An Autonomous Mission Controller for Earth Observing Satellites

Cédric Pralet, Charles Lesire, Jean Jaubert

An Autonomous Mission Controller for Earth Observing Satellites

Cédric Pralet and **Charles Lesire**
ONERA, Université de Toulouse
2 av. Edouard Belin, BP 74025
F-31055 Toulouse Cedex 4, France

Jean Jaubert
CNES
18 av. Edouard Belin
31401 Toulouse Cedex 9, France

Abstract

This paper presents an autonomous controller developed for managing the activities of a new generation of Earth Observing Satellites (EOSs). This controller uses a hierarchy of reactors as in previously existing architectures, and it exploits specific reasoning procedures at the level of each reactor to get fast deliberations on-board. It is able to take into account the arrival of urgent acquisition requests, late cloud predictions, and information about the real volume of data, while meeting several operational requirements from the end-users.

1 Introduction

The mission of commercial Earth Observing Satellites (EOSs) consists in making observations of some areas at the Earth surface and in downloading collected data to ground reception stations. For most of these satellites, mission plans are first built on the ground at the level of the mission center, and then sent to the satellite by ground control stations. These plans contain sequences of time-tagged telecommands that must be executed while the satellite is orbiting around the Earth. Nowadays, the search for performance leads to the development of a new generation of commercial EOSs. As in the seminal NASA contributions on *Deep Space 1* (Muscetolla et al. 1998) and *Earth Observing 1* (Chien et al. 2004), the objective is to get autonomous satellites that quickly adapt their behavior depending on the conditions encountered at execution time, instead of waiting for plans sent by the ground only a few times a day.

In this paper, we consider three situations in which adapting plans directly on board is relevant. First, many images realized by EOSs cannot be exploited due to the *presence of clouds* over observed areas. To improve on this point, we consider a scenario where the satellite regularly receives meteorological predictions from ground stations or from geostationary relay satellites. These predictions consume a low level of uplink bandwidth and can be received much more frequently than time-tagged plans. Second, due to on-board data compression, the *exact volume of observation data* is not known beforehand, especially due to the uncertainty about the presence of clouds on some parts of the images. Our goal is then to deliver data more quickly to the end-users by exploiting the real volumes of data files, instead of executing a timed-tagged download plan built on the ground based on conservative maximum volume assumptions for all images. Third, we consider scenarios in which *urgent acquisition requests* can be synthesized by the satellite following on-board detections, or received from the ground or from relay satellites using a low level of uplink bandwidth. The goal for the satellite is then to be reactive and to autonomously integrate the realization of these requests in the current plan, possibly by canceling low priority operations.

In this paper, we present techniques used for answering these needs for a new generation of autonomous EOSs. The main contribution is an instantiation of an architecture composed of a hierarchy of modules which use heterogeneous procedures to deliberate and which manage static memory allocation issues and action interruptibility in a clean way. We do not consider the functional layer of the satellite, real-time execution aspects, or fault detection, identification and recovery. Instead, we focus on the on-board deliberation capabilities and on the interaction with execution.

The paper is organized as follows. Sect. 2 recalls related works, Sect. 3 motivates our design choices, Sect. 4 describes the problem to solve, Sect. 5 gives the architecture of the EOS mission controller, Sect. 6 details the plan adaptation procedures, Sect. 7 details how planning and execution are interleaved, and Sect. 8 describes our first experiments.

2 Related Works

The development of autonomous mission controllers is not a new topic in the space domain. Almost all systems developed in this context use so-called *timeline-based reasoning*, which consists in using *timelines* for representing the evolution of relevant state attributes of the system and its environment, such as the states of instruments or the amount of resources available. Each timeline is composed of a sequence of *tokens*, which correspond to temporal intervals over which a state attribute takes a given value.

In the space domain, several operational autonomous mission controllers were already deployed:

- the seminal *Remote Agent* (Muscetolla et al. 1998; Jónsson et al. 2000) which managed to take the full control of the NASA DS-1 probe launched in 1998; it used constraint-based reasoning and flaw resolution techniques for progressively building temporally flexible plans satisfying all constraints required between tokens;

- CASPER (Chien et al. 1999b; 1999a; 2014; Knight et al. 2001; Estlin et al. 2000), which was successfully used for NASA satellites EO-1 launched in 2000 and IPEX launched in 2013; for IPEX, CASPER reacted autonomously, during at least one year of operations, to the real on-board resource usages, to the real durations of activities, to the real data volumes, and to events detected on images; to do this, CASPER is called regularly (using a continuous planning strategy) and iteratively repairs conflicts on the current plan; it does not manipulate temporally flexible plans, but it contains repair procedures which can move activities; CASPER also exploits hierarchical decompositions of activities to perform short-term planning for low-level activities and long-term planning for high-level ones;
- the VAMOS experiment of the DLR BIROS satellite launched in 2016 (Wörle and Lenzen 2013); given a baseline ground plan represented as a set of successive timeline blocks, VAMOS is able to autonomously activate *timeline extensions* on-board depending on the amount of resources available at execution time and to react to event detection by quickly generating new timeline extensions from a catalog of predefined extensions; to make these operations very quickly, VAMOS uses decision rules based on resource thresholds precomputed on the ground;
- the three ESA PROBA satellites, launched in 2001, 2009, and 2013; for instance, PROBA-1 is able to realize autonomous maneuvers for realizing observations on areas whose coordinates are provided by the ground, and PROBA-V uses on-board decision rules to activate the acquisition flow for imaging lands (Ilsen et al. 2014);
- the autonomous controllers used for the NASA Martian rovers; for these rovers, mission plans are built on the ground and on-board reasoning is used for autonomous navigation tasks, for opportunistic science in the middle of navigation tasks, and for coping with activities whose duration is longer than expected based on some clean up operations if the real cumulated duration of a block of activities is longer than a given margin (Gaines et al. 2016).

Besides these deployed systems, several other experimental autonomous controllers have been tested.

- As in the Remote Agent and IxTeT-eXeC (Lemai and Ingrand 2004), some experimental controllers execute *temporally flexible plans*. This is the case for the GOAC architecture (Fratini et al. 2011; Cesta et al. 2012) and for the SanchoExpress on-board executive (Victoria et al. 2015). Both of them manipulate plans defined as timelines where transition times between tokens are subject to minimum and maximum temporal distance constraints. Simple temporal network reasoning is then used to propagate the earliest possible start time of each transition and to detect potential conflicts. Other experimental controllers manipulate temporally flexible plans defined as partial order schedules (Policella et al. 2004): see (Pralet et al. 2014) for an application to flexible download plans for EOSs and (Chi et al. 2018) for an application to the flexible execution of the Mars 2020 rover activities.
- As in CASPER, other prototypes use *iterative plan repair*, but with a specific effort to use predefined intelligent repair rules to decrease the computational effort. This is the case for MEXEC developed for the future Europa Clipper mission near Jupiter (Verma et al. 2017), and for the TVCR system studied for ExoMars (Woods et al. 2006).
- As in VAMOS, other approaches combine ground planning and on-board reasoning by using, at the level of the flight software, a set of *resource thresholds computed on the ground* to decide whether additional activities can be activated online (Maillard et al. 2015; 2016). In the space domain, only a few work has been realized on plans containing explicit conditional branches (Washington, Golden, and Bresina 2000).
- Some other autonomous controllers follow a different approach, where execution is interleaved with planning based on a *fast greedy search* that has a reduced worst-case time complexity (Khatib et al. 2013; Chien and Troesch 2015; Pouly, Jouanneau, and Olhagaray 2014; Rabideau and Benowitz 2017; Pralet, Infantes, and Verfaillie 2013; Rabideau, Chien, and Laren 2009). Such greedy searches manipulate consistent plans and use an iterative *selection-insertion* mechanism. At each step, the latter selects one candidate activity a and tries to insert a into the plan. If the insertion succeeds, a new current plan is obtained, otherwise a is rejected and the next activity is considered. The contributions that follow this approach differ in the way the next activity is selected and in the way insertions are made. For replanning in case of addition of a new goal, some of these approaches restart from an empty plan (Chien and Troesch 2015), while others use incremental processing, for instance by restarting from a plan where only the lowest priority activities are removed (Rabideau, Chien, and Laren 2009).

In another direction, the definition of autonomous controllers in the space domain requires the definition of a robotic architecture composed of several modules. Some approaches use a three-tier architecture involving a deliberative layer, an executive layer, and a functional layer. Other approaches are inspired by the work on IDEA (Muscettola et al. 2002) and T-REX (McGann et al. 2008), which allow for a more flexible interleaving between planning and execution. Basically, a T-REX agent controls a set of control loops called *reactors*. The latter manipulate timelines, and to progressively build a globally consistent plan they exchange goals defined as desired future timeline values. At each deliberation cycle, a T-REX agent tries to synchronize the view of the timelines among the reactors, dispatches goals between the reactors, and finally allocates some time to each reactor to deliberate. In the space domain, the T-REX approach was used for the *APSI deliberative reactor* of the GOAC architecture developed for ESA, where the APSI framework is used to deliberate (Fratini et al. 2011; Cesta et al. 2012). More recently, it was also used in ERGO (Ocón et al. 2018), where a PDDL planner is used by a mission reactor, and where the autonomy level can range from reactive to deliberative behaviors.

3 Towards an Autonomous EOS Controller

To deal with our autonomous EOS scenario, we developed a timeline-based mission controller whose main design choices are given below.

First, we use simple decision rules rather than black-box multi-criteria planning procedures. One reason for this is that the mission controller must bring some guarantees on how preferences of the end-users are taken into account. For instance, high priority requests must always be preferred to low priority ones, the geocentric pointing (pointing in the direction of the Earth center) must be preferred when there are sufficiently long periods without observation, and the devices (the mass memory, the acquisition instrument, and the emission antenna) must be switched off when possible but not too often for long-term reliability issues. Also, the mission controller is an embedded software which must not use dynamic memory allocation. This means that the used memory must be reserved at the initialization step, and not during the execution of the decision procedures. This is why we do not use black-box planners which do not offer such a guarantee.

Second, we use the T-REX approach that decomposes the mission manager into several reactors. This allows to have several smaller subproblems for which specific reasoning procedures can be used instead of a global problem which is harder to solve. This is particularly useful because EOSs have low computational capabilities (typically 100 times slower that a modern laptop), and the time available to make decisions is limited to avoid missing acquisition and download opportunities while the satellite is moving on its orbit. The T-REX approach also leads to a modular architecture where reactors can be more easily changed, reused, and validated. We also make some efforts to get incremental deliberation procedures that quickly deal with goal updates and avoid replanning from scratch.

Third, in our architecture, a reactor can manipulate flexible plans internally but we do not have a global simple temporal network representing the set of all feasible plans. One reason for this choice is that in the EOS scenario considered, postponing an activity is not just a matter of temporal constraints, essentially due to *complex state constraints*. One of them is related to the kinematic features of agile satellites, which use gyroscopic actuators to move around their gravity center along the three axes (roll, pitch, and yaw). Also, acquisition and download activities can be performed in parallel but there is a coupling between the pointing of the satellite used for observing and the pointing required for downloading data to a given ground station (download is not possible if the angle between the current satellite pointing and the satellite-station vector is too high). To take these state constraints into account, we fix the timings of acquisition activities when building the download plan.

Fourth, as in CASPER, we manipulate both long-term high-level plans (coarse acquisition and download activities) and short-term low-level plans (detailing when the devices must be switched on and off). To get an explicit relationship between high-level activities and low-level ones, we exploit *hierarchical decompositions* of goal activities.

Fifth, to perform planning and execution in parallel, we make a clear distinction between the activities whose execution is committed and the activities whose presence in the plan can still be changed if needed, and a clear distinction between the activities that can be interrupted and those which cannot, as in IxTeT-eXeC (Lemai and Ingrand 2004).

4 Planning Problem Description

At each step, the autonomous controller must consider:

- a set of observation requests $\mathcal{R}$; each request $r \in \mathcal{R}$ has a priority, a duration, and a set of possible realization time windows; each request $r \in \mathcal{R}$ also has an estimated data volume (exact volume if data has already been acquired and compressed) and a boolean cloud cover prediction for each possible realization window;
- the kinematic capabilities of the satellite; to simplify the presentation, we only consider maneuvers in terms of roll angles (pointing of the satellite to the right or to the left of its ground trace); we use a simple model where the minimum transition duration between two successive observation requests r_1, r_2 is obtained from the difference between the roll angle $RollEnd_{r_1}$ obtained at the end of r_1 and the roll angle $RollStart_{r_2}$ required at the start of r_2, divided by a mean rotation speed around the roll axis;[1]
- a set of ground reception stations $\mathcal{S}$, with for each station $s \in \mathcal{S}$ a set of time windows during which downloading data is possible; observing and downloading is parallel is possible, but only if the pointing used for observing is not too far (in terms of angular distance along the roll axis) from the direction of the satellite-station vector;
- the features of the devices (the observation instrument, the mass memory, and the emission antenna), including the fixed durations required to switch on and off each device.

Fig. 1 shows, at two different steps of the execution, the kind of plans obtained from the autonomous EOS controller presented in this paper. In Fig. 1, the current execution time is represented by the red vertical dashed line, and the current start time of the planning horizon is represented by the blue vertical dashed line. Several timelines are used:

- **pointing**, which represents the evolution of the pointing of the satellite; possible tokens are $\texttt{MAN}(\rho, \rho')$ (maneuver from roll angle ρ to roll angle ρ', in magenta), GEO (geocentric pointing, in blue), and $\texttt{ACQ}(\rho, \rho')$ (pointing for realizing an acquisition whose start and end roll angles are ρ and ρ' respectively, in red);
- **memWrite**, which represents the unique identifier of the file in which acquisition data are being written; possible tokens are $\texttt{WRITE}(\texttt{f})$ (when file number f is written, in red) and IDLE (when no file is written, in gray);
- **memRead**, which represents the unique identifier of the file from which downloaded data is being read; possible tokens are $\texttt{READ}(\texttt{f})$ (when file number f is read, in orange) and IDLE (when no file is read, in gray);

[1] The approach can be extended to deal with fully agile satellites which can move around the three axis (roll, pitch and yaw), with time-dependent effects in terms of minimum transition durations.

- **instrState**, **memoryState**, and **antennaState**, which represent the evolution of the state of each device; for these timelines, the possible tokens are `ON_A` (ON state used for supporting acquisition and download activities, in green), `ON_M` (ON state maintained between two activities, in green too; contrarily to `ON_A`, the `ON_M` token can be interrupted if needed), `OFF` (OFF state of the device, in gray), `OFF_TO_ON` (warming of the device, in yellow), and `ON_TO_OFF` (ON to OFF transition, in yellow too).

Fig. 1 also shows the visibility windows available over three stations. Globally, each plan contains (1) a sequence of acquisitions interleaved with maneuvers and geocentric pointings, (2) download activities realized during station visibility windows, (3) operations over the devices to have the mass memory and the instrument ready during acquisitions, and to have the mass memory and the antenna ready during download activities. It is also possible to see that multiple planning horizons are used, with a long-term horizon for the global acquisition and download plans, and a short-term horizon for the detailed management of the devices.

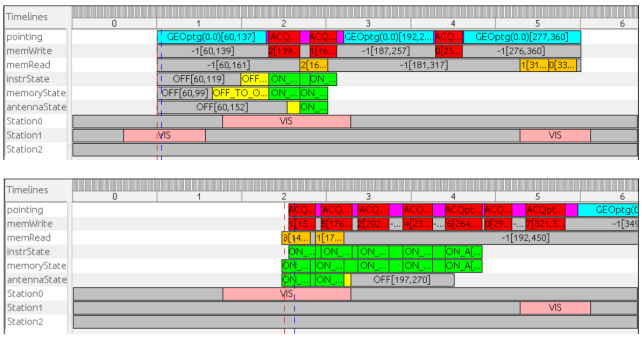

Figure 1: Timelines used by the mission controller

5 Reactor-based Decisional Architecture

To build the plans presented before, the EOS autonomous controller uses the acyclic architecture provided in Fig. 2, which is composed of 10 reactors. To transform high-level goals into fine-grain plans, the reactors exchange goal requests (solid lines) and information on the system state (dashed lines). A reactor can also cancel a goal previously sent to another reactor.

At the highest level of the hierarchy, the *Mission Reactor* is responsible for managing the arrival of new observation requests. For each new request received, the *Mission Reactor* sends one acquisition goal to the *Acquisition Reactor* and one download goal to the *Download Reactor*.

At the lowest level, the *Pointing Monitor*, the *Instrument Monitor*, the *Memory Monitor*, and the *Antenna Monitor* are respectively responsible for managing the commands concerning the pointing of the satellite, the observation instrument, the mass memory, and the data emission antenna. These low-level reactors directly interact with the functional layer of the EOS. They receive time-tagged commands and they simply order these commands chronologically to send them to the devices when needed.

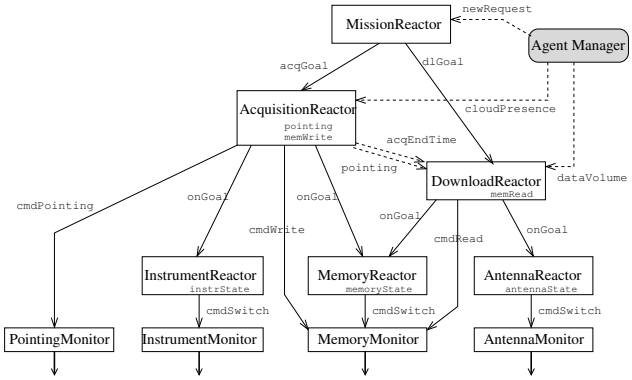

Figure 2: Reactor-based architecture

The *Acquisition Reactor* is responsible for producing a high-level acquisition plan from the set of candidate acquisition goals it receives and from updated cloud cover predictions over the areas to be observed. It sends "ON goals" to the *Memory* and *Instrument Reactors*, to request the mass memory and the observation instrument to be switched on during observations. It also sends pointing commands to the *Pointing Monitor*, and write commands to the *Memory Monitor* (command to write the data flow generated by the observation instrument during a fixed time window).

The *Download Reactor* is responsible for producing a download plan from the set of candidate download goals it receives, from all inputs on the real data volume after compression, from the end dates of acquisitions (to ensure that for each request, data download is realized after data acquisition), and from the pointing of the satellite synthesized by the *Acquisition Reactor*. Such a pointing can indeed limit download opportunities over a given station. The *Download Reactor* sends ON goals to the *Memory* and *Antenna Reactors*, to request that the mass memory and the antenna must be switched on for downloading data.

The *Instrument Reactor*, the *Memory Reactor*, and the *Antenna Reactor* are respectively responsible for managing the ON/OFF state of the observation instrument, the mass memory, and the emission antenna. To achieve the ON goals received, they compute switch-on/switch-off commands that are then sent to the device monitors. These reactors are instantiations of a generic *Device Reactor*.

As in T-REX, each timeline is owned by a unique reactor. For example, as shown in Fig. 2, the **memoryState** timeline is owned by the *Memory Reactor*. The reactors interact through goal transmissions and read operations over external timelines (*e.g.*, the *Download Reactor* listens to updates on the pointing managed by the *Acquisition Reactor*).

Goal Decompositions The decomposition of high-level goals into subgoals is shown in Fig. 3. The decomposition also associates with each goal g the set of all possible tokens that might be useful over timelines to achieve g. All goals and their tokens are created at the initialization of the flight software, to avoid dynamic memory allocation. Fig. 3 shows the 6 types of goals manipulated:

- the *request* goal, which covers one acquisition goal and one download goal;
- the acquisition goal (*acqGoal*), which covers:
  - tokens `IDLE` and `WRITE` over the **memWrite** timeline; the first one is used to model an idle state of the data writing process between the end of the previous acquisition and the start of the acquisition associated with the goal; the second one is used to describe that data is written in a file during the acquisition;
  - commands used to start and end the recording of the acquisition flow (*cmdStartWrite* and *cmdEndWrite*);
  - ON goals over the mass memory and the instrument (*memOnGoal* and *instrOnGoal*);
  - pointing commands, including *cmdManeuverToGeo* used for reaching a geocentric pointing between the last acquisition realized and the acquisition considered (command used only if there is a sufficient time gap between the two acquisitions), *cmdManeuverToAcq* used for realizing a maneuver just before the acquisition considered, and *cmdScanAcq* used for scanning a ground area; four tokens are associated with the **pointing** timeline to describe the evolution of the pointing from these commands;
- the download goal (*dlGoal*), whose decomposition is similar to the decomposition of an acquisition goal; the main difference is that it does not involve elements related to the pointing of the satellite;
- the ON goals over the three kinds of devices (*instrOnGoal*, *memOnGoal*, *antennaOnGoal*); each ON goal has fixed start and end times; it covers tokens and commands that can be used to enforce the goal (see the algorithms described in the following section).

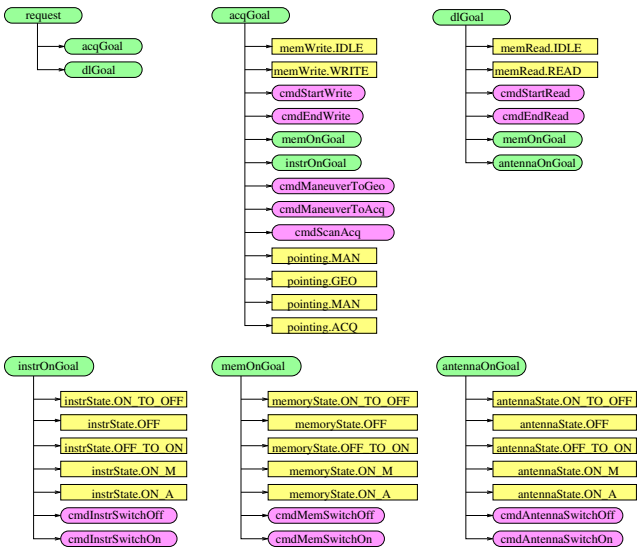

Figure 3: Decomposition of goals into subgoals and tokens potentially used for realizing them (high-level goals in green, atomic low-level goals in magenta, tokens in yellow)

Even if it is not explicitly represented here, each goal has parameters which describe its features. There is also a relationship between the parameters of a goal and the parameters of the subgoals and tokens it covers. For instance, for an *acqGoal* realized over time slot $[150, 173]$, subgoal *memOnGoal* has $[150, 173]$ as a required time slot. Similarly, given an acquisition goal, if the roll angle at the start of the acquisition is 24 degrees, then the *cmdManeuverToAcq* command also has as a target the 24 degrees roll angle.

Data Structures Associated With Reactors In the autonomous EOS controller, each goal type is handled by a unique reactor and each reactor handles a finite set of potential goals. For instance, there is a constant giving the maximum number of acquisition goals which can be handled by the system. If this number is exceeded, low priority goals can automatically be rejected. Also, for the sake of future incremental decision procedures, each goal has an update status: `ADDED` in case of an addition of the goal, `REMOVED` in case of a cancellation by the reactor which requested the goal, `UPDATED` in case of change in the parameters of the requested goal, and `NONE` otherwise.

Additionally, one token is used for each timeline to represent its initial state, and static data structures are used to represent the final evolution of timelines. As an illustration, the *Acquisition Reactor* can use a *cmdManeuverToGeo* command to realize a maneuver to the geocentric pointing at the end of the planning horizon, together with some associated tokens over the pointing timeline. The acquisition and download reactors can use `IDLE` tokens at the end of their planning horizon. Each device reactor can used a *cmdSwitchOff* command to switch the device off after the last activity, together with tokens over the device state timeline.

6 Plan Adaptation Procedures

To handle the set of reactors, the autonomous controller uses a `planningStep()` function which sequentially calls the plan adaptation procedures of each reactor, from high-level reactors to low-level reactors. From a generic point of view, it makes reactors deliberate in a topological order of the DAG whose arcs correspond to goal and timeline value transmissions between reactors (note that the *Download Reactor* is at a lower level than the *Acquisition Reactor* due to read operations over the **pointing** and **acqEndTime** timelines). We now detail the deliberation procedure used by each reactor. We exclude the *MissionReactor* which only forwards acquisition and download subgoals without making any decision. We also exclude the *Instrument*, *Memory*, *Antenna*, and *Pointing Monitors* which only maintain a chronological ordering of the goals they receive.

Acquisition Reactor: Greedy Hierarchical Adaptation The *Acquisition Reactor* computes the sequence of acquisitions which must be successively realized by the satellite. It uses a generic temporal constraint manager (Pralet 2017) to automatically obtain earliest and latest start times of successive acquisitions, based on the realization window chosen for each of them and on the required roll angle transitions

between acquisitions. Internally, the reactor handles temporally flexible plans, but the ON goals sent to low-level reactors are time-tagged (by the earliest start and end times of acquisitions).

Given the current plan, the *Acquisition Reactor* uses the following steps to adapt its internal plan and the sub-goals sent to lower level reactors (see Fig. 4 for an illustration):

1. read all acquisition goals updates (with status `REMOVED`, `ADDED`, or `UPDATED`); remove from the plan all goals whose status equals `REMOVED` or `UPDATED`, and add to the set of *active goals* all goals having the `ADDED` status; in Fig. 4, this step removes acquisition A from the plan due to the cloud cover prediction over A;
2. apply a greedy *selection-insertion* search: at each step, select an unplanned active goal that has no cloud cover and whose priority is the highest, and break ties by considering first the oldest activated goals; to insert the acquisition a associated with the goal:
   - consider the next possible acquisition window w for a in the chronological order; to free some place for a, remove all acquisitions a' of the plan which have a priority lower than a and which overlap w in the earliest start time plan (taking into account roll angle transitions); in Fig. 4, when the insertion of F is considered, this step removes observations B and C from the plan;
   - try to insert a into the plan at a position which offers the highest temporal flexibility in the realization of a, to favor future acquisition insertions;
   - if the insertion of a succeeds, then a new current plan is obtained; the acquisitions removed for inserting a can possibly be reinserted at future steps of the greedy search; in Fig. 4, this occurs for observation B, as shown in the final plan obtained, but not for C;
   - if the insertion of a fails, come back to the state of the plan before the test of the insertion of a in window w; try to insert a in the next window if one exists, or consider the next candidate acquisition for insertion;
3. in the plan, add intermediate geocentric pointings as soon as the distance between two successive acquisitions is greater than a given threshold, and come back to a geocentric pointing at the end of the planning horizon; add maneuvers to fill the **pointing** timeline;
4. from the earliest start time plan obtained, derive all updates on goals to be sent to the lower level reactors.

The incrementality of the previous procedure could be improved, for instance by avoiding the recomputation of all geocentric pointings each time a new plan is synthesized.

Download Reactor: Chronological Decision Rules The *Download Reactor* restarts from an empty download plan when there is a change in the end time of an acquisition or in the data volume after compression. It also rebuilds a plan when a change in the pointing profile invalidates one download activity of the current plan. The *Download Reactor* deliberates following decision rules that fill the download plan in a chronological way, by considering at each step

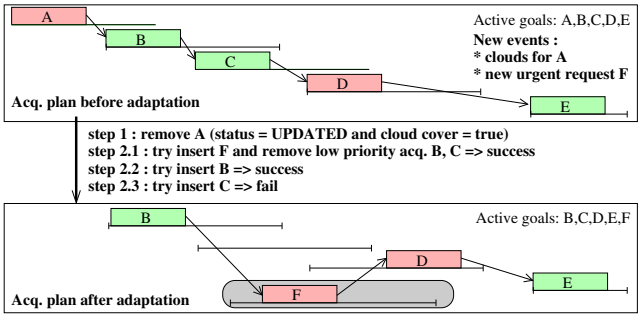

Figure 4: Adaptation of the acquisition plan (prioritary acquisitions in red, non prioritary ones in green)

a download goal (*dlGoal*) associated with a highest priority request (greedy selection-insertion mechanism). The deliberation procedure is illustrated in Fig. 5 and works as follows.

1. Based on the pointing timeline obtained from the *Acquisition Reactor*, the set of available station visibility windows is reduced in order to get *effective* visibility windows during which downloading data is compatible with the pointing used for observing (angle between the satellite pointing and the satellite-station direction less than a given threshold). In Fig. 5, the visibility window of station 1 is split into two effective visibility windows.
2. To initialize the chronological construction of a plan, the current time t is set to the maximum between the start time of the first effective station window and the first date at which both the mass memory and the emission antenna can be in the ON state, given the initial state of the device state timelines.
3. At each step, the planning algorithm selects the highest priority download goal whose realization can start at time t and finish before the end of one effective station window available at t. Ties are broken by considering first the download goals associated with data whose acquisition time is the lowest (the oldest data).

   (a) If one such download goal g exists, it is enqueued at the end of the download plan and the new current time is set to the end time of the realization of g. In Fig. 5, this decision rule first inserts the download $dl(A)$ of acquisition A when the current time equals $t3$, and then the download $dl(B)$ of acquisition B when the current time equals $t4$. The download of A is inserted first because A has a highest priority. The download of B is realized before the download of F because $dl(F)$ is not a candidate download yet when the current time equals $t4$, since the end time of acquisition F has not been reached yet.

   (b) If no such download goal exists, the current time t is moved forward to the minimum between the next acquisition end time and the next start time of an effective download window. The first case occurs when the current time is changed from $t6$ to $t7$ in Fig. 5.
4. The algorithm stops when there is no more effective station window or download goal over the planning horizon.

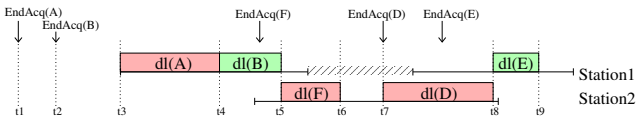

Figure 5: Successive insertions of downloads in the download plan during a deliberation of the *Download Reactor*

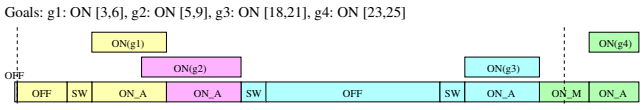

Figure 6: Plan built by a *Device Reactor* to satisfy ON goals over fixed time intervals (SW tokens correspond to ON_TO_OFF and OFF_TO_ON transitions)

Device Reactors: Chronological Decision Rules A *Device Reactor* restarts from an empty plan when an *ON goal* is updated (added, removed, or moved). It then deliberates over a short-term horizon to produce a so-called *ON/OFF plan* for the device. The deliberation procedure is illustrated in Fig. 6 and uses the following steps.

1. The ON goals are ordered by increasing start times. Some of these goals come from the *Acquisition Reactor*, while others come from the *Download Reactor*, hence there can be a temporal overlapping between them.
2. At each step, the next ON goal g over a given temporal window $[t_1, t_2]$ is considered. Let δ denote the duration required by an OFF_TO_ON transition. Two cases are analyzed to deliberate.

   (a) In the first case, the last token on the device state timeline takes value OFF or ON_TO_OFF and finishes at time t. If it is not possible to have the device in the ON state at t_1 ($t_1 - t < \delta$), the ON goal is rejected. Otherwise, the device is kept off until time $t_1 - \delta$, switched on at $t_1 - \delta$, and used in the ON state over $[t_1, t_2]$ (insertion of one OFF token over $[t, t_1 - \delta]$, one OFF_TO_ON token over $[t_1 - \delta, t_1]$, and one ON_A token over $[t_1, t_2]$). In Fig. 6, these steps are used for planning the realization of goal g1 (yellow).

   (b) In the second case, the last token on the device state timeline takes value ON_A (active ON), ON_M (maintain ON), or OFF_TO_ON. If $t \geq t_2$, no operation is needed. Otherwise, if $t \geq t_1$, a token is added to use the device in the ON state until time t_2 (insertion of an ON_A token over $[t, t_2]$, used for instance when inserting goal g2 in Fig. 6). Otherwise, the reactor decides whether the device must be temporarily switched off just after t and then switched on a short time before t_1 (option used for the insertion of goal g3 in Fig. 6), or whether it must be maintained on from t to t_1 (option used for the insertion of goal g4 in Fig. 6). The choice between the two options is realized by switching the device off when $t_1 - t$ exceeds a predefined threshold. At any step, the tokens added on timelines are those which are statically associated with goals, as defined in Fig. 3. Note that in Fig. 6, goal g4 is located after the end of the planning horizon (vertical bars on the figure), but it must be taken into account to determine whether the device must be switched off between the end of g3 and the end of the planning horizon.
3. If the end time of the planning horizon is not reached, commands and tokens are added to switch the device off just after the last token of the device state timeline and to keep it off until the end of the planning horizon.

The incrementality of the previous procedure could be improved to avoid building ON/OFF plans from scratch each time an input ON goal changes. However, the operations realized already have a low worst-case time complexity. Note also that for building an ON/OFF plan, there is no need to use a complex constraint-based or PDDL-based planner. The dedicated procedure proposed is both intuitive for the end-users and fast. Moreover, it is generic since it is used for three devices, the only parameter to set being the duration of ON_TO_OFF and OFF_TO_ON transitions, and the threshold defining when a device must be temporarily switched off.

Goal Planning Failures Goal planning failure is not an issue for the reactors used. Indeed, even if the sets of acquisition and download goals requested by the *Mission Reactor* usually lead to over-constrained planning problems, having unplanned goals is not an issue for the *Mission Reactor*. Moreover, the *Acquisition* and *Download Reactors* integrate constraints that ensure that the ON goals they request will always be feasible by the *Device Reactors*.

In a more general context, when a reactor r fails in planning a goal g requested by another reactor r', reactor r' receives a message expressing that the realization of g has not been planned. At the next deliberation cycle, r' can decide to update its own plan accordingly. Such planning steps can be iterated to try to converge to a state where all requested goals are planned. Even if the total allocated deliberation time does not suffice to reach a globally consistent plan, the plans of low-level reactors still remain valid, and in the worst case the satellite will just plan some useless operations.

7 Management of the Execution

Goal Execution Status As in the APSI Deliberative Reactor (Fratini et al. 2011; Cesta et al. 2012), each goal has an execution status which allows to determine whether it needs to be taken into account by the deliberation procedures. In our case, the execution status of an active goal takes value BUFFERED when the execution of the goal has not started yet, EXECUTING when the realization of the goal is ongoing, EXEC_SUCCESS when the execution of the goal has succeeded, and EXEC_FAIL when it has failed. In our current prototype, we do not consider execution failures for which an additional analysis module explaining the reasons of the failure would be needed, together with potential reconfiguration mechanisms.

At the moment, our autonomous EOS controller is not used on a real satellite or connected to a real processor used in the space domain. Instead, we just simulate the execution of all basic commands. As commands *cmdStartWrite*, *cmdEndWrite*, *cmdStartRead*, and *cmdEndRead* are executed within a single execution cycle, they have a null execution

duration and immediately get status EXEC_SUCCESS when their execution is triggered. Command *cmdScanAcq* used for scanning an observation area gets status EXECUTING when it is activated, and status EXEC_SUCCESS when the duration of the associated acquisition is elapsed. Similarly, all commands used for switching devices on and off get status EXECUTING when they are activated, and status EXEC_SUCCESS once the fixed duration of the associated device state transition is elapsed. The same kind of simulation can be made for commands *cmdManeuverToGeo* and *cmdManeuverToAcq*.

The execution status of an acquisition goal becomes EXECUTING when the associated WRITE token starts, and EXEC_SUCCESS when this token ends. This implies that for the deliberation process, the realization of an acquisition goal can always be changed until its execution starts, even if previous operations were performed such as switching on devices for the sake of this goal. This brings more flexibility to insert the realization of last-minute high priority requests. Similarly, the status of a download goal becomes EXECUTING when the associated READ token starts, and EXEC_SUCCESS when this token ends. The status of an ON goal expressed on devices (handled by the *Memory*, *Instrument*, and *Antenna Reactors*) also takes value EXECUTING and EXEC_SUCCESS respectively at the start and end times associated with this goal. Last, the execution status of a request becomes EXECUTING when the execution of the associated acquisition goal starts, and status EXEC_SUCCESS when the execution of the associated download goal ends.

Interleaving Execution and Planning Let t denote the current time (the execution time). As in T-REX, if the deliberation process of a reactor r is triggered at time t, then this process considers a planning horizon starting at time $t' = t + \lambda_r$ where λ_r is called the latency of reactor r. Another deliberation process will be triggered at time t', to obtain deliberations over a rolling horizon.[2] At time t, the part of the plan located over time interval $[t, t'[$ is committed. In our implementation, the execution is simulated at all execution cycles $t'' \in [t, t'[$ to update the execution status of commands and goals.

The deliberation process removes all tokens ending strictly before t'. One difficulty is then that for a timeline tl, time t' might be placed in the middle of a non-interruptible token tk. Non interruptible tokens are MAN(ρ, ρ') and ACQ(ρ, ρ') for the **pointing** timeline, READ and WRITE for the **memRead** and **memWrite** timelines, and ON_TO_OFF, OFF_TO_ON, ON_A for the device state timelines. In this case, the deliberation process makes a copy of tk and records it as the initial token for timeline tl for the new planning horizon. Otherwise, if token tk is interruptible, the reactor makes a copy of tk and records it as the initial token for timeline tl, but the end time of this token is set to t', the start time of the planning horizon. As in T-REX, each reactor r then builds a plan over temporal window $[t', t' + \pi_r]$, where π_r is called the planning horizon of r (with $\pi_r \geq \lambda_r$). In the end, planning and execution are synchronized based on commit window $[t, t'[$, on goal execution status, and on the initial state computed for each timeline at t'.

[2]Recent strategies defined for M2020 schedulers that mix fixed cadence and event-driven deliberations could be considered for improving the efficiency of the approach (Chi et al. 2018).

8 Experiments

The implementation of our EOS autonomous controller is based on several generic features for representing timelines, tokens, goal decompositions, reactors having planning and execution functions, goal activation and deactivation, and commitments. We used the prototype developed both on hand-written scenarios to test particular configurations and on randomly generated scenarios that validate the behavior of the prototype on a wider range of situations.

Each scenario (hand-written or random) defines the number of acquisitions and their features (priority, duration, roll angles, initial cloud cover prediction, maximum data volume, acquisition windows), the number of ground stations and their features (zenith angle of the satellite, download windows), and the features of the devices (durations of the transitions between the ON and OFF states, and maximum time gap for leaving the device ON between two usages).

Each scenario also defines all events that will be received by the autonomous controller over the planning horizon, including (1) the arrival of new acquisition requests (a scenario specifies the time at which each request is received by the controller), (2) the arrival of new cloud cover predictions (a scenario specifies the time at which each prediction is received), (3) the real volume of data associated with each acquisition (this real volume information is received by the controller at the end of each acquisition realized). In random scenarios, the parameters of these events are randomized, as well as the features of candidate observations and ground reception stations.

During our first experiments, we were able to visualize the events received by the autonomous controller, the evolution of the current plan (same view as in Fig. 1), and the concurrent execution of actions. For scenarios involving 10 requests, 3 priority levels, and 3 stations, each deliberation of the full controller (*i.e.* over all reactors) requires at most 3 milliseconds to converge to a new current plan on a standard laptop (Intel i5, 1.2GHz, 4GBRAM).

9 Conclusion

This paper presented an autonomous mission controller built for a new generation of EOS. One of its main strength is its capacity to use efficient reasoning procedure at the level of each reactor of the architecture. We should now test the approach on scenarios involving satellites which are agile along the three axis, and on scenarios involving memory and energy limitations. One challenge is also to formally validate the behavior of the controller, by validating first the behavior of each individual reactor, and then the interactions between the reactors. In this direction, it would helpful to use a formal language for describing the reactor architecture and the goal decompositions. The last point would be to add new modules providing to each reactor the current state observed or estimated for the timelines it owns.

References

Cesta, A.; Fratini, S.; Orlandini, A.; and Rasconi, R. 2012. Continuous planning and execution with timelines. In *ISAIRAS'12*.

Chi, W.; Chien, S.; Agrawal, J.; Rabideau, G.; Benowitz, E.; Gaines, D.; Fosse, E.; Kuhn, S.; and Biehl, J. 2018. Embedding a scheduler in execution for a planetary rover. In *ICAPS'18*.

Chien, S., and Troesch, M. 2015. Heuristic onboard pointing re-scheduling for an Earth observing spacecraft. In *IWPSS'15*.

Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 1999a. Integrated planning and execution for autonomous spacecraft. In *IAC'99*.

Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 1999b. Using iterative repair to increase the responsiveness of planning and scheduling for autonomous spacecraft. In *IJCAI'99 PLAN Workshop*.

Chien, S.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davies, A.; Lee, R.; Mandl, D.; Frye, S.; Trout, B.; Hengemihle, J.; Agostino, J. D.; Shulman, S.; Ungar, S.; Brakke, T.; Boyer, D.; Gaasbeck, J. V.; Greeley, R.; Doggett, T.; Baker, V.; Dohm, J.; and Ip, F. 2004. The EO-1 autonomous science agent. In *AAMAS'04*.

Chien, S.; Doubleday, J.; Thompson, D. R.; Wagstaff, K. L.; Bellardo, J.; Francis, C.; Baumgarten, E.; Williams, A.; Yee, E.; Fluitt, D.; Stanton, E.; and Piug-Suari, J. 2014. Onboard autonomy on the intelligent payload experiment (IPEX) cubesat mission: a pathfinder for the proposed HyspIRI mission intelligent payload module. In *ISAIRAS'14*.

Estlin, T.; Rabideau, G.; Mutz, D.; and Chien, S. 2000. Using continuous planning techniques to coordinate multiple rovers. *Electronic Transactions on Artificial Intelligence* 4(A).

Fratini, S.; Cesta, A.; Benedictis, R. D.; Orlandini, A.; and Rasconi, R. 2011. APSI-based deliberation in goal oriented autonomous controllers. In *ASTRA'11*.

Gaines, D.; Anderson, R.; Doran, G.; Huffman, W.; Justice, H.; Mackey, R.; Rabideau, G.; Vasavada, A.; Verma, V.; Estlin, T.; Fesq, L.; Ingham, M.; Maimone, M.; and Nesnas, I. 2016. Productivity challenges for Mars rover operations. In *PlanRob'16*.

Ilsen, S.; Gerrits, D.; Vrancken, D.; Naudet, J.; Mellab, K.; Santandrea, S.; Laroche, T.; and Verheyden, A. 2014. PROBA-V: The example of onboard and onground autonomy. In *AIAA/USU'14*.

Jónsson, A. K.; Morris, P.; Muscettola, N.; and Rajan, K. 2000. Planning in interplanetary space: Theory and practice. In *AIPS'00*.

Khatib, L.; Frank, J.; Smith, D.; Morris, R.; and Dungan, J. 2013. Interleaved observation execution and rescheduling on Earth observing systems. In *the ICAPS'03 Workshop on Plan Execution*.

Knight, R.; Rabideau, G.; Chien, S.; Engelhardt, B.; and Sherwood, R. 2001. CASPER: Space exploration through continuous planning. *Intelligent Systems IEEE* 16(5):70–75.

Lemai, S., and Ingrand, F. 2004. Interleaving temporal planning and execution in robotics domains. In *AAAI'04*.

Maillard, A.; Verfaillie, G.; Pralet, C.; Jaubert, J.; Sebbag, I.; and Fontanari, F. 2015. Postponing decision-making to deal with resource uncertainty on Earth-observation satellites. In *IWPSS'15*.

Maillard, A.; Verfaillie, G.; Pralet, C.; Jaubert, J.; Sebbag, I.; Fontanari, F.; and L'Hermitte, J. 2016. Adaptable data download schedules for agile Earth-observing satellites. *Journal of Aerospace Information Systems* 13(8).

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2008. A deliberative architecture for AUV control. In *ICRA'08*.

Muscetolla, N.; Nayak, P. P.; Pell, B.; and Williams, B. 1998. Remote Agent: to boldly go where no AI system has gone before. *Artificial Intelligence* 103:5–47.

Muscettola, N.; Dorais, G.; Fry, C.; Levinson, R.; and Plaunt, C. 2002. IDEA: Planning at the core of autonomous reactive agents. In *IWPSS'02*.

Ocón, J.; Buckley, K.; Colmenero, F.; Bensalem, S.; Dragomir, I.; Karachalios, S.; Woods, M.; Pommerening, F.; and Keller, T. 2018. Using the ERGO framework for space robotics in a planetary and an orbital scenario. In *ISAIRAS'18*.

Policella, N.; Smith, S.; Cesta, A.; and Oddi, A. 2004. Generating robust schedules through temporal flexibility. In *ICAPS'04*.

Pouly, J.; Jouanneau, S.; and Olhagaray, P. 2014. Autonomous mission planning in space : Mission benefits and real-time performances. In *ERTS*^{2}.

Pralet, C.; Verfaillie, G.; Maillard, A.; Hébrard, E.; Jozefowiez, N.; Huguet, M.-J.; Desmousceaux, T.; Blanc-Paques, P.; and Jaubert, J. 2014. Satellite data download management with uncertainty about the generated volumes. In *ICAPS'14*.

Pralet, C.; Infantes, G.; and Verfaillie, G. 2013. A generic constraint-based local search library for the management of an electromagnetic surveillance space mission. In *ICAPS'13 Application showcase*.

Pralet, C. 2017. An incomplete constraint-based system for scheduling with renewable resources. In *CP'17*.

Rabideau, G., and Benowitz, E. 2017. Prototyping an onboard scheduler for the Mars 2020 rover. In *IWPSS'17*.

Rabideau, G.; Chien, S.; and Laren, D. M. 2009. Tracatable goal selection with oversubscribed resources. In *IWPSS'09*.

Verma, V.; Gaines, D.; Rabideau, G.; Schaffer, S.; and Joshi, R. 2017. Autonomous science restart for the planned Europa mission with lightweight planning and execution. In *IWPSS'17*.

Victoria, J. M. D.; Yeomans, B.; Gao, Y.; and Stryk, O. V. 2015. Autonomous mission planning and execution for two collaborative mars rovers. In *ASTRA'15*.

Washington, R.; Golden, K.; and Bresina, J. 2000. Plan execution, monitoring, and adaptation for planetary rovers. *Electronic Transactions on Artificial Intelligence* 4(A).

Woods, M.; Long, D.; Baldwin, L.; Aylett, R.; Wilson, G.; Ward, R.; and Vituli, R. 2006. On-board planning and scheduling for the ExoMars mission. In *DASIA'06*.

Wörle, M., and Lenzen, C. 2013. Ground assisted onboard planning autonomy with VAMOS. In *IWPSS'13*.