MULTIAGENT-BASED SCHEDULING AND EXECUTION OF ACTIVITIES FOR SPACE SYSTEMS

Guido Sangiovanni[(1)], Simone Farè[(2)], Francesco Amigoni[(2)], Michèle Lavagna[(1)]

[(1)] Dipartimento di Ingegneria Aerospaziale – Politecnico di Milano, via La Masa, 34, 20156, Milano, Italy,
Email: sangiovanni@aero.polimi.it; lavagna@aero.polimi.it
[(2)] Dipartimento di Elettronica e Informazione – Politecnico di Milano,Piazza L. da Vinci, 32, 20133, Milano, Italy,
Email: cymon@tiscali.it; amigoni@elet.polimi.it

ABSTRACT

This paper presents the design and the preliminary experimental validation of an Artificial Intelligence-based scheduler intended to run onboard a space satellite. We developed the architecture of the proposed system according to a multiagent paradigm in which each agent is associated to a subsystem of the satellite. In this way, a number of benefits, including robustness, easy reuse of agents, and the possibility for the designer to focus on a small portion of the problem at a time, can be exploited. The experimental scenario is offered by Palamede, a Low Earth Orbit satellite under development at the Politecnico di Milano Aerospace Engineering Department.

1. INTRODUCTION

A completely autonomous space system must be capable of identifying the goals of the mission, planning its activities to reach these objectives, executing and monitoring the planned actions, detecting the presence of failures, and deploying some recovery strategies [1][2][5][6][7]. A lot of techniques to solve these problems, especially the planning and scheduling ones, are nowadays diffused in the Artificial Intelligence field. While planning means to build effective courses of actions to be undertaken in order to reach some goals, scheduling selects among alternative plans to appropriately manage resources concurrently needed by different actions in the plan [3]. It is currently required that, in the real world applications, these two aspects be integrated [4].
An increasing interest has been devoted to *distributed problem solving* in which a number of problem solvers, or *agents*, work together extending their own capabilities thanks to cooperation [9]. This paradigm is extremely useful when problems can be better modelled by a collection of agents that share resources. The cooperative parallelism generally improves the efficacy of single agents. Some methodologies have been created for dealing with such systems in the last years [8].

In this paper a spacecraft is seen as a collection of subsystems, each one represented by an agent, physically and functionally detached but highly interconnected and interdependent in order to reach the goals of the mission. The multiagent paradigm has been effectively used for dealing with the problem of the satellite management. Indeed a collection of agents that work and act locally on a common global problem is more robust than a single manager. The separation of functions, the decentralization of the activities and responsibilities, and the parallelism enable each agent to quickly answer to *local* unforeseen events or faults, without involving the overall system. Moreover, the multiagent paradigm allows the designer to focus on small portions of the problem at a time, simplifying the analysis of complex problems. For these reasons, in space applications the multiagent approach is increasingly employed, also because some future missions will be characterized by the presence of formations or constellations of satellites working together [7].
The main contribution of this paper is a two-level distributed scheduler for a multiagent system managing a spacecraft. The scheduling problem is formalized as a Distributed Constraint Satisfaction Problem (DCSP) at the system level [11] and as a Constraint Satisfaction Problem (CSP) at the agent level [10].
After a sketch of the overall architecture of the system, the distributed scheduler will be presented in Section 2. A description of the algorithms and of their implementation follows. The experimental scenario is presented in Section 4 and some performances analyses of the system are reported in Section 5. Related works and conclusions complete the paper.

2. OVERALL ARCHITECTURE

This work is the first step in a project aimed at developing a multiagent system for the management of a completely autonomous spacecraft. In Fig. 1 can be noticed that a single agent is designed to be composed of four different modules: a planner/negotiator, a scheduler, an executor, and a monitor: this is a very common high level layered architecture for agents in

Proc. of 'The 8th International Symposium on Artifical Intelligence, Robotics and Automation in Space - iSAIRAS', Munich, Germany.
5-8 September 2005, (ESA SP-603, August 2005)

space applications [1]. In general, a module interacts only with the adjacent modules in the vertical layered structure. All the agents of the system, or *agency*, share the same architecture. Each agent is responsible of a single subsystem or an on-board device of the spacecraft. The corresponding (i.e., at the same level) modules of different agents create a planner, a scheduler, an executor, and a monitor, respectively, that are distributed.

In this paper, we focus on the distributed scheduler that is the link between the physical world (the level at which the executor operates) and the deliberative abstract space of long-term plans. Its implementation is a first step toward proving the advantages of the overall approach – robustness, parallelism, reactivity, and management of shared resources – and analyzing the drawbacks related to its complexity and to the interactions between different modules and different agents.

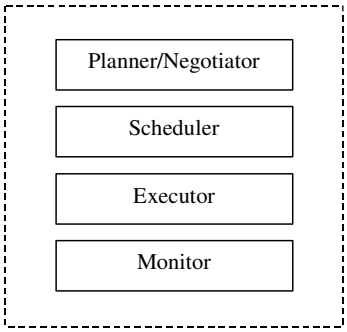

Fig. 1. Modules of an agent.

The distributed scheduler is made up of the scheduler modules of N agents, each one in charge of a set of activities characterized by duration and resource consumption and subject to some constraints. For example, in order to take a picture, before the Grab activity can be performed, the camera should be on, so the Camera ON activity should already be started. Both activities and constraints are currently user-defined, but they could be the output of the distributed planner.

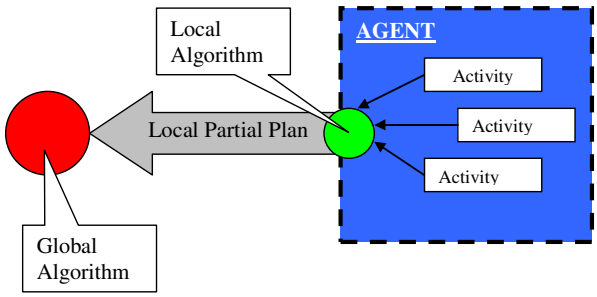

Fig. 2. Local and global levels of processing.

There are two different levels of processing in the scheduling process: the *local* one in which each agent determines the sequence of its internal activities according to their ordering constraints, and the *global* one in which external constraints are considered together with consistent consumption of shared resources (Fig. 2). At the local level, each agent produces a partial plan using a CSP algorithm based on dynamic backtracking [13]; at the global level, the agents harmonize their partial plans using a DCSP algorithm, called asynchronous weak commitment [11]. The final plan resulting from this cooperation respects the constraints on the resource consumption and on the activities sequencing. As soon as a plan is built, it is sent to an executor that performs the activities of the plan and, if errors occur, applies some fault-recovery procedures.

To take into account resources (i.e., electrical energy stored into the battery, on board memory, …), a manager agent is inserted into the agency for each resource: it is the only entity that has relationship with the physical resource. This manager is a wrapper that allows a better management of the information flows from and to the resource, hiding its real status and its intrinsic complexity to other agents. Moreover, a manager could also perform specific tasks, such as monitoring activities, *ad hoc* management and forecasting, or trend analysis. In the current implementation, we have a single resource: the electrical energy provided by batteries and solar arrays. However, other managers could be added, provided that the interactions protocol among the agents is modified accordingly.

3. ALGORITHMS AND THEIR IMPLEMENTATION

Both the local and the global algorithms used in this work belong to the CSP framework, with the difference that the latter is distributed: for details refer to [10][11]. The scheduling problem is formalized in this way:

1) there is a set of M activities to be allocated by the whole agency;
2) this set is partitioned by the user into N subsets; each subset of activities is relative to an agent;
3) each activity has one variable representing its starting time, whose time values can be taken from a finite set (the scheduling horizon); an activity is characterized also by other parameters like the duration;
4) each agent is characterized by a global variable, which is the vector of local starting times of its local activities;
5) the activities can have local or global constraints that involve other activities or resources (a local constraint is known only to an agent, while a

global constraint is known to all agents whose activities are constrained).

This formalization allows treating the global scheduling problem both as a DCSP, because each agent is responsible for its global variable, and, at the same time, as a collection of N CSPs local to the agents.
Solving the global problem amounts to find an assignment of values (i.e., starting times) to all the variables consistent with all the constraints, thus obtaining a solution both for the local CSPs and the global DCSP at the same time. It has to be noticed that the choice of using a local CSP method allows DCSP handling multiple local variables in a very easy way. Moreover, the proposed methodology transforms a problem of exponential complexity into a set of still exponential but smaller problems, thus obtaining an increase in performances.

3.1 The Global Algorithm

As previously said, the global algorithm of the distributed scheduler is a modification of the *Asynchronous Weak Commitment* (AWC) algorithm that we have chosen among other alternatives like *Asynchronous Backtracking* (AB) and *Distributed Breakout* (DB) [11]. According to AWC algorithm, each agent chooses an assignment to its variables and exchanges this information with the others, sending *ok()?* messages. Information coming from other agents is locally stored into a data structure called *agent_view*. Each agent has a priority: in the case of *nogood* situations, i.e., whenever some inconsistency of assignments with constraints is revealed, an agent increases its own priority and sends a new assignment to the others. A heuristic called *min-conflict* is used for choosing in an optimized way the agent that has to change its assignment.
It has been already proved that AWC is more efficient than AB because it can revise a bad decision without exhaustive search but by changing the priority order of agents dynamically [12]. On the other side, DB outperforms the AWC search when problem instances are critically difficult, because DB does an intensive analysis of all the possibilities of each agent before changing its values [11]. The system proposed in this paper automatically does this local work by using the local CSP algorithm, so DB is unnecessary. AWC could be also partially centralized by introducing an agent that receives and evaluate the *nogood* messages, and this property fits well with the proposed architecture.
The most important innovations we introduced with respect to the standard AWC described in [11] and [12] are the use of the resource manager, the behavior in case of *nogood* situations, and the use of a local CSP algorithm for the choice of new assignments to the variables.

In Fig. 3 the beginning of the scheduling activity is depicted. The agency is composed of N scheduling agents (blue circles) and the manager that represents the resource (the electrical energy). The manager determines the resource availability over a future time window and then transmits this information (red arrows) to the other agents that start local scheduling (DB_R means Dynamic Backtracking with information about resource; this algorithm is explained later). According to the AWC method, each agent maintains the current value assignments of other agents, namely their local plans, in the *agent_view*. At the beginning, each *agent_view* is empty because the agents don't know the commitments of each other. Moreover, global constraints are not taken into account, so every partial plan is built in order to satisfy only local constraints.

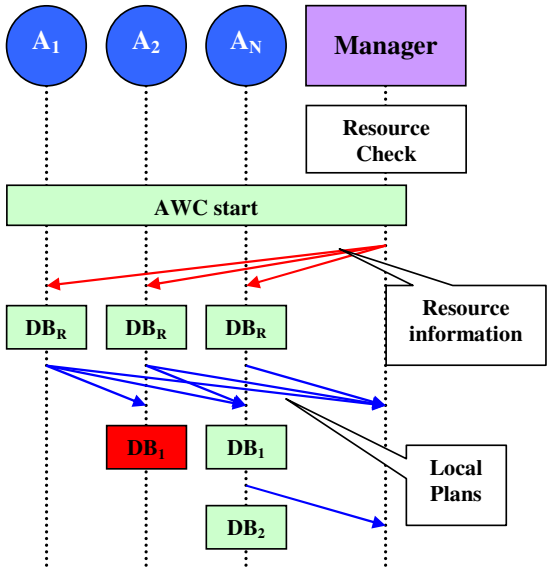

Fig. 3. Sequence of actions performed and messages exchanged by agents.

The instantiation order of agents creates the priority system required by the AWC. An agent changes its local plan if it is not consistent with the local plan of any higher priority agents. If no local plans is consistent with those of higher priority agents, the agent increases its priority and sends a *nogood* message to all the other agents and to the manager. A *nogood* message contains its current local plan and the new priority of the agent. This protocol of interaction guarantees the asynchronous modifications of an agent's *agent_view* that are driven by the arrival of others' local plans with associated priorities.

Following the AWC protocol, each agent sends its local plan to those with lower priority but also to the manager (blue arrows): these plans are inserted into the *agent_view* of the receiving agents.
After such a message is received by an agent, two cases are possible:

1) the plan received by the agent is consistent with all the global constraints involving the receiving agent so no modification is required (in Fig. 3 the green boxes DB_1 and DB_2, where the number refers to the agent from which the new partial plan comes);
2) the plan violates some of the global constraints involving the receiving agent, so the local algorithm is again used but with a higher level of information about other agents (red DB_1); if the new local plan is consistent it is sent again to the agents with lower priority and to the manager. Whenever a consistent local plan could not be found, the agent performs a relaxation of the constraints. This operation involves only resources and global activities and has the effect that, when the agent checks the consistency of the plan, it tolerates small extra resource consumption beyond the limit and does not take into account global constraints. The new relaxed plan contains some flaws, so a *nogood* situation is encountered. Also in this case, the agent augments its priority and transmits its new plan, thus forcing the other agents to follow its assignments.

In the standard AWC algorithm, *nogood* situations are recorded to guarantee the completeness by preventing loops. With the proposed methodology, the local algorithm sets the values of the variables so the completeness is guaranteed at the local level, but not globally because *nogood* situations are neither communicated to other agents nor recorded. In practice, the number of possible local schedules is very large so the probability of entering a loop is almost zero.
By receiving every local plan generated by the agents, the manager can monitor the evolution of the search from the resource's point of view and can stop the process when a solution is found. In this case, the manager communicates to the agents the local plans to which they should commit with during execution.

3.2 The Local Algorithm

The local algorithm of agents is an improved version of the Dynamic Backtracking method [13]. For every activity of an agent, the Elimination Explanation (EE) matrix registers the time positions that are not allowed for the activity because constrained by other activities, with an indication of these constraining activities in order to backjump to them and change their assignments, if necessary. The local algorithm iteratively chooses an activity, creates its EE matrix, and check if allowable positions are present: if yes, the time location with the maximal level of available resource is chosen. This heuristic is called *maximum availability heuristic*. When no time window matches the requirements, the algorithm backjumps to the scheduling of the activity responsible of the flaw by analyzing the EE matrix. A constraint processor is also implemented, allowing the pre-processing of constraints before the creation of the EE matrix: it identifies cases where no solution exists, putting the agent into a *nogood* situation.

3.3 Tools and Languages for implementation

The JADE (Java Agent DEvelopment framework) open source software platform for peer-to-peer agent-based application, fully implemented in Java language and distributed by TILAB, has been used for the development of the system. It complies with FIPA specifications, which means that the communication protocols are standardized and already available [15]. All the knowledge (the activities, their characteristics, the scheduling and execution parameters...) is coded in XML.

4. TEST CASE: THE MICROSATELLITE PALAMEDE

Palamede is a Low Earth Orbit satellite under development at the Politecnico di Milano Aerospace Engineering Department. Its main task is to take pictures of the Earth surface and transmit them to a ground station. We associated an agent to each subsystem of the satellite. More precisely, the agency we developed is composed by the Camera agent, which manages the activities of taking pictures, the ADCS agent, which controls the Attitude Determination and Control System of the satellite, and the ORBCOMM agent, which manages the communication flows between Palamede and the ground station via the ORBCOMM transponder. The onboard electrical energy is provided by five body-mounted solar arrays and a Li-Ion battery assembly. By applying a semi-regulated bus system topology, the nominal mode power is provided by the solar arrays; whenever extra power is available, it is used for batteries recharging, nominally discharged during eclipses. To take into account the power consumption in the scheduling activities, a manager agent is associated to this depletable resource: the Battery agent.
In order to accomplish their tasks in a realistic scenario, agents should perform the activities reported in Fig. 4. The durations and power consumptions are those expected in the real spacecraft: as the total expected power produced by the solar arrays will be

approximately 20 W and the on-board data handling subsystem continuously needs 9 W, it is clear that the management of the power resource is very critical.

Constraints related to activities are temporal constraints, both global (when involving activities assigned to different agents) and local (when involving activities assigned to a single agent), and belong to the following classes:

1) *TimeConstraint(Activity a, char boa, int instant)* – the activity *a* must be scheduled before or after *instant* if *boa* is respectively *B* or *A*;
2) *Eclipse Constraint(Activity a, char code, int min_dist, int max_dist)* – *a* must be inserted before, after, during or out of eclipses according to the value of *code* (*B,A,D, or X*), with a certain distance within the range specified in last parameters;
3) *OverlapConstraint(Activity a, Activity b, boolean avoid)* – if *avoid* is *false a* must overlap *b*;
4) *OrderConstraint(Activity a, Activity b, String code, int min_dist, int max_dist)* – the order constraint between *b* and *a* depends upon the value of *code* (e.g. *SAE* means *a* Starts After the End of *b*).

Constraints related to resources are global so they are handled directly by resource manager that checks the aggregate demand of power.

Agent	Activity	Duration [s]	Power Consumption [W]
Camera	Camera On	90	3
	Grab	30	1.6
	GPS	30	2.3
ORBCOMM	ORBCOMM On	600/120	1
	Transmit Photo	450	25.2
	Transmit HK	30	25.2
Battery	Charge	300	9
	Discharge	1800/150	-12/-18
ADCS	Read Attitude	30	0.5
	Control Attitude	120	3.8

Fig. 4. Activities of the agents of Palamede.

5. EXPERIMENTAL RESULTS

We have performed several experiments giving to the scheduler some goals to be reached. These goals correspond to the activities to be performed by the satellite in a temporal horizon of two orbits: the number of photos to be taken, the number of data and HK (HouseKeeping) transmissions and the battery charging periods. Processing the high-level requests, e.g., take two photos, a special component of the scheduler infers all the low-level activities needed to perform them and defines the constraints among these activities. In the future the distributed planner will perform this operation. The indicators of performance of the system are the fraction of the successful schedules and the time required to reach a consistent plan.

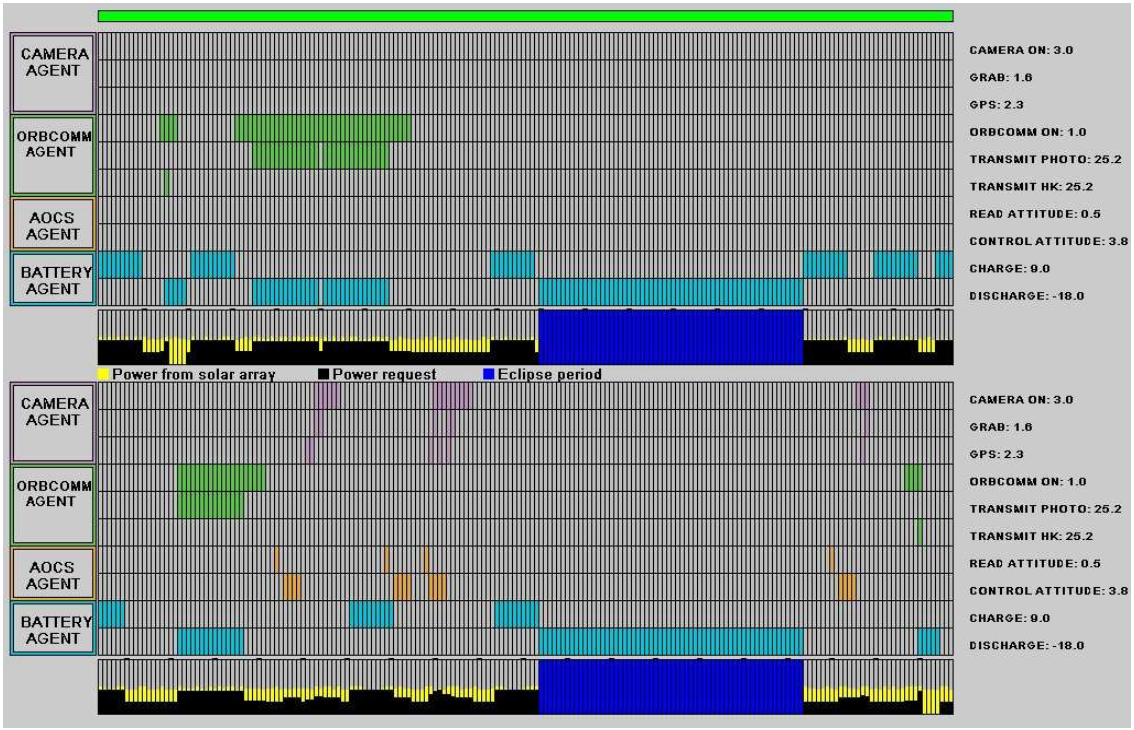

Fig. 5. A test schedule resulting from the global merging of local plans

Fig. 5 shows an example of schedule that allows the spacecraft turning on camera three times for grabbing six pictures, and transmitting housekeeping data and photos respectively two and three times; moreover, the battery is recharged in eight periods of five minutes each. Since at the moment the agency is not able to manage the state of the satellite before and after the scheduling horizon, no order constraints among goals have been inserted and the distributed scheduler has to respect only temporal and resources constraints, as previously described. The proposed system finds a solution to this test case in almost the totality of trials (499/500) with an average time for finding a solution of about 8 seconds. It has to be remarked that the problem is characterized by 52 variables and almost 200 binary constraints, so it is not trivial.

In order to better evaluate the performances of the distributed scheduler, 192 scenarios have been created by combining the number of goal activities to be reached: number of periods of camera on could be 2 or 3, number of photos to be taken could be 3, 5, 7 or 9, number of transmissions of HK could be 2 or 4 and those of pictures is comprised between 2 and 5, and number of charges could be 4, 6 or 8. The scheduler has to analyze 10 times (trials) each scenario. Tests were performed with a computer equipped with a Pentium III 500 MHz processor and 128 Mb RAM. A timeout of 3 minutes has been given to the system for solving each trial: whenever this limit was reached the trial was stopped and declared unsuccessful. About the 60% of the attempts (1180 on 1920) gave consistent plans before the timeout. 76 on 192 scenarios have been always solved, while in 26 on 192 scenarios (~14%) a solution was never found. An analysis of the performances of the system has been done from the point of view of the number of activities required to be scheduled. Changing from 2 to 3 the number of periods during which the camera is turned on does not change significantly neither the percentage of successful cases nor the average solving time; the same happens when the number of battery charges is changed.

The incidence on performances of the number of transmissions both of HK and of pictures is lower than that of the grabs. Fig. 6 and Fig. 7 show that the percentage of success decreases with the increasing of the number of transmissions requested, passing from 73% to 47% for photo (for 2 to 5) and from 70% to 50% for HK (for 2 to 4). The average solving time of success increases too, from 40 to 51 and from 37 to 53 seconds, respectively. Globally, the performances are not getting worst so much by increasing the number of transmissions. This is a very important result because the activities involving transmission are the more expensive in term of power consumption, thus requiring the use of battery. This positive result demonstrates the efficiency of the maximum availability heuristic used in the local algorithm and also of the overall methodology for the management of the resource.

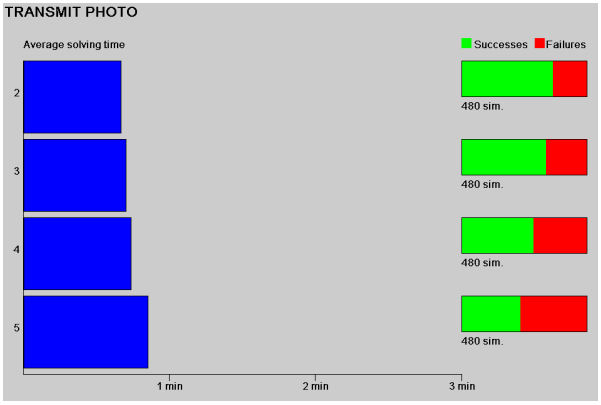

Fig. 6. Performances versus the number of photo transmissions.

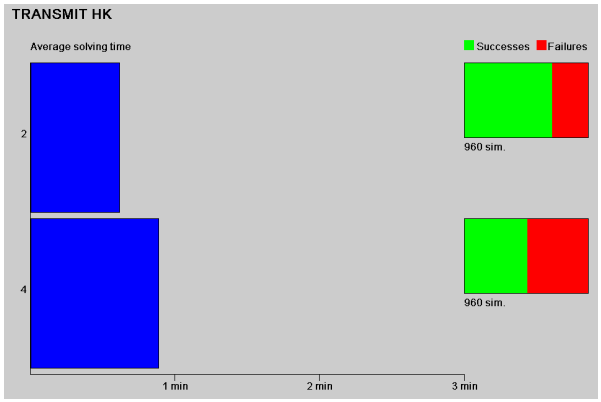

Fig. 7. Performances versus the number of HK transmissions.

The analysis of the performances related to grab activities (i.e., the actions of taking pictures) shows that both the percentage of failures and the average solving time increase with the number of photos required (Fig. 8). These activities are characterized by the shortest duration and very low power consumption but also by several constraints with external activities managed by other agents (red arrows in Fig. 9): according to the real mission plan, an attitude control action has to be done before two grabs and photo cannot be taken during eclipse. Grab is the most constrained activity, representing a very critical node in the network of relationship among actions.

The introduction of the timeout is a heuristic that allows avoiding useless computation in case of impossible solution; as already explained, this is necessary because our adaptation of the standard AWC makes the global algorithm not complete. In order to evaluate the incidence of this parameter on the whole system, the 26 failed scenarios were considered again with a timeout of 5 minutes (instead of 3). With 7 trials

for each scenario, the scheduler with extended timeouts has solved 15 out of 26 problems (~58%), proving the high sensitivity to this parameter of the system and the difficulty to set a general limit between hard and impossible scenarios.
As already explained in previous sections, the proposed architecture could be very efficient facing faults and unforeseen events (e.g., modification of the mission goals or unavailability of a device). Whenever an event makes a local plan inconsistent, consequences upon the global one are expected. The multiagent paradigm allows solving this problem exploiting decentralization: the single agent could use its local CSP method in order to adjust the previous plan according the new scenario without changing others' assignments. Whenever the solution to the problem requires an intervention at the agency level, the global AWC algorithm is used. Note that the same approach on two levels could be used also for the execution phase.

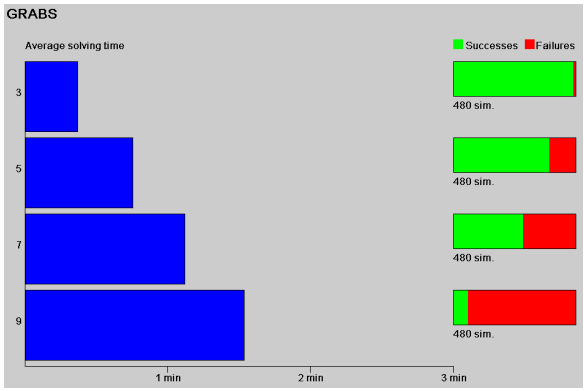

Fig. 8. Performances versus the number of grabbing activities.

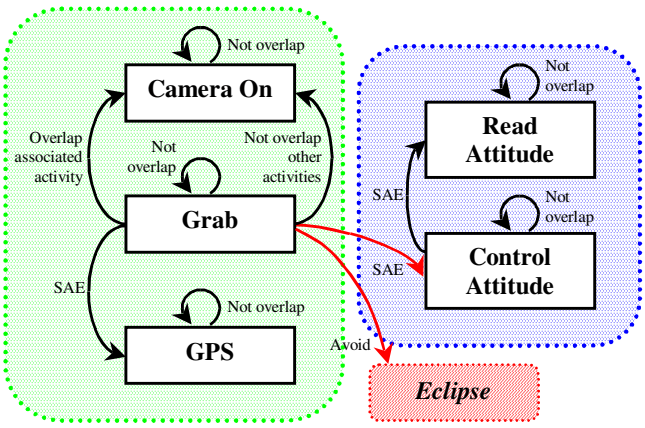

Fig. 9. Local and global constraints of the grab activity: the Camera agent activities are in the green set, those of the ADCS agent are in the blue one.

6. RELATED WORKS

A lot of works has been done with focus on the autonomy of spacecrafts. NASA's Remote Agent probably represents the most relevant benchmark; it has been implemented on-board during the Deep Space One mission (1998) and it is able to plan, execute, and monitor the activities and recover from some failures. It has not a dedicated scheduler but one of its modules is an integrated planner/scheduler that generates a set of time-based event-based activities, known as tokens [1].
ASPEN and CASPER constitute other relevant approaches: the first is a planner/scheduler system realized for managing ground operation while the latter is its soft real-time version that performs a continual re-planning using the iterative repairing methodology [5][6].
The scheduler proposed in this paper is not comparable directly to these three systems but it is possible to recognize the same constraint-based interval approach of [7]. The main difference is that, in this approach, the variables are the starting and the ending time of activities.
NASA is also studying extensions of the above architectures: the multiagent paradigm has been chosen to obtain relevant enhancements. In [7] an agency has been built starting from the planning level using the Hierarchical Task Network paradigm, whereas in our work the attention is currently focused on the scheduling and on the robustness allowed by the multiagent approach.

7. CONCLUSION

The preliminary validation of the proposed distributed scheduler is satisfactory, even if a more detailed analysis has to be performed.
A simple distributed executor has been realized too with the aim of analyzing some fault-recovery procedures. The execution modules are completely reactive and no communication among them is present, similar to RAPS [14]. Whenever a local failure occurs, the execution modules tries to perform the activity again until the scheduled ending time is reached.
Next steps of the work will be the improvement of the distributed executor and the implementation of complex fault-recovery procedures capable of taking full advantage from the multiagent paradigm. Last step of the project will be the creation of a distributed planner/negotiator that will complete the overall architecture.

8. REFERENCES

1. N. Muscettola, P. Nayak, B. Pell and B. Williams, "Remote Agent: To Boldy Go Where No AI System Has Gone Before", *Artificial Intelligence* 103 (1998) 5-47.

2. M. Lavagna, G. Sangiovanni, A. Da Costa, "Modelization, failures identification and high-level recovery in fast varying non-linear dynamical systems for space autonomy", *Proc. 6th Cranfield Conference on Dynamics and Control of Systems and Structures in Space*, Riomaggiore, Italia, 18-22 July 2004, p.541-550.

3. M. Zweben, et al., *Intelligent Scheduling*, Morgan Kaufmann editor, 1994.

4. D. E. Smith, J. Frank, A. K. Jonsson, Bridging the Gap Between Planning and Scheduling, *Knowledge Engineering Review*, 15(1), 2000.

5. S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, D. Tran , "ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling," *SpaceOps 2000*, Toulouse, France, June 2000.

6. S. Chien, R. Knight, A. Stechert, R. Sherwood, G. Rabideau, "Integrated Planning and Execution for Autonomous Spacecraft," *Proc. of the IEEE Aerospace Conference (IAC)*, Aspen, CO, March 1999.

7. S. Das, P. Gonsalves, R. Krikorian, W. Truszkowski, "Multi-Agent Planning and Scheduling environment for enhanced spacecraft autonomy", *Proc. of 5th International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS99)*, Noodwijk, March 1999.

8. M. Wooldridge, *An Introduction to MultiAgent Systems*, Wiley&Sons ed., Chichester, England, February 2002.

9. E. Durfee, V. Lesser, D. Corkill, Trends in Cooperative Distributed Problem Solving, *IEEE Trans. On Knowledge and Data Engineering* vol. 1 n. 1 March 1989.

10. V. Kumar, "Algorithms for Constraint-Satisfaction Problems: A Survey", *AI Magazine* 13(1):32-44, 1992.

11. M. Yokoo, K. Hirayama, "Algorithms for Distributed Constraint Satisfaction: A Review", *Autonomous Agents and Multi-Agent Systems*, vol 3, n.2, 2000.

12. M. Yokoo, E. H. Durfee, T. Ishida, K. Kuwabara, "Distributed Constraint Satisfaction Problem: Formalization and Algorithms", *IEEE Trans. on Knowledge and Data Engineering*, vol.10, No.5, 1998.

13. M. L. Ginsberg, "Dynamic backtracking", *Journal of AI Research*, 1:25--46, 1993.

14. R. J. Firby, "Task Networks for Controlling Continuous Processes", *Proc. of the Second International Conference on AI Planning Systems*, Chicago IL, June 1994.

15. F. Bellifemine, A. Poggi, G. Rimassa, JADE - A FIPA compliant agent framework, In *Proc. Fourth International Conference on the Practical Application of Intelligent Agent and Multi Agent Technology (PAAM99)*, pp. 97-108, London, U.K., 1999.