CAAM : A Model of Resource-Bounded Reasoning Agent for Terrain Analysis in C3I Systems

Thierry SALVANT (1&2), Stéphan BRUNESSAUX (1) and Alain GRUMBACH (2)

(1) MATRA Sysèmes & Information
6, rue Dewoitine BP14 - 78142 Vélizy Cedex - FRANCE
E-Mail: tiri@matra-ms2i.fr, stephan@mcs-vdr.fr
Tel: (33)1.34.63.76.03 - Fax: (33)1.34.63.76.23

(2) Ecole Nationale Supérieure des Télécommunications
46, rue Barrault 75634 Paris Cedex 13 - FRANCE

Abstract

The multi-agent approach has been successfully used for solving more and more complex problems. Recent work carried out in that field has shown the benefits it provides for building decision support applications in a complex real-time environment.
We will herein present some work in the field of Command, Control, Communications and Information (C3I) systems, that has been carried out in the context of the $30 million European EUCLID RTP6.1 project.
We describe agent-based techniques that have been developed for providing time-critical decision support in a realistic operational context. This has been done by integrating and adapting anytime reasoning techniques to a multi-agent framework. The main objectives of the Cooperative Anytime Agent Model (CAAM) we have proposed were to allow agents to make use of time and computational resources in an optimal manner, and to allow the overall system to take time constraints into account by adapting the accuracy of its results to the time available.

Introduction

The multi-agent approach has been successfully used for solving more and more complex problems. Recent work carried out in that field has shown the benefits it provides for building decision support applications in a complex real-time environment. We will present in this paper results of experiments we have been carried out to assess a model of resource-bounded reasoning agent called CAAM (Cooperative Anytime Agent Model), that has been implemented for performing time-critical terrain analysis in the context of route planning. Such a terrain analysis includes line-of-sight visibility calculations, calculation of the shortest viable route between two points and extrapolation of possible future movements considering kinematic properties [Salvant, 1997]. All these calculations are applied on a 60km x 60km coastal area of Norway, with a military-required optimal precision of 100m x 100m for terrain elevation data, and may have to be performed taking account of time constraints.

This work has been applied in the field of Command, Control, Communications and Intelligence systems (C3I) and has been carried out in the context of the $30 million-worth EUCLID RTP6.1 project. EUCLID, which stands for EUropean Coordination for the Long term In Defence, is a European research programme started in 1990. The 5-year long RTP6.1 project has been carried out by the GRACE consortium grouping together 18 companies within 7 nations. It aims at defining future intelligent C3I workstations.

Motivation for using resource-bounded reasoning

The agents we will describe in this paper are part of the GRACE multi-agent based decision support system which run on 4 Ethernet-networked Sun Sparc workstations. Such a decision support system includes other agents specialized in providing support for: compiling the current tactical picture, analyzing the tactical threat, generating orders and so on. All these last activities may also have to be performed under time pressure, but their nature and the techniques available make far less difficult their implementation considering deadlines, if we compare to the implementation of terrain analysis calculations that we will detail below.

The point here is that, as responsible for the implementation of agents for terrain analysis, we are not able to consider as many computation resources as we would like to. This is due to economical reasons, combined with the fact that such functionalities could have to be added later to existing systems, with non extensible computation resources. Given the architecture of the GRACE system and the constraints related to the use of the underlying hardware resources, the work we will describe has been done considering agents with one non-shared computational resource, and capable of performing concurrently terrain analysis calculations.

Our terrain analysis calculations have two main characteristics that make their use very limited in time-constrained situations.

Firstly, there is a very high variance between the computation time required for each type of calculation. In our context, a line-of-sight visibility calculation can use from 5 to 60 CPU seconds and from 40 seconds to 8 minutes considering a maximum range of respectively 10Km and 25Km (these measures have been made on a 75MHz SS20 processor).

For a given range, the CPU time is directly related to the shape of the terrain where the calculation is made. The calculation will take far less time in a very narrow fjord than from a location with few obstacles around.

These measures show the limitation of such calculations for its use in time-critical situations. Indeed, in real situations, the user may ask many requests to be performed concurrently: it is for example realistic in our context that the user asks for more than 5 terrain calculations and waits for all the results to be able to take a decision. This can at least imply non acceptable delays for delivery of the results, if we consider the limited number of CPU available. Indeed, given the rate at which a tactical situation evolves, a result is rarely useful for the user if delivered after 2 or 3 minutes, and even less if the situation is particularly critical.

Secondly, the limitation is related to predictions of CPU consumption. The limitations are at the level of each calculation, and at the level of the number of requests that may need to be performed concurrently.

For the former, the key element that makes a good and quick prediction difficult to make is the terrain. For line-of-sight visibility calculations, at a given range, the CPU consumption is directly function of the resulting area that can be seen. A precise prediction can therefore be potentially as difficult to obtain and as time consuming as the calculation itself. The same happens considering path calculation or extrapolation of possible movements. Intuitively, the solution is to have an "idea" of the shape - i.e. an approximation - of the optimal result that needs to be precise enough to have an acceptable prediction, but not too precise to get it in a sensible delay.

For the later, the key element is the external situation that the user can of course not entirely control. The number or requests he may have to start can therefore not be predictable, because situation dependent. It can be very small for a long period, making the delays for getting the optimal solutions (at the 100mx100m precision) almost always acceptable. But this number can suddenly very high, making the optimal solutions not feasible in time.

These two limitations, related to high variance between the computation time required and their estimation, make not feasible and not desirable to compute the optimal answer. It is not feasible considering deadlines, and not desirable considering the predictions of required CPU.

Approach to the problem

The approach has been presented in [Salvant, 1997]. Basically, it consists in implementing each of the terrain based calculation as an anytime algorithm.

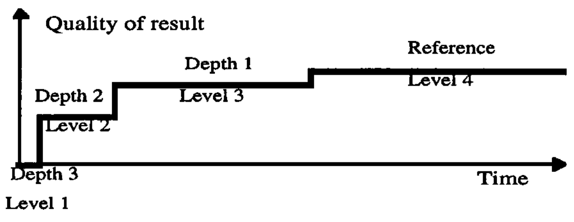

Figure 1: Chosen 4-level profile

This has been done by introducing 3 more terrain representations obtained after approximation of the optimal one, and executing the same algorithm successively on each of these representation (figure 1). Each approximated representation also covers the same 60Km x 60Km square area. Their precision are respectively 200x200m (depth 1 approximation - 300x300 cells), 400x400m (depth 2) and 800x800m for the most approximated one (depth 3).

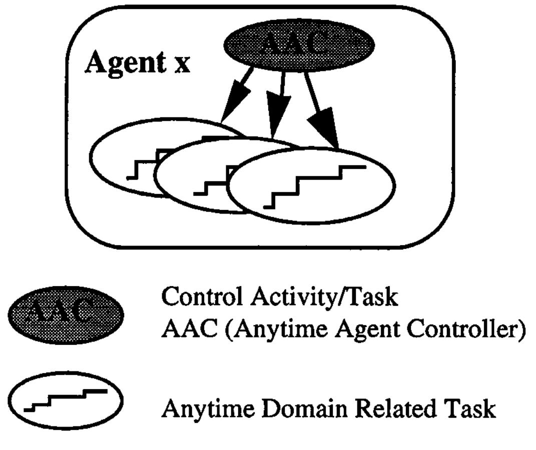

Figure 2 : Anytime agent tasks

Each CAAM agent has one control activity/task for controlling the anytime tasks with such a profile (figure 2). This task is performed by a so called Anytime Agent Controller (AAC). It is responsible for terminating the anytime tasks on the deadline, for providing the available results if required and scheduling them [Salvant, 1997] - a modified version of the Earliest Deadline First algorithm [Chetto, 1989] as well a modified version of the Deliberative Scheduling algorithm [Boddy and Dean, 1994] have been implemented.

A mode can be set for the control task allowing it to deliver multiple intermediate results before the deadline. This might be required considering some cases where a progressive/anytime task requires, to perform a calculation at a given approximation level, a result from another task at the same level. This might be also required by the user who might need intermediate results.

For example, a user may need to compute a path to go from one location to another, avoiding as much as possible entering in the areas that can be seen from the destination. Here, the task for path calculation needs first to know what are these visible areas, that have to be computed by another task performing line-of-sight visibility calculations. In order to perform each calculation of the various approximation levels, the first task needs the corresponding results of the line-of-sight visibility calculation made at the destination location (figures 3a, 3b, 3c). Each of these figures shows the results calculated at the 3 first levels of approximation. On that example, the criteria for the path calculation is to enter as late as possible into the visible area. The total CPU times to get these results shown in the figures are respectively 2 sec., 15 sec. and 1.5 min. At the level of reference (no approximation), 11 min. are required.

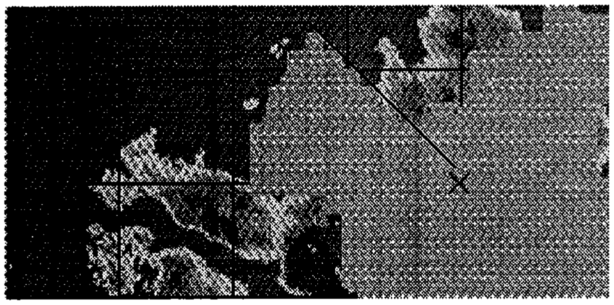
Figure 3a : Level 1

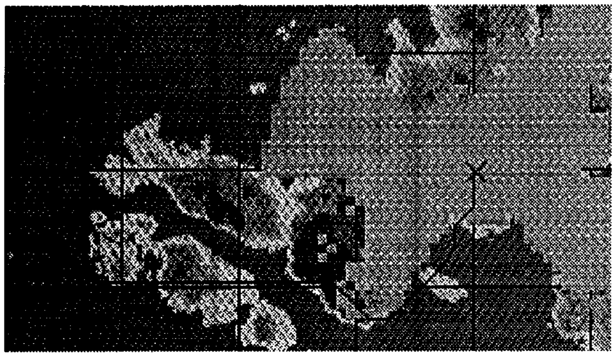
Figure 3b : Level 2

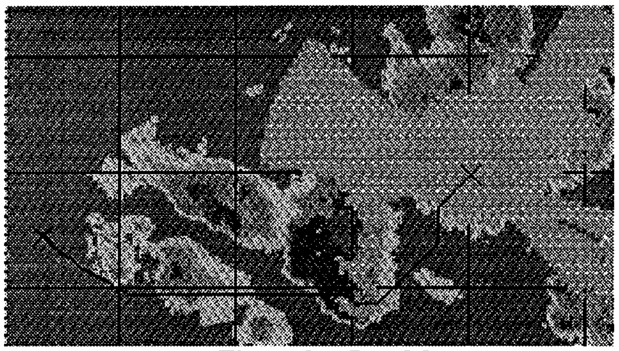
Figure 3c : Level 3

This scenario can appear when considering one agent capable of both calculations, or two different agents, each of them capable of performing one of the two. We will discuss this point later.

This implies a strong requirement on the way the solutions evolves. Indeed, the approximations always have to be conservative compared to reality, ensuring that the cruder the approximation is, the more conservative the result is. By conservative, we mean that, for a specific problem, if an acceptable decision can be made by a task part of an agent (human or software), based on a result calculated by another agent at a given level of approximation, this decision will always be acceptable even if not optimal. It will still not invalidated by subsequent, more accurate results, although these may identify better solutions.

For a line-of-sight visibility calculation, considering a hostile - respectively a friend position - the deeper the approximation is, the larger -respectively the smaller - the resulting visible area has to be. Moreover, for every level, the corresponding area has to be included into the area at the next - respectively the previous - level. For a path, if the criteria is the length, the paths given successively at each level have to be shorter - respectively longer - considering a friendly - respectively hostile - unit.

For the specific example of figures 3x, the criteria for the path is to enter as late as possible into the visible area. The decision to go to the destination location will be taken depending on the distance between this location and the location where the mobile enters into the area. Either this distance is bellow a given threshold, and it is safe to go, or it is above the threshold and it may be dangerous to try to reach the destination.

For the line-of-sight visibility calculation, ensuring that the deeper the approximation is, the larger the resulting visible area is, and ensuring that each area at one level is included in the area calculated at the following level, will guarantee that this distance will decrease with the time allocated to the calculation.

For that purpose, two types of approximation have been done at each level: one considering the terrain always higher from one approximation to a deeper one, and the other lower. The former is called approximation max, the later approximation min.

Limitations and benefits of the approach

We present some measures that have been done to assess the approach considering the evolution of the successive results, and the benefits it provides in order to make predictions of CPU consumption during run-time.

Line-of -sight visibility calculation

By nature of the representation, it is not possible to guarantee at 100% that the resulting areas will evolve as required.

Table 1 gives the measures to assess this error on an approximation max. For this type of approximation, the error corresponds to the percentage of area that is not included in the one at the next level (the results are comparable for the approximation min). The values show that these errors are acceptable.

Table 1 Approximation max (120 measures)

Levels	Average error (%)	Average difference to the average error	Min. Error (%)	Max. Error (%)
0 -> 1	7.66	2.55	0.00	20.00
1 -> 2	4.87	1.36	0.93	13.23
2 -> 3	3.03	1.10	0.00	10.64

Table 2 gives average values of the coefficients to get the CPU consumption at one level knowing the one at the previous level.

Table 2 Approximation max (120 measures)

Levels	Average coefficient	Average difference	Average difference(%)
1 -> 2	7.88	0.419	5.31 %
2 -> 3	8.02	0.197	2.45 %
3 -> 4	8.05	0.114	1.42 %

These measures show some very interesting benefits of this simple approach considering predictions of CPU consumption. Indeed, for a range between 10Km and 25Km, the CPU needed measured in seconds at the first, second and third levels are respectively in [0.05 : 0.9], [0.4 : 6.3] and [3.5 : 50]. This means that it is possible to make "very" quickly a prediction after the first level of approximation with an average error of 5.3%, then refine progressively at run-time this prediction as other levels are performed.

Path calculation

The algorithm we have implemented is a modified version of the Dijkstra algorithm to find an optimal path in a graph. Several criteria can be used to find a path, such as length, average altitude along the path, and minimum penetration in a given area. For the first criteria, the terrain approximations guarantee that the length will either always get shorter (if we consider the friend forces) or longer (if we consider the hostile forces) as long as the algorithm finds the optimal solution for one level. For the second criteria, they will guarantee that the average altitude will be lower (if we consider the friend forces) or higher (if we consider the hostile forces). For the last criteria, it will depend on the area to be avoided, that results from a line-of-sight visibility calculation. Further studies will consider the evolution of results combining altogether these criteria.

The measures on the coefficients to get the CPU consumption at one level knowing the consumption at the previous level give similar results, even if they are less precise. For each level, the value of the coefficient is around 5 with an average error of 20%. We will show measures that show that, even with this error, the predictions are still better than considering only the distance between the starting location and the destination.

Mechanisms to control the computational resources in CAAM

As said in the section describing the approach to the problem, each agent is able to perform two types of task scheduling with the profile described. For each, the scheduling is performed each time a new task is to be run, or a task has finished to perform a level of approximation or a task deadline has been modified.

The modified Earliest Deadline First algorithm is as follows. As soon as there are tasks that have not finished reasoning at the first level of approximation, the control task suspends all the other tasks that have finished it, in order to run concurrently (sharing the agent CPU) only the tasks at the first level. When all the tasks have performed their first level of approximation, then the control task determines all the task levels at the second level that have not been accomplished, then put them in the schedule following the "earliest deadline first" rule. A level is however not put in the schedule if it can not be performed before the deadline of its task - the estimation of CPU requirement is then available for each task level and taken into account. This step is repeated considering, successively at the third and fourth level, the task levels that has not been accomplished. After the schedule is performed, only the first task level of the schedule is executed on the agent CPU.

The adaptation of the Deliberative Scheduling is as follows: it is the same as the previous one considering the task levels of the first level. For the other levels, they are scheduled as described in [Boddy and Dean, 1994], considering all the slopes equal to zero.

We have measured the CPU used by the control task to perform these scheduling algorithms. For both, it is small compared to the CPU used by the progressive/anytime tasks, and has a negligible influence on the predictions, compared to those dues to the errors we have measured (table 2). For both scheduling algorithms, the performance is related to these errors.

We propose an approach to address the problem we have already mentioned. This problem considers a task performing a path calculation that needs as input at a level an area to avoid which is calculated by a task performing line-of-sight visibility calculation on the same level (figures 3x).

We have considered two cases: one agent performing all the terrain based calculations, and more than one agent, each one being able to perform a sub-set of these calculations, but being the only one with these capabilities.

In the first case, the scheduling is performed simply by taking these dependencies into account. In the second case, the idea is to make the server agent sending some predictions about real-time delay at which it is able to deliver intermediate results, given its current scheduling. These predictions are then used by the client waiting for the server results at each level, in order to optimize its own scheduling. Taking the example we have described above, the agent with line-of-sight visibility calculation capabilities will send, at run-time and after each scheduling, its expectations about the delivery of the results at each level, taking the new scheduling into account (figure 4). These predictions will then be used by the agent with path calculation capabilities that has made the request in order to compute a path avoiding as much as possible the visible area.

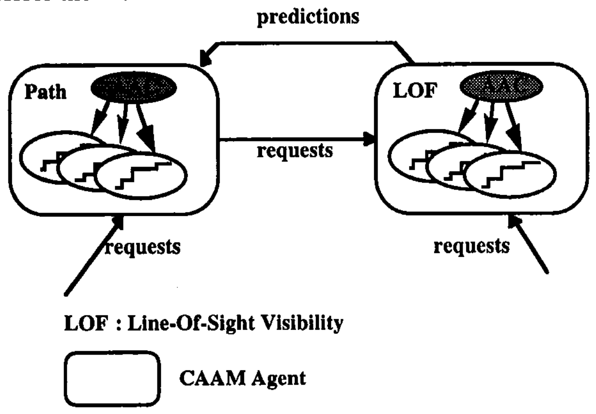

Figure 4 : Agents cooperation

The main assumption behind this is that both agents may have also to answer to unpredictable requests made by other agents of the system. If one agent knows when some tasks should receive the answers they are waiting for, it will be able to know that for example one other task, with lower priority but ready to be executed/continued, should not be started even if the CPU time it requires to finish its current level is smaller than the delay before its deadline. It could realize that this CPU required is still too important considering the time when the waiting tasks with higher priority will receive the answer to their requests, and considering the CPU they will required by to finish their current level. Indeed, with the predictions, the agent will be able to assess when and for how long time the task ready to be executed will be interrupted, and then maybe realize that it will not be able to finish its current level. In that case, it will execute the next task in the schedule with lower priority that has some chance to provide another result before its own deadline.

We have made some statistics to assess the benefits of using the modified Earliest Deadline First scheduling policy we have proposed.

The test has consisted in generating a script of 500 requests like the one illustrated on figures 3x, to be answered by a system of 2 CAAM agents. One performing line-of-sight visibility calculation, the other performing path calculation. Each request is sent to the first agent that makes a request to the second in order to get the visible areas it needs to compute an optimal path. Each request has a deadline of 3 minutes, and follows the previous one after a random generated delay between 30 seconds and 3 minutes. This simulates a realistic use of this decision support facility by one user for more than 10 hours.

The same script has been used firstly with the 2 agents not performing and then performing the modified Earliest Deadline First scheduling policy. In the former case, the tasks share the agent CPU until their deadline without being preempted.

The results are shown in table 3. The second row of the table gives the average number of levels reached before the deadlines. The third row gives the average delay needed to get the results at each level.

Table 3 Without / With Scheduling

Levels	1	2	3	4
Average reached	500 / 500 +0%	500 / 500 +0%	412 / 500 +21%	21 / 170 +700%
Average delay (sec.)	5.2 / 4.4 - 15%	25.4 / 16.9 -33%	100 / 95 -5%	161 / 125 -22%

This table shows that using the scheduling improves the average quality of the answers delivered by the system

(170 calculations were able to reached the last level compared to 21). Moreover, it improves the reactivity allowing the system to provide results quicker.

Acknowledgments. This work has been done in the context of the GRACE EUCLID RTP6.1 collaboration agreement.

References

Adelantado M.; and Boniol F. 1993. Programming distributed reactive systems: a strong and weak synchronous coupling. In Proceedings of the 7th Workshop on distributed algorithms, Lausanne.

Body M.; and Dean T.L. 1994. Deliberation scheduling for problem-solving in time-constraint environments. *Artificial Intelligence* V67 no 2, June 1994.

Brunessaux S.; Charpillet F.; Haton J.P.; and Le Mentec J.C. 1992. ATOME-TR: Une architecture à base de connaissances multiples et orientée temps-réel, (in French),*Génie Logiciel & Systèmes Experts*, No 28, Sept.1992.

Chetto H.; and Chetto 1989. M. Some results of the earliest deadline scheduling algorithms. IEEE Transactions On Software, 6-15, No 10

Cohen P.; Greenberg M.; and Hart D. 1990. Real-Time problem solving in the Phoenix environment. Coins Technical Report 90-28.

Horvitz E. 1997. Reasoning about beliefs and actions under computational resource constraints. In Proceedings of the Workshop UAI 87.

Ingrand F.F.; Coutance V. 1993. PRS - REAKT: Procedural reasoning versus Blackboard architecture for Real-Time reasoning. In Proceedings of Avignon.

Lalanda P.; Charpillet F.; and Haton JP. 1992. Une architecture temps réel à base de tableau noir, (in French), In Proceedings of. Avignon 92, Vol.1, 671-682.

Mouaddib A; Charpillet F,; and Haton J.P. 1993. Real-time engine of messages for multi-agents architecture. In Proceedings of. Avignon, 589-599.

Mouaddib A.; and Zilberstein S. 1995. Knowledge-based anytime computation. In Proceedings of. IJCAI 95, 775-781.

Mouaddib A.; and Galone J. 1996. Progressive scheduling for real-time artificial intelligence tasks. In Proceedings of the 4th Annual Workshop On Real-Time Applications.

Musliner D.J.; Durfee E.H.; and Shin K.G. 1993. CIRCA: a Cooperative Intelligent Real-Time Control Architecture. IEEE Trans.Sys, Man.and Cybernetics, Vol.23, No6, Nov/Dec 93.

Musliner D.J.; Hendler J.A.; Agrawala A.K.; Durfee E.H.; Strosnider J.K.; and Paul C.J. 1995. The challenge of real-time AI. IEEE 0018-9162, Jan 95.

Salvant T., and Brunessaux St. 1997. Multi-Agent Based Time-Critical Decision Support for C3I Systems. In Proceedings of. Practical Application of Intelligent Agent and Multi-Agent Technology, PAAM'97, April 97, London

Salvant T., Brunessaux St. and Grumbach A. 1996. Coopération d'agents anytime pour la plannification en environnement temps-réel complexe. Proc. 3ièmes journées francophones IAD&SMA, Port Camargues, France (in French).

Zilberstein S. 1996. Using anytime algorithms in intelligent systems. AI Magazine.