Artificial Agents and Logic Programming

Gerd Wagner gw@inf.fu-berlin.de

Inst.f.Informatik, Universität Leipzig,
Augustusplatz 10-11, 04109 Leipzig, Germany.
http://www.informatik.uni-leipzig.de/~gwagner

Abstract. Artificial agents represent a new paradigm in software engineering and Artificial Intelligence. As complex software-controlled systems they are capable of flexible autonomous behavior in dynamic and unpredictable environments. Over the past few years, researchers in computer science have begun to recognise that the technology of artificial agents provides the key to solving many problems in distributed computing and intelligent control, for which traditional software engineering techniques offer no solution. The field of logic programming includes many important concepts, such as declarativity, unification, meta-logic programming, and deduction rules, from which the new technology of multiagent systems can benefit.

1 Introduction

Although the idea of agent systems is intuitively appealing, and there are a number of implemented systems that claim to realize this popular idea, the basic concepts underlying these systems are often not well-understood, and no attempt is made to define them in a rigorous fashion. The lack of conceptual clarity in the field of agent systems severely hinders the scientific and technological progress. Agent theories proposed so far often suffer from the academic syndrom of a pure theory based on conceptual and ontological stipulations which are not grounded in the practice of information processing but rather follow traditional philosophical abstractions such as the prominent 'possible worlds' semantics. For these theories, it is not clear how to relate them to the programming point of view needed in building real systems.

The theory of artificial agents we propose is built upon the basic components and operations of agent programming systems. It should be construed as an attempt of a practical theory which aims at establishing relevant contributions to the conceptual and software engineering foundations of agent systems. The basic components of an agent, such as its knowledge and perception systems, action and reaction rules, tasks and intentions, are precisely defined in accordance with their operational semantics as programming constructs. In particular, we suggest that the knowledge base of an agent is neither a collection of standard logical formulas (as proposed in many logic-based agent theories), nor a set of simple attribute-value-sentences (as in many implemented systems), but rather an extension of the information system paradigm of relational databases. This implies that an agent may reason nonmonotonically on the basis of the Closed-World

Assumption, and it may process various types of information (such as temporal and fuzzy information) in order to deal with dynamic and noisy environments.

2 What is an Agent ?

There is a growing tendency in the software industry to call certain new types of software 'agents', just because

- they are perceived 'intelligent', e.g. because they provide customized response behavior such as the Firefly^{TM} user interface 'agent'
- they are perceived 'autonomous', e.g. because
  - they are running in the background (like daemons) without immediate user interaction and feedback, such as Lotus Notes^{TM} 'agents';
  - their execution is not bound to a single place (host), but they can be interrupted any time and resumed after they have migrated to another location, such as the *mobile* 'agents' of IBM (called 'Aglets'), or *Agent Tcl* [Gra96].

It should be clear that these systems, though they may incorporate some innovative software technology, do not satisfy any specific conceptual requirement qualifying them as agents. They should therefore rather be called by their traditional (hypeless) names: *(intelligent) user interfaces*, *daemons*, and *mobile objects/processes* or *remote programming*. As argued in [Pet96], any appeal to such obscure notions like 'intelligence' or 'autonomy' is not of much help in justifying the use of a new technical term such as 'agent'.

There are, however, several serious attempts to define the concept of an agent, putting different emphasis on different aspects of agency. Two such attempts, representing two important research camps, are the *software engineering* and the *knowledge representation* approach. We also mention a third category of agent models, the *cognitive science* approach.

2.1 The Software Engineering Approach

Software engineering research is concerned with the synthesis of artificial information processing systems. It largely proceeds by developing new software architectures and techniques, and by building and evaluating prototype implementations on the basis of the newest technology available.

Genesereth and Ketchpel, in [GK94], define: *An entity is a software agent if and only if it communicates correctly in an agent communication language (ACL) like KQML.* Such a language is based on *typed messages*. It is essential that the language contains message types capturing all basic communication acts. In contrast to the application-specific messages in object-oriented programming, ACL message types are application-independent and allow true software interoperability. In addition to application-independent message types, the communication architecture of a multiagent system should be rather *peer-to-peer*, and not client-server, as pointed out in [Pet96].

2.2 The Knowledge Representation Approach

Knowledge representation research develops formal conceptualizations of human cognitive functions for the purpose of reconstructing them as software systems. It includes a large body of theoretical (abstract logic) work which is often more inclined to pure theoretizing than to practical applications. However, there are several important fields where knowlegde representation meets software engineering: notably databases, knowledge systems, and logic programming.

Shoham, in [Sho93], defines the *mentalistic* agent model: *An agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments.* Unfortunately, this classical characterization of agents neglects the dynamic aspects of agency, in particular perception and action which are fundamental notions of the following definition: *Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions.* [HR95]

2.3 The Cognitive Science Approach

Cognitive science is concerned with the analysis of natural information processing systems (such as animals and humans). Its main concern is the investigation and modeling of human cognitive competences.

The Belief-Desire-Intention (BDI) model of agents, usually attributed to [RG91], as well as related work on 'rational' agents (e.g., [CL90, Sin95]), is based on work in philosophy, cognitive science and AI on *intentional attitudes*. In these approaches, primary importance is given to various notions of desire, goal and intention while perception, reaction and communication are often neglected. Various multi-modal logics, based on a multitude of accessibility relations, have been proposed as the formal semantics of BDI agents. Computationally, these logics are highly complex. Conceptually, they abstract away from the functional components of agents and are rather concerned about issues of cognitive science from an *external* ('objective' observer) point of view, and not about software engineering and knowledge representation issues, and the internal perspective of an agent designer.

The schematic use of accessibility relations for each mental component, and the various attempts to establish 'axioms' for them, however, do not shed much light on the semantics of these notions. In many cases, the proposed 'axioms' are either trivial, such as $GOAL(p) \supset BEL(GOAL(p))$, or they are problematic, such as $GOAL(p) \supset BEL(p)$, both from [RG91].

Although it follows from the above remarks, that the modal-logic 'BDI' approach is not grounded in the practice of information processing but rather follows traditional philosophical abstractions, such as the 'posible worlds' semantics, there are many agent system implementations refering to it as their theoretical basis (in fact, it seems to be the currently most popular theory in this area). "Hence, the implemented BDI systems have tended to use the three

major attitudes as data structures, rather than as modal operators." [Rao96]. The problem with this is that it creates a "large gap between theory and practice", as admitted in [Rao96].

Rao [1996] concedes that BDI logics "have shed very little light on the practical problems". As an attempt to bridge the gap between theory and practice of BDI agents, he proposes a logic-programming-like language, called *AgentSpeak(L)*, which allows to specify a certain type of reactive agent whose KB is a set of literals upon which classical inference is performed (i.e. there is no form of the CWA). Similar to our semantic account of KP agents, Rao defines a transition system semantics for his AgentSpeak(L) agents.

3 The Need for Formal Concepts

There are a number of prototype agent systems implementing the central functions of agents, such as beliefs, perceptions, actions, typed message peer-to-peer communication, etc. However, in these systems, the theory of agency embodied by the agent system is expressed only as code, with the relationships among the agent's beliefs, perceptions, goals, intentions, and actions left implicit in the implementation. If the program changes, then so may the embodied theory. The program code, be it Prolog, C++ or Java, is usually too low-level to be mapped onto readable specifications of the essential agent functionality. It is therefore difficult to reason with and about such an implicit theory or program. And it is difficult to obtain a functional understanding of such a system at the level of an explicit agent theory which abstracts away from implementation details.

In order to get a deeper understanding of a new idea, and in order to make further progress in its development, it is essential to establish formal concepts and methods whose properties can be mathematically analyzed. Only formal concepts can serve as a non-ambiguous and platform-independent reference framework for comparisons and further extensions which are necessary for any real progress.

However, we should not attempt to establish a formal definition of an agent in general. This is not necessary, and probably even impossible, as there is also no definition of *what is a number* in mathematics, but only definitions of specific kinds of numbers capturing important cases, such as natural or rational numbers. The same applies to databases: there is no formal definition of what is a database in general, but only of specific kinds of databases, such as relational or deductive databases.

While we can certainly not find a generic definition of *the agent*, we should find out what are the important cases of agent types to be captured by precise mathematical definitions. Such a conceptualization can only be successful if it is based on a sufficiently rich body of practical experience.

3.1 Knowledge- and Perception-Based (KP) Agents

While we can associate *implicit* notions of *goals* and *intentions* with any "intentional system", be it natural or artifical (according to D. Dennett), it is only

the *explicit* notion (of a goal or an intention) which counts for an artificial agent from the programming point of view. Having an explicit goal requires that there is some identifiable data item in the agent program which represents exactly this goal, or the corresponding sentence. Having explicit goals makes only sense for an agent, if it is capable of generating and executing plans in order to achieve its goals. Simple agents, however, which are purely *reactive*, do not generate and execute plans for achieving explicit goals assigned to them at run time (i.e. do not behave *pro-actively*), but only react to events according to their reactive behavior specification. Of course, a reaction pattern can be viewed as encoding a certain task or goal which is implicit in it. But unlike explicit goals, such implicitly encoded tasks have to be assigned to the agent at design time by programming/hardwiring them into the agent system.

So what are the basic components shared by all important – and even very simple – types of agents ? At any moment, the state of any such agent comprises *beliefs* (about the current state of affairs) and *perceptions* (of communication and environment events), and possibly other components such as tasks/goals, intentions, obligations, emotions, etc. While the agent's beliefs are represented in its *knowledge base* (KB), its perceptions are represented (in the form of incoming messages) in its *event queue* (EQ). We obtain the following picture:

$$\text{agent state} = \text{beliefs} + \text{perceptions} + \ldots$$

or, formally,

$$A = \langle KB, EQ, \ldots \rangle$$

And the state of a purely reactive agent may very well consist of just these two components, and nothing else:

$$\text{reactive agent specification} = \text{reaction patterns} + \text{initial state}$$
$$\text{reactive agent state} = \text{beliefs} + \text{perceptions}$$
$$(\text{or, formally, } A = \langle KB, EQ \rangle)$$

Technically, the beliefs in a KB are expressions in some representation language. For instance, they may be simple attribute/variable=value pairs like

$$\mathit{MyName} = 007, \text{ or}$$
$$\mathit{FaxNo[sunshine\_travel\_agency]} = 8132,$$

such as in a conventional program, or atomic sentences like

$$\mathit{I\_am(007)}, \text{ or}$$
$$\mathit{travel\_agency('Sunshine',\ 'Malibu',\ 8132)},$$

such as the table rows in a relational database, or the facts in a Prolog program. In certain cases, beliefs may have to be qualified, e.g. by a degree of uncertainty, a valid-time span, or a security classification, like in

$$\text{diagnosis(network\_component, faulty)} : \mathit{very\_likely}$$
$$\text{connection(switch\_1, switch\_2)} \ @ \ [1.1.97\text{–}1.1.98,\ 1.2.98\text{–}\infty]$$
$$\text{agent(007, 'James Bond', 0815)} \ / \ \mathit{top\_secret}$$

Perceptions may have the form of typed messages labeled with their origination, such as the environment event message

$$\langle \text{ observed(dog(approaching, 300m):0.7), camera\_1 } \rangle,$$

or the communication event message

$$\langle \text{ tell(travel\_agency('Sunshine', 'Malibu', 8132)), 007 } \rangle$$

and FIFO-buffered in the event queue EQ.

Thus, all interesting types of artificial agents are *knowledge- and perception-based (KP)*,[1] but only the more sophisticated (pro-active) agents will have (explicit) goals and intentions.

A KP agent is a software-controlled system whose state comprises beliefs and perceptions. If an agent, in addition to beliefs and perceptions, has any further components, it may be called *KP* agent*. The basic functionality of a KP agent comprises a knowledge system, a perception (event handling) system, and the capability to represent and perform reactions in order to be able to react to events. The behavior of a KP agent is purely reactive since it has no (explicit) tasks or goals to pursue. KP agent systems can be formally modelled by the nondeterministic interleaving of perception and reaction in a labelled state transition system

Notice that a general model of KP agents will have to account for the syntactic and semantic variety of simple and qualified beliefs. Thus, standard first order logic is certainly not adequate for the knowledge system of a KP agent.

Below, we will present a rule-based model of KP agents, called *vivid reagents*, which is generic in the sense that it treats *KB* and *EQ* as black-boxes, but requires

1. that the *KB* of an agent is a conservative extension of a relational database, and
2. that reactions are specified by means of rules.

In the agent-oriented programming language AGENT0, defined in [Sho93], an agent is specified by its initial beliefs, its 'capability rules', and its 'commitment rules'. AGENT0 agents are a particular form of KP agents: their reactions to incoming messages are specified by their commitment rules (which are a particular form of reaction rules).

3.2 Knowledge-Perception-Task-Intention (KPTI) Agents

A Knowledge-Perception-Task-Intention (KPTI) agent is a software-controlled system whose state comprises beliefs/knowledge, perceptions, tasks and intentions. If an agent, in addition to these, has any further components, such as obligations and emotions, it may be called *KPTI** agent.

[1] There is no good reason to construct agents without memories/beliefs, i.e. without knowledge representation, even if they are to be primarily reactive. There may be forms of 'intelligence without representation' [Bro91], but why should one do without the representation of memory and beliefs provided that it enhances the functionality and is not too costly ?

The basic functionality of a KPTI agent comprises, in addition to the functions of a KP agent, the capability to represent and perform actions in order to be able to generate and execute plans. Notice that we make the important distinction between action and reaction: actions are deliberatively planned in order to solve a task or to achieve a goal, while reactions are triggered by communication and environment events. Reactions may be immediate and independent from the current belief state of the agent but they may also depend on the result of deliberation. In any case, they are triggered by events which are not controlled by the agent.

The combination of reactive and proactive behavior in KPTI agent systems can be formally modelled by the nondeterministic interleaving of the five basic *mental transitions*: perception, reaction, planning, plan execution, and replanning.

A *vivid agent* is a KPTI agent

1. whose knowledge base is a conservative extension of a relational database, and
2. whose behavior is represented by means of *action* and *reaction rules*.

The basic components of an agent, such as its knowledge and perception systems, action and reaction rules, tasks and intentions, are defined in accordance with their operational semantics as programming constructs.

4 Inter-Agent Communication

The most basic inter-agent communication acts are those needed for

1. supplying new information (TELL)
2. query answering (ASK-* and REPLY-*, where * = IF, ONE, ALL)
3. requesting certain actions (REQ-DO, CONFIRM, DISCONFIRM)

These message types should be included in the core of any agent communication language (ACL), and any agent theory should include a formal account for them.

The use of *TELL* for supplying new information to an agent is related to the data manipulation commands of SQL and Prolog. An SQL `INSERT` of a new row $\langle a, b, c\rangle$ into the table p, or a Prolog `assert(p(a,b,c))`, corresponds to sending a *TELL(p(a,b,c))* message with the atomic sentence $p(a, b, c)$ as content. Sending a *TELL*($\neg p(a, b, c)$) with the negated sentence $\neg p(a, b, c)$ as content corresponds to an SQL `DELETE` of the respective row, or a Prolog `retract`. This correspondence assumes, however, that the agent deals only with complete information and may therefore work with a relational database as its underlying knowledge system.

Similarly, *ASK-ALL*($p(x, y, z)$) corresponds to the SQL query command SELECT x, y, z FROM p which delivers the collection of all answer substitutions in the form of a table. An *ASK-IF* leads to an if-answer such as *yes*, *no*, or *unknown*. Finally, an *ASK-ONE* yields a single (possibly non-deterministic) answer substitution.

SQL databases may request certain actions from other databases by means of *remote procedure calls*, but there is no standard (application-independent) language for such calls.

A C L	**S Q L**	**Prolog**
TELL($p(a,b,c)$) TELL($\neg q(a)$)	INSERT INTO p VALUES (a,b,c) DELETE FROM q WHERE $x = a$	assert($p(a,b,c)$). retract($q(a)$).
ASK-IF($p(a,b,c)$) ASK-IF($\neg p(a,b,c)$)	n.a. n.a.	?- $p(a,b,c)$. ?- not $p(a,b,c)$.
ASK-ONE($p(x,y,z)$)	n.a.	?- $p(X,Y,Z)$.
ASK-ALL($q(x)$)	SELECT x FROM q	?- findall($X, q(X), Ans$).

Table 1. Correspondences between communication acts and SQL/Prolog.

Similar to the KQML model of communication[2], we assume that the following requirements are met by any KP agent system:

- Agents may interact asynchronously with more than one other agent at the same time.
- Agents are known to one another by their symbolic names, rather than their IP addresses. There may be special agents, called *facilitators*, which provide address information services in order to facilitate communication.
- An agent communicates verbally with other agents: actively by sending, and passively by receiving, typed messages.[3]
- Messages may be sent over network links, or via specific radio links, or, similar to human communication, by means of audio signals. The transport mechanism is not part of the communication model of vivid agents. Certain assumptions about message passing, however, are necessary or useful:
  - When an agent sends a message, it directs that message to a specific addressee.
  - When an agent receives a message, it knows the sender of that message.
  - The order of messages in point-to-point communication is preserved.
  - No message gets lost.
- Message types are defined by a *communication event language* based on speech act theory.
- The arguments of a message (i.e. the 'propositional content' of the corresponding communication act) may affect the mental state of both the sender and the receiver.

Communication in multiagent systems should be based on the *speech act theory* of Austin and Searle [Aus62, Sea69], an informal theory within analytical

[2] See, e.g., [Lab96].

[3] In addition, physical agents may have non-verbal forms of communication, e.g. by means of perception.

philosophy of language. The essential insight of speech act theory was that an utterance by a speaker is, in general, not the mere statement of a true or false sentence, but rather an *action* of a specific kind (such as an assertion, a request, a promise, etc.). Therefore, logic alone is not sufficient for a semantic account of verbal communication.

5 Formal Semantics of KP Agents

The requirement of a formal semantics does not necessarily mean a *possible-worlds*-semantics, such as the many multi-modal logics proposed for 'rational', or BDI, agents (see, e.g., [CL90, RG91, Sin95]). An alternative, but not less formal, approach consists of an *operational* (i.e. transition system) semantics based on mathematical definitions of

1. The *agent state* A (including beliefs KB and perceptions EQ), together with a notion of an *agent state formula* F, and an *inference relation* $\vdash$ between agent states and agent state formulas which may be used to express that the property described by the formula F holds in the agent state $A = \langle KB_A, EQ_A, \ldots \rangle$ as $A \vdash F$. For the purpose of refering to specific components of the agent state A, corresponding meta-predicates are introduced. For instance, if p is a belief of agent A, i.e. $KB_A \vdash p$, or if $\langle m(c), j \rangle$ is the current perception represented by a message type m, content c, and origination j, i.e. $head(EQ_A) = \langle m(c), j \rangle$, then we may define

$$\begin{aligned} A \vdash \mathsf{B}p \;\; \text{iff} \;\; KB_A \vdash p \\ A \vdash \mathsf{rM}[m(c), j] \;\; \text{iff} \;\; head(EQ_A) = \langle m(c), j \rangle \end{aligned}$$

   where the meta-predicates B and rM stand for *belief* and perception (*receive message*).
2. The agent behavior by means of certain operations that may transform the agent state and the environment. Since our internal account of KP agents does not include the 'real' state of the environment (but only some necessarily incomplete representation of it in the agent's knowledge base), the behavior can be described by an algorithmic function *React*, transforming an agent state A to a new state A' by processing the incoming event messages in EQ taking into account the current beliefs in KB:

$$A' = \mathrm{React}(A)$$

Together, these definitions form a labelled state transition system describing the temporal evolution of an agent. Any fixed number of such agent systems can be aggregated and forms a closed multiagent system (MAS) which is again a transition system where subscripted belief and perception operators are introduced to refer to beliefs and perceptions of specific agents A_i according to

$$\begin{aligned} \langle A_1, A_2, \ldots, A_n \rangle \vdash \mathsf{B}_i p \;\; \text{iff} \;\; A_i \vdash \mathsf{B}p \\ \langle A_1, A_2, \ldots, A_n \rangle \vdash \mathsf{rM}_i[m(c), j] \;\; \text{iff} \;\; A_i \vdash \mathsf{rM}[m(c), j] \end{aligned}$$

Notice that for introducing these belief operators there is no need for any modal logic whatsoever. The MAS transition system $\mathcal{S}$ is the basis for the definition of further notions (like in concurrency semantics) such as

1. execution histories ('runs') as possibly infinite sequences of transitions;
2. fairness (qualifying those histories as intended where all enabled agents eventually proceed);
3. assertion formulas with the help of two history operators:
   (a) the invariance assertion $\mathbf{Inv}(F)$ expresses the fact that the state formula F holds in all evolving states of $\mathcal{S}$.
   (b) the leads-to assertion $F \rightsquigarrow G$ expresses the fact that whenever F holds at some point in a history, then there is a later state in this history where G holds.
4. a satisfaction relation between (multi-)agent systems $\mathcal{S}$ and correctness (i.e. safetey and progress) properties expressed by means of assertion formulas: a property is satisfied by $\mathcal{S}$ if it holds in all fair histories of $\mathcal{S}$.

It is important to note that the operational semantics sketched above does neither imply a specific representation language for beliefs or perceptions, nor any specific query language for forming sentences p, nor any specific structure or architecture for the knowledge base KB. The expressions KB, EQ, p, m, c, etc. are black-boxes in this sense. Also, it was not necessary to make any commitment with respect to agent behavior operations determining the possible transitions of $\mathcal{S}$. It is only assumed that the information content of a KB can be queried by means of an (algorithmic) inference relation $\vdash$, and that the state of the event queue can be effectively checked.

5.1 Formal Semantics of Communication Acts

It is an illusion to believe that there is a unique semantic definition of communication acts like there is one for sentential connectives in classical logic. Rather, the semantics of communication acts depends on the behavior properties of the communicating agents. For instance, the semantics of TELL and REPLY depends on whether the receiver may assume that the sender is honest or not. The following collection of properties may serve as an example of an ACL standard.[4] We call it 'GoodAgents'. Notice that the two cases of GoodAgents with or without meta-beliefs are treated separately.

GoodAgents should be aware of their *information competence*, i.e. they should know about which information items they do have complete information (and may therefore apply the Closed-World Assumption), and about which items they don't. This concernes their correct behavior in the case of negative information causing them to answer *no* or *unknown*.[5]

In the sequel, material implication is denoted by $\supset$. The letter p stands for a sentence, while α stands for an action.

[4] Such as strived for by the *Foundation for Intelligent Physical Agents (FIPA)*.
[5] See the definition of the answer operation in relational factbases below.

GoodAgents should be *honest*, *reliable*, *non-vacuous* and *cooperative*. These properties are formalized now with respect to the communication acts TELL, ASK-IF, REPLY-IF, REQ-DO, CONFIRM and DISCONFIRM. The following postulates express these properties and, at the same time, stipulate a semantics for these communication acts. They may be checked for any implemented multiagent system by the formal verification method of *assertional reasoning*.[6]

Honesty

GoodAgents should only TELL someone something if they believe it, and they should believe what they REPLY. Both conditions are safety properties:

$$\mathbf{Inv}(\mathsf{rM}_i[\mathit{TELL}(p), j] \supset \mathsf{B}_j p)$$

$$\mathbf{Inv}(\mathsf{rM}_i[\mathit{REPLY\text{-}IF}(p, \mathit{yes}), j] \supset \mathsf{B}_j p)$$
$$\mathbf{Inv}(\mathsf{rM}_i[\mathit{REPLY\text{-}IF}(p, \mathit{no}), j] \supset \mathsf{B}_j \neg p)$$
$$\mathbf{Inv}(\mathsf{rM}_i[\mathit{REPLY\text{-}IF}(p, \mathit{unknown}), j] \supset \neg \mathsf{B}_j p \wedge \neg \mathsf{B}_j \neg p)$$

Notice that secure agents which withhold confidential information in order to protect it from unauthorized query access (see, e.g., [Wag97]) are excluded from the above definition of honesty (they are only honest to authorized receivers).

In the case of agents with meta-beliefs, we can require in addition that if an agent is told something it adopts the meta-belief that the sender believes what it has told:

$$\mathsf{rM}_i[\mathit{TELL}(p), j] \rightsquigarrow \mathsf{B}_i \mathsf{B}_j p$$

$$\mathsf{rM}_i[\mathit{REPLY\text{-}IF}(p, \mathit{yes}), j] \rightsquigarrow \mathsf{B}_i \mathsf{B}_j p$$
$$\mathsf{rM}_i[\mathit{REPLY\text{-}IF}(p, \mathit{no}), j] \rightsquigarrow \mathsf{B}_i \mathsf{B}_j \neg p$$
$$\mathsf{rM}_i[\mathit{REPLY\text{-}IF}(p, \mathit{unknown}), j] \rightsquigarrow \mathsf{B}_i(\neg \mathsf{B}_j p \wedge \neg \mathsf{B}_j \neg p)$$

Reliability

GoodAgents can safely be assumed reliable, i.e. it is rational to believe what they tell (because they are competent and honest). That is, if a GoodAgent is told someting by another GoodAgent, it should adopt this as a new belief:

$$\mathsf{rM}_i[\mathit{TELL}(p), j] \rightsquigarrow \mathsf{B}_i p$$

$$\mathsf{rM}_i[\mathit{REPLY\text{-}IF}(p, \mathit{yes}), j] \rightsquigarrow \mathsf{B}_i p$$
$$\mathsf{rM}_i[\mathit{REPLY\text{-}IF}(p, \mathit{no}), j] \rightsquigarrow \mathsf{B}_i \neg p$$

Non-Vacuity

Questions are non-vacuous:

$$\mathbf{Inv}(\mathsf{rM}_i[\mathit{ASK\text{-}IF}(p), j] \supset \neg \mathsf{B}_j p \wedge \neg \mathsf{B}_j \neg p)$$

In the case of agents with meta-beliefs, we can require in addition that *TELL* is non-vacuous:

$$\mathbf{Inv}(\mathsf{rM}_i[\mathit{TELL}(p), j] \supset \neg \mathsf{B}_j \mathsf{B}_i p)$$

[6] Further details can be found in [Sha93, Wag96].

Cooperativity

GoodAgents always reply:

$$\begin{aligned}\mathsf{rM}_i[\mathit{ASK\text{-}IF}(p), j] \rightsquigarrow\ & \mathsf{rM}_j[\mathit{REPLY\text{-}IF}(p, \mathit{yes}), i] \\ & \vee\ \mathsf{rM}_j[\mathit{REPLY\text{-}IF}(p, \mathit{no}), i] \\ & \vee\ \mathsf{rM}_j[\mathit{REPLY\text{-}IF}(p, \mathit{unknown}), i]\end{aligned}$$

Requests are confirmed or disconfirmed:

$$\mathsf{rM}_i[\mathit{REQ\text{-}DO}(\alpha), j] \rightsquigarrow \mathsf{rM}_j[\mathit{CONFIRM}(\alpha), i] \vee \mathsf{rM}_j[\mathit{DISCONFIRM}(\alpha), i]$$

6 Vivid Agents

A *vivid agent* is a KPTI agent whose knowledge base is a conservative extension of a relational database, and whose behavior is represented by means of *action* and *reaction rules*. The basic functionality of a vivid agent comprises a knowledge system (including an update and an inference operation), and the capability to represent and perform actions in order to be able to generate and execute plans. Since a vivid agent is 'situated' in an environment with which it has to be able to communicate, it also needs the ability to react in response to environment events, and in response to communication events created by the communication acts of other agents. We formalize the combination of these reactive and proactive aspects of agent behavior by nondeterministic interleaving of perception, reaction, planning and plan execution, resp. action. Notice that we make the important distinction between action and reaction: actions are deliberatively planned in order to solve a task or to achieve a goal, while reactions are triggered by environment and communication events. Reactions may be immediate and independent from the current belief state of the agent but they may also depend on the result of deliberation. In any case, they are triggered by events which are not controlled by the agent.

Our theory of vivid agents is based on the internal or *subjective* view of the world inhabitated by them. This means that there is no need for a notion of objective time, or for the distinction between knowledge ('true' belief) and belief. In contrast, these concepts are essential to external or *objective* theories of agents such as [CL90] or [FHMV95]. While our subjective theory of agents corresponds to the programming point of view, objective theories try to capture the perspective of an external, eternal and perfect observer of the world, that is the perspective of God.

We do not assume a fixed formal language and a fixed logical system for the knowledge base of an agent.[7] Rather, we believe that it is more appropriate to choose a suitable knowledge system for each agent individually according

[7] It is important to recognize that for information and knowledge processing, unlike classical first-order logic for mathematics, there is no ONE TRUE LOGIC, but many different logical systems accounting for different kinds of knowledge such as temporal, uncertain, confidential, inconsistent, disjunctive, deductive, etc.

to its domain and its tasks. In simple cases, a relational database-like system (admitting of atomic sentences only) will do the job, while in more involved cases one may need the ability to process, in addition to simple facts, (disjunctive or gradual) uncertain information, temporal information, or even such advanced capabilities as deductive query answering and abductive reasoning.

The knowledge system of a vivid agent will be nonmonotonic, since one needs the Closed-World Assumption, and negation-as-failure, in any practical system. Notice that this departs from the use of standard logics (enriched by various modal operators) which is common in many other logical approaches to agent modeling. Vivid agents can be obtained by extending vivid knowledge systems through the addition of action and reaction rules, i.e. one can 'plug in' any suitable knowledge system for constructing a specific agent system. Since our definition of action and reaction rules applies to all kinds of knowledge systems, this makes vivid agents scalable. Our rule-based approach to agent specification is more computational than modal logic approaches based on possible worlds semantics because it refers to the actual components of agent systems needed in programming and not to philosophical abstractions.

The combination of a knowledge base with action and reaction rules yields an *executable specification* of an agent, or of a multi-agent system. This is similar to the idea of PROgramming in LOGic where programs have both a procedural and a declarative reading. Our concept of vivid agents, thus, is able to narrow the gap between agent theory and practical systems, a gap which seems to be insuperable in many other logic-based approaches.

While certain agents may have rather limited capabilities, others are quite complex. We call the simplest form of a vivid agent a *reagent*. A reagent does not have explicit goals and intentions but only beliefs about the current state of affairs. It reacts to events in its environment, taking into account what it currently believes. A reagent updates its beliefs and draws inferences from them by applying the respective operations of the vivid knowledge system it is based on.

6.1 Vivid Knowledge Systems

The knowledge system of a vivid agent is based on three specific languages: L_{KB} is the set of all admissible knowledge bases,[8] L_{Query} is the query language, and L_{Input} is the set of all admissible inputs, i.e. those formulas representing new information a KB may be updated with. In a diagnosis setting, L_{Input} may be $\{test(\_,\_), diagnoses(\_,\_)\}$, where *test* is used to update other agents' test results and *diagnoses* to update the agents' diagnosis results. While the input language defines what the agent can be told (i.e. what it is able to assimilate into its KB),

[8] It seems to be unrealistic to allow for arbitrary formulas in a KB for a number of reasons: a KB concept has to be a conservative extension of that of relational databases; it has to provide for negation-as-failure and for some kind of CWA mechanism; the amount of 'disjunctiveness' of a KB needs special care; there will be null values rather than existential quantifiers; etc.

the query language defines what the agent can be asked. Where L is a set of formulas, L^0 denotes its restriction to closed formulas (sentences). Elements of L^0_{Query}, i.e. closed query formulas, are also called *if-queries*.

A **knowledge system**[9] $\boldsymbol{K}$ consists of three languages and two operations: a knowledge representation language L_{KB}, a query language L_{Query}, an input language L_{Input}, an inference relation $\vdash$, such that $X \vdash F$ holds if $F \in L^0_{\text{Query}}$ can be inferred from $X \in L_{\text{KB}}$, and an update operation Upd, such that the result of updating $X \in L_{\text{KB}}$ with $F \in L^0_{\text{Input}}$ is the knowledge base $\mathsf{Upd}(X, F)$.

We now present two basic examples of knowledge systems: relational databases and factbases. While the former can only be used under the very strong assumption of complete information about all represented predicates (implying perfect – i.e. competent, reliable and honest – information sources), the latter allow to represent incomplete predicates in addition to complete ones. In many agent domains, it will be necessary to represent and reason with various forms of incomplete information. Relational factbases, and *extended logic programs*, are therefore important extensions of relational databases and normal logic programs.

Relational Databases

A finite set of ground atoms corresponds to a relational database. For instance, in diagnosis a relational database may contain observations and connections of the system to be diagnosed: $X_1 = \{hi(s_1), conn(s_1, s_2)\}$, may represent the information that switch s_1 is high and that it is connected to switch s_2. As a kind of natural deduction from positive facts an inference relation $\vdash$ between a database X and an if-query is defined in the following way:

$$\begin{array}{ll} (a) & X \vdash a \text{ if } a \in X \\ (\neg a) & X \vdash \neg a \text{ if } a \notin X \end{array}$$

Notice the non-monotonicity of $(\neg a)$. Negation in relational databases corresponds to *negation-as-failure*. For example, $X_1 \vdash hi(s_1) \wedge \neg hi(s_2)$. Because of its built-in general Closed-World Assumption, a relational database X answers an if-query F by either yes or no: the answer is yes if $X \vdash F$, and no otherwise.

Updates are insertions, $\mathsf{Upd}(X, a) := X \cup \{a\}$, and deletions, $\mathsf{Upd}(X, \neg a) := X - \{a\}$, where a is an atom. For instance,

$$\mathsf{Upd}(X_1, \neg hi(s_1) \wedge hi(s_2)) = \{conn(s_1, s_2), hi(s_2)\}$$

describes a possible transaction.

The knowledge system of relational databases is denoted by $\boldsymbol{A}$.[10] **Knowledge systems extending $\boldsymbol{A}$ conservatively are called vivid**. Positive vivid knowledge systems use a general Closed-World Assumption, whereas general

[9] See also [Wag95].

[10] $\boldsymbol{A}$ stands for $\boldsymbol{A}$tomic.

vivid knowledge systems employ specific Closed-World Assumptions (and possibly two kinds of negation). For instance, $\boldsymbol{A}$ can be extended to a general vivid knowledge system by allowing for literals instead of atoms as information units (see below). Further important examples of positive vivid knowledge systems are temporal, fuzzy and disjunctive databases. All these kinds of knowledge bases can be extended to *deductive knowledge bases* by adding deduction rules of the form $F \leftarrow G$ [Wag95]. The semantics of deductive knowledge bases is determined by *stable generated* models [HW97].

Relational Factbases and Extended Logic Programs

A knowledge base consisting of a consistent set of ground literals (viewed as positive and negative facts) is called a *relational factbase*. In a relational factbase, the CWA does not in general apply to all predicates, and therefore in the case of a non-CWA predicate, negative information is stored along with positive. This allows to represent predicates for which the KB does not have complete information.

The schema of a factbase stipulates for which predicates the CWA applies by means of a special set *CWRel* of relation symbols. Explicit negative information is represented by means of a *strong* negation $\neg$. For instance, in the factbase

$$\begin{aligned} CWRel &= \{conn\} \\ X_2 &= \{conn(s_1, s_2),\ \neg hi(s_1)\} \end{aligned}$$

the CWA applies only to the predicate *conn* representing the connection of components, i.e. if it is not positively confirmed that two components are connected we assume that they are not. In contrast to this, the CWA does not apply to *hi* anymore. Now we can distinguish the two cases that we have explicitly observed that a switch is not high and that we do not have information about the switch. I.e. $X_2 \vdash \neg hi(s_1)$ means that switch s_1 is observed to be not-high (i.e. low), whereas $X_2 \vdash \sim hi(s_2)$ only expresses that we cannot infer s_2 to be high, which means that it is either not high or that there is no information. As a kind of natural deduction from positive and negative facts an inference relation $\vdash$ between a factbase X and an if-query is defined in the following way:

$$\begin{array}{ll} (\neg a) & X \vdash \neg p(c) \text{ if } \neg p(c) \in X \\ (\sim a) & X \vdash \sim p(c) \text{ if } p(c) \notin X \\ (\neg_{\mathrm{CWA}}) & X \vdash \neg p(c) \text{ if } p \in CWRel \ \ \& \ \ X \vdash \sim p(c) \end{array}$$

where $p(c)$ stands for an atomic sentence with predicate p and constant (tuple) c. $\sim$ and $\neg$ are also called *weak* and *strong* negation. Note that, since X is consistent, the strong negation implies the weak negation:

$$X \vdash \neg F \text{ implies } X \vdash \sim F$$

Compound formulas are treated according to DeMorgan and double negation rules.[11] A factbase X answers an if-query F by yes if $X \vdash F$, by no if $X \vdash \neg F$,

[11] Inference in factbases corresponds to predicate circumscription in partial logic, i.e. to preferential entailment based on minimal coherent partial models.

and by unknown otherwise. Updates are recency-preferring revisions:

$$\mathsf{Upd}(X, p(c)) := \begin{cases} X \cup \{p(c)\} \text{ if } p \in \mathit{CWRel} \\ X - \{\neg p(c)\} \cup \{p(c)\} \text{ otherwise} \end{cases}$$
$$\mathsf{Upd}(X, \neg p(c)) := \begin{cases} X - \{p(c)\} \text{ if } p \in \mathit{CWRel} \\ X - \{p(c)\} \cup \{\neg p(c)\} \text{ otherwise} \end{cases}$$

The knowledge system of relational factbases is denoted by $\boldsymbol{F}$. The extension of $\boldsymbol{F}$ by adding deduction rules leads to *extended logic programs* with two kinds of negation.

An **extended logic program** consists of a factbase and a set of deduction rules of the form

$$l_0 \leftarrow l_1, \ldots, l_m, \sim l_{m+1}, \ldots, \sim l_n \ (0 \leq m \leq n)$$

where each l_i is a positive or negative fact ($l_i = a | \neg a$, $0 \leq i \leq n$).

Inference in extended logic programs can be defined model-theoretically as preferential entailment based on stable generated partial models [HW97, HJW97] or, equivalently, by the fixpoint semantics of *answer sets* [GL90].

6.2 Reagents

Simple vivid agents whose mental state comprises only beliefs and perceptions, and whose behavior is purely reactive, i.e. not based on any form of planning and plan execution, are called *reagents*. A reagent $\mathcal{A}$ on the basis of

1. a vivid knowledge system $\boldsymbol{K} = \langle L_{\mathrm{KB}}, \vdash, L_{\mathrm{Query}}, \mathsf{Upd}, L_{\mathrm{Input}} \rangle$,
2. an environment and communication event language L_{PEvt} and L_{CEvt} whose union is denoted by L_{Evt}, and
3. an action language L_{Act}

is a tripel $\mathcal{A} = \langle X, EQ, RR \rangle$, consisting of

1. a knowledge base $X \in L_{\mathrm{KB}}$,
2. an event queue EQ recording environment and communication events in the form of incoming messages, and
3. a set RR of *reaction rules* which code the reactive and communicative behavior.

A multi-reagent system is a tuple of reagents:

$$\mathcal{S} = \langle \mathcal{A}_1, \ldots, \mathcal{A}_n \rangle$$

Reaction rules encode the behavior of vivid agents in response to environment event messages created by the agent's perception subsystems, and to communication event messages created by communication acts of other agents. We distinguish between epistemic, physical and communicative reaction rules, and call the latter *interaction rules*. The following table describes the different formats of reaction rules:

epistemic	$\mathit{Eff} \leftarrow \mathsf{recvMsg}[\varepsilon, S],\ \mathit{Cond}$
physical	$\mathsf{do}(\alpha),\ \mathit{Eff} \leftarrow \mathsf{recvMsg}[\varepsilon, S],\ \mathit{Cond}$
communicative	$\mathsf{sendMsg}[\eta, R],\ \mathit{Eff} \leftarrow \mathsf{recvMsg}[\varepsilon, S],\ \mathit{Cond}$

The event condition $\mathsf{recvMsg}[\varepsilon, S]$ is a test whether the event queue of the agent contains the message ε sent by some perception subsystem of the agent or by another agent identified by S, where $\varepsilon \in L_{\mathrm{Evt}}$ represents an environment or a communication event. The epistemic condition $\mathit{Cond} \in L_{\mathrm{Query}}$ refers to the current knowledge state, and the epistemic effect $\mathit{Eff} \in L_{\mathrm{Input}}$ specifies an update of the current knowledge state. In a physical reaction, $\mathsf{do}(\alpha)$ calls a procedure realizing the action α. In a communicative reaction, $\mathsf{sendMsg}[\eta, R]$ sends the message $\eta \in L_{\mathrm{CEvt}}$ to the receiver R.

In general, reactions are based both on perception and on knowledge. Events are represented by incoming messages.[12] We identify a communication act with the corresponding communication event which is perceived by the addressee of the communication act.

Reaction rules are triggered by events. The agent interpreter continuously checks the event queue of the agent. If there is a new event message, it is matched with the event condition of all reaction rules, and the epistemic conditions of those rules matching the event are evaluated. If they are satisfiable in the current knowledge base, all free variables in the rules are instantiated accordingly resulting in a set of triggered actions with associated epistemic effects. All these actions are then executed, leading to physical actions and to sending messages to other agents, and their epistemic effects are assimilated into the current knowledge base.

6.3 Defining the Execution of Reagents

We propose a *perception-reaction cycle* as the execution model of a reagent system. Informally, it consists of the following steps:

1. Get the next message from the event queue, and check whether it triggers any reaction rules. If it does not, then repeat 1, else continue.
2. For each of the triggered reaction rules, check whether its epistemic condition is satisfied; if it is, assimilate the epistemic effect of the triggered action into the knowledge base, and in case it is
   (a) a physical action, execute it by calling the associated procedure.
   (b) a communicative action, execute it by sending the corresponding message to the specified addressee.
3. Continue with step 1.

[12] In a robot, for instance, appropriate perception subsystems, operating concurrently, will continuously monitor the environment and interpret the sensory input. If they detect a relevant event pattern in the data, they report it to the knowledge system of the robot in the form of an environment event message.

Kowalski, in [Kow95], has proposed to use the formalism of meta-logic programming to define "the observation-thought-action cycle of an agent that combines the ability to perform resource-bounded reasoning, which can be interrupted and resumed any time, with the ability to act when it is necessary". We make use of several of Kowalski's suggestions, in particular his inferability meta-predicate *demo*, and his update meta-predicate *assimilate*, from [Kow79]. Notice, however, that in our treatment these meta-predicates are based on our knowledge system concepts, allowing for various degrees of expressiveness and various kinds of logical inference, and are therefore more general than in Kowalski's proposal.

We propose the following *cycle* procedure as a Prolog-style meta-logic specification of a reagent:

```
cycle( KB)
← newEvent( Evt),
   findall( ActEff, (reaction(ActEff,Evt,Cond), demo(KB,Cond)), ActEffs),
   perform( ActEffs, KB, KB′),
   cycle( KB′).

perform( [], KB, KB).

perform( [Act/Eff | ActEffs], KB, KB′)
← execute( Act),
   assimilate( Eff, KB, KB1),
   perform( ActEffs, KB1, KB′).

execute( noAct).        % EPISTEMIC ACTION = only assimilate

execute( do(Act))       % PHYSICAL ACTION
← call( Act).

execute( send(Msg,To))      % COMMUNICATIVE ACTION
← pvm_send( To, 1, Msg).    % implemented in PVM-Prolog
```

Here, reaction rules are represented as triples $\langle Act/Eff, Evt, Cond\rangle$ in the table *reaction*. A null action *noAct* is used to represent epistemic actions as *noAct/Eff*. An incoming event message *Evt* is popped from the message queue, and subsequently matched with suitable reaction rules. If the precondition *Cond* of a rule matching *Evt* holds in the current knowledge state, expressed by *demo(KB, Cond)*, the epistemic effect *Eff* associated with the action *Act* is assimilated into the knowledge base, the physical or communicative action *Act* is performed by means of appropriate procedure calls, and *cycle* starts over with the updated knowledge base KB′. The *demo* and *assimilate* meta-predicates are formally related to our knowledge system concepts of inference and update:

$$\begin{aligned} demo(\mathrm{KB}, Cond) &:\Longleftrightarrow \mathrm{KB} \vdash Cond \\ assimilate(Eff, \mathrm{KB}, \mathrm{KB}') &:\Longleftrightarrow \mathrm{KB}' = \mathsf{Upd}(\mathrm{KB}, Eff) \end{aligned}$$

We have implemented a multi-reagent system on top of PVM-Prolog. In [SW97], we show that reagents (with extended logic progams as their knowledge system) can achieve distributed model-based diagnosis requiring sophisticated forms of reasoning (including default rules and abduction) and inter-agent communication.

6.4 Vivid Agents

While the behavior of a reagent consists alone in its reactions to perception and communication events, proactive agents can in addition generate and execute plans in order to achieve their goals. The proactive behavior repertoire of a vivid agent is represented by means of (epistemc, physical and communicative) *action rules*:

epistemic	$\mathit{Eff} \leftarrow \mathit{Cond}$
physical	$\mathsf{do}(\alpha),\ \mathit{Eff} \leftarrow \mathit{Cond}$
communicative	$\mathsf{sendMsg}[\eta, R],\ \mathit{Eff} \leftarrow \mathit{Cond}$

A planning problem on the basis of a knowledge system $\boldsymbol{K}$ and a set of action rules AR is given by

1. a knowledge base $X_0 \in L_{\mathrm{KB}}$, representing the initial situation, und
2. a goal $G \in L^0_{\mathrm{Query}}$, for which a plan has to be generated.

The simplest notion of a plan consists of a sequence π of elementary actions represented by instantiated action rules. Such a sequence of action rules can be viewed as a composition of update functions transforming the initial state X_0 into the state $\pi(X_0)$ where the goal G is achieved:

$$\pi(X_0) \vdash G$$

A vivid agent on the basis of a vivid knowledge system $\boldsymbol{K}$ and event and action languages L_{PEvt}, L_{CEvt}, and L_{Act}, is a tripel $\mathcal{A} = \langle M, RR, AR\rangle$, consisting of

(M) a **mental state** $M = \langle X, TL, CI, EQ\rangle$, with
  (X) a knowledge base $X \in L_{\mathrm{KB}}$,
  (EQ) an event queue $EQ \in (L^0_{\mathrm{Evt}})^*$,
  (TL) a set of *tasks* $TL \subseteq L^0_{\mathrm{Query}}$, and
  (CI) a set of *current intentions* $CI = \{G_1/P_1, \ldots, G_n/P_n\}$, consisting of goal/plan pairs G_i/P_i, such that $P_i \in AR^*$ is a plan to achieve $G_i \in L^0_{\mathrm{Query}}$;

(RR) a set $RR \subseteq (L_{\mathrm{CEvt}} \cup L_{\mathrm{Act}}) \times L_{\mathrm{Input}} \times L_{\mathrm{Evt}} \times L_{\mathrm{Query}}$ of **reaction rules**, and

(AR) a set $AR \subseteq (L_{\mathrm{CEvt}} \cup L_{\mathrm{Act}}) \times L_{\mathrm{Input}} \times L_{\mathrm{Query}}$ of **action rules**, representing the behavior repertoire available to the agent.

An agent specification $\mathcal{A}$ is executed by interleaving the reactive behavior determined by *X, EQ, RR* with the proactive behavior determined by *X,TL,CI,AR*. A simple but natural stipulation of priorities would be to prefer perception and reaction over planning and plan execution.

As a possible solution to the problem of resource bounds and real-time planning, we have developed a system where (re)action execution and planning are performed concurrently, see [SMCW97].

7 Conclusion

Logic programming can play an important role in agent systems:

Declarativity: The idea of *executable specifications* also applies to agent systems.

Unification: In the action- and reaction-rule-based specification of agent behavior the unification concept of Prolog is essential.

Deduction Rules: The knowledge base of advanced agents can represent intensional predicates and heuristics by means of Prolog-style deduction rules.

Meta-Programming: The execution model of agents can be specified by means of a Prolog meta-interpreter.

However, the modeling of agent systems requires substantial additions and extensions of the conceptual framework of logic programming:

Agents are much broader. Agent systems are much broader than logic programs. They may subsume logic programs (deduction rules) as a specific type of knowledge system.

Normal logic programs are inadequate. Since in many cases agents have to deal with incomplete predicates, they rather need extended instead of normal logic programs in their knowledge system. In fact, they even need the extension of extended logic programs: viz the combination of weak and strong negation with (fuzzy, temporal, security, reliability, etc.) qualifications

Reasoning is not the most important thing for agents. Perception, reaction and communication are more fundamental for agents than sophisticated forms of reasoning such as achieved, e.g., by abductive and disjunctive logic programs.

Agents need a dynamic semantics. Unlike the static semantics of normal logic programs based on stable generated Herbrand models, agent systems need a dynamic semantics such as the transition system semantics proposed in [Wag96].

References

[Aus62] J.L Austin. *How to Do Things with Words.* Harvard University Press, Cambridge (MA), 1962.

[Bro91] R.A. Brooks. Intelligence without reason. In *Proc. IJCAI'91*, pages 569–595, 1991.

[CL90] P.R. Cohen and H.J. Levesque. Intention is choice with commitment. *AI*, 42(3), 1990.

[FHMV95] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge (MA), 1995.

[GK94] M.R. Genesereth and S.P. Ketchpel. Software agents. *Communication of the ACM*, 37(7):48–53, 1994.

[GL90] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *Proc. of Int. Conf. on Logic Programming*. MIT Press, 1990.

[Gra96] R.S. Gray. Agent tcl: A flexible and secure mobile agent system. In *Proc. of Fourth Annual Usenix Tcl/Tk Workshop*, pages 9–23. http://www.cs.dartmouth.edu/ agent/papers/tcl96.ps.Z, 1996.

[HJW97] H. Herre, J. Jaspars, and G. Wagner. Partial logics with two kinds of negation as a foundation of knowledge-based reasoning. In D.M. Gabbay and H. Wansing, editors, *What Is Negation ?* Oxford University Press, Oxford, 1997.

[HR95] B. Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72:329–365, 1995.

[HW97] H. Herre and G. Wagner. Stable models are generated by a stable chain. *J. of Logic Programming*, 30(2):165–177, 1997.

[Kow79] R.A. Kowalski. *Logic for Problem Solving*. Elsevier, 1979.

[Kow95] R.A. Kowalski. Using meta-logic to reconcile reactive with rational agents. In *Meta-Logics and Logic Programming*. MIT Press, 1995.

[Lab96] Y. Labrou. *Semantics for an Agent Communication Language*. PhD thesis, University of Maryland Graduate School, 1996.

[Pet96] C.J. Petrie. Agent-based engineering, the web, and intelligence. *IEEE Expert*, 1996.

[Rao96] A.S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. Van de Velde and J.W. Perram, editors, *Agents Breaking Away, LNAI 1038*, volume 1038 of *LNAI*, pages 42–55. Springer-Verlag, 1996.

[RG91] A.S. Rao and M.P. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proc. KR-91*, San Mateo (CA), 1991. Morgan Kaufmann.

[Sea69] J.R. Searle. *Speech Acts*. Cambridge University Press, Cambridge (UK), 1969.

[Sha93] A.U. Shankar. An introduction to assertional reasoning for concurrent systems. *ACM Computing Surveys*, 25(3):225–262, 1993.

[Sho93] Y. Shoham. Agent-oriented programming. *AI*, 60:51–92, 1993.

[Sin95] M.P. Singh. *Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communication*, volume 799 of *LNAI*. Springer-Verlag, 1995.

[SMCW97] M. Schroeder, R. F. Marques, J. C. Cunha, and G. Wagner. CAP – concurrent action and planning: Using PVM-prolog to implement vivid agents. In *Proc. of Practical Applications of Prolog (PAP'97)*, 1997.

[SW97] M. Schroeder and G. Wagner. Distributed diagnosis by vivid agents. In *Proc. 1st Int. Conf. On Autonomous Agents (Agents'97)*. ACM Press, 1997.

[Wag95] G. Wagner. From information systems to knowledge systems. In W. Hesse E.D. Falkenberg and A. Olivé, editors, *Information System Concepts*, pages 179–194, London, 1995. Chapman & Hall.

[Wag96] G. Wagner. A logical and operational model of scalable knowledge- and perception-based agents. In W. Van de Velde and J.W. Perram, editors, *Agents Breaking Away (Proc. of MAAMAW'96)*, pages 26–41. Springer-Verlag, LNAI 1038, 1996.

[Wag97] G. Wagner. Multi-level security in multiagent systems. In P. Kandzia and M. Klusch, editors, *Cooperative Information Agents (Proc. of CIA'97)*, pages 272–285. Springer-Verlag, LNAI 1202, 1997.

This article was processed using the LaTeX macro package with LLNCS style