A Real-Time Agent Model in an Asynchronous-Object Environment[1]

Z. GUESSOUM*, M. DOJAT**
*Equipe OMC, LAFORIA-IBP, Université Paris 6, Boîte 169,
4, place de Jussieu, F-75252 PARIS
**INSERM Unité 296, Faculté de Médecine
8, avenue du Général Sarrail, 94010 CRETEIL FRANCE
e-mail: {guessoum, dojat}@laforia.ibp.fr

Abstract

To build intelligent control systems for real-life applications, we need to design software agents which combine cognitive abilities to reason about complex situations, and reactive abilities to meet hard deadlines. We propose an operational agent model which mixes AI techniques and real-time performances. Our model is based on an ATN (Augmented Transition Network) to dynamically adapt the agent's behavior to changes in the environment. Each agent uses a production system and is provided with a synchronization mechanism to avoid the possible inconsistencies of the asynchronous execution of several rule bases. Our agents communicate by message-passing and are implemented in an asynchronous-object environment. We report on the use of our agent model in intensive care patient monitoring.

Key Words

Multi-Agent, Actors, Real-Time, ATN, Production Rules, Object-Oriented Language, Artificial Ventilation.

Résumé

Le contrôle de systèmes dynamiques nécessite de définir des agents qui allient des capacités cognitives, pour raisonner sur des situations complexes, à des capacités réactives, pour respecter des contraintes temporelles temps réel. Nous proposons un modèle d'agent opérationnel qui possède ces caractéristiques. Il repose sur l'utilisation d'un ATN (Augmented Transition Network) pour adapter son comportement aux changements qu'il détecte dans son environnement. Chaque agent possède pour raisonner un mécanisme d'inférences d'ordre 1. Il gère un graphe de dépendances pour assurer une cohérence entre les différents raisonnements effectués par les agents. Nos agents communiquent par envoi de messages et sont implémentés dans un environnement d'objets asynchrones. Nous rapportons une application de notre modèle au monitorage de la ventilation artificielle des patients en soins intensifs.

[1] Cet article a été publié dans : Lecture Notes in Artificial Intelligence 1038, Agents Breaking Away, Walter Van de Velde and John W. Perram (eds.), Netherlands, January 1996.

A Real-Time Agent Model in an Asynchronous-Object Environment

Z. GUESSOUM[*], M. DOJAT[**]
*Equipe OMC de J-F Perrot, LAFORIA-IBP, Université Paris 6, Boîte 169,
4, place de Jussieu, F-75252 PARIS
**INSERM Unité 296, Faculté de Médecine
8, avenue du Général Sarrail, 94010 CRETEIL FRANCE
e-mail: {guessoum, dojat}@laforia.ibp.fr

Abstract

To build intelligent control systems for real-life applications, we need to design software agents which combine cognitive abilities to reason about complex situations, and reactive abilities to meet hard deadlines. We propose an operational agent model which mixes AI techniques and real-time performances. Our model is based on an ATN (Augmented Transition Network) to dynamically adapt the agent's behavior to changes in the environment. Each agent uses a production system and is provided with a synchronization mechanism to avoid the possible inconsistencies of the asynchronous execution of several rule bases. Our agents communicate by message-passing and are implemented in an asynchronous-object environment. We report on the use of our agent model in intensive care patient monitoring.

Key Words

Multi-Agent, Actors, Real-Time, ATN, Production Rules, Object-Oriented Language, Artificial Ventilation.

1 Introduction

Artificial intelligence (AI) techniques are well adapted to perform tasks such as diagnosis, design and classification. To build intelligent control systems for real-life applications, AI techniques must also handle specific real-time aspects, such as resource limitations and guarantee of timely response. This leads to the emerging research area of "real-time AI" [Charpillet and Théret 1994; Garvey and Lesser 1994; Musliner et al. 1995].

Multi-Agent Systems (MASs) seem well adapted to model a variety of real applications. To be useful for real-time domains, MASs must (1) handle asynchronous events, (2) manage resource overload and time constraints and (3) ensure a control of distributed autonomous entities. We propose here an agent model which achieves these goals by integrating smoothly so-called reactive abilities (to meet hard deadlines) and cognitive abilities (to act rationally by using knowledge and to reach a fixed goal when constraints are relaxed).

1.1 Blackboards & Actors

MASs proposed in the literature rely on two approaches to achieve communication between agents: 1) the blackboard model introduced in the Hearsay-II system [Erman et al. 1980] and used in several recent systems [Hayes-Roth et al. 1992; Bussmann and Demazeau 1994; Charpillet and Boyer 1994] ; and 2) the actor model initially proposed by [Hewitt 1977] which developed into various concurrent languages [Agha 1986; Yonezawa et al. 1986; Yokote and Tokoro 1987; Ferber and Briot 1988] .

In the blackboard approach, knowledge sources use available information without knowing its origin and produce information without worrying about its destination, whereas in the actor approach, an actor communicates information directly to another actor via message-passing. As observed by [Nii et al. 1989], blackboard systems are prone to inefficiency problems, which are to be solved by parallelization, e.g. with actors. Actors may thus be considered as the basic element for building agents. Further, the combination of the actor concept and the object paradigm leads to the notion of "agent-oriented programming" [Shoham 1993], which is the framework of the present paper. We start from an environment which provides objects, classes (with inheritance), production rules, and processes.

1.2 Cognitive, Reactive and Hybrid Models

Several agent models have been proposed. Two main approaches can be distinguished: *cognitive* and *reactive* (see [Ferber 1995]). In the cognitive approach, each agent contains a symbolic model of the outside world, about which it develops plans and makes decisions in the traditional (symbolic) AI way. In the reactive approach, on the other hand, simple-minded agents react rapidly to asynchronous events without using complex reasoning.

Many researchers have suggested that neither a completely reactive nor a completely cognitive approach is suitable for building complete solutions for real-life applications. Hybrid models, such as *TouringMachines* [Ferguson 1992], *InteRRaP* [Müller and Pischel 1994] and the model presented in [Bussmann and Demazeau 1994], have been proposed to combine the advantages of both reactive and cognitive models. In these models, agents are decomposed in a set of modules which can in turn be of a reactive or cognitive nature. However, the problem with such models is that of controlling the interactions between these fundamentally different modules: reactive and cognitive modules, so to say, do not live with the same time scale, which makes it difficult to integrate the different temporal sequences.

1.3 Our Model

Our hybrid agent model relies on a first layer made up of interactive modules that can be either reactive or cognitive, and interact asynchronously and concurrently. A higher level *supervision* module controls these interactions via an ATN so as to impose a global temporal sequencing. As we shall see, this time scale is based on the firing of individual rules as its basic unit. Modules of the base layer include the *perception* module (reactive), the *reasoning* module (cognitive) and the *communication/action* module (which can be either reactive or cognitive).

In our model, each reasoning module owns a (first order, forward chaining) rule base, which represents the bulk of its cognitive abilities. Control of the reasoning process is achieved by a set of metarules (called a metabase) also owned by the module. These control metarules operate on objects that are also accessed by the supervisor's ATN, thus establishing the supervision link at the level of the individual rule firing. As opposed to the blackboard approach, where control is achieved through a separate architecture [Hayes-Roth 1985], in our model each individual agent carries its own control knowledge, represented by the metabase of its reasoning module.

As regards real-time response, our model is adequate provided a time granularity of single rule firing. This seems to be acceptable in many industrial applications [Barachini and Grenec 1993]. As our application example will demonstrate, it exhibits:

- Ability to monitor in real-time information provided by the environment and to adapt its behavior to respond in real-time to changes in the environment.
- Ability to coordinate its actions in accordance with other agents actions to avoid inconsistencies.

The purpose of this paper is to present our model and to show its use to design a real-time prototype for intensive care unit patient monitoring. Section 2 presents the main Smalltalk-based tools of our implementation: *Actalk* and *NéOpus*. Section 3 describes the collective behavior of a society of agents. Section 4 describes the individual modules of our model. In Section 5, we report on the application to patient monitoring. Finally, we discuss the advantages of our approach to design real-time MASs.

2 Technical Context

We have opted for an environment which combines object-oriented programming and production rules. We use *Smalltalk-80* as our base language. This enables our implementation to make use of various software components (alias frameworks) such as the Smalltalk Discrete Event Simulation Package (see [Guessoum 1995]) for observing the temporal behavior of the system under simulated real-time conditions (see 5.4 below). We have used *RPC-Talk* [Wolinski 1994], which provides *Smalltalk-80* with RPC (Remote Procedure Call) facilities, to put any number of machines at the service of our MASs. Our actors are built with *Actalk* [Briot 1989, Briot 1995] a generic platform for implementing various actor models in *Smalltalk-80*. *Actalk* may be seen as the foundation stone of our model. The rule bases and metabases of the reasoning modules use *NéOpus* [Pachet 1992; Pachet 1995], a first order inference engine completely embedded in *Smalltalk-80*. We have added a few features to customize these components to meet the specific needs of our model.

2.1 Actalk

Among other things, *Actalk* allows to transform ordinary Smalltalk objects into actors. Asynchronism, a basic principle of actor languages, is implemented by enqueuing the received messages into the mail box, thus dissociating message reception from its interpretation. In *Actalk*, an actor is composed of three objects:

- an instance of class `Address` represents the mail box of the actor. It defines the way messages will be queued for later interpretation;
- an instance of class `Activity` represents the internal activity of the actor. It provides autonomy to the actor. It owns a Smalltalk process which continuously removes messages from the mail box and launches their interpretation by the actor;
- an instance of class `ActiveObject` represents the behavior of the actor, i.e. the way individual messages will be interpreted.

To build an agent with *Actalk*, all one has to do is to create the three components (call them *ad*, *act* and *actObj* respectively) and put them together as an actor by sending the message *active* to *actObj*. Customizing *Actalk* therefore means defining subclasses of `Address`, `Activity` and `ActiveObject`. See 5.2 for some details on our realization.

2.2 NéOpus

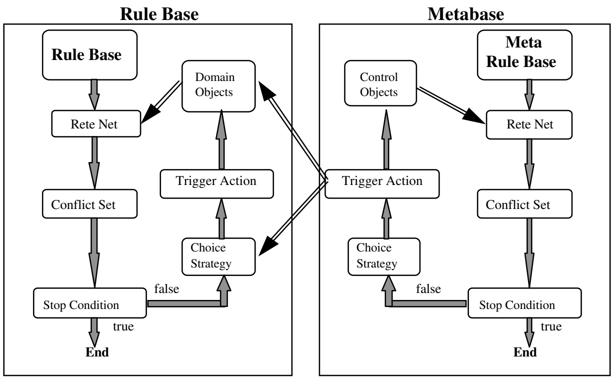

Fig. 1. Control with metarules in NéOpus.
This figure highlights the organizational similarity between domain knowledge and control knowledge. The fired metarules act on the firing strategy of rules, on the stop condition and on the objects moving in the associate rule-base net.

NéOpus realizes a neat integration of rule-based programming with *Smalltalk-80*. It uses the Rete algorithm to compile rule bases. One of its prominent features is declarative specification of control with metarules [Pachet and Perrot 1994]. Metarules are similar to rules and operate on so-called control objects. With each rule base there may be associated a metabase which controls the firing of its rules (see Fig. 1).

A number of metabases have been designed by *NéOpus* users to define standard types of control. In particular, specific metabases have been built to reason on critical situations [Dojat and Pachet 1992]. We have adapted some of them to our needs by changing a few individual metarules.

3 Relations Between Agents

We are interested in situations where an agent shares a collection of resources with other agents. Thus, agents must adapt themselves to take advantages of resources as needed, but must coordinate their actions to avoid inconsistencies. Researchers have evolved a range of approaches to coordinate a collection of autonomous entities. [Durfee et al. 1987; Gasser 1992] describe the mechanisms that improve the network coherence: organization, exchanging meta-level information, local and multi-agent planning, and explicit analysis and synchronization.

With a synchronization mechanism as proposed in [Ishida 90], each agent protects itself against conflicts or redundant actions concerning the other agents, at a cost of (1) reduced concurrency, and (2) synchronization overhead. However, if the level of dependency is low, and the granularity of actions is high, this mechanism can provide useful coordination, as observed by [Gasser 1992]. This is the case for our model. The granularity of our agents is high, each agent possesses its own rule base and the agent's rules are fired sequentially. Therefore dependency and interference between individual rules of the same agent do not have to be considered. Thus the dependency between agents is low.

There remains to avoid inconsistencies between different agents. To do so, we implement Ishida's dependency mechanism with a technique which improves its overall performance via a better anti-interference mechanism.

3.1 The Dependency Graph

Following Ishida's ideas, we distinguish two kinds of objects: local objects used by a single agent, and global objects used by several agents. The principle of the dependency mechanism may be defined as follows:

- each agent is provided with a dependency graph (Ishida' terminology) giving for each global object the list of other agents which use it;
- the dependency graph is used by the communication module of the agent to inform the other agents about the modification, creation or removal of global objects by the reasoning module. It is updated gradually when rules are triggered: if a global object is removed by rule actions, it is also automatically removed from the dependency graph and a message is sent to the other agents to update their graphs.

Inter agents conflicts resulting from access to global objects are thus avoided.

3.2 Anti-Interference Mechanism

Interference exists among two rules if there is a global object that both rules access and at least one modifies. The principle of our anti-interference mechanism may be defined as follows:

- each agent is provided with a list of those global objects that it is currently modifying (objects-in-use);
- each agent is provided with the collection of the objects-in-use lists of the other agents;
- we add two steps in the inference engine cycle:
  * *test*: before the firing of the selected rule, the agent verifies that no global object that is modified by the selected rule is also being modified by some other agent (as is apparent from its objects-in-use list collection). In this case, the global objects that are modified by the selected rule are added to its own

objects-in-use list and the rule is triggered, otherwise another fireable rule is selected;

∗ <u>updateList</u>: after the firing of the rule, the agent removes from its own *objects-in-use* list those global objects that were added to its in the <u>*test*</u> phase, i.e. the global objects that were modified by the rule.

The proposed solution prevents conflicts and redundant actions. It avoids also the synchronization messages used by Ishida.

4 Our Real-Time Agent Model

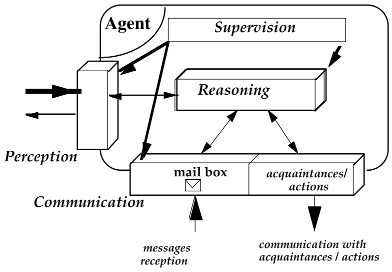

Fig. 2. Agent model

4.1 The Supervision Module

This module synchronizes the execution of concurrent actions of the other modules. It relies on two notions: *states* and *transitions*. States qualify the context as perceived by the other modules. Changes in the context are reflected as transitions between states. Now, these states and transitions naturally build up an ATN [Woods 70]. The ATN is a synthetic and deterministic representation of the agent's behavior.

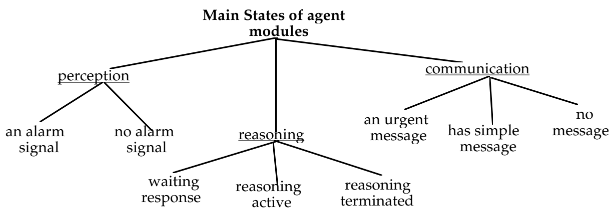

Fig. 3. Main states of agent modules
Different states correspond to each module. The combination of these states defines the global agent's state.

Each transition links an input state with an output state. The various signals received by the agent's modules represent the conditions of transition and the actions of transition change the state of the various modules (activate reasoning, terminate reasoning, ...). When these conditions are verified, the transition actions are executed and the agent's state is modified.

4.2 The Perception Module

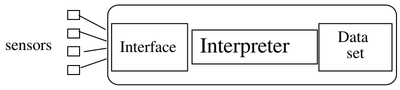

Fig. 4. The perception module

The perception module manages the interactions between the agent and its environment. It monitors sensors, translates and filters sensed data in accordance with the instructions of the reasoning or supervision modules (see Fig. 4). It may package information concerning the same phenomenon to facilitate interpretation. The data set obtained is used mainly by the reasoning module.

4.3 The Reasoning Module

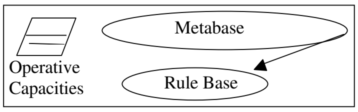

Fig. 5. The reasoning module

This module is responsible for generating adequate responses to the messages transmitted by the communication module, or to the changes detected by the perception module. To do this it relies on two kinds of capacities: operative, represented by the standard behavior of the associated Smalltalk objects (procedures, alias *methods* in the Smalltalk terminology), and cognitive, embodied in a NéOpus-based asynchronous production system [Guessoum 1994]. This production system mainly comprises: (1) a rule base which includes objects describing the agent's environment and rules representing suitable operations over these objects; (2) an inference engine which includes the dependency and anti-interference mechanisms; and (3) a metabase which provides a declarative representation of the control of reasoning.

4.4 The Communication/Action Module

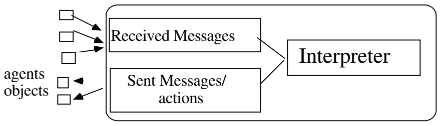

Fig. 6. The communication/action module

This module allows the agent to receive and to send messages asynchronously. It filters the received messages, determines their priority (LIFO, FIFO,...) and the type of treatment to give them. It owns the list of the agent's acquaintances. It sends them messages with various modes corresponding to specific protocols (*urgent, answer needed, ...*). The messages and their modes are provided by the reasoning module.

This module also effects the direct actions (modification of the environment via effectors) and indirect actions (information transmission to other agents) as directed by the reasoning module.

5 From Actalk Objects to Agents

In this section, we illustrate the implementation of our model with an application: intensive care unit patient monitoring.

5.1 Application: Artificial Ventilation Control

The system deals with patients suffering from respiratory insufficiency, assisted with mechanical ventilation. The problem is to monitor in real-time various ventilation signals (tidal volume (Vt), respiratory rate (RR) and expired-CO_2 pressure (PCO_2)), in order to diagnose the patient current state and to adapt the mechanical assistance accordingly. To perform this task, it is necessary to develop a complex temporal reasoning to diagnose the time-course of the patient's status [Dojat and Sayettat 1995]. In alarming situations such as hypoventilation or apnea the current therapy must be modified quickly (1 second). A first system, NéoGanesh, is in use at the hospital Henri Mondor (Créteil near Paris) [Dojat et al. 1995]. The extension based on a distributed architecture using our agent model, aims at increasing the system reactivity and incorporating additional distributed medical expertise.

5.2 Agents

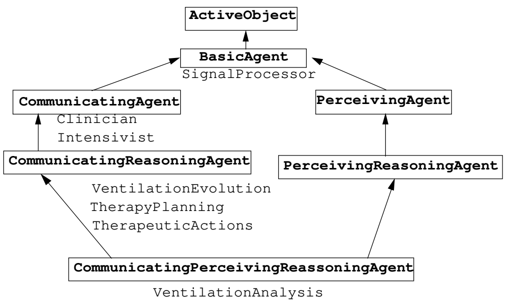

Fig. 7. Excerpt from the agents hierarchy used for artificial ventilation control application

All "intelligent" agents have the same general structure but they differ in 1) their sensor-driving layer: perception and communication/action modules; 2) their behavior: the know-how, the domain and control knowledge. Agents are complex entities and a hierarchy of agents (see Fig. 7) may be built according to several criteria, such as type of the sensor-driving structures, know-how and type of cognitive functions. `BasicAgent`, root of the classes describing our agent behavior, is encapsulated according to the Actalk principle to be transformed in an actor.

Note that, for example, agent *SignalProcessor* has only a simple behavior to process data acquisition. Whereas, agent *VentilationEvolution* exhibits a complex behavior to appreciate the time-course of the patient's ventilation.

Supervision Module

The supervision module represents the kernel of agent. Its main activity is to interpret its ATN. This is effected by the method `setProcess` redefined in subclass `AgentActivity` of the *Actalk* standard class `Activity`.

In Actalk , `Activity` manages and transmits the messages which are interpreted by `ActiveObject`. The method `setProcess`, which creates a process to remove continuously the messages buffered in the mail box, is redefined in `AgentActivity`. The new role of this process is to interpret the ATN. The ATN itself is defined in subclass `BasicAgent` of *Actalk*'s `ActiveObject`.

For example, the ATN of *VentilationEvolution* (see Fig. 8) gives priority to urgent messages such as alarms. In state 1, the condition "no message" leads to the action "wait" and to the persistence in state 1. The condition "an urgent message" leads to the action "read mailbox" whatever the state may be.

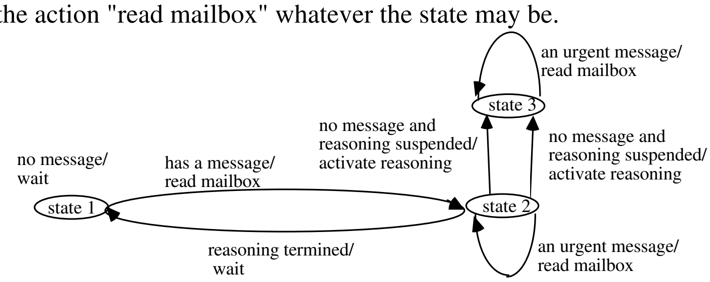

Fig. 8. *VentilationEvolution* ATN

Perception Module

It is mainly characterized by a list of sensors and an instance method which interprets the sensors data, compares them with the previous ones and transmits the new data to the reasoning module.

For example, several sensors (CO_2, RR, Vt) are connected to *SignalProcessor*. It scans the different sensors at a variable sample frequency, interprets the data and transmits the result to *VentilationAnalysis*.

Reasoning Module

To implement the anti-interference mechanism we have redefined the NéOpus inference engine cycle. The new cycle includes the two steps *test* and *updateList* (see 3.2).

The statements, that inform the Rete net of NéOpus about modifying, removing and adding of objects, have been redefined. The new statements use the dependency graph (see 3.1) to inform the agent itself and the other agents indicated by this graph.

For example, the rule base of *VentilationEvolution* uses a global object (`VentilatoryState`) which is also filtered by the rules of *TherapeuticActions*. Indeed, these two agents have dependency graphs to inform each other about the modification of this object.

The metarules fire the rules which deal with alarming situations preferentially to other fireable rules. They operate on objects such as *stop condition*, *time* and *state*. These objects are of course referenced and modified by the ATN to etablish the supervision link at the rule level.

Communication Module

This module follows the ABCL/1 model [Yonezawa 86] as realized in Actalk by Briot.

The only technical difficulty lies with the continuation of asynchronous messages [Agha 1986] when a reply is expected by the agent. We propose to endow such messages with a process which waits for the answer (class `MessageWithSenderAndReceiver`, subclass of the *Actalk* standard class `MessageWithSender` which appears in the last version of *Actalk* [Briot 1995]).

5.3 Asynchronous Agents Management

Each agent is mapped to a process which interprets its supervisor's ATN. The execution model of all agents is closely related to the process management by *Smalltalk-80*. In the latter, a scheduler (`Processor`) manages processes with a simple mechanism based on priority levels. It does not stop an active process i.e. it is not preemptive. Therefore, each agent must of its own will free the processor to give a chance to other agents (sociability) by inserting the expression `Processor yield`.

In order to accommodate several agents and/or modules on a single processor, we must simulate parallelism. To do so we choose a process allocation strategy at two levels: supervision module level (simulation of parallelism between agents) and perception, reasoning and communication modules level (simulation of the internal parallelism of the agent).

At the supervision module level, the agent suspends its activity after each transition. A message `self suspendBehavior` (which includes a `Processor yield`) is sent at the end of each transition by the ATN interpreter.

At the reasoning module level, process control is performed after each rule firing. At the communication and the perception modules, the process control is effected after each mail box reading and at the end of the method `scan`.

5.4 An Example of Real-Time Functioning

After each ATN transition, the agent suspends its activity to scan the state changes of the perception, reasoning and communication modules. Depending on the nature of change, the agent may suspend its current reasoning process to deal with the new perceived data or new received messages.

Suppose our *SignalProcessor* is scanning its sensors with period 8 seconds. It detects an alarming symptom at 12:07:31 pm. Then it sends an alarm signal (urgent message) to *VentilationAnalysis.* The latter suspends its reasoning process, sends a message to *SignalProcessor* to increase its sample frequency and then reactivates its reasoning. *SignalProcessor* increases its frequency, but the alarm is still present. After 4 seconds, i.e. at 12:07:35 pm, it sends another signal to *VentilationAnalysis*. When receiving this second signal, *VentilationAnalysis* starts an analysis, diagnoses a persistent apnea at: 12:07:35 pm and informs *TherapeuticActions*. This diagnosis takes some milliseconds. *TherapeuticActions* starts its actions at 12:07:36 and ends at 12:07:37 pm.

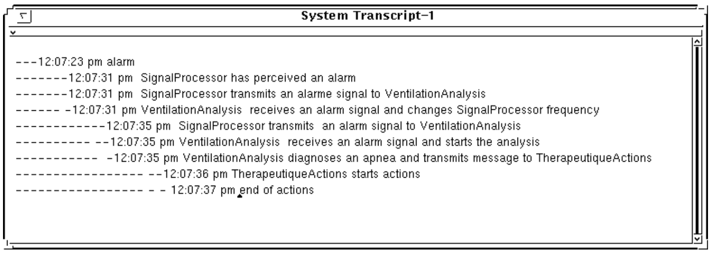

Fig. 9. Example of alarm processing (simulation data).

The total response time is the sum of transmission time, reaction time, parsing time and action time (see Fig. 10). The transmission time belongs to the interval [0, 8 sec] (8 sec is the period of *SignalProcessor*). The reaction time is the elapsed time between the perception of the alarm signal by *SignalProcessor* and the date at which the signal is taken into account by *VentilationAnalysis*. It corresponds to the run time of an ATN transition. The parsing time is the elapsed time between the date at which the message has been received by *VentilationAnalysis* and the date at which the signal has been parsed by its reasoning module. It corresponds to the run time of an ATN transition to deal with the urgent message sent by *SignalProcessor* and the firing of 4 metarules to activate the rules which parse the alarm signal. The action time is the elapsed time between the date at which the message has been received by *TherapeuticActions* and the end of the action on the ventilator. It corresponds to the run time of an ATN transition to deal with the urgent message sent by *VentilationAnalysis*, the activation of 2 metarules and one rule to act on the ventilator to change the therapy.

The total response time is about 14 seconds which is an excellent response time.

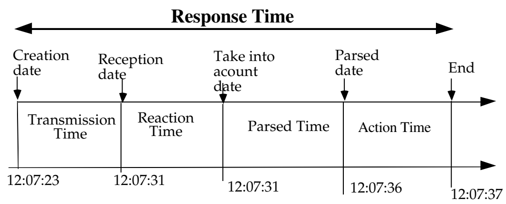

Fig. 10. Important dates in the life of an event

6 Discussion

As we saw earlier, realistic agent models require the combination of reactive and cognitive abilities. Using such hybrid models for real-time application needs to allow interactions between cognitive modules and reactive modules which have different time scales. The solution we propose has two main characteristics. First, we introduce a supervision module based on an ATN to control the interactions between modules that can be reactive or cognitive and to impose a global temporal sequencing. Second, the parallelism granularity of the time scale is based on the firing of a rule.

We have chosen an environment which combines actor, objects, production rules and processes. We benefit from the well known mechanisms of objects-oriented programming. Thus, by using the inheritance mechanism, the model can be easily extended. For example, to study communication between agents, we may integrate a specific module based on speech acts (such as [Bouron 1992]). Our system may be seen as a "framework" in the sense of object-oriented software engineering [Johnson 1994].

The experimental application describes here reuses the whole of the operational NéoGanesh system. We plan to run it in the same medical environment as NéoGanesh, and thus to obtain real-life performance measurements. Note that recent works (which rely, for the most part, on blackboards [Hayes-Roth et al. 1992; Quaglini et al. 1992; Sukuvaara et al. 1993]), so far lacks in clinical experiments.

Beside the medical application, we are using our platform to develop a manufacturing process simulator [Guessoum 1995] and an economic modeling [Guessoum and Durand 1995]. Other applications are currently under development.

Finally, our model uses well known distributed paradigms such as "access to resources" and "mail box messages communication". To avoid inconsistencies and conflicts, it uses dependency and anti-interference mechanisms. It would be interesting to explore other approaches such as organization, exchanging meta-level information and local planning.

Acknowledgment

The authors would like to thank **J-F Perrot** for his many useful criticisms and suggestions. His detailed comments have greatly improved the quality of this paper.

References

[Agha 1986] G. Agha. *Actors: a model of concurrent computation in distributed system*s. Cambridge MA (USA), MIT Press 1986.

[Barachini and Granec 1993] F. Barachini and R. Granec. *Productions systems for process control: advances and experiences*. Applied Artificial Intelligence 7: 301-316, 1993.

[Bouron 1992] T. Bouron. *What architecture for communication among computational agents?* Report LAFORIA 92/35, November 1992.

[Briot 1989] J-P. Briot. *Actalk: a testbed for classifying and designing actor languages in the Smalltalk-80 environment*. Proc. of ECOOP'89, Cook, p. 109-130, 1989.

[Briot 1995] J-P. Briot. *Modélisation et classification de langages de programmation concurrente à objets : l'expérience Actalk*. Proc. of LMO'95, Grenoble, 1995.

[Bussmann and Demazeau 1994] S. Bussmann and Y. Demazeau. *An agent model combining reactive and cognitive capabilities*. Proc. of IEEE International Conference on Intelligent Robots and Systems - IROS' 94, München, 1994.

[Charpillet and Boyer 1994] F. Charpillet and A. Boyer . *Incorporating AI techniques into predictable real-time systems: Reakt outcome*. Proc. of 14ème journées internationales Avignon'94, Avignon, p. 121-135, 1994.

[Charpillet and Théret 1994] F. Charpillet and P. Théret. *I.A. et temps réel*. Bulletin de l'AFIA 17: 19-42, 1994.

[Dojat and Pachet 1992] M. Dojat and F. Pachet. *NéoGanesh: an extensible Knowledge-Based System for the Control of Mechanical Ventilation*. 14th IEEE-EMBS, Paris, p. 920-921, 1992.

[Dojat et al. 1995] M. Dojat M. Harf, D. Touchard M. Laforest, H. Lemaire and L. Brochard. *Clinical evaluation of a knowledge-based system providing ventilatory management and decision for extubation during weaning from mechanical ventilatio*n. American Journal of Respiratory and Critical Care Medicine : to appear, 1995.

[Dojat and Sayettat 1996] M. Dojat and C. Sayettat. *A realistic model for temporal reasoning in real-time patient monitoring*. Applied Artificial Intelligence : to appear in vol. 10 n°2, 1996.

[Durfee et al. 1987] E. H. Durfee, V. Lesser, D. D. Corkill. *Coherent Cooperation Among Communicating Problem Solvers*. IEEE Transactions on Computers 36(11): 1275-1291, 1987.

[Erman et al. 1980] L. D. Erman, F. Hayes-Roth, V. Lesser. *The Hearsay II speech understanding system: integrating knowledge to resolve uncertainty*. ACM Computing Surveys 12 (2), 1980.

[Ferber 1995] J. Ferber. *Les systèmes multi-agents, vers une intelligence collective*. InterEditions, Paris, 1995.

[Ferber and Briot 1988] J. Ferber and J-P. Briot. *Design of concurrent language for distributed artificial intelligence*. Proc. of International Conference on Fifth Generation Computer Systems, Tokyo, Icot, p. 755-762, 1988.

[Ferguson 1992] I. A. Ferguson. *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*. PhD thesis, Clare Hall, University of Cambridge, UK.

[Gasser 1992] L. Gasser. *An Overview of DAI*. In Distributed Artificial Intelligence. N. M. Avouris and L. Gasser (eds.), Klewer Academic Publisher, Boston, 1992.

[Garvey and Lesser 1994] A. Garvey and V. Lesser. A survey of research in deliberative real-time artificial intelligence. Journal of Real-Time Systems 6 (3): 313-347, 1994.

[Guessoum 1994] Z. Guessoum. *Systèmes asynchrones de production*. Proc. of 2ème Journées IADSMA'94, Voiron, p. 169-180, 9-11 may, 1994.

[Guessoum 1995] Z. Guessoum. *A framework integrating an object-oriented multi-agent system and discrete event simulation*. Proc. of First LAAS ICCS'95, Beirut, p. 165-173, 1995.

[Guessoum and Durand 1995] Z. Guessoum and R. Durand. *Un système multi-agents pour modéliser l'évolution économique*. Report LAFORIA, to appear, 1995.

[Hayes-Roth 1985] B. Hayes-Roth. *Blackboard architecture for control*. Artificial Intelligence 26: 251-321, 1985.

[Hayes-Roth et al. 1992] B. Hayes-Roth, R. Washington, D. Ash, R. Hewett, A. Collinot, A. Vina and A. Seiveur. *Guardian: a prototype intelligent agent for intensive-care monitoring*. Artificial Intelligence in Medicine 4: 165-185, 1992.

[Hewitt 1977] C. Hewitt. *Viewing control structures as patterns of passing messages*. Artificial Intelligence 8 (3): 323-364, 1977.

[Ishida 1990] T. Ishida. *Methods and effectiveness of parallel rule firing*. IEEE Conf. on Artificial Intelligence Applications, Washington, p. 116-122, 1990.

[Johnson 1994] Johnson R. *How to Develop Frameworks*, Tutorial notes (10), 8th ECOOP'94, Bologna, 1994.

[Müller and Pischel 1994] J.P. Müller and M. Pischel. *Modelling reactive behaviour in vertically layered agent architectures.* Proc. of ECAI'94, p. 709-713, Amsterdam, (NL), 1994.

[Musliner et al. 1995] D. J. Musliner, J. A. Hendler, A. K. Agrawala, E. H. Durfee, J.K. Strosnider and C.J. Paul.. *The Challenge of Real-Time AI*. Computer (January): 58-66, 1995.

[Nii et al. 1989] H. P. Nii, N. Aiello, J. Rice. *Experiments on Cage and Poligon: measuring the performance of parallel blackboard systems*. Distributed Artificial Intelligence. L. Gasser and M. N. Huhns. San Mateo (Ca), Morgan Kaufmann. 2: 319-383, 1989.

[Pachet 1992] F. Pachet. *Représentation de connaissances par objets et règles : le système NéOpus*. PhD thesis, LAFORIA, Paris 6, 1992.

[Pachet and Perrot 1994] F. Pachet and J-F. Perrot. *Rule Firing with MetaRules*. Proc. of SEKE'94, Jurmala, Latvia, p. 322-329, 1994.

[Pachet 1995] F. Pachet. *On the Embeddability of Production Rules in Object-Oriented Languages*. Journal of Object-Oriented Programming 8(4): 19-24, 1995.

[Quaglini et al. 1992] S. Quaglini, R. Bellazi, C. Brzuini, M. Stefanelli and G. Barosi. *Hybrid Knowledge-Based Systems for Therapy Planning*. Artificial Intelligence in Medicine 4: 207-226, 1992.

[Shoham 1993] Y. Shoham. *Agent-Oriented Programming*. Artificial Intelligence 60: 139-159, 1993.

[Sukuvaara et al. 1993] T. I. Sukuvaara, M. E. Sydänmaa, H.O. Nieminen, A. Heikelä and E. M. Koski. *Object-Oriented Implementation of an Architecture for Patient Monitoring*. IEEE Transactions in Biology Engineering 12 (4): 69-81, 1993.

[Wolinski 1994] F. Wolinski. *RPC-Talk : une librairie RPC pour Smalltalk, Introduction à RPC et utilisations de RPC-Talk*. Report LAFORIA 94/26, November, 1994;

[Woods 1970] W. Woods. *Transition Network Grammar for Natural Language Analysis*. Communication of Association of Computing Machinery 13 (10): 591-606, 1970.

[Yokote and Tokoro 1987] Y. Yokote and M. Tokoro. *Experience and Evolution of Concurrent Smalltalk*. Proc. of OOPSLA'87, Orlando (USA), Special issue of SIGPLAN notices, ACM, p. 406-415, 1987.

[Yonezawa et al. 1986] A. Yonezawa, J-P. Briot, E. Shibayama. *Object-Oriented Concurrent Programming in ABCL/1*. Proc. of OOPSLA'86, Portland (USA), Special issue of SIGPLAN notices, ACM, p. 258-268, 1986.