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

Preliminary Report†

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

1 Introduction **2**

2 Knowledge Systems **3**

3 Action Rules and Planning **5**
3.1 Representing Actions 6
3.2 Planning 7

4 Reaction and Interaction Rules **8**
4.1 Private Reaction Rules 8
4.1.1 Example: A Reactive and Deliberative Elevator 9
4.1.2 Example: A Purely Reactive Elevator 9
4.1.3 Verifying the Reactive Elevator Specification 10
4.2 Interaction Rules 10
4.2.1 Example: Communicating Elevators 11
4.2.2 Incomplete and Inconsistent Information 12

*This work has been done during a stay at Univ. Nova de Lisboa under an HCM research grant of the EU. The author wants to thank Luis M. Pereira for his hospitality and support.

†A shorter version containing the single-agent material will be presented at the ModelAge'96 workshop, and one containing the multi-agent material will be presented at the MAAMAW'96 workshop and will be included in the proceedings to be published by Springer-Verlag.

5 Specification, Execution and Verification of Single-Agent Systems 13
5.1 Operational Semantics of Private Reaction Rules 14
5.2 Defining an Execution Model by Meta-Programming 15
5.3 Agent Queries and Mental Inputs 17
5.4 Agent Systems as State Transition Systems 17
5.5 Assertional Reasoning 18
5.6 Proof Rules for Assertional Reasoning 19
5.7 Examples of Verification 20

6 Multi-Agent Systems 20
6.1 Operational Semantics of Action and Reaction Rules 21
6.2 Multi-Agent Systems as State Transition Systems 22
6.3 Assertional Reasoning 24

7 Open Issues and Limitations 25

8 Related Work 25

9 Conclusion 27

1 Introduction

A *vivid agent* is a software-controlled entity whose state is represented by a knowledge base, and whose behavior is represented by means of *action* and *reaction rules*. Following [Sho93], we shall view the state of an agent as consisting of mental components, such as beliefs and intentions. The basic functionality of a vivid agent comprises an update and an inference operation, and the possibility 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.

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 system will do the job, while in more involved cases one might need the ability to process, in addition to simple facts, (disjunctive or gradual) uncertain information, or temporal information, or even such advanced capabilities as deductive query answering and inconsistency handling. The

[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, secure, partial, disjunctive, deductive, active, etc.

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. Since vivid knowledge systems are scalable, one can 'plug in' any suitable one for constructing a specific agent system. In any case, our definition of action and reaction rules is general enough to apply to all kinds of knowledge systems, and thus makes vivid agents scalable as well.

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 ($\vee$), weak negation (alias negation-as-failure, denoted by $-$), strong negation ($\sim$), and the truth constant 1; the predicate symbols $p, q, r, \ldots$; the constant symbols $c, d, \ldots$ and variables $x, y, \ldots$. For the sake of simplicity we shall not consider functional terms but only variables and constants. An *atom* a is an atomic formula, it is called *proper*, if $a \neq 1$. *Literals* are either atoms or strongly negated atoms, $l = a|{\sim}a$. *Extended literals* are either literals or weakly negated literals, $e = l|{-}l$. We use $a, b, \ldots$, $l, k, \ldots$, $e, f, \ldots$, and $F, G, H, \ldots$ as metavariables for atoms, literals, extended literals and well-formed formulas, respectively. A variable-free expression is called *ground*. The set of all proper atoms (literals, extended literals) of a given language is denoted by At (Lit, XLit). If $\mathcal{F}$ is a set of logical functors, say $\mathcal{F} \subseteq \{1, -, \sim, \wedge, \vee, \rightarrow, \exists, \forall, \ldots\}$, 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. Free(F) denotes the set of free variables of F.

We shall 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 inputs, i.e. those formulas representing new information a KB may be updated with. 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 quintuple:*[3]

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

[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 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 KS in terms of query and input processing was already implicitly present in Belnap's [1977] view of a KS. In [Lev84] it was proposed as a 'functional approach to knowledge representation'. In [Wag94a, Wag95] the concept of knowledge systems was further extended and used as an integrating framework for knowledge representation and logic programming.

where the inference relation $\vdash \subseteq L_{\mathrm{KB}} \times L^0_{\mathrm{Query}}$*, together with the update operation* $\mathsf{Upd} : L_{\mathrm{KB}} \times L^0_{\mathrm{Input}} \to L_{\mathrm{KB}}$*, satisfy for any* $X \in L_{\mathrm{KB}}$*,*

(KS1) $X \vdash 1$*, and* $\mathsf{Upd}(X, 1) = X$*.*

(KS2) $L_{\mathrm{Input}} \subseteq L_{\mathrm{Query}}$*.*

(KS3) $\mathsf{Upd}(X, F) \vdash F$*, for any* $F \in L^0_{\mathrm{Input}}$ *which is consistent with* X*.*[4]

In the sequel, we shall sometimes simply write 'KB' in formal expressions standing for an arbitrary knowledge base $X \in L_{\mathrm{KB}}$.

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

$$\mathsf{Upd} : L_{\mathrm{KB}} \times 2^{L_{\mathrm{Input}}} \to L_{\mathrm{KB}}$$

which has to be defined in such a way that for any finite $A \subseteq L_{\mathrm{Input}}$, $\mathsf{Upd}(\mathrm{KB}, A) = \mathsf{Upd}(\mathrm{KB}, \bigwedge A)$. We sometimes write $\mathrm{KB} + F$ as an abbreviation of $\mathsf{Upd}(\mathrm{KB}, F)$, resp. $\mathrm{KB} - F$ as an abbreviation of $\mathsf{Upd}(\mathrm{KB}, -F)$.

Example 1 (Relational Databases) *A KB consisting 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* $X \subseteq \mathrm{At}$ *and an if-query is defined in the following way:*

$$\begin{array}{lrl}
(\vdash a) & X \vdash a & \textit{if} \quad a \in X \\
(\vdash -a) & X \vdash -a & \textit{if} \quad a \notin X \\
(\vdash \wedge) & X \vdash F \wedge G & \textit{if} \quad X \vdash F \ \ \& \ \ X \vdash G \\
(\vdash \vee) & X \vdash F \vee G & \textit{if} \quad X \vdash F \ \textit{or} \ X \vdash G
\end{array}$$

This inductive definition is completed by assuming the following DeMorgan-style rewrite rules:

$$\begin{array}{lll}
-(F \vee G) & \longrightarrow & -F \wedge -G \qquad -\,-F \longrightarrow F \\
-(F \wedge G) & \longrightarrow & -F \vee -G
\end{array}$$

For instance, $X_1 \vdash m(P, L) \wedge r(P) \wedge -r(L)$*. Updates are insertions,* $\mathsf{Upd}(X, a) := X \cup \{a\}$*, and deletions,* $\mathsf{Upd}(X, -a) := X - \{a\}$*, of atoms* $a \in \mathrm{At}$*. For a consistent* $E \subseteq \mathrm{At} \cup \overline{\mathrm{At}}$*, we have* $\mathsf{Upd}(X, E) = X \cup E^+ - E^-$*. For instance,*

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

describes a possible transaction. The KS of relational databases, denoted by $\boldsymbol{A}$*, is then defined as*

$$\boldsymbol{A} \;=\; \langle 2^{\mathrm{At}},\, \vdash,\, L(-, \wedge, \vee),\, \mathsf{Upd},\, \mathrm{At} \cup \overline{\mathrm{At}} \,\rangle$$

[4] According to some notion of consistency associated with the abstract knowledge system. E.g., one might want to exclude contradictory pieces of information from this reflexivity principle: $\mathsf{Upd}(\{\sim p\}, p) \nvdash p$. We shall not discuss this issue in the present paper, however.

Example 2 (Defeasible Factbases) *A KB consisting of ground literals (viewed as positive and negative facts) is called a* factbase. *For instance, the factbase*

$$X_2 = \{r(S), r(P), s(S), {\sim}s(L), {\sim}s(P),\ m(P,L),\ m(T,S)\}$$

may represent the information that Susan and Peter are residents, Susan is a smoker, Linda and Peter are nonsmokers, Peter is married with Linda, and Tom is married with Susan. In a defeasible factbase, contradictory items invalidate (i.e. neutralize) each other. $\mathrm{Inc}(X) = X \cap \widetilde{X}$ *collects all contradictory information units of a factbase* X. *As a kind of defeasible deduction from positive and negative facts an inference relation* $\vdash$ *between a factbase* $X \subseteq \mathrm{Lit}$ *and a ground formula is defined in the following way:*[5]

$$\begin{array}{rc} (\vdash l) & X \vdash l \quad \textit{if} \quad l \in X \ \ \& \ \ \tilde{l} \notin X \\ (\vdash -l) & X \vdash -l \quad \textit{if} \quad l \notin X \\ (\vdash \wedge) & X \vdash F \wedge G \quad \textit{if} \quad X \vdash F \ \ \& \ \ X \vdash G \end{array}$$

For instance, one might ask "who is married with a nonsmoking non-resident ?", i.e. $X_2 \vdash m(x,y) \wedge {\sim}s(y) \wedge -r(y)$. *Updates are insertions and deletions of literals* $l \in \mathrm{Lit}$*:*

$$\begin{array}{rcl} \mathsf{Upd}(X, l) & := & X \cup \{l\} \\ \mathsf{Upd}(X, -l) & := & X - \{l\} \end{array}$$

We also add a unary functor standing for recency-preferring revision*:*

$$\begin{array}{rcl} \mathsf{Upd}(X, *l) & := & X - \{\tilde{l}\} \cup \{l\} \\ X \vdash *l & \textit{if} & X \vdash l \end{array}$$

The knowledge system of defeasible factbases is then defined as

$$\boldsymbol{F}_d \ := \ \langle\, 2^{\mathrm{Lit}},\ \vdash,\ L(*, -, {\sim}, \wedge),\ \mathsf{Upd},\ \mathrm{Lit}^{*,-} \,\rangle$$

Knowledge systems extending $\boldsymbol{A}$ conservatively are called *vivid*. Positive vivid knowledge systems have only negation-as-failure and use a general Closed-World Assumption, whereas general vivid knowledge systems employ two kinds of negation and specific Closed-World Assumptions. For instance, $\boldsymbol{A}$ can be extended to a general vivid knowledge system, called *factbases*, by allowing for literals instead of atoms as information units. In factbases, one can represent inexact predicates which is a particular form of incomplete knowledge. 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 disjunctve factbases. All these kinds of knowledge bases can be extended to *deductive knowledge bases* by adding deduction rules of the form $F \leftarrow G$ where $F \in L_{\mathrm{Input}}$ and $G \in L_{\mathrm{Query}}$ (see, e.g., [Wag94b]).

3 Action Rules and Planning

Action rules have the general form of *Action* $\leftarrow$ *Condition*.[6] We shall distinguish between action and reaction rules. While action rules are used to represent the elementary action types of an agent, and to generate and execute plans, thus determining

[5] This inductive definition is completed by DeMorgan-style rewrite rules including double negation rules such as ${\sim}{-}a \longrightarrow a$. See [Wag94b].

[6] Action rules (in the form of 'event-condition-action' rules) were also proposed as an extension to databases, now called *active databases*, see e.g. [McCD89].

the *proactive* behavior, reaction rules are used to specify the *reactive* behavior of an agent which is based on both perception and knowledge in our account.

3.1 Representing Actions

Depending on the resp. domain and the capabilities and tasks of an agent, only certain actions can be performed by it. Besides communication acts, an agent can perform *private actions* by means of its effectors. An action type corresponds to a set of *action rules* which represent the different execution conditions and effects of different situation contexts. We denote the set of all elementary private action types available to an agent by L_{Act}. They correspond to private action rules of the form

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

In addition, there are communicative action rules

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

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 elementary action; $\mathsf{do}(\alpha)$ is a procedure call in order to execute the private action $\alpha \in L_{\mathrm{Act}}$, and r_α is the name of the private action rule; and likewise $\mathsf{sendMsg}[\eta, R]$ is a procedure call to execute the communicative action $\eta \in L_{\mathrm{CEvt}}$ where R identifies the receiver of the outgoing message, and r_η is the name of the communicative action rule.

We require that the precondition *Cond* is an evaluable formula and that all free variables of the epistemic effect formula *Eff* occur also in *Cond*.[7] If 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 resp. action rule r, thus obtaining its *epistemic projection* $\dot{r}$ which is called an *epistemic action rule*.

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* applies to X, i.e. $X \vdash \mathit{Cond}$.

Definition 2 (Rule Application) *Let $E \in L_{\mathrm{Input}}$, $C \in L_{\mathrm{Query}}$. An epistemic action rule $r : E \leftarrow C$ represents an update function, i.e. a mapping $r : L_{\mathrm{KB}} \rightarrow L_{\mathrm{KB}}$, whose application is defined as*

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

Notice that in the case of a non-applicable rule, we get by definition

$$r(X) = \mathsf{Upd}(X, \emptyset) = \mathsf{Upd}(X, \bigwedge \emptyset) = \mathsf{Upd}(X, 1) = X$$

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 $\dot{r}(X)$.

[7] 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*, resp. *domain-independent*, formulas.

Example 3 (Elevator Actions) *In an elevator scenario, we have the elementary actions of going one floor up, or one floor down:*

$$\begin{aligned} \mathsf{oneup}: &\quad at(x+1) \wedge -at(x) \leftarrow at(x) \wedge x < topfloor \\ \mathsf{onedown}: &\quad at(x-1) \wedge -at(x) \leftarrow at(x) \wedge x > 0 \end{aligned}$$

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*

$$\mathsf{oneup}(X_0) = X_0 - at(3) + at(4) = \{at(4)\}$$

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.

Observation 1 (Atomicity) *All elementary actions have to be executable* atomically. *That is, once an agent starts executing an elementary action, other agents cannot influence the execution or observe intermediate points of the execution. This means that 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*.*

3.2 Planning

We shall 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 reachieved over and over again. Tasks are typically assigned to an agent at design time, or at run-time via commands or requests from users or other agents.

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

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

where P is applied to X_0 as a composed function.

Example 4 (The Deliberative Elevator) *Let* $L_{\mathrm{Act}} = \{\mathsf{oneup}, \mathsf{onedown}\}$ *consist of the two actions defined above, 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:*

$$P = \mathsf{oneup} \circ \mathsf{oneup} \circ \mathsf{oneup} \circ \mathsf{oneup}$$

the execution of which solves the task:

$$P(X_0) = \mathsf{oneup}(\mathsf{oneup}(\mathsf{oneup}(\mathsf{oneup}(\{at(3)\})))) = \{at(7)\}$$

Notice that our concept of planning on the basis of action rules in knowledge systems can be viewed as a generalization of the STRIPS[8] paradigm which corresponds to planning on the basis of $\boldsymbol{A}$, the knowledge system of relational databases. 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.

4 Reaction and Interaction Rules

Reaction rules control the behaviour of a vivid agent in response to perception events related to the environment, or to communication events caused by other agents. We distinguish between private and communicative reaction rules, and call the latter *interaction rules*. *PEvt* and *CEvt* denote the resp. sets of perception and communication event schemas, *Evt* denotes their union, and L_{PEvt}, L_{CEvt}, and L_{Evt} the resp. event languages.

4.1 Private Reaction Rules

A private reaction rule has the form $\mathsf{do}(\alpha)$, *Eff* $\leftarrow$ $\mathsf{recvMsg}[\varepsilon, S]$, *Cond*, where

1. *Eff* $\in L_{\mathrm{Input}}$, and *Cond* $\in L_{\mathrm{Query}}$, are logical formulas expressing the epistemic effects, resp. the precondition, of the reaction;

2. $\alpha \in L_{\mathrm{Act}}$ is an elementary action expression;

3. ε represents either an elementary perception event created by the perception subsystem with identifier S, i.e. $\varepsilon \in L_{\mathrm{PEvt}}$, or a communication event created by another agent identified by S, i.e. $\varepsilon \in L_{\mathrm{CEvt}}$. If there is only one perception subsystem, the identifier S will be omitted, i.e. we write $\mathsf{recvMsg}[\varepsilon]$ when $\varepsilon \in L_{\mathrm{PEvt}}$.

The premise $\mathsf{recvMsg}[\varepsilon, S]$ is simply a test whether the event queue contains the incoming message ε addressed to the agent either by the perception subsystem, or by another agent, identified by S. Thus, perception and communication events will be represented by incoming messages.[9]

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* $= 1$. 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). It will be more difficult in knowledge systems with deduction rules (e.g. deductive databases).

[8] See [FN71].

[9] In a robot, for instance, appropriate perception subsystems, operating concurrently, will continously 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.

4.1.1 Example: A Reactive and Deliberative Elevator

As in the previous example, we assume two actions, $L_{\text{Act}} = \{\mathsf{oneup}, \mathsf{onedown}\}$, and one perception event type, $L_{\text{PEvt}} = \{req(x)\}$. Whenever the elevator is requested to serve floor k, signalled by the event $req(k)$, this creates a task $req(k)$ for which a plan will be generated and subsequently executed.

We need the following mental reaction rule providing for the creation of new tasks:

$$\mathsf{T}req(x) \leftarrow \mathsf{rM}[\text{req}(x)]$$

4.1.2 Example: A Purely Reactive Elevator

In order to describe an elevator as a reactive agent we use three action types: $L_{\text{Act}} = \{mvup, mvdown, halt\}$; and two perception event types: $L_{\text{PEvt}} = \{req(x), arrAt(x)\}$. 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 $req(k)$, this will be recorded in the KB by means of the fluent $req(k)$.

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

α_1 : $\mathsf{do}(\text{mvup}),\ req(x) \wedge -at(y) \wedge up(y)$
$\leftarrow\ \mathsf{recvMsg}[\text{req}(x)],\ at(y) \wedge x > y \wedge x < topfloor$

α_2 : $\mathsf{do}(\text{mvdown}),\ req(x) \wedge -at(y) \wedge down(y)$
$\leftarrow\ \mathsf{recvMsg}[\text{req}(x)],\ at(y) \wedge x < y \wedge x > 0$

α_3 : $up(x) \wedge -up(x-1)$
$\leftarrow\ \mathsf{recvMsg}[\text{arrAt}(x)],\ req(y) \wedge x < y \wedge x < topfloor$

α_4 : $down(x) \wedge -down(x+1)$
$\leftarrow\ \mathsf{recvMsg}[\text{arrAt}(x)],\ req(y) \wedge y < x \wedge x > 0$

α_5 : $\mathsf{do}(\text{halt}),\ at(x) \wedge -req(x) \wedge -up(x-1) \wedge -down(x+1)$
$\leftarrow\ \mathsf{recvMsg}[\text{arrAt}(x)],\ req(x)$

where do is the key word for private actions (realized by means of procedure calls). 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 resp. *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. recvMsg[req(5)], we get in the first step

$$X_1 = \alpha_1(X_0) = \{req(5),\ up(3)\}$$

and the elevator is moving upwards. When it arrives at the fourth floor, $arrAt(4)$ is signalled, and the reaction rule α_3, causing the continuation of moving up, is applicable

$$X_2 = \alpha_3(\{req(5),\ up(3)\}) = \{req(5),\ up(4)\}$$

Finally, the fifth floor is reached, i.e. $\mathsf{recvMsg}[\mathrm{arrAt}(5)]$ holds, and α_5 is applicable:

$$X_3 = \alpha_5(\{req(5),\ up(4)\}) = \{at(5)\}$$

4.1.3 Verifying the Reactive 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 $req(k)$ 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 $\mathsf{recvMsg}[\mathrm{req}(k)]$ hold. Since the assertion holds trivially for $k = m$ we may assume that $k \neq m$. Thus, in the first step either α_1 or α_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 asume 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, $up(x)$ leadsto $\mathsf{recvMsg}[\mathrm{arrAt}(x+1)]$, the message $arrAt(m+1)$ will be received. This triggers α_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 α_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 α_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 $k < m$. □

4.2 Interaction Rules

An interaction rule has the form

$$\mathsf{sendMsg}[\eta, R],\ \mathit{Eff} \ \leftarrow\ \mathsf{recvMsg}[\varepsilon, S],\ \mathit{Cond}$$

where ε, S, *Eff*, and *Cond*, i.e. the triggering event, the sender, the epistemic effect and the precondition formula are defined like for private reaction rules, and the action $\mathsf{sendMsg}[\eta, R]$ represents a communication act realized by asynchroneous message passing, such that $\eta \in L_{\mathrm{CEvt}}$, and R identifies the receiver of the message. 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 Peirce has 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.

4.2.1 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, $L_{\mathrm{PEvt}} = \{\mathrm{req}(x), \mathrm{arrAt}(x), \mathrm{timeout}\}$, we now also have communication events in order to avoid collisions and deadlocks, i.e. for guaranteeing *safety* and *progress*. The elevators use two communication acts for coordination: requesting the permission to approach, and granting the permisssion, expressed by $L_{\mathrm{CEvt}} = \{reqPerm, grPerm\}$. This leads to the following specification, including a communication protocol.

Elevator a needs only the rules $a_1, \ldots, a_9$, while elevator b needs only the rules $b_1, \ldots, b_9$. If a is $at(1)$ and reveives 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:

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

If a receives a request from b to give permission for floor 2, and a is $at(1)$ or moves down from floor 2, it grants the requested permission; likewise, if b is requested for permission from a, and is currently $at(3)$ or moving up from floor 2:

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

However, 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:

$$\begin{array}{ll} a_3: & permreq(b) \leftarrow \mathsf{recvMsg}[\mathrm{reqPerm}, b],\ at(2) \vee up(1) \\ b_3: & permreq(a) \leftarrow \mathsf{recvMsg}[\mathrm{reqPerm}, a],\ at(2) \vee down(3) \end{array}$$

Then, if one of the elevators is halting $at(2)$, we assume the arrival of a timeout message after a certain time triggering a request to depart in order to avoid a deadlock. The second posibility to avoid a deadlock is a service request before the timeout as described by the rules $(a7, b7)$ below.

$$\begin{array}{ll} a_4: & \mathsf{do}(\mathrm{mvdown}),\ req(1) \wedge down(2) \wedge -at(2) \leftarrow \mathsf{recvMsg}[\mathrm{timeout}],\ at(2) \\ b_4: & \mathsf{do}(\mathrm{mvup}),\ req(3) \wedge up(2) \wedge -at(2) \leftarrow \mathsf{recvMsg}[\mathrm{timeout}],\ at(2) \end{array}$$

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; likewise b when $at(3)$:

$$\begin{array}{ll} a_5: & \mathsf{do}(\text{mvup}),\ up(1) \wedge -at(1) \ \leftarrow\ \mathsf{recvMsg}[\text{grPerm}, b],\ at(1) \wedge req(2) \\ b_5: & \mathsf{do}(\text{mvdown}),\ down(3) \wedge -at(3) \ \leftarrow\ \mathsf{recvMsg}[\text{grPerm}, a],\ at(3) \wedge req(2) \end{array}$$

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

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

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

$$\begin{array}{ll} a_7: & \mathsf{sendMsg}[\text{grPerm}, b],\ -permreq(b) \\ & \leftarrow\ \mathsf{recvMsg}[\text{req}(1)],\ at(2) \wedge permreq(b) \\ b_7: & \mathsf{sendMsg}[\text{grPerm}, a],\ -permreq(a) \\ & \leftarrow\ \mathsf{recvMsg}[\text{req}(3)],\ at(2) \wedge permreq(a) \end{array}$$

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

$$\begin{array}{ll} a_8, b_8: & \mathsf{do}(\text{halt}),\ at(x) \wedge -req(x) \wedge -up(x-1) \\ & \leftarrow\ \mathsf{recvMsg}[\text{arrAt}(x)],\ req(x) \wedge up(x-1) \\ a_9, b_9: & \mathsf{do}(\text{halt}),\ at(x) \wedge -req(x) \wedge -down(x+1) \\ & \leftarrow\ \mathsf{recvMsg}[\text{arrAt}(x)],\ req(x) \wedge down(x+1) \end{array}$$

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 defeasible factbases for domains where incomplete and inconsistent information has to be processed.

4.2.2 Incomplete and Inconsistent Information

We show how the possible difference in epistemic strength between a piece of information conveyed by communication and one obtained through one's own perception may be captured in a defeasible factbase. The former leads to an epistemic update, i.e. is balanced against incompatible information, while the latter leads to a recency-preferring revision, i.e. overrides incompatible information.

We consider the following scenario (derived from example 2). Special agent Cooper is a nonsmoker and he beliefs that JB (alias 007) is a smoker, i.e. his beliefs are $X_0 = \{\sim s(C), s(JB)\}$. JB has decided to quit smoking, and therefore tells Cooper that he is now a nonsmoker. Cooper reacts by applying the rule

$$r_1: \quad F \leftarrow \mathsf{recvMsg}[\text{tell}(F),\ S],\ -\sim reliable(S)$$

i.e. he updates his beliefs accordingly:

$$X_1 = r_1(X_0) = \mathsf{Upd}(X_0, \sim s(JB)) = \{\sim s(C), s(JB), \sim s(JB)\}$$

Consequently, Cooper no longer believes that JB smokes: $X_1 \not\vdash s(JB)$, but he also does not believe that JB is a nonsmoker now: $X_1 \not\vdash \sim s(P)$, i.e. he has become skeptical with respect to JB's smoking. However, after observing that JB smokes, he believes again in JB's smoking, since he trusts his own observations more than any news being told by someone else, consequently processing them according to

$$r_2: \quad *F \leftarrow \mathsf{recvMsg}[\text{observed}(F),\ \mathit{vision}]$$

leading to the belief change

$$X_2 = r_2(X_1) = \mathsf{Upd}(X_1, *s(JB)) = \{\sim s(C), s(JB)\}$$

5 Specification, Execution and Verification of Single-Agent Systems

In this section, we first define our vivid agent model, and the operational semantics of private reaction rules. Then, we describe our execution model in the form of a meta-logic program, followed by a brief presentation of the transition system semantics of vivid agents, which is the basis for a verification formalism using concepts from Hoare logic and linear time temporal logic.

A vivid agent consists of

1. a mental state $M = \langle X, TL, PTL, CI \rangle$, comprising
   (a) a knowledge base X representing its *beliefs*,
   (b) a list of *tasks* $TL$the agent has to carry out once to sovle them, and a list of permanent tasks PTLwhich have to be checked and carried out periodically.
   (c) a list of goal/plan pairs $CI = [G_1/P_1, \ldots, G_n/P_n]$, where P_i is a plan to achieve G_i, called *current intentions*;
2. an event queue EQ recording perception and communication events in the form of incoming mesages,
3. a set AR of *action rules* describing the available action types which together with the goal list and task list are the basis for planning and plan execution, and
4. a set RR of *reaction rules*, consisting of private reaction and interaction rules which code the reactive and communicative behavior.

In a single-agent domain, there is no need for interaction rules, since there is no inter-agent communication.

Every knowledge system $\boldsymbol{K} = \langle L_{\text{KB}}, \vdash, L_{\text{Query}}, \mathsf{Upd}, L_{\text{Input}} \rangle$ can be extended to an *agent knowledge system* $Ag\boldsymbol{K}$ by adding a set of private action schemes Act, and a set of perception event schemes $PEvt$, which together with a knowledge base schema form the schema of a single-agent domain. In $Ag\boldsymbol{K}$, an agent is specified as a 4-tuple $\langle M, EQ, AR, RR \rangle$, consisting of a *mental state* M, an event queue EQ, and a set of action and reaction rules

$$AR \subseteq L_{\text{Act}} \times L_{\text{Input}} \times L_{\text{Query}}$$
$$RR \subseteq L_{\text{Act}} \times L_{\text{Input}} \times L_{\text{PEvt}} \times L_{\text{Query}}$$

The *mental state* M of an agent comprises its beliefs, its tasks, and its current intentions: $M = \langle X, TL, PTL, CI\rangle$, such that $X \in L_{\mathrm{KB}}$, and $TL, PTL \subseteq L^0_{\mathrm{Query}}$, and for each goal/plan pair G/P from CI, $G \in L^0_{\mathrm{Query}}$, and P is a sequence of action rules, $P \in AR^*$. The tuple $\langle TL, PTL, CI\rangle$ is called *intention state*. The *agent state* consist of the mental state and the event queue of the agent, i.e. it is a pair $\langle M, EQ\rangle$, such that EQ is a sequence (list) of instantiated event expressions from L^0_{Evt}. 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.

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

5.1 Operational Semantics of Private Reaction Rules

We briefly describe the operational semantics of private reaction rules now making the parameters of private actions and of events explicit by writing $\alpha(V) \in L_{\mathrm{Act}}$, and $\varepsilon(U) \in L_{\mathrm{Evt}}$, such that U, V are suitable lists of parameters. For brevity, we use the following notation for a private reaction rule:

$$\mathsf{do}(\alpha(V)),\ E \ \leftarrow\ \mathsf{rM}[\varepsilon(U), S],\ C$$

Let u be any list of ground terms, such that $\varepsilon(u)$ is an admissible event expression. Whenever there is a new message, say $\varepsilon(u)$, popped from the event queue, i.e. $\mathsf{rM}[\varepsilon(u), s]$ holds, where s is the sender responsible for the event, it is matched with the event condition of all suitable reaction rules with event condition $\mathsf{rM}[\varepsilon(U), S]$ such that $S = s$, and for some substitution σ_0, we have $U\sigma_0 = u$. All shared variables are instantiated accordingly in the precondition formula C (including the variable S for the sender), yielding the substitution variant $C\sigma_1$ where $\sigma_1 = \sigma_0 \cup \{S/s\}$. In this way one obtains a set of action rules

$$\mathsf{do}(\alpha(V\sigma_1)),\ E\sigma_1 \ \leftarrow\ C\sigma_1$$

which is then fired for every inferable ground instance $C\sigma_1\sigma_2$ of $C\sigma_1$ leading both to the update of the original knowledge state X,

$$X' = \mathsf{Upd}(X, \{E\sigma_1\sigma_2 : X \vdash C\sigma_1\sigma_2\})$$

and to the set of concurrent private actions,

$$\{\mathsf{do}(\alpha(V\sigma_1\sigma_2)) : X \vdash C\sigma_1\sigma_2\}$$

After both the update actions and the private actions of all applicable rules have been carried out, the current entry pointer of the event queue moves to the next entry.

The concurrent execution of all reactions triggered by the event $\langle\varepsilon(u), s\rangle$ corresponds to the following transition of the agent state $\langle M, EQ\rangle$:

$$RR(M,\ [\langle\varepsilon(u), s\rangle|RestQ]) = \langle RR_{\varepsilon(u),s}(M),\ RestQ\rangle$$

where the set $RR_{\varepsilon(u),s}$ of action rules triggered by the event $\langle\varepsilon(u), s\rangle$ is defined as

$$RR_{\varepsilon(u),s} = \{\mathsf{do}(\alpha(V\sigma)), E\sigma \leftarrow C\sigma \ \ :\ \ [\mathsf{do}(\alpha(V)), E \leftarrow \mathsf{rM}[\varepsilon(U), S], C] \in RR \ \ \&\ \ u = U\sigma_0\ \ \&\ \ \sigma = \sigma_0 \cup \{S/s\}\}$$

and the concurrent execution of a set of action rules R in the mental state M yields the update

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

where $\dot{R} = \{\dot{r} : r \in R\}$ denotes the epistemic projection of R.

Example 5 *In the reactive elevator example above, mental states comprise only beliefs, and the reactive behavior is specified by $RR = \{\alpha_1, \ldots, \alpha_5\}$, yielding the following transitions:*

$$\begin{aligned}
RR(\{at(3)\}, [\text{req}(5)]) &= \langle RR_{\text{req}(5)}(\{at(3)\}), []\rangle \\
&= \langle \alpha_1(\{at(3)\}), []\rangle = \langle \{req(5),\ up(3)\}, []\rangle \\
RR(\{req(5),\ up(3)\}, [\text{arrAt}(4)]) &= \langle RR_{\text{arrAt}(4)}(\{req(5),\ up(3)\}), []\rangle \\
&= \langle \alpha_3(\{req(5),\ up(3)\}), []\rangle = \langle \{req(5),\ up(4)\}, []\rangle \\
RR(\{req(5),\ up(4)\}, [\text{arrAt}(5)]) &= \langle RR_{\text{arrAt}(5)}(\{req(5),\ up(4)\}), []\rangle \\
&= \langle \alpha_5(\{req(5),\ up(4)\}), []\rangle = \langle \{at(5)\}, []\rangle
\end{aligned}$$

5.2 Defining an Execution Model 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 shall 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 calcuclus* 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 single-agent system:

```
cycle( [], KB, [G | RestTL], [])          % GOAL-DIRECTED PLANNING
← not demo( KB, G),                 % check if new goal is not satisfied
   plan( P, KB, G),                 % generate a plan for it
   getNewEvents( Evts),        % check for events which might have occured meanwhile
   cycle( Evts, KB, [RestTL], [G/P]).

cycle( [], KB, TL, [ G/[R] | RestCI ])       % PLAN EXECUTION
← action( R, Act, Eff, Cond),       % 'R' is the action rule name.
   demo( KB, Cond),
   assimilate( KB, Eff, KB′),
   do( Act),
   getNewEvents( Evts),
   cycle( Evts, KB′, TL, RestCI).
```

```
cycle( [], KB, TL, [ G/[R | RestP] | RestCI ])       % PLAN EXECUTION
← action( R, Act, Eff, Cond),       % 'R' is the action rule name.
   demo( KB, Cond),
   assimilate( KB, Eff, KB′),
   do( Act),
   getNewEvents( Evts),
   cycle( Evts, KB′, TL, [ G/RestP | RestCI ]).

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

cycle( [Evt | RestEvts], KB, TL, CI )       % REACTIVE BEHAVIOR
← reaction( Act, Eff, Evt, Cond),
   demo( KB, Cond),
   assimilate( KB, Eff, KB′),
   do( Act),
   getNewEvents( NewEvts),
   append( RestEvts, NewEvts, Evts),
   cycle( Evts, KB′, TL, CI).
```

The last cycle clause provides for immediate reactions in response to events. The reactive behavior of each agent is specified by its set of reaction rules represented as 4-tuples in the table *reaction*. An incoming event message is popped from the message queue, and subsequently matched with suitable reaction rules. If their precondition is satisfied with respect to the current knowledge state, expressed by *demo(KB, Cond)*, its epistemic effects are assimilated into the KB. The *demo* and *assimilate* meta-predicates are formally defined on the basis of our knowledge system concepts of inference and update by

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

After the assimilation has succeeded, the action is performed, *do(Act)*, by means of an appropriate procedure call. If the action succeeds, *cycle* looks for new events, and starts over with the updated knowledge base KB′.

The first three cycle clauses (goal-directed planning and plan execution) are responsible for the deliberative and proactive behavior of the agent. Plans are generated in order to solve tasks. The plan execution cycle clause is responsible for the proactive behavior of the agent. 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), goal-directed planning starts (i.e. there is time now for deliberation). This leads to the construction of a new current plan.

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. This is an issue of future research.

5.3 Agent Queries and Mental Inputs

We extend the query language of an agent's knowledge system in order to allow also querying the current perceiption events (i.e. received messages).

Definition 3 (Agent Queries) *Let* $L' = L_{\mathrm{Query}} \cup \{\mathsf{T}F : F \in L_{\mathrm{Query}}\} \cup \{\mathsf{rM}[\varepsilon] : \varepsilon \in L_{\mathrm{Evt}}\}$. *Then,* $L_{\mathcal{AQ}} = L'(-, \wedge, \vee)$ *denotes the set of agent query formulas.*

Inference for agent queries on the basis of an agent state $A = \langle M, EQ\rangle$ with $M = \langle X, TL, CI\rangle$ is defined as follows.

$$\begin{array}{lcll} A & \vdash & F & \text{iff} \quad X \vdash F \\ A & \vdash & \mathsf{T}F & \text{iff} \quad F \in TL \\ A & \vdash & -\mathsf{T}F & \text{iff} \quad F \notin TL \\ A & \vdash & \mathsf{rM}[\varepsilon] & \text{iff} \quad \varepsilon = head(EQ) \\ A & \vdash & -\mathsf{rM}[\varepsilon] & \text{iff} \quad \varepsilon \neq head(EQ) \end{array}$$

Likewise we extend the input language by allowing also for mental inputs, such as tasks.

Definition 4 (Mental Iputs) *Let* $L' = L_{\mathrm{Input}} \cup \{\mathsf{T}F : F \in L_{\mathrm{Query}}\} \cup \{-\mathsf{T}F : F \in L_{\mathrm{Query}}\}$. *Then,* $L_{\mathrm{MInp}} = L'(\wedge)$ *denotes the set of mental input formulas.*

Updating with mental inputs amounts to adding or retracting tasks:

$$\begin{array}{rcl} \mathsf{Upd}(M, F) & = & \langle \mathsf{Upd}(X, F), TL, CI\rangle \\ \mathsf{Upd}(M, \mathsf{T}F) & = & \langle X, TL + [F], CI\rangle \\ \mathsf{Upd}(M, -\mathsf{T}F) & = & \langle X, TL - F, CI\rangle \end{array}$$

where $TL - F$, the result of retracting F from TL, is defined in the obvious way (assuming that F occurs in TL).

5.4 Agent Systems as State Transition Systems

In this section, we generalize concepts and methods from the theory of concurrent systems (see, e.g., [Sha93]), and adapt them to agent systems. Modeling agent systems as state transition systems allows to reason about them using Hoare logic and a fragment of linear-time temporal logic.

The temporal behavior of an agent system can be described by means of a *labeled transition system* where we have five kinds of transitions transforming an agent state $A = \langle M, EQ\rangle$ with $M = \langle X, TL, CI\rangle$:

(Perception) An incoming perception event message ε yields the transition $A \xrightarrow{\varepsilon} A'$, where $EQ' = EQ + [\varepsilon]$, and $M' = M$.

(Reaction) A (possibly concurrent) reaction RR_ε in response to an event $\varepsilon = head(EQ)$ yields the transition $A \xrightarrow{RR_\varepsilon} A'$, where $A' = \langle RR_\varepsilon(M),\ tail(EQ)\rangle$.

(Planning) If $CI = []$, or for all G/P in CI, $head(P)$ is not executable, and $TL = [G|RestT]$, then deliberative planning yields the transition $A \xrightarrow{plan(G)} A'$, where $A' = A$, except that $TL' = RestT$, and $CI' = [G/P] + CI$ such that P is a plan to achieve G, i.e. $P(X) \vdash G$.

(Action) If $CI = [G/P|RestCI]$, a planned action $r = head(P)$ which is executable, i.e. $X \vdash Cond(r)$, yields the transition $A \xrightarrow{r_+} A'$, where $A' = A$, except that $X' = \dot{r}(X)$, and if $RestCI \neq []$, then $CI' = [G/tail(P)|RestCI]$, otherwise $CI' = RestCI$.

(Replanning) If $G/P = head(CI)$, a planned action $r = head(P)$ which is not executable, i.e. $X \nvdash Cond(r)$, yields the transition $A \xrightarrow{r_-} A'$, where $A' = A$, except that $P' = N + P$ if there is a plan $N \in AR^*$ such that $N(X) \vdash Cond(r)$, or otherwise $CI' = tail(CI) + [G/P]$ (i.e. the currently unachievable goal G is postponed).

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 satisfied, i.e. $CI = []$, or cannot be executed. 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).

Notice that we assume that planning (resp. replanning) is always successful, and that planning, (planned) actions, and (concurrent) reactions can be atomically executed.

Definition 5 (Execution History) *An* agent execution history *is a chain of state transitions $A_0 \xrightarrow{\tau_0} A_1 \xrightarrow{\tau_1} \ldots$, 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 an agent system $\mathcal{A}$ is denoted by* $\mathrm{Hist}(\mathcal{A})$.

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

5.5 Assertional Reasoning

Definition 6 (Hoare Triples) *Let t be a transition of $\mathcal{A}$, and let $F, G \in L^0_{\mathcal{AQ}}$ be agent queries. A* Hoare triple *has the form $F \overset{t}{\Rightarrow} G$, and the following meaning: for any agent state A, $A \vdash F$ implies $t(A) \vdash G$, where $t(A)$ is the state obtained by applying the transition t to A.*

In order to be able to state fairness assumptions for the environment of an agent, and to prove correctness assertions with respect to an agent specification, we define two operators Inv, and $\rightsquigarrow$, which represent a fragment of linear-time temporal logic.[10]

Definition 7 (Invariance and Leads-To Assertions) *Let $H \in \mathrm{Hist}(\mathcal{A})$, and let $F, G \in L^0_{\mathcal{AQ}}$. We say that H satisfies the* invariance assertion $\mathsf{Inv}(F)$*, symbolically $H \models \mathsf{Inv}(F)$, iff for all $A \in H$, $A \vdash F$. We say that H satisfies the* leads-to assertion *$F \rightsquigarrow G$, symbolically $H \models F \rightsquigarrow G$, if for all states $A \in H$ with $A \vdash F$, there is a later state $A' \in H$, $A \leq A'$, such that $A' \vdash G$. An assertion ϕ holds in $\mathcal{A}$, symbolically $\mathcal{A} \models \phi$, if for all $H \in \mathrm{Hist}(\mathcal{A})$, $H \models \phi$.*

Definition 8 (Single-Agent Domain Description) *A single-agent domain description consists of an agent specification $\mathcal{A}$ and a set of assertions Φ expressing assumptions about the environment of the agent. Such a domain description satisfies an assertion ϕ, if ϕ is satisfied by all histories of $\mathcal{A}$ satisfying Φ. Formally,*

$$\mathcal{A}, \Phi \models \phi \quad \textit{iff} \quad \text{f.a. } H \in \mathrm{Hist}(\mathcal{A}) : H \models \Phi \Rightarrow H \models \phi$$

In the reactive elevator domain description, for instance, the environment asumptions have been

$$\Phi = \{up(x) \rightsquigarrow \mathsf{rM}[\mathrm{arrAt}(x+1)],\ down(x) \rightsquigarrow \mathsf{rM}[\mathrm{arrAt}(x-1)]\}$$

5.6 Proof Rules for Assertional Reasoning

Claim 1 *Let $\mathcal{A} = \langle M_0, EQ_0, AR, RR\rangle$ be an agent system, such that $A_0 = \langle M_0, EQ_0\rangle$ is the initial state. Then, $\mathcal{A} \models \mathsf{Inv}(F)$, if $A_0 \vdash F$, and for every transition t of $\mathcal{A}$, it holds that $F \stackrel{t}{\Rightarrow} F$.*

This can be proved by straightforward induction on the length of histories.

The correctness of the following proof rule is obvious.

Claim 2 (A1) *$\mathcal{A} \models F \rightsquigarrow G$, if $F \stackrel{t}{\Rightarrow} G$ holds for every transition t of $\mathcal{A}$.*

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

$$\frac{F \rightsquigarrow H \quad H \rightsquigarrow G}{F \rightsquigarrow G} \qquad \frac{F \rightsquigarrow H \quad G \rightsquigarrow H}{F \vee G \rightsquigarrow H}$$

For a comprehensive treatment 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.

Claim 3 (A2) *Let $\langle W, <\rangle$ be a wellfounded ordering. Then, $\mathcal{A} \models F \rightsquigarrow G$, if*

$$\begin{array}{ll} (i) & \mathcal{A} \models F \rightsquigarrow G \vee H(m) \quad \textit{for some } m \in W \\ (ii) & \mathcal{A} \models H(m) \rightsquigarrow G \vee H(n) \quad \textit{for some } n < m \end{array}$$

Notice that this rule was implicitly used in our correctness proof of the reactive elevator specification (see also below).

[10] See [MP92], where $\Box F$ corresponds to $\mathsf{Inv}(F)$, and $\Box(F \rightarrow \Diamond G)$ corresponds to $F \rightsquigarrow G$.

5.7 Examples of Verification

Suppose we want to prove that the reactive elevator system $\mathcal{A}$, with $A_0 = \langle X_0, EQ_0 \rangle = \langle \{at(m)\}, [] \rangle$, never attempts to move downwards from the 0th floor. This safety property can be expressed by the following invariance assertion.

Claim 4 $\mathcal{A} \models \mathsf{Inv}(-down(0))$.

Proof: Obviously, $A_0 \vdash -down(0)$. It remains to show for every reaction rule α_i $(i = 1, \ldots, 5)$, that $-down(0) \stackrel{\alpha_i}{\Rightarrow} -down(0)$. Since $A \vdash -down(0)$ iff $down(0) \notin X$, we have to show that $down(0) \notin X$ implies $down(0) \notin \alpha_i(X)$. This is obvious for α_1, α_3, and α_5, since these transitions do not add any information about *down*. Suppose that $down(0) \in \alpha_2(X)$, implying that for some x,

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

which is a contradiction. Finally, suppose that $down(0) \in \alpha_4(X)$, implying that

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

which is again a contradiction. □

Claim 5 *For $k < m$,*

$$\begin{array}{lcl} \boldsymbol{A} & \models & req(k) \wedge down(m) \leadsto at(k) \vee down(m-1) \\ \boldsymbol{A} & \models & down(m) \leadsto at(k) \vee down(m-1) \end{array}$$

By the proof rule (A2), this yields the desired conclusion that

$$req(k) \wedge down(m) \leadsto at(k)$$

6 Multi-Agent Systems

An agent knowledge system $Ag\boldsymbol{K}$ can be extended to a homogeneous *multiagent knowledge system* $MA\boldsymbol{K}$ by adding a set of communication event schemas *CEvt* which represents the language of inter-agent communication. In $MA\boldsymbol{K}$, the knowledge base of an agent is a 4-tuple $\mathcal{A} = \langle M, EQ, AR, RR \rangle$, such that AR and RR now also comprise communicative (re)action rules:

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

A 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: $S = \langle A_1, \ldots, A_m \rangle$.

In the next section, we shall show that $\mathcal{S}$ is not only a formal, but also an *executable specification* of a multi-agent system.

6.1 Operational Semantics of Action and Reaction Rules

We briefly describe the operational semantics of interaction rules

$$\mathsf{sM}[\eta(T), R],\ E \ \leftarrow\ \mathsf{rM}[\varepsilon(U), S],\ C$$

Let u be any list of ground terms, such that $\varepsilon(u)$ is an admissible event expression. Whenever there is a new message, say $\varepsilon(u)$, popped from the event queue, i.e. $\mathsf{rM}[\varepsilon(u), s]$ holds, where s is the sender responsible for the event, it is matched with the event condition of all suitable reaction and interaction rules with event condition $\mathsf{rM}[\varepsilon(U), S]$ such that $S = s$, and for some substitution σ_0, we have $U\sigma_0 = u$. All shared variables are instantiated accordingly in the precondition formula C (including the variable S for the sender), yielding the substitution variant $C\sigma$ where $\sigma = \sigma_0 \cup \{S/s\}$. In this way one obtains a set of communicative action rules

$$\mathsf{sM}[\eta(T\sigma), R\sigma],\ E\sigma \ \leftarrow\ C\sigma$$

which is then fired for every inferable ground instance $C\sigma\tau$ of $C\sigma$ leading both to the update of the original KB state X,

$$X' = \mathsf{Upd}(X, \{E\sigma\tau : X \vdash C\sigma\tau\})$$

and to the set of concurrent communication acts

$$\{\mathsf{sM}[\eta(T\sigma\tau), R\sigma\tau] : X \vdash C\sigma\tau\}$$

After the actions of all applicable rules have been carried out, the current entry pointer of the event queue moves to the next entry.

The concurrent execution of a set of action rules AR in the mental state M yields the update

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

where $\dot{AR} = \{\dot{r} : r \in R\}$. In addition, we shall need the set of all communication events created by a set of action rules AR applicable in M, and addressed to agent j

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

We can now define the concurrent execution of a set of actions by agent $\mathcal{A}_i$ as the application of the resp. set of action rules AR to the multi-agent system state S at component i:

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

where

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

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

Definition 9 (Act) *When agent i executes a planned action from its current plan, we obtain the following transition:*

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

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

$$\mathrm{Act}(i, S) \ = \ head(P)(i,\ \langle \ldots, \langle X_i,\ TL_i,\ RestCI_i,\ EQ_i\rangle, \ldots\rangle)$$

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

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

where $A\sigma = \mathsf{sM}[\eta(T\sigma), R\sigma]$, or $A\sigma = \mathsf{do}(\alpha(V\sigma))$. 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 10 (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 6 *In the communicating elevator scenario, where $\mathcal{S} = \langle \mathcal{A}_1, \mathcal{A}_2\rangle$, if the initial agent states are $A_1^0 = \langle\{at(1)\}, [\mathrm{req}(2)]\rangle$, resp. $A_2^0 = \langle\{at(2)\}, [\mathrm{req}(3)]\rangle$, and the reaction rules are $RR_1 = \{a_1, \ldots, a_9\}$, resp. $RR_2 = \{b_1, \ldots, b_9\}$, we obtain the following transitions:*

$$\begin{aligned}\mathrm{React}(1, S_0) &= RR_1^{\mathrm{req}(2)}(1, \langle\{at(1)\}, []\rangle, \langle\{at(2)\}, [\mathrm{req}(3)]\rangle\rangle\\ &= \langle\langle a_1(\{at(1)\}), []\rangle, \langle\{at(2)\}, [\mathrm{req}(3), \mathrm{reqPerm}]\rangle\rangle\\ &= \langle\langle\{at(1), req(2)\}, []\rangle, \langle\{at(2)\}, [\mathrm{req}(3), \mathrm{reqPerm}]\rangle\rangle\\ \mathrm{React}(2, S_0) &= RR_2^{\mathrm{req}(3)}(2, \langle\langle\{at(1)\}, [\mathrm{req}(2)]\rangle, \langle\{at(2)\}, []\rangle\rangle\\ &= \langle\langle\{at(1)\}), [\mathrm{req}(2)]\rangle, \langle b_6(\{at(2)\}), []\rangle\rangle\\ &= \langle\langle\{at(1)\}), [\mathrm{req}(2)]\rangle, \langle\{req(3), up(2)\}), []\rangle\rangle\\ S_1 &= \mathrm{React}(1, \mathrm{React}(2, S_0)) = \mathrm{React}(2, \mathrm{React}(1, S_0))\\ &= \langle\langle\{at(1), req(2)\}), []\rangle, \langle\{req(3), up(2)\}), [\mathrm{reqPerm}]\rangle\rangle\\ S_2 &= \mathrm{React}(2, S_1)\\ &= \langle\langle\{at(1), req(2)\}), [\mathrm{grPerm}]\rangle, \langle\{req(3), up(2)\}), []\rangle\rangle\\ S_3 &= \mathrm{React}(1, S_2)\\ &= \langle\langle\{up(1), req(2)\}), []\rangle, \langle\{req(3), up(2)\}), []\rangle\rangle\end{aligned}$$

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.*

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

6.2 Multi-Agent Systems as State Transition Systems

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*. As in the case of single-agent systems, we have five kinds of transitions transforming a mulit-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 (possibly concurrent) reaction of agent i in response to an event $\varepsilon = head(EQ_i)$ yields the transition $S \xrightarrow{i,RR_i^{\varepsilon}} S'$, where $S' = \mathrm{React}(i, S)$.

(Planning) If $CI_i = []$, or for all G/P in CI_i, $head(P)$ is not executable, then deliberative 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), $TL_i' = tail(TL_i)$.

(Action) If $G/P = head(CI_i)$, an action $r = head(P)$ which is executable, i.e. $M_i \vdash Cond(r)$, and planned by agent i, yields the transition $S \xrightarrow{i,r_+} S'$, where $S' = \mathrm{Act}(i, S)$.

(Replanning) If $G/P = head(CI_i)$, a planned action $r = head(P)$ which is not executable by agent i, i.e. $M_i \not\vdash Cond(r)$, 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 $CI_i' = tail(CI_i) + [G/P]$ (i.e. the currently unachievable goal G is postponed).

Notice that we assume that planning and plan execution may only take place if there is no need to react in response to an event, and that planning, (planned) actions, and (concurrent) reactions can be atomically executed.

Definition 11 (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 12 (Enabled Agent) *We say that an agent $\mathcal{A}_i$ is enabled if its event queue EQ, its list of current intentions CI, or its task list TLis nonempty, or if one of its permanent tasks has to be (re-)achieved: for some $G \in PTL_i : X_i \not\vdash G$.*

Definition 13 (Fairness) [11]

1. *A finite history of a multi-agent system $\mathcal{S}$ is called* fair, *if in its final state no agent is enabled.*

[11] In [Sha93], this is called *weak fairness* as opposed to a stronger notion of fairness which we shall not consider.

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 $\mathrm{FHist}(\mathcal{S})$.

6.3 Assertional Reasoning

Definition 14 ($\mathcal{S}$-Queries) *Let $\mathcal{S}$ be a homogeneous multi-agent system, and let $L = \{\mathsf{B}_i F : F \in L_{\mathrm{Query}}\} \cup \{\mathsf{T}_i F : F \in L_{\mathrm{Query}}\} \cup \{\mathsf{PT}_i F : F \in L_{\mathrm{Query}}\} \cup \{\mathsf{rM}_i[\varepsilon] : \varepsilon \in L_{\mathrm{Evt}}\}$. Then, $L_{\mathcal{S}\mathrm{Q}} = L(-, \wedge, \vee)$ denotes the set of multi-agent system 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.

$$
\begin{array}{lcll}
S & \vdash & \mathsf{B}_i F & \text{iff} \quad X_i \vdash F \\
S & \vdash & -\mathsf{B}_i F & \text{iff} \quad X_i \nvdash F \\
S & \vdash & \mathsf{T}_i F & \text{iff} \quad F \in TL_i \\
S & \vdash & -\mathsf{T}_i F & \text{iff} \quad F \notin TL_i \\
S & \vdash & \mathsf{PT}_i F & \text{iff} \quad F \in PTL_i \\
S & \vdash & -\mathsf{PT}_i F & \text{iff} \quad F \notin PTL_i \\
S & \vdash & \mathsf{rM}_i[\varepsilon] & \text{iff} \quad \varepsilon = head(EQ_i) \\
S & \vdash & -\mathsf{rM}_i[\varepsilon] & \text{iff} \quad \varepsilon \neq head(EQ_i)
\end{array}
$$

As for single-agent systems, we get the analogous notion of Hoare triples also for multi-agent systems.

Definition 15 (Hoare Triples) *Let t be a transition of $\mathcal{S}$, and let $F, G \in L^0_{\mathcal{S}\mathrm{Q}}$ be $\mathcal{S}$-queries. A* Hoare triple *has the form $F \overset{t}{\Rightarrow} G$, and the following meaning: for any state S, $S \vdash F$ implies $t(S) \vdash G$, where $t(S)$ is the state obtained by applying the transition t to S.*

Assertions about multi-agent systems refer only to fair histories.

Definition 16 (Invariance and Leads-To Assertions) *Let $H \in \mathrm{Hist}(\mathcal{S})$, and let $F, G \in L^0_{\mathcal{S}\mathrm{Q}}$. We say that H satisfies the* invariance assertion $\mathsf{Inv}(F)$, *symbolically $H \models \mathsf{Inv}(F)$, iff for all $S \in H$, $S \vdash F$. We say that H satisfies the* leads-to assertion *$F \rightsquigarrow G$, symbolically $H \models F \rightsquigarrow G$, if for all states $S \in H$ with $S \vdash F$, there is a later state $S' \in H$, $S \leq S'$, such that $S' \vdash G$. An assertion ϕ holds in $\mathcal{S}$, symbolically $\mathcal{S} \models \phi$, if for all fair histories $H \in \mathrm{FHist}(\mathcal{S})$, $H \models \phi$.*

Definition 17 (Multi-Agent Domain Description) *A multi-agent domain description $\mathcal{S}, \Phi$ consists of a multi-agent system specification $\mathcal{S}$, and a set of assertions Φ expressing assumptions about the environment of the agents. Such a domain description satisfies an assertion ϕ, if ϕ is satisfied by all fair histories of $\mathcal{S}$ satisfying Φ. Formally,*

$$\mathcal{S}, \Phi \models \phi \quad \textit{iff} \quad \text{f.a. } H \in \mathrm{FHist}(\mathcal{S}) : H \models \Phi \Rightarrow H \models \phi$$

Example 7 *For the communicating elevators system from example 4.2.1, we might want to prove the safety property, that it is never the case that the lower elevator goes up to the second floor while the upper one goes down to the second floor, or is currently halting at the second floor, expressed as*

$$\mathcal{S}, \Phi \models \mathsf{Inv}(-[\mathsf{B}_1 up(1) \wedge (\mathsf{B}_2 down(3) \vee \mathsf{B}_2 at(2))])$$

where $\Phi = \{\mathsf{B}_i up(k) \rightsquigarrow \mathsf{rM}_i[\mathrm{at}(k+1)],\ \mathsf{B}_i down(k) \rightsquigarrow \mathsf{rM}_i[\mathrm{at}(k-1)]\}$. *Also we might want to prove*

$$\mathcal{S}, \Phi \models \mathsf{rM}_i[\mathrm{req}(2)] \rightsquigarrow \mathsf{B}_i at(2)$$

expressing the progress property that both elevators will eventually serve every request from floor 2.

7 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. 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 interleave the single planning steps with perception and reaction which would amount to interruptable planning. This still has to be investigated. It was also out of scope of the present paper to investigate the possibility of (re)action rules with *mental* preconditions and effects, such as rules for communicating new tasks, or dropping old goals.

8 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 action rules may be compared with Shoham's *conditional action statements*. 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 [FW94], a first-order temporal logic is proposed for the purpose of verification.

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 also knowledge about the occurence of actions and about the precedence of situations), and b) action rules in the form of *static effect laws*. A domain description can be queried about the validity of 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 occurence of actions. In other respects, however, it is less powerful: 1) while in our modelling of scalable agents any knowledge system may be used, they restrict state knowledge to atomic sentences like in the knowledge system $\boldsymbol{A}$ of relational databases (thus, the epistemic effects of actions have to be atomic); 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.

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.[12] 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. Specifications in these logics are completely abstract and, as opposed to

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

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.

In recent years, 'hybrid' agent architectures combining reactive and deliberative behavior have become increasingly important (see, e.g., [BS92, 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.

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, proactive, reactive and communicative behavior in this framework. We also outlined methods of assertional reasoning for single-agent and multi-agent systems. We hope that our model can serve as a basis for the formalization of more high-level concepts such as values, emotions, social structures, social laws, etc.

Acknowledgements: Thanks to J.J. Alferes, J.N. Aparicio, C. Damasio, R. Li, and L.M. Pereira for stimulating discussions in our working group.

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.

[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 Centre, 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.

[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.

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

[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.

[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.

[Wag94a] G. Wagner: *Vivid Logic – Knowledge-Based Reasoning with Two Kinds of Negation*, Springer Lecture Notes in AI **764** (1994).

[Wag94b] G. Wagner: *Disjunctive, Deductive and Active Knowledge Bases*, LWI Report 22/1994, Freie Universität Berlin; available as `ddakbs.dvi.gz`, resp. `ddakbs.ps.gz`, via ftp from `ftp.inf.fu-berlin.de` in the directory `/pub/reports/wagner` (notice that the file has to be 'gunzip'ed).

[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.

[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.

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

Table 1: Two elevators in a shaft.