Scheduling Activity in an Agent Architecture

Ignacio Soto
Dpt. Tecnologías de las Comunicaciones
Universidad Carlos III de Madrid
c/Butarque 15
28911 Leganés (Madrid)
Spain
isoto@it.uc3m.es

Abstract

Agents for applications in dynamic environments require artificial intelligence techniques to solve problems to achieve their objectives. For example, they must develop plans of actions to carry out missions in their environment, in other words, to achieve some state in the world. But also, the agents must fulfill real-time requirements that arise because the characteristics of the applications and the dynamism of the environment. In this paper we analyze the use of a schedule of activity in an agent architecture to control the resources (time) needed by agents to accomplish their objectives.

1 Introduction

An agent must achieve objectives in dynamic and complex environments. To achieve these objectives it must carry out a series of tasks. We call task to a schedulable and executable procedure. A task can be computational, i.e., one that tries to find out other tasks which once executed will eventually let the agent achieve its objectives. Or a task can embody actions in the real world and/or perceptions of the environment.

On the other hand the activity of the agent is conditioned by real-time requirements:

1. The application can have real-time constraints: the agent must fulfill each objective before its deadline.
2. The agent must be reactive in front of events in the environment. Some will need an immediate response by the agent to guarantee its own security, others will allow for deliberation to deal with them (to find out which tasks to execute associated with them).
3. The behavior of the agent must be robust in the sense of always doing useful work. If it has not resources to fulfill all its objectives, it must try to fulfill its most important ones, while not being distracted by objectives it cannot achieve.

Requirement 2 has been the main aim for agent architectures that have been used to build agents that need to interact with a real world environment (for example, controlling robots). Less effort seems to have been made to deal with requirements 1 and 3 (but see section 5 in which we compare our work with other approaches).

In section 2 we describe an agent architecture to fulfill the requirements mentioned above. This agent architecture is based on the blackboard model. We identify the characteristics that this model offers that, we believe, are useful for building intelligent agents that combine the use of different artificial intelligence techniques with real-time requirements. And then, we propose modifications to the basic model that are needed to fulfill these requirements. In particular, we propose that, to be able to deal with resource constraints of high level objectives (missions) of the agent, the agent architecture can benefit from having an schedule of the predicted activity to achieve those objectives. In section 3 we describe the role of the schedule of tasks that defines the activity of the agent and how can be built under real-time constraints. In section 4 we present experimental results about the behavior of the architecture using the schedule. In section 5 we compare the role of the schedule in our agent architecture with the role that plans play in other agent architectures, and comment on other related work. And finally, in section 6 we summarize our results and give directions for future research.

2 Agent Architecture

Our research group has been working in developing an agent architecture to fulfill the requirements mentioned in the introduction. This architecture is called AMSIA.

AMSIA is based on the blackboard model (Corkill, 1991; Carver and Lesser, 1992; Hayes-Roth, 1988; Pfleger and Hayes-Roth, 1997). Using this model, we can divide the knowledge of our agents in a series of Knowledge Sources (KSs). This division has several advantages:

1. **Distribution**: first, of course, we are dividing the activity needed to solve a problem. The parts should be easier to build than the complete solution. Moreover, incremental and/or hierarchical reasoning is natural in this model.

2. **Software reuse**: each part solves a problem and so, it can be reused in different situations where the problem appears and/or in different applications (Hayes-Roth et al., 1995). Application programmers can take the basic architecture and bring or build knowledge sources to deal with their domain problems.

3. **Flexibility**: it allows the agent to use different reasoning methods. Each knowledge source is independent from the others and can be built in any form needed by the application. The knowledge sources doesn't communicate directly. The only restriction is that a knowledge source must be capable of understanding the representation of the knowledge in which it is interested and that will have been left in the blackboard by other knowledge sources.

4. **Estimation of resource requirements**: the division of the activity needed to solve a problem in parts makes easier to estimate resource requirements. The agent can do this estimation separately for each part, and it can compensate the resource use of different parts. Also, real-time artificial intelligence techniques, such as anytime algorithms or approximate processing, can be integrated smoothly in knowledge sources.

In AMSIA, we have refined the traditional blackboard model with two new properties:

1. All the activity in the system is explicitly scheduled. With the term activity we refer both to actions in the real world and to actions internal to the agent (i.e. reasoning activities including planning). This is the base to control the use of resources.

2. We make independent in the agent the following of a line of activity which, at the same time, generates possibilities of activity for the future, from the decision of what line of activity must be followed.

We believe that the second property defines an important division needed to achieve real-time performance. The line of activity of an agent represents its committed resources. It defines a behavior with some profit for the agent. Choosing future lines of action is the act of committing resources to achieve some profit. The separation of these two activities allows the agent to control its opportunism.

In the past we have explored achieving this division using a multiprocessor architecture for our agent (Soto et al., 1997, 1998). We used a processor to follow a line of activity and offer new ones; and another to analyze the possibilities that were created by the agent by following its line of activity, and to choose the future line of activity of the system. We continue working in this architecture but, in this paper, we explore another approach to the problem, namely we study how AMSIA achieves the mentioned division in time, and not with the use of two processors. In this architecture the own schedule of future activity of the agent must include time to consider and choose among possibilities of future activity. This is not easy because there are situations in which the agent doesn't know when possibilities for future activity are going to be opened. We study how to deal with this situation in next section.

To predict future activity the agent must use planning techniques. In AMSIA, reasoning tasks can create plans of objectives; and control tasks can translate those to plans of tasks (to achieve the objectives), assign them resources, and introduce them in the schedule. Decisions can be delayed simply by using a reasoning task to decide what to do about an objective in the right moment, perhaps extend it in a series of sub-objectives. Changes in the plan of objectives are easy because they are in the blackboard and can be accessed by any task. Changes in the method (task) to achieve an objective are also easy because the alternative tasks are kept associated with the corresponding objective.

Figure 1 shows the conceptual model of AMSIA. Notice:

1. Control and execution are independent activities according with property two above, but both of them get its time of execution from the schedule that defines the activity of the system.

2. Both control and domain actions have preconditions. This is a check to ensure that the conditions expected by the task to be executed are really so when it is going to be executed. If they are not, the task is not executed and an external (see bellow) event is generated. Soto et al. (1998) presents a more detailed discussion of this issue.

3 Scheduling Tasks in AMSIA

3.1 Construction of the Schedule

To have a schedule of activity allows AMSIA to control the use of resources. The problem is how to build this schedule.

In AMSIA, activity is triggered by events. These events signal that something interesting has happened. They represent changes in the blackboard that can be consequence of a reasoning activity or of perceptions in a broad sense: we consider perceptions readings from sensors but also messages from other agents or a timer that expires.

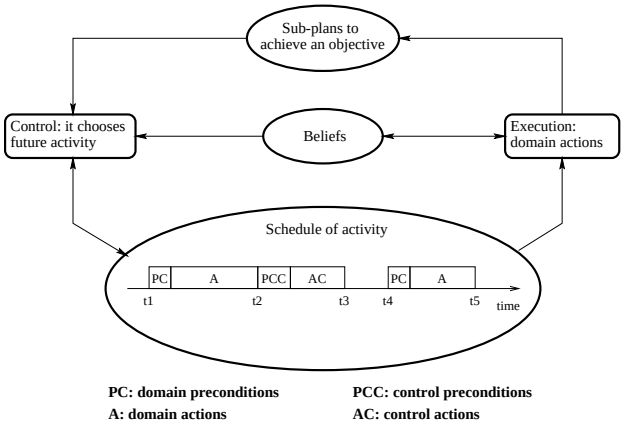

Figure 1: The Conceptual Model of the Agent Architecture

For each event there will be a number of KSs whose knowledge can be useful in that situation. The agent identifies those KSs, creates tasks based on them, builds possible sequences of those tasks to do the work needed in front of the event, and then it must add one of the sequences to the global schedule that defines its (of the agent) future activity. Different sequences will make different trade-offs in resource usage and quality of expected results. The schedule registers the resources allocated to the tasks. In our implementation the only resource considered is time and so, it is kept in the schedule the instants before which the execution of each task must begin and end.

The activity needed to deal with an event (identify KSs, create tasks, build sequences of tasks, and introduce one in the schedule) is too complex to be done in a fixed or negligible time. Instead, this activity must be scheduled itself, i.e., a task to deal with the event, to do that activity, must be included in the schedule. To do so, we divide the events in two different kinds:

- internal: events internal to the reasoning flow of the agent;
- external: events external to that flow.

Internal events are created by the reasoning activity of the agent. They show the need/possibility of using new tasks to develop the reasoning work in which the agent is involved. For example, the execution of a task in certain level of abstraction can discover that it is needed the execution of several tasks in a lower level of abstraction. So, internal events can be anticipated by the agent and it must include in the schedule of activity a task to deal with them.

But there are also events that aren't produced by the reasoning activity of the agent. We call them external events. Examples are certain situations perceived in the environment, or a message from other agent. The situation is the same as before in the sense that the agent needs to execute a task to deal with the events. The difference is that the agent cannot anticipate these events and so, it cannot have in the schedule tasks to deal with them. The solution is that, when an external event is received by the agent, asynchronously, it must include a task in its schedule to deal with it.

The agent can control its openness and reactivity in front of events because it decides when and how it is going to deal with them.

The scheduler works with the algorithm that is shown in figure 2.

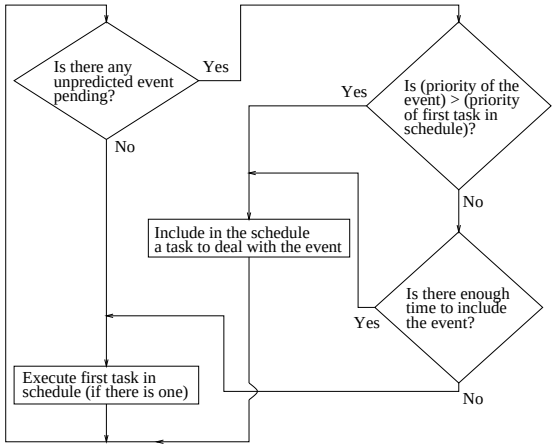

Figure 2: Algorithm of the scheduler

The scheduler is non-preemptive (it works between tasks, not when an event is received, which is reasonable in deliberative tasks but see section 6 conclusions and future work) and dynamic (of course it doesn't know the future time of arrival of new events to the agent).

3.2 Example of Schedule Construction

In figure 3 it is shown an example of the algorithm working. We begin with an empty schedule. An external event is received and, hence, the scheduler adds a task to the schedule to deal with it. To assign time to this task the scheduler has the information of the kind of event and (possibly) the time that has spent in tasks to deal with the same kind of events in the past (more on this later). This task is then executed resulting (in this example) in the scheduling of two new tasks. The scheduler algorithm is run, as there is no new external events, the next task in the schedule (task number two in the figure) is run. This is a deliberative task and as a result of its reasoning activity internal events are generated.

There are not external events and so next task (task three) is executed. This task is in the schedule to deal with the internal events generated by task two. As a result, new tasks are added to the schedule.

Task four is again a deliberative task that generates internal events. A point to notice is that the agent can predict the time that is going to need to execute not only task four but also the tasks that task four identifies for

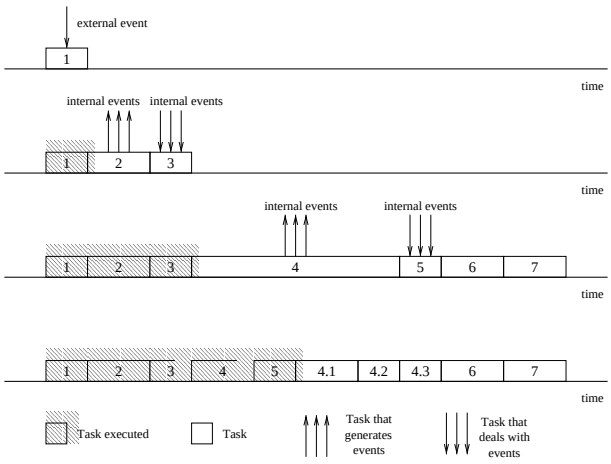

Figure 3: An example of a schedule

execution. This is useful because is a reserve of resources that allows to know early if the agent is going to have resources enough to execute the plan, and it simplifies the work of scheduling the tasks identified by the execution of task four.

3.3 Estimations of Execution Time of Tasks

An important problem is how to assign time to the tasks in the schedule, mainly because most of them are deliberative or represent complex actions in the environment (not a primitive action but a reactive module to achieve some state in the environment). We are not trying to answer this question here. Our architecture offers the means to apply the solutions proposed elsewhere. For example:

- Anytime algorithms: they can be interrupted at any moment and they guarantee to offer a result, although more time of execution will mean results with more quality. They have associated performance profiles that indicate the expected quality of results in function of the time of execution. Tasks can be constructed as anytime algorithms giving the tasks that add them to the schedule the flexibility of assigning them time to get certain quality. And tasks to deal with external events can be anytime algorithms so they can be executed the available time.

- Approximate processing: our architecture integrates very easily the possibility of having several methods to do the same thing. The task that deals with the event will choose according to resource constraints and quality requirements. We can also have several methods to deal with external events and use an heuristic in the scheduler to choose among them.

The control mechanism of AMSIA schedules sequences of tasks (and not individual tasks) and so, real-time artificial intelligence (Musliner et al., 1995; Garvey and Lesser, 1994) techniques can be applied.

Usually we will use estimations for the execution time of tasks. These estimations will be based in the history of the agent and can be changed dynamically. This is necessary both because the dynamism of the environment that can condition the time needed to do some task, and because, using learning techniques, the agent can learn to do certain tasks faster.

Of course estimations can be wrong. There are two protections to errors in the estimations of time of execution of tasks in our architecture:

1. Little deviations can be compensated with available time in the schedule or with execution time of other tasks of the same plan.

2. Greater deviations can be dealt by using monitoring. A great deviation will be detected and an external event will be generated to repair the schedule. Currently we do monitoring between tasks because we do not consider preemption. The tasks themselves must be build so that they have a maximum execution time (but see future work in section 6).

Moreover, AMSIA supports an hierarchical application of knowledge using internal events to identify tasks to work in other level of abstraction. This is interesting also because when the agent has a plan at a certain level of abstraction, it has resources (time) assigned to it. The execution of the tasks at that level generates tasks in a lower level that define more exactly the resource needs (possibly inside the resources previously reserved, see tasks 4, 4.1, 4.2, 4.3 in figure 2, although perhaps with some kind of adjustment). Then, as the agent spends more time in a plan, it has more exact idea of the resource requirements of that plan and, so, it is less probable that the agent had to abort the plan due to underestimation of resource requirements.

Also it is important that the reasoning model of the agent is incremental, the agent has a plan (schedule) and it works adding and removing pieces to that schedule. Resource estimations are not global, hence, they are easier to do and to compensate in case of error.

3.4 Conflicts in Resources Assignment

It is possible that, when the control mechanism of AMSIA tries to introduce a sequence of tasks in the schedule, there are not resources (time) enough to do it. To solve these conflicts, the control mechanism of AMSIA scores all the sequences of tasks. The score depends on the plan the sequence of tasks is trying to achieve, and the particular tasks that are part of the sequence. When there is a conflict, the control mechanism tries to free time in the schedule by removing the sequences of tasks with the smallest score and that are in conflict with the one that it is being introduced. External events are generated to signal

the removing of these sequences of tasks, and so, later it can be considered their re-introduction. This is an heuristic process, but it only happens when there are resource conflicts and it favors the most important plans.

4 Experimental Work

In this section we are going to show the results of an experiment developed to study the robustness of our agent in front of errors in the estimations of the duration of the tasks of the schedule.

We have implemented the proposed agent architecture modifying BBK (Brownston, 1995), a C++ implementation of the blackboard architecture for control (Hayes-Roth, 1988), and adding the mechanisms described in this paper. We have applied it to control a simulated robot (a modified version of the Khepera simulator (Michel, 1996)) 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. Not all the missions are of the same importance to the agent, in case of resource shortage it is better for the agent to abandon missions with low importance to favor the accomplishment in time of missions of higher 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.

To operate in this environment and to successfully accomplish its missions the agent needs to implement several functionalities. It must be able to act: to move (using its two motors), and to repair faults. It must be able to sense: obstacles in its path, the state of a fault, and messages telling the agent the missions that it must accomplish. It must be able to *reason*: planning how to accomplish its missions, path planning for discovering how to go to its destinations, and diagnosis of faults (using an expert system). All this functionality is implemented as knowledge sources in our architecture. For example, the agent has a knowledge source for going from one point to another, this knowledge source controls the speed of the motors of the robot and attends to its sensors. Robot sensors offer raw data that must be processed by the knowledge source to deliver symbolic information.

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 $\bar{t}_{OD}$.
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* $= \frac{\textit{Score obtained by the agent}}{\textit{Total score offered to the agent}} \times 100$.

   where,

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

   Missions accomplished refers to those accomplished *before* their deadlines.
2. *Mission effectiveness* $= \frac{M_{aa}}{T_{oa}} \times 100$

   where, M_{aa} is the number of missions accomplished by the agent, and T_{oa} is the total number of missions offered to the agent.
3. *Importance effectiveness* $= \frac{M_{aahi}}{T_{oahi}} \times 100$

   where, M_{aahi} is the number of missions accomplished by the agent of the highest importance, and T_{oahi} is the total number of missions offered to the agent of the highest importance.

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:

Factor	values
Mission dynamism	high, low
Importance range	medium
Deadline	big
Obstacle dynamism	low
Time estimation	high, medium, low, very_low

Table 1: Independent variables in the experiment

Missions dynamism $= high \Rightarrow \bar{t}_M = 275$
Missions dynamism $= low \Rightarrow \bar{t}_M = 600$
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 $\bar{t}_{OD} = 100$

Time estimation *medium* means that the average execution time of each task (measured in the preliminary experiments) is used as estimation of the expected execution time of that task. Time estimation *high* means that estimations 15% over the average values are used, *low* means 15% under the average values, and *very_low* 25% under the average values.

The results of the experiment are shown graphically in figure 4, where we have separated the situation with dynamism high and low. An analysis of variance shows that the factor *time estimation* has significant influence in the three dependent variables: effectiveness (for missions dynamism=low F=4.5673, P=0.0171; and for missions dynamism=high F=4.4002, P=0.0194), missions effectiveness (for missions dynamism=low F=5.2067, P=0.0031; and for missions dynamism=high F=4.0605, P= 0.0253), and importance effectiveness (for missions dynamism=low F=7.0520, P=0.0031; and for missions dynamism=high F=7.9768, P=0.0018).

The shape of the curves in figure 4 is what we expected. The architecture achieves a profit of its time estimations, hence, the effectiveness measurements have a maximum at one point, and go down at both sides of that point. If time estimations are too high, this results in that missions which could have been tried are not, because the agent thinks that it has not enough resources. If estimations are too low, the agent tries missions that finally are not achieved because of lack of resources (or they are achieved after their deadlines).

However, when the missions dynamism is low, the maximum of effectiveness and mission effectiveness is not achieved using as time estimations the average time of execution of tasks, but a lower value. The reason for this is the flexibility that the agent architecture has to deal with errors in time estimations. If missions dynamism is high the agent architecture has more problems to deal with error in time estimations, there are few time available in the schedule and the missions in it are of high importance.

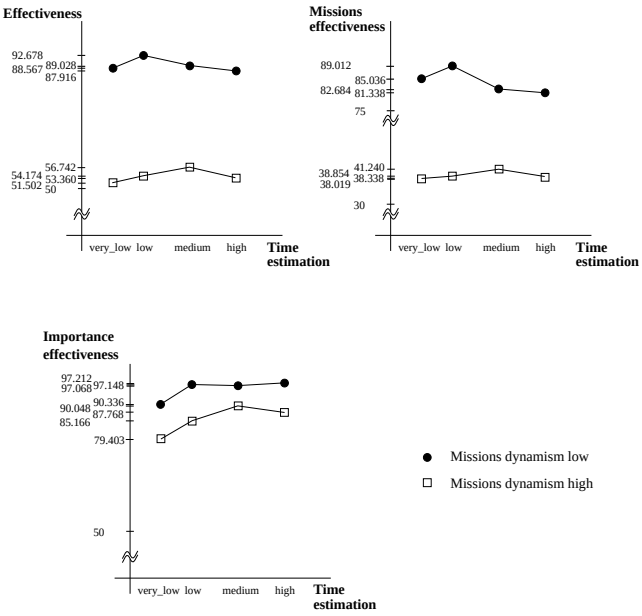

Figure 4: Results of the experiment

The only solution left is to use tasks with less quality (but that need less time) to achieve the missions. The problem is that these tasks sometimes are going to fail preventing the achievement of the mission.

We can conclude the following from this experiment:

1. The estimation of execution time of the tasks has influence in the performance of the agent architecture. Hence, a better estimation improves the performance. However, errors in estimations doesn't provoke an abrupt fall in performance because the mechanisms that the architecture has to deal with these situations.

2. As the missions dynamism (the number of missions that the agent is facing) is decreased, it is better to be optimistic in time estimations. These allows the agent to try more missions, and it has enough flexibility to deal with situations of error in the time estimations. If mission dynamism is increased, time estimations must be more exact to get higher performance. Notice that the agent architecture can calculate dynamically the estimations of the time of execution of its tasks; for example, it can be more or less conservative according to the perceived missions dynamism.

5 Related Work

Plans or schedules have different roles in different agent architectures.

Reactive architectures, as the subsumption architecture (Brooks, 1985), don't use plans, and so, it doesn't

seem easy, using this kind of architecture, to build an agent to fulfill certain real-time requirements of high level objectives.

Hybrid architectures as InteRRaP (Fischer et al., 1995; Müller, 1996), TouringMachines (Ferguson, 1992), or RemoteAgent (Gamble Jr. and Simmons, 1998), use a reactive 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 makes plans to achieve the high level objectives of the agents, negotiate with other agents, etc. These plans are built off-line and, afterwards, executed. 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). Nonetheless, the idea of a reactive layer to manage the direct interaction with the environment seems a good one (see future work in section 6).

IRMA (Bratman et al., 1988; Pollack et al., 1994) 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. But, to keep openness in front of external events, an override process allows options incompatible with the current plan but highly promising to pass the filtering process to let the agent deliberate about then (about changing the current plan). 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.

Our work differs from recent advances in planning and scheduling (as for example in Chien et al. (1998)) in that our main aim is in the integration of planning and execution. In fact, in our system, planning is an activity as any other and must compete for the resources of the agent, the result of this activity are plans that guide the future behavior of the system. Plans keep its causal structure and can be analyzed or modified at any time, but the schedule is highly committed to simplify control operations and because replanning is based on the plans, not on the schedule. AMSIA can adapt its planning activity to the circumstances (for example it can choose a predefined plan because there is not time to generate a better one).

As it was mentioned before, techniques such as anytime algorithms (Garvey and Lesser, 1994) and how to build a solution to a problem using a number of anytime algorithms (Zilberstein, 1996), and approximate processing (Lesser et al., 1988) and how to build a solution to a problem based on different methods of different tasks (Garvey and Lesser, 1993), are easily integrated in AMSIA.

6 Conclusions and Future Work

In this paper we have analyzed the role of a schedule of activity to guide the behavior of an agent. This agent must use different reasoning methods under real-time requirements associated with its high level objectives.

All the activity in AMSIA is explicitly scheduled as a way of controlling the use of resources. Also, the activity to choose a line of action is separated from the activity of following that line of action and offering new possibilities for future action. We believe this is an important property for agents that must fulfill real-time requirements. The line of action focuses the attention of the agent that, independently, considers changing that line of action, i.e., it keeps its opportunism. In other work (Soto et al., 1997, 1998) we have explored the idea of separating these activities in hardware. In this paper we explore the division of these activities in time. To do so, the activity needed to choose a line of action must be included as a series of tasks in the schedule of the agent. A mechanism (external events) is added to deal with unexpected events, i.e., to include tasks in the schedule to consider what to do in front of those events.

Also, there are options for AMSIA that we want to explore:

- The use of a preemptive scheduler. This means that we need to be able to interrupt the execution of tasks. The problem is that it is not easy to keep the consistence of the knowledge in the blackboard when a reasoning task is interrupted. There are solutions as using sections of code where an interrupt is impossible to make changes in the knowledge state of the system.
- We have used our agent architecture to control a simulated robot. In a real environment we will need a reactive layer to augment the reactivity in face of contingencies.
- We want to extend the information that is kept in the schedule. For example, it will be interesting to register other temporal constraints for the execution of tasks. Although this will complicate the heuristics used in schedule construction, this is not a critical problem because this activity is also scheduled.

Acknowledgements

The author is grateful to Mercedes Garijo and Carlos Ángel Iglesias for useful comments on this work. The author also wishes to thank the anonymous reviewers for useful comments on the abstract of this paper.

References

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

Rodney 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.

Lee Brownston. *BBK Manual*. Knowledge Systems Laboratory, Stanford University, September 1995. Report No. KSL 95-70.

Norman Carver and Victor Lesser. The evolution of blackboard control architectures. Technical Report UM-CS-92-071, University of Massachusetts, Amherst, 1992.

Steve Chien, Benjamin Smith, Gregg Rabideau, Nicola Muscettola, and Kanna Rajan. Automated planning and scheduling for goal-based autonomous spacecraft. *IEEE Intelligent Systems*, September/October 1998.

Daniel D. Corkill. Blackboard systems. *AI Expert*, 6(9), September 1991. Also as *Technical Report, Blackboard Technology Group Inc.*

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

Klaus Fischer, Jörg P. Müller, and Markus Pischel. Unifying control in a layered agent architecture. In *Proceedings of the IJCAI 1995 Workshop on Agent Theories, Architectures, and Languages*, 1995. Also as DFKI GmbH Technical Memo RR-94-05, http://www.dfki.uni-sb.de/mas/papers.

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

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

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

Barbara Hayes-Roth. A blackboard architecture for control. In Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 505–540. Morgan Kaufmann Publishers, 1988.

Barbara Hayes-Roth, Karl Pfleger, Philippe Lalanda, Philippe Morignot, and Marko Balabanovic. A domain-specific software architecture for adaptative intelligent systems. *IEEE Transactions on Software Engineering*, 21(4):288–301, April 1995.

Victor R. Lesser, Jasmina Pavlin, and Edmund Durfee. Approximate processing in real-time problem solving. *AI Magazine*, 9(1):49–61, Spring 1988.

Olivier Michel. *Khepera Simulator Package version 2.0*. University of Nice Sophie-Antipolis, March 1996. Freeware mobile robot simulator downloadable from http://wwwi3s.unice.fr/~om/khep-sim.html.

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

David J. Musliner, James A. Hendler, Ashok K. Agrawala, Edmund H. Durfee, Jay K. Strosnider, and C. J. Paul. The challenges of real-time AI. *IEEE Computer*, January 1995.

Karl Pfleger and Barbara Hayes-Roth. An introduction to blackboard-style systems organization. Technical Report KSL-98-03, Knowledge Systems Laboratory, Stanford University, 1997. http://www-ksl.stanford.edu/publications/index.html.

Martha E. Pollack, David Joslin, Nunes Arthur, Sigalit Ur, and Eithan 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.

Ignacio Soto, Manuel Ramos, F. Javier González, and Ángel Viña. Arquitectura multiprocesador para sistemas inteligentes adaptativos. In Martín Llamas, José J. Pazos, and Manuel J. Fernández, editors, *Actas de las V Jornadas de Concurrencia*, pages 161–175, Vigo, Junio 1997. Servicio de publicaciones de la Universidad de Vigo. (In Spanish).

Ignacio Soto, Manuel Ramos, and Ángel Viña. A control mechanism to offer real-time performance in an intelligent system. In Ethem Alpaydin, editor, *Proceedings of the International ICSC Symposium on Engineering of Intelligent Systems (EIS'98)*. ICSC, February 1998.

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