An Agent Architecture that Unifies Rationality with Reactivity

Robert Kowalski and Fariba Sadri
Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, UK
{rak,fs}@doc.ic.ac.uk

Abstract

In this paper we analyse the similarities and differences between rational and reactive agent architectures, and propose a unified architecture that aims to capture both as special cases. For this purpose we employ a proof procedure as the thinking component of the agent. The proof procedure combines definitions with integrity constraints. It uses definitions, in logic programming manner, for "rational" reduction of goals to subgoals. It uses integrity constraints for reactive, condition-action rule behaviour. We also employ a resource-bounded formalisation of the proof procedure, so that the agent's thinking can be interrupted and resumed to record and assimilate observations as input and to execute atomic actions as output.

1 Introduction

The traditional notion of a rational agent in Artificial Intelligence focuses on the agent's reasoning process and downplays its interaction with the environment. This notion has been challenged in recent years by critics, such as Brooks [1991], who argue that, on the contrary, intelligence resides in the ability of an agent to react appropriately to changes in its environment.

There have been many attempts, perhaps the most notable of which is that at DFKI [Fisher *et al.*, 1996], to combine rational and reactive capabilities within a single agent architecture. All of these are hybrid architectures, which integrate the two capabilities without attempting to unify them.

In this paper we propose a logic-based agent architecture that unifies rationality with reactivity. Rationality is achieved by using definitions to reduce goals to subgoals. Reactivity is achieved both by using integrity constraints to simulate condition-action rules and by formulating the proof procedure in such a way that it can be interrupted to make observations and execute actions.

The rest of the paper is structured as follows. Sections 2 and 3 discuss the cycles of rational and reactive agents, respectively. Section 4 outlines the proof procedure of the unified agent. Section 5 defines the unified cycle that uses the proof procedure as its thinking component. Section 6 argues that the unified cycle subsumes the other two. Section 7 shows how complex actions can be defined in terms of lower-level actions. Section 8 concludes.

2 Rational Agents

A traditional database or knowledge base contains information in symbolic, often logical form. Such a knowledge base can receive inputs, which are updates or queries. It may check whether updates satisfy *integrity constraints*. However, the only output it can generate is in the form of answers to queries. The amount of resources it might need to deduce such answers is unbounded.

The traditional approach to robotics in Artificial Intelligence uses such a knowledge base to represent an agent's beliefs. In addition to beliefs, the agent has goals, which it seeks to achieve by constructing a *plan*, which in the simplest case consists of a sequence of atomic actions. Only after generating one or more complete plans does the agent choose one and begin to execute it.

Thus planning is separated from plan execution. Because the world changes and because the agent may have incomplete or incorrect knowledge of its environment, actions which are planned may fail when the attempt is made to execute them. For this reason, the agent needs to update its knowledge base after every action and replan if necessary.

We will argue later that updates, queries and other requests can all be treated uniformly as observations that are input from the environment.

At the top-most level, the cycle of a rational agent can be expressed in the form:

To cycle at time T,

i) observe any input at time T,
ii) record any such input,
iii) check the inputs for satisfaction of integrity constraints by reasoning forwards from the input,
iv) solve (or re-solve) goals by constructing a plan (or replanning), using for steps (iii) and (iv) a total of R units of time,
v) select a plan from among the alternatives, and select from the plan an atomic action which can be executed at time T+R+2,
vi) execute the selected action at time T+R+2 and record the result,
vii) cycle at time T+R+3.

For simplicity, both here and throughout the remainder of the paper, we assume that the total amount of time involved in any steps numbered (i) and (ii) is one unit combined, and for any steps numbered (v) and (vi) is one unit each. We assume also that the agent executes only one atomic action at a time.

The amount of resources *R* such an agent might spend on thinking in steps (iii) and (iv), both checking the integrity of inputs and solving goals, is unbounded.

Critics of the traditional knowledge based approach argue that the lack of a bound on the resources *R* renders the traditional, rational agent architecture unfeasible. They argue, moreover, that generating complete plans before beginning to execute them is generally futile, because of the unpredictability of the environment.

Some rational agent architectures, such as the "practical" realisation of the BDI architecture of Rao and Georgeff [1992], mitigate the shortcomings of the rational agent cycle by restricting the amount of reasoning that is performed in one agent cycle. We employ a similar, but somewhat more general approach in our own unified agent cycle.

3 (Generalised) Reactive Agents

Perhaps the simplest way to achieve reactivity is to regulate the behaviour of an agent by condition-action rules. At any given time, the agent simply matches the conditions of its rules against observations input from the environment and immediately attempts to execute an action in the conclusions of the rules. If several rules have their conditions satisfied, then a choice has to be made from among the resulting actions.

Typically, such a purely reactive agent has no explicit goals. Whatever goals it achieves are implicit in the emergent behaviour of the agent. The agent might have no knowledge base in which to record its observations. The environment in which the agent is situated might serve, in effect, as its own world model. In an extreme case, the agent might not even change its internal state from one cycle to the next.

Example 3.1

The behaviour of a reactive agent could be governed by the following single rule, without the agent having any explicit goals, beliefs or internal state:

if *there is an intruder* ***then*** *raise an alarm.*

The rule is triggered by an input observation of an intruder and generates an action of raising an alarm as output to the environment. The use of the rule achieves a goal, i.e. maintaining security, which is implicit rather than explicit.

We shall argue later that condition-action rules can be viewed as integrity constraints, and that integrity constraints are goals. Thus in this example, the rule of raising an alarm whenever there is an intruder can be regarded as an explicit goal, which is a subgoal of the implicit goal of maintaining security.

Example 3.2

The two rules

if *clear ahead* ***then*** *move forward*
if *obstacle ahead* ***then*** *turn right*

implicitly achieve the goal of moving while avoiding obstacles, without the agent having any explicit representation of the goal. Such rules, possibly augmented with an updatable local state, are typical of the stimulus-response rules of Brooks' [1991] subsumption architecture.

Example 3.3

The generalised condition-action rule

if *Agent requests do(self, Act, T2) at time T1*
and *Agent is friendly*
and *can do(self, Act, T2)*
and *T1<T2*
then *do(self, Act, T2)*

requires more powerful reasoning mechanisms and gives rise to more powerful capabilities than the simpler rules of examples 3.1 and 3.2. The rule is triggered by an input request occurring at (*transaction*) time *T1*, but its remaining conditions are verified by consulting a knowledge base or by matching later inputs observed before time *T2* . The action itself is a *commitment*, which is executed when the transaction time becomes equal to the *domain* time *T2* of the action which is requested.

Notice that the request, which could be a simple query, is treated like any other input. It is for this reason that we do not distinguish between inputs which are updates and inputs which are queries or other requests.

The rule in this example is typical of those which govern the behaviour of agents in Shoham's [1993] AgentO. Such agents possess beliefs, to verify conditions of rules, but no explicit goals, other than the goals associated with ensuring that the rules are obeyed.

In our attempt to unify rationality with reactivity, we formulate the two cycles in a way which emphasises their similarities. For this purpose, we formulate a cycle for the generalised reactive case, which accommodates all of the three examples above within a single generalised condition-action rule interpreter:

To cycle at time T,

i) observe any input at time T,
ii) (optionally) record any such input,
iii) match conditions of condition-action rules with the inputs,
iv) (optionally) verify any remaining conditions of the rules using information in the knowledge base, using for steps (iii) and (iv) a total of R units of time,
v) select an atomic action which can be executed at time T+R+2 from the conclusions of rules all of whose conditions are satisfied,
vi) execute the selected action at time T+R+2, and (optionally) record the result,
vii) cycle at time T+R+3.

Assuming that optional steps are made mandatory, the two cycles share the same outer shell, steps (i), (ii), (vi) and (vii). Otherwise, they differ in three respects.

First, they differ in the amount of resources *R* they use in steps (iii) and (iv). These resources are unbounded for the rational agent, but limited to a small amount for the reactive agent.

Limiting the resources *R* used by the reactive agent is easy in cases, like examples 3.1 and 3.2, where there is no knowledge base and where conditions are verified simply by matching them with the input. It is more difficult in cases, like example 3.3, where reasoning is needed to verify some of the conditions. One practical solution, employed in AgentO, is to restrict the form of the knowledge base, so that conditions can be verified efficiently, within some predefined small amount of time *n*.

Second, in step (v), the two cycles differ in the way they choose atomic actions to execute. The rational agent first selects a complete plan of atomic actions from among the alternatives and then attempts to execute a selected atomic action from the beginning of the plan. The reactive agent selects an atomic action from among the conclusions of the rules all of whose conditions are satisfied. However, both agents perform a form of conflict resolution among alternatives before making a commited choice.

Simple reactive agents, of the kind illustrated in examples 3.1 and 3.2, always select an action in step (v) and attempt to execute it in step (vi), if there is a rule all of whose conditions are satisfied. But more general reactive agents, with condition-action rules of the kind illustrated in example 3.3, might have domain times associated with their actions. In such a case, no action might be selected for execution, even when there is a rule whose conditions are fully satisfied, because the domain time of the action might be later than the transaction time.

The first and second differences between the two cycles are both due to the difference between the unlimited resources required by the rational agent compared with the limited resources employed by the reactive agent.

In our attempt to unify rationality with reactivity, we will define a unified cycle which eliminates the differences due to resource limitations. For this purpose, we employ a resource-bounded, interruptable and resumable formalisation of the proof procedure of the rational agent, outlined in [Kowalski, 1995] and developed by [Davila, 1996].

The **third** difference between the two cycles is their "knowledge representations". This is the difference we will focus on in this paper. The reactive agent employs condition-action rules, optionally augmented with a knowledge base for verifying conditions of rules. The rational agent, on the other hand, employs a knowledge base of beliefs together with goals, which include integrity constraints.

To eliminate the differences due to knowledge representation, we will employ in the unified cycle a proof procedure which combines definitions, for goal reduction, with integrity constraints, for condition-action rule behaviour.

4 The Proof Procedure

Before defining the unified agent cycle, we outline the proof procedure for the propositional case. The proof procedure is based on that of [Fung, 1994; Kowalski and Fung, 1997; Wetzel *et al.*, 1995], and can be viewed as a hybrid of the proof procedures of Console *et al.* [1991] and Denecker and De Schreye [1992]. It was originally developed with the goal of unifying abductive logic programming, constraint logic programming and semantic query optimisation. A similar proof procedure has been used for abductive planning in the event calculus [Kowalski and Sergot, 1986] by Denecker and De Schreye [1993] and Jung *et al*. [1996].

Definitions are used to represent the completely defined predicates of an agent's beliefs. They have the form of generalised logic programs in if-and-only-if form:

(*i*) $G \leftrightarrow D_1 \vee \ldots \vee D_n \quad n \geq 0.$

If $n=0$ the disjunction, called the *body* of the definition, is equivalent to *false*. The disjuncts D_i are conjunctions

(ii) $C_1 \wedge \ldots \wedge C_m \quad m \geq 1$

where each conjunct C_i is either an atom (possibly *true*) or an implication of the form

(iii) $B_1 \wedge \ldots \wedge B_p \rightarrow E_1 \vee \ldots \vee E_q \quad p,q \geq 0.$

where the *conditions* B_i are atomic formulae and the *conclusion* has the same form as the body of a definition, i.e. the disjuncts E_i are conjunctions of the form (ii). If $p=0$ the conditions are equivalent to *true*. If $q=0$ the conclusion is equivalent to *false*. A denial $\neg B$ is represented as an implication of the form $B \rightarrow false$.

Goals statements are used to represent the goals of an agent. They have the same form

$$D_1 \vee \ldots \vee D_n$$

as the body of a definition.

Integrity constraints are implications of the form (iii). Every integrity constraint is a conjunct of every disjunct of every goal statement. Thus, integrity constraints are effectively global goals, which must always be satisfied.

In our unified agent cycle, generalised condition-action rules of the form

if B_1 **and** ... **and** B_p **then**
[do(self, Act_{11}, T_{11}) **and** ... **and** do(self, Act_{1n1}, T_{1n1})]
or ... **or**
[do(self, Act_{m1}, T_{m1}) **and**...**and** do(self, $Act_{mnm,,}T_{mnm}$)]

are represented by integrity constraints of the form

$B_1 \wedge \ldots \wedge B_p \rightarrow$
[do(self, Act_{11}, T_{11}) $\wedge$ ... $\wedge$ do(self, Act_{1n1}, T_{1n1})]
$\vee$ ... $\vee$
[do(self, Act_{m1}, T_{m1}) $\wedge$ ... $\wedge$ do(self, Act_{mnm} ,T_{mnm})]

where all variables are implicitly universally quantified, with scope the implication in which they occur.

The actions in the conclusion of integrity constraints need not be atomic, as in the constraint

$$at(Agent, A, T) \wedge intruder(Agent) \wedge at(self, B, T) \rightarrow do(self, move(B,A), [T, T+10])$$

where *move* is the complex action defined in section 7. Such a constraint cannot be replaced by one involving atomic actions only, because *move* is recursive.

Integrity constraints can also express prohibitions, such as no agent can do two things at the same time:

$$do(Agent, Act_1, T) \wedge do(Agent, Act_2, T) \wedge Act_1 \neq Act_2 \rightarrow false .$$

In the propositional case, the *proof procedure* has four inference rules, each of which transforms a goal statement into an equivalent one. The proof procedure terminates when no further inference rules can be applied. The resulting goal statement is equivalent to the initial goal statement.

When the proof procedure is used for planning, when it terminates, each disjunct of the final goal statement contains an alternative plan, which solves the top-level goal.

The inference rules are:

i) **Goal reduction** uses a definition

$G \leftrightarrow D_1 \vee \ldots \vee D_n$

(**case 1**) to replace a goal statement of the form[1]

$(G \wedge G') \vee D$ by $((D_1 \vee \ldots \vee D_n) \wedge G') \vee D$.

(**case 2**) to replace a goal statement of the form

$((G \wedge G' \rightarrow D') \wedge G'') \vee D$
by $((D_1 \wedge G' \rightarrow D') \wedge \ldots \wedge (D_n \wedge G' \rightarrow D') \wedge G'') \vee D$.

In case 1, goal reduction needs to be followed by the splitting rule, described below, to put the resulting formula in goal statement form:

$(D_1 \wedge G') \vee \ldots \vee (D_n \wedge G') \vee D$.

Case 2 implicitly uses the equivalence

$((B \vee C) \rightarrow A) \leftrightarrow (B \rightarrow A) \wedge (C \rightarrow A)$.

ii) **Splitting** uses distributivity to replace a formula of the form

$(D \vee D') \wedge G$ by $(D \wedge G) \vee (D' \wedge G)$.

iii) **Propagation** replaces a goal statement of the form

$(G \wedge (G \wedge G' \rightarrow D') \wedge G'') \vee D$
by $(G \wedge (G \wedge G' \rightarrow D') \wedge (G' \rightarrow D') \wedge G'') \vee D$.

Logically, when the implication $(G \wedge G' \rightarrow D')$ is an integrity constraint or an implication derived from an integrity constraint and G is the record of an input, propagation contributes to integrity verification.

iv) **Logical equivalences** replace a subformula by another formula which is both logically equivalent and simpler. These include the following equivalences used as rewrite rules:

$G \wedge true \leftrightarrow G$	$G \wedge false \leftrightarrow false$
$D \vee true \leftrightarrow true$	$D \vee false \leftrightarrow D$.
$(true \rightarrow D) \leftrightarrow D$	$(false \rightarrow D) \leftrightarrow true$

[1] Here and elsewhere, we assume the commutativity and associativity of conjunction and disjunction. In particular, when we write a goal statement in the form $(G \wedge G') \vee D$, we intend that $G \wedge G'$ may be any disjunct in the goal statement and that G may be any conjunct in the disjunct.

Definitions are used only for completely defined predicates. Incompletely defined predicates, such as abducibles, constraints and inaccessible, externally defined data, have to be treated differently. Following Denecker and De Schreye [1993], the simplest way to deal with them is to approximate their definitions by means of integrity constraints, which are included, therefore, as conjuncts in every disjunct of every goal statement.

Incompletely defined predicates include input predicates, which describe observations, and output predicates which record the result of actions attempted by the agent. By recording observations as integrity constraints in goal statements, the propagation inference rule implements the triggering of condition-action rules by the input as a special case. The verification of the remaining conditions of condition-action rules is performed by other propagation steps or by case 2 of goal-reduction.

In the general case, the proof procedure can reason with definitions containing certain patterns of universal and existential quantifiers, as illustrated by the following

Example 4.1

Given the definition

maintain-security «
" T[intruder at time T ® $T'[do(self, raise-alarm, T') Ù T < T' < T+10]].

and the goal statement

maintain-security Ù intruder at time 1

goal reduction replaces the goal statement by the new goal statement

" T [intruder at time T ®
$T'[do(self, raise-alarm, T') Ù T < T' < T+10]]Ù intruder at time 1

Propagation, instantiating *T* to *1*, adds the new subgoal

$T'[do(self, raise-alarm, T') Ù 1< T' < 1+10]

to the goal statement.

Notice that the implication obtained as the result of goal reduction has the form of an integrity constraint which represents the condition-action rule of example 3.2.

This example illustrates a general phenomenon: Given a goal statement *G* and goal-reduction rules *KB* we can, in many cases, replace both by an equivalent conjunction of integrity constraints *IC*. In many cases these integrity constraints represent generalised condition-action rules. The integrity constraints are equivalent to *KB* and *G*, in the sense that the proof procedure using *IC* terminates with the same goal statements as the proof procedure using *KB* and *G*.

In effect, the integrity constraints *IC* are obtained from *G* and *KB* by performing at "compile time" inferences that would otherwise need to be performed at "run time". For this reason, the run time execution of *IC* is more efficient than that of *G* and *KB* . The process of transforming *G* and *KB* into *IC*, called partial evaluation [Lloyd and Shepherdson, 1991], is a powerful, but standard logic programming technique.

5 A Unified Agent Cycle

The proof procedure described in the previous section is the thinking component of our unified agent cycle. The top-most level of the cycle has the form:

To cycle at time T,

i) observe any input at time T,
ii) record any such input,
iii) resume the proof procedure by propagating the inputs,
iv) continue applying the proof procedure, using for steps (iii) and (iv) a total of R units of time,
v) select an atomic action which can be executed at time T+R+2 from among the alternatives,
vi) execute the selected action at time T+R+2 and record the result,
vii) cycle at time T+R+3.

Here, as in the reactive agent cycle, the amount of resources *R* available in steps (iii) and (iv) is the smaller of some predefined amount *n* and the time taken to reach a state where no further inference rules can be applied.

Example 5.1

Suppose we have the definition of example 4.1 and the cycle starts at time 1 with the initial goal

maintain-security .

Given an observation of an intruder at time 1, the recording of the observation will be completed at time 2 in step (ii) by conjoining it to the goal. Provided the bound *n* on the resources *R* is sufficiently large, goal reduction and propagation will be performed in step (iv), resulting in the addition to the goal statement of the subgoal

$T'[do(self, raise-alarm, T') Ù 1< T' < 1+10] .

Provided *1+R+2*, i.e. *R+3*, is not later than 11, the agent will attempt to raise an alarm at time *R+3*. If the attempt is successful, the record

do(self, raise-alarm, R+3)

is conjoined to the goal. Otherwise, the record

do(self, raise-alarm, R+3)® false

is conjoined instead.

The proof procedure has additional inference rules, such as *factoring*, to deal with variables and quantifiers in the non-propositional case. In this example, in the case where the attempted action is successful, the *factoring* rule will generate a disjunction corresponding to the two cases, *T' = R+3* or *T' ¹ R+3*. The second case would be useful in a more complicated example where a successful action may need to be retried later if other actions fail.

In the case where the attempted action fails, propagation with the implication which records the action's failure will effectively add the "constraint" *T' ¹ R+3* to the goal statement, which then contains

$T'[do(self, raise-alarm, T') Ù 1<T'<1+10 Ù T' ¹ R+3].

This allows the action to be retried in a future cycle, if time allows.

In more complicated examples where the goal statement is a disjunction and the selected atomic action is one of several candidates for selection, the result of trying the selected action is, in effect, broadcast to every disjunct. Whether the action succeeds or fails, the inference rules correctly process the record on every disjunct. In particular, if any disjunct contains actions which are inconsistent with the result of the action, then the proof procedure will add *false* to the disjunct. The logical equivalences, in turn, will eliminate the disjunct from the goal statement completely.

6 The unified cycle subsumes the other cycles

In order to compare the rational and unified cycles we assume that both cycles use the same proof procedure, outlined in section 4. It is then trivial to see that for any initial goal statement, knowledge base and input, if one iteration of the rational cycle terminates after *r* units of time, then there is a bound *n, n=r,* under which the unified cycle also terminates with the same resulting goal statement.

More interestingly, still assuming that the two cycles use the same proof procedure, but assuming that each iteration of the unified cycle is bounded by a small amount of time *n*, we can show that for any initial goal statement, knowledge base and appropriate inputs, if one iteration of the rational cycle terminates after *r* units of time with a plan all of whose actions eventually succeed, then the unified cycle also terminates after *r/(n+3)* iterations with a goal statement which contains a corresponding plan among the conjunctions of one of its disjuncts. For this purpose we also need to assume that the rational cycle has available among its inputs accurate predictions of the future corresponding to the incrementally obtained inputs of the unified cycle.

It is also possible to show that the unified agent cycle subsumes the reactive agent cycle. Because the two cycles employ the same restricted resources *R* in steps (iii) and (iv), they differ only with respect to their knowledge representations. It is easy to see that in the special case where there are no definitions and, therefore, all integrity constraints, observations and other facts are contained as conjuncts of every disjunct of every goal statement, propagation in steps (iii) and (iv) of the unified cycle simulates steps (iii) and (iv) of the generalised reactive agent cycle. It follows that, provided the actions in the condition-action rules are additive, in the sense that they do not delete any definitions, observations or other facts, the unified cycle subsumes the generalised reactive cycle.

The unified cycle can also simulate cases where the condition-action rules are not additive by employing a declarative representation of actions and their effects, as in the situation calculus or event calculus. In such calculi, the deletion of a fact is represented as the termination of the persistence of that fact as the result of recording a terminating action.

7 Complex Actions

The feasibility of our proposed unified agent architecture depends in large measure on the form of the agent's goal-reduction rules and integrity constraints. The goal-reduction rules, in particular, need to construct partial plans incrementally, so that execution can begin before all the atomic actions in a plan have been generated.

For this purpose, the goal-reduction rules have to provide greater detail about the beginning of a plan than they do about its end. This is illustrated by the following example.

Example 7.1

The following definition reduces the goal of moving from one place, *A*, to another place, *B*, to the subgoals of first taking a single step to a next location, *C*, and then moving from *C* to *B*:

do(Agent, move(A,B), [T1,T2]) « [A=B Ù T1=T2]
Ú [next(A,C) Ù clear(C, T) Ù T1£T£T2 Ù
do(Agent, step(A,C), T) Ù do(Agent, move(C,B), [T,T2])]

Here *T1* and *T2* are the earliest start time and the latest finish time, respectively, for the action *move(A,B)*.

Execution of actions generated by the definition can start after the first three conditions of the second disjunct have been eliminated by goal reduction. The time *T* for the execution of *step(A,C)* must be early enough for the remainder of the partial plan, *move(C,B)*, to be completed before time *T2*.

We assume that such scheduling of actions is performed by a separate control layer which decides how to implement both the non-determinism of the proof procedure and of the selection step (v) of the agent cycle. The control layer needs to decide, therefore, not only when to attempt the atomic action *step(A,C)*, but also which location, *C*, from among the alternative next locations, to select. The control layer can use the same evaluation function both to guide the search strategy of the proof procedure and to select among alternative disjuncts of the current goal statement when commitments to actions need to be made.

The definition of complex actions in terms of other actions, illustrated in this example, is similar to the definition of complex actions in Golog [Levesque *et al*., 1997]. Davila [1996] also uses similar definitions as the target language into which he compiles programs written in a procedural language. The procedural language allows complex actions to be defined using such operations as sequential and parallel composition, conditionals and iteration.

8 Conclusions

In this paper we have outlined an attempt to reconcile the traditional notion of an intelligent agent as one which plans its actions rationally in advance with the contrary notion of an intelligent agent as one which reacts to changes in its environment.

This work extends earlier work [Kowalski, 1995; Kowalski and Sadri, 1996] with the same objective. In this paper, in section 6, we have argued that the unified agent cycle subsumes the rational and the reactive cycles. However, we have not explored the extent to which the unified cycle improves the other cycles. This, together with the development of applications to multi-agent systems, remains for future work.

References

[Brooks, 1991] Rodney Brooks. Intelligence without reason. In *Proceedings of IJCAI 91*, Australia, Morgan Kaufmann, 1991, pp. 569-595.

[Console, *et al.*, 1991] Luca Console, D. Theseider Dupre, Pietro Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation* 2(5), 1991, pp 661-690.

[Davila, 1996] Jacinto Davila. REACTIVE PASCAL and the event calculus: a platform to program reactive, rational agents. In *Proc. Workshop on Reasoning about Actions and Planning in Complex Environments*, Bonn, 1996.

[Denecker and De Schreye, 1992] Marc Denecker and Danny De Schreye. SLDNFA: an Abductive Proof Procedure for Normal Abductive Programs. In *Proc. International Joint Conference and Symposium on Logic Programming*, 1992, pp 686-700.

[Denecker and De Schreye, 1993] Marc Denecker and Danny De Schreye. Representing Incomplete Knowledge in Abductive Logic Programming. In *Proc. International Symposium on Logic Programming*, 1993, pp 147-163.

[Fisher *et al.*, 1996] Fischer, K., Mueller, J.P. and Pische, M. AGenDA: A General Testbed for DAI Applications. *Foundations of DAI*, Jennnings, N. and O'Hare G.M.P. (eds.) John Wiley & Sons, Inc. 1996.

[Fung, 1994] Tse Ho Fung. A modified abductive framework. In *Proceedings of Logic Programming Workshop*, WLP'94, N. Fuchs and G. Gottlob (eds.), 1994.

[Fung and Kowalski, 1996] Tse Ho Fung and Robert Kowalski. The IFF Proof Procedure for Abductive Logic Programming. Department of Computing, Imperial College, 1996.

[Jung *et al.*, 1996] Jung, C., Fischer, K. and Alister Burt. Nonlinear Planning Using an Abductive Event Calculus. DFK Report, 1996.

[Kowalski, 1995] Robert Kowalski. Using meta-logic to reconcile reactive with rational agents. In *Meta-Logic and Logic Programming.* K. Apt and F. Turini (eds.), MIT Press, 1995, pp. 227-242

[Kowalski and Sadri, 1996] Robert Kowalski and Fariba Sadri. Towards a Unified Agent Architecture that Combines Rationality with Reactivity. *Proceedings of International Workshop on Logic in Databases*, San Miniato, Italy, Springer-Verlag, 1996.

[Kowalski and Sergot, 1986] Robert Kowalski and Marek Sergot. A Logic-based Calculus of Events. In *New Generation Computing*, Vol. 4, No.1, February 1986, pp. 67-95.

[Levesque *et al.*, 1997] Hector J. Levesque, Raymond Reiter, Yves Lesperance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A Logic Programming Language for Dynamic Domains. To appear in the *Journal of Logic Programmin*g, 1997.

[Lloyd and Shepherdson, 1991] John W. Lloyd and John C. Shepherdson. Partial evaluation in logic programming. *Journal of Logic Programming*, vol. 11, 1991, pp. 217-242.

[Rao and Georgeff, 1992] Anand S. Rao and Michael P. Georgeff. An abstract architecture for rational agents. *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning, KRR92*, Boston, 1992.

[Shoham, 1993] Yoav Shoham. Agent-oriented programming. *AI Journal*, vol. 60, no. 1, March 1993, pp. 51-92.

[Wetzel *et al.*, 1995] Gerhard Wetzel , Robert A. Kowalski and Francesca Toni. A Theorem-Proving Approach to CLP. In *Workshop Logische Programmierung*, Krall A., Geske U. (eds.), vol. 270 of GMD-Studien, September 1995, pp. 63-72.