Managing Deliberation and Reasoning in Real-Time AI Systems

Félix Ingrand, Michael P Georgee

Managing Deliberation and Reasoning in Real-Time AI Systems*

François Félix Ingrand
Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
E-mail: felix@ai.sri.com

Michael P. Georgeff
Australian AI Institute
1 Grattan Street
Carlton, Victoria 3053
Australia
E-mail: georgeff@aaii.oz.au

Abstract

This paper describes some recent research[1] on architectures for situated (embedded) systems that need to deliberate and reason in real time. One of the most difficult problems in the design of such architectures is how to manage the reasoning performed by such a system while still meeting the real-time constraints of the problem domain. We present an architecture, based on the Procedural Reasoning System (PRS), that provides mechanisms for the management and control of deliberation and reasoning in real-time domains. In particular, we show how deliberation and reasoning strategies can be represented in the form of metalevel plans, and describe an interpreter that selects and executes these in a way that retains bounded reaction time. In addition, this approach allows us to represent different types of situated systems by varying the metalevel deliberation strategies. Finally, we provide some statistical measures of performance for one such type of situated system applied to a complex real-time application.

1 Introduction

The design of reasoning and planning systems that are situated (embedded) in real-time, dynamic environments has recently been the focus of expanded research efforts in artificial intelligence. A critical issue is to identify the architectural features that would enable such systems to exhibit rational behavior in these domains. In this paper, we describe a uniform architecture that we believe addresses many of the difficult problems in this area.

Computer systems, like human beings, have resource limitations: they have only partial knowledge of their environment and bounded computational (or reasoning) capabilities. When the systems are situated in dynamic environments, these limitations become important, because the environment may change in significant ways while a system attempts to gather more information or to reason about what actions to pursue, given the information it already has. If the system (or agent) does not act in a timely manner, it may not be able to recover from a deteriorating situation or may miss positive opportunities.

One way to cope with stringent time constraints is to determine ahead of time how the system should act in every possible situation [Kaelbling, 1987; Rosenschein and Kaelbling, 1986]. However, in domains requiring complex responses to different patterns of events, it is unlikely that such precompilation of plans of action will be practically possible. In such cases, the system must be able to reason about what courses of action to pursue as it observes the changing environment and performs its various tasks.

In particular, at any given time, the system will have to decide what tasks are important enough to initiate or continue, choose among the various means for accomplishing each task, and determine how to order the chosen tasks for execution. In some cases, these decisions may be relatively simple and straightforward. But in other cases they may involve consideration of the likelihood of success of the task, the utility of performing the task, the resources required, the task's expected execution time, the availability and reliability of information upon which the performance of the task depends, the task's dependence on other tasks that are also to be performed, etc.

Moreover, these deliberative tasks themselves are subject to the same constraints on time and information as any other task the system is performing. Thus, the agent will need to decide when and how to seek more information, when and how to deliberate, and when to simply go ahead and act on the basis of whatever reasoning and deliberation it has already performed using whatever information it has at the time. And these deliberations, in turn, need to be reasoned and deliberated about.

An important question, then, is to determine how one can design a situated system that provides for the execution and management of such deliberative processes, yet meets the real-time demands and information constraints of its environment. In this paper, we describe how one such architecture, the Procedural Reasoning System (PRS), provides the mechanisms for handling this problem.

2 An Architecture for Situated Deliberation

The architecture of a PRS module or agent consists of (1) a database containing the system's current beliefs about the world; (2) a set of current goals; (3) a library of plans, called Knowledge Areas (KAs), which describe particular sequences of actions and tests that may be performed to achieve given goals or to react to certain situations; and (4) an intention structure, consisting of a [partially] ordered set of all those plans chosen for execution. An interpreter or inference mechanism manipulates these

*This paper has appeared in the Proceedings of the 1990 DARPA Workshop on Innovative Approaches to Planning, San Diego, CA

[1]This research is supported by the National Aeronautics and Space Administration, Ames Research Center, under Contract No. NAS2-12521.

components, selecting an appropriate plan (KA) based on system beliefs and goals, placing those selected KAs on the intention structure, and finally executing them.

PRS interacts with its environment both through its database, which acquires new beliefs in response to changes in the environment, and through the actions that it performs as it carries out its intentions. Different instances of PRS, running asynchronously, can be used in an application that requires the cooperation of more than one subsystem.

The PRS interpreter runs the entire system. From a conceptual standpoint, it operates in a relatively simple way. At any particular time, certain goals are established and certain events occur that alter the beliefs held in the system database. These changes in the system's goals and beliefs trigger (invoke) various KAs. One or more of these applicable KAs will then be chosen and placed on the intention structure. Finally, PRS selects a task (intention) from the root of the intention structure and executes *one step* of that task. This will result in either the performance of a primitive action, the establishment of a new subgoal, or the conclusion of some new belief.

At this point the interpreter cycle begins again: the newly established goals and beliefs trigger new KAs, one or more of these are selected and placed on the intention structure, and again an intention is selected from that structure and partially executed.

PRS has several features that make it particularly powerful as a situated reasoning system, including: (1) the semantics of its plan (procedure) representation; (2) its ability to construct and act upon partial (rather than complete) plans; (3) its ability to pursue goal-directed tasks while at the same time being responsive to changing patterns of events in bounded time; (4) its facilities for managing multiple tasks in real-time; (5) its default mechanisms for handling stringent real-time demands of its environment; and (6) its metalevel (or reflective) reasoning capabilities. Some of these features have been discussed in earlier reports and papers [Georgeff and Ingrand, 1989; Georgeff and Ingrand, 1990a; Georgeff and Ingrand, 1990b; Georgeff and Lansky, 1986; Rao and Georgeff, 1990]. In this paper, we consider in more detail the way the system architecture supports deliberative reasoning and provide some statistics on the system's real-time performance capabilities.

3 Making Decisions in Real Time

At each interpreter cycle, the changing beliefs and goals of PRS trigger certain KAs (plans) which, upon execution, either perform certain primitive actions or modify the internal state (the beliefs, goals, and intentions) of the system. At this level of abstraction, PRS acts like a situated automaton [Rosenschein and Kaelbling, 1986].

However, one of the most critical aspects of the PRS architecture is the way in which its beliefs, goals, and intentions evolve and change over time. It is here that a number of strong commitments in the design of PRS have been made and these, we believe, are crucial to its successful performance as a situated, real-time system.

Given that the system needs to be able to deliberate in various ways and at various times, one of the most difficult problems to overcome is how to reduce the amount of potential deliberation that need be undertaken. In particular, how can we avoid deliberation on every action (internal or external) taken by the system, and, recursively, how can one avoid or reduce deliberation on those deliberation processes themselves?

Most existing situated reasoning systems use one or a combination of the following approaches:

1. The system does not allow any form of deliberation—the considerations important to such deliberation are compiled into the triggering parts of the plans or knowledge sources themselves [Firby, 1989]
2. The deliberation is performed at one level only and is done at every cycle irrespective of the constraints on time and information existing at that moment in time [Hayes-Roth, 1989]
3. The deliberation occurs at one level only and is performed by a separate module of the system, unconstrained by the real-time demands of the application and thus not bounded in reaction or response time [Dodhiawala *et al.*, 1989; Fehling and Wilber, 1989; Hayes-Roth, 1989].

PRS takes a quite different approach. We consider that the first task of the system should be *to keep the number of options open to deliberation under control.* To achieve this, the PRS interpreter uses certain default decision-making mechanisms that are stringently bounded in execution time. For example, once a certain means has been chosen for achieving a particular goal, and as long as the system has not already failed to achieve the goal using those means, that means *will not be reconsidered* — despite possible changes in the environment that may indicate the existence of better options. These option-reducing decision mechanisms execute in bounded time and, in most real-world situations, substantially reduce the set of options available for deliberation. Some of the more important of these mechanisms are discussed elsewhere [Georgeff and Ingrand, 1989].

Of course, even after this filtering of options, some options remain open to consideration. Furthermore, the filtering may have removed some options that should really have been considered more carefully. Thus, it is necessary to provide the system with a capability for performing a possibly unbounded amount of deliberation and for reconsidering some of the options that have possibly been discarded by the default decision mechanisms.

In PRS, both of these tasks are achieved by the use of so-called metalevel KAs. Metalevel KAs use exactly the same knowledge representation as application-level KAs; they differ only in that they operate on the system's internal state (i.e., its beliefs, goals, and intentions)[2] rather than the external world.

The metalevel KAs are brought to bear on any particular problem by means of their invocation criteria. These criteria may depend both on conditions obtaining in the external world and, more typically, on conditions relating to the internal state of the system. Such conditions might include, for example, the applicability of multiple KAs in the current situation, the failure to achieve a certain goal, or the awakening of some previously suspended intention.

The body of a metalevel KA can be used to represent any kind of decision-making procedure and can be

[2] It is important to note that these include beliefs goals, and intentions toward various properties of the system state, such as the number of applicable KAs at the current time point, the success or otherwise of a particular KA instance, the ordering of the intention structure, or the status of some specific intention.

of arbitrary complexity. However, because it is executed in the same manner as any other KA, it will be interrupted whenever any external events modify the system's beliefs or goals. The system can thus continue to react in bounded time, irrespective of the complexity of the decision procedure. This is unlike other existing situated reasoning systems, whose bound on reaction time is determined by the complexity of the decision-making procedures incorporated in the system.

Moreover, further metalevel KAs can be invoked to make decisions about the decision-making procedures themselves. Again, the representation of these higher levels of metalevel procedures is as for any other procedure, and the system's reaction time remains bounded. Of course, one has to be careful in the design of such metalevel procedures if one wants the system to *respond* to events — rather than just notice them — in some given time frame.

It is also important to note that the decision-making behavior of PRS is strongly influenced by the choice of the invocation conditions of metalevel KAs. For example, if these conditions are such that the decision-making metalevel KAs are frequently invoked, PRS will act in a *cautious* manner, spending more time making decisions than otherwise [Bratman *et al.*, 1988]. If, on the other hand, these metalevel KAs are rarely invoked, PRS will act in a *bold* manner, rapidly choosing its actions in response to the changing world in which it is embedded. Thus, by varying the metalevel KAs, we can study different types of situated systems and determine which are best suited for which problem domains.

The question remains as to how to invoke the metalevel KAs and how to ensure their execution as appropriate. We look at this problem in the next section.

4 Invoking MetaLevel Procedures

Our aim in designing PRS was to hardwire as little as possible into the interpreter; i.e., to make it as simple as possible. This provides us with the potential to investigate many different types of agents simply by varying the default decision procedures and the metalevel KAs.

The main loop of the system interpreter determines which KAs are applicable and chooses which to place on the intention structure. It can be viewed as the topmost metalevel KA; it is the final arbiter of which KAs reach the intention structure and thus which can be executed.

The major problem is how to allow KAs to be deliberated upon by other [metalevel] KAs and how to place the chosen ones on the intention structure. The basis of our approach is to allow the main interpreter loop to place at most *one* KA on the intention structure and to require that it be placed at the root of that structure.

At first sight, this seems unduly restrictive—one often wants to attend to more than one task, and one often wants to order these tasks for later execution rather than have them executed immediately (which placing KAs at the root of the intention structure entails). The way around this problem lies in the metalevel KAs: *these are the means by which one can place multiple intentions on the intention structure and order them as one pleases.*

The next problem is how to actually invoke metalevel KAs. The difficulty is that, while some of the invocation conditions of metalevel KAs will be known at the beginning of each selection cycle, others (such as the number of KAs applicable at a given moment) can only be determined part way through this cycle. The way in which we solve this problem is to allow the system to continue to reflect on its changing beliefs about its own internal state within a single cycle of the interpreter, breaking out of this self-reflection only when the process of KA activation ceases.

Figure 1 shows a simplified version of the main interpreter loop. Its purpose is to select a KA, place it on the intention structure, and invoke its execution (of which we have more to say later). The basic idea of the algorithm is that the system continuously reflects on itself until no new KAs are applicable. When this state is reached, a KA is chosen at random from those applicable at the previous reflection cycle. If there are no KAs to choose from (i.e., the set of applicable KAs is empty), the execution phase is invoked and the outer cycle repeated. Otherwise, the chosen KA is placed on the intention structure, the execution phase invoked, and the outer cycle repeated.

To enable this scheme to work, the system has to determine which KAs are applicable on each self-reflection cycle. This information becomes a new system belief. In particular, on each cycle, the system concludes a belief about the set of KAs applicable on that cycle, expressed as `(soak` x`)`, where x is the list of applicable KAs. It is then determined whether or not the acquisition of this new belief (i.e., `(soak` x`)`), and possibly other events, triggers any new [metalevel] KAs. If it does, the system acquires a new belief about the applicability of these metalevel KAs. In fact, it does so simply by updating the belief `(soak` x`)` so that the list x now contains exactly those metalevel KAs that are now applicable. (The previous belief about applicable object-level KAs is removed from the database and so, in a sense, is forgotten. However, if needed, it can be captured in the variable bindings of the invoked metalevel KAs.)

As PRS places no restrictions upon the invocation conditions of metalevel KAs, it is quite possible that more than one metalevel KA will be invoked at this stage. If this happens, we shall now be left with the problem of deciding which of these metalevel KAs to invoke. There are a number of possible solutions to this problem. One would be simply to select one of the metalevel KAs at random, on the assumption that all are equally good at making the decision about which object-level KAs should be invoked. Another alternative would be to preassign priorities to the metalevel KAs and to invoke the one with the highest priority. However, in keeping with our aim of providing maximum flexibility, the solution we chose to adopt is to allow further metalevel KAs to operate on these lower-level metaKAs in the same way that the lower-level metaKAs operated on the object-level KAs.

The process of invoking metalevel KAs is thus continued until no further KAs are triggered. At that point, there may still be a set of applicable KAs from which to choose. It is then, and only then (i.e., only after failing to find any more applicable metalevel KAs) that we select one of these KAs at random.

Thus it is seen that, when more than one KA is applicable, and in the absence of any information about what is best to do, the system interpreter defaults to selecting one of these KAs at random. With no metalevel KAs, the system would thus randomly select one of the applicable object-level KAs. However, one usually provides metalevel KAs to help make an informed choice about the object level KAs. The applicable metalevel KAs themselves are subject to the same default action (i.e., one

```
(loop ;Loop continuously.
      do (loop for soak = (set-of-applicable-ka) ;Set soak to the set of applicable KAs
               when (or previous-soak soak)
                    do (conclude-fact '(soak ,soak)) ;Post the soak metalevel fact
               if (null soak) ;No new KAs are applicable
                  then if (null previous-soak)
                           ;If previous soak is empty then either no KAs were relevant
                           ;or there is nothing to do (no new goals).
                           then ;Continue to execute the intentions in the Intention Structure
                                (activate-intention-structure)
                                return ;Exit the reflective loop.
                           else;Else, intend one of the KAs selected randomly,
                                (intend (select-randomly previous-soak))
                                ;Go and execute something in the intention structure,
                                (activate-intention-structure)
                                ;Set previous-soak to nil
                                (setq previous-soak nil)
                                return) ;Exit the reflective loop
                  else (setq previous-soak soak)) ;Swap previous-soak and soak
               do (get-new-facts)) ;Get any new facts generated by metalevel matching
      (get-new-facts-goals-messages)) ;Get any new messages, goals or facts
```

Figure 1: KA and Intention Selection in PRS

will be randomly selected) unless there are yet other metalevel KAs available to make a choice among them. In the end, at some level in the meta-hierarchy, the default action will be taken.

Once selected, the chosen KAs must be inserted into the intention structure. If a selected KA arose due to an external goal or a new belief, it will be inserted into the intention structure as a new intention at the root of the structure. For example, this will be the case for any metalevel KA that is invoked to decide among some set of applicable lower-level KAs. Otherwise, the KA instance must have arisen as a result of some subgoal of some existing intention, and will be "grown" (i.e., attached) as a subKA of that intention. Finally, we are left with the execution phase. This is relatively straightforward. First, an intention at one of the (possibly multiple) roots of the intention structure is selected for further execution. The next step of that intention will comprise either a primitive action or one or more unelaborated subgoals. If the former, the action is directly initiated; if the latter, these subgoals are posted as new goals of the system.[3]

While we have focused above on metalevel KAs that react to changes in the type or number of applicable KAs, other beliefs about the environment or the internal system state can trigger other kinds of metalevel KAs. For example, beliefs about changing intentions could trigger metalevel KAs to reorder the intention structure, or beliefs about failed goals could trigger a metalevel KA to deliberate on the utility of reattempting the goal.

5 Measures of Performance

Definitions of real-time systems revolve around the notion of *response time*. For example, Marsh and Greenwood [Marsh and Greenwood, 1986] define a real-time system as one that is "predictably fast enough for use by the process being serviced," and O'Reilly and Cromarty [O'Reilly and Cromarty, 1985] require that "there is a strict time limit by which the system must have produced a response, regardless of the algorithm employed." This measure is most important in real-time applications; if events are not handled in a timely fashion, the operation can go out of control.

Response time is the time the system takes to recognize and respond to an external event. Thus, a bound on *reaction time* (that is, the ability of a system to recognize or notice changes in its environment) is a prerequisite for providing a bound on response time. PRS has been designed to operate under a well-defined measure of reactivity. Because the interpreter continuously attempts to match KAs with any newly acquired beliefs or goals, the system is able to notice newly applicable KAs after every primitive action it takes.

Some useful performance metrics for evaluating the performance of real-time situated systems are provided by Dodhiawala [Dodhiawala *et al.*, 1989]. Not all of these are of relevance in the applications to which PRS has so far been applied, but the following five probes provide important measures of performance:

1. *sending-time(e)* is the time at which an event e is signalled
2. *receiving-time(e)* is the time at which e is received by the system
3. *begin-ack-time(e)* is the time at which e is noticed by the system
4. *end-soak-time-cycle(e)* is the time at which all the events occurring in the same cycle as e have been noticed and the corresponding set of applicable KAs determined
5. *event-execution-time(e)* is the time at which the first action following KA selection has terminated
6. *event-response-time(e)* is the time at which the execution of all the procedures initiated by e has terminated.

Then we define:

$R1 =$ *receiving-time(e)* $-$ *sending-time(e)*,

$R2 =$ *begin-ack-time(e)* $-$ *receiving-time(e)*,

[3] In fact, the execution algorithm is somewhat more complicated than we indicate here. For example, it needs to handle in different ways the failure and success of attempting to accomplish its goals, what goals need to be reestablished, etc.

$R3$ = *end-soak-time-cycle(e)* − *begin-ack-time(e)*,

$R4$ = *event-execution-time(e)* − *end-soak-time-cycle(e)*,

$R5$ = *event-response-time(e)* − *sending-time(e)*

Assuming a bounded number of events occurs in any time interval, we can prove that $R1$, $R2$, $R3$, and $R4$ are bounded. $R1$ is the time used to communicate the event to the system and is bounded by definition of the communication function (independently of PRS). The operations performed in $R3$ and $R4$ form a cycle, so $R2$ is actually bounded by $R3 + R4$. So if we prove that $R3$ and $R4$ are bounded, we can conclude that $R2$ is also bounded.

$R4$ is bounded by the maximum time required to execute the longest primitive action in PRS or the time required to post a goal. The time to post a goal is bounded by definition. Therefore, the bound on $R4$ is determined by the choice of primitive actions and thus by the user. As the user can choose any level of granularity he or she desires, this bound can be made arbitrarily small. (In the application described below, a maximum action execution time of one second was found to be quite satisfactory, though other applications may well require finer granularity.)

$R3$ is the time used by the system to parse the invocation part and the context part of relevant KAs. As we have a bounded number of events and a bounded number of KAs, we can guarantee that $R3$ is bounded.[4]

To estimate the bound on $R2$, let p be an upper bound on the execution times of the primitive actions that the system is capable of performing. Also assume that n is an upper bound on the number of events that can occur in unit time, and that the PRS interpreter takes at most time t to select the set of KAs applicable to each event occurrence. The maximum *reactivity delay*, Δ_R, is then given by $\Delta_R = p + y \times t$, where y is the maximum number of events that can occur during the reaction interval. We have $y = \Delta_R \times n$ and thus obtain $\Delta_R = p/(1 - nt)$ where we assume that $t < 1/n$. This means that, provided the number of events that occur in unit time is less than $1/t$, PRS will notice *every* event that occurs [that is capable of triggering some KA] and is guaranteed to do so within a time interval Δ_R.

Because metalevel procedures are treated just like any other, they too are subject to being interrupted after every primitive metalevel action. Thus, reactivity is guaranteed even when the system is choosing between alternative courses of action or performing deliberations of arbitrary complexity.

$R5$ is the time one would like most to see bounded. However, as the time taken to respond to an event can be arbitrarily large, no such guarantee can be given in general. Let's consider this in a little more detail.

Having reacted to some event, it is necessary for the system to respond to this event by performing some appropriate action. As the system can be performing other tasks at the time the critical event is observed, a choice has to be made concerning the possible termination or suspension of those tasks while the critical event is handled. Furthermore, if there are a number of different ways in which the event can be handled, it might be necessary to choose among those alternatives.

Such choices can be made by appropriate metalevel KAs. However, in general, these decision procedures may take an unbounded amount of time to reach a decision. There are two possible ways to overcome this problem. One is to require that all decision procedures complete in a bounded time. In many domains, this provides adequate decision-making capability and yields a bound on response time. As a particular case, it is not difficult to construct metalevel KAs that yield the same functionalities as Ladder Logic.[5]

Alternatively, one could construct a special metalevel KA to act as a task scheduler. This KA would have the capability to preempt all executing decision tasks (and any other tasks for that matter) within a bounded time and to begin execution of an event handler. It could utilize whatever information was available (such as any incremental decisions made by anytime decision algorithms [Dean and Boddy, 1988]) to select the most appropriate event handler and the manner in which to suspend or terminate other tasks. It could also take into account the different constraints on response time that may exist in different situations. The only requirement is that this KA have a guaranteed upper bound on execution time.

In summary, PRS is guaranteed to react to critical events in a bounded time interval. With appropriate metalevel and application-level KAs, it is also possible to guarantee a bound on response time.

6 Experimentation

As mentioned earlier, one of the valuable features of this design is the ability to realize different types of situated systems by varying the default decision rules and the metalevel procedures. In particular, one could then examine the behavioral properties of different types of agents in different environments. We have begun this process by creating one particular type of agent [Georgeff and Ingrand, 1989] and applying it to various real-time applications. In this section, we briefly describe one such application and provide statistics on the performance of the system.

The application domain we choose for experimentation is the task of malfunction handling for the Reaction Control System (RCS) of NASA's space shuttle. This is a relatively complex propulsion system that is used to control the attitude of the shuttle. A wide range of problems can occur in this system, and, in a normal shuttle mission, no less than four mission controllers are continuously monitoring and controlling its operation.

Two PRS modules (agents) were used for the application. The resulting system was able to detect and recover from most of the possible malfunctions of the RCS, including sensor faults, leaking components, and regulator and jet failures. It is currently under testing at NASA's Johnson Space Center.

The following performance measures have been made on a SUN Sparcstation, with 20 megabytes of central memory, running Sun Common Lisp, development Environment 4.0.0 Beta-0, Sun4 Version for SunOS 4.0. The code was not optimized by the compiler, and the probing itself affects system performance (the probes defined in Section 5 are activated for every event and goal posted by PRS).

For the series of tests given below, we ran the following RCS scenarios: a pressure transducer failure, a regulator

[4] As selection of KAs does not involve any general deduction beyond unification and evaluation of a Boolean expression, an upper bound does indeed exist.

[5] Ladder Logic is one of the most widely used program languages for real-time systems.

Name	RCS	INTERFACE
Percent	8.07	36.33
Facts	20	49
Meta-Facts	357	2364
Goals	202	1257
Messages	66	355
Relevant-KAs	923	8359
Applicable-KAs	105	1108
Intentions	6	14
Satisfied Goals in DB	28	164
Total Run Time	00:00:31	00:02:29

Figure 2: Performance Statistics in the RCS application

failure with both regulators open, and a leaking manifold. This set of scenarios exercises most of the major features of PRS and is representative of the kind of problems occurring in the RCS system. The whole test set took approximately six minutes to run.

Figure 2 shows some statistics on the run. The *Percent* measurement indicates how busy the PRS agents were. During the six minutes, RCS ran for 31 seconds, and INTERFACE for 2 minutes 29 seconds. Clearly, each PRS module has plenty of time to work on other problems. (On this machine, with this configuration, this application can be run three times faster than real time without any difficulty). The *Facts*, *Metalevel Facts*, *Goals* and *Messages* indicate the flow of input to the two PRS modules. We have separated metalevel facts, such as `(soak ...)`, and application facts. The statistics on the goals and messages refer only to the application level. *Relevant-KAs* represents the number of relevant KAs (selected by the indexing mechanism as being potentially applicable) and *Applicable-KAs* represents the number of KAs that were actually applicable. *Intentions* indicates the number of intentions the PRS agent has formed, and *Satisfied Goals in DB* represents the number of goals that were directly satisfied in the database (and thus did not require KA activation).

Figure 3 shows the values of $R2$, $R3$, and $R5$ (see Section 5). All the values are given in sixtieths of a second. The average $R2$ are usually very low (a few sixtieths of a second), and even the maximum values stay under one second.[6] $R3$ is also quite small and never exceeds one second. The values of $R5$, which represents response time, are very difficult to interpret. This is because many of the procedures executed in the RCS application are supposed to "wait" for certain external events to occur. For example, certain procedures require the system to wait for the pressure to drop under 300 psi, or to wait for the astronauts to flip a switch.[7] Nevertheless, the experience of and the evaluation of the system by mission controllers show that PRS executes its procedures much faster than either an astronaut or a mission controller could. Moreover, in this application, metalevel KAs have been written to ensure that the most important procedures get executed first, thus guaranteeing that the response time is shortest for the most urgent procedure [Georgeff and Ingrand, 1990b].

[6]The high maximum value can be explained by the quantum (300 ms) of the scheduler used under SUN lisp 4.0. That means that if both PRS modules are runnable, one will have to wait at least 300 ms before getting a chance to run.

[7]These waits are asynchronous and do not block system execution.

RCS				
	Number of facts	Average	Distribution	Maximum
$R2$	486	3.04	6.34	42.
$R3$	486	2.55	4.38	36.
$R5$	7	2222.37	2630.15	6863.
	Number of goals	Average	Distribution	Maximum
$R2$	202	0.57	2.03	21.
$R3$	202	2.16	3.77	26.
$R5$	186	341.21	783.54	5983.
INTERFACE				
	Number of facts	Average	Distribution	Maximum
$R2$	2982	4.23	13.14	94.
$R3$	2982	5.84	10.67	52.
$R5$	124	441.20	479.65	2355.
	Number of goals	Average	Distribution	Maximum
$R2$	1257	0.42	1.05	21.
$R3$	1257	1.88	2.23	41.
$R5$	1165	33.11	115.06	1494.

$$\text{Average} = \overline{x},$$
$$\text{Distribution} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}} \text{ and}$$
$$\text{Maximum} = \max(x_1, \ldots, x_n)$$

Figure 3: $R2$, $R3$, and $R5$ for the RCS application

7 Review of related works

Some researchers have sought to deal with resource limitations in dynamic environments by considering all contingencies at design time. This approach obviates the need for explicit reasoning at execution time: all such reasoning is effectively *compiled* into the structure of the executing program [Agre and Arge, 1987; Brooks, 1986; Firby, 1989; Rosenschein and Kaelbling, 1986; Kaelbling, 1987]. It is very likely that these techniques are optimal in certain applications. However, many researchers believe that, in complex domains, the knowledge-compilation approach will lead to brittle, inflexible systems if used without any real-time deliberative processing [D'Ambrosio and Fehling, 1989; Doyle, 1988; Pollock, 1989].

Blackboard architectures have been used in certain systems that are intended to perform real-time behavior[Dodhiawala *et al.*, 1989; Hayes-Roth *et al.*, 1989]. They use a collection of knowledge sources (tasks) sharing a common data structure. There are a number of interesting features of these systems that could be important in providing fast response in real-time domains that do not require significant amounts of deliberation. However, in current blackboard systems, the actions carried out by the system are not interruptible. This poses serious problems for maintaining realistic bounds on reaction time whenever complex or lengthy tasks need to be performed [Georgeff and Lansky, 1986]. Keeping the blackboard consistent when knowledge sources are asynchronous is also a serious problem that has yet to be addressed. In addition, most blackboard architectures use an agenda of pending tasks that are run serially. The problem is that the agenda manager (i.e., the component that deliberates on what tasks to execute, how to execute them, and when to execute them) is invoked in each cycle and for each task present on the agenda. Thus it runs with considerable overhead, again seriously restricting the real-time capabilities of the system. Moreover, it is difficult to include any lengthy deliberation procedures

in blackboard architectures, and there are no mechanisms for reasoning about the deliberation processes themselves.

Schemer-II [Fehling and Wilber, 1989] is in some way similar to PRS, but utilizes specific managers and handlers (deliberation processes) to control the system. As with the blackboard approach, these task handlers cannot reason about themselves. Consequently, the architecture is not as general or flexible as PRS. However, it is an interesting approach and may be optimal for some real-time domains.

8 Conclusion and Future Developments

In this paper, we have attempted to show how the uniform knowledge representation for both application-level knowledge and metalevel knowledge, the default decision rules, and the algorithm used for handling metalevel procedures provide a good framework for managing deliberation and reasoning in real-time environments. We have presented some results regarding the real-time performance of the system when used in a real application (RCS), and briefly reviewed some related works.

Although we have presented an architecture that supports real-time deliberation and reasoning, we have so far not investigated how different default decisions and different metalevel strategies affect system behavior; nor have we examined sufficient real-time domains to determine which kind of situated system best suits which kind of domain. The current RCS application used a set of default decision rules and metalevel procedures that proved to be particularly successful *in that domain*. While we believe these to be of wide applicability, that conjecture has yet to be tested.

Of particular interest would be to incorporate as metalevel procedures various algorithms that have recently been proposed for deliberating in real-time environments. These include the work of Whitehair and Lesser on approximate reasoning [Lesser *et al.*, 1989], Dean and Boddy's work in anytime algorithms [Dean and Boddy, 1988], and the work of a number of researchers [Agogino and Ramamurthi, 1989; Horvitz *et al.*, 1988; Russell and Wefald, 1989] in decision-analysis techniques.

We intend to explore some of these issues in our future research. In particular, by varying metalevel strategies, we aim to experiment with different types of system (such as the IRMA agent architecture [Bratman *et al.*, 1988]) in different kinds of environments, thus leading to a better understanding of situated systems and agent rationality.

References

[Agogino and Ramamurthi, 1989] A. M. Agogino and K. Ramamurthi. Real time reasoning about time constraints and model precision in complex distributed mechanical systems. In *Proceedings of the AAAI Symposium on Limited Rationality*, pages 96–100, Stanford, California, U.S.A, 1989.

[Agre and Arge, 1987] P. E. Agre and D. Arge. Pengi: An implementation of a theory of activity. In *Proceedings of the National Conference on Artificial Intelligence*, pages 268–272, Seattle, Washington, 1987.

[Bratman *et al.*, 1988] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computation Intelligence*, 4(4), 1988.

[Brooks, 1986] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, pages 14–23, 1986.

[D'Ambrosio and Fehling, 1989] B. D'Ambrosio and M. Fehling. Resource bounded-agents in an uncertain world. In *Proceedings of the AAAI Symposium on Limited Rationality*, pages 13–17, Stanford, California, U.S.A, 1989.

[Dean and Boddy, 1988] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 49–54, Saint Paul, Minnesota, 1988.

[Dodhiawala *et al.*, 1989] R. Dodhiawala, N. S. Sridharan, P. Raulefs, and C. Pickering. Real-time AI systems: A definition and an architecture. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 256–261, Detroit, Michigan, U.S.A, August 1989.

[Doyle, 1988] J. Doyle. Artificial intelligence and rational self-government. Technical Report CS-88-124, Carnegie Mellon University, Pittsburgh, Pa., 1988.

[Fehling and Wilber, 1989] M. R. Fehling and B. M. Wilber. Schemer-II: An architecture for reflective, resource-bounded problem solving. Technical Report 837-89-30, Rockwell International Science Center, Palo Alto Laboratory, Palo Alto, California, U.S.A, 1989.

[Firby, 1989] R. James Firby. *Adaptative Execution in Complex Dynamic Worlds*. PhD thesis, Yale University, Department of Computer Science, Yale University, May 1989.

[Georgeff and Ingrand, 1989] M. P. Georgeff and F. F. Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 972–978, Detroit, Michigan, U.S.A, 1989.

[Georgeff and Ingrand, 1990a] M. P. Georgeff and F. F. Ingrand. Real-time reasoning: The monitoring and control of spacecraft systems. In *Proceedings of the Sixth IEEE Conference on Artificial Intelligence Applications*, Santa Barbara, California, U.S.A, March 1990.

[Georgeff and Ingrand, 1990b] M. P. Georgeff and F. F. Ingrand. Research on procedural reasoning systems. Final Report, Phase 2, for NASA Ames Research Center, Moffet Field, California, U.S.A, Artificial Intelligence Center, SRI International, Menlo Park, California, U.S.A, June 1990.

[Georgeff and Lansky, 1986] M. P. Georgeff and A. L. Lansky. Procedural knowledge. *Proceedings of the IEEE Special Issue on Knowledge Representation*, 74:1383–1398, 1986.

[Hayes-Roth *et al.*, 1989] B. Hayes-Roth, R. Washington, R. Hewett, M. Hewett, and A. Seiver. Intelligent monitoring and control. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 243–249, Detroit, Michigan, U.S.A, August 1989.

[Hayes-Roth, 1989] B. Hayes-Roth. Architectural foundations for real-time performance in intelligent agents. Technical Report KSL 89-63, Knowledge Systems Laboratory, Department of Computer Science, Stanford University, Stanford, California, U.S.A 94305, December 1989.

[Horvitz *et al.*, 1988] E. J. Horvitz, J. S. Breese, and M. Henrion. Decision theory in expert systems and artificial intelligence. *Journal of Approximate Reasoning*, 2:247–302, 1988.

[Kaelbling, 1987] L. P. Kaelbling. An architecture for intelligent reactive systems. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 395–410. Morgan Kaufmann, Los Altos, California, U.S.A, 1987.

[Lesser *et al.*, 1989] V. R. Lesser, D. D. Corkill, R. C. Whitehair, and J. A. Hernandez. Focus of control through goal relationships. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 497–503, Detroit, Michigan, U.S.A, August 1989. International Joint Conference on Artificial Intelligence.

[Marsh and Greenwood, 1986] J. Marsh and J. Greenwood. Real-time AI: Software architecture issues. In *Proceedings of the IEEE National Aerospace and Electronics Conference*, pages 67–77, Washington, DC, 1986.

[O'Reilly and Cromarty, 1985] C. A. O'Reilly and A. S. Cromarty. "Fast" is not "real-time" in designing effective real-time AI systems. In *Applications of Artificial Intelligence II*, pages 249–257, Bellingham, Washington, 1985. Int. Soc. of Optical Engineering.

[Pollock, 1989] J. L. Pollock. Oscar: A general theory of rationality. In *Proceedings of the AAAI Symposium on Limited Rationality*, pages 96–100, Stanford, California, U.S.A, 1989.

[Rao and Georgeff, 1990] A. S. Rao and M. P. Georgeff. Intention and rational commitment. Technical Report 8, Australian AI Institute, Carlton, Australia, 1990.

[Rosenschein and Kaelbling, 1986] S. J. Rosenschein and L. P. Kaelbling. The synthesis of digital machines with provable epistemic properties. In *Proceedings of the Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 83–98, 1986.

[Russell and Wefald, 1989] S. Russell and E. Wefald. Principles of metareasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Ontario, 1989.