A NEW DESIGN APPROACH OF SOFTWARE ARCHITECTURE FOR AN AUTONOMOUS OBSERVATION SATELLITE

Jérôme Gout, Sara Fleury

LAAS-CNRS
7 Av. du Colonel Roche - 31077 Toulouse Cedex 4 - France
Jerome.Gout@laas.fr Sara.Fleury@laas.fr

Hervé Schindler

Matra Marconi Space
31 Av des Cosmonautes - 31402 Toulouse Cedex 4 - France
Herve.Schindler@tls.mms.fr

Abstract

The new generation of satellites are complex autonomous systems that present similarities with autonomous robots. This paper analyses a design approach and tools taken from robotic research. It focuses on the knowledge representations handled by the different architecture components and on the problems arising from the integration of the decision capacities (incremental reactive planner). Formal models of actions and tasks from which local representations of a varied nature are automatically derived, ensure the consistency of the data.

The approach has been evaluated using a real system specification and a complete architectural instance has been implemented from the low level real-time control routines to the hight level missions.

1 Introduction

The new generation of satellites have to fit constraints that have important repercussions on the design philosophy: lighter satellites, reduction of the design process duration, use of "standard" components, simplification of the on-site maintenance (in particular the heavy control from the central ground station), and so on. These constraints lead to satellite designs which manage both *reactive and decision capacities on-board* the system, that is *autonomous satellites.*

In this paper, we propose an approach along with tools to design software architectures which are inspired by work in autonomous robot. To illustrate and demonstrate the relevance of this approach, we have considered, in collaboration with Matra Marconi Space[1], the example of an autonomous observation satellite.

The main objective of these new satellites is to allow *direct access* to end-users using the World Wide Web through ground stations situated all around the world. Thus, unlike SPOT satellites for instance, it must offer high level interactions (e.g., "Take a photo of Noordwijk between 9AM and 10AM local time") and must be able to integrate all the client requests.

Consequently, the system must be able to manage the *planning* of the actions required to accomplish the missions (maneuvers, image processing, data down-loadings, ...) and their *execution control*, including failure recovery and redundancy management, *on-board.*

In order to integrate all these capacities we propose a generic software architecture structured in two main hierarchic levels (section 2). A lower functional level which embeds all the basic capabilities of the system (device control, servo-control, monitoring, etc) is controlled by an upper decision level that plans and controls the execution of the operations required to accomplish a mission.

We consider the elaboration of a real-time, modular and controllable functional level, using tools such as the Generator of Modules GenoM, a mastered operation and we therefor focus on the decisional level. Indeed, if the organization of this level is also well defined and if different tools exist to implement its components, the actual realization of such complex systems still raises important difficulties, the major ones being:

- *the knowledge representations:* the different components of the architecture have to handle and to share data of a varied nature (static models, dynamic state vectors, numerical/symbolical data,

[1]This collaboration has been supported by the Région Midi-Pyrénées (France) within the project SyDRE **Sy**stèmes **D**istribués **R**éactifs **E**mbarqués (On-Board Reactive Distributed Systems).

etc). To avoid redundancies and to ensure the consistency of the system we propose a unified knowledge representation associated with an automatic synthesis of the local models (section 3).

- *the master of algorithm complexity* which requires automatic synthesis based on validated models and the use of generic tools and control algorithms (section 4.2).
- *the integration of a reactive temporal planner* in the decision level raises problems related to *incremental planning* and to the synchronization between the future plan (elaborated from a predicted state and models of actions) and the ongoing execution (section 4.3).

A complete integration based on a simulated satellite will illustrate our approach (section 5).

2 Software Architecture Overview

In order to reconcile both decision and real-time capacities on board an autonomous robot, a generic software architecture composed of 2 hierarchic levels has been developed [1] :

- At the lower level a reactive distributed *functional level* embeds all the operational functions (control of devices, processing, ...).
- At the upper level, the *decisional level* decides which actions are to be executed according to the mission and the state of the system and controls their execution at the lower level.

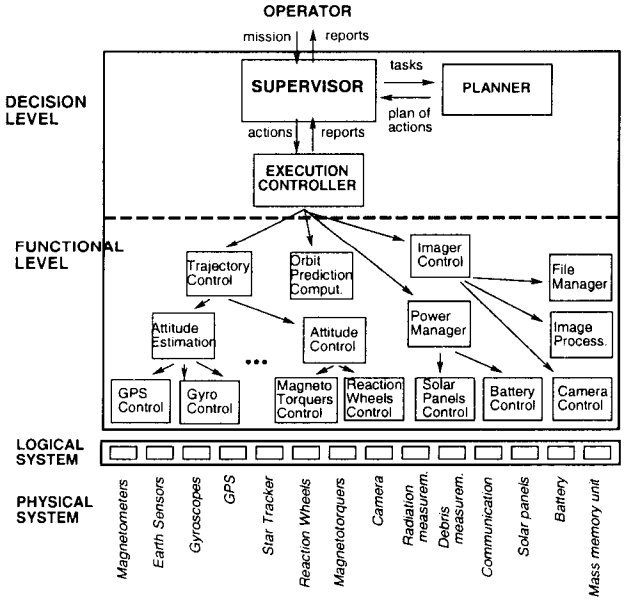

Figure 1: A generic architecture instantiated for an autonomous observation satellite. The functional level embeds about 20 modules generated by GenoM.

2.1 The Decision Level

The decisional part is responsible for mission management and for the control of the on-board system: it has to interpret the mission, to plan the adequate sequence of actions according to the current state of the system and to control on-board execution. It is composed of three entities: a supervisor, a planner and an execution controller. The planner is used as a resource by the supervisor which actually interacts with the next level, controls the execution of the plan and reacts to incoming events. An instance of this level will be presented in the following sections.

2.2 The Functional Level

The actions are executed at the functional level which embeds all the operation functions like the control of the various hardware devices (magneto-meters, earth sensors, gyroscope, gps, reaction wheels, ...), and also processing like orbit prediction or image processing. This level is organised as a network of modules: the functions are embedded in independent modules that have the responsibility of physical or logical resources.

The modules are capable of performing a number of specific services by processing inputs from, or acting on, physical robot devices and/or other modules. The services are parameterised and activated asynchronously through a non-blocking client/server protocol: a relevant *request*, that may include input parameters, applies to every service of each module. Thus requests start processings. The end of the service is marked by a reply returned to the client that includes an *execution report* and possibly data results.

For this application we have developed about 20 modules: one basic module for each hardware device (sensors, actuators, payloads, communication), one for the orbit prediction computation, and several to estimate and to servo-control the attitude using the previous basic ones (Figure 1).

Every module at the functional level is an instance of a generic model. They are automatically generated using the generator of modules GenoM which simplifies the design process and ensures a correct implementation (see [2]).

3 Knowledge Representations

A unified and consistent knowledge representation is fundamental to design and implement complex high level software architectures. Only a unified and formalized knowledge representation:

- ensures the consistency of the local representation of the different architectural components;
- allows the use of generic, and thus validated, architectural components;
- allows automatic code synthesis.

However if all the components of the architecture (i.e., the supervisor, the execution controller and the planner for the proposed architecture) *in fine* reason on, or handle, the same low level functional operators of the functional level, they do not consider the same properties of these operators.

Whereas the activities of the functional level are essentially characterized by numerical processing, the decisional level needs an abstract and symbolical representation (effects, conditions, resources used) to organize their execution.

Moreover, each component of this decision level requires specific knowledge:

- for planning purposes the planner needs to know their effects, their (pre-)conditions, their duration, their resource consumption, using or production, etc.;
- the supervisor that supervises the correct execution of the plan of actions and implements the failure recovery needs to know all potential malfunctions of every action.
- and finally the execution controller needs to know how to control (start, stop or parameterize) these actions at the lower level.

From these considerations we have elaborated *action models* that can be seen as an abstraction of module requests. Thus these actions modelise the low level operators and fill the gap between the functional and the decisional levels.

The hight level missions will be realized by combining these basic actions into *tasks*. The tasks are sequences of actions that allow us to predefine complex operators. They fill the gap between the hight level missions and the on-board capacities (the actions).

The actions represent the basic platform-dependent capacities of the system (control of hardware devices, servo-control, monitoring, filtering, etc), whereas the tasks represent complex application-depend functional capacities.

Action and task representations must be formal to allow automatic synthesis and reasoning both for the planning and the supervising processes.

3.1 Action Representation

The decisional level has to decide which of the operational functions (ie, of the requests) are to be executed at the lower level. However, it can not reason directly on the request descriptions provided by GenoM. These descriptions are only functional and do not integrate information related to their conditions or effects on the state vector of the agent. Moreover this data cannot be added to the module description as they depend on the application context (eg: according to the situation, satellite maneuvers can be allowed or not during image acquisitions).

Thus, from a bottom-up view point, the requests have been enriched with the actions: actions are extensions of module requests with semantic information.

The description of an action is basically composed of two parts :

- structural and functional information (name, requests involved, all possible termination status, ...) to control the execution of the action when required;
- resource and logical information (effects on the resources, conditions and effects on agent state) which allow reasoning about the usage and consequences of the action.

Actions are defined as a list of (`attribute: value`) couples containing executive service name (`service`), the non-nominal possible terminations (`end_slots`), resource usage/production and consumption (`uses, consumes, effects`); the conditions and effects specifications (`assertions, effects`).

The following example is the action `CAMERA` that allows to take an image:

```
action CAMERA {
  service      : take_image;
  concurrence  : interrupt;
  end_slots    : cam_hard_failed, cam_soft_failed;
  uses         : camera(1);
  consumes     : power(20)@start,
                 mmu(100)@start;
  produces     : image(1)@ok;
  assertions   : on_zone() = ?zone in [start,ok];
  effects      : image(?zone) = taken @ok;
};
```

The actions are characterized by the following properties:

- An action starts on the controllable[2] event `start/-`.
- The end of an action is associated with the terminal contingent event[3] `-/end`.
- The terminal event is always associated with a report that characterizes how the action has ended (the default report is `ok`).
- An action may be interrupted by the controllable event `kill/-`.
- An action may produce intermediate contingent events. The default intermediate event `-/started` confirms its starting.

Actions are the smallest entity handled at the upper levels of the architecture. The decisional level can

[2]This event is controllable from the decisional level view point. The controllable events are noted `evt/-`.

[3]This event is contingent from the decisional level view point. The contingent events are noted `-/evt`.

act on the system only using the controllable events `start/-` and `kill/-`, and the evolution of the system is perceptible (measurable) only through the incoming contingent event `-/started` and `-/end` (associated with their terminal reports). The state vector of the system is the integration of all these events.

From the textual description of an action a graphical representation can be derived (Figure 2). This representation will be used to define tasks.

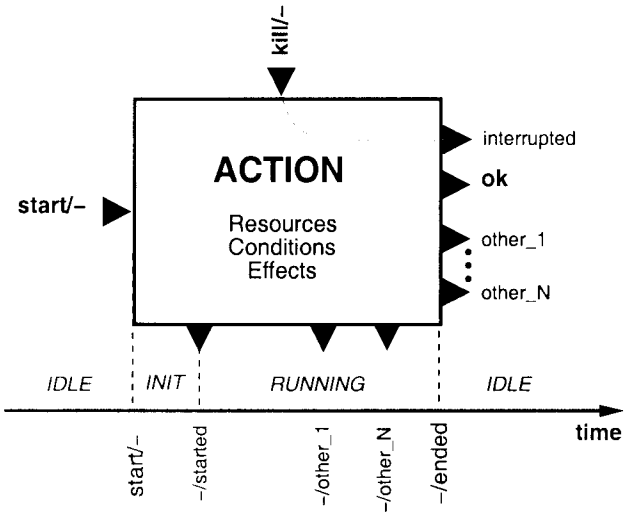

Figure 2: *Graphic and synthetic view of action. The time runs from left to right. The incoming black arrows (or slots) receive the* `start/-` *(on the left) and the* `kill /-` *(on the top) events. The outgoing ones export the events produced by the action during its execution (on the bottom) or at its end (on the right). The end slots on the right are exclusive and allow expression of conditional tasks according to termination reports.*

3.2 Task Representation

The task model is defined as a complex combination of actions, expressing control and ordering information of actions. Formally, a task can be defined as $\{\{\mathcal{A}_i\}, \mathcal{R}\}$, $\{\mathcal{A}_i\}$ being a set of actions and $\mathcal{R}$ being a partial order relation between these actions.

The tasks are designed by the operator using an intuitive graphical tool called `TaskBuilder`. Within this environment, the actions are composed using their graphical representation: "contingent" slots (ie, intermediate or terminal slots) are linked to "controllable" ones (ie, start or kill slots). In such a way, one can express part of known plans or skeletons (partial graphs) of actions that represent complex satellite processes, including failure detections and recovery actions (using the different termination slots).

Figure 3 presents a simple example of a task with three different actions. The nominal process of this task involves two actions: `CAMERA` to take an image and `DOWN_LOAD` to down-load it to a ground station. In case of camera failure the `CHECK_UP` action is invoked.

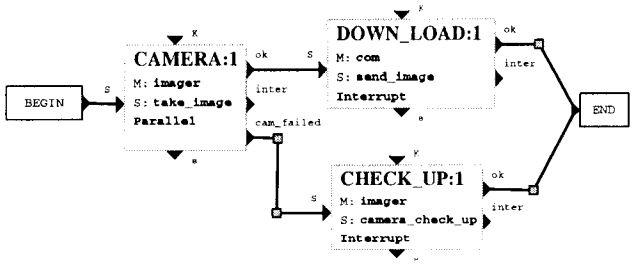

Figure 3: *An example of task composed of three actions. According to the report of the* `CAMERA` *action, the system down-loads the image or checks the equipment.*

3.3 Automatic synthesis of the models

The action and task formal models have been designed to unify the representations handled in the architecture. From the action and task descriptions `TaskBuilder` *automatically produces* (Figure 4):

- the upper execution controller procedures that allow execution of the different actions by sending the adequate module requests;
- the planning operators for the planner that are elaborated from both the resource and logical information on the actions involved in the task, and the partial order relation between these actions;
- the supervision procedures elaborated from the relations between the all the events and their termination or intermediate reports.

Because of the limitations of planning algorithm (*i.e.* no handling of conditional plans) the planning operators derived from the complete task model, contain only the subset of nominal actions and their relations. The supervision procedures integrate the complete description including nominal and non-nominal events and actions. The contingent events that are not explicitly considered in the task description, implicitly aim to a failure state and a survival mode of the satellite if they occur during the task execution.

4 Decision Level Integration

4.1 The execution controller

The execution controller, or executive, interfaces the functional and the decision levels. It is a purely reactive system without reasoning. It controls the module requests according to the "controllable" events coming from the supervisor, and returns "contingent" events from the execution reports. An action can involve several module requests coordinated by the executive.

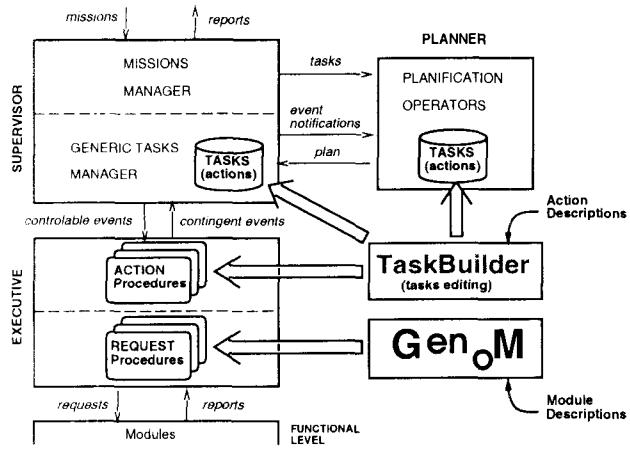

Figure 4: *Overview of the origin and the distribution of the knowledge representations (in bold) and the dynamically exchange data (in italics) in the architecture.*

The executive maintains a state vector of the functional level and manages the conflicts between module requests.

The executive is written with PRS and its procedures have been entirely produced by the couple of systems: `TaskBuilder` and **GenoM** (Figure 4).

4.2 The Supervisor

As presented in section 2, the supervisor is actually the conductor of the decisional level. It is in relation with:

- the users, through mission requests;
- the planner, through planning requests and the returned solution plans;
- the execution controller, through action events.

The supervision system is divided into two different subsystems (Figure 1). The upper one deals with the clients' requests. It receives mission observations, processes them taking into account client priority and flight over target area. According to these data the supervision sends a planning request to the planner. This part is application dependent as it contains all the application specificities due to the interface between users and the autonomous satellite.

The lower subsystem is in charge of the execution and the supervision of the plan produced by the planner. Unlike the upper one, this subsystem is *generic*. Following the dynamic task plan and the task models produced by `TaskBuilder`, it starts actions by sending the adequate events to the executive and integrates returned feedbacks to make the plan progress and to control its execution.

The plan maintained by the supervisor is composed of the 3 following parts:

- Static description of task produced from the formal model by `TaskBuilder`.
- Numerical temporal data: the planner returns a temporal window for each event corresponding to an action event contained in the newly inserted tasks of the plan
- Symbolic temporal relations between events of different tasks.

The two last parts of the plan are dynamically produced by the planner in response to planning requests.

The supervision of the plan is achieved by analysing received feedbacks from the executive. According to the execution report of the actions, the plan progresses following nominal or non-nominal branches of the complete static description. A fatal error occurs when there are no action operations (starting or interruption) associated with a received execution report. The satellite switches automatically to a survival mode, closing all client connections and waiting for the maintainer's intervention.

Our supervision system is implemented using a tool called PRS (for Procedural Reasoning System) ([3, 4]).

4.3 Planner

The asynchronous arrival of numerous client requests, their strong temporal constraints (ie, the communication or image acquisition temporal windows), the inaccuracy of the orbit prediction in long term, the important resource constraints (eg, shared hardware devices, images storage capacities before their download, limited energy capacities between two battery rechargings with the solar panels, etc) call for an efficient *incremental temporal planner.*

This incremental planning has been elaborated upon IxTeT, a temporal planner developed in our research group ([5]).

A temporal planner. IxTeT is a general and open planner (see [6] for an adapted planner). Its formalism is based on a reified temporal logic that defines several temporal predicates on both state and resource attributes. State attributes are handled with the `event` predicate to express a change in the world, whereas the persistence can be expressed by the `hold` predicate. Concerning resource management, one can express resource usage during a determined time interval (`use` predicate) or resource production (`produce`) and consumption (`consume`). IxTeT's algorithms are sound and complete. Until now IxTeT has not been integrated with a supervisor as an incremental reactive planner.

An incremental planner. The incremental planner, based on an IxTeT's kernel, has to maintain a global historical plan of all missions sent by the clients, updating it for each new request.

Thus, the planner is a plan server running concurrently to the supervisor: the supervisor sends a plan-

ning request to the planner and get back a new temporal plan. This dynamic link between the supervisor and planner builds the planning problem online, using the planning operators synthesized by `TaskBuilder` from the task descriptions.

A reactive planner. Another problem to integrate the planner/supervisor couple is related to the synchronisation of the plans. In order to predict the future state of the system, the planner maintains a global plan elaborated from the task and action *models* that will quickly diverge from the real executed plan without synchronisation. Thus, several synchronisation operators have been added allowing the supervisor to update the planner plan:

- the planner gets the real execution date of all events as they occur,
- the resource and logic states predicted by the planner are updated by the supervisor according to the *real* feedback of the actions,
- the planner can retract tasks from its plan in case of failures or mission abortion.

In return, the planner informs the supervisor about a new plan insertion resulting from a new planning request. A translation between the planning representation and the supervision one is necessary, including completion of a nominal plan by failure recovery actions. Therefore, the supervisor plan model is dynamically updated.

Finally, the planning time process must be bounded to ensure the global system dynamics.

5 Application to an Autonomous Satellite

The presented methodology and tools to design the software architecture of autonomous systems have been evaluated on a future autonomous observation satellite project.

To run the whole system, we have implemented a simulator to emulate the physical system: the earth rotation, the orbital motion of the satellite (including noises), the energy consumption of the hardware devices, the occurrence of failures, and so on.

An example of a user mission is presented bellow. It corresponds to an image acquisition request with parameters including: the target area (`Toulouse`), some constraints for the image acquisition (local time interval, maximum inclination, type of camera ...) and on-board image processing, the client identification (in particular to know where to down-load the image) and the request priority (in case of resource saturation).

```
(IMAGE-RQST (ZONE TOULOUSE 43.62 1.45 30
                    12:00 13:00)
            (IMAGE HIGH NONE NONE (..))
            878 12)}
```

Once selected by the supervisor, this client request is translated to a planning request and sent to the planner.

This mission uses the task `take_image` which contains 4 main steps involving 8 basic actions:

1. satellite orientation (`SLEW` action)
2. image acquisition (`CAMERA`, `ZONE_IN` and `ZONE_OUT` actions)
3. data processing (`IMAGE_PROC` action)
4. processed image down-loading to the client's ground station (`DOWNLOAD`, `ZONE_IN` and `ZONE_OUT` actions).

`ZONE_IN` and `ZONE_OUT` are monitoring actions that allow detection of the entrance or the exit of the satellite over a given area (for photographic or for communication purpose): a contingent event is returned to the decisional level once the monitored condition is satisfied.

Note that a single action may involve several module requests at the functional level. For instance, the `SLEW` action requires several sensors and actuators to estimate and control the attitude of the satellite.

The average time taken by the planner to find the solution plan is about 2 seconds[4]. The resultant plan is presented in figure 5. In this example the request has been integrated in a plan that already contained 2 previous user requests. Only three planning operators have been used here (the two first operations of the `take_image` task have been gathered) `TAKE_IMAGE`, `DSP`, `DOWN_LOAD`.

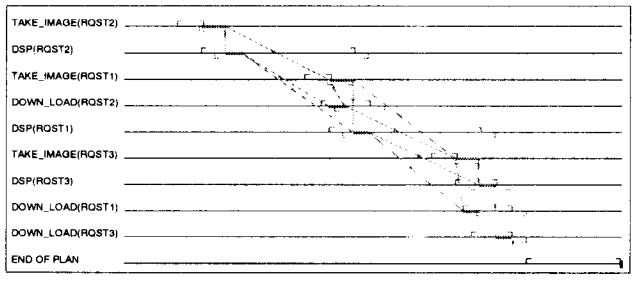

Figure 5: *An example of temporal plan. The operators (listed on the left) of three different tasks (*`RQST1`, `RQST2`, `RQST3`*) are intertwined. The arrows represent their precedence constraints.*

[4]Let us recall that the time allocated to the planner to find out a solution is bounded. The request is put back to the mission queue if no solution is found.

The planner translates the plan as a chronology of events for the supervisor. The following PRS facts give an illustration of that data. The first group shows symbolic temporal constraints between events belonging to different tasks.

```
(EPP 1 (EVENT TI END CAMERA 1) 1 (EVENT TI END ORBCP_OUT 1))
(EPP 1 (EVENT TI END CAMERA 1) 2 (EVENT TI START SLEW 1))
(EPP 1 (EVENT TI END ORBCP_IN 1) 1 (EVENT TI START ORBCP_OUT 1))
```

The next 3 facts express the temporal window of each event of the plan.

```
(ETW 1 (EVENT TI START ORBCP_IN 1) 300 3889 642)
(ETW 1 (EVENT TI START ORBCP_OUT 1) 3600 4189 -1)
(ETW 1 (EVENT TI START SLEW 1) 0 3589 235)
```

And finally, to complete this data dynamically produced by the planner, the supervisor uses PRS procedures, synthesized by `TaskBuilder` from the static description of the tasks (pair of condition/action-like rules).

```
(ETP (.(EVENT TI END CAMERA 1) ok .)(EVENT TI KILL ORBCP_OUT 1))
(ETP (.(EVENT TI END ORBCP_OUT 1) inter .)(EVENT TI START DSP 1))
(ETP (.(EVENT TI END DSP 1) ok .)(END))
```

Figure 6 is a screen copy of an experiment session. One can distinguish the supervisor (top left), the execution controller (top right), the user console (bottom left), the planner (bottom right) and the simulator in the center showing the satellite and the current area being flown over.

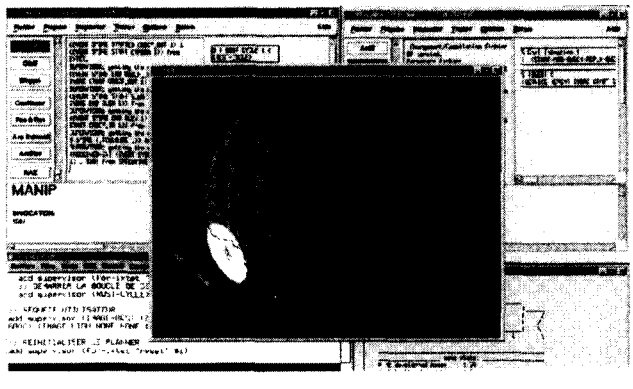

Figure 6: *A snapshot of the experiment running*

6 Conclusion

The objective of this study is to analyze and demonstrate concepts and tools taken from research in robotics to design the new generation of autonomous satellites. Formal hierarchic models of knowledge representations (*i.e.* **action** and **task**) have been proposed and allowed to:

- eliminate the description redundancies,
- automatically produce the derived instances handled by the different architecture components,
- ensure the consistency between model instances.

Concerning the deliberative processes, we have implemented the integration of planning within the dynamic loop of execution/supervision. This includes an incremental planning based on IxTeT and an extension of the planning capabilities (necessity to extend the task insertion control).

Actions and tasks describe both nominal and non-nominal situations that are managed by the supervisor according to the feedback received from the functional level.

The executive and the lower part of the supervisor are generic and handle models automatically synthesized by `GenoM` and `TaskBuilder`. This simple the integration procedure masters the complexity of the system.

The approach has been evaluated using a real specification, and a complete architectural instance has been implemented from the low level real-time control routines to the highest level missions.

References

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *Special Issue of the International Journal of Robotics Research on Integrated Architectures for Robot Control and Programming*, 17(4):315–337, April 1998. Rapport LAAS N97352, Septembre 1997, 46p.

[2] S. Fleury, M. Herrb, and R. Chatila. Genom: A tool for the specification and the implementation of operating modules in a distributed robot architecture. In *International Conference on Intelligent Robots and Systems (IROS'97), Grenoble, France*, 1997.

[3] F. Ingrand, M. Georgeff, and A. Rao. An architecture for real-time reasoning and system control. *IEEE Expert, Knowledge-Based Diagnosis in Process Engineering*, 7(6):p. 33–44, December 1992.

[4] F. Ingrand, R. Chatila, R. Alami, and F. Robert. Prs: A high level supervision and control language for autonomous mobile robots. In *IEEE Int. Conf. on Robotics and Automation (ICRA'96)*, Minneapolis (USA), June 1996.

[5] M. Ghallab and H. Philippe. A compiler for real-time Knowledge-based Systems. In *International Workshop on Artificial Intelligence for Industrial Applications,Hitachi City, Japan*, May 1988.

[6] P. Dago. *Extention d'algorithmes dans le cadre des problèmes de satisfaction de contraintes valuées: application à l'ordonnancement de systèmes satellitaires.* PhD thesis, Ecole National Supérieure de l'Aéronautique et de l'Espace, France, 1997.