Agenten und Multiagentensysteme

Complex Adaptive Systems

Prof. Dr. Michael Köhler-Bußmeier

HAW Hamburg, Department Informatik

Version vom 9. September 2016

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Gliederung

Multiagentensysteme (MAS)

Sicht der **verteilten künstlichen Intelligenz** (VKI)
(engl. distributed artificial intelligence, DAI) auf MAS:

Nach [Wei99]

"DAI is the study, construction, and application of multiagent systems, that is, systems in which several interacting, intelligent agents pursue some set of goals or perform some tasks."

Zentrale Aspekte von Agentensystemen:

1. Erstens handelt es sich bei den Systemkomponenten um Agenten, d.h. um lose gekoppelte Einheiten, die im Rahmen ihrer Möglichkeiten autonom handeln.
2. Zweitens sind die Systemeinheiten intelligent, worunter man grob verstehen kann, dass Agenten in der Lage sind, auch in verschiedenen Umgebungen flexibel zu agieren, dass sie in der Lage sind, rational zu planen, und dass sie die Fähigkeit besitzen zu lernen.
3. Drittens spielt die Interaktion der Agenten eine zentrale Rolle, da nicht nur die Einheiten verteilt sind, sondern auch Fähigkeiten, Daten und Ressourcen.

Gliederung

Konzeptionelle Abgrenzung der VKI zur KI

- KI stellt mit dem **intelligenten Problemlösen** eines einzelnen situierten Systems die kognitive Aspekte in den Vordergrund.
- Für die VKI besteht der Kontext eines Agenten nun wiederum aus Agenten. Damit steht hier der Interaktionszusammenhang zwischen den Agenten im Vordergrund.
- Intelligenz des Ganzen ergibt sich aus dem Zusammenspiel der Teile.
- Neuausrichtung in Hinblick auf die Nachbardisziplinen: Während die KI der Psychologie und der Neurologie nahesteht, orientiert sich die VKI in Richtung der Sozial- und Wirtschaftswissenschaften.

Agent vs. Objekt

- Diese Eigenschaften grenzen Agenten von Objekten ab:
  1. Objekte sind weder autonom
  2. noch lernend
  3. oder zielverfolgend.
- Die engste Verbidung finden Agenten noch zu den aktiven Objekten [Nie93], bei denen jedem Objekt ein eigener Kontrollfluss (engl. thread) zugeordnet wird.
- Agenten sind ebenso spezifischer als Expertensysteme.

Gliederung

Interaktionszusammenhänge

Interaktionszusammenhänge sind sie zweiseitig ausgerichtet:

- Ausrichtung auf die Mikro-Ebene: Agenten, der in die Interaktionszusammenhänge eingebettet ist und dessen Handeln somit kontexttualisiert wird.
- Makro-Zusammenhänge auf Ebene des Gesamtsystems: Es sind nicht die Handlungen des einzelnen relevant, sondern vielmehr der Prozess mit seinen Mustern insgesamt.
- Diese Mikro-Makro-Dualität wird uns in den weiteren Ausführungen begleiten.

Die Möglichkeit – und meist auch der Zwang – zur Interaktivität der Agenten bildet das Leitmotiv für die VKI-Forschung:
„Wer kommuniziert wann mit wem worüber?“

Koordination als verteilter Algorithmus

- Koordination durch Protokolle: verteilte Algorithmen
- Typischerweise ist aus der Protokollsicht jeder Agent des Systems ein potentieller Interaktionspartner.
- Offensichtlich steigt mit der Anzahl der Agenten die Komplexität, geeignete Interaktionspartner zu finden, die Ziele mit ihnen abzustimmen und die Pläne konfliktfrei zu gestalten.
- Ziel muss aber $O(n)$ sein, da man dann die Reaktionszeiten auch bei steigenden Agentenzahlen konstant halten kann, indem man die Zahl der Rechner im MAS-Netzwerk entsprechend aufstockt.
- Typische verteilte Algorithmen weisen Kommunikationskomplexitäten von $O(n \log n)$ oder $O(n^2)$ auf.
- Dieser Zuwachs an Koordinierungsaufwand ist als **Skalierungsproblematik** der VKI bekannt.
- Benötigt: Strukturen, die den Koordinationsprozess regeln.

Die Skalierungsproblematik

- Die Skalierungsproblematik geht somit über die Ebene einzelner Agenten hinaus.
  Sie lässt sich nur auf der Systemebene behandeln.
- Man muss sich mit der Formation von Teams und der Organisationsstruktur von MAS beschäftigen.
- Postulat: Die Skalierungsproblematik ist dann zu lösen, wenn die Organisationsstruktur explizit repräsentierter Bestandteil des Systems ist und das Systems aktiv auf diese Einfluss nehmen kann.
- "Thus, the current state of the art is challenged by MASs that are larger or where the magnitude or speed of agent population variability confounds one overall design.
  To tackle both these problems we hypothesize that MASs should be **self-building** (able to determine the most appropriate organizational structure for the system by themselves at run-time) and **adaptive** (able to change this structure as their environment changes)." [TJ01]
- **Reflexivität** des MAS: Die Organisation wird zur Variablen erklärt, die durch das Handeln der Akteure geformt wird.
- Stichwort: **lernende Organisation**

Gliederung

Agenten als soziale Akteure

Definition

Agenteneigenschaften:

1. Agenten sind in einer Umgebung situierte Einheiten.
2. Jeder Agent ist autonom, d.h. im Rahmen der operationalen Randbedingungen (interne wie externe) ist er in seiner Handlungswahl frei. Restriktionen sind nur von ihm selbst auferlegte.
3. Jeder Agent ist in dem Sinne intelligent, als dass er zielverfolgend ist und dass hiermit eine gewisse Flexibilität bezüglich der Zielrealisierung verbunden ist, so dass ein Agent auch auf sich verändernde Randbedingungen reagieren kann.
4. Agent der VKI: Die Umgebung besteht aus anderen Agenten.
5. Aus der Tatsache, dass ein Agent in der Lage ist, auf die Umwelt einzuwirken, folgt die **Interaktivität**.

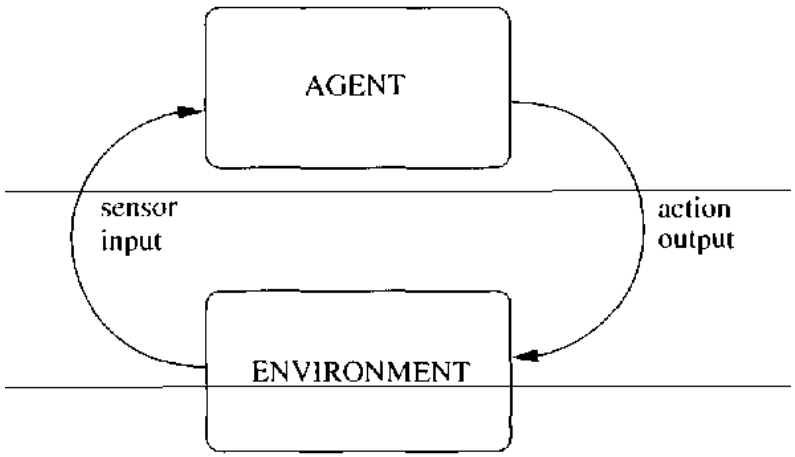
Gliederung

Gliederung

Situated agents: environment

Wooldridge and Jennings (1995).

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



In most domains of reasonable complexity, an agent will not have complete control over its environment.
From the point of view of the agent, this means that the same action performed twice in apparently identical circumstances might appear to have entirely different effects, and in particular, it may fail to have the desired effect.

Functional vs. reactive systems

- Originally, software engineering concerned itself with what are known as 'functional' systems.
  A functional system is one that simply takes some input, performs some computation over this input, and eventually produces some output.
- Unfortunately, many computer systems that we desire to build are not functional in this sense.
- Typically, the main role of reactive systems is to maintain an interaction with their environment, and therefore must be described (and specified) in terms of their on-going behaviour. (Pnueli, 1986)
- Reactive systems are harder to engineer than functional ones.
  Perhaps the most important reason for this is that an agent engaging in a (conceptually) non-terminating relationship with its environment must continually make local decisions that have global consequences.

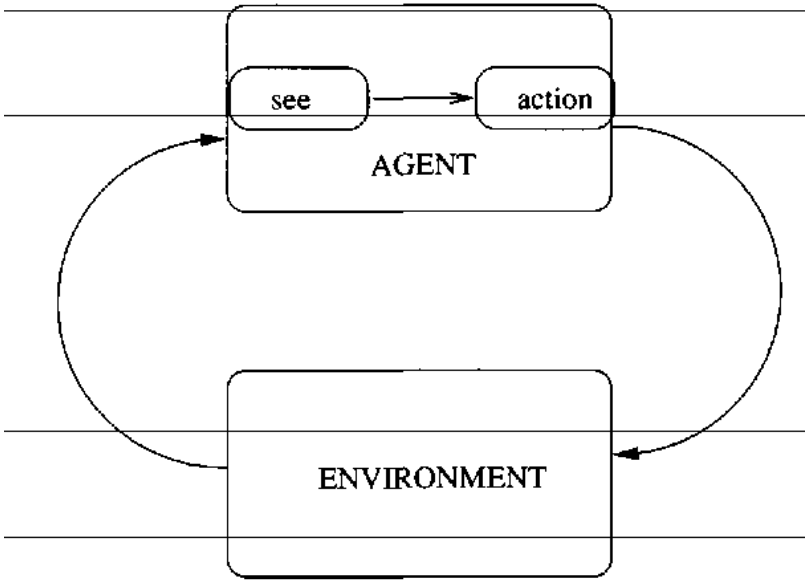
Intelligent Agents

Definition (Wooldridge and Jennings (1995))

Capabilities of intelligent agents:

- Reactivity. Intelligent agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their design objectives.
- Proactiveness. Intelligent agents are able to exhibit goal-directed behaviour by taking the initiative in order to satisfy their design objectives.
- Social ability. Intelligent agents are capable of interacting with other agents (and possibly humans) in order to satisfy their design objectives.

Perception - Action

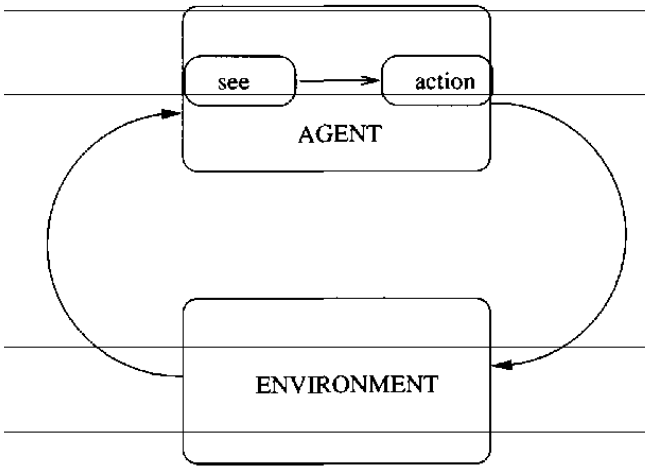


The function *see* captures the agent's ability to observe its environment, whereas the *action* function represents the agent's decision-making process.

The see function might be implemented in hardware in the case of an agent situated in the physical world: for example, it might be a video camera or an infrared sensor on a mobile robot.

For a software agent, the sensors might be system commands that obtain information about the software environment.

Perception - Action



The output of the see function is a percept - a perceptual input. Let Per be a (non-empty) set of percepts.

Then see is a function

$$see : E \rightarrow Per$$

which maps environment states to percepts, and action is a function

$$action : Per* \rightarrow Ac$$

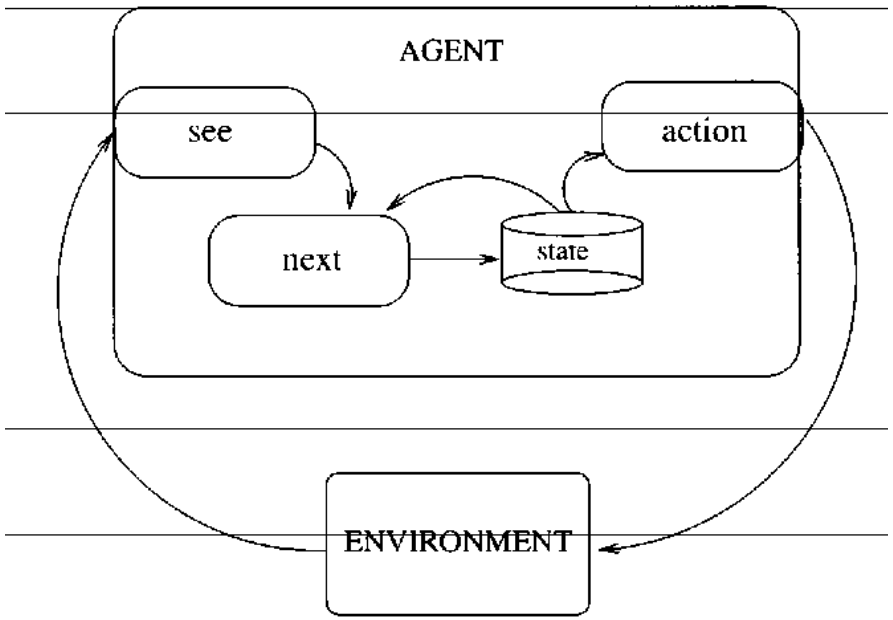
which maps sequences of percepts to actions.

An agent Ag is now considered to be a pair $Ag = (see, action)$, consisting of a see function and an $action$ function.

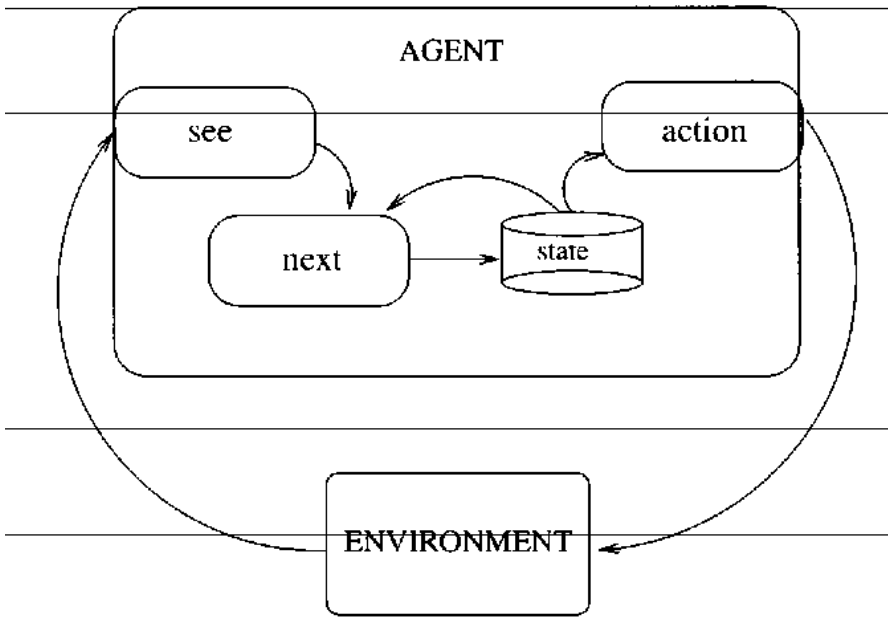
Agents with state

We have so far modelled an agent's decision function as from **sequences** of environment states or percepts to actions.

Agents have some internal data structure, which is typically used to record information about the environment state and history.



Agents with state



The action-selection function *action* is defined as a mapping

$$action : I \rightarrow Ac$$

from internal states to actions.

An additional function *next* is introduced, which maps an internal state and percept to an internal state:

$$next : I \times Per \rightarrow I.$$

Gliederung

Agentenarchitekturen

Aufbau eines Agenten

- Da Zielverfolgung und Planung ein zentraler Gegenstand der KI ist, entstammen viele Strukturierungsansätze aus dem Bereich der formalen Logik.
- Hier werden Modallogiken zur Wissensrepräsentation [LL00] oder
- zur Spezifikation der temporalen Aktivitäten [WJ95, Woo00] verwendet.
- praktische Systeme: METAMEM [BFG+90], Congolog [GLL00]

Ein weit verbreiteter Ansatz ist das Design von Agenten nach dem BDI-Paradigma [Bra87, RG91]:

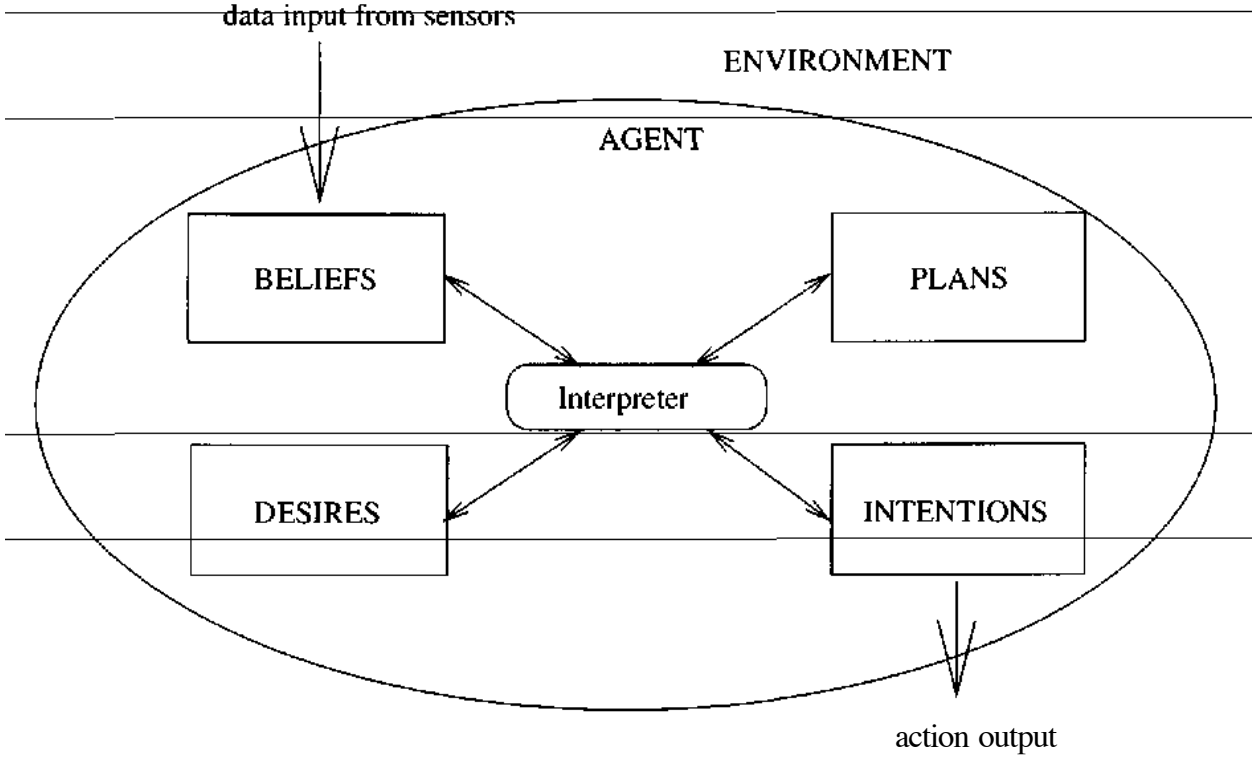
- Wissen (engl. beliefs),
- Wünsche (engl. desires) sowie
- Absichten (engl. Intentions) eines Agenten

Eine Abwendung vom logischen Paradigma stellt der reaktive Ansatz dar (beliebt bei Echtzeitsystemen).

The Procedural Reasoning System

The Procedural Reasoning System (PRS), originally developed at Stanford Research Institute by Michael Georgeff and Amy Lansky, was perhaps the first agent architecture to explicitly embody the BDI paradigm

In the PRS, an agent does no planning from first principles. Instead, it is equipped with a library of pre-compiled plans.



PRS: Blocks World in JAM

```
GOALS:
  ACHIEVE blocks_stacked;
FACTS:
  FACT ON "Block5" "Block4";      FACT ON "Block4" "Block3";
  FACT ON "Block1" "Block2";      FACT ON "Block2" "Table";
  FACT ON "Block3" "Table";       FACT CLEAR "Block1";
  FACT CLEAR "Block5";            FACT CLEAR "Table";
Plan: {
  NAME: "Top-level plan"
  GOAL: ACHIEVE blocks_stacked;
  CONTEXT:
  BODY:     ACHIEVE ON "Block3" "Table";
            ACHIEVE ON "Block2" "Block3";
            ACHIEVE ON "Block1" "Block2";
}
Plan: {
  NAME: "Stack blocks that are already clear"
  GOAL: ACHIEVE ON $OBJ1 $OBJ2;
  CONTEXT:
  BODY:     ACHIEVE CLEAR $OBJ1;
            ACHIEVE CLEAR $OBJ2;
            PERFORM move $OBJ1 $OBJ2;
  UTILITY: 10;
  FAILURE: EXECUTE print "\n\nStack blocks failed!\n\n";
}
Plan: {
  NAME: "Clear a block"
  GOAL: ACHIEVE CLEAR $OBJ;
  CONTEXT: FACT ON $OBJ2 $OBJ;
  BODY:     ACHIEVE ON $OBJ2 "Table";
  EFFECTS: RETRACT ON $OBJ2 $OBJ;
  FAILURE: EXECUTE print "\n\nClearing block failed!\n\n";
}
```

Logik-Architekturen

- Modalitäten sind Operatoren der Logik, die sich auf den Wahrheitswert von Formeln beziehen.
- Dies ist in der Prädikatenlogik nicht möglich, da man in ihr keine Prädikate über Formeln ausdrücken kann.
- Der folgende Ausruck ist demnach ungeeignet, um auszudrücken, dass Alice glaubt, dass Tweety ein Vogel sei:

$$\text{believes}(\mathit{alice}, \text{is-a-bird}(\mathit{tweety}))$$

- Das Problem besteht darin, dass die Argumente des Prädikats $\mathit{believes}$ Terme sein müssen, $\text{is-a-bird}(\mathit{tweety})$ jedoch eine Formel ist.

Modalitäten der Modal-Logik

- Die klassischen Modalitäten sind die der Notwendigkeit und die der Möglichkeit [HC84].
- Eine Formel ϕ ist **notwendigerweise** wahr, wenn man sich keine Welt vorstellen kann, in der ϕ nicht wahr wäre: $\Box\,\phi$
- ϕ ist **möglicherweise** wahr, wenn Welten vorstellbar sind, in denen ϕ wahr ist, aber auch welche, in denen das nicht gilt: $\Diamond\,\phi$
- Erweitert man die Aussagenlogik um diese beiden Modalitäten, so erhält man die propositionale Modallogik.
- Um die Semantik der Modalitäten exakt festzuhalten, fordert man axiomatisch die Gültigkeit gewisser Formeln:

Name	Axiom	Bedingung
K	$\Box\,(\phi \Rightarrow \psi) \Rightarrow (\Box\,\phi \Rightarrow \Box\,\psi)$	
N	$\phi \Rightarrow \Box\,\phi$	
T	$\Box\,\phi \Rightarrow \phi$	reflexiv
D	$\Box\,\phi \Rightarrow \Diamond\,\phi$	seriell
4	$\Box\,\phi \Rightarrow \Box\,\Box\,\phi$	transitiv
5	$\Diamond\,\phi \Rightarrow \Box\,\Diamond\,\phi$	Euklid

Modalität des Wissens

- Notwendigkeit ist nicht die einzige Interpretation von $\Box$.
- Interpretiert man die Modalität $\Box\,\phi$ als „ϕ wird gewusst“, so kann man hiermit eine Logik des Glaubens/Wissens formalisieren.
- Dies hat zur Konsequenz, dass aus dem Notwendigkeitsaxioms $\phi \Rightarrow \Box\,\phi$ folgt, dass **alle** Tautologien gewusst werden.
- Außerdem ist Wissen unter Implikation abgeschlossen, d.h. alle gültige Folgerungen $\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow \phi$ werden gewusst.
- Gilt das D-Axiom ($\Box\,\phi \Rightarrow \Diamond\,\phi$ und $\Diamond\,\phi \equiv \neg\Box\,\neg\phi$), so ist Wissen widerspruchsfrei.
- Gilt das T-Axiom ($\Box\,\phi \Rightarrow \phi$), so können nur Wahrheiten gewusst werden.
- Gilt das 4-Axiom ($\Box\,\phi \Rightarrow \Box\,\Box\,\phi$), so besteht Wissen über das Gewusste.
- Gilt das 5-Axiom ($\neg\Box\,\neg\phi \Rightarrow \Box\,(\neg\Box\,\neg\phi)$), so besteht Wissen über das Ungewusste.

BDI-Logik: Überzeugungen (Beliefs)

In der **BDI-Logik** werden die Überzeugungen (**beliefs**), die Bedürfnisse (engl. **desires**) und die Absichten eines Agenten (engl. **intentions**) eines Agenten durch Modalitäten modelliert [PJN02].

- Die Tatsache, dass der Agent a zum Zeitpunkt t der Überzeugung ist, dass ϕ gilt, wird als $\mathbf{Bel}(a, \phi)(t)$ dargestellt. Es gelten die Axiome KD45:

$$\begin{aligned}
\mathbf{Bel}(a, \phi)(t) \wedge \mathbf{Bel}(a, (\phi \Rightarrow \psi))(t) &\Rightarrow \mathbf{Bel}(a, \psi)(t) \\
\mathbf{Bel}(a, \phi)(t) &\Rightarrow \neg\mathbf{Bel}(a, \neg\phi)(t) \\
\mathbf{Bel}(a, \phi)(t) &\Rightarrow \mathbf{Bel}(a, \mathbf{Bel}(a, \phi))(t) \\
\neg\mathbf{Bel}(a, \phi)(t) &\Rightarrow \mathbf{Bel}(a, \neg\mathbf{Bel}(a, \phi))(t)
\end{aligned}$$

BDI-Logik: Bedürfnisse und Ziele (desires and goals)

- Die Bedürfnisse und Ziele eines Agenten drücken die Zustände der Welt aus, die er gern erreichen möchte.
- Die Tatsache, dass der Agent a hat zum Zeitpunkt t das Bedürfnis hat, ϕ zu erreichen, wird als $\mathbf{Des}(a, \phi)(t)$ dargestellt,
- Ziele als $\mathbf{Goal}(a, \phi)(t)$.
- Der Unterschied zwischen Zielen und Bedürfnisse liegen darin, dass Bedürfnisse inkonsistent und sogar unerfüllbar sein können,
- während Ziele als Menge **konsistenter und realisierbarer** Zustände definiert sind, von denen man annehmen kann, dass der Agent sie erreichen kann.
- Agenten haben keine den Überzeugungen widersprechenden Ziele.

BDI-Logik: Absichten (intentions)

- Die Absichten eines Agenten sind immer auf einen bestimmten erstrebenswerten Zustandes der Welt ausgerichtet.
- Besitzt ein Agent die individuelle Absicht, einen Zustand zu erreichen, so ist er sich selbst gegenüber verpflichtet (**self-commitment**), entsprechend zu handeln.
- Die Absicht des Agenten a, zum Zeitpunkt t einen Zustand, in dem ϕ gilt, anzustreben, wird $\mathbf{Int}(a, \phi)(t)$ notiert.
- Ein Agent besitzen keine Absichten, die seinen Überzeugungen widersprechen:

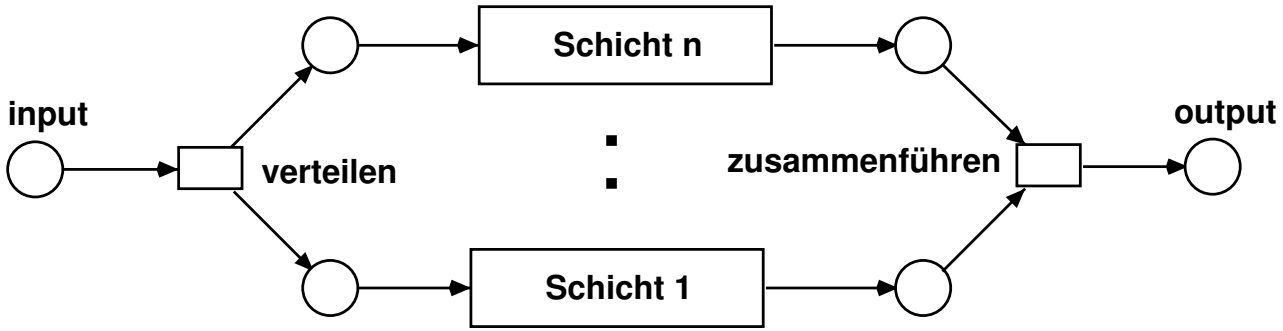
$$\mathbf{Int}(a, \phi)(t) \Rightarrow \neg \mathbf{Bel}(a, \neg\phi)(t)$$

Reaktive Architekturen

- Abkehr von planenden Ansätzen
- Agenten bauen kein Weltmodell auf, speichern keine – oder nur geringe – Zustandsinformation und treffen ihre Wahl anhand einer Liste einfacher Wenn-Dann Regeln.
- Vorteile: geringe Darstellungsgröße und kurze Reaktionszeiten.
- Anwendungsbereich: Robotik [Bro90].
- Schlagwort: Schwarmintelligenz [KE01]
- Tierpopulationen (wie Ameisenkolonien, Fisch- oder Vogelschwärme) in der Lage sind, global kohärentes (um nicht zu sagen: intelligentes) Verhalten aufweisen
- Das einzelne Individuum ist dazu kognitiv aber gar nicht in der Lage.
- So kann eine einzelne Ameise keine Wegewahloptimierung bei der Futtersuche betreiben, die Kolonie als ganzes schon.
- Neu: Software defined Networks

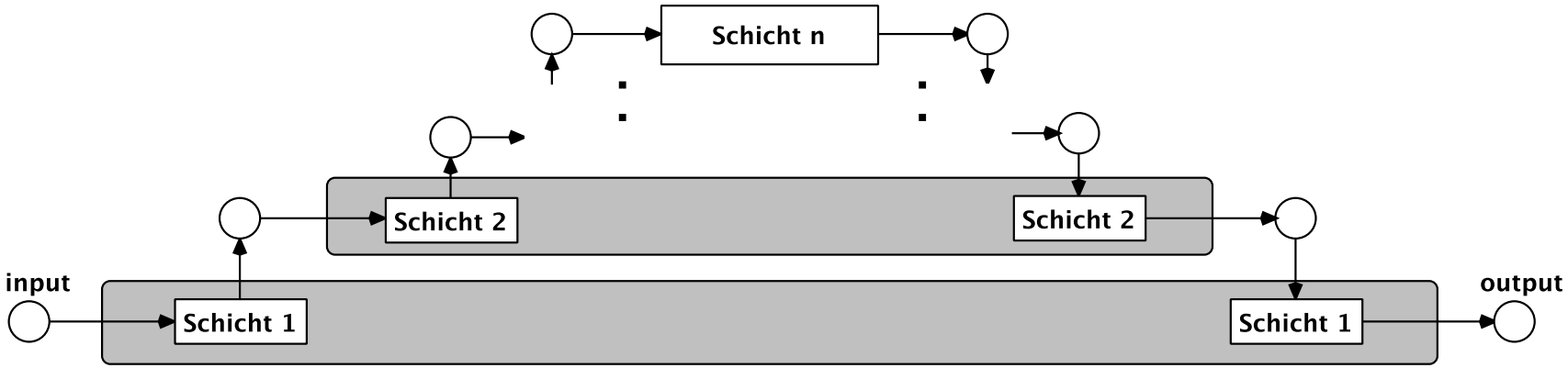
Gliederung

Horizontale Agentenarchitektur



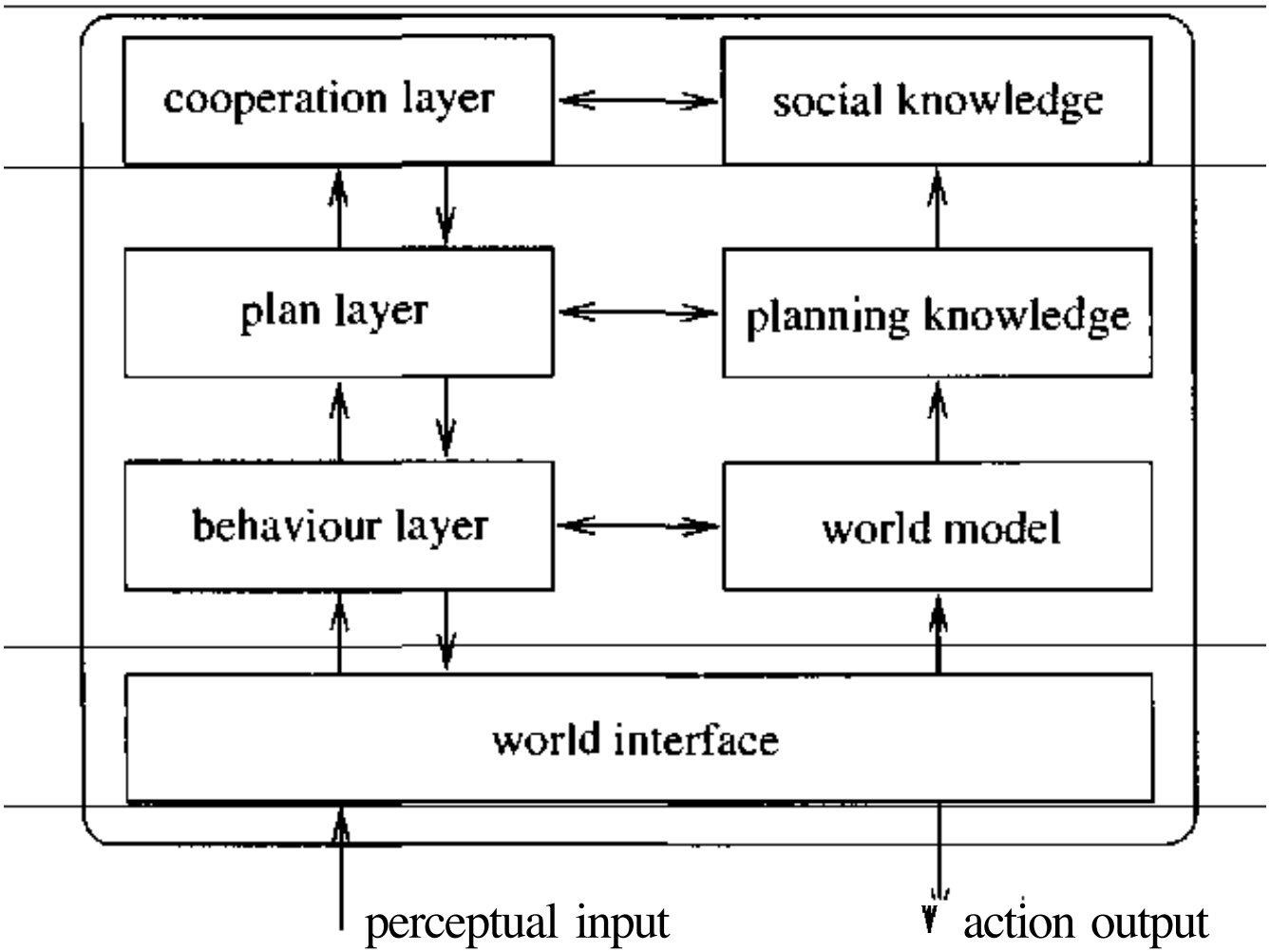
- Weitere typische Ansätze stellen die Schichtenarchitekturen dar. Hierbei unterscheidet man zwischen sogenannten horizontalen und vertikalen Schichtungen.
- Bei der horizontalen Schichtung erhalten alle Planungsschichten die gleiche Eingabe.
- Ihre Ergebnisse werden dann zu einer Gesamtentscheidung zusammengesetzt.
- Beispiel für eine horizontale Schichtung: Touring-Machines [Fer92].

Vertikale Agentenarchitektur



- Vertikale Architekturen bearbeiten die Eingabe und geben ihre Ergebnisse an die nächste Ebene weiter.
- Man unterscheidet Systeme, bei denen die Bearbeitung in der letzten Schicht zu Ende ist (one-pass),
- von solchen, bei denen die letzte Schicht ihre Ergebnisse wieder zurück reicht, so dass sie die gesamte Schichtung in rückwärtiger Richtung durchläuft (two-pass).
- Beispiel für eine vertikale Schichtung: INTERARP [MP94]

InteRRaP - a vertically layered two-pass agent architecture



Gliederung

Agentenorientierte Sprachen

Daneben existieren als Erweiterung der objektorientierten Programmiersprachen auch agentenorientierte Sprachen wie:

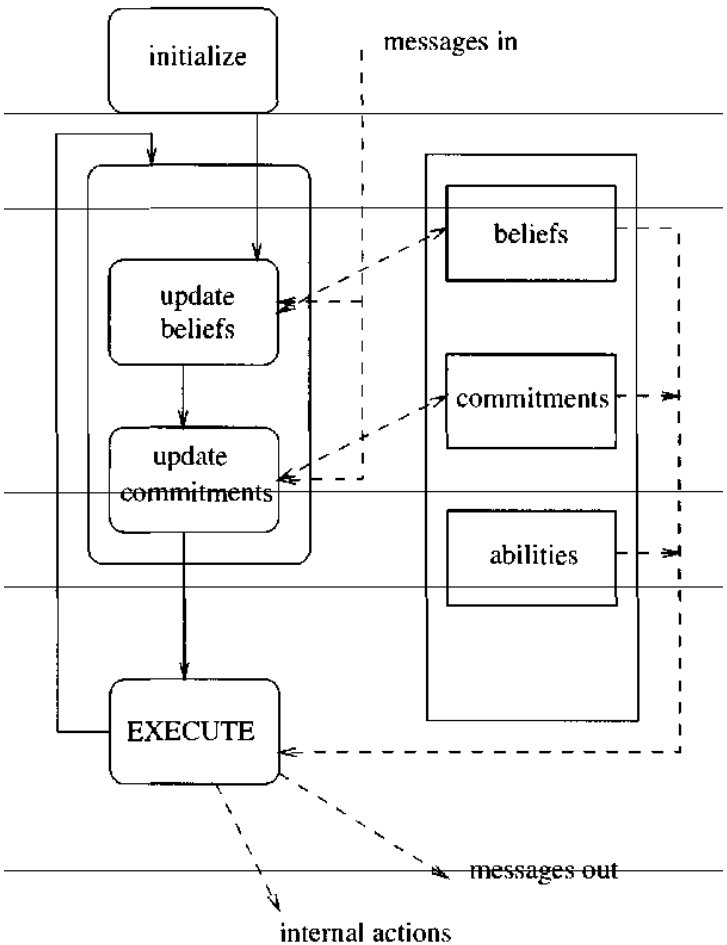
- Agent-0 [Sho90]
- Agentspeak [Rao96]
- JASON[BHW07]
- JADE [BPR01]
- ....

Diese unterstützen die Darstellung von Wissen, sowie eine mit Prolog verwandte, Regel-basierte Programmierung (WENN-DANN)

Gliederung

The flow of control in Agent0

Agent0 [Shoham 93]: programming agents in terms of **mentalistic notions** (such as belief, desire, and intention).



The operation of an agent can be described by the following loop:

- (1) Read all current messages, updating beliefs - and hence commitments - where necessary.
- (2) Execute all commitments for the current cycle where the capability condition of the associated action is satisfied.
- (3) Goto (1).

A Commit Rule in Agent0

```
COMMIT(
  ( agent, REQUEST, DO(time, action)
  ), ;;; msg condition
  ( B,
    [now, Friend agent] AND
    CAN(self, action) AND
    NOT [time, CMT(self, anyaction)]
  ), ;;; mental condition
  self,
  DO(time, action) )
```

This rule may be paraphrased as follows:

If I receive a message from *agent* which *requests* me to *do action* at *time*, and I *believe* that

- agent is currently a friend;
- I can do the action;
- at time, I am not committed to doing any other action,

then *commit* to *doing action* at *time*.

Gliederung

Jason

Jason is the interpreter for an extended version of AgentSpeak.

AgentSpeak(L) is a logic-based agent-oriented programming language, which is aimed at the implementation of reactive planning systems.

Some of the features available in Jason are:

- speech-act based inter-agent communication (and annotation of beliefs with information sources);
- annotations on plan labels, which can be used by elaborate (e.g., decision theoretic) selection functions;
- the possibility to run a multi-agent system distributed over a network (using SACI, but other middleware can be used);
- fully customisable (in Java) selection functions, trust functions, and overall agent architecture (perception, belief-revision, inter-agent communication, and acting);
- straightforward extensibility (and use of) by means of user-defined "internal actions";
- clear notion of multi-agent environments, which can be implemented in Java (this can be a simulation of a real environment, e.g., for testing purposes before the system is actually deployed).

Examples of AgentSpeak Plans for a Bomb-Disarming Robot

```
skill(plasticBomb).
skill(bioBomb).
~skill(nuclearBomb).

safetyArea(field1).

@p1
+bomb(Terminal, Gate, BombType) :  skill(BombType)
   <-  !go(Terminal, Gate);
       disarm(BombType).

@p2
+bomb(Terminal, Gate, BombType) :  ~skill(BombType)
   <-  !moveSafeArea(Terminal, Gate, BombType).

@p3
+bomb(Terminal, Gate, BombType) :  not skill(BombType) &
                                   not ~skill(BombType)
   <-  .broadcast(tell, alter).

@p4
+!moveSafeArea(T,G,Bomb) :  true
   <-  ?safeArea(Place);
       !discoverFreeCPH(FreeCPH);
       .send(FreeCPH, achieve,
             carryToSafePlace(T,G,Place,Bomb)).
:
```

Gliederung

Concurrent MetateM

The Concurrent MetateM language (developed by Michael Fisher) is based on the direct execution of logical formulae.

A Concurrent MetateM system contains a number of concurrently executing agents, each of which is able to communicate with its peers via asynchronous broadcast message passing.

Each agent is programmed by giving it a temporal logic specification of the behaviour that it is intended the agent should exhibit.

Concurrent MetateM: Logic

Propositional MetateM Logic (PML):

Operator	Meaning
$\bigcirc \varphi$	φ is true 'tomorrow'
$\bullet \varphi$	φ was true 'yesterday'
$\Diamond \varphi$	at some time in the future, φ
$\Box \varphi$	always in the future, φ
~~$\blacklozenge \varphi$~~	~~at some time in the past, φ~~
$\blacksquare \varphi$	always in the past, φ
$\varphi \mathcal{U} \psi$	φ will be true until ψ
$\varphi S \psi$	φ has been true since ψ
$\varphi \mathcal{W} \psi$	φ is true unless ψ
$\varphi \mathcal{Z} \psi$	φ is true zince ψ

$$\Diamond important(Janine)$$

means 'sometime in the future, Janine will be important'.

$$(\neg\ friends(us))\ \mathcal{U} apologize(you)$$

means 'we are not friends until you apologize'. And, finally,

$$\bigcirc apologize(you)$$

Three agents: rp, rc1, and rc2.

$$rp(ask1, ask2)[give1, give2]:$$
$$\bigcirc ask1 \Rightarrow \Diamond give1;$$
$$\bigcirc ask2 \Rightarrow \Diamond give2;$$
$$\text{start} \Rightarrow \Box\neg (give1 \wedge give2).$$

$$rc1(give1)[ask1]:$$
$$\text{start} \Rightarrow ask1;$$
$$\bigcirc ask1 \Rightarrow ask1.$$

$$rc2(ask1, give2)[ask2]:$$
$$\bigcirc (ask1 \wedge \neg ask2) \Rightarrow ask2.$$

- The agent rp is a 'resource producer: it ran 'give' to only one agent at a time, and will commit to eventually give to any agent that asks.
  Agent rp will only accept messages ask1 and ask2, and can only send give1 and give2 messages.
- The interface of agent rc1 states that it will only accept give1 messages, and can only send ask1 messages.
  The rules for agent rc1 ensure that an ask1 message is sent on every cycle – this is because start is satisfied at the beginning of time.
  This triggers the second rule in the next round, and so on.
- The interface for agent rc2 states that it will accept both ask1 and give2 messages, and can send ask2 messages.
  The single rule for agent rc2 ensures that an ask2 message is sent on every cycle where, on its previous cycle, it did not send an ask2 message, but received an ask1 message (from agent rc1).

Literatur I

H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens.

Metatem: a framework for programming in temporal logic.
In *REX workshop: Proceedings on Stepwise refinement of distributed systems: models, formalisms, correctness*, pages 94–129, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge.

Programming Multi-Agent Systems in AgentSpeak using Jason.
Wiley, 2007.

Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa.

Developing multi-agent systems with JADE.
In Cristiano Castelfranchi and Yves Lespérance, editors, *Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL-2000)*, volume 1986 of *Lecture Notes in Computer Science*, pages 89–103. Springer-Verlag, 2001.

M. E. Bratman.

Intentions, Plans, and Practical Reason.
Harvard University Press, Cambridge, 1987.

Rodney A. Brooks.

A robust layered control system for a mobile robot.
pages 2–27, 1990.

N. Carriero and D. Gelernter.

Linda in context.
Communications of the ACM, 32:444–458, 1989.

Literatur II


Edmund H. Durfee and Victor R. Lesser.
Partial global planning: A coordination framework for distributed hypothesis formation.
IEEE Transactions on Systems, Man, and Cybernetics, 21(5):1167–1183, September 1991.


I. A. Ferguson.
Towards an architecture for adaptive, rational, mobile agents.
In *Decentralized Artificial Intelligence*, volume 3, pages 249–262. Elsevier Science, 1992.


FIPA.
FIPA 97 Specification, Part 2 - Agent Communication Language.
Technical report, `http://www.fipa.org`, Oktober 1998.


Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque.
Congolog, a concurrent programming language based on the situation calculus.
Artificial Intelligence, 121(1-2):109–169, 2000.


George E. Hughes and Maxwell J. Cresswell.
A companion to modal logic.
Methuen, 1984.


James Kennedy and Russell C. Eberhart.
Swarm intelligence.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.


The DARPA Knowledge Sharing Initiative External Interfaces Working Group.
Specification of the KQML Agent-Communication Language.
Draft, DARPA, Juni 1993.

Literatur III

Hector Levesque and Gerhard Lakemeyer.

The logic of knowledge bases.
MIT Press, Cambridge, Massachusetts, 2000.

Jörg P. Müller and Markus Pischel.

An architecture for dynamically interacting agents.
Journal of Intelligent and Cooperative Information Systems, 3(1):25–45, 1994.

Oscar Nierstrasz.

Composing active objects.
In *Research directions in concurrent object-oriented programming*, chapter 5. MIT Press, 1993.

Uche Ogbuji and Roxane Ouellet.

DAML reference.
online: http://www.xml.com/pub/a/2002/05/01/damlref.html, 2002.

Pietro Panzarasa, Nicholas R. Jennings, and Timothy J. Norman.

Formalizing collaborative decision-making and practical reasoning in multi-agent systems.
Journal of Logic and Computation, 12(1):55–117, 2002.

Anand S. Rao.

AgentSpeak(L): BDI agents speak out in a logical computable language.
In Rudy van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, 1996.

Literatur IV

Anand S. Rao and Michael P. Georgeff.

Modeling rational agents within a BDI-architecture.
In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Morgan Kaufmann, 1991.

John R. Searle.

Speech acts: An essay in the philosophy of Language.
Cambridge University Press, 1970.

Yoav Shoham.

Agent-oriented programming.
Technical report, Stanford, Calif.: Department of Computer Science, Stanford University, 1990.

Rudi Studer, Andreas Hotho, Gerd Stumme, and Raphael Volz.

Semantic web – state of the art and future directions.
Künstliche Intelligenz, 3:5–9, 2003.

Reid G. Smith.

The contract net: A formalism for the control of distributed problem solving.
In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, 1977.

Michael K. Smith, Chris Welty, and Deborah L. McGuinness.

Owl web ontology language guide. w3c recommendation.
http://www.w3.org/TR/owl-guide/, 2004.

Literatur V

Phillip J. Turner and Nicholas R. Jennings.

Improving the scalability of multi-agent systems.
In *Proceedings of the First International Workshop on Infrastructure for Scalable Multi-Agent Systems*, volume 1887 of *Lecture Notes in Computer Science*, page 246ff. Springer-Verlag, 2001.

Gerhard Weiß, editor.

Multiagent systems: A modern approach to Distributed Artificial Intelligence.
MIT Press, 1999.

Michael J. Wooldridge and Nicholas R. Jennings.

Agent theories, architectures, and languages: a survey.
In M. J. Wooldridge and N. R. Jennings, editors, *Intelligent Agents*, pages 1–22. Springer-Verlag, 1995.

Michael Wooldridge.

Reasoning about Rational Agents.
Intelligent robotics and autonomous agents. MIT Press, Cambridge, Massachussetts/London, 2000.