

A generic modular architecture for the control of an autonomous spacecraft

G rard Verfaillie

ONERA, Toulouse, France
Gerard.Verfaillie@onera.fr

Marie-Claire Charmeau

CNES, Toulouse, France
Marie-Claire.Charmeau@cnes.fr

Abstract

In this paper, we present the result of the work that has been done in the CNES-ONERA-LAAS AGATA project on a generic modular architecture for the control of an autonomous spacecraft. The main features of the proposed architecture are (1) its genericity which goes far beyond the space domain, (2) its adequacy for closed loop control of any (sub)system (state and objective tracking, decision making, decision execution, and supervision), (3) its ability to take into account requirements for quality and reactivity of the control, (4) its modularity, (5) the encapsulation of data and control it guarantees inside each module. After describing the proposed architecture, we discuss its impact on deliberative tasks such as on-line decision making.

The AGATA project

The AGATA project (Autonomy Generic Architecture: Tests and Applications) brings engineers and researchers from CNES¹, ONERA², and LAAS-CNRS³ together around the global objective of increasing spacecraft autonomy (Charmeau & Bensana 2005).

One may want to increase spacecraft autonomy to reduce the need for a permanent control of spacecraft by human operators on the ground. For that, one tries for example to replace the orbital manoeuvres which are still often decided, computed, and controlled from the ground by orbital manoeuvres which can be decided, computed, and controlled autonomously on-board according to the current state of the spacecraft and of its mission and according to the current deviation from the reference orbit (Lamy *et al.* 2004). One tries also to equip the spacecraft with autonomous capabilities for failure detection and identification and for system reconfiguration, in order to avoid, in case of failure, sending for a ground control center which is not always reachable

¹CNES: Centre National d'Etudes Spatiales, French Space Agency, www.cnes.fr

²ONERA: Office National d'Etudes et de Recherches A rospatiales, French Research Agency in Aeronautics and Space, www.onera.fr

³LAAS-CNRS: Laboratoire d'Analyse et d'Architecture des Syst mes du Centre National de la Recherche Scientifique, Laboratory for Analysis and Architecture of Systems of the French Research Center, www.laas.fr

and available and thus to improve on the system dependability.

One may want also to increase spacecraft autonomy to improve on the quantity and quality of the collected data and to speed up data acquisition and delivery. For example, an Earth surveillance or observation satellite is most of the time out of visibility of its control center. If it is able to detect in front of it ground phenomena such as forest fires or volcanic eruptions, it will be able to decide autonomously to perform finer observations of the detected phenomena and to manage conflicts between candidate observations, even when it is out of visibility of its control center (Damiani, Verfaillie, & Charmeau 2005b). If it is able to detect in front of it the actual cloud coverage, it will be able to perform optical observations on only cloud free ground areas and to avoid performing useless observations on cloudy areas (Lachiver *et al.* 2001). If it is able to roughly analyze observation data after acquisition, it will be able to avoid storing and downloading poor quality data and thus to save mass memory space and downloading time (Chien *et al.* 2005).

What is true for satellites orbiting at low altitude around the Earth is still truer for spacecraft exploring planets or comets (ESA-Mars-Express ; ESA-Rosetta) (either mobile or immobile, either in orbit around the planet or the comet or on its surface). To visibility problems, problems resulting from communication delay and rate must be added. Equipping the spacecraft with autonomous decision-making capabilities for system reconfiguration in case of failure or for management of resources such as energy and memory may avoid time consuming communication loops between the spacecraft and the ground and thus improve on the global data acquisition capacity over the whole mission period.

To that, the new requirements coming from formation flying mission projects must be added: sets of spacecraft orbiting in close formation with very strong relative position and orientation constraints to perform for example interferometric observations of the Earth or of the space (Chien *et al.* 2001; ESA-Darwin). For such missions, it may be hoped that formation configuration keeping and changes be performed autonomously by the whole spacecraft formation.

The AGATA project started in 2005 for an expected duration of 3 years. Its aim is the design, implementation, and experiment of a ground simulator of an autonomous spacecraft, which can show that available techniques allow a

spacecraft to be autonomously and correctly controlled. One of the first tasks of the project was the definition of a generic architecture for the control of an autonomous spacecraft, on which one can rely for the development of more specialized components such as state tracking or decision making.

This paper is a synthesis of the current result reached by the group of the AGATA project working on the architecture definition. This group benefited from inputs from various people from CNES, ONERA, and LAAS-CNRS and especially from Michel Lemaître from ONERA and Félix Ingrand from LAAS-CNRS.

Section 2 (*Expected qualities of a control architecture*) lists the qualities that are expected from a control architecture of an autonomous spacecraft. Section 3 (*Overview of existing architectures*) is a non exhaustive survey of classical architectures in the space, aeronautics, and robotics domains. Section 4 (*A generic control module*) presents the schema of a generic control module, along with its various components. Section 5 (*A decomposition example*) shows an example of decomposition of the control architecture of an Earth surveillance and observation satellite into specialized control modules, and Section 6 (*Two module examples*) provides the reader with a more precise description of two of these modules. Section 7 (*Impact on decisional tasks*) discusses the impact of such an architecture on deliberative tasks such as state tracking and decision making. Finally, Section 8 (*Conclusion*) concludes with the work that remains to be done.

Expected qualities of a control architecture

In order to make a sensible choice in terms of architecture, we listed the qualities that we expect *a priori* from the control architecture of an autonomous spacecraft. The result is the following one:

1. *genericity*, that is independence (1) from the type of mission (observation, surveillance, exploration . . .), (2) from the system or sub-system to be controlled (detection, observation, data memorization, data downloading . . .), (3) from the kind of task to be controlled (either physical task, or information processing task), (4) from the kind of control to be exerted (selection and scheduling of tasks to be executed, decomposition of these tasks into elementary tasks, decision upon task starting times, upon used resources, or upon other execution parameters . . .), and (5) from the way this control is exerted (predefined program, execution of a predefined plan with or without on-line replanning capabilities, decision rules, execution of a predefined policy, on-line decision-making . . .);
2. *adequacy for close loop control*: this implies the architecture be built from the four basic components of any closed-loop control system: system state tracking, objective tracking, decision making, and decision execution;
3. *ability to take into account quality and reactivity requirements*: usually, the control of a system must satisfy as well as possible requirements in terms of quality (good quality decisions with regard to the current mission objectives) and reactivity (decisions delivered in due time);

the proposed architecture must allow these possibly contradictory requirements and the trade-off between them to be taken into account;

4. *modularity*: this means that, for the sake of complexity limitation and of control reactivity, one wishes to decompose the control of the whole system into autonomous communicating control modules, as far as independence between sub-systems permits it; with such a decomposition, each module is responsible for a sub-system, that is for a subset of the system features (which are usually modelled as state variables or time-lines), as well in terms of tracking as in terms of control; for example, the orbit control module of a spacecraft is responsible for the tracking of all the features of the orbital position and for their control in case of deviation from the reference orbit; moreover, two modules cannot be both responsible for a same system feature;
5. *encapsulation of data and control*, a usual requirement in all the object-based or agent-based frameworks: this means that, given a decomposition into various control modules, no information about the state of a feature f can be obtained without an information request to the module m that is responsible for f ; such a requirement guarantees the consistency of the views that various modules in the architecture may have of f at a given time; the same way, this means that no control on the state of f can be exerted without a control request to m ; such a requirement forbids unauthorized possibly contradictory attempts to control f coming from various modules in the architecture and guarantees that possible conflicts between control requests be managed by m ;
6. *reflexivity*: the control tasks are specific, but they are also information processing tasks; they may take time; this may be the case of some state tracking or decision making tasks; they must consequently themselves be controlled, what is usually performed by a control supervision task;
7. *simplicity*: the proposed architecture must be based on mere principles and concepts, easily explainable and understandable;
8. *absence of useless restrictions*: it must finally impose no useless restriction in terms of approaches, formalisms, methods and algorithms possibly used in the architecture, especially for state tracking and decision making.

Overview of existing architectures

Among the architectures used for the control of autonomous engines in the space, aeronautics, and robotics domains, one can first distinguish between centralized and decentralized (modular) ones.

Among the first ones (centralized), one finds the most classical ones in the robotics domain (LAAS architecture (Alami *et al.* 1998), *Remote Agent* architecture (Muscatella *et al.* 1998) . . .), which are built on three levels: a first *deliberative* level in charge of mission tracking and of generation and maintenance of activity plans, a second *reactive* level in charge of execution triggering and

tracking according to the activity plans, and a third *functional* level which encapsulates all the basic functionalities of the physical system (sensors and actuators). In these architectures, planning is the core of the control system and nothing is *a priori* executed on the physical system via the functional level unless the deliberative level planned it.

Still among the first ones (centralized), one finds less classical ones, which are built around a pre-programmed *supervisor* (Barrouil & Lemaire 1999), in charge of execution triggering and tracking of all the possible tasks: physical ones or information processing ones, such as planning for example. In these architectures, supervision is the core of the control system.

Among the second ones (decentralized), one finds various attempts to decompose the whole control into various control modules. This is the case with the GENOM architecture (Fleury, Herbb, & Chatila 1994), which allows the whole functional level to be implemented in a modular and homogeneous way. This is also the case with the IDEA architecture (Muscettola *et al.* 2002), which proposes a unique schema for any decision module at any level, with the following components in any module:

1. *Model*, which involves the models associated with all the actions that can be activated by the module;
2. *Plan Database*, which involves all the information about past, present, and future actions and states, including the current goals and plan;
3. *Plan Runner*, which can be seen as a module supervisor;
4. *Reactive Planner*, which is in charge of planning, performed regularly within a limited time over a limited temporal horizon ahead.

It can be observed that, except the *Remote Agent* architecture (Muscettola *et al.* 1998), these architectures do not set an explicit place for system and environment state tracking. It can be observed too that, except the IDEA architecture (Muscettola *et al.* 2002) and some extensions of the LAAS architecture (Lemai & Ingrand 2004), they do not light the way for the connexion between deliberative tasks, such as state tracking and decision making, whose duration may be not well controlled, and reactive tasks, whose duration can be controlled. In most of the cases, the reactivity of the whole control system, including deliberative tasks, remains an open question.

The architecture we propose is inspired from GENOM and IDEA, because, as both are, it is modular. It is also inspired from the *Remote Agent* one, because it is built on the same four basic components: state tracking, objective tracking, decision making, and decision execution. With regard to IDEA to which it is the closest one, it proposes a different finer organization of each module, which fits better, we hope, the main requirements in terms of system control. It proposes also a generic schema for exchanges between modules, based on information and control requests.

A generic control module

Figure 1 shows the schema of a generic control module (we will refer to as *M*), itself organized into generic compo-

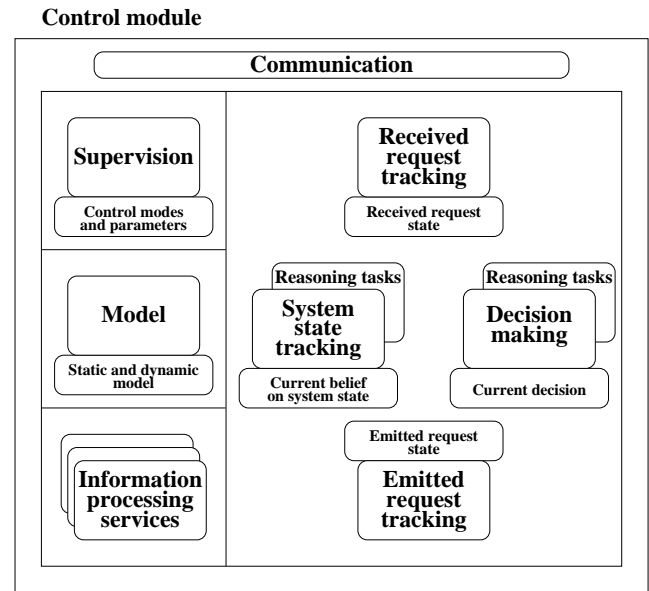


Figure 1: Schema of a generic control module.

nents. The core of this module is a set of four components, dedicated to closed-loop control: *received request tracking*, *system state tracking*, *decision making*, *emitted request tracking* components.

The *received request tracking* component is in charge of the receipt, of the pre-processing, and of the tracking of the control requests emitted by other modules to the *M* module. It maintains a *current state of the received requests*.

The *emitted request tracking* component is the emission counterpart of the previous component. It is in charge of the pre-processing, of the emission, and of the tracking of the control requests emitted by the *M* module to other modules, to physical systems, or to information processing services. It maintains a *current state of the emitted requests*.

Among the control requests, some of them may be imperative, for example a request of activation of an instrument. But, some of them may not be, for example an observation request emitted to the module responsible for observation management. In the latter case, they are often associated with something that indicates their importance: weight, priority ... Generally, requests are also associated with information about their validity period, for example immediately and for the time being, from a time to another time, or any-time between a time and another time. Their content is also highly variable, for example to perform this action, to get in this mode, to get this state, or to meet these constraints. Some of them are ordinary control requests, handled by the *received request tracking* component. Other ones are special requests aiming at changing the control mode of the module itself (a form of meta-control). They are in this case handled by the *supervision* component.

The *system state tracking* component is in charge of the tracking of the state of the features for which the *M* module is responsible, that is in charge of the maintenance of a belief

about the state of these features. This belief can take various forms: a specific state without any uncertainty, a set of possible states, a distribution of probability, possibility, or any form of uncertainty over the set of possible states, a total or partial order on the the set of possible states . . . The maintenance of this belief uses information coming from physical systems and from the *system state tracking* components of other modules. It may call for complex computations whose duration may be not well controlled, but we can assume to have an anytime behavior: result available any time, at least after a minimum time, result quality increasing with time. It maintains a *current belief about the state of the system*.

The *decision making* component is in charge of deciding on the emission of control requests to other modules, physical systems, or information processing services. It answers the question: which requests at which time? To answer this question, it uses information coming from other components of the *M* module: current state of the received requests, current state of the emitted requests, current belief about the state of the system. It uses also information coming from physical systems and from the *system state tracking* components of other modules. As the *system state tracking* component does, it may call for complex computations whose duration may be not well controlled, but we can assume to have an anytime behavior: result available any time, at least after a minimum time, result quality increasing with time. It maintains a *current decision on request emission*.

On top of these four basic components, a *supervision* component controls the module itself, what we could refer to as a *meta-control*: initialisation, surveillance and control of the various components in the module, via a pre-determined set of modes and of control parameters which set the activation rate of the various components.

The *model* component is optional. If it exists, it gathers all the static and dynamic characteristics of the features of the system for which the *M* module is responsible: for example, for an orbit control module, the minimum and maximum power of the thrusters or the set of equations that constrain the orbital position and attitude at the beginning of the thrust, the thrust direction, the thrust duration, the thrust power, and the impact on orbit and attitude.

The *information processing service* component is optional too. If it exists, it gathers all the information processing services associated with the *M* module: for example, for an orbit control module, the services dedicated to the anticipated computing of orbital positions, eclipse or visibility windows.

The *communication* component is dedicated to exchanges with other control modules, physical systems, or information processing services: information or control *requests*, *information* about changes in system state or request status, *alerts*, *data* about current system state or request status.

Figure 2 focuses on the flow of requests and information about request status and system state between modules in the architecture. One can observe a top-down flow of control requests. After reception and preprocessing by the *received request tracking* component, control requests are decomposed into more elementary ones by the *decision making* component, which sends the result of decomposition to the *emitted request tracking* component. In the opposite bottom-up di-

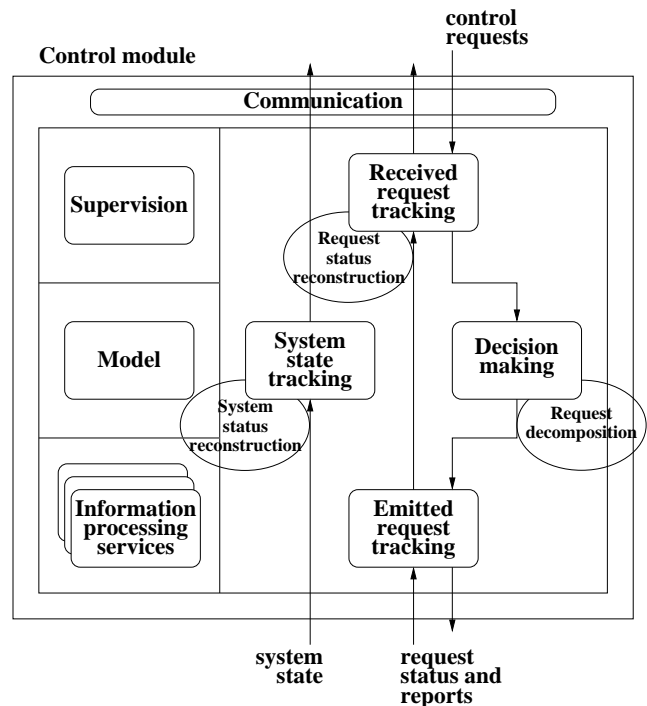


Figure 2: Flow of requests and of information about request status and system state.

rection, status of emitted requests and reports on them, received by the *emitted request tracking* component, allow the *received request tracking* component to reconstruct the status of the received requests and to build reports on them. In the same bottom-up direction, information about the system state received from other modules allows the *system state tracking* component to build a belief about the *system state*, limited to the features the *M* module is responsible for.

A decomposition example

Figure 3 shows an example of decomposition into control modules of the control of a Earth surveillance and observation satellite (Damiani, Verfaillie, & Charneau 2005b). The main tasks this satellite must perform are (1) detection of ground phenomena such as for example forest fires or volcanic eruptions via a wide swath detection instrument in front of the satellite, (2) analysis of detection data in order to distinguish for example between forest fires and volcanic eruptions, (3) observation of selected ground areas via a narrow swath observation instrument and a mobile sight mirror, (4) compression, memorization, and down-loading of observation data, (5) communication with the ground to send alarms after ground phenomenon detection or to receive requests for the tracking of specific ground areas, and finally (6) keeping the satellite in a geocentric attitude on its reference orbit.

One finds a natural hierarchical decomposition with (1) at the top level a *satellite* module responsible for everything must be coordinated at the satellite level, (2) at the second

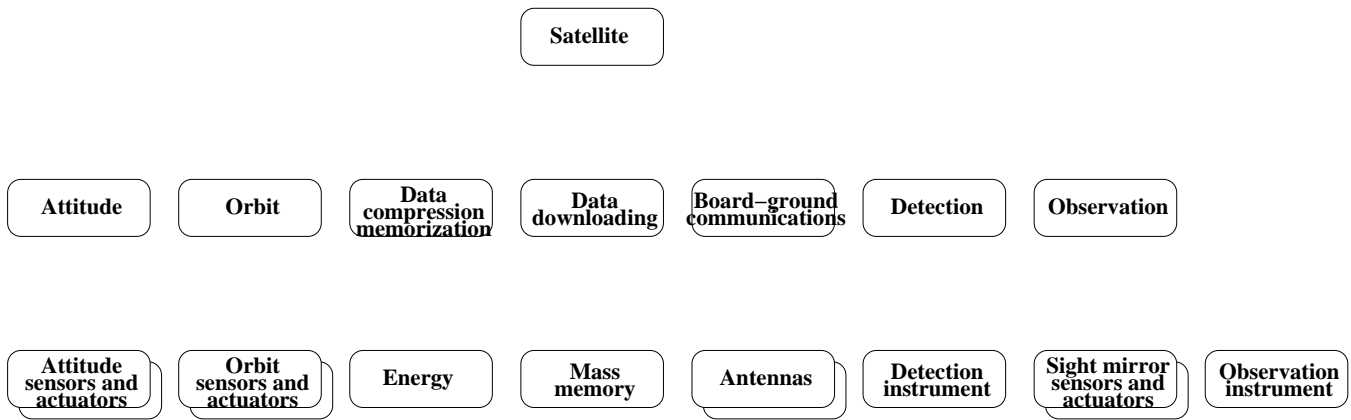


Figure 3: Example of a modular architecture for the control of an Earth surveillance and observation satellite.

level the main functionalities the satellite must perform (detection, observation ...), and (3) at the third level, all the physical functionalities the satellite can provide (detection instrument, sight mirror, observation instrument ... that is what is usually referred to as the *functional* level). It must be stressed that this is only an example of decomposition and that many other decompositions could be considered on the same satellite.

Figure 4 adds to the previous figure information about the control requests that can be sent from a module to another one. One can observe that, according to the hierarchical decomposition, control requests are sent from a module at a given level to a module at a lower or equal level. For example, the *observation* module, responsible for the management of observations, can send control requests to the *sight mirror*, *observation instrument*, and *data compression and memorization* modules. One can also observe that some modules may receive control requests from various other modules. This is for example the case with the *mass memory* module which may receive requests from the *data compression and memorization* and *data down-loading* modules. In such a case, possible conflicts (leading for example to a memory overflow) must be managed, either by the module that receives requests (in this case, the *mass memory* module, *a posteriori* control), or by another module on top of the modules that emit requests (in this case, the *satellite* module, *a priori* control). One can finally observe that the *energy* module receives no request. This results from the assumption that, in this satellite, energy is not controlled, but only consumed by the other equipments. In such conditions, the *energy* module aims only at maintaining the current level of available energy, as well as possible forecast. The same way as with the *mass memory* module, possible conflicts about the use of energy must be managed on top of the modules that require it.

Figure 5 adds to Figure 3 information about the exchanges of data between modules about the system state. One can observe that, still according to the hierarchical decomposition, data goes from a module at a given level to a module at a higher or equal level. For example, the *observation*

module, responsible for the management of observations, receives data from the *orbit*, *energy*, *memory*, *sight mirror*, and *observation instrument* modules, in order to decide upon observations on the basis of the current and foreseen state of the orbital position, of the energy, of the memory, of the sight mirror, and of the observation instrument. It should be noted that such a representation makes no assumption about the way data exchanges between modules are physically performed.

Two module examples

We present in this section two illustrative examples of control modules, still associated with an Earth surveillance and observation satellite.

The first one (Figure 6) is an example of module at the functional level (the *sight mirror* module) which encapsulates a sensor (which measures the current mirror orientation) and an actuator (a motor which allows the mirror orientation to be changed).

The second one (Figure 7) is an example of module at a higher level (the *observation* module) which is responsible for the management of observations and can send control requests to and receive information from the previous one as well as to and from other modules.

About the *sight mirror* module, a very mere one, one can observe that the *system state tracking* component produces a belief about the current orientation of the mirror as a function of a measure of this orientation and of information about the motor state. Then, when a mirror orientation request is received, the *decision making* component uses this belief and the result of the computation of the necessary mirror movement to build a mirror movement request it sends to the physical system.

About the *observation* module, a more complex one, one can first observe that it involves no own *system state tracking* component. In fact, the *observation* module does not need to produce a belief about any system feature. It uses the beliefs produced by other modules: the *orbit*, *energy*, *memory*, *sight mirror*, and *observation instrument* ones which provide it with information about the current and foreseen orbital

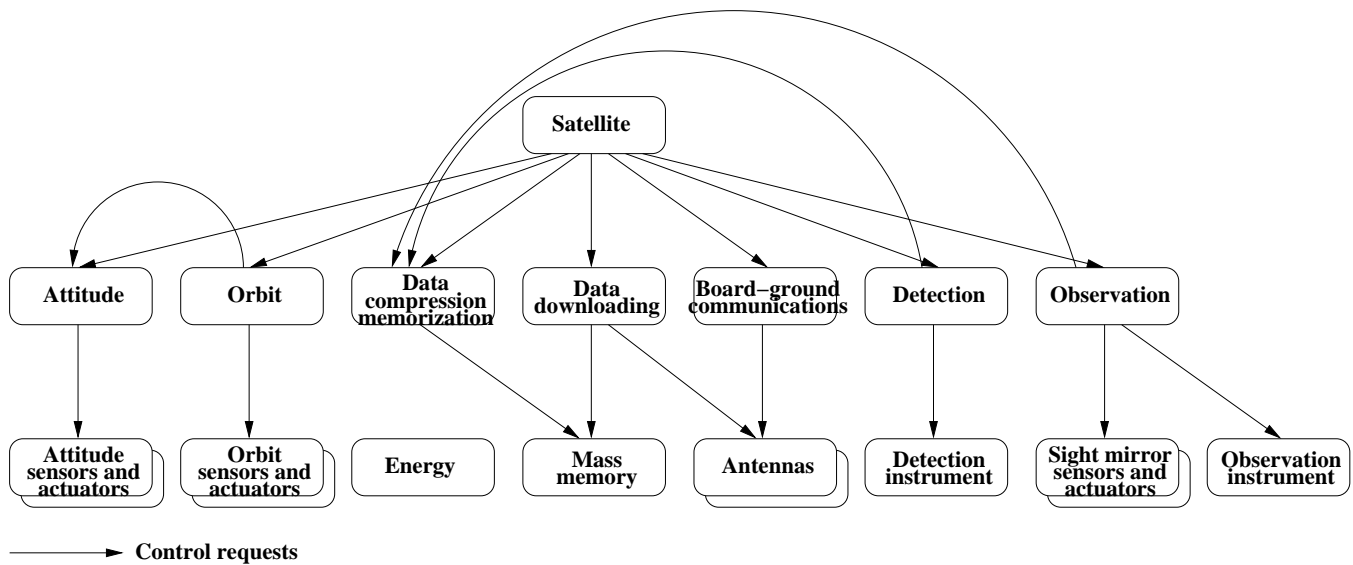


Figure 4: Control requests from module to module.

position, energy and memory levels, sight mirror and observation instrument states. Then, the *decision making* component uses this information and the observation requests it has received to decide upon the observations it performs. For that, it can use any technique that can provide it with a *a priori* good decisions: hand-written decision rules, pre-computed policy, on-line planning, on-line decision-making ... See for example (Damiani, Verfaillie, & Charneau 2005a). Finally, it transforms decision into coordinated control requests to the *sight mirror*, *observation instrument*, and *data compression and memorization* modules.

Impact on decisional tasks

Among the expected qualities of a control architecture, control reactivity is one of the most important ones. One wants not only that decisions made in each module be of good quality, that they lead the system in the right direction, towards satisfaction of the user objectives. One wants also that these decisions be produced in due time. Let us assume for example that the *decision making* component of the *observation* module sends to the *emitted request tracking* component of the same module a request for an observation o , but that it does that after the satellite on its orbit flew over the ground area associated with o . Even if this decision may be of good quality, delivering it too late is of no practical use.

This requirement in terms of reactivity has a strong impact on the deliberative tasks in each module, such as state tracking or decision making. Roughly speaking, an *any-time* behavior (Zilberstein 1996) is wished for. This implies that a result be available any time, at least after a minimum computing time, and that result quality globally increases with time. But, at least with regard to decision making, it is necessary to go further and to take into account possible changes in system state and mission objectives, as well

as uncertainty about these changes. In a surveillance and observation mission for example, new ground phenomena may appear, new observation requests can be emitted by the ground, planned observations may fail ... To face such a situation with changes and uncertainties about changes, various significantly different approaches have been proposed.

Planning and scheduling people (Ghallab, Nau, & Traverso 2004; Baptiste, Pape, & Nuijten 2001) propose to build plans that take or do not take into account uncertainty about the way system state and objectives may evolve (classical, robust, flexible, or conditional plans ...). In case of a change which invalidates the current plan, the classical approach consists in building a new plan, often by trying and repairing the previous one, with a first objective of limiting as much as possible the time used to produce a new plan, and sometimes a second objective of modifying as less as possible the previous plan. Such an approach has four main drawbacks: (1) the question of the choice of the temporal horizon on which to build a plan (the planning horizon) remains often without any sensible answer; (2) even if negative changes (for example, an equipment failure) are taken into account, the positive ones (for example, a resource level higher than expected) are rarely exploited; (3) in a highly uncertain situation, replanning may become very frequent and one can wonder about the usefulness of building plans that are immediately invalidated by the execution; (4) nothing can guarantee that building a new plan by repairing a previous one can take a time that is compatible with the system reactivity requirements, especially when objectives must absolutely be reached; in the worst case, plan repairing complexity is similar to plan building complexity (from scratch).

In another direction, *Sequential decision making* people (Puterman 1994; Sutton & Barto 1998) propose to build off-line (manually or automatically, by using either models, or simulations and learning) policies, that are functions that

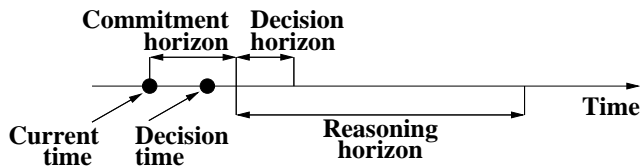


Figure 8: Commitment, decision, and reasoning horizons.

often merged or hidden:

1. the *commitment* horizon, on which decisions have been made and corresponding actions have been possibly already triggered off and on which no change in the decisions is possible or desirable; except in special situations, such as execution failure, one does not wish for example to stop an observation that has been already triggered off; thus, one refrains from deciding any new observation that would start before its end;
2. the *decision* horizon, on which one wishes to make effective decisions; this may be for example only the next observation; this may be also the next k ones, or the temporal horizon until the end of the current satellite revolution or until the next communication window with a control center; by definition, *commitment* and *decision* horizons have an empty intersection;
3. the *reasoning* horizon, on which one reasons to make the best possible decisions on the chosen *decision* horizon; this may be for example the next k observations, or the temporal horizon until the next data down-loading window, or until the next satellite day period; by definition, the *reasoning* horizon includes the *decision* one.

Figure 8 shows this various horizons. It should be noted that, although time is one of the main parameters, these horizons can be defined by parameters other than time. For example, the *reasoning* horizon may be limited to the next k observations of priority greater than a given threshold, if one wants to reason only on the most important observations.

Roughly speaking, the problem is, before the end of the *commitment* horizon, to make the best possible decisions on the *decision* horizon, by using the reasoning performed on the *reasoning* horizon. With classical planning and scheduling, decision and reasoning horizons are often merged. In sequential decision making, the *decision* horizon is limited to the next decision and *reasoning* horizon is far larger, finite or not.

Distinguishing between these various horizons paves the way for numerous trade-offs that can take into account the dynamics of the state of the system and of the objectives, the uncertainty about this dynamics, the reactivity requirements, the complexity of the decision problem, and the available computing resources.

For example, if the dynamics of the state of the system and of the objectives are high and the uncertainty about these dynamics is important, one will be inclined to make decisions as late as possible on as small as possible *decision* horizons to benefit from the most up-to-date information and to avoid committing on too large *commitment* horizons. Moreover, if

reactivity requirements are strong and computing resources are limited, one will be inclined to limit the size of the *reasoning* horizon (trade-off favouring reactivity at the expense of quality). In fact, one will be able to adapt the size of the *reasoning* horizon to the time that is actually available for reasoning. One can find in (Damiani, Verfaillie, & Charmeau 2005b) an example of application of such an approach to the problem of management (on-board, on-line, and *anytime*) of the observations to be performed by an autonomous surveillance and observation satellite.

Conclusion

We have presented the main features of a generic modular architecture for the closed-loop control of an autonomous spacecraft and more widely of any engine or system. We concluded by showing that it is possible to work out the apparent contradiction between classical approaches based on *planning-replanning* and on *sequential decision making*, via the notions of *commitment*, *decision*, and *reasoning* horizons.

The next steps of this work will consist in formalizing, implementing, and validating this generic modular architecture by using frameworks, languages, and tools such as *UML* (Fowler 2004), *synchronous languages* (Benveniste *et al.* 2003), or *AltaRica* (Arnold *et al.* 1999). For that, we are currently studying a generic well-defined way of connecting in each module reactive components such as the four basic components (*received request tracking*, *system state tracking*, *decision making*, *emitted request tracking*) with deliberative components such as *planning* over large temporal horizons or *diagnosis* from observations collected over large temporal horizons, that are called by the *system state tracking* and *decision making* reactive components. Following the ideas developed in the CIRCA architecture (Musliner, Duffee, & Shin 1993), the objective is to get a control system which offers strong guarantees in terms of quality and reactivity of the response at a basic control level (for example, to avoid failure states), but offers only guarantees in terms of reactivity, not in terms of quality, at the higher mission level, where the goal is to optimize the mission return.

After that, our plans are to instantiate this modular architecture on the reference missions of the AGATA project (satellite constellation for Earth surveillance and observation, agile Earth observation satellite, orbiter for planet exploration, satellite formation for scientific observation) in order to assess its ability to correctly structure the control of various spacecraft in the context of various missions.

References

- Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An Architecture for Autonomy. *The International Journal of Robotics Research* 17(4):315–337.
- Arnold, A.; Point, G.; Griffault, A.; and Rauzy, A. 1999. The AltaRica Formalism for Describing Concurrent Systems. *Fundamenta Informaticae* 40(2-3):109–124.
- Baptiste, P.; Pape, C. L.; and Nuijten, W. 2001. *Constraint-based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers.

- Barrouil, C., and Lemaire, J. 1999. Advanced Real-time Mission Management for an AUV. In *Proc. of the SCI NATO Symposium on Advanced Mission Management and System Integration Technologies for Improved Tactical Operations*.
- Benveniste, A.; Caspi, P.; Edwards, S.; Halbwachs, N.; Guernic, P. L.; and de Simone, R. 2003. The Synchronous Languages Twelve Years Later. *Proc. of the IEEE* 91(1):64–83.
- Charmeau, M.-C., and Bensana, E. 2005. AGATA: A Lab Bench Project for Spacecraft Autonomy. In *Proc. of the 8th International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS-05)*.
- Chien, S.; R.Sherwood; Burl, M.; Knight, R.; Rabideau, G.; Engelhardt, B.; Davies, A.; Zatocha, P.; Wainwright, R.; Kuplar, P.; Cappelaere, P.; Surka, D.; Williams, B.; Greeley, R.; Baker, V.; and Doan, J. 2001. The Techsat-21 Autonomous Spacecraft Constellation. In *Proc. of the 6th International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS-01)*.
- Chien, S.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davies, A.; Mandl, D.; Frye, S.; Trout, B.; Shulman, S.; and Boyer, D. 2005. Using Autonomy Flight Software to Improve Science Return on Earth Observing One. *Journal of Aerospace Computing, Information, and Communication*.
- Damiani, S.; Verfaillie, G.; and Charmeau, M.-C. 2005a. A Continuous Anytime Planning Module for an Autonomous Earth Watching Satellite. In *Proc. of the ICAPS-05 Workshop on Planning under Uncertainty for Autonomous Systems*.
- Damiani, S.; Verfaillie, G.; and Charmeau, M.-C. 2005b. Cooperating On-board and On the ground Decision Modules for the Management of an Earth Watching Constellation. In *Proc. of the 8th International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS-05)*.
- ESA-Darwin. Darwin mission. <http://sci.esa.int/darwin/>.
- ESA-Mars-Express. Mars Express mission. <http://sci.esa.int/marsexpress/>.
- ESA-Rosetta. Rosetta mission. <http://sci.esa.int/rosetta/>.
- Fleury, S.; Herbb, M.; and Chatila, R. 1994. Design of a modular architecture for autonomous robot. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA-94)*.
- Fowler, M. 2004. *UML distilled: a brief guide to the standard object modelling language*. Addison Wesley.
- Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.
- Lachiver, J.; Laherrère, J.; Sebbag, I.; Bataille, N.; and Bret-Dibat, T. 2001. System Feasibility of Onboard Clouds Detection and Observations Scheduling. In *Proc. of the 6th International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS-01)*.
- Lamy, A.; Charmeau, M.-C.; Laurichesse, D.; Grondin, M.; and Bertrand, R. 2004. Experiment of Autonomous Orbit Control on the DEMETER Satellite. In *Proc. of the 18th International Symposium on Space Flight Dynamics (ISSFD-04)*.
- Lemai, S., and Ingrand, F. 2004. Interleaving Temporal Planning and Execution in Robotics Domains. In *Proc. of the 19th National Conference on Artificial Intelligence (AAAI-04)*.
- Muscettola, N.; Nayak, P.; Pell, B.; and Williams, B. 1998. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence* 103(1-2):5–48.
- Muscettola, N.; Dorais, G.; Fry, C.; Levinson, R.; and Plaunt, C. 2002. IDEA: Planning at the Core of Autonomous Reactive Agents. In *Proc. of the 3rd NASA International Workshop on Planning and Scheduling for Space*.
- Musliner, D.; Durfee, E.; and Shin, K. 1993. CIRCA: A Cooperative Intelligent Real-time Control Architecture. *IEEE Transactions on Systems, Man, and Cybernetics* 23(6):1561–1574.
- Puterman, M. 1994. *Markov Decision Processes, Discrete Stochastic Dynamic Programming*. John Wiley & Sons.
- Sutton, R., and Barto, A. 1998. *Reinforcement Learning*. MIT Press.
- Zilberstein, S. 1996. Using Anytime Algorithms in Intelligent Systems. *AI Magazine* 17(3):73–83.