Composing and Monitoring Non Deterministic Design-to-time Methods

François Charpillet, Anne Boyer

Composing and Monitoring Non Deterministic Design-to-time Methods

François CHARPILLET and Anne BOYER

CRIN-CNRS et INRIA-Lorraine,
BP 239, 54506 Vandœuvre, France
e-mail: charp@loria.fr,
tel: (33) 83-59-20-81, fax: (33) 83-41-30-79

Abstract

Guaranteed response time is one of the important issues encountered in designing a real-time system. This problem has been studied with a new view by the AI community, which so far has proposed different paradigms. Anytime algorithms, Approximate processing, Design-to-time Scheduling and Progressive Reasoning are the most popular. All of them rely on a trade-off between run-time and quality of results. In the framework of the ESPRIT project n° 5146 and 7805 REAKT (REAl time Knowledge Tool), we have developed such a model called PROGRESS (PROGressive REasoning SyStem). This approach makes it possible to manage AI tasks with hard and soft deadlines, provided that multiple methods are available for the tasks the system has to solve. Thus, PROGRESS is closed to design-to-time real-time scheduling but it extends this approach for harder real-time constraints such that the system has the ability to react, meet hard and soft deadlines, stay alert to incoming events and reset task priorities according to changes in workload or resource availability. For this purpose, we have defined a new task model such that a task is not *a priori* defined at the time of its activation but step by step in the course of its execution. It is conceived as a process that gradually integrates changes and developments in the situation and in availability of resources. When unforeseen tasks have to be included in the schedule because of the occurrence of an unexpected event, the resulting overhead is dynamically accounted for an adaptation of on-going tasks. An on-going task can be reactively adapted as the subtasks composing the task are interruptible (as anytime algorithm can do). A task being constructed dynamically by composing design-to-time methods chosen in a library, we have developed a new deliberative scheduling algorithm which allocates to each component of the task the computation time which maximizes the output quality of the task.

Key-Words

Real-time AI, Real-time Design-to-time scheduling, Anytime algorithms.

Introduction

Motivations and Scope of the paper

The functions to be automated in applications such as process control, pilot's associate, medical monitoring, robotics are more and more complex. Introducing Artificial Intelligence techniques into conventional systems appears as a promising approach to deal with this complexity. However, most of these applications require a real-time approach. This requirement raises great difficulties that have considerably reduced the scope of knowledge-based systems. While real-time systems require predictable and continuous operations, AI techniques rely on time-consuming algorithms, with unpredictable or highly variable performances. Even though task execution speed is of course an essential feature, it is not the only parameter to consider. As mentioned in [Dodhiawala et al., 1989], the ability of the system to react and meet deadlines (timeliness), its ability to stay alert to incoming events (responsiveness) and to reset task priorities according to changes in workload or resource availability (graceful adaptation) are imperative features in a knowledge processing approach to address real-time applications.

The European Esprit Projects REAKT (EP 5146 and 7805) comes within this scope. The primary objective of those projects was to develop a set of tools and the associated methodology to apply knowledge-based systems in real-time domains. The projects goals (Mensch et al., 1994) were to produce definitions, specifications and prototypes of various techniques, to be eventually integrated into a toolkit to develop, deploy and maintain efficiently knowledge-based modules which can be embedded into real-time applications. Research areas of particular interest included deliberative real-time artificial intelligence which were identified as a key element to provide guaranteed response time, as well as temporal reasoning and coherence management mechanisms. The methodological work was mainly focused on the implications of real-time issues on the modelling and development of knowledge-based systems.

The purpose of this paper is to present the PROGRESS model on which relies the management of hard and soft deadlines in the REAKT architecture. The management of

deadlines is a crucial issue that has been widely addressed by the real-time community. The main results of this research comes from scheduling techniques. But these approaches are well-suited to domains where the needs and availability of resources are predictable. But it is clear that AI methods have high variance in their response time. The worst-case computational time may be several orders of magnitude than the average time. Then, using worst-case values in scheduling could result in severe underutilization of computation resources. An other drawback of conventional scheduling is the poor skill in dealing with saturated situations assuring that important tasks are executed.

To avoid these problems the AI community has proposed different paradigms. Anytime algorithms (Boddy and Dean,1988) Russell (Zilberstein and Russel, 1991-1992), Flexible Computation (Horvitz, 1987), design-to-time(Garvey and Lesser, 1992-93-94) and progressive reasoning (Mouaddib et al 94-95) are the most popular. All of them rely on a trade-off between runtime and result quality. This enables a system to operate under bounded resources while using unbounded AI methods. In the field of real time scheduling, some researchers are investigating a similar approach: the *imprecise computation* as described in (Liu *et al.*, 1991) (Chung *et al.*,1990).

We propose such a model called PROGRESS that makes it possible to manage AI tasks with hard and soft deadlines, provided that multiple methods are available for the tasks which the system has to solve. Thus, PROGRESS is close to design-to-time real-time scheduling but it extends this approach for harder real-time constraints. In PROGRESS a task is not *a priori* defined at the time of its activation but step by step in the course of its execution. It is conceived as a process which gradually integrates changes and developments in the situation and in the availability of resources. When unforeseen tasks have to be included in the schedule because of the occurrence of an unexpected event, the resulting overhead is dynamically accounted. An on-going task can be reactively adapted as the subtasks composing the task are interruptible as anytime algorithm can do.

REAKT ARCHITECTURE

A Real Time Blackboard Architecture

The REAKT architecture is based on the blackboard paradigm. A blackboard based architecture consists of a number of agents which communicate with each other through a shared database called blackboard. Such an architecture provides a framework to co-ordinate the work of a set of agents which are instanciated from knowledge sources (KS). A controller mediates the execution of enabled agents with respect to externally and internally events in order to notify important modifications in the system. Classical blackboard architectures even if they offer a good starting point need to be extended to deal with real-time applications. Therefore, the REAKT architecture introduces the following concepts:

- Multiprocessing: The components of REAKT are designed as independent processes. This implies the possibility of pre-empting active tasks to run more important ones in terms of urgency or priority.
- Progressive deepening reasoning: This is a control strategy which monitors the depth (degree of refinement) of tasks in order to meet the deadlines.
- Advanced temporal reasoning capacities: it contributes mainly to the ability of the system in anticipating the future.

Architecture Description

A REAKT application is made up of an agent community implemented as independent processes which communicates through a blackboard structure managed by a Knowledge Data Manager (KDM). The architecture has been strongly influenced by the deadline management issue. To that end, a two layer architecture has been defined (see figure 2) enabling to design hard real-time systems made up of periodic and sporadic tasks dealing with problems that require guaranteed response time, while an expert server uses the remaining time to reason about high-level problems that requires powerful but unpredictable reasoning techniques.

More precisely, in the REAKT task model, each task τ is decomposed into three parts: *mandatory*, *optional* and *action* (cf. Figure 1) [Audsley et al. 91]. The mandatory part, activated at Rm_τ, is in charge of providing a first-level solution with a guaranteed response time; the optional part then tries to improve this result, using more complex reasoning mechanisms; the action part is activated before the task deadline D_τ, possibly pre-empting the optional part, and uses the best available solution to execute the necessary actions.

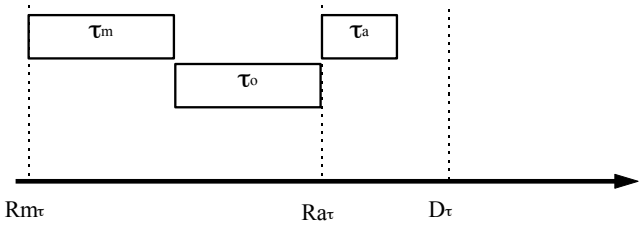


Figure1 - Decomposition of a task into mandatory, optional and action parts .

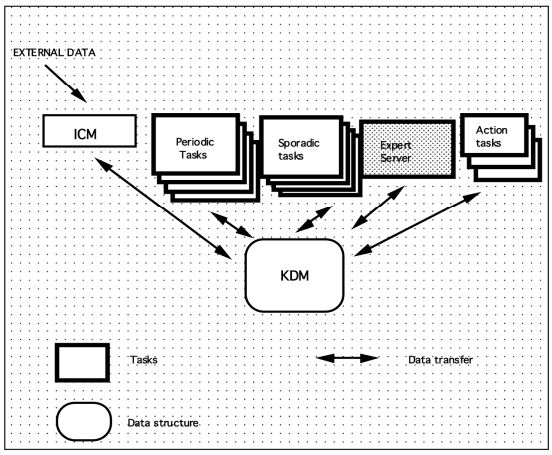


Figure -2- Global architecture of REAKT.

Important characteristics of a task include:

- The type of the task: either *periodic*, when the task activation pattern repeats itself, or *sporadic*, when the task is triggered once in response to a particular event.
- The *worst-case execution time* of both the mandatory and action parts of the task.
- The temporal characteristics of the task: *period* and *deadline* for a periodic task, *minimum inter-arrival time* and *deadline* for a sporadic task.
- The name of the *intention structure* (Lalanda *et al*., 1992) to be used to execute the optional part.

The REAKT Kernel relies on a two-layer control strategy, with a first level in charge of scheduling the real-time mandatory and action parts of each task, and a second level responsible for optimising the solution quality by running optional parts (expert system) in the remaining time.

The role of the first level real-time control layer is twofold: it must guarantee the response time of the mandatory and action parts of all tasks in an application, but should also maximise the amount of time available for optional second-level activities, in charge of improving the solution quality. For a given task, the basic principle is to first execute the mandatory part, then the optional part, and delay as much as possible the beginning of the action part, while of course executing it before the task deadline. The scheduling algorithm used in the real-time control layer is based on the *slack time server* algorithm [Lehoczky and Ramos-Thuel 93], as its characteristics are close to the REAKT requirements:

- The original goal of slack time server algorithm is to maximise the CPU allocation to soft aperiodic tasks, which is more or less equivalent to maximise the amount of time available for optional processing.
- The basic principle of the algorithm is to delay as much as possible the execution of periodic tasks while guaranteeing their deadlines. This is exactly what needs to be achieved in REAKT for action tasks.
- The type of the task: either *periodic* (i.e. the task The algorithm is optimal for the deadline-monotonic priority assignment, in the sense that no other algorithms can save more time for optional processing.
- The slack-time server algorithm is able to recover CPU time unused by stochastic real-time tasks, i.e. tasks which execute faster than their worst-case execution time.

The slack time server algorithm has been adapted to the REAKT task. The details of the slack time server algorithm we have developed are available in (Mensch and Charpillet, 1996).
The second level of the architecture performs the optional tasks which are dynamically provided by the expert server. It is made up with tasks which are either refinement of the periodic or sporadic tasks or by independent tasks with soft deadlines. This paper is focused on the refinement level we have defined in REAKT.
Whatever the components are (i.e. periodic tasks, sporadic tasks, expert server), they communicate through a common data area, or blackboard, managed by the knowledge data manager (KDM). The blackboard feeds and is fed by:

- the ICM (Intelligent Communications Manager) which receives data from external sources;
- periodic or sporadic tasks ;
- the expert server ;
- action tasks which provide the external world with the solution.

The important point to notice is that the slice of CPU time allocated to the expert server will grow dynamically when the activation conditions (i.e., event arrival) of sporadic tasks are not met.

Second Layer Scheduling issues

Given the above description of the architecture and of the knowledge representation, we can abstract the second layer task model, called PROGRESS.
Progress relies on the following assumptions :

- several methods, which requires various computation times, are available to solve a given problem,
- for each method, it is possible to estimate the computation time even inaccurately (The computation time can be viewed for tasks with high variance CPU time as a control feature which reflect the utility of achieving the method),
- It is assumed that the longer the time of task execution, the better the result,
- tasks are supposed to be pre-emptable,
- the response to a given event is computed by a sequence of methods, some of them being time-constrained,

- the response to a given event can be computed in several ways which depends on the context. All these ways are known, but the one which is going to be activated is determined step by step during the execution,
- even if a task is unexpectedly interrupted, a result is available.

The availability of several methods with various response times is essential to optimize the behaviour of the system and adapt it to the stress imposed by the situation. Our workload model can be viewed as a set of pre-emptable tasks called intentions, each being organized as a tree of steps, each step being broken down into a set of agents implementing the set of available methods to solve the same problem. They are sorted by increasing computation time. The fastest one is called « the first level agent » and others are called « optional agents ». Thus, an intention represents a set of possibilities to carry out the corresponding task _ the successful execution of an intention consists in determining step by step a path from the root of the tree to one of its leaves.

The problem of scheduling such intentions is complex and belongs to the problem of scheduling non deterministic dependent tasks. In order to manage deadlines, different policies could be adopted. The first one, inspired from design-to-time approach could determine at intention creation time the refinement level of each step. Even if this policy is attractive, it is not suitable. Computing *a priori* the refinement level of a step requires a good estimation of the execution time of agents because over-evaluation would lead to CPU time waste. Indeed, if an active agent has to be sped up to meet deadlines (due to unforeseen additional computation workload), it has to be cancelled and replaced by an agent with lower-level quality. In this case, the time spent for executing the deleted agent is lost.

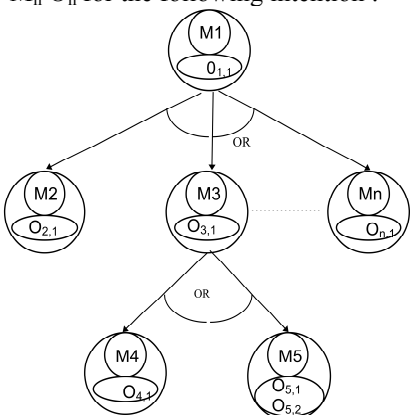
The solution we adopt is inspired by the process used to transform contract algorithms into interruptible anytime algorithms. This consists in considering a step as a process obtained by chaining available agents together, from the first level agent to the one with the best level of refinement. So, at first sight, the execution of a step behaves like an interruptible anytime algorithm. Unlike interruptible anytime algorithms based on contract algorithms, we do not control the time required for producing the first result with the lower quality. All we can say is that we encourage the developer of an application to design the first-level agent to be as fastest and predictable as possible. Indeed, the first-level agent must be executed to completion if we want to be able to get a minimal result for the step. If not possible, the completion of the corresponding intention is not achieved and the current time spent to execute the intention is lost. Thus the main objective of scheduling is to ensure that the first-level agents required to execute the intention can be definitively completed. If no schedule exists, the intention is either discarded (its exception is fired) or kept while computation time is recovered by cancelling a less important intention. Based on this model, the purpose of the scheduling algorithm is to determine the existence of a feasible schedules that meet the timing constraints of all steps (by executing at least the first-level agent), and find one that minimizes the number of discarded optional agents. Such a schedule is said to satisfy the 0/1 constraint because optional agents are either completed or discarded. Unfortunately, the general problem of finding such an optimal schedule is NP-complete. Thus we have paid attention to finding a heuristic which finds an approximate schedule in polynomial time. Even, if not optimal the schedules we obtain are of good value mainly because our heuristic correct the drawback of using worst-case computation times.

Controlling Progressive Reasoning

Objective and definitions

Let {I1, …,IN} be a set of N active intentions. An intention is known by a weight number wi, which measures the importance of the intention, a tree of steps, a date of creation, the current step.

Let {Si1, ..., SiN_i}be the set of steps defining the intention Ii. Each step Sij is characterised by a deadline dij, a weight number wij measuring the importance of the step (if not specified, wij = wi), a set of successor nodes in {Si1, ..., SiN_i}given by the function called succ. A step Sij is broken down into subtasks: the first-level agent Mij and the refinement agents {Oij1, …, $Oijk_{ij}$}. The computation times of the first-level agent and refinement agents are known and are referred as mij for the computation time of the first-level agent and {Oij1, …, $Oijk_{ij}$}for the computation times of the refinement agents. So, considering an intention as a tree of steps, our computational model takes a disjunctive form: $M_1\ O_{1,1}$ (($M_2\ O_{2,1}$ / $M_3\ O_{3,1}$ ($M_4\ O_{4,1}$ / $M_5\ O_{5,1}\ O_{5,2}$)……/ $M_n\ O_n$ for the following intention :



Because an intention is a tree of steps, each intention can be executed in different ways, depending on the chosen path in the tree. A path is a sequence of steps joining the root to a leaf. There are as many possible paths as leaves.

Let us select one path per intention. Let S be the set of all the steps of the selected paths. A schedule is the sorted sequence of the elements of S with respect to the deadlines. Guaranteeing that all active intentions could be executed, whatever the chosen path in each intention is, consists in verifying that there exits a feasible schedule of all steps belonging to the chosen paths. The set of all schedules accounting for all the possible paths in all the intentions is called the set of **possible schedules**. If such a schedule is admissible w.r.t. deadlines, we say the schedule is **admissible**.

Deadline and release time modifications

Let us now consider the first problem to solve: the schedulability of a set of intentions whatever the way each intention will be executed. The problem is to determine the existence of an admissible sequence of steps in these intentions. An admissible sequence is a sequence that verifies the precedence constraints in each intention and the deadlines of each step firing at least the first level agent. Referring to the results about dependant tasks scheduling in (Chetto et al., 90), we can extend the process of deadlines, ready times modifications and the schedulability test.

The tree defining an intention specifies a partial order < between a step and its successors. A step must be completed before one of its successor begins. Then, the given deadline dij of a step Sij cannot be later than dik of any step Sik such that Sik<Sij. Because the time mij required to execute Sij is known, we can state that the deadline of S_{ij} cannot be later than the deadline of Sik minus mik. Then, the modified deadline of the step Sij is: $d_{ij}^* = \text{Min}\ \{d_{ij}, \text{Min}\ \{dik^* - mik / Sik > Sij\}\}$.

It is important to point out that the modified deadlines are computed in the worst case as we take into account all possible successors of a step (although only one will be executed). The ready time rij of a step Sij is modified in the same way.

Theorem

Let S={Si1, ..., SiN_i} be a set of pre-emptable tasks and < be a partial order over S.

Let S*={S*i1, ..., S^*iN_i} be a set of independent tasks such that:

$m^*ij = mij$,

$rij^* = \text{Max}\ \{rij, \text{Max}\{rik^* + mik / Sij > Sik\}\}$,

$d_{ij}^* = \text{Min}\ \{d_{ij}, \text{Min}\ \{dik^* - mik / Sik > Sij\}\}$.

S is schedulable if and only if :

$\forall j = 1, \ldots, N_i, \forall i = 1, \ldots, N_i$ such that $ri^* \le rj^*$, $di^* \le dj^*$

$$\sum_{rk^* \ge ri^*,\ dk^* \le dj^*} mk \le dj^* - ri^*$$

for all steps k=(a,b), j=(c,d), i=(e, f) where a, c, e represent an intention and b, d, f a step in the corresponding intentions; i, j, k belong to a same possible schedule.

The complexity of the schedulability test depends highly on the complexity of the algorithm enumerating all necessary inequations :

$$\sum_{rk^* \ge ri^*,\ dk^* \le dj^*} mk \le dj^* - ri^*$$

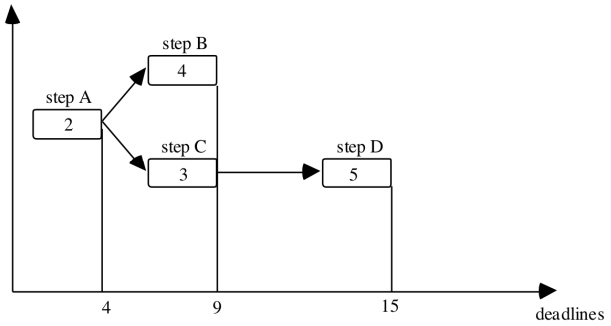
in which the sum stands for the total duration of the set of tasks that can be executed between a release time ri and a deadline dj and belonging to the same possible schedule. Enumerating all possible schedules obviously has a high complexity. Clearly, if the number of intentions is N and L the number of leafs (paths) per intention, the enumeration requires L^{N-1} iterations for each given dij, rkl. Fortunately, the enumeration of all possible schedules is not necessary because only the worst case duration is of interest. Let us notice that a possible schedule is made of partial paths, each one belonging to the intentions with steps having a deadline prior to the one considered in the equations. Among the different partial paths of interest for the equation only the path with the greatest duration has to be taken into account for a given intention. The paths of interest are those whose begin time is greater than the release rij and whose step has a deadline which is less than the deadline dkl, where rij and dkl are the deadline and release time considered in the equation.

Monitoring Progressive reasoning algorithm

The schedulability test given in the previous paragraph neither calculates a schedule nor optimizes the level of refinement allocated to steps. It only proves, that for a given set of time allocated to each steps in S*, there exists a possible schedule. So the trade-off between quality versus response time remains. We propose in this paragraph an algorithm addressing this problem. It is derived from the schedulability test and relies on a local scheduling strategy, that favours the current step of each intention. This algorithm is not optimal as we cannot afford to implement an NP_complete algorithm. *For the sake of simplicity we present in this paper the case with no release-time.*

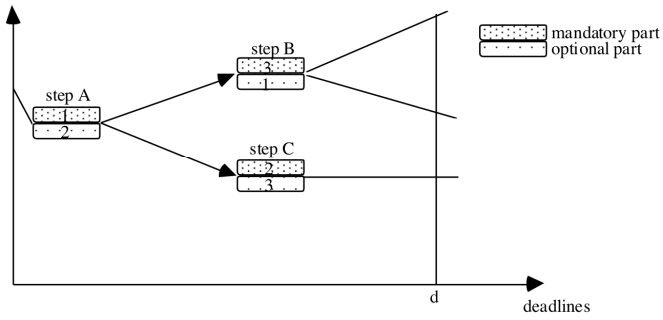
For all on-going intentions, the algorithm computes the maximum amount of time that can be allocated to each current step, such that all intentions can be completed with respect to the deadlines in the worst case, each step being executed at the lowest refinement level. For this purpose each deadline has to be considered (we can have several deadlines per intention to check). We have to check the correctness of the sum of the worst case computation time which is needed by each ongoing intention from the current step until the considered deadline. We define the worst-case for a given deadline d, for a given intention i currently executing the step s, as the maximum possible execution time to reach a step s', s' having a deadline less or equal to

d, and being on a path containing s. The following figure examplifies this definition :



From step	to deadline	worst-case path is	worst-case duration is
A	4 9 15	A AB ACD	2 6 10
B	9	B	4
C	9 15	C CD	3 8
D	15	D	5

As we introduced progressive deepening, it makes sense to enlarge the notion of worst-case. For a given step, and for a given deadline, there are several interleaved worst-cases, depending on the number of refinement levels we planned to execute for each step. The following figure describes the different worst-cases we have to consider when each step has a first level part and a single refinement part :



Considering only the step A, the worst-cases associated with the deadline d are :

depth A	depth B	depth C	worst-case path	worst-cases durations
2	2	2	AC	8
2	2	1	AB	7
2	1	2	AC	8
2	1	1	AB	6
1	2	2	AC	6
1	2	1	AB	5
1	1	2	AC	6
1	1	1	AB	4

Most cases are of little interest for the scheduling problem. Knowing the level of refinement of future steps is useless, because we do not know the future (i.e. the worst path, considered here, has a low probability to be the one that will be executed). For this reason, the schedulability test consisting of the two following phases is satisfactory:

I. a schedulability test only considering the first-level parts.
   a) if it fails, among the intentions involved in the scheduling test, the one with the lowest importance is removed
   b) if it succeeds, go to the second phase.

II. a schedulability test considering both first-level and refinement parts of ongoing steps and first level only for future steps.

As a result, in our example, the worst-case table for the step A is now reduced to:

deadline	path	duration
3	A2	2
3	A1	1
9	A2B1	5
9	A1B1	4
12	A2C1	4
12	A1C1	3

Schedulability test algorithm :

Let I be a list containing all the intentions to be scheduled.
Let $Deadl_S$ be a set containing the deadlines of the steps following s in the plan s belongs to.
Let s_i be the current step of the intention i.
Let D be a null-initialized list of deadlines.
Let current be the current time.
Let cumulated be the cumulated time of all the worst-cases for a given deadline.

Let WC (s,d,n) be a function returning the worst-case computation time needed by an intention to execute the corresponding path starting from s, and finishing by a step s' whose deadline is d', d'≤d ; the step s executing n levels of refinement.

```
for each i in I
   D = D U Deadl_si
endfor
for each d in D
   cumulated = 0
   for each i in I
      cumulated = cumulated + WC (s_i,d,1)
   endfor
   if d - current < cumulated
   then    test has failed
```

endif
endfor

For the first phase of the schedule, the worst-case we want is the one concerning only the first-level parts of each steps. If the test fails, the intention with the lowest importance is removed and the test is redone until it succeeds otherwise there's no more intention.
The second phase of the schedule is dedicated to compute for all intentions, the maximum time that can be allocated to each current step. The schedulability test is applied with :
cumulated = cumulated +WC(si, d, nb_levels(si))
nb_levels(si) is initially set to the maximum number of refinement level for each si. When the schedulability test fails the less important intention is chosen to be approximated (by decreasing the number of refinement level for si). This is the purpose of the following algorithm.

Level Deletion Algorithm

For a given intention i , a given deadline d, and a given cumulated time.
Let nb_levels (s) be a function returning the number of levels planned for the step s.
Let decrease_nb_levels (s) be a function decreasing as a side effect the number of levels planned for step s.
while (cumulated > d - current) **and** (nb_levels (s_i)>1)
  cumulated = cumulated - WC (s_i, d, nb_levels (s_i))
  decrease_nb_levels (s_i)
  cumulated = cumulated + WC (s_i,d,nb_levels (s_i))
endwhile

After the two tests have succeed, the current step of each intention contains the number of levels planned for the execution to come. Then, the steps are scheduled using the Earliest Deadline First algorithm.

Discussion

This heuristic takes advantage of the over-estimation of estimated computation time by favouring current steps while preserving the remaining steps. A first point advocates this approach: deliberating on cases which may not occur is useless. As we can notice that, the further a step in an intention is, the less its probability to be fired is : an intention being a tree, the probability to reach a step is function of the number of its alternatives, it is suitable to put at a disadvantage further steps. A second point is related to the chaining of steps. The quality of results produced by intentions dependents on the quality of all the steps fired during its execution. If we point out that a step uses results produced by previous steps, it is clear that the quality of a step is related to its entry, i.e. to the quality of results produced by previous steps. Then it is not suitable to refine steps whose previous ones are not. However, let us notice that in some specific cases (we are in the worst-case and there is no remaining time), this strategy leads to a rather inadequate solution quality, the first steps being totally achieved, the last ones being completely approximated.

Conclusion

An important aspect of the REAKT project is to deal with an inherent characteristic of real-time systems, i.e. the ability to account for timing constraints. We have developed a new model termed progressive deepening. It is based on a trade-off between the quality versus computation time. The model we propose is a tactical approach and belongs to multiple methods. It is closed to the task model TÆMS but extended to generate tasks termed intentions with better real-time behaviour. A task is not *a priori* defined at the time of its activation but step by step in the course of its execution. It is designed as a process which gradually integrates changes and developments in the situation and in the availability of resources.
To make trade-offs of solution quality versus time, a scheduler has been defined. The problem of intentions scheduling is complex and belongs to the problem of scheduling dependent tasks. We have extended the schedulability test procedure proposed by Chetto (Chetto *et al.*, 1990). Relying on this schedulability test, we have developed a progressive deepening mechanism which is inspired by the process used to transform contract algorithms into interruptible anytime algorithms. This consists in considering a step as a process obtained by stringing available agents together from the first level agent to the one with the best level of refinement. So, at first sight, the execution of a step has the behaviour of an interruptible anytime algorithm. The main objective of the scheduling is to ensure that the first-level agents required to execute the intention can be all completed. If no schedule exists the intention is either discarded and its exception is fired or computation time is recovered by cancelling a less important intention. Such a schedule is said to satisfy the 0/1 constraint. Unfortunately, the problem of finding such an optimal schedule is NP-complete in general. Thus we paid attention to finding a heuristic computes approximated schedule in polynomial time. Even if they are not optimal, the schedules we obtain are of valuable value mainly because our heuristic fixes the drawback of using worst-case computation times.
The REAKT environment provides both tools and complete methodology consisting of a set guidelines and support tools to assist the REAKT user throughout the application development life cycle.
The MORSAF demonstrator has been successful. The industrial relevance of this demonstrator comes from its ability to manage potentially dangerous and harmful

situations, thus ensuring better performances and a higher security degree of the plant. The integration of a knowledge-based system seems to be a promising technique towards such a goal. The techniques developed for the demonstrator could be applied to many other environments. We are currently investigating the steel industry and Robotics . An application for performing diagnosis in a steel plants will be developed and will allow to assess the benefits of the REAKT technology in this application domain. An additional objective of this application will be to extend the REAKT technology with signal processing and behavioral diagnosis capabilities.

ACKNOWLEDGEMENTS

The authors acknowledge the REAKT partners for their comments and their participation in the global architecture described in this paper. The partners involved in the project REAKT are Thomson-CSF (France),Grupo de Mecánica del Vuelo, S.A. (Spain), Marconi (United Kingdom), Etnoteam (Italy), Computas eXpert Systems (Norway), Syseca (France), Universidad Politécnica de València (Spain) and CRIN-INRIA (France). This work is partially sponsored by the Commission of the European Communities under the ESPRIT II programme (Project Ref. N. 5146 in its first phase and N.

References

Audsley, N.C, Burns, A. and Welling, A.J. « Incorporating unbounded algorithms into predictable real-time systems », *Technical Report*, Universisty of York, UK, 1991.

Boddy and Dean. "An Analysis of Time Dependent Planning", Proc. 7th AAAI, pp. 49-54, 1988.

Chetto, Silly and Bouchentouf. "Dynamic scheduling of Real-Time tasks under precedence constraints." The International Journal of Time-Critical Computing Systems., Vol. 2, No 3, pp.181-4, 1990.

Chung and Liu . "Scheduling Periodic Jobs that allow Imprecise Results", IEEE Transactions on Computers, Vol. 39, No. 9, 1990.

Dodhiawala, Sridharan, Raulefs and Pickering. "Real-Time AI Systems: A Definition and An Architecture", Proc. 11th IJCAI, Detroit, 1989.

Garvey and Lesser."Design-to-time Scheduling", IEEE Transactions on Systems, Man and Cybernetics, Special Issue on Planning, Scheduling and Control, Vol. 23, N°6, 1992.

Garvey and Lesser. "A Survey of Research in Deliberative Real-Time Artificial Intelligence", Real-Time Systems, Vol. 6, N°2, 1993.

Garver, Decker and Lesser. "A Negociation Based Interface Between a Real-Time Scheduler and a Decision-Maker", UMASS CS Technical Report 94-08, 1994.

E.J. Horvitz, «Reasoning about Beliefs and Actions under Computational Resources Constraints». *Proc. 1987 Workshop on Uncertainty in Artificial Intelligence*, Seattle, Washington, 1987.

P. Lalanda, F. Charpillet and J. -P. Haton. A Real Time Blackboard Based Architecture. *Proceedings 10th European Conference on Artificial Intelligence*, Vienna (Austria), août 1992.

Lehoczky J. and Ramos-Thuel, S. « An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems », *IEEE Symposium on Real-Time Systems,* 1993.

Liu, Lin, Shih and Yu. "Algorithms for Scheduling Imprecise Computations". IEEE Computer, pp 58-68, 1991.

A. Mensch and F. Charpillet. Scheduling in the REAKT kernel: combining predictable and unbounded computations for maximizing solution quality in real-time knowledge based systems, To be published in proc. of RTS&ES'96, Paris, 10-12 Janvier 1996.

Mensch, Kersual, Crespo, Charpillet and Pessi. “REAKT: real-time architecture for time-critical knowledge-based systems”. Intelligent Systems Engineering Journal, Autumn 94, pp 153-167.

Mouaddib, Charpillet, Haton. "GREAT: a Model of Progressive Reasoning for Real-Time Systems", in IEEE International Conference on Tools for AI, 1994.

Mouaddib, Zilberstein. "Knowledge Based anytime computation" in IJCAI'95, Montreal.

Russell and Zilberstein. "Composing Real Time systems" Proc. 12th IJCAI, Sydney, 1991.

Zilberstein and Russell. "Efficient Resource-Bounded Reasoning in AT-RALPH, Proc. first AIPS, pp260, 266, 1992.