Procedural Reasoning versus Blackboard Architecture for Real-Time Reasoning

Félix Ingrand, Vianney Coutance

Procedural Reasoning versus Blackboard Architecture for Real-Time Reasoning *

François Félix Ingrand
LAAS/CNRS
7, Avenue du Colonel Roche
31077 Toulouse Cedex, France
Tel: (33) 61 33 63 46
E-mail: felix@laas.fr

Vianney Coutance
ACS Technologies
5, Place du Village d'Entreprises,
31674 Labege Cedex, France
Tel: (33) 62 24 99 20
E-mail: vianney@acs.dialexis.fr

Abstract

In the last years we have witnessed an increasing interest in AI systems able to perform reasoning and actions in real-time environments. However, the issue of combining advanced reasoning under real-time constraints remains a challenge. Nevertheless, some architectures or systems are now emerging as potential tools to tackle this problem. Among them, one can find the Blackboard Architecture, and the Procedural Reasoning System.

In this paper, we shall give an account of what a real-time reasoning system should be. Then, we will focus on the Blackboard Architecture and the Procedural Reasoning System approaches and study exactly what makes them particularly well suited for real-time reasoning. We shall also compare them at the light of various real-time applications in which they have been used. At last, we will argue that, by design, PRS seems to offer better characteristics to allow real-time reasoning, whereas the characteristics which make the Blackboard architecture well suited for real-time reasoning are usually added onto the original architecture.

We conclude with a short survey of other systems and architectures used for real-time reasoning.

Keywords: Architectures and languages for AI, real-time AI, Procedural Reasoning, Blackboard Architecture.

Résumé

Durant ces dernières années, on a pu constater un intérêt croissant pour les systèmes d'IA capables de raisonner et d'agir dans des environements temps réel. Toutefois, les problèmes résultant de l'utilisation de techniques avancées de raisonnement en temps réel subsistent. Malgré cela, quelques systèmes et architectures apparaissent qui s'intéressent de près à ces problèmes. Parmi eux, on trouve l'architecture Tableau Noir et les Systèmes de Raisonnement Procédural.

Cet article introduira d'abord une caractérisation de ce que peut ou doit être un système de raisonnement temps réel. Nous présenterons alors l'architecture Tableau Noir (BB) et les Systèmes de Raisonnement Procédural (PRS), en étudiant de plus près ce qui les rend particulièrement aptes au raisonnement en temps réel. Ensuite, nous les comparerons, au vue des applications temps réel pour lesquelles ils ont été utilisés. Enfin, nous argumenterons que de par sa conception, PRS semble offrir de meilleurs caractéristiques pour le raisonnement en temps réel, alors que celles qui rendent l'architecture Tableau Noir adaptée aux applications temps réel ont en pour la plupart été rajoutées à l'architecture originale.

Nous conluerons par un rapide survol des autres systèmes et architectures utilisés pour le raisonnement temps réel.

Mots clés: Architectures et langages pour l'IA, IA temps réel, Raisonnement Procédural, Tableau Noir.

*This paper has been published in the Proceedings of the Thirteenth International Conference on Artificial Intelligence, Avignon, 1993.

1 Introduction

In the past years, we have witnessed an ever increasing interest in AI systems able to perform their tasks in real-time (RT) environments. There are various reasons to this growing interest, but the main one is probably the introduction of AI systems in areas requiring RT properties. Among them, one can find applications for monitoring and control of industrial processes, automatic medical supervision, pilot assistance, and so on. Another reason for the increasing interest in RT AI systems is probably the ever increasing power of available computers which allow traditionally slow AI reasoning systems to perform at an acceptable speed. However, as pointed out by the authors in [O'Reilly and Cromarty, 1985], "Fast" is not "Real-Time"...and this newly available computational power only solves a small part of the problem.

Indeed, the design of Real-Time Reasoning Systems (RTRS) gœs far beyond the issue of using fast machines. In section 2, we will define what a RTRS is or, more accurately, which temporal properties a RTRS may have to prove or guarantee, to consider its use suitable for a particular application.

Section 3 will present PRS, a general architecture for representing and reasoning about procedures in dynamic domains. We show elsewhere [Ingrand and Coutance, 1993] that, by design, and under some well defined assumptions, the PRS main loop satisfies one important temporal property: an upper bound on the reaction time. In this paper, we will also present some aspects of its architecture, seldom presented, but which greatly participate in PRS ability to handle RT constraints.

In Section 4, we will present the Blackboard (BB) Architecture and some BB implementations used in RT environments. We will see that most of the characteristics which make these various implementations well suited for RT environments have been added on to the original architecture.

Section 5 will present a comparison of the two approaches; PRS and BB, with an emphasis on the BB system used in REAKT [Lalanda *et al.*, 1992b].

In Section 6, we will present other systems and architectures which address the issue of RTRS. This short survey ranges from "Compilation Techniques" to "High Level Architecture".

2 What is a Real Time Reasoning System?

The definition of a real-time reasoning system (RTRS) has been the subject of many publications, and is still a controversy in the AI community. From our point of view, there is no general definition of a RTRS, there are only reasoning systems which satisfy some temporal properties.

To some people, it is not reasonable to "view real-time performance as a provable, guaranteed, universal property of the agent" [Hayes-Roth, 1989]. Other think that an upper bound on the response time is always required [Kaelbling, 1988], even if the price to pay is an almost complete loss of flexibility at run-time.

We believe that a so-called RTRS must guarantee some temporal properties. In fact, the very essence of the real-time system domain is that one can prove some temporal properties before hand. People will only find interest in RTRS, and will trust them if, and only if, for a given application, one can prove given properties about them. Which patient would accept to be under the control of an automatic respirator system which cannot guarantee a bound on reaction time? Which pilot will accept to fly a plane if his assistant may fail whenever he needs it most. One cannot trust a system if we can only guarantee that it works in most or average situations, but may loose events or exceed a given reaction time from time to time.[1]

The problem remains to decide which temporal properties we need, what do we want to prove and what is the trade off we are willing to make to guarantee these required properties? For example, in some applications, one needs to prove that there exists an upper bound on the response time, and indeed, there is plenty of "traditional computer science" literature on systems which can guarantee such a thing [Benveniste and Berry, 1991]. However few reasoning systems are able to guarantee this property. The cause is not lost, we can in some situations use a more flexible mechanism while still guaranteeing some basic temporal properties.

Coming back to the RTRS definition, most people commonly expect a real-time system to give an appropriate response in a bounded, known in advance, amount of time. Although this is desirable, it is not possible for most AI reasoning systems. Among the various underlying reasons, the most obvious one is that, contrary to standard programming systems, the control is not defined as imperative and therefore, it is very tedious, if not impossible, to predict exactly which execution path will be taken.

2.1 Time intervals

We shall now define a number of time intervals which will be of interest to define temporal properties. As shown on Figure 1, we can define the following time intervals for a given event:

Transmission time is the elapsed time between the date of creation of the event and the date at which the event arrives in the system input buffers, which is asynchronously accessible by the external modules. In other words, we recognize the existence of a buffer, where external modules can put events for the RTRS when desired. From time to time, the RTRS picks up all the events in this buffer and parses them.

[1] Note that this type of situation tends to appear exactly when you are in a critical phase, where many events occur at the same time and the risks of exceeding the capabilities of the RTRS are high.

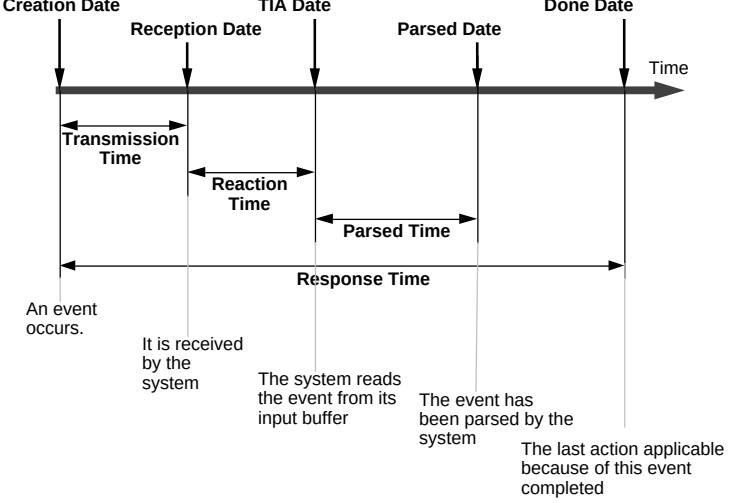

Figure 1: Important dates in the life of an event

Reaction time is the elapsed time between the date of arrival of the event in the input buffer and the date at which the event is taken into account by the system (or picked up from the input buffer). In other words, for each event, it is the time spent waiting in the input buffer.

Parsed time is the elapsed time between the date at which the event is taken into account by the system and the date at which the event has been parsed by the triggering mechanism of the RTRS.

Response time is somewhat more complex to define. It basically depends on what we consider to be a response to an event, i.e., what type of solution we want to use in response to an event. The solution to arbitrary problems can be arbitrary complex...By using internal states, most systems "build up" the response over more than one cycle (except for "reflex" or 0(1) computation). Thus, it is certainly not reasonable to define the response time as the time elapsed between the date of creation and the date at which the *first action to be performed, because of this event, has completed. We define it at as the elapsed time between the date of creation of the event and the date at which the* last action to be performed, because of this event, has completed.

Note that for a given event, reaction time is always inferior or equal to the response time, which means that an upper bound must be found on the former if the latter is to be bounded.

These intervals being defined, here is a non exhaustive list of some temporal properties which can participate to the design and the validation of a RTRS:

- upper bound on reaction time, which allows the system to guarantee that it will detect every event in a given amount of time,
- upper bound on transmission time, which insures that the event will arrive after a given amount of time in the input buffer,
- upper bound on first response time, which insures a bound on the first response given to an event,
- upper bound on last response time, which guarantees that all responses to any event will be delivered in a predefined amount of time.

We believe that the definition of a RTRS (or at least its validation) should be based on some of these properties, or derived ones.

3 Description of PRS

3.1 General Description

The Procedural Reasoning System (PRS) can be described as a generic architecture for representing and reasoning about actions and procedures in dynamic domains. It has already been applied to various tasks with real-time demands, that range from malfunction monitoring for different subsystems of NASA's space shuttle [Ingrand *et al.*, 1992], to the diagnosis, monitoring and control of telecommunications networks [Wesley *et al.*, 1991], the control of a mobile robot [Revillod, 1992], and system control for a surveillance aircraft [Ingrand *et al.*, 1989].

The architecture of a PRS kernel is composed of three main elements:

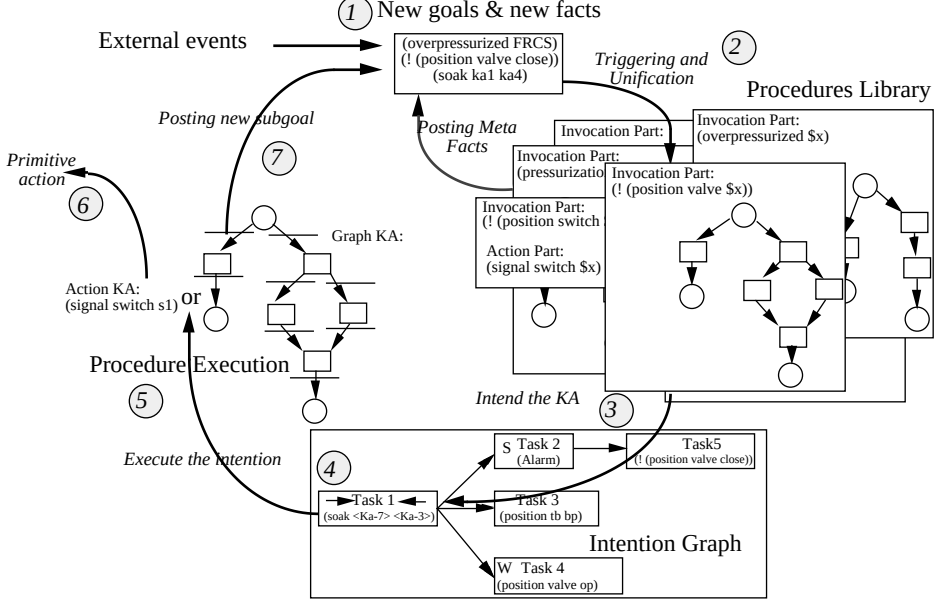

Figure 2: PRS Interpreter

1. a database contains the system current beliefs about the world and is constantly and automatically updated as new events appear,

2. a library of plans (or procedures), called Knowledge Areas (KAs), describes particular sequences of actions and tests that may be performed to achieve given goals or to react to certain situations,

3. an intention graph consists of a [partially] ordered set of all plans or tasks to be executed at runtime.

3.2 The Interpreter

As shown in Figure 2, an interpreter (inference mechanism) manipulates these components. It receives new events and internal goals (1), selects an appropriate plan (KA) based on these new events, goals, and system beliefs (2), places the selected KA on the intention graph (3), chooses a current intention among the roots of the graph (4) and finally executes *one step* of the active KA in the selected intention (5). This can result in a primitive action (6), or the establishment of a new goal (7).

The PRS kernel 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.

In PRS, goals are descriptions of desired tasks or behaviors. In the logic used by PRS, the goal to achieve a certain condition `C` is written as `(! C)`; to test for the condition is written as `(? C)`; to wait until the condition is true is written as `(^ C)`; to maintain C is written as `(# C)`; to assert the condition C is written as `(=> C)`; and to retract the condition C is written as `(~> C)`. For example, the goal to close a valve `v1` could be represented as `(! (position v1 cl))`, and to test for it being closed as `(? (position v1 cl))`.

Knowledge about how to accomplish given goals or to react to certain situations is represented in PRS by declarative procedure specifications called *Knowledge Areas* (KAs). Each KA consists of a *body*, which describes the steps of the procedure, and an *invocation condition*, which specifies under which situations the KA is useful. Together, the invocation condition and the body of a KA express a declarative fact about the results and utility of performing certain sequences of actions under certain conditions [Georgeff and Lansky, 1986].

The set of KAs in a PRS application system not only consists of procedural knowledge about a specific domain, but also includes *metalevel* KAs (See Figure 3) — that is, KAs able to manipulate other KAs, beliefs, goals, and intentions of PRS itself. The use of metalevel KAs range from methods for choosing among multiple applicable KAs, to insure mutual exclusion on critical resource, or to compute the amount of additional reasoning that can be undertaken, given the real-time constraints of the problem domain. To achieve such goals, these metalevel KAs make use of information about KAs, goals, facts that is contained in the system database or in the property slots of the KA. For example, the Meta KA presented on Figure 3 will insure that any procedure invoked because of an external event, such as an external fact, will be intended.[2]

[2]This type of KA is usually required in monitoring and control applications.

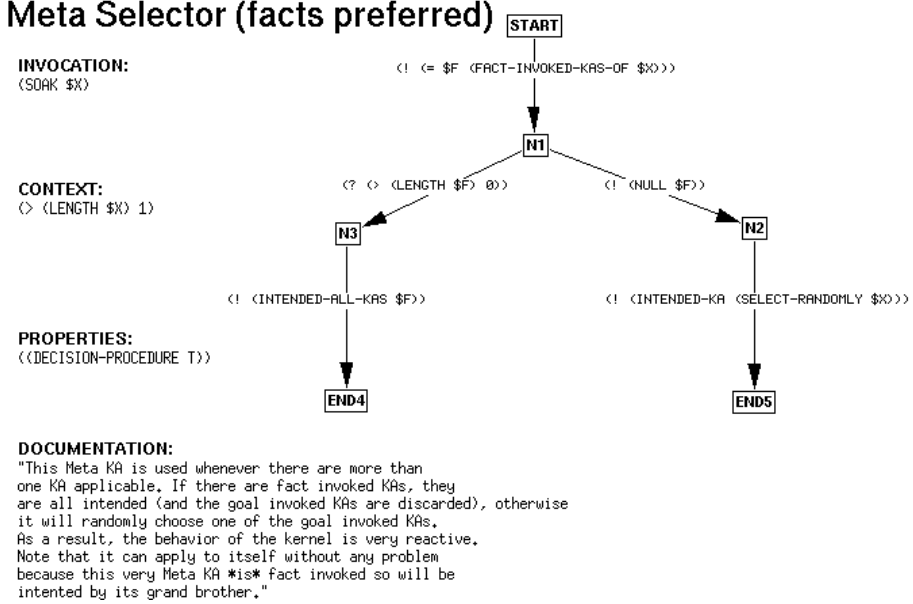

Figure 3: A Simple Meta Level KA

3.3 PRS Main Loop

If one must understand the way meta level KAs get executed, the PRS main loop is of great interest.

```
while (TRUE) {                                  /* Loop for ever. */
        check_input_buffer();                   /* Check the input buffer. */
        shift_facts_goals();                    /* Get new facts and new goals. */
        soak = find_soak();                     /* Look for new applicable KAs. */
        while (! (list_empty(soak))) {          /* While we have Applicable KAs. */
                post_soak_meta_facts(soak);     /* Post the Meta Facts. */
                previous_soak = soak;           /* Save the previous soak value. */
                shift_facts_goals();
                soak = find_soak();
        }
        if (! (list_empty(previous_soak))) {
          post_soak_meta_facts(soak);
          intend(select_randomly(previous_soak));   /* Intend one randomly. */
        }
        current_intention = choose_intention();     /* Choose an intention to execute. */
        execute_intention(current_intention);       /* Execute one step of the intention. */
        previous_soak = soak;
}
```

It is basically composed of one meta level reasoning loop inside the normal main loop. The goal of this inner loop is to determine the successive Sets Of Applicable KAs (SOAK), at the light of the concluded beliefs on the previous SOAK. This inner loop stops when the SOAK is empty, i.e. when no applicable KA is found.

3.4 Other Aspects of PRS

Many aspects of the PRS architecture have been designed to address real-time reasoning. For example the system provides mechanisms to handle, in constant time, operations which can be in $O(n)$, such as a look up mechanism in the fact database or the procedure library. In most applications, we can find in advance the maximum number of facts (or at least find an upper bound) which will ever be concluded in the database, and the number of KAs which will be stored in the library. There exist mechanisms (mostly hashing mechanisms) which can thus guarantee an upper bound on an access time.

Moreover, other mechanisms, although seldom presented, prove to be very useful, if not critical in RT applications. We present some of them in the following sections:

Term indexing mechanism in the database. The PRS database uses a term indexing mechanism. The detail of this mechanism, and its complexity can be found in [Stieckel, 1987]. In short, all the facts are indexed in such a way that "consult" operations can be done in constant time. Moreover, this mechanism is particularly useful when one consults the database with facts containing variables (which is most often the case with PRS). However, there is a tradeoff, as the deletion and the conclusion of a fact can be rather expensive. Nevertheless, the overall gain in speed is substantial.

Functional predicates. Functional predicates are predicates which can be expressed as a function of a subset of their arguments which gives the rest of their arguments as a result. In other words, for such a predicate P: $((P\ x_1 \ldots x_n) \wedge (P\ y_1 \ldots y_n) \wedge x_1 = y_1, ..x_j = y_j) \Rightarrow x_i = y_i, i = j+1, \ldots, n$
In this case, P is functional with respect to the first j arguments. In PRS, when a predicate is declared functional, the database automatically dœs the bookkeeping and removes previous values. For example if we consider the following statement: `(pressure tk1 245)`, there is, most likely, only one possible value of pressure at one time, therefore any previously recorded pressure value for `tk1` should be discarded. In this case, if we received `(pressure tk1 250)` few minutes (or seconds) ago, it will be removed from the database.

Triggering mechanism. The triggering mechanism, i.e., the mechanism which finds the currently applicable KAs, has also been optimized for dynamic environments. The KA syntax and semantic require the Invocation Part to contain a condition on triggering goals or facts. In other words, only the presence of one of these goals or facts may render this KA applicable. When procedures are loaded and compiled in the system, several hash tables are built and used by the kernel to very quickly retrieve (in constant time) the procedures which are triggered by a particular fact or a particular goal. Therefore, the kernel dœs not have to scan the whole library of procedures for applicable procedures, but only a subset of those procedures which are "relevant" to a new fact or a new goal. Of course, this dœs not prevent the system from using a full unification to check the applicability of the relevant procedures afterwards, but this is then done on a very small subset of the set of KAs.

Asynchronous Action. The measure of the reaction time of PRS heavily depends on the time of the longest action. Therefore, one of the main concern of a PRS application designer is to keep the actions as short as possible. However, some operations can either take a long time to execute, or may need to synchronize with external events (such as waiting until a user types the response to a question). For this type of action, the system provides a mechanism which allows an action to return before completion, with a particular value which is interpreted by the kernel as "I do not have the result yet, call me back later".

C Language, Real-time Unix. The most recent version of PRS (called C-PRS) has been written in C under Unix. Most people would not consider this as a serious argument for its ability to be used as a RTRS. Indeed from a research point of view, this argument dœs not hold. But from a practical point of view, if you need to put the kernel on the 680X0 board of your mobile robot, you cannot afford a Lisp interpreter. In fact, from our experience with previous versions of PRS written in Lisp, it appears that the overall performances of the C version are far better, and the most visible effect is the size of the C-PRS kernel which is less than 400 Kbytes on a Sparcstation, while the Lisp one was over 10 Mbytes. In many applications, one needs to run more than one C-PRS kernel in parallel (communicating with a message passing mechanism), and in some cases, one even has tens of C-PRS cooperating on a workstations network. It is then obvious that fast and small kernels are preferable even for research study.

As presented above, PRS numerous mechanisms and features participate to its ability to be used in dynamic, real-time environments. Nevertheless, the main characteristics that make it particularly powerful as an embedded reasoning system remain: (1) The semantics of its plan (procedure) representation, which is important for verification and maintenance; (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. Most of these features have been discussed in earlier reports and papers [Ingrand and Georgeff, 1990, Georgeff and Ingrand, 1989, Georgeff and Ingrand, 1990].

4 Blackboard Architecture

Blackboard architectures [Hayes-Roth, 1989] have been heavily used in various systems with real time constraints like BB1 in the Guardian system [Hayes-Roth *et al.*, 1989], RT1 [Dodhiawala *et al.*, 1989] for the pilot associate, REAKT [Lalanda *et al.*, 1992a], or ATOME-TR [Mentec and Brunessaux, 1992].

4.1 General presentation

The Blackboard architecture is generic and provides an interesting framework to develop control and monitoring systems. In short, a Blackboard system is composed of the following modules:

1. the blackboard, a common solution space shared by the Knowledge Sources (KS),
2. the Knowledge Sources (KS), which are mostly independent knowledge modules working on their "part" of the problem, reading their data and writing their results in the blackboard,
3. the Control, which is in charge of selecting the appropriate KSs for execution.

This framework is in fact too general to provide powerful mechanisms to handle RT constraints. Therefore, it must be modified or extended. For example, none of the default mechanisms of the blackboard architecture allows, a priori, a bound on reaction time. Different "techniques" have been used to allow the BB system designer to use its application in a real time environment.

In [Dodhiawala *et al.*, 1989], the authors present RT1, a Blackboard architecture used in the DARPA Pilot Associate program. This particular BB implementation uses four event channels with different priorities to allow the processing of more important events before others.

In [Hayes-Roth *et al.*, 1989] and [Hayes-Roth, 1989], the authors present the Guardian system, a system to monitor patients under ventilator (an artificial breathing machine) in a surgical intensive care unit. Here also there are extensions to the original BB paradigm to handle real-time data. The application is organized around various modules specialized in a particular subtask: scheduler, executor, agenda manager, etc. All these modules are conceptually interesting, but from an implementation point of view, it is rather difficult to figure out what dœs what. Moreover, there seems to be serious design flaws which may lead the system to forget about some events, mainly when the input buffers are full. Another problem we see in this particular implementation is the large overhead induced by the scheduler at each cycle to reorder the applicable KSs.

4.2 REAKT

The REAKT system presented in [Lalanda *et al.*, 1992a, Lalanda *et al.*, 1992b] is also based on a Blackboard architecture. We shall concentrate on this particular implementation which seems to be the most advanced to handle RT problems.

The BB present in REAKT has been enhanced in the following mechanism:

1. The addition of interruptible KSs. This is certainly the most significant change in terms of improvement to handle real-time constraints. In a RT environment, the computation performed in a particular KS should not take an arbitrary long time, otherwise, the reaction time (and consequently the response time) of the system could be arbitrary long. Therefore the use of interruption mechanisms appears to be necessary.

2. The use of the intention concept. The introduction of this notion allows the control mechanism to focus on and reason about tasks instead of individual rules. Intention offers a better level of abstraction when the control needs to deal with tasks, or with sets of rules dealing with a particular goal.

3. The use of RETE algorithm in KSs. This change allows a faster triggering mechanism to find out the applicable rules.

5 Comparison of the two approaches

If we compare the two approaches, and particularly the REAKT BB system and PRS, we find some interesting common points. However, there are some subtle differences which we believe give an edge to the PRS architecture.

The interruption mechanism in REAKT is also present in PRS (by design). When procedures are executed, the control gœs back to check the input buffer after each goal posting, or each action execution. As a result, the reaction time of the system is very small, and there are even mechanisms to enable actions to give back control when they take too long to complete.

The BB architecture uses the KS representation. Each of these KSs is a set of rules organized in a single module because of their relation with the problem this particular KS is solving. In PRS, a similar partition of an application can also be considered by using multiple C-PRS kernels which can easily communicate between each other.

In REAKT, there is the possibility of using a particular KS to control the execution of the other KSs. This is done in a particular intention: the control intention. We believe that the PRS meta level mechanism is somewhat more general and more flexible because it provides more control on the main loop of the kernel (KA applicability, intending KA, re shuffling the intention graph, etc). Moreover, this mechanism is designed in such a way that it is "self" interruptible.

The notion of intention, introduced in PRS in 1987, is present in both architectures and appears to be very practical when it comes to deal with change of focus or tasks management. However, its use appears to be more advanced in PRS, as an *intention graph* is built from the set of intentions. This intention graph, from which only the root can be executed at any time, allows the user to easily implement and represent complex priority mechanisms. Moreover, meta level KAs can change dynamically the layout of this graph to implement a particular configuration of execution. For example, after a serious alarm, a meta level KA may reorganize the intention graph to explicitly put the new intention, which takes care of the alarm, before all the other intentions.

The knowledge representation used in both systems is quite different. Most BB systems use a rule based representation, while PRS uses a procedural representation of knowledge. We think that, in most real-time applications, such as monitoring and supervision of complex processes, the procedural representation is better suited as it enables the user to keep explicit the control structure embedded in the knowledge and in the various

operational procedures. In fact, for this type of application, there often exist books of procedures and plans to execute in specific and critical situations. Moreover, when it comes to write meta level control, it is easier to prove its validity if it is written as a procedure then a set of rules.

At last, but not least, PRS guarantees some temporal properties providing some well defined assumptions. We show in [Ingrand and Coutance, 1993], that an upper bound on reaction time is a prerequisite to most temporal properties (such as the bound on response time), and we show that a bound on reaction time can be guaranteed.

If we consider μ_{Max}, the maximum frequency of arrival of events, and T_{Pars}, the parsing time, i.e., the time spent looking for KAs and choosing one KA.

If there is a bound on the parsing time T_{Pars}, and $\mu_{Max} \times T_{Pars} < 1$, and the meta level KAs (i.e., the KAs which may trigger in the inner loop of PRS main loop), do not loop on themselves, and the set of applicable KAs in this inner loop monotonically decreases in at most n inner loop, then it exists a bound on reaction time.

Note that this condition dœs not depend on the longer action of the system: T_{Exec}^{Max}. However the value of the reaction time depends on it.[3]

The upper bound on reaction time is then[4]: $T_{Rea}^{Max} = \frac{T_{Exec}^{Max}+c}{1-\mu_{Max} \times T_{Pars}}$

To our knowledge, no such study or analysis has been done with Blackboard based real-time reasoning system[5] while it appears to be a prerequisite to most temporal properties.

6 Other Systems and Architectures

Various other reasoning systems and architectures offer some capabilities to handle real time constraints while executing. In some cases, the system or the language is defined in such a way that some real time properties can be identified, as is the case for Synchronous Systems. Nevertheless, few reasoning system architectures or algorithms allow any provable real time property, which remains the real criteria from our point of view.

6.1 Synchronous Systems and Compilation Techniques

Synchronous systems and systems using compilation techniques can usually guarantee an upper bound on reaction time and even on response time. However, the tradeoff is very high and "response time" is to be taken in a very restrictive way. All these systems are unable to compute, in a predictable bounded time, arbitrary operations as simple as factorial, just because everything needs to be known in advance and can therefore be compiled in a finite state automata of reasonable size. Moreover, these approaches are often too restrictive. They do not provide much flexibility at runtime and, in most cases, the produced result is only interesting if the produced automata has been parallelized by compiling it in an electronic circuit.

The synchronous approach can intuitively be defined as using a model where the outputs of the system are produced synchronously with their inputs...Even if this seems rather difficult to implement, it turns out to be easier to describe and analyze temporal properties for this system than for the asynchronous approach. An abundant literature flourishes on these systems, but seldom shows interest for the AI field and the Reasoning Systems we deal with in this paper.

The Rex/Gapps approach presented in [Kaelbling, 1988] is in fact similar and can be placed in the family of real-time synchronous systems. The user can describe a set of automata which completely defines all the system operations. However, a couple of critics can be made to this approach. The produced automata is the "worst case" automata, i.e., all computations are done in every cycle, whatever inputs are fed in the system. If this is not critical in a parallelized solution (such as a silicium chip), this can be rather annoying if the automata is interpreted on a sequential machine sharing resources with other processes. Another problem with this approach is that the semantic of goals presented in Gapps dœs not hold very well in a compiled environment,[6] for example, there is no difference (from the automata point of view) between an "achieve" or "maintain" goal.

The KHEOPS system [Ghallab and Philippe, 1988], using expert rules in propositional logic, is also a system with a bound on reaction and response time. Here also, the tradeoff is important, but at least, the produced decision tree is optimal since only the computation required to produce the result is done.

For all the systems presented above, response time and reaction time are the same. However, we should keep in mind that a response in this case can only be obtained by at most one transition of the state automata (which is a finite automata).

6.2 High Level Architecture

Other systems such as Phœnix [Cohen *et al.*, 1989] or RAP presented in [Firby, 1989] are intended to be used, or have been used, in real-time environments. However, none of them seems to provide any proof of their ability

[3] Note that to exhibit this value, we computes it in the worst case. Therefore, this value is in fact less important than its existence.

[4] c is a constant corresponding to the time needed to intend a new condition and to choose one intention to execute.

[5] Of course, this dœs not mean that it cannot be done.

[6] After all, is a compiled goal still a goal?

to guarantee a bound on reaction time or other temporal properties.

In [Musliner *et al.*, 1993], the authors present a cooperative intelligent real-time control architecture (CIRCA). They propose a cooperative architecture, in which they introduce two separate subsystems: an AI reasoning system[7] and a real-time subsystem which cooperate while still addressing the problem each system is designed for. This is actually a very interesting and promising approach. Some recent work done at LAAS [Revillod, 1992] attempts to implement a somewhat similar architecture by combining PRS and the KHEOPS system presented above, in a multi layered architecture, where each layer guarantees a number of temporal properties: at the lowest level KHEOPS guarantees response time, while upper layers using PRS provide more advanced reasoning with still bounded reaction time, and if desired, other temporal properties implemented with meta level KAs.

To conclude this section on compiled systems and advanced architectures, let us point out that strong restrictions prevent high level architecture knowledge systems from being compiled in situated automata. For example, if we consider the assumptions we make to still guarantee reaction time in PRS, why cannot we just compile the whole system in a huge automata? If this is theoretically possible, it is in practice unreasonable. The number of states necessary to represent a group of application KAs in a propositional logic can still be reasonable. However, if we later consider the meta level reasoning, i.e., all the possible ways and orders in which these KA instances can be applicable and executed, we are likely to quickly reach an unreasonable number of states.

7 Conclusion

In this paper we have shown that it is rather difficult to define what a Real-Time Reasoning System is. It appears to be more correct to define a number of temporal properties, and then to prove that a particular reasoning system satisfies any given property under some specific assumptions.

We have presented the PRS system and the Blackboard architecture, which have both been used in real-time applications. The two approaches are quite different, but have a number of interesting characteristics in common. However, we believe that some of these critical characteristics are better handled by the PRS architecture, either because of its original design, or because of more generic mechanisms.

We concluded with an overview of some of the current systems and architectures which have been used or developed to be used in real-time environments. If some of them can guarantee high level temporal properties, such as a bound on response time, it is usually done by using some compilation techniques which tend to freeze all the reasoning at compile time. Other systems are able to carry on more advanced reasoning, but usually lack formal proofs of their ability to satisfy some of the temporal properties presented.

We believe that the RTRS research area will grow and become ever more accepted in the AI community (and more generally in the computer science community), if and only if RTRS designers can prove that their systems satisfy the temporal properties their applications require.

As for PRS, a commercial version, written in C and named C-PRS [Coutance and Ingrand, 1992] is available and runs on various Unix platforms, under the X11/Motif graphic environment. It is used in various applications ranging from telecommunication network management to industrial process control and supervision. C-PRS also runs under VxWorks (a real-time Unix operating system), and among other tasks, we are using it on board the various mobile robots at LAAS as a system to do control execution and plans refinement in the MARTHA Esprit project.

References

[Benveniste and Berry, 1991] A. Benveniste and G. Berry. The Synchronous Approach to Reactive and Real-time Systems. *Proceedings of the IEEE*, 79(9):1270–1282, 1991.

[Cohen *et al.*, 1989] P. R. Cohen, M. L. Greenberg, D. M. Hart, and A. E. Howe. Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments. *Artificial Intelligence Magazine*, 10(3):32–48, 1989.

[Coutance and Ingrand, 1992] V. Coutance and F. F. Ingrand. *C-PRS Development Environment Manual.* ACS Technologies, 5, Place du Village d'Entreprises, BP 556, 31674 LABEGE Cedex, France, 1992.

[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.

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

[7] The AI reasoning system main loop is in fact very similar to the one used in PRS as described in [Ingrand and Georgeff, 1990].

[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, 1990] 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 Lansky, 1986] M. P. Georgeff and A. L. Lansky. Procedural Knowledge. *Proceedings of the IEEE Special Issue on Knowledge Representation*, 74:1383–1398, 1986.

[Ghallab and Philippe, 1988] M. Ghallab and H. Philippe. A compiler for real-time Knowledge-based Systems. In *International Workshop on Artificial Intelligence for Industrial Applications*, Hitachi City, Japan, May 1988. IEEE.

[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, December 1989.

[Ingrand and Coutance, 1993] F. F. Ingrand and V. Coutance. Real-Time Reasoning using Procedural Reasoning. Technical Report 93-104, LAAS/CNRS, Toulouse, France, 1993.

[Ingrand and Georgeff, 1990] F. F. Ingrand and M. P. Georgeff. Managing Deliberation and Reasoning in Real-Time AI Systems. In *Proceedings of the 1990 DARPA Workshop on Innovative Approaches to Planning*, Santa Diego, California, U.S.A, November 1990.

[Ingrand *et al.*, 1989] F. F. Ingrand, J. Goldberg, and J. D. Lee. SRI/Grumman Crew Members' Associate Program: Development of an Authority Manager. Final Report, Artificial Intelligence Center, SRI International, Menlo Park, California, U.S.A, 1989.

[Ingrand *et al.*, 1992] F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An Architecture for Real-Time Reasoning and System Control. *IEEE Expert, Knowledge-Based Diagnosis in Process Engineering*, 7(6):34–44, December 1992. Also available as LAAS Technical Report 92-521.

[Kaelbling, 1988] L. P. Kaelbling. Goals as Parallel Program Specifications. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 60–65, Saint Paul, Minnesota, U.S.A, 1988.

[Lalanda *et al.*, 1992a] P. Lalanda, F. Charpillet, and J-P. Haton. A Real-time Blackboard based architecture. In *Proceedings of the Tenth European Conference on Artificial Intelligence*, Vienna, Austria, 1992.

[Lalanda *et al.*, 1992b] P. Lalanda, F. Charpillet, and J-P. Haton. *Conduite du raisonnement dans un système à base de tableau noir temps réel.* PhD thesis, University of Nancy, CRIN, Nancy, France, December 1992.

[Mentec and Brunessaux, 1992] J-C. Le Mentec and S. Brunessaux. Improving Reactivity in a Blackboard Architecture with Parallelism and Interruptions. In *Proceedings of the Tenth European Conference on Artificial Intelligence*, Vienna, Austria, 1992.

[Musliner *et al.*, 1993] D. J. Musliner, E. H. Durfee, and K. G. Shin. CIRCA: A Cooperative Intelligent Real-Time Control Architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6), 1993. To Appear.

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

[Revillod, 1992] F. Revillod. Une Architecture Décisionelle pour le Contrôle d'un Robot Autonome. Rapport de Stage Ecole Supérieure d'Aéronautique et de l'Espace, LAAS/CNRS, Toulouse, France, September 1992. In french.

[Stieckel, 1987] Mark Stieckel. Term Indexing Database. Technical Report 87-5, SRI International, Menlo Park, California, U.S.A, 1987.

[Wesley *et al.*, 1991] L. Wesley, F. F. Ingrand, J. Rushby, J. Garcia Luna, and J. D. Lee. Application of PRS to Network Management System. Final Report, Artificial Intelligence Center, SRI International, Menlo Park, California, U.S.A, 1991.