17th International Conference on Space Operations, Dubai, United Arab Emirates, 6 - 10 March 2023.

SpaceOps-2023, ID # 246

Proactive-reactive on-board planning for complete goal-oriented automation

Riccardo Maderna[a]*, Gabriele Giordana[a]

[a] *AIKO S.r.l., Via dei Mille 22, 10123, Torino, Italy*, name@aikospace.com
* Corresponding Author

Abstract

This paper presents an approach to goal-oriented automation of satellites through the integration of proactive and reactive on-board planning. The approach enables long-term planning and leverages the timelines formalism to manage concurrent processes, causal and temporal constraints, and different types of resources. The algorithm's output is a flexible plan that can adapt to execution uncertainties and discrepancies in resource consumption, resulting in an increased effectiveness of the entire mission and reducing response latency to events. The algorithm has been tested in simulated Earth Observation scenarios and has been shown to perform better than traditional ground-based mission control.
Keywords: Automated planning, Autonomous satellite operations, Earth observation, Timeline-based planning

1. Introduction

Satellites play a vital role in various industries such as telecommunications, weather forecasting, and earth observation. However, their operations are time-consuming and challenging due to the constraints imposed by their remote location and limited resources. In recent years, there has been a growing interest in developing autonomous systems for satellites to improve their efficiency and reduce reliance on ground-based control. One approach to achieving this is by using on-board planning, which refers to the process of generating and executing plans on the satellite itself, rather than relying on ground-based control. This enables the satellite to make decisions independently considering situational awareness, actual resource consumption, and self-diagnosis in prioritizing high-level goals and defining the optimal task plan. As a result, on-board re-planning ensures the maximization of mission goals within operational constraints and compensates for the reliance on communication with the ground stations, especially due to the limited frequency of transmission.

Traditional on-board planning approaches have primarily focused on the reactive repair of schedules computed on the ground in response to contingencies. Notable examples are VAMOS [1] by the German Aerospace Centre, implemented as part of the FireBIRD mission, which allows for simple onboard reaction to telemetry measurements and event detection by choosing from pre-calculated timeline alternatives. NASA's Jet Propulsion Laboratory's MEXEC [2] is designed to be a multi-mission onboard planning and execution software. It receives goals from the ground specified as task networks, which are transformed into conflict-free schedules based on state information. CASPER [3] focuses on high responsiveness through iterative repair. Following an anomaly, CASPER fixes flaws in the existing plan until a feasible one is found. [4] describes a just-in-time reactive planning architecture able to determine what to do over the next few minutes based on the latest system state and in response to triggering events. In [5], an autonomous recovery algorithm based on constraint programming is presented, which repairs a ground schedule after an error in processing an image taken by an EO satellite.

However, purely reactive approaches easily lead to suboptimal performance due to their myopic decision-making. They only respond to the immediate situation and fail to consider the effects on mission goals or against potential future contingencies. Conversely, proactive planning can inject long-term reasoning to fulfil mission goals. However, purely proactive planning can be inflexible and unable to respond to unexpected events. Instead, the integration of planning and execution leveraging flexibility, replanning, and utility maximization has been shown to achieve the best results for tightly constrained space missions that must address significant uncertainty [6].

In this paper, we propose a proactive-reactive on-board planning approach for complete goal-oriented autonomy on satellites [7]. The proactive part transforms the mission goals into a long-term flexible plan of activities that complies with operational constraints. The reactive part executes the flexible plan in real time to pursue two main goals. First, it exploits flexibility to absorb uncertainties and contingencies. Second, it refines activities to maximize the mission return based on local information. Additionally, a reasoning engine continuously evaluates the validity of planned goals and triggers replanning when major discrepancies are detected.

The proposed approach leverages the timelines formalism [8] to manage concurrent activities, causal and temporal constraints, and the availability of different types of onboard resources. While most of the well-known planners in this field were designed to fit specific mission requirements, the proposed solution aims to be applicable

to a wide range of missions, thanks to a robust integration of domain-independent techniques and the possibility to refine the application with domain-specific knowledge.

The remainder of the paper is organized as follows. Section 2 presents the overall architecture and the functional chain. Then, Section 3 and Section 4 detail the proactive and reactive planning loops, respectively. An example scenario is presented in Section 5 to depict the algorithm behaviour and discuss preliminary performance. Finally, Section 6 draws the conclusions and path forward.

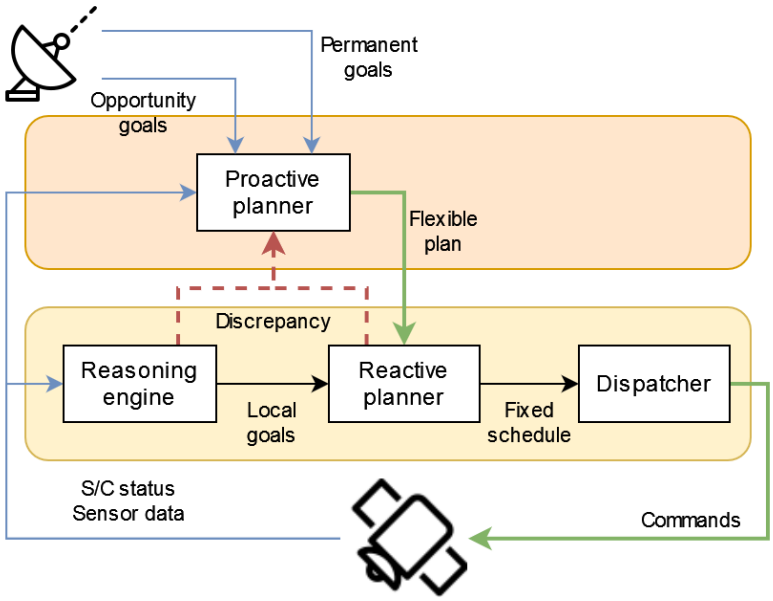

Fig. 1. Overall solution architecture highlighting the proactive (orange, top) and reactive (yellow, bottom) loops.

2. Solution overview

The general architecture, which is shown in Fig. 1, is composed of two main layers, each one corresponding to a planning loop with different objectives, frequencies, and horizons. Consequently, the proposed architecture relies naturally on several formalisms and processing approaches, which meet the functional requirements of each level.

The proactive layer is composed of one single component named Proactive Planner, which is implemented leveraging the timeline-based planning formalism. Its main function is to transform mission goals into a long-term flexible plan of activities that can span multiple orbits. The optimal set of mission goals to pursue is dynamically obtained by adjusting the priority of mission goals based on situational awareness, i.e., by considering operational constraints, the current system state, and its predicted evolution. Moreover, the Planner can generate new tasks to pursue secondary objectives and increase the mission outcome. The resulting flexible plan can be seen as an envelope of deterministic schedules, each one representing a fixed-in-time sequence of actions or commands. This flexibility allows for absorbing uncertainties during execution without the need for replanning, which strongly reduces the computational load of the whole planning pipeline and increases the predictability of the satellite's behaviour.

Once a valid long-term plan is available, the reactive layer is responsible for refining and executing it based on real-time data and events. To ensure reactivity, it works at high frequency on a short horizon, which is typically of a few minutes. The pipeline is composed of three components. The core is constituted by the Reactive Planner, which has two main functions. First, it exploits flexibility to absorb uncertainties and contingencies, which means selecting at any time the optimal deterministic local schedule from the flexible plan. Second, it provides a procedural system for task refinement to maximize the mission return based on local information. The latter step is needed to dynamically transform the higher-level activities treated by the Proactive Planner into a list of executable commands to be dispatched to the platform. The decisions taken by the Reactive Planner are informed by local goals inferred by a Reasoning Engine that processes all the information available on board, such as the state of the spacecraft, the results of edge data processing, the occurrence of external and internal events, failures, and so on. On the other side, the output commands from the Reactive Planner are fed to a Dispatcher unit that is responsible for the prompt transmission of time-tagged commands for execution.

In parallel, both the Reasoning Engine and the Reactive Planner look for discrepancies between the execution and the long-term plan, which make the latter invalid. For instance, an extremely long duration of an uncontrollable

activity might push the end of such activity outside the flexibility bounds (which is a case detected by the Reactive Planner), or the optimal local goal might be in contrast with the planned goal due to the emergence of a failure (which is a case detected by the Reasoning Engine). When a discrepancy is found, the algorithm searches for possible explanations to inform the Proactive Planner and trigger a replanning.

The proposed solution has not been developed to fit a specific mission. Conversely, it aims to be applicable to diverse mission concepts. Therefore, a particular emphasis has been put to design fully configurable algorithms through a robust integration of domain-independent techniques and the possibility to refine the application with domain-specific knowledge. Indeed, an automated planning system based on artificial intelligence differs from a traditional decision supporting system as the input specifies objectives ('what') rather than 'how'. Then, it is left to the planning process to come up with 'how' to achieve those objectives (i.e., with planned activities) leveraging a symbolic representation of the problem. The mission scenario can be easily specified as a configuration for the onboard planning system, comprising objectives, available activities, inputs, resources, and constraints. A clear and natural language-like syntax has been defined for this purpose to minimize the effort in tailoring the software to specific user needs.

3. Proactive planning loop

As anticipated in the previous Section, the Proactive Planner is implemented using timeline-based planning [8]. This formalism has been regarded as a powerful solution for mission planning thanks to its expressivity and capabilities in managing resources and task allocation. Additionally, the timeline-oriented paradigm is very close to the way problems and constraints are naturally represented in space applications, which eases usability and transparency from users.

The timelines-based approach assumes that the world is modelled as a set of entities, named state variables, whose properties can vary in time according to some internal logic or because of external inputs. Examples of state variables are the orientation of a spacecraft, its operating mode, and the visibility status of a ground station. State variables are either external, which describe exogenous elements on which the planner has no control (e.g., ground station visibility), or internal, which describe controllable states (e.g., the spacecraft orientation). The evolution in time of state variables is described by Timelines, which are defined as a sequence of tokens. In turn, tokens are (flexible) time intervals where the variable holds a single value. A chronicle fully describes the status of the planning problem by collecting all the timelines, together with constraints and resource availability.

The search space of the planning algorithm is constituted by the space of chronicles. Starting from a partial chronicle representing the current state, the planner iteratively tries to resolve flaws - i.e., constraint violations - in the current chronicle until a complete and feasible one is found. Finally, a flexible activity plan is described by the timelines and the constraints among tokens.

In order to manage both controllable and uncontrollable temporal relations among activities and external events, the chronicle maintains a Simple Temporal Network with Uncertainty (STNU). Temporal relations cover uncertain token durations, task precedence constraints, and synchronization rules among tasks and with external timelines. A STNU is consistent if every flexible timepoint can be given a value without breaking any constraint. In addition, for a plan to be executed, the STNU must also be dynamically controllable [9]. This property is maintained throughout the planning process. Furthermore, the planner manages different types of resources to guarantee the feasibility of the produced mission plan. For instance, it supports binary resources that can be either occupied or free (e.g., specific payloads or subsystems), capacity resources (e.g., available power), and consumable ones (e.g., propellant, onboard storage, or battery level). Whereas binary resources can be modelled as state variables (with associated timelines), dedicated resource managers are implemented to maintain the expected resource utilization profile of capacity and consumable ones.

3.1 Configuration and inputs

The user can configure the planning problem according to their specific concept of operations and platform characteristics. In principle, this configuration shall be done once per mission as it pertains to information that remains the same across subsequent planning requests, such as the definition of state variables and resources. Nevertheless, the configuration can be updated at any time during the mission to reflect changes in mission objectives or platform capabilities. Table 1 lists the mission configuration inputs expected by the Proactive Planner. Instead, when a replanning is triggered, the Proactive Planner gathers information about the current state of the system to build the initial chronicle, which is listed in Table 2.

In addition to configuration and state information, the definition of mission goals is fundamental. The Proactive Planner supports two types of goals, which are named permanent and opportunity goals, respectively. Permanent goals capture the value of performing certain activities throughout the mission. For instance, they can express

survivability goals (e.g., keeping the spacecraft batteries charged), or data retrieval ones (e.g., maximizing payload data acquisition and prioritizing downlink if there are payload data to be transmitted). Usually, permanent goals remain valid throughout the mission, but can be adapted to reflect changes in mission objectives or platform capabilities, like the planner configuration inputs. Conversely, opportunity goals describe goals that are valuable only when achieved at a prescribed time. A clear example is user requests for camera imaging acquisitions of a target of interest. Opportunity goals are specified as a prioritized list of time-bound activities that increase the plan value if included in the chronicle. No prior check on the presence of conflicts among goals and other activities is needed, but the planner evaluates which is the best subset of opportunities to include while obtaining a feasible plan.

Table 1. Proactive Planner configuration inputs

Name	Description	Data Structure	Example
Planning horizon	Time span covered by the plan	Int	120
State variables	List of state variables	(Name, type, [*state*], [*transition*])	('orientation', internal, [s_1, s_2, …], [τ_1, τ_2, …])
- State	Feasible state for a state variable	(Name, min_duration, max_duration, controllability)	('downlink', 2, ∞, true)
- Transition	Feasible transition between states of a state variable	(Origin, destination)	(s_1, s_2)
Synchronization rules	Describe relations between two states belonging to different state variables	[(operator, source, target)]	(During, 'observation', 'nadir_pointing')
Resources	List of available resources	[*resource*]	[r_1, r_2, …]
- Resource	Description of a resource	(Name, type, lower_bound, upper_bound)	('battery', consumable, 26.0, 32.0)
Resource usage	Map between state and resource usage	(*State*, *resource*, usage)	Consumption: (s_1, r_1, -20.0) Generation: (s_1, r_2, 150.0)

Table 2. Proactive Planner planning inputs

Name	Description	Data Structure	Example
External timelines	Complete specification of external timelines	[*timeline*]	[T_1, T_2, …]
- Timeline	Sequence of tokens	[*token*]	[t_1, t_2, …]
- Token	Description of a token	(state_name, start, end)	('visible', 0, 8)
Resource level	Current availability level of resources	[(*resource*, level)]	[(r_1, 45.3), (r_2, 121.0)]
Mandatory activities	List of activities that must be inserted in the plan	[*token*]	[t_3, t_4, ….]

3.1 Main planning routine

The main elements that are manipulated in the algorithm are the chronicle, flaws, resolvers, and the exploration stack. The chronicle describes the current partial plan, as already defined. Flaws are defects in a chronicle that must be solved to obtain a complete and feasible plan. A resolver for a flaw defines modifications to the current chronicle that can solve the flaw, for example by adding tokens or constraints. Different flaw classes exist, which are outlined in the following:

- *Conflict flaw:* tokens in timelines cannot overlap; a conflict flaw exists between two overlapping tokens.
  Resolvers: a conflict between two tokens is resolved by posting a new precedence constraint to induce an ordering between the two tokens. Solving all conflict flaws induces an ordering of all tokens in the timeline, which is useful to search and solve all other flaws.
- *Synchronization flaw:* all synchronization rules among state variable values associated with tokens in the chronicle's timelines must hold. A synchronization flaw exists for each unsatisfied synchronization rule.
  Resolvers: a synchronization flaw is resolved by applying the required temporal constraints, depending on the synchronization operator. One resolver exists for each target token present in the timeline. In case the target belongs to an internal timeline, an additional resolver consists in adding a new target token that satisfies the synchronization rule.

- *Resource limit flaw*: resource usage must be limited to its capacity at any time; a resource limit flaw exists when the resource level at a given time point is outside the capacity bounds. Since token durations are uncertain, flaws are searched in the worst-case resource utilization profile.
  Resolvers: resource flaws can be solved in different ways. First, by constraining consumer (producer) tokens to start after (end before) the flaw. Second, by reducing (increasing) the duration of a consumer (producer) token before the flaw. Third, by adding a new producer token before the flaw.
- *Opportunity flaw:* formally speaking, opportunity is not a flaw, but it is convenient to treat it as such. One opportunity flaw per opportunity goal is generated to evaluate the inclusion of each goal in the chronicle.
  Resolvers: the solution to the opportunity flaw is either to include (adding an appropriate new token) or discard (i.e., do nothing) the associated opportunity goal.
- *Unsupported token flaw*: all tokens in timelines must be causally supported. A-priori-supported tokens represent known facts and usually define the initial state of timelines. All other tokens are supported if there exists a connected predecessor token describing a feasible state transition. The ultimate result of supporting tokens is to close the gaps in the timeline.
  Resolvers: a token is supported by connecting it to a valid preceding token. The supporting token can already exist in the timeline or be generated by the resolver. The former choice goes in the direction of consolidating the final chronicle, the latter performs a generative search to seek permanent goals in the plan.

Finally, the exploration stack tracks the history of plan space exploration. i.e., the list of all applied resolvers, that is needed to manage backtracking in case of inconsistency. During backtracking, all resolvers applied between the current chronicle and the last decision point are retracted in inverse chronological order. A pseudocode overview of the planning routine is the following:

```
Plan(initial_state, final_state, goal_list):
    Goals ← Initialize(goal_list)
    Chronicle ← Initialize(initial_state, final_state)
    while Flaws
        Flaw ← GetHighestPriorityFlaw(Chronicle, Goals)
        Resolvers ← ComputeResolvers(Flaw)
        if |Resolvers| > 1:
            Exploration Stack ← Add(Decision Point)
            Resolver ← GetHighestPriorityResolver(Resolvers)
        Exploration Stack ← Add(Resolver)
        Apply(Chronicle, Resolver)
        if Chronicle is inconsistent:
            Backtrack to the previous decision point
    return ExtractPlan(Chronicle)
```

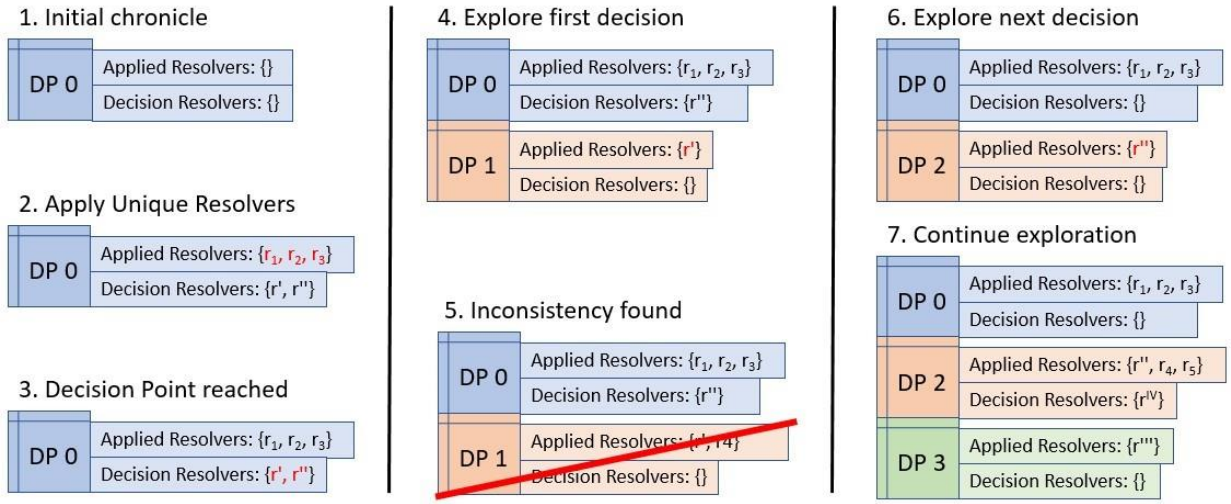

Fig. 2. Example of Execution Stack evolution

When multiple flaws are present in a chronicle, the choice of the first flaw to solve is mainly guided by the fail-first principle: all flaws must be solved but those with fewer resolvers are more constrained, and it is advisable to address them first. This encourages early failure and reduces the average depth of branches in the search tree. Conversely, when multiple resolvers are present for a given flaw, the choice is guided by a trade-off between the effect the resolver has on permanent goals (to maximize if positive) and how much it constrains the search space (to minimize). Figure 2 depicts an example evolution of the exploration stack during the initial iterations of the main planning routine.

4. Reactive planning loop

The role of the proactive planning loop is to refine the flexible plan generated by the Proactive Planner into a sequence of fixed-in-time commands for the platform. To do so, it exploits flexibility to absorb uncertainties and contingencies and refines activities to maximize the mission return based on local information. Therefore, a key feature of the reactive loop is the ability to collect information about the current state of the mission and infer local goals to guide the refinement process.

This role is fulfilled by the Reasoning Engine, which is the first element in the reactive pipeline (see Fig. 1). It implements an expert system, i.e., an AI software that uses scenario and system-level information stored in a knowledge base to solve problems that would usually require a human expert, thus preserving its knowledge in a database. An inference engine is applied to the knowledge base to derive information starting from already-known facts. Lower-level information is queried during the inference process until a known fact is encountered, thus reconstructing the actual system and mission knowledge state. The Reasoning Engine embeds different fuzzy logic functionalities at input level. The concept of fuzzy logic is extremely important to represent the fact that experts make decisions based on imprecise information; in this sense, these kinds of models are mathematical means of representing vagueness and imprecise information. This formalism fosters transparency in the onboard decision-making process by extracting explanations at a different abstraction level, closer to the form of knowledge familiar to a human expert.

Local goals determined by the Reasoning Engine are fed to the Proactive Planner. However, when the mission state differs too much from the expected one by the Proactive Planner, the determined local goals might be in contrast with the planned activities. In this case, a replanning trigger might be raised to generate a new long-term plan based on the new mission state.

The first role of the Proactive Planner is to extract a schedule starting from the flexible plan. Table 3 lists the real time inputs received by the Proactive Planner. From the temporal point of view, the schedule is continuously updated in response to the evolving system state to deal with minor uncertainties, i.e., those that remain in the foreseen flexibility bounds of the plan. Instead, when major discrepancies are detected, the generation of a new plan is triggered [10]. Discrepancies may lie in the duration of activities, the availability level of resources, and the occurrence of unexpected exogenous events or failures.

In addition to that, the Proactive Planner exploits hierarchical decomposition methods (defined, e.g., via hierarchical task networks [11]) to refine the high-level activities planned by the Proactive Planner into low-level tasks that can be transformed into a sequence of time-tagged commands by means of simple procedures. The definition of decomposition rules includes conditional choices based on local goals input, which are dynamically evaluated online. For instance, an Earth observation satellite can continuously adapt the acquisition frequency based on cloud detection information. Practically, this results in an observation activity being refined by a series of payload configuration commands determined in real-time. Clearly, the local refinement of the flexible plan has an impact on the actual state of the system, e.g., on the actual resource usage, which can lead to the insurgence of a discrepancy.

As the last step in the reactive pipeline, the Dispatcher is a simple component that takes the schedule of time-tagged commands and dispatches them to the platform.

Table 3. Reactive Planner planning inputs

Name	Description	Data Structure
Flexible plan	Proactive Planner output	*chronicle*
Clock	Current time	Time stamp
Events observations	Command execution and uncontrollable events acks used to scan the flexible plan	(Event, time stamp)
Local goals	Reasoning engine output	[(goal ID, parameters)]

5. Example scenario

In the following, a benchmark scenario is used to exemplify the algorithm behaviour and highlight improved performance with respect to common satellite planning methods. The scenario refers to a small satellite Earth observation mission, whose main objective is the acquisition of user-requested targets on the Earth's surface. Additional data can be acquired over wide regions of interest as a secondary objective to increase mission return. Onboard payload data processing can evaluate the quality of acquired data (e.g., cloud detection) and can be used to adapt the frequency of acquisition to spare spacecraft resources. The proactive planning problem is modelled with two external timelines that describe eclipses and ground station visibility, respectively, and two internal timelines. The latter are:

- the operating mode timeline, whose feasible states are "idle", "observation", "monitoring", "downlink", "ground control".
- the spacecraft orientation timeline, whose feasible states are "solar panels pointing sun", "antenna pointing ground station", "camera pointing target", "slewing".

The battery level and memory storage occupation are tracked as consumable resources. Acquisition requests are modelled as opportunity goals, whereas extra data takes are modelled as a permanent goal that increases the value of observation activity. At the reactive level, the observation activity can be refined into high- and low-frequency acquisition based on payload data processing and resource availability, whereas the monitoring activity can locally switch to observation mode in particularly favourable or relevant conditions. Moreover, every high-level activity is decomposed into elementary tasks and then low-level commands via predefined procedures.

Figure 4 gives an example of proactive planning output. Specifically, Figure 4(a) depicts the initial chronicle. The first row shows the list of opportunity goals to include (priority information is not shown). The external timelines, in blue, are already consistent and complete, whereas internal ones have only the initial status defined, together with a few mandatory slots reserved from ground-commanded activity (e.g., to run periodic tests and maintenance). Red and green tokens represent unsupported and supported tokens, respectively. Light colour represents the feasible interval of flexible time points. At the bottom, the worst-case foreseen battery profile is depicted as an example of a consumable resource. Figure 4(b) depicts the partial plan after the evaluation of all opportunity goals. One can notice that not all opportunities have been inserted due to conflict with other tasks, insufficient resources, or because they lead to inconsistent plans that have been backtracked. A complete and feasible plan is shown in Fig. 4(c), where one can notice that all tokens are supported (all are green) and a margin of flexibility is still present. Clearly, the amount of flexibility in the Proactive Planner output highly depends on the uncertainty of the scenario (e.g., the presence of activities with uncontrollable duration, such as pre-heating periods).

Figure 5 gives notable examples of the behaviour of the reactive planning loop. The first row highlights the presence of an opportunity goal, which is reflected by the planned observation activity in the timeline (second row). The observation token covers at least the requested observation, but flexibility is present in the plan before switching to the monitoring activity. The expected and measured battery charge levels are depicted in blue with a dashed and solid line, respectively. The bottom row shows the refined plan determined by the Reactive Planner. The initial situation is shown in Fig. 5(a), where the ongoing observation activity is refined as high-frequency acquisition based on local goals from the Reasoning Engine. The observation continues adapting the acquisition frequency to local goals until the end of the opportunity goal (Fig. 5(b)). Low-frequency acquisitions consume less battery and lead to a surplus of battery charge with respect to the expected one. Therefore, the Reactive Planner decides to extend the observation to maximize Permanent Goals (Fig. 5(c)). When the battery lowers (or the amount of data stored in the memory increases over the downlink budget), a new local goal suggests to the Reactive Planner to switch to the next activity.

6. Discussion and conclusions

The present work described a proactive-reactive approach for onboard automated planning. The solution is highly flexible and can be applied to a wide variety of missions. The proactive loop allows for long-term, goal-oriented planning while the reactive loop enables local decision-making and plan refinement in view of uncertainty and contingencies.

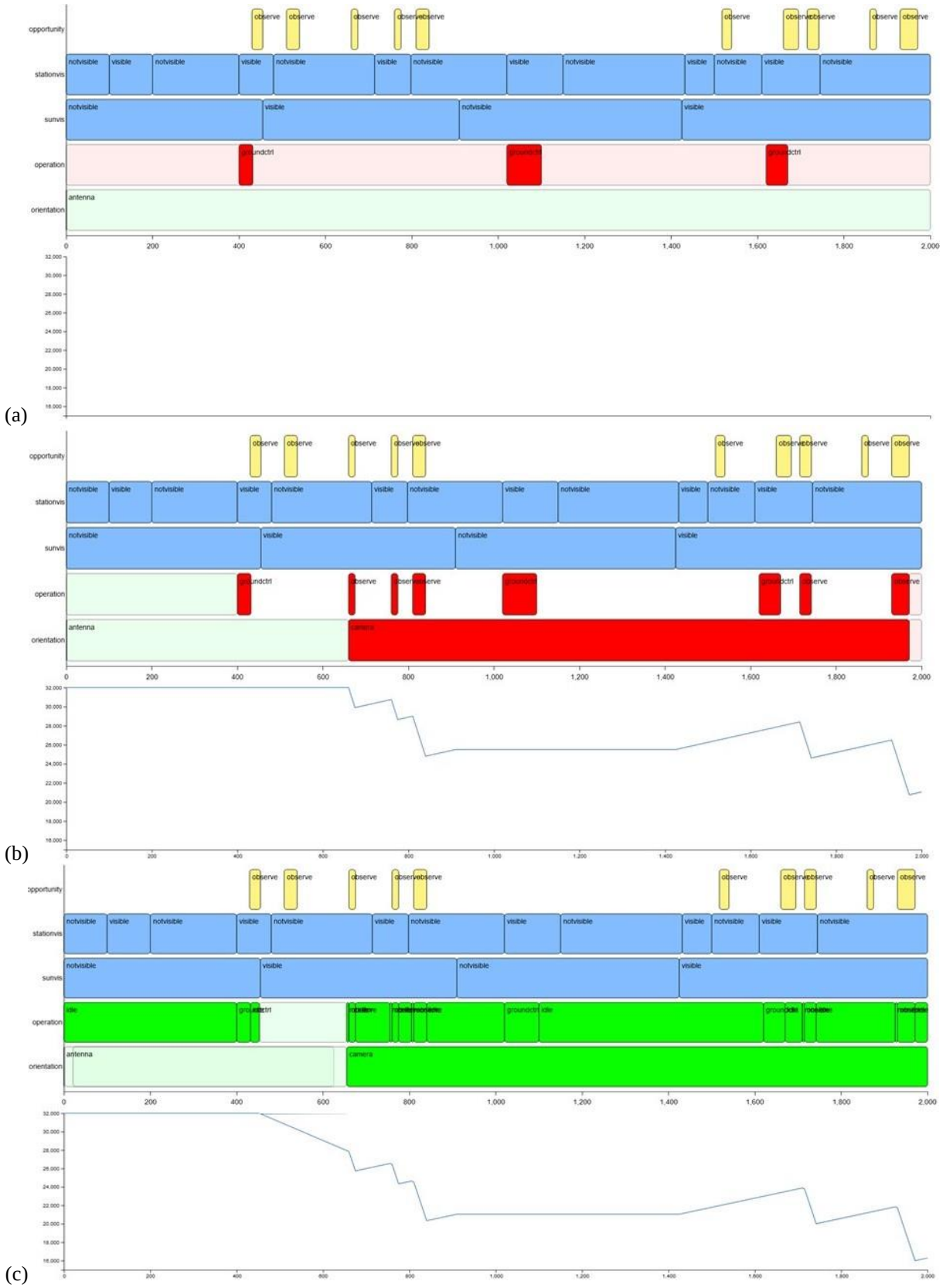

Fig.4. Example of proactive planning process from initial Chronicle to final plan. (a) Initial Chronicle, (b) Status after the insertion of all feasible opportunity goals, (c) Final flexible plan

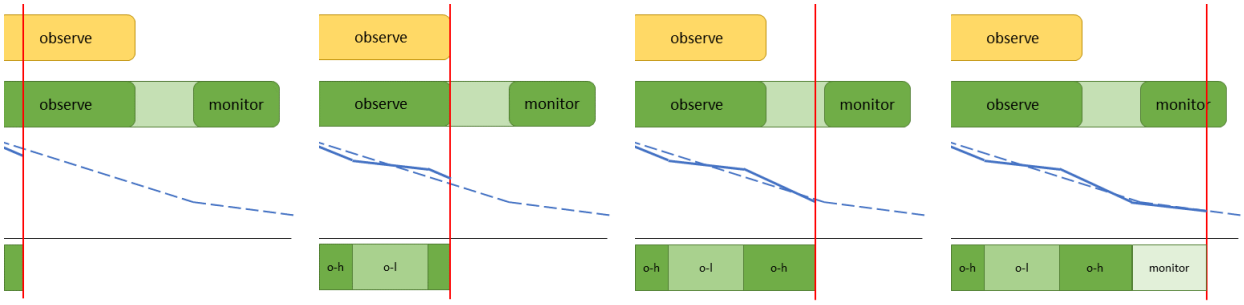

Fig.5. Example of reactive planning process: (a) observation activity refined as high-frequency acquisition, (b) observation continue switching frequency based on local goals, (c) plan flexibility allows extending the observation activity to exploit the surplus battery, (d) when the battery lowers, the Reactive Planner switch to the next activity.

On-board planning improves mission return if compared to results achievable with mission planning on the ground by optimizing payload and resource usage based on timely information that is only available on board. Both proactive and reactive loops contribute to this in different ways. Additionally, by increasing the level of onboard autonomy, new opportunities for space missions become available, which cannot be achieved with traditional ground-in-the-loop operations cycles due to communication constraints, dynamic environmental conditions, or limited mission lifetime. Also, the effort in ground mission operations is strongly reduced since a detailed activity plan is no more required. However, a change in the current operations processes, practices, and tools may be necessary to fully exploit onboard autonomy [12]. Specifically, the uplink phase should focus on communicating mission goals to the onboard planning system and assess the potential impact of such goals on the spacecraft state. In turn, the downlink phase should be used to reconstruct and explain decisions made by autonomy and assess the resulting spacecraft's state.

References

[1] M. T. Wörle and C. Lenzen, 'Ground Assisted Onboard Planning Autonomy with VAMOS', p. 9.
[2] M. Troesch et al., 'MEXEC: An Onboard Integrated Planning and Execution Approach for Spacecraft Commanding', International Conference on Automated Planning and Scheduling (ICAPS 2020), p. 9.
[3] S. Knight, G. Rabideau, S. Chien, B. Engelhardt, and R. Sherwood, 'Casper: space exploration through continuous planning', *IEEE Intell. Syst.*, vol. 16, no. 5, pp. 70–75, 2001, doi: 10.1109/MIS.2001.956084.
[4] E. Herz, D. George, T. Esposito, and K. Center, 'Onboard Autonomous Planning System', in SpaceOps 2014 Conference, Pasadena, CA, May 2014. doi: 10.2514/6.2014-1783.
[5] C. Powell and A. Riccardi, 'On-board re-planning of an earth observation satellite for maximisation of observation campaign goals', 73rd International Astronautical Congress (IAC 2022), p. 11, 2022.
[6] D. Wang, J. A. Russino, C. Basich, and S. Chien, 'Analyzing the Efficacy of Flexible Execution, Replanning, and Plan Optimization for a Planetary Lander', Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, vol. 32, p. 9, 2022.
[7] ECSS-E-ST-70-11C – Space segment operability
[8] Ghallab, M., Nau, D., and Traverso, P., Deliberation with Temporal Models, Cambridge University Press, 2016, p. 114–154. https://doi.org/10.1017/CBO9781139583923.006.
[9] Morris, P., Muscettola, N., & Vidal, T. (2001). Dynamic control of plans with temporal uncertainty.
[10] H. Muñoz-Avila, D. Dannenhauer, and N. Reifsnyder, 'Is Everything Going According to Plan? Expectations in Goal Reasoning Agents', AAAI, vol. 33, pp. 9823–9829, Jul. 2019, doi: 10.1609/aaai.v33i01.33019823.
[11] C. Qi, D. Wang, H. Muñoz-Avila, P. Zhao, and H. Wang, 'Hierarchical task network planning with resources and temporal constraints', Knowledge-Based Systems, vol. 133, pp. 17–32, Oct. 2017, doi: 10.1016/j.knosys.2017.06.036.
[12] T. Vaquero, 'A Knowledge Engineering Framework for Mission Operations of Increasingly Autonomous Spacecraft', 32nd International Conference on Automated Planning and Scheduling (ICAPS 2022), p. 8, 2022.