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

Z. GUESSOUM*, M. DOJAT**
*LAFORIA-IBP, Université Paris 6, Boîte 169, 4, place de Jussieu, 75252 PARIS, FRANCE
**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 behaviour 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-Agents, 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 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 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. Indeed, each agent must integrate smoothly both reactive abilities, -to meet hard deadlines, and cognitive abilities, -to act rationally by using knowledge and -to reach a fixed goal when constraints are more relaxed.

MASs proposed in the literature rely on two approaches for communicating between agents: 1) the *blackboard* model initially introduced in Hearsay-II [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 proposed by [Hewitt 1977] on the basis of 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 origine and produce information without worrying about its fate, whereas in the actor approach, actors communicate with each other via message-passing. Actors may be considered as the basic element for building agents. The combination of the actor concept and the object paradigm leads to the notion of "*agent-oriented programming*" [Shoham 1993] . On this basis, we have designed a real-time agent model embedded in an object-oriented environment where properties such as encapsulation and inheritance are respected.

The purpose of this paper is to present our model and to show how it was used to design a real-time prototype for Intensive Care Unit (ICU) patient monitoring. Section 2 presents the proposed real-time agent model and details the use of an ATN to adapt the agent behaviour to asynchronous changes in its environment. Section 3 describes the behaviour of this model in the bosom of a society of agents. Section 4 presents the tools we used: *NéOpus* and *Actalk*. In Section 5, we report the application of our model to patient monitoring. Finally, we discuss the advantages of our approach to design real-time MASs.

2 OUR REAL-TIME AGENT MODEL

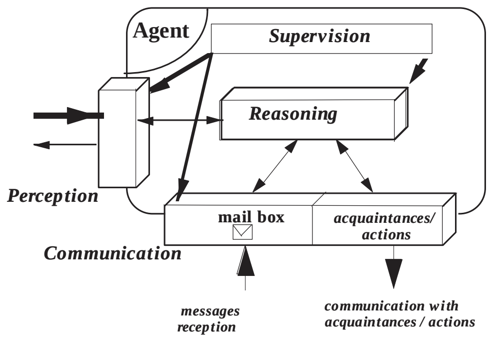


Figure 1: Agent model

Our agent model (see Figure 1) relies on a first layer including three modules: the *communication* module, the *reasoning* module and the *perception* module. These three modules act asynchronously and concurrently. A high-level *supervision* module allows the agent to reason on the states of the other modules. The reasoning and the supervision modules represent respectively the *decisional* and *meta decisional* levels. The following sub-sections detail each module.

2.1 The Supervision Module

This module has two functions. Firstly, it supervises the interactions between the other modules to control their behaviour. Secondly, it synchronizes the execution of concurrent actions to avoid inconsistencies. It relies on two notions: *states* and *transitions*. States define the context in which the events introducing changes occur and transitions define the agent's response to these events.

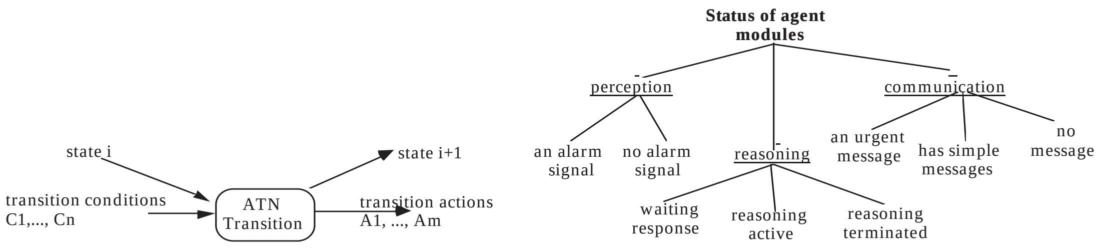


Figure 2: The supervision module.
Different status correspond to each module. Each combination of these status or their negation defines an agent state.

The supervision module is represented as an ATN [Woods 70] which defines the set of possible transitions linking different states. Each transition links input and output states, the various signals received by the agent modules represent the conditions of transition (see Figure 2) and the actions of transition change the status of the various modules (activate reasoning, terminate reasoning, ...). The agent's state is a combination of the status of each module. When these conditions are verified, the transition actions are executed and the agent's state is modified. The ATN is a declarative and deterministic representation of agent's bahaviour.

2.2 The Perception Module

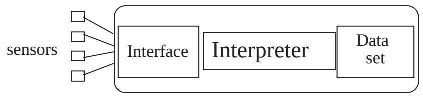


Figure 3: The perception module

The perception module manages the interactions between the agent and its environment. It monitors sensors, translates and filters sensed data according to the instructions of the reasoning or supervision modules (see Figure 3). It may group information concerning the same phenomenon to facilitate interpretation. The data set obtained is essentially used by reasoning.

2.3 The Reasoning Module

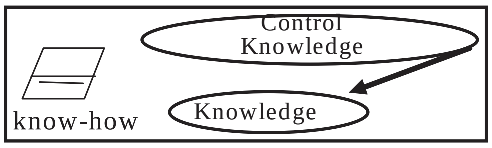


Figure 4: The reasoning module

This module defines the **adapted** response depending on the messages transmitted by the communication module, or on the changes detected by the perception module. This response is a function of its *know-how* (operative competences) and its *knowledge* (cognitive competences). Thus, this module includes an asynchronous production system [Guessoum 1994]. This system mainly comprises: (1) a knowledge base which includes objects describing the agent's environment and rules representing suitable operations over these objects; (2) an inference engine which includes a mechanism to avoid inconsistencies due to the asynchronous execution of several rule-bases; and (3) a meta-knowledge base which provides a declarative representation of the control of reasoning.

2.4 The Communication Module

The communication module (see Figure 5) 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 accord them. It sends messages to its **acquaintances** with various protocols (selective, with acknowledgement, synchronous, asynchronous). The protocol is determined by the reasoning module.

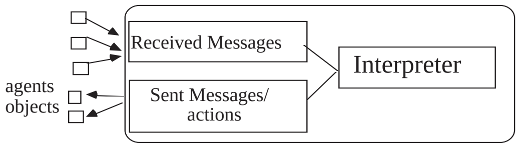


Figure 5: The communication module

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

2.5 Real-Time Characteristics

The proposed model provides real-time characteristics at different levels:

• **Reactivity**: Via an asynchronous perception module, the agent detects real-time alarming situations. This feature contributes to improve the agent's response to changes.

• **Adaptability:** 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.

• **Real-Time Reasoning:** Our agent model uses a first-order forward chaining inference engine called NéOpus [Pachet 1995] based on the Rete Algorithm to compile rule-bases. The original key mechanisms of NéOpus include a declarative specification of control with meta rules. The execution control of a rule-base is seen as a problem which requires some forms of knowledge. Thus specific declarative control knowledge may be introduced to reason on critical situations [Dojat and Pachet 1992].

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 coordinating 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 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 [Gasser 1992]. This is the case for our model. The granularity of our agents is high: each agent possesses its own rule-base. The agent's rules are fired sequentially and the internal dependency and internal interference do not have to be considered. Thus the dependency between agents is low. To avoid inconsistencies between asynchronous agents reasoning, we exploit two graphs namely a dependency graph and an inference graph initally introduced by [Ishida 90].

3.1 Dependency mechanism

We distinguish two kinds of objects: *local objects* used by a single agent, and *global objects* used by several agents. The principle of dependency mechanism may be defined as follows:

• each agent is provided with a dependency graph giving for each global object the list of agents which use it.

• after each global-object modification, the agent informs the other agents indicated by its dependency graph.

The dependency graph is used by the communication module to inform the other agents about the modification, creation or removal of global objects. It is updated progressively when rules are triggered: if a global object is removed by the rule actions, it is also automatically removed from the dependency graph and a message is sent to the other agents to update their graphs.

3.2 Interference mechanism

Interference exists among two rules R1 and R2 if there is a global object O such as R1 modifies O and R2 modifies O, R1 filters O and R2 modifies O, or R2 filters O and R1 modifies O.

The principle of interference mechanism may be defined as follows:

• each agent is provided with an interference graph, giving for each rule the list of agents that have rules interfering with this rule.

• each agent is provided with a model of other agents giving for each agent the list of global objects in use. This model is introduced to avoid the synchronization messages used by Ishida.

• we enrich the inference engine cycle of two steps:

- lock: before the rule firing, the agent tests if the global objects, that the selected rule uses, are not locked by the other agents. In this case, the agent the rule is triggered, otherwise it selects another fireable rule.
- unlock: This action is executed after each rule firing, it cancels the effects of the action *lock*.

4 INTEGRATION OF OUR MODEL IN AN ASYNCHRONOUS OBJECT ENVIRONMENT

We have opted for an environment which combines object-oriented programming and production rules. We have used *Smalltalk-80* and *Actalk* [Briot 1989] which introduces the notion of actor in *Smalltalk-80,* the foundation stone of our model. The reasoning module uses *NéOpus* [Pachet 1995] , a first order inference engine completely embedded in Smalltalk-80. We have extended these three basic components to implement our model and to propose a platform to build real-time MASs.

4.1 Actalk

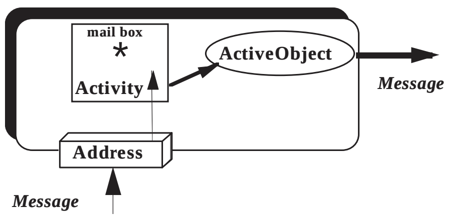


Figure 6: the actor model in *Actalk*
The figure shows the three components of an actor: Address, Activity and ActiveObject.

Actalk allows to join harmoniously objects and actors from an initial kernel. In *Actalk*, an actor is composed of three objects (see Figure 6):

• an object (Address) describes the address (mail box) of an active object. It defines the way message transmissions will be interpreted.

• an object (Activity) describes the internal activity of the active object. It provides autonomy to the actor. It defines a process which removes continuously the messages of the mail box and sends up their interpretation by the active object.

• an object (ActiveObject) describes the actor behaviour. It computes the messages.

Asynchronism, a basic principle of actor languages, is implemented by **enqueuing** the received messages into the mail box. It dissociates the message reception from its interpretation.

4.2 NéOpus

NéOpus introduces rule-based programming into object-oriented programming Smalltalk-80. The three mechanisms specific to this integration are: (1) natural typing, a mechanism which allows the pattern matcher to consider direct instances of classe as well as instances of its subclasses to be mached by the variables of a rule; (2) rule base inheritance: a rule-base inherits of rules of its super bases that can be redefined and/or extended and (3) declarative specification of control with meta rules. Meta rules are similar to rules and are assembled in metabases. A metabase controls the rule firing of the associated rule-base (see Figure 7).

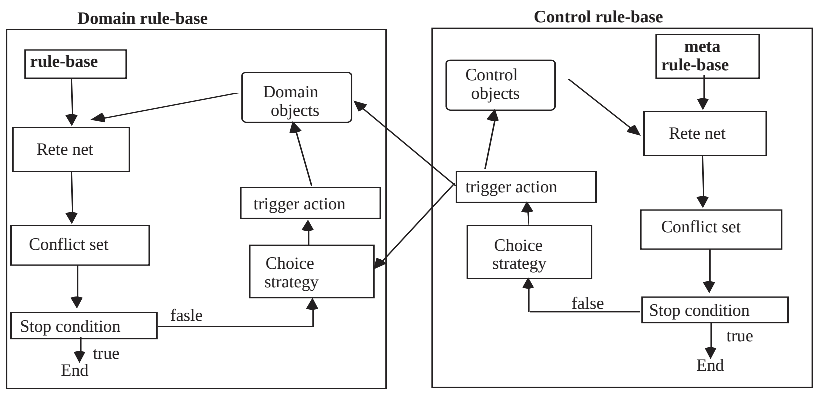


Figure 7: The control with meta rules in NéOpus.
This figure highlights the organisational similarity between domain knowledge and control knowledge. The fired meta rules act on the firing strategy of rules, on the stop condition and on the objects moving in the associate rule-base net.

5 FROM ACTALK OBJECTS TO AGENTS

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

5.1 Application: Artificial Ventilation Control

Briefly, 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 used at the hospital Henri Mondor (Créteil) [Dojat et al. 1995]. An extension of the current system based on a distributed architecture using our agent model, should be interesting to increase the system reactivity and to incorporate new distributed medical expertises. The recent architecture for patient monitoring relies, for the most existing systems, on blackboard [Hayes-Roth et al. 1992; Quaglini et al. 1992; Sukuvaara et al. 1993]. We explore the use of actors paradigm for patient monitoring.

5.2 Agents

BasicAgent, root of the classes describing our agent behaviour, is encapsulated according to the *Actalk* principle to be transformed in active object. All the agents have the same general structure but they differ in 1) their sensory-driving layer: perception and communication modules; 2) their behaviour: the know-how, the domain and control knowledge. Agents are complex entities and the taxonomy of agents (see Figure 8) may be realized according to several criteria, such as type of the sensory-driving structures, know-how, type of cognitive functions, ...

```
ActiveObject
    BasicAgent
            SignalProcessor
            CommunicatingAgent
                    Clinician
                    Intensivist
            PerceivingAgent
                    CommunicatingReasoningAgent
                            VentilationEvolution
                            TherapyPlanning
                            TherapeuticActions
                    PerceivingReasoningAgent
                            CommunicatingPerceivingReassoningAgent
                                    VentilationAnalysis
```

Figure 8: Excerpt from the agents taxonomy used for articificial ventilation control application

In the monitoring of artificial ventilation application, the agents complexity is variable. For example, the agent *SignalProcessor* has a simple bahaviour to process data acquisition. At the opposite,*VentilationEvolution* agent has a complex bahaviour to appreciate the time-course of patient's ventialtion.

5.2.1 Supervision Module

The supervision module represents the agent kernel. It controls the other modules by using an ATN. To implement this module, we have reused the *Actalk* kernel. We have defined two classes: AgentActivity (subclass of Activity) and BasicAgent (subclass of ActiveObject).

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, has been redefined in AgentActivity. The new role of this process is to interpret the ATN.

```
!BasicAgent methodsFor: 'atn interpreter'!
interpreter
|state|
state:= atn initialState.
state = atn finalState whileFalse:[state := atn transitionAt: state]

!AgentActivity methodsFor: 'process management'!
setProcess
^process := [[true] whileTrue: [self activeObject interpreter]] newProcess
```

The ATN of *VentilationEvolution* manages different states of the communication and reasoning modules. For example, in state 1, the condition "no message" drives to the action "wait" and to the persistence in the state 1. The condition “an urgent message” drives to the action “read mailbox” whatever the state may be. This agent gives priority to urgent messages such as alarms.

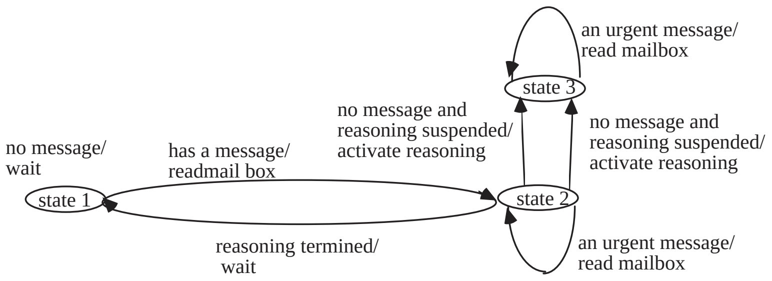


Figure 9: *VentilationEvolution* ATN

5.2.3 Perception Module

It is mainly characterized by a list of sensors and an instance method scan. This method allows to interpret the sensors data, to compare them with the old one and to transmit new data to the reasoning module.

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

5.2.3 Reasoning Module

The reasoning module represents the agent know-how and knowledge. The first ones are represented by a list of methods; the second by a NéOpus rule-base. The latter is subclass of LocalRuleSet if the used objects are local or DependInterfRuleSet in the other case. The latter implements the dependency and interference mechanisms.

For example, the reasoning module of *VentilationEvolution* agent has a rule-base (subclass of DependInterfNeOpusRuleSet) and a metabase. The rule-base uses a global object (VentilatoryState) which is also filtered by the rules of *TherapeuticActions* agent. Indeed, these two agents have dependency graphs to inform each other about the modification of this object.

```
!VentilationEvolutionRules methodsFor: 'continuity'!
partialContinuity

|VentilatoryState s1 s2 s3 |

s3 persistent.
s1 sameAs: s3.
s2 contradicts: s3.
s2 between:e1 and: s3.
s3 durationInExpertise >1.
s2 dureeInExpertise  <= 1.

actions
s1 validity: (DiscontinuousInterval with: s1 occurenceDate with: s3 occurenceDate).
s2 remove.s3 remove.
s1 increaseOf: s3 duration.
```

Figure 10: Example of a rule used by*VentilationEvoultion* agent that matches global objects

5.2.4 Communication Module

The class representing this module derives from the class Address of *Actalk*. It manages explicitly the various received or sent messages. It is founded on the ABCL/1 model [Yonezawa 86]. The main used message types are the asynchronous message sending, the synchronous message sending and the broadcasting message sending.

The synchronous messages are implemented as active objects. Their activity consists in transmitting the message, waiting for the result and then forwarding it to the sender.

```
!MessageWithSenderAndReceiver methodsFor: 'synchronous send'!
synchronousSendTo: anAgent
anAgent synchronousSend: aMessage.
self process: [resultat isNil whileTrue: [self wait]] new process.
sender forwardResult: self
```

5.3 Asynchronous Agents Management

Each agent is mapped to a process which interprets the corresponding ATN. The execution model of all agents is closely related to the processes management by the *Smalltalk* virtual machine. 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 free voluntarily the processor to give a chance to other agents (sociability) by inserting the expression Processor yield.

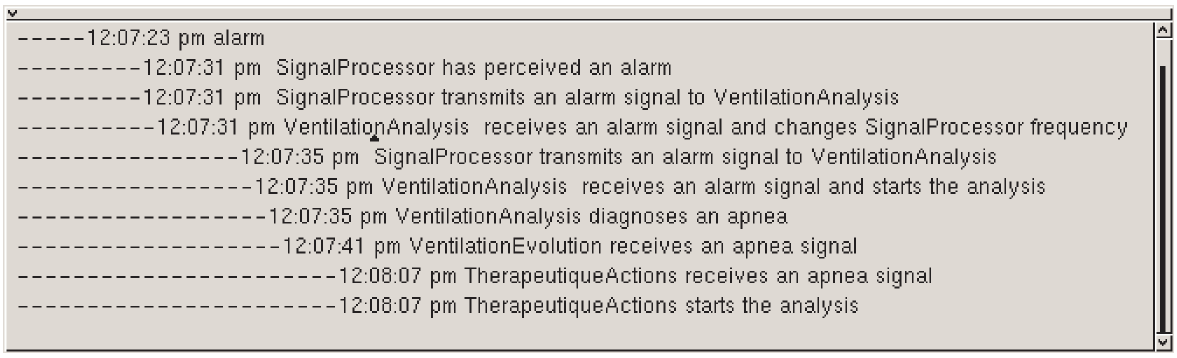
To simulate parallelism we chose a process allocation control at two levels: supervision module level (simulation of parallelism between agents) and perception, reasoning and communication modules level (simulation of the internal parallelism of the agent).

```
!BasicAgent methodsFor: 'atn interpreter'!
interpreter
|state|
state:= atn initialState.
state= atn finalState whileFalse:
                [state:= atn transitionAt: state. self suspendBahaviour].
```

At the supervision module level, the agent suspends its activity after each transition. A message suspendBahaviour (including Processor yield) is performed at the end of each transition by the ATN interpreter.

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

5.4 Real-Time Functioning



```
-----12:07:23 pm alarm
---------12:07:31 pm  SignalProcessor has perceived an alarm
---------12:07:31 pm  SignalProcessor transmits an alarm signal to VentilationAnalysis
----------12:07:31 pm VentilationAnalysis  receives an alarm signal and changes SignalProcessor frequency
----------------12:07:35 pm  SignalProcessor transmits an alarm signal to VentilationAnalysis
-----------------12:07:35 pm VentilationAnalysis  receives an alarm signal and starts the analysis
------------------12:07:35 pm VentilationAnalysis diagnoses an apnea
-------------------12:07:41 pm VentilationEvolution receives an apnea signal
-----------------------12:08:07 pm TherapeutiqueActions receives an apnea signal
-----------------------12:08:07 pm TherapeutiqueActions starts the analysis
```

Figure 11: Example of alarm processing.

Figure 11 describes a situation where *SignalProcessor* perceives an alarm signal. The signal (urgent message) is then forwarded to *VentilationAnalysis* which suspends its reasoning process, and 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. It sends another signal to *VentilationAnalysis*. When receiving this second signal, *VentilationAnalysis* starts an analysis, diagnoses a persistent apnea and informs *TherapeuticActions*. To analyse and process in real-time an alarm signal, *TherapeuticActions* needs two kinds of knowledge: meta rules and rules. The meta rules used fire the rules which deal with alarming situations preferentially to other fireable rules.

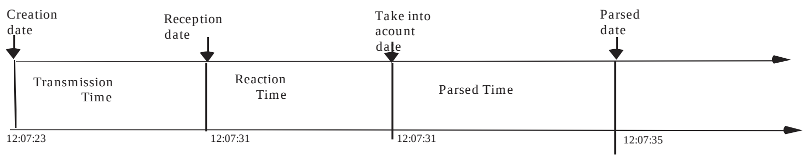


Figure 12: Important dates in the life of an event

Figure 12 gives an example of the different steps for processing an alarm signal. The transmission time belongs to the interval [0, 8 sec] (8 sec is the frequency 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 parsed 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*, the firing of 4 meta rules to activate the rules which process the alarm situation. Then the appropriate rule is triggered. One ATN transition and the activation of 5 rules (4 metarules and one rule) are required to act on the ventilator to change the therapy.

6 DISCUSSION

In this paper we have proposed a real-time agent model. We have used an object-oriented programming and mixed two techniques: production systems and actors and have introduced mechanisms to guarantee real-time response. In addition to the medical application, the model has been used to develop a manufacturing process simulator [Guessoum 1995] and is currently used to develop an economical modelling system.

6.1 Characteristics of the model

The proposed model has the following characteristics:

• Ability to monitor in real-time information provided by the environment.

• Ability to adapt its behaviour by using the supervision module on an ATN. This allows to describe declaratively and concisely the dynamic changes of the behaviour to respond in real time to changes in the environment.

• Internal and external consistencies: - the internal consistency is ensured by the supervision module and - the external consistency is ensured by two management mechanisms of dependency and interference graphs.

• We have chosen a granularity at the rules level that seems to be acceptable in several industrial applications [Barachini and Grenec 1993].

• The control is distributed between agents: each agent owns an ATN and a metabase for the control of its reasoning.

• Contrary to blackboard architectures for control [Hayes-Roth 1985] which are based on an opportunistic control, our control architecture always prefers meta rules to rules.

6.2 Gains of the Object-Oriented Programming

• By using the inheritance property, we can define several coexisting actors with various abilities: reactive actors or cognitive agents.

• We have extended the discrete event simulation kernel of Smalltalk to study the temporal behaviour of MASs [Guessoum 1995] in a simulated real world.

• Various concurrent object programming mechanisms can be tested to define the mechanism (or the combination of mechanisms) suitable for a specific application.

• Each actor is an autonomous entity by means of the activate process for its messages management or its ATN interpretation.

6.3 Extensions

• The implemented agents use dependency and interference graphs to avoid inconsistency and conflicts. It seems interesting to explore the other approaches such as organization, exchanging meta-level information and local planning.

•Our model can be easily extended. For example, to study communication between agents, we may integrate a specific module based on speech acts such as the module presented by [Bouron 1992].

•Our plate-form has been applied to the medical application detailled in this paper. We use it to develop a manufacturing process simulator [Guessoum 95] and an economical modelling system is under development.

7 BIBLIOGRAPHY

[Agha 1986] **G. Agha**. Actors: a model of concurrent computation in distributed systems. 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 19992] **T. BOURON.** What architecture for communicatios among computational agents. Report LAFORIA 92/35, novembre 1992;

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

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

[Charpillet and Boyer 1994] **F. Charpillet and A. Boyer** . *Incorporating AI techniques into predictable real-time systems: Reakt outcome*. 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 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Paris, pp. 920-921, 1992.

[Dojat et al. 1995] **M. Dojat et al.** *Clinical evaluation of a knowledge-based system providing ventilatory management and decision for extubation during weaning from mechanical ventilation*. 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] **Edmund H. Durfee, Victor R. Lesser, Daniel D. Corkill**. Coherent Cooperation Among Communicationg Problem Solvers. IEEE Transactions on Computers 36(11): 1275-1291 ,1987.

[Erman et al. 1980] **Erman L D, 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*. InterEdition, France, 1995.

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

[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*. Journées Intelligence Artificielle Distribuée et Systèmes Multi-Agents, Voiron, 9-11 mai, 1994.

[Guessoum 1995] **Z. Guessoum**. *A framework integrating an object-oriented multi-agent system and discrete event simulation*. First LAAS International Conference, Beirut, pp. 165-173, 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, et al.** *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.

[Musliner et al. 1995] **D. J. Musliner, et al.** The Challenge of Real-TIme AI. Computer (January): 58-66, 1995.

[Pachet 1995] **Pachet F**. *On the embeddability of production rules in object-oriented languages*. Journal of Object-Oriented Programming (june), 1995.

[Quaglini et al. 1992] **S. Quaglini, et al.** *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, et al.** Object-oriented implementation of an architecture for patient monitoring. IEEE Transactions in Biology Engineering 12 (4): 69-81, 1993.

[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*. Object-Oriented Programming Systems, Languages and Applications, 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*. Object-Oriented Programming Systems, Languages and Applications, Portland (USA), Special issue of SIGPLAN notices, ACM, p. 258-268,1986.