A Hybrid Agent Model: a Reactive and Cognitive Behavior

Zahia Guessoum
Equipe RTCD, LAFORIA-IBP, Université Paris 6, Boîte 169, 4 Place de Jussieu, F-75252 PARIS
guessoum@laforia.ibp.fr
http://www-laforia.ibp.fr/~guessoum

Abstract: To model complex systems, software agents need to combine cognitive abilities to reason about complex situations, and reactive abilities to meet hard deadlines. We propose an operational hybrid agent model which mixes well known paradigms (objects, actors, production rules and ATN) and real-time performances. This agent model combines cognitive and reactive modules. The cognitive module uses a rule-based system and is provided with a synchronization mechanism to avoid the possible inconsistencies of the asynchronous execution of several rule bases. The reactive asynchronous perception and communication modules allow the agent to dynamically adapt its behavior to changes in the environment. To manage the interactions between these fundamentally different modules (reactive and cognitive), we use an ATN-based module.

Key Words: Agents, Reactive, Cognitive, Hybrid, Object-Oriented, Actors, Production Rules, ATN.

1 Introduction

Several agent models have been proposed. Two main approaches can be distinguished: cognitive and reactive (see [20, 7]). 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.

Neither a completely reactive nor a completely cognitive approach is suitable for building complete solutions for real-life applications. Hybrid models [8, 3] 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.

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 schedules 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.

In our model, each reasoning module owns a 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 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.

The purpose of this paper is to present our hybrid agent model and its implementation. Section 2 presents the main Smalltalk-based tools of our implementation: Actalk and NéOpus. Section 3 describes our hybrid agent model. It presents the individual modules, the control mechanisms which are used and the real-time characteristics of this model. In Section 4, we report on the implementation with a concurrent object language. In Section 5, we report on the implementation with a concurrent object language. In Section 5, we report on the application of our model to patient monitoring and to economic agents evolution modeling. Finally, we discuss the advantages of our model to design and implement multi-agent systems (MASs).
2 Technical Context

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 [11]) for observing the temporal behavior of the system under simulated real-time conditions. We have used RPC-talk [19], 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 [2] 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 [18], 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

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 actor activity. 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.

2.2 NéOpus

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 [17]. 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.2).

![Fig. 2 Control with metarules in NéOpus.](image)

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 [5]. We have adapted some of them to our needs by changing a few individual metarules.

3 A Hybrid Agent Model

As underlined in the introduction, our model relies on two layers made up of modules. The first layer modules describe the agent activities. The second layer includes only one module: the supervision module. The latter manages the interaction between the first layer modules.

In this section, we describe the modules of the two layers, the mechanisms control we used and the real-time characteristics of this model.
3.1 The first layer modules

In this section, we describe three examples of modules of the first layer: the perception, the reasoning and the communication/action modules.

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. It may package information concerning the same phenomenon to facilitate interpretation. The data set obtained is used mainly by the reasoning module.

The Reasoning 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 [10]. 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.

The Communication/Action 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).

3.2 The Supervision Module

This module allows the agent to schedule its internal activities according to its world state. It 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. The ATN is a synthetic and deterministic representation of the agent's behavior.

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 (reasoning terminated, has urgent message, ...) and the actions of transition (activate reasoning, terminate reasoning, ...) change the state of the various modules. When these conditions are verified, the transition actions are executed and the agent's state is modified.

3.3 Control Mechanisms

Intelligent agents with high degree of autonomy should perform well complex tasks in a dynamic environment for extended periods of time. Here we propose a control mechanisms at two levels. At the agent level, we propose a self-control to achieve autonomy and to coordinate its own activities. At the multi-agent level, we propose a coordination mechanism to avoid redundant actions and conflicts between agent.

3.3.1 Self-control

The self-control is managed at three levels (see Fig. 4): 1) at the supervision module level, an ATN allows to specify the agents behavior in accordance to the observed internal states; 2) at the reasoning module level, a metalevel architecture specifies declaratively the agent reasoning strategy and 3) at the internal parallelism level, a management of the processes associated to the asynchronous concurrent modules allows to specify the available resources allocation.

3.3.1.1 Supervision Module level: Adaptive Control

The supervision module allows the agent to dynamically adapt its behavior the its universe changes. The adaptation corresponds to a new organization mode of the agent in its decisional relations, interactions with its environment or relations with other agents. The agent is able to control its abilities (communicational, behavioral, ...) according to its current environment.

3.3.1.2 Reasoning Module level: Intelligent Control

One of our main hypothesis is: The reasoning control
strategies can be separated from the domain knowledge, but they must be supervised by the agent itself. Unlike blackboard approaches [16] our agents encapsulate the two kinds of knowledge: domain and control knowledge.

The system NéOpus we use to represent the agent reasoning capacities provides a declarative control architecture. The latter is founded on the following reflexivity principle: The control of a rule-base execution is fully supervised by an other metalevel rule base called metabase. It provides a declarative specification of control with metarules. With each rule base there may be associated a metabase which controls the firing of its rules. As system complexity increases, such control mechanism will need, for instances, state estimators. Thus, control objects are used to give reasoning these states estimators. They are used by the metarules and can be also be accessed by the supervisor's ATN, thus establishing link at the reasoning process.

3.3.1 Internal Parallelism level: Process Management

Each agent is mapped to a process. The execution model of all agents is closely related to the process management by Smalltalk-80. In the latter, a scheduler (Processor) does not stop an active process i.e. it is not preemptive. Therefore, each agent must of its own free the processor to give a chance to other agents 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: at the supervision module level (simulation of parallelism between agents) and at the 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. 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.

3.3.2 Coordination mechanism

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. [4, 9] describe the mechanisms that improve the network coherence: organization, exchanging metalevel information, local and multi-agent planning, and explicit analysis and synchronization.

With a synchronization mechanism as proposed by Ishida in [15], each agent protects itself against conflicts and 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 L. Gasser. 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.

3.3.2.1 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.

3.3.2.2 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:

* **updateList**: 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 test 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.

### 3.4 Real-Time Characteristics of the proposed Agent Model

The proposed hybrid agent model has the main characteristics of a real-time system: 1) Reactivity and adaptability and 2) Anytime reasoning to guarantee a solution in a bounded time.

#### 3.4.1 Reactivity and Adaptability

Reactivity and adaptability are crucial characteristics in the complex process control such artificial ventilation monitoring. The main problem to solve in such systems is the real-time reactivity to asynchronous changes of the environment and the adaptability of the agent behavior to the state changes provoked by these changes.

The characteristics of the proposed model which gives it reactivity and adaptability are the following:

- the use of asynchronous perception module allows to supervise in real-time a large information sources of the agent environment,
- the use of asynchronous communication module allows to respond in real-time to the other agents requests,
- the use of an ATN-based supervision module allows the agent to observe and to adapt its behavior to its universe changes as indicated by its asynchronous modules.

These characteristics give limited time reactivity to our agents. 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 [1].

#### 3.4.2 Anytime Reasoning

The anytime reasoning is a promising solution based on the anytime algorithms [21]. Thus, we have provided our agents with anytime reasoning abilities to guarantee a fixed time-consuming. This reasoning assumes that there are several solutions with different qualities each of which represents approximate view of the final solution [16].

![Schematic structure of the progressive reasoning](image1)

**Fig. 5** Schematic structure of the progressive reasoning

![A single execution cycle of the real-time metabase](image2)

**Fig. 6** A single execution cycle of the real-time metabase

The proposed technique is founded on the anytime reasoning on the one hand and on two main features of the rule-based system we use on the other hand. These main features are: 1) the declarative reasoning control and 2) the inheritance of rule bases. We propose to decompose each agent rule base in several packages. The first package provides fast solution but often of worst quality. The addition of the other packages and the context enrichment improves the solution quality (see Fig. 5).

To implement this progressive reasoning, we have defined and implemented a metabase which uses several deliberation stages where the solution quality is improved progressively (see Fig. 5).

To update the deadline, we have redefined the rule firing method. The new one allows to update, at each cycle, the remaining time.

### 4 From Concurrent Objects to Agents

To implement the proposed autonomous agent model,
we use Actalk. In the realized environment, an agent is an actor composed by the following objects (see Fig. 7):

- An object (instance of the class AgentActivity subclass of Activity) which describes the agent activity.
- An object (instance of the class BasicAgent subclass of ActiveObject) which describe the agent ATN structure.
- A set of objects which describe the different modules describing the agent behavior.

Fig. 7 Different objects of the implemented agent

The two first objects implement the supervision module of the agent and the last subset implements its first layer. The class describing the communication module derives from the class Address of Actalk.

In Actalk, an object Activity manages the messages which are interpreted by another object ActiveObject. In our model, the agent activity is described by an ATN which schedules the different activities (perception, reasoning and communication/action). So, we have redefined the instance method body used by createProcess which creates a process to take out continuously the messages present in the actor mailbox.

```smalltalk
: Activity methodsFor: 'activity setting'
body
[true] whileTrue: [self acceptNextMessage]
createProcess
^[self body] newProcess
```

The new function of this process is to interpret the agent ATN.

```smalltalk
:AgentActivity methodsFor: 'activity setting'
body
[true] whileTrue: [self atnInterpreter]
```

The use of an object-oriented concurrent language allows to benefice of the inheritance mechanism. To realize a social agents hierarchy, we have reused this mechanism. The implemented agents can have a simple behavior (reactive) or a complex one (cognitive or hybrid). The agents have the same general structures, but they differ in:

- Their sensor-driving layer, they don't have all a perception module and a communication module.
- The behavior, the know-how, the domain and control knowledge they own.

Several agent classes belonging to different complexity levels have been defined (see Fig. 8).

Fig. 8 Classes describing agent structures

At each class is associated an ATN to define the behavior type. Each class implements one or several agent creation functions.

5 Applications

To validate the operational environment (DIMA) which is based on the proposed hybrid agent model, we have implemented three applications: 1) manufacturing process simulator [11]; 2) NéoGanDi [12]: a multi-agent system to control mechanical ventilation and 3) Meveco [13]: a multi-agent system to model economic agents evolution.

In this section, we briefly describe NéoGanDi and Meveco.

5.1 NéoGanDi

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-CO2 pressure (PCO2)), 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 [6]. In alarming situations such as hypoverntilation 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) [6]. The extension based on a distributed architecture using our agent model, aims at increasing the system reactivity and incorporating additional distributed medical expertise.

*Fig. 9 Overview of the mechanical ventilation agents*  

*NéoGanDi* is composed by 7 agents (see Fig. 9). All these "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. For example, agent *SignalProcessor* has only a simple behavior to process data acquisition. Whereas, agent *Classifier* exhibits a complex behavior to appreciate the time-course of the patient’s ventilation.

This experimental application 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 [14]), so far lacks in clinical experiments.

In order to evaluate this new version, we have realized 1000 experimentations (on a station SUN SPARC 10). The results show that the total response time of the expert system is better than the total response time of a not distributed system (*NéoGanDi* implemented on only one machine). This is principally due to the use of asynchronous messages (use of a mail box).

<table>
<thead>
<tr>
<th></th>
<th>Response Time (Ordinary situation)</th>
<th>Response Time (Alarming situation)</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>NéoGanesh</em></td>
<td>4232 ms s</td>
<td>4067 ms</td>
</tr>
<tr>
<td><em>NéoGanDi</em></td>
<td>4738 ms</td>
<td>3475 ms</td>
</tr>
<tr>
<td>(1 machine)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>NéoGanDi</em></td>
<td>4240 ms</td>
<td>2156 ms</td>
</tr>
<tr>
<td>(2 machines)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Tab. 1 : Comparison of *NéoGanesh* and the two versions of *NéoGanDi*

However, the total response time of *NéoGanesh* in alarming situation is improved of ~ 57 % when using the distributed version of *NéoGanDi*. This improvement is not only due to the use of two machines, it is also due to the use of our model. For example, the use of an ATN has significantly improved the time reactivity of the system.

### 5.2 Meveco

Standard economic literature is centered around the paradigm of homogeneity. Investors, consumers or traders are supposed to be identical in beliefs. Such homogeneity deprives economic agents of their essential features: these agents have different interpretations of the surrounding world.

Meveco is a multi-agent system to model the evolution of economic agents which are modeled as autonomous, cognitive, adaptive and heterogeneous agents. Unlike mathematical and expert systems approaches, ours allows a comparative and incremental evaluation of their relevance to the observed phenomena and their validity. It allows also to determine the most relevant strategy.

*Fig. 10 Overview of the economic agents*

These agents combine cognitive abilities to reason about competition, and reactive abilities to interpret the surrounding world. Each agent owns a model of the other agents. It scans the market to have some data (for example price and quality of the different products) and builds the model of the others.

The implemented system have provided results which have allowed to demonstrate the importance of the existence of heterogeneous, intelligent economic agents for the robustness of the system to abrupt external perturbations.

### 6 Conclusion

As we saw earlier, realistic agent models require the combination of reactive and cognitive abilities. Using such hybrid models for real-life applications 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 actors, objects, production rules and processes. We benefit from the well known mechanisms of object-oriented programming. Thus, by using the inheritance mechanism, the model can be easily extended. For example, to study communication between agents, we may define new subclasses to integrate a specific module based on speech acts.

Finally, in this paper, we have limited our study of real-time aspects to the agent level. Real-time agents are necessary to most real-life applications but they are not sufficient to real-time applications. It seems very interesting to study how the agents society cooperate to solve a global problem in real-time.

7 References


