PRIMES: Progressive Reasoning and Intelligent Multiple Methods System *

Jean-François Dauchez, Abdel-Illah Mouaddib, Éric Grégoire
CRIL/IUT de Lens-Université d'Artois
Rue de l'Université S.P. 16, F-62307 Lens Cedex France
{dauchez,mouaddib,gregoire}@cril.univ-artois.fr

Abstract

In this paper, PRIMES (Progressive Reasoning and Intelligent multiple MEthods System), a new architecture for resource-bounded reasoning that combines a form of progressive reasoning and the so-called multiple methods approach is presented. Each time-critical reasoning component is designed in such a way that it delivers an approximate result in time whenever an overload or a failure prevents the system from producing the most accurate result.

The architecture of PRIMES is presented, which includes a cooperative control module using a new incremental scheduling algorithm allowing both progressive reasoning and multiple intelligent methods to coexist. In this way, we hope to extend the actual scope of these basic real-time systems to more real-world application domains.

1 Introduction

One recent active research direction in real-time AI has concerned the development of large applications or architectures that embody real-time aspects in many components. The eventual goal is to reach overall real-time performance through several resource-bounded components. To this end, several architectures have been developed, most notably Guardian [9], Phoenix[10], CIRCA [22], TAEMS[2], RT-SOS[19; 15], REAKT[13] and other reactive systems such as [5], [12]. Indeed, most of these systems allow one to build real-time AI components that are to be assembled in order to deliver larger real-time application systems. Two major issues arise in the development of these systems. First, real-time AI components dedicated to particular and domain-specific real-time problems are to be built. Second, new techniques are to be defined to guide the behavior of these components. Accordingly, recent research in the real-time AI community has focused on:

- defining and elucidating particular useful real-time techniques. The most popular classes of these techniques are anytime algorithms [1; 23], multiple methods [3; 6] and progressive reasoning [14] ones.
- using these techniques as backbones to assemble real-time AI systems. The RT-SOS [19] and REAKT [13] systems are made of progressive reasoning components, TAEMS[2] uses components representing multiple methods whereas Zilberstein and Russell proposed a system composed with anytime algorithms [24].

All these systems offer *one* formalism for implementing resource-bounded reasoning. Accordingly, their expressiveness and their actual use in large various application domains are somewhat restricted. In this paper, a real-time system implementing both a progressive reasoning approach (an anytime one [20]) and a multiple methods one is presented. First, let us briefly recall the basic difference between these two alternative forms of resource-bounded reasoning.

- The multiple methods approach: different available methods delivering solutions of an increasing quality, each of them requiring a specific non-interruptible amount of computation time [6].
- The progressive reasoning or anytime algorithms: solutions are approximated by constructing a rough one and by refining it through a hierarchy of reasoning levels that can be interrupted at any time [18].

Problems that cannot be approximated are addressed through a unique method or level of reasoning. The combination of the above two approaches should contribute in increasing their expressiveness and in allowing more real-world problems to be addressed. However, when embedding resource-bounded components based on progressive reasoning or multiple methods in large systems,

*This work has been supported by the Ganymède-II project of the Contract Plan Etat/Nord-Pas-De-Calais and by the MENESR.

the problem of controlling and guiding their behavior can become much more complicated. In order to solve this problem a new cooperative control module is proposed. It normalizes the representation of the various resource-bounded components and then uses a scheduling algorithm for guiding their behavior. The construction of such a system is motivated by different applications like real-time world-wide web services, railways control, flexible operating system [11] and navigation robots. PRIMES is particularly dedicated to real-time World-Wide Web services and navigation robots.

To summarize, a new architecture is proposed, that is based on a cooperative control module allowing one to guide the behavior of resource-bounded components. These components are designed as progressive reasoning and as multiple methods ones. A new scheduling algorithm that is able to guide both components is proposed. This scheduling algorithm can be seen as a trade-off between the *Incremental Scheduling Algorithm* [15; 21] and the *Design-to-Time Scheduler* [6]. By combining several such techniques, it is hoped that this system will allow more application domains to be addressed in dynamic and critical hard real-time situations.

The paper is organized as follows: the second section presents the architecture model of PRIMES and its different communicating modules. The third section reviews different scheduling algorithms and their properties. An incremental multiple methods scheduling algorithm dedicated to guide both progressive reasoning and multiple methods is given. Section four illustrates how PRIMES meets usual real-time requirements. The general conclusion is given in section 5, together with perspectives for further research and applications.

2 The architecture of PRIMES

The architecture of PRIMES is based on different communicating modules (Figure 1). It includes a *Library of reasoning components* that contains problem-solving components, each of them being based on either progressive reasoning or on multiple methods techniques, a *triggering mechanism* that maps the set of goals of the system to a set of reasoning components, a *calendar* containing the schedule of components to execute, a *decision-maker* that constructs a schedule and the *timer* that synchronizes the execution of the schedule and updates it, if necessary, after the end of one component execution.

2.1 Library of reasoning components

The library, hand-coded by the application designer, contains various domain-specific reasoning components. Reasoning components can be based either on *progressive reasoning* that can be interrupted at any time and

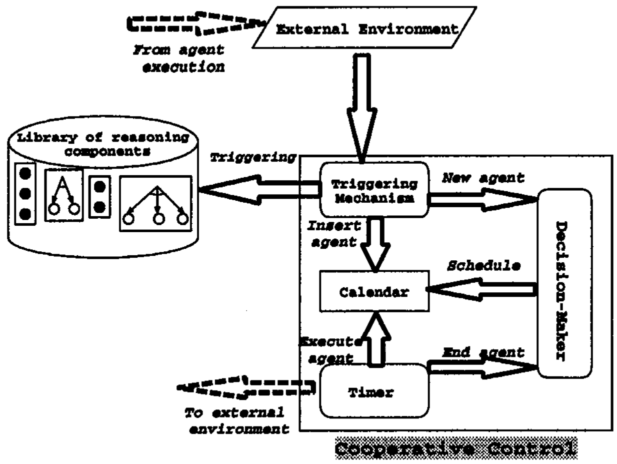

Figure 1: Architecture of PRIMES

that it is dedicated to goals that can be achieved at different levels of details, or on *multiple methods* that are non-interruptible and that are dedicated to goals that can be achieved by different methods with different qualities (Figure 2). For the other goals, the reasoning component consists of one reasoning level.

• A progressive reasoning component α is represented

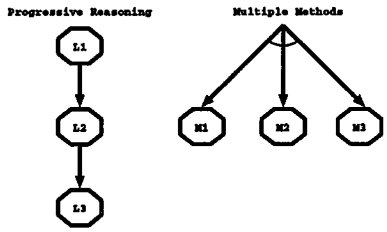

Figure 2: Reasoning component structures

by a linear precedence-constraint graph made of successive levels L^i_α: the level L^i_α can begin its execution only after the level L^{i-1}_α is completed. The level L^i_α is thus the immediate successor of L^{i-1}_α, and the output of L^{i-1}_α is one of the inputs of L^i_α. When a level L^i_α is interrupted before completing its processing, the result from the level L^{i-1}_α is delivered.

• A multiple methods component α is represented by an OR-tree of methods M^i_α. The OR-tree consists in activating one of its methods M^i_α where the method M^i_α is longer and more complete that the method M^{i-1}_α. Any method M^i_α is non-interruptible.

In the following, it is shown how PRIMES guides, through its *Decision-Maker*, the behavior of reasoning components based on both techniques.

2.2 Triggering mechanism

The *Triggering Mechanism* is responsible for the interaction between the system and the external environment.

It receives messages from the external environment, conveying data describing new facts about the world. The *Triggering Mechanism* analyzes the current situation and then generates the goals to be achieved. Afterwards, the *Triggering Mechanism* creates, for each goal, an *agent* to execute an instance of the appropriate reasoning component selected from the library. *An agent is thus defined as an instance of a reasoning component that is created to achieve a goal.* The choice of the reasoning components to be assigned to a specific goal category is described inside a control rule base, which is up to the application designer.

For example, let us consider an office surveillance robot application. We assume that the *Triggering mechanism* receives the data `object O detected at location X`. The outputs generated from these data take the form of two goals: {`Go_to(X)`, `Look_for_Object(O)`}. The *Triggering Mechanism* creates an agent that contains an instance of the reasoning component `Navigate_to` that achieves the goal `Go_to(X)` and an instance of the reasoning components `Get_objet`, `Analyze_object` that achieves the goal `Look_for_Object(O)`, respectively.

Created agents are put in the *Calendar* by the *Triggering Mechanism* and are to be scheduled by the *Decision-Maker*.

2.3 Timer

The *Timer* reasons about its real-time clock and the time constraints of agents. It is responsible for the following tasks:

- *Execution of agents*: the *Timer* uses a real-time clock to synchronize the execution of agents. This latter receives the *begin_time* of the first agent in the *Calendar*. Afterwards, it compares this time to the current time got from the real-time clock. An execution event is fired when the current time matches the *begin_time* of the first agent in the calendar. In this case, the agent is retracted from the calendar and its execution is started. The execution of an agent modifies the state of the external environment and then a new situation is assessed.
- *Removing agents*: the *Timer* has a list of events corresponding to the deadlines of the agents. An event is fired when one deadline in the list is met and then the corresponding agent that cannot be executed is removed from the *Calendar*.
- *Monitoring execution*: at the end of an agent execution, the *Timer* uses the consumed execution time to update the schedule. It then updates its list of events by inserting the *begin_time* of the first agent in the *Calendar*. Furthermore, the *Timer* sends a message to the *Decision-Maker* to indicate the modification in the schedule.

2.4 Decision-Maker

This module is responsible for constructing a schedule of agents in the *Calendar*. It is based on scheduling algorithms and is activated in two situations:

- *Arrival of a new agent*: At the receipt of a message from the *Triggering Mechanism*, the *Decision-Maker* performs its scheduling algorithm to determine the execution window of the new agent by defining its *begin_time* and its *end_time*.
- *Updating the schedule*: At the receipt of a message from the *Timer*, the *Decision-Maker* reschedules agents in the *Calendar* by adapting their levels of approximation according to the deviation from the predetermined length of time occurred during the past execution of agents. The *Decision-Maker* does not start its scheduling from the beginning but revises the current schedule by increasing/decreasing levels of approximation depending on gained/lost time during execution. In the next section different algorithms dedicated to this module are presented and discussed.

2.5 Calendar and cooperative control

The *Calendar* contains the set of agents created by the *Triggering Mechanism*, each of them with its required time window. Such a window is represented by a pair (*begin_time*, *end_time*) defined by the *Decision-Maker* module. The *Calendar* is a memory ensuring interaction between the different modules of the cooperative control. Indeed, the *Triggering Mechanism* inserts new agents in the *Calendar* that the *Decision-Maker* schedules and that the *Timer* executes. These different modules interact through message-passing and the *Calendar* shared memory. These two communication mechanisms ensure the cooperation between control modules.

The *Triggering Mechanism* is responsible for selecting reasoning components from the *library* and activates the *Decision-Maker* to perform its scheduling algorithm. This latter is responsible for constructing the schedule by adjusting the approximation level of different components to ensure that the overall system meets hard deadlines and also achieves the system goals as closely as possible. The *Timer* is responsible for monitoring the execution of agents in the *Calendar*. The *Timer* executes the agents one by one. It updates (i.e. advances or delays) the schedule in the *Calendar* when the execution of an agent is deviated from the predetermined length of time. The *Timer* sends this modification of the schedule to the *Decision-Maker* so that agents are rescheduled when deadlines are violated. The modules of cooperative control communicate in an asynchronous manner. Indeed, the *Triggering Mechanism* analyses asynchronous messages coming from the external environment and activates the *Decision-Maker*. The *Timer* is activated as soon as an important event occurs. Indeed, its list of events contains important dates at which it must start the execution of agents or indicating that deadlines are reached.

Such a form of cooperative management and control, as illustrated in Figure 1, ensures performance trade-offs to be made based on resource limitations. Indeed, thanks to the interactions between its cooperative control modules combined with the flexible structure of its reasoning components, the system guarantees that it will produce a solution in timely fashion, with a traded level of approximation. One salient features of PRIMES lies in its ablility to guide both the behavior of components based on a progressive reasoning or on multiple methods. In this respect, a scheduling algorithm able to support both techniques and meet the usual main requirements of real-time intelligent systems is presented in the next section.

3 Scheduling issues arising from PRIMES specific real-time requirements

The cooperative control module must manage the system so that the real-time requirements are met. *Timeliness* requires this module to propose a schedule that guarantees all the agents' time constraints. The *Decision-Maker* module is responsible for constructing these schedules. It is based on scheduling algorithms that must be able to manage a dynamic situation (i.e. a new agent arrives) without rescheduling, taking the information gathered during execution into account. These algorithms should support unexpected interrupts when an important event occurs and return an approximate schedule. Furthermore, they must deal with both progressive reasoning and multiple methods techniques. In the following, the scheduling algorithms, *Design-to-Time* introduced by Garvey *et al.* [6] to schedule multiple methods components, and the *Incremental Scheduling* introduced by Mouaddib *et al.* [15; 21] to schedule progressive reasoning components are reviewed. An Incremental multiple methods scheduling algorithm dedicated to progressive reasoning and multiple methods techniques is also introduced. In the following sections these algorithms, their performance and main characteristics are described.

In the following a set $\mathcal{A}$ containing n reasoning components α, β,..., γ sorted according to their deadlines is considered. The problem is to define, for each reasoning component, the optimal approximation level while all deadlines stay respected.

3.1 Incremental scheduling algorithm

In this section, a scheduling algorithm dedicated to progressive reasoning components is presented [21]. Both its formal framework and its processing mode are described.

Preliminary definitions:

Progressive reasoning level formulation Actually, each progressive reasoning component α is a composite one made of progressive reasoning levels L^i_α. Each progressive reasoning level L^i_α is characterized by its required, a-priori, computation time $C_{L^i_\alpha}$ and the intrinsic value of solution quality $V_{L^i_\alpha}$.

Utility of progressive reasoning level The utility $U_{L^i_\alpha}$ of a reasoning level L^i_α is defined as follows:

$$U_{L^i_\alpha} = V^i_\alpha - Cost(C_{L^i_\alpha})$$

where $Cost(C_{L^i_\alpha})$ the cost of consuming an amount of time $C_{L^i_\alpha}$. The utility concept is then used to classify the different possible levels. Actually, a utility-based approach is defined to determine the level of reasoning to be selected, and to allow for a scheduling revision when execution is slower or faster than predicted.

Adopted structure As indicated above, a list $\mathcal{A}$ of n reasoning components α, β,..., γ sorted according to their a-priori deadlines is considered, using an *Earliest-Deadline-First* scheduling algorithm. The schedule will be computed progressively, i.e. level by level. At each iteration step, the algorithm attempts to extend a tentative schedule by allowing additional levels of reasoning to be taken into account.

Figure 3 illustrates the wave-like approach to this incremental construction. The current tentative schedule, noted $\mathcal{E}$, may already involve several reasoning levels for the various components. We call *Frontier*, noted $\mathcal{F}$, the set of the immediate successor levels for all the components of $\mathcal{E}$. The elements of $\mathcal{F}$ will be considered for inclusion in the schedule at the next iteration step.

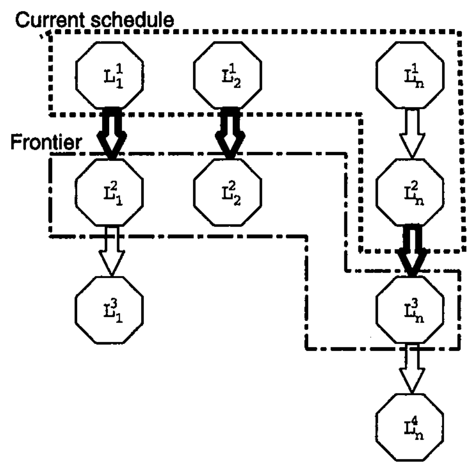

Figure 3: A wave-like structure

The scheduling algorithm

Accordingly, the schedule is constructed step by step through a series of expansion cycles. Initially, a tentative schedule that contains the first level of reasoning L_α^1 for each component α ($\alpha \in \mathcal{A}$) is considered. This schedule is then refined progressively. At each expansion cycle, all the levels of the *frontier* are tentatively introduced, allowing one additional level of reasoning for each component. When this expansion succeeds (i.e. when no deadline is violated), a new expansion cycle is undertaken. When an expansion cycle fails to deliver a schedule respecting all deadlines, levels exhibiting the least utility are discarded. This processing is repeated until the set $\mathcal{F}$ is empty, i.e. until no further expansion is possible. Let us stress that this algorithm can be interrupted at any time while still delivering a scheduling.

This algorithm is thus based on the following steps:

• *Initialization step*:
First, the schedule $\mathcal{E}$ is empty and the *frontier* $\mathcal{F}$ is intialized with the first reasoning level L_α^1 of all components α ($\alpha \in \mathcal{A}$). Accordingly, the first expansion cycle will consist in constructing a preliminary schedule with all reasoning levels L_α^1 of $\mathcal{F}$.

$$\mathcal{E} = \emptyset \text{ and } \mathcal{F} = \{L_\alpha^1 \mid \forall\, \alpha \in \mathcal{A}\}$$

• *Expansion step*:
This step consists in extending $\mathcal{E}$ to all the levels belonging to $\mathcal{F}$:

$$\mathcal{E} = \mathcal{E} \uplus \mathcal{F} \text{ and } \mathcal{F} = \emptyset$$

The operator $\uplus$ allows one to insert additional levels of reasoning to the reasoning components of the schedule by respecting the progressive structure of components (Figure 3) [21]. The *feasibility test* step is then invoked to verify whether no deadline is violated.

• *The feasibility test step*:
Let us note D_α the deadline of a reasoning component α. The expansion steps fails to deliver a schedule when at least one deadline D_γ is violated.

$$\exists\, \gamma \in \mathcal{A}: \left(\sum_{\{\delta\, \in\, \mathcal{A},\ D_\delta\, \leq\, D_\gamma\}} \sum_{i=1}^{i=d_\delta} C_{L_\delta^i}\right) > D_\gamma,$$

where $L_\delta^{d_\delta}$ represents one last level introduced in $\mathcal{E}$.

When such a failure is encountered, the *approximation* step is invoked. Otherwise, the *new frontier* step is invoked to compute a new set $\mathcal{F}$.

• *Approximation step*:
The level with the least utility, noted L_{min}, is discarded when an expansion cycle fails to deliver a schedule with all the deadlines respected. It is selected among the levels of reasoning L_δ^k inserted by the last expansion cycle:

$$L_{min} = arg(MIN_{L_\delta^k}(U_{L_\delta^k}))$$

Accordingly, the total required time of the schedule is reduced from the computation time of L_{min}. Afterwards, the *feasibility test* is called again.

• *New frontier step*:
Whenever it exists, the immediate successors of each reasoning level inserted by the last expansion cycle is inserted in the frontier $\mathcal{F}$.
The *expansion* step is invoked if the frontier is not empty. Otherwise, the algorithm stops and returns the current schedule $\mathcal{E}$.

3.2 Design-to-Time scheduling algorithm

Design-to-Time was introduced in [6] to generalize the approximate processing developed in [3]. Let us briefly describe the principles behind this scheduling algorithm. The interested readers can find more details in [6].

Preliminary definitions:

Multiple solution methods formulation Each reasoning component α has multiple solution methods M_α^i available for solving a problem, where the increasing method number i entails a longer but more complete method. Each method M_α^i has an estimated computation time $C_{M_\alpha^i}$ and one intrinsic value of solution quality $V_{M_\alpha^i}$. $C_{M_\alpha^{i+1}}$ is longer than $C_{M_\alpha^i}$ while $V_{M_\alpha^{i+1}}$ is greater than $V_{M_\alpha^i}$.

Utility of a method The utility of each method $U_{M_\alpha^i}$ obeys a similar definition as the one presented in §3.1.

The scheduling algorithm

This algorithm constructs a schedule of agents in $\mathcal{A}$ that meets the timing constraints and maximizes the quality of the agents. To this end, it schedules the methods with the highest quality and then tries to ensure that no constraint is violated. If no schedule can be found, the scheduler changes the problem-solving method of the least important agent to use a faster but less accurate method. This reduces the total run-time of the schedule. This processing is repeated until no deadline is violated. Whenever all the agents are approximated to their quickest and less accurate methods and no schedule is found, then the scheduler discards some agents (of the least importance) from the schedule until this later becomes feasible. Such an algorithm constructs a schedule with the maximum possible quality without missing any deadline.

3.3 Incremental scheduling algorithm for multiple methods

The most salient feature of the incremental algorithm lies in its ability to deliver a schedule at any time. Consequently, it should be suitable for applications requiring a bounded-resource schedule that can be interrupted unexpectedly.
Let us recall that our motivation behind the development

of PRIMES was twofold. First, we wanted our scheduling algorithm (and the schedule itself) to be interruptible while delivering a schedule anyway. Accordingly, we selected the incremental scheduling approach. Second, we wanted to accomodate both forms of progressive reasoning and multiple methods.

Since the *incremental scheduling* approach deals with progressive reasoning only, two possible ways to overcome this limitation were available. First, we could have tried to represent multiple methods under the form of a progressive reasoning structure. No clear and easy way to accomplish this seems available. Consequently, we went on representing progressive reasoning under the form of multiple methods and on adapting the *incremental scheduling* accordingly. In the following, such an original approach is presented.

From progressive reasoning to multiple methods The main goal of this mapping is to take advantage of algorithms guiding the behavior of anytime algorithms for guiding multiple methods. In [7], it is proposed to represent an anytime algorithm with multiple methods and use *Design-to-Time*. In this sense, a straightforward way to encode progressive reasoning under the form of multiple methods is proposed in [16]. Indeed, the hierarchical structure of progressive reasoning containing d levels is mapped to multiple methods as follows (Figure 4):

- The 1^{st} method: the M_α^1 method consists in activating the first level L_α^1 of the reasoning level hierarchy. This method is the fastest but the less accurate one.

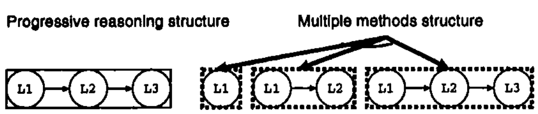

Figure 4: Mapping from progressive reasoning to multiple methods

- The i^{th} method: the method M_α^i consists in activating $\{L_\alpha^1, L_\alpha^2, \ldots, L_\alpha^i\}$. This method is faster but less accurate than the method M_α^{i+1}.
- The last method: the method M_α^d consists in activating all the reasoning levels of the hierarchy. This method is the slowest but the most accurate one.

This mapping allows one to interpret a progressive reasoning agent as a multiple methods agent.

Incremental Scheduling algorithm for multiple methods With the mapping presented above, we can represent all reasoning components as multiple methods components. Then, we could directly use the *Design-to-Time* scheduler. However, it is advantageous to use the *Incremental Scheduler* because of its high performance and its suitability to critical time pressure situations. Consequently, a version of the *Incremental Scheduling* algorithm to multiple methods is required. It uses $\mathcal{E}$ as a set of current scheduled methods while $\mathcal{F}$ contains the immediate successors of the scheduled methods. The algorithm consists in scheduling, first, the fastest and less accurate methods of the agents. When a schedule is found, the scheduler improves it by changing scheduled methods M_α^i in $\mathcal{E}$ with their respective immediate successor methods M_α^{i+1} in $\mathcal{F}$ that are longer and more precise. In the following the basic steps of this algorithm are described:

- *Initialization step*:

The schedule is initialized with the fastest but less precise methods of the reasoning components.

$$\mathcal{E} = \{M_\alpha^1 \mid \forall \alpha \in \mathcal{A}\}$$

Then, go to the *the feasibility test* step.

- *Expansion step*:

The frontier $\mathcal{F}$ becomes the new current tentative schedule:

$$\mathcal{E} = \mathcal{F}$$

Go to the *feasibility test* step.

- *Feasibility test*:

Let us note D_α the deadline of a reasoning component α. The expansion step fails to deliver a schedule when at least one deadline D_γ is violated.

$$\exists \gamma \in \mathcal{A} : (\textstyle\sum_{\{\delta \in \mathcal{A},\, D_\delta \leq D_\gamma\}} C_{M_\delta^i}) > D_\gamma, \text{ where } M_\delta^i \in \mathcal{E}.$$

If the schedule fails go to the *approximation step*, else go to the *new frontier* step.

- *Approximation step*:

When no schedule is found, the algorithm replaces the method M_α^i exhibiting the least utility by its immediate predecessor M_α^{i-1} (when it exists), which is faster but less precise, and thus leads the total run-time of the schedule to be reduced. Formally, the method M_{min}^k to be replaced is selected in $\mathcal{E}$ in such a way that:

$$M_{min}^k = arg(MIN_{M_\beta^j \in \mathcal{E}}(U_{M_\beta^j}))$$

When k matches 1 (i.e. when M_{min}^1 is selected to be replaced), the scheduler discards the agent *min*.
Go to the *feasibility test* step.

- *New frontier step*:

Whenever it exists, the immediate successor of each reasoning component in $\mathcal{E}$ is inserted in the frontier $\mathcal{F}$, i.e.

$$\mathcal{F} = \{M_\alpha^{j+1} \mid \forall M_\alpha^j \in \mathcal{E}\}$$

Then, the *expansion* step is invoked if the frontier is not empty. Otherwise, the algorithm stops and returns the currently obtained schedule $\mathcal{E}$.

3.4 Complexity and suitability of the algorithms

The complexity of the *Design-to-Time* and the *Incremental Scheduling* algorithms are studied in the "worst-case" S_w and the "best-case" S_b. By the "worst-case" S_w, we mean the hard critical time situation where only the first levels of approximation (the first reasoning level for progressive reasoning or the fastest and less precise methods) are schedulable, while the best-case S_b is the situation where a schedule is found using the deepest level of approximate reasoning.

Let K be the average number of methods or reasoning levels for one reasoning component. The following table describes the time-complexity results for *Design-to-Time* and the new *Incremental Scheduler*. In particular, it shows us that the *Incremental Scheduler* is more efficient than *Design-to-Time* in constrained situations (with a K factor).

	S_w	S_b
Design-to-Time	$O(Kn^2)$	$O(n)$
Incremental Scheduler	$O(n^2)$	$O(Kn)$

Although *Design-to-Time* is in turn more efficient in underconstrained situations, we believe that the *Incremental Scheduler* should often be preferred also in these situations because it accomodates both progressive reasoning and multiple methods approaches.

4 How does PRIMES meet standard real-time requirements?

In [4] Dodhiawala *et al.* outlined the major requirements of real-time intelligent systems. In this section, how PRIMES meets these requirements is described:

- **Responsiveness:** This property lies in the ability of the system to stay alert to incoming events. Since the interactive real-time *Triggering Mechanism* module is primarily driven by external inputs, PRIMES recognizes when such an input is available, through message-passing between the *Triggering Mechanism* and the *Decision-Maker*, which decides when this new event is processed. Indeed, the *Decision-Maker* is expected to have agents that check for all important events as frequently as necessary with messages received from the *Triggering Mechanism.* The software *cooperative control* interruption allows one to embed a reactive behavior in PRIMES. Indeed, the knowledge in *Triggering Mechanism* encodes resource-bounded reasoning components to activate and to react to a given situation. Furthermore, PRIMES reasons, through its *Decison-Maker*, about the resource required for activated components. However, PRIMES provides more guaranteed performance than reactive systems that simply run as fast as they can and can thus coincidently be real-time [4] without guaranteeing any real-time performance.
- **Timeliness:** This property lies in the ability of the system to react so that deadlines are met. Through the scheduling process of its *Decision-Maker* module, PRIMES achieves this property by adapting the approximation level of its resource-bounded reasoning components. The *Timer* is up to execute the most critical agent. Indeed, it gets the first agent in the *Calendar* and executes it when its *begin_time* is reached. The *Decision-Maker* and *Timer* include rudimentary mechanisms of temporal reasoning. Indeed, the *Decision-Maker* reasons about temporal relations between time points. This module must sort agents according to their deadlines and reason about their temporal windows. The *Timer* includes a simple form of temporal reasoning driven by a local clock that allows one to detect reached deadlines and *begin_time* of the first agent in the *Calendar.* Furthermore, the cooperation between the *Decision-Maker* and the *Timer* allows the monitoring of progress of the resource-bounded components to be conducted. We believe that these features represent a significant contribution compared to existing systems such as e.g. CIRCA [22] and PRS [8].
- **Graceful adaptation:** This property lies in the ability of the system to adapt the priority of the agents according to the workload or resource availability. PRIMES, through its *Decision-Maker* combined with the resource-bounded reasoning of components, allows one to adapt the level of approximation of its problem-solving components according to the available resource. The low levels from a component are retracted when a schedule is not found. In this respect, PRIMES offers more flexibility than many existing systems. Indeed, PRIMES is more flexible than CIRCA [22] in the sense that when no schedule is found, PRIMES flexes the details of its reasoning components while CIRCA reasons with another subset of agents of the initial set. Furthermore, PRIMES integrates multiple methods and progressive reasoning to reach more expressiveness to address real-world problems.

5 Conclusion and open issues

In this paper, an architecture embedding resource-bounded reasoning components using both the progressive reasoning and the multiple methods approaches has been presented. In particular, a cooperative control module has been described that allows the system to reason at different levels of detail through hierarchies of reasoning levels and multiple methods. This combination of techniques increases the scope of the system but makes it more complex to manage. To address this last issue, an algorithm that appears as a trade-off between the incremental processing of the *Incremental Scheduler* and the *Design-to-Time* algorithms has been proposed. This algorithm is more suitable for critical time situations than *Design-to-Time*. It is able to guide the behavior of both progressive reasoning and multiple methods components. It also can be interrupted at any time and returns a schedule. PRIMES, the system implementing this architecture, is written in C. Future works concern various directions: (1) Developing a user interface allowing a designer to hand-code its applications and assessing the system in more real-world applications, (2) handling duration uncertainty in cooperative control [21] to improve the monitoring of resource-bounded components, (3) using both scheduling algorithms, *Incremental Scheduling* and *Design-to-Time* and selecting, through incremental negotiation [17] between the *Triggering Mechanism*, the *Decision-Maker* and the *Timer*, the most suitable one with respect to the current state of PRIMES. This negotiation should assess different parameters affecting the work load such as the length of the *Calendar* and the *arrival frequency* of external events. (4) PRIMES must take interactions between agents into account.

References

[1] T. Dean and M. Boddy. An analysis of time-dependent planning. In *AAAI-88*, pages 49–54, 1988.

[2] K. Decker, A. Garvey, M. Humphrey, and V. Lesser. Real-time control architecture for an approximate processing blackboard system. *The Journal of Pattern Recognition and Artificial Intelligence*, 7(2):265–284, 1993.

[3] K. Decker, V. Lesser, and R. Whithair. Extending a blackboard architecture for an approximate processing. *The Journal of Real-Time Systems*, 2(1-2):47–79, 1990.

[4] R. Dodhiawala, N. Sridharan, P. Raulefs, and C. Pickering. Real time AI system: Definition and architecture. In *IJCAI*, pages 256–261, 1989.

[5] E. Durfee. A cooperative approach to planning for real-time control. In *Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 277–283, 1990.

[6] A. Garvey and V. Lesser. Design-to-time real-time scheduling. *IEEE Transactions on systems, Man, and Cybernetics*, 23(6), 1993.

[7] A. Garvey and V. Lesser. Design-to-time for anytime algorithms. In *Workshop Anytime algorithms and deliberative scheduling*, 1995.

[8] M. Georgeff and F. Ingrand. Decision-making in an embedded reasoning system. *IJCAI-89*, pages 972–978, 1989.

[9] B. Hayes-Roth. Architectural foundation for real-time performance in intelligent agents. *Journal of Real-Time Systems*, 2(1), 1990.

[10] A. Howe, D. Hart, and P. Cohen. Addressing real-time constraints in the design of autonomous agents. *Journal of Real-Time Systems*, 2(1/2):81–97, 1990.

[11] D. Hull, W. Feng, and J. Liu. Operating system support for imprecise computation. In *AAAI Fall Symposium on Flexible Computation*, pages 96–99, 1996.

[12] F. F. Ingrand and M. Georgeff. Managing deliberation and reasoning in real-time AI systems. In *Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 284–291, 1990.

[13] P. Lalanda. Conduite de raisonnement dans système à base de blackboard temps réel. *PhD, University of Nancy I, (in French)*, 1992.

[14] A.-I. Mouaddib. Contribution au raisonnement progressif et temps réel dans un univers multi-agents. *PhD, University of Nancy I, (in French)*, 1993.

[15] A.-I. Mouaddib. Progressive goal-directed reasoning for real-time systems. *Engineering intelligent systems*, 3(2):67–77, 1995.

[16] A.-I. Mouaddib. Progressive reasoning in intelligent systems. In *AAAI Fall Symposium on Flexible Computation, Research Summary Report*, pages 183–185, 1996.

[17] A.-I. Mouaddib. Progressive negotiation for time-constrained autonomous agents. In *First ACM International Conference on Autonomous Agents*, pages 8–16, 1997.

[18] A.-I. Mouaddib, F. Charpillet, and J. Haton. Approximation and progressive reasoning. In *AAAI Workshop on Imprecise Computation*, pages 166–170, 1992.

[19] A.-I. Mouaddib, F. Charpillet, J. Haton, and Y. Gong. Real-time specialist society. In *IEEE Conference on Intelligent Control and Instrumentation*, pages 751–754, 1992.

[20] A.-I. Mouaddib and S. Zilberstein. Knowledge-based anytime computation. In *IJCAI*, pages 775–781, 1995.

[21] A.-I. Mouaddib and S. Zilberstein. Handling duration uncertainty in meta-level control for progressive reasoning. In *IJCAI-97*, 1997.

[22] D. Musliner, E. Durfee, and K. Shin. Circa: A cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man and Cybernetics*, 1993.

[23] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.

[24] S. Zilberstein and S. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1-2):181–213, 1996.