Vivid Agents: How They Deliberate, How they React, How They Are Verified*

Gerd Wagner (gw@inf.fu-berlin.de)
Institut für Informatik, Universität Leipzig,
Augustusplatz 10-11, D-04109 Leipzig, Germany

Abstract

We propose a model of an agent which is both logical and operational. Our model of *vivid agents* takes into account that agents need not only the ability to draw inferences but also to update their current knowledge state, to represent and to perform (and to simulate the execution of) actions in order to generate and execute plans, and to react and interact in response to perception and communication events. We illustrate our formalization of this basic functionality of an agent by means of examples. We also show how our model fits into the transition system semantics of concurrent reactive systems by identifying the five basic transitions of vivid agent systems: perception, reaction, planning, action, and replanning.

Contents

*The present paper improves and extends [Wag96a]. Much of this work was performed while the author was a research fellow at Univ. Nova de Lisboa with an HCM grant of the EU. The author wants to thank Luis M. Pereira for his hospitality and support.

1 Introduction

A *vivid agent* is a software-controlled system whose state is represented by a knowledge base, and whose behavior is represented by means of *action* and *reaction rules*. Following [McC79, Sho93], the state of an agent is described in terms of mental qualities, such as beliefs and intentions. 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 [F+95]. While our subjective theory of agents corresponds to the programming point of view, objective theories try to capture the perspective of an eternal, external 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.[1] 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 inconsistency handling. 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, resp. 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.

2 Knowledge Systems

The language of knowledge systems consists of the logical functors conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), exclusive disjunction $|$, and the truth constant true; predicate symbols such as $p, q, r, \ldots$; constant symbols such as $c, d, \ldots$; and variables such as $x, y, \ldots$. For simplicity, we shall not consider functional terms but only variables and constants. An *atom* a is an atomic formula, it is called *proper*, if $a \neq \mathsf{true}$. *Literals* are either atoms or negated atoms, $l = a | \neg a$. We use $a, b, \ldots, l, k, \ldots$, and $F, G, H, \ldots$ as metavariables for atoms, literals, and well-formed formulas, respectively. A variable-free expression is called *ground*. The set of all proper atoms (literals) of a given language is denoted by At (Lit). If $\mathcal{F}$ is a set of connectives, say $\mathcal{F} \subseteq \{\mathsf{true}, \neg, \wedge, \vee, |, \supset, \exists, \forall, \ldots\}$, then $L(\mathcal{F})$ denotes the respective set of well-formed formulas. Where L is a language (a set of formulas), L^0 denotes its restriction to closed formulas (sentences).

We use three specific languages: L_{KB} is the set of all admissible knowledge bases of a knowledge system,[2] L_{Query} is the query language, and L_{Input} is the set of all admissible

[1] 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.

[2] 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

inputs, i.e. those formulas representing new information a KB may be updated with. 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. Elements of L^0_{Query}, i.e. closed query formulas, are also called *if-queries.*

Definition 1 (Knowledge System) *An abstract knowledge system* $\boldsymbol{K}$ *is a septuple:*[3]

$$\boldsymbol{K} = \langle L_{\text{KB}}, \vdash, L_{\text{Query}}, \mathsf{Ans}, L_{\text{Ans}}, \mathsf{Upd}, L_{\text{Input}} \rangle$$

where $\vdash \subseteq L_{\text{KB}} \times L^0_{\text{Query}}$ *is called* inference relation*,* $\mathsf{Ans} : L_{\text{KB}} \times L_{\text{Query}} \to L_{\text{Ans}}$ *is called* answer operation*, and* $\mathsf{Upd} : L_{\text{KB}} \times L^0_{\text{Input}} \to L_{\text{KB}}$ *is called* update operation*, satisfying for any* $X \in L_{\text{KB}}$*,*

(KS1a) $X \vdash \mathsf{true}$*, and* $\mathsf{Upd}(X, \mathsf{true}) = X$.
(KS1b) $X \vdash F$ *iff* $\mathsf{Ans}(X, F) = \mathsf{yes}$.
(KS2) $L_{\text{Input}} \subseteq L_{\text{Query}}$.
(KS3) $\mathsf{Upd}(X, F) \vdash F$*, for any* $F \in L^0_{\text{Input}}$ *which is compatible with* X*.*

In the sequel, we sometimes omit Ans and L_{Ans}, and write a knowledge system simply as a quintuple.

In many cases, it is useful to be able to update by a set of inputs and we therefore 'overload' the symbol Upd to denote also this more general update operation

$$\mathsf{Upd} : L_{\text{KB}} \times 2^{L^0_{\text{Input}}} \to L_{\text{KB}}$$

which has to be defined in such a way that for any finite $A \subseteq L^0_{\text{Input}}$, $\mathsf{Upd}(X, A) = \mathsf{Upd}(X, \bigwedge A)$.

We now present several examples of basic knowledge systems for agents. The simplest one is the knowledge system of relational databases which requires complete and consistent information about the predicates of the application domain, implying that all information sources are absolutely reliable and honest.

2.1 Relational Databases

A finite set of ground atoms corresponds to a relational database. For instance, $X_1 = \{r(S), r(P), m(P, L)\}$, may represent the information that both Peter and Susan are residents, and that Peter is married to Linda. As a kind of natural deduction from positive facts an inference relation $\vdash$ between a database $X \subseteq \text{At}$ and an if-query is defined in the following way:

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

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.

[3]The formulation of a knowledge system in terms of query and input processing was already implicit in [Bel77]. In [Lev84] it was proposed as a 'functional approach to knowledge representation'. In [Wag94, Wag95] the concept of knowledge systems was further extended and used as an integrating framework for knowledge representation and logic programming.

Compound if-queries, involving conjunction and disjunction, are handled in the standard way. Negated compound if-queries are treated by simplification according to the DeMorgan rules and double negation elimination. Inference in relational databases corresponds to preferential entailment based on minimal models. For instance, $X_1 \vdash m(P,L) \wedge \neg(r(P) \wedge r(L))$. Because of its built-in general Closed-World Assumption, a relational database X answers an if-query F by either yes or no:

$$\mathsf{Ans}(X,F) = \begin{cases} \mathsf{yes} \text{ if } X \vdash F \\ \mathsf{no} \text{ otherwise} \end{cases}$$

Updates are insertions, $\mathsf{Upd}(X,a) := X \cup \{a\}$, and deletions, $\mathsf{Upd}(X,\neg a) := X - \{a\}$, of atoms $a \in \mathrm{At}$. For a consistent set of literals E, we have $\mathsf{Upd}(X,E) = X \cup E^+ - E^-$, where E^+ contains the positive, and E^- contains the negative literals of E. For instance,

$$\mathsf{Upd}(X_1, \neg m(P,L) \wedge m(P,S)) = \{r(S),\, r(P),\, m(P,S)\}$$

describes a possible transaction. In the presence of integrity constraints $IC \subseteq L^0_{\mathrm{Query}}$, updates must not violate any of them, and thus $\mathsf{Upd}(X,F)$ is the knowledge base *closest* to X satisfying both F and all $G \in IC$.

The knowledge system of relational databases, denoted by $\boldsymbol{A}$, is then defined as

$$\boldsymbol{A} \;=\; \langle 2^{\mathrm{At}},\, \vdash,\, L(\neg,\wedge,\vee),\, \mathsf{Ans},\, \{\mathsf{yes},\mathsf{no}\},\, \mathsf{Upd},\, \mathrm{Lit}\rangle$$

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, called *relational factbases*, 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, which can be extended to corresponding general vivid knowledge systems called temporal, uncertain and disjunctive factbases. All these kinds of knowledge bases can be extended to *deductive knowledge bases* by adding deduction rules of the form $F \leftarrow G$ (see [Wag95]).

2.2 Relational Factbases

A knowledge base consisting of a consistent set of ground literals (viewed as positive and negative facts) is called a *relational factbase*. In a 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. For instance, the factbase

$$\begin{aligned} \mathit{CWRel} &= \{r\} \\ X_2 &= \{r(S), r(P), s(S), \neg s(L), \neg s(P),\, m(P,L),\, m(T,S)\} \end{aligned}$$

may represent the information that Susan and Peter are residents, Susan is a smoker, Linda and Peter are nonsmokers, Peter is married to Linda, and Tom is married to

Susan. Only the predicate *resident* is assumed to be completely represented, i.e. subject to the CWA, since the local KB does not have complete information about smokers and marriages while it has complete information about residents.

As a kind of natural deduction from positive and negative facts an inference relation $\vdash$ between a factbase $X \subseteq \mathrm{Lit}$ and an if-query is defined in the following way:

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

Compound formulas are treated like in $\boldsymbol{A}$. For instance, one may ask "who is married with a nonsmoking non-resident ?", i.e. $X_2 \vdash m(x,y) \wedge \neg s(y) \wedge \neg r(y)$. An if-query F is answered according to

$$\mathsf{Ans}(X,F) = \begin{cases} \mathsf{yes} \text{ if } X \vdash F \\ \mathsf{no} \text{ if } X \vdash \neg F \\ \mathsf{unknown} \text{ otherwise} \end{cases}$$

Updates are recency-preferring revisions:

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

The knowledge system of relational factbases, denoted by $\boldsymbol{F}$, is defined as

$$\boldsymbol{F} = \langle 2^{\mathrm{Lit}}, \vdash, L(\neg,\wedge,\vee), \mathsf{Ans}, \{\mathsf{yes},\mathsf{no},\mathsf{unknown}\}, \mathsf{Upd}, \mathrm{Lit}\rangle$$

Inference in $\boldsymbol{F}$ corresponds to preferential entailment based on models which are minimal with respect to *CWRel*, or equivalently, to predicate circumscription [McC80].

2.3 Unreliable Information

In relational databases and factbases it is assumed that all information sources authorized to enter new information into the knowledge base are completely reliable and honest, and therefore new information always overrides old information (in relational databases it is even assumed that all information suppliers have complete information). In many domains, however, and especially in many agent scenarios, this assumption is not realistic, and different sources of information have to be distinguished in terms of authorization, reliability and honesty. While reliability concerns the frequency of (unintended) errors, honesty is a matter of benevolence and truthfulness. The knowledge system of *source-labelled (SL-) factbases* represents a principled approach to the problem of reliability and honesty. Its main principles are:

1. A SL-factbase assigns reliability degrees $\rho(s) \in [-1,1]$ to its information sources s. Complete reliability is expressed by $\rho(s) = 1$, or $\rho(s) = -1$. If $\rho(s) = 0$, the source s is completely unreliable, and hence completely uninformative. If $\rho(s) < 0$, the source s is not honest, and $\rho(s) = -1$ means that s always lies. Reliability degrees correspond to possibilistic certainty values, see [Wag96c]. For

a completely reliable piece of information l, the source s is not recorded together with l since this is unnecessary. Instead, l is assimilated into the SL-factbase like in relational factbases. An unreliable piece of information l is recorded together with its source s as $l{:}s$.

2. Information about Closed-World predicates is only accepted from completely reliable sources, i.e. if $p \in CWRel$,

$$\mathsf{Upd}(X, p(c){:}s) = \begin{cases} X \cup \{p(c)\} \text{ if } \rho(s) = 1 \\ X - \{p(c)\} \text{ if } \rho(s) = -1 \\ X \text{ otherwise} \end{cases}$$

3. If the information source is qualified as completely reliable, it leads to recency-preferring revision:

$$\mathsf{Upd}(X, l{:}s) = \begin{cases} X - \{\tilde{l}\} \cup \{l\} \text{ if } \rho(s) = 1 \\ X - \{l\} \cup \{\tilde{l}\} \text{ if } \rho(s) = -1 \end{cases}$$

where $l = a|\neg a$ stands for a literal, and $\tilde{l}$ denotes its complement.

4. More reliable information overrides less reliable:

$$\mathsf{Upd}(X, l{:}s) = \begin{cases} X - \{l{:}s', \tilde{l}{:}s'\} \cup \{l{:}s\} \text{ if } l{:}s' \in X \text{ or } \tilde{l}{:}s' \in X, \text{ and } |\rho(s)| > |\rho(s')| \\ X \text{ if } l{:}s' \in X \text{ or } \tilde{l}{:}s' \in X, \text{ and } |\rho(s)| < |\rho(s')| \end{cases}$$

5. New information from the same honest source as the old one leads to a recency-preferring revision, while old information from a liar is preferred to new information from the same liar:

$$\mathsf{Upd}(X, l{:}s) = \begin{cases} X - \{\tilde{l}{:}s\} \cup \{l{:}s\} \text{ if } \tilde{l}{:}s \in X \text{ and } \rho(s) > 0 \\ X \text{ if } \tilde{l}{:}s \in X \text{ and } \rho(s) < 0 \end{cases}$$

6. If new information contradicts old one and stems from a different source with the same degree of reliability, this leads to mutual neutralization:

$$\mathsf{Upd}(X, l{:}s) = X \cup \{l{:}s\}$$

if $\tilde{l}{:}s' \in X$ and $\rho(s') = \rho(s)$, or $l{:}s' \in X$ and $\rho(s') = -\rho(s)$.

An answer to an if-query is either yes, no, or unknown, or an uncertainty-qualified yes, no, or yes-and-no. For Closed-World predicates $p \in CWRel$,

$$\mathsf{Ans}(X, p(c)) = \begin{cases} \mathsf{yes} \text{ if } p(c) \in X \\ \mathsf{no} \text{ otherwise} \end{cases}$$

$$\mathsf{Ans}(X, \neg p(c)) = \begin{cases} \mathsf{yes} \text{ if } p(c) \notin X \\ \mathsf{no} \text{ otherwise} \end{cases}$$

For non-CWA predicates,

$$\mathsf{Ans}(X,l) = \begin{cases} \mathsf{yes} \ \text{ if } \ l \in X \\ \mathsf{no} \ \text{ if } \ \tilde{l} \in X \\ \mathsf{unknown} \ \text{ if } \ l{:}s \notin X \ \ \& \ \ \tilde{l}{:}s \notin X \\ \mathsf{yes} : \rho(s) \ \text{ if } \ l{:}s \in X \ \ \& \ \ \tilde{l}{:}s' \notin X \\ \mathsf{no} : \rho(s) \ \text{ if } \ \tilde{l}{:}s \in X \ \ \& \ \ l{:}s' \notin X \\ \mathsf{yes\text{-}and\text{-}no} : \rho(s) \ \text{ if } \ l{:}s \in X \ \ \& \ \ \tilde{l}{:}s' \in X \end{cases}$$

Below, in section 4.4, we will present an example of processing unreliable information.

3 Action Rules

Action rules have the general form of *Action* ← *Condition*.[4] We distinguish between action and reaction rules. While action rules can be used to represent the elementary action types of an agent, and to generate and execute plans, thus determining the *proactive* behavior, reaction rules are used to specify the *reactive* behavior of an agent.

Depending on the resp. domain and the capabilities and tasks of an agent, only certain actions can be performed by it. We distinguish between epistemic, physical and communicative actions. Epistemic actions only change the beliefs of an agent. Physical actions change the environment, but they may also lead to a corresponding belief change if the agent is aware of their epistemic effects and if their recording is relevant to its operation. A physical action is therefore represented by a procedure call (affecting the available actuators) and an epistemic effect formula. Communicative actions may change the beliefs of all involved agents. They are realized by means of asynchronous message passing via the available communication links (either audio or radio links in the case of robots, or network links in the case of embedded systems and software agents).

Action rules combine declarative queries (or inference) with state change (or update). Since in general, the effects of an action may be context-dependent, an action is represented by a set of action rules expressing the dependence of different effects on different preconditions. The execution of an action in a situation described by the knowledge base X is realized by firing the corresponding action rule whose precondition *Cond* holds in X, i.e. $X \vdash \mathit{Cond}$.

3.1 Epistemic Action Rules

Epistemic action rules have the form

$$r \ : \ \mathit{Eff} \ \leftarrow \ \mathit{Cond}$$

where $\mathit{Eff} \in L_{\mathrm{Input}}$, and $\mathit{Cond} \in L_{\mathrm{Query}}$, are logical formulas expressing an epistemic effect and a precondition, and r is the name of the action rule. We require that *Cond* is an *evaluable* formula and that all free variables of *Eff* also occur in *Cond*.[5]

[4] Certain forms of action rules were proposed under the name of 'production rules' in expert systems, and under the name of 'event-condition-action' rules as an extension of databases, now called *active databases*, see e.g. [McCD89].

[5] Rules of this form, where all free variables of the conclusion occur in the premise, and the premise is evaluable, are called *range-restricted*. In the sequel, we assume that rules are always range-restricted. See [vGT91] for the notion of *evaluable* or *domain-independent* formulas.

Definition 2 (Rule Application) *An epistemic action rule $r : E \leftarrow C$ represents an update function, i.e. a mapping $r : L_{\mathrm{KB}} \to L_{\mathrm{KB}}$, with*

$$r(X) \;:=\; \mathsf{Upd}(X,\ \{E\sigma : \sigma \textit{ is a substitution s.th. } X \vdash C\sigma\})$$

Notice that in the case of a non-applicable rule, we get (by definition, resp. convention)

$$r(X) = \mathsf{Upd}(X, \emptyset) = \mathsf{Upd}(X, \bigwedge \emptyset) = \mathsf{Upd}(X, \mathsf{true}) = X$$

Also, if a rule $r : E \leftarrow C$ does not have free variables, then

$$r(X) = \begin{cases} \mathsf{Upd}(X, E) \text{ if } X \vdash C \\ X \text{ otherwise} \end{cases}$$

If a rule $r : E \leftarrow C$ has free variables, say $\mathrm{Free}(C) = \{x_1, \ldots, x_n\}$, then we also write its name as the expression $r(x_1, \ldots, x_n)$ in order to be able to refer to specific instantiations of it by means of $r(c_1, \ldots, c_n)$ where the free variables x_i in r are substituted by constants c_i.

Example 1 *Referring to the database X_1 from 2.1, we consider the epistemic action rule*

$$r_1 : r(y) \leftarrow (m(x, y) \vee m(y, x)) \wedge r(x)$$

expressing the fact that everyone getting married with a resident becomes also a resident. Applying it to X_1 yields

$$r_1(X_1) = X_1 \cup \{r(L), r(T)\}$$

since $(m(x, y) \vee m(y, x)) \wedge r(x)$ can be inferred from X_1 with the two substitutions $\{x/P, y/L\}$ and $\{x/S, y/T\}$. Alternatively, we could obtain the same result by successively applying the two ground instances $r_1((P, L)$ and $r_1(S, T)$ of $r_1(x, y)$ to X_1:

$$r_1(X_1) = r_1(P, L) \circ r_1(S, T)(X_1) = r_1(P, L)(r_1(S, T)(X_1))$$

3.2 Physical and Communicative Action Rules

Physical action rules have the form

$$r \;:\; \mathsf{do}(\alpha),\ \mathit{Eff} \;\leftarrow\; \mathit{Cond}$$

while communicative action rules have the form

$$r \;:\; \mathsf{sendMsg}[\eta, R],\ \mathit{Eff} \;\leftarrow\; \mathit{Cond}$$

where $\mathit{Eff} \in L_{\mathrm{Input}}$, and $\mathit{Cond} \in L_{\mathrm{Query}}$, are logical formulas expressing the epistemic effects, resp. precondition, of the action; $\mathsf{do}(\alpha)$ calls procedure $\alpha \in L_{\mathrm{Act}}$ affecting some actuators available to the agent; $\mathsf{sendMsg}[\eta, R]$ is a procedure call to execute the communication act η from the communication event language L_{CEvt} such that R identifies the receiver of the outgoing message η; and r is the name of the resp. action rule. We assume that a communication act $\mathsf{sendMsg}[\eta, R]$ is realized by asynchronous message passing through perfect channels, but abstract away from the channel in our formal treatment below.[6]

When we are only interested in the epistemic representation of an action we omit the execution command $\mathsf{do}(\alpha)$, resp. $\mathsf{sendMsg}[\eta, R]$, in the action rule r, thus obtaining its *epistemic projection* $\dot{r}$. The execution of an action which is represented by the action rule r in a situation described by a knowledge base X corresponds on the epistemic level to the application of $\dot{r}$ to the current knowledge state X, and yields the updated knowledge state $\dot{r}(X)$.

Example 2 (Elevator Actions) *In an elevator scenario, we may have the elementary actions of going one floor up, or one floor down, i.e.* $L_{\mathrm{Act}} = \{\mathrm{oneup}, \mathrm{onedown}\}$, *represented by the action rules*

$$\begin{array}{ll} r_u: & \mathsf{do}(\mathrm{oneup}),\ at(x+1) \wedge -at(x) \ \leftarrow\ at(x) \wedge x < topfloor \\ r_d: & \mathsf{do}(\mathrm{onedown}),\ at(x-1) \wedge -at(x) \ \leftarrow\ at(x) \wedge x > 0 \end{array}$$

Let the current situation, where the elevator is halting at the 3rd floor, be described by $X_0 = \{at(3)\}$, *and let topfloor* $= 10$. *Then, performing the action* oneup *corresponds to the rule application*

$$\dot{r}_u(X_0) = \dot{r}_u(3)(X_0) = X_0 - \{at(3)\} \cup \{at(4)\} = \{at(4)\}$$

In this example, it is assumed that the action of moving one floor upwards is immediate, i.e. without duration. In a more realistic modeling, such as the one below, one would rather define the action of *starting* to move upwards.

4 Reaction and Interaction Rules

Reaction rules encode the behavior of an agent 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*. L_{PEvt} and L_{CEvt} denote the perception and communication event languages, and $L_{\mathrm{Evt}} = L_{\mathrm{PEvt}} \cup L_{\mathrm{CEvt}}$. The following table describes the different reaction rule formats:

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

where $\varepsilon \in L_{\mathrm{Evt}}$ represents a perception or a communication event created by some perception subsystem of the agent or by another agent identified by S, and thus the event condition $\mathsf{recvMsg}[\varepsilon, S]$ is a test whether the event queue of the agent contains the message ε sent by S. Both perception and communication events will be represented by incoming messages.[7]

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

[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.

In general, reactions are based both on perception and on knowledge. Immediate reactions do not allow for deliberation. They are represented by rules with an empty epistemic premise, i.e. $Cond = \mathsf{true}$. Timely reactions can be achieved by guaranteeing fast response times for checking the precondition of a reaction rule. This will be the case, for instance, if the precondition can be checked by simple table look-up (such as in relational databases and factbases). It will be more difficult in knowledge systems with deduction rules (e.g. deductive databases).

Notice that we identify a communication act with the corresponding communication event which is perceived by the addressee of the communication act.Typically, the set of possible communication acts includes *telling*, *asking*, *replying*, and more. As noted in [Pei76], different communication acts (also called 'speech acts' in the literature) may have the same propositional content:

> One and the same proposition may be affirmed, denied, judged, doubted, inwardly inquired into, put as a question, wished, asked for, effectively commanded, taught, or merely expressed, and does not thereby become a different proposition.

4.1 Example: An Elevator as a Reagent

In order to describe an elevator as a purely reactive agent we use three action types: $L_{\mathrm{Act}} = \{mvup, mvdown, halt\}$; and two perception event types: $L_{\mathrm{PEvt}} = \{reqTo(x), arrAt(x)\}$. Since there is no need for inter-agent communication, $L_{\mathrm{CEvt}} = \emptyset$. Whenever the elevator, while moving up or down, arrives at some floor k, this is signalled by the environment as the perception event $arrAt(k)$. Whenever the elevator is requested to serve floor k, signalled by the event $reqTo(k)$, this will be recorded in the KB by means of the fluent $req(k)$. The state of the elevator is described by means of the fluents $at(x)$, $up(x)$, and $down(x)$, i.e. predicates representing the information that the elevator is currently halting at floor x, moving upwards from floor x, or moving downwards from floor x.

The behavior of the elevator can be specified by means of the following reaction rules:

$$
\begin{array}{ll}
r_1: & \mathsf{do}(\mathrm{mvup}),\ req(x) \wedge \neg at(y) \wedge up(y) \\
 & \leftarrow\ \mathsf{recvMsg}[\mathrm{reqTo}(x)],\ at(y) \wedge x > y \wedge x < topfloor \\
r_2: & \mathsf{do}(\mathrm{mvdown}),\ req(x) \wedge \neg at(y) \wedge down(y) \\
 & \leftarrow\ \mathsf{recvMsg}[\mathrm{reqTo}(x)],\ at(y) \wedge x < y \wedge x > 0 \\
r_3: & up(x) \wedge \neg up(x-1) \\
 & \leftarrow\ \mathsf{recvMsg}[\mathrm{arrAt}(x)],\ req(y) \wedge x < y \wedge x < topfloor \\
r_4: & down(x) \wedge \neg down(x+1) \\
 & \leftarrow\ \mathsf{recvMsg}[\mathrm{arrAt}(x)],\ req(y) \wedge y < x \wedge x > 0 \\
r_5: & \mathsf{do}(\mathrm{halt}),\ at(x) \wedge \neg req(x) \wedge \neg up(x-1) \wedge \neg down(x+1) \\
 & \leftarrow\ \mathsf{recvMsg}[\mathrm{arrAt}(x)],\ req(x)
\end{array}
$$

Notice that for this simple elevator system we do not have to distinguish between the request made by pushing the button in the elevator in order to move it to the target floor, and the request made by someone waiting for the elevator to come to his floor. In both cases, since this type of elevator has no recording facility, requests are only accepted if the elevator is currently halting at some floor. Notice also that the behavior of the elevator does not depend on its knowledge about moving up or down because it completely relies on the signals *arrAt(x)* from the environment. We have included the *up* and *down* information in the epistemic effects of the reaction rules, however, in order to be able to prove below the *correctness* of this specification which requires to make the interaction with the environment explicit.

Assuming the initial situation $X_0 = \{at(3)\}$ and an incoming request from the fifth floor, i.e. $\mathsf{recvMsg}[\mathrm{reqTo}(5)]$, we get in the first step

$$X_1 = \dot{r}_1(X_0) = \{req(5),\, up(3)\}$$

and the elevator is moving upwards. When it arrives at the fourth floor, $arrAt(4)$ is signalled, and the reaction rule r_3, causing the continuation of moving up, is applicable

$$X_2 = \dot{r}_3(\{req(5),\, up(3)\}) = \{req(5),\, up(4)\}$$

Finally, the fifth floor is reached, i.e. $\mathsf{recvMsg}[\mathrm{arrAt}(5)]$ holds, and r_5 is applicable:

$$X_3 = \dot{r}_5(\{req(5),\, up(4)\}) = \{at(5)\}$$

4.2 Verifying the Elevator Specification

We might want to prove that the elevator specification is correct in the sense that every service request when the elevator is idle will be served. This means that we have to show that for any k and any m with $0 \leq k, m \leq topfloor$, the event $reqTo(k)$ leads to $at(k)$, if $at(m)$ holds. The latter condition requires that the elevator must currently be halting, i.e. idle, at some floor. Let $X_0 = \{at(m)\}$, and let $\mathsf{recvMsg}[\mathrm{reqTo}(k)]$ hold. Since the assertion holds trivially for $k = m$ we may assume that $k \neq m$. Thus, in the first step either r_1 or r_2 is triggered yielding either of the states $X_1 = \{req(k), up(m)\}$, or $X_1 = \{req(k), down(m)\}$. It suffices, therefore, to prove that whenever $req(k) \wedge (up(m) \vee down(m))$ holds, this leads (in finite time) to a state where $at(k)$ holds.

We assume that the environment behaves reasonably, i.e. that whenever the elevator is moving up or down, after some time it will receive the signal (in the form of an incoming message) that it has arrived at the next floor from the environment.

Proof by induction on $|k - m|$: Let $|k - m| = 1$. If $k = m + 1$, then $X_1 = \{req(m+1), up(m)\}$, and the elevator is moving upwards. Since by assumption, $up(x)$ leads to $\mathsf{recvMsg}[\mathrm{arrAt}(x+1)]$, the message $arrAt(m+1)$ will be received. This triggers r_5 and leads to $X_2 = \{at(k)\}$. Likewise, if $k = m-1$, i.e. $X_1 = \{req(m-1), down(m)\}$, the elevator is moving downwards. After some time the event $arrAt(m-1)$ will be signalled. This triggers r_5, and leads to $X_2 = \{at(k)\}$.

Now assume that the specification is correct for $|k-m| = n \geq 1$, and let $|k-m| = n+1$. If $k > m$, we have $X_1 = \{req(k), up(m)\}$, and the elevator is moving upwards. After some time the event $arrAt(m+1)$ is signalled, triggering r_3, and yielding $X_2 = \{req(k), up(m+1)\}$. By the induction hypothesis, since $|k - (m+1)| = n$, this leads to $X_{n+2} = \{at(k)\}$. An analogous argument holds for $k < m$. $\square$

4.3 Example: Communicating Elevators

In the communicating elevator scenario, we assume that two elevators a and b operate in the same shaft and must not collide. For simplicity, we shall consider the case with three floors. Elevator a serves floors 1 and 2, while elevator b serves floors 2 and 3, and hence the critical zone is floor 2. The states of the elevators are described by means of the indexical fluents $at(x)$, $up(x)$, and $down(x)$, i.e. indexical predicates representing the information that the resp. elevator is currently halting at floor x, moving upwards from floor x, or moving downwards from floor x.

In addition to perception events, $L_{\mathrm{PEvt}} = \{\mathrm{reqTo}(x), \mathrm{arrAt}(x)\}$, we now also have communication events in order to avoid collisions and deadlocks. The elevators use two communication acts for coordination: requesting the permission to approach, and granting the permission, expressed by $L_{\mathrm{CEvt}} = \{reqPerm, grPerm\}$. A special fluent $turn(x)$ controls whose turn it is when both want to go to floor 2 at the same time. We assume that initially $turn(a)$ holds for both elevators. This leads to the following specification, including a coordination protocol.

The behavior of elevator a is specified by the rules $a_1, \ldots, a_{10}$, while that of elevator b is specified by the dual rules $b_1, \ldots, b_{10}$. If a is $at(1)$ and receives a request to serve floor 2, it records the service request, and asks b for permission to go to floor 2; likewise if b is $at(3)$, and receives a request to serve floor 2:

$$\begin{array}{rl} a_1: & \mathsf{sendMsg}[\mathrm{reqPerm}, b],\ req(2) \leftarrow \mathsf{recvMsg}[\mathrm{reqTo}(2)],\ at(1) \wedge \neg req(2) \\ b_1: & \mathsf{sendMsg}[\mathrm{reqPerm}, a],\ req(2) \leftarrow \mathsf{recvMsg}[\mathrm{reqTo}(2)],\ at(3) \wedge \neg req(2) \end{array}$$

If a receives a request from b to give permission for floor 2, and a is $at(1)$ and has not itself received a request to serve floor 2, or a moves down from floor 2, it grants the requested permission; likewise, in the case of b:

$$\begin{array}{rl} a_2: & \mathsf{sendMsg}[\mathrm{grPerm}, b] \leftarrow \mathsf{recvMsg}[\mathrm{reqPerm}, b],\ (at(1) \wedge \neg req(2)) \vee down(2) \\ b_2: & \mathsf{sendMsg}[\mathrm{grPerm}, a] \leftarrow \mathsf{recvMsg}[\mathrm{reqPerm}, a],\ (at(3) \wedge \neg req(2)) \vee up(2) \end{array}$$

If a receives a request from b to give permission for floor 2, and a is $at(1)$ and has itself received a request to serve floor 2, but it is b's turn, it grants the permission to b and changes the $turn$ fluent so that in the next competition for floor 2 it will be a's turn (and dually for b):

$$\begin{array}{rl} a_3: & \mathsf{sendMsg}[\mathrm{grPerm}, b],\ turn(a) \wedge \neg turn(b) \\ & \leftarrow \mathsf{recvMsg}[\mathrm{reqPerm}, b],\ at(1) \wedge req(2) \wedge turn(b) \\ b_3: & \mathsf{sendMsg}[\mathrm{grPerm}, a],\ turn(b) \wedge \neg turn(a) \\ & \leftarrow \mathsf{recvMsg}[\mathrm{reqPerm}, a],\ at(3) \wedge req(2) \wedge turn(a) \end{array}$$

If a receives a request from b to give permission for floor 2, and a is $at(1)$ and has itself received a request to serve floor 2, and it is indeed a's turn, it records the request in order to keep in mind that it has to be answered later on (when leaving the critical zone), and it changes the $turn$ fluent:

$$\begin{array}{rl} a_4: & permreq(b) \wedge turn(b) \wedge \neg turn(a) \\ & \leftarrow \mathsf{recvMsg}[\mathrm{reqPerm}, b],\ at(1) \wedge req(2) \wedge turn(a) \\ b_4: & permreq(a) \wedge turn(a) \wedge \neg turn(b) \\ & \leftarrow \mathsf{recvMsg}[\mathrm{reqPerm}, a],\ at(3) \wedge req(2) \wedge turn(b) \end{array}$$

If a is $at(2)$ or moving up from 1 to 2 when requested for permission, it is not able to grant permission but only to record the request for a later answer:

$$\begin{array}{rl} a_5: & permreq(b) \leftarrow \mathsf{recvMsg}[\mathrm{reqPerm}, b],\ at(2) \vee up(1) \\ b_5: & permreq(a) \leftarrow \mathsf{recvMsg}[\mathrm{reqPerm}, a],\ at(2) \vee down(3) \end{array}$$

If a is granted the permission by b to go to 2 when $at(1)$ with a pending service request $req(2)$, it moves up:

$$\begin{array}{rl} a_6: & \mathsf{do}(\text{mvup}),\ up(1) \wedge \neg at(1) \ \leftarrow\ \mathsf{recvMsg}[\text{grPerm}, b],\ at(1) \wedge req(2) \\ b_6: & \mathsf{do}(\text{mvdown}),\ down(3) \wedge \neg at(3) \ \leftarrow\ \mathsf{recvMsg}[\text{grPerm}, a],\ at(3) \wedge req(2) \end{array}$$

If a or b receives a service request when $at(2)$, it can immediately depart:

$$\begin{array}{rl} a_7: & \mathsf{do}(\text{mvdown}),\ req(1) \wedge down(2) \wedge \neg at(2) \ \leftarrow\ \mathsf{recvMsg}[\text{reqTo}(1)],\ at(2) \\ b_7: & \mathsf{do}(\text{mvup}),\ req(3) \wedge up(2) \wedge \neg at(2) \ \leftarrow\ \mathsf{recvMsg}[\text{req}(3)],\ at(2) \end{array}$$

If a or b receives a service request when $at(2)$, while there is a pending permission request, it grants permission:

$$\begin{array}{rl} a_8: & \mathsf{sendMsg}[\text{grPerm}, b],\ \neg permreq(b) \ \leftarrow\ \mathsf{recvMsg}[\text{req}(1)],\ at(2) \wedge permreq(b) \\ b_8: & \mathsf{sendMsg}[\text{grPerm}, a],\ \neg permreq(a) \ \leftarrow\ \mathsf{recvMsg}[\text{reqTo}(3)],\ at(2) \wedge permreq(a) \end{array}$$

Finally, if a or b has reached the floor for which it has a pending service request, it halts and updates its KB accordingly:

$$\begin{array}{rl} a_9, b_9: & \mathsf{do}(\text{halt}),\ at(x) \wedge \neg req(x) \wedge \neg up(x-1) \\ & \leftarrow\ \mathsf{recvMsg}[\text{arrAt}(x)],\ req(x) \wedge up(x-1) \\ a_{10}, b_{10}: & \mathsf{do}(\text{halt}),\ at(x) \wedge \neg req(x) \wedge \neg down(x+1) \\ & \leftarrow\ \mathsf{recvMsg}[\text{arrAt}(x)],\ req(x) \wedge down(x+1) \end{array}$$

In order to guarantee fairness of the system, we have to assume that whenever one of the elevators is $at(2)$ there will eventually be a request causing it to depart from floor 2. This could be achieved by appropriate timeouts, for instance.

4.4 Incomplete and Inconsistent Information

In all elevator examples we have tacitly used the knowledge system $\boldsymbol{A}$ of relational databases. This was sufficient for the simple forms of elevator knowledge where we do not have to deal with incomplete information. The following example illustrates the use of SL-factbases for domains where incomplete and inconsistent information has to be processed.

We show how the possible difference in the epistemic weight between a piece of information conveyed by inter-agent communication, and one obtained through perception may be captured in a SL-factbase. While the former is balanced against other incompatible information, the latter overrides incompatible information. The following rules express the epistemic reactions to the communication event *tell* and to the perception event *observed*:

$$\begin{array}{rrl} r_1: & F{:}Sender & \leftarrow\ \mathsf{recvMsg}[\text{tell}(F),\ Sender] \\ r_2: & F{:}vision & \leftarrow\ \mathsf{recvMsg}[\text{observed}(F),\ vision] \end{array}$$

We consider the following scenario. There are three agents: 007, his British secretary (s), and special agent Cooper (C). Cooper considers 007 as a pretty consistent liar, and consequently assigns a negative reliability degree to him: $\rho(007) = -0.5$. 007 knows that both his secretary and Cooper are truthful and very reliable, and consequently

$a_1:$	$\mathsf{sM}[\text{reqPerm}, b],\ req(2)\ \leftarrow\ \mathsf{rM}[\text{reqTo}(2)],\ at(1) \wedge \neg req(2)$
$b_1:$	$\mathsf{sM}[\text{reqPerm}, a],\ req(2)\ \leftarrow\ \mathsf{rM}[\text{reqTo}(2)],\ at(3) \wedge \neg req(2)$
$a_2:$	$\mathsf{sM}[\text{grPerm}, b]\ \leftarrow\ \mathsf{rM}[\text{reqPerm}, b],\ (at(1) \wedge \neg req(2)) \vee down(2)$
$b_2:$	$\mathsf{sM}[\text{grPerm}, a]\ \leftarrow\ \mathsf{rM}[\text{reqPerm}, a],\ (at(3) \wedge \neg req(2)) \vee up(2)$
$a_3:$	$\mathsf{sM}[\text{grPerm}, b],\ turn(a) \wedge \neg turn(b)\ \leftarrow\ \mathsf{rM}[\text{reqPerm}, b],\ at(1) \wedge req(2) \wedge turn(b)$
$b_3:$	$\mathsf{sM}[\text{grPerm}, a],\ turn(b) \wedge \neg turn(a)\ \leftarrow\ \mathsf{rM}[\text{reqPerm}, a],\ at(3) \wedge req(2) \wedge turn(a)$
$a_4:$	$permreq(b) \wedge turn(b) \wedge \neg turn(a)\ \leftarrow\ \mathsf{rM}[\text{reqPerm}, b],\ at(1) \wedge req(2) \wedge turn(a)$
$b_4:$	$permreq(a) \wedge turn(a) \wedge \neg turn(b)\ \leftarrow\ \mathsf{rM}[\text{reqPerm}, a],\ at(3) \wedge req(2) \wedge turn(b)$
$a_5:$	$permreq(b)\ \leftarrow\ \mathsf{rM}[\text{reqPerm}, b],\ at(2) \vee up(1)$
$b_5:$	$permreq(a)\ \leftarrow\ \mathsf{rM}[\text{reqPerm}, a],\ at(2) \vee down(3)$
$a_6:$	$\mathsf{do}(\text{mvup}),\ up(1) \wedge \neg at(1)\ \leftarrow\ \mathsf{rM}[\text{grPerm}, b],\ at(1) \wedge req(2)$
$b_6:$	$\mathsf{do}(\text{mvdown}),\ down(3) \wedge \neg at(3)\ \leftarrow\ \mathsf{rM}[\text{grPerm}, a],\ at(3) \wedge req(2)$
$a_7:$	$\mathsf{do}(\text{mvdown}),\ req(1) \wedge down(2) \wedge \neg at(2)\ \leftarrow\ \mathsf{rM}[\text{reqTo}(1)],\ at(2)$
$b_7:$	$\mathsf{do}(\text{mvup}),\ req(3) \wedge up(2) \wedge \neg at(2)\ \leftarrow\ \mathsf{rM}[\text{req}(3)],\ at(2)$
$a_8:$	$\mathsf{sM}[\text{grPerm}, b],\ \neg permreq(b)\ \leftarrow\ \mathsf{rM}[\text{req}(1)],\ at(2) \wedge permreq(b)$
$b_8:$	$\mathsf{sM}[\text{grPerm}, a],\ \neg permreq(a)\ \leftarrow\ \mathsf{rM}[\text{reqTo}(3)],\ at(2) \wedge permreq(a)$
$a_9, b_9:$	$\mathsf{do}(\text{halt}),\ at(x) \wedge \neg req(x) \wedge \neg up(x-1)$
	$\leftarrow\ \mathsf{rM}[\text{arrAt}(x)],\ req(x) \wedge up(x-1)$
$a_{10}, b_{10}:$	$\mathsf{do}(\text{halt}),\ at(x) \wedge \neg req(x) \wedge \neg down(x+1)$
	$\leftarrow\ \mathsf{rM}[\text{arrAt}(x)],\ req(x) \wedge down(x+1)$

Table 1: Two elevators in a shaft.

assigns $\rho(C) = \rho(s) = 0.9$. Both Cooper and 007 consider their visual perceptions as completely reliable: $\rho(vision) = 1$.

Being busy with the difficult 'Yale shooting problem', Cooper is is told by 007 that the turkey (t) is dead, leading to the following update of Coopers initial knowledge base $X_C^0 = \emptyset$:

$$X_C^1 = \mathsf{Upd}(X_C^0, dead(t) : 007) = \{dead(t) : 007\},$$

Initially, Cooper does not know whether the turkey is dead, i.e. $\mathsf{Ans}(X_C^0, dead(t)) =$ $\mathsf{unknown}$, but now, since he reckons with the fact that 007 frequently lies, he beliefs that the turkey is probably not dead:

$$\mathsf{Ans}(X_C^1, dead(t)) = \mathsf{yes} : -0.5$$

And indeed, Cooper manages the same day to observe the turkey near the main university building and updates his KB accordingly:

$$X_C^2 = \mathsf{Upd}(X_C^1, \neg dead(t) : vision) = \{\neg dead(t)\}$$
$$\mathsf{Ans}(X_C^2, dead(t)) = \mathsf{no}$$

Meanwhile 007 has been told by his secretary that the turkey is dead:

$$X_{007}^1 = \mathsf{Upd}(\emptyset, dead(t) : s) = \{dead(t) : s\}$$
$$\mathsf{Ans}(X_{007}^1, dead(t)) = \mathsf{yes} : 0.9$$

But then special agent Cooper tells him that the turkey is alive which leads to an implicit neutralization:

$$X_{007}^2 = \mathsf{Upd}(X_{007}^1, \neg dead(t) : C) = \{dead(t) : s,\ \neg dead(t) : C\}$$
$$\mathsf{Ans}(X_{007}^2, dead(t)) = \mathsf{yes\text{-}and\text{-}no} : 0.9$$

since both his secretary and Cooper have the same credibility for him.

This concludes our discussion of reasoning with incomplete and possibly inconsistent information. We now define the formal concept of a *reagent*, i.e. a purely reactive vivid agent.

5 Specification, Execution and Verification of Reagents

In our model, a reagent $\mathcal{A} = \langle X, EQ, RR\rangle$ on the basis of a knowledge system $\boldsymbol{K} = \langle L_{\mathrm{KB}}, \vdash, L_{\mathrm{Query}}, \mathsf{Upd}, L_{\mathrm{Input}}\rangle$, and event and action languages L_{PEvt}, L_{CEvt}, and L_{Act} consists of

(X) a knowledge base $X \in L_{\mathrm{KB}}$,

(EQ) an event queue EQ being a list of instantiated event expressions from L^0_{Evt}, and

(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*, consisting of epistemic and physical reaction and interaction rules which code the reactive and communicative behavior of the agent.[8]

[8]We assume that RR is a consistent encoding of reactive behavior in the sense that whenever a set of reaction rules is triggered by an event, the resulting actions are compatible with each other, i.e. the epistemic effects associated with them do not cancel out each other.

A multi-reagent system (MRS) is a tuple of reagents:

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

The state of a reagent $\mathcal{A}_i$ consists of its knowledge state X_i and its event queue EQ_i, i.e. it is a pair $A_i = \langle X_i, EQ_i \rangle$. The global state S of a MRS is the corresponding tuple of agent states: $S = \langle A_1, \ldots, A_n \rangle$.

We use the Prolog style list notation for event queues: $EQ = [\varepsilon | RestQ]$, where the head of the list, $\varepsilon = head(EQ)$, denotes the current event, and $RestQ = tail(EQ)$ denotes the tail of EQ. We write $EQ_1 + EQ_2$ for the concatenation of EQ_1 and EQ_2, and $EQ + \varepsilon$ instead of $EQ + [\varepsilon]$.

5.1 Operational Semantics of Reaction Rules

The operational semantics of reaction rules

$$\begin{array}{rcl} E & \leftarrow & \mathsf{rM}[\varepsilon(U), S],\ C \\ \mathsf{do}[\alpha(V)],\ E & \leftarrow & \mathsf{rM}[\varepsilon(U), S],\ C \\ \mathsf{sM}[\eta(V), R],\ E & \leftarrow & \mathsf{rM}[\varepsilon(U), S],\ C \end{array}$$

where U and V are suitable lists of parameters, and rM (and sM) abbreviate $\mathsf{sendMsg}$ (and $\mathsf{recvMsg}$), is defined in two steps. First, it has to be defined how reaction rules are triggered by events leading to a set of corresponding action rules (without an event condition). Then, it has to be defined how such a set of action rules is executed, i.e. how it changes the system state.

The concurrent execution of a consistent set of action rules A in the knowledge state X yields the update

$$A(X) = \mathsf{Upd}(X,\ \{E\sigma : (E \leftarrow C) \in \dot{A}\ \ \&\ \ X \vdash C\sigma\})$$

where $\dot{A} = \{\dot{r} : r \in A\}$, and σ is any substitution instantiating all free variables of C. In addition, we need the set of all communication events created by a set of action rules A which are applicable in X, and addressed to agent j

$$CE_j(A, X) = \{\eta(V\sigma)\ :\ (\mathsf{sM}[\eta(V), R], E \leftarrow C) \in A\ \ \&\ \ X \vdash C\sigma\ \ \&\ \ R\sigma = j\}$$

We can now define the concurrent execution of a set of actions by agent $\mathcal{A}_i$ as the application of the corresponding set of action rules A to the MRS state $\langle A_1, \ldots, A_n \rangle$ at component i:

$$A(i,\ \langle A_1, \ldots, A_n \rangle) = \langle A'_1, \ldots, A'_n \rangle$$

where

$$\begin{array}{rcll} A'_i & = & \langle A(X_i),\ EQ_i + CE_i(A, X_i) \rangle & \\ A'_j & = & \langle X_j,\ EQ_j + CE_j(A, X_i) \rangle & \quad (j \neq i) \end{array}$$

We assume here that the set of concurrent communication events resulting from a reaction is serializable in the sense that the order of their arrival at an event queue does not matter. If A is a singleton, $A = \{r\}$, we simply write $r(i, \langle A_1, \ldots, A_n \rangle)$ instead of $A(i, \langle A_1, \ldots, A_n \rangle)$.

Given a set of reaction rules RR, the set of action rules triggered by the event $\langle\varepsilon(u), s\rangle$ is denoted $RR^{\varepsilon(u),s}$, where

$$\begin{array}{rcl} RR^{\varepsilon(u),s} & = & \{A\sigma, E\sigma \leftarrow C\sigma \;\; : \;\; (A, E \leftarrow \mathsf{rM}[\varepsilon(U), S], C) \in RR \\ & & \qquad\qquad\qquad\quad \& \;\; u = U\sigma_0 \;\; \& \;\; \sigma = \sigma_0 \cup \{S/s\}\} \end{array}$$

and A stands for a physical or communicative action, i.e. $A\sigma = \mathsf{do}[\alpha(V\sigma)]$, or $A\sigma = \mathsf{sM}[\eta(V\sigma), R\sigma]$. Referring to a specific agent i with reaction rule set RR_i, we write RR_i^{Evt} instead of $(RR_i)^{Evt}$.

Finally, the concurrent execution of all reactions of agent i triggered by an event $Evt = \langle\varepsilon(u), s\rangle$ corresponds to a transition of $\mathcal{S}$, denoted $React(i, S)$, which is given by the application of the set of action rules RR_i^{Evt} at component i:

$$\begin{array}{rcl} \mathrm{React}(i, S) & = & \mathrm{React}(i,\, \langle \ldots, \langle X_i,\, [Evt|RestQ]\rangle, \ldots\rangle) \\ & = & RR_i^{Evt}(i,\, \langle \ldots, \langle X_i,\, RestQ\rangle, \ldots\rangle) \end{array}$$

In the case of a single-reagent system $\mathcal{A}$, with state $A = \langle X, EQ\rangle$, we omit subscripts and simply write React(A).

Example 3 *In the reactive elevator example above, the reactive behavior is specified by $RR = \{r_1, \ldots, r_5\}$, yielding the following transitions:*

$$\begin{array}{rcl} \mathrm{React}(\{at(3)\},\, [\mathrm{reqTo}(5)]) & = & \langle RR^{\mathrm{req}(5)}(\{at(3)\}),\, []\rangle \\ & = & \langle \dot{r}_1(\{at(3)\}),\, []\rangle = \langle\{req(5),\, up(3)\},\, []\rangle \\ \mathrm{React}(\{req(5),\, up(3)\},\, [\mathrm{arrAt}(4)]) & = & \langle RR^{\mathrm{arrAt}(4)}(\{req(5),\, up(3)\}),\, []\rangle \\ & = & \langle \dot{r}_3(\{req(5),\, up(3)\}),\, []\rangle = \langle\{req(5),\, up(4)\},\, []\rangle \\ \mathrm{React}(\{req(5),\, up(4)\},\, [\mathrm{arrAt}(5)]) & = & \langle RR^{\mathrm{arrAt}(5)}(\{req(5),\, up(4)\}),\, []\rangle \\ & = & \langle \dot{r}_5(\{req(5),\, up(4)\}),\, []\rangle = \langle\{at(5)\},\, []\rangle \end{array}$$

5.2 Defining the Execution of Reagents by Meta-Programming

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. While the treatment of actions in [Kow95] is based on the *event calculus* of [KS86], we propose the more operational formalism of action and reaction rules.

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
```

The reactive behavior of each agent is specified by a set of reaction rules represented as triples $\langle Act/Eff, Evt, Cond\rangle$ in the table *reaction*. 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 defined on the basis of our knowledge system concepts of inference and update:

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

5.3 Reagent Systems as Transition Systems

Assuming the principle of *nondeterministic interleaving* of (possibly concurrent) state changing events, the temporal behavior of a MRS can be described by means of a *labelled transition system*, where we have two kinds of transitions transforming a state $S = \langle A_1, \ldots, A_n\rangle$ with $A_i = \langle X_i, EQ_i\rangle$:

(Perception) An incoming message ε representing a perception event from perception subsystem j, and received by agent i, yields the transition $S \xrightarrow{i,\varepsilon,j} S'$, where $S' = S$, except that $EQ_i' = EQ_i + \langle \varepsilon, j\rangle$.

(Reaction) A reaction of agent i in response to an event $Evt = head(EQ_i)$ yields the transition $S \xrightarrow{i,RR_i^{Evt}} S'$, where $S' = \mathrm{React}(i, S)$. If the only applicable reaction rule of the triggered rule set RR_i^{Evt} is r, we simply write $S \xrightarrow{i,r} S'$.

Notice that we assume that all transitions are *atomic*. That is, once an agent is getting involved in a transition, other agents cannot influence the transition or observe intermediate points of it. In combination with the interleaving principle this implies that the simultaneous execution of actions is *serializable* in the following sense: if two elementary actions, say α_1 and α_2 are executed concurrently, then the net effect is either that of α_1 followed by α_2, or α_2 followed by α_1. The gain of the interleaving assumption consists in the reduction of the number of possible execution histories to consider in correctness proofs. In general, however, there may be agent programs which cannot be serialized.

Definition 3 (Execution History) *An* execution history *of a MRS $\mathcal{S}$ is a chain of state transitions $S^0 \xrightarrow{\tau_0} S^1 \xrightarrow{\tau_1} \ldots$, where each τ_i corresponds to one of the transitions listed above. A history can be finite or infinite. By definition, a finite history ends in a state. The set of all histories of $\mathcal{S}$ is denoted by* $\mathrm{Hist}(\mathcal{S})$.

The states S^i are linearly ordered in a history $H \in \mathrm{Hist}(\mathcal{S})$. We write $S \in H$ if S occurs in H, i.e. if $S = S^i$ for some $i \geq 0$. We write $S \leq S'$, if $S = S^i$, $S' = S^j$, and $i \leq j$.

Definition 4 (Enabled Reagent) *We say that a reagent $\mathcal{A}_i$ is* enabled *if its event queue is nonempty, $EQ_i \neq []$.*

Definition 5 (Fairness) *A finite history of a multi-reagent system $\mathcal{S}$ is called* fair,[9] *if in its final state no reagent is enabled. An infinite history H of $\mathcal{S}$ is called* fair, *if for all reagents $\mathcal{A}_i$, either H contains an infinite number of $\mathcal{A}_i$-reaction transitions, or it contains an infinite number of states where $\mathcal{A}_i$ is not enabled.*

We denote the set of all fair histories of $\mathcal{S}$ by $\mathrm{FHist}(\mathcal{S})$.

5.4 Assertional Reasoning

Assertional reasoning about multi-agent systems allows to specify and prove *safety* and *progress* properties of such systems. As concurrent systems, multi-agent systems are much more complex, and harder to understand, than sequential programs. We show that the method of assertional reasoning, originally developed for sequential programs by Floyd, Hoare, and Dijkstra, can be used to analyze the dynamics of multi-agent systems.

Definition 6 ($\mathcal{S}$-Queries) *Let $\mathcal{S}$ be a MRS, and let $L' = \{\mathsf{B}_i F : F \in L_{\mathrm{Query}}\} \cup \{\mathsf{rM}_i[\varepsilon] : \varepsilon \in L_{\mathrm{Evt}}\}$. Then, $L_{\mathcal{S}\mathrm{Q}} = L'(\neg, \wedge, \vee)$ denotes the set of $\mathcal{S}$-query formulas.*

Inference for $\mathcal{S}$-queries on the basis of a state $S = \langle A_1, \ldots, A_n \rangle$ with $A_i = \langle X_i, EQ_i \rangle$, is defined as follows.

$$
\begin{array}{lcllcl}
S & \vdash & \mathsf{B}_i F & \text{iff} & X_i \vdash F \\
S & \vdash & \neg \mathsf{B}_i F & \text{iff} & X_i \not\vdash F \\
S & \vdash & \mathsf{rM}_i[\varepsilon] & \text{iff} & \varepsilon = head(EQ_i) \\
S & \vdash & \neg \mathsf{rM}_i[\varepsilon] & \text{iff} & \varepsilon \neq head(EQ_i)
\end{array}
$$

[9] This is sometimes also called *weak fairness* as opposed to a stronger notion of fairness which we shall not consider.

Definition 7 (Hoare Triples) *Let τ be a transition of $\mathcal{S}$, and let $\phi, \psi \in L^0_{\mathcal{S}\mathrm{Q}}$ be $\mathcal{S}$-queries. A* Hoare triple *has the form $\phi\{\tau\}\psi$, and the following meaning: for any state S, if $S \vdash \phi$, then $\tau(S) \vdash \psi$, where $\tau(S)$ is the state obtained by applying the transition τ to S.*

In order to be able to prove correctness assertions with respect to a behavior specification, we define two operators Inv, and $\leadsto$, which represent a fragment of temporal logic.[10] Assertions about multi-reagent systems refer only to fair histories.

Definition 8 (Invariance and Leads-To Assertions) *Let $H \in \mathrm{Hist}(\mathcal{S})$, and let $\phi, \psi \in L^0_{\mathcal{S}\mathrm{Q}}$. We say that H satisfies the* invariance assertion $\mathsf{Inv}(\phi)$, *symbolically $H \models \mathsf{Inv}(\phi)$, iff for all $S \in H$, $S \vdash \phi$. We say that H satisfies the* leads-to assertion *$\phi \leadsto \psi$, symbolically $H \models \phi \leadsto \psi$, if for all states $S \in H$ with $S \vdash \phi$, there is a later state $S' \in H$, $S \leq S'$, such that $S' \vdash \psi$. An assertion ϕ is entailed by $\mathcal{S}$, symbolically $\mathcal{S} \models \phi$, if for all fair histories $H \in \mathrm{FHist}(\mathcal{S})$, $H \models \phi$.*

Definition 9 (Multi-Reagent Domain Description) *A multi-reagent domain description consists of a MRS specification $\mathcal{S}$, and a set of assertions Φ expressing assumptions about the environment of the agents. Such a domain description entails an assertion ϕ, if ϕ is satisfied by all fair histories of $\mathcal{S}$ satisfying Φ. Formally,*

$$\mathcal{S}, \Phi \models \phi :\Longleftrightarrow \textit{for all } H \in \mathrm{FHist}(\mathcal{S}), \textit{ if } H \models \Phi, \textit{ then } H \models \phi.$$

We briefly state the soundness of several basic proof rules for assertional reasoning.

Claim 1 (INV) *Let $\mathcal{S}$ be a MRS, such that $S^0 = \langle A^0_1, \ldots, A^0_n \rangle$, with initial states $A^0_i = \langle X^0_i, EQ^0_i \rangle$. Then, $\mathcal{S} \models \mathsf{Inv}(\phi)$, if $S^0 \vdash \phi$, and for every transition τ of $\mathcal{S}$, it holds that $\phi\{\tau\}\phi$.*

Claim 2 (LT1) *$\mathcal{S} \models \phi \leadsto \psi$, if $\phi\{\tau\}\psi$ holds for every transition τ of $\mathcal{S}$.*

Further proof rules, such as the following, allow to infer leads-to assertions from other assertions:

$$\text{(Transitivity)} \quad \frac{\phi \leadsto \chi \quad \chi \leadsto \psi}{\phi \leadsto \psi} \qquad \text{(Left-Or)} \quad \frac{\phi \leadsto \chi \quad \psi \leadsto \chi}{\phi \vee \psi \leadsto \chi}$$

For a comprehensive discussion of such rules, see [MP92].

The following rule for proving leads-to assertions by wellfounded iteration of other proof rules is also well-known from the literature.[11]

Claim 3 (LT2) *Let $\langle W, < \rangle$ be a wellfounded ordering. Then, $\mathcal{S} \models \phi \leadsto \psi$, if*

$$\begin{array}{ll} (i) & \mathcal{S} \models \phi \leadsto \psi \vee \chi(m) \quad \textit{for some } m \in W \\ (ii) & \mathcal{S} \models \chi(m) \leadsto \psi \vee \chi(n) \quad \textit{for some } n < m \end{array}$$

[10] See [MP92], where $\Box\phi$ corresponds to $\mathsf{Inv}(\phi)$, and $\Box(\phi \supset \Diamond\psi)$ corresponds to $\phi \leadsto \psi$.

[11] See, e.g., [Sha93].

5.5 Examples of Verification

Suppose we want to prove that the single elevator system $\mathcal{A}$ from 4.1, with $A_0 = \langle X_0, EQ_0\rangle = \langle\{at(m)\}, []\rangle$, never attempts to move downwards from the ground floor. This safety property can be expressed by the following invariance assertion.

Observation 1 $\mathcal{A} \models \mathsf{Inv}(\neg down(0))$.

Proof: Obviously, $A_0 \vdash \neg down(0)$. It remains to show for every reaction rule r_i $(i = 1, \ldots, 5)$, that $\neg down(0)\,\{r_i\}\,\neg down(0)$. Since $A \vdash \neg down(0)$ iff $down(0) \notin X$, we have to show that $down(0) \notin X$ implies $down(0) \notin r_i(X)$. This is obvious for r_1, r_3, and r_5, since these transitions do not add any information about *down*. Suppose that $down(0) \in r_2(X)$, implying that for some x,

$$A \vdash \mathsf{rM}[\mathrm{reqTo}(x)] \wedge x < 0 \wedge x > 0$$

which is a contradiction. Finally, suppose that $down(0) \in r_4(X)$, implying that

$$A \vdash \mathsf{rM}[\mathrm{arrAt}(0)] \wedge 0 > 0$$

which is again a contradiction. $\Box$

The progress property that the elevator when currently halting at floor m will eventually serve every request is expressed by $req(k) \wedge at(m) \leadsto at(k)$. Since this is trivially the case for $k = m$, we only have to prove it for $k < m$ and $k > m$. In both cases the elevator will start to move towards the requested floor, either downwards or upwards, and it is therefore sufficient to prove the following property.

Observation 2 $\mathcal{A}, \Phi \models req(k) \wedge (up(m) \vee down(m)) \leadsto at(k)$.

Proof: We consider only the case of moving down. For $k < m$, it follows from

$$\begin{array}{rcl} \Phi & = & \{up(k) \leadsto \mathsf{rM}[\mathrm{at}(k+1)],\ down(k) \leadsto \mathsf{rM}[\mathrm{at}(k-1)]\,\} \\ \mathcal{A}, \Phi & \models & req(k) \wedge down(m) \leadsto at(k) \vee down(m-1) \\ \mathcal{A}, \Phi & \models & down(m) \leadsto at(k) \vee down(m-1) \end{array}$$

by the proof rule (LT2) that $\mathcal{A}, \Phi \models req(k) \wedge down(m) \leadsto at(k)$. $\Box$

In order to prove correctness properties of the two-elevator system $\mathcal{S}$, we need the following observation about the possible states of an elevator:

$$\mathcal{S} \models \mathsf{Inv}[(\mathsf{B}_a at(1) | \mathsf{B}_a up(1) | \mathsf{B}_a at(2) | \mathsf{B}_a down(2)) \wedge \neg \mathsf{B}_a up(2) \wedge \neg \mathsf{B}_a at(3) \wedge \neg \mathsf{B}_a down(3)]$$

A dual version of this observation holds for elevator b. By the exclusive disjunction $\mathsf{B}_a at(1) | \mathsf{B}_a up(1) | \mathsf{B}_a at(2) | \mathsf{B}_a down(2)$, describing the definite state of elevator a, we can infer, for instance, that $X_a^k \vdash \neg up(1) \wedge \neg at(2)$ if $X_a^k \vdash at(1) \vee down(2)$, where k is any stage in the execution history of $\mathcal{S}$. .

We might want to prove *mutual exclusion* for the communicating elevators system, i.e. the safety property that it is never the case that the lower elevator goes up to the second floor, or is currently halting at the second floor, while the upper one goes down to the second floor, or is currently halting at the second floor, expressed as follows.

Observation 3 $\mathcal{S} \models \mathsf{Inv}(-[(\mathsf{B}_a up(1) \vee \mathsf{B}_a at(2)) \wedge (\mathsf{B}_b down(3) \vee \mathsf{B}_b at(2))])$

Proof: According to the proof rule (INV) we have to show that for every fair history $H = \langle S^0, S^1, S^2, \ldots \rangle$, where $S^i = \langle A_a^i, A_b^i \rangle$ with $A_j^i = \langle X_j^i, EQ_j^i \rangle$, it holds that for

$$F = (\neg \mathsf{B}_a up(1) \wedge \neg \mathsf{B}_a at(2)) \vee (\neg \mathsf{B}_b down(3) \wedge \neg \mathsf{B}_b at(2))$$

we have $S^0 \vdash F$, and $S^{k+1} \vdash F$ if $S^k \vdash F$. Notice that $S^k \vdash F$ iff $X_a^k \vdash F_a$ or $X_b^k \vdash F_b$, where $F_a = \neg up(1) \wedge \neg at(2)$, and $F_b = \neg down(3) \wedge \neg at(2)$.

It is easy to see that $X_a^0 \vdash F_a$, and consequently $S^0 \vdash F$. We assume now that $S^k \vdash F$, i.e. $X_a^k \vdash F_a$ or $X_b^k \vdash F_b$.

Case A: $X_a^k \vdash F_a$, implying $X_a^k \vdash at(1)|down(2)$. Since a perception transition does not change the beliefs of an agent, it is sufficient to consider reactions only.

Case Aa: $X_a^k \vdash at(1)$. The possible reactions are a_1, a_2, a_3, a_4, and a_6. For any $i = 1, 2, 3, 4$, and $S^k \xrightarrow{a_i} S^{k+1}$, it holds that $X_a^{k+1} \vdash at(1)$, and consequently $X_a^{k+1} \vdash F_a$. For $S^k \xrightarrow{a_6} S^{k+1}$, we obtain $X^{k+1} \vdash at(1)$ or $X^{k+1} \vdash up(1)$. The former implies again that $X^{k+1} \vdash F_a$, and so it remains to check the latter case which may arise in three ways: $X^{k+1} \vdash up(1)$ iff $S^k \vdash \mathsf{rM}_a[grPerm, b]$ iff $S^{k-1} \xrightarrow{b_2} S^k$ or $S^{k-1} \xrightarrow{b_3} S^k$ or $S^{k-1} \xrightarrow{b_7, b_8} S^k$.

Case Aa1: If $S^{k-1} \xrightarrow{b_2} S^k$, then $X_b^k \vdash at(3) \vee up(2)$, and consequently $X_b^{k+1} \vdash F_b$.

Case Aa2: If $S^{k-1} \xrightarrow{b_3} S^k$, then $X_b^k \vdash at(3)$, and consequently $X_b^{k+1} \vdash F_b$.

Case Aa3: If $S^{k-1} \xrightarrow{b_7, b_8} S^k$, then $X_b^k \vdash up(2)$, and consequently $X_b^{k+1} \vdash F_b$.

Case Ab: $X_a^k \vdash down(2)$ entailing the possible reactions a_2 and a_{10}. If $S^k \xrightarrow{a_2} S^{k+1}$, then $X_a^{k+1} \vdash down(2)$, and consequently $X_a^{k+1} \vdash F_a$. If $S^k \xrightarrow{a_{10}} S^{k+1}$, then $X_a^{k+1} \vdash at(1)$, and consequently $X_a^{k+1} \vdash F_a$.

Case B: $X_b^k \vdash F_b$, implying $X_b^k \vdash at(3)|up(2)$. The proof of case B is dually analogous to case A. □

Observation 4 *Also we might want to prove for $x = a, b$ that*

$$\mathcal{S}, \Phi \models \mathsf{rM}_x[\text{reqTo}(2)] \leadsto \mathsf{B}_x at(2)$$

expressing the progress property that both elevators will eventually serve every request from floor 2. Here we need the following environment assumptions:

$$\begin{aligned} \Phi \;=\; & \{\mathsf{B}_a up(1) \leadsto \mathsf{rM}_a[\text{arrAt}(2)],\ \mathsf{B}_b up(2) \leadsto \mathsf{rM}_b[\text{arrAt}(3)]\} \\ & \cup \{\mathsf{B}_a down(2) \leadsto \mathsf{rM}_a[\text{arrAt}(1)],\ \mathsf{B}_b down(3) \leadsto \mathsf{rM}_b[\text{arrAt}(2)]\} \end{aligned}$$

guaranteeing that the environment correctly informs the elevator about its arrival at every floor. Notice that the fairness assumption implies that whenever one of the elevators is halting at the critical floor, it will eventually depart, requiring that $\mathcal{S}, \Phi \models \mathsf{B}_a at(2) \leadsto \mathsf{rM}_a[\text{reqTo}(1)]$, *and* $\mathcal{S}, \Phi \models \mathsf{B}_b at(2) \leadsto \mathsf{rM}_b[\text{reqTo}(3)]$.

6 Vivid Agents

While reagents are only able to react in response to perception and communication events, full agents can in addition generate and execute plans in order to solve certain tasks assigned at design time, or communicated at run-time.

6.1 Planning

We distinguish between normal and permanent tasks, both of which require planning. While normal tasks exist only temporarily until they have been carried out, permanent tasks persist and have to be re-solved over and over again. Tasks are initially assigned to an agent at design time, or communicated to it at run-time.

In order to solve a task $G \in L^0_{\text{Query}}$, in a situation described by $X_0 \in L_{\text{KB}}$, an agent generates a suitable plan P being a sequence of instantiated action rules $r_1(\vec{c_1}), \ldots, r_n(\vec{c_n})$, such that when the corresponding sequence of actions is performed in X_0, it leads to a situation $P(X_0) \in L_{\text{KB}}$ where G holds:

$$P = r_n(\vec{c_n}) \circ \ldots \circ r_1(\vec{c_1}) \ , \quad \text{and} \quad P(X_0) \vdash G$$

where P is applied to X_0 as a composed function.

Example 4 (The Planning Elevator) *Let the action rules r_u and r_d represent the elevator actions oneup and onedown from example 2, and let the current situation be described by $X_0 = \{at(3)\}$. If the elevator has to perform the task $at(7)$ in order to pick up a passenger at the seventh floor, it could generate the following (admittedly not very exciting, but optimal) plan:*

$$P = r_u(6) \circ r_u(5) \circ r_u(4) \circ r_u(3)$$

the execution of which solves the task:

$$P(X_0) = r_u(6)(r_u(5)(r_u(4)(r_u(3)(\{at(3)\})))) = \{at(7)\}$$

Notice that our concept of planning on the basis of action rules in knowledge systems can be viewed as a generalization of the STRIPS paradigm.[12] Therefore, the frame problem is solved in the same way as in STRIPS: by means of a *minimal change* policy incorporated in the update operation of a knowledge system.

6.2 Specification of Vivid Agents

In this section, we define our concept of a vivid agent which extends that of a reagent by adding tasks and intentions, and allowing for planning and plan execution.

On the basis of a knowledge system $\boldsymbol{K} = \langle L_{\text{KB}}, \vdash, L_{\text{Query}}, \mathsf{Upd}, L_{\text{Input}} \rangle$, and event and action languages L_{PEvt}, L_{CEvt}, and L_{Act}, a vivid agent is specified as a 4-tuple $\mathcal{A} = \langle M, EQ, AR, RR \rangle$, consisting of

(M) a *mental state* $M = \langle X, TL, PTL, CI \rangle$, comprising

- (a) a knowledge base $X \in L_{\text{KB}}$ representing its *beliefs*,
- (b) a set of *tasks* $TL \subseteq L^0_{\text{Query}}$ the agent has accepted to carry out, and a set of permanent tasks $PTL \subseteq L^0_{\text{Query}}$ which have to be checked and carried out continuously,
- (c) a set of *current intentions*, i.e. goal/plan pairs, $CI = \{G_1/P_1, \ldots, G_n/P_n\}$, where $P_i \in AR^*$ is a plan to achieve $G_i \in L^0_{\text{Query}}$;

[12] The STRIPS system of [FN71] corresponds to planning on the basis of $\boldsymbol{A}$, the knowledge system of relational databases.

(EQ) an *event queue* EQ recording perception events, i.e. EQ is a sequence of elements from L^0_{PEvt},

(AR) a set AR of *action rules* describing the available action types which together with the task lists TL and PTL and the current intentions CI are the basis for the proactive behavior (i.e. deliberative planning and plan execution), and

(RR) a set RR of *reaction rules* encoding the reactive and communicative behavior.

We extend the query and the input language of the underlying knowledge system in order to allow also for queries and inputs concerning the tasks and the current intentions of an agent.

Definition 10 (Mental Queries) *Let $F \in L_{\mathrm{Query}}$. The mental query language L_{MQ} is defined as the smallest superset of L_{Query} containing $\mathsf{T}F$, $\mathsf{PT}F$, and $\mathsf{I}F$, and being closed with respect to formula formation under $\neg$, $\wedge$, and $\vee$.*

Inference for mental queries on the basis of a mental state $M = \langle X, TL, PTL, CI\rangle$ is defined as follows:

$$
\begin{array}{lclll}
M & \vdash & F & \text{iff} & X \vdash F \\
M & \vdash & \mathsf{T}F\ (\mathsf{PT}F) & \text{iff} & F \in TL\ \ (F \in PTL) \\
M & \vdash & \neg\mathsf{T}F\ (\neg\mathsf{PT}F) & \text{iff} & F \notin TL\ \ (F \notin PTL) \\
M & \vdash & \mathsf{I}G & \text{iff} & G/P \in CI \text{ for some plan } P \\
M & \vdash & \neg\mathsf{I}G & \text{iff} & \text{there is no } P\text{, s.th. } G/P \in CI
\end{array}
$$

Definition 11 (Mental Inputs) *Let $F \in L_{\mathrm{Query}}$. The mental input language L_{MI} is defined as the smallest superset of L_{Input} containing $\mathsf{T}F$, $\neg\mathsf{T}F$, $\mathsf{PT}F$, $\neg\mathsf{PT}F$, and $\neg\mathsf{I}F$, and being closed with respect to formula formation under $\wedge$.*

Updating with mental inputs concerns beliefs, tasks, and intentions. While tasks can be added and retracted through mental input, current intentions, which arise internally through goal selection and planning, can only be dropped but not added. Let $F \in L_{\mathrm{Input}}$, and $G \in L_{\mathrm{Query}}$. Then,

$$
\begin{array}{rcl}
\mathsf{Upd}(M, F) & = & \langle \mathsf{Upd}(X, F), TL, PTL, CI\rangle \\
\mathsf{Upd}(M, \mathsf{T}G) & = & \langle X, TL + G, PTL, CI\rangle \\
\mathsf{Upd}(M, \neg\mathsf{T}G) & = & \langle X, TL - G, PTL, CI\rangle \\
\mathsf{Upd}(M, \mathsf{PT}G) & = & \langle X, TL, PTL + G, CI\rangle \\
\mathsf{Upd}(M, \neg\mathsf{PT}G) & = & \langle X, TL, PTL - G, CI\rangle \\
\mathsf{Upd}(M, \neg\mathsf{I}\, G) & = & \langle X, TL, PTL, CI - G/P\rangle
\end{array}
$$

where $TL \pm G$, $PTL \pm G$, and $CI - G/P$, i.e. the result of adding/retracting G from the resp. set, is defined in the obvious way. We can now extend the form of action and reaction rules by allowing for mental queries as preconditions, and mental inputs as the effects of actions and reactions. Thus, action rules AR correspond to 3-tuples, and reaction rules RR to 4-tuples of the following form

$$
\begin{array}{l}
AR \subseteq (L_{\mathrm{Act}} \cup L_{\mathrm{CEvt}}) \times L_{\mathrm{MI}} \times L_{\mathrm{MQ}} \\
RR \subseteq (L_{\mathrm{Act}} \cup L_{\mathrm{CEvt}}) \times L_{\mathrm{MI}} \times L_{\mathrm{Evt}} \times L_{\mathrm{MQ}}
\end{array}
$$

The triple $\langle TL, PTL, CI\rangle$ is called *intentional state*. The *agent state* consist of the mental state and the event queue of the agent, i.e. it is a pair $\langle M, EQ\rangle$.

In the next sections, we shall show that agent specifications $\langle M, EQ, AR, RR\rangle$ are directly executable.

6.3 Defining the Execution of Vivid Agents by Meta-Programming

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

```
cycle([ KB, [], PTL, [] ])          % PLANNING for permanent tasks
← newEvent( noEvt)
   member( G, PTL),
   not demo( KB, G),                % check if goal is not satisfied
   plan( P, KB, G),                 % generate a plan for it
   cycle([ KB, [], PTL, [G/P] ]).

cycle([ KB, [G | RestTL], PTL, [] ])          % PLANNING for temporary tasks
← newEvent( noEvt)
   not demo( KB, G),                % check if goal is not satisfied
   plan( P, KB, G),                 % generate a plan for it
   cycle([ KB, [RestTL], PTL, [G/P] ]).

cycle([ KB, TL, PTL, [G/[R] | RestCI] ])        % EXECUTING the last action of a plan
← newEvent( noEvt)
   action( R, Act, Eff, Cond),      % 'R' is the action rule name.
   demo( KB, Cond),
   epistemic_assimilate( Eff, KB, KB′),
   execute( Act),
   cycle([ KB′, TL, PTL, RestCI ]).

cycle([ KB, TL, PTL, [G/[R|RestP] | RestCI] ])        % PLAN EXECUTION
← newEvent( noEvt)
   action( R, Act, Eff, Cond),      % 'R' is the action rule name.
   demo( KB, Cond),
   epistemic_assimilate( Eff, KB, KB′),
   execute( Act),
   cycle([ KB′, TL, PTL, [G/RestP|RestCI] ]).

cycle([ KB, TL, PTL, [G/[R|RestP] | RestCI] ])        % REPLANNING
← newEvent( noEvt)
   action( R, Act, Eff, Cond),
   not demo( KB, Cond),
   plan( P, KB, Cond),
   append( P, [R|RestP], NewPlan),
   cycle([ KB, TL, [G/NewPlan|RestCI] ]).
```

Notice that the above five cycle clauses for planning, plan execution and re-planning only apply if there is no event in the event queue. If there is an event, the following cycle clauses define the reactive behavior of the agent like in the case of reagents except that reactions now may affect the entire mental state of an agent which was confined to beliefs in the case of reagents.

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

perform( [], M, M).        % all reactions performed

perform( [Act/Eff | ActEffs], M, M′)
← execute( Act),
   assimilate( Eff, M, M1),
   perform( ActEffs, M1, M′).
```

The first five cycle clauses (planning and plan execution) are responsible for the proactive behavior of the agent. Plans are generated in order to solve tasks. If all events have been processed, and the current plan is nonempty (i.e. there are still planned actions not yet performed), plan execution is invoked. For simplicity, we do not consider the case where no plan can be found, and where the agent is therefore not able to achieve its goals.

If all events have been processed, and the current plan is empty (i.e. it has been fully executed), planning starts (i.e. there is time now for deliberation). This leads to the construction of a new plan to be added to the list of current intentions.

Notice that this high-level specification is not committed to a particular planning mechanism. It only supposes that there is a meta-predicate $plan(P, \mathrm{KB}, G)$ expressing that P is a plan to achieve goal G on the basis of KB. For real-time behavior, there should be some mechanism to interrupt planning in a logically controlled way whenever new events, requiring timely reaction, happen.

6.4 Multi-Agent Systems as Transition Systems

A closed multi-agent system $\mathcal{S}$ is a tuple of agents:

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

The state of agent $\mathcal{A}_i$ is denoted by $A_i = \langle M_i, EQ_i \rangle$. The global state S of a multi-agent system $\mathcal{S}$ is the corresponding tuple of agent states: $S = \langle A_1, \ldots, A_m \rangle$.

The application of a set of action rules A to the mental state M yields the update

$$A(M) = \mathsf{Upd}(M, \{E\sigma : E \leftarrow C \in \dot{A} \;\; \& \;\; M \vdash C\sigma\})$$

where $\dot{A} = \{\dot{r} : r \in A\}$. Likewise, we define the set $CE_j(A, M)$ of all communication events created by a set of action rules A applicable in M and addressed to agent j in the same way as for reagents (see 5.1). We can then define the concurrent execution of a set of actions by agent $\mathcal{A}_i$ as the application of the resp. set of action rules A to the multi-agent system state S at component i:

$$A(i,\, \langle A_1, \ldots, A_m\rangle) = \langle A'_1, \ldots, A'_m\rangle$$

where

$$\begin{aligned} A'_i &= \langle A(M_i),\, EQ_i + CE_i(A, M_i)\rangle \\ A'_j &= \langle M_j,\, EQ_j + CE_j(A, M_i)\rangle \qquad (j \neq i) \end{aligned}$$

If A is a singleton, $A = \{r\}$, we again simply write $r(i,S)$ instead of $A(i,S)$.

Definition 12 (Act) *When agent i executes a planned action from its current intentions, we obtain the following transition:*

$$\begin{aligned} \mathrm{Act}(i,S) &= \mathrm{Act}(i,\, \langle A_1, \ldots, \langle X_i,\, TL_i,\, [G/P|RestCI_i],\, EQ_i\rangle, \ldots, A_m\rangle) \\ &= r(i,\, \langle A_1, \ldots, \langle X_i,\, TL_i,\, [G/tail(P)|RestCI_i],\, EQ_i\rangle, \ldots, A_m\rangle) \end{aligned}$$

if $tail(P) \neq []$, otherwise

$$\mathrm{Act}(i,S) = r(i,\, \langle A_1, \ldots, \langle X_i,\, TL_i,\, RestCI_i,\, EQ_i\rangle, \ldots, A_m\rangle)$$

where $r = head(P)$.

Finally, the concurrent execution of all reactions of agent i triggered by an event $Evt = \langle \varepsilon(u), s\rangle$ corresponds to a transition of $\mathcal{S}$, denoted $React(i,S)$, which is given by the application of the set of action rules RR_i^{Evt} at component i.

Definition 13 (React)

$$\begin{aligned} \mathrm{React}(i,S) &= \mathrm{React}(i,\, \langle A_1, \ldots, \langle M_i,\, [Evt|RestQ]\rangle, \ldots, A_m\rangle) \\ &= RR_i^{Evt}(i,\, \langle A_1, \ldots, \langle M_i,\, RestQ\rangle, \ldots, A_m\rangle) \end{aligned}$$

Example 5 *In the communicating elevator scenario, where $\mathcal{S} = \langle \mathcal{A}_1, \mathcal{A}_2\rangle$, if the initial agent states are $A_1^0 = \langle\{at(1)\},\, [\mathrm{reqTo}(2)]\rangle$, resp. $A_2^0 = \langle\{at(2)\},\, [\mathrm{reqTo}(3)]\rangle$, and the reaction rules are $RR_1 = \{a_1, \ldots, a_{10}\}$, resp. $RR_2 = \{b_1, \ldots, b_{10}\}$, we obtain the following transitions:*

$$\begin{aligned} \mathrm{React}(1,S_0) &= RR_1^{\mathrm{req}(2)}(1,\, \langle\langle\{at(1)\}, []\rangle,\, \langle\{at(2)\}, [\mathrm{req}(3)]\rangle\rangle \\ &= \langle\langle a_1(\{at(1)\}), []\rangle,\, \langle\{at(2)\}, [\mathrm{reqTo}(3),\, \mathrm{reqPerm}]\rangle\rangle \\ &= \langle\langle\{at(1),\, req(2)\}, []\rangle,\, \langle\{at(2)\}, [\mathrm{reqTo}(3),\, \mathrm{reqPerm}]\rangle\rangle \end{aligned}$$

$$\begin{aligned} \mathrm{React}(2,S_0) &= RR_2^{\mathrm{req}(3)}(2,\, \langle\langle\{at(1)\},\, [\mathrm{req}(2)]\rangle,\, \langle\{at(2)\}, []\rangle\rangle \\ &= \langle\langle\{at(1)\}), [\mathrm{reqTo}(2)]\rangle,\, \langle b_6(\{at(2)\}), []\rangle\rangle \\ &= \langle\langle\{at(1)\}), [\mathrm{reqTo}(2)]\rangle,\, \langle\{req(3),\, up(2)\}), []\rangle\rangle \end{aligned}$$

$$\begin{aligned} S_1 &= \mathrm{React}(1,\, \mathrm{React}(2,S_0)) = \mathrm{React}(2,\, \mathrm{React}(1,S_0)) \\ &= \langle\langle\{at(1),\, req(2)\}), []\rangle,\, \langle\{req(3),\, up(2)\}),\, [\mathrm{reqPerm}]\rangle\rangle \end{aligned}$$

$$\begin{aligned} S_2 &= \mathrm{React}(2,\, S_1) \\ &= \langle\langle\{at(1),\, req(2)\}), [\mathrm{grPerm}]\rangle,\, \langle\{req(3),\, up(2)\}), []\rangle\rangle \end{aligned}$$

$$\begin{aligned} S_3 &= \mathrm{React}(1,\, S_2) \\ &= \langle\langle\{up(1),\, req(2)\}), []\rangle,\, \langle\{req(3),\, up(2)\}), []\rangle\rangle \end{aligned}$$

We assume that $up(x)$ *leads in finite time to* $\mathsf{recvMsg}[\mathrm{arrAt}(x+1)]$, *and therefore elevator* a *will receive the message* $\mathrm{arrAt}(2)$, *and elevator* b *will receive* $\mathrm{arrAt}(3)$, *i.e.*

$$\begin{aligned} S_4 &= \langle\langle\{up(1),\, req(2)\}\rangle,\, [\mathrm{arrAt}(2)]\rangle,\, \langle\{req(3),\, up(2)\}\rangle,\, [\mathrm{arrAt}(3)]\rangle\rangle \\ S_5 &= \mathrm{React}(1,\, \mathrm{React}(2, S_4)) \\ &= \langle\langle\{at(2)\}\rangle, []\rangle,\, \langle\{at(3)\}\rangle, []\rangle\rangle \end{aligned}$$

Assuming the principle of *nondeterministic interleaving* of (possibly concurrent) state changing events, the temporal behavior of a multi-agent system can be described by means of a *labelled transition system.* In contrast to reagent systems, we now have five kinds of transitions transforming a multi-agent system state $S = \langle A_1, \ldots, A_m\rangle$ with $A_i = \langle M_i, EQ_i\rangle$, and $M_i = \langle X_i, TL_i, PTL_i, CI_i\rangle$:

(Perception) An incoming event message ε at agent i yields the transition $S \xrightarrow{i,\varepsilon} S'$, where $S' = S$, except that for A_i, $EQ_i' = EQ_i + \varepsilon$.

(Reaction) A reaction of agent i in response to an event $\varepsilon = head(EQ_i)$ yields the transition $S \xrightarrow{i,RR_i^\varepsilon} S'$, where $S' = \mathrm{React}(i, S)$.

(Action) If there is $G/P \in CI_i$, such that the planned action $r = head(P)$ is executable, i.e. $M_i \vdash Cond(r)$, this yields the transition $S \xrightarrow{i,r+} S'$, where $S' = \mathrm{Act}(i, S)$.

(Replanning) If none of the current intentions is executable, and $G/P = head(CI_i)$ with $r = head(P)$, re-planning yields the transition $S \xrightarrow{i,r-} S'$, where $S' = S$, except that $CI_i' = G/P' + tail(CI_i)$ such that $P' = N + P$, if there is a plan $N \in AR_i^*$ such that $N(M_i) \vdash Cond(r)$, or otherwise $CI_i' = tail(CI_i)$, i.e. the unachievable goal G is being dropped from the set of current intentions.

(Planning) Planning by agent i either a) for a goal $G = head(TL_i)$, or otherwise b), if $TL_i = []$, for some $G \in PTL_i$ such that $M_i \not\vdash G$, yields the transition $S \xrightarrow{pl(i,G)} S'$, where $S' = S$, except that $CI_i' = [G/P] + CI_i$ such that P is a plan to achieve G, i.e. $P(M_i) \vdash G$, and in case a), $TL_i' = tail(TL_i)$.

Typically, one would give priority to perception and reaction over action and planning, i.e. a simple control policy would only allow to execute a planned action, if there is no event requiring a reaction, and it would only allow to generate a new plan if the current intentions have all been carried out, i.e. $CI = []$, or cannot be executed, i.e. for all G/P in CI, $head(P)$ is not executable. There may be reasons, however, to suspend reactions in certain cases in favor of urgent actions (this topic is out of scope of the present paper).

Definition 14 (Execution History) *An* execution history *of a multi-agent system is a chain of state transitions* $S_0 \xrightarrow{\tau_0} S_1 \xrightarrow{\tau_1} \ldots$, *where each* τ_i *corresponds to one of the transitions listed above. A history can be finite or infinite. By definition, a finite history ends in a state. The set of all histories of a multi-agent system* $\mathcal{S}$ *is denoted by* $\mathrm{Hist}(\mathcal{S})$.

For multi-agent system histories, we need the notion of *fairness.*

Definition 15 (Enabled Agent) *We say that an agent $\mathcal{A}_i$ is enabled if its event queue EQ_i, its list of current intentions CI_i, or its task list TL_i is nonempty, or if one of its permanent tasks has to be (re-)solved: for some $G \in PTL_i : X_i \not\vdash G$.*

Definition 16 (Fairness)

1. *A finite history of a multi-agent system $\mathcal{S}$ is called* fair*, if in its final state no agent is enabled.*
2. *An infinite history H of $\mathcal{S}$ is called* fair*, if for all agents $\mathcal{A}_i$, either H contains an infinite number of non-perception transitions of $\mathcal{A}_i$, or it contains an infinite number of states where $\mathcal{A}_i$ is not enabled.*

We shall denote the set of all fair histories of $\mathcal{S}$ by FHist($\mathcal{S}$).

6.5 Assertional Reasoning

Definition 17 ($\mathcal{S}$-Queries) *Let $\mathcal{S}$ be a multi-agent system, and let $L' = \{\mathsf{B}_i F : F \in L_{\text{Query}}\} \cup \{\mathsf{T}_i F : F \in L_{\text{Query}}\} \cup \{\mathsf{PT}_i F : F \in L_{\text{Query}}\} \cup \{\mathsf{rM}_i[\varepsilon] : \varepsilon \in L_{\text{Evt}}\}$. Then, $L_{\mathcal{S}\text{Q}} = L'(\neg, \wedge, \vee)$, i.e. the Boolean closure of L', denotes the set of $\mathcal{S}$-query formulas.*

Inference for $\mathcal{S}$-queries on the basis of a state $S = \langle A_1, \ldots, A_m \rangle$ with $A_i = \langle M_i, EQ_i \rangle$, and $M_i = \langle X_i, TL_i, PTL_i, P_i \rangle$, is defined as follows:

$$\begin{array}{lllll} S & \vdash & \mathsf{B}_i F & \text{iff} & X_i \vdash F \\ S & \vdash & \mathsf{T}_i F & \text{iff} & F \in TL_i \\ S & \vdash & \mathsf{PT}_i F & \text{iff} & F \in PTL_i \\ S & \vdash & \mathsf{rM}_i[\varepsilon] & \text{iff} & \varepsilon = \mathit{head}(EQ_i) \end{array}$$

and finally $S \vdash \neg\phi$ iff $S \not\vdash \phi$, and $S \vdash \phi \wedge (\vee) \psi$ iff $S \vdash \phi$ and (or) $S \vdash \psi$.

As for reagent systems, we get the analogous notion of Hoare triples $\phi\{\tau\}\psi$, invariance assertions $\mathsf{Inv}(\phi)$, leads-to assertions $\phi \leadsto \psi$, and domain descriptions $\mathcal{S}, \Phi$ also for multi-agent systems.

7 Related Work

Although we depart considerably from it, our present work has been much inspired by Shoham's [Sho93] proposal of *agent-oriented programming (AOP).* Our reaction rules may be compared with Shoham's *commitment rules*, and our action rules with *capability rules.* The major differences seem to be that 1) our work is based on our theory of knowledge systems (allowing for negation-as-failure in the query language, and for various forms of updating, including deletion/contraction), while Shoham's AGENT-0 system is defined on the basis of temporal fact knowledge using a kind of standard logical inference and recency-preferring revision of facts; 2) there is no genuine concept of actions in Shoham's AOP, actions are simply represented as facts, and therefore conditional action statements in AOP are not able to account for the epistemic effects of an action which is, on the other hand, essential in our account of actions; 3) AOP is much more ambitious about temporal reasoning using appropriate modalities, and it uses a system clock for synchronization purposes, while our communication acts are realized by asynchronous message passing.

In [Woo92], Wooldridge has defined a formal model of a multi-agent system along the same lines we have followed. Instead of a knowledge system in our sense, he uses the 'deduction model of belief' from [Kon86], and extends it by adding a *belief revision function* corresponding to the update operation of our knowledge systems. His *action rules* correspond to our epistemic action rules, and his *message rules* correspond to our communicative action rules. However, perception and reaction are not explicitly considered. Also, the mental state of an agent in this model consists only of beliefs, and since there is no consideration of goals, or tasks, the agent model does not account for planning and plan execution. Wooldridge presents an interleaved execution model for his multi-agent system model, but he does not define fairness conditions for it. In subsequent work, [Woo95], Wooldridge refined and extended his model and linked the execution histories of a multi-agent system to a dense time line in order to use the temporal logic of reals for specification and assertional reasoning.

In [FW94], a first-order temporal logic-based programming system, CONCURRENT METATEM, is proposed for the purpose of specification and verification. It is shown how to specify and reason about the *Contract Net Protocol*. The *program rules* of this formalism have some similarity with our action rules, but there is no explicit concept of events and event-triggered rules.

Gelfond et al. (see [GL92], or more recently [BGP95]) have proposed an *action domain description language* which allows to represent a) three kinds of facts describing in addition to the basic propositional knowledge which is timestamped by situation names also knowledge about the occurrence of actions and about the precedence of situations, and b) epistemic action rules which are called *causal effect laws*. A domain description can be queried about the validity of timestamped facts, and about the hypothetical effects of actions. Gelfond et al. suggest to use their formalism "in designing intelligent agents capable of planning in a changing environment". Their approach is more general than ours with respect to the explicit use of situation names, and its possibility to record not only state knowledge but also the occurrence of actions. In other respects, however, it is less powerful: 1) while in our modelling of agents any knowledge system may be used, they restrict state knowledge to literal sentences, and thus, the epistemic effects of actions cannot be uncertain or indefinite; 2) our approach accounts for the interplay of perception, reaction, planning, and plan execution, while they only consider planning and plan execution; 3) they only consider the single-agent scenario without communication, whereas we account for multi-agent scenarios with inter-agent communication. In addition, we outline methods of assertional reasoning for agent systems which allow to prove the correctness of a specification with respect to safety and progress properties.

Our framework of assertional reasoning for multi-agent systems is a generalization of the approach of [Sha93] for concurrent systems based on the temporal logic concepts of Hoare [Hoa69], and Manna and Pnueli [MP92]. Since in the theory of concurrent systems, knowledge states consist of the values of state variables, there is no consideration of more complex states such as those of knowledge bases and the related concepts of inference and update. While the system model of [Sha93] contains only one kind of state changing event, corresponding to assignment statements, we have to consider five kinds of transitions in order to take into account all relevant state changes of a multi-agent system: perception, reaction, planning, plan execution, and re-planning.

Liveness and fairness properties for multi-agent systems were defined in an abstract automata-theoretic setting in [Bur93]. Execution histories are treated as elements of a formal language (being a prefix-closed subset of the set of all finite sequences of events/actions). While this approach abstracts away from the internal structure and cognitive operations of agents, our work is primarily concerned with this internal structure from the conceptual and programming point of view.

There is a large body of rather abstract work on multi-modal logics modeling the mental attitudes of agents in the tradition of Hintikka's possible worlds semantics for knowledge and belief.[13] The connection of this work to the operational concepts needed in the design and implementation of agents is not clear, however. In [RG93], for instance, Rao and Georgeff propose to use three-dimensional modal logics, called *BDI logics*, for the specification and verification of 'abstract agent-oriented systems'. These logics, however, lack genuine concepts of action and reaction. Thus, by being not able to account for the interplay of perception, reaction and action, they are rather limited, although, by means of their three-dimensional possible worlds semantics, they are very complex.[14] Specifications in these logics are completely abstract and, as opposed to our specification formalism, not executable. Verification, however, is rather important for executable, and not for abstract specifications. Finally, by lacking a concept of communication, BDI logics cannot model multi-agent systems.

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 reagents whose KB is a set of literals upon which classical inference is performed (i.e. there is no form of the CWA). What Rao calls a 'plan', is rather a specific form of a reaction rule (which may be viewed as a pre-defined plan to react made up at design time). AgentSpeak(L) does, in fact, not allow planning and pro-active plan execution. Similar to our semantic account of vivid agents, Rao defines a transition system semantics for his AgentSpeak(L) reagents.

In recent years, 'hybrid' agent architectures combining reactive and deliberative behavior have become increasingly important (see, e.g., [BS92, BHS93, Mue94]). This indicates that, in practice, neither the traditional purely deliberative logic-based approaches, nor the more recently fashionable purely reactive approaches are appropriate for the design and validation of agent systems. Our proposal of *vivid agents* with action and reaction rules provides a theoretical foundation for such hybrid architectures. In fact, [Mue94] uses a certain form of action rules, called *pattern of interaction*, and an action-rule-based update operation, called *mechanism execution function*, in outlining the formal model of the hybrid architecture InteRRap. In [BHS93], an architecture for representing and executing communication protocols in multi-agent systems is proposed. However, no suggestion is made how to integrate communication with query answering and updating in a logical way, and no formalism for proving the correctness of protocols is presented.

[13] See, e.g., [Wer88, CL90, RG91, Sin94].

[14] "Hence, the implemented BDI systems have tended to use the three major attitudes [Belief,Desire,Intention] as data structures, rather than as modal operators."[Rao96]

8 Open Issues and Limitations

Several issues had to be left open. First, we did not consider the possibility of simultaneous actions in planning. Second, we did not say which planning procedures (e.g. partial order planning, abstraction, etc.) would be appropriate for vivid agents. Third, we did not consider the possible unability of an agent to find a plan for a specific goal. Fourth, we had to omit the question how to distinguish important events (for which it is urgent to react adequately) from less important ones. Fifth, we did not show how sophisticated forms of communication can be achieved by vivid agents in accordance with speech act theory. All these issues are orthogonal to the points of our paper; they are treated elsewhere, and may be incorporated in our approach.

More significant, however, is the lack of a method to interrupt planning when new events call for timely reaction. A possible solution of this problem, where (re)action execution and planning are performed concurrently, is proposed in [SW96].

9 Conclusion

The concept of vivid agents is a powerful extension of the concept of knowledge bases. It combines static knowledge in the form of a declarative knowledge base with dynamic knowledge in the form of action and reaction rules. As a rule-based approach, it is more 'computational' than modal logic approaches based on possible worlds semantics.

We have shown how to model deliberative, pro-active, reactive and communicative behavior in this framework. We also outlined methods of assertional reasoning for vivid agent systems which may allow to prove safety and progress properties for critical applications.

A first case study [SW97] shows that the concept of vivid agents can be successfully applied to practical problems. A programming language for vivid agent systems, called VIVA, is currently being developed and implemented [Wag96b].

We hope that our agent model can serve as a basis for the formalization of further high-level concepts such as values, emotions, social structures, social laws, etc., which will be the topic of our future research.

Acknowledgements: Thanks to Daniela A. Plewe and Michael Schroeder for stimulating discussions. I am also grateful to Michael Gelfond for pointing out a shortcoming in an earlier version of the specification of the two-elevator system, and to Michael Wooldridge for helpful comments on a preliminary version of the paper.

References

[BGP95] C. Baral, M. Gelfond and A. Provetti: Representing Actions (Part One) – Laws, Observations and Hypothesis, Preprint, 1995.

[Bel77] N.D. Belnap: A Useful Four-valued Logic, in G. Epstein and J.M. Dunn (Eds.), *Modern Uses of Many-valued Logic*, Reidel 1977, 8–37.

[Bur93] H.-D. Burkhard: Liveness and Fairness Properties in Multi-Agent Systems, *Proc. of IJCAI'93*, 1993, 325–330.

[BHS93] B. Burmeister, A. Haddadi and K. Sundermeyer: Generic Configurable Cooperation Protocols for Multi-Agent Systems, *Proc. of MAAMAW'93*, Springer-Verlag, 1993.

[BS92] B. Burmeister and K. Sundermeyer: Cooperative Problem-Solving Guided by Intentions and Perception, in Y. Demazeau and J.P. Müller (Eds.), *Decentralized Artificial Intelligence*, Elsevier/North-Holland, 1992.

[CL90] P.R. Cohen and H.J. Levesque: Intention is Choice with Commitment, *Artificial Intelligence* **42**:3 (1990).

[Dee94] S.M. Deen (Ed.), *Proc. 2nd Int. Working Conf. on Cooperating Knowledge-Based Systems*, DAKE Center, University of Keele, 1994.

[FN71] R.E. Fikes and N. Nilsson: "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", *Artificial Intelligence* **5**:2 (1971), 189–208.

[FW94] M. Fisher and M. Wooldridge: Specifying and Executing Protocols for Cooperative Action, in [Dee94], 295–306.

[vGT91] A. van Gelder and R.W. Topor: Safety and Translation of Relational Calculus Queries, *ACM Transactions on Database Systems* **16**:2 (1991), 235–278.

[GL92] M. Gelfond and V. Lifschitz: Representing Actions in Extended Logic Programs, in *Proc. of Joint Int. Conf. and Symp. on Logic Programming 1992*, MIT Press, 1992, 559–573.

[F+95] R. Fagin, J. Halpern, Y. Moses and M. Vardi: *Reasoning about Knowledge*, MIT Press, Cambridge (MA), 1995.

[Hoa69] C.A.R. Hoare: An Axiomatic Basis for Computer Programming, *Communications of the ACM* **12**:10 (1969), 576–583.

[Kon86] K. Konolige: *A Deduction Model of Belief*, Pitman and Morgan Kaufmann, San Mateo (CA), 1986.

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

[KS86] R.A. Kowalski and M. Sergot: A Logic-Based Calculus of Events, *J. of New Generation Computing* **4** (1986), 67–95.

[Kow95] R.A. Kowalski: Using Meta-Logic to Reconcile Reactive with Rational Agents, Meta-Logics and Logic Programming, MIT Press, 1995.

[Lev84] H.J. Levesque: Foundations of a Functional Approach to Knowledge Representation, *AI* **23**:2 (1984), 155-212.

[MAAMAW96] W. Van de Velde and J.W Perram (Eds.), *Agents Breaking Away*, Proc. of MAAMAW'96, Springer Lecture Notes in AI **1038** (1996).

[McCD89] D.R. McCarthy and U. Dayal: The Architecture of an Active Database Management System, *Proc. ACM SIGMOD-89*, 1989, 215–224.

[McC79] J. McCarthy: Ascribing Mental Qualities to Machines, in M. Ringle (Ed.), *Philosophical Perspectives in Artificial Intelligence*, Humanities Press, 1979, 161–195.

[McC80] J. McCarthy: Circumscription – A Form of Non-Monotonic Reasoning, *AI* **13**:1/2 (1980), 27–39, 171–172.

[MP92] Z. Manna and A. Pnueli: *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag, New York, 1992.

[Mue94] J.P. Müller: A Conceptual Model of Agent Interaction, in [Dee94], 213–233.

[Pei76] C.S. Peirce: Mathematical Philosophy, Vol. IV of C. Eisele (Ed.), *The New Elements of Mathematics*, Mouton Publishers, The Hague/Paris, 1976.

[Rao96] A.S. Rao: 'AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language', in [MAAMAW96], 42–55.

[RG91] A.S. Rao and M.P. Georgeff: Modeling Rational Agents within a BDI-Architecture, in J. Allen, R. Fikes and E. Sandewall (Eds.), *Proc. of KR-91*, Morgan Kaufmann, San Mateo (CA), 1991.

[RG93] A.S. Rao and M.P. Georgeff: A Model-Theoretic Approach to the Verification of Situated Reasoning Systems, in *Proc. of IJCAI'93*, Morgan Kaufmann, San Mateo (CA), 1993, 318–324.

[Sha93] A.U. Shankar: An Introduction to Assertional Reasoning for Concurrent Systems, *ACM Computing Surveys* **25**:3 (1993), 225–262.

[Sho93] Y. Shoham: Agent-Oriented Programming, *Artificial Intelligence* **60** (1993), 51–92.

[Sin94] M.P. Singh: *Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communication*, LNAI 799, Springer-Verlag, Heidelberg, 1994.

[SW96] M. Schröder and G. Wagner: Concurrent Action and Planning – Towards an Implementation of VIVA, position paper for *Workshop on Foundations of Multi-Agent Systems (FOMAS'96)*, 1996.

[SW97] M. Schröder and G. Wagner: Distributed Diagnosis by Vivid Agents, *Proc. Int. Conf. on Autonomous Agents*, ACM Press, 1997.

[Wag94] G. Wagner: *Vivid Logic – Knowledge-Based Reasoning with Two Kinds of Negation*, Springer Lecture Notes in AI **764** (1994).

[Wag95] G. Wagner: From Information Systems to Knowledge Systems, in E. Falkenberg (Ed.), *Proc. of IFIP Conf. on Information System Concepts (ISCO-3)*, Chapman & Hall, London, 1995.

[Wag96a] G. Wagner: A Logical and Operational Model of Scalable Knowledge- And Perception-Based Agents, in [MAAMAW96], 26–41.

[Wag96b] G. Wagner: VIVA Knowledge-Based Agent Programming, technical report, Univ. Leipzig, 1996, (available as ftp://www.informatik.uni-leipzig.de/∼gwagner/VIVA.ps.gz).

[Wag96c] G. Wagner: Possibilistic and Extended Logic Programs, technical report, (available as ftp://www.informatik.uni-leipzig.de/∼gwagner/poss.ps.gz).

[Wer88] E. Werner: Towards a Theory of Communication and Cooperation for Multi-Agent Planning, in M.Y. Vardi (Ed.), *Proc. of 2nd Conf. on Theoretical Aspects of Reasoning about Knowledge*, Morgan Kaufmann, San Mateo (CA), 1988, 129–144.

[Woo92] M. Wooldridge: *The Logical Modelling of Computational Multi-Agent Systems*, PhD thesis, Dep. of Computation, Manchester Metropolitan University, Manchester, UK, 1992.

[Woo95] M. Wooldridge: "This is MyWorld: The Logic of an Agent-Oriented Testbed for DAI, in M. Wooldridge and N.R. Jennings (Eds.), *Intelligent Agents: Theories, Architectures and Languages*, Springer LNAI **890** (1995), Springer-Verlag, 160–178.