Self-synchronization of Cooperative Agents in a Distributed Environment*

Sergio Ilarri[1]**, Eduardo Mena[1], and Arantza Illarramendi[2]

[1] IIS Department, Univ. of Zaragoza, Maria de Luna 3, 50018 Zaragoza, Spain
{silarri, emena}@unizar.es
[2] LSI Department, Univ. of the Basque Country, Apdo. 649, 20080 Donostia, Spain
jipileca@si.ehu.es

Keywords: coordinated behaviour, distributed cooperative agents

Abstract. Multiagent systems have been widely used in applications that are inherently distributed such as information retrieval, network diagnosis, and distributed vehicle monitoring. The use of multiple cooperative agents offers some important advantages (such as parallelism, robustness and scalability) for those applications. However, the synchronization of those multiple agents is an important challenge, specially when they are executing on different computers.
In this paper we present a formal description of a mechanism based on deadline commitments to synchronize agents in a distributed application. In our proposal, agents synchronize themselves reacting to changes in the environment. The proposal considers a loose coupling among agents with the goal of maintaining the autonomy and independency of the involved agents.

1 Introduction

Multiagent systems have been widely used in applications that are inherently distributed such as information retrieval [2], network diagnosis [1], and distributed vehicle monitoring [10, 3, 4, 9]. The use of multiple cooperative agents offers some important advantages (such as parallelism, robustness and scalability) for those applications. However, the synchronization of those multiple agents is an important challenge, specially when they are executing on different computers.

The synchronization techniques we describe in this paper are suitable to multi-agent applications that satisfy the following conditions:

1. *The main task is decomposed in several subtasks.* That is, an agent is in charge of performing such a main task. To achieve its goal, this agent creates as many agents as it needs, possibly in different computers. Each agent could create its own network of agents to achieve its own goal, and so on. Thus, agents are organized in a multi-layered, *hierarchical architecture*.

* This work was supported by the CICYT project TIC2001-0660 and the DGA project P084/2001.
** Work supported by the grant B132/2002 of the Aragón Government and the European Social Fund.

2. *Each task returns a certain result.* This could be a simple flag indicating success or failure, or information obtained from the environment. Agents at the bottom layer gets information from the environment to perform its tasks. Agents at an intermediate layer use the results received from its created agents as an input for their tasks. The process performs recursively until the root agent gets the final result of the main task (that will be shown to the user or stored for later use).
3. *The task must be performed periodically*, with a certain *execution frequency*, as the result of the task is based on data obtained from a dynamic environment, that changes along time.

In this paper we present a formal description of a mechanism based on deadline commitments to synchronize agents in a distributed application. In our proposal, agents synchronize themselves reacting to changes in the environment. The proposal considers a loose coupling among agents with the goal of maintaining the autonomy and independency of the involved agents. We also explain how to manage problems due to clock desynchronization and network delays, that could lead some agents to lose their deadline.

In Section 2 and Section 3 we explain our synchronization technique based on the use of deadlines. In Section 4 we describe how agents decide when they have to start their task. In Section 5 we describe some related work. Finally, some conclusions and future work are shown in Section 6.

2 Dealing with Deadlines

As explained in [6], it is necessary to coordinate the agents in a multi-agent application in order to avoid the generation of suboptimal solutions, the waste of computational/communication resources (due to the generation of redundant, unneeded or poorly timed results) and, in the worst case, the failure in getting a result or getting a wrong result.

We explain in this section the basics of the technique we propose to coordinate agents in a distributed application in an efficient way. The idea is that each agent must provide its creator agent with a result obtained as late as possible (in order to maximize the novelty of the data it considers to get that result) but within the required time (to return a new result with the required frequency).

To assure timely communications, *every agent in the network will be associated to a certain deadline that indicates the time at which a new result must be available to its creator agent.* The problem of keeping the final result up-to-date can be solved by the network of agents by communicating new results to the root agent *right before* a new result for the main task is required.

In Figure 1 we show how agents interact across layers. We focus on an intermediate agent at layer i, $Agent_i$, created by a certain $Agent_{i-1}$ at layer $i-1$. We explain the main steps to consider in order to manage deadlines: (1) $Agent_{i-1}$ communicates to $Agent_i$ the deadline for layer i; (2) $Agent_i$ receives its deadline and calculates the deadline of agents at layer $i+1$, by considering its own deadline and its estimated task delay; (3) $Agent_i$ communicates to agents at layer

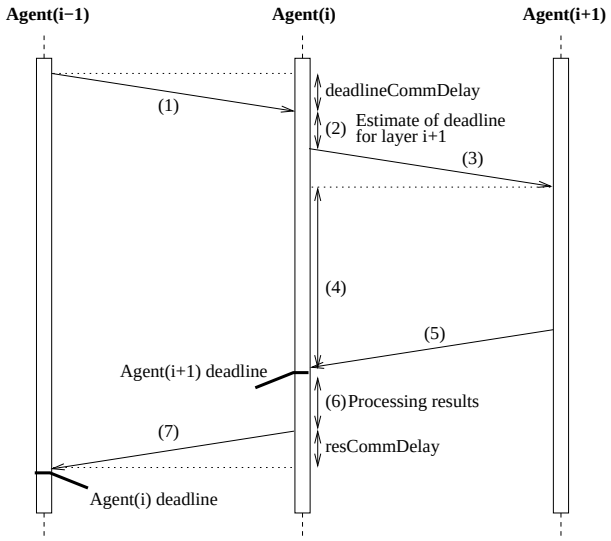

Fig. 1. Interactions among agents

$i+1$ their deadline; (4) $Agent_i$ waits until the deadline of layer $i+1$ comes (to meet its own deadline, $Agent_i$ has to begin processing the received results right after they are obtained from layer $i+1$); (5) each $Agent_{i+1}$ sends its result to $Agent_i$; (6) $Agent_i$ processes the received results, and (7) $Agent_i$ sends its result to $Agent_{i-1}$. Agents at the lowest layer do not receive results from any agent, but they obtain data directly from the environment.

As mentioned before, every agent adopts a deadline-based approach to perform its task and communicate its result, instead of a waiting-based approach in which every agent would wait until it has received all the necessary results before processing them and communicating its own result. We consider that our solution has two major advantages:

- *A final result can be obtained despite partial failures in the system.* An agent can perform its task despite it has not received the result from some of its underlying agents, as using the last received result from such an agent could be a suitable approach in some applications.
- *A consistent snapshot of the environment is considered* by associating each agent with a time instant at which its task must finish. In this way, we synchronize the moment at which results from a certain layer are obtained. For example, let us consider a multiagent application whose goal is to get the average temperature of a campus by averaging temperatures obtained by agents located in different buildings. If a waiting-based approach is used, we could, for example, get a temperature of 20 degrees that never happened before, because we could average temperatures measured at different time instants. In general, using a waiting-based approach it would be difficult that

all the involved agents get their data within the same small time window (to get a consistent result).

To calculate deadlines, every agent must be aware of the time it needs to perform its tasks. The different types of delays in which an agent incurs are given by Definition 1 and Definition 2.

Definition 1. *We call task delay of an agent x at layer i created by a certain agent y at layer $i-1$, to the time it spends performing the task of processing the results received from its underlying agents and communicating its own result to its creator agent:*

$$taskDelay_{x@i} = processDelay_{x@i} + resCommDelay_{y@i-1}^{x@i}$$

Definition 2. *We call deadline change delay of an agent x at layer i that creates several agents z at layer $i+1$, to the time it spends performing the tasks it must do when there is a change in its delays; these tasks are the estimation of a new deadline for agents z and the communication of this deadline to agents z:*

$$deadlineChangeDelay_{x@i} = estimationDelay_{x@i} + deadlineCommDelay_{z@i+1}^{x@i}$$

Agents at the same layer can have different delays because they can execute on different computers. Besides, the delays can change along time: communication tasks can become more or less time expensive (as the size of the results to transmit can vary, the network load can change, etc.) and the time needed to perform the processing of results can also change (it could depend on the size of the results to process and even on its values). Thus, agents will keep track of the delays in which they incur in order to be able to estimate future delays. Based on its estimated delays, an agent will obtain the deadline of its underlying agents. In the following, we will use the $\widehat{\ }$ symbol (e.g., $\widehat{taskDelay}_{x@i}$) to denote estimations of values.

Definition 3. *We call absolute deadline of an agent y at layer $i+1$ to the time instant at which it should have made its result available to its creator agent x at layer i. It is given by:*

$$absDeadline_{y@i+1} = absDeadline_{x@i} - \widehat{taskDelay}_{x@i}$$

As agents in a multiagent system could execute on different computers[1], dealing with absolute deadlines would require to keep the internal clocks of these computers synchronized. We consider that this is a very strong requirement, difficult to achieve in heterogeneous environments, specially when some of the computers to synchronize are wireless-connected. In the following, we will use a superscript like $x@i$ to indicate that a certain time instant is measured with regard to the computer clock where a certain agent x (at layer i) resides.

[1] Even they could move among computers, if they are mobile agents.

Definition 4. *The clock shift or desynchronization between the clocks of the computers where agents x (at layer i) and agent y (at layer j) reside, at a certain time instant t, is given by:*

$$shiftClock_{y@j}^{x@i} = t^{x@i} - t^{y@j}$$

Just a slight clock shift will make some agents to miss their deadlines, as they try to do their tasks as late as possible to maximize the novelty of the data they use in their tasks. Consequently, deadlines communicated among layers must be relative (e.g., "I want your result in ten seconds") rather than absolute (e.g., "I want your result ready at 13:05:20"). An agent calculates the relative deadline of its underlying agents as indicated in Definition 5

Definition 5. *We call relative deadline of a certain agent y at layer $i+1$ to the time interval that will remain before the next absolute deadline of y comes once it receives this (relative) deadline from its creator agent x at layer i. It is given by:*

$$relDeadline_{y@i+1} = absDeadline_{y@i+1}^{x@i} - tDeadlineEst_{x@i}^{x@i} - \widehat{deadlineChangeDelay}_{x@i}$$

where $tDeadlineEst_{x@i}$ denotes the time instant at which agent x (at layer i) starts estimating the relative deadline of its underlying agents.

3 Obtaining Absolute Deadlines from Relative Deadlines

As we have seen, deadlines are communicated in a relative manner. An agent will transform the relative deadline it receives into an absolute deadline in order to know the time instant at which it must finish its task. The transformation is performed as indicated in Definition 6.

Definition 6. *We call absolute deadline estimation to the way in which an agent y (at layer $i+1$) estimates its own absolute deadline from the relative deadline that it receives:*

$$\widehat{absDeadline}_{y@i+1}^{y@i+1} = tDeadlineRec_{y@i+1}^{y@i+1} + relDeadline_{y@i+1}$$

where $tDeadlineRec_{y@i+1}$ denotes the time instant at which agent y (at layer $i+1$) receives its relative deadline from its creator agent.

In Corollary 1 we prove that the formula given in Definition 6 is a good estimation of the absolute deadline of an agent.

Theorem 1. *The error in the absolute deadline considered by a certain agent equals the error committed by its creator agent when it estimated its deadline change delay.*

Proof. We will consider an agent y (at layer $i+1$) created by a certain agent x (at layer i). According to Definition 5, the relative deadline of agent y is calculated by agent x at time instant $tDeadlineEst$, as follows:

$$relDeadline_{y@i+1} = absDeadline^{x@i}_{y@i+1} - tDeadlineEst^{x@i}_{x@i} - \widehat{deadlineChangeDelay}_{x@i}$$

According to Definition 4, we have:

$$tDeadlineRec^{y@i+1}_{y@i+1} = tDeadlineRec^{x@i}_{y@i+1} + shiftClock^{y@i+1}_{x@i}$$

The previous equality can be further decomposed by considering that agent y receives its new relative deadline after x has estimated such deadline and communicated it to y, that is:

$$tDeadlineRec^{x@i}_{y@i+1} = tDeadlineEst^{x@i}_{x@i} + deadlineChangeDelay_{x@i}$$

By Definition 4, we have:

$$absDeadline^{y@i+1}_{y@i+1} = absDeadline^{x@i}_{y@i+1} + shiftClock^{y@i+1}_{x@i}$$

By using the previous equalities in Definition 6, we obtain:

$$\widehat{absDeadline}^{y@i+1}_{y@i+1} = absDeadline^{y@i+1}_{y@i+1} + (deadlineChangeDelay_{x@i} - \widehat{deadlineChangeDelay}_{x@i})$$

□

Corollary 1. *The absolute deadline estimation given by Definition 6 is a precise estimation of the absolute deadline of an agent (given by Definition 3), assuming that a precise estimation of the deadline change delay is provided.*

Proof. By hypothesis, it holds that

$$deadlineChangeDelay_{x@i} = \widehat{deadlineChangeDelay}_{x@i}$$

Thus, from Theorem 1:

$$\widehat{absDeadline}^{y@i+1}_{y@i+1} = absDeadline^{y@i+1}_{y@i+1}$$

□

4 Waiting the Deadline

We explain in this section how an agent determines how much time it has to wait before performing its tasks.

Definition 7. *We define the spare time of an agent y (at layer $i+1$) at time instant t as the time interval that the agent will wait before performing its tasks. It is given by:*

$$spareTime^{t}_{y@i+1} = absDeadline^{y@i+1}_{y@i+1} - t^{y@i+1}_{y@i+1} - taskDelay_{y@i+1}$$

Definition 8. *We call task finalization time to the time instant at which an agent finishes its tasks (in a certain refreshment period), according to the computer clock where such an agent resides. If an agent y (at layer $i+1$) waits T time units after time instant $t^{y@i+1}_{y@i+1}$ before performing its tasks, it holds:*

$$tEndTask^{y@i+1}_{y@i+1} = t^{y@i+1}_{y@i+1} + T + taskDelay_{y@i+1}$$

Proposition 1. *At time instant t, the maximum time interval that an agent y (at layer $i+1$) can wait at time instant t before performing its task is given by $spareTime^{t}_{y@i+1}$ (see Definition 7).*

Proof. Let us assume that at instant $t^{y@i+1}_{y@i+1}$ the agent y waits T time units before performing its task. From Definition 7, we have:

$$absDeadline^{y@i+1}_{y@i+1} = t^{y@i+1}_{y@i+1} + spareTime^{t}_{y@i+1} + taskDelay_{y@i+1}$$

According to Definition 8, it holds:

$$tEndTask^{y@i+1}_{y@i+1} = t^{y@i+1}_{y@i+1} + T + taskDelay_{y@i+1}$$

To keep the agent from missing its deadline, it must hold:

$$tEndTask^{y@i+1}_{y@i+1} <= absDeadline^{y@i+1}_{y@i+1} \Longleftrightarrow$$
$$T \leq spareTime^{t}_{y@i+1}$$

□

Corollary 2. *At time instant t, an agent y (at layer $i+1$) must wait $spareTime^{t}_{y@i+1}$ (as defined in Definition 7) before performing its tasks.*

Proof. In order to consider data as new as possible, the agent must perform its task as late as possible. According to Proposition 1, the maximum amount of time that agent y can wait at time instant t before performing its tasks is $spareTime^{t}_{y@i+1}$. Consequently, that is exactly the time interval that the agent should wait. □

Definition 9. *We define the estimated spare time of an agent y (at layer $i+1$) at time instant t as follows:*

$$\widehat{spareTime}^{t}_{y@i+1} = absDeadline^{y@i+1}_{y@i+1} - t^{y@i+1}_{y@i+1} - \widehat{taskDelay}_{y@i+1}$$

Theorem 2. *The value of* $\widehat{spareTime}^t_{y@i+1}$ *(given by Definition 9) is a good approximation to* $spareTime^t_{y@i+1}$ *(given by Definition 7), assuming that a good estimation of the task delay is provided.*

Proof. A good estimation of delays implies that:

$$|taskDelay_{y@i+1} - \widehat{taskDelay}_{y@i+1}| < \epsilon$$

for a small value of ϵ.

From Definition 7 and Definition 9, we have:

$$|\widehat{spareTime}^t_{y@i+1} - spareTime^t_{y@i+1}| = |taskDelay_{y@i+1} - \widehat{taskDelay}_{y@i+1}|$$

Therefore:

$$|\widehat{spareTime}^t_{y@i+1} - spareTime^t_{y@i+1}| < \epsilon$$

□

Proposition 2. *The estimation of delays must be pessimistic for the calculation of the estimated spare time of an agent, as otherwise an agent could miss its deadline.*

Proof. Let us consider an agent y at layer $i+1$. According to Definition 8, and assuming that the agent waits its estimated spare time (as indicated by Corollary 2), it holds:

$$tEndTask^{y@i+1}_{y@i+1} = t^{y@i+1}_{y@i+1} + \widehat{spareTime}^t_{y@i+1} + taskDelay_{y@i+1}$$

From Definition 9, we have:

$$absDeadline^{y@i+1}_{y@i+1} = t^{y@i+1}_{y@i+1} + \widehat{spareTime}^t_{y@i+1} + \widehat{taskDelay}_{y@i+1}$$

The agent will not miss its deadline iff:

$$tEndTask^{y@i+1}_{y@i+1} \leq absDeadline^{y@i+1}_{y@i+1} \iff taskDelay_{y@i+1} \leq \widehat{taskDelay}_{y@i+1}$$

□

Thus, an agent waits its spare time (estimated by the agent as shown in Definition 9) before performing its task. To avoid that a slight increase in the estimated delays makes an agent to miss its deadline, the agent could wait something less than what its estimated spare time indicates.

In Theorem 3 we prove that an agent should start processing the received results on the arrival of these results from its underlying agents. This is due to the fact that every agent performs its task as late as possible.

Theorem 3. *The time instant at which an agent x (at layer i) starts its task equals the deadline of its underlying agents y (at layer $i+1$).*

Proof. Agent x starts its task after waiting its estimated spare time once it knows which its next deadline is, that is:

$$tStartTask_{x@i}^{x@i} = tDeadlineRec_{x@i}^{x@i} + \widehat{spareTime}_{x@i}$$

where, for the sake of readability, we have omitted the superscript for the spare time.

Using the formula given in Definition 9 with time instant $tDeadlineRec_{x@i}^{x@i}$, we get:

$$tStartTask_{x@i}^{x@i} = absDeadline_{x@i}^{x@i} - \widehat{taskDelay}_{x@i} = absDeadline_{y@i+1}^{x@i}$$

□

5 Related Work

In [10, 9] a network of distributed sensors is used to track the location of moving objects by triangulating measures from three or more sensors. It is necessary that the measures obtained to triangulate a moving object's location refer to (approximately) the same (as recent as possible) time instant. They propose, as we do, an approach based on deadline commitments to reevaluate periodically the location. Although they mention the problem that would suppose to have desynchronized computers, no solution is proposed. Nevertheless, they tackle the problem in a more generic way, as negotiation among agents is considered instead of having agents with fixed roles.

In [7, 8], mobile agents are proposed as a mechanism to perform data integration in a distributed sensor network in an efficient way. However, as far as we know, they do not propose a mechanism to assure that the measures that they are integrating refer to (approximately) the same time instant.

6 Conclusions and Future Work

In this paper, we have provided a formal description of a mechanism based on deadline commitments to synchronize agents in a distributed application. The main features of our approach are the following:

- Agents adapt themselves to changes in the environment (changes in network or processing delays).
- A loose coupling among agents is considered, allowing them to work in an asynchronous and independent way.
- The system could work even when some agents in the system cannot do their tasks in time.

– Every agent performs its task as late as possible in order to consider data as recent as possible.

As future work, we plan to analyze how to deal with situations where the required execution frequency cannot be achieved. Some preliminary work for a specific application (a location-dependent query processor) is shown in [5]. We plan to study how to make good estimations of parameters such as the network delays by using past measured values.

References

1. A. Bieszczad, T. White, and B. Pagurek. Mobile agents for network management. *IEEE Communications Surveys*, 1998.
2. Brian Brewington, Robert Gray, Katsuhiro Moizumi, David Kotz, George Cybenko, and Daniela Rus. Mobile agents in distributed information retrieval. In Matthias Klusch, editor, *Intelligent Information Agents*. Springer-Verlag: Heidelberg, Germany, 1999.
3. S. Ilarri, E. Mena, and A. Illarramendi. A system based on mobile agents for tracking objects in a location-dependent query processing environment. In *Twelfth International Workshop on Database and Expert Systems Applications (DEXA'2001), Fourth International Workshop Mobility in Databases and Distributed Systems (MDSS'2001), Munich (Germany)*, pages 577–581. IEEE Computer Society, ISBN 0-7695-1230-5, September 2001.
4. S. Ilarri, E. Mena, and A. Illarramendi. Monitoring continuous location queries using mobile agents. In *Sixth East-European Conference on Advances in Databases and Information Systems (ADBIS'2002), Bratislava (Slovakia)*. Springer Verlag LNCS, September 2002.
5. S. Ilarri, E. Mena, and A. Illarramendi. Dealing with continuous location-dependent queries: Just-in-time data refreshment. In *First IEEE Annual Conference on Pervasive Computing and Communications (PerCom), Dallas Fort-Worth (Texas)*. IEEE Computer Society, to appear, March 2003.
6. Victor R. Lesser. Cooperative multiagent systems: A personal view of the state of the art. *Knowledge and Data Engineering*, 11(1):133–142, 1999.
7. Hairong Qi, S. Sitharama Iyengar, and Krishnendu Chakrabarty. Distributed multi-resolution data integration using mobile agents. *IEEE Transactions on Systems, Man and Cybernetics (Part C):Applications and Reviews*, 31(3):383–391, August 2001.
8. Hairong Qi, S. Sitharama Iyengar, and Krishnendu Chakrabarty. Multisensor data fusion in distributed sensor networks using mobile agents. In *Proc. Intl. Conf. Information Fusion*, pages 11–16, August 2001.
9. Régis Vincent, Bryan Horling, Victor Lesser, and Thomas Wagner. Implementing soft real-time agent control. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 355–362, Montreal, Canada, 2001. ACM Press.
10. Regis Vincent, Bryan Horling, Roger Mailler, Jiaying Shen, Kyle Rawlins, and Victor Lesser. SPT: Distributed sensor network for real time tracking. In *Fifth International Conference on Autonomous Agents, Montreal (Canada)*, May 2001.