CPU Sharing for Autonomous Time-Aware Agents

Cédric Dinont[1,2], Emmanuel Druon[2], Philippe Mathieu[1], Patrick Taillibert[3]

1: Laboratoire d'Informatique Fondamentale de Lille
59655 Villeneuve-d'Ascq Cedex - FRANCE
2: Institut Supérieur de l'Electronique et du Numérique
41 Bd Vauban 59046 Lille Cedex - FRANCE
3: Thales Aerospace Division
2 Avenue Gay Lussac 78990 Elancourt - FRANCE

Abstract: In this paper, we investigate time-aware agents programming. A time-aware agent is able to manage several tasks working on disparate data. It reasons about its tasks durations and it is also able to cooperate with other agents to share CPU resources in order to meet its deadlines. Time-awareness should not be at the expense of agent autonomy, to which we must pay attention at programming and execution levels. Moreover, time-aware agents may need to quickly react to data provided by sensors. Taking into account these objectives leads us to introduce two agent classes: extrovert agents which are always turned towards the external world and introvert agents which allow to guarantee tasks durations. We show that our architecture enables autonomous time-aware agents to share a single CPU. We also introduce a Time Services Agent (TSA) which collects CPU usage requests and returns time slots ensuring deadlines meeting.

Keywords: Autonomous agents, CPU reservation, reactive processing.

1. Introduction

1.1 Autonomous Cognitive Agents

One of the major concepts brought by the agent paradigm is autonomy [1]. It can be defined as agent capacity to decide itself of its actions. The respect of temporal constraints in a context of autonomy raises a lot of difficulties, either at agent or MAS level. As we evolve in the Distributed Artificial Intelligence field [2], the reasoning of our cognitive agents can make use of complex algorithms to solve planning problems for example. A problem search space can have critical regions: for a small variation on input data, the computing time to find the solution can tremendously vary and even become prohibitory. Needed processor resources are thus very variable and difficult to envisage. Numerous problems are solved by using more or less powerful heuristics. It is not easy to predict that a heuristic will give a better result and/or more quickly than another heuristic, just according to the input data. When the choice of a particular heuristic becomes too hard, one comes to start several of them at the same time. We stop them as soon as one of them has found a satisfying solution. Our proposal makes it possible, among others, to answer this kind of problems.

We expect from agents that they are aware of their environment. They must permanently be turned towards the external world to quickly take into account changes in the environment and quickly answer other agents messages. For software agents, it consists in regularly having enough processor time which they will use to update their state according to their environment. Guessoum and Dojat presented in [3] an agent model which mixes cognitive and reactive behaviour aiming at meeting hard deadlines.

We now define active and inactive periods of the agents and we associate an "agent time" clock to agents which allows to evaluate the quantity of work that an agent has already carried out at a given time instant.

Definition 1 (Active agent / inactive agent) *An agent is inactive over a period of time if it does not have any access to the processor during this period. Otherwise, it is active over the considered period.*

Definition 2 (Agent time) *The agent time advances at the rhythm of the operations executed by the agent.*

The agent time belongs to each agent: there is an "agent time" clock per agent. Each can advance at a different rhythm. Note that if agents use agent time to reason about their processing durations, their reasoning will be independent of the CPU power and the other agents that may share

the CPU. This thus allows to program agents regardless of the execution environment. However, possible reasoning could only be local to the agents. We will later introduce another concept to allow reasoning at MAS level.

As we have just seen, agents get involved in processings which durations (in agent time) are not necessarily known before they were executed. An agent that wants to have control on what it is doing will not execute its processings completely and in one run. It will rather decide to start them during a certain time, analyze the partial result obtained and possibly start them again for a new period of time.

Definition 3 (Processing / task) *A processing is a computation work. A task is the execution of a processing during a fixed agent time.*

A processing can be done with one or several tasks. When a task ends, a new task can continue the processing where it was stopped.

1.2 Time-Aware Agents

Cognitive agents continuously reason to know which goals to achieve and how to achieve them. Time is a part of the reality agents must be aware of to make right decisions. We thus define a time-aware agent as follows:

Definition 4 (Time-aware agent) *An agent is time-aware if its behaviour depends on time which runs.*

An agent does not only need to reason about time to be aware of time that runs. It also needs to reason about the passing of time during its actions and decision making. The knowledge of the passing of time is gained by consulting the time from a clock. The chosen clock does not matter. Moreover, it is not necessary to be the same for all the time-aware agents of a same MAS.

A time-aware agent is able to plan its processing, estimate their durations in agent time, launch tasks to carry out these processings, monitor them and make decisions according to its observations [4]. An agent which breaks a processing into several tasks will, for example, be able to detect that the processing lasts for an unusually long time. It needs to be able to go on working without the data it was waiting for from a processing output if it chooses to stop it before it is finished. Let us also notice that an agent does not need to be always active to be time-aware.

Taking into account the duration of processing is a recurrent problem in the AI and MAS fields [5]. *Anytime algorithms* [6] have a function indicating the answer quality according to time. When they exist for a given problem, their use allows agents to reason about the tasks durations they allocate to problems and even to make it a rule to meet constraints on solution quality. Zilberstein et al. present in [7] a meta level to reason about the uncertainties linked with the durations and the answer qualities. Adelantado et al. add in [8] a reactive mechanism to anytime agents that enables them to be used in embedded real-time applications. Other reasoning can also be done on the necessary compromise between computing time and answer utility as in [9]. Wagner details in [10] design-to-time proposal [11] and its extension design-to-criteria [12] which allows to take into account the processing durations and other criteria for planning. Finally, Prouskas presents in [13] an extension of April which reifies two different levels of temporal constraints: temporal constraints between agents and between humans and agents. The periods of time considered are longer in the case of interactions with humans. Real-time constraints are also less strong when they are linked to a human agent. Prouskas also gives a definition for time-aware agent that is appropriate to applications where temporal constraints are dictated by the interactions with human agents.

We finally notice that a real-time process executed on a real-time system is not inevitably time-aware. It was just programmed to give an answer in a bounded time.

1.3 Agents Sharing One CPU

An architecture for time-aware agents must provide the means to deal with several agents sharing one CPU.

One of the roles of the operating system is to manage the scheduling of tasks on the available processors. This is differently done according to seeked properties. Thus, systems like Windows or Unix, said to be time-sharings, guarantee that all processes will be able to regularly have a quantum of processor time. What is searched for here is the illusion, at user abstraction level, that all processes are excuted at the same time. Unfortunately, these systems do not guarantee anything on tasks deadlines.

This guarantee is on the other hand given by real-time systems, but this is often at the expense of the illusion that processes are always active. Indeed, algorithms classically used like Earliest Deadline First (EDF) [14, 15] will instead launch a task until it is finished and then go to the next one rather than to assign small quantums of time to each task. We do not investigate more real-time systems as our purpose is to run our agent systems on top of classical systems used in workstations.

The management of the conflicts of processor allocation is also a significant issue which is among others approached by [16]. In a MAS, these conflicts are preferably managed at multi-agent level. We also can use interaction and negociation mechanisms to process them while taking into account the autonomy of agents. If this is the operating system which decides what to do in case of conflict, it will make decisions which will go against the principle of autonomy. For example, and that is the most common case, consider that the operating system or the MAS framework uses priorities to manage conflicts. When an inconsistency appears in the scheduling, we see the authoritative ousting of the tasks of weaker priority. However, we want, when an agent decides to launch a task, to have the guarantee that it will finish as foreseen. That must be valid for all the agents, without any distinction. The principle of autonomy goes against the management of conflicts by priorities and submission to the authority of a third party. Indeed, two agents can rightly think that their tasks have high priorities. If there is a conflict for the execution of a new task, it is preferable that an agent decides to sacrifice itself by stopping one of its tasks or that this is the new task which cannot be carried out if no agent wants to sacrifice itself.

Ultimately, we want agents to be turned towards the external world and to be able to ensure temporal constraints in a context of autonomy and of sharing a single CPU. Because of their autonomy, agents can decide to launch a task and want it to be finished before a given date. We need a means to ensure that when an agent execute a task, other agents let it enough processor resources to finish its task before its deadline i.e. we need a means to know if the set of temporal constraints linked to active tasks is satisfied. We also need a way to manage conflicts which arise when a new task request brings an inconsistency.

2. Proposed Architecture

For the remainder of this paper, we choose a granularity such that an agent has only one execution path i.e. one agent can be executed by one process or one thread but cannot consist of several concurrent processes or threads.

2.1 General Description

We classify agents into two exclusive classes defined in the next paragraph: extrovert agents and introvert agents. These classes are characterized by the deadline separating two consultations of their mailboxes. Extrovert agents reason about the temporal behaviour of introvert agents. They also delegate to them the realization of long processings as tasks. They finally have a link which allows them to suspend and to start again introvert agents at any time.

The Time Services Agent (TSA), described in section 3, is a particular extrovert agent. Other extrovert agents have to interact with it when they want to launch a task. It collects temporal constraints of all tasks carried out by introvert agents and it commits itself so that they are checked. In particular, it finds a schedule for the tasks and indicates the corresponding execution time slots to introvert agents. If they respect them, the TSA guarantees that all tasks terminate before their deadlines.

2.2 Agent Classes

Definition 5 (Extrovert / introvert agent) *An agent is extrovert if the agent time which separates two checks of its mailbox is bounded. Otherwise, it is introvert.*

This definition implies that an extrovert agent cannot blindly commit itself in long processings which would prevent it from consulting its mailbox regularly. It is sometimes possible to envisage the regular consultation of the messages in a processsing loop. In this case, an agent will be able to launch long processings. That is however not possible in all cases, in particular when using legacy code or when using processings like constraints solving. For this kind of processings, once launched, it is generally impossible to suspend them just to consult the mailbox. The extrovert agents will delegate the execution of the processings that cannot be interrupted to introvert agents.

We can also notice that an extrovert agent cannot have idle periods longer than the limit fixed between two checks of the mailbox. On the contrary, an introvert agent can have unlimited idle periods. Afterwards, we will particularly focus on extrovert agents that are permanently active. Timesharing is the only property required from the operating system so that agents can be regarded as constantly active. Indeed, if we place all the agents at the same priority with respect to the operating system scheduler, each agent will have regularly a little processor time. Just like the user of such a system who considers that its various programs work permanently, we can consider that some agents are permanently active. That obviously only becomes true for sufficiently long periods of time.

2.3 Respect of Temporal Constraints

Separation into two agent classes makes it possible to reach part of the goal that we set ourselves: to carry out long pro-

cessings while having agents permanently turned outside. That is not enough to ensure that the long processings check temporal constraints like deadlines. As we will further see, the deadline management will be done by means of a particular agent which will indicate which tasks must be carried out and which tasks must be paused at a given moment. That implies that introvert agents can be paused and started again as needed. The agents mail system cannot be used for this purpose, because once an introvert agent has started the execution of a task, it does not react any more to the messages which accumulate in its mailbox. Thus, an additional means becomes necessary to ensure the suspensibility of the agents.

Definition 6 (suspensibility) *An agent can be suspended if it is possible at any time to order him to become active or inactive.*

A suspensible agent remains autonomous, e.g. it can decide not to answer a message, but it has to suspend itself if it is asked to. When an agent is suspended, it is inactive and thus does not use the processor any more. Especially, it is not able any more to give its progress report to the extrovert agent which controls it. If the knowledge of this report is needed by the temporal reasoning done by the extrovert agent, it must take advantage of its conscious moments to communicate its progress report to it.

We note that introvert agents have to be suspensible so that a management of deadlines can be set up. We should accept that agents lose a part of their autonomy, here by agreeing to become inactive, so that the other agents can be carried out and thus respect their deadlines. This property is not needed for all agents. It does not concern extrovert agents since they must be constantly active.

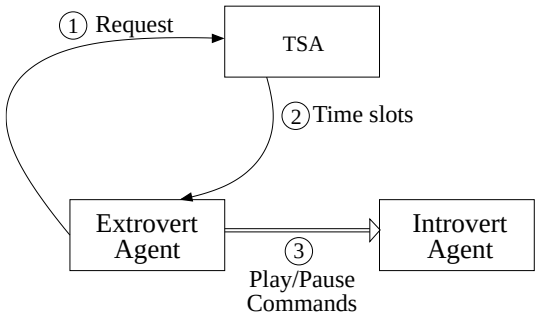

Figure 1: **Main relations between the TSA, extrovert agents and introvert agents.**

We finally note that agent properties described up to now (active, time-aware, extrovert and suspended) are intrinsic to the agent and independent of the execution environment.

3. The Time Services Agent

We have seen that each agent has its own clock indicating its "agent time". It also needs to refer to an unspecified clock to be aware of time. The TSA collects tasks deadlines from the agents. These deadlines must all refer to a common clock. For this purpose, we define the "TSA time".

Definition 7 (TSA time) *The TSA time is a common time to all agents that use the services of the TSA.*

We would often choose the system clock to give the TSA time, but it may also be a virtual clock which gives it, for example in the case of simulations. We have seen that the agent time could be used by a reasoning that needs to be independent of the processor and the other agents. On the contrary, the TSA time is used by a reasoning that depends on the current CPU load.

3.1 Task Request Protocol

An agent which wants to allocate a task to advance a processing does a request to the TSA in the form $[TE,\ TCs]$ where TE is the asked agent time for the task and TCs is a list of temporal constraints. The TSA evaluates if the new temporal constraints do not bring inconsistency. If an inconsistency is detected, the TSA returns a refusal to execute the task to the requesting agent. The last can send a new request if it wants to. If the request does not bring inconsistency, the TSA returns the list of time slots (in TSA time) to the requesting agent during which the task can be carried out. The requesting agent must honour these time slots by sending orders of suspension and resuming towards the agent which will carry out the task. An agent can at any moment tell the TSA that it gives up using some of the time slots which were attributed to it.

3.2 Scheduling

Required Properties: The scheduler that we need must have particular properties. It is constructed on top of the operating system scheduler. It can take into account at its level that several tasks can be carried out in parallel. Especially, it must particularly manage always active extrovert agents. It also must manage deadlines, while being easily extensible to other temporal constraints.

We consider that it is interesting to begin tasks as soon as possible. The behaviour of a scheduler like EDF, which will entirely execute the task whose deadline is the earliest and then turn to the next task, is not adapted to the resolution of a problem by various agents running in parallel different heuristics. In this kind of application, it is interesting to start the various heuristics as soon as possible and to stop the resolution as soon as an agent has found the required solution.

Modeling: Requests are sudmitted as $[TE_i,\ DL_i]$. TE_i is the agent time to affect to task i before deadline DL_i. For TE_i, we may also say that it is the total processing energy to assign to task i. We consider n requests whose deadlines are sorted in ascending order: $DL_i \leq DL_{i+1}, \quad 1 \leq i \leq n-1$. We get n intervals: [T_0 to DL_1, DL_1 to DL_2, ...,DL_{n-1} to DL_n], with T_0 the scheduling beginning date. We also consider a fixed number NEA of always active extrovert agents.

Our aim is to respect deadlines: for interval i, the goal is thus to finish task i before the end of interval i. We also want to give as much as possible processor time to tasks $i+1$ to n, as soon as possible in the interval. We split the considered interval into subintervals corresponding to all possibilities to run task i in parallel with one or several other tasks from $i+1$ to n. For the ith interval, there are 2^{n-i} subintervals. We arrange the subintervals in decreasing order of the number of tasks included in them. For a subinterval j of the interval i having T tasks, each task is given a share $P_{i,j}$ of the available CPU power equal to $\frac{1}{T+NEA}$.

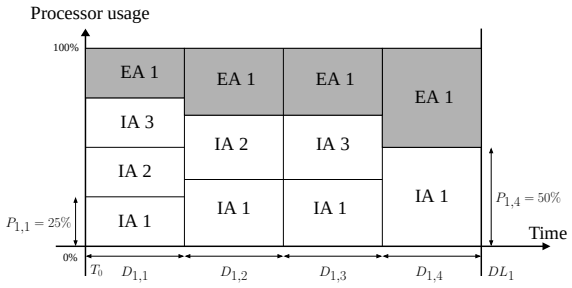

Figure 2: **Partitioning of the first interval with 3 introvert agents and 1 extrovert agent.**

This partitioning of an interval i fits well with our goal: priority is given to finishing task i since it appears in all the subintervals and if we can execute one or more other tasks in parallel to task i, it will be done as soon as possible in the interval. The problem for the ith interval thus consists in determining the durations $D_{i,j}$, $D_{i,j}$ being the duration of the subinterval j of the interval i. $D_{i,j}$ can equal 0.

Resolution Preparation: The resolution is done interval by interval. For each interval, we calculate the variables needed to constitute a system of equations and inequations. We then resolve it with the simplex algorithm [17].

We first define the limits of the considered interval. The end date always equals DL_i. The real beginning date BD_i can differ from DL_{i-1} if task $i-1$ has finished before the end of its interval. It is obtained by:

$$BD_1 = T_0 \qquad \text{and}$$
$$BD_i = \sum_{j=1}^{2^{n-i-1}} D_{i-1,j}, \quad 1 \leq j \leq 2^{n-1}$$

Let $L_{i,j,k}$ be 1 if task k has got one of its parts in subinterval j of interval i and 0 otherwise. And let $E_{i,k}$ be the total real energy allocated to task k in interval i.

$$E_{i,k} = \sum_{j=1}^{2^{n-k}} L_{i,j,k} P_{i,j} D_{i,j}$$

We can calculate $RE_{i,k}$, the remaining energy for task k started from interval i included as follows:

$$RE_{1,k} = TE_k \qquad \text{and}$$
$$RE_{i,k} = RE_{i-1,k} - E_{i-1,k}, \quad 2 \leq i \leq n, \quad 1 \leq k \leq 2^{n-1}$$

Since we solve the problem interval by interval, we need to ensure that we correctly choose the tasks which will be executed in a particular interval i. Indeed, for the tasks which would not be able to be entirely executed in their intervals, we need to take into account in the previous intervals that we have to affect them a part of the available energy. An example in paragraph 3.2.5 illustrates that.

Let SRE_i be the minimal energy to affect in interval i to other tasks than the ith. And let $C_{i,k}, \quad i+1 \leq k \leq 2^{n-1}$ be 1 if task k is in the list of tasks for which we need to affect a minimum of energy in interval i. The calculation of the $C_{i,k}$ and of SRE_i is done by going back from the last interval to interval i. We first initialize SRE_i and the $C_{i,k}$ to 0. At interval m, we update SRE_i and the $C_{i,k}$ as follows. Let M_m be the processor energy available in interval m. $M_m = P(DL_m - BD_m)$ with P the processor energy by unit of time. If $RE_{i,m} + SRE_i \leq M_m$, then we set SRE_i and the $C_{i,k}$ to 0. Else, we add $(RE_{i,m} - M_m)$ to SRE_i and we set $C_{i,m}$ to 1.

Resolution: We now have determined all needed numerical values. We only have to determine the variables $D_{i,j}$. The linear program to be solved is:

$$Max\ z = \sum_{j=1}^{2^{n-i}} [(2^{n-i} - j + 1) D_{i,j}] \quad [1]$$

$$D_{i,j} \geq 0, \quad 1 \leq j \leq 2^{n-i} \quad [2]$$

$$\sum_{j=1}^{2^{n-i}} D_{i,j} \leq DL_i - BD_i \quad [3]$$

$$E_{i,i} = RE_{i,i} \quad [4]$$

$$E_{i,k} \leq RE_{i,k}, \quad j+1 \leq k \leq 2^{n-1} \quad [5]$$

$$\sum_{k=j+1}^{2^{n-1}} (C_{i,k} E_{i,k}) \geq SRE_i \quad [6]$$

[1] We want to maximize the weighted sum of the durations $D_{i,j}$ of interval i. $D_{i,j}$ is weighted by a greater factor than the one of $D_{i,j+1}$.

[2] All durations are positive.

[3] Task i finishes before its deadline: the sum of the durations of its parts is lower or equal to the duration of the current interval.

[4] The sum of the energies of the parts of task i in interval i equals the remaining energy to affect to task i.

[5] We cannot affect more energy to a task in interval i than what remains for this task from this interval.

[6] We must devote the quantity SRE_i of the supplied energy in the current interval for the tasks k whose variable $C_{i,k}$ is at 1.

Resolution Examples: We consider in this paragraph that the processor is able to execute 1000 operations per second. Task requests are given as $[TE, DL]$. TE is a duration in agent time, given as a number of instructions to execute. DL is the deadline for the task in TSA time given in seconds.

Our first example makes use of three introvert agents and no extrovert agent. Requests are: $[3000ops, 4s]$, $[2700ops, 6s]$ and $[2000ops, 8s]$. We point out that it is not possible to entirely execute task 2 only in the second interval. This one lasts two seconds and has thus an execution capacity of only 2000 operations. It is necessary to carry out at least 700 operations of task 2 in the first interval. Figure 3 illustrates that: we must stop task 3 which turned in parallel to 1 and 2 so that the sum of the parts of task 2 in the first interval can reach the value of 700 operations.

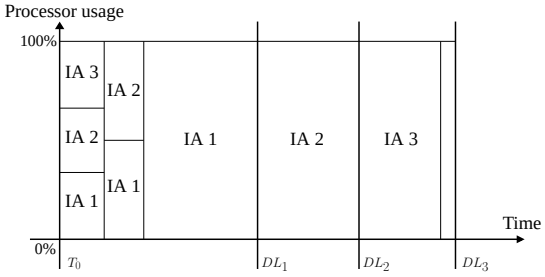

Figure 3: **Taking into account, in the resolution for an interval, of constraints on the remaining intervals.**

For our second example, let us take two extrovert agents and following requests for three introvert agents: $[700ops, 4s]$, $[1000ops, 6s]$ and $[1300ops, 8s]$. Figure 4 shows that we can finish task 1 before the end of its interval while carrying out tasks 2 and 3 in parallel. The new beginning date of the second interval is not DL_1 any more, but the sum of the $D_{1,j}$. It is the same for the beginning date of the third interval since task 2 finishes before the date DL_2.

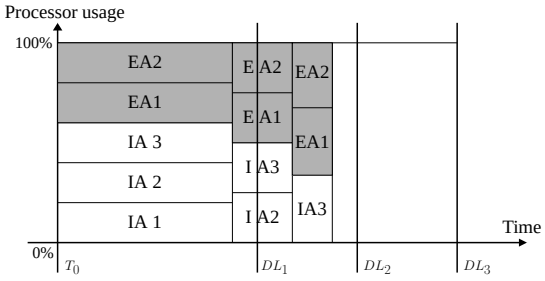

Figure 4: **Changes of intervals beginning dates.**

3.3 Implementation

Each one of our agents executes in its own system process. Extrovert agents callbacks occuring at limits of the time slots which were attributed to them, as well as the sending of the suspension and waking up orders to the introvert agents, were implemented using system signals.

We have studied the limits of our scheduler. The $O(2^n)$ complexity mainly comes from the splitting into subintervals. The computing time quickly becomes prohibitory. It is in practice however possible to reduce this complexity by studying the set of constraints. Indeed, it is often possible to determine, just after the calculation of SRE_i and the $C_{i,k}$ that some tasks will not be able to be executed in the interval i considered. If the sum $SRE_i + RE_{i,i}$ equals the duration of interval i, we do not need to consider other tasks

than task i and tasks k for which $C_{i,k} = 1$. Moreover, we need to limit the number of tasks in an interval so that the solution found does not bring too much splitting. It must remain a difference between the orders of magnitude of the time periods considered by the system scheduler and the TSA scheduler. The latter must not encroach on the work of the first by pausing and resuming too often the agents.

In the scheduling algorithm, we give to all agents a $\frac{1}{nia+NEA}$ share of the processor, given nia the number of introvert agents running at the instant considered. The hypothesis that extrovert agents use the whole computing capacity given to them is too strong when extrovert agents are limited to automata that only process incoming messages. Choosing another hypothesis on the processor usage done by extrovert agents is easily done when attributing the $P_{i,j}$.

4. Related work

Wooldridge and Jennings noticed in [18] that MAS systems are often constructed with simulated parallelism on a single computer and then deployed on the distributed production environment. They argue that several problems appear when going to real distribution. However, they forget to mention that numerous problem actually appear when several autonomous agents have to share the CPU power available on a single computer. As a consequence, time management in MAS with AI tasks has been and still is a challenge. Three major approaches have been proposed to solve this problem.

First, AI tasks may be embedded in a real-time system[19]. This approach guarantees hard deadlines, but restricts the range of AI techniques to the ones that can be simplified to provide bounded response times. As said before, this approach is not the one we want to follow: we want our agents to run on classical systems and we want to use a broad spectrum of AI techniques.

Second, the system may comprise a control level that uses the full range of AI techniques and a real-time subsystem that schedules tasks with deadlines [20]. Although this approach guarantees deadlines for the scheduled bounded-time tasks, it does not guarantee deadlines for system goals.

Third, AI systems may be extended with real-time features. *Anytime algorithms* are an example of solutions proposed in this area. Lalanda and Hayes-Roth propose in [21] another solution to this approach, based on the work of Garvey and Lesser on design-to-time [22]. Their assumption is that goals can be decomposed into a sequence of tasks, and that different methods can be used to perform a task. The actual method to perform a task is choosen at runtime by the meta-control system, according to the different methods properties and the temporal constraints assigned to the agent goals.

Our system falls into the second approach, while allowing the use of the techniques proposed in the third approach. It can be distinguished from others solutions. First, we allow several agents to share a single CPU while other techniques only manage the tasks of a single agent. So, in our context, we have to take care of the autonomy of the agents. We also want that our agents reason about the duration of computational tasks while several other techniques are used to manage tasks that are not computational.

5. Conclusion

We have proposed a definition for time-aware agents and an architecture which allows autonomous time-aware agents to share a single CPU. Extrovert agents are constantly active and turned towards the external world. These agents can decide at any time to launch long processings for which they do not necessarily know the duration. They delegate them to introvert agents as tasks. Introvert agents execute their tasks before turning towards the external world. They also accept to be suspended at any time. For agents sharing a single CPU, the guarantee of temporal constraints satisfiability falls to a specific agent: the Time Services Agent. We define the communication protocol used to interact with it and an algorithm to verify the constraints system satisfiability and to schedule tasks. The TSA only brings a centralization of data and not of decision. Agents decide themselves which amount of CPU they will use. The main role of the TSA is to point out inconsistencies in the scheduling.

Our architecture allows easier programming of time-aware agents. Agent time allows some reasoning to be independent of the execution environment. On the other hand, TSA time allows reasoning to take into account the activity of the other agents. These results will be useful for users who want to control processing durations in AI based MAS applications and especially for applications where the same data provided by sensors is processed by several heuristics.

6. Future work

Time-awareness implies to estimate the duration of processings planned by the agent. If we want to be able to implement in agents a broad spectrum of algorithms, from AI or not, it is necessary to have a generic system that evaluates the processing durations.

The current protocol returns a refusal message when an inconsistency is detected in the set of temporal constraints. An extension being currently developed is that it would rather return a list of changes to be done to the various task requests previously posted. Thus, interactions or negociations could be opened with other agents as in [23] to ask them to give up some of their requirements.

We propose an architecture which allows several agents to share a single CPU. Distributing our system could be done by using a TSA by CPU and introducing interactions between each TSA for load balancing by agent migration. We can reuse the work done by on the DECAF distributed multiagent system[24].

7. References

[1] S. Joseph and T. Kawamura: "*Agent Engineering*", chapter Why Autonomy Makes the Agent, World Scientific Publishing, 2001.

[2] K. P. Sycara, A. Pannu, M. Williamson, D. Zeng and K. Decker: "*Distributed Intelligent Agents*", IEEE Expert, 11(6):36–46, 1996.

[3] Z. Guessoum and M. Dojat: "*A Real-Time Agent Model in an Asynchronous-Object Environment*", In R. van Hoe, editor, Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, 1996.

[4] R. Vincent, B. Horling, V. Lesser and T. Wagner: "*Implementing Soft Real-Time Agent Control*", Proceedings of the 5th ICAA, pages 355–362, June 2001.

[5] A. Garvey and V. Lesser: "*A Survey of Research in Deliberative Real-Time Artificial Intelligence*", Technical report, 1993.

[6] J. Grass: "*Reasoning about computational resource allocation*", Crossroads, 3(1):16–20, 1996.

[7] S. Zilberstein and A. I. Mouaddib: "*Reactive Control of Dynamic Progressive Processing*", In Proceedings of the 16th IJCAI, 1999.

[8] M. Adelantado and S. de Givry: "*Reactive/Anytime Agents - Towards Intelligent Agents with Real-Time Performance*", In IJCAI'95 Workshop on Anytime Algorithms and Deliberation Scheduling, 1995.

[9] E. Horvitz and G. Rutledge: "*Time-dependent utility and action under uncertainty*", In Proceedings of the 7th conference on Uncertainty in artificial intelligence, pages 151–158, 1991.

[10] T. Wagner: "*Toward Quantified, Organizationally Centered, Decision Making and Coordination*", PhD thesis, University of Massachusetts, 2000.

[11] A. Garvey and V. Lesser: "*Design-to-time Scheduling with Uncertainty*", Technical report, 1995.

[12] T. Wagner and V. Lesser: "*Design-to-Criteria Scheduling: Real-Time Agent Control*", Proceedings of AAAI 2000 Spring Symposium on Real-Time Autonomous Systems, 2000.

[13] K. Prouskas: "*Real-Time Extensions of April for Time-Aware Multi-Agent Systems Programming*", PhD thesis, University of London, 2002.

[14] C. L. Liu and J. W. Layland: "*Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*", Journal of the ACM, 20(1):46–61, 1973.

[15] Stankovic, Spuri and Ramamritham: "*Deadline scheduling for real-time systems: EDF and related algorithms*", 1998.

[16] L. C. DiPippo, V. F. Wolfe, L. Nair, E. Hodys and O. Uvarov: "*A Real-Time Multi-Agent System Architecture for E-Commerce Applications*", In ISADS, pages 357–364, 2001.

[17] R. Faure: "*Précis de recherche opérationnelle*", Dunod, 1979.

[18] M. Wooldridge and N. R. Jennings: "*Pitfalls of Agent-Oriented Development*", In K. P. Sycara and M. Wooldridge, editors, Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98), pages 385–391, New York, 9–13, 1998.

[19] S. Forrest, K. P. Gostelow, F. H. Roth and D. M. Smith: "*Concepts, methods, and languages for building timely intelligent systems*", Real-Time Syst., 2(1-2):127–148, 1990.

[20] D. J. Musliner, E. H. Durfee and K. G. Shin: "*CIRCA: A Cooperative Intelligent Real Time Control Architecture*", IEEE Transactions on Systems, Man, and Cybernetics, 23(6):1561–1574, 1993.

[21] P. Lalanda and B. Hayes-Roth: "*Deadline Management in Intelligent Agents*", Technical Report KSL 94-27, Knowledge Systems Laboratory, May 1994.

[22] A. J. Garvey and V. Lesser: "*Design-to-Time Real-Time Scheduling*", IEEE Transactions on Systems, Man and Cybernetics, 23(6):1491–1502, November/December 1993.

[23] A. Garvey, K. Decker and V. Lesser: "*A Negotiation-based Interface Between a Real-time Scheduler and a Decision-Maker*", Technical Report UM-CS-1994-008, 1994.

[24] J. R. Graham: "*Real-time Scheduling in Distributed Multiagent Systems*", PhD thesis, University of Delaware, 2001.