An Agent Architecture to fulfill Real-Time Requirements

Ignacio Soto*
Dpt. Tecnologías de las Comunicaciones, Universidad Carlos III de Madrid
c/ Butarque 15, 28911 Leganés, Spain

Mercedes Garijo, Carlos A. Iglesias
Dept. Ing. de Sistemas Telemáticos - Universidad Politécnica de Madrid
ETSI Telecomunicación, Ciudad Universitaria s/n, 28040 Madrid, Spain

Manuel Ramos
Dpt. Tecnologías de las Comunicaciones, Universidad de Vigo
Campus Universitario s/n, 36200 Vigo, Spain

isoto@it.uc3m.es, (mga,cif)@gsi.dit.upm.es, mramos@ait.uvigo.es

ABSTRACT

In this paper we present AMSIA, an agent architecture that combines the possibility of using different reasoning methods with a mechanism to control the resources needed by the agent to fulfill its high level objectives. The architecture is based on the blackboard paradigm which offers the possibility of combining different reasoning techniques and opportunistic behavior. The AMSIA architecture adds a representation of plans of objectives allowing different reasoning activities to create plans to guide the future behavior of the agent. The opportunism is in the acquisition of high-level objectives and in the modification of the predicted activity when something doesn't happen as expected. A control mechanism is responsible for the translation of plans of objectives to concrete activities, considering resource-boundedness. To do so, all the activity in the agent (including control) is explicitly scheduled, but allowing the necessary flexibility to make changes in the face of contingencies that are expected in dynamic environments. Experimental work is also presented.

1. INTRODUCTION

This article deals with agents that must carry out missions in dynamic and complex environments. This agent domain takes into account possible relationships between the agents' missions. These relationships can be causal (doing some part of a mission influences how to carry out other missions) or due to the usage of common limited resources (it is possible that the agent lacks the resources to carry out all its missions). Missions have as relevant parameters their priority (importance for the agent) and deadline. We are going to evaluate the agents not only for accomplishing missions, but for accomplishing them in time.

The article introduces AMSIA, an agent architecture that supports real-time requirements, and it is based on the blackboard model since it provides an easy way of integrating different reasoning techniques and opportunistic behavior. Planning techniques have been combined with the blackboard model for considering the *resources* needed by the agent to accomplish its missions. Planning is a basic technique for the agent to reason about how to achieve its objectives. Also, plans are the base to be able to control the use of resources (if we have not a plan to do something, it is difficult to know if we are going to have resources to do it). Nevertheless, the use of plans in dynamic environments has its problems. On the one hand, it is useless to make detailed plans in advance, it is usually better to delay decisions of how to do certain things until the moment in which that decision is needed. In other words, hierarchical plans, which have different levels of abstraction are appropriate for these environments. Moreover, certain situations can make a plan invalid, hence it must be easy to extend, modify or, if nothing better, cancel a plan (and develop another).

Our architecture combines activity guided by plans of objectives, with opportunism in acquiring high-level objectives (missions) and in building, modifying, and extending the plans. Also, all the activity in the architecture is explicitly scheduled, both reasoning activity (including building plans) and activity in the environment (actions and perceptions), with the resources needed to carry it out.

In section 2 we describe AMSIA, the proposed agent architecture. In section 3 we present experimental results obtained with an agent built using the architecture. The agent controls a robot in a simulated environment. In section 4 we compare our work with related approaches. And, finally, in section 5 we summarize our conclusions and propose future

*This work was partially done while the first author was visiting the Área de Ingeniería Telemática of the University of Vigo.

Agents 2000 Barcelona Spain

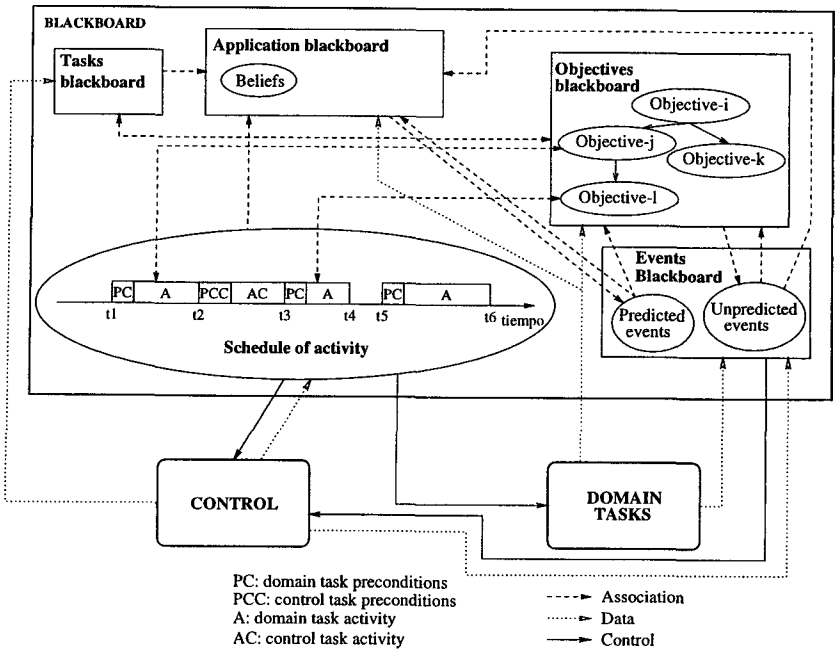

Figure 1: The AMSIA agent architecture

lines of research.

2. THE AMSIA AGENT ARCHITECTURE

The proposed agent architecture can be seen in figure 1. This architecture allows the agent to execute the tasks corresponding to its missions. Tasks can embody computational activities (as, for example, reasoning activities) or actions in the real world and/or perceptions of the environment (in this case, the task is an abstraction of a high level action in the environment for the architecture, in fact it is a reactive module to do something in the world)[1].

The main components of the AMSIA architecture are the events blackboard, the objectives blackboard, the knowledge sources corresponding to tasks, and the schedule of activity.

The motor of activity in the architecture are events (AMSIA is event-driven). Events are abstractions that the architecture uses to signal opportunities of activity, and they are stored in the events blackboard. In our architecture an agent will have KSs interested in the events. When an event is produced, a task will be created based in the corresponding KS. The events can be created as the result of reasoning activity (by reasoning KSs, the events signal that there is symbolic information that the agent can follow), and as the result of sensor activity (by sensor KSs, the events signal a contingency in the environment that is relevant for the agent).

2.1 Plans of objectives in AMSIA

The objectives of the agent are modeled as special objects in the blackboard. Its creation or modification generates internal events. Most of the knowledge (the corresponding KSs) in the architecture is indexed by the objectives it allows to achieve. A KS interested in an event associated with an objective can directly achieve the objective or can be a planning activity (in the sense that the corresponding task generates a sub-plan of objectives to progress in achieving it).

Each objective is specified in the architecture by the following information:

1. **Name**: As any other object in the blackboard, the objects *objective* have a unique name that distinguishes them from any other object. Hence it is possible to have two or more objectives of the same kind activated in the architecture, and their names can be used to distinguish among them.
2. **Class**: All the objects *objective* created by applications are subclasses of the object *Objective* defined in the architecture. The class defines the kind of objective to which the object refers.
3. **UnreacheableP**: This attribute allows a task to indicate that, in principle, it is not possible to achieve this objective, stating that the plan where it belongs is invalid. This is a useful information for re-planning. More information about the reason because of it is not possible to achieve the objective will be left in the blackboard.
4. **The following links** (all the links have an inverse indicated in brackets):
   (a) **ToParameter (ToObjective)**: links to blackboard objects that hold information related to the objective. These objects contain a representation of the particular situation specified by the

[1]In blackboard terminology, tasks are instances of Knowledge Sources (KSs).

objective, that domain tasks can understand and analyze.

(b) **Need (NeededFor)**: links to objectives which must be achieved before it is possible to try to achieve this one. Hence, this link defines a causal relationship between the objectives pointed by links with this name and this objective.

(c) **Before (After)**: this kind of link defines an order relationship between this objective and those pointed by links with this name.

(d) **ToSubPlan (ToFatherObjective)**: link to a plan that, possibly, allows to advance in the achievement of this objective, i.e., this link defines a decomposition of an objective in a (sub)plan of objectives. If an objective has several *ToSubPlan* links, it is because there are several alternative plans (*OR* relationship) to try to achieve it.

Figure 2 shows an example of a structure of a plan of objectives (inverse links are omitted).

The structure of objects *objective* allows to represent partial order and hierarchical plans. But with these important singularities:

1. The steps in the plans are not directly activities, but specify them, because there will be a set of possible tasks (KS instances) associated with each step (objective) of the plan. These tasks are alternatives to achieve the objective or to create sub-plans with the same end.

2. There is not a fixed algorithm to create the plans. There is not an algorithm that extends the mentioned structure until something executable is reached. On the contrary, all the steps in the plan specify executable tasks, but these tasks can have the mission of generating sub-plans. In this way, different reasoning algorithms can be combined. The standard structure of objects *objective* is what allows the integration of the different algorithms and the use of the plans to guide the future activity of the agent. Using the objective and the objects pointed by *ToParameter* link, the planning tasks can understand what a plan means and how to modify or expand it.

3. It is the control mechanism of the architecture which is in charge of choosing particular tasks for achieving an objective (or follow a plan of objectives). It is usual to have different tasks associated with the same objective. Different tasks represent different mechanisms of reasoning or different methods to execute actions in the environment, usually with different relationships between response quality and needed resources. The control mechanism will choose among the possibilities to maximize the quality considering the available resources.

The representation of the plans is, in fact, a tree of objectives. This has the interesting property that at any point, the control mechanism can choose between extending an objective (a plan) or finishing it, just by choosing the adequate task, i.e., we can associate emergency tasks with objectives (at any level of the plan) that do some kind of minimal action when there are not resources to extend the objective. Hence, this tree structure favors an anytime reasoning.

2.2 Control mechanism

2.2.1 Schedule of activity

As it has been told, the control mechanism of the architecture is in charge of deciding which tasks to execute. It works on the events that signal opportunities of activity. These events can mean that a plan/sub-plan has been created (or modified) or that a situation has been detected that represents an opportunity for the agent although not linked (initially, at least) with its current objectives. The control mechanism uses the events to create tasks based in the knowledge sources interested in those events. The knowledge sources in our architecture have a *resource specification*, which allows the control mechanism to calculate the resources needed by the tasks.

The control mechanism works building a schedule of activity that defines the tasks that are going to be executed by the system and the resources needed by those tasks. In the current implementation the only resource considered is time but other resources could be equally considered.

The construction of the schedule is incremental, the control mechanism incorporates sequences of tasks as the result of events produced by new plans for the system or new situations to analyze. It is responsibility of the control mechanism to keep the schedule feasible in the sense that the agent must have enough resources to carry it out. In the current implementation, as the only resource considered is time, the schedule registers the instants before each task must begin and finish its execution, and the estimated time of execution of the tasks. The sequences of tasks are introduced in the schedule according to the links of its objectives to objectives of tasks in the schedule (i.e., under application directions if they exist) or using EDF (Earliest Deadline First) criterion applied to missions.

The control mechanism doesn't reason about how to achieve objectives. It only chooses among possible sequences of tasks. Domain tasks reason and create plans of objectives, control only translates those plans into sequences of tasks and introduces them in the schedule. At any point it is easy to change a task to do something, because the alternatives are kept associated to the corresponding objective, and the tasks in the schedule are linked to their objectives.

When there are resource conflicts (i.e., there are not resources to execute the sequences of tasks corresponding to the objectives of the system), the control mechanism has to deal with them. The goal of the control mechanism is to maximize the profit of the line of activity of the agent. To do so, it always tries to favor the sequences of tasks corresponding to more important plans and to use tasks to carry out those plans which offer more quality. The control mechanism can remove sequences of tasks corresponding to less important plans to favor the introduction of those corresponding to more important ones. This is not an optimum

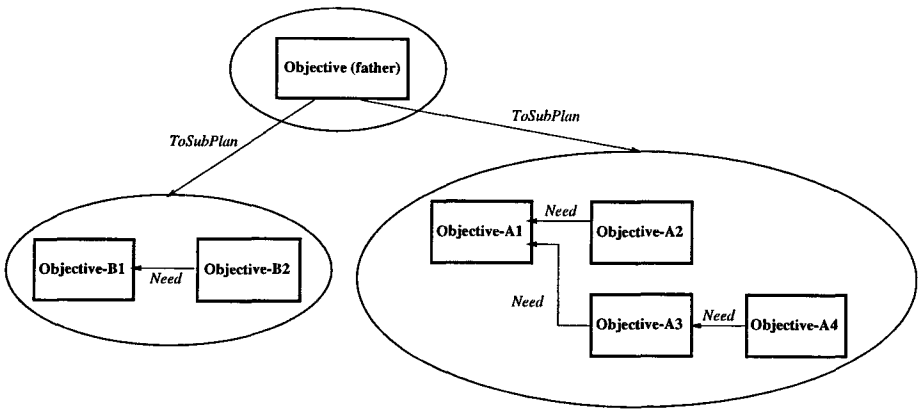

Figure 2: A plan of objectives in the architecture

process, not all the combinations of remove/introduce tasks are tried. The priority is to introduce the tasks of the more important plans removing those of less importance that use resources needed by the more important plan. Afterwards, if there are resources, those plans with less importance are considered to be re-introduced in the schedule. Notice that the control mechanism has the flexibility of choosing tasks that need less resources to follow a plan although it can mean less quality in the achieved objectives. A more detailed description of how it works can be found in [19].

To decide between sequences of tasks the control mechanism scores them taking into account both the importance of the objectives of the plan and the quality offered by the tasks used to follow it.

An important point is that the operations of the control mechanism cannot be done in a fixed or negligible time. Hence, the control operations must be scheduled themselves. To do so, the control operations are divided in tasks (control tasks that are instances of control knowledge sources). Each control task analyzes an event or a related set of events (for example, those associated with the creation of a plan of objectives).

2.2.2 *The short-term scheduler*

Some domain tasks (e.g., planning tasks) are specified (its corresponding knowledge source) as creating events, and so control tasks can be scheduled (with its needed resources) with them. But other events cannot be predicted (e.g., a situation that indicates a new opportunity for the agent as the reception of a message). Control tasks to deal with these events must be introduced dynamically in the schedule of activity of the agent without risking its feasibility. Our agent architecture uses a short-term scheduler that works with the algorithm shown in figure 3

Basically, it tries to introduce control tasks to deal with unpredicted events while there is time in the schedule or if the pending events are more important than the tasks in the schedule. When there are not pending events, it chooses the first task in the schedule for execution.

2.3 Reactivity

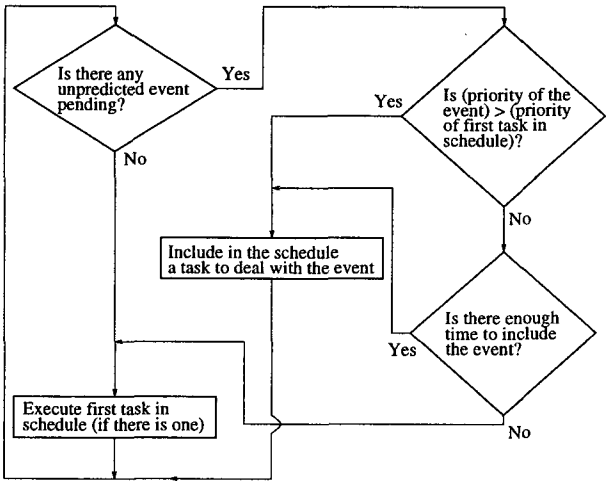

Figure 3: The algorithm of the short-term scheduler

The proposed architecture has limited reactivity because a contingency that the task being executed is not prepared to handle, will not be analyzed till the execution of the task is finished. By adding a reactive layer, the reactivity is increased giving fast responses to contingencies. Also, this permits to deal with the continuous activity of a real environment translating it to discrete events that our architecture can manage. We have then (see figure 4) a two-layer horizontal architecture (only the reactive layer has access to sensors and actors). This reactive layer will be built as a series of concurrent skills (as in [1]). The concurrent skills will be activated and deactivated by the tasks (of our proposed architecture) in charge of interacting with the environment. Conceptually, the reactive layer doesn't change the vision that the architecture presented in this paper has, only it uses a more suitable interface to the environment (a specialized skill of the reactive layer) instead of directly the actors or sensors.

3. EXPERIMENTAL WORK WITH AMSIA

3.1 The simulated environment

We have implemented the AMSIA agent architecture modifying BBK [4], a C++ implementation of the blackboard

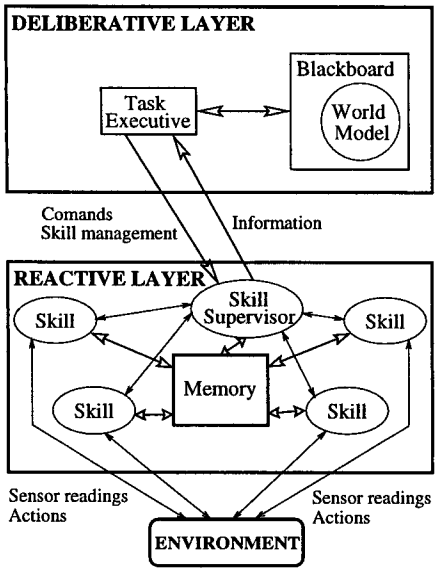

Figure 4: The agent architecture with a reactive layer

architecture for control [10], and adding the mechanisms described in this paper. We have applied it to control a simulated robot (a modified version of the Khepera simulator [15]) that receives requests to carry out missions in the environment. The missions have the following characteristics:

- A deadline: each mission must be accomplished by the agent before its deadline.
- An importance: each mission has an associate importance.
- A destination: the environment presented by the simulator is a collection of rooms. Missions consist of going to a room (destination) and make a fault diagnosis and repair there. Information needed by the robot to do the diagnosis can be obtained only if it is in the destination room.

First, we identify the factors that can influence in the performance of the agent:

1. Dynamism: the dynamism is configured in the simulator by two parameters:
   (a) missions dynamism: the ratio of appearance of new missions. Modeled by an exponential distribution with mean $\bar{t}_M$.
   (b) obstacle dynamism: the ratio of appearance of obstacles that can make more difficult or make impossible the accomplishment of some missions, modeled by an exponential distribution with mean $\bar{t}_O$. And the life of those obstacles, modeled by an exponential distribution with mean $\bar{t}_{OD}$.

Factor	values
Missions dynamism	high, medium, low
Importance range	medium
Deadline	big
Obstacle dynamism	low

Table 1: Independent variables in the experiment

2. Deadline: how is the deadline associated with missions. The deadline is modeled by an exponential distribution shifted to the right t_{MD} and with mean $\bar{t}_{MP}$.
3. Range of importance: the importance of missions is distributed uniformly between **0** and I_{max}.

The variables that we use to measure the performance of our agent in a certain interval of time are:

1. *Effectiveness* $= \frac{S}{T} \times 100$.

   where S is the score obtained by the agent and T is the total score offered to it. The score is calculated as follows:

$$score = \sum_{\text{missions accomplished}} (importance_{mission} + 1) \quad (1)$$

   Missions accomplished refers to those accomplished *before* their deadlines.

2. *Mission effectiveness* $= \frac{M}{T_m} \times 100$

   where M is the number of missions accomplished by the agent and T_m is the total number of missions offered to it.

3. *Importance effectiveness* $= \frac{M_h}{T_h} \times 100$

   where M_h is the number of missions of the highest importance accomplished by the agent and T_h is the total number of missions of the highest importance offered to it.

We wanted to measure the performance of the agent in stationary state, so we did preliminary experiments and use them to decide the time of the simulation (15000 seconds), the number of samples in each condition (5), and the suppressed samples to avoid the transitory state. Also we used the preliminary experiments to determinate interesting values of the factors that influence the performance of the agent in the experiment. The values chosen for the experiment are shown in table 1.

The categories in table 1 correspond to the following values (in tenths of second) of the parameters in the simulator:

Missions dynamism = *high* $\Rightarrow \bar{t}_M = 275$

Missions dynamism = *medium* $\Rightarrow \bar{t}_M = 600$

Missions dynamism = *low* $\Rightarrow \bar{t}_M = 925$

Importance range = *medium* $\Rightarrow I_{max} = 5$

Deadline = *big* $\Rightarrow t_{MD} = 3000$ and $\bar{t}_{MP} = 10000$

Obstacle dynamism = *low* $\Rightarrow \bar{t}_O = 1000$ and $\bar{t}_{OD} = 100$

3.2 The experiment

In this section we present the results of comparing the performance of two agents, one built using our architecture and the other built using BBK[2]. This last agent uses the abstract plans of the blackboard architecture for control [10] to favor the most important missions among those the agent thinks it has resources to achieve, hence we call it IMP agent. Also, the IMP agent favors always finishing an on-going mission while it thinks it can achieve it (this is a useful strategy in the conditions of the experiment, see [18]).

To compare two agents that use architectures with the same style facilitates a fair comparison because both use the same knowledge, the same knowledge sources. On the other side, comparing our architecture with others farther from its architectural style would be more challenging but would give more interesting results.

The results of the experiment are presented in figure 5. We used t-tests to see the statistic significance of the differences shown in figure 5 (see table 2). We separate the results by *missions dynamism* because this factor has a strong influence in the dependent variables (hardly surprising, given the way in which were defined).

The agent built using the AMSIA architecture in this paper gets a significant better *effectiveness* that the IMP agent. Our agent and the IMP agent has similar *importance effectiveness*. This is interesting, our agent is able to get the same performance in the most important missions that the IMP agent (an agent which is precisely favoring the most important missions). But, our agent can use the mechanisms presented in this paper to achieve other missions without risking the performance in the most important ones (see that our agent has a better *missions effectiveness* that the IMP agent for all the range of *missions dynamism*).

4. RELATED WORK

There are a series of agent architectures developed to face the problem of interaction with dynamic environments.

Reactive architectures as the subsumption architecture [3] offer good performance in the interaction with the environment but it is not clear how they can adapt their behavior to great changes in the environment or in their missions. Moreover, it doesn't seem easy, using this kind of architecture, to build an agent to fulfill certain real-time requirements of high level objectives.

[2]Each sample was taken running an agent and the simulator in a PC Pentium-II, 233 MHz, 32 MB RAM, using Linux 2.0.30.

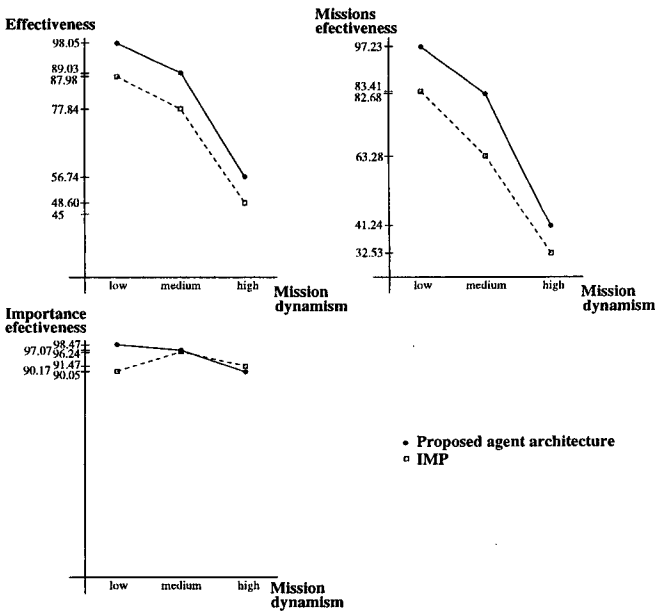

Figure 5: Results of the experiment

Hybrid architectures such as InteRRaP [16], TouringMachines [5] or Remote Agent [17; 6] use a reactive layer (or module) to ensure the security of the agent in front of events in the environment that can mean a risk to the agent. The reactive layer offers actions quickly to ensure the survival of the agent while the deliberative layer/s make plans to achieve the high level objectives of the agents, negotiate with other agents, etc. But deliberative actions are not scheduled themselves and so it is difficult to offer guarantees of global real-time requirements (specifically, it is difficult to adapt the reasoning to real-time constraints).

IRMA [2; 18] is a deliberative architecture thought to deal with resource-boundedness in the reasoning of the agent. The main procedure to do this is to use the plan of intentions that defines what the agent intends to do as a guide for the reasoning of the agent, limiting in that way its possibilities of reasoning. Options for deliberation are filtered to avoid loosing much time in deliberation. The idea is that the less promising options are discarded faster with the filtering process than if the agent deliberate about them. Options incompatible with the current plan of intentions are filtered this way. An override process allows that some options incompatible (with the current plan) but highly promising can pass the filter process and, so, the agent can deliberate about them. Much of the work with IRMA is to show the advantages of the filtering mechanism for a resource-bounded agent. Notice that in our agent architecture the global schedule effectively directs where the agent is going to spend its reasoning resources. The role of the filtering-override processes is played by the scheduler and how it deals with external events. But reasoning activity is scheduled and so the agent has the flexibility of choosing among different reasoning methods according with the circumstances, of deciding when to deliberate and how about a particular event, and of integrating several objectives and divide the resources among them.

Missions dynamism = low			
Dependent variable	**degrees of freedom**	**t**	**p**
Effectiveness	8	14.628	0.000
Missions effectiveness	8	15.391	0.000
Importance effectiveness	8	5.976	0.000
Missions dynamism = medium			
Dependent variable	**degrees of freedom**	**t**	**p**
Effectiveness	8	8.641	0.000
Missions effectiveness	8	14.007	0.000
Importance effectiveness	8	0.460	0.658
Missions dynamism = high			
Dependent variable	**degrees of freedom**	**t**	**p**
Effectiveness	8	5.060	0.001
Missions effectiveness	8	7.191	0.000
Importance effectiveness	8	-0.558	0.592

Table 2: Results of t-tests in the experiment

AIS [11; 12] is an agent architecture based in a blackboard model that has been modified to deal with real-time requirements. The last version of AIS uses two layers: a reactive layer and a cognitive (deliberative) layer. The deliberative layer tries to fulfill real-time requirements by using a satisfying cycle for the traditional cycle of execution of blackboards architectures [11]. This cycle is used to identify KSARs useful for the system in the current moment, to determine which KSAR (task) is going to be executed next and to execute it. The idea is to limit the time of execution of this cycle making it work as an anytime algorithm that can be interrupted if a deadline approaches or if a solution good enough has been found. Our work is based in this architecture but it must be noted that our agent architecture allows more flexibility because the agent decides when and how to deal with the events. For example, our agent can follow efficiently a schedule of tasks while the AIS agent must confirm its decision of following the schedule each time it finishes the execution of a task.

PRS [13; 9] is a system for reactive planning. A set of procedures describes the actions to take in certain situations in the environment or to accomplish objectives. The procedures have a fixed format and they are executed step by step. Reactivity in PRS is very fast, because several procedures can be active at the same time, and the PRS interpreter can choose to execute the step of the most appropriate procedure (according to metalevel knowledge). Plans in PRS, as in AMSIA, are trees of objectives. But in PRS the only planning mechanism is script-based. PRS has been used in the agent architecture Cypress [20] as a robust executor [21] for a planner, to control the interaction with the environment. AMSIA is thought for a different cognitive level than PRS, it sacrifices reactivity to be able to get involved in complex reasoning activities. AMSIA also offers facilities (specification of resources needed by knowledge sources and a control mechanism that deals with them) to manage the resources needed to accomplish its missions. PRS lacks these facilities probably because its interest is in reactivity and robust plan execution.

It is interesting to mention also that techniques such as anytime algorithms [7] and how to build a solution to a problem using a number of anytime algorithms [22], and approximate processing [14] and how to build a solution to a problem based on different methods of different tasks [8], are easily integrated in our architecture.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have proposed AMSIA, an architecture for agents that must combine the use of different artificial intelligence techniques with the ability to fulfill real-time requirements associated with its high level objectives.

The blackboard paradigm, in which this architecture is based, offers interesting properties to build agents for dynamic and complex environments. Namely, because of the division in knowledge sources we have modularity, being easy to add or remove them from the system; also because of the knowledge sources it is easy to combine different reasoning techniques which is useful in open environments (for example, this allows the use of existing software wrapping it in a knowledge source); and we have opportunistic behavior which seems appropriate to operate in dynamic environments.

But this properties are not enough for an agent. First, an agent is going to reason about what to do, and it is necessary to use the results of this reasoning to guide the behavior of the agent. Also, the agent must control its opportunism if it must be sensitive to real-time requirements of its high level objectives. In AMSIA the applications are given the possibility of creating objectives to guide the behavior of the agent. The control mechanism translates them to concrete activity taking into account resource limitations. All the activity in the architecture is explicitly scheduled to control the use of resources. The opportunism is in the acquisition of new high level objectives; in creating, extending and modifying plans of objectives; and in dealing with unexpected contingencies. The time spent in analyzing new opportunities is managed according to the available resources.

Preliminary experimental results with a simulator gave us indications that the mechanisms introduced in our architecture are useful to operate successfully in a dynamic environment. This environment requires different reasoning techniques to carry out the missions presented to the agent.

Also, the missions have a deadline after which it is not interesting the achievement of the mission.

Several lines of future research are open. One of the most interesting is using our simulator to compare agents built using different architectures. We have presented such a comparison in this paper but its interest is limited by the fact that the two compared architectures are of the same style (blackboards).

Other line of future research in what we are specially interested is in using AMSIA to study multiagent collaboration under real-time requirements.

6. REFERENCES

[1] R. P. Bonasso, D. Kortenkamp, D. P. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. In *Proceedings of the 1995 IJCAI Workshop on Agent Theories, Architectures, and Languages*, August 1995. http://tommy.jsc.nasa.gov/er/er6/mrl/projects/arch.

[2] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computacional Intelligence*, 4:349–355, 1988. http://bert.cs.pitt.edu/pollack/distrib/guide.html.

[3] R. A. Brooks. A robust layered control system for a mobile robot. Technical Report A. I. Memo 864, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, September 1985. http://www.ai.mit.edu/people/brooks/papers.html.

[4] L. Brownston. *BBK Manual.* Knowledge Systems Laboratory, Stanford University, September 1995. Report No. KSL 95-70.

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

[6] E. B. Gamble Jr. and R. Simmons. The impact of autonomy technology on spacecraft software architecture: A case study. *IEEE Intelligent Systems*, September/October 1998.

[7] A. Garvey and V. Lesser. A survey of research in deliberative real-time artificial intelligence. *Real-Time Systems*, 6(3):317–347, May 1994.

[8] A. J. Garvey and V. R. Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6), November/December 1993.

[9] A. Haddadi and K. Sundermeyer. Belief-desire-intention agent architectures. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence.* John Wiley & Sons, 1996.

[10] B. Hayes-Roth. A blackboard architecture for control. In A. H. Bond and L. Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 505–540. Morgan Kaufmann Publishers, 1988.

[11] B. Hayes-Roth. Architectural foundations for real-time performance in intelligent agents. *Real-Time Systems*, 2(1/2):99–125, May 1990.

[12] B. Hayes-Roth. An architecture for adaptative intelligent systems. *Artificial Intelligence*, 72(1-2), January 1995.

[13] F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, December 1992.

[14] V. R. Lesser, J. Pavlin, and E. Durfee. Approximate processing in real-time problem solving. *AI Magazine*, 9(1):49–61, Spring 1988.

[15] O. Michel. *Khepera Simulator Package version 2.0.* University of Nice Sophie-Antipolis, March 1996. Freeware mobile robot simulator downloadable from http://diwww.epfl.ch/lami/team/michel/khep-sim/.

[16] J. P. Müller. *The Design of Intelligent Agents, A Layered Approach*, volume 1177 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag, Berlin, 1996.

[17] B. Pell, D. E. Bernard, S. A. Chien, E. Gat, N. Muscettola, P. P. Nayak, M. D. Wagner, and B. C. Williams. An autonomous spacecraft agent prototype. In W. L. Johnson, editor, *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, CA USA, February 1997. ACM Press.

[18] M. E. Pollack, D. Joslin, N. Arthur, S. Ur, and E. Ephrati. Experimental investigation of an agent commitment strategy. Technical Report 94-31, Department of Computer Science, University of Pittsburgh, June 1994. http://bert.cs.pitt.edu/~pollack/distrib/tileworld.html.

[19] I. Soto, M. Ramos, and Á. Viña. A control mechanism to offer real-time performance in an intelligent system. In E. Alpaydin, editor, *Proceedings of the International ICSC Symposium on Engineering of Intelligent Systems (EIS'98).* ICSC, February 1998.

[20] D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI*, 7(1), 1995. http://www.ai.sri.com/~cypress.

[21] S. Źilberstein. Resource-bounded sensing and planning in autonomous systems. *Autonomous Robots*, 3:31–48, 1996. http://anytime.cs.umass.edu/.

[22] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996. http://anytime.cs.umass.edu/.