Reactive/Anytime Agents
—
Towards Intelligent Agents with Real-Time Performance

M. Adelantado, S. De Givry

Onera-Cert/Deri
Department of Computer Science
2, Avenue Edouard Belin
31055 Toulouse Cedex, France
e-mail : Martin.Adelantado@cert.fr
fax : (33) 62–25–25–93

Topics

Reactive agents, deliberative agents, rational agents, real-time decision-making, anytime computing, reactive programming, meta-reasoning

Abstract

To face the requirements of complex and time-critical applications, several systems for reasoning about time and in time, have been proposed by the AI community: reactive systems provide reactivity to their environment at the cost of short-sighted decision making, and conversely, deliberative systems involve more sophisticated decision making at the cost of response times. Recently, several AI researchers have proposed the concept of anytime algorithms as a means to meet timing constraints in time-critical applications. An anytime algorithm is an iterative refinement algorithm that can be interrupted and asked to provide an answer at any time. A major requirement of real-time embedded systems that has not been directly addressed by the AI community, is predictability because random timing failures may lead to catastrophic situations. Several reactive languages have been developed to face such real-time requirements for pure reactive agents. The objective of this paper is twofold: first of all, we describe a unified view of anytime computing as a framework for designing deliberative time-constrained agents. Secondly, we show how reactive programming and anytime computing gives the opportunity to associate reactive and deliberative agents to achieve flexibility and responsiveness. More generally, our work is an attempt to motivate a closer relationship between the AI community and the real-time programming community. This work is supported by the French Ministry of Defense (DRET).

1. Introduction

Historically, the field of artificial intelligence has not been directly concerned with real-time performance. Consequently, few systems are capable of reasoning in time and about time [Laf 88]. Currently, a great deal of research effort is spent to design timely intelligent agents, and two major approaches are proposed [Hay 94]. The first approach requires some limitations in complexity of AI techniques providing bounded response times. This approach, called **embedded AI**, makes no distinction between AI and non-AI task, and supports conventional scheduling theory for guaranteeing timing constraints. In order to use the full range of AI techniques, a second approach has been suggested. This approach, called **extended AI** consists in adapting conventional AI techniques with real-time properties such as timeliness. The main advantage of extended AI lies in the fact that sophisticated reasoning techniques achieving flexibility are not abandoned. Recently, a new paradigm has been suggested to design intelligent agents with real-time performance. This paradigm is based on **anytime** algorithms which are able to produce results that improve monotonically over computation time [Dea 88]. This computing approach falls in the last category, and provides a great flexibility in the definition of an intelligent agent.

In the meantime, the real-time programming community spend a great deal of research effort to develop methodologies and tools capable of enhancing some major properties of embedded real-time systems, such as predictability, responsiveness and dependability [Ger 94], [Gau 94]. In particular, **predictability**, defined by A. Bestavros as "the ability to foretell that an implementation will not violate a set of specified reliability and timeliness requirements", is a highly desirable property of responsive embedded systems [Bes 93].

In this paper we motivate a closer relationship between the field of AI and the domains of interest of the real-time programming community. More precisely, we believe that the **reactive** programming approach of real-time systems can be associated with the emergent **anytime** model of computation, to achieve intelligent agents with real-time performances such as responsiveness and flexibility. We define a **responsive** agent as one which reacts to external inputs in a timely, dependable and predictable manner. This paper is structured as follows: the next section outlines the major approaches proposed by the AI community for designing intelligent agents offering real-time performances. Section 3 gives a unified point of view of anytime computing, through a survey of some relevant works. In section 4, we discuss the main features of reactive programming, and in section 5, we examine how reactive programming and anytime computing can cooperate. In section 6, we summarize our conclusions and identify future research directions.

2. Extending intelligent agents for achieving real-time performances

The challenge with real-time intelligent agents, is that the worst-case execution times of the tasks are often unknown or are much larger than their average-case execution times [Str 94]. Therefore, conventional scheduling models which rely on allocating the worst-case execution time for all tasks, are of little interest. Historically, there are at least two different approaches for designing time-constrained intelligent agents, which might be called the **reactive** approach, and the **deliberative** approach [Dea 91], [Bod 94], [Gar 93a]. The reactive community adopts the approach of using the current situation to find the best action according to a set of pre-compiled reactions. This approach is an attempt to cope with time constraints by limiting run-time deliberation and by making the decision by design. Reactive Planning [Agr 87], [Bro 91], and Universal Planning [Sho 87] are well-known paradigms for designing intelligent agents operating in time-critical environments. Although reactive agents provide **timeliness** and **flexibility** through interactions with their environment, the growth of the compiled situations table makes that this approach do not seem well-suited to face the requirements of applications of interesting size. Furthermore, reactive systems do not support explicit treatment of timing constraints.

In response to such shortcomings, the concept of **deliberative** agent has been suggested to provide flexible and adaptive intelligent agents embedded in real-time environments. Deliberative agents are systems that make decisions at run-time through an explicit control of low-level reasoning tasks. Such a meta-level control of computation, generally referred to as **meta-reasoning**, assumes that an agent has limited computational capabilities and is capable of reasoning about the cost of deliberation and the benefits of action. Recently, Russell and Wefald developed a normative decision-theoretic approach of meta-reasoning for control of inference, supporting a view of artificial intelligence as a constrained optimization problem [Rus 89], [Rus 91a]. Their model of meta-reasoning is based on the optimization of an **utility** function defined over the states of the world and including the real-time properties of the environment and the features of the situation in which a decision is taken. The diagram in figure 1, taken from [Rus 91a], illustrates the difference between ideal, optimizing and optimal action selection.

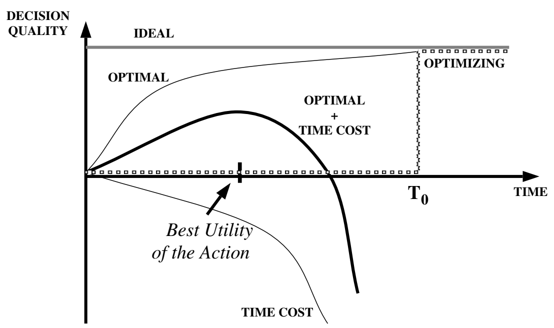

Figure 1

An ideal agent selects **instantaneously** the best utility action. Conversely, the best utility decision of an optimizing agent is only obtained when the full decision selection mechanisms have completed, that is to say at time T_0. The optimal line shows the value of the decision, so called intrinsic utility by Russell and Wefald, assuming that the corresponding action is taken immediately. Finally, the time cost line illustrates the effect of the environment in lowering the utility of decisions as time progresses. The true utility of a decision, that is to say of taking an action at a particular time, is then a function of the value of the decision and the time cost.

Russell and Wefald suggest that under certain assumptions, the intrinsic utility of an action can be separated from considerations of time pressure [Rus 91a]. A real-time agent supporting the meta-reasoning model of Russell and Wefald, can be then view as a utility driven agent maximizing the true utility of taking an action at a particular time. In that case, the time cost function has to take into account the difference between **soft** and **hard** deadlines. More precisely, the time cost function depicted by the diagram in figure 1, corresponds to a soft deadline.

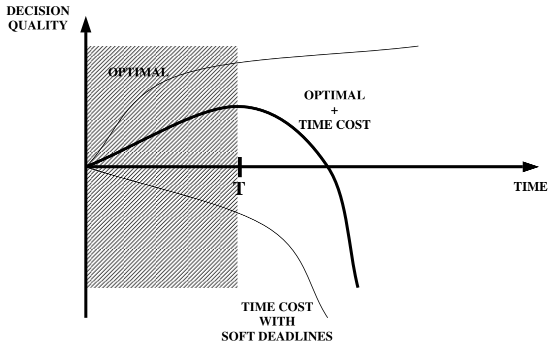

Figure 2

When the agent has to face unpredictable **hard** deadlines, the best utility decision depends on the instant when the deadline occurs. In figure 2, we assume that the best true utility $\mathbf{U_{best}}$ of the decision from the agent point of view, is obtained at time **T** when there are only soft deadlines. Suppose now that a hard deadline may occur at time $\mathbf{T_{hard}}$. If $\mathbf{T_{hard}}$ is greater than **T**, the true utility of the decision remains $\mathbf{T_{best}}$, and the action has not to be delayed after time **T**. Otherwise, if $\mathbf{T_{hard}}$ is less than **T**, the agent has to act before $\mathbf{T_{hard}}$, obtaining then a true utility $\mathbf{U_{hard}}$, such as $\mathbf{U_{hard} <= U_{best}}$. Such a model of deliberative agent, referred to as **time-constrained deliberative** agent, needs being enable to capture the best intrinsic utility decision in order to meet timing constraints.

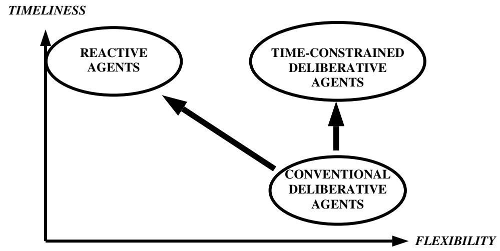

Figure 3

Figure 3 highlights, in an oversimplified way, the expected properties of agents corresponding to the different approaches proposed for achieving real-time performance in extending intelligent agents. Conventional deliberative agents are intelligent agents with unlimited computational and temporal resources. Reactive agents achieve responsiveness and timeliness at the cost of flexibility. Finally, the model of time-constrained deliberative agent is an attempt to achieve timeleness, without abandoning flexibility through complex representations of the world and run–time deliberation strategies. Although the meta-reasoning model of Russell and Wefald seems well-suited to support time-constrained deliberative agents, at least two fundamental questions arise. The first one is what kind of low-level reasoning tasks are needed. The second one concerns specially the real-time programming community, and deals with how expressing the meta-level control of computation, and how ensuring a level of predictability to achieve a **responsive** agent. We believe that the emergent **anytime** model of computation provides a framework to answer the first question, and that **reactive** programming languages and their associated formal tools, give the opportunity of achieving predictable meta-reasoning. In the following sections we motivate that claim.

3. Anytime computing: a unified approach

3.1. Anytime algorithms: related works

The term **"anytime"** was coined by Th. Dean and M. Boddy, to qualify algorithms beeing able to produce an approximate result at any time. Such algorithms are expected to return results whose **quality** improves gradually as computation time increases [Dea 88], [Bod 91]. What Dean and Boddy call quality of a result, is exactly the **intrinsic utility** of the decision procedures proposed by Russell and Wefald. The focus of Dean and Boddy work has been on deliberation scheduling applied to time-dependent planning problems [Bod 91]. In this area, they proposed several scheduling algorithms to explicitly allocate resources to anytime tasks, so as to maximize a global value of the computations performed by an agent. A **performance profile** providing probabilistic information about the quality of the result, is always associated with an anytime algorithm. Since that time, many related works have been proposed.

Flexible inferences are the basic components of the computational model proposed by Horvitz to implement decision-theoretic reasoning about what next action has to be performed. Such an anytime decision procedure is able to return a result whose **object-related value** is an increasing continuous function of time spent in computation [Hor 87], [Hor 89], [Hor 90]. The main focus of Horvitz deals with finding an optimum balance between time spent doing reasoning and time spent acting, in health care domains.

Lesser, Pavlin and Durfee investigated **approximate processing** as an approach allowing a run-time selection of approximations to meet timing constraints with predictable effects on the quality of the results [Les 88]. Several approximation strategies have been experimented on **DVMT** (**D**istributed **V**ehicle **M**onitoring **T**estbed). Recently, Garvey and Lesser suggested the **design-to-time** processing approach as a generalization of approximate processing [Gar 93b]. Design-to-time supports multiple solution methods that are available for tasks, those methods making tradeoffs in solution quality versus time. The main issue is then to design a solution to a problem so as to maximize its quality within the available time.

The last but not the least work on anytime algorithms has been recently conducted by Russell and Zilberstein in [Rus 91b], and by Zilberstein in [Zil 93]. Their work gives a framework for structuring a computation as a combination of several anytime algorithms. Such a process, called **compilation** of anytime algorithms, provides specified amounts of runtime to each basic component of the system. Figure 4 describes in an oversimplified way, the compilation of anytime components.

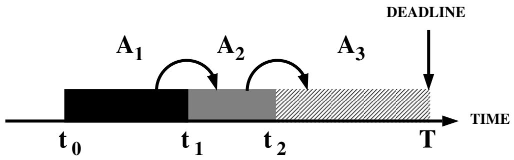

Figure 4

Assume that an agent has to combine 3 anytime algorithms A_1, A_2 and A_3, the procedure A_1 producing approximate results for A_2, which then elaborates approximate results for A_3. The compilation process allocates amounts of runtime to A_1, A_2 and A_3 so as to maximize the quality of the result produces by A_3 when the deadline T occurs. Such a process requires the knowledge of the deadline T and is based on the availability of the performance profiles associated with each elementary component. Because the amounts of runtime are directly calculated by the compiler, the resulting anytime algorithms are called **contract** anytime algorithms.

Garvey and Lesser have identified some relevant properties of anytime algorithms, contract algorithms and design-to-time computation [Gar 93b]. More precisely, anytime algorithms produce a result at any time and are expected to produce better results as they are given more time. Contract algorithms are anytime algorithms which may produce no useful results if interrupted before their allocated time. Conversely, design-to-time approach needs the availability before run-time, of a set of solution methods with discrete duration and expected quality values.

Shih, Liu and Chung have examined the real-time scheduling problem with a model of **imprecise computations**, assuming that each task is decomposed into a mandatory part and an optional part [Liu 91], [Shi 92]. Their scheduling approach consists in exactly scheduling the mandatory parts of all the tasks, and to compute the optional parts only if more time is available. Consequently, this model requires hard deadlines for mandatory parts, and soft deadlines for optional parts. Several on-line scheduling algorithms minimizing the total error of a set of imprecise tasks, have been experimented [Shi 92]. Although this approach has received a great deal of scepticism, it seems well-suited to model some conventional algorithms.

As a pedagogical illustration, let us give a brief analysis of sorting problem through the well-known heapsort algorithm. Horvitz has developed a comprehensive analysis of sorting to illustrate the problem of assigning utility values to partial computations [Hor 90]. His work is based on the analysis of several polynomial-time sorting strategies, such as selectionsort, shellsort and bubblesort. Although heapsort does not make instantly partial results available, it is able to be interrupted at any time, as soon as a mandatory part has completed.

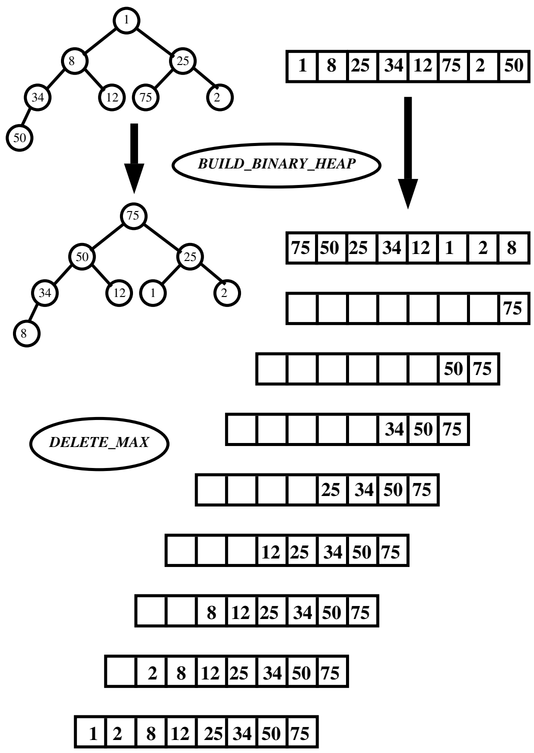

Figure 5

Figure 5 shows the behavior of heapsort algorithm applied to the {1, 8, 25, 34, 12, 75, 2, 50} set of 8 unsorted elements. The basic strategy of heapsort is to build first of all, a binary heap by using only one array. This first phase is called **build_binary_heap** phase. Heapsort performs next 8 **delete_max** operations, by swapping at each step, the last element in the heap with the first, decrementing the heap size, and building a new heap. Notice that we change the ordinary property of the binary heap, so that the parent has a larger key than the childs. Adopting such a property provides a more typical increasing sorted order. Each "delete_max" step increments then the number of contiguous elements, starting from the high end of the array, that are currently in the positions they will occupy after the array has been completely sorted. Such an objective quality of partial results, is referred to as **high-end completion** by Horvitz [Hor 90].

During the "build_binary_heap" operation, no useful partial results are available. But, each **delete_max** operation increases the high-end completion function, making available a partial sorted array of higher quality, at each step. Therefore, the **build_binary_heap** operation can be viewed as a mandatory part while the eight **delete_max** operations represent the optional part of the sorting procedure.

More generally, imprecise computations can be viewed as a special case of anytime algorithms, for which a default result is not instantly available.

3.2. Anytime computing and reactive/anytime agents

Anytime algorithms, flexible computations, contract algorithms, design-to-time problem solving model and imprecise computations share the same underlying model of computation, called **anytime computing**, which is based on the paradigm that it is better to have a crude and incomplete solution in time than one that is fully elaborated but too late. Within his model of **operational rationality**, Zilberstein defined anytime computing as "a model of computation that uses anytime algorithms as basic components and includes mechanisms for **automatic** scheduling the components so as to maximize a certain objective function" [Zil 93]. This model of operational rationality is interesting in that it simplifies the design of large real-time systems. Unlike conventional approches which try to meet a set of timing constraints through scheduling strategies, anytime computing provides a framework to maximize the utility of a decision when an agent has to operate under time pressure.

We have a less restrictive view of anytime computing because we focus our attention on achieving a level of predictability for real-time agents. We need then programming approaches and formal tools to foretell that an implementation will not violate a set of timeliness requirements. We define then anytime computing as a model of computation including meta-level mechanisms to **explicitly** control anytime algorithms, in such a way that the best satisficing decision that meet timing constraints is always selected. Notice that such a definition is not contradictory to the theory of operational rationality. Our approach of anytime computing provides a different methodology to develop real-time systems, in order to achieve a set of predictable temporal properties.

In other words, **operational rationality** provides a methodology to design what we have called in section 1, **time-constrained deliberative agents**, achieving timeliness and flexibility. What we want to do, is to extend the concept of time-constrained deliberative agent so that some predictable behavioral properties emerge. Our work can be viewed as an attempt to implement time-constrained deliberative agents, using more predictable programming formalisms. Another difference with operational rationality lies in the fact that we investigate more explicit mechanisms to control base-level inferences. Furthermore, we believe that fundamental concepts such as **utility** of a decision, or **quality** of a partial result returned by an anytime algorithm, are domain-specific and application-dependent functions. Therefore, some utility-driven mechanisms need to be directly expressed by the designer, according to the unpredictable run-time behavior of the environment.

From a functional point of view, the model of time-constrained deliberative agent we suggest, is organized as depicted by the diagram in figure 6.

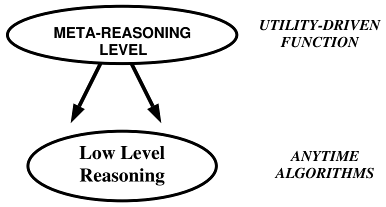

Figure 6

A meta-reasoning level controls at run-time, a set of anytime algorithms, in an utility-driven way. From a programming point of view, our model of time-constrained deliberative agent, will be referred to as **reactive/anytime** agent in the remainder of this paper. We define a **reactive/anytime** agent as a time-constrained deliberative agent supporting an **anytime** model of computation, and involving a **reactive** behavior of the meta-reasoning level. Therefore, the expected properties of a reactive/anytime agent are the conventional properties of a time-constrained deliberative agent plus a **predictable** behavior of the meta-reasoning function. Notice that the term **reactive** involves both conventional properties of reactive agents, and more specific programming approaches of reactive systems. Those programming approaches are described in the next section.

4. Reactive programming approach of real-time systems

Reactive systems have been defined by Harel and Pnueli as systems that maintain an ongoing relationship with their environment [Har 87]. Such systems cannot be naturally described in terms of a function computing a terminal state from an initial state. Therefore, several languages have been proposed to reactive systems programming: an imperative synchronous language, Esterel [Ber 92], two data-flow synchronous languages, Lustre and Signal [Ben 93], a graphical specification formalism, Statecharts [Har 87], and an extension of the C language, Reactive C [Bou 91]. In this section, we focus our attention on the underlying paradigm of reactive systems programming, shared by all these formalisms. A more comprehensive overview of reactive programming can be found in the special issue of the proceedings of the IEEE [IEEE 91].

Reactive languages use a **logical** time divided into **instants** which are the moments when programs react. Furthermore, a reactive program communicates with its environment by **signals**. Let us illustrate the model of reactive programming, considering the example of figure 7, written in Esterel style.

```
loop
  trap END in
    await SECOND ;
    emit ALARM ;
    exit END ;
  ||
    await BUTTON ;
    emit ACTION ;
    exit END ;
  end
end
```

Figure 7

First of all, this program handles two input signals, SECOND and BUTTON, and two output signals, ALARM and ACTION. This example illustrates the use of the trap statement and the parallel operator ("||"). The trap END statement defines a block exited when "exit END" is executed. In this case, the block is terminated as soon as SECOND or BUTTON or both become present. When both signals are simultaneously present, ALARM and ACTION are both emitted according to the semantics of the parallel operator of Esterel. A run-time behavior of this reactive program is highlighted by the diagram in figure 8.

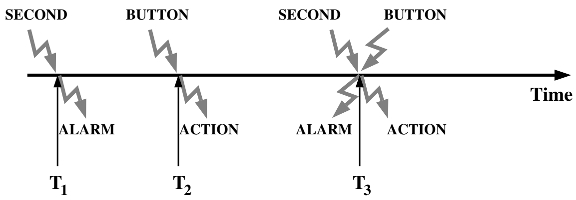

Figure 8

The logical instants when that program reacts are T_1, T_2 and T_3. Reactive programming supports some interesting properties for designing time-critical reactive systems for which the temporal behavior at run-time, has to be proved formally. The first property is that such reactive formalisms support a mathematical semantics. The second one deals with the compilation process. A brief overview of the compilation phase of a reactive program is given in the figure 9. The compiler produces a **sequential finite automaton** whose temporal behavior is strictly equivalent to the temporal behavior expected by the designer. In particular, the automaton provides entries for proving the temporal behavior of the program, using current automata verification tools, such as the **AUTO** system [Ver 87]. In other words, concurrency in a reactive program source is "sequentialized", avoiding at run-time, conventional problems that arise with parallel languages such as Ada. The sequential finite automaton generated by the compiler, can be then considered as a **compiled scheduler**.

More conventional languages such as Ada may be used for programming pure reactive systems. But run-time concurrency in Ada needs a scheduler whose temporal behavior can not be easily proved in a formal way. In particular, there is no assurance that the program will always respect the order of events. More generally, priority inversions or deadlocks are possible, and a statement such as "delay(n)" never ensures that the process will be delayed exactly n time units. Reactive programming allows logical concurrency between tasks at the design level, but considers that run-time concurrency is neither mandatory nor desirable when we have to design highly predictable systems.

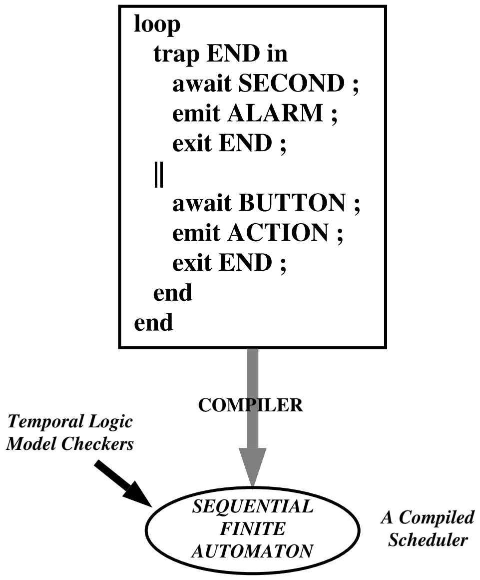

Figure 9

Unlike some specification formalisms, reactive languages such as Esterel are directly executable since the sequential automaton generated by the compiler, is written in C or Ada languages.

5. Reactive/Anytime agents

We define a **reactive/anytime** agent as a deliberative-time constrained agent supporting an **anytime** model of computation and including a **reactive programming** approach of the meta-reasoning level. From a functional point of view, a reactive/anytime agent is organized as depicted in the figure 10.

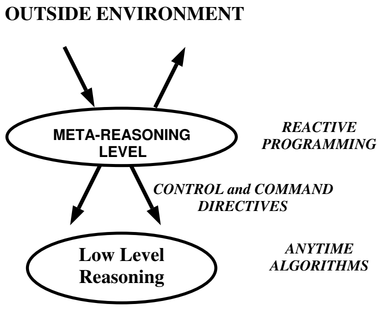

Figure 10

A reactive kernel written in a reactive language controls a set of anytime tasks through a set of control and command directives. According to the reactive programming approach, the reactive kernel maintains a continuous relationship with its environment. At run-time, the reactive kernel follows an **utility driven** behavior, handling a set of anytime tasks so as to maximize an objective utility function. It is important to emphasize that the meta-reasoning level of a reactive/anytime agent has to be directly expressed by the designer. The control of the anytime components by the reactive kernel is then driven by the calculation of the utility value of a decision, according to the current situation, the time pressure, and the performances profiles of the basic components.

Temporal predictability is achieved when it is possible to formally prove that the system will respect timing constraints, and particularly that 100% of hard real-time tasks will respect their deadlines. Such a proof is usually given by a scheduler which performs a run-time scheduling of a set of tasks, according to the model of tasks (release times, periodic or aperiodic tasks, deadlines, worst-case execution times). Such a conventional approach of predictability has limited applicability for intelligent systems that operate in highly unpredictable environments involving dynamic timing constraints. First of all, because worst-case execution times of tasks are often unknown, or much larger than their average-case execution times. Secondly, because deadlines are often only known at run-time, and then, predictability cannot be formally proved at compile-time.

As mentioned above, the reactive programming approach supports a set of tools which provide the ability to foretell that a set of timeliness requirements will be respected. Our model of reactive/anytime agent allows then to formally prove the temporal behavior of the meta-reasoning level. Let us discuss the example given in figure 11, written in Esterel language style.

```
LOOP
  AWAIT I1
  START T1(......)
  DO
    HALT
  WATCHING S1
   CONSULT T1(current_result)
   KILL T1
   EMIT O1(current_result)
ENDLOOP
||
LOOP
 AWAIT I2
  START T2(......)
  DO
    HALT
  WATCHING S2
   CONSULT T2(current_result)
   KILL T2
   EMIT O2(current_result)
ENDLOOP
```

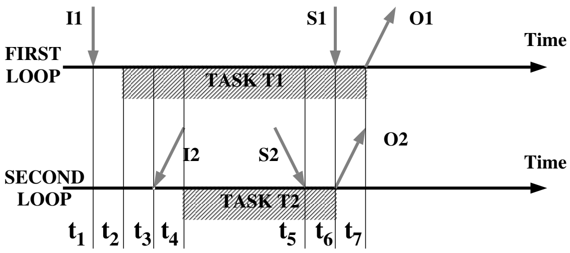

Figure 11

For the sake of clarity, the meta-reasoning reactive kernel does not include the calculation of an utility function. This example only illustrates why combining reactive programming and anytime computing could achieve a level of predictability. The reactive program defines two parallel loops whose temporal behavior are respectively illustrated by the two arrows. The first loop (resp. the second loop) waits for an input signal I1 (resp. I2) and starts then a task T1 (resp. T2). On arrival, of signal S1 (resp. S2) the body of the watching statement is "killed". Therefore, the current result of task T1 (resp. T2) is collected by the kernel, and the corresponding output signal O1 (resp. O2) is emitted.

First of all, well-known problems dealing with true concurrency between the two parallel loops do not arise since the compiler produces a **sequential** finite automaton. Therefore, the temporal behavior of the reactive kernel can be formally proved. In particular, the instants when statements such as starting, consulting or killing a task, are performed, can be predicted. For instance, we can foretell that signal O1 will be emitted exactly at time t_7 if the input signal S1 is present at time t_6. Moreover, we can assume that the delay $\Delta t = t_7 - t_6$, corresponding to the time spent in collecting the current results of task T1, depends on architectural features and can be upper bounded. Notice that input signals S1 and S2 are considerd as aperiodic events which cannot be predicted at compile-time.

Assuming that the worst-case execution time of task T1 is C_{T1}, predictability will be achieved iff the delay $D=t_6-t_1$ is greater than C_{T1}. In most cases, C_{T1} will be much larger than the average-case execution time. Therefore, it will not be possible to formally foretell that a set of timeliness requirements will be respected at run-time, since in the general case, D will be less than C_{T1}. Conversely, if task T1 (resp. T2) is an anytime algorithm, we can formally prove that an approximate result will be performed by T1 (resp. T2) for any value of D. It is important to emphasize that reactive/anytime agents involve a new approach of predictability based on the expected quality of the result elaborated by a task and given by its performance profile. We refer to such property as **anytime responsiveness**. Anytime responsiveness considers that the meta-reasoning level of a reactive/anytime agent is predictable, in that the system will be able to produce a **satisficing** response at any time. Anytime responsiveness differs from conventional predictability in that a quantitative measurement of timeliness is available. This measurement gives the expected quality of the responses over the available computation time.

Unlike **operational rationality** theory developed by Zilberstein in [Zil 93], **anytime responsiveness** makes no restrictive assumptions about the model of anytime algorithms. We consider that the model of imprecise computations has to be included as a basic component of an anytime responsive agent, when the worst-case execution time of the mandatory part is realistic. On the other hand, anytime responsiveness leaves to the programmer the decision of what quality of a result is acceptable, and provides some explicit tools to perform tradeoffs between accuracy and time, depending on which is more critical. This approach has several disadvantages compared to the model of operational rationality. Nevertheless, we believe that our approach is well-suited for designing adaptive and responsive agents that have to operate with highly unpredictable environments.

A **reactive/anytime** approach of real-time AI can be viewed as an example of what Musliner and Hendler call cooperative real-time AI systems [Mus 94]. Those systems support a separate approach of AI and real-time subsystems, both systems cooperating in achieving overall desirable behaviors and expected properties.

6. Conclusions and future issues

In this paper we propose a model of reactive/anytime agent as a framework to design intelligent agents with real-time performance. We believe that reactive/anytime agents give the opportunity to associate **reactive** and **deliberative** approaches to achieve flexibility, timeliness and a level of predictability. In other words, the reactive/anytime approach is an attempt to design both **flexible** and **responsive** agents. This claim is schematically represented by the diagram in figure 12, where agents are situated according to their expected properties.

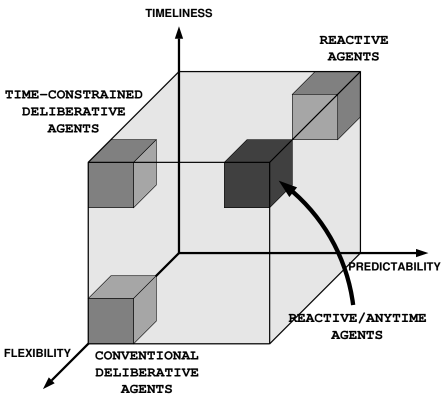

Figure 12

Pure reactive agents are responsive (timely and predictable), but provide poor flexibility. Conventional deliberative agents are flexible, but neither timely nor predictable. Real-time deliberative agents extend conventional deliberative agents to achieve timeliness. Finally, the anytime model of computation supported by a reactive/anytime agent provides flexibility and timeliness, while a reactive programming approach of the meta-reasoning function allows to achieve a level of predictability.

The model of deliberative time-constrained agent we suggest is not only an abstract model of computation, since we are currently experimenting the ability to express the control of a set of anytime basic components, in a reactive way. We have then developed a testbed, called **Saturne**, which is described in depth in [Ade 95]. Saturne may be considered as a rapid prototyping tool for designing time-critical applications in terms of a distributed network of reactive/anytime agents, using the Esterel reactive language [Ade 94a], [Ade 94b]. Related works include the **asynchronous transfer of control** mechanism, recently proposed by Giering and Baker, to integrate the model of imprecise computations into Ada [Gie 94]. The difference with our approach lies in the choice of the Ada 9X emerging revised Ada language standard which not provides formal tools to foretell a set of timeliness requirements.

Current work address the following main issues: how can one determine and represent the performance profile of a given anytime algorithm ? What kind of tools have to be included into current reactive formalisms to explicitly monitor a set of anytime tasks, in an utility-driven manner. We are investigating such issues with a realistic time-critical application as part of the "Copilote Electronique" Pilot's Associate Project supported by Dassault-Aviation French Company.

References

[Ade 94a] Adelantado M., Boniol F., "Synchronous/anytime programming for distributed reactive systems", Second Workshop on Parallel and Distributed Real-Time Systems, Cancun, Mexico, April 28–29, 1994

[Ade 94b] Adelantado M., Boniol F., "Controlling real-time asynchronous tasks with Esterel synchronous languages", (short paper), 13th ACM Symposium on Principles of Distributed Computing (PODC' 94), Los Angeles, California, August 14–17, 1994

[Ade 95] Adelantado M., Boniol F., Pinault D., "A simulation environment for designing distributed reactive/deliberative time-constrained applications", 28th Annual Simulation Symposium, Phoenix (AZ), April 9–13, 1995

[Agr 87] Agre P. E., Chapman D., "Pengi: an implemantation of a theory of activity", Proceedings of the Sixth National Conference on Artificial Intelligence, pp. 268–272, Seattle, Washington, July 1987

[Ben 93] Benveniste A., Caspi P., Le Guernic P., Halbwachs N., "Data-flow synchronous languages", IRISA Technical Report, n° 768, October 1993

[Ber 92] Berry G., Gonthier G., "The Esterel synchronous programming language: design, semantics, implementation", Science of Computer Programming, Vol. 19, n° 2, 1992

[Ber 93] Berry G., Ramesh S., Shyamasundar R. K., "Communicating reactive processes", Proceedings of the 20th ACM Conference on Principles of Programming Languages, Charleston, Virginia, 1993

[Bes 93] Bestavros A., "Cleopatra: building responsive systems from physically-correct specifications", Tecnical Report, Department of Computer Science, Boston University, 1993

[Bly 93] Blyte J., Reilly W. S., "Integrating reactive and deliberative planning for agents", Technical Report CMU-CS-93–155, May 1993

[Bod 91] Boddy M., "Solving time-dependent problems: a decision-theoretic approach to planning in dynamic environments", PhD Dissertation, Brown University, Providence, Rhode Island 02912, Rapport CS-91–06, Mai 1991

[Bod 94] Boddy M., Dean Th., "Deliberation scheduling for problem solving in time-constrained environments", Artificial Intelligence 67, pp. 245–285, 1994

[Bou 91] Boussinot F., "Reactive-C: an extension of C to program reactive systems", Software Practice and Experience, Vol. 21(4), April 1991

[Bro 91] Brooks R. A., "Intelligence without reason", Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, pp. 569–595, Sydney, Australia, 1991

[Dea 88] Dean Th., Boddy M., "An analysis of time-dependent planning", Proceedings of the Seventh National Conference on Artificial Intelligence, pp. 49–54, Minneapolis, Minnesota, 1988

[Dea 91] Dean Th., Wellmann M. P., "Planning and control", Morgan Kaufmann Publishers, San Mateo, California, 1991

[Gar 93a] Garvey A., Lesser V. R., "A survey of research in deliberative real-time artificial intelligence", Department of Computer Science, University of Massachusetts, UMass Technical Report 93–84, November 19, 1993

[Gar 93b] Garvey A., Lesser V. R., "Design-to-time real-time scheduling", IEEE Transactions on Systems, Man and Cybernetics, 23(6), 1993

[Gau 94] Gautier T., Le Guernic P., Maffeis O., "For a new real-time methodologie", IRISA Technical Report, n° 870, October 1994, Rennes, France

[Ger 94] Gerber R., "Languages and tools for real-time systems: problems, solutions and opportunities", University of Maryland Technical Report, CS-TR-3362, UMIACS-TR-94–117, October 1994

[Gie 94] Giering E. W., Baker T. P., "Ada 9x asynchronous transfer of control: applications and implementation", ACM Sigplan Workshop on Languages, Compiler and Tool Support for Real-Time Systems, Orlando, Florida, 1994

[Hal 93] Halbwachs N., "Synchronous programming of reactive systems", Kluwer Academic Publishers, 1993

[Har 87] Harel D., "Statecharts: a visual formalism for complex systems", Science of Compiler Programming, Vol. 8, n° 3, 1987

[Hay 94] Hayes–Roth B., Lalanda Ph., "Deadline management in intelligent agents", Stanford University, Knowledge Systems Laboratory, Technical Report n° KSL 94–27, May 1994

[IEEE 91] Proceedings of of IEEE, Another Look at Real-Time Programming, September 1991

[Hor 87] Horvitz E. J., "Reasoning about beliefs and actions under computational resource constraints", Proceedings of the Workshop on Uncertainty in Artificial Intelligence", Seattle, Washington, 1987

[Hor 89] Horvitz E. J., Suermondt H. J., Cooper G. F., "Bounded conditioning: flexible inference for decisions under scarce resources", Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence, Windsor, Ontario, 1989

[Hor 90] Horvitz E. J., "Computation and action under bounded resources", PhD Dissertation, Department of Computer Science and Medecine, Stanford University, California, Report n STAN-CS-92–1430, December 1990

[Laf 88] Laffey T. J., Cox P. A., Schmidt J. L., Kao S. M., Read J. Y., "Real-time knowledge based systems", AI Magazine, Vol. 9, n° 1, Spring 1988

[Les 88] Lesser V. R., Pavlin J., Durfee E., "Approximate processing in real-time problem solving", AI Magazine, 9(1), pp. 49–61, Spring 1988

[Liu 91] Liu J. W., Shih W-K., Yu A., C-S., Chung J-Y. C., Zhao W., "Algorithms for scheduling imprecise computations", IEEE Computer 24, 5, pp. 58–68, May 1991

[Mus 94] Musliner D. J., Hendler J. A., Agrawala A. K., Durfee E. H., Strosnider J. K., Paul C. J., "The challenges of real-time AI", University of Maryland Technical Report CS-TR-3290, UMIACS-TR-94–69, June, 1994

[Rus 89] Russell S. J., Wefald E. H., "Principles of metareasoning", Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning, R. J. Brachman et al. (eds.), Morgan Kaufmann, San Mateo, California, 1989

[Rus 91a] Russell S. J., Wefald E. H., "Do the right thing: studies in limited rationality", MIT Press, Cambridge, Massachusetts, 1991

[Rus 91b] Russell S. J., Zilberstein S., "Composing real-time systems", Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, pp. 212–217, Sydney, Australia, 1991

[Shi 92] Shih W-K., "Scheduling in real-time systems to ensure graceful degradation: The imprecise computation and the deferred-deadline approaches", PhD Dissertation, Department of Computer Science, Report n° UIUCDCS-R-92–1765, University of Illinois at Urbana Champaign, 1992

[Sho 87] Shoppers M., "Universal plans for reactive robots in unpredictable environments", Proceedings of the International Conference on Artificial Intelligence, IJCAI-87, 1987

[Str 94] Strosnider J. K., Paul C. J., "A structured view of real-time problem solving", AI Magazine, pp. 45– 66, Summer 1994

[Ver 87] Vergamini D., "Verification de reseaux d'automates finis par equivalences observationnelles: le système AUTO", PhD Dissertation, Nice University, 1987

[Zil 93] Zilberstein S., "Operational rationality through compilation of anytime algorithms", PhD Dissertation, University of Berkeley, Report UCB:CSD-93–743, 1993