CONSTRAINT-BASED LAYERED PLANNING AND DISTRIBUTED CONTROL FOR AN AUTONOMOUS SPACECRAFT FORMATION FLYING

Jean-Clair Poncet(1), Christophe Guettier(2), Gérard Le Lann(3), Eric Bornschlegl (4)

(1)Axlog Ingénierie
19-21, rue du 8 mai 1945, F-94110 Arcueil
jean-clair.poncet@axlog.fr

(2)Xerox Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304
guettier@parc.xerox.com

(3)INRIA
B.P. 105, 78153 Le Chesnay Cedex, France
gerard.le_lann@inria.fr

(4)ESA/ESTEC
Keplerlaan 1, P.O Box 299, NL-2202 AG Noordwijk
eric.bornschlegl@esa.int

INTRODUCTION

For autonomous space systems, particularly satellite-borne systems, the minimization of human intervention in the system loop exacerbates the classical requirements of safety, survivability and availability. Furthermore, such systems call for tighter "close loop interleaving" between embedded operation, planning and decision with finer grain real time control and command. For a set of autonomous spacecraft flying in formation (FF), group management introduces some additional complexity, especially regarding global command and control.

Defining spacecraft formation in terms of a Multi-Agent System (MAS) enables the integration of automatic reasoning functions into a distributed model of architecture. Locally, an agent can sense, reason, operate and actuate using its knowledge from different levels of automation retroactively. Globally, with intelligent coordination protocols, a multi-agent system can elaborate collaborative behaviors to achieve a common goal. However, to be integrated in future spacecraft systems, reasoning algorithms and coordination protocols must be well suited and powerful enough to solve real-size decision, operation and control problems as well as meet timeliness and liveliness properties despite the distributed nature of the underlying system, and despite occurrence of failures.

The approach we have adopted separates the applicative semantics from the system semantic. On the one hand, by following a Proof-Based System Engineering method, one can design a computer-based system which provably meets its specification, i.e. which behaves as expected. On the other hand, planning functions based on Constraint Programming (CP) [9,10] are strengthening Multi-Agent Systems for dedicated domains, such as space and aeronautic. In fact, the approach allows powerful specifications and an efficient solving of collaborative and coordination problems [7,8]. Furthermore, when specifying and solving planning problems using (CP), one can prove a correct usage of applicative devices and on-board resources along spacecraft operations. Despite being co-designed, application and system components can be easily combined and integrated so as to build a space system.

ARCHITECTURES AND DESIGNS FOR AUTONOMOUS FORMATIONS

In order to meet those requirements proper to autonomous space systems, designs must be proven correct, for behaviors remarkably adaptive and responsive to partially unknown environments. By mapping a multi-agents architecture onto a spacecraft formation, it is possible to combine state-of-the-art in real-time fault-tolerant distributed systems with automatic reasoning.

At first, it is necessary to know how to architect and design computer-based systems that meet those requirements deriving from autonomy. Proof-based system engineering (SE) is a critical discipline in this respect. Proof-based SE [5] rests on such mature theories as, e.g., the real-time scheduling theory and the distributed algorithms theory. Results established in these fields are essential in addressing successfully the challenges involved with establishing designs which are provably correct, despite postulating the absence of global knowledge, as well as failure occurrence. Proofs of such properties as safety, liveness, timeliness, dependability, rest on mathematical analyses that are well mastered. A significant number of problems in the area of real-time distributed fault-tolerant computing are closed. It turns out that (1) this vast amount of knowledge is partly ignored by scientists other than theoreticians in computer science, (2) some

of the solutions that are available match some essential problems raised with autonomous systems. Conversely, autonomy raises new issues, i.e. new open problems in computer science. These are the incentives for setting up multi-expertise teams, a sensible approach to achieving a much-needed cross-fertilization.

Recent experience in proof-based SE, gathered through a number of real world projects, shows that it is possible to conduct an application requirements "capture" phase satisfactorily, that is ending up with an unambiguous specification <A> of the application problem considered, as well as an unambiguous specification <X> of the computing problem that matches <A>. In our case, applications are autonomous missions and spacecraft formations. <A> embodies the application semantics, while <X> embodies system semantics. Lack of ambiguity is a reachable goal, despite the fact that an autonomous mission involves some – possibly large – degree of uncertainty.

An unambiguous specification is tractable through scientific means (techniques, tools). In other words, a design specification – a solution for a computing problem which has been "captured" – is another unambiguous specification [S] which provably solves <X>. Design specifications are modular. If so desired, modules may be mapped onto a distributed architecture. The fact that every module (a software program, a piece of hardware) comes with a complete and unambiguous specification greatly simplifies the (usually daunting) global verification task. It suffices to verify the implementation of each module specification, in isolation from each other. Also, a specification such as [S] can be refined (i.e. modularized) as much as desired, in order to bring the complexity of global solution S down to some tractable level. Proof obligations met while conducting a design phase guarantee that the problems stated in <X>, which inevitably result from asynchrony, concurrency, lack of central locus of control, partial failures, variability in processing and communication delays – to name a few – have been correctly addressed and solved.

Basically, there are three classes of generic problems which are relevant to autonomous spacecraft formations, namely distributed computing, fault-tolerance, real-time scheduling. One prominent generic composite problem is that of real-time distributed coordination. Emerging solutions for that problem combine solutions established over the last ten years for "simpler" problems, such as (fault-tolerant) uniform consensus/atomic broadcast [3] with solutions established more recently for on-line distributed scheduling of real-time tasks [1]. Major difficulties lie with the choice of appropriate computational models, failure models, and load models, which must match reality as much as possible (i.e. with very "high" coverage), without designers ending up faced with impossibility results, or led to contemplate solutions of exponential complexity.

MULTI-AGENT SYSTEMS

Agent systems have been widely investigated in the last ten years [13]. They have been proven to be a powerful mean to model the behavior of distributed autonomous or partially autonomous entities interacting together and with the environment. The agent concept has already been used in the space domain to improve autonomy techniques during the Deep Space One mission [12]. However, with autonomous formation flying, one must consider distributed systems, where several autonomous entities are supposed to act together in order to satisfy mission objectives. Following a proof-based SE approach, we propose to investigate the class of hybrid multi-agent architectures designed for the formation management, command and control problems. Within this context, a global problem <A> has been decomposed into independent sub-problems <A1>, <A2>,..,<An> according to local and temporal criteria. A hierarchy reflects this problem decomposition and structures the multi-agent systems. Furthermore, some of the problems are relevant to decision, planning and scheduling theories, while others are relevant to distributed systems theory.

Agent Architecture

Inside a formation, one agent is associated to each spacecraft, giving a perfect mapping between spacecraft system and Multi-Agent System. Each agent encompasses platform, payload, hardware and software of a spacecraft. According to standard denomination of agent types, it corresponds to an hybrid reactive-deliberative architecture, where functionalities are dispatched onto three distinct levels (Figure 1). According to its position in the system's hierarchy, skills can be different from one agent to another. Figures 2 and 3 illustrate template leader agent architecture and template subordinate agent architecture. These templates could be instantiated differently from one agent to another, according to the expected system structure and agent's position in this system.

Figure 1 illustrates this architecture. At the highest level, decisional procedures and functions implement the deliberative and social parts of agent. This high level is made of four main components. One component consists in a generic mission planner that can build medium and long term plans for the whole formation. Another component is a behavior based control component that determines the behavior of the agent according to the current situation. A knowledge base component stores all facts and beliefs of the agent about the current formation and environment situation. Finally, a general control component is in charge of the activation management and the synchronization of actions of the three other components. A lower level includes the automated components of an agent architecture:

Command and Control (C&C), Fault Detection Isolation and Recovery (FDIR), Platform and Payload Systems (PPS, including sensors), and Communication. A large part of an agent's reactive behavior relies on these automated low-level components. Macro sequences of tasks performed at the C&C and PPS levels are represented using non-deterministic finite state automata. Typical tasks are Pointing, Calibrating, Attitude and Orbit Control, Emergency or Communications (Figure 3), most of them corresponding to a continuous control. Tasks activation depends on the current state of the automata and is performed by the real-time executive. At the bottom of this three level stack, one finds the middleware, system-ware and hardware layers. To support tasks execution, the middleware layer combines a constraint solver (based on AC5 algorithms), a real-time executive, as well as reliable communication and consensus services. These architectural features are discussed in the sequel.

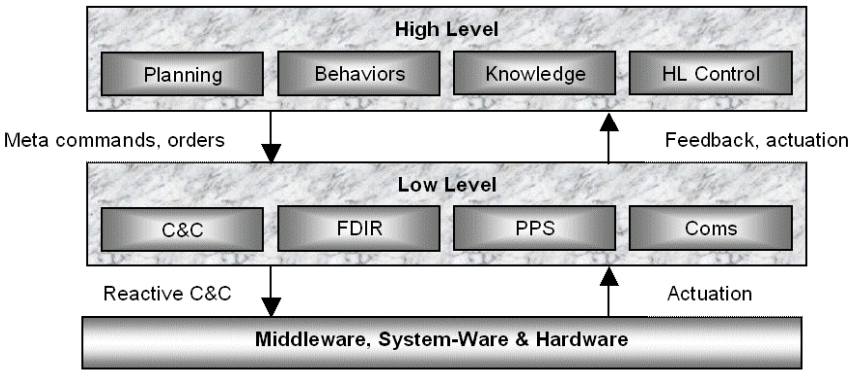

Figure 1 ─ Three-level agent architecture

Distributed Architecture

Inside the MAS mapped onto the spacecraft formation, each agent has a role, directing and imposing its behavior all along the formation course. This role is statically assigned to each agent at the creation of the system, but role assignment can be changed according to possible external and/or unpredicted events. The way roles are dynamically re-assigned to agents depends on the system definition. Basically, many agent organizations are possible, from a completely structured hierarchy to a full negotiation oriented system. In the case of spacecraft formation, it is important to minimize the communication between spacecraft (and thus between agents) as they are power consuming. A pyramidal hierarchy made of a single leader and a set of subordinates enables one to dispatch the system roles over agents, while minimizing the necessary amount of communication and ensuring the formation safety and liveness.

Agent Hierarchy

In this approach, the MAS is structured according to a hierarchical tree. At the top node of the system tree, a single leader agent is in charge of mission planning according to the mission goals that have to be satisfied, spacecraft resource availability and realization possibilities. Leaves of that tree-based system hierarchy are devoted to other agents, considered as subordinates of the leader. A subordinate receives the mission plan and shall realize the included sub-plan that has been computed for it. However, in order to be able to compute a valid mission plan, the leader of a formation shall be permanently provided with the operational situation of each of its subordinates. Thus a status reporting mechanism from one spacecraft to others enables each agent of the system to be continuously informed of the situation and availability of each other agent. This reporting conducted regularly, at frequencies that depend on the system evolution speed.

Using such a hierarchical approach enables one to restrict the main communications to the broadcasting of plans' data and status reporting. This is considerably less expensive in time and resource consumption than the negotiation protocols (for example the Contract Net Protocols family) required realizing efficient mission planning for a non-hierarchy based distributed system.

However, negotiation protocols are always required to the dynamical autonomous reconfiguration of the agent system. When the agent acting as the leader is lost (out of communication reach, or down), remaining agents shall be able to reconfigure themselves in order to maintain the system structure and thus the viability of the formation. This requires electing a new leader using some distributed fault-tolerant leader election algorithm, which is a particular instance of a consensus problem. Therefore, as long as there are enough agents able to take on the role of a leader, computer or communication failures do not jeopardize the outcome of a mission.

Collaboration, cooperation & synchronization

As a side effect of centralized mission planning, sub-problems like collaboration, cooperation [8] and part of the synchronization can't completely rely onto agent knowledge exchange at planning time and execution time. Thus, the centralized mission planning shall solve part of collaboration and coordination, stating constraints relative to each partial plan of subordinate agents. In a collaborative maneuver, several spacecraft will have to act together in order to meet the same global goal. It is the responsibility of the mission planning to split this goal into several independent sub-goals and to allocate them to different agents of the formation. While doing this, the mission-planning function is able to compute the amount of local and shared resources that will be necessary to realize these goals and then to predict resource consumption up to the plan horizon.

Identically, synchronization of agents during the plan execution cannot completely rely on message passing and other agent-to-agent communication and localization means. In order to ensure that all participants will conduct phases of the plan requiring synchronization of several agents at the right time, the planning function shall introduce constraints between the related partial plans. These constraints link together sub goals of the same goal in order to guarantee that they will be executed in the same time interval. At the execution time, when several agents shall synchronize themselves for the realization of a common goal, effective synchronization is made using a barrier algorithm.

CONSTRAINT BASED PLANNING AND OPERATION COMMAND

Constraint programming techniques [10][11] have been shown to be highly effective, predictive and modular for solving combinatorial problems related to autonomy, such as on-board planning and scheduling or optimized control. Expanded with incremental approaches to constraint solving, constraint programming can tackle reactive forms of planning, scheduling and control. Currently, the complete planning and scheduling of spacecraft activities for long term autonomous periods satisfying hard reactivity properties is out of the scope of conventional constraint solvers. To fulfill mission goals, various formations, spacecraft and control constraints have to be considered altogether by the on-board system. For example, a global consensus must be reached for operations coordination (maneuvers, thrusting…) and synchronization (observations, warm-up…), compliant with trajectories, energy consumption, payload allocation and systems states. In theory, guaranteeing the completeness of the solving leads to the consideration of the global problem in a single formulation. In practice, according to present on-board processing power, such global problems are beyond the reach of constraint solvers and must be broken down in independent sub-problems, using upper-approximations, sufficient conditions and knowledge abstractions [8]. Then, resulting independent sub-problems can be distributed over several spacecraft, leading to a more adaptive and reliable architecture, where local knowledge can be processed locally. Additionally, this kind of hierarchical plans enables one optimizations of processing resources among the whole formation.

The aim of long-term and medium-term mission planning is to define a macro sequence of actions that meets the mission goals while ensuring the global formation liveness, safety, and timeliness. According to the type of the mission, such as Low Earth Orbit (LEO) formations of spacecraft or deep space probes, a mission goal can be an observation of a ground area, a landing over a celestial body, a specific attitude to adopt at a defined time or any other kind of mission objective. Resulting actions shall be defined according to time and resource criteria, including the orbitography computing required determining the set of possible navigation points and trajectories. Thus, mission planning for a spacecraft formation comes down to solving a schedule of goal realizations, corresponding to operation and command actions, with resource constraints. This requires finding an optimized command and actuation sequence in order to meet mission objectives in reasonable time. Additionally, planning must be reactive enough to prevent the formation from catastrophic failure in case of unpredicted events in uncertain conditions of evolution.

Centralized Long-Term Mission Planning

According to these characteristics, a goal can only be realized in given space locations characterized by a set of potential long-term navigation points (associated with feasible time intervals). A goal description also encompasses a worst-case duration and some resource requirements. The navigation points where goals can be met are defined according to the formation potential pre-defined potential trajectories (processed from on-board devices such as a star tracker sensing results, or communicated by ground orbitography experts). Global mission planning determines and optimizes the selection of the navigation points that shall be reached by the formation. It parameterizes goals execution along the trajectory path (such as navigation or thrust sequence) for each spacecraft according to a trade off between goal realization possibilities and trajectory determination.

The mission planner is based on a multiple models approach [9], where each model abstracts a sub-part of the overall problem. Examples of such models are the navigation through pre-defined trajectories, approximation of resource consumption, goals execution and allocation. Additional agent-oriented goals over-constrain the global problem,

insuring the consistency of the long-term plan between spacecraft [7][8]. In addition to the constraint solver, those global long-term plans are elaborated using global search techniques that process a concurrent solving and constraints propagation over the set of models. Either the mission planning function is activated when new goals must be fulfilled by the formation or when the monitoring subsystem detects drifts or failures in the formation. In that case, the search control uses the existing plan as a heuristic and combines the global search with a repair algorithm in order to recover the situation or to stabilize the formation in a safe mode (Figure 2).

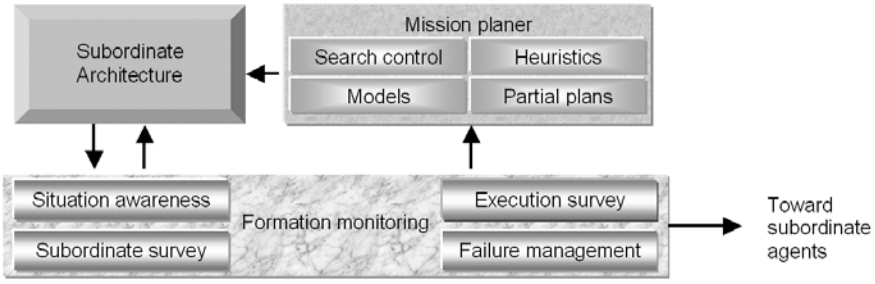

Figure 2 ─ Template leader architecture

Local Short-Term Spacecraft Planning

Each spacecraft, including the leader, comprises a set of equipments like actuators (thrusters, gyro-wheels…), sensors (star tracker, lasers, radars…), processing and communication systems. In our approach, each goal is a specification of an expected state to be reached in a given timeframe. An equipment state corresponds to a set of periodic/sporadic independent tasks subject to real-time constraints (periods are assumed equal to deadlines). Also, application-centric invariants, such as mutual exclusion (the thruster cannot be activated while a payload is on) and causal dependencies between states (a warm-up always precede a payload measure if the temperature is too low) are global constraints required by the spacecraft design. Achieving a goal consists in finding a set of command and control sequences for each device that satisfies global constraints as well as feasibility conditions of real-time task processing. To reach a given state, the equipment is represented as a non-deterministic automaton using a constraint modelbased representation (Figure 3). Then, the local control and command problem is to find dynamically a deterministic control sequence for all agent equipments. The solving over the set of constraint-based models considers predefined limits for on-board processing load and searches for a set of commands in the resulting feasible domain to provide a deterministic automaton. Within a control-loop, this approach is far more adaptive than static deterministic automata (like those proposed by formal languages, or Petri nets approaches), which generally encodes domain heuristics. Even more, deterministic heuristics can still be applied gradually to achieve the level of responsiveness required by the environment. The template architecture presented in Figure 3 relies strongly on constraint solver capabilities. In addition to optimized command & control sequences, it can solve various secondary goals relevant to mission performances and safety/survivability. For example, when a fault is detected and diagnosed, it is possible to assert the fault consequences by keeping the constraint model-based knowledge consistent. Also, it is possible to anticipate automata behavior for further leader monitoring (Figure 2).

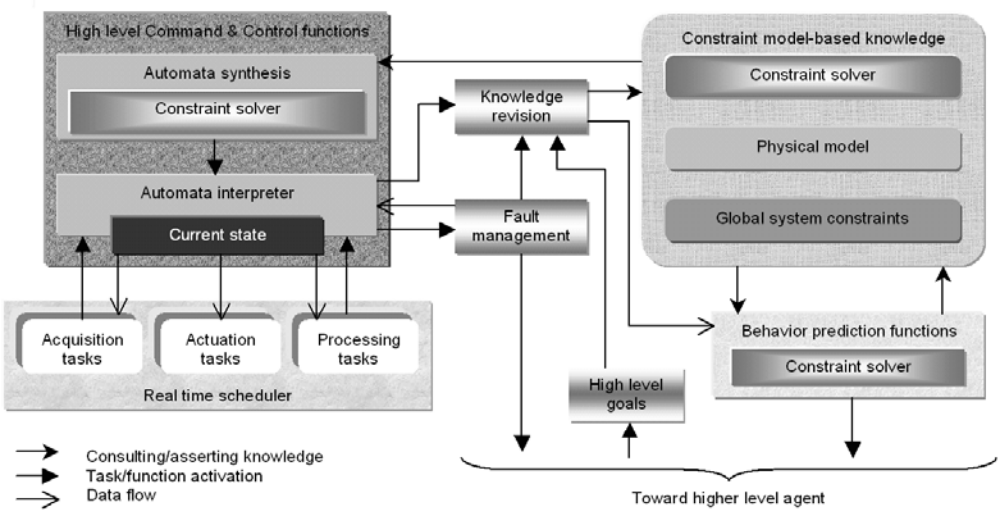

Figure 3 ─ Template subordinate

DISTRIBUTED COMPUTING SYSTEM

As indicated previously, there are three classes of generic problems which are relevant to computer-based systems used for autonomous spacecraft formations, namely distributed computing, fault-tolerance, real-time scheduling. The exact specification of which instances of such problems are to be tackled is denoted as <X>. We use the term "process" to refer to any kind of application-level computational and/or communication activity, and the term "processor" to refer to any kind of technological apparatus capable of running some process(es). Every processor is equipped with some system-ware, which contains all the protocols and algorithms needed for a correct behavior of a formation. With current (off-the-shelf, customized) technology, what we refer to as system-ware (or executives) encompasses operating systems kernels and middleware.

Distributed computing addresses issues raised with concurrent executions of processes, every process being distributed over some number of processors. The most difficult issues arise whenever processes share updateable persistent variables. Reads and writes performed at run-time over such variables must meet application-defined invariants (safety properties), which translates into required properties such as serializability or linearizability. It is almost never the case that, in nowadays systems, there are no (application-level) processes which share updateable persistent variables. Autonomous spacecraft formations are not unique in this respect.

Hence, systems used for autonomous spacecraft formations must be equipped with an executive that contains some form of concurrency control. For example, despite conflicting accesses to shared variables caused by interleaved executions of distributed processes, every run (or system history) is serializable, i.e. equivalent to some sequential execution of processes. Of course, concurrency control algorithms are designed so as to run asynchronously over any number of processors. Popular algorithms make use of logical timestamps/tickets, with or without locking (of persistent variables) [2]. Fault-tolerant computing addresses issues raised with the occurrence of partial failures at run-time. Algorithmic solutions fall in two broad categories: fault detection-and-recovery, fault masking. Depending on which computational models, which failure models, and which dependability properties are considered, fault-tolerant computing problems may have optimal solutions, or proven solutions, or may be proven impossible (e.g., no deterministic solution) [6]. Problems that are relevant to autonomous spacecraft formations and that have proven, sometimes optimal, solutions are "leader election", "uniform consensus/atomic broadcast". For example, there are (distributed) algorithms which are themselves immune to partial failures, and which emulate the equivalent of a unique, perpetual, leader agent, out of some number of asynchronous (and failing) agents.

More complicated are those issues raised with the need to "synchronize" replicated processes whenever some failure occurs. Synchronization may be needed for the reason that (replicated) shared variables were in the course of being modified when a failure occurred. Backward- or forward-error recovery cannot be contemplated unless one knows "where" to resume some aborted process. A particularly interesting form of "synchronization" is known under the name of uniform consensus. The paradigm is as follows, in its most interesting setting, that of asynchronous or partially synchronous models (It is reasonably obvious that there is one computational model which is to be disregarded when designing a system intended for autonomous spacecraft formations, and that is the synchronous model, because of its

very poor coverage). Initially, every process proposes some value. A process may fail while broadcasting its value to other processes. Every non-failed process must eventually pick up one of the values received, and apply it (this is called "decision"). Uniform consensus requires that the same (received) value is decided by every process that decides – and possibly fails right after, in some finite time (albeit unbounded).

Uniform atomic broadcast is uniform consensus with the additional requirement that whenever multiple instances of consensus are run concurrently, the order after which decisions are applied is unique system-wide.

We will see below that systems intended for autonomous spacecraft formations must be equipped with an executive that contains the equivalent of a real-time consensus/atomic broadcast algorithm.

Real-time process scheduling addresses issues raised with required timeliness properties, such as meeting absolute or relative strict deadlines, or strictly bounded jitters, for process termination, or for issuing unrecoverable state transitions (e.g., signaling the environment). Many scheduling problems raised with centralized systems are closed, or have proven solutions, under the form of "good" algorithms, which achieve sufficient feasibility conditions not too "far away" from necessary and sufficient conditions (problem-wise). Earliest-deadline-first is a well known example. Most centralized scheduling problems, in the presence of resources shared in mutual exclusion by processes, are NP-hard. Not too many "good" algorithms have been published so far. There are reasons for this.

Not too surprisingly, current state-of-the-art regarding real-time scheduling in distributed systems is even more limited. Local scheduling decisions made concurrently by executives running over processors are not independent (causal dependencies among processes, serializability/linearizability constraints, non-preemptable resources, and so on). The intrinsically highly combinatorial nature of scheduling problems can only be exacerbated in the presence of distributed control. Add to this the requirement that scheduling decisions – that are (at least partially) made on-line – must be correct also in the presence of partial failures.

Fortunately, it turns out that there is way out of these problems, and that is composite algorithm. Very intuitively, if we know how to "merge" some concurrency control algorithm, some "good" centralized scheduling algorithm, and some (fault-tolerant) consensus/atomic broadcast algorithm, then we should be able to circumvent those limitations that plague existing schedulers and executives, especially commercial off-the-shelf executives.

Also, almost as a "by-product", one solves the real-time uniform consensus/atomic broadcast problem, a problem that arises when the eventual termination property is replaced with a timeliness property.

Pushing forward these ideas, one identifies a prominent generic composite problem, which is that of real-time distributed coordination, a generalization of real-time uniform consensus/atomic broadcast [4].

Back to autonomous spacecraft formations. Examples of services which call for real-time uniform consensus/atomic broadcast algorithms are: system reconfiguration, agreement on a set of sensor values, agreement on unrecoverable actuators commands, global time, mutual consistency of replicated data (needed to achieve "high" availability of critical and/or historical data).

Examples of services which call for real-time distributed coordination are distributed process scheduling, mission planning or mission re-planning.

To conclude, it seems unavoidable that computer-based systems intended for autonomous spacecraft formations must be equipped with a distributed executive that would contain some of the algorithms briefly introduced in the above. Research and experiments are underway in these areas. Very promising results have been established recently regarding ultra-fast uniform consensus/coordination.

EXPERIMENTS

Mission Planning

The mission planning system has been first tested with a Low Earth Orbit (LEO) flying formation that has been supposed to be in charge of earth surface observation. According to the planning models defined in [7,8], several tests have been run for different levels of problem complexity including: optimization without resource constraints, optimization under resource constraints, optimization without periodic goalsor optimization including periodic goals (10%). The complexity has also been modulated according to the number of goals to schedule in the mission plan.

Resource Models	Periodic Goals	Variables	Constraints	Best Solution	End Search
NO	NO	24609	35188	16.46 s	16.55 s
NO	YES	27039	37919	21.96 s	22.07 s
YES	NO	229219	240251	81.68 s	82.32 s
YES	YES	237995	247977	105.38 s	106.30 s

Table 1 ─ Results for scheduling 150 goals for a five spacecraft formation in LEO

Results in Table 1 show the problem's complexity and the resolution time for scheduling 150 goals (observations, maneuvers, orbit changes, …) for a LEO formation of five spacecraft. Obviously, resource models introduce a large complexity in the planning problem, but resolution times remain good even when both resource management and periodic goals are introduced. Moreover, this table shows only the best solution for each sort of problem. A first solution with acceptable quality ratio can be found each time in a very short time (less than a second). For deep space missions, planning is made harder due to the determination of trajectory among possible navigation points (which is of course not the case for the LEO mission as trajectory is reduced to a simple elliptic orbit). Nevertheless, it is possible to schedule 40 mission goals for a five spacecraft formation with a trajectory graph including 20 vertices less than 80 seconds.

Local Planning

Local planning models have been tested with different automata. The example below corresponds to an automaton controlling the thrust of an electrical propulsion system (Figure 4). Transitions between states can be through commands or environment events. The planner attempts to define the best way to travel along the automaton according to initial temperature of the engine in order to reach a required speed.

As illustrated in Table 2, the structure of the solution found depends on initial conditions. These conditions are invariants of physical models associated with the engine components. The planning horizon is expressed in number of possible events (transitions of the automaton). The resolution time remains quite reasonable as no specific search nor heuristic has been used in this example. The increased amount of time to find the second solution (20° initial) is due to the higher complexity of the second solution that includes a warm-up sequence. The solution of each problem encompasses the resource consumption for each state of the planned path in the automaton.

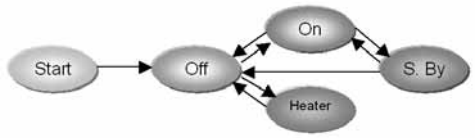

Figure 4 ─ Test automaton

T initial (C)	Planning Horizon	Resolution Time	Nb Transitions (required to reach the speed)
40°	15	1.3s	4
20°	15	3.5s	5

Table 2 ─ Results for automaton instantiation

CONCLUSION

ACKNOWLEDGEMENTS

We thank the European Space Agency (ESA-ESTEC) for supporting and supervising the major part of the work, which has led to the results presented in this paper. We thank Lara Crawford and Yi Shang for their comments on this paper.

REFERENCES

[1] P. Carrere, J.-F. Hermant, G. Le Lann, “In Pursuit of Correct Paradigms for Object-Oriented Real-Time Distributed Systems", 2nd IEEE ISORC Symposium, May 1999, 271-279.

[2] P. Bernstein, V. Hadzilacos, N. Goodman, «Concurrency Control and Recovery in Database Systems", Addison-Wesley Pub., ISBN 0-201-10715-5, 1987, 370 p.

[3] T. Chandra, S. Toueg, « Unreliable Failure Detectors for Reliable Distributed Systems », Journal of the ACM, vol. 43(2), March 1996, 225-267.

[4] J.-F. Hermant, G. Le Lann, "Asynchronous Uniform Consensus in Real-Time Distributed Systems", submitted for publication, March 2001.

[5] G. Le Lann, "Proof-Based System Engineering and Embedded Systems", Lecture Notes in Computer Science n° 1494, G. Rozenberg, F. Vaandrager, Eds., Springer-Verlag Pub., Oct. 1998, 208-248.

[6] N. Lynch, "Distributed Algorithms", Morgan Kaufmann Pub., ISBN 1-55860-348-4, 1996, 870 p.

[7] E. Bornschlegl, C. Guettier, J-C. Poncet, "Automatic Planning for Autonomous Spacecrafts Constellation", in proceedings of 2nd NASA Workshop on Planning & Scheduling for Space, San Francisco USA, March 16-18 2000.

[8] C. Guettier, J-C. Poncet, "Multi-Levels Planning for Spacecraft Autonomy", in proceedings of 6thInternational Symposium on Artificial Intelligence, Robotics and Automation in Space, Montreal Canada, June 18-21 2001.

[9] V. Saraswat, D. Bobrow and J. de Kleer, "Infrastructure for Model-based computing", Technical Report, Xerox PARC, CA, March 93.

[10]P. van Hentenryck, V. Saraswat and Y. Deville, "Design , Implementation and Evaluation of the Constraint Language CC(FD), In Constraint Programming: Basics and Trends", A. Podelski Ed., Chatillon sur Seine, Springer-Verlag LNCCS 910, pp. 68-90, 1995.

[11]J. Jaffar and J-L Lassez, Constraint Logic Programming, In Proc. Of the 14th ACM Symposium on Princples of Programming Languages, Munich, Germany, 1987.

[12]D.E. Bernard and al. "Design of the remote agent experiment for spacecraft autonomy", In Proc. Of the IEEE Aerospace Conference, 1998, ASPEN, CO.

[13]J.M.P. O'Hare, N.R. Jennings, "Foundations of Distributed Artificial Intelligence", 6th generation Computer Technology Series, ISBN 0-471-00675-0, 1996.