dish
R0, R1	Chicken
R2, R3	Salad
R4, R5	Pasta

Quelle: Ohko (1996)

Training:

Das Ziel des Trainings wurde bereits genannt - ein Modell der Fähigkeiten anderer Roboter aufzubauen, um gezielt zu kommunizieren. Die Agenten bedienen sich dazu des Cased-based Reasoning. Sie lernen schrittweise mit jedem neuen Beispiel (online). Die Autoren des o. g. Papers nannten das Verfahren „Addressee Learning“ („Adressiertes Lernen“).

*Algorithmus (vereinfacht):* Eine Aufgabenbeschreibung (task) ist eine Liste mit Attribut-Wert-Paaren und einer eindeutigen Id  $T = (A_1V_1, A_2V_2, \dots, A_{m-1}V_{m-1}, taskid : id)$ . Der Algorithmus läuft wie unter 6.1.1 beschrieben ab. Die Funktion zur Bestimmung des Abstands zweier Fälle lautet:

$$(1) \text{Dist}(A_{1i}, V_{1i}, A_{2j}, V_{2j}) = W(A_{1i}, A_{2j}) \text{Equal}(V_{1i}, V_{2j})$$

$$(2) W(A_{1i}, A_{2j}) = 1, \text{ wenn } A_{1i} = A_{2j}, \text{ sonst } 0$$

$$(3) \text{Equal}(V_{1i}, V_{2j}) = 1, \text{ wenn } V_{1i} = V_{2j}, \text{ sonst } 0$$

$$(4) \text{Similarity}(T_1, T_2) = \sum_{i=1}^m \sum_{j=1}^n \text{Dist}(A_{1i}, V_{1j}, A_{2i}, V_{2j})$$

Für alle Beispiele mit einer Ähnlichkeit über einer festzulegenden Schranke bestimmt der Agent zusätzlich einen Performancewert<sup>67</sup>. Auf dessen Ermittlung soll aber nicht weiter eingegangen werden. Der Lemming sendet seine Anfrage an den Roboter mit der höchsten Performance.

Ergebnisse (ohne Abbildung):

Die Definition der Performance für den Fall, dass eine Aufgabe ausgeführt werden kann, lautet:  $\text{Performance}(R, t, T) = \max(0, C - (\text{EndTime}(T) - t))$  (R: Roboter; T: Task; t: aktuelle Zeit; EndTime: Zeit, zur der die Aufgabe beendet wurde; C: im Verhältnis zu  $(\text{EndTime}(T) - t)$  große Konstante). Bei Nichtausführung erhält der Agent eine Bestrafung von -1. Je höher der

<sup>67</sup> Der Faktor wird durch die Auswertung von Parametern in den ausgetauschten Messages des jeweiligen Beispielfalls ermittelt.

Funktionswert für Performance( $R, t, T$ ) ausfällt, desto besser. Bei der Ermittlung der Kommunikationslast berücksichtigte man folgende Punkte:

- Broadcast-based system: Nachricht zur Kontraktanschreibung \* Anzahl *aller* Roboter, Nachrichten der Bieter (*alle Lemminge mit „passenden“ Gerichten*), Kontrakt, Report
- Lemming system: Nachricht zur Kontraktanschreibung \* Anzahl der *adressierten* Roboter, Nachrichten der Bieter, Kontrakt, Report

Es wurden drei Experimente durchgeführt. Sechs Lemminge mussten vier Gäste bedienen. Es konnte zwischen Huhn, Salat und Pasta gewählt werden. Die Roboter kannten die Speise, welche sie selbst trugen (siehe Tab. 6-1), nicht aber die der anderen. Bestellte ein Gast ein Gericht, welches der Lemming nicht selbst liefern konnte, musste er mit den anderen Agenten verhandeln. Die Roboter waren zur Kooperation verpflichtet. Des Weiteren variierte die Anzahl der vorrätigen Gerichte der Lemminge. Alle bewegten sich mit der gleichen Geschwindigkeit. Die Bestellungen wurden per Zufall verteilt. Eine Simulation war nach 40 bearbeiteten Aufgaben pro Lemming beendet. Für jedes Experiment wurden die Durchschnittswerte nach 30 Wiederholungen ermittelt. Als Vergleichswert diente ein System, indem Auftragsanschreibungen nur per broadcast „herausgegeben“ wurden.

Der erste Versuch verlief wie oben beschrieben, ohne jegliche Beschränkung hinsichtlich der Anzahl der Gerichte. Im zweiten änderte man nach 20 erledigten Aufgaben die Kapazität von Roboter R1, R3 und R5 auf null. Im dritten wurde dieser Schritt zufällig vor dem Erreichen der Marke von 20 ausgeführten Aufträgen gemacht. Nach der 20. Task wurden dann alle Speisen wieder aufgefüllt. Es zeigte sich, dass die Kommunikationslast im Vergleich zum broadcast-based system in allen drei Fällen um ca. 50% gesenkt werden konnte. Die Effizienz der Aufgabenausführung (durchschnittliche Wartezeit) dagegen verbesserte sich nicht. Sie war sogar etwas geringer als der Vergleichswert. Ein weiteres Problem trat auf, wenn neue, unbekannte Roboter in das MAS kamen (siehe Experiment 3). Diese wurden erst wahrgenommen, wenn aufgrund einer fehlgeschlagenen Suche nach ähnlichen Situationen im Speicher oder dem Absagen der für die Aufgabe in Frage kommenden Roboter ein broadcast gestartet wurde.

### **Durch Kommunikation das Lernen verbessern**

Wie bereits mehrfach erwähnt, besitzen Agenten oft nur unvollständige Informationen über ihre Umgebung. Generell kann deshalb Kommunikation zum Zweck des Informationsaustauschs als eine Möglichkeit des Lernens in MAS angesehen werden, da neues Wissen akquiriert wird (vgl. Weiss (1999a)). Die damit verbundenen Fragen sind entweder durch den Entwickler oder das MAS selbst zu beantworten:

- Welche Informationen sollen ausgetauscht werden? Was ist von Interesse?
- Wann „lohnt“ sich Kommunikation? (Kommunikation ist mit Kosten verbunden!)
- Mit wem soll kommuniziert werden? (1:1 oder 1:n? Wer besitzt interessante Informationen?)
- Auf welcher Basis findet die Kommunikation statt? (Protokoll, Sprachen, Ebene der Kommunikation)

Die lokale Sicht der Agenten auf ein Problem birgt die Gefahr eines Fehlschlags bei der Bearbeitung einer Lernaufgabe in sich, z. B. der Ausführung von Plänen. Sie können deshalb versuchen, dieses Risiko durch die Beschaffung zusätzlicher Informationen zu reduzieren. Die dazu notwendige Kommunikation kann auf verschiedenen Ebenen stattfinden:

- low-level communication: Die Kommunikation beschränkt sich auf relativ einfache „Frage-und-Antwort“ Interaktionen, um fehlende (Stücke von) Informationen auszutauschen.
- high-level communication: Es findet eine sehr komplexe Interaktion statt, z. B. Verhandlungen oder gegenseitiges Erklären. Die Informationen werden kombiniert und so neues Wissen generiert. Diese Art des Lernens ähnelt der von Menschen.

Voraussetzung für Agenten-Kommunikation auf hohem abstrakten Niveau ist eine gemeinsame Ontologie (siehe auch Abschnitt 3.5). In offenen Systemen, wie z. B. dem WWW, stellt das ein Problem dar. Dem versucht man durch verschiedene Maßnahmen zu begegnen. Die bekannteste dürfte wohl die Initiative „Semantic Web“<sup>68</sup> sein. Eine weitere Möglichkeit besteht im gegenseitigen Erlernen von Ontologien, wie z. B. in „Learning to Share Meaning in a Multi-Agent System“ (Williams (2004)). In diesem Fall lernen Agenten schrittweise, durch gegenseitiges Erklären der Bedeutung von Objekten aus der gemeinsamen Umgebung, die Begriffswelt des jeweiligen Partners.

Ob Kommunikation das Lernen letztlich verbessert, kann jeweils nur im Einzelfall durch Klärung nachstehender Punkte (Benchmarking) erfolgen:

- Wie schnell werden die Lernziele mit und ohne Kommunikation erreicht?
- Sind die erreichten Ergebnisse von gleicher Qualität?
- Wie komplex ist der Lernvorgang mit und ohne Kommunikation?

Die Kombination von Kommunikation mit Lernverfahren muss sehr sorgfältig vorgenommen werden, da der Austausch von Information oder Wissen auch seine negativen Seiten hat, wie erhöhte Kosten bzw. Aufwand, größere Fehleranfälligkeit usw. (vgl. Weiß (1999a)).

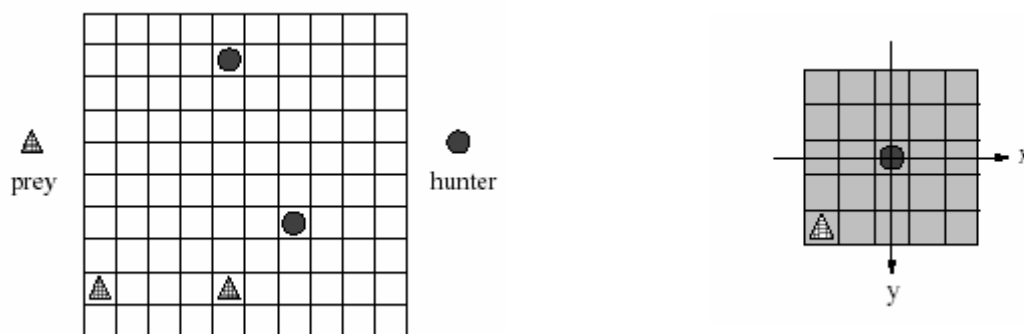
---

<sup>68</sup> Für weitere Informationen siehe <http://www.w3.org/2001/sw/>.

### Beispiel 10 – Verbesserungen beim Lernen von Koordinationswissen durch den Austausch von Informationen

*Multi-agent reinforcement learning: Independent vs. cooperative agent (Tan (1993))*

Das Paper beschäftigt sich mit der Frage, inwieweit low-level communication zwischen Agenten den Lernvorgang für Koordinationswissen verbessern kann. Dazu werden Vergleiche zwischen unabhängigen und kooperierenden Agenten gezogen, welche gleiche Aufgaben lösen müssen. Als Domain wurde die Jäger-Beute Welt gewählt (siehe linker Teil der Abb. 6-19). In einem 10\*10 Einheiten großen Gitter befinden sich Jäger und Beute. Pro Zeiteinheit kann ein Schritt vorwärts, rückwärts, nach rechts oder links erfolgen. Agenten mit einer Beuterolle bewegen sich zufällig. Die Jäger können die Beute nur in einem bestimmten Umfeld (Größe kann variiert werden, siehe rechter Teil der Abb. 6-19<sup>69</sup>) wahrnehmen. Die Beute gilt als gefangen, wenn Jäger und Beute das gleiche Feld besetzen. Ein run (Experiment) besteht aus mehreren trials. Beim ersten trial eines runs werden die Startpositionen zufällig initialisiert. Danach beginnen nur noch die Jäger in zufälligen Ausgangspositionen, die Beute behält die ihren bei. Ein trial ist beendet, wenn die *erste* Beute gefangen wurde.



Quelle: Tan (1993)

**Abb. 6-19:** Gitterwelt Beispiel 10

#### Training:

Das Ziel des Trainings besteht im möglichst schnellen „Erlegen“ der Beute. Die Jäger lernen inkrementell durch jeden trial mittels RL (Q-Learning).

*Algorithmus:* Das Q-Learning wurde bereits des Öfteren beschrieben (siehe Punkt 6.1.1). Für jeden Erfolg gibt es eine Auszahlung von +1, für alle sonstigen Schritte eine Auszahlung von -0,1. Während der Trainingsphase wird eine Aktion  $a_i$  mit der folgenden Wahrscheinlichkeit gewählt (Boltzmann-Verteilung): 
$$p(a_i|x) = \frac{e^{Q(x,a_i)/T}}{\sum_{k \in \text{actions}} e^{Q(x,a_k)/T}}.$$

<sup>69</sup> Der rechte Teil der Abb. 6-19 zeigt einen Jäger mit der visuellen Tiefe von 2 Einheiten.

Der Temperatur T kommt dabei die Aufgabe zu, auch zufällige Entscheidungen zu generieren. Nach Abschluss des Trainings wird jedoch die Aktion mit dem größten Wert für  $Q(x, a)$  ausgeführt. Die Parameter betragen  $\beta = 0,8$  /  $\gamma = 0,9$  und  $T = 0,4$ .

#### Ergebnisse:

Die Tab. 6-2 zeigt die Ergebnisse der „Jagd“ (nach 200 trials) für lernende und zufällig agierende Agenten bei einer visuellen Tiefe von 4 Einheiten. Die Frage ist nun, ob sich die Effizienz der Jäger durch Austausch von Informationen bzw. eine Zusammenarbeit verbessern lässt.

**Tab. 6-2:** Auswertung I Beispiel 10

N-of-prey / N-of-hunters	1/1	1/2
Random Hunters	123,08	56,47
Learning Hunters	25,32	12,21

Quelle: Tan (1993)

In einem weiteren Experiment konnten die Jäger ihre aktuellen Wahrnehmungen austauschen. Des Weiteren war ihnen ihre relative Position zueinander bekannt. Wie die Tab. 6-3 zeigt, führte das, abhängig von der Wahrnehmungstiefe, zu unterschiedlichen Ergebnissen. Die linke Spalte beinhaltet die durchschnittliche Anzahl der Schritte während des Trainings, die rechte nach Abschluss der Lernphase<sup>70</sup>. Gut zu erkennen ist, dass bei einer visuellen Tiefe von 2 Einheiten Kommunikation sogar negative Effekte zeigt<sup>71</sup>. Die Beute befindet sich zu kurz im Sichtfeld des Jägers, um daraus Rückschlüsse ziehen zu können.

Im letzten Experiment nutzten die Agenten (a) die gleiche Q-Matrix (Abb. 6-20, linker Teil) bzw. (b) tauschten diese nach einer gewissen Anzahl von Schritten aus (Abb. 6-20, rechter Teil). Die visuelle Tiefe betrug in beiden Versuchen 4 Einheiten. Bei anderen Sichtweiten unterschieden sich die Ergebnisse in (a) nicht wesentlich voneinander. In (b) hingegen war ein Einfluss festzustellen. Bei einer Sichtweite von beispielsweise 2 Einheiten war der Austausch der Q-Matrix alle 50 Schritte von Vorteil. Generell brachte der Informationsaustausch auf diesem Level Performanceverbesserungen mit sich.

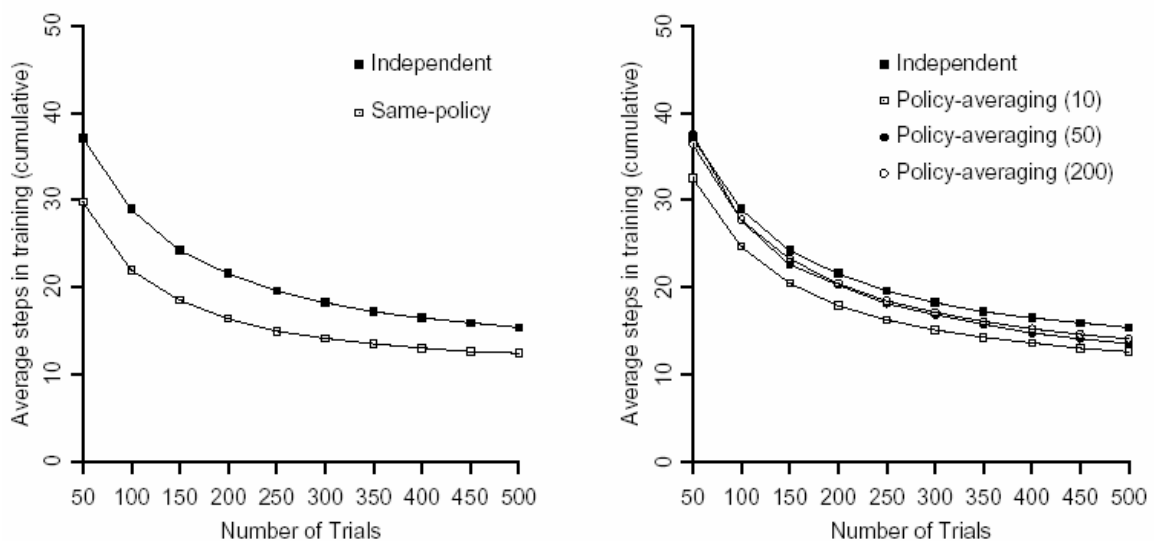
<sup>70</sup> Die Anzahl der Schritte wurde beim Training alle 50 trials festgestellt. Nach dem Erreichen der Konvergenzgrenze bildete man den Durchschnitt. Während des Tests flossen die 5 letzten von 1000 trials in den Durchschnittswert ein. Training und Test unterscheiden sich durch die Art der Aktionsselektion.

<sup>71</sup> Es sei noch einmal an die Bemerkung von (Weiss (1999a)) erinnert, dass die Kombination von Kommunikation und Lernen sorgfältig abgewogen werden muss.

**Tab. 6-3:** Auswertung II Beispiel 10

	Visual Depth	Average Steps to Capture a Prey	
		Training	Test
Independent agents	2	20,38 (+/- 0,57)	24,04 (+/- 1,00)
Mutual-scouting	2	25,20 (+/- 0,79) (worse)	24,52 (+/- 1,24)
Independent agents	3	14,65 (+/- 0,53)	16,04 (+/- 0,56)
Mutual-scouting	3	14,02 (+/- 0,75) (same)	12,98 (+/- 0,65)
Independent agents	4	12,21 (+/- 0,65)	11,53 (+/- 0,61)
Mutual-scouting	4	11,05 (+/- 0,56) (better)	8,83 (+/- 0,78) (better)

Quelle: Tan (1993)



Quelle: Tan (1993)

**Abb. 6-20:** Auswertung III Beispiel 10

Die Verbindung von Lernen und high-level communication führt zu Lernformen, wie sie bei der Zusammenarbeit von Menschen in Arbeitsgruppen auftreten (siehe auch weiter oben bzw. Abschnitt 6.1.2). Die Umsetzung derartiger kognitiver Konzepte wurden bis dato aufgrund ihrer Komplexität<sup>72</sup> durch die DAI weitestgehend nicht verfolgt (vgl. Weiß (1999a), Weiss (1999b), Williams (2003)). Demzufolge finden sich dazu nur wenige Arbeiten in der Literatur. Ein

<sup>72</sup> Aufgrund dessen sind diese Ansätze natürlich auch sehr rechenintensiv. Ein Problem, das sich bei einer größeren Anzahl von Agenten im MAS noch verschärft.

Ansatz („Adaption based on cooperative learning in multi-agent systems“ (Sian (1993))) sei kurz beschrieben.

Eine Gruppe von Agenten nutzt ein Blackboard als Kommunikationsmedium. Verwendet wird eine einfache Sprache aus neun Operatoren. Mit Hilfe dieser können Hypothesen vorgeschlagen, bearbeitet und zurückgezogen werden<sup>73</sup>. Im vorliegenden (konstruierten) Fall „diskutierten“ die Agenten über die Preisentwicklung von Kaffee, Tee und Kakao in Abhängigkeit vom Wetter in verschiedenen Erzeugerländern. Nach einer Phase der Datensammlung und Auswertung (jeder separat für sich) erzielten sie in der anschließenden Konversation Übereinstimmung darin, dass die Preise für ein Produkt (Feldfrucht) steigen, wenn es im Haupterzeugerland desselben widriges Wetter gibt. Im Unterschied zum Beispiel 9 handelt es sich hier um „*shared understanding*“ anstatt von „*shared information*“.

---

<sup>73</sup> Beispielsweise können Agenten eigene Hypothesen durch andere Agenten überprüfen lassen oder aus der „Diskussion“ heraus neue gewinnen.

## 7 Effiziente Technologien für das Agenten-Training

Agenten mit der Fähigkeit zum Lernen auszustatten, bedeutet zusätzlichen Aufwand, z. B. durch:

- Die Programmierung von Algorithmen
- Einem eventuellen Aufbau oder die Erweiterung einer Wissensbasis
- Das Bereitstellen einer Trainings- oder Testumgebung
- Die Durchführung des Trainings und die Evaluation der Ergebnisse

Sind bei den Entwicklern von MAS nicht auch die entsprechenden Kompetenzen für die genannten Punkte vorhanden, verschärft sich das Problem noch. Deshalb wäre eine Unterstützung bei der Integration von Lern- und Trainingsverfahren in MAS durch Agenten-Entwicklungswerkzeuge sinnvoll, gewissermaßen als Verbesserung der *Prozesseffizienz und Ressourceneffizienz* für diese spezielle Phase der Softwareentwicklung bei agentenbasierten Systemen. Anhand von fünf Beispielen wird exemplarisch untersucht, welche Möglichkeiten z. Z. verfügbare Entwicklungsumgebungen bieten. Bei der Auswahl der Testkandidaten wurde versucht, einen möglichst breiten Querschnitt von Ansätzen zu treffen.

Die jedem Abschnitt vorangestellte Tabelle verschafft einen Überblick über die besprochene Software. Für weitergehende Informationen allgemeiner Art wird auf die Web-Seiten des jeweiligen Herstellers verwiesen.<sup>74</sup> Der Punkt „Komponenten für integriertes Lernen“ beinhaltet die jeweiligen Hilfestellungen zur effizienteren Programmierung von adaptiven Agenten.

### 7.1 Agent Building and Learning Environment (ABLE)

ABLE ist für die Entwicklung hybrider Agenten vorgesehen und kann auf allen Systemen zum Einsatz kommen, die Java ab Version 1.3 unterstützen. Nicht zu verwechseln ist das Toolkit mit einem früheren Projekt von IBM (jetzt Open-Source), den Java-Aglets. ABLE spielt eine wichtige Rolle im Rahmen der IBM-Initiative „Selbstheilende Software“.

---

<sup>74</sup> Quellen der nachfolgenden Ausführungen sind ebenfalls die diversen Dokumentationen und Tutorials, die entweder im Lieferumfang enthalten oder über die entsprechenden Web-Seiten verfügbar sind. Bei Ausnahmen wird darauf hingewiesen.

**Tab. 7-1:** Überblick ABLE

Technische Voraussetzungen	Java ab Version 1.3
Version / Lieferumfang	Version 2.1 / Framework, Agent Editor, Dokumentation, Beispiele, Plattform (Runtime-Environment), intelligente Softwarekomponenten
Lizenz	Kostenlos für interne Testzwecke
Hersteller	IBM
Plattform erfüllt FIPA-Standard	Ja (FIPA 97)
Kommunikation via	Sprache: FIPA-ACL (und Java-Events) Protokolle: HTTP, RMI
Wesentliche Features	<ul style="list-style-type: none"> <li>• Graphische Unterstützung beim Erstellen (und Testen) von agentenbasierten Applikationen (Voraussetzung: Verwenden von JavaBeans)</li> <li>• Intelligente Softwarekomponenten für ML und Schlussfolgern</li> <li>• Able Rule Language (ARL), Able RuleSet Editor</li> </ul>
Link	<a href="http://www.alphaworks.ibm.com/tech/able">http://www.alphaworks.ibm.com/tech/able</a>

### Komponenten für integriertes Lernen

Wie bereits dem Namen zu entnehmen ist, lag ein wesentliches Augenmerk bei der Entwicklung von ABLE auf der Integration von Lernverfahren. Bevor jedoch näher darauf eingegangen wird, kurz zur Architektur von AbleAgents. Diese werden aus AbleBeans (JavaBeans<sup>75</sup>) mittels „data flow connections“, „event connections“ und/oder „property connections“ zusammengesetzt<sup>76</sup> (siehe Abb. 7-1). Dadurch unterscheidet sich der Ansatz von anderen Herangehensweisen. Viele Entwicklungswerkzeuge geben BDI-Agenten als kleinste Basiseinheit vor. Die Lösung von IBM bietet allerdings mehr Flexibilität und Skalierbarkeit.

Der Able Agent Editor dient als graphisches Tool für das Erstellen, Bearbeiten und Testen von Agenten und seiner Bestandteile (siehe Abb. 7-1). Die mitgelieferte Bibliothek intelligenter Softwarekomponenten umfasst folgende Bestandteile (Auswahl):

<sup>75</sup> JavaBeans sind normierte, wieder verwendbare Softwarekomponenten in Java.

<sup>76</sup> Ein AbleAgent kann natürlich wiederum andere Agenten enthalten („Matroschka-System“). AbleBeans können auch separat in eigenen Anwendungen eingesetzt werden.

- Data Beans: zuständig für Datenimport und -export (SQL-Datenbanken, Textfile)
- Learning Beans: Es sind vor allem ML-Algorithmen implementiert worden, die zur Klassifikation und Voraussage eingesetzt werden können. Learning Beans kommen oft in Verbindung mit Rules Beans zum Einsatz (leichtgewichtige DM-Agenten).
- Rules Beans: ABLE bietet verschiedene Inferenzmaschinen sowie eine Planungskomponente an. Zur Definition und hierarchischen Strukturierung der Regelbasen (AbleRuleSets, Wissensbasen) wird die durch IBM entwickelte Sprache Able Rule Language (ARL) genutzt. ARL kann Java-Objekte instanziiieren, manipulieren sowie Java-Funktionen aufrufen.
- Agents: „komplette“ Agenten mit spezieller Funktionalität, z. B. Genetische Suche, Clustering, Remote-Zugriff auf andere Agenten u. a.

Ein Vorteil von ABLE ist die Fähigkeit, subsymbolische und symbolische Verfahren in einem AbleAgent zusammenzuführen.

#### Beispiel: Assistent zur Entscheidungsunterstützung (DM-Agent)

Im Beispielfall geht es um die Unterstützung von Entscheidungsträgern bei der Kreditvergabe. Dem Agenten werden die Daten bereits vergebener Kredite präsentiert, zusammen mit der Angabe, ob diese ordnungsgemäß zurückgezahlt wurden oder nicht (Trainingsdaten). Learning Beans versuchen auf Grundlage des Inputs eine Funktion zu approximieren<sup>77</sup>, welche die Klassifizierung in positive und negative Beispiele beschreibt (überwachtes Lernen). Diese wird dann die Basis zur Voraussage der Kreditwürdigkeit neuer Kreditanträge.

Das nächste Anwendungsbeispiel beschreibt den Einsatz von AbleAgents zur Überwachung und Steuerung von Hard- oder Software.

#### Beispiel: Autotune Agent

Heutige Computersysteme werden immer komplexer. Demzufolge wird es auch immer komplizierter, diese zu überwachen und zu steuern. Autonome Agenten können Administratoren bei diesen Aufgaben unterstützen. Durch Sensoren kontrolliert ein Agent den aktuellen Systemzustand, z. B. die Prozessorleistung oder die Anzahl der laufenden Tasks (im Falle eines Servers). Da er mit einem Satz von Metriken (oder einem Modell des Systems) ausgestattet ist, kann der Agent Abweichungen vom Normzustand feststellen. Sollte das der Fall sein, schlussfolgert der Autotune Agent entsprechend seines Wissens und nimmt gegebenenfalls Veränderungen an spezifischen Einstellungen vor, um den gewünschten Zustand wiederherzustellen. Über seine Sensoren kann er den Erfolg oder Misserfolg seiner Aktionen

---

<sup>77</sup> Z. B. durch NN. Der Output wird mit „semantischen Labeln“ versehen und kann im Anschluss mittels einer Inferenzmaschine ausgewertet werden.

ermitteln und erforderlichenfalls eine andere Strategie verfolgen. (Der Autotune Agent wurde prototypisch für einen Apache Webserver implementiert. In ähnlicher Art und Weise funktioniert die IBM Server Diagnostik.)

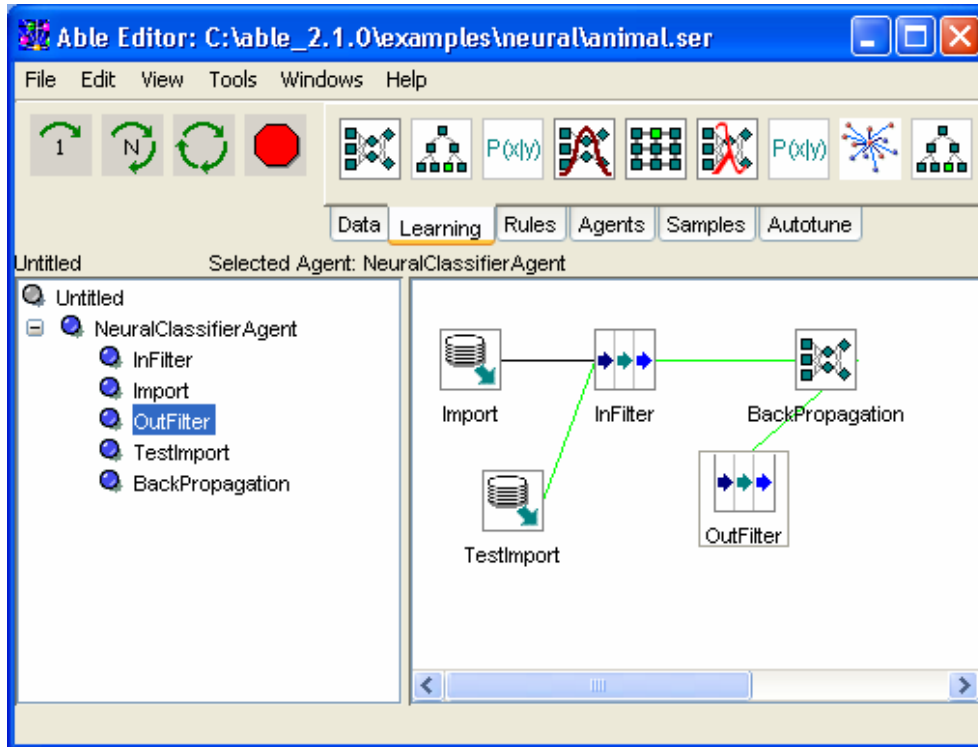


Abb. 7-1: Able Agent Editor

## 7.2 JAVA DEvelopment Framework (JADE)

JADE ist eine javabasierte, frei verfügbare Entwicklungsumgebung. Durch den flexiblen Aufbau von JADE-Agenten (siehe Abb. 7-2) wird der Entwickler nicht auf eine spezielle Agentenarchitektur (reaktiv, BDI-Agent, hybrid) festgelegt. JADE ist auf allen Systemen mit Java in der Version 1.4 und aufwärts lauffähig. In Verbindung mit der Integration der Lightweight Extensible Agent Platform (LEAP) kann es auch auf mobilen Endgeräten und Terminals eingesetzt werden.

**Tab. 7-2:** Überblick JADE

Technische Voraussetzungen	Java ab Version 1.4
Version / Lieferumfang	Version 3.2 / Framework, Dokumentation, Beispiele, Plattform (Runtime-Environment), Agenten für Standardaufgaben
Lizenz	GNU Library or Lesser General Public License (LGPL)
Hersteller	Telecom Italia Lab
Plattform erfüllt FIPA-Standard	ja
Kommunikation via	Sprache: FIPA-ACL Protokolle: HTTP, IIOP, RMI ...
Wesentliche Features	<ul style="list-style-type: none"> <li>• Graphische Unterstützung vor allem beim Testen und Debugging von agentenbasierten Applikationen</li> <li>• Ontologiemanager</li> <li>• Viele eingebaute Kommunikations- und Interaktionsprotokolle</li> <li>• Integration von LEAP</li> <li>• Unterstützung von Java Expert Systems Shell (JESS)</li> </ul>
Link	<a href="http://jade.tilab.com/">http://jade.tilab.com/</a>

### Komponenten für integriertes Lernen

JADE verfügt über *keine* explizite Unterstützung von Lern- und Trainingsverfahren. Vorgesehen ist die Integration mit JESS, einem in Java implementiertem Expertensystem. JESS kann Java-Objekte erstellen und manipulieren, Java-Methoden aufrufen sowie Java-Events verarbeiten. Ein Anwendungsbeispiel für das Zusammenwirken von JADE, JESS und weiteren Technologien ist die Agent Academy (siehe Abschnitt 7.5). Daneben gibt es für JADE-Agenten ein optionales built-in-Verhalten (behaviour), welches über die JADE-Bibliothek verfügbar gemacht wird (siehe auch Abb. 7-2). Beim Eingang jeder Nachricht sucht dann eine JESS-Engine nach Fakten in einer JESS-Regelbasis<sup>78</sup> (KnowledgeBase), welche den Umgang mit dieser beschreiben. Dadurch kann das Verhalten der Agenten entsprechend dem Inhalt der Nachrichten angepasst werden (Aufruf oder Unterbrechen von behaviors).

<sup>78</sup> Die Regelbasen lassen sich zur Laufzeit austauschen.

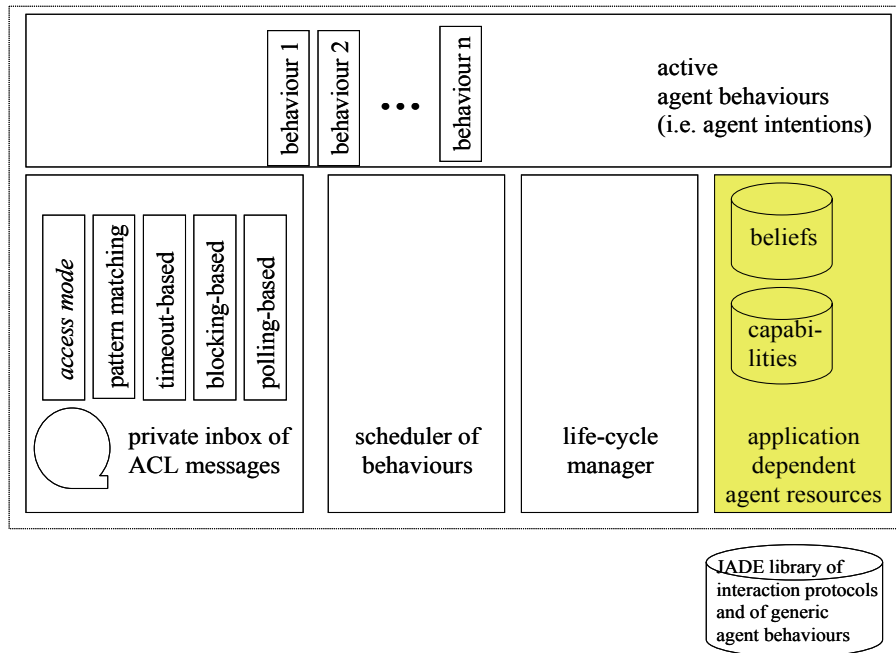


Abb. 7-2: Architektur eines JADE-Agenten

### 7.3 JACK™ Intelligent Agents

Die Agent Oriented Software Group (Hersteller) bezeichnet JACK als führend auf dem Gebiet der kommerziellen Agentenentwicklungssysteme. JACK ist sowohl für die Entwicklung kognitiver, als auch reaktiver Agenten ausgelegt und läuft auf PDA's, Laptop's, PC's oder High-End Rechnern, die Java ab Version 1.3 (und höher) unterstützen.

Tab. 7-3: Überblick JACK

Technische Voraussetzungen	Java ab Version 1.3, 2 MB freier Festplattenspeicher
Version / Lieferumfang	Version 4.1 / Suite von Entwicklungswerkzeugen, Dokumentation, Beispiele, Compiler, Plattform (Runtime-Environment)
Lizenz	Kommerzielles Produkt
Hersteller	Agent Oriented Software Group
Plattform erfüllt FIPA-Standard	Keine Angabe (nein)
Kommunikation via	Sprache: keine (Versenden von Objekten mittels Binärformat [JACOB]) Protokoll: TCP-basiert

Wesentliche Features	<ul style="list-style-type: none"> <li>• Graphische Unterstützung beim Erstellen, Testen und Debugging von agentenbasierten Applikationen</li> <li>• Agentenorientierte Programmiersprache, die auf Java aufsetzt (JACK Agent Language, JAL)</li> <li>• Graphische Editoren zur Bearbeitung von (Team-)Plänen und Rollen mit automatischer Codegenerierung</li> <li>• Agenten können schlussfolgern</li> <li>• u. a.</li> </ul>
Link	<a href="http://www.agent-software.com">http://www.agent-software.com</a>

### **Komponenten für integriertes Lernen**

Eine explizite Unterstützung für das Training bzw. Lernen in MAS ist *nicht* vorgesehen. Trotzdem werden dem Programmierer einige Hilfestellungen bei der Entwicklung intelligenter adaptiver Agenten gegeben.

#### Weltmodell von BDI-Agenten

So genannte BeliefSets enthalten das Wissen des Agenten über seine Welt. Sie werden durch den Entwickler in Form eines tupelbasierten Modells offline codiert (JAL), welches die Grundlage für alle Schlussfolgerungsprozesse durch die integrierte Inferenzmaschine bildet. Die logische Konsistenz eines BeliefSet wird automatisch überprüft und sichergestellt. Das geschieht auch bei Änderungen, die online durch den Agenten selbst vorgenommen werden. Das Update des BeliefSets löst automatische Events aus, die z. B. dazu genutzt werden können, den Planungsvorgang des Agenten neu zu starten.

#### (Team-)Pläne / „Meta-level reasoning“

Ein Plan beschreibt eine Folge von Aktionen, die der Agent ausführt, wenn ein bestimmtes Ereignis eintritt. Jeder Plan kann nur einen bestimmten Ereignistyp behandeln. Für einen Ereignistyp kann es aber mehrere Pläne geben. „Normale“ Events führen dazu, dass der erste relevante (für diesen Event zutreffende) und anwendbare (wird durch Aufruf der context()-Methode festgestellt [Abprüfen von Bedingungen]) Plan ausgeführt wird. Diese Vorgehensweise eignet sich für die Modellierung reaktiven Verhaltens. Für adaptives sind die „BDI-Events“ von Bedeutung. Sie erlauben das längerfristige Verfolgen von Zielen. Der Unterschied zur „normalen“ Ereignisverarbeitung besteht zum einen in der Art und Weise der Auswahl eines Plans aus einem Set von Plänen, zum anderen im Umgang mit Situationen, in denen ein Plan fehlschlägt.

- Schlussfolgerungen auf einer Meta-Ebene (Meta-level reasoning): Ein „Meta-Plan“ beschreibt, wie die Auswahl aus einer Menge von relevanten und anwendbaren Plänen durchgeführt werden soll, beispielsweise durch Nutzen von Heuristiken.
- Auswahl eines alternativen Plans, falls der Originalplan fehlschlägt<sup>79</sup>: Der Agent versucht, sein Ziel auf mehreren Wegen zu erreichen.
- Erneutes Zusammenstellen der Menge von relevanten und anwendbaren Plänen bei einer Änderung der BeliefSets

Das Lernen erfolgt also nach folgendem Muster: Beim Versagen eines Plans wird ein weiterer (soweit vorhanden) getestet. Jeder zur Ausführung gekommene Plan wird durch Attribute (z. B. erfolgreich, fehlgeschlagen, weitere Parameter) gekennzeichnet, welche vor einer Wiederverwendung ausgewertet werden (Meta-level reasoning). Die vorgestellten Techniken können auch für Gruppen von Agenten (Teams) verwendet werden. Pläne können mit dem Plan Graph Editor visualisiert und komfortabel bearbeitet (u. a. automatische Codegenerierung) werden.

## 7.4 Soar

Soar kann auf PC's unter den Betriebssystemen Windows, Macintosh und Linux betrieben werden. Im Gegensatz zu allen anderen vorgestellten Toolsets liegt der Schwerpunkt auf der Konstruktion einzelner intelligenter Agenten. Die Infrastruktur für Kommunikation und Koordination zwischen den Agenten (Basis eines MAS) muss durch den Programmierer bereitgestellt werden (Sie ist, soweit vorhanden, rudimentärer Natur [siehe weiter unten]). Das Gleiche gilt für den Support von verteilten Anwendungen. Soar eignet sich für Agenten mit kognitiver Architektur (BDI-Agent). Die Interaktion mit der (simulierten) Umwelt wird explizit durch spezielle Input/Output-Funktionen übernommen (ermöglichen auch die Anbindung an andere Applikationen). Die Soar-Engine verarbeitet dabei Regeln und Sensorinformationen so effizient, dass Soar-Agenten auch in zeitkritischen (real-time) Anwendungen eingesetzt werden können, z. B. Flugsimulatoren oder Ego-Shootern.

---

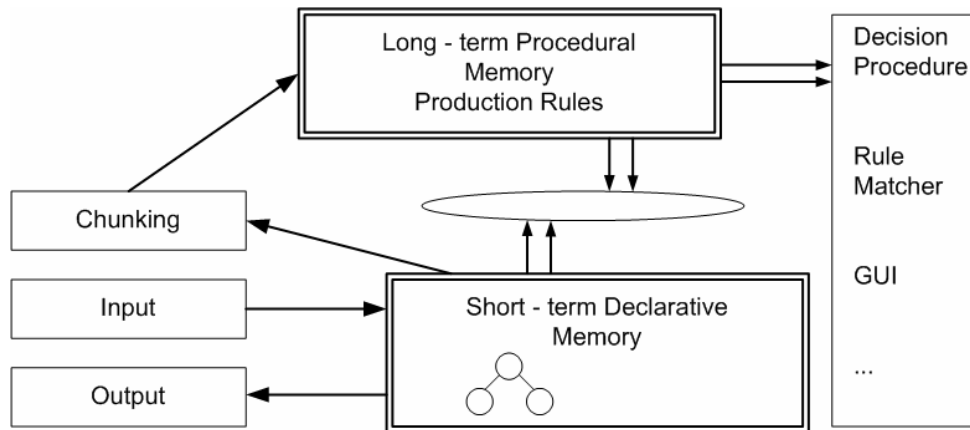
<sup>79</sup> Hinweis: Bei der Verarbeitung von „normalen“ Ereignissen wird bei Nichtausführbarkeit/Abbruch des gewählten Plans einfach zum nächsten Event übergegangen.

**Tab. 7-4:** Überblick Soar

Technische Voraussetzungen	keine speziellen, basiert auf Tcl – im Paket enthalten
Version / Lieferumfang	Version 8.5.2 / graphische Entwicklungstools, Dokumentation, Tutorial, Beispiele, Framework, Runtime-Environment
Lizenz	BSD Licence (Open Source)
Hersteller	Carnegie Mellon University, University of Michigan u. a.
Plattform erfüllt FIPA-Standard	nein
Kommunikation via	Sprache: keine (Tcl-Funktionsaufrufe, Soar I/O) Protokolle: keine
Wesentliche Features	<ul style="list-style-type: none"> <li>• Graphische Werkzeuge für das Erstellen, Testen und Debugging von intelligenten Agenten</li> <li>• Agenten verfügen „von Haus aus“ über die Fähigkeit zum Lernen (Chunking) und Planen</li> </ul>
Link	<a href="http://sitemaker.umich.edu/soar">http://sitemaker.umich.edu/soar</a>

### Komponenten für integriertes Lernen

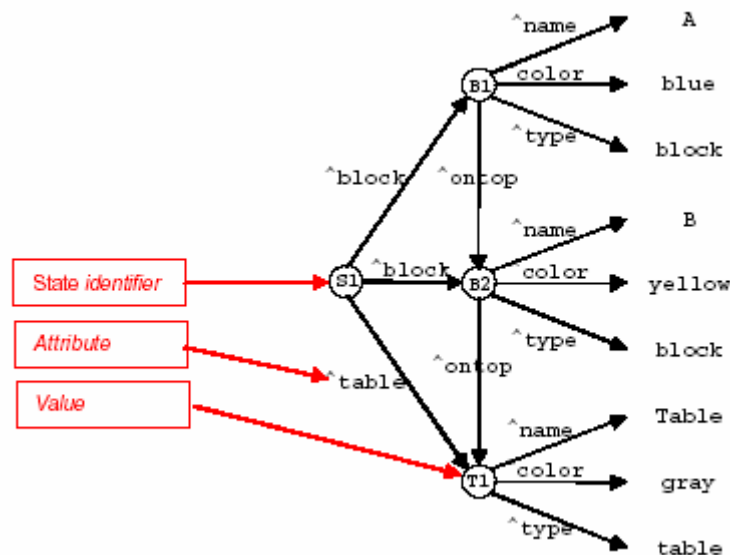
Der durch Soar zur Verfügung gestellte automatische Lernmechanismus für Agenten (Chunking) steht in einem engen Zusammenhang mit der zugrunde liegenden Agenten-Architektur (siehe Abb. 7-3). Langfristiges Wissen (unabhängig von den aktuellen Zielen des Agenten) befindet sich im Long-term Memory (LTM) und wird in Form von production rules (PR) abgelegt. Dagegen beschreibt das im Short-term (Working) Memory (WM) vorhandene dynamische Wissen den gegenwärtigen Zustand der Umwelt in der Struktur von hierarchisch organisierten „gelabelten“ Graphen (siehe Abb. 7-4). Ein Umweltzustand wird dabei indirekt durch die Menge aller Objekte mit ihren Attribut-Wert-Paaren im WM beschrieben. Manipulieren PR's das WM, spricht man von Operatoren. Zustandsänderungen können auch durch externe Einflüsse, z. B. Sensorinput, erfolgen.



Quelle: Laird (2004)

**Abb. 7-3:** Architektur eines Soar-Agenten

Funktionsweise eines Soar-Agenten: Ist eine PR feuerbereit (alle Vorbedingungen sind erfüllt), werden die von der Regel beschriebenen Aktionen sofort ausgeführt. Stehen mehrere PR's zur Disposition, durchlaufen sie einen Auswahlprozess. Nur diejenigen Aktionen, welche durch die „Sieger-Regel“ definiert werden, kommen dann zur Ausführung. Das zielgerichtete Lösen von Problemen erfolgt durch Suche in einem Zustandsraum. Ausgehend von einem Startzustand wird der Problemraum durch Tiefensuche expandiert. Wie zwischen Zuständen gewechselt wird, legen die Operatoren fest.



**Abb. 7-4:** Beschreibung von Umweltzuständen mittels gelabelter Graphen

Das Lernen in Soar erfolgt durch so genanntes Chunking, einer Form des EBL (siehe Abschnitt 6.1.1). Stark vereinfacht kann es wie folgt beschrieben werden: Die Suchpfade auf dem Weg zum Ziel (oder Teilzielen) werden aufgezeichnet. Endet die Suche nach einer Problemlösung in einer „Sackgasse“ (widersprüchliche oder unzulängliche Operatoren [führen nicht zum Ziel]), wertet Soar die bisherigen Schritte aus (Meta Reasoning) und versucht neue PR zu kreieren („Abkürzungen“; Es erfolgt eine Neukombination von vorhandenem Wissen.), sodass die augenblickliche Situation bewältigt werden kann. Bei der Konfrontation des Agenten mit einem gleichen (und nur ähnlichen) Problem, löst er dieses dann effizienter. Der Erfolg der Lernmethode ist stark abhängig vom Umfang und der Korrektheit der Wissensbasis.

## 7.5 Agent Academy

Die Agent Academy wurde durch ein Konsortium von Universitäten und Firmen bis zum 01. November 2003 als EU-Forschungsprojekt entwickelt. Seit diesem Zeitpunkt wird sie als Open-Source-Initiative weiterbetrieben.

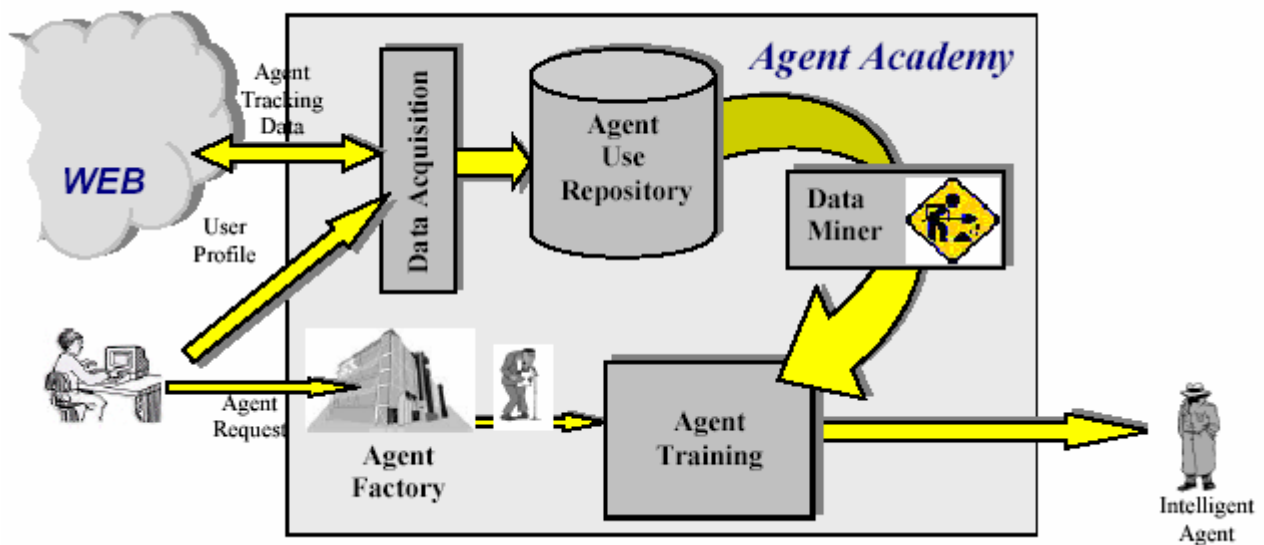
**Tab. 7-5:** Überblick Agent Academy

Technische Voraussetzungen	Java ab Version 1.4, ant ab Version 1.5.1, PostgreSQL-Datenbank ab Version 7.2, J2EE kompatibler Web-Server und diverse andere
Version / Lieferumfang	Version 1.2 / verschiedene graphische Entwicklungstools, Dokumentation, Beispielanwendung, Framework und Plattform (Runtime-Environment) basierend auf JADE, Agenten mit spezifischen Aufgaben innerhalb MAS (Agent Academy)
Lizenz	GNU Library or Lesser General Public License (LGPL)
Hersteller	Agent Academy Consortium
Plattform erfüllt FIPA-Standard	ja
Kommunikation via	Sprache: FIPA-ACL Protokolle: HTTP, IIOP, RMI u. a.
Wesentliche Features	<ul style="list-style-type: none"> <li>• Graphische Unterstützung beim Erstellen, Testen und Debugging von agentenbasierten Applikationen</li> <li>• Ontologiemanager</li> <li>• Datenbank, Speicherung von relevanten (mit den Aktionen der Agenten in Beziehung stehenden) Umweltdaten</li> </ul>

	<ul style="list-style-type: none"> <li>• DM-Agent, Umwandlung des extrahierten Wissens in JESS-Regeln</li> <li>• Agenten können schlussfolgern</li> </ul>
Link	<a href="http://agentacademy.iti.gr/">http://agentacademy.iti.gr/</a>

### Komponenten für integriertes Lernen

Der durch die Agent Academy verfolgte Ansatz unterscheidet sich grundlegend von den bisherigen. Zum Verständnis der nachfolgenden Ausführungen ist die Kenntnis des strukturellen Aufbaus der Academy von Vorteil (siehe Abb. 7-5; Erklärungen sind in die Beschreibung des Trainings mit eingebunden).



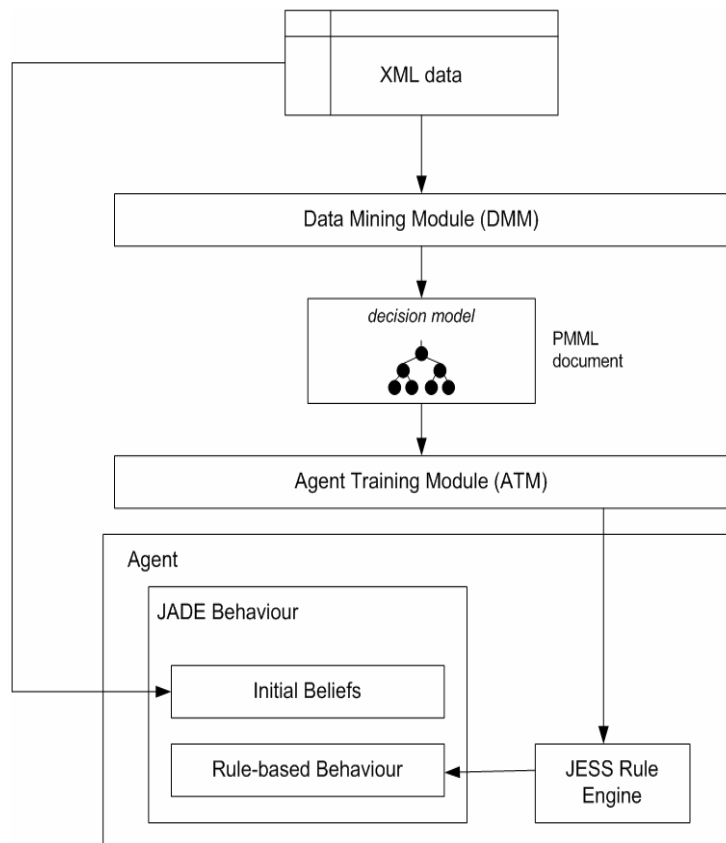
Quelle: Mitkas (2002)

**Abb. 7-5:** Aufbau Agent Academy

Durch die Agent Academy können neue Agenten „geschult“ oder vorhandene Agenten einem Retraining unterzogen werden. Der Trainingsprozess (siehe Abb. 7-5 und Abb. 7-6) verläuft idealisiert wie folgt:

- Die Agent Academy erhält den Auftrag, einen Agenten zu trainieren. Dazu spezifiziert der Auftraggeber (Application Developer) über ein Interface (Protege'-2000 Front End) die funktionalen Anforderungen im Zusammenspiel mit der Agent Factory (AF). Anschließend wird der Basis-Code für den untrainierten Agenten (UA) generiert.

- Sofern erforderlich, wird danach das Agent Training Module (ATM) benachrichtigt. Dieses bestimmt anhand der Kundenanforderungen, über welches Verhaltensrepertoire (behaviors; Einzelne Verhaltensregeln sind als Java-Klassen abgelegt [bzw. JESS-RuleSet].) der Agent verfügen muss. Das ATM teilt sein Ergebnis dem UA mit. Dieser besitzt die Fähigkeit, die erforderlichen Klassen (behaviors) zu laden und steht damit für den Einsatz bereit. Da die Agenten nun die Fähigkeit zum Schlussfolgern benötigen, erhalten sie außerdem eine JESS-Engine.
- Den Kern der Agent Academy bildet das Agent Use Repository (AUR). Hier werden relevante (mit den Aktionen der Agenten in Beziehung stehende) Umweltdaten gesammelt. Der Aufbau bzw. die Pflege (Update) des AUR ist ein permanenter Prozess, an dem eine große Anzahl mobiler Agenten beteiligt ist. Gesteuert werden sie vom Data Acquisition Module. Die im AUR vorhandenen Daten werden durch Data Mining Module (DMM) bearbeitet, um nützliches Wissen (hier Entscheidungsmodelle für die Aktionsauswahl) zu extrahieren. Gefundene Modelle werden durch das ATM in behaviors (JESS-RuleSets) übersetzt.



Quelle: Mitkas (2003)

**Abb. 7-6:** Schematische Darstellung des Trainings in der Agent Academy

## 8 Bewertung und Auswahl effizienter Trainingsverfahren

Wie bereits in Kapitel 5 dargestellt, gibt es z. Z. nur wenige Ansätze und quantitative Erfahrungen zur Softwaremessung agentenbasierter Systeme. Dort wurden auch die auftretenden Probleme einer *generelleren* Bewertung der Effizienz von Trainingsverfahren besprochen. Dementsprechend schwer fällt es, *allgemeine* Aussagen zur Auswahl von Lern- und Trainingsformen unter Effizienzaspekten zu treffen. In einem ersten Schritt kann man Übersichten wie die Tab. 8-1 und Tab. 8-2 zusammenstellen. Sie listen die Anwendungsbeispiele für effizientes Training aus Kapitel 6 noch einmal auf. Im Allgemeinen führt es zu Steigerungen der Lern- und Adaptionseffizienz sowie nachfolgend der Aktions- und Leistungseffizienz (*Produkteffizienz/Performance*) bzw. entsprechender Größen in MAS.

**Tab. 8-1:** Effiziente Trainingsmethoden I

Einzelner Agent		Effizienzverbesserung / Ansatz
Aktionselektion		
1	Emailfilter MAXIMS	Verbesserung der Anzahl der richtig bearbeiteten (klassifizierten) Emails durch größer werdende Erfahrungsbasis (Trainingsbeispiele) / Case-Based Reasoning und RL
2	Roboter-Navigation	Finden eines (fast) optimalen Weges unter Vermeidung von Zusammenstößen; wiederholtes Lösen des gleichen Problems / RL bzw. Classifier-Systeme
Agenten-Wissen		
3	Modellbildung - das Relevante lernen	Vergrößerung der Wissensbasis durch Sammeln von Erfahrungen; (zielorientertes) Experimentieren des Agenten; Beschränkung auf relevantes Wissen / Lernen eines statistischen Modells

In MAS werden zusätzlich Verbesserungen der *Koordinations-, Kommunikations- oder Interaktionseffizienz* (u. a.) erreicht.

Tab. 8-2: Effiziente Trainingsmethoden II

Multi-Agenten-System		Effizienzverbesserung /Ansatz
<b>Koordination</b>		
4	CIRL, Koordination ohne Kommunikation	Block-Welt; Agenten bewegen einen Block entlang eines vorgegebenen Pfades zur Zielposition; die Anzahl der notwendigen Schritte sinkt; in Kooperation oder Konkurrenz; wiederholtes Lösen des gleichen Problems / RL
5	Gruppen koordinieren	Block-Welt; Agenten stapeln Blöcke nach vorgegebenen Regeln; die Anzahl der notwendigen Schritte sinkt; wiederholtes Lösen des gleichen Problems / RL
6	Rollen in Organisationen lernen	Agenten lernen, in welcher Rolle sie den höchsten Nutzen für die Gruppe erzielen (Design eines Dampfkondensators); wiederholtes Lösen des gleichen Problems / RL
7	Präferenzen anderer Agenten lernen	Terminplanung; Reduzierung des Kommunikations- und Koordinationsaufwands durch Voraussage der Präferenzen anderer Agenten; Online / Bayesian Classifier
8	Entscheiden, wann und wie man Aktionen koordiniert	Gitternetzwelt; Agenten optimieren ihren Nutzen durch Auswahl des „richtigen“ Koordinationsmechanismus; wiederholtes Lösen des gleichen Problems / RL
<b>Kommunikation</b>		
9	Roboter-Koordination	Roboter; Aufgaben können nur effizient in der Gruppe gelöst werden; Reduzierung der dazu notwendigen Kommunikation; Verbesserungen durch größer werdende Erfahrungsbasis / Cased-Based Reasoning
10	Kommunikation zur Verbesserung der Koordination	Jäger-Beute-Welt; Agenten tauschen ihre Erfahrungen aus, um effizienter beim „Jagen“ zu werden; wiederholtes Lösen des gleichen Problems / RL

(Williams (2003)) verfolgen eine ähnliche Herangehensweise. Sie stellen einen Bezug zwischen Praxisbeispielen, verwendeten Trainingsverfahren und Einsatzdomänen her (siehe Tab. 8-3).

**Tab. 8-3:** Effiziente Trainingsmethoden III

Reinforcement Learning	RoboCup, Aufzugskontrolle, Netzwerk Routing, Wegsuche, Jäger-Beute-Welt, Block-Welt, Roboter-Navigation
Rule Based Learning	RoboCup, Jäger-Beute-Welt, Roboter-Navigation, Board Game (gegnerische Strategien lernen)
Neuronale Netze	RoboCup, Jäger-Beute-Welt, Pool Balancing (Gleichgewichtsproblem), Virtual Life
Genetische Algorithmen	Jäger-Beute-Welt, Board Game
Statistische und andere Lernmethoden	Traffic Signal Control, Jäger-Beute-Welt, Board Game, Marktplatz (Verhandlungen), Rollen in Organisationen

Quelle: vgl. Williams (2003)

Derartige Übersichten verfolgen das Ziel<sup>80</sup>, Trends bei der Verwendung von Lern- und Trainingsmethoden aufzudecken<sup>81</sup>. Dem liegt die Annahme zugrunde, dass bei der Auswahl des jeweils zum Einsatz gekommenen Verfahrens Effizienzkriterien ausschlaggebend waren (Rationalitätsprinzip).

Schwachstellen des gewählten Vorgehens sind die nachstehenden Punkte:

- Ist die o. g. Annahme berechtigt?
- Es gibt keine standardisierten Problemklassen.
- Die Einteilung erfolgt subjektiv durch Autoren.
- Es wird nur eine geringe Anzahl von Fällen untersucht (statistische Relevanz).
- Die Aufstellungen sind statische Schnappschüsse (zeitpunktbezogene Auswertungen).

Trotzdem können derartige Übersichten wertvolle Anhaltspunkte liefern.

Aus den in der Literatur verfügbaren Erfahrungen lassen sich weiterhin einige empirische Aussagen gewinnen (Faustregeln):

<sup>80</sup> in Ermangelung anderer Möglichkeiten (siehe Abschnitt 5.1 bzw. Dumke (2000))

<sup>81</sup> Gleichzeitig lassen sich so auch „weiße Flecken“ finden – Themenkomplexe, für die erst wenige Lern- und Trainingsverfahren entwickelt wurden.

- Bei der Entscheidung für oder gegen ein spezielles Lernverfahren ist der durch die zu lösende Aufgabe und die speziellen Gegebenheiten vor Ort vorgegebene Rahmen von großer Bedeutung, wie z. B. die technischen Voraussetzungen, Anforderungen an die MAS-Architektur, der Problemtyp, die Kompetenzen der Mitarbeiter usw. (vgl. Brenner (1998)).
- Es ist zu prüfen, ob nicht herkömmliche Systeme eingesetzt werden können. „Indeed, many of the systems that have been build using agent technology could likely have been built just as easily with nonagent techniques.“ (Wooldridge (1999)).
- In Domänen, die viel Hintergrundwissen erfordern (Verhandlungen, Organisationen), haben regelbasierte Systeme Vorteile. In „einfachen“ Welten (Block-Welt, Jäger-Beute-Welt) empfiehlt sich die Verwendung von Formen des RL oder des statistischen Lernens (vgl. Laird (2004)).
- A priori Wissen beschleunigt das Lernen (Russel (1995)), da die Anzahl der Hypothesen im Lösungsraum abnimmt (Aussortieren von falschen Hypothesen) bzw. noch einfachere gefunden werden können.
- Inkrementelle Verfahren sind zur kontinuierlichen Integration neuer Beispiele in große Fallsammlungen vorteilhaft (vgl. Puppe (2001)).

Die Einschätzung von Agenten-Entwicklungsumgebungen in Bezug auf eine potentielle Steigerung der Effizienz bei der praktischen Umsetzung von Lernprozessen (beispielsweise durch Verbesserung der CASE-Effizienz [*Prozesseffizienz*] oder Entwicklereffizienz [*Ressourceneffizienz*]) gestaltet sich einfacher. Die Beurteilung kann anhand von objektiven Produktmerkmalen erfolgen. Ein Problem stellen allerdings wiederum die fehlenden *agentenorientierten* Standards dar. Eine Möglichkeit der Bewertung zeigen (Weiß (2005)) auf. Sie evaluieren Entwicklungsmethodologien und -umgebungen für MAS (ohne den Aspekt Lernen/Training) unter Bezugnahme auf das Bearbeiten gleicher Aufgabenstellungen (gemeinsames Anwendungsszenario). Die Basis bilden Mess- und Bewertungsformen des OOSE (erweitert um agentenspezifische Kriterien).

Außerdem können als Ausgangspunkt für weitere Überlegungen wiederum Aufstellungen dienen, welche einen Überblick der verfügbaren Toolsets und ihrer Merkmale<sup>82</sup> geben.

---

<sup>82</sup> Eine Eingrenzung auf relevante Attribute erscheint sinnvoll.

**Tab. 8-4:** Effiziente Trainingstechnologien

Entwicklungsumgebung		Effizienzverbesserung / Ansatz
1	ABLE	grundlegende ML-Verfahren sowie verschiedene Inferenzmaschinen werden als separate Bausteine (wieder verwendbare Softwarekomponenten) zur Verfügung gestellt; Baukastensystem
2	JADE	keine oder nur geringe Unterstützung, z. B. durch eine mögliche Integration von JESS (Inferenzmaschine) und graphische Tools zum Bearbeiten von Wissensbasen, Ontologien etc.
3	JACK	keine oder nur geringe Unterstützung, z. B. durch eine integrierte Inferenzmaschine und graphische Tools zum Bearbeiten von Plänen, Wissensbasen, Ontologien etc.
4	Soar	in Agenten(architektur) integrierte(s) Lernverfahren (Chunking, Form des EBL) für einzelne Agenten
5	Agent Academy	Bereitstellung trainierter Agenten aufgrund von Benutzerspezifikationen; erforderliche Wissensbasis wird durch DM-Methoden generiert

Bei der Auswahl eines Software-Werkzeugs darf der Punkt „Lernen/Training“ nicht separat betrachtet werden. Wichtig ist dessen *umfassende* Unterstützung des Softwareentwicklungsprozesses bzw. dessen *Integration* in eine bestehende Infrastruktur. Weiterhin sollten projektspezifische Gegebenheiten eine Rolle spielen.

## 9 Schlussbetrachtung und Ausblick

In dieser Arbeit wurde ein Überblick über effiziente Trainingsverfahren in MAS gegeben. Anhand von Praxisbeispielen konnte die Leistungssteigerung lernfähiger Agenten exemplarisch nachvollzogen werden. Die Entwicklung geht dabei von „einfachen“ Algorithmen hin zu komplexen, die die Zusammenarbeit in menschlichen Arbeitsgruppen nachempfinden (kognitive Konzepte) oder hierarchische Strukturen (layered learning) verwenden.

Weiterhin wurden Agenten-Entwicklungsumgebungen hinsichtlich ihrer Unterstützung bei der effizienten Programmierung adaptiver Agenten untersucht. Dabei waren verschiedenste Ansätze feststellbar. Sie bedürfen einer weiteren intensiven, praktischen Erprobung. Außerdem sind funktionale Erweiterungen wünschenswert. Eine Durchsetzung der Toolsets hängt allerdings nicht zuletzt von der Akzeptanz der Agenten-Technologie im Allgemeinen ab.

Die Softwaremessung agentenbasierter Systeme steht erst am Anfang der Entwicklung. Ihre Bedeutung wird aber immer mehr erkannt. Z. Z. fehlen noch *agentenorientierte* Standards und quantitative Erfahrungen. Dementsprechend schwer fällt es, *allgemeine* Aussagen zur Bewertung und Auswahl von Lern- und Trainingsverfahren unter Effizienzaspekten zu treffen. Es wurden erste Bewertungsformen der Effizienz für das Agenten-Training aufgezeigt und diskutiert. In diesem Zusammenhang gibt es zahlreiche offene Fragen und Probleme, die Gegenstand zukünftiger Arbeiten sein sollten:

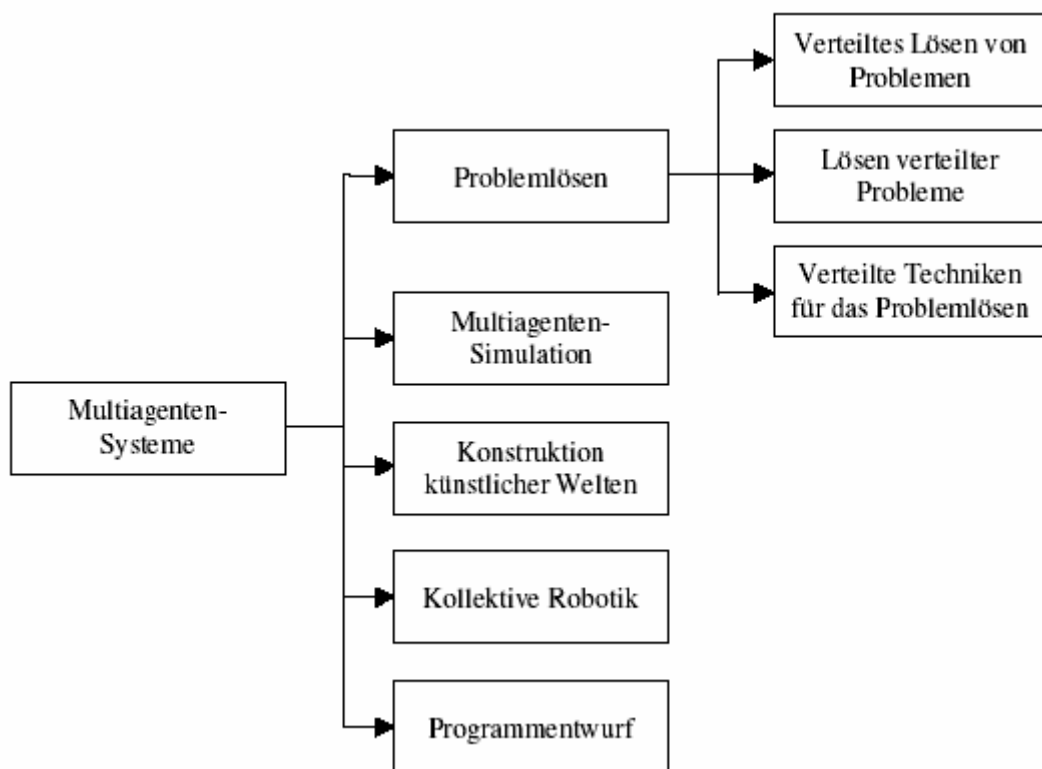
- Definieren von einheitlichen Problemklassen (Einsatzdomänen) für MAS; Auf deren Basis könnten Effizienzmessungen von Trainingsmethoden durch die Bearbeitung von Standardaufgaben erfolgen.
- Schaffung von agentenorientierten Mess- und Bewertungsstandards
- Schaffung von Bibliotheken mit Kennzahlen bereits umgesetzter Projekte für die Effizienz des Trainings in MAS (Möglichkeit zum Benchmarking)
- Kataloge mit verfügbaren Lernalgorithmen bzw. Bausteinen zur Integration in eigene Programme
- Verbesserte Toolunterstützung bei der Softwaremessung, z. B. durch die erwähnte Selbstmessung mittels Messagenten.

Die Agenten-Technologie besitzt ein großes Potential, wie diese Arbeit gezeigt hat. Zur breiteren Akzeptanz und Anwendung in der (industriellen) Praxis ist es allerdings notwendig, verbindliche Standard zu schaffen sowie Sicherheitsbedenken auszuräumen.

## Anhang

### A Einsatzdomänen von MAS

Ferber unterteilt die Anwendungsbereiche für MAS in fünf grundsätzliche Kategorien (siehe Abb. A-1):



Quelle: Ferber (2001)

**Abb. A-1:** Eine Klassifikation der verschiedenen Anwendungstypen für Multi-Agenten-Systeme

Die folgende Tabelle listet wichtige Einsatzdomänen von Agenten auf. Sie soll einen ersten Eindruck der vielfältigen Möglichkeiten der Agenten-Technologie geben. Ein Teil der genannten Beispiele stammt aus dem universitären Umfeld und besitzt Prototypcharakter. Es gibt aber auch eine ganze Reihe von Anwendungen, die den Sprung hin zur kommerziellen Nutzung geschafft haben. Ein aktuelles Beispiel liefert die Raumfahrt (<http://eo1.gsfc.nasa.gov/>). Agenten werden von der NASA zur Fernüberwachung von Erderkundungssatelliten eingesetzt. (Hinweis: Die Reihenfolge bei der Aufzählung der Domänen spiegelt nicht deren Bedeutung wieder.)

**Tab. A-1:** Einsatzgebiete von Agenten

Domäne	Beispiel
Automatisches Design	RAPPID (Responsible Agents for Product-Process Integrated Design)
autonome Roboter	RoboCup (Roboter-Fußball)
Business process management	ADEPT (Agent-Based Business Process Management)
E-Learning	Semantische eLearning-Agenten (FH Hannover)
Electronic Commerce	ShopBots bei amazon, elektronische Marktplätze – Kashbar (MIT)
Entertainment	Quake (Ego-Shooter, Computergegner)
Informationssuche	Meta-Crawler
Interface-Agenten	VIENA (Virtual Environments and Agents, Uni Bielefeld)
Luftverkehrsüberwachung	Flughafen Sydney (OASIS)
Logistik	<a href="http://www.industrieanzeiger.de/O/108/Y/81600/VI/30307008/default.aspx">http://www.industrieanzeiger.de/O/108/Y/81600/VI/30307008/default.aspx</a> (Materialflusssteuerung; Siemens, Fraunhofer Institut)
Medizin	ADAPT (Adaptive Multi Agent Process Planning & Coordination of Clinical Trials, Uni Hohenheim)
Militär	Flugsimulator Tec-Air (US-Army, Soar)
Persönliche Assistenten	MIA (Mobile Information Assistant)
Produktionssteuerung	Production 2000+ (Daimler-Chrysler)
Raumfahrt	Software der Sonde „Deep Space One“ (NASA)
Real-time control systems	ARCHON (ARchitecture for Cooperative Heterogeneous ON-line systems)
Scheduling	AARIA (Autonomous Agents for Rock Island Arsenal)
Simulation	Simulation des Straßenverkehrs der Schweiz mit MASSIVE

## Literaturverzeichnis

- Austin (1962) Austin, J. L.: How to do things with words. Cambridge, Harvard University Press, 1962
- Bigus (2001) Bigus, J.; Bigus, J.: Intelligente Agenten mit Java programmieren. eCommerce und Informationsrecherche automatisieren. München u. a., Addison-Wesley, 2001
- Brenner (1998) Brenner, W.; Zarnekow, R.; Wittig, H.: Intelligente Softwareagenten. Grundlagen und Anwendungen. Berlin u. a., Springer Verlag, 1998
- Brooks (1986) Brooks, R. A.: A robust layered control system for a mobile robot. In: IEEE Journal of Robotics and Automation, RA-2(1986)1, S. 14-23
- Brooks (1990) Brooks, R. A.: The Behaviour Language: Users Guide. In: M.I.T. Artificial Intelligence Laboratory, AI Memo 1227
- Brooks (1991) Brooks, R. A.: Intelligence without representation. In: Artificial Intelligence, 47(1991), S. 139-159
- Ciancarini (2001) Ciancarini, P.; Wooldridge, M. J.: Agent-Oriented Software Engineering. Lecture Notes in Computer Science 1957, Berlin u. a., Springer-Verlag, 2001,
- Dilger (2003) Dilger, W.: Multiagentensysteme. Vorlesung WS 2003/2004, <http://www.tu-chemnitz.de/informatik/HomePages/KI/>, 06. Dezember 2004
- Doran (1997) Doran, J. E.; Franklin, S.; Jennings, N. R.; Norman, T. J.: On Cooperation in Multi-Agent Systems. In: The Knowledge Engineering Review, 12(3), 1997, S. 309-314
- Drescher (1991) Drescher, G.: Made Up Minds: A Constructivist Approach to Artificial Intelligence. Cambridge, MIT Press, 1991
- Dumke (2000) Dumke, R.; Koeppel, R.; Wille, C.: Software Agent Measurement and Self-Measuring Agent-Based Systems. Preprint Nr. 11, Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, 2000
- Dumke (2003) Dumke, R.: Software Engineering: Eine Einführung für Informatiker und Ingenieure. Systeme, Erfahrungen, Methoden, Tools. 4. Aufl., Wiesbaden, Vieweg Verlag, 2003
- Dumke (2003a) Dumke, R.: Vorlesung Verteilte Systementwicklung (2003), <http://ivs.cs.uni-magdeburg.de/~dumke/STV/ST3-06.html>, 15. August 2004
- Ferber (2001) Multiagenten-Systeme: Eine Einführung in die Verteilte Künstliche Intelligenz. München u. a., Addison Wesley, 2001
- Finin (1994) Finin, T.; Fritzon, R.; McKay, D.; McEntire, R.: KQML as an Agent Communication Language. In: Proceedings of the third International Conference on Information and Knowledge Management (CIKM'94), ACM Press, 1994

- Fisher (1995) Fisher, M.: Representing and executing agent-based systems. In: (Wooldridge (1995)), S. 307-323
- Foner (1994) Foner, N. L.; Maes, P.: Paying Attention to What's Important: Using Focus of Attention to Improve Unsupervised Learning. M.S. Thesis, Massachusetts Institute of Technology, 1994
- Franklin (1996) Franklin, S.; Graesser, A.: Is it an Agent, or just a Program?: A Taxonomie for Autonomous Agents. In: Müller, J. P., Wooldridge, M. J.; Jennings, N. R. (Hrsg.): Intelligent Agents III. Proceedings of the ECAI'96 Workshop on Agent Theories, Architectures, and Languages. Berlin et al., Springer Verlag, 1997, S. 21-35
- Garland (2004) Garland, A.; Alterman, R.: Autonomous Agents that Learn Better to Coordinate. In: Autonomous Agents and Multi-Agent Systems, Vol. 8, Issue 3 (2004), S. 267-301
- Glaser (2002) Glaser, N.: Conceptual modelling of multi-agent systems: The CoMoMAS engineering environment. In: Multiagent Systems, Artificial Societies, and Simulated Organizations (MASA), Vol. 4, Kluwer Academic Publishers, 2002
- Huhns (1999) Huhns, M. N.; Stephens, L. M.: Multiagent Systems and Societies of Agents. In: Weiss (1999), S. 79-120
- Jennings (1998) Jennings, N. R.; Sycara, K.; Wooldridge, M. J. (1998): A Roadmap of Agent Research and Development. In: Int. Journal of Autonomous Agents and Multi-Agent Systems, (1998), 1, S. 7-38
- Jennings (2003) Excelente-Toledo, C. B.; Jennings, N. R.: Learning When and How to Coordinante. In: Web Intelligence and Agent System, IOS Press, Vol. 1 (2003), Issue 3-4, S. 203-218
- Klügl (2004) Klügl, F.: Verteilte Künstliche Intelligenz: Grundlagen der Multiagentensysteme. Vorlesung SS 2004, <http://ki.informatik.uni-wuerzburg.de/teach/ss-2004/vki/>, 07. November 2004
- Knapik (1998) Knapik, M.; Johnson, J.: Developing Intelligent Agents for Distributed Systems., McGraw-Hill Education, 1998
- Kozierok (1993) Kozierok, R.; Maes, P.: A learning interface agent for scheduling meetings. In: Proceedings of the ACM SIGCHI International Workshop on Intelligent User Interfaces, ACM Press, 1993, S. 81-88
- Laird (2004) Laird, J. E.; Nason, S.: Soar-RL: Integrating Reinforcement Learning with Soar. ICCM 2004 (Pittsburgh, USA), <http://ai.eecs.umich.edu/people/laird/recent-research.html>, 13. Dezember 2004
- Lashkari (1994) Lashkari, Y.; Metral, M.; Maes, P.: Collaborative Interface Agents. In: Proceedings of AAI'94, Seattle 1994, <http://agents.media.mit.edu/publications/aaai-ymp/aaai.html>, 13. Dezember 2004

- Lesser (1995) Lesser, V. R.; Prasad, M. V. N.; Lander, S. E.: Learning Organizational Roles in a Heterogeneous Multi-agent System. In: Technical Report: UM-CS-1995-035, University of Massachusetts, 1995
- Lettmann (2000) Lettmann, T.; Schulz, A.: Vorlesung/Projektgruppe Intelligente Agenten I, WS 2000/2001, <http://www.wcs.upb.de/cs/ag-klbue/de/courses/> 13. Dezember 2004
- Lind (2001) Lind, J.: Iterative software engineering for multiagent systems: The MASSIVE method. In: Lecture Notes in Computer Science, Vol. 1994, Springer-Verlag, Berlin u. a., 2001
- Maes (1994) Lashkari, Y.; Metral, M.; Maes, P.: Collaborative Interface Agents. In: Proceedings of the Twelfth National Conference on Artificial Intelligence. Seattle (WA). AAAI Press, 1994
- Maes (1994) Maes, P.: Modeling Adaptive Autonomous Agents. In: Artificial Life Journal, 1(1994), Vol. 1+2, No. 9
- Maes (1995) Maes, P.: Artificial Life Meets Entertainment: Lifelike Autonomous Agents. In: Communications of the ACM, Vol. 38, No. 11, S. 108-114
- Mitchell (1997) Mitchell, T.: Machine Learning. McGrawHill, 1997
- Mitkas (2002) Mitkas, A. P.; Dogac, A.: An Agent Framework for Dynamic Agent Retraining: Agent Academy. e2002, Prag, <http://agentacademy.iti.gr/publications.htm>, 13. Dezember 2004
- Mitkas (2004) Mitkas, A. P.; Kehagias, D. Symeonidis, A. L.; Athanasiadis, I. N.: Framework for Constructing Multi-Agent Applications and Training Intelligent Agents. <http://agentacademy.iti.gr/pdf/aose2003.pdf>, 13. Dezember 2004
- Morin (1977) Morin, E.: La Méthode (1): la Nature de la Nature. Paris, Le Seuil, 1977
- Müller (1996) Müller, J. P.: The Design of Intelligent Agents. A Layered Approach. Berlin et al., Springer Verlag, 1996
- Newell (1980) Newell, A.: Physical symbol systems. In: Cognitive Science 4, S. 185-203
- Nwana (1996) Nwana, H., S.: Software Agents: An Overview. In: Knowledge Engineering Review, Vol. 11, No. 3, S. 205-244
- Ohko (1996) Ohko, T.; Hiraki, K.; Anzai, Y.: Addressee Learning and Message Interception for Communication Load Reduction in Multiple Robot Environments. In Weiß, G. (Hrsg.): Lecture Notes In Computer Science. Selected papers from the Workshop on Distributed Artificial Intelligence Meets Machine Learning. Learning in Multi-Agent Environments. Berlin u. a.; Springer-Verlag, 1996, S. 242-258

- Punter (2001) Punter, T.: Software Measurement with the GQM in Small and Medium Enterprise. In: Software-Metriken in der Praxis. Tagungsband des DASMA-Konferenz Metrikon 2001, Aachen, Shaker Verlag, 2001
- Puppe (2001) Puppe, F.: Vorlesung VKI, WS 2001/2002, <http://ki.informatik.uni-wuerzburg.de/teach/index.shtml>, 13. Dezember 2004
- Rao (1991) Rao, A. S.; Georgeff, M. P.: Modeling rational agents within a BDI-Architecture. In: Technical Report 14, Australian AI Institute, Carlton, 1991
- Rao (1995) Rao, A. S.; Georgeff, M. P.: BDI Agents: From Theory to Practise. In: Proceedings of the First International Conference on Multi-Agent-Systems (ICMAS), San Francisco, 1995
- Rumelhart (1986) Rumelhart, D. E.; Hinton, G. E.; Williams, R. J.: Learning internal representations by error propagation. In: Parallel Distributed Processing, Vol. 1, Cambridge, MIT Press, 1986
- Russel (1995) Russel, S. J.; Norvig, P.: Artificial Intelligence: A Modern Approach. London et al., Prentice Hall, 1995
- Sen (1998) Sen, S.; Sekaran, M.: Individual learning of coordination knowledge. In: Journal of Experimental and Theoretical Artificial Intelligence, Vol. 10 (1998), No. 3, S. 333-356
- Shoham (1993) Shoham, Y.: Agent oriented programming. In: Artificial Intelligence, Vol. 60, S. 51-92, 1993 (1)
- Sian (1991) Sian, S. S.: Adaption based on kooperative learning in multi-agent systems. In: Müller, J. P.; Demazeau, Y. (Hrsg.): Decentralised AI, Vol. 2 (1991), S. 257-272
- Stojanov (1996) Stojanov, S.: A Multi-Agent System for the Solution of Statistical Problems. In: Workshop „Concurrency, Specification, and Programming“, Berlin, S. 181-189
- Stone (2000) Stone, P.: Layered Learning in Multiagent system: A Winning Approach to Robotics Soccer. Cambridge, MIT Press, 2000
- Tan (1993) Tan, M.: Multi-agent reinforcement learning: Independent vs. cooperative agent. In: Proceedings of the Tenth International Conference on Machine Learning, S. 330-337, 1993
- Thomas (1993) Thomas, S. R.: PLACA, an Agent Oriented Programming Language. PhD thesis, Computer Science Department, Stanford University, Stanford, CA 94305
- Turing (1950) Turing, A.: Mind. A Quarterly Review of Psychology and Philosophy. In: Computing machinery and intelligence, Vol. 59 (1950), S. 433-460
- Venkatesh (1996) Venkatesh, S.; Bui, H. H.; Kieronska, D.: Learning other agents' preferences in multi-agent negotiation using the Bayesian classifier. International Journal of Cooperative Information Systems, Vol. 8(1999), No. 4, S. 275-294

- Wanner (2004) Wanner, L.: Vorlesung Intelligente Software Agenten, SS 2004, <http://www.iis.uni-stuttgart.de/lehre/ss04/IA/>, 13. Dezember 2004
- Weiß (1993) Weiß, G.: Learning to coordinate actions in multi-agent systems. In: Proceedings of the 13<sup>th</sup> International Joint Conference on Artificial Intelligence, Vol. 1, S. 311-316, Morgan Kaufmann Publication, 1993
- Weiss (1999) Weiss, G. (Hrsg.): Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge et al., MIT Press, 1999
- Weiss (1999a) Weiss, G.; Sen, S.: Learning in Multiagent Systems. In: Weiss (1999), S. 259-298
- Weiß (1999b) Weiß, G.; Dillenbourg, P.: What is „multi“ in multiagent learning? In: Dillenbourg, P. (Hrsg.): Collaborative learning. Cognitive and computational approaches. Pergamon Press, 1999, S. 64-80
- Weiß (2001) Weiß, G.: Agentenorientiertes Software Engineering. Rubrik "Das aktuelle Schlagwort". Informatik Spektrum, Band 24, Heft 2, S. 98-101
- Weiß (2005) Weiß, G.; Jakob, R.: Agentenorientierte Softwareentwicklung. Methoden und Tools. Berlin u. a., Springer-Verlag, 2005
- Wellmann (1995) Wellman, M.: A Computational Market Model for Distributed Configuration Design. In: Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM), Vol. 9 (1995)
- Wille (2001) Wille, C.; Dumke R.; Stojanov, S.: Performance Engineering in Agent-Based Systems: Concepts, Modelling and Examples. In: Current Trends in Software Measurement. Proceedings of the 11<sup>th</sup> International Workshop on Software Measurement, Montreal/Canada 2001, Shaker Verlag, 2001, Seite 153-180
- Wille (2002) Wille, C.; Dumke, R.: Quality Assurance in Agent-Based Systems. Current State and Open Problems. Preprint Nr. 4, Otto-von-Guericke-Universität Magdeburg, 2002
- Wille (2002a) Wille, C.; Dumke, R.; Stojanov, S.: New Measurement Intentions in Agent-based Systems Development and Application. In: Dumke et al. (Hrsg.): Software Measurement and Estimation. Proceedings of the 12<sup>th</sup> International Workshop on Software Measurement, Magdeburg, Shaker-Verlag, 2002, S. 203-227
- Wille (2002b) Wille, C.; Dumke, R.; Stojanov, S.: Softwaremessung und -Bewertung für agentenbasierte Systementwicklung und Anwendung. In: Dumke/Rombach (Hrsg.): Software-Messung und Bewertung. Wiesbaden, Deutscher Universitätsverlag, 2002, S. 219-253,
- Wille (2004) Wille, C.; Brehmer, N.; Dumke, R.: Software Measurement of Agent-Based Systems. An Evaluation Study of the Agent Academy. Preprint Nr. 3, Otto-von-Guericke-Universität Magdeburg, 2004

- Williams (2003) Williams, A. B.; Ren, Z.: A Survey of Multiagent Learning. <http://www.engineering.uiowa.edu/~zren/research.html>, 13. Dezember 2004
- Williams (2004) Williams, A: Learning to Share Meaning in a Multi-Agent System. In: Autonomous Agents and Multi-Agent Systems, Vol. 8 (2004), S. 165– 193
- Wood (2001) Wood, M. F.; DeLoach, S. A.: An Overview of the Multiagent Systems Engineering Methodology. In: Ciancarini (2001), S. 1-28
- Wooldridge (1995) Wooldridge, M. J.; Jennings, N. R.: Intelligent Agents. Proceedings of the ECAI'94 Workshop on Agent Theories, Architectures, and Languages, Vol. 890, Berlin et al., Springer-Verlag, 1995
- Wooldridge (1995a) Wooldridge, M. J.; Jennings, N. R.: Agent Theories, Architectures, and Languages: A Survey. In: (Wooldridge (1995)), S. 1-39
- Wooldridge (1999) Wooldridge, M. J.: Intelligent Agents. In: (Weiss (1999)), S. 27-77
- Wooldridge (2000) Wooldridge, M. J.; Jennings, N. R.; Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. In: Autonomous Agents and Multi-Agents Systems, Vol. 3, Issue 3 (September 2000), S. 285-312
- Wooldridge (2001) Wooldridge, M. J.; Ciancarini, P.: Agent-Oriented Software Engineering: The State of the Art. In: Ciancarini (2001), S. 1-28

## **Abschließende Erklärung**

Ich versichere hiermit, dass ich die vorliegende Diplomarbeit selbständig, ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Magdeburg, den 21. Dezember 2004

Thomas Wesche