# Towards Timed Automata and Multi-Agent Systems

G. Hutzler, H. Klaudel and D.Y. Wang

LaMI, UMR 8042, Université d'Evry-Val d'Essonne/CNRS
523, Place des Terrasses 91000 Evry, France
{name}@lami.univ-evry.fr

**Abstract.** The design of reactive systems must comply with logical correctness (the system does what it is supposed to do) and timeliness (the system has to satisfy a set of temporal constraints) criteria. In this paper, we propose a global approach for the design of adaptive reactive systems, i.e., systems that dynamically adapt their architecture depending on the context. We use the timed automata formalism for the design of the agents' behavior. This allows evaluating beforehand the properties of the system (regarding logical correctness and timeliness), thanks to model-checking and simulation techniques. This model is enhanced with tools that we developed for the automatic generation of code, allowing to produce very quickly a running multi-agent prototype satisfying the properties of the model.
**Keywords.** agent oriented software engineering, formal models, agent oriented programming

## 1 Introduction

Real-time reactive systems are defined through their capability to continuously react to the environment while respecting some time constraints. In a limited amount of time, the system has to acquire and process data and events that characterize its temporal evolution, make appropriate decisions and produce actions. Thus, the robustness of the system relies on its capability to present appropriate outputs (logical correctness) at an appropriate date (timeliness). Such applications are often critical. Their hardware and software architectures have to be specified, developed and validated with care. Then, they are set in order for the system to have a determinist and predictable behavior. The interest of multi-agent systems in this context may be considered as limited, especially because of autonomy and proactivity properties generally attributed to agents. In fact, the decision step in real-time systems is very often hidden and examples of usages of multi-agent paradigm in the real-time context [3, 18] exploit the distributed aspects of multi-agent systems much more than the autonomy aspects.

In this paper, we aim at addressing systems in which time constraints are neither critical (obtaining a response a little bit later than specified is acceptable) nor strict (when a normal delay of response is exceeded, the result is not

immediately worthless but its value decreases more or less quickly with time). Another characteristic of such systems is the variability and unpredictability of treatments to process and their priority, but also of the availability of active entities (processors) in charge of processing. In such a context of dynamic scheduling in distributed systems, there is no solution yet capable to guarantee the respect of timing constraints. Our purpose is then to design this scheduling so as to optimize the compromise between the respect of logical correctness and timeliness, possibly by loosening some constraints when all of them cannot be satisfied simultaneously.

More precisely, rather than scheduling in its classical understanding, our concern here is the problem of adaptive reconfiguration of the processing chain during the execution. This reconfiguration can occur according to the available resources (sensors, processors, effectors), to the wished logical correctness, to the measured timeliness and to the events occurring in the environment. But, instead of doing this in a centralized manner, the agents will need to control the reconfiguration themselves, in addition to their normal activity of data processing.

Our objective here is to propose a complete approach, from a software engineering point of view, for the design of adaptive multi-agent systems. It covers all stages of software life cycle, from an abstract specification of the application architecture to a testable implementation, including formal verification of properties and simulation. The method is based on the formalism of timed automata [1], which allows to express systems as a set of concurrent processes satisfying some time constraints (section 3). We show that this formalism may be used in order to model a multi-agent system from the angle of data processing as well as that of dynamic treatment chain reconfiguration (section 4). Then, we show how model-checking and simulation may be used to verify selected properties of the system and analyze a priori its behavior (section 5). Finally, we address the problem of semi-automated translation from a timed automata specification to executable agents (section 6). But before giving more details about this work, it is necessary to give some words of explanation about our target application and its specificities.

## 2 Target application and objectives

The context in which we develop our approach is the project that we call *Dance with Machine* [12]. This project aims at staging a real-time dialogue between a human dancer-actor and a multimodal multimedia distributed cognitive system. The role of the latter is to achieve in real-time the captation and analysis of the performance of the dancer, and to build a multimedia answer to it. This answer may consist in visual animations projected on screens around the dancer, musical sequences, or actions by robots or other physical objects. We consider this application as a metaphorical transposition of the kind of interactions that we may forecast between human users and communicating objects. This is called *Ambient Cognitive Environments* (ACE), i.e., physical environments in which

perception, processing and action devices have to organize dynamically and in a cooperative way in order to provide users with natural interaction and extended services.

The computerized setup is composed of a set of processors equipped with communication capabilities. They may also be connected to sensors (video cameras, biometric sensors, localization sensors, etc.) or effectors (screens, loudspeakers, engines, etc.). Each processor may run one or several agents, each of them being specialized for a specific kind of treatment. Data retrieved from the sensors must be handled by several agents before being converted into actions. Agents' work is to analyze, synthesize and transform the data that they get. Data produced by an agent are then transmitted to other agents in order to continue the processing. The data are finally used to generate pictures, sounds or actions, either when the analysis is precise enough, or when the available time is too limited. Figure 1 shows a very simplified view of this process. Only one perception modality is represented, which corresponds to a video camera.

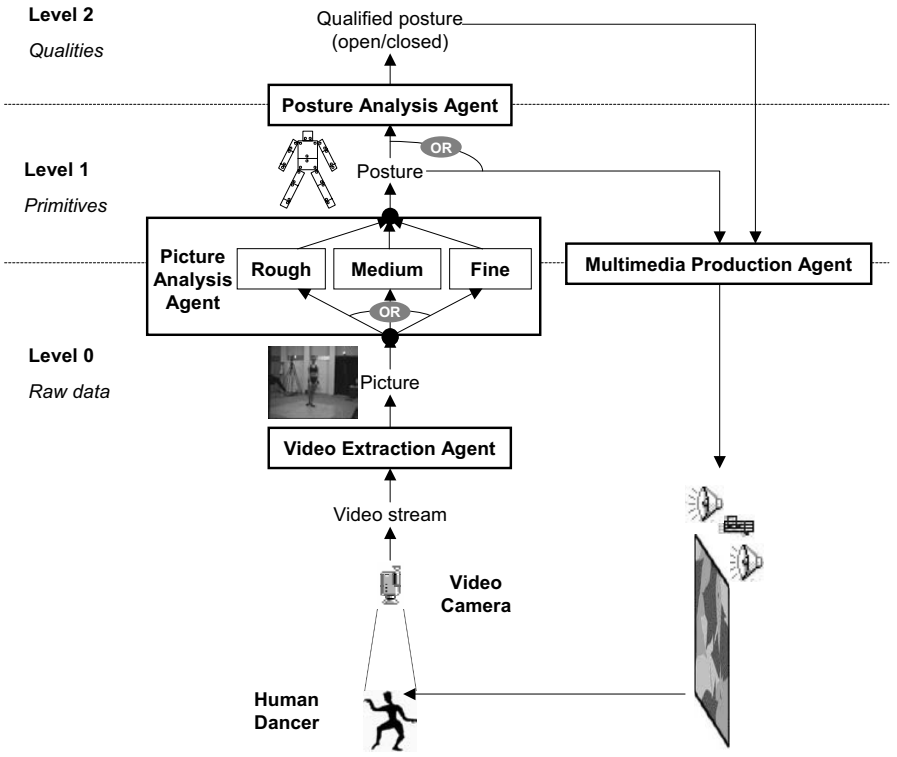


**Fig. 1.** Global architecture of the processing chain in the project "Dance with Machine".

The use of agents in this context is justified by the distributed nature of

the application (captation, processing and action are distributed among several objects and processors). But the main reason why we use agents is to make the whole system adaptive in various contexts: when components are added or removed, when the global behavior of the system must change, or when time constraints are not met by the system. The main time constraint that the system should respect is the latency, i.e., the time between the acquisition of data by sensors, and the production of corresponding actions by the system, under one form or another. This latency should of course be kept as low as possible so that the reaction of the system seems instantaneous (at least very quick). On the other hand, the analysis of the dancer's performance should be kept as precise and thorough as possible. These two constraints are potentially contradictory since a precise and thorough analysis can take significantly more time than a rough and superficial one. The quality of an analysis can be measured along two complementary dimensions: the precision (for the measure of a parameter of the performance) and the thoroughness (when optional treatments are possible, a superficial processing will be limited to what is compulsory).

Our main purpose is to allow a very quick evaluation of various strategies in the control of the processing chain, in order to produce an efficient agent-based implementation of the system. We achieve it using a formal model of the system along with tools that we developed to automate the implementation of a functional prototype. Model-checking allows to verify that the systems complies to the specified constraints (latency, non-blocking, sequentiality of treatments, etc.). Simulation, for its part, allows to evaluate the quality of the compromise between logical correctness (is the quality of processing satisfactory?) and timeliness (does the system comply to time constraints?).

## 3 Introduction to timed automata

Real-time systems may be specified using numerous dedicated methods and formalisms. Most of them are graphical semi-formal notations allowing a state machine representation of the behavior of the system. Among the most popular formalisms, we may quote Grafcet [7], SA/RT [17], Statecharts [8], UML/RT [5]. Such visual representations do not enable to verify the properties of systems and it is necessary to associate a formal semantics to them, based in general on process algebras [9], Petri nets [6] or temporal logics [15]. Proposing a new formalism is not our intention here. On the contrary, we prefer to examine the potential benefit of real-time specification and verification techniques in the design and the programming of agent-based reactive systems. We chose for this purpose to use timed automata [1]. This formalism has the advantage to be relatively simple to manipulate and to possess adequate expressivity in order to model time constrained concurrent systems. Moreover, there exists for this model powerful implemented tools (e.g., UPPAAL [13]) allowing model-checking and simulation.

### 3.1 Standard model

A timed automaton is a finite state automaton provided with a continuous time representation through real-valuated variables, called *clocks*, allowing to express time constraints. Generally, a timed automaton is represented by an oriented graph, where the nodes correspond to states of the system while the arcs correspond to the transitions between these states. The time constraints are expressed through *clock constraints* and may be attached to states as well as to transitions. A clock constraint is a conjunction of atomic constraints which compare the value of a clock $x$, belonging to a finite set of clocks, to a rational constant $c$. Each timed automaton has a finite number of *states* (locations), one of them being distinguished as *initial*. In each state, the time progression is expressed by a uniform growth of the clock values. In that way, in a state at each instant, the value of the clock $x$ corresponds to time passed since the last reset of $x$. A clock constraint, called an *invariant*, is associated to each state. It has to be satisfied in order for the system to be allowed to stay in this state. Transitions between states are instantaneous. They are conditioned by clock constraints, called *guards*, and may also reset some clocks. They may also carry labels allowing synchronization. An example of timed automaton and a corresponding possible execution is shown in figure 2.

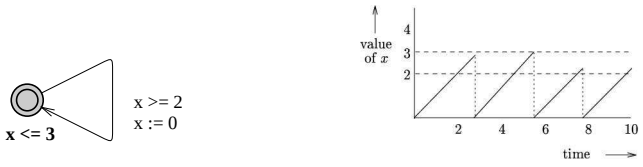


**Fig. 2.** Example of a timed automaton, where $x$ is a clock. The guard $x \geq 2$ and the invariant $x \leq 3$ imply that the transition will fire after 2 and before 3 time units passed in the state.

The behavior of a complex system may be represented by a single timed automaton being a product of a number of other timed automata. The set of states of this resulting automaton is the Cartesian product of states of the component automata, the set of clocks is the union of clocks, and similarly for the labels. Each invariant in the resulting automaton is the conjunction of the invariants of the states of the component automata, and the arcs correspond to the synchronization guided by the labels of the corresponding arcs.

### 3.2 Extensions in UPPAAL

We use UPPAAL for our modelling; a detailed presentation of this tool may be found in [13]. We remind here only the main characteristics and extensions with respect to the standard model [1]. In UPPAAL, a timed automaton is a finite

structure handling, in addition to a finite set of clocks evolving synchronously with time, a finite set of integer-valuated and Boolean variables. A model is composed of a set of timed automata, which communicate using binary synchronization through transition labels and a syntax of emission/reception. By convention, a label $k!$ indicates the emission of a signal on a channel $k$. It is supposed to be synchronized with the signal of reception, represented by a complementary label $k?$. Absence of synchronization labels indicates an internal action of the automaton. The execution of the model starts in the initial configuration (corresponding to the initial state of each automaton with all variable values set to zero), and is a succession of reachable configurations. The configuration change may occur for three reasons:

- by time progression corresponding to $d$ time units in the states of the components, provided that all the state invariants are satisfied. In the new configuration, the clock values are increased by $d$ and the integer variables do not change;
- by a synchronization if two complementary actions in two distinct components are possible, and if the corresponding guards are satisfied. In the new configuration, the corresponding states are changed and the values of clocks and of integer variables are modified according to the reset and update indications;
- by an internal action if such an action of a component is possible, it may be executed independently of the other components: the state and the variables of the component are modified as above.

Another peculiarity of UPPAAL, useful in expressing a kind of synchronicity of moves, is the notion of "committed" states, labelled in the figures by a special label C; see, for instance, the state *Choice* in the first automaton of figure 5. In such a state, no delay is permitted. This implies an immediate move of the concerned component. Thus, two consecutive transitions sharing a committed state are executed without any intermediate delay.

UPPAAL allows simulating systems specified in this way, detecting deadlocks and to verify, through model-checking, various reachability properties. Typically, it can answer the questions like "starting from its initial state, does the system reach a state where a given property is satisfied?", "starting from its initial state, is a given property always true?", or "starting from its initial state, can the system reach a given state in a given delay?".

## 4 Modelling a decentralized reactive system

As stated earlier, timed automata allow to model systems as a set of concurrent processes. We will detail gradually in the sequel the way they may be applied to our case study. The behavior of our agents consists in receiving and processing input data in order to generate and send new outputs. The processing has a duration, considered as fixed, and has to be performed repeatedly. The corresponding model is shown in figure 3.

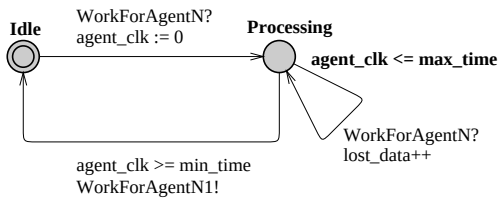


**Fig. 3.** A model of a simple agent.

Initially, the agent is waiting for new data in the state *Idle*. It starts processing on reception of the signal *WorkForAgentN* passing to the state *Processing*. It comes back to the state *Idle* at the end of its treatment, which takes a time comprised between *min_ time* and *max_ time*. The following agent is informed then (through the synchronisation on the channel *WorkForAgentN1*), that it can start processing.

This simple model presents however the following drawback: if a new treatment request comes to an agent when it is already processing, the corresponding data is lost. The number of such events is counted by incrementing the variable *lost_ data*. Nevertheless, the loop at the state *Processing* is necessary to avoid deadlocks which may occur if the situation described above happens. A solution can be to introduce an additional state playing the role of a buffer (see figure 4).

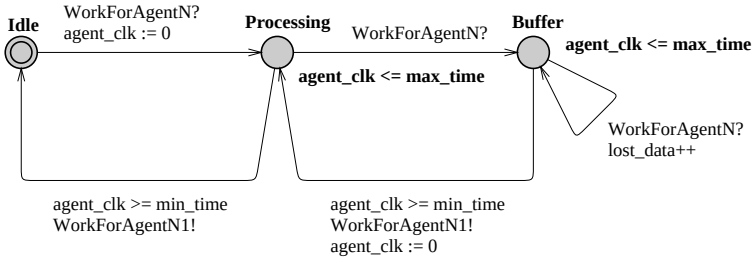


**Fig. 4.** A model of an agent with a buffer.

Now, if a new request arrives to the agent while it is in the state *Processing*, it passes to the state *Buffer*. Then, it comes back to the state *Processing* at the end of the treatment, in order to start a next one. If a new request comes when the agent is already in the state *Buffer*, then the corresponding data is lost. At this stage, we shall still take into account the fact that a few modules (corresponding to various precisions of the processing) are available and may be used to analyze the dancer's posture. A first approach consists in duplicating the agent in charge

of the corresponding treatment by associating to each copy a different duration constant. However, when a new data is available, it is transmitted to one of the agents chosen in a non-deterministic way. Thus, it is necessary to incorporate in the agent a controller responsible for choosing between different treatment modules. This solution is represented in figure 5.

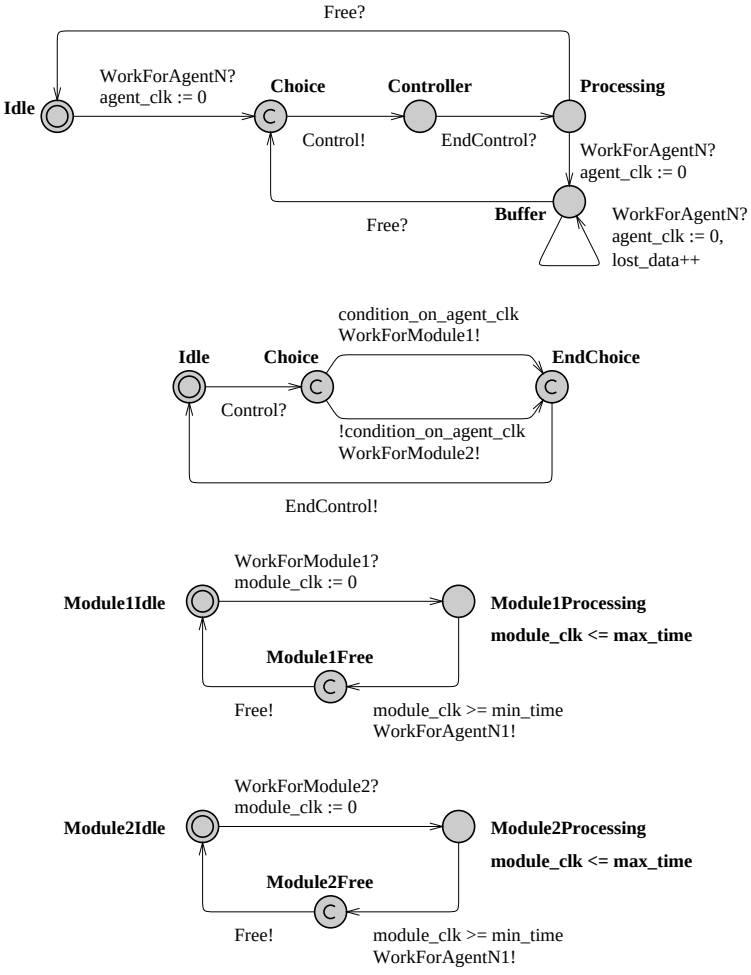


**Fig. 5.** A model composed of a generic agent, a controller module and two treatment modules.

When some data is ready to be processed, the controller module passes in the state *Choice*. The agent chooses to execute a treatment module depending on

the value of the boolean expression *condition_ on_ agent_ clk*. When the chosen module achieves processing, it informs about it the next agent in the processing chain, then it informs the controller by sending the signal *Free*.

## 5 Verification and simulation

The controller presented in the previous section needs of course to be instantiated by fixing explicitly the criteria determining the choice between treatment modules. We present three different strategies that may be considered and address verification and simulation experiences which may be accomplished for some interesting properties. The particular context considered for this study is explained in figure 6.

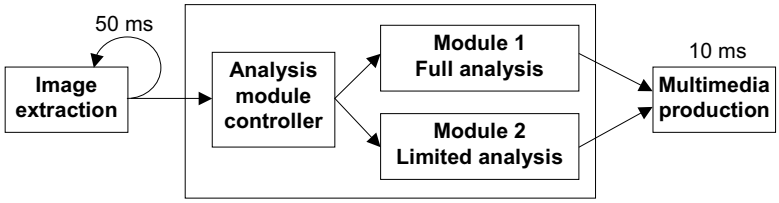


**Fig. 6.** A simplified scheme of the processing chain.

The extraction agent produces an image every 50 ms, which has to be treated by the agent in charge of the analysis. This treatment should be performed either by a module capable to accomplish a complete analysis or by a module which can do only a partial one but taking less time ($t_{treatment_2} < t_{treatment_1}$). The controller has to be designed in such a way that it could be possible to conciliate two potentially contradictory criteria: analyzing all images or, in other words, avoiding loosing too many of them (timeliness) and performing a maximum of complete analyzes (logical correctness).

### 5.1 Different strategies of choice

The first proposal is not really a strategy but we give it as a reference. It consists only in systematically alternating the two treatment modules.

In order to minimize the loss of images, the idea is to anticipate, when the agent performs the choice ($t_{choice}$), the date when the agent will receive a new image to analyze while it has already an image in its buffer and has not terminated its current analysis ($t_{loss}$). This is possible since the frequency of arrivals of new images is constant. Thus, in the second strategy, the module 1 will be chosen if and only if $t_{treatment_1} < t_{loss} - t_{choice}$.

In order to maximize the number of complete analyzes, one can loosen the

previous constraint by allowing to use the module 1 even if its execution will necessarily entail a loss of an image. In the third strategy, the module 1 will be chosen if and only if $t_{treatment_1} < (t_{loss} - t_{choice}) * coef$, where $coef$ fixes the limits of allowance.

### 5.2 Results

For each strategy, it is possible to check with UPPAAL that the system satisfies certain properties. In particular, we checked that:

- there is no deadlock: A[ ] not deadlock;
- there is no image lost: A[ ] lost_data == 0;
- the ratio of the choice of module 1 is grater than a given threshold: A[ ] (nb1 * 100 / (nb1 + nb2 + lost_data)) > 50).

Moreover, it is possible to simulate the system during a given number of cycles and to check experimentally the ratio of lost images and images which could be analyzed completely versus treatment times $t_{treatment_1}$ and $t_{treatment_2}$, as shown in figure 7.

Model-checking techniques allow to verify formally and automatically if some properties of the system, considered as important, are satisfied in all possible system evolutions. On the other hand, simulation permits to obtain some empirical evaluation of performances of the system in terms of logical correctness and timeliness, depending on the characteristics of treatment modules and on the applied strategy. This allows also envisaging a supplementary control level for the agent in charge of the image analysis. This corresponds to a kind of "meta-strategy" which could adapt dynamically the strategy of choice depending on various constraints and fixed objectives.

## 6 Automated code generation

After having validated the model of the multi-agent system, both formally and experimentally, the next stage of development corresponds to translating it into an executable prototype. In order to do so, a naive idea could consist in implementing each timed automaton by a thread, since they are models of concurrent processes. Nevertheless, for a same agent modelled by several automata, it could involve several synchronization and lead to decline sensibly its performances, which could be awkward for a reactive system. Thus, a first step consists in performing first a synchronized product of all automata describing the same agent in order to transform it next into a skeleton of an application. The compiler that we developed produces this synchronized product by performing also a number of optimizations in order to minimize the size of the resulting automaton. Each agent is modelled consequently by a unique timed automaton, which can be translated into an executable form in several steps. First, only the finite state automaton aspects of the given timed automaton are considered. The states where it is necessary to let the time progress are assumed to correspond to some

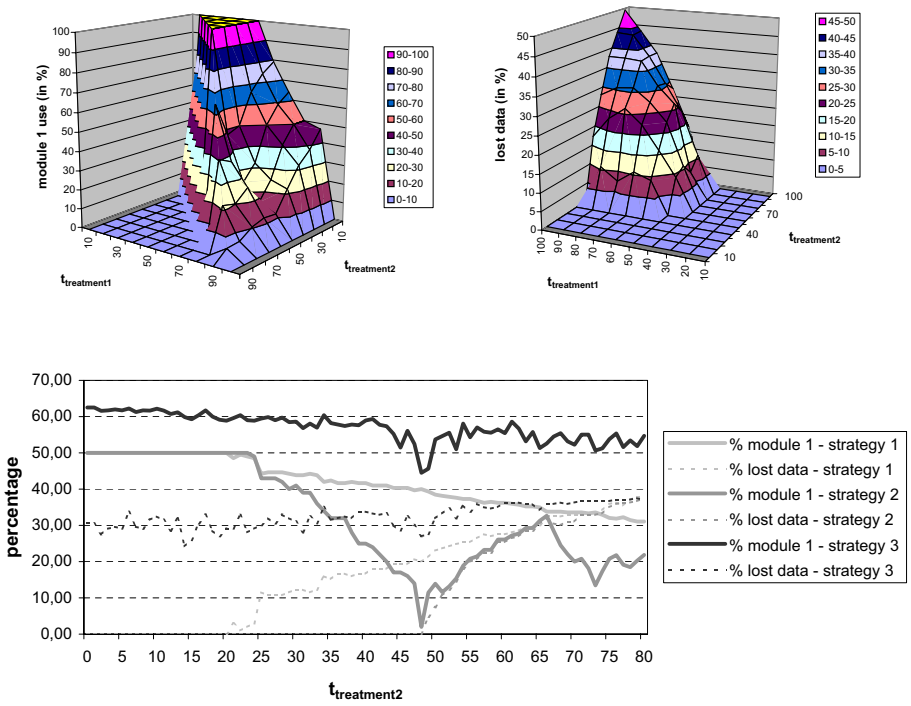


**Fig. 7.** The ratio of images analyzed with the module 1 (on the left) and the ratio of lost images (on the right), obtained for the second strategy and various values of time of treatment for modules 1 and 2. On the bottom, a comparison of the three strategies for $t_{treatment_1} = 80ms$ and *coef*= 1.25, for various values of $t_{treatment_2}$.

treatments. Our compiler translates it in terms of a state in which the agent does a break (which is supposed to be replaced by the corresponding treatment module when it is available). Finally, the synchronization signals between automata are associated to communications between the corresponding agents.

## 7 Conclusion

We presented in this paper a complete approach, from the software engineering point of view, for the modelling of adaptive real-time systems based on the multi-agent paradigm. The usage of timed automata specification and verification techniques played here a central and unifying role. We showed how this formalism, thanks to its capabilities to model concurrent processes having time constraints, can be adapted in order to represent multi-agent systems. Moreover, we demonstrated that it could be possible to model in a modular way an agent controller, capable to make decisions depending on some fixed objectives.

The advantage one can take from this formal specification is twofold: First,

it is possible to check the model against various kinds of deadlock (or timelock) and more generally, against any property coming from a non-respect of time constraints, and avoid this way some problems at a very early stage of development. Second, it is worthwhile to take advantage of timed automata representation of the system in order to generate automatically application skeletons. To do so, we developed a specific compiler which, taking an XML representation of the timed automata specification, produces a skeleton based on the JADE multi-agent platform [4]. This prototype is finally used to validate choices made previously, during modelling and implementation, and to review and modify some of them if necessary.

Finally, the general purpose of this work consists in exploiting the approach described in this paper, the design patterns and the composition tools, in order to facilitate the design of an entire system. These design patterns could be coupled with machine learning techniques for the exploration of parameter spaces, in order to optimize agent behaviors when the model becomes more complex. Also, it would be interesting to develop an experimental protocol in order to validate, on the real prototype, the properties observed on the model. In this context, the presented work, even if it is at a preliminary stage, demonstrates however the feasibility of this approach and allows to foresee favorably the development of powerful and complete tools dedicated to the implementation of reactive multi-agent systems.

## References

1. Alur R., Dill D. L., A Theory of Timed Automata, in Theoretical Computer Science, Vol. 126, No. 2, pp. 183-236, 1994.
2. Arai T., Stolzenburg F., Multiagent systems specification by UML statecharts aiming at intelligent manufacturing, in AAMAS'2002, pp. 11-18, 2002.
3. Attoui A., Les systèmes multi-agents et le temps-réel, Eyrolles, 1997.
4. Belleifemine F., Caire G., Poggi A., Rimassa G., JADE - A White Paper, http://sharon.cselt.it/projects/jade/papers/WhitePaperJADEEXP.pdf, 2003.
5. Douglass B. P., Real-Time UML: Developing Efficient Objects for Embedded Systems, Addison-Wesley-Longman, Reading, MA, 1998.
6. Elmstrøm R., Lintulampi R., Pezze M., Giving Semantics to SA/RT by Means of High-Level Timed Petri Nets, in RTSJ, Vol. 5, No. 2/3, pp. 249-271, 1993.
7. Groupe AFCET Systèmes Logiques. Pour une représentation normalisée du cahier des charges d'un automatisme logique, in RAII, Vol. 61 & 62, 1977.
8. Harel D., Statecharts : A Visual Formalism for Complex Systems, in Science of Computer Programming, Vol. 8, 1987.
9. Harel D., Pnueli A., Schmidt J. P., Sherman R., On the Formal Semantics of Statecharts, LICS 1987, pp. 54-64, 1987.
10. Hatley D. J. , Pirbhai I., Strategies for Real Time System Specification, Dover Press, 1987.
11. S. Horstmann and G. Cornell. Core Java 2, Vol. 1 & 2, Prentice Hall, 1999.
12. Hutzler G., Gortais B., Joly P., Orlarey Y., Zucker J.-D., J'ai dansé avec machine ou comment repenser les rapports entre l'homme et son environnement, in JFIADSMA'2002, pp.147-150, Hermès Science, 2002.

13. Larsen K. G., Pettersson P., Yi W., UPPAAL in a Nutshell, in Springer International Journal of Software Tools for Technology Transfer, 1(1-2), pp. 134-152, 1998.
14. Occello M., Demazeau Y., Baeijs C., Designing Organized Agents for Cooperation with Real-Time Constraints, in CRW'98, pp. 25-37, Springer-Verlag, 1998.
15. Manna Z., Pnueli A., The Temporal Logic of Reactive and Concurrent Systems: Specification, Springer-Verlag, 1991.
16. Soler J., Julian V., Rebollo M., Carrascosa C., Botti V., Towards a Real-Time Multi-Agent System Architecture, in COAS, AAMAS'2002, 2002.
17. Ward P., Mellor S., Structured Development for Real-Time Systems, Prentice-Hall, 1985.
18. Wolfe V. F., DiPippo L. C., Cooper G., Johnston R., Kortman P., Thuraisingham B., Real-Time CORBA, in IEEE TPDS, Vol. 11, no. 10, 2000.