Conceptual Foundations of Artificial Agents

Gerd Wagner[1]

December 23, 1996

[1]gw@inf.fu-berlin.de, Institut für Informatik, Universität Leipzig, Augustusplatz 10-11, D-04109 Leipzig, Germany.

Contents

Chapter 1

General Introduction

As Rodney Brooks has pointed out, many AI theories of knowledge representation have only been proposed to solve "anomalies within formal systems which are never used for any practical task" [Bro91]. These rather academic theories typically make, or follow, conceptual and ontological stipulations which are not grounded in the practice of information processing. Useful theories, on the other hand, are built on the basic components and operations constituting their domain of application.

The theory of artificial agents developed in this book should be construed as an attempt of a practical theory which aims at establishing relevant contributions to the conceptual and software engineering foundations of multiagent systems.

1.1 Practical Theory and Theory-Based Practice

In computer science, the relationship between theory and practice seems to be uneasy in comparison to other sciences, such as, e.g., chemistry or electrical engineering. This may be partly created by the strong influence of mathematicians (in theoretical computer science) who don't care much about real systems, but also by the obvious success of ingenious programmers and infamous hackers who don't care much about theory. It may indicate that computer science is 'very special', but it is more likely just a sign of its immaturity.

1.1.1 Do the Real Thing

Why should we try to develop a theoretical basis for agent systems rather than immediately develop agent systems themselves on our favorite platforms ? Programmers tend to be impatient. They don't want to wait for suitable theories. They prefer to do the real thing right away.[1] And it is exactly this pioneer-

[1] Only Dijkstra's theoretically motivated insistence that GOTO statements have to be considered harmful has eventually led to more programming discipline in the software industry.

ing behavior which has led to many new ideas and techniques in computer science. But 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. To see this, let's take a brief look at the history of two other fields of computer science: the success story of database technology, and the still-no-success (= failure?) story of expert systems.

1.1.2 The Success of Database Systems

In the sixties and seventies, pushed by the need to store and process large data collections, powerful database software based on the file system technology available at that time has been developed. These types of systems have been named *hierarchical* and *network* databases, referring to the respective type of file organization. Although these systems were able to process large amounts of data efficiently, their limitations in terms of flexibility and ease of use were severe. Those difficulties were caused by the rather low-level data operations of hierarchical and network databases dictated by their respective file organization. Thus, both database models have later on turned out to be cognitively inadequate. Only through the formal conceptualization of *relational databases* by Codd in the early seventies could the inadequacy of the then prevailing database technology be overcome. Only the logic-based formal concepts of the relational database model have led to more cognitive adequacy, and have thus constituted the conceptual basis for further progress (towards deductive, OO, temporal, etc. databases). Unlike much of theoretical computer science, Codd's theory of relational databases is an example of a practical theory.

1.1.3 The Difficulties of Expert Systems

In the field of expert systems, on the other hand, there has never been a conceptual breakthrough like that of the relational database model. It seems that the field is still at the 'pre-conceptual' level of hierarchical and network databases, and there seems to be almost no measurable progress since MYCIN. There is neither a formal definition of an 'expert system', nor is there any clear semantics of expert system rules. The field has rather developed a variety of notions (such as 'production rules,' or 'certainty factors') which have often not been clearly defined, or have not been based on a logical semantics. Lacking a clear (and formally defined) conceptual framework, it is difficult for the expert system community to compare different systems and to identify shortcomings and measure

Dijkstra also reports (in [SL95]): *The programmers didn't like the idea [of verification] at all because it deprived them of the intellectual excitement of not quite understanding what they were doing.*

progress. One may argue that these problems are due to the inherent difficulties of knowledge representation and processing, but it rather seems that the expert system community has failed to develop cognitively adequate formal concepts and has even failed to learn from its neighbor fields of logic programming and deductive databases which have successfully developed a clear concept of rules based on a logical semantics.

So, the question for the field of multiagent systems really is: does it learn its lesson from the database and expert system stories, or is it going do repeat the mistakes of the expert system field ?

1.1.4 What is an Agent ?

There are several informal definitions of the concept of an agent, putting different emphasis on different aspects of agency. A definition which is compatible with the model of vivid agents is

> "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." (Hayes-Roth, 1995)

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.

1.2 Vivid Agents

A *vivid agent* is a software-controlled system whose state consists of components such as beliefs, goals and intentions, which have a *mentalistic semantics* in the sense of McCarthy and Shoham [McC79, Sho93], 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 perception 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 perception and communication events. Reactions may be immediate and independent from the current believe 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 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.[2] Rather, we believe that it is more appropriate to choose a suitable knowledge system for each agent individually according 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

[2]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, active, etc.

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.

1.3 An Overview of Vivid Agents

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.

1.3.1 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, [3] 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), 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*.

An **abstract knowledge system**[4] $\boldsymbol{K}$ 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 $F \in L^0_{\mathrm{Query}}$ can be inferred from $X \in L_{\mathrm{KB}}$, and an update operation Upd, such that the result of updating $X \in L_{\mathrm{KB}}$ with $F \in L^0_{\mathrm{Input}}$ is the knowledge base $\mathsf{Upd}(X, F)$.

We now present two basic examples of knowledge systems: relational databases and factbases.

[3] 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.

[4] See also [Wag95].

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:

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

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 $\boldsymbol{A}$.[5] Knowledge systems extending $\boldsymbol{A}$ conservatively are called *vivid*. Positive vivid knowledge systems use a general Closed-World Assumption, whereas general 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, uncertain 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].

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

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

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

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:

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

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.[6] A factbase X answers an if-query F by yes if $X \vdash F$, by no if $X \vdash \neg F$, and by unknown otherwise. Updates are recency-preferring revisions:

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

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

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

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

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

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

1.3.2 Specification of Reagents

Simple vivid agents whose mental state comprises only beliefs, and whose behavior is purely reactive, i.e. not based on any form of planning and plan execution, are called *reagents*. A reagent $\mathcal{A} = \langle X, EQ, RR \rangle$, on the basis of a knowledge system $\boldsymbol{K}$ consists of

1. a knowledge base $X \in L_{\mathrm{KB}}$,
2. an event queue EQ being a list of instantiated event expressions, and
3. a set RR of *reaction rules*, consisting of epistemic and physical reaction and interaction rules which code the reactive and communicative behavior of the agent.

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 perception events created by the agent's perception subsystems, and to communication events created by communication acts of other agents. We distinguish between epistemic, physical and communicative reaction rules, and call the latter *interaction rules*. We use L_{PEvt} and L_{CEvt} to denote the perception and communication event languages, and $L_{\mathrm{Evt}} = L_{\mathrm{PEvt}} \cup L_{\mathrm{CEvt}}$. The following table describes the different formats of epistemic, physical and communicative reaction rules:

$$\begin{array}{rcl} \mathit{Eff} & \leftarrow & \mathsf{recvMsg}[\varepsilon, S],\ \mathit{Cond} \\ \mathsf{do}(\alpha),\ \mathit{Eff} & \leftarrow & \mathsf{recvMsg}[\varepsilon, S],\ \mathit{Cond} \\ \mathsf{sendMsg}[\eta, R],\ \mathit{Eff} & \leftarrow & \mathsf{recvMsg}[\varepsilon, S],\ \mathit{Cond} \end{array}$$

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 a perception 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. Perception and communication events are represented by incoming messages.[7] We

[7] 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 a perception event message.

identify a communication act with the corresponding communication event which is perceived by the addressee of the communication act.[8]

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.

1.3.3 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.

1.4 Inter-Agent Communication

Similar to the KQML model of communication[9], we assume that the following requirements are met by any vivid 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.

[8] We assume that communication channels are perfect in the sense that no message gets lost, and in point-to-point communication all messages are received in the order they have been sent.

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

- An agent communicates verbally with other agents: actively by sending, and passively by receiving, typed messages.[10]
- 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 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.

In the vivid agent model, the semantics of communicative actions is rather determined by

1. a mentalistic model of agents, defining their *mental state*, together with a notion of mental conditions and mental effects of actions,
2. a satisfaction relation between mental states and mental conditions,
3. an operation that assimilates mental effects into a mental state,
4. the assignment of a mental precondition and a mental effect to each action, and
5. associating with each type of communicative action a type of reaction (of the addressee of a communication act).

[10] In addition, there may be non-verbal forms of communication, e.g. by means of perception.

In our *reagent* model, for instance, the mental state of an agent consists only of beliefs (represented in a KB), the mental satisfaction relation and the mental assimilate operation are $\vdash$ and Upd, and communicative actions are represented by means of reaction rules.

1.5 VIVA Knowledge-Based Agent Programming

VIVA is a rule-based agent-oriented programming language. It adopts many concepts from SQL and Prolog such as, e.g., the distinction between the schema and the state of an agent, or the use of facts and rules with negation-as-failure, logical variables and unification. In addition to the well-known *deduction rules* of Prolog, VIVA employs *action rules* for representing the proactive behavior repertoire, and *reaction rules* for representing the reactive behavior of an agent.

A first prototype interpreter (not yet suitable for distribution) has been implemented on the basis of PVM-Prolog. It is currently used to evaluate basic constructs of the language. We plan to implement a fully-fledged VIVA interpreter on the basis of *Visual Prolog*, and make it available as as an executable for Windows, OS/2 and LINUX PCs.

1.6 Main Contributions

The main contributions of this work are the concepts of:

Vivid Knowledge Systems – a comprehensive framework of *knowledge representation and reasoning* integrating many diverse disciplines such as nonmonotonic and uncertain reasoning, and various kinds of qualified information such as temporal, unreliable, and confidential information. While the basic principles of vivid knowledge systems were already discussed in [Wag94b], the present work contains a much more elaborate presentation. In particular, we

1. establish the novel concepts of *fuzzy* and *possibilistic databases* and logic programs – a new formalization of reasoning with uncertainty within our knowledge system framework based on a new compositional definition of *possibilistic logic*;
2. formalize *multi-level security* in knowledge systems;
3. propose a new semantics of deduction rules in terms of stable grounded and *stable generated closures* inspired by the notion of *stable generated models* of [HW97].

Reagents – simple agents, based on a vivid knowledge system, whose mental state comprises only beliefs and whose behavior is purely reactive (including high-level communication acts). Reagents are defined as triples

$\langle \text{KB}, EQ, RR \rangle$ consisting of a knowledge base KB, an event queue EQ and a set of reaction rules RR.

Vivid Agents – reactive and proactive agents, based on a vivid knowledge system, whose mental state comprises beliefs, tasks and intentions. Vivid agents are defined as quadruples $\langle M, EQ, RR, AR \rangle$, where the mental state M comprises in addition to the agent's beliefs (KB) further mental components (such as tasks and intentions), and the set of action rules AR represents the pro-active behavior repertoire.

VIVA – a rule-based agent-oriented programming language in the tradition of SQL and Prolog, turning the vivid agent model into a practical programming system.

Acknowledgements: Many thanks to Heinrich Herre, Michael Schroeder and Jan Jaspars for the cooperation and stimulating exchange during this work. I am also grateful to Luis Farinas del Cerro, Michael Gelfond, Luis Moniz Pereira and Michael Wooldridge for their support and valuable comments on preliminary material of this book. Finally thanks to Daniela A. Plewe for a stimulating discussion on the question of planning reactions which I will take up in future work.